Forecasting Daily River Flow

ABHISHEK KUMAR

September 2024

School of Mathematics,
Cardiff University

A dissertation submitted in partial fulfilment of the requirements for **MSc (Data Science & Analytics)** by taught programme, supervised by **Owen Jones.**

Acknowledgements

Firstly, I would like to thank for my Project Supervisor Owen Jones who has provided valuable guidance. His insight and advice make able to understand and tackled unseen problems with data. His advice ensured that project progress is on right direction.

Secondly, I would like to thank for National River Flow Archive at UK Centre for Ecology & Hydrology and Natural Resources Wales which are Measuring authority at station for data Acquisition.

Abbreviation

NSE:	Nash–Sutcliffe efficiency
R-squared:	Coefficient of determination
MAPE:	Mean absolute percent error
MSE:	Mean squared error
MAE:	Mean absolute error
I_A:	Willmott's Index of Agreement
R:	Correlation coefficient
MLP-NN:	Multi-layer perceptron neural network
LSTM:	Long short-term memory neural network
XGBoost:	Extreme gradient boosting
CV:	Coefficient of variation
CART:	Classification and Regression Tree

Executive Summary

This dissertation investigates next-day river flow forecasting using both traditional linear regression models and advanced machine learning techniques, applied to the Taff River at Pontypridd (station ID: 57005) and its seven active tributaries. For data collection GDF and CDR data were obtained from the National River Flow Archive and ranged from 2006-05-26 to 2017-12-31. Comprehensive data preprocessing measures encompassing data cleaning, de-seasonalisation, data transformation (log transformation and Box-Cox transformation), and stepwise selection (forward and Backward techniques) were employed to choose input variables to improve model performance.

The study thus focuses on Linear Regression analysis (LR), Support Vector Regression (SVR), Neural Networks (NN), and Generalized Additive Models (GAM). The Random Search Cross-Validation method was used to tune the hyperparameters for the SVR model, and this change yielded considerable improvement. This study provided evidence that the SVR model, especially when fitted to Box-Cox transformed data, yielded lower prediction errors compared to other models in the three-evaluation metrics, namely, R-squared, MAE, and MAPE. This is an indication that the Neural Network model which performed well was not fully optimized and could be an area of further research. Improving the accuracy of the model could be done by fine-tuning the Neural Network or trying an ensemble of SVR and NN instead of the simple merging done in this dissertation.

The study addresses a key research gap: comparison of simple linear regression idea with the most sophisticated machine learning algorithms in the context of river flow prediction. It notes that the use of non-linear models including the SVR offers better performance than the linear regression in determining the relationship in hydrological data. This project fills a gap in hydrological modelling by demonstrating the impact of data preprocessing and hyperparameter optimization on model performance.

In conclusion, the presented work indicates that the application of the more sophisticated machine learning model, namely SVR, leads to river flow forecasting improvement in the Taff River catchment area. These results imply the need for careful attention to preprocessing and the choice of appropriate hyperparameters.

Future research should focus on optimization of neural network and exploring ensemble learning techniques to combine strengths multiple models to enhance more accurate predictions in context of river flow forecasting.

Table of Contents

Acknowledgements..... 1
Abbreviation 2
Executive Summary..... 3
Table of Contents 4
1. Introduction..... 7
2. Literature Review 9
2.1. Importance of daily river level forecasting..... 9
2.2. Data Acquisition and Data Preprocessing practice for river level forecasting..... 10
2.3. Input Variable Selection for Lagged Data River Level Forecasting……..... 11
2.4. Method of Model building and Evaluation in Daily River Forecasting..... 12
2.4.1. Support Vector Regression (SVR) Applications..... 12
2.4.2. Linear Regression Models in river flow Forecasting…..... 12
2.4.3. Hybrid Machine Learning model in river flow Forecasting..... 13
2.4.4. Other models investigated against Linear Regression Model for river flow forecasting..... 14
3. Methodology……..... 15
3.1. Data collection and initial geographical setup…..... 15
3.2 Data Preparation and Cleaning……..……..... 16
3.2.1 Data formatting of daily flow and rainfall dataset..... 16
3.2.2 Data merging of daily flow and rainfall data at each gauge……..…….16
3.2.3 Numerical Summary of each gauge after gdf and cdr data merge……..... 17
3.2.4 Inspection of Missing values and Data Redundancy after data merging..... 18
3.2.5 Handling Missing Values…….……...18
3.2.5.1 Exclusion of Gauge with Insufficient Data..... 18
3.2.5.2 Alignment of Dataset Time Frames……..... 21
3.2.5.3 Imputation of Remaining Missing Values..... 22
3.2.6 Inspection of Outlier and Nature of data..... 22
3.2.7 Combined all dataset in single dataframe……..... 25
3.3 Exploratory Data Analysis..... 26
3.3.1 Relation of daily flow and daily rainfall at various gauges…..... 27
3.3.2 Relation between daily flow vs Daily across various gauge..... 35
3.3.3 Correlation Analysis between daily flow and daily Rainfall..... 37

3.3.4 Distribution Analysis of Daily flow and Rainfall at Each gauge..... 38
3.4 Data Transformation..... 41
3. 4.1 Log Transformation.....41
3.4.2 Box-Cox Transformation…….44
3.5 Variance and Mean Analysis..... 49
3.6 Moving Average to find underlying trend in data……..... 51
3.7 Time Series Decomposition to find seasonality or trend (STL Decomposition) 56
3.8 Autocorrelation Function and Partial Autocorrelation Function..... 64
3.9 Model Building……...69
3.9.0 Automatic Stepwise Selection of Input Variables..... 69
3.9.1 Linear Regression model…….69
3.9.2 Support Vector Regression (SVR) Model..... 70
3.9.3. Neural Network Model..... 71
3.9.4 Generalized Additive Model (GAM)..... 67
3.10 Model Evaluation Metrics..... 72
3.11 Evaluation of SVR using cross-validation method.....73
3.12. Model Optimization and Hyperparameter Tuning..... 73
4. Result and Comparison of Models……...……...73
4.1 Split of data for Training and Testing…….....73
4.2 Model Experiment..... 74
4.2.1 Variable selection at Log transformation data..... 74
4.2.1.1 Linear Regression (LR) model at log transformed data……74
4.2.1.2 Support Vector regression (SVR) model at log transformed data….....77
4.2.1.3 Neural Network (NN) model at log transformed data.....80
4.2.1.4 Generalised additive model (GAM) at log transformed data.....82
4.2.1.5 Model comparison on log transformed data..... 84
4.2.1.6 Model optimization of SVR model by features scaling and hyperparameter tuning…….86
4.2.2 Variable Selection at Log transformation with de-seasonalized data……. 87
4.2.2.1 SVR model at log transformed + de-seasonalised data…….87
4.2.3 Variable Selection after applied box-cox transformation…….……...88
4.2.3.1 SVR model after applied box-cox transformation……88
4.3 Model comparison of all optimized SVR model……90
4.4 Next Day flow Prediction……90

5. Conclusion…….90
6. References……..…….91
7. Appendix……94

1. Introduction

Prediction of river level is important in water resource management, flood and droughts effects (Jamei *et al.*, 2022). Accurate forecasts are even crucial for flood protection, water supply, and demand, as well as for preparation to the future hydrologic events (Granata and Di Nunno, 2023). With the advancement in technology, computational approaches have been enhanced enabling the usage of enhanced models embracing both rainfall and river flow in different evolving tributaries thus enhancing the perception of river systems. This has become more important as the extent of fluctuation in the river levels would determine the amount of water to be expected during floods.

The Traditional techniques, especially, Linear Regression (LR) model, has been practiced for river level prediction. However, these approaches raise difficulties in identifying nonlinear dynamism, which is characteristic of river systems (Fernandes *et al.*, 2023). The belief in linearity of the independent and dependent variable relationship reduces LR capability in addressing variable influencing factors like rainfall, land absorption and human activity. Advanced models are required because river systems involve multiple multifaceted dependencies between flow across gauges and temporal lags Machine learning (ML) models, by contrast, offer greater flexibility in handling non-linear relationships and can better accommodate the multifactorial nature of river catchments (Zakaria *et al.*, 2023).

However, it became apparent that traditional models have their drawbacks and are widely used despite that. Since LR fails to consider nonlinear relationship, the major issues are in computation of river levels especially in conditions of changes in rainfall regime and water supply from artificial reservoirs (Zhang *et al*., 2018). Therefore, the use of higher order models including Support Vector Regression (SVR), Neural Network (NN) and Generalised Additive Models (GAM) is considered to be the solution. These models can integrate spatial and temporal information of gauges providers in the catchment areas and gives more detail about the rainfall events and river responses (Kedam et al., 2024). The staffers ensure that the models are fed with data from various tributaries, filling gaps left behind by traditional approaches of modelling.

The focus of this study is on developing and testing models to predict next-day river levels by using both traditional and ML approaches. The analysis includes data from the National River Flow Archive, specifically focusing on the River Taff catchment area and its tributaries. By combining historical river

flow and rainfall data, this study aims to improve the accuracy of predictions. The comparison of LR with ML models highlights the latter's ability to capture complex, non-linear relationships between rainfall and river levels.

One of the main challenges in river level forecasting is improving prediction accuracy to ensure effective flood management and water resource allocation. This research aims at identifying the challenges of the Traditional models and examining the possibilities of the sophisticated methods ML techniques for increasing the credibility of prediction. The findings will inform future strategies for more efficient management of water resources, especially in areas susceptible to flooding, will be informed by the findings, the Research aims at closing the gap between Traditional and ML models, which is useful in improving the level of accuracy of river level forecasting following changes in environmental factors.

2. Literature Review

2.1. Importance of daily river level forecasting

River level forecasting is significant for water resource planning, flood control and irrigation planning on the daily basis. It helps in the protection of human life, reduction of risks in the occurrence of adverse weather conditions and proper management and distribution of water. (Granata and Di Nunno, 2023) state that accurate prediction of the river flow is significant in different fields including agriculture and hydropower to mitigate the impacts of floods or droughts that lead to revenue losses and threats to aquatic life.

To give early warnings and take further preventive measures, it is essential to make a proper forecast amidst flood management, especially in regions where there is no integrated flood warning system for heavy rainfall, (Jamei *et al.*, 2022). Even though higher accuracy in river flow prediction is achieved by machine learning models mainly due to their non-linear nature and flexibility in incorporating changes in conditions (Zakaria *et al.*, 2023), it is still a challenge to forecast river level daily, especially in regions with complex hydrological environments.

Although, plenty of improvements have been made in the case of the forecasting methods, there is no daily forecast of river level for Taff River and its branches in Cardiff and Pontypridd area. Currently, services such as the Met Office and Natural Resources of Wales provide flood risk advisories that provide up to five days' prognosis but refrain from individual and daily river level prognosis.

The Objective of this research aims to address this gap by carrying out a modelling approach on daily river levels of the Taff River and its tributaries with the aid of machine learning algorithms. This study also employed historical flow and rainfall data with intention of developing, validating and comparing the model result which may lead to a better and specific next day ahead river level forecasting tool to support flood mitigation and water resources management in the area. An application of machine learning models like Support Vector Regression, Generalised additive Model and neural networks in such systems as here proposed should go a long way in enhancing the accuracy of these forecasts and be revealed as an extremely useful addition.

2.2. Data Acquisition and Data Preprocessing practice for river level forecasting

Many recent research on river flow to accurate forecasting have been based strongly on the following aspects: Efficient data gathering, cleaning, and preprocessing. (Kedam *et al.*, 2024) claimed the significance of accruing data from more than one station over a long-time span, saying that large datasets are prerequisite for the setup of accurate predictive models, in his study, they have used handled missing values and outliers and more attention on the features scaling like standard Scaler and some cognizance of non-stationary and non-linear contact points in the data. In the same way, (Serrano-López *et al.*, 2024) also concerned with the river flow and collected height data from different sources, they were more careful about the quality of data and made it suitable for modelling and filling of missing value by KNN regressor for large gaps and interpolation for small gaps in this study also outlier was handled by using moving average with window size of 25.

However, these studies highlight the need to devote more time to data collection and analysis in areas of interest, especially concerning the daily river level prediction of the Taff River and its tributaries. Existing solutions lack optimal solution to address issues of fusion and preprocessing of datasets for focusing localized daily predictions.

This dissertation aims to estimate daily river level of the Taff River and its major tributaries with the help of a large set of data from several gauges. Using such data preprocessing techniques as detection and handling of outliers, handling missing data, features scaling with MinMax Scaler, this research seeks to enhance the accuracy of the localized predictions with the view of providing a more local specific management on water resource and flood risk.

Concerning missing data, both (Serrano-López *et al.*, 2024) and (Zakaria *et al.*, 2023) applied imputation methods to deal with missing values in the datasets. However, more detailed techniques to resolve the areas of the catchment and therefore adaptation of the models for daily forecasting seem to be required. These techniques are used in this research with emphasis on retaining inter-conversion consistency of data measured on different gauges, which is important in generating daily prediction. Also addressed the procedure of handling missing values after the integration of multiple gauges.

Concerning outliers, (Kedam *et al.*, 2024) utilized StandardScaler for balanced scaling while (Tawakuli *et al.*, 2024) employed Grubbs' test and Interquartile Range methods. Although these methods work well, their use is not very beneficial in short-term forecasts for individual subordinate

rivers such as the Taff. This research enhances these methods and fine-tunes them to suit hydrologic conditions of the Taff River basin to guarantee that data input to the model is credible.

2.3. Input Variable Selection for Lagged Data River Level Forecasting

Variable selection is vital in establishing suitable models for river flows predictions especially if one is working with data shifted across several days. (Akbarian, Saghafian and Golian, 2023) underlined the significance of reducing the dimensionality by addressing the issue of managing a large set of potential predictors especially in monthly streamflow forecasting. but they also highlighted some disadvantages including the fact that it is very prone to overfitting especially when the size of data is small and the fact that it takes a very long time to manage so many variables. To address these problems, their work used Pearson correlation analysis (PCA) and Recursive Feature Elimination (RFE) to reduce the dimensionally of input matrix by removing the predictors with high collinearity and low selection importance. Despite these challenges, the approach sought to strike a fine line between the model's complexity and accuracy when making the predictions though the issue of overfitting arose especially when not well addressed.

(Hussain and Khan, 2020) also stressed the significance of selecting appropriate predictors in the study of monthly river flow prediction. To identify appropriate lagged values for the river flow data, they utilized ACF and PACF and incorporated the appropriate lags into their model inputs. It is essential to apply this process to reveal the temporal dependencies of the data to enhance the model's capacity to predict future river flows.

(Lian *et al.*, 2021) study selected chosen optimal lag with the help of PACF function and was more inclined towards the application of PCA method to analyse and reduce the input variables to their most fundamental form that can provide accurate forecast. In their study on multi-day-ahead streamflow forecasting they used PCA to reduce dimensionality of the input space and observed the gains and reduction in risks of overfitting. When PCA was combined with other sophisticated models like the LSTM, the predictive capability was improved by emphasizing on the considerably relevant features of the data.

While these studies effectively address feature selection for lagged data, there is limited research focused on stepwise selection algorithm to select better input variables for number of lagged days to daily river level forecasting.

The purpose of this study is to determine by applying these feature selection techniques, including ACF/PACF analysis and stepwise selection techniques, to daily river level forecasting. By focusing on the Taff River and its tributaries, the integration of these feature selection methods with Linear regression model and advanced models such as Generalized Additive Models (GAM) and Support Vector Regression (SVR), Neural Network and compared result among these models.

2.4. Method of Model building and Evaluation in Daily River Forecasting

Different research has also shown that integration of other ML algorithms such as SVR, ANN, RF and XGBoost with meta-heuristic optimization algorithms improves accuracy of the forecast of river flow as well as the stability of the models.

2.4.1. Support Vector Regression (SVR) Applications

(Nguyen, Huu and Li, 2015) used the Support Vector Regression (SVR) to predict changes in the water level of Mekong River, a comparison of the accuracy of the model offered by LASSO and Random Forest. It also found out that SVR was able to capture non-linear relationship better than other models in terms of RMSE and MAE. (Malik et al., 2020) have enhanced SVR with gamma test (GT) for the selection of variables and applied meta-heuristic algorithms namely Harris Hawks Optimization (HHO), Bayesian Optimization (BO), Multi-verse Optimizer (MVO), Ant Lion Optimization (ALO), Spotted Hyena Optimizer (SHO) and Particle Swarm Optimization (PSO). These hybrid models were evaluated using RMSE, Scatter Index (SI), Coefficient of Correlation (COC), Willmott Index (WI) and graphically, with the SVR-HHO model demonstrating superior performance in daily streamflow prediction.

2.4.2. Linear Regression Models in river flow Forecasting

Numerous researchers have investigated the utilization of linear regression models, frequently as a reference point for evaluating more advanced machine learning (ML) models. In the different Iranian basins studies of streamflow forecasting, (Akbarian, Saghafian and Golian, 2023) used Multiple Linear Regression (MLR). However, while employing the MLR is very easy due to its clear interpretable structure, the model often fails when comparing it with non-linear models like Artificial Neural Networks (ANN), Random Forest (RF), or XGBoost. To demonstrate that the chosen nonlinear models more accurately predict the results compared to the MLR model, the evaluation criteria such as NSE, NRMSE, and KGE were used in the study. This trend was particularly apparent in the regions with

complex hydrological parameters, and which could not be properly described by the linear MLR model.

2.4.3. Hybrid Machine Learning model in river flow Forecasting

In the recent past, there are number of key developments in use of ML models for river flow prediction as concerns enhance accuracy of predictions and handling of non-linear characteristics of water data. In the present paper, a model integration of CatBoost with Genetic Algorithm (GA) has been done to predict daily river flow in Sakarya Basin of Turkey using univariate time series data by (Kilinc *et al.*, 2023). This model outperformed the basic models which includes LSTM & Linear Regression while comparing the RMSE & R^2 values. Similarly, in (Cui et al., 2020), the authors proposed the Emotional Neural Network (ENN) to predict the hourly river flow for Mary River, Australia and its performance was found better than the models such as Minimax Probability Machine Regression (MPMR), Relevance Vector Machine (RVM) and Multivariate Adaptive Regression Splines (MARS). Variables chosen by the PACF function as the input were very encouraging since ENN can capture the subtle and non-linear relationship between the data in real-time hydrological forecasting. In these studies, the authors emphasize the need to combine machine learning with optimization algorithms to improve the river flow estimations and their reliability.

(Jamei *et al.*, 2022) studied the flood forecast ability using time-varying filter-based empirical mode decomposition (TVF-EMD) in combination with CART and Machine learning models such as LSTM, CFNN, MARS and GBDT of Australia's Clarence River. According to evaluation criteria of R, MAPE, KGE, RMSE and NSE, they pointed out that CART-TVF-EMD-CFNN model had a better performance over others in Short-term prediction, hence they clearly indicated that there is a limitation with highly non-stationary data using LSTM.

In a similar vein, (Zakaria *et al.*, 2023) evaluated MLP-NN, LSTM, and XGBoost for the use of forecasting in the Muda River of Malaysia. Based on RMSE, MAE, MAPE, and R^2, the most accurate model was MLP-NN, while LSTM offered superior long-term forecasting. While XGBoost proved efficient with the least time, its accuracy was the lowest among all the classifiers used. These studies demonstrate the need for careful selection of the right models and evaluation metrics in water level forecasting.

2.4.4. Other models investigated against Linear Regression Model for river flow forecasting

(Tsakiri, Marsellos and Kapetanakis, 2018) introduced a hybrid model combining Artificial Neural Networks (ANN) and Multiple Linear Regression (MLR) for flood prediction in the Mohawk River, New York. Author's study shown robustness of ANN and assessed the effectiveness of MLR. The study found that ANN models applied to decomposed time series outperformed MLR in predictive accuracy, especially in capturing non-linear relationships. The use of time series decomposition was critical in improving the model's performance, as evidenced by better R-Squared and MSE.

(Latt and Wittenberg, 2014) employed Stepwise Multiple Linear Regression (SMLR) and Artificial Neural Networks (ANN) to analysed flood forecasting in Myanmar's Chindwin River. Based on ACF, PACF, cross correlations CCF, and stepwise selection for input variables, they noted that ANN was marginally superior to SMLR and had higher R^2, lower RMSE, MAPE, and CV for extreme floods. Likewise, (Chieu *et al.*, 2024) demonstrated that the Kien Giang River's water level can also be predicted by models such as LR and SVR. Thus, the analysis found that on R^2, NSE, and RMSE, the LR model outperformed more complex models, which were overfitting, because the LR model was able to correctly estimate the linear response of water levels to rainfall.

Despite these developments in the studies, there are still limitations on the different comparison of linear regression with the popular machine learning models under different hydrological conditions. The present work addresses those gaps with the use of traditional and machine models, applying stepwise selection and hyperparameter tuning for the prediction of next day river levels using metrics such as, MAPE, MAE, MSE and R^2.

3. Methodology

3.1. Data collection and initial geographical setup

Data was collected for this study from National River flow archive which maintained by UK Centre for Ecology & Hydrology where Measuring authority at each gauge Natural Resources Wales. Focused on Gauged Daily flow (GDF) and catchment daily rainfall (CDR) data across Multiple gauges. The primary gauge station utilized for next-day river flow forecasting was the Taff at Pontypridd (station 57005). In addition, data were collected from seven tributary gauge stations within the catchment, including Taf Fechan at Taf Fechan Reservoir (station 57001), Taf Fawr at Llwynon Reservoir (station 57002), Taff at Merthyr Tydfil (station 57015), Taff at Fiddlers Elbow (station 57007), Cynon at Abercynon (station 57004), Rhondda at Trehafod (station 57006), and Rhondda Fawr at Tynewydd (station 57017) as depicted in Figure 1.0 (page 17). Daily flow at gauges was measured in m3/s and daily rainfall measured in mm. Discharge data were measured in cubic meters per second (m^3/s) for the daily flow at these stations while precipitation data were taken in millimeters (mm) for the daily rainfall.

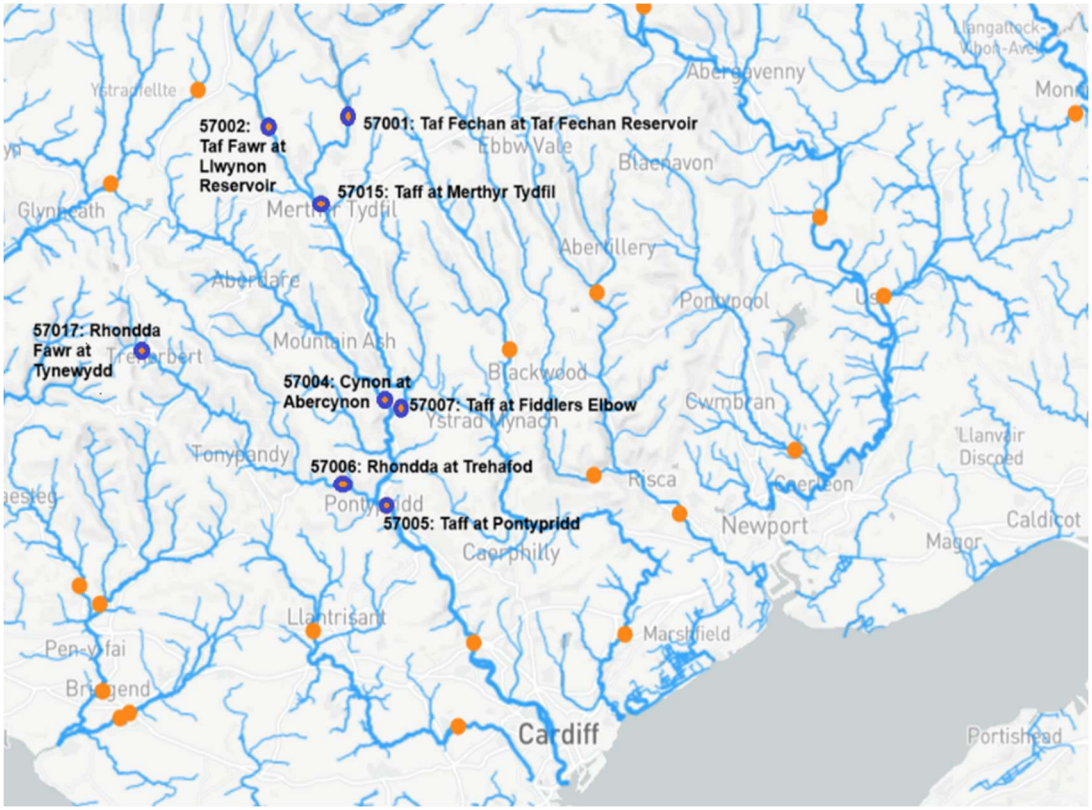

Figure:1.0, gauges used for river flow forecasting (National River Flow Archive, no date)

The catchment areas corresponding to each gauge station are as follows:
57005-Taff at Pontypridd: 454.8 km²
57006-Rhondda at Trehafod: 100.5 km²
57017-Rhondda Fawr at Tynewydd: 16.6 km²
57004-Cynon at Abercynon: 106 km²
57007-Taff at Fiddlers Elbow: 194.5 km²
57015-Taff at Merthyr Tydfil: 104.1 km²
57002-Taf Fawr at Llwynon Reservoir: 43 km²
57001-Taf Fechan at Taf Fechan Reservoir: 33.7 km²

The dataset was initially read into a Python environment, whereby the GDF and CDR data sets were combined into one continuous data frame for each gauge station separately. This was necessary to bring the data in datasets into a realm of homogeneity to enable subsequent processes such as cleaning and analysis.

3.2 Data Preparation and Cleaning

The data cleaning and data pre-processing phases were critical to the assessment of quality and usability of the datasets in the current study that sought to build robust river flow forecasting models. The measures taken are outlined below:

3.2.1 Data formatting of daily flow and rainfall dataset

Upon initial inspection of the original datasets, it was noted that there were metadata entries in Gauged Daily Flow (GDF) and Catchment Daily Rainfall (CDR) dataset that needed to be eliminated. To counter this, the data was restructured for each gauge. In more detail, extracting data from both the GDF and the CDR dataset involved selecting information only from the 19th row down to the first and second columns, where the date and measurement values were indicated. The date was started in the first column, whereas the second column was used to store the GDF or CDR values. This reformatting procedure was systematically performed on all the eight gauges, and each gauge's dataset was stored separately in a dataframe for further analysis.

3.2.2 Data merging of daily flow and rainfall data at each gauge

The next stage involved merging the two distinct DataFrames—one containing daily flow data and the other daily rainfall data—into a single cohesive dataframe for each gauge. An 'inner join' method was employed, using the date as the common key, to ensure that only matched dates between the two datasets were retained (author, year). Such an approach was necessary to eliminate the inclusion of missing values within the GDF or CDR columns. Subsequently, type conversions were made for the purpose of transforming the 'date' column into a datetime format and the 'gdf' and 'cdr' columns into floats. Further, a new column termed as 'gauge' was included in each of the DataFrames, which captured the unique station identification number of the location (for instance, 57005 for Taff at Pontypridd). This was done to ensure that the dataset was of equal quality across all the gauges before proceeding with the next steps in the analysis

3.2.3 Numerical Summary of each gauge after gdf and cdr data merge

After performing the above steps of data merging, a statistical summary was obtained for each gauge to gain a general perspective on the data. Results of this summary provided the count, mean, std for GDF values and count and mean, std for CDR values. Table (1.0) present numerical summary of each gauge which helped to identify data consistency and anomalies.

	GDF			CDR		
Gauge	Data count	mean	std	Data count	mean	std
57005	17259	20.720056	27.358210	17259	5.344696	9.331123
57006	16955	5.899018	7.739532	17259	6.443687	11.302441
57017	731	1.156085	1.306841	731	6.830369	10.990441
57004	21642	4.409308	6.155301	22007	5.257218	9.460269
57007	16316	6.989491	9.668518	16316	4.931907	8.544934
57015	14337	3.796097	6.116896	14337	5.402260	9.210841
57002	25821	1.098884	1.852487	31503	5.482570	9.406618
57001	22777	0.799590	1.528575	29645	5.489664	9.382976

3.2.4 Inspection of Missing values and Data Redundancy after data merging

In analysing merged data frames, the first step done was a check for missing values and duplicate data. Table 2.0 below depict, the number of missing values as well as the cases of duplicated data among the gauges. The inspection also pointed that certain feature like gauges 57006, 57004, 57002 and 57001 were having missing values which need to be handle properly for the best management of the dataset.

Gauge id:		57005	57006	57017	57004	57007	57015	57002	57001
GDF	Missing values	0	304	0	365	0	0	5682	6868
	Redundancy	0	0	0	0	0	0	0	0
CDR	Missing values	0	0	0	0	0	0	0	0
	Redundancy	0	0	0	0	0	0	0	0

3.2.5 Handling Missing Values

3.2.5.1 Exclusion of Gauge with Insufficient Data

After analysing the missing data patterns of all the gauges as shown above. The analysis also showed that there were huge disparities in the availability of the Gauged Daily Flow (GDF) data at various gauges. These gaps were represented on time series plots, thus making it easier to understand when data was missing (Figures:2.1-2.8, page:29-21). This was noticed because there existed long gaps in the data owing to events that have included reconstruction work and main gap highlighted in the metadata of the given data set. Based on these gaps as well as realizing that eliminating one gauge could greatly affect the results a decision was made to eliminate gauge 57017 which had data for only two years. If this gauge was included the sample size of the other gauges would have been reduced enormously thus reducing the overall robustness and reliability of the model in the study.

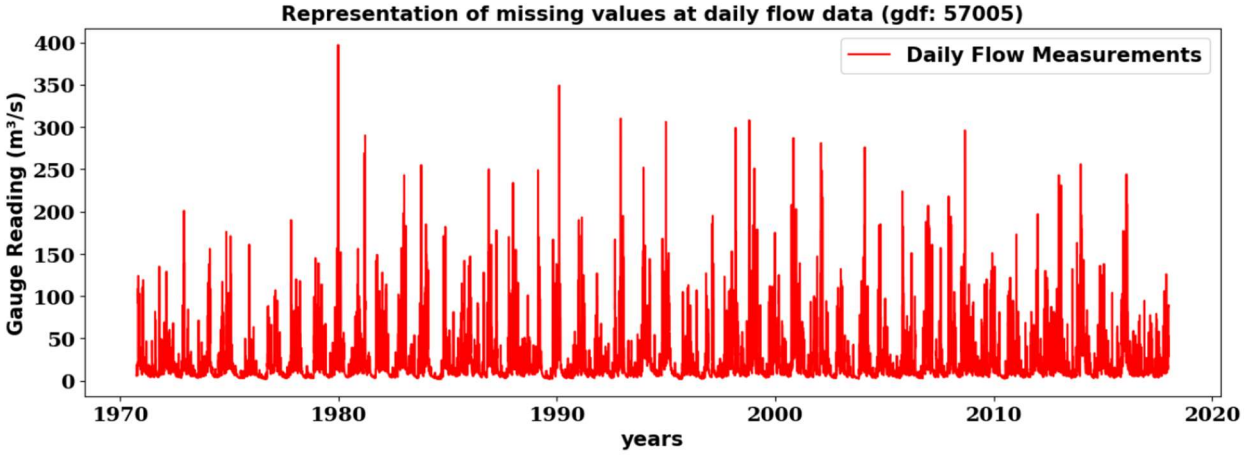

Figure:2.1

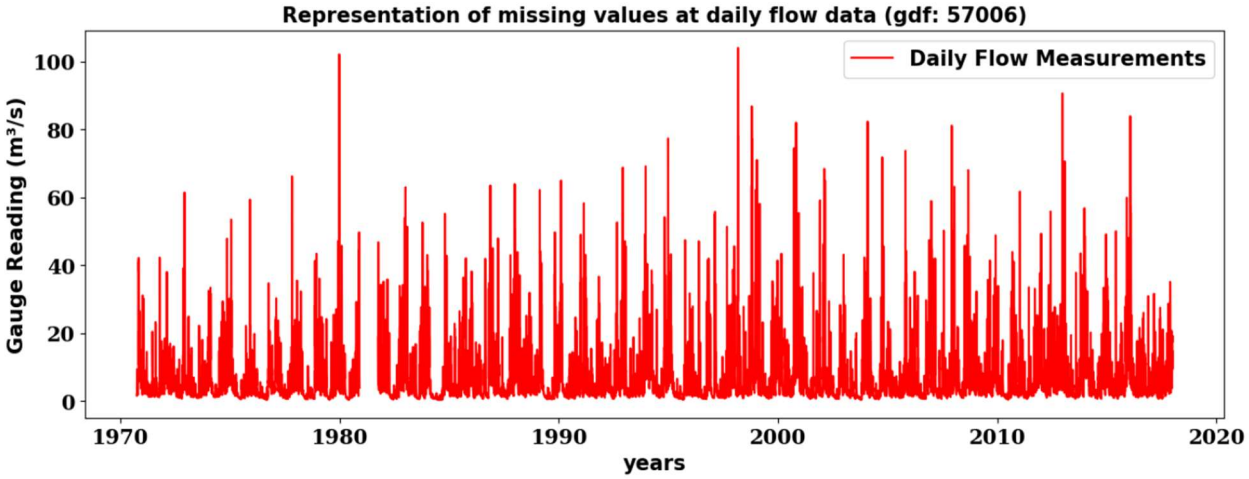

Figure:2.2

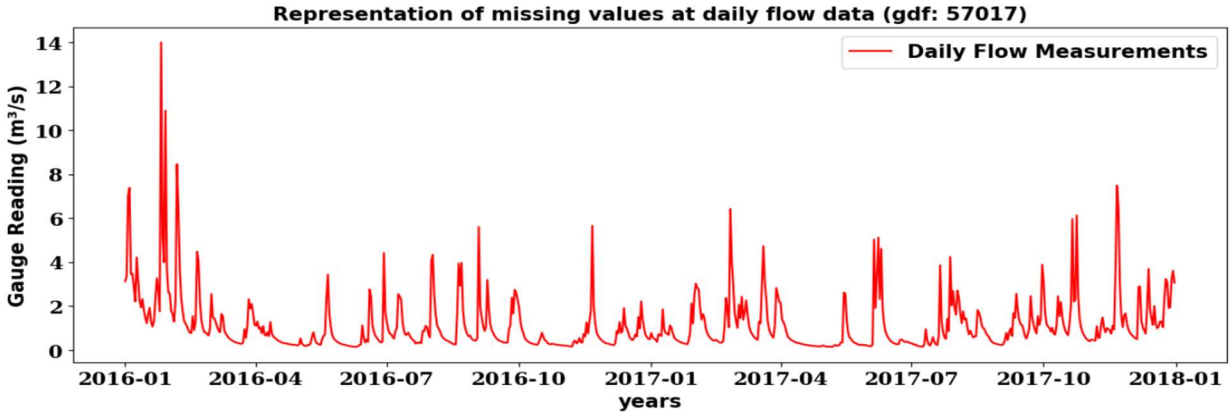

Figure:2.3

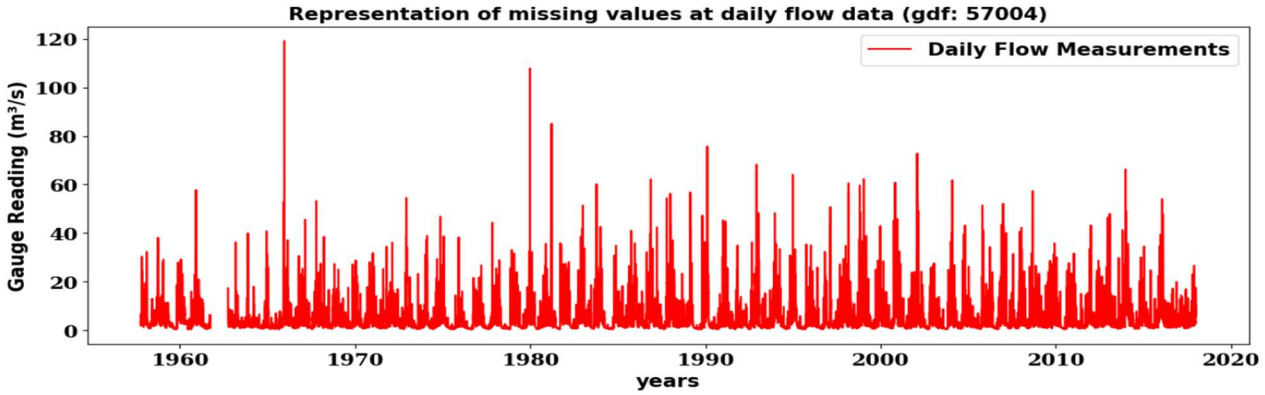

Figure:2.4

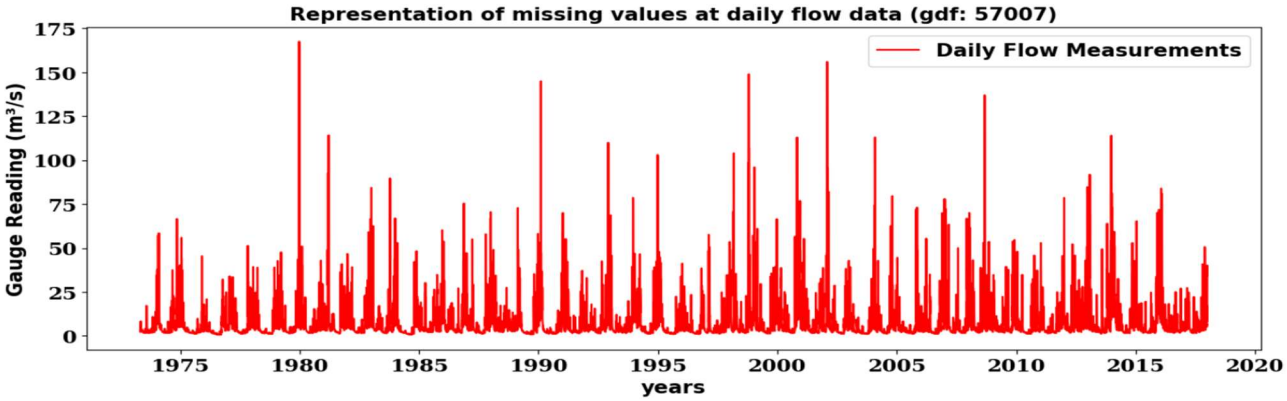

Figure:2.5

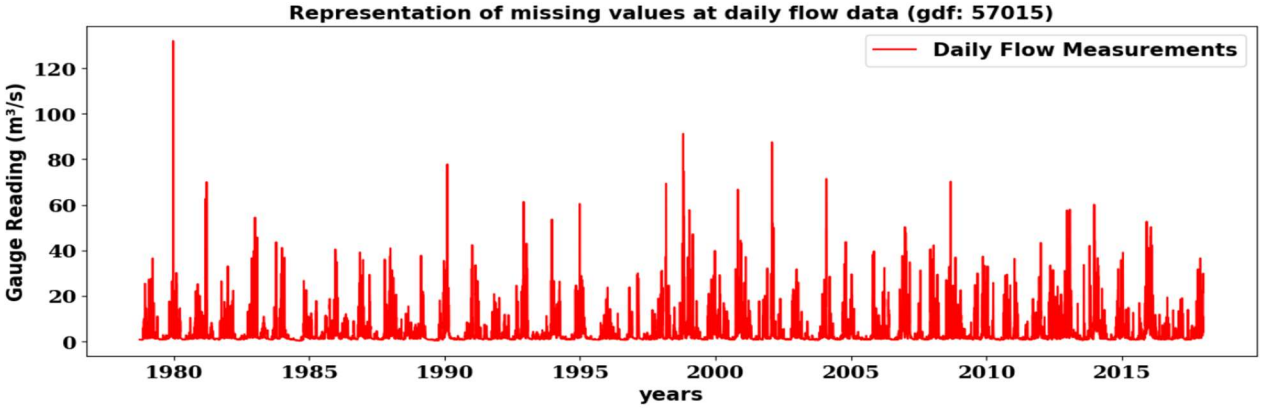

Figure:2.6

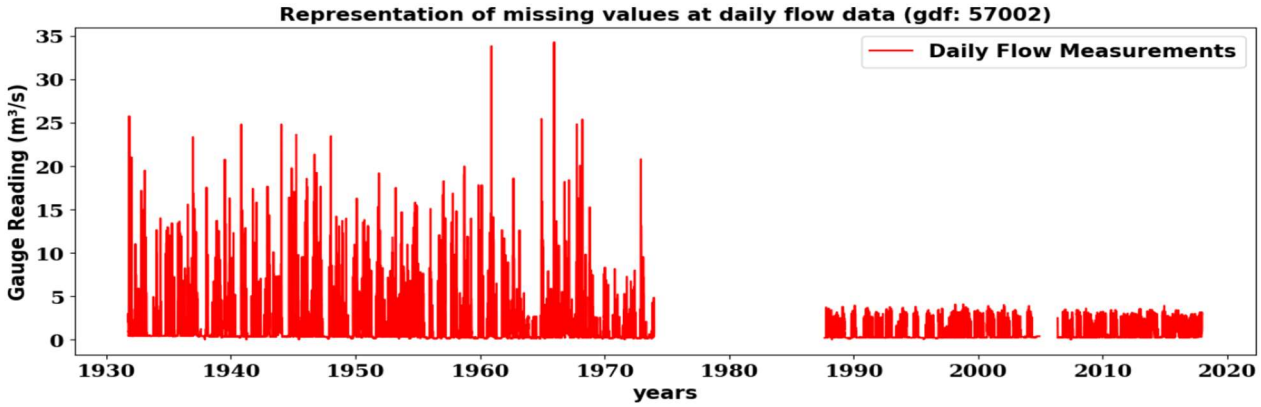

Figure:2.7

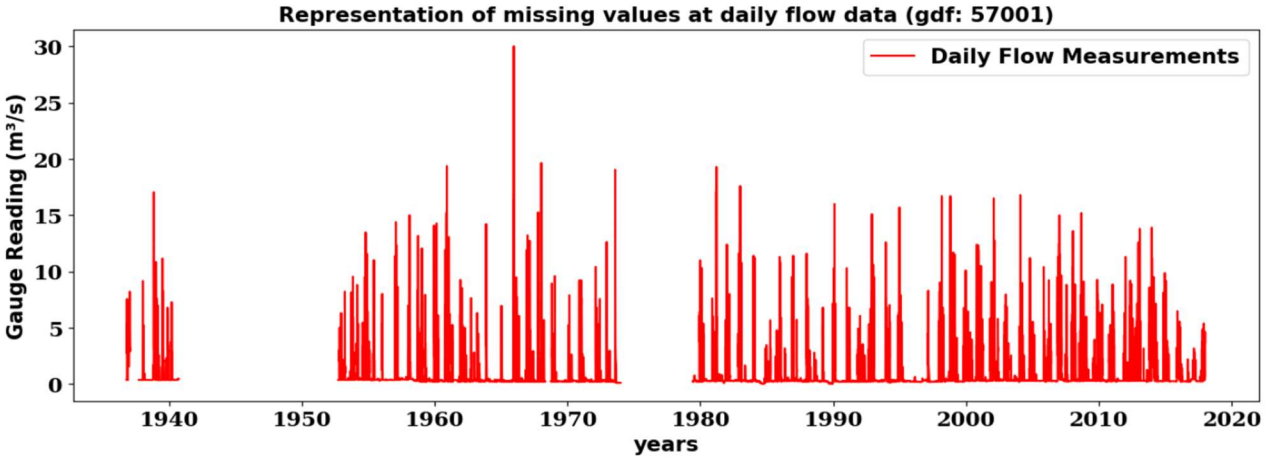

Figure:2.8

3.2.5.2 Alignment of Dataset Time Frames

As seen in (Figure:3.0, Page:22), it was clear that missing values were a common occurrence for most of the gauges to account for missing data for the purpose of achieving consistent data across the gauges, with the observation of (Figure:3.0, Page:22), gauge 57002 was used and the dataset was brought to standard by realigning the other gauges with data from this gauge. In selection of data for the study, data was chosen from 2006-05-26 to 2017-12-31 because data on rainfall for the subsequent dates was unavailable. This was achieved by bringing all datasets from the different gauges into this timeframe thus making sure that the final dataset was complete for analysis.

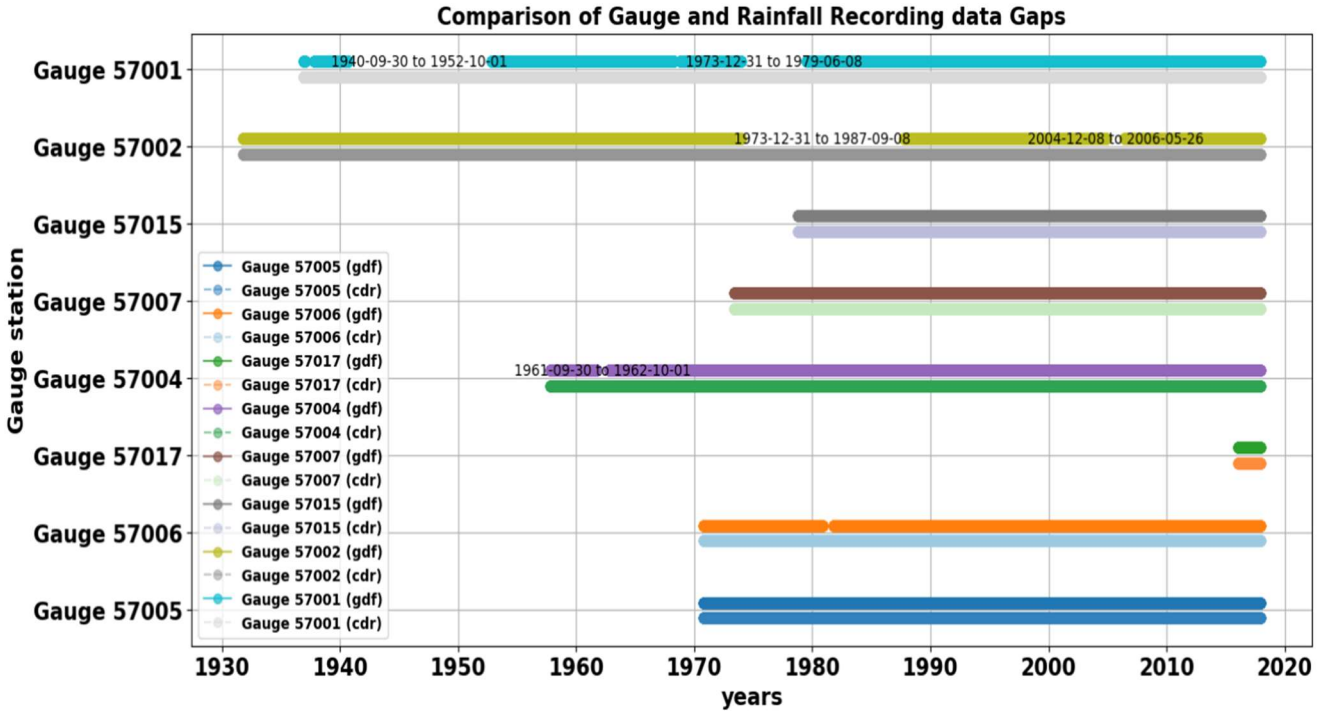

Figure:3.0

3.2.5.3 Imputation of Remaining Missing Values

After the alignment, only six missing values were found in the gauge 57002 GDF dataset. The steps that were taken to address missing values were the forward fill and backward fill method (author, year). The forward fill method was used because it ensured that there was no interruption in the data by using the last recorded values to carry forward, which proves especially helpful when it comes to sequences that have some missing data. Another technique also known as backward- fill was also used where missing values were filled by subsequent values. These methods were chosen because they provide a direct way of preserving important formats of the time series when extracting without forcing modification of sets to follow a certain sequence.

At this stage, all eight gauges datasets were in the individual data frames where each data frame comprised of the date, CDR, GDF, and the gauge number. This structured organization of the data helps in replicability of the data cleaning process and assists in analysis.

3.2.6 Inspection of Outlier and Nature of data

To first evaluate for any possible outliers in the Gauged Daily Flow (GDF) and Catchment Daily Rainfall (CDR) data of the selected gauges, box plots were employed, as described below in (Figures:4.1-4.7, page23-25). When analysing the data given by the box plots, it was clear that the distribution of both GDF and CDR values was right skewed, and there were many highly influential

observations, which lay outside the interquartile level. These extreme values, although highlighted as outliers by the box plots (author, year), probably include important hydrological events including floods or heavy rainfall and are not necessarily because of error or anomalous circumstances. Since all the gauges exhibited right skewness, the following steps were taken to perform additional measures by creating distribution plots and data transformation.

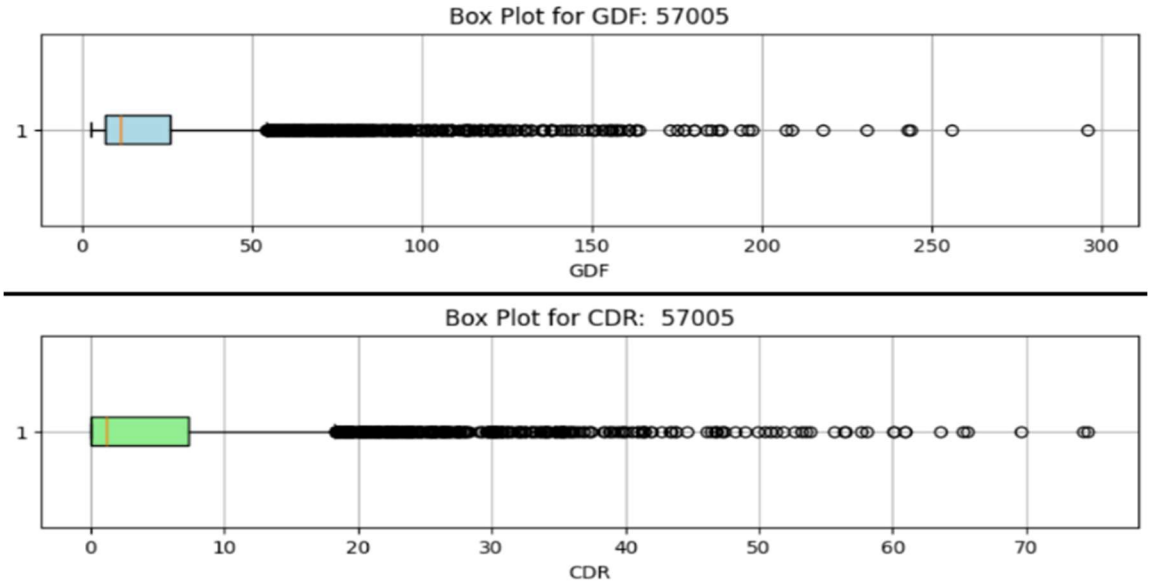

Figure:4.1

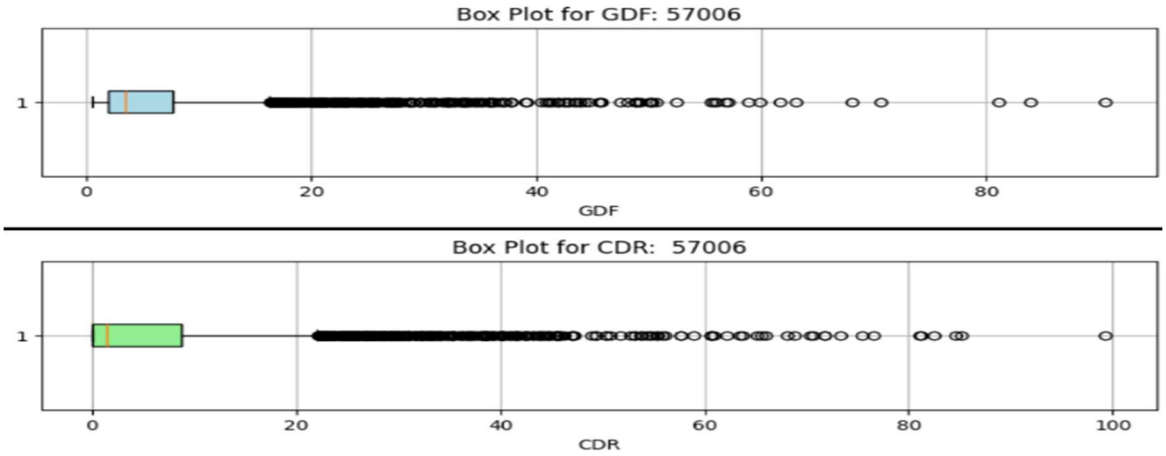

Figure:4.2

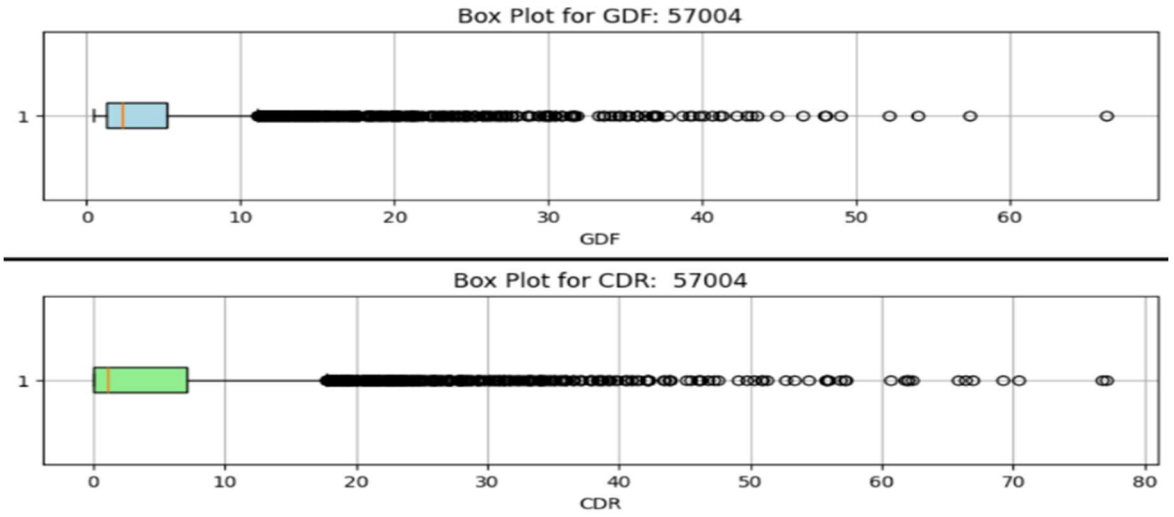

Figure:4.3

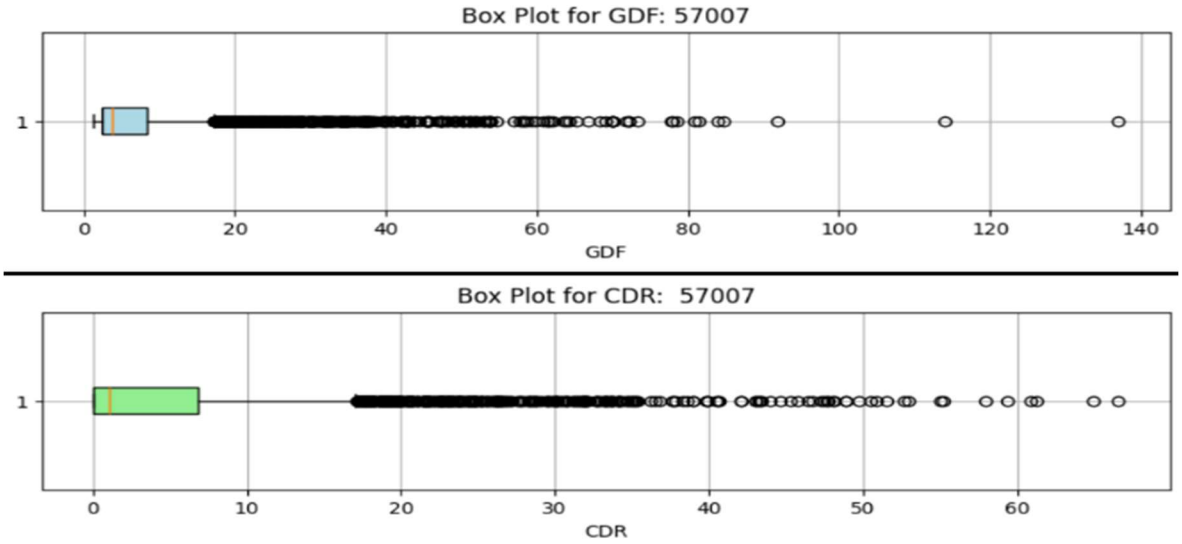

Figure:4.4

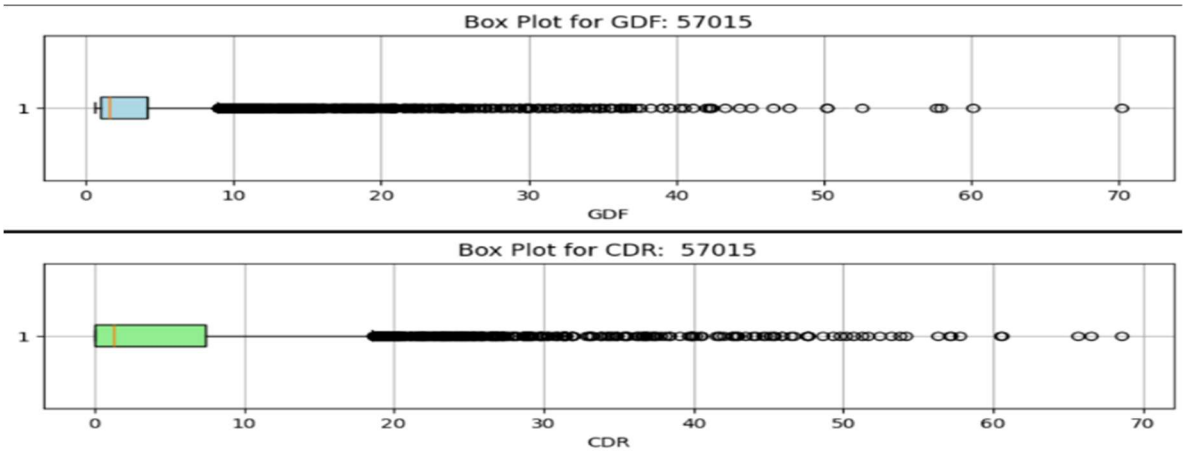

Figure:4.5

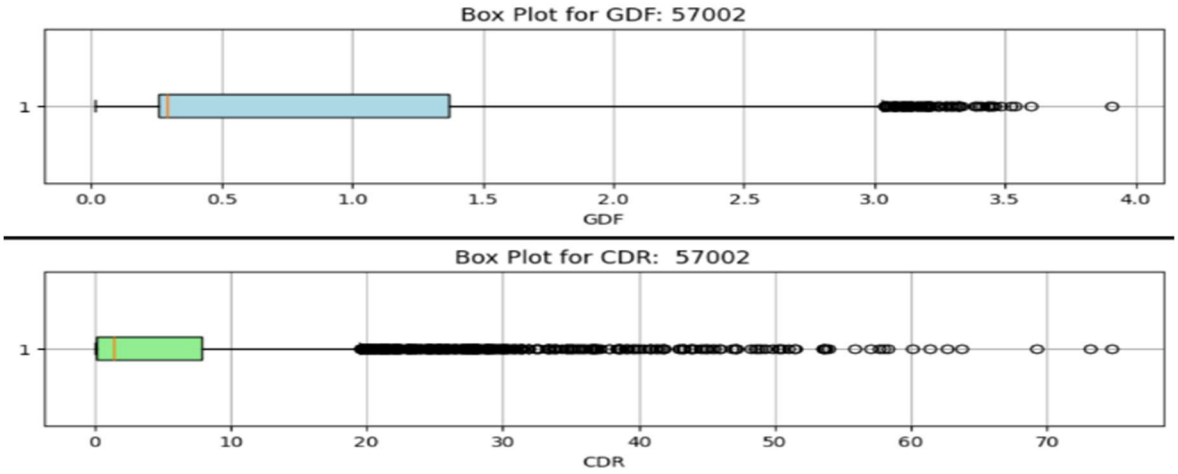

Figure:4.6

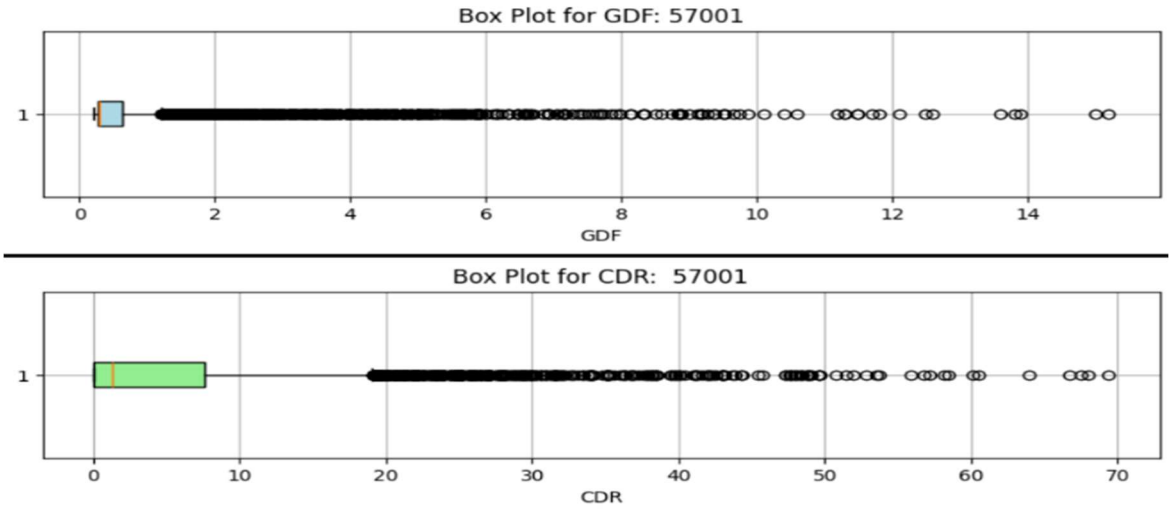

Figure:4.7

3.2.7 Combined all dataset in single dataframe

To carry out further analyses and for the purpose of efficient computation, each of the gauges' data set was merged into one final data frame. This process excluded Gauge 57017 given that it only had limited data to offer which, in effect, might have well skewed the results.

Two approaches were considered for this consolidation:

Direct Merge by 'Date': This method would have involved merging datasets on the date column, producing a dataframe with a substantial increase in rows, thus extending processing times.

Optimized Column Renaming and Merging: To mitigate the computational burden, each dataset's cdr and gdf columns were renamed to include their respective gauge identifiers. The gauge column was then omitted from each data set followed by merging the DataFrames using the inner join where the 'date' was the key column used in merging. This approach resulted in lowering several rows and facilitated the management of data.

This gave a merged dataframe as the final output with the data being processed without any duplicity thus making the processing time efficient. (Figure:5.1, Page 27) below shows the final dataframe structure that was important for the subsequent Exploratory Data Analysis (EDA) or model building.

date	cdr_57015	gdf_57015	cdr_57002	gdf_57002	cdr_57006	gdf_57006	cdr_57007	gdf_57007	cdr_57004	gdf_57004	cdr_57005	gdf_57005	cdr_57001	gdf_57001
2006-05-26	3.3	11.80	3.5	2.480	6.6	12.80	2.9	20.10	3.1	11.590	4.1	55.1	3.5	2.970
2006-05-27	6.5	8.62	6.4	2.370	8.6	10.20	5.9	15.60	5.9	7.969	6.5	41.6	7.0	2.400
2006-05-28	0.5	4.54	0.5	1.110	0.5	7.11	0.4	9.87	0.4	5.069	0.4	27.4	0.7	1.340
2006-05-29	1.1	2.99	1.3	0.591	1.5	5.63	0.8	7.32	1.1	3.902	1.0	20.8	1.1	0.762
2006-05-30	0.0	2.13	0.0	0.358	0.0	4.56	0.0	5.89	0.0	3.200	0.0	16.2	0.0	0.483
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-12-27	2.2	5.79	2.7	1.600	1.3	9.38	1.7	11.00	1.4	6.390	1.5	34.5	2.2	1.170
2017-12-28	18.2	3.77	18.2	0.986	22.3	9.57	17.5	8.28	19.7	5.518	19.0	28.5	19.4	0.634
2017-12-29	8.4	17.40	8.9	2.950	13.3	19.30	8.7	24.40	10.2	14.560	10.5	73.4	8.3	2.700
2017-12-30	24.6	29.80	25.5	3.140	23.5	19.30	24.1	40.10	24.6	16.770	24.1	89.4	26.5	4.640
2017-12-31	17.7	21.50	19.2	2.950	11.0	17.60	14.7	34.10	13.6	17.450	13.3	87.7	18.9	3.690

Figure:5.1

3.3 Exploratory Data Analysis

The Exploratory Data Analysis (EDA) phase was conducted to gain a deeper understanding of the relationships and patterns present within the dataset, which comprises Gauged Daily Flow (GDF) and Catchment Daily Rainfall (CDR) across multiple gauges. This analysis aimed to uncover the underlying structure of the data, identify relation between flow, rainfalls, distribution patterns, identify significant trends, and assess the correlations between various hydrological variables.

3.3.1 Relation of daily flow and daily rainfall at various gauges

Scatter plots were used to analyse the relationship between Gauged Daily Flow (GDF) and Catchment Daily Rainfall (CDR) for various gauge stations. The objective of this study was to assess cases of any discernible relationship or trends that reflect the dependence of the flow of the river within the catchment area on rainfall.

(Figures:6.1-6.7, Page:28-34) presents the scatter plots of the daily rainfall that was recorded at each gauge and the river flows. Generally, higher rainfall was associated with increased river flow, although the relationship exhibited considerable variability, particularly at higher rainfall levels. This non-linear relationship indicates that other hydrological processes are at work in the system including runoff production, rate of water infiltration into the soil, and effects of the tributaries upstream.

The scatter plots also helped identify that one particular gauge demonstrated a much tighter relation between the rainfall and the river flow, while other gauges same type of more scattered relationship. This variability may suggest that there are differences in the extents to which different parts of the catchment are reliant on rainfall.

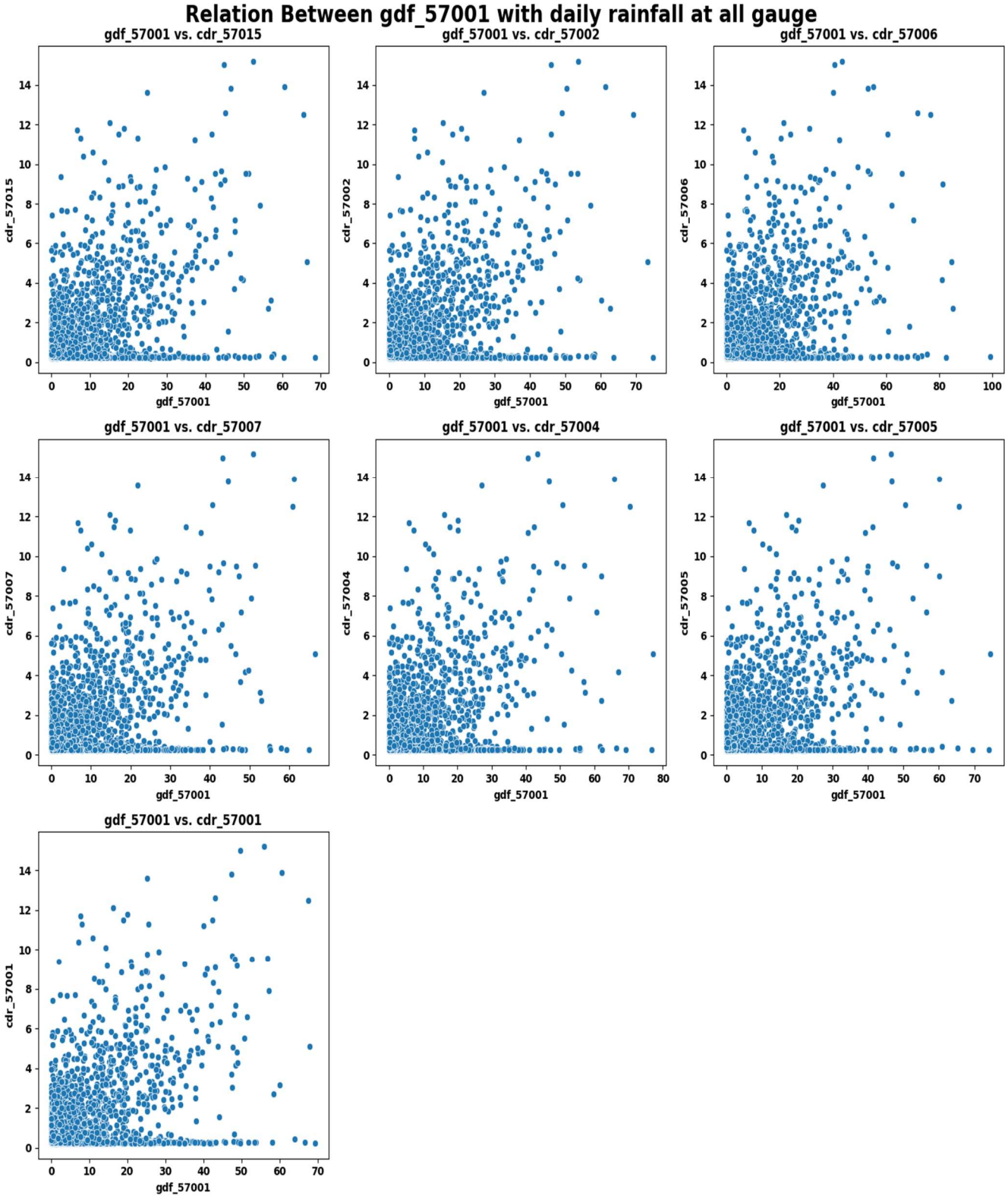
Relation Between gdf_57001 with daily rainfall at all gauge
gdf_57001 vs. cdr_57015
gdf_57001 vs. cdr_57002
gdf_57001 vs. cdr_57006
gdf_57001 vs. cdr_57007
gdf_57001 vs. cdr_57004
gdf_57001 vs. cdr_57005
gdf_57001 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57001

Figure:6.1

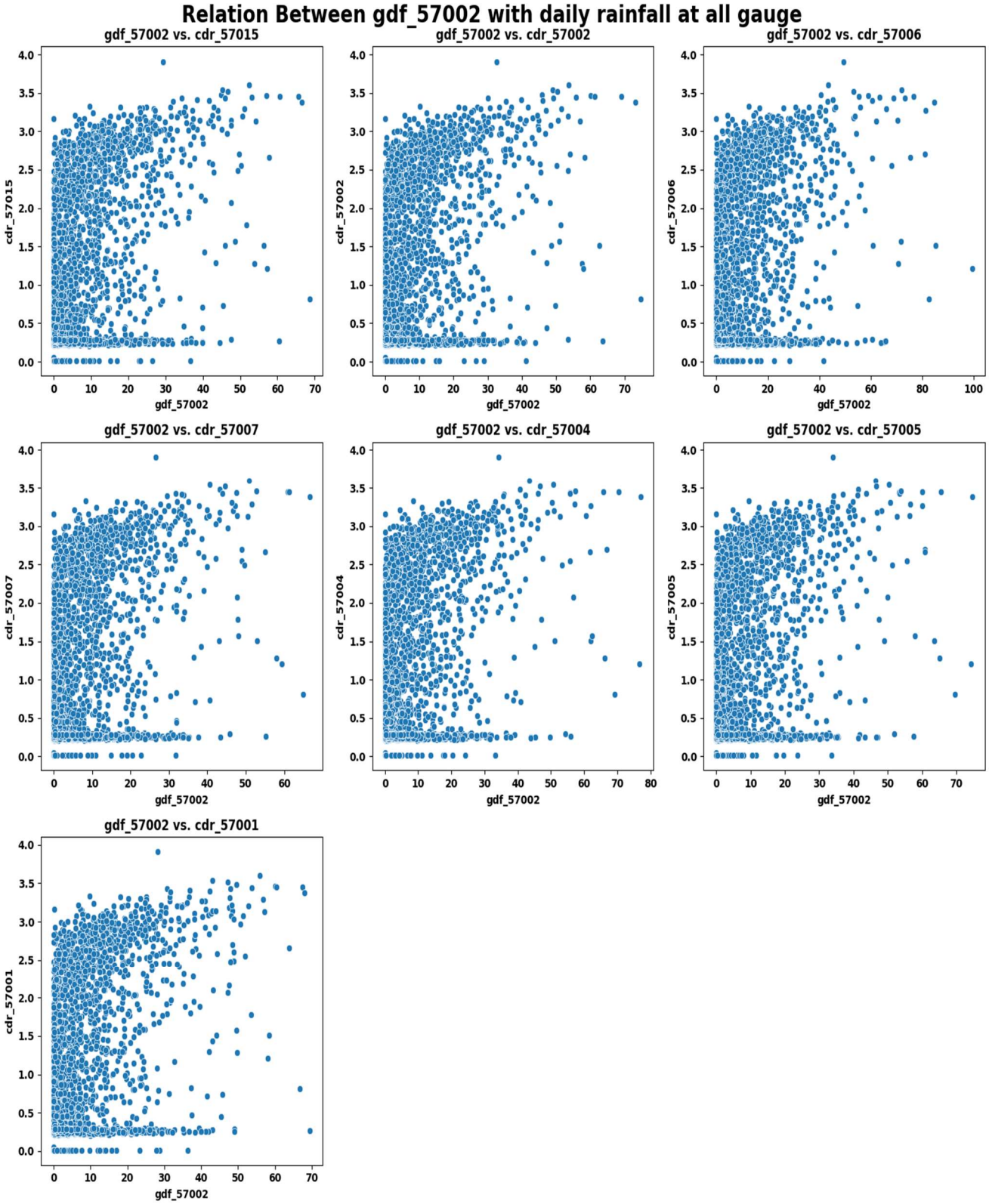
Relation Between gdf_57002 with daily rainfall at all gauge
gdf_57002 vs. cdr_57015
gdf_57002 vs. cdr_57002
gdf_57002 vs. cdr_57006
gdf_57002 vs. cdr_57007
gdf_57002 vs. cdr_57004
gdf_57002 vs. cdr_57005
gdf_57002 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57002

Figure:6.2

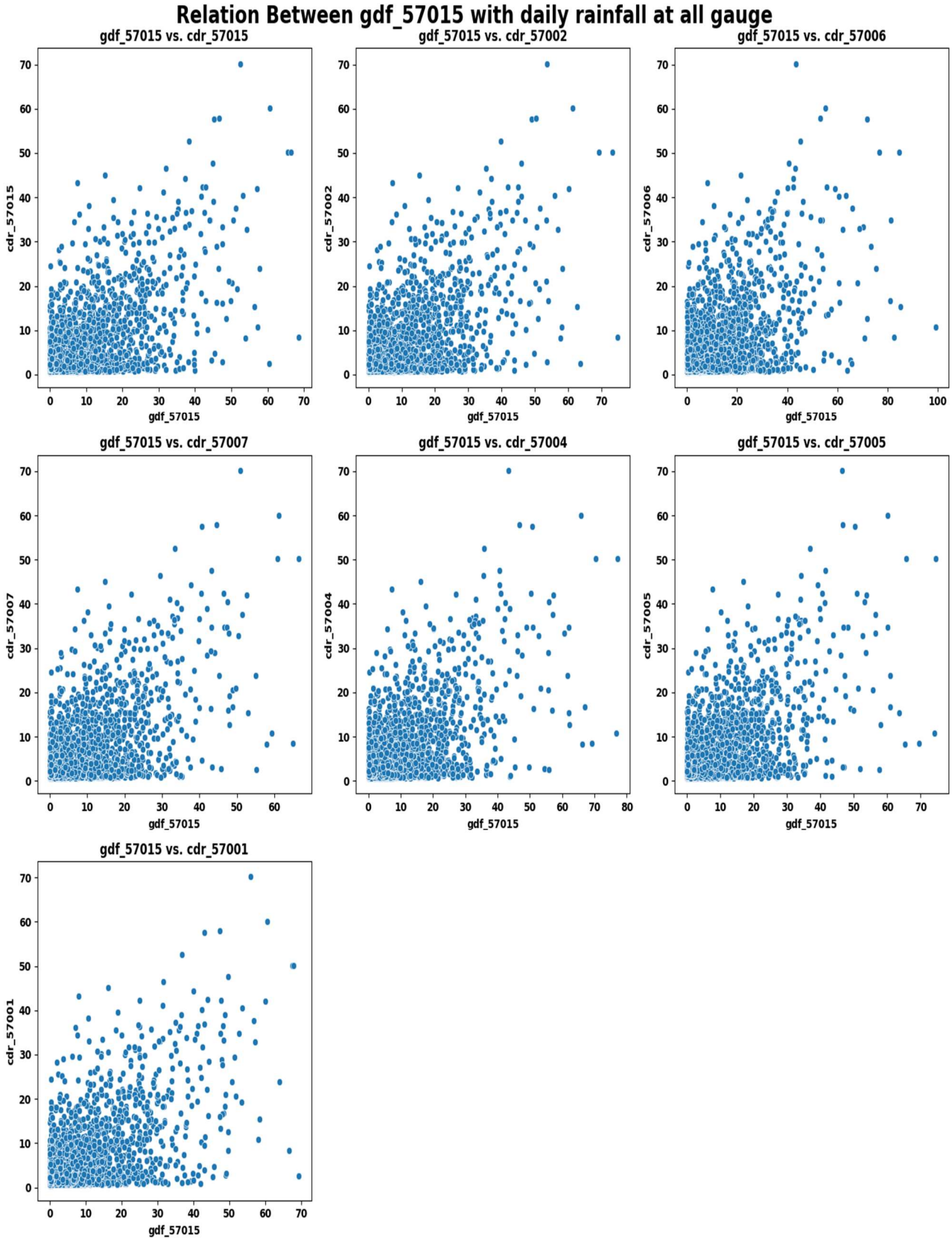
Relation Between gdf_57015 with daily rainfall at all gauge
gdf_57015 vs. cdr_57015
gdf_57015 vs. cdr_57002
gdf_57015 vs. cdr_57006
gdf_57015 vs. cdr_57007
gdf_57015 vs. cdr_57004
gdf_57015 vs. cdr_57005
gdf_57015 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57015

Figure:6.3

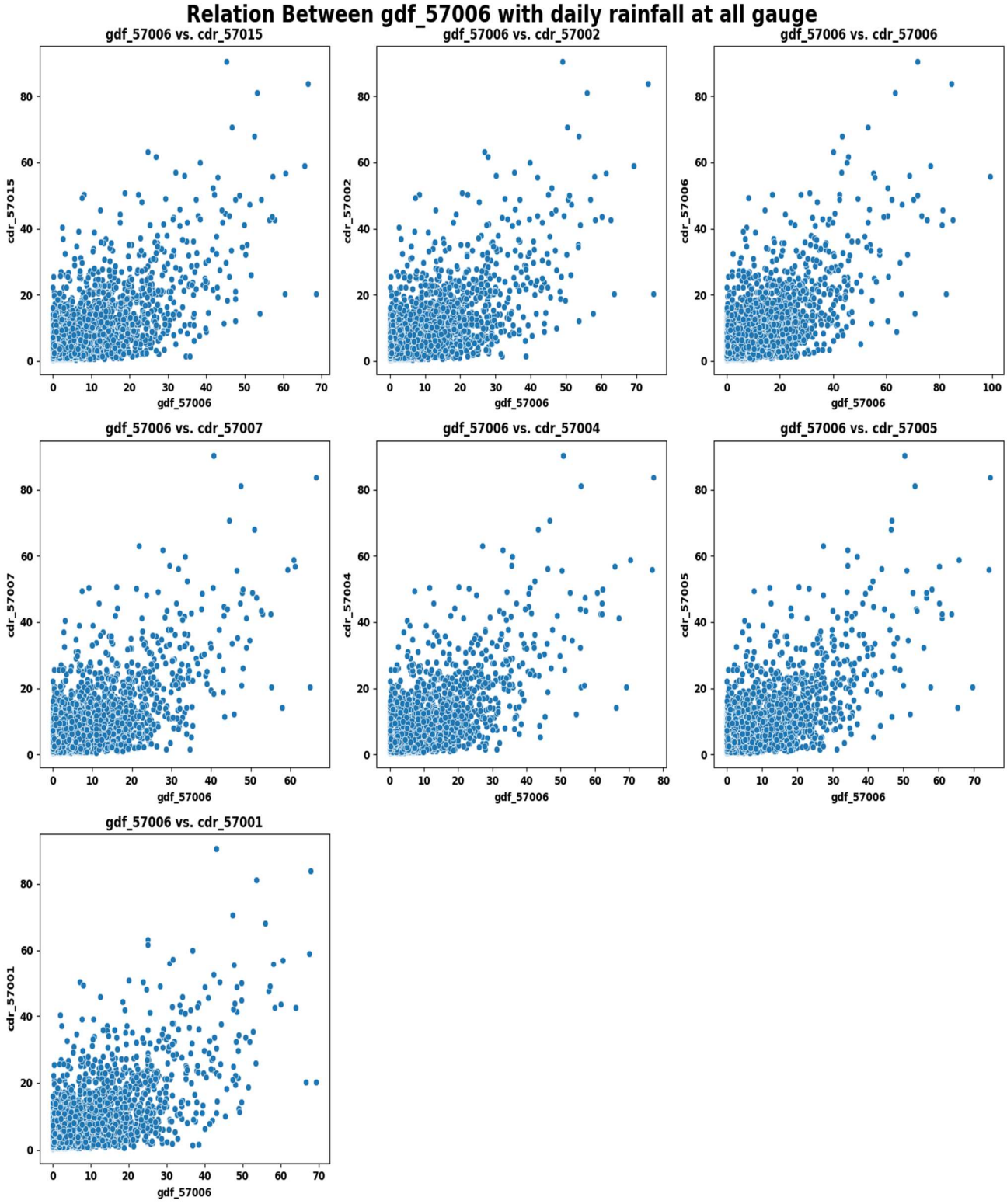
Relation Between gdf_57006 with daily rainfall at all gauge
gdf_57006 vs. cdr_57015
gdf_57006 vs. cdr_57002
gdf_57006 vs. cdr_57006
gdf_57006 vs. cdr_57007
gdf_57006 vs. cdr_57004
gdf_57006 vs. cdr_57005
gdf_57006 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57006

Figure:6.4

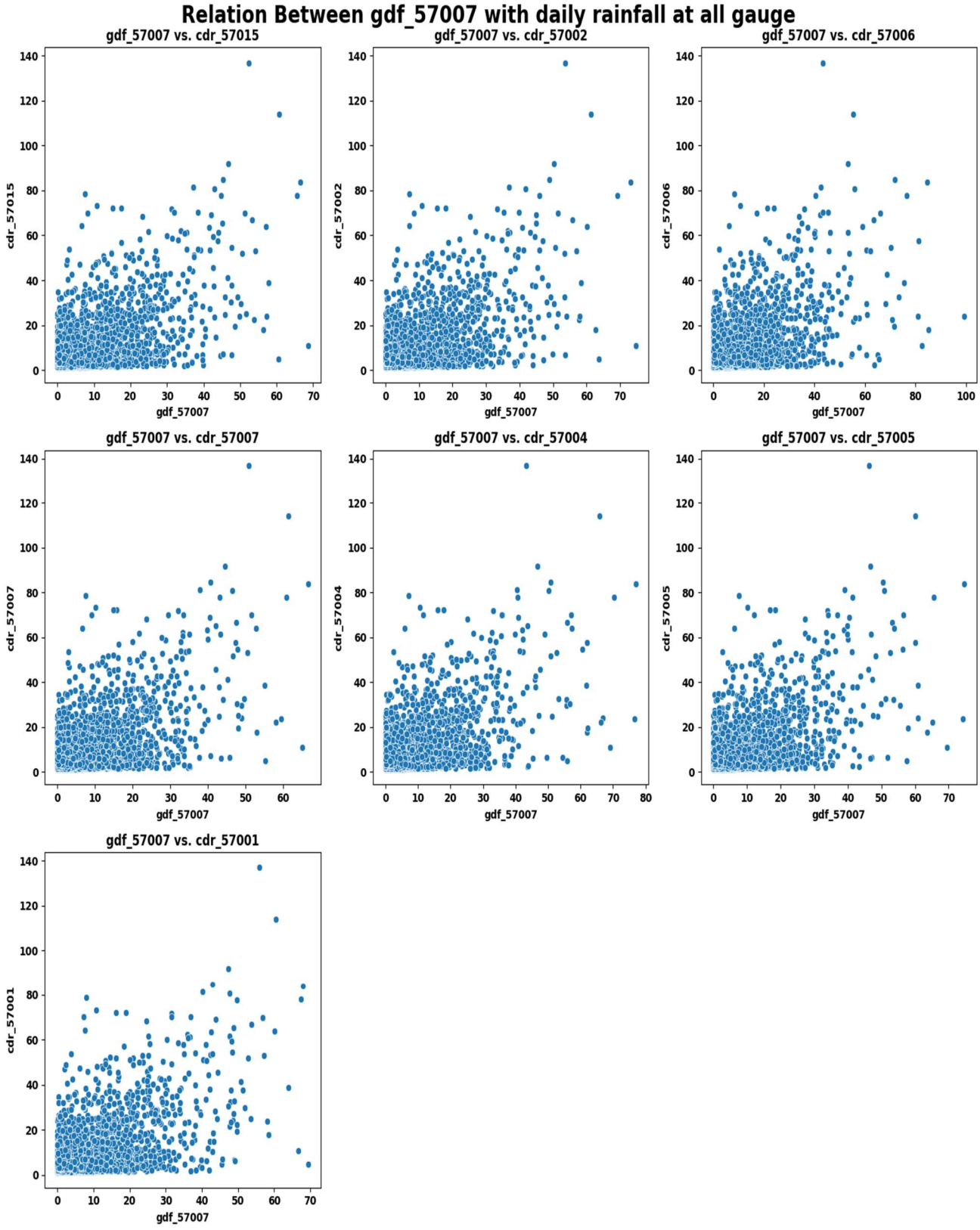
Relation Between gdf_57007 with daily rainfall at all gauge
gdf_57007 vs. cdr_57015
gdf_57007 vs. cdr_57002
gdf_57007 vs. cdr_57006
gdf_57007 vs. cdr_57007
gdf_57007 vs. cdr_57004
gdf_57007 vs. cdr_57005
gdf_57007 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57007

Figure:6.5

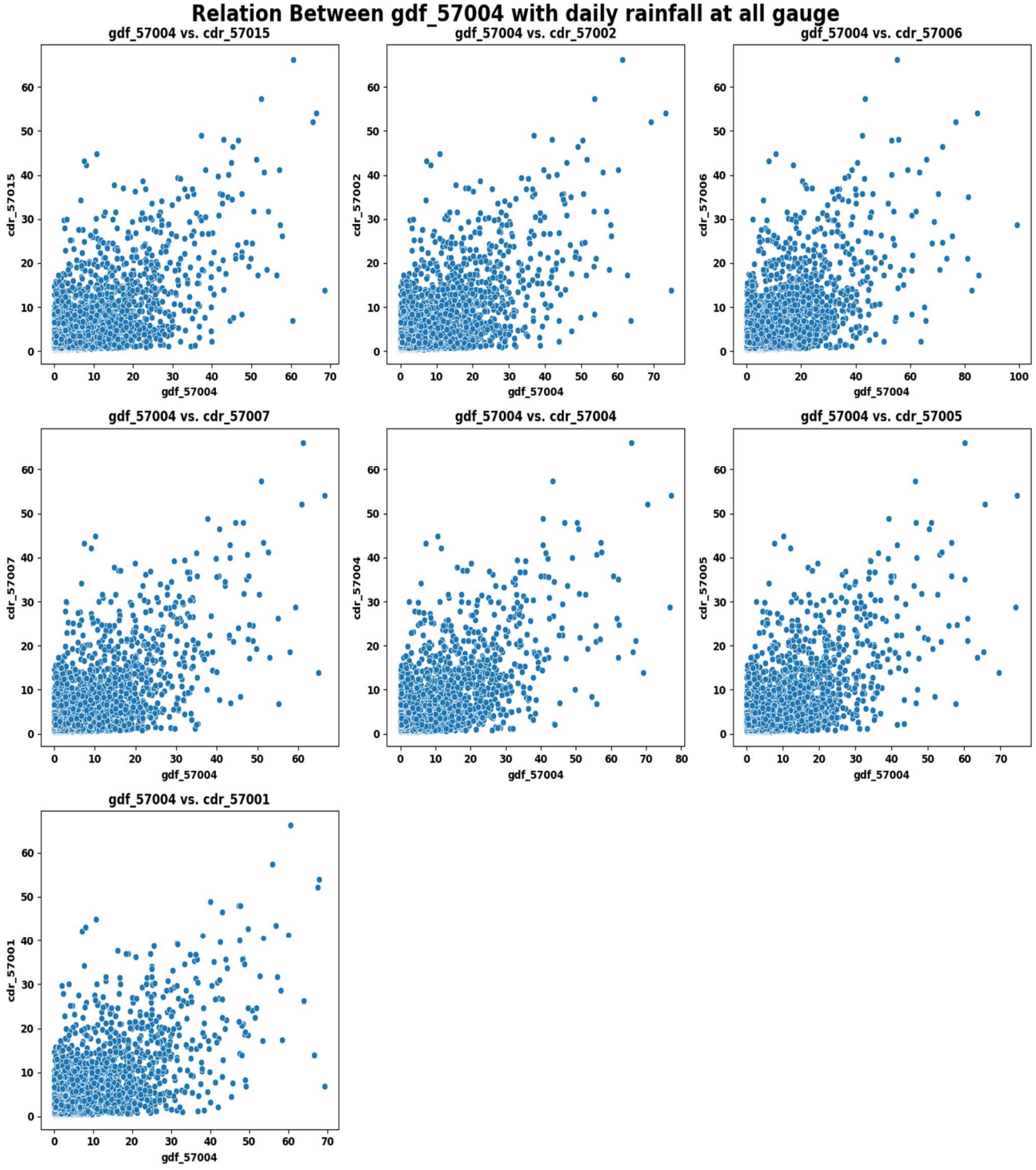
Relation Between gdf_57004 with daily rainfall at all gauge
gdf_57004 vs. cdr_57015
gdf_57004 vs. cdr_57002
gdf_57004 vs. cdr_57006
gdf_57004 vs. cdr_57007
gdf_57004 vs. cdr_57004
gdf_57004 vs. cdr_57005
gdf_57004 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57004

Figure:6.6

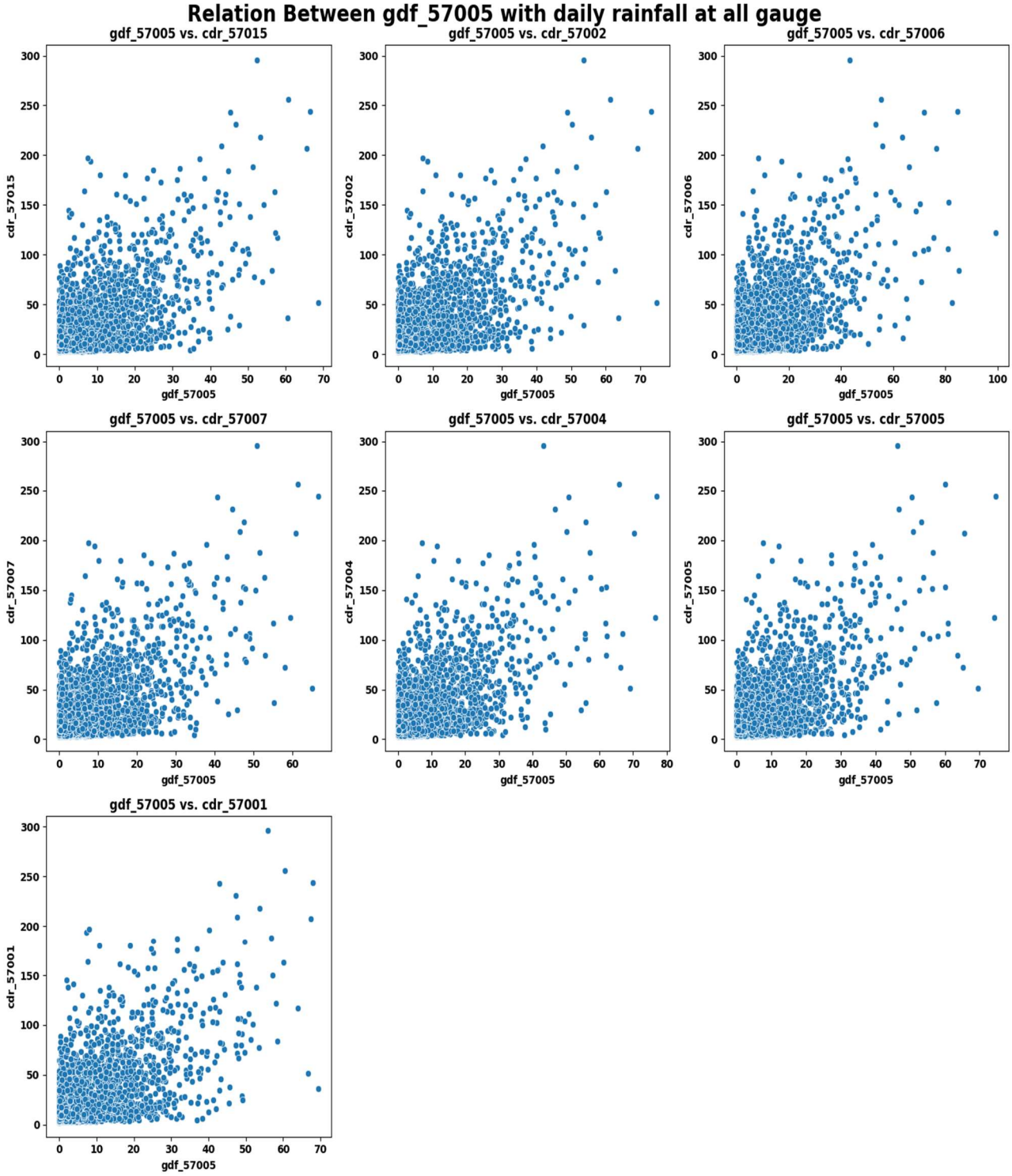
Relation Between gdf_57005 with daily rainfall at all gauge
gdf_57005 vs. cdr_57015
gdf_57005 vs. cdr_57002
gdf_57005 vs. cdr_57006
gdf_57005 vs. cdr_57007
gdf_57005 vs. cdr_57004
gdf_57005 vs. cdr_57005
gdf_57005 vs. cdr_57001
cdr_57015
cdr_57002
cdr_57006
cdr_57007
cdr_57004
cdr_57005
cdr_57001
gdf_57005

Figure:6.7

3.3.2 Relation between daily flow vs Daily across various gauge

To analyse the relationship between daily flow measurements of the several gauges within the catchment, a pair plot was developed with the aim of analysing the relationships of daily flow measurements between different gauges. The pair plot is shown in (Figure:7.1, Page:36).

The scatter plots within the pair plot matrix also show varying degrees of correlation between the measures. Notably:

- Gauge 57005 and gauge 57004 are almost perfectly linear, which means that there is a direct positive relationship between them. This implies that these two gauges with Gauge 57005 being the principal river where next-day flow will be forecasted, and Gauge 57004 as a substream, feel the same hydrological conditions. This relation was important for the forecasting model because it assumed that Gauge 57004 contained substantial predictive information for Gauge 57005.

- As it is also elucidated through Gauge 57007 plot with Gauge 57005 and Gauge 57004, there is a nearly direct correlation between them. Since Gauge 57007 is another tributary, this goes well with the logical assembly of the flow dynamics in this catchment area.

- A similarly positive and approximately linear relationship is also present when comparing Gauge 57015 to Gauge 57007. This suggests that another tributary Gauge 57015 also have similar flow regime as Gauge 57007 and may be important in analysing the total impacts of the tributary flows on the main river.

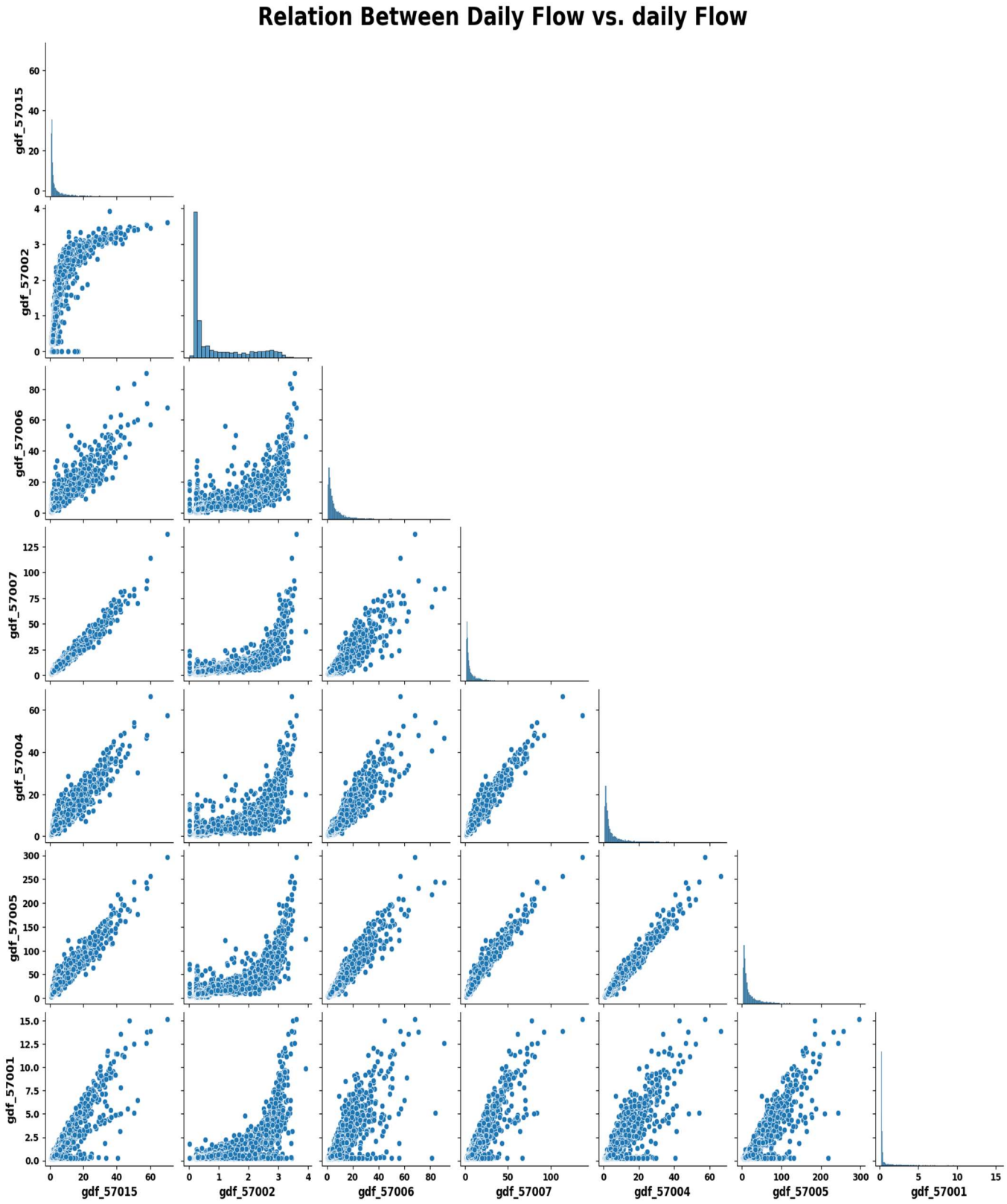
Relation Between Daily Flow vs. daily Flow
gdf_57015
gdf_57002
gdf_57006
gdf_57007
gdf_57004
gdf_57005
gdf_57001

figure 7.1

3.3.3 Correlation Analysis between daily flow and daily Rainfall

To analyse the strength of the correlation between GDF (flow) and CDR (rainfall) over the chosen gauges, a heatmap was created with more emphasis placed on those gauges that are significant in the prediction of the next day's river flow at gauge 57005 (gdf_57005). The heatmap depicts the correlation coefficients where the values are between 0 and 1, with values nearer to 1 representing a strong positive correlation and values nearer to 0 representing a strong weaker correlation.

As presented in the (Figure:8.1, page:37), several key correlations were observed:

- gdf_57005 shows that it is highly correlated to other flow gauges: gdf_57004, gdf_57006, gdf_57007, and gdf_57015. This suggests that flow at these tributaries tends to be involved in the determination of the flow at the main river gauge (57005).
- Moreover, gdf_57001 and gdf_57002 had significant correlations with gdf_57005, but the coefficients were slightly lower compared to the above-mentioned gauges.
- Correlations between CDR (rainfall) and GDF (flow) were comparatively lower, indicating that although rainfall is a useful parameter, the flow relationships between the gauges have a greater influence and potential in predicting river flow at 57005 with better precision.

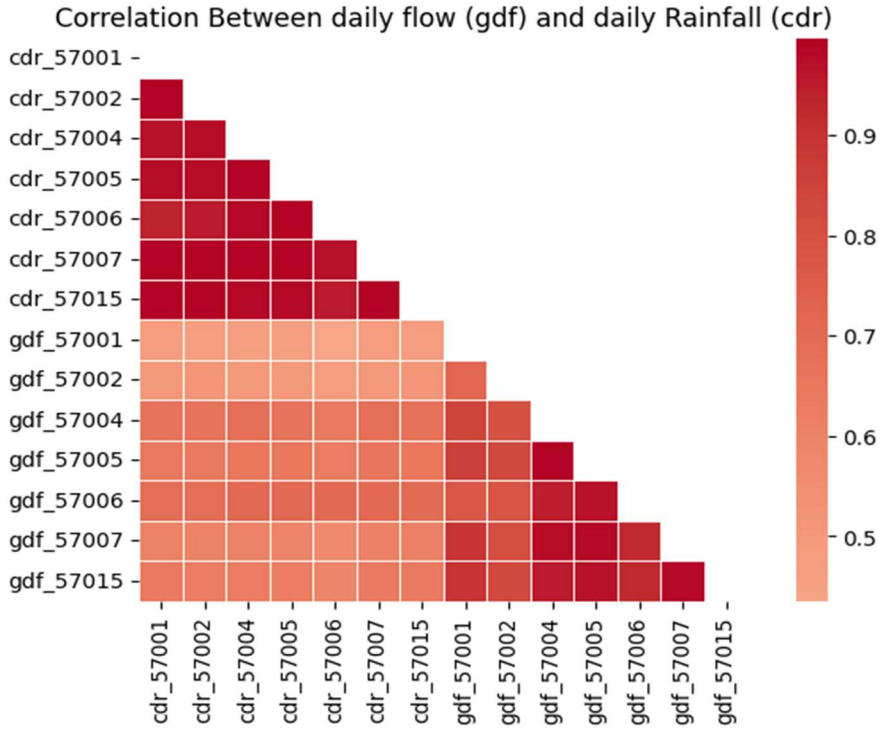

Figure:8.1

3.3.4 Distribution Analysis of Daily flow and Rainfall at Each gauge

To examine the distribution of Gauged Daily Flow (GDF) and Catchment Daily Rainfall (CDR) over all the gauges, the Kernel Density Estimation (KDE) plots were created as presented in (Figures:9.1-9.7, page:39-41). Based on KDE plots provided below, based on KDE plots provided below, GDF and CDR have right skewed characteristics across all the gauges, which agrees with findings from the previous outlier analysis interpreted via box plots.

This skewness is evidence of the influence of rare hydrological occurrence including high rainfall and flows in the rivers. This skewed nature of these distributions served to strengthen the argument for data transformation techniques such as log or Box-Cox transformations, which improve the efficiency of existing models for prediction.

However, it was imperative to understand that the distribution of gdf_57002 was skewed from the other gauges. As for the other gauges, their distribution looked as if it has a long tail on the high flow side whereas gdf_57002 was significantly fewer high values and a very tight cluster around the low flow values. This pattern indicated that the observed flow characteristics at gauge 57002 might be related to different hydrological conditions or, possibly, to localized factors which was different from the conditions affecting the rest of the gauges and should probably be investigated during the modelling phase.

The CDR frequency distributions reveal a high percentage of low rainfall days specifically, near zero which was characteristic for temperate climate regions that experience many dry days. The same can be said for the GDF distributions, which also show a high frequency of low flows values and a lesser number of near high flows suggesting that, as with most rivers, most days was generally characterized by normal, low flows with high flows being infrequent events.

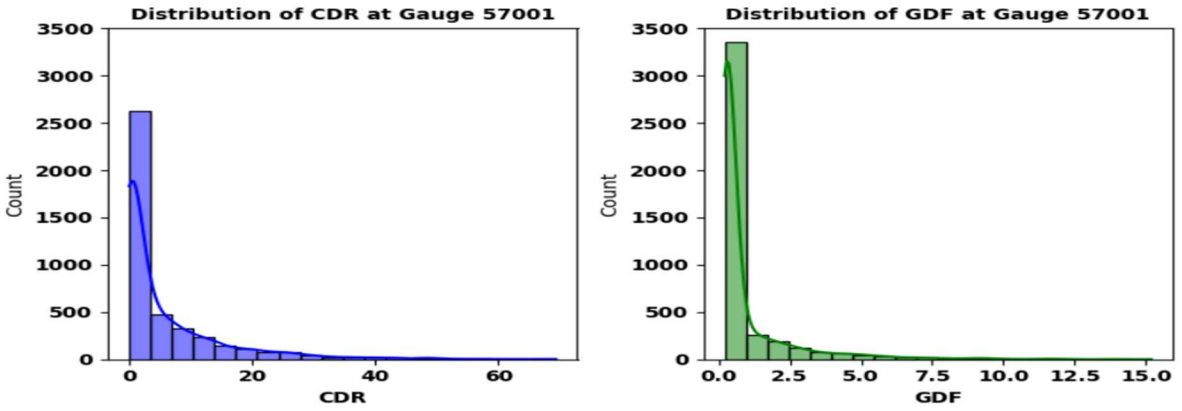

Figure:9.1

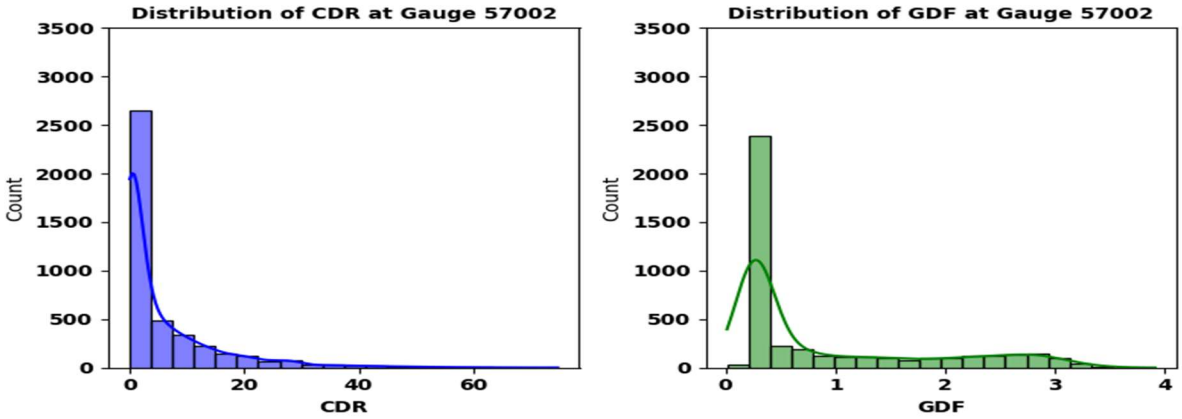

Figure:9.2

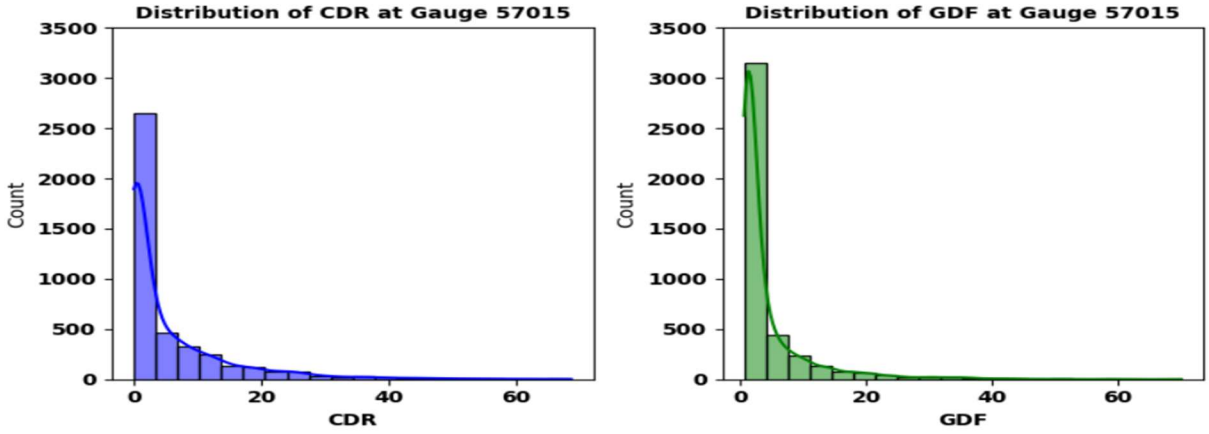

Figure:9.3

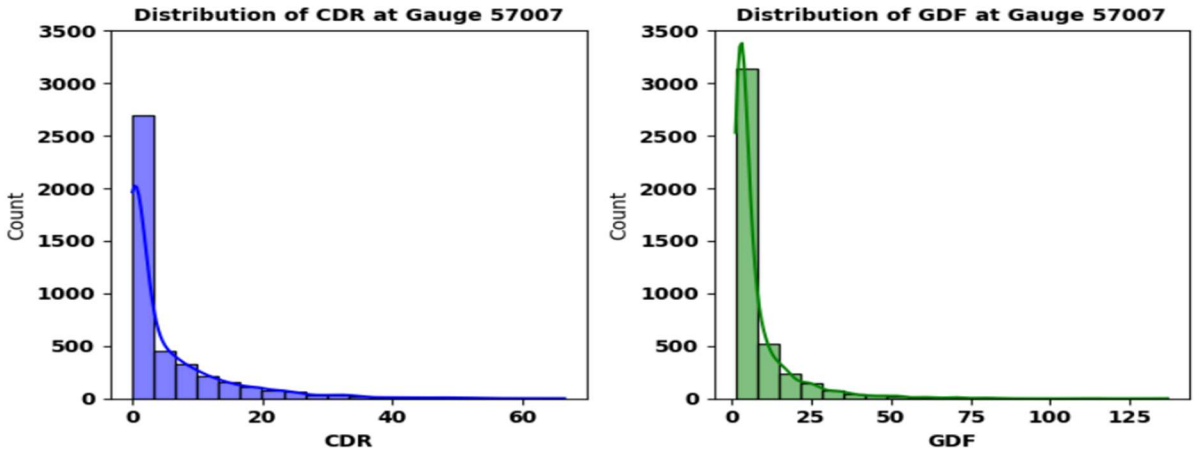

Figure:9.4

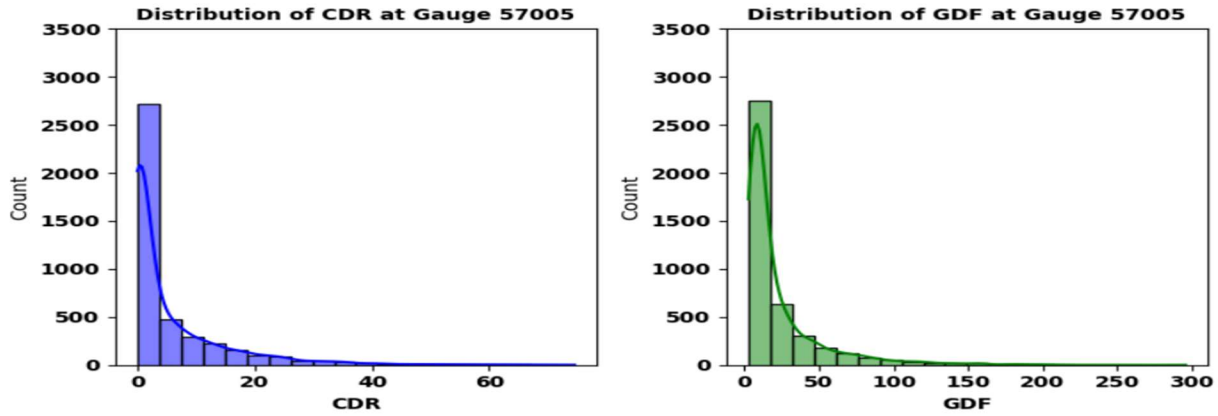

Figure:9.5

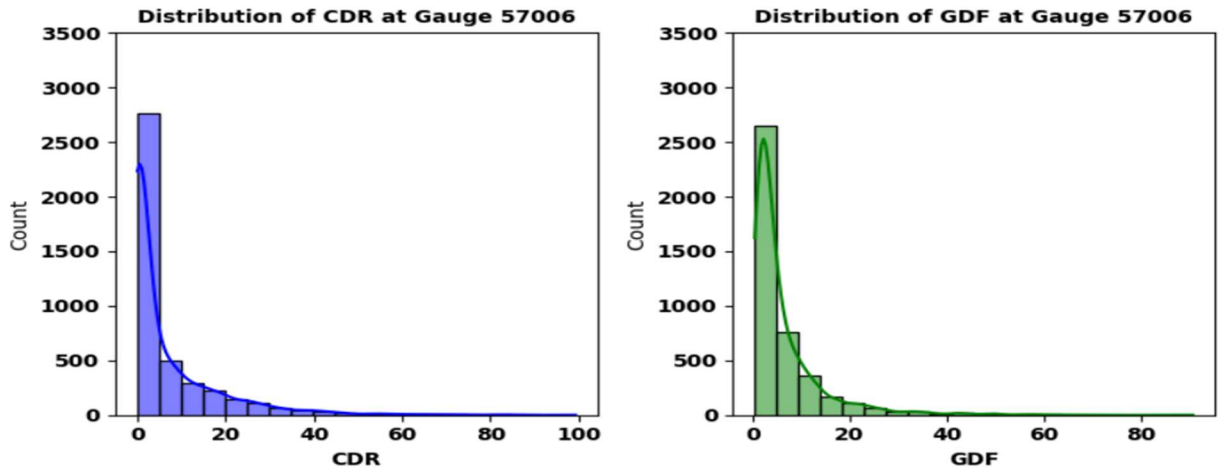

Figure:9.6

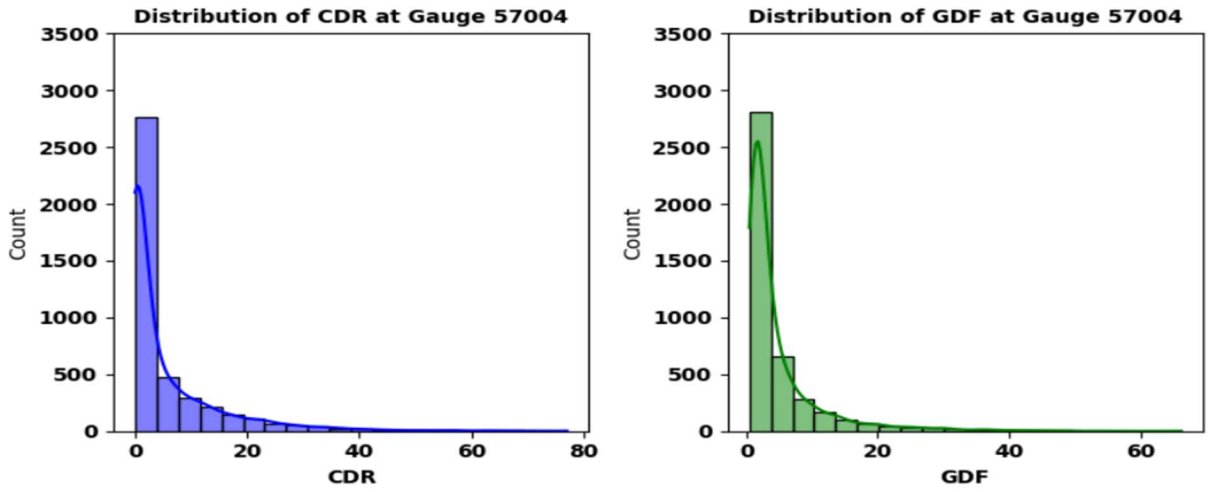

Figure:9.7

3.4 Data Transformation

As a result of skewness in the distribution of GDF and CDR the data was transformed to improve normality as this is often necessary for many statistical models.

3. 4. 1 Log Transformation

The first technique with which Log Transformation was employed. For GDF, the transformation was done using the log (x) tool, that is log to the base 10. As for CDR, the data contained zero values, so a small constant was added to the values followed by transformation log(1+x). This was done with intent to minimize right skewness of data which is crucial for model such as Linear Regression.

This is illustrated in (Figure:10.1-10.7, page:42-44) where application of log transformation has generally led to reduction in right skewness and made the distributions more normal. This adjustment is necessary since a lot of predictive models work better when the input data has a normal distribution.

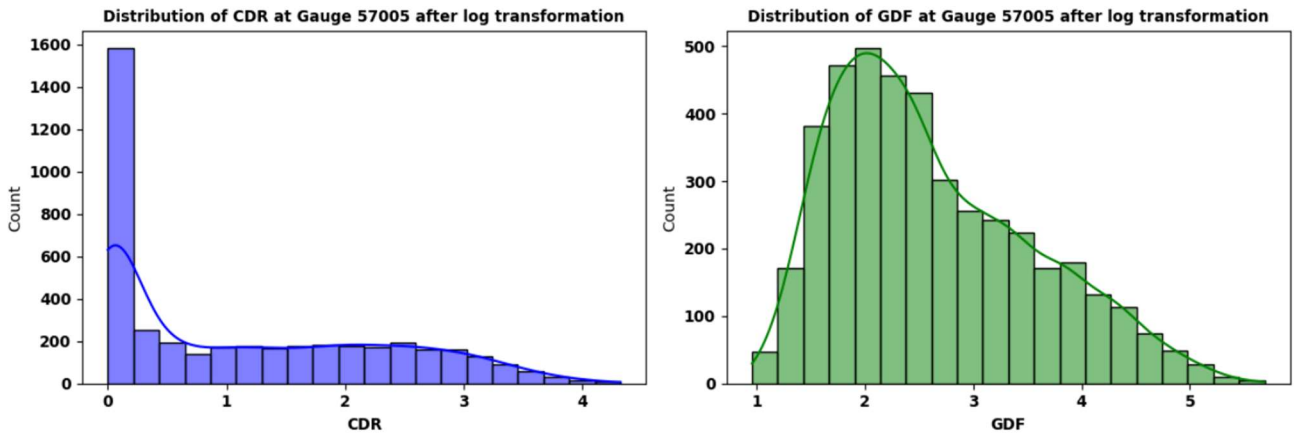

Figure:10.1

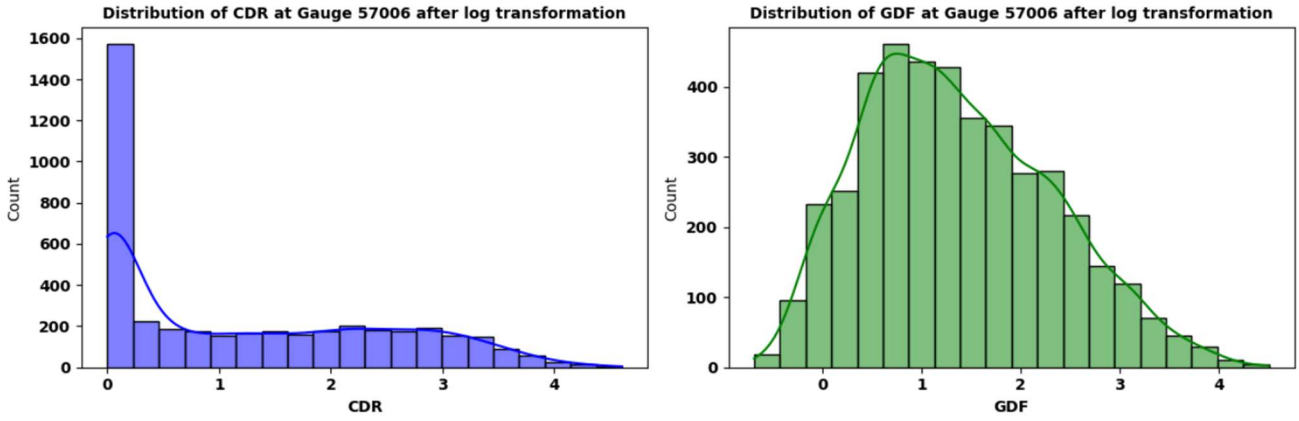

Figure:10.2

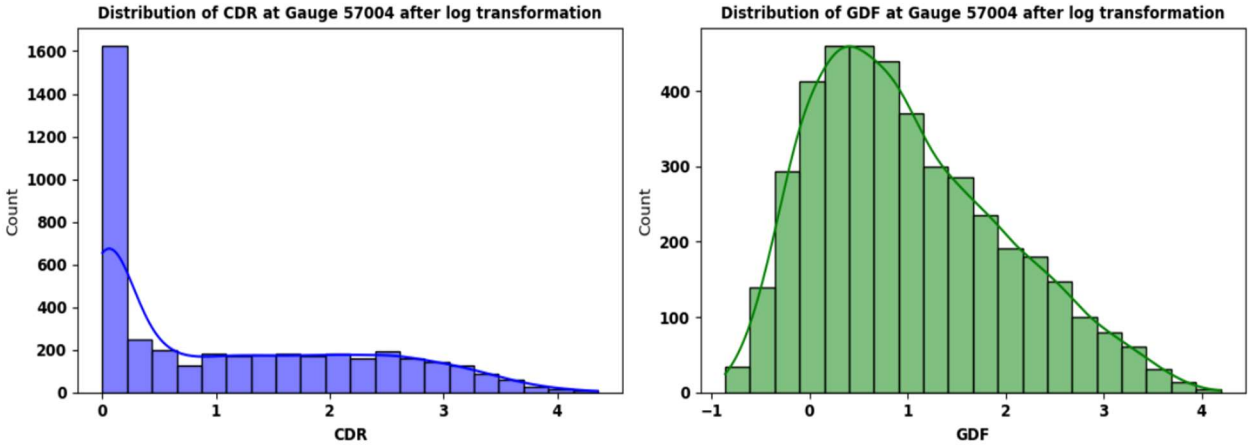

Figure:10.3

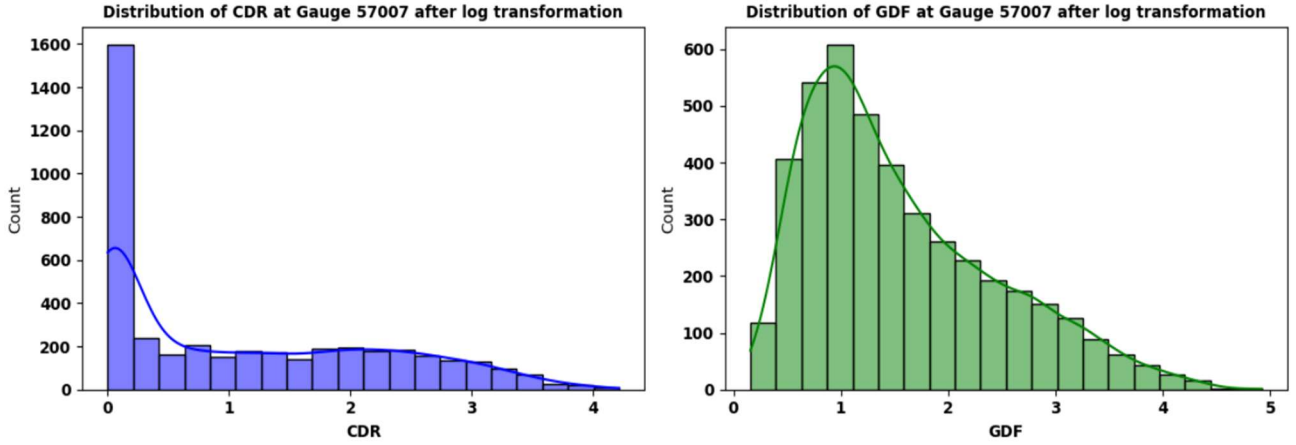

Figure:10.4

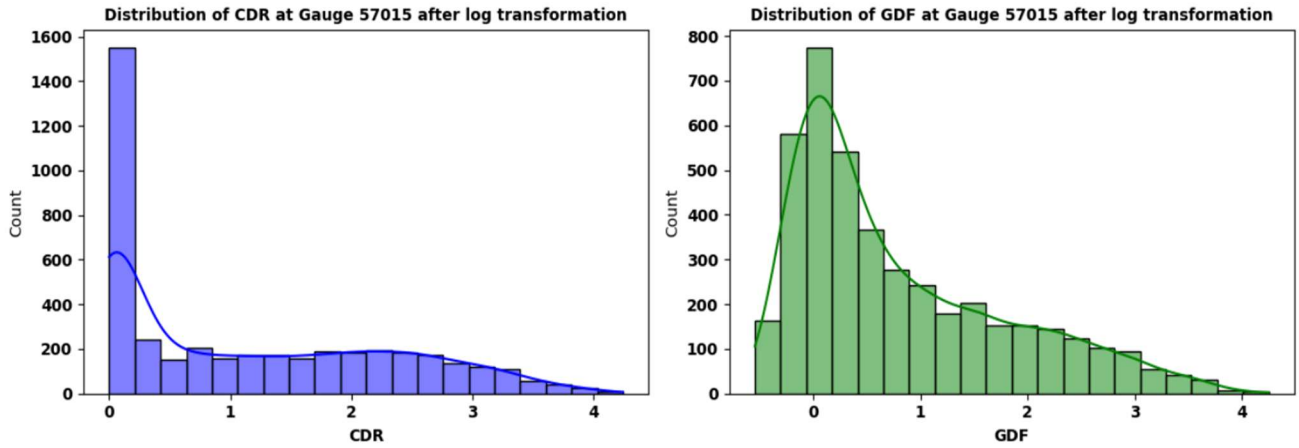

Figure:10.5

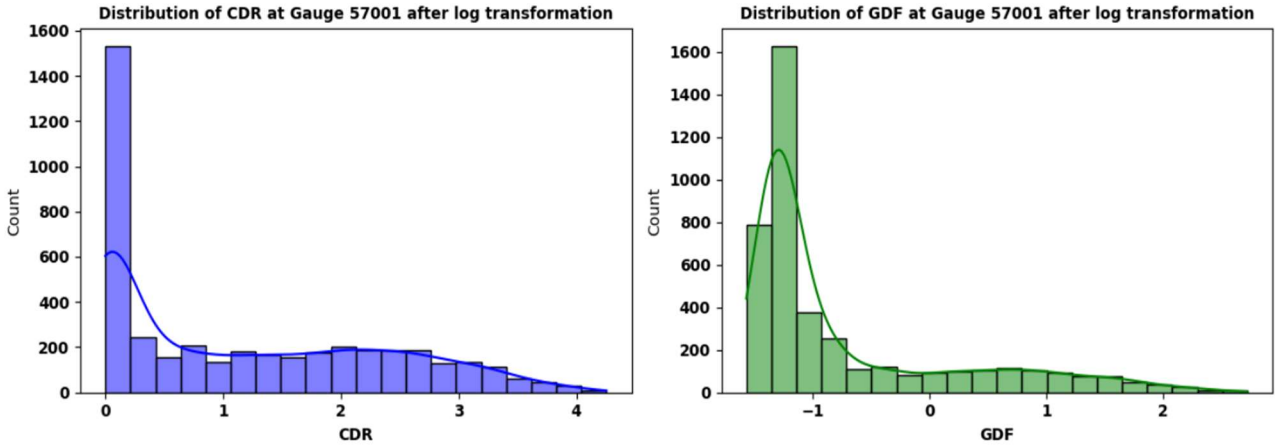

Figure:10.6

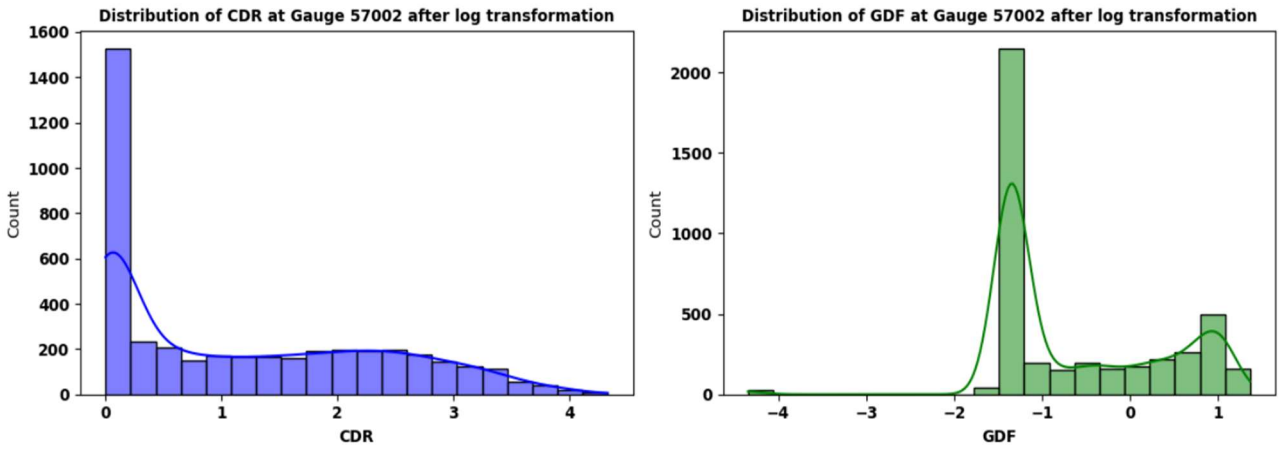

Figure:10.7

3.4.2 Box-Cox Transformation

Besides log transformation, the Box-Cox transformation was applied to the cleaned data to stabilize variance and achieve a more normal distribution, particularly for non-negative data. It is defined as:

$$y(\lambda) = y(\lambda) = \frac{y^{\lambda}-1}{\lambda}, for\ \lambda \neq 0$$

λ: parameter that determines the nature of the transformation. When λ=0, the transformation becomes equivalent to the natural logarithm.

The Box-Cox transformation offers flexibility by adjusting λ to shape the data distribution. The goal of applying these transformations was to make the data more normally distributed, enhancing the reliability of statistical models. Distribution plots in (Figures:11.1-11.7, page:45-47) show significant improvement in GDF symmetry across gauges 57005, 57006, 57004, and 57007.

Although the Box-Cox transformation improved the shape of the CDR distributions, it did not entirely eliminate the skewness, though it made it less pronounced. This step was crucial for improving the suitability of the data for statistical analysis.

Box plots created after the Box-Cox transformation (Figures:12.1-12.7, page:47-49) further illustrate the reduction in outliers and the shrinking of whiskers, highlighting the mitigation of extreme values.

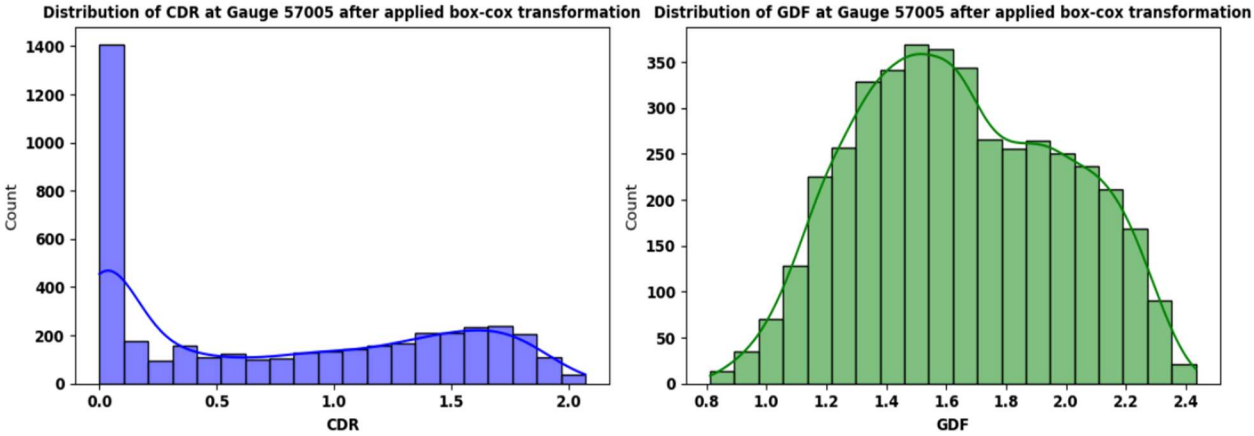

Figure:11.1

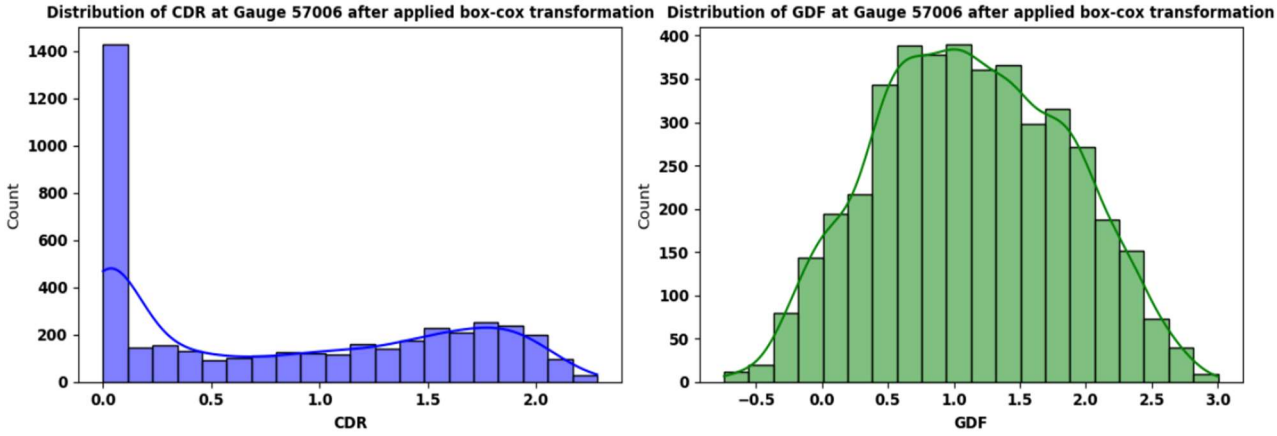

Figure:11.2

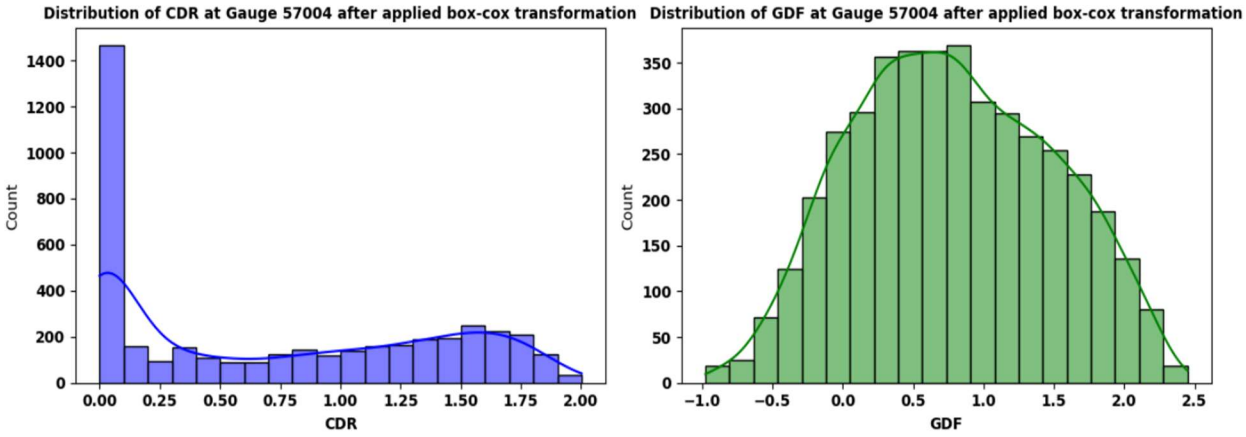

Figure:11.3

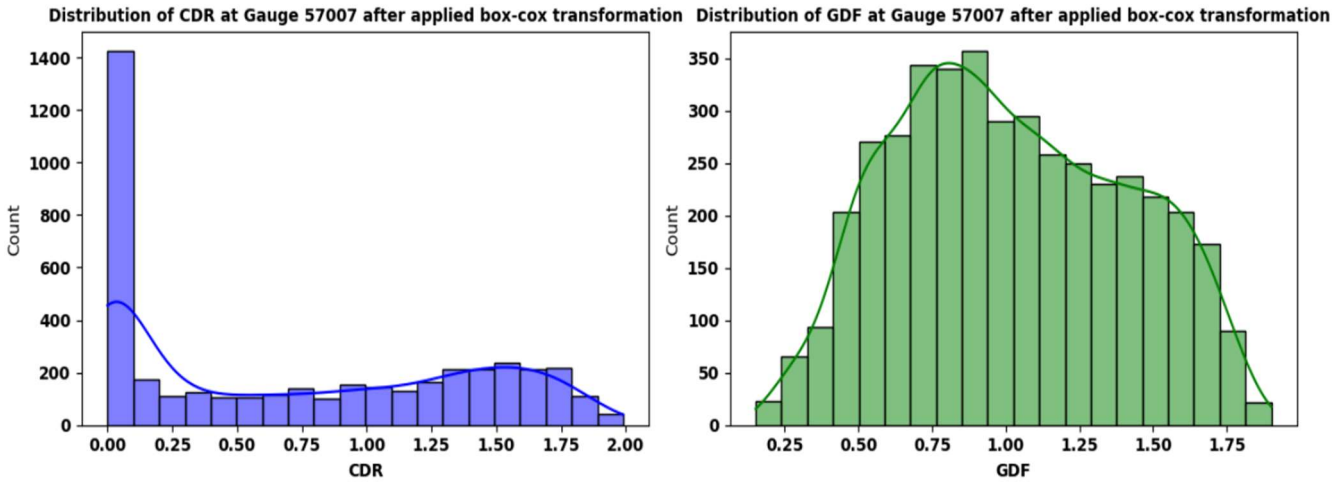

Figure:11.4

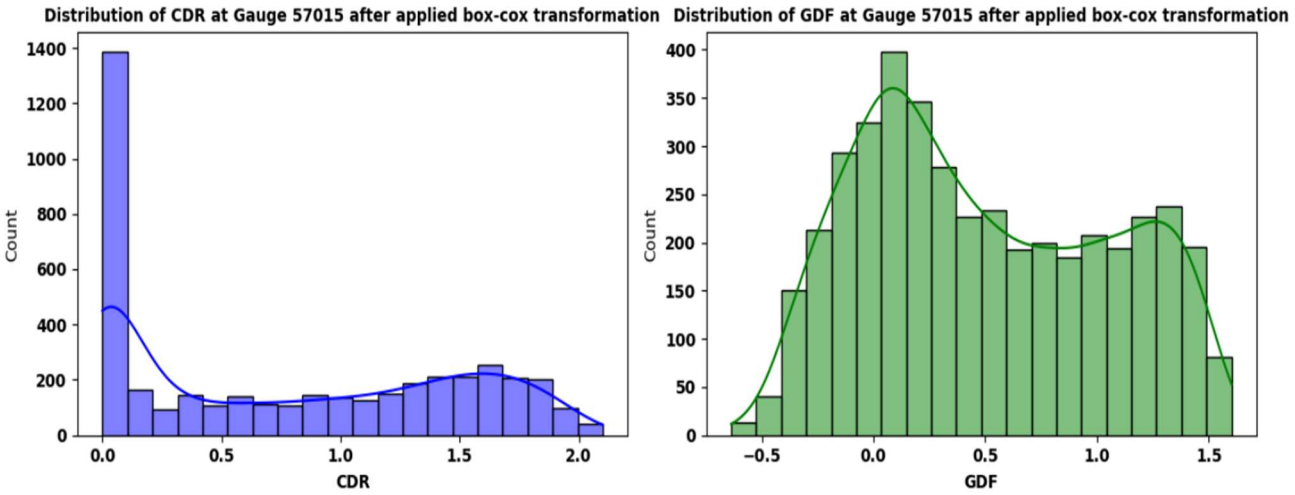

Figure:11.5

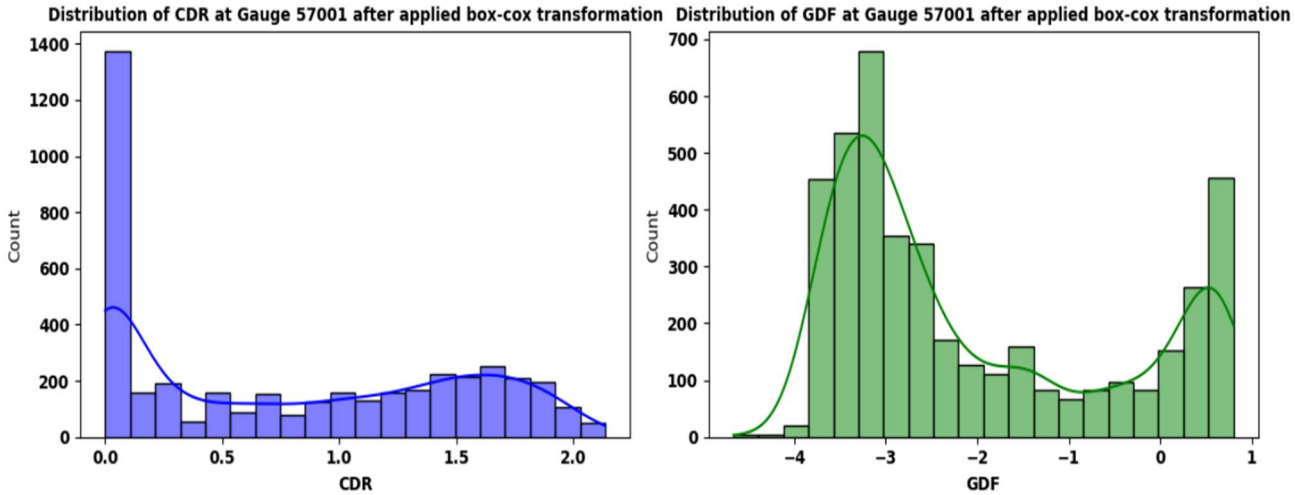

Figure:11.6

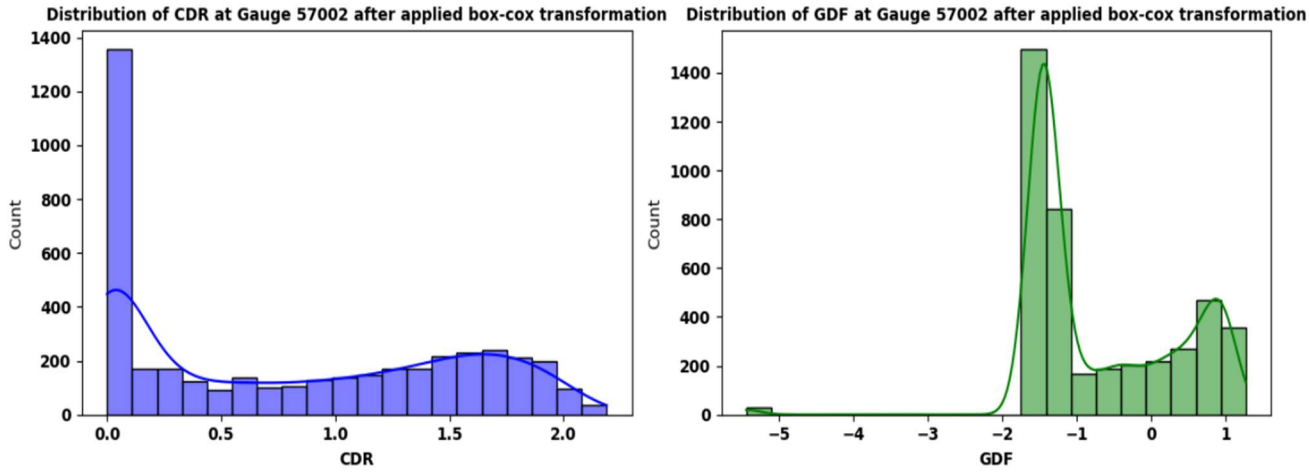

Figure:11.7

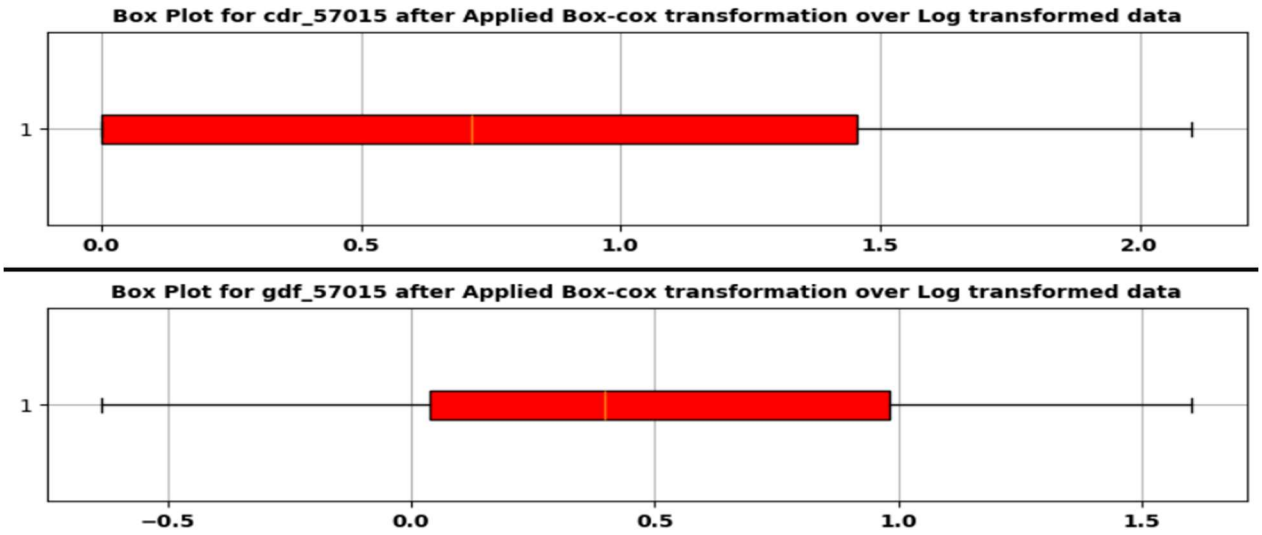

Figure:12.1

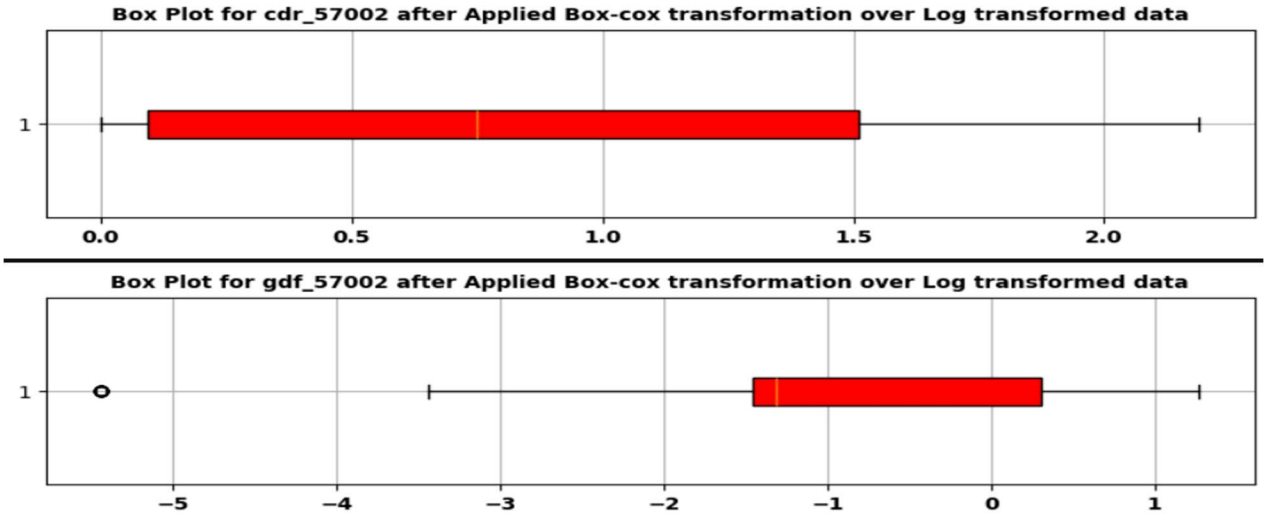

Figure:12.2

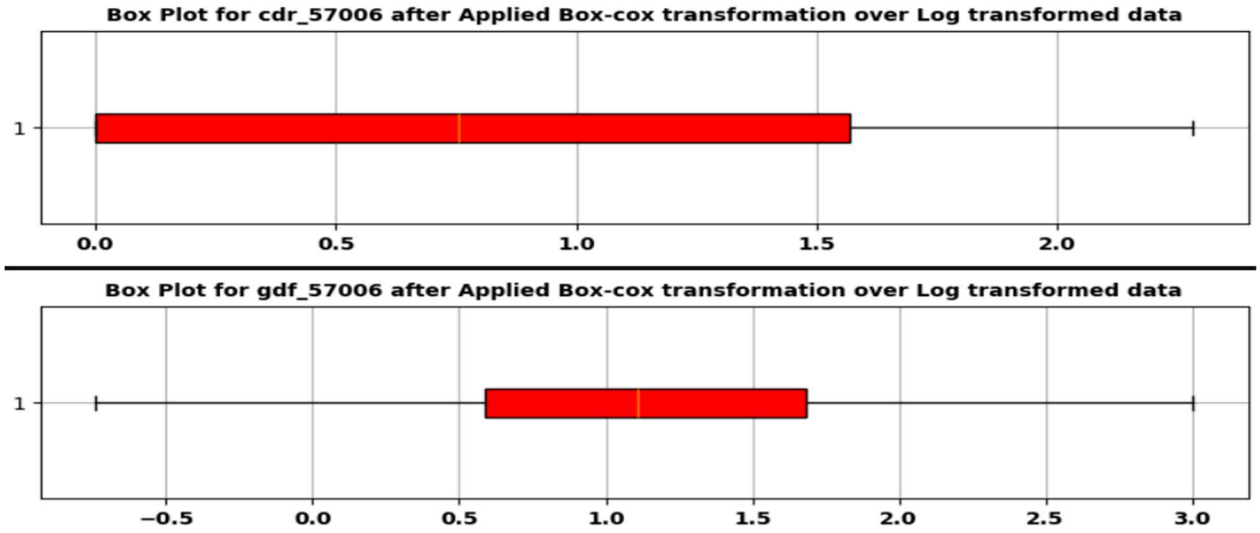

Figure:12.3

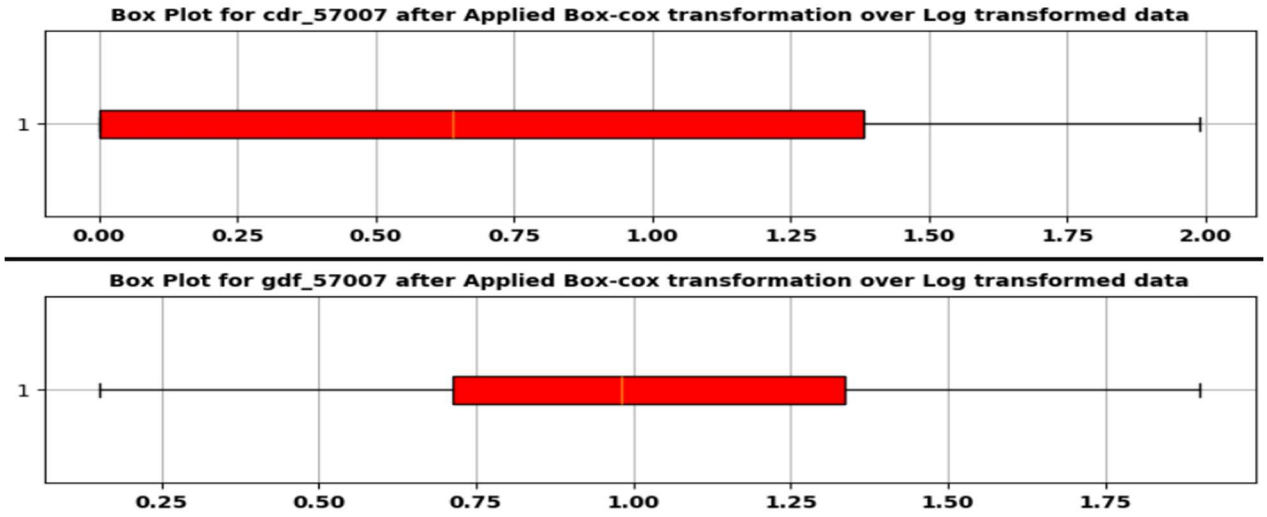

Figure:12.4

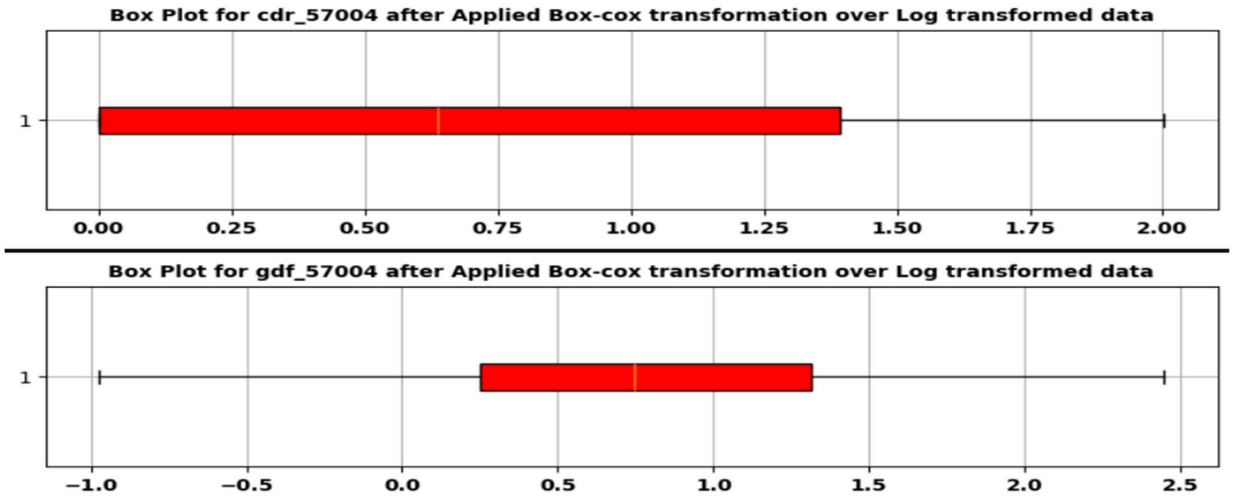

Figure:12.5

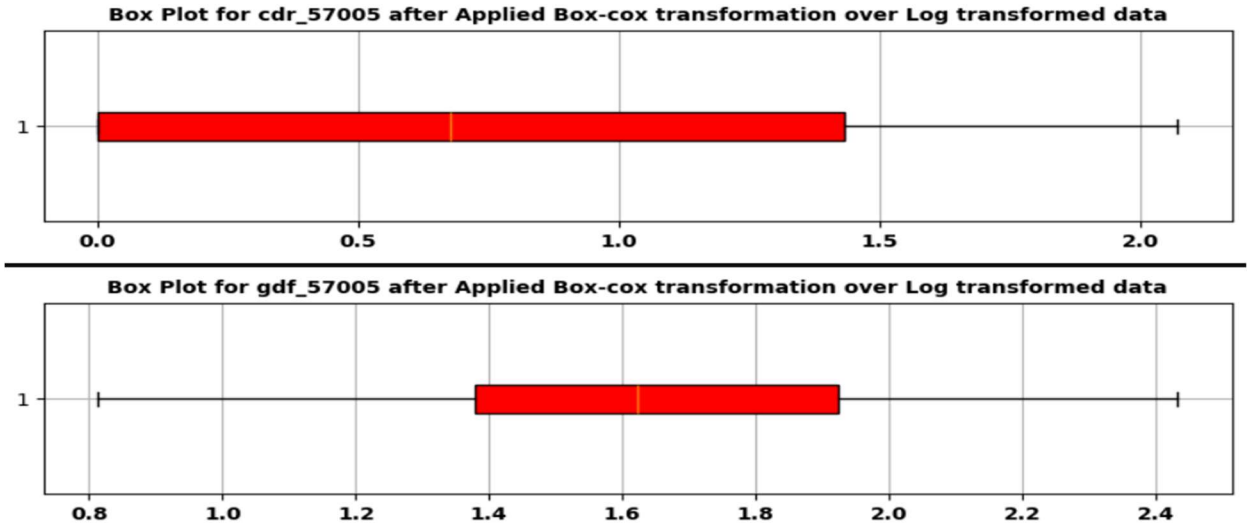

Figure:12.6

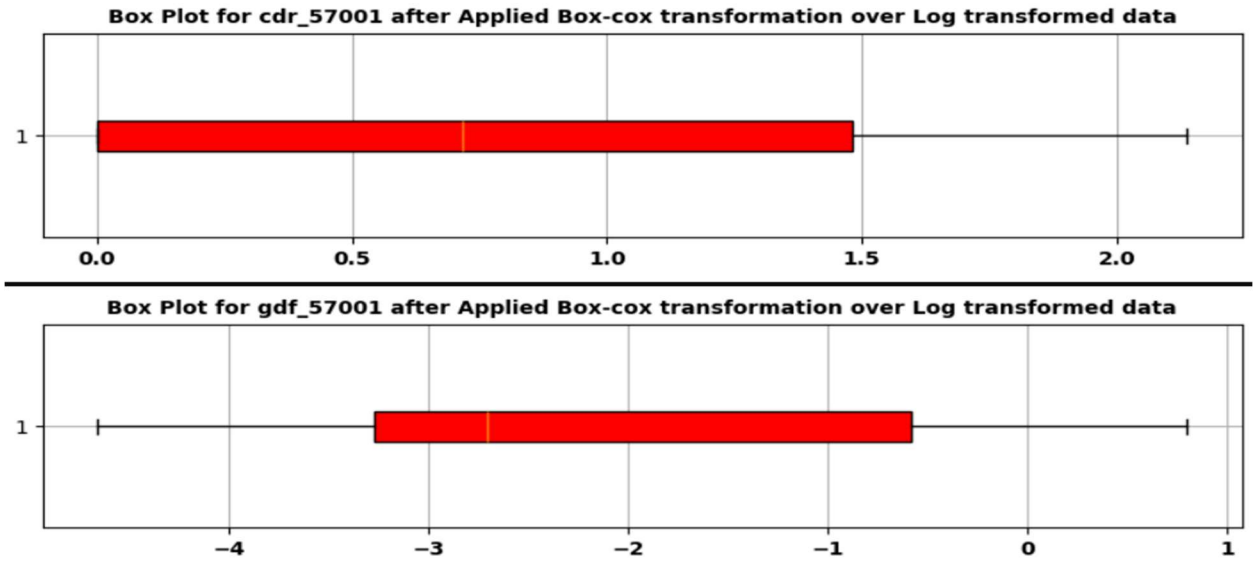

Figure:12.7

3.5 Variance and Mean Analysis

To assess the changes that these transformations bring into the temporal character of the data, a comparative analysis of variance and mean was conducted before and after the log transformation. CDR and GDF data were calculated as variance and mean for the 30-day window, for each gauge though with more emphasis on gauge 57005 in (figures:13.1-13.4, page:50-51).

Before Log Transformation: The mean level had less fluctuations and remained consistently low during high and low flow rates while the variance which demonstrated fluctuations depicted high

variances, especially during high flow periods. Such a difference pointed to the fact that the original variable was heteroscedastic.

After Log Transformation: The variance was reduced in the variation stability sense, which implied a decrease in the peaks identified in the intensity contours. Mean became closer to variance especially in the transformed data implying that the structure was more homoscedastic. Such alignment was crucial in enhancing the efficiency and reliability of forecasting models.

These transformations have not only helped in making normality of the data distributions more appropriate but also, they have been utilized in stabilizing variance to make the data more appropriate for the following model building stage.

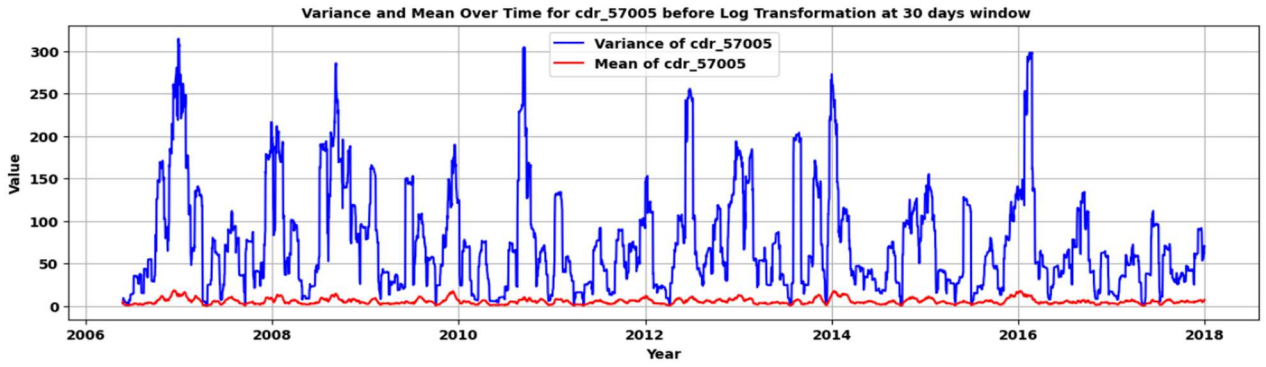

Figure:13.1

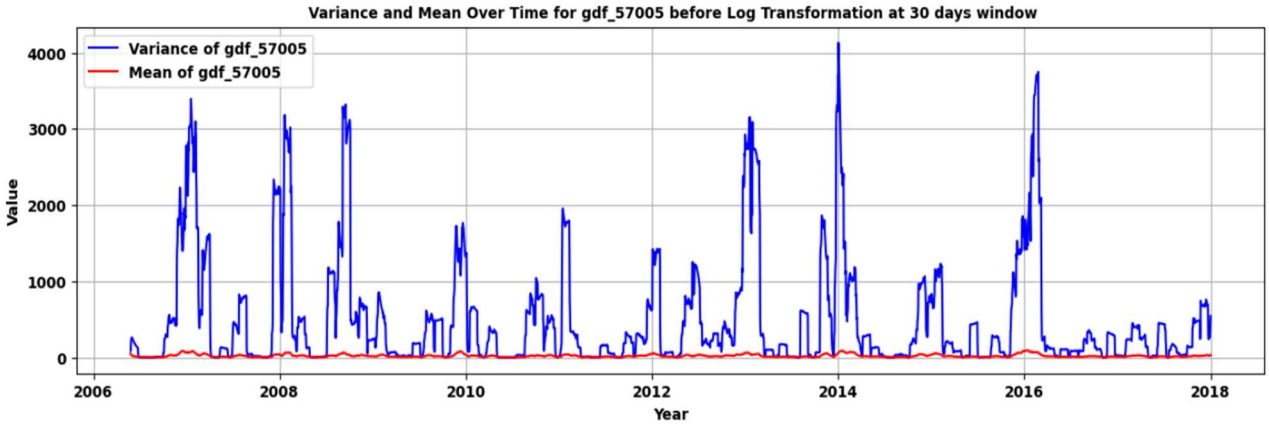

Figure:13.2

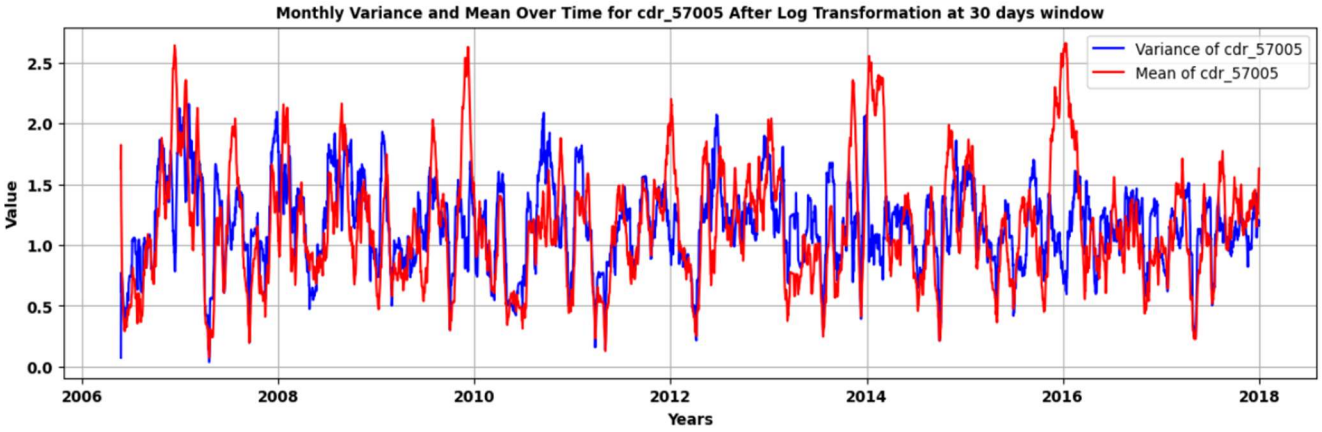

Figure:13.3

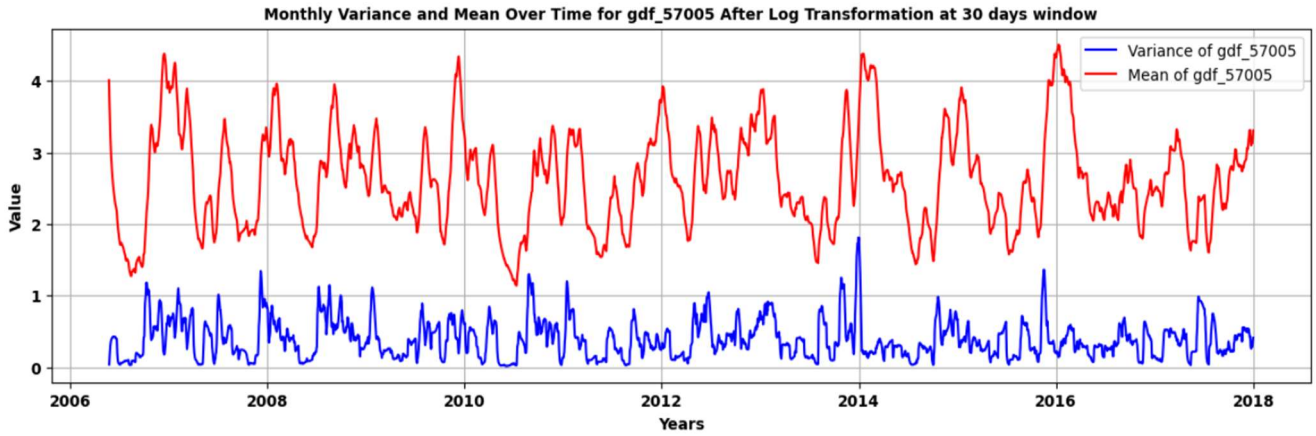

Figure:13.4

3.6 Moving Average to find underlying trend in data

The moving average (MA) technique is one of the most basic methodological approaches in time series analysis, which is used to average out short-term fluctuations to make long-term trends more visible. In this analysis, four types of moving averages were used on the log transformed river flow data for multiple gauges: 7 MA, Double 7 MA, 30 MA, and Weighted MA. All these methods help to eliminate noise with longer periods like the 30 MA able to smooth out more of the noise and thus enable one to capture the trend.

From the moving average plots of each gauge (Figures:14.1-14.14, Page:52-56), no long period increasing or decreasing trends were observed in the river flow data.

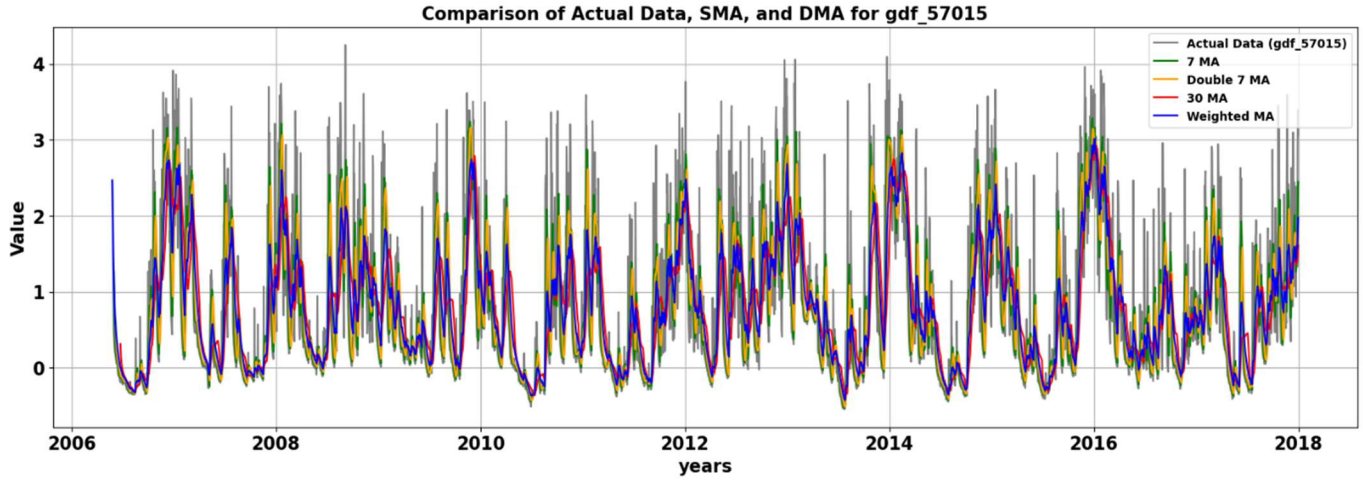

Figure:14.1

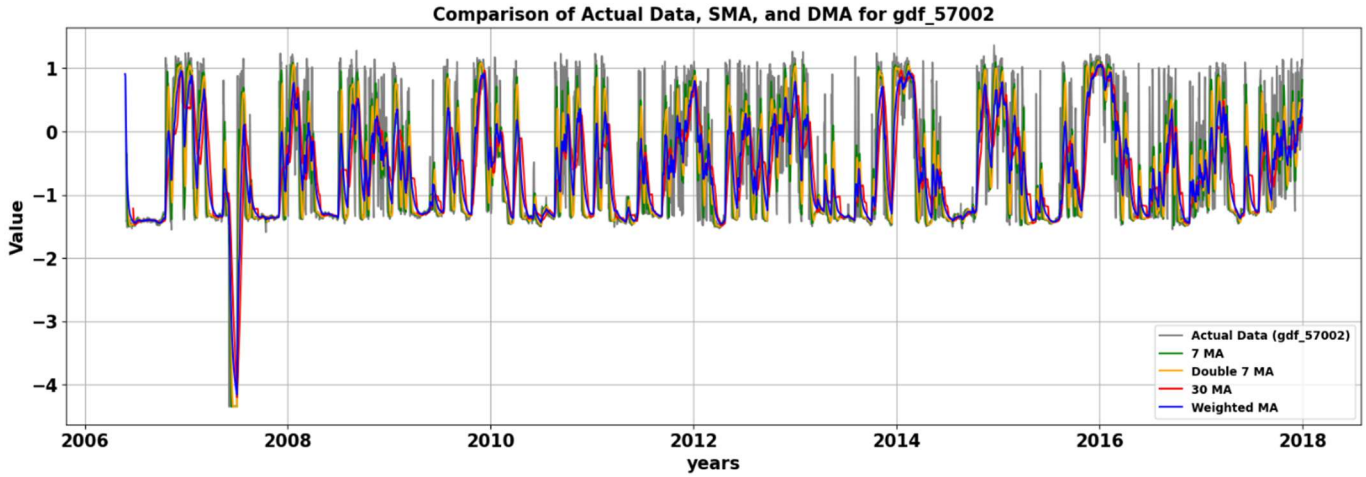

Figure:14.2

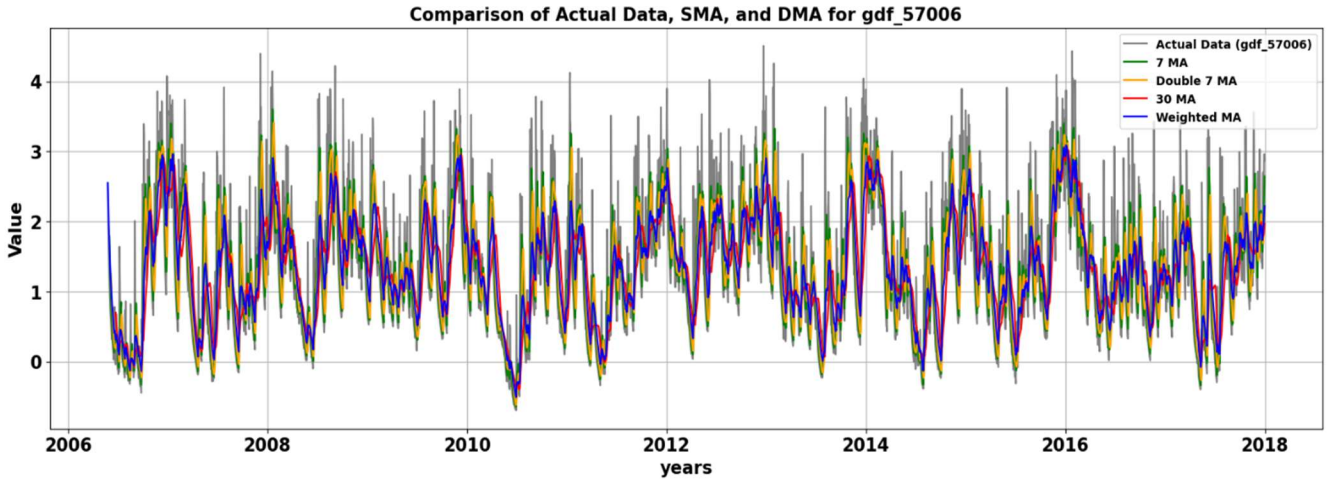

Figure:14.3

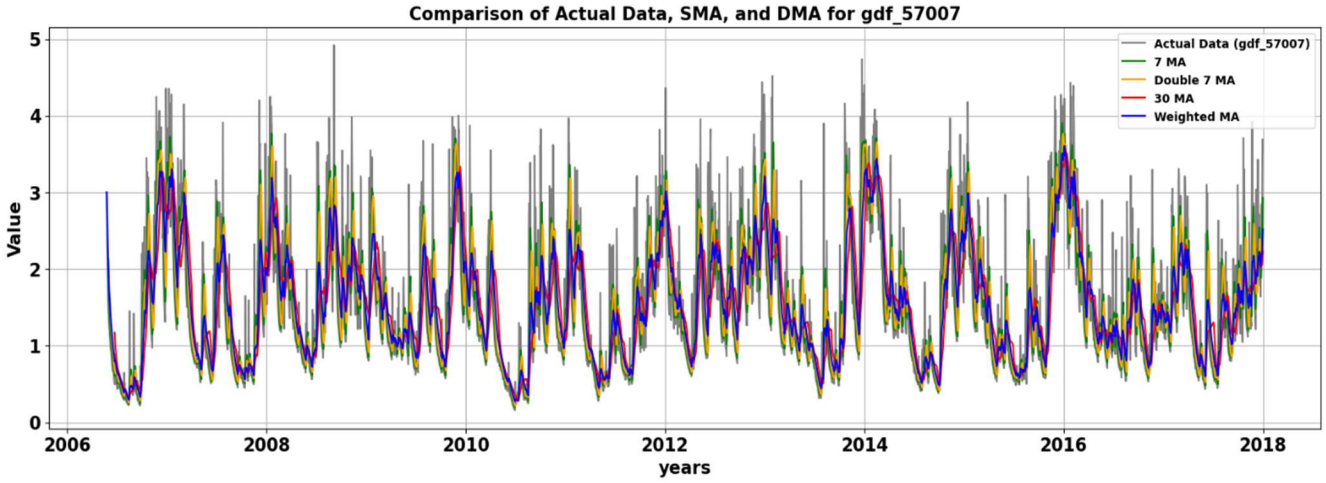

Figure:14.4

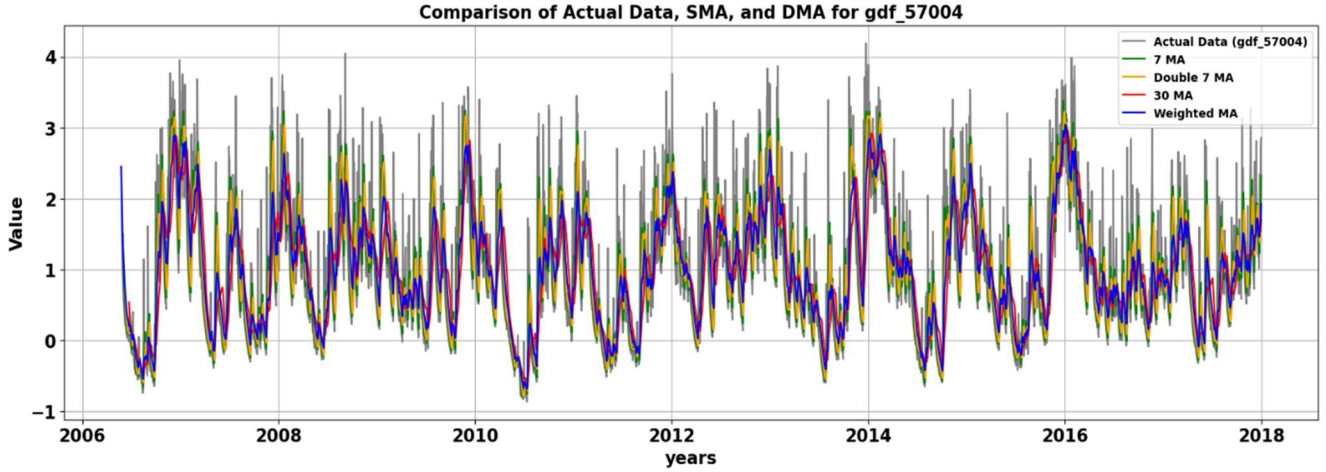

Figure:14.5

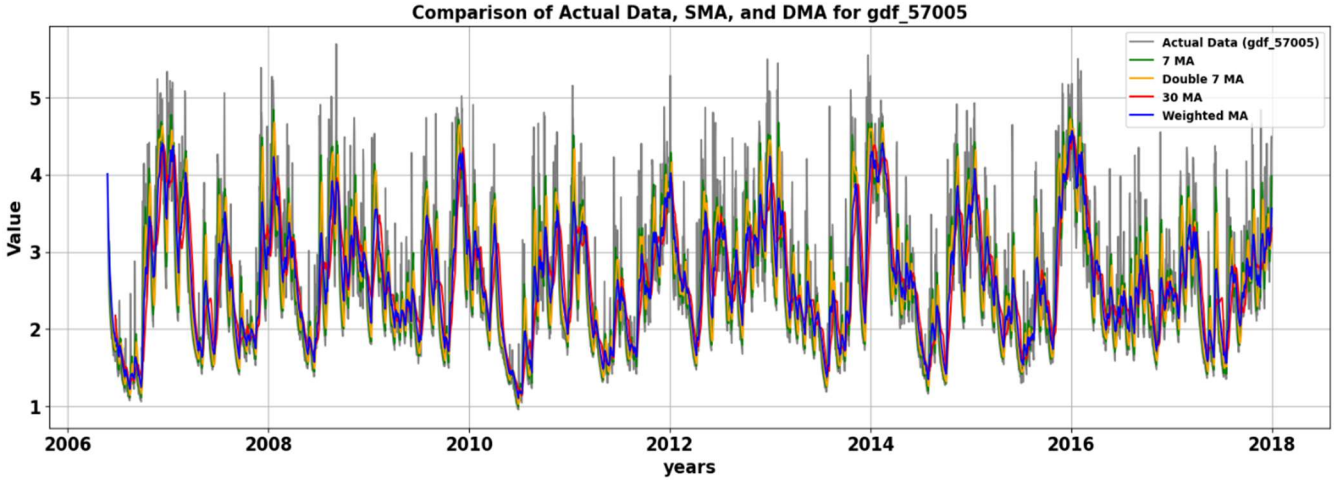

Figure:14.6

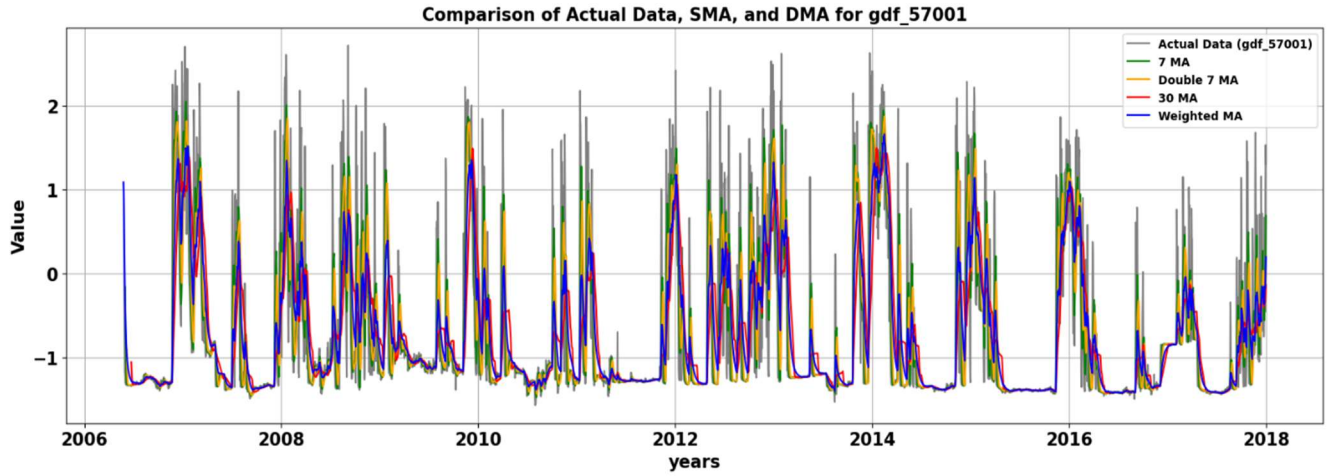

Figure:14.7

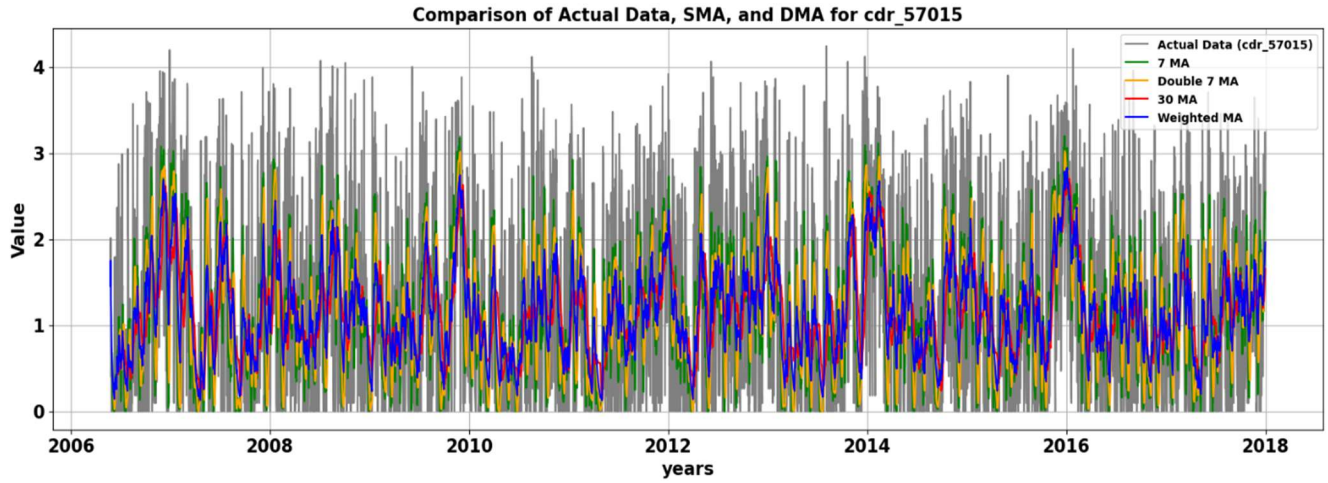

Figure:14.8

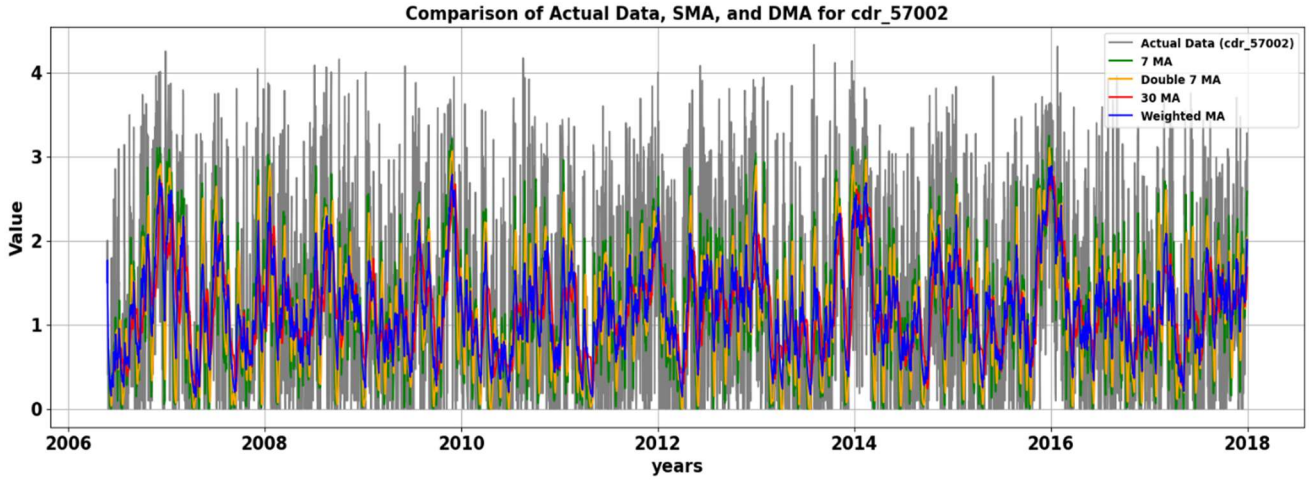

Figure:14.9

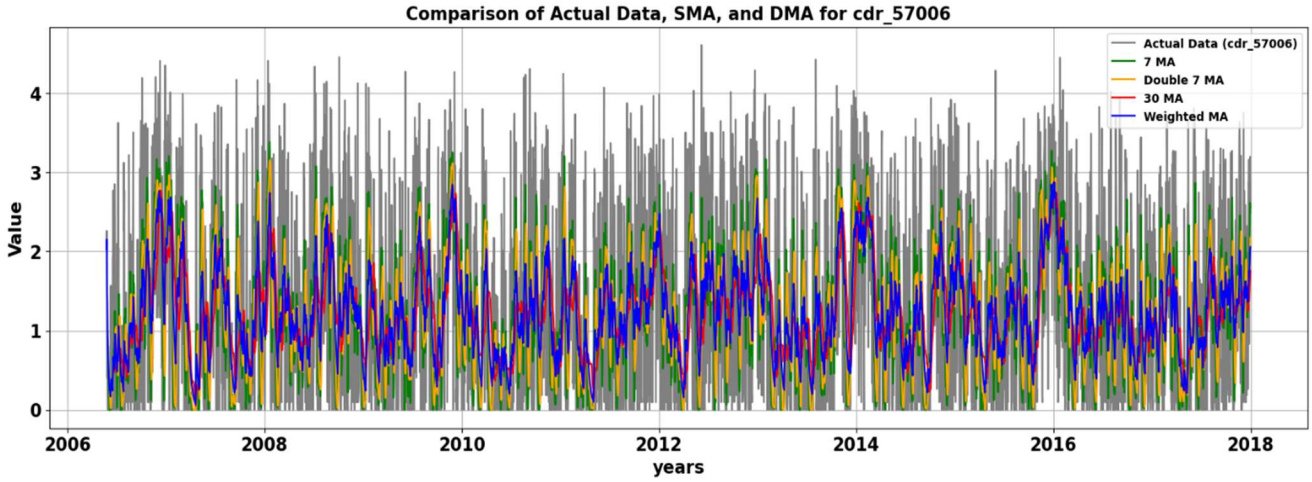

figure:14.10

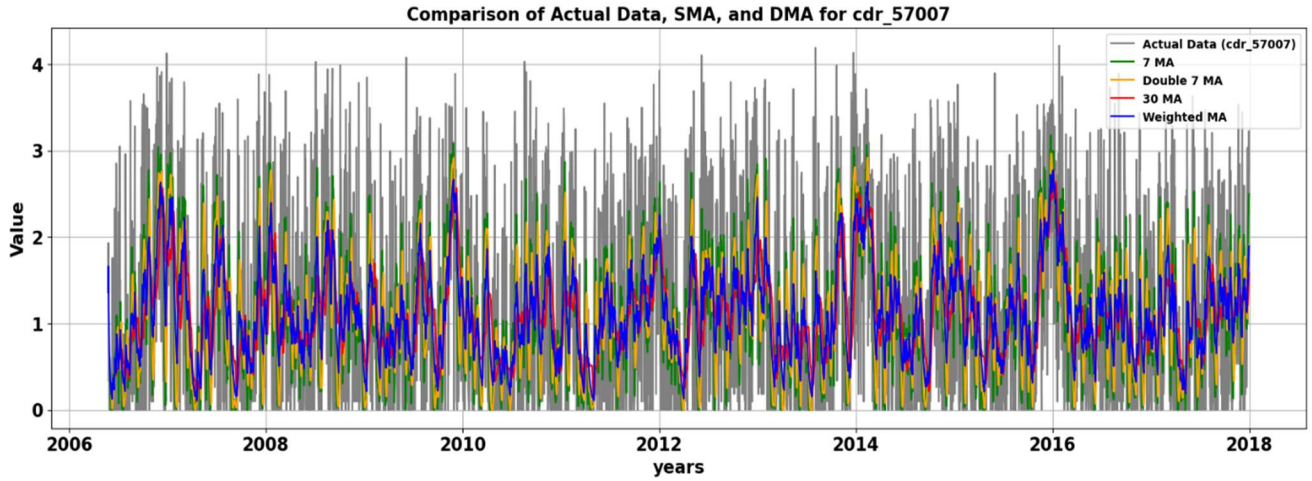

Figure:14.11

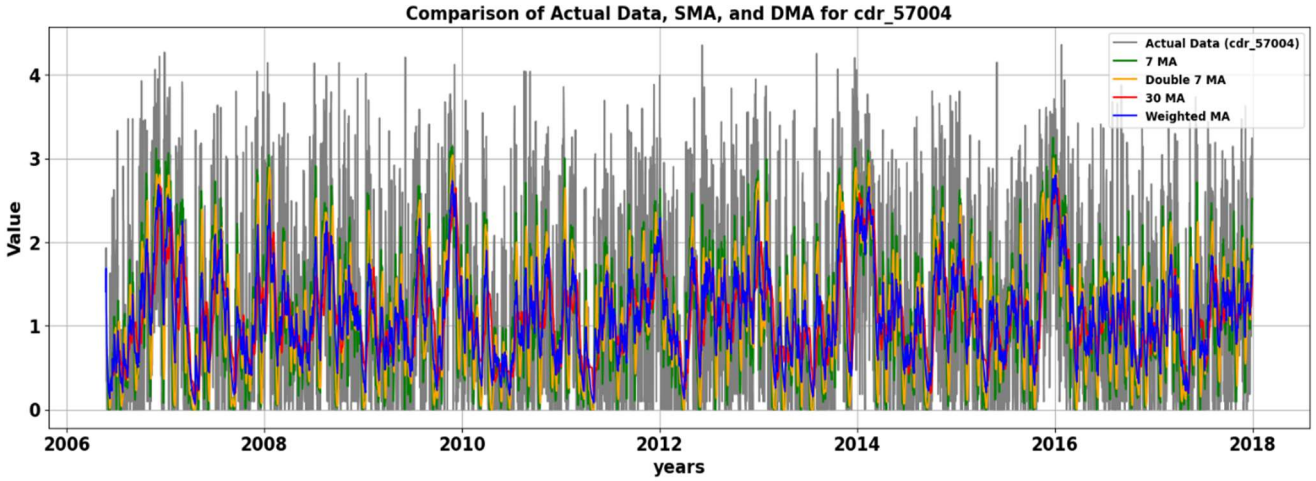

Figure:14.12

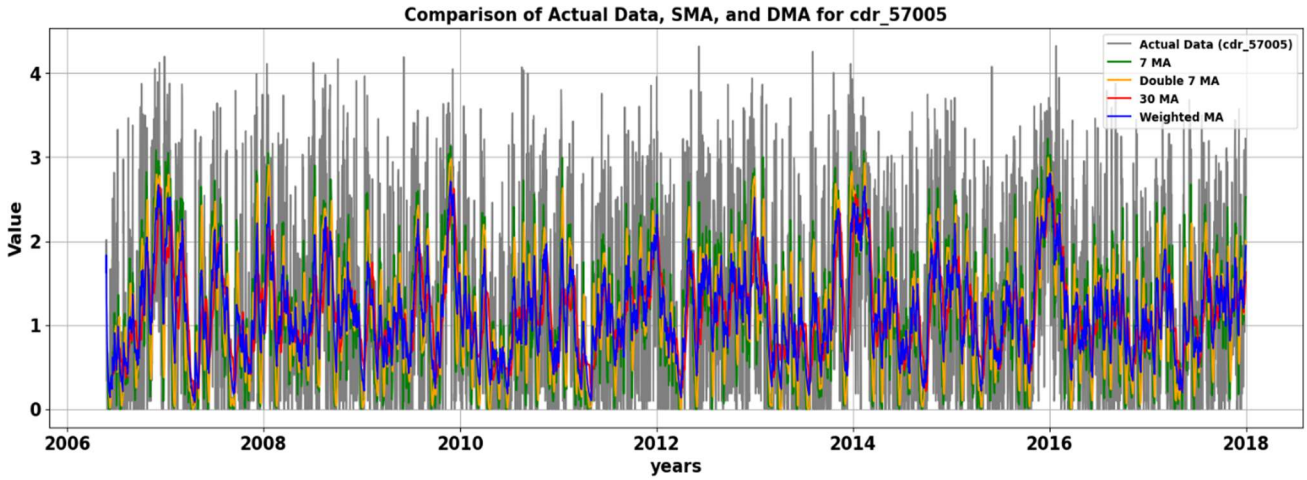

Figure:14.13

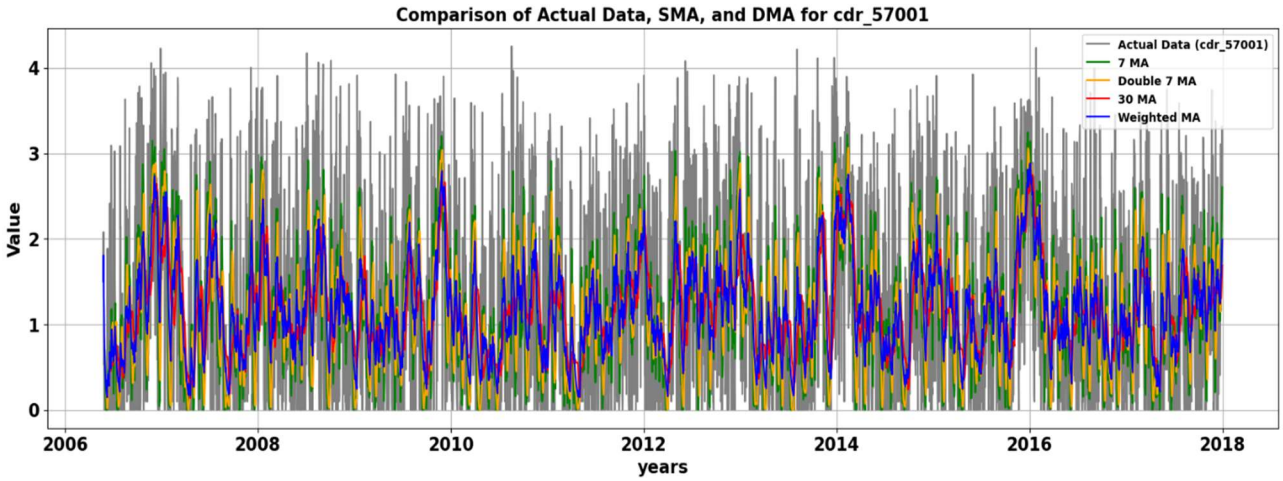

Figure:14.14

3.7 Time Series Decomposition to find seasonality or trend (STL Decomposition)

From the obtained series of river flow and area rainfall, Seasonal-Trend Decomposition using LOESS (STL) was used to decompose the data to reveal the underlying cyclic behaviour. STL allows for the decomposition of a time series into three components: trend, seasonality, and residuals with the seasonality was set at 31 days to cover months. This decomposition was done on the log-transformed data.

(Figures:15.1-15.14, page:57-59) presented the STL decomposition of each gauge, The seasonal constituents present regular cycles over years which indicate the variation that recurs each year That is, there are strong indications of seasonality.

Furthermore, to visually examine the pattern of the last three years the seasonal plots were created (Figures:16.1-16.14, page:59-64). These plots also show the reliability of seasonality, particularly, the low flow was observed around the month between April to September and high flow was around November to February.

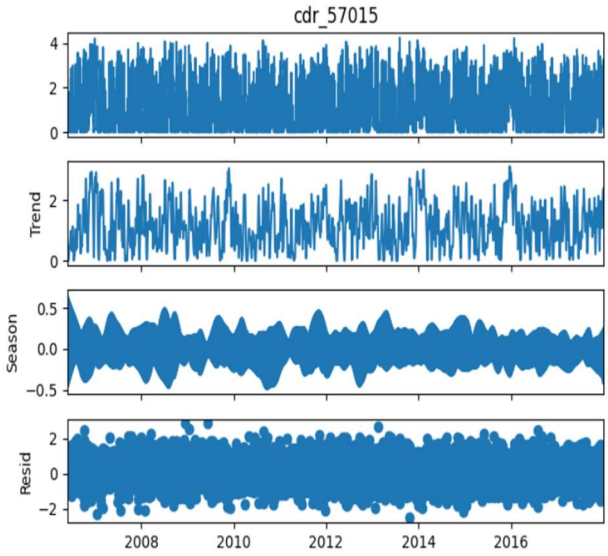

Figure:15.1

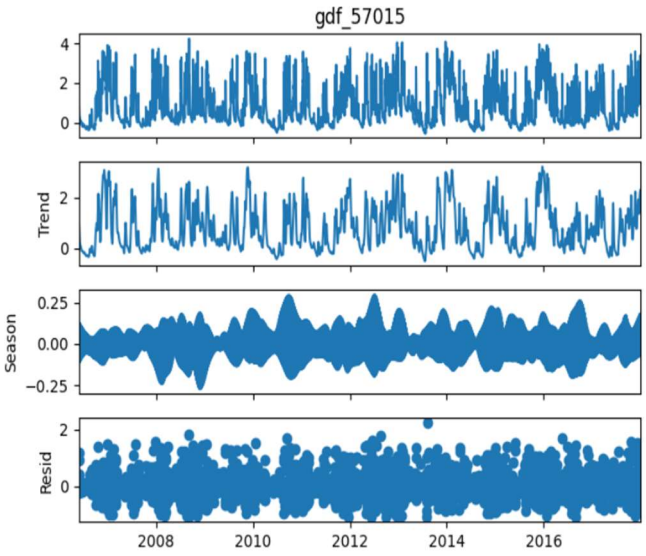

figure:15.2

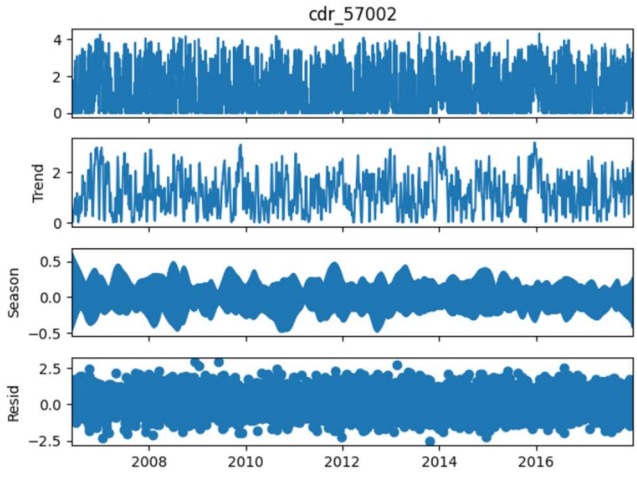

Figure:15.3

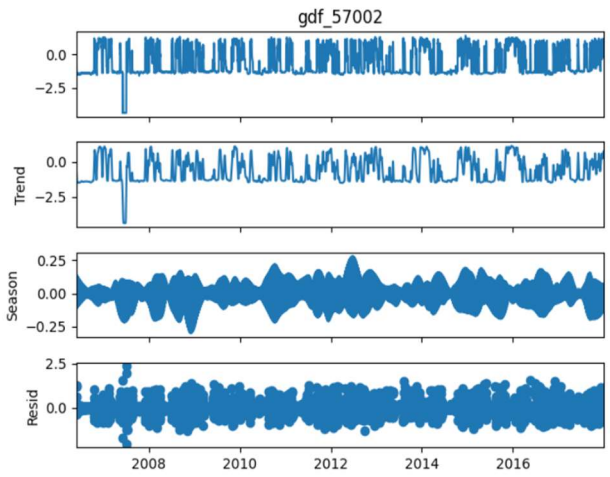

Figure:15.4

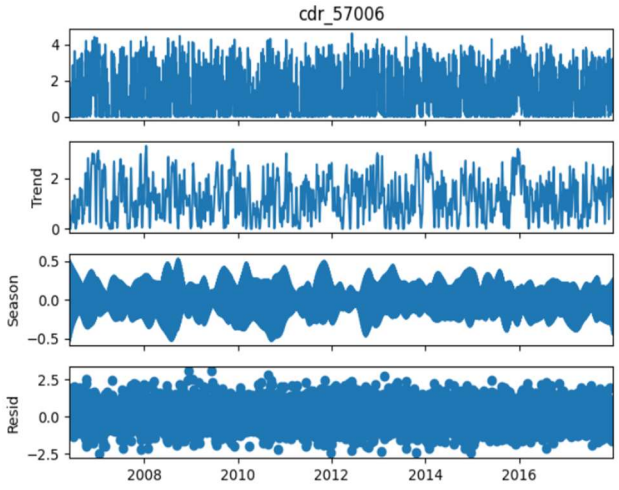

Figure:15.5

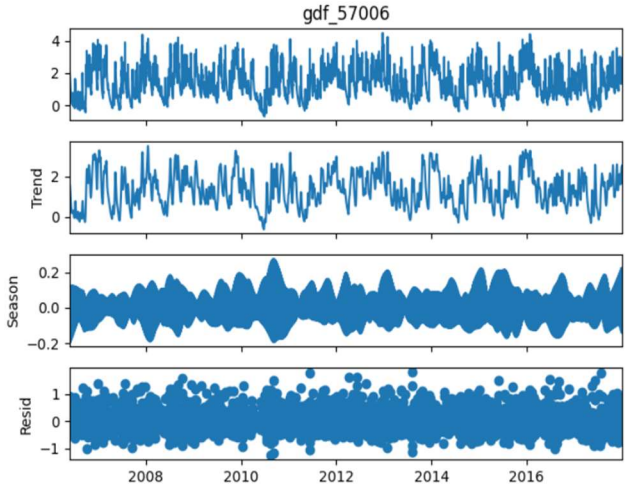

Figure:15.6

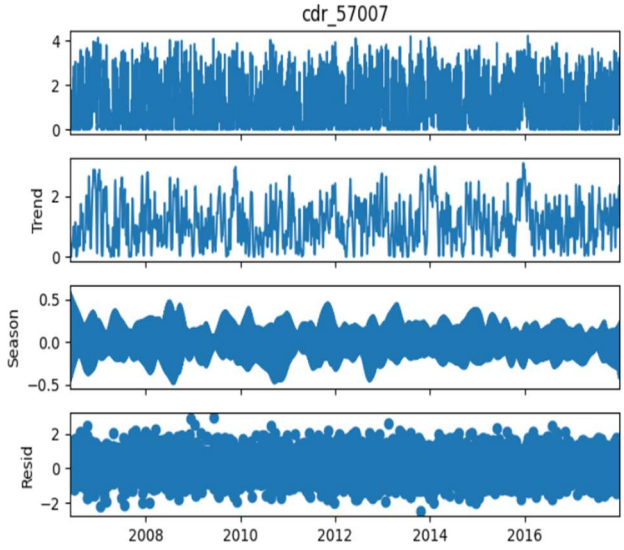

Figure:15.7

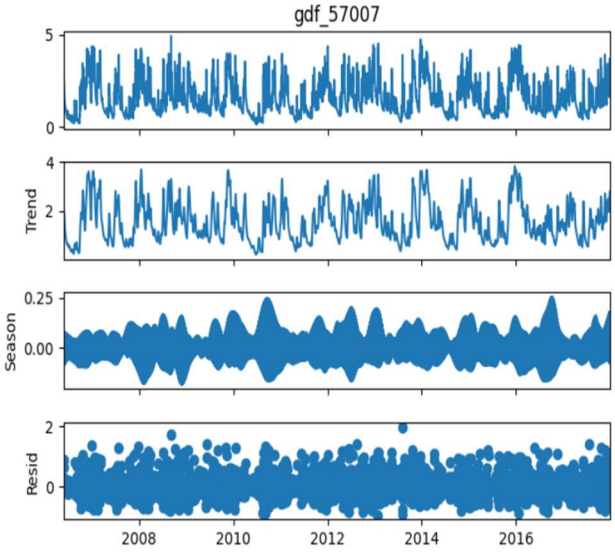

Figure:15.8

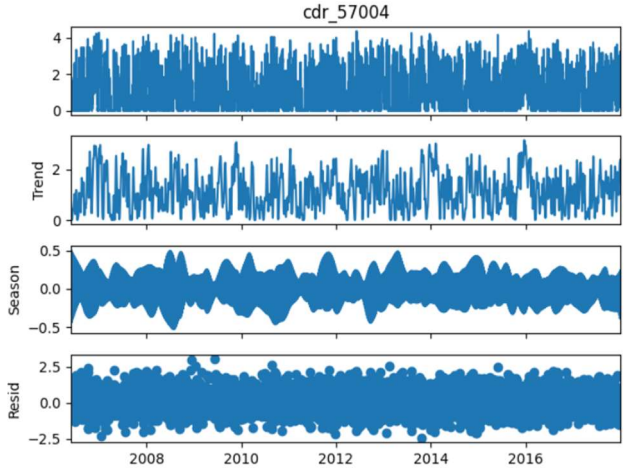

Figure:15.9

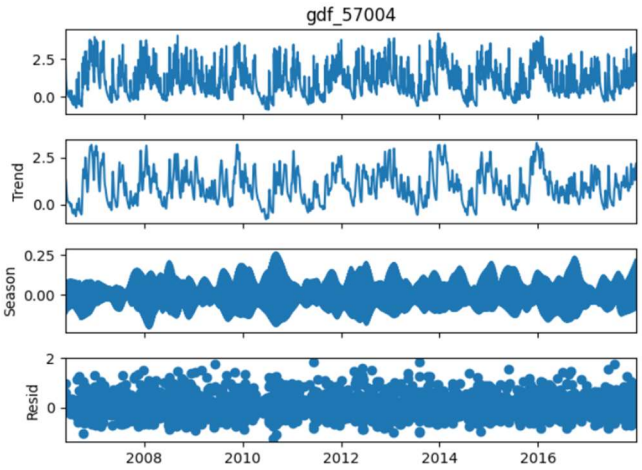

Figure:15.10

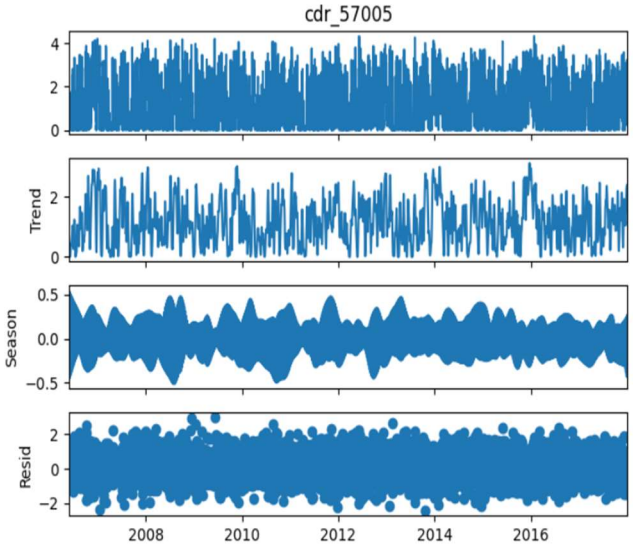

Figure:15.11

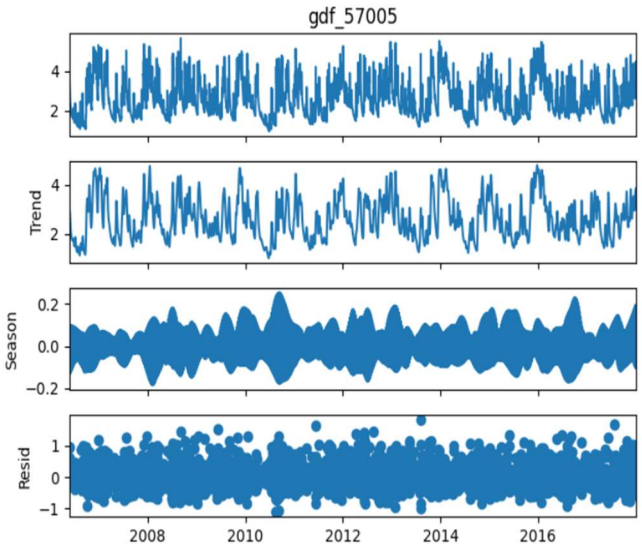

Figure:15.12

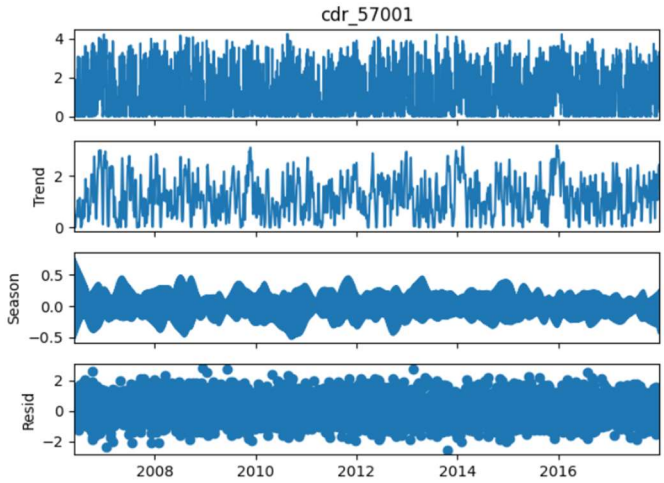

Figure:15.13

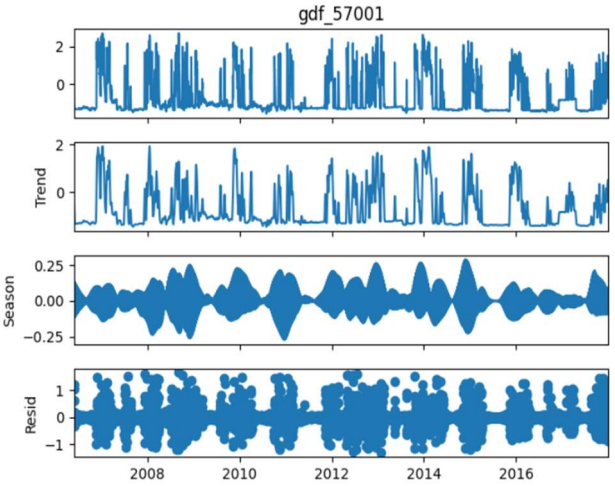

Figure:15.14

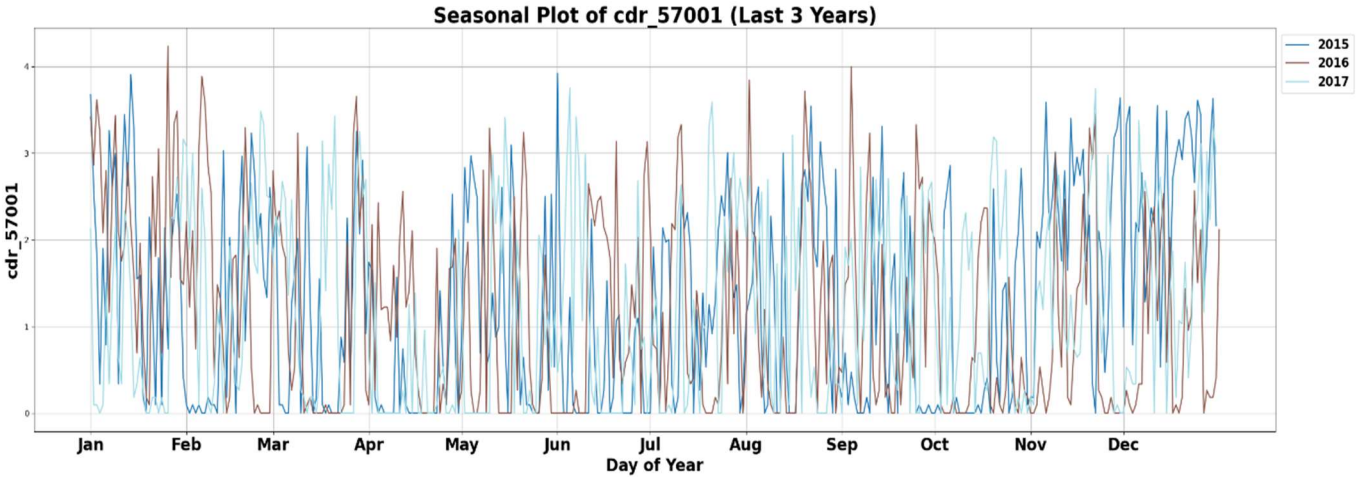

Figure:16.1

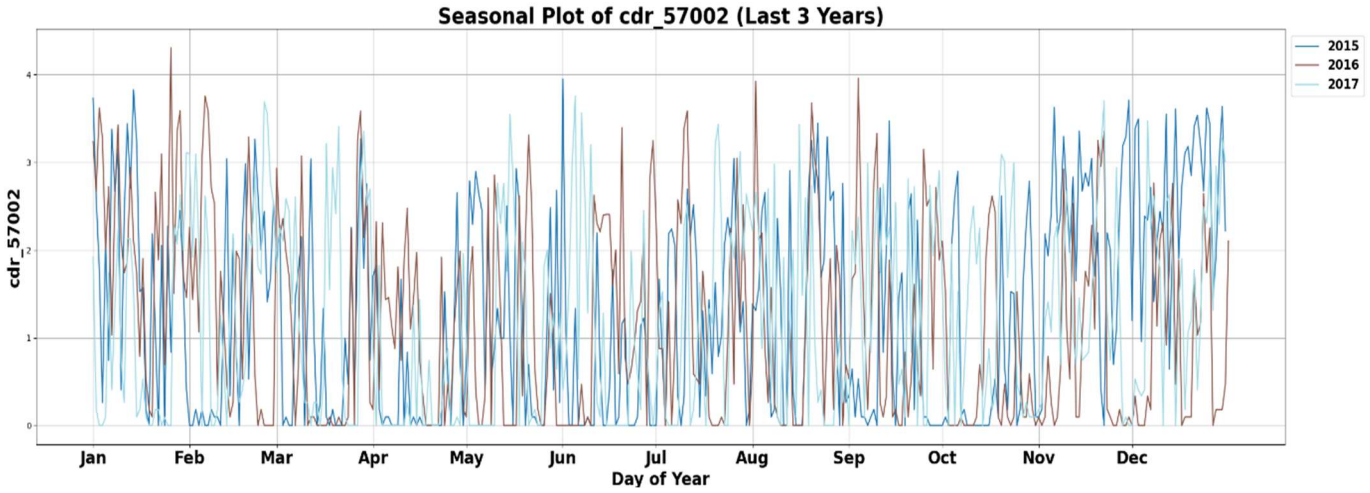

Figure:16.2

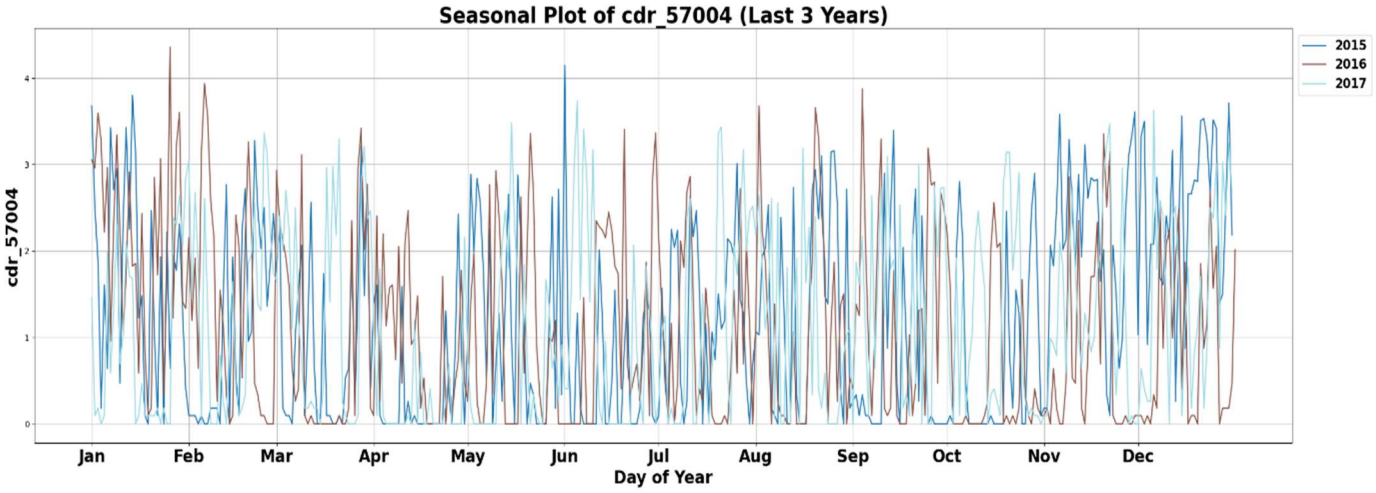

Figure:16.3

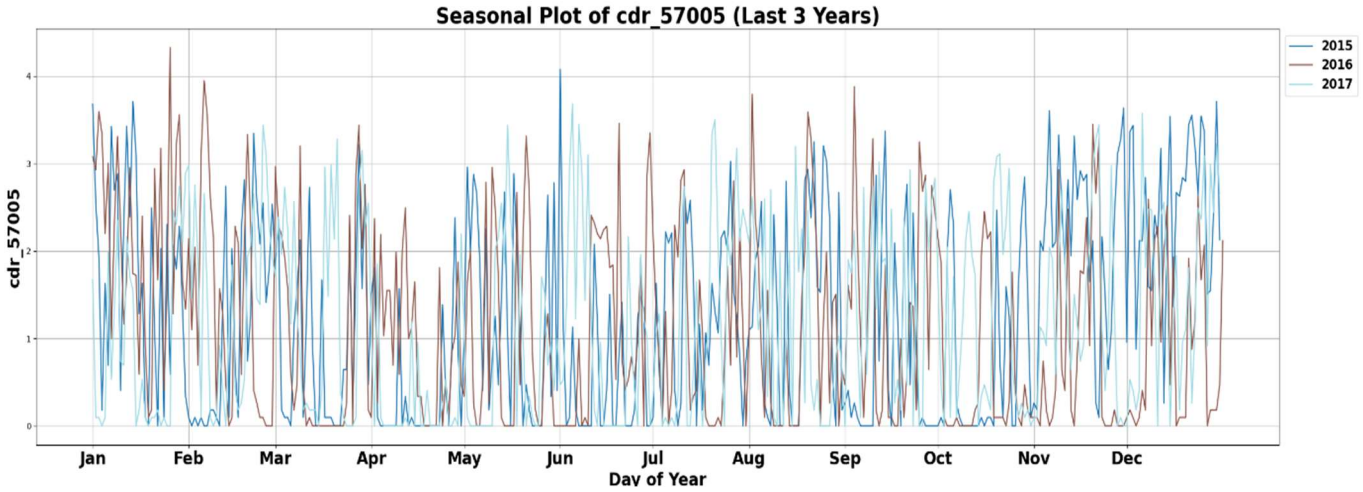

Figure:16.4

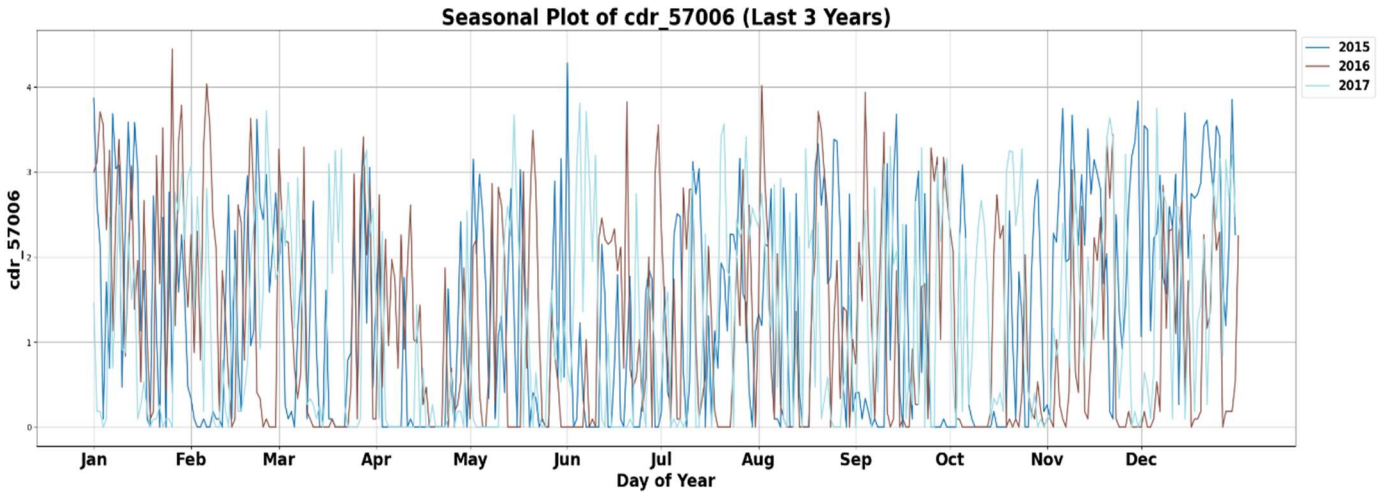

Figure:16.5

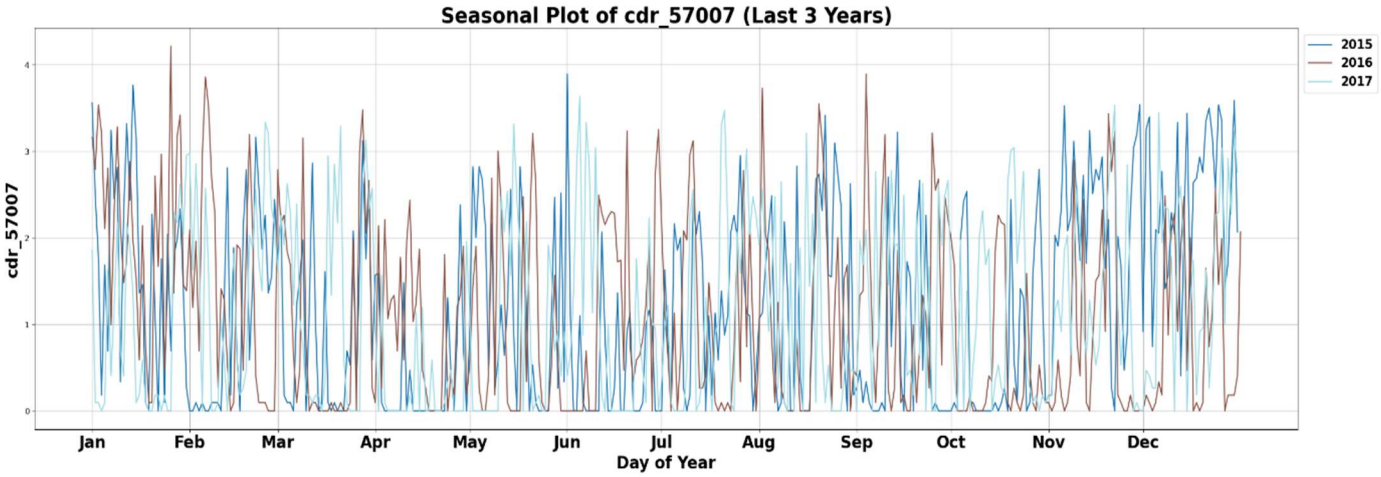

Figure:16.6

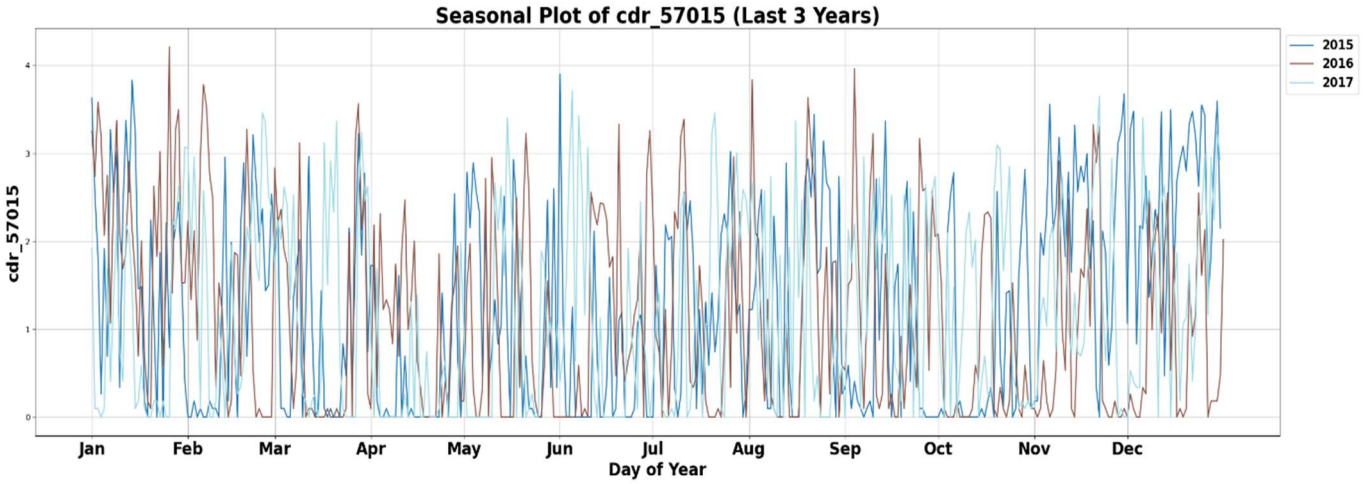

Figure:16.7

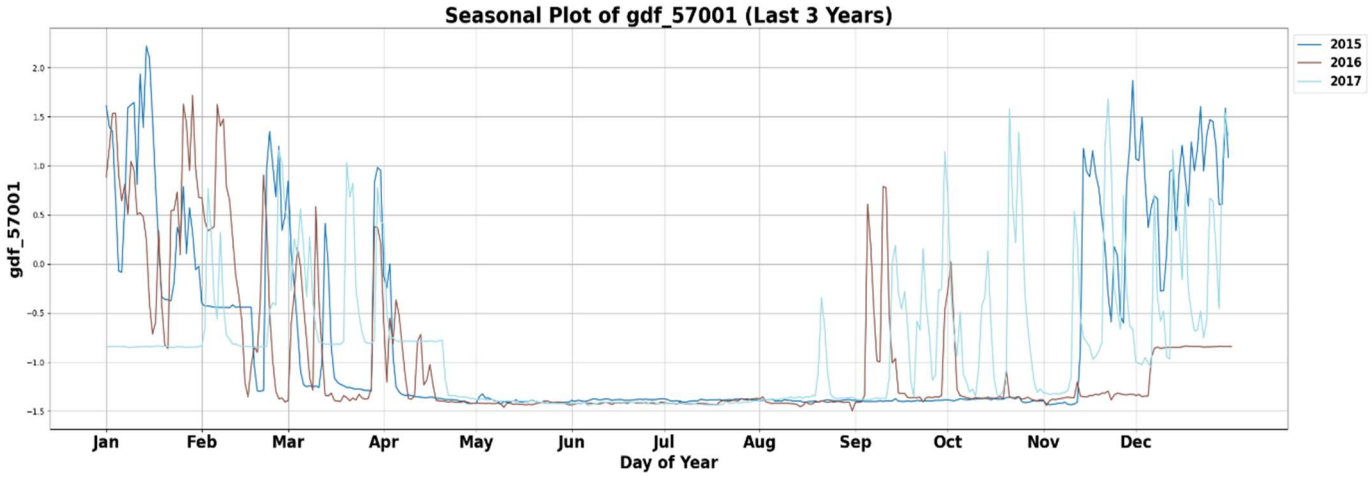

Figure:16.8

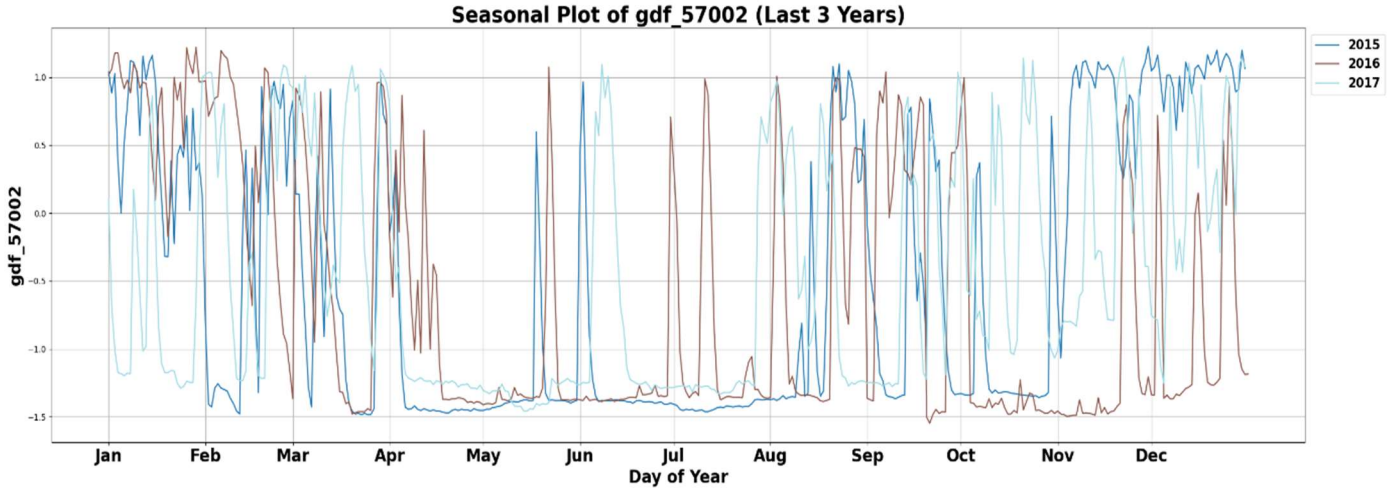

Figure:16.9

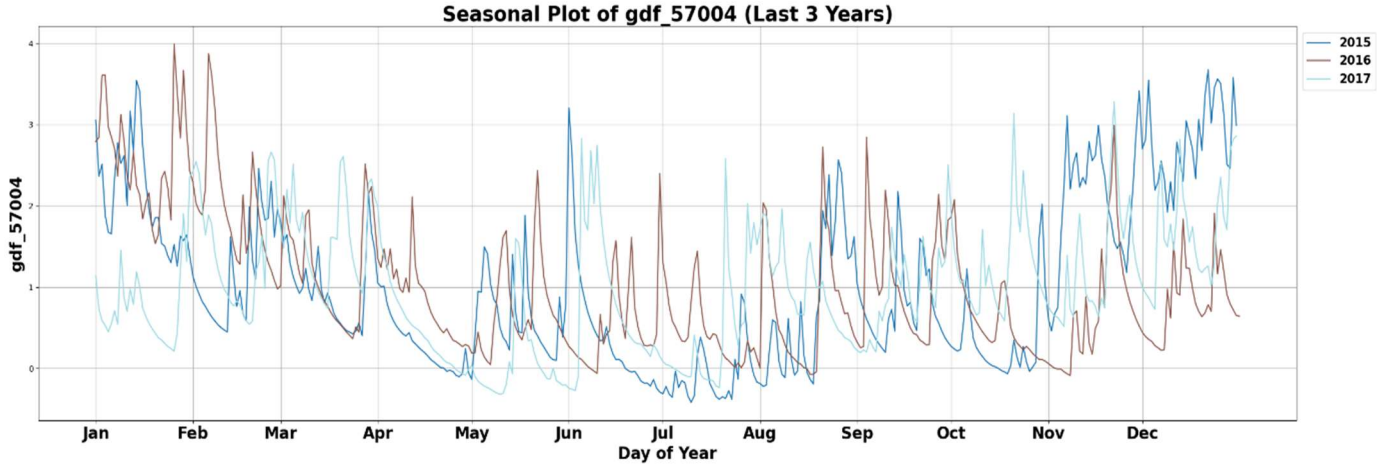

Figure:16.10

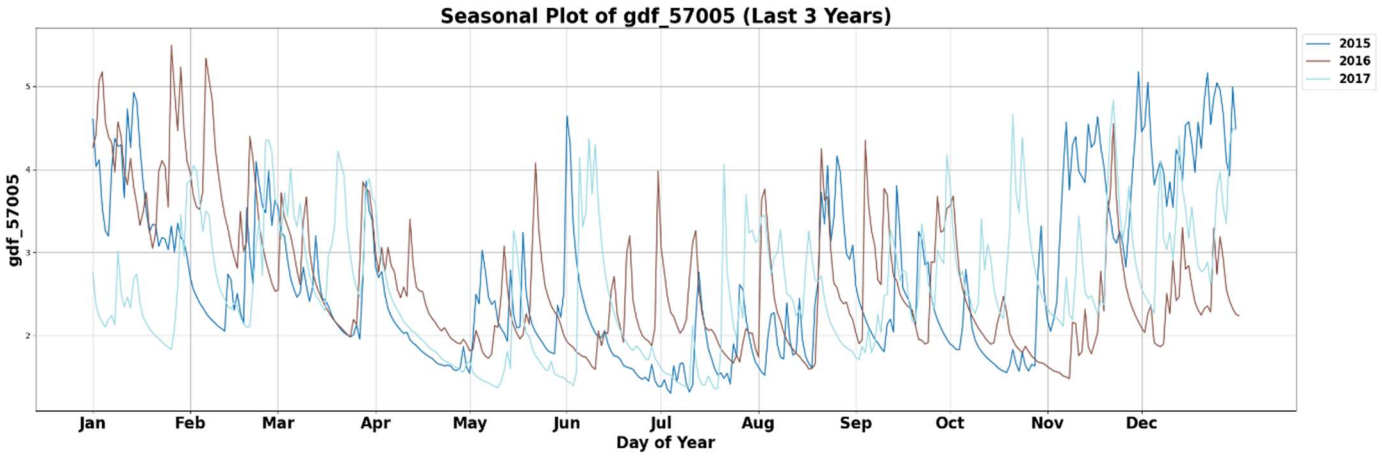

Figure:16.11

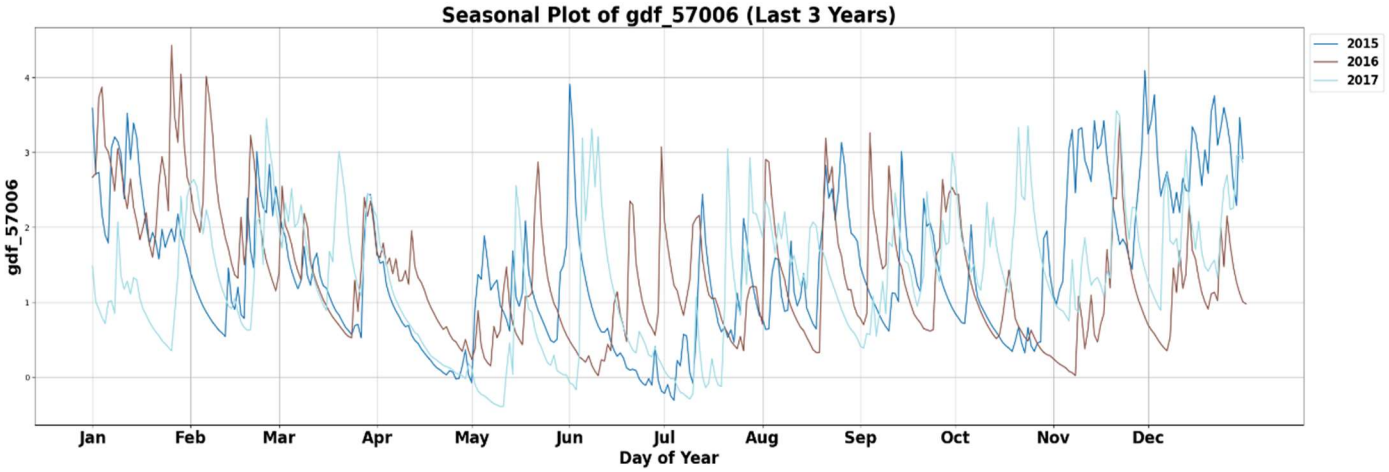

Figure:16.12

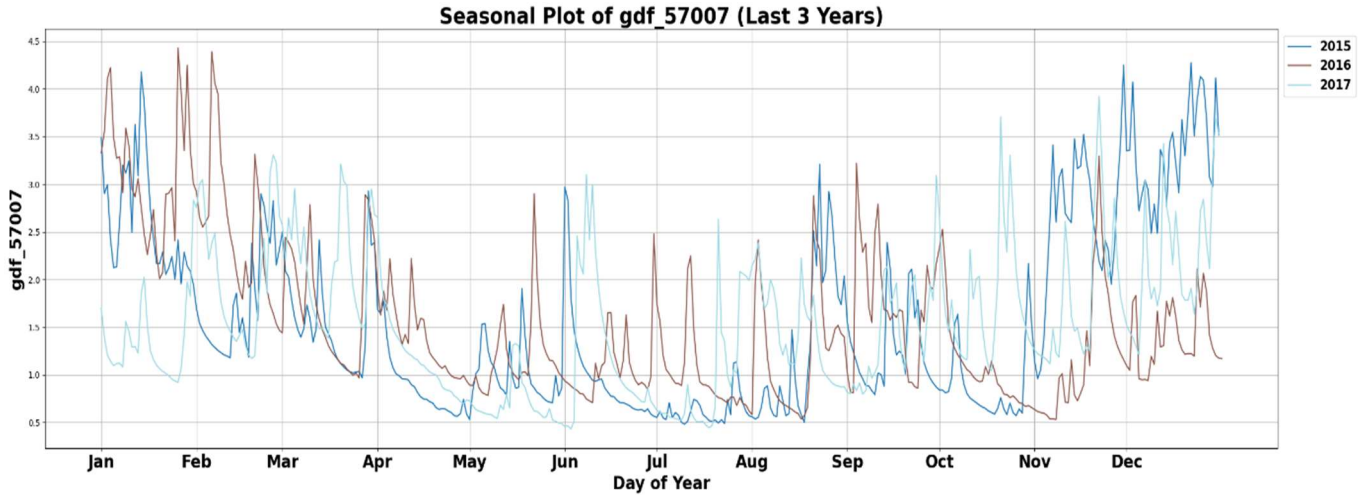

Figure:16.13

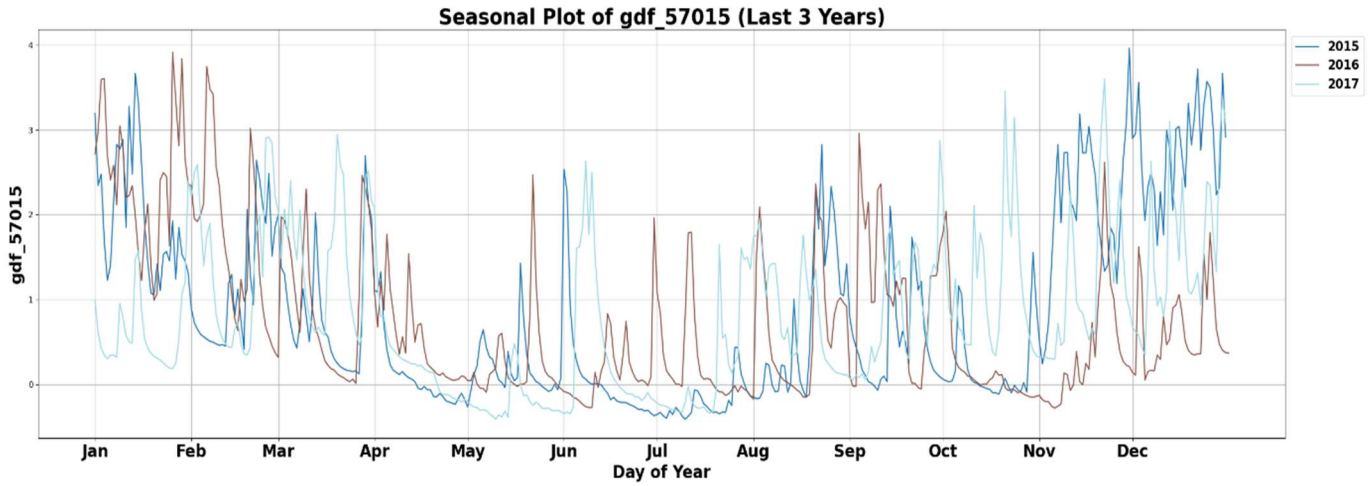

Figure:16.14

3.8 Autocorrelation Function and Partial Autocorrelation Function

Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) (figures:17.1-17.14, page:64-69) were created for both rainfall and daily flow data post logarithm transformation. As evidenced from the ACF plots below, autocorrelation is still present at various lags and therefore, it is difficult to select an appropriate number of lags simply based on ACF. However, from the PACF plots more clarity is observed in the number of peaks and the number of significant spikes was observed up to 4 lags in the rainfall Data set. The PACF for daily flow data provides different significant lags for different gauges up to lag 5th only. While some plots can experience negative spikes because of the influence of negative values coming from the log transformation step if the original data include near-zero values.

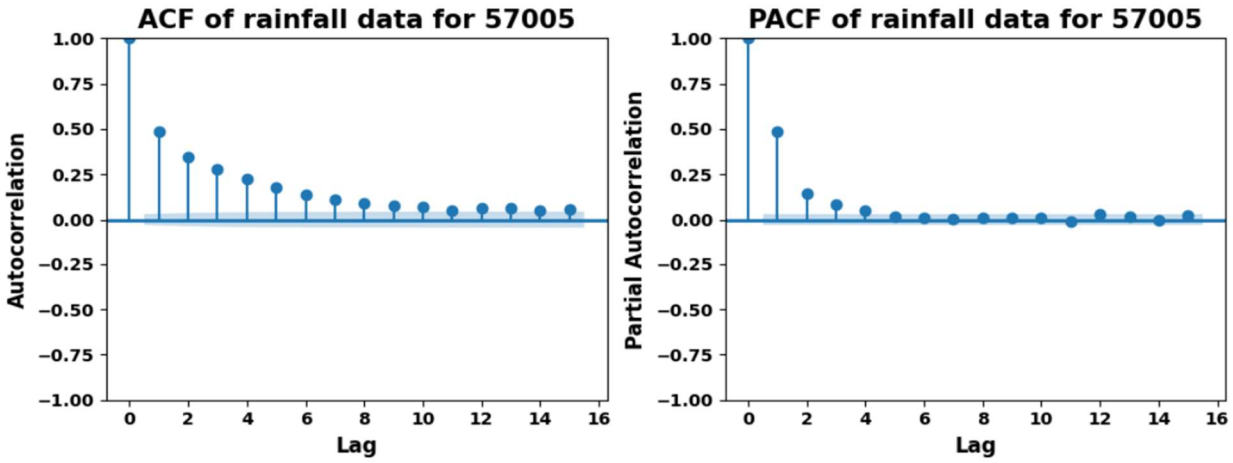

Figure:17.1

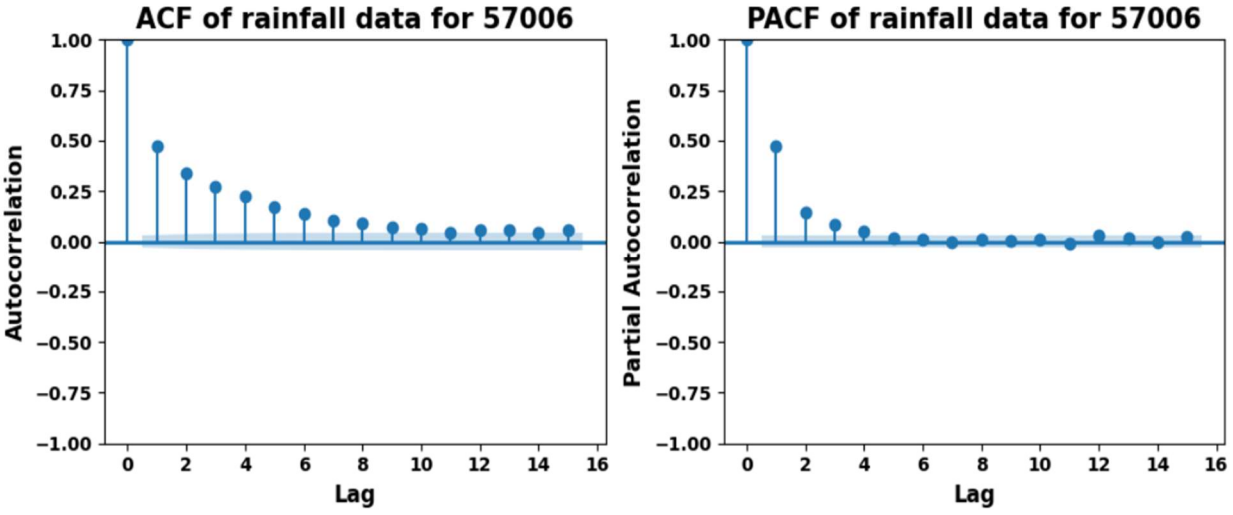

Figure:17.2

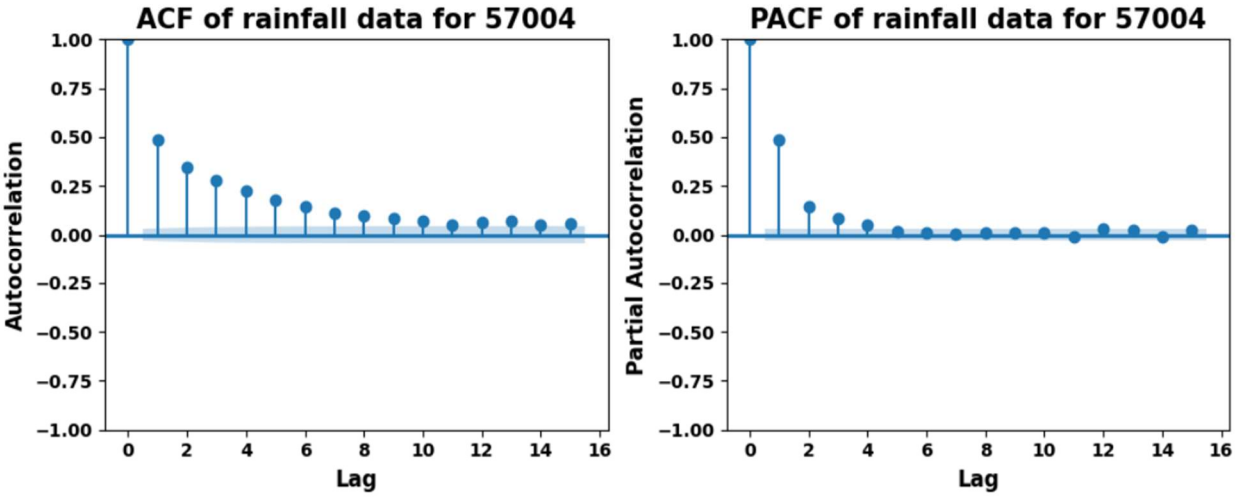

Figure:17.3

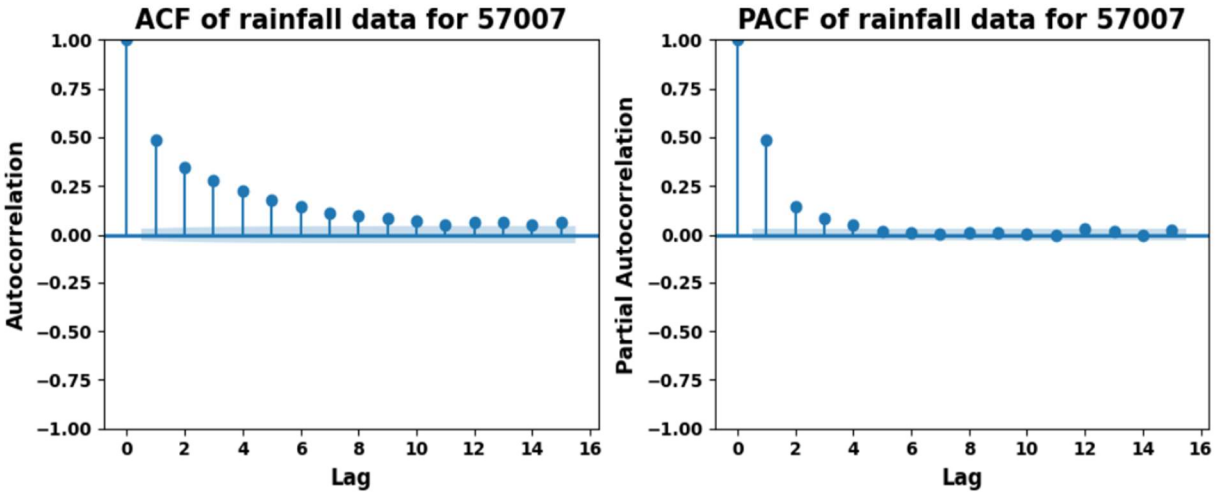

Figure:17.4

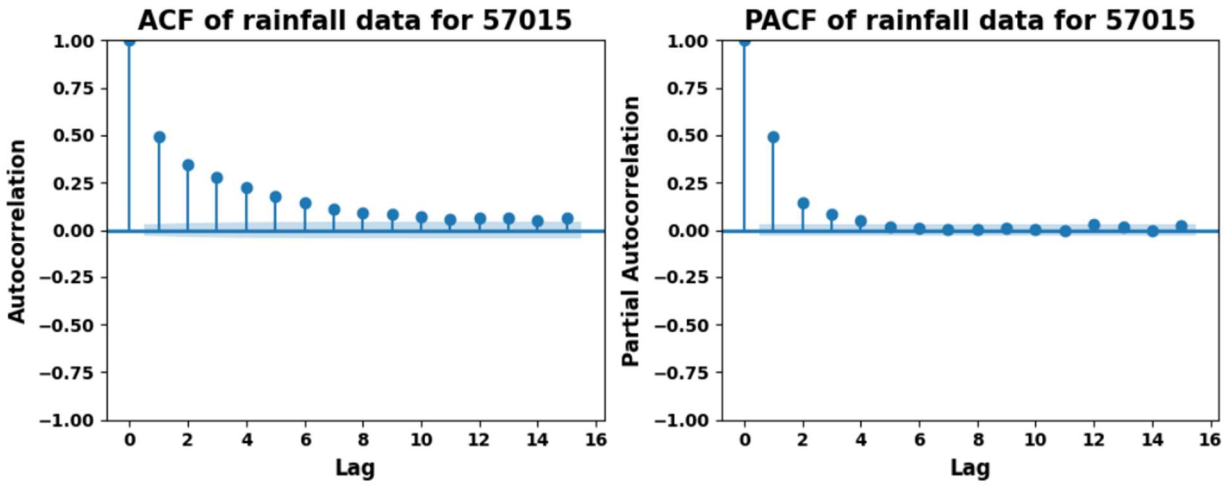

Figure:17.5

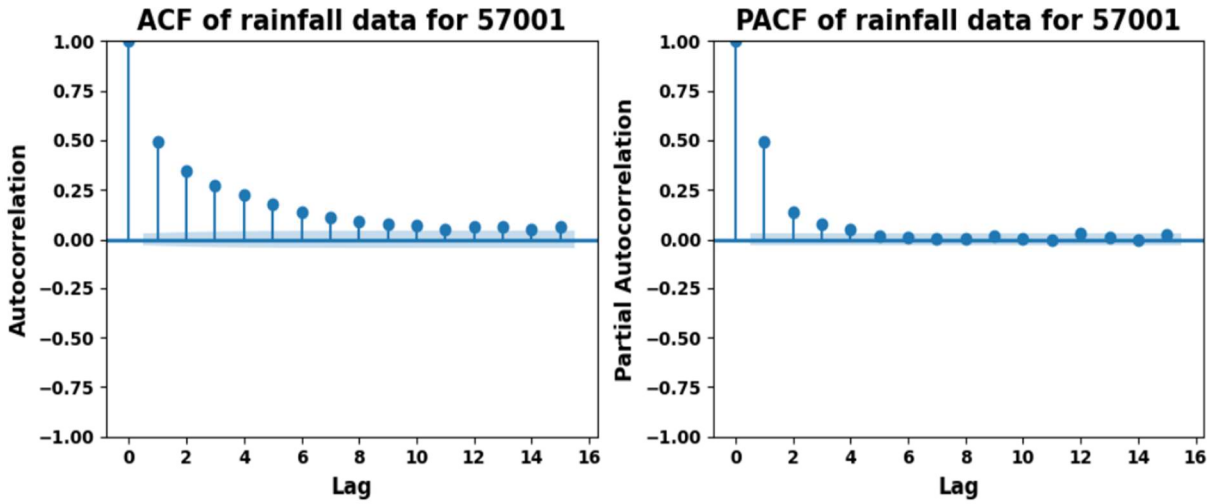

Figure:17.6

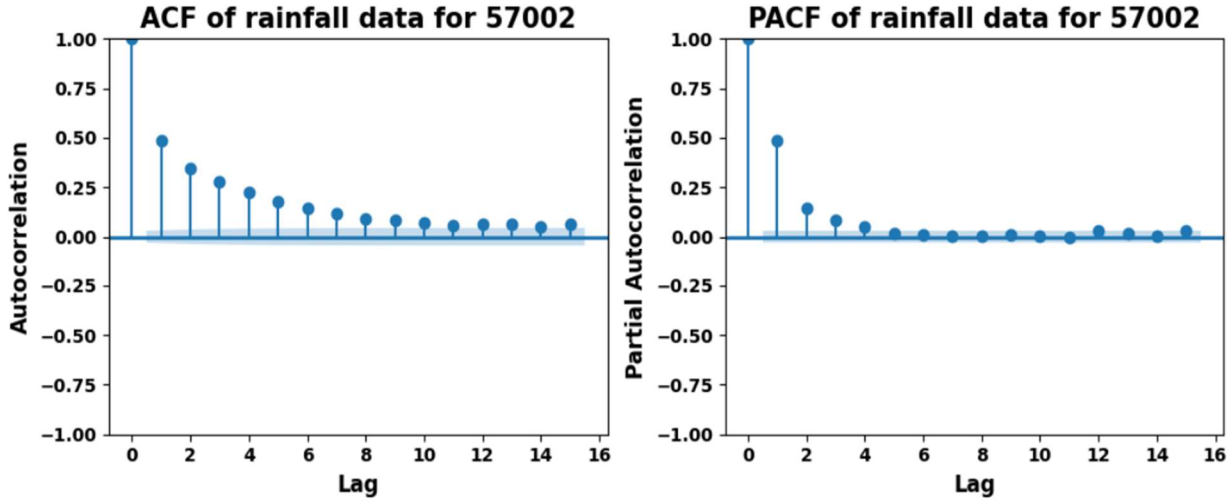

Figure:17.7

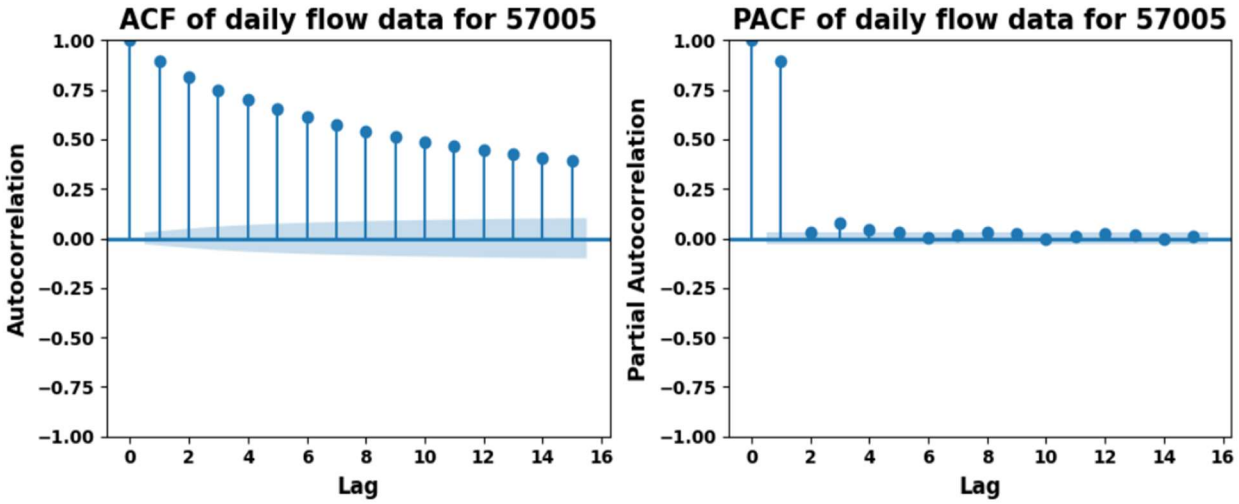

Figure:17.8

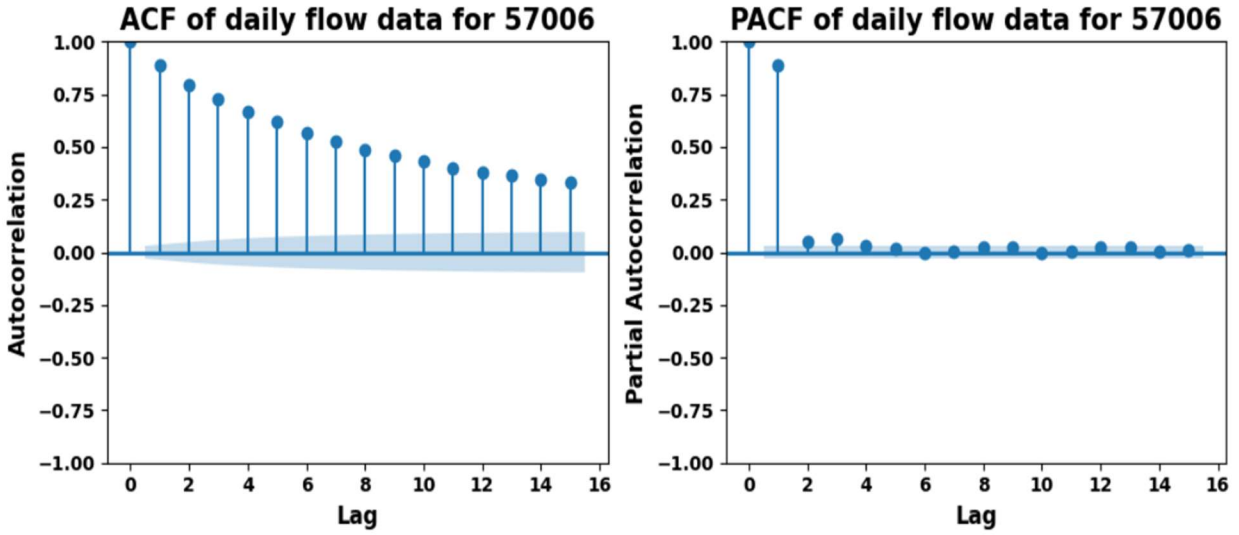

Figure:17.9

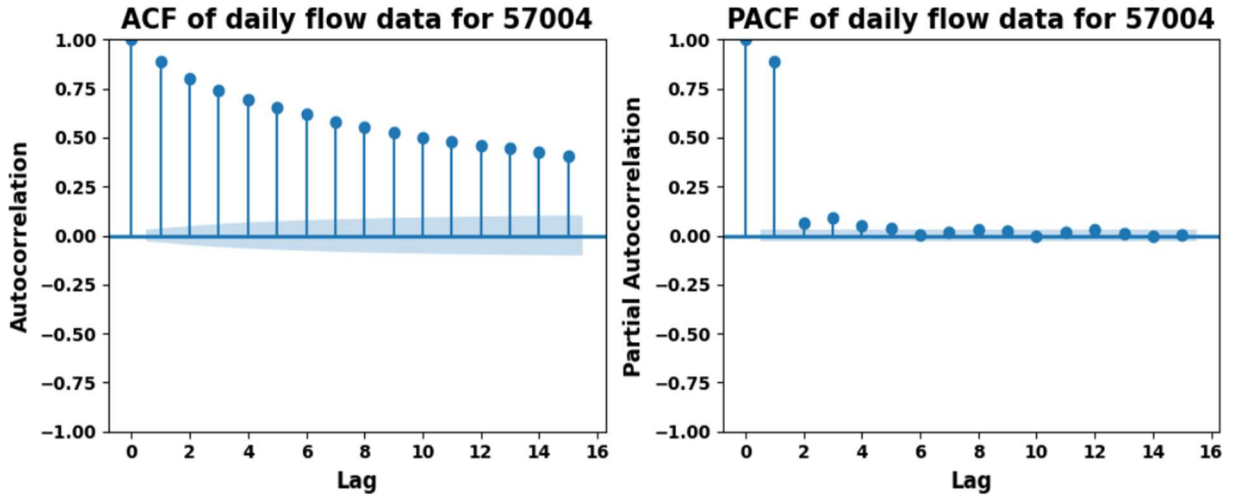

Figure:17.10

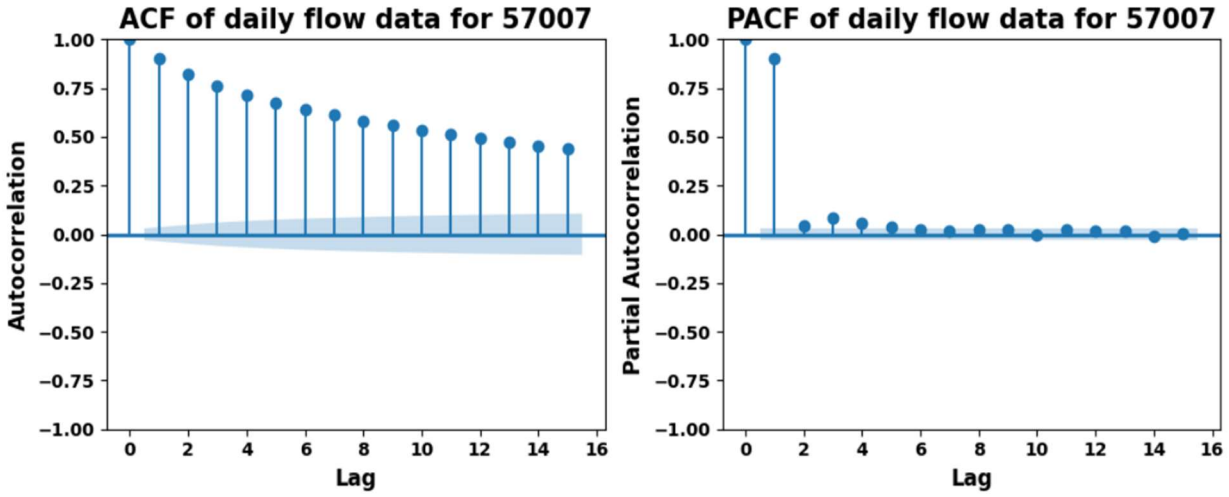

Figure:17.11

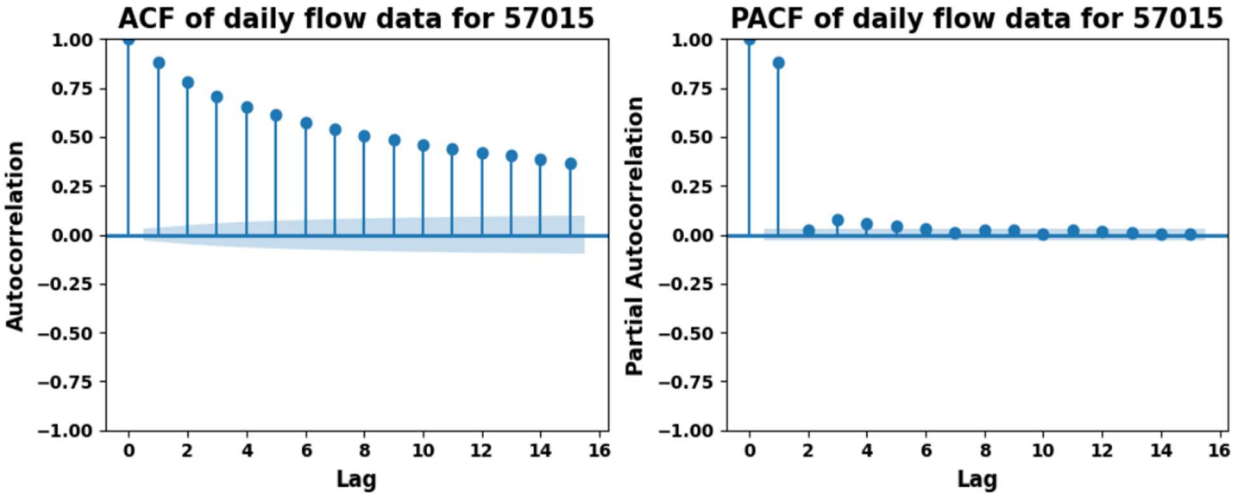

Figure:17.12

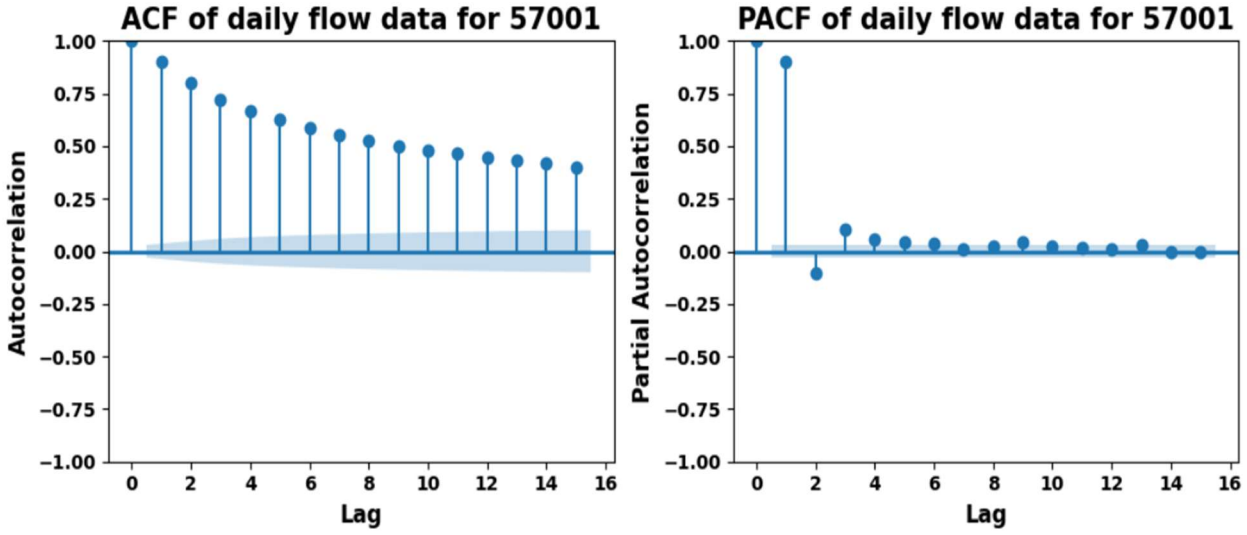

Figure:17.13

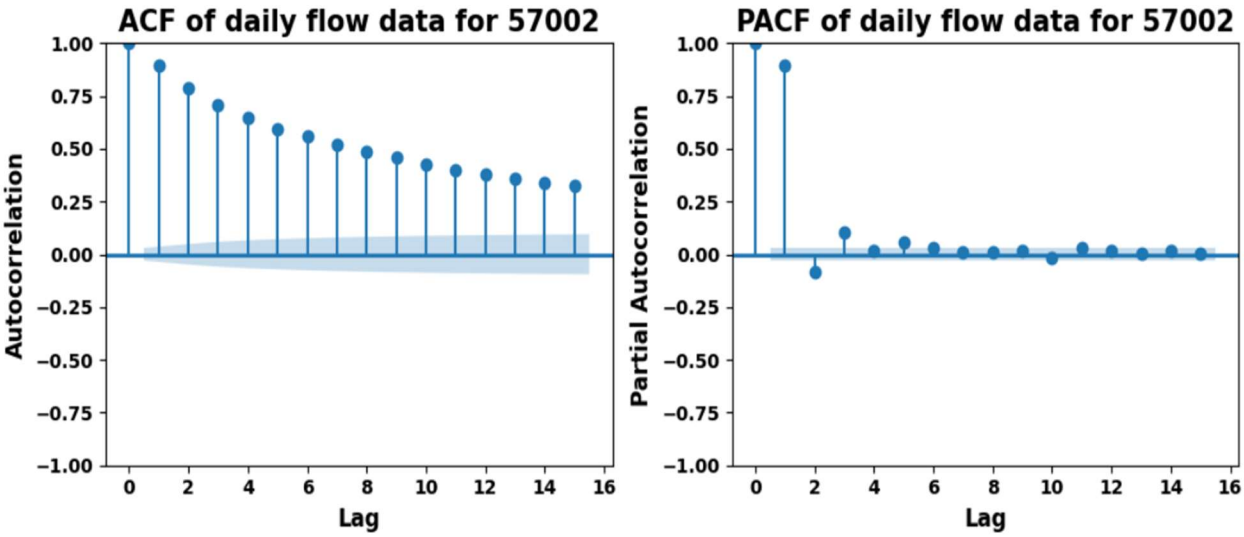

Figure:17.14

3.9. Model Building

3.9.0 Automatic Stepwise Selection of Input Variables

This approach combined the stepwise addition of significant variables ($p < 0.05$) in the model, namely forward selection, followed by the removal of insignificant variables ($p > 0.05$) in the model, namely backward elimination (Smith, 2018) and (James *et al.*, 2021). It goes on until the changes in the variables, whether inclusion or exclusion, happen in a cyclic manner. Although this method is very efficient one of the disadvantages is that it does not examine through all regimes of the variables and thus one might miss the global optimum of the predictor set. Rather, it works on local optimization in the series of variable adding and removing based on the relevance of each. Selection carried out in the last step was directed in such a way that models which model enhance accuracy and minimize complexity.

3.9.1 Linear Regression model

The baseline forecasting approach of river flow in this dissertation employed multiple linear regression to estimate and predict the river flow using a set of input predictors derived from the lagged records of the flow and rainfall in the tributaries. Multiple linear regression is like simple linear regression, but instead of using only one independent variable to predict one variable, the dependent variable that is, several independent variables are used (Gambella, Ghaddar and Naoum-Sawaya, 2021). This model supposes a linear relationship between the input features which contain flow and rainfall data of the

previous days and the output variable that is the river flow on the next day. The relationship can be expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Y: Dependant variable

β0: Intercept

βn: coefficients associated with each independent variable Xn

ϵ: error term

Assumption:

- **Linearity:** To confirm the linearity of the relationships between the independent variables; flow of the previous days and rainfall and the dependent variable next day flow, the residuals versus fitted plot test was employed.
- **Independence:** durbin-watson test was used to test for any kind of autocorrelation in the residuals. If the value here was close to 2 then the residuals were believed to be independent of other observations in the model under test.
- **Homoscedasticity:** The Residuals vs. Fitted Plot also helped in conducting the test for homoscedasticity. A constant spread of residuals across levels of the independent variables suggested that this assumption was satisfied.
- **Normality of Residuals:** The normality of each of the residuals was checked by following Histogram of Residuals and Q-Q Plot.

3.9.2 Support Vector Regression (SVR) Model

In this dissertation the use of Support Vector Regression (SVR) was considered as a more complex model for river flow forecasting. SVR is an extension of the Support Vector Machines (SVM) algorithms that are suited to solve regression problems tasks (Comito and Pizzuti, 2022). This happens through finding a hyperplane that optimizes the data, considering a certain tolerance level which is determined by the parameter, ϵ; the model aim to minimises errors outside of this tolerance level which provide more flexibility in handling non-linear relationships between the variables relationships (Ji *et al.*, 2022).

Key attributes of SVR with major reference to its components are as indicated below.

- **Kernel Function:** SVR works in a way of transforming inputs into a higher dimensional space through kernel functions to represent non-linear relationship. This includes the linear, polynomial and the radial basis function (RBF) kernels. During hyperparameter tuning of this

project, several kernels were analysed in a bid to ascertain which kernel is most suitable for the river flow forecasting for which RBF was suggested to be the most suitable.

- **Regularization Parameter (C):** Determines the trade-off between the goodness of fit to the training data and the complexity of the model. An increase in value of C leads to reduced training error while it has a disadvantage of overfitting. While performing the hypers parameter tuning in this project, several tested C values to get the best performance.
- **Epsilon (ϵ):** Erects the range of error which does not attract any penalties for prediction wrongs. The lower value of ϵ corresponds to higher possibility of getting a better fit on the data while a higher epsilon gives more generalized model. value of epsilon was optimized in such a way that it fitted the actual river flow data during the training while preventing the model from overfitting.

To diagnose the performance of the SVR model, standard residual diagnostics were used like Residual vs Fitted Plot, Histogram of Residuals, actual vs predicted value plot, learning curve, comparison plot of actual test value vs predicted value and cross validation.

3.9.3. Neural Network Model

In this dissertation, the predictor of River flows was developed using the multi-layer feedforward neural network (known as multi-layer perceptron) to capture the non-linear relationship that prevails (Svozil, Kvasnicka and Pospichal, 1997).

The neural network architecture developed for this project consisted of three layers:

- **Input Layer:** The input layer constituted neurons that stood for the chosen input variables that were lagged variables of river flow and rainfall.
- **Hidden Layers:** They are mainly consisting of two hidden layers with 64 neurons in the first layer and 32 neurons in the second layer respectively The ReLU ((Rectified Linear Unit) activation function was applied to apply non-linearity.
- **Output Layer:** The output layer was comprised a single neuron due to the nature of the problem tackled: predicting continuous target measure, namely river flow

The aimed model was trained with Adam optimizer and mean squared error (MSE) was applied as the loss function. To reduce overfitting, early stopping was used where the training process is interrupted the moment that there is no enhancement in the validation loss for the next ten epochs.

Feature scaling was performed using MinMaxScaler to ensure that the data used was standardized. Techniques such as training and validation loss graphs were used to check convergence and the actual test data, and the river flow value was compared with the predicted value for aesthetic analysis.

3.9.4 Generalized Additive Model (GAM)

The next model used to handle non-linear relationships between inputs and target variable that was GAM model. GAM are extended version of linear models in which each input variable can have its own smooth, non-linear function (Dubos *et al.*, 2022). The model structure is given by:

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots f_n(x_n) + \epsilon$$

where Y is the river flow, fn(Xn) represents the smooth functions applied to the predictor variables, and ϵ is the error term.

The GAM was then fitted using the LinearGAM function within R and a lambda parameter is used to adjust the level of smoothing of the penalized splines in the model. In GAM applied in this study, the smoothing parameter λ was 2. 0 was used to obtain a better balance between the flexibility of the model and its smoothness. A higher λ provides a better fit hence reducing the chances of overfitting while a lower λ increases the size of the model hence the chances of overfitting.

Limitation: GAM model provides more flexibility to capture non-linearity, but it assumes that these relationships are smooth. This can be limiting when dealing with number of variable datasets, like river flow, where abrupt changes may occur.

3.10 Model Evaluation Metrics

In this dissertation, models performance was evaluated using several key metrics: which are; R-squared (R^2), Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean absolute percentage error (MAPE).

- **R-squared (R^2)** is the degree of variation in the dependent variable explained by independent variables where the greater the value, the better the fit.
- **MSE** computes the average squared deviation of the difference between the observed and predicted values with better performance indicated by smallest values.
- **MAE** that involves the computation of average magnitude of errors in absolute value is less volatile than MSE in case of presence of outliers in the model.

- **MAPE** presents errors in terms of percentage hence can be used to compare the model's performance even across different scales.

3.11 Evaluation of SVR using cross-validation method.

Cross validation techniques where 5-fold cross validation used in this dissertation in a bid to check the ability of the Support Vector Regression (SVR) model for generalization. The data set was split into five sets of data, the system trained was trained on 4 sets and tested on the 5th set. this process was repeated five times, with each fold serving as the test set once. The cross-validation gave out other measures like (MSE), (R^2) and (MAPE) which gave a more robust assessment of the accuracy of the elaborated model. While cross-validation principle has been adopted for the purpose of model evaluation, the actual training of the final model was not affected by it. Instead, the model was trained on the full dataset after cross-validation to ensure its performance on unseen data.

3.12. Model Optimization and Hyperparameter Tuning

When choosing the best SVR model, three methods of preprocessing data before entering into the model: StandardScaler, RobustScaler, MiniMaxScaler. Among them MinMaxScaler gave better solution. This method normalises data to a specified range (between 0 to 1) which benefits the model as some kernel functions of SVR such as RBF improve with scaled data. However, StandardScaler and RobustScaler could not result in the same level of enhancement on the same data and with the same SVR model because of the impact of different scales of features.

Hyperparameter tuning was done by RandomizedSearchCV as opposed to grid search due to the exploration of a wider solution space in one search. Random search was chosen due to computational complexity since the method does not search for every combination but randomly evaluates them. This made it possible to fine-tune the C, epsilon, and gamma values to produce a definite improvement on the SVR model when used in next-day river flow prediction.

4. Result and Comparison of Models

4.1 Split of Data for Training and Testing

The obtained cleaned dataset was split 70-30 where the 70% was used in model building and fine-tuning while the 30% was used to evaluate the model on unseen data approach enabled the assessment of how well the models generalize as well as the accuracy of the models.

4.2 Model Experiment

The current research used to develop predictive models for river flow predictions using three experimental sets. In the first experiment, log-transformed data was used to create models. In the second, seasonal influences were removed from the log-transformed data by applying de-seasonalization, the third experiment used the Box-Cox transformation in an attempt to further normalize the data. These configuration setups were aimed to measure the effects of the different transformation methods and preprocessing strategies on the models.

4.2.1 Variable selection at Log transformation data

According to the PACF plots, Catchment Daily Rainfall (CDR) requires 4 lags, and 5 lags are needed for Gauged Daily Flow (GDF) in the modelling process. These lagged values were then used in the model and the stepwise technique for the selection of the appropriate input variables for building the model was applied. The automatic stepwise selection technique identified the following significant variables for use in the models**:** 'gdf_57004_lag_1', 'gdf_57007_lag_2', 'gdf_57006_lag_4', 'cdr_57004_lag_1', 'gdf_57006_lag_1', 'cdr_57001_lag_2', 'gdf_57006_lag_5', 'cdr_57015_lag_4', 'cdr_57005_lag_1', 'gdf_57004_lag_2', 'gdf_57007_lag_5', 'gdf_57015_lag_5', 'gdf_57015_lag_1', 'gdf_57007_lag_1', 'cdr_57005_lag_4', 'gdf_57002_lag_2'.

4.2.1.1 Linear Regression (LR) model at log transformed data

The AIC of the model is 2371, which is quite acceptable where lower values suggest better results. There was no overfitting as seen in both the training and the testing phase with the test set having R-squared of 0.8827. This result metric is presented in the table below:

Set	R-squared	MSE	MAPE
Training result	0.84977	0.12885	0.08209
Testing result	0.88270	0.10052	0.07423
Cross-validation	0.88000	0.10254	0.07305

Cross-validation, using a 5-fold approach, confirmed this consistency, with cross-validated R-squared aligning closely with the test set. However, the model does not meet all linear regression assumptions. Figures 18.2 and 18.3 reveal deviations from normality in the Q-Q plot and right-skewness in the

residuals. The Durbin-Watson statistic (1.94479) suggests independence of residuals is satisfied which close to 2.0.

The residuals vs. fitted plot (Figure:18.2, Page:75) shows an even distribution around zero, affirming the linearity assumption. (Figures:18.4-18.5, Page:76) confirm no significant autocorrelation, and the comparison of actual vs. predicted values (Figures:18.6-8.7, Page:76) reflects good agreement.

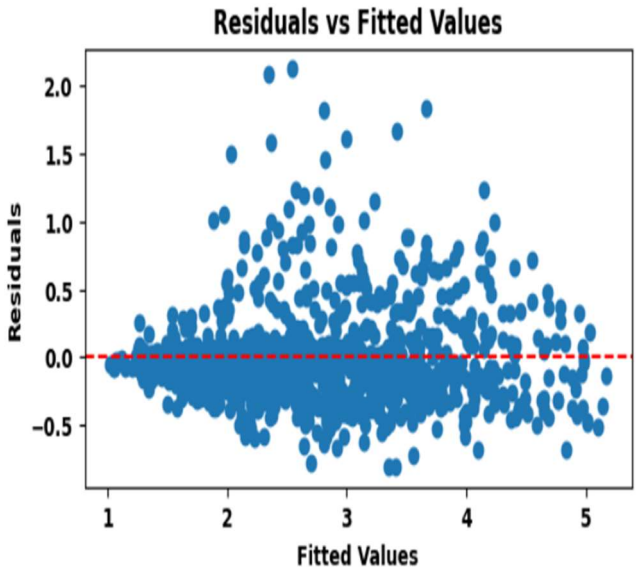

Figure:18.1

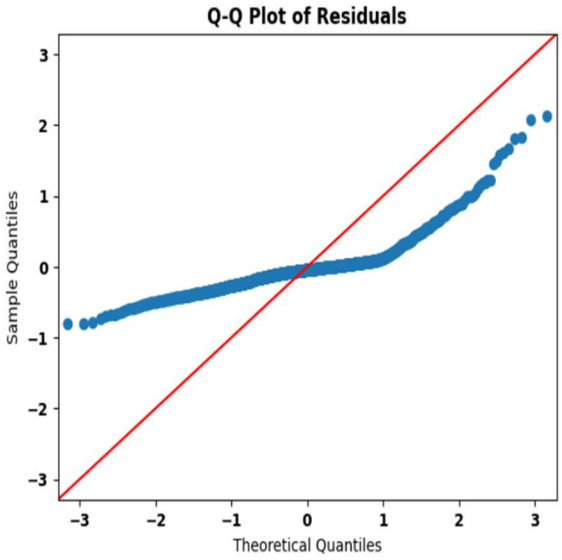

Figure:18.2

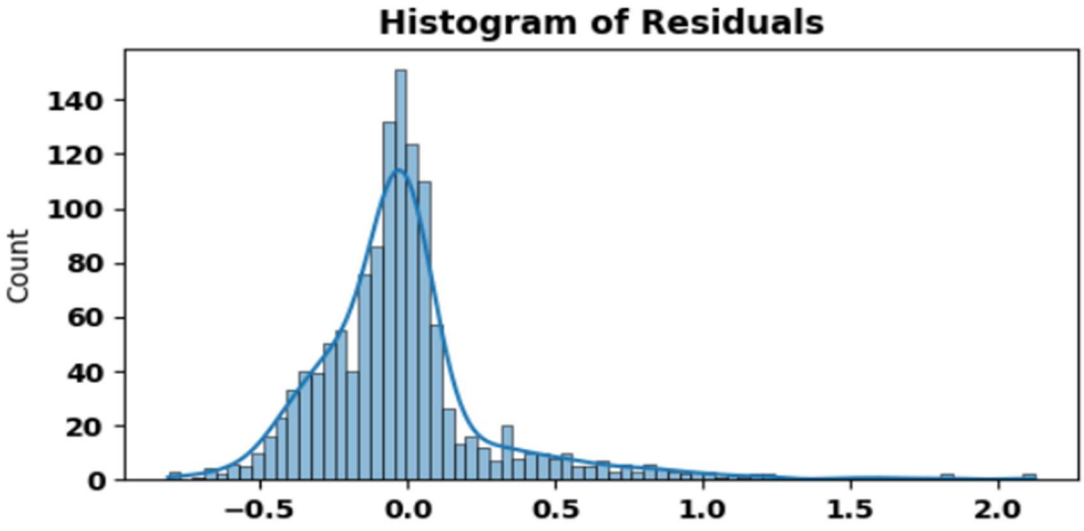

Figure:18.3

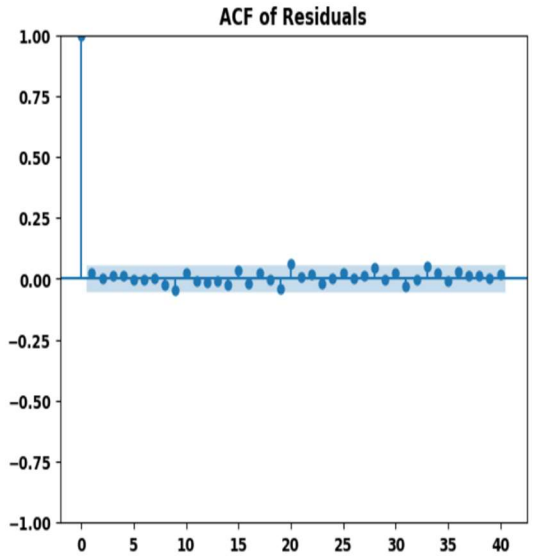

Figure:18.4

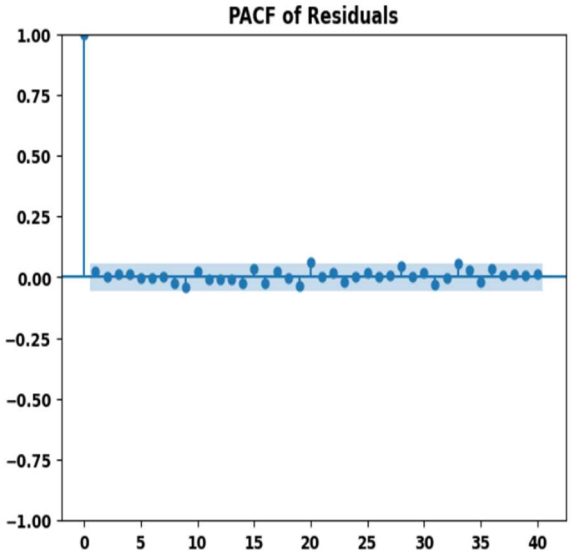

Figure:18.5

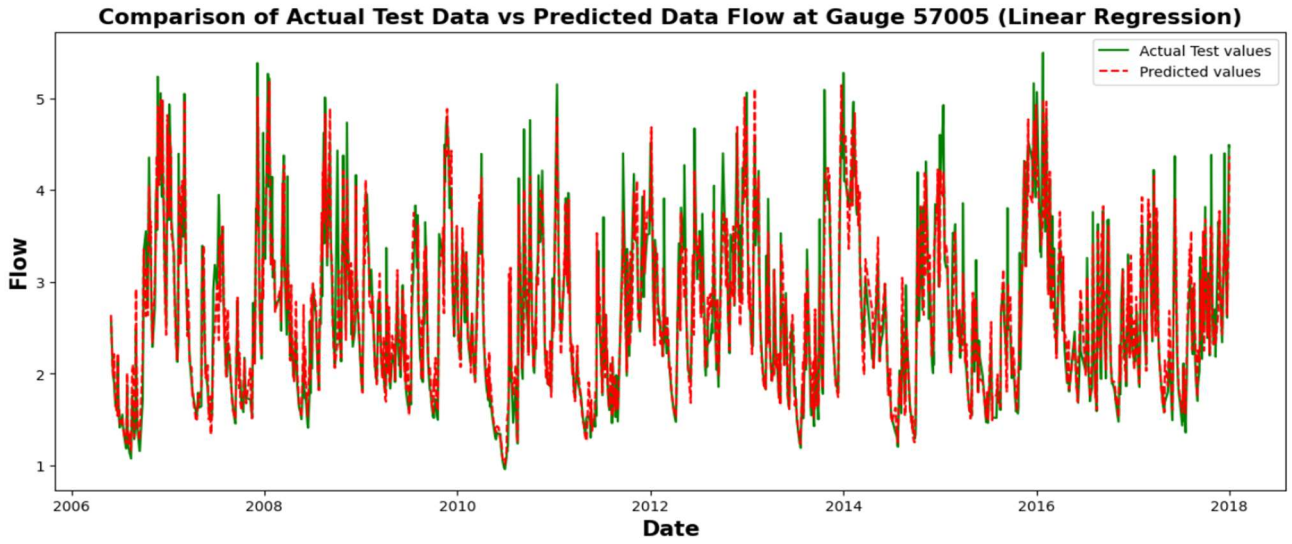

Figure:18.6

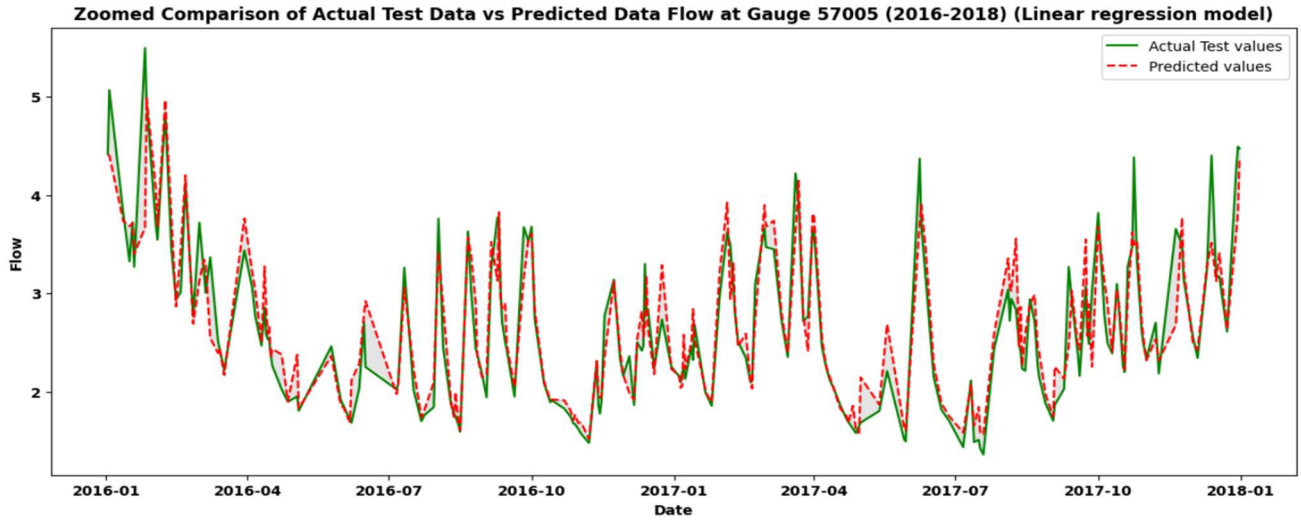

Figure:18.7

4.2.1.2 Support Vector regression (SVR) model at log transformed data

The support vector regression (SVR) model was applied to the log-transformed data to forecast river flow. The model used a radial basis function (RBF) kernel with hyperparameters set as follows: C=1.0, ϵ=0.1, gamma = scale. These hyperparameters were chosen to balance the trade-off between model complexity and error margin, with the RBF kernel capturing non-linear relationships in the data. The model exhibited strong performance on both training and test datasets, with no evidence of overfitting. R-squared is also slightly better than baseline Linear regression model. Cross-validation (5-fold) results further validated the model's generalizability. The key performance metrics are as follows:

	R-squared	MSE	MAPE
Training result	0.87093	0.11070	0.06184
Testing result	0.88603	0.09766	0.06031
Cross-validation	0.85755	0.12171	0.06662

Although the SVR model demonstrated good performance, the learning curve (Figure:19.1, Page:77) shows minor discrepancies between the training and cross-validation errors, suggesting the possibility of enhancing the model with more data. The plot of residuals against predicted values (Figure:19.2, Page:77) indicates the absence of any discernible patterns, which provides evidence in favour of the assumption of linearity. However, the histogram of residuals (Figure:19.3, Page:78) shows a mild right skew, Actual test value vs predicted value plot (figure:19.4, Page:78); minor deviations are visible but do not significantly affect the overall prediction. As shown in (Figure:19.5-19.6, Page:79) slightly better prediction observed as compared to previous LR model.

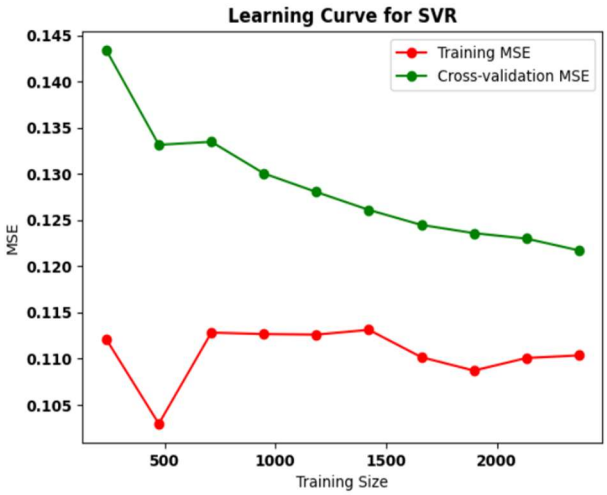

Figure:19.1

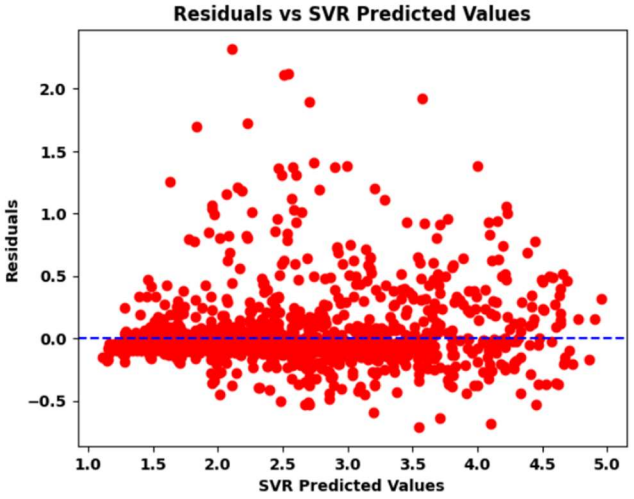

Figure:19.2

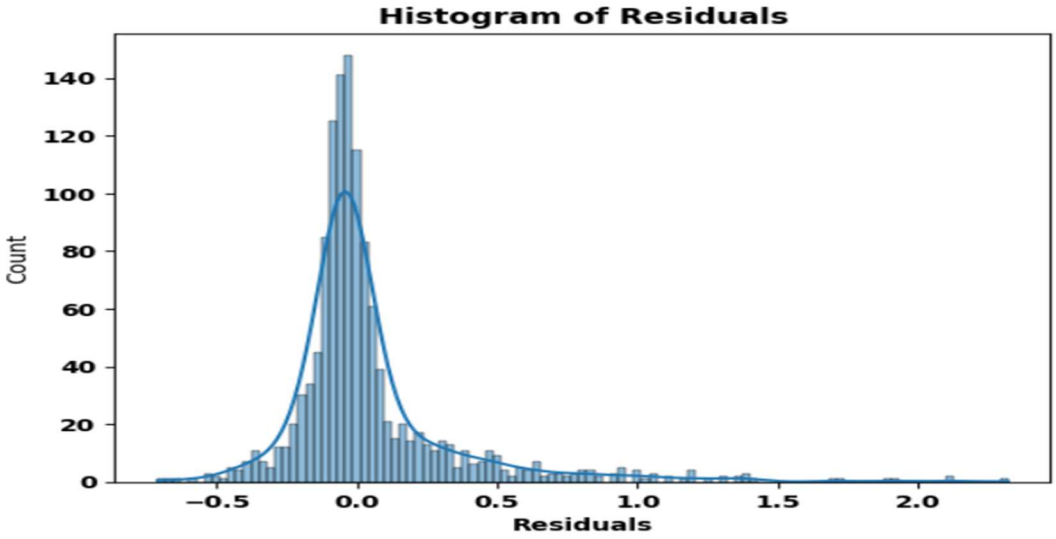

Figure:19.3

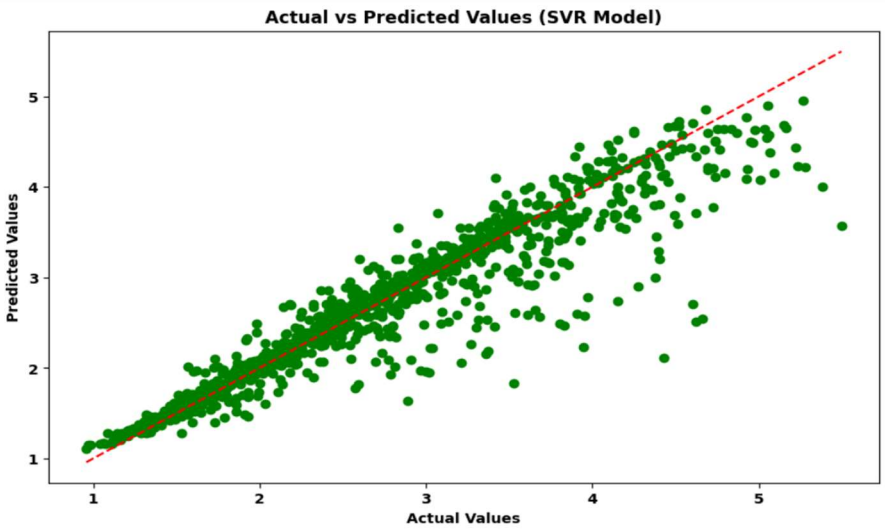

Figure:19.4

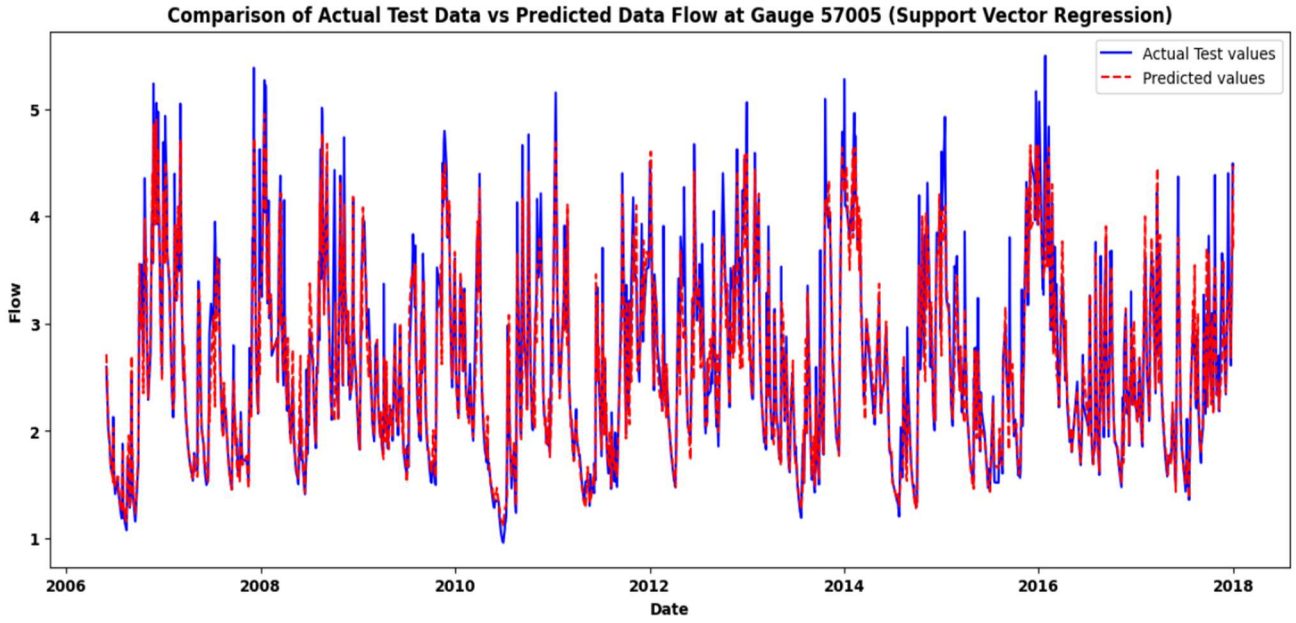

Figure:19.5

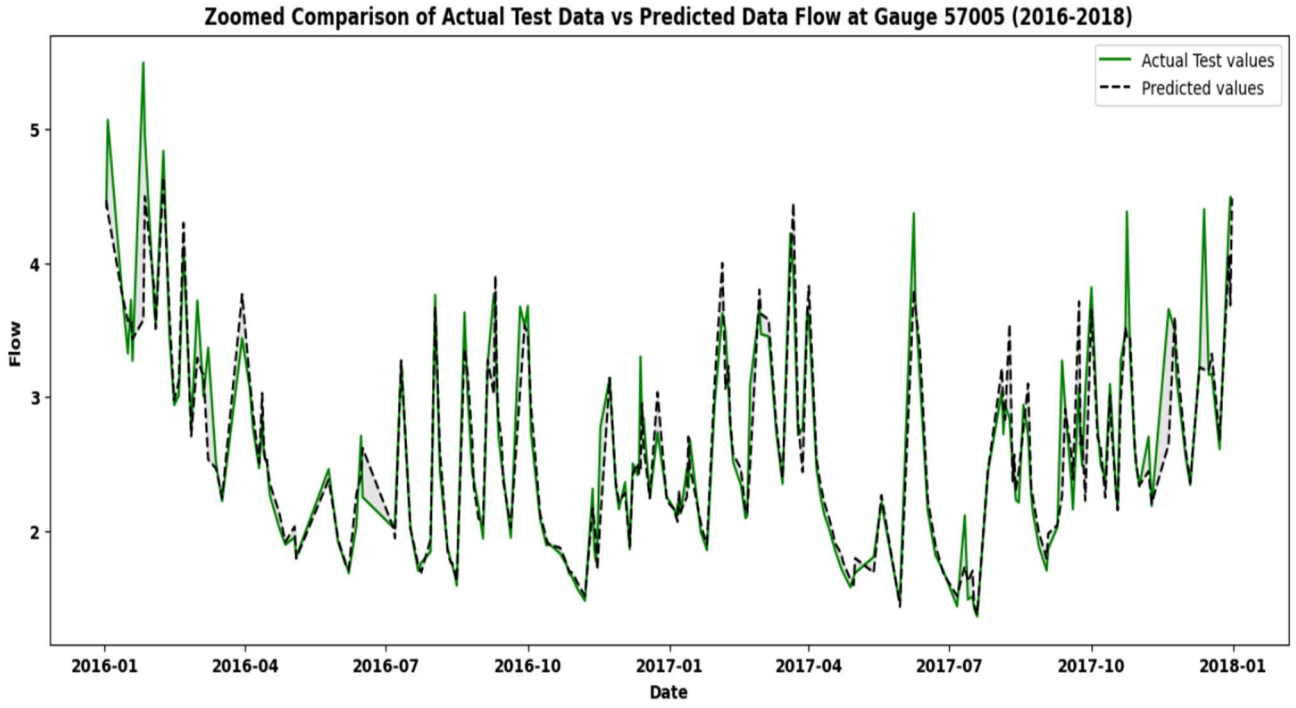

Figure:19.6

4.2.1.3 Neural Network (NN) model at log transformed data

The neural network (NN) model applied to the log-transformed data yielded strong performance metrics. The key performance metrics are as below in table demonstrating good generalization

	R-squared	MSE	MAPE
Training result	0.85815	0.12166	0.07603
Testing result	0.88534	0.09825	0.06872

The learning curve (Figure:20.1, Page:80) shows convergence between training and validation losses, indicating no overfitting. The actual vs. predicted plots reveal close alignment between the predicted and observed values, further confirmed by the zoomed-in comparison for the 2016-2018 period (Figure:20.3, Page:81), showing consistent accuracy in forecasting river flow at gauge 57005.

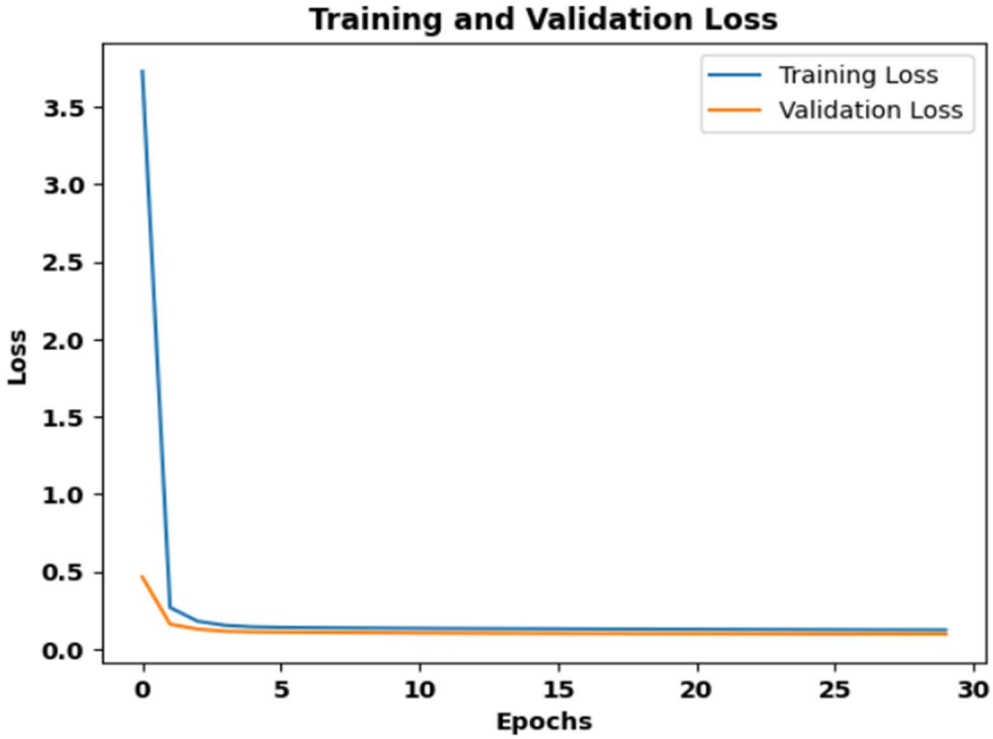

Figure:20.1

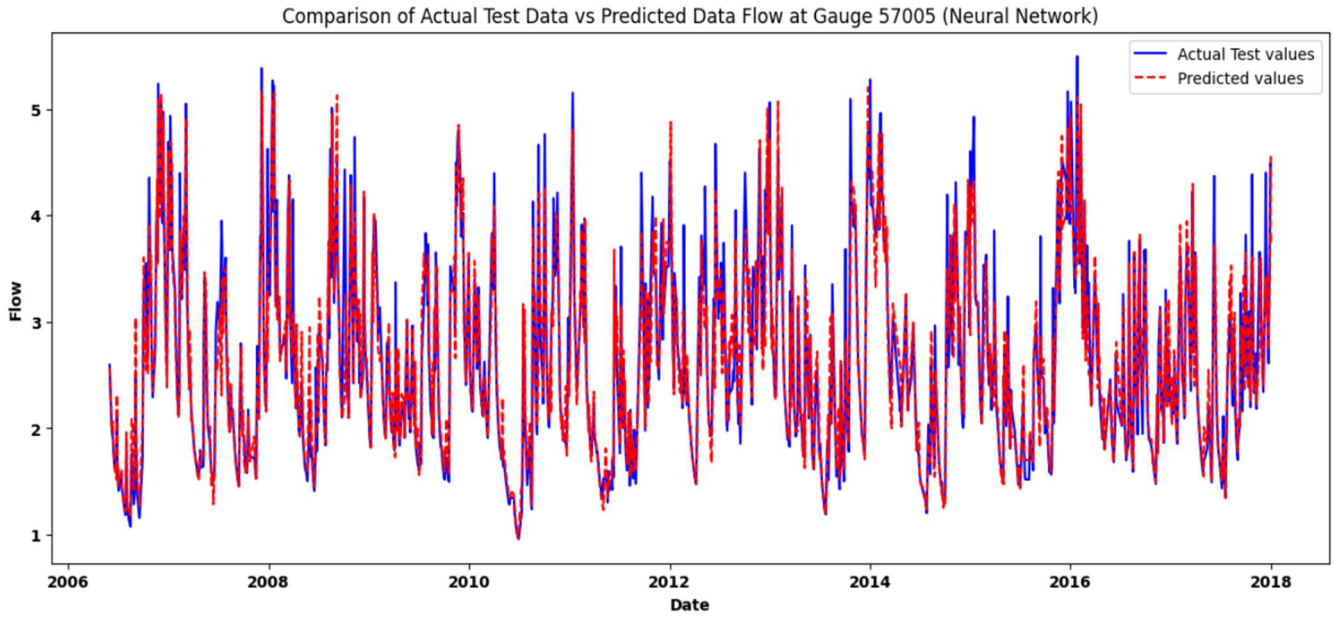

Figure:20.2

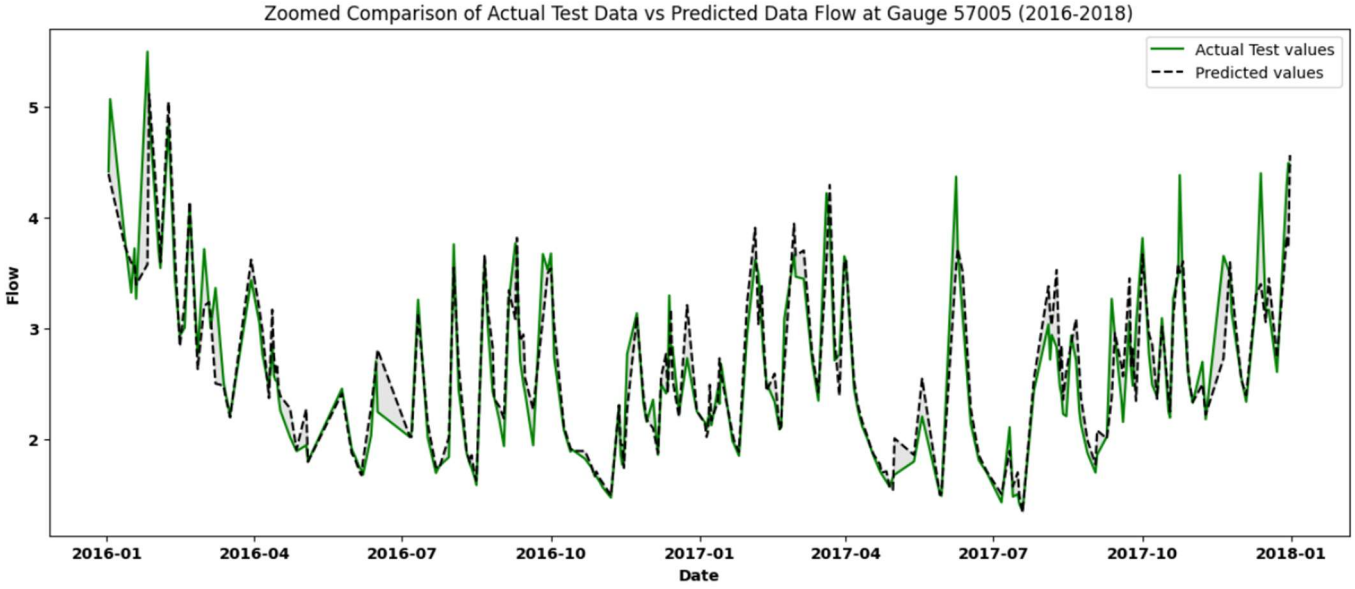

Figure:20.3

4.2.1.4 Generalised additive model (GAM) at log transformed data

The log-transformed data was analysed using the Generalised Additive Model (GAM). The performance of the Generalised Additive Model (GAM) can be summarised as follows:

	R-squared	MSE	MAPE
Training result	0.87513	0.10710	0.07567
Testing result	0.87801	0.10454	0.07557

These measures demonstrate that the model exhibits comparable performance on both the training and test data, suggesting the absence of overfitting. The scatter plot of residuals against predicted values (figure 21.1) does not exhibit any discernible trend, indicating that the model effectively captures the majority of the information included in the data. The plot (figure:21.2) demonstrates a significant connection between the predicted and actual test values, providing additional evidence for the model's validity.

In addition, the histogram of residuals (figure:21.3) verifies that the residuals exhibit a near approximation to a normal distribution, albeit with a minor rightward skew. The contrast between the actual test data and the predicted data (figures:21.4) illustrates that the model accurately captures the underlying trend. (Figure:21.5), which provides a close-up comparison for the years 2016-2018, further demonstrates the GAM's ability to accurately estimate the flow throughout this time frame.

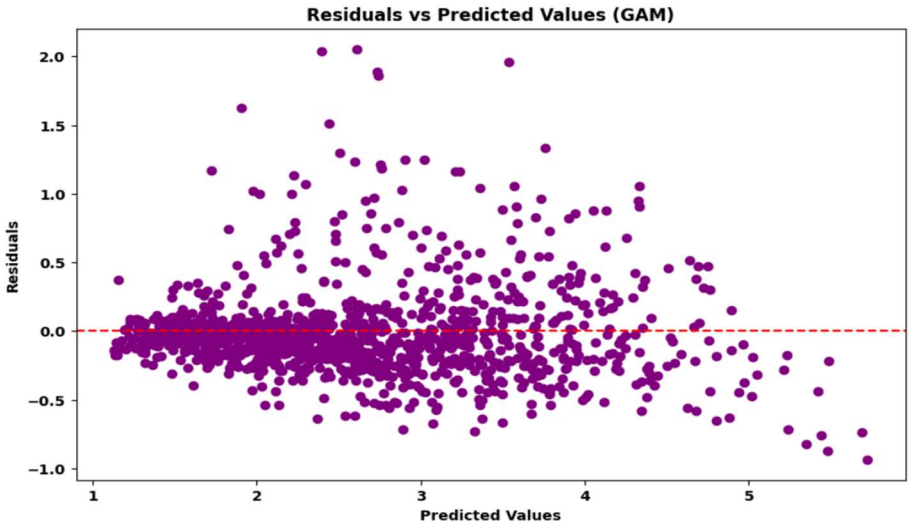

Figure:21.1

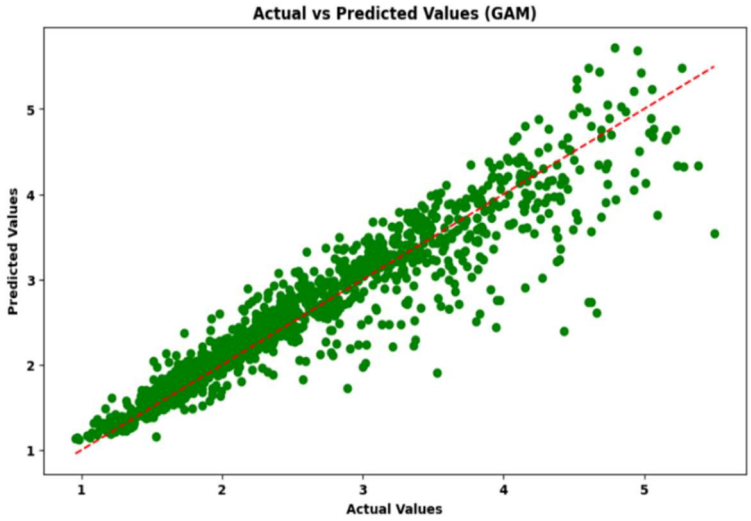

Figure:21.2

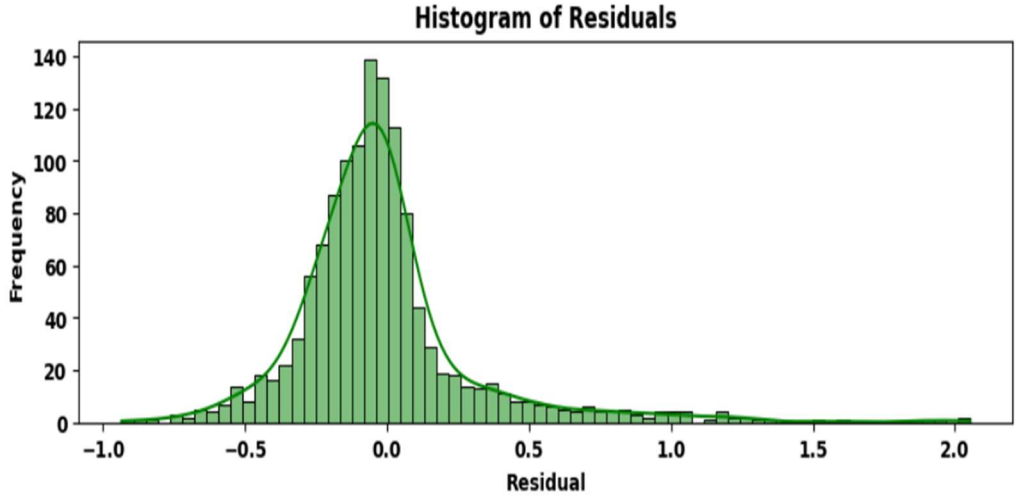

Figure:21.3

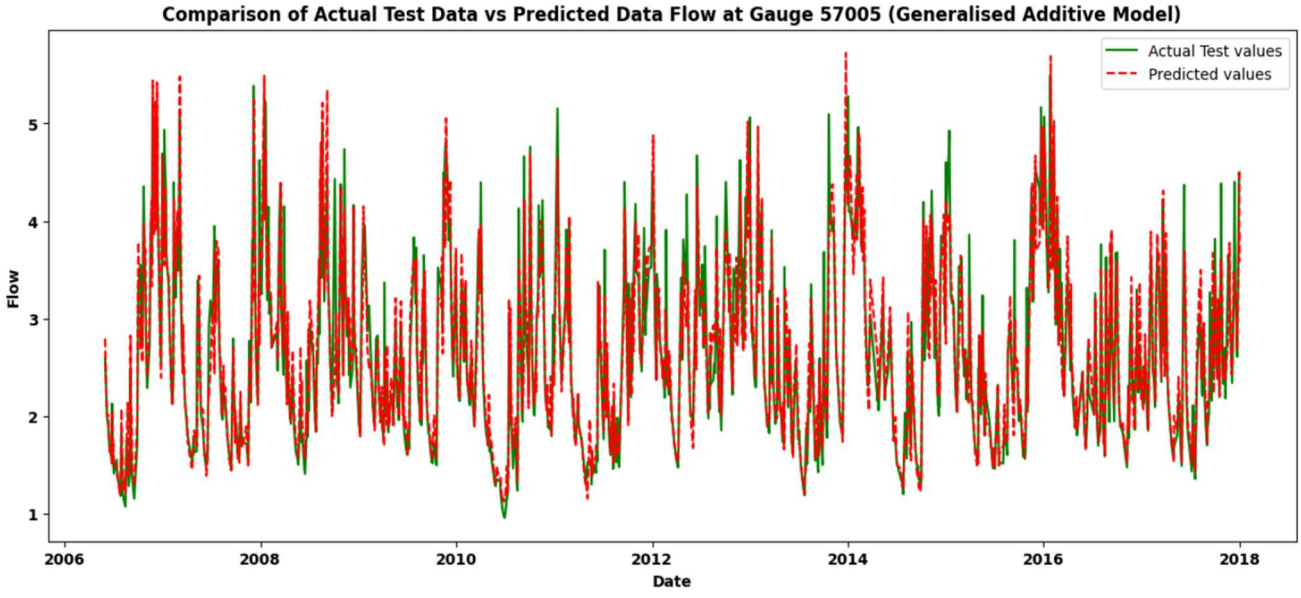

Figure:21.4

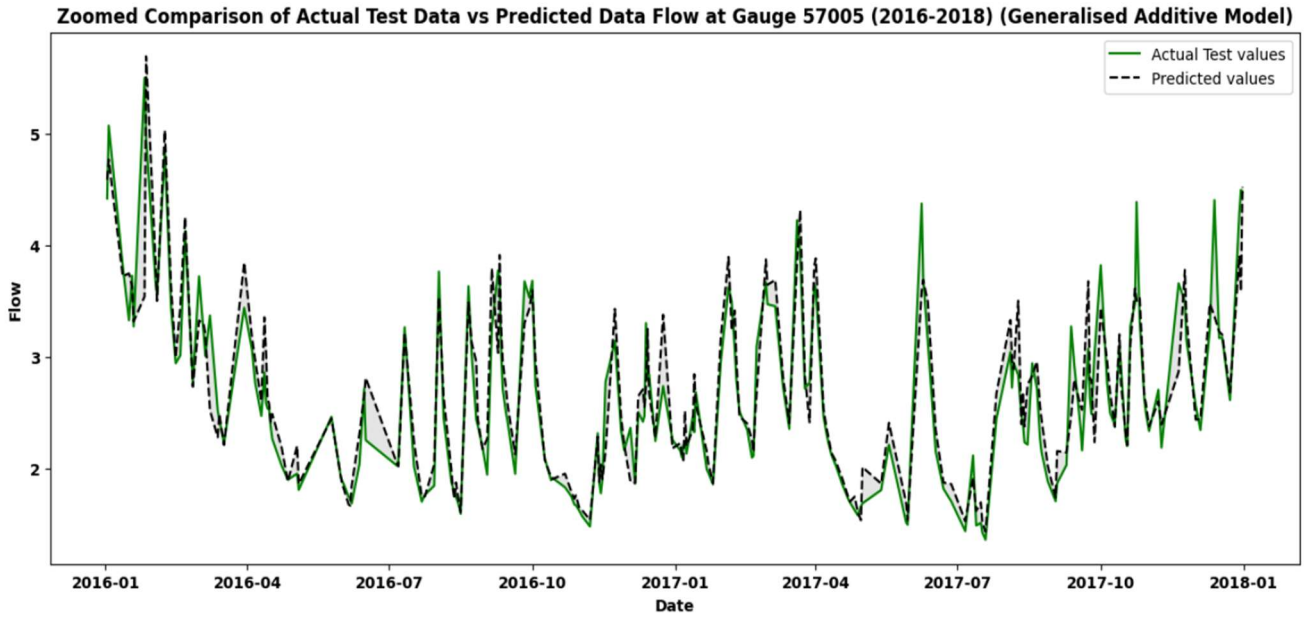

Figure:21.5

4.2.1.5 Model comparison on log transformed data

The comparison of models on the log-transformed data (Table, Figures:22.1-22.2, Page:85) shows that the (SVR) model performed the best, with the lowest MAE (0. 174208) and MAPE (0. 060309). The Neural Network (NN) followed with an MAE of 0. 192524 and MAPE of 0. 068723, while Linear Regression (LR) showed moderate performance The Generalized Additive Model (GAM) had the highest error rates making it the least effective. These results highlight SVR as the most suitable model for river flow prediction.

(Figure:22.3, Page:86) shows a zoomed comparison of actual vs. predicted values for all models over 2016-2018. The SVR model closely follows the actual values, especially during peak flows, supporting the results in Table and Figures:22.1-22.2, where SVR outperformed the other models.

Model	MAE	MAPE
LR	0.202526	0.074229
SVR	0.174208	0.060309
NN	0.192524	0.068723
GAM	0.207902	0.075571

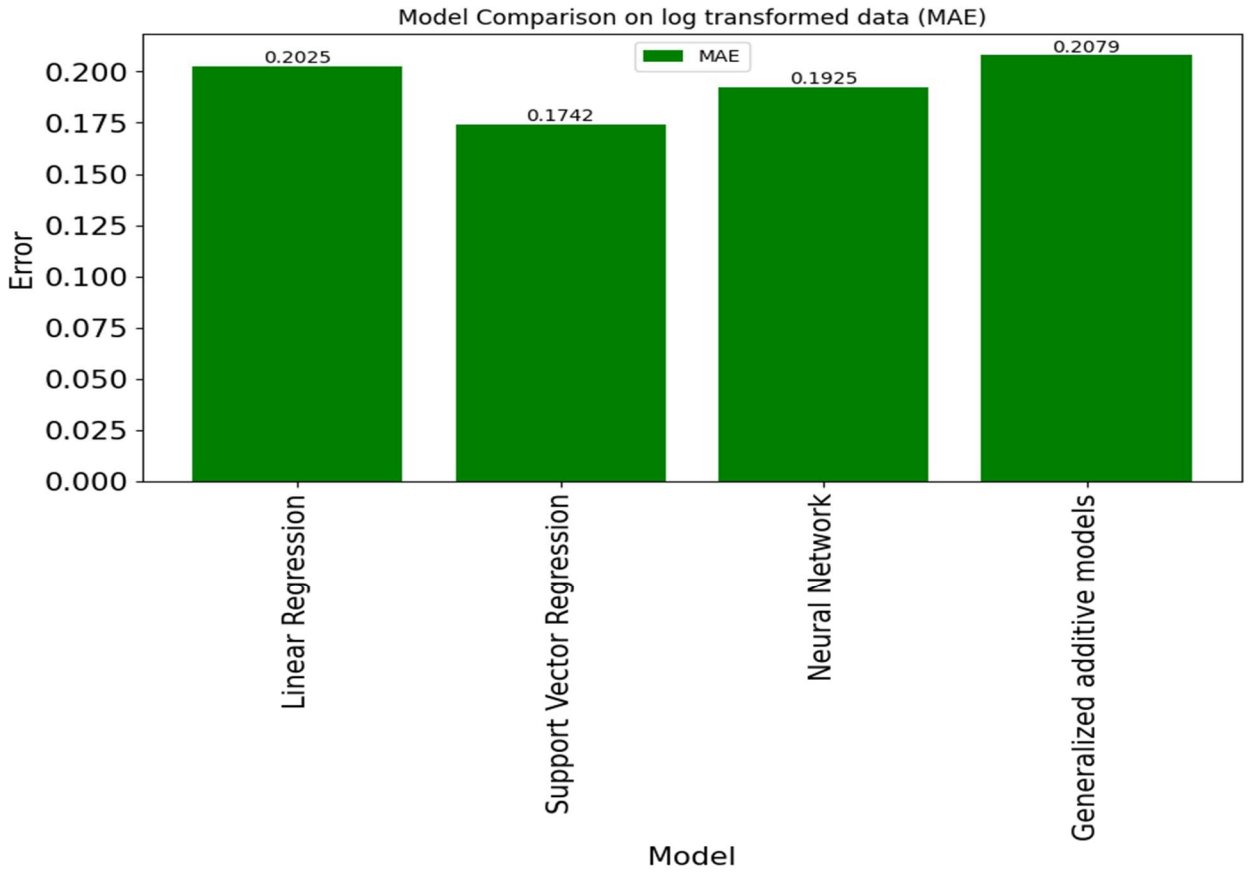

Figure:22.1

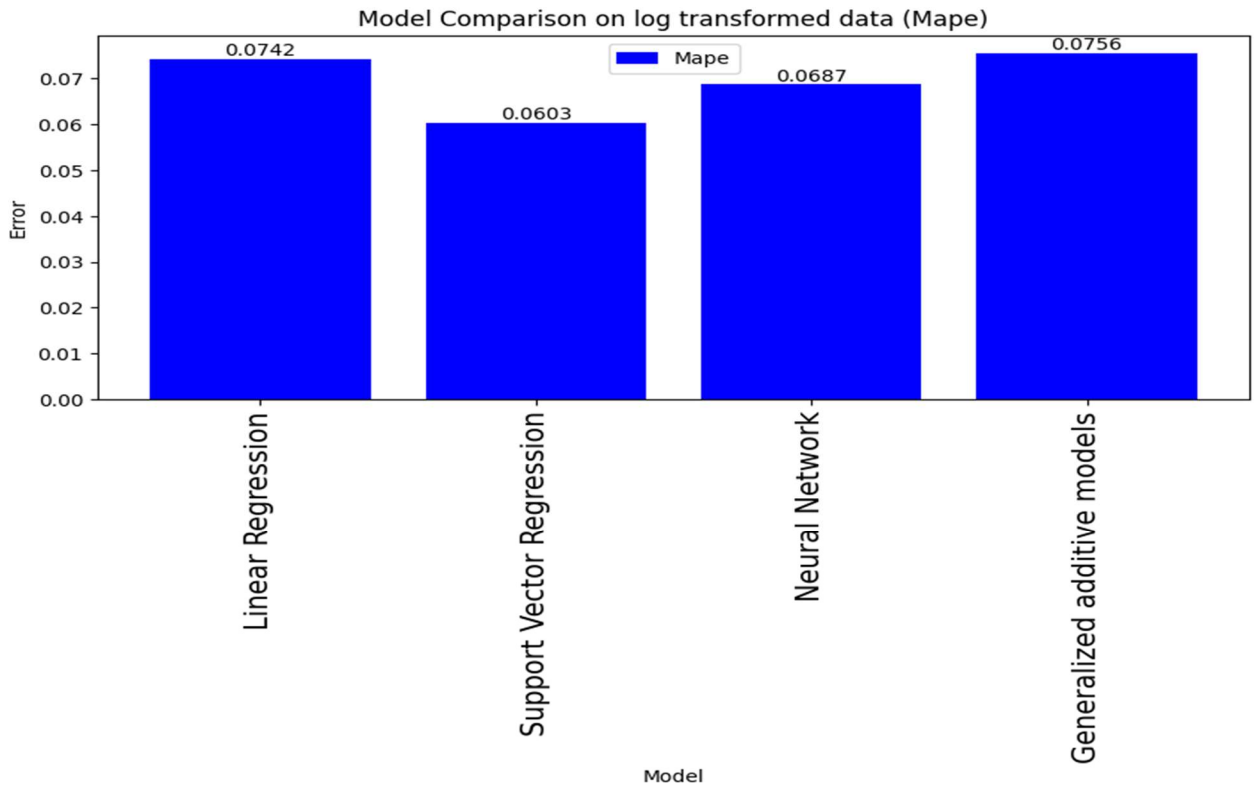

Figure:22.2

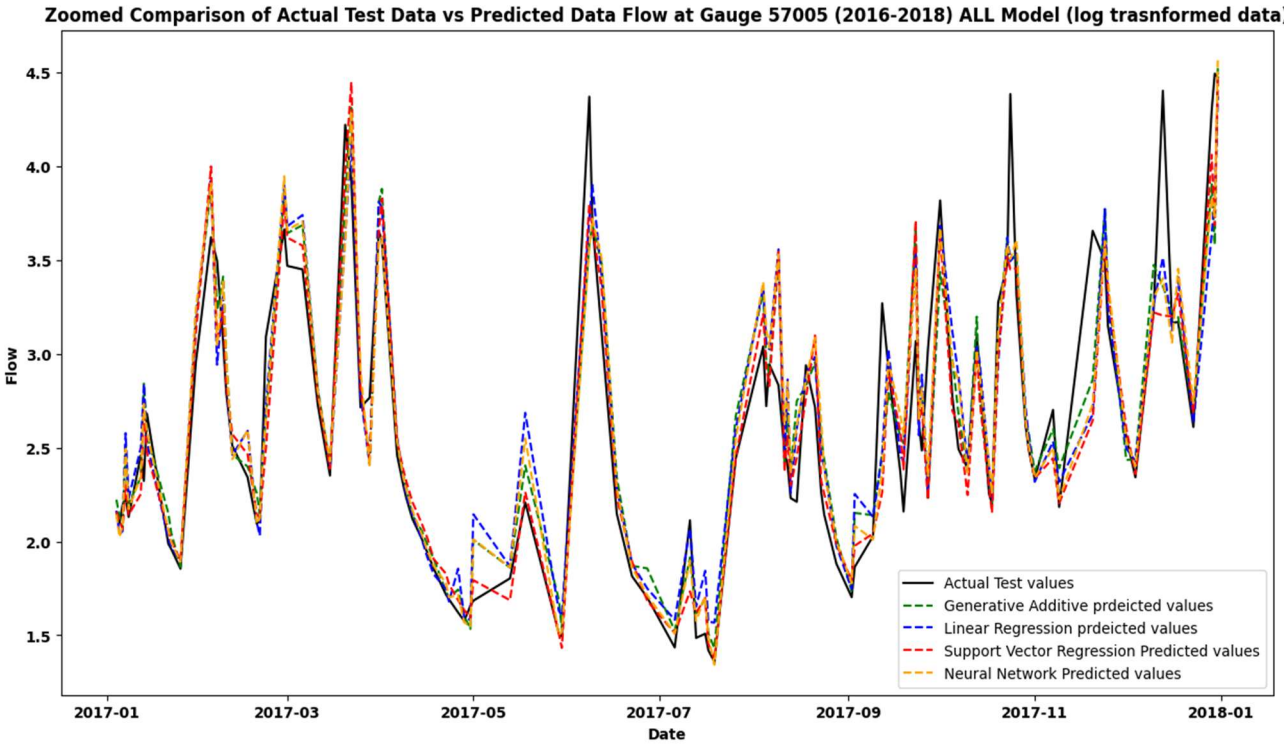

Figure:22.3

4.2.1.6 Model optimization of SVR model by features scaling and hyperparameter tuning

It was decided for concentrating on the optimisation of the SVR model was based on its superior performance in prior comparisons, where it demonstrates lower MAE and MAPE values compared to other models. Consequently, further studies were conducted exclusively on SVR to investigate its potential for enhancing next-day flow forecasting.

Three different scaling methods—Standard Scaler, Robust Scaler, and Min-Max Scaler—were tested. Among these, the Min-Max Scaler yielded the best improvement in model performance. Result show in below table:

SVR Model Comparison After Features Scaling				
	R-squared	MSE	MAE	MAPE
Standard Scaler	0.88554	0.09808	0.17563	0.06085
Robust Scaler	0.88516	0.09841	0.17659	0.06113
Minmax Scaler	0.88596	0.09772	0.17384	0.06023

After hyperparameter tuning using the random search technique, the best parameters identified for the SVR model **were 'C': 52.23487, 'epsilon': 0.01039, 'gamma': 0.26178, and 'kernel': 'rbf'**. These adjustments improved the model's performance, reducing **MAE** to **0.16932** and **MAPE** to **0.05704.** (Figure:23.1, Page:23.1) illustrates the closer alignment between predicted and actual test values after optimization, reflecting the enhanced accuracy in flow forecasting.

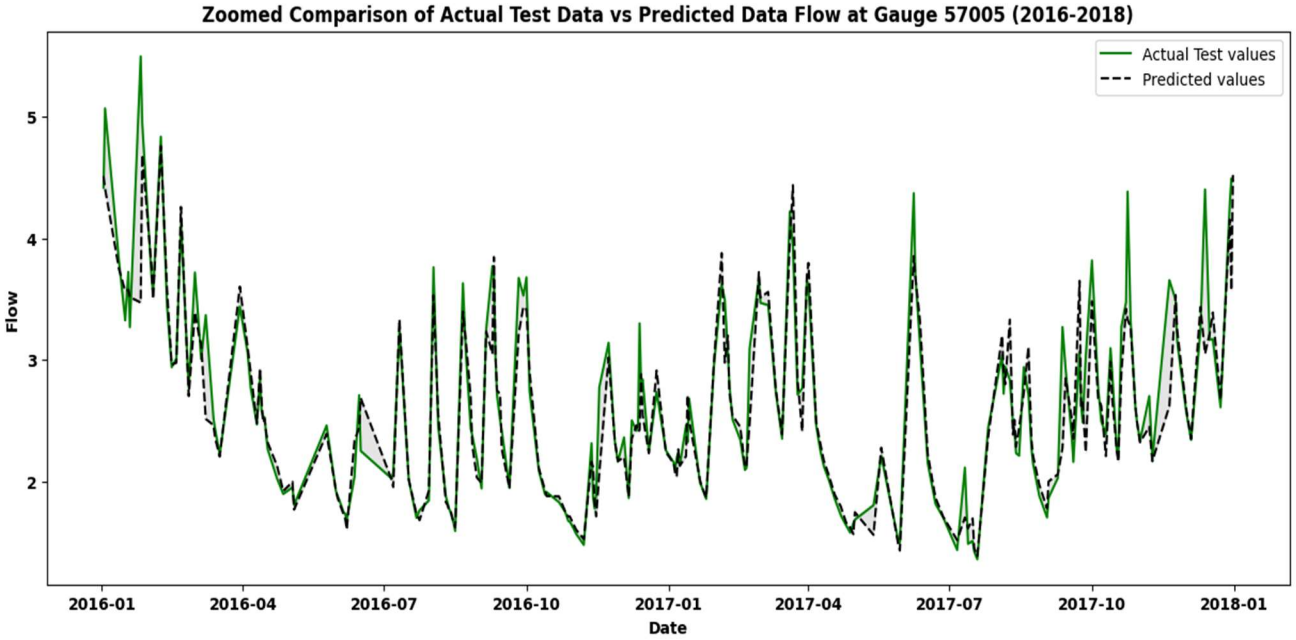

Figure 23.1

4.2.2 Variable Selection at Log transformation with de-seasonalized data

Again, conducted stepwise selection techniques with log transformation followed by de-seasonalised data that identified the following significant variables for use in the models: 'gdf_57004_lag_1', 'gdf_57007_lag_2', 'gdf_57006_lag_4', 'cdr_57004_lag_1', 'gdf_57004_lag_5', 'gdf_57006_lag_1', 'cdr_57007_lag_2', 'gdf_57006_lag_5', 'cdr_57015_lag_4', 'gdf_57002_lag_1', 'gdf_57004_lag_2', 'cdr_57005_lag_1', 'gdf_57002_lag_4'.

4.2.2.2 Support Vector regression (SVR) model at log transformed + de-seasonalised data

When the model was trained on this dataset, the R-squared value improved compared to the previously optimized model. However, both MAE and MAPE increased, indicating a rise in the overall average prediction error. This suggests that while R-squared showed improvement, the model's accuracy in minimizing average errors was not enhanced through MinMax scaling and hyperparameter tuning

when compared to the performance of the log-transformed optimized model. The results are summarized in the table below.

SVR model trained at de-seasonalised data				
Followed step for SVR model	R-squared	MSE	MAE	MAPE
1st Attempt	0.89210	0.09174	0.18115	0.06532
2nd Minmax scaling	0.89214	0.09170	0.18115	0.06532
3rd hyperparameter tuning	0.88983	0.09367	0.16788	0.06422

4.2.3 Variable Selection after applied box-cox transformation

Again, conducted stepwise selection techniques with applied box-cox transformation that identified the following significant variables for use in the models: 'gdf_57015_lag_1', 'gdf_57004_lag_1', 'cdr_57001_lag_2', 'gdf_57007_lag_5', 'gdf_57006_lag_1', 'gdf_57007_lag_2', 'cdr_57004_lag_1', 'gdf_57007_lag_1', 'gdf_57006_lag_5', 'gdf_57004_lag_4', 'gdf_57015_lag_3', 'gdf_57007_lag_3', 'gdf_57004_lag_2', 'gdf_57007_lag_4'.

4.2.3.1 Support Vector regression (SVR) model after applied box-cox transformation

When the model was trained on Box-Cox transformed data, the R-squared value improved compared to the previously optimized model. Additionally, both MAE and MAPE decreased, indicating that the model better explains the variance while also reducing the average prediction error. In comparison to the models trained on log-transformed and de-seasonalized data, the Box-Cox transformed model performed better in all aspects, including MAE, MAPE, and R-squared. With the application of Min-Max scaling and hyperparameter tuning, the model was further optimized using the following parameters: **'C': 52.23487, 'epsilon': 0.01039, 'gamma': 0.39610, 'kernel': 'rbf'**. The results of this comparison are summarized in the accompanying below table. (Figure:24.1, Page:89) provides a zoomed-in view of the comparison between the actual test data and predicted values, highlighting the model's improved performance. Learning curve shown in (Figure:24.2, Page:89), no sign of overfitting at last optimized model. Additionally actual vs predicted plots (figure:24.3, Page:89) evident to better generalising model.

SVR model trained after applied Box-cox transformation				
Followed step for SVR model	R-squared	MSE	MAE	MAPE
1st Attempt	0.90368	0.01146	0.07820	0.04827
2nd Minmax scaling	0.90164	0.01142	0.07877	0.04890
3rd hyperparameter tuning	0.90889	0.01084	0.05985	0.03510

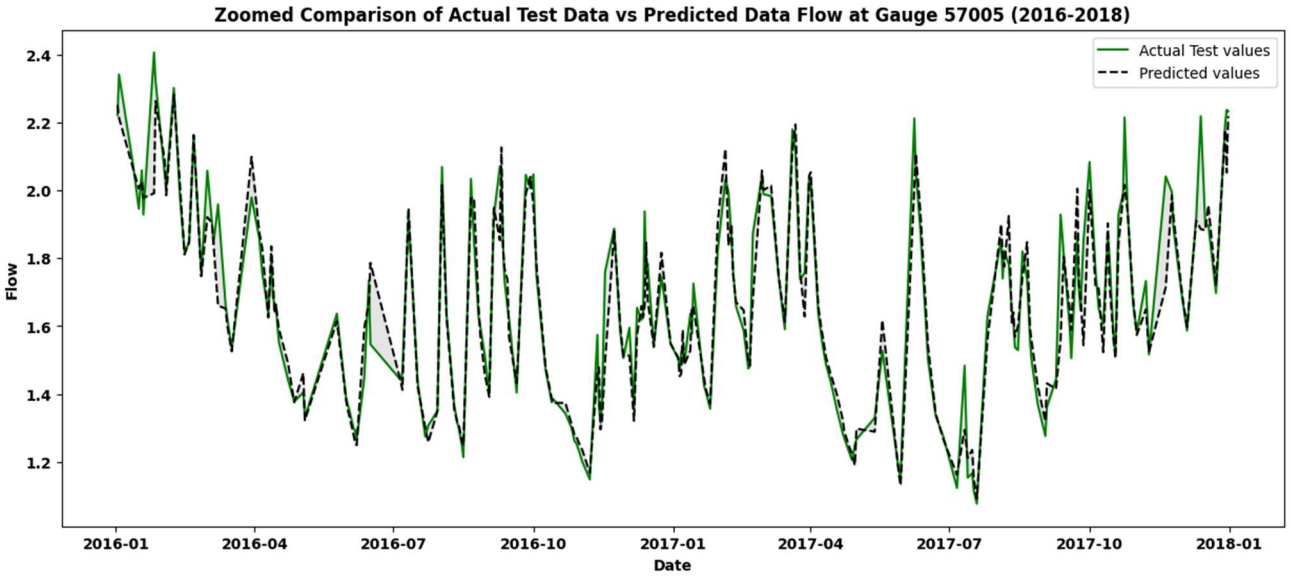

Figure:24.1

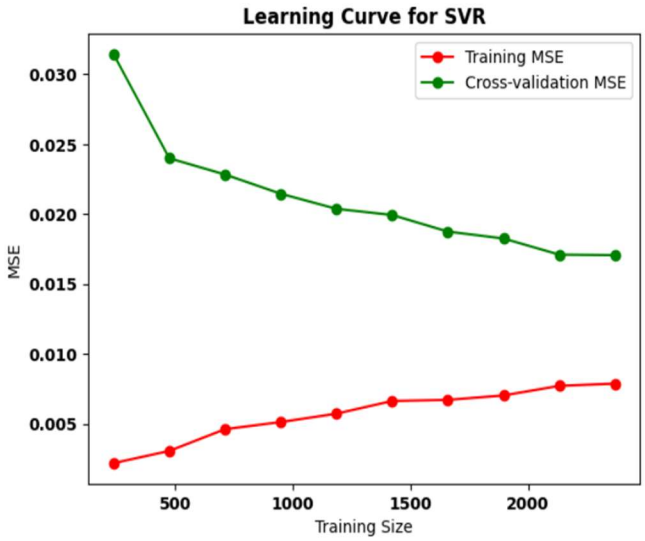

Figure:24.2

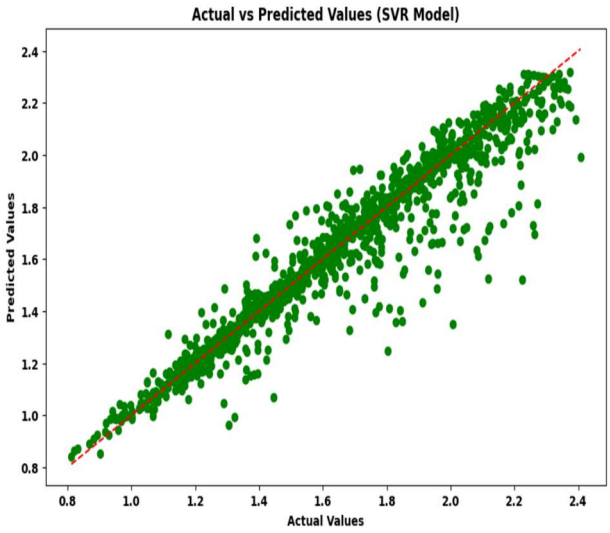

Figure:24.3

4.3 Model comparison of all optimized SVR model

Model trained on SVR model by using random search hyperparameter method by find finding best parameters, in which model trained at box-cox transformed data is performed outperforming, comparison as shown in below table.

Comparison of Optimized SVR model by Hyperparameter tuning		
Followed step for SVR model	MAE	MAPE
Log transformed data	0.16932	0.05704
De-seasonalized data	0.18014	0.06422
Box-cox transformed data	0.05985	0.03510

4.4 Next Day flow Prediction

The SVR model trained on Box-Cox transformed data outperformed all other models and was selected for next-day river flow forecasting. The predicted flow value for the next day (2018-01-01), where the cleaned dataset concluded, was 81.60591 on the original scale.

5. Conclusion

The effectiveness of the SVR model trained on the Box-Cox transformed data in next day river flow prediction compared with that of the Linear Regression model. De-seasonalization was least sensitive to short term forecasts but Neural Networks were ranking second and could possibly be best tuned by adjusting the parameters. combined techniques that affected model performance were the forward and backward selection both concerning input variables. Further research should highlight further improvements of Neural Networks approximation, effects of the ensemble method, and selection of input variables; However, the long-term forecast and impact of environmental changes are promising objectives in future studies.

6. References

Akbarian, M., Saghafian, B. and Golian, S. (2023) 'Monthly streamflow forecasting by machine learning methods using dynamic weather prediction model outputs over Iran,' *Journal of Hydrology*, 620, p. 129480. https://doi.org/10.1016/j.jhydrol.2023.129480.

Chieu, T.Q. *et al.* (2024) 'Prediction of the water level at the Kien Giang River based on regression techniques,' *River*, 3(1), pp. 59–68. https://doi.org/10.1002/rvr2.71.

Comito, C. and Pizzuti, C. (2022) 'Artificial intelligence for forecasting and diagnosing COVID-19 pandemic: A focused review,' *Artificial Intelligence in Medicine*, 128, p. 102286. https://doi.org/10.1016/j.artmed.2022.102286.

Cui, F. *et al.* (2020) 'Newly explored machine learning model for river flow time series forecasting at Mary River, Australia,' *Environmental Monitoring and Assessment*, 192(12). https://doi.org/10.1007/s10661-020-08724-1.

Dubos, V. *et al.* (2022) 'Short-term forecasting of spring freshet peak flow with the Generalized Additive model,' *Journal of Hydrology*, 612, p. 128089. https://doi.org/10.1016/j.jhydrol.2022.128089.

Fernandes, A.C.P. *et al.* (2023) 'Water quality predictions through linear regression - A brute force algorithm approach,' *MethodsX*, 10, p. 102153. https://doi.org/10.1016/j.mex.2023.102153.

Gambella, C., Ghaddar, B. and Naoum-Sawaya, J. (2021) 'Optimization problems for machine learning: A survey,' *European Journal of Operational Research*, 290(3), pp. 807–828. https://doi.org/10.1016/j.ejor.2020.08.045.

Granata, F. and Di Nunno, F. (2023) 'Neuroforecasting of daily streamflows in the UK for short- and medium-term horizons: A novel insight,' *Journal of Hydrology*, 624, p. 129888. https://doi.org/10.1016/j.jhydrol.2023.129888.

Hussain, D. and Khan, A.A. (2020) 'Machine learning techniques for monthly river flow forecasting of Hunza River, Pakistan,' *Earth Science Informatics*, 13(3), pp. 939–949. https://doi.org/10.1007/s12145-020-00450-z.

Jamei, M. *et al.* (2022) 'Forecasting daily flood water level using hybrid advanced Machine Learning based Time-Varying Filtered Empirical Mode Decomposition approach,' *Water Resources Management*, 36(12), pp. 4637–4676. https://doi.org/10.1007/s11269-022-03270-6.

James, G. *et al.* (2021) 'Linear model selection and regularization,' in *Springer texts in statistics*, pp. 225–288. https://doi.org/10.1007/978-1-0716-1418-1_6.

Ji, C. *et al.* (2022) 'Early identification of abnormal deviations in nonstationary processes by removing non- stationarity,' in *Computer-aided chemical engineering/Computer aided chemical engineering*, pp. 1393–1398. https://doi.org/10.1016/b978-0-323-85159-6.50232-3.

Kedam, N. *et al.* (2024) 'River Stream Flow Prediction through Advanced Machine Learning Models for Enhanced Accuracy,' *Results in Engineering*, p. 102215. https://doi.org/10.1016/j.rineng.2024.102215.

Kilinc, H.C. *et al.* (2023) 'Daily Scale River Flow Forecasting Using Hybrid Gradient Boosting Model with Genetic Algorithm Optimization,' *Water Resources Management*, 37(9), pp. 3699–3714. https://doi.org/10.1007/s11269-023-03522-z.

Latt, Z.Z. and Wittenberg, H. (2014) 'Improving flood forecasting in a developing Country: A comparative study of Stepwise multiple linear regression and artificial neural network,' *Water Resources Management*, 28(8), pp. 2109–2128. https://doi.org/10.1007/s11269-014-0600-8.

Lian, Y. *et al.* (2021) 'Climate-driven model based on Long Short-Term memory and Bayesian optimization for multi-day-ahead daily streamflow forecasting,' *Water Resources Management*, 36(1), pp. 21–37. https://doi.org/10.1007/s11269-021-03002-2.

Malik, A. *et al.* (2020) 'Support vector regression optimized by meta-heuristic algorithms for daily streamflow prediction,' *Stochastic Environmental Research and Risk Assessment*, 34(11), pp. 1755–1773. https://doi.org/10.1007/s00477-020-01874-1.

National River Flow Archive (no date) *Search data*. https://nrfa.ceh.ac.uk/data/search (Accessed: September 8, 2024).

Nguyen, T.-T., Huu, Q.N. and Li, M.J. (2015) *Forecasting Time Series Water Levels on Mekong River Using Machine Learning Models*, *2015 Seventh International Conference on Knowledge and Systems Engineering (KSE)*. Ho Chi Minh City, Vietnam: IEEE. https://doi.org/10.1109/kse.2015.53.

Serrano-López, F. *et al.* (2024) 'Modeling river flow for flood forecasting: A case study on the TeR River,' *Applied Computing and Geosciences*, 23, p. 100181. https://doi.org/10.1016/j.acags.2024.100181.

Smith, G. (2018) 'Step away from stepwise,' *Journal of Big Data*, 5(1). https://doi.org/10.1186/s40537-018-0143-6.

Svozil, D., Kvasnicka, V. and Pospichal, J. (1997) 'Introduction to multi-layer feed-forward neural networks,' *Chemometrics and Intelligent Laboratory Systems*, 39(1), pp. 43–62. https://doi.org/10.1016/s0169-7439(97)00061-0.

Tawakuli, A. *et al.* (2024) 'Survey:Time-Series Data Preprocessing: A survey and an Empirical analysis,' *Journal of Engineering Research* [Preprint]. https://doi.org/10.1016/j.jer.2024.02.018.

Tsakiri, K., Marsellos, A. and Kapetanakis, S. (2018) 'Artificial neural network and multiple linear regression for flood prediction in Mohawk River, New York,' *Water*, 10(9), p. 1158. https://doi.org/10.3390/w10091158.

Zakaria, M.N.A. *et al.* (2023) 'Exploring machine learning algorithms for accurate water level forecasting in Muda River, Malaysia,' *Heliyon*, 9(7), p. e17689. https://doi.org/10.1016/j.heliyon.2023.e17689.

7. Appendix

The presented code, sequence of actions involved in data cleaning, exploratory data analysis, and stepwise selection. Please take note informed that the provided code snippet is partial. The complete code, including the building process of the model and the optimisation processes, has been submitted separately to the university supervisor .

1. Import all used libraries

```
!pip install tensorflow
!pip install pygam
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from itertools import combinations
from scipy.stats import boxcox
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import STL
import seaborn as sns
import random
import torch
from statsmodels.tools.eval_measures import aic
import tensorflow as tf
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error, make_scorer
from sklearn.model_selection import train_test_split, cross_val_score, learning_curve, KFold, RandomizedSearchCV
import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson
from sklearn.svm import SVR
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from tensorflow.keras.callbacks import EarlyStopping
from scipy.stats import uniform
from pygam import LinearGAM, s

'''imported marked to add notes in bold, Italic...'''
from IPython.display import Markdown
def bold(string):
    display(Markdown(string))
```

2. Load Data

```
# add file path
dir_57001_cdr = pd.read_csv('57001_cdr.csv')
dir_57001_gdf = pd.read_csv('57001_gdf.csv')
dir_57002_cdr = pd.read_csv('57002_cdr.csv')
dir_57002_gdf = pd.read_csv('57002_gdf.csv')
dir_57004_cdr = pd.read_csv('57004_cdr.csv')
dir_57004_gdf = pd.read_csv('57004_gdf.csv')
dir_57005_cdr = pd.read_csv('57005_cdr.csv')
dir_57005_gdf = pd.read_csv('57005_gdf.csv')
dir_57006_cdr = pd.read_csv('57006_cdr.csv')
dir_57006_gdf = pd.read_csv('57006_gdf.csv')
dir_57007_cdr = pd.read_csv('57007_cdr.csv')
dir_57007_gdf = pd.read_csv('57007_gdf.csv')
dir_57015_cdr = pd.read_csv('57015_cdr.csv')
dir_57015_gdf = pd.read_csv('57015_gdf.csv')
dir_57017_cdr = pd.read_csv('57017_cdr.csv')
dir_57017_gdf = pd.read_csv('57017_gdf.csv')
```

3. Data Inspection and Data Cleaning

3.1 Data inspected of all gauges

3.1.1 Gauge station (57005 - Taff at Pontypridd)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57005 (which is main river)**')
dir_57005_cdr.head(30)
```

```
bold('**First 30 instance of daily flow\'s raw data of gauge 57005 (which is main river)**')
dir_57005_gdf.head(30)
```

- raw data file contain metadata with actual dataset which need to separte from metadata (57005, Taff at Pontypridd)

```
daily_rainfall_catchement_57005 = dir_57005_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57005 = pd.DataFrame({
    'date': daily_rainfall_catchement_57005['file'],
    'cdr': daily_rainfall_catchement_57005['timestamp']
})
daily_gauge_catchement_57005 = dir_57005_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57005 = pd.DataFrame({
    'date': daily_gauge_catchement_57005['file'],
    'gdf': daily_gauge_catchement_57005['timestamp']
})
print('-----------------------------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57005 (Taff at Pontipridd-Parent of 57004, 57006, 57007)**')
print('-----------------------------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57005)
print('-----------------------------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57005 (Taff at Pontipridd-Parent of 57004, 57006, 57007)**')
print('-----------------------------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57005)
```

```
bold('**Structure summary of dataframe 57005**')
print('-----------------')
print('Rainfall at 57005')
print('-----------------')
print(df_daily_rainfall_catchement_57005.info())
print('-----------------')
print('|| Daily flow at 57005 ||')
print('-----------------')
print(df_daily_gauge_catchement_57005.info())
```

```
bold('**Description summary of dataframe 57005**')
print('-----------------------')
print('|| Rainfall at 57005 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57005.describe())
print('-----------------------')
print('|| Daily flow at 57005 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57005.describe())
```

```
bold('**Merge DataFrames 57005 on \'time\' column using inner join**')
merged_df_57005 = pd.merge(df_daily_rainfall_catchement_57005, df_daily_gauge_catchement_57005, on='date', how='inner')
print("Inner Join Result:")
print(merged_df_57005)
```

```
merged_df_57005['gauge'] = 57005
merged_df_57005
```

```
# converting type of attritute
merged_df_57005['date'] = pd.to_datetime(merged_df_57005['date'])
merged_df_57005['cdr'] = pd.to_numeric(merged_df_57005['cdr'])
merged_df_57005['gdf'] = pd.to_numeric(merged_df_57005['gdf'])
merged_df_57005.info()
```

```
print('--------------------------------')
bold('**|| Numerical Summary of 57005 ||**')
print('--------------------------------')
merged_df_57005.iloc[0:,1:3].describe()
```

```
print('**********************************')
bold('**|| Checking missing value in 57005 ||**')
print('**********************************')
missing_values_57005 = merged_df_57005.isna().sum()
missing_values_57005
```

```
print('**********************************')
bold('**|| Checking duplicacy in data 57005 ||**')
print('**********************************')
duplicate_values_count_57005 = merged_df_57005.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57005}')
```

3.1.2 Gauge station (57006 - Rhondda at Trehafod)

```
bold('**First 30 instance of water rainfall\'s raw data of gauge 57006 (which is tributaries river)**')
dir_57006_cdr.head(30)
```

```
bold('**First 30 instance of water daily flow\'s raw data of gauge 57006 (which is tributaries river)**')
dir_57006_gdf.head(30)
```

```
daily_rainfall_catchement_57006 = dir_57006_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57006 = pd.DataFrame({
    'date': daily_rainfall_catchement_57006['file'],
    'cdr': daily_rainfall_catchement_57006['timestamp']
})
daily_gauge_catchement_57006 = dir_57006_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57006 = pd.DataFrame({
    'date': daily_gauge_catchement_57006['file'],
    'gdf': daily_gauge_catchement_57006['timestamp']
})
print('-------------------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57006 (Rhondda at Trehafod- parent of 57017)**')
print('-------------------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57006)
print('-------------------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57006 (Rhondda at Trehafod - parent of 57017)**')
print('-------------------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57006)
```

```
bold('**structure summary of dataframe 57006**')
print('-----------------------')
print('|| Rainfall at 57006 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57006.info())
print('-----------------------')
print('|| Daily flow at 57006 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57006.info())
```

```
bold('**Description summary of dataframe 57006**')
print('-----------------------')
print('|| Rainfall at 57006 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57006.describe())
print('-----------------------')
print('|| Daily flow at 57006 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57006.describe())
```

```
bold('**Merge DataFrames 57006 on \'time\'column using inner join**')
merged_df_57006 = pd.merge(df_daily_rainfall_catchement_57006, df_daily_gauge_catchement_57006, on='date', how='inner')
print("Inner Join Result:")
merged_df_57006['gauge'] = 57006
print(merged_df_57006)
```

```
# converting type of attritute
merged_df_57006['date'] = pd.to_datetime(merged_df_57006['date'])
merged_df_57006['cdr'] = pd.to_numeric(merged_df_57006['cdr'])
merged_df_57006['gdf'] = pd.to_numeric(merged_df_57006['gdf'])
merged_df_57006.info()
```

```
print('--------------------------------')
bold('**|| Numerical Summary of 57006 ||**')
print('--------------------------------')
merged_df_57006.iloc[0:,1:3].describe()
```

```
print('**********************************')
bold('**|| Checking missing value in 57006 ||**')
print('**********************************')
```

```
missing_values_57006 = merged_df_57006.isna().sum()
missing_values_57006


print('**********************************')
bold('**|| Checking duplicacy in data 57006 ||**')
print('**********************************')
duplicate_values_count_57006 = merged_df_57006.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57006}')
```

3.1.3 Gauge station (57017 - Rhondda Fawr at Tynewydd)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57017 (which is tributaries river)**')
dir_57017_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57017 (which is tributaries river)**')
dir_57017_gdf.head(30)


daily_rainfall_catchement_57017 = dir_57017_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57017 = pd.DataFrame({
    'date': daily_rainfall_catchement_57017['file'],
    'cdr': daily_rainfall_catchement_57017['timestamp']
})
daily_gauge_catchement_57017 = dir_57017_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57017 = pd.DataFrame({
    'date': daily_gauge_catchement_57017['file'],
    'gdf': daily_gauge_catchement_57017['timestamp']
})
print('--------------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57017  (Rhondda Fawr at Tynewydd-child)**')
print('--------------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57017)
print('--------------------------------------------------------------------------------')
bold('**mMasurement of daily flow (m3/s) for station 57017(Rhondda Fawr at Tynewydd-child)**')
print('--------------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57017)


bold('**structure summary of dataframe 57017**')
print('-----------------------')
print('|| Rainfall at 57017 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57017.info())
print('-----------------------')
print('|| Daily flow at 57017 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57017.info())


bold('**Description summary of dataframe 57017**')
print('-----------------------')
print('|| Rainfall at 57017 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57017.describe())
print('-----------------------')
print('|| Daily flow at 57017 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57017.describe())


bold('**Merge DataFrames 57017 on \'time\'column using inner join**')
merged_df_57017 = pd.merge(df_daily_rainfall_catchement_57017, df_daily_gauge_catchement_57017, on='date', how='inner')
print("Inner Join Result:")
merged_df_57017['gauge'] = 57017
print(merged_df_57017)


# converting type of attritute
merged_df_57017['date'] = pd.to_datetime(merged_df_57017['date'])
merged_df_57017['cdr'] = pd.to_numeric(merged_df_57017['cdr'])
merged_df_57017['gdf'] = pd.to_numeric(merged_df_57017['gdf'])
merged_df_57017.info()


print('--------------------------------')
bold('**|| Numerical Summary of 57017 ||**')
print('--------------------------------')
merged_df_57017.iloc[0:,1:3].describe()
```

```
print('**********************************')
bold('**|| Checking missing value in 57017 ||**')
print('**********************************')
missing_values_57017 = merged_df_57017.isna().sum()
missing_values_57017


print('**********************************')
bold('**|| Checking duplicacy in data 57017 ||**')
print('**********************************')
duplicate_values_count_57017 = merged_df_57017.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57017}')
```

note: need to drop station 57017 because there is very less data point for modelling but still need look forward at this point not very clear

3.1.4 Gauge station (57004 - Cynon at Abercynon)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57004 (which is tributaries river)**')
dir_57004_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57004 (which is tributaries river)**')
dir_57004_gdf.head(30)


daily_rainfall_catchement_57004 = dir_57004_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57004 = pd.DataFrame({
    'date': daily_rainfall_catchement_57004['file'],
    'cdr': daily_rainfall_catchement_57004['timestamp']
})
daily_gauge_catchement_57004 = dir_57004_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57004 = pd.DataFrame({
    'date': daily_gauge_catchement_57004['file'],
    'gdf': daily_gauge_catchement_57004['timestamp']
})
print('-------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57004 (Cynon at Abercynon-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57004)
print('-------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57004 (Cynon at Abercynon-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57004)


bold('**structure summary of dataframe 57004**')
print('-----------------------')
print('|| Rainfall at 57004 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57004.info())
print('-----------------------')
print('|| Daily flow at 57004 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57004.info())


bold('**Description summary of dataframe 57004**')
print('-----------------------')
print('|| Rainfall at 57004 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57004.describe())
print('-----------------------')
print('|| Daily flow at 57004 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57004.describe())


bold('**Merge DataFrames 57004 on \'time\'column using inner join**')
merged_df_57004 = pd.merge(df_daily_rainfall_catchement_57004, df_daily_gauge_catchement_57004, on='date', how='inner')
print("Inner Join Result:")
merged_df_57004['gauge'] = 57004
print(merged_df_57004)


# converting type of attritute
merged_df_57004['date'] = pd.to_datetime(merged_df_57004['date'])
merged_df_57004['cdr'] = pd.to_numeric(merged_df_57004['cdr'])
merged_df_57004['gdf'] = pd.to_numeric(merged_df_57004['gdf'])
merged_df_57004.info()
```

```
print('--------------------------------')
bold('**|| Numerical Summary of 57004 ||**')
print('--------------------------------')
merged_df_57004.iloc[0:,1:3].describe()


print('**********************************')
bold('**|| Checking missing value in 57004 ||**')
print('**********************************')
missing_values_57004 = merged_df_57004.isna().sum()
missing_values_57004


print('**********************************')
bold('**|| Checking duplicacy in data 57004 ||**')
print('**********************************')
duplicate_values_count_57004 = merged_df_57004.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57004}')
```

3.1.5 Gauge station (57007 - Taff at Fiddlers Elbow)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57007 (which is tributaries river)**')
dir_57007_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57007 (which is tributaries river)**')
dir_57007_gdf.head(30)


daily_rainfall_catchement_57007 = dir_57007_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57007 = pd.DataFrame({
    'date': daily_rainfall_catchement_57007['file'],
    'cdr': daily_rainfall_catchement_57007['timestamp']
})
daily_gauge_catchement_57007 = dir_57007_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57007 = pd.DataFrame({
    'date': daily_gauge_catchement_57007['file'],
    'gdf': daily_gauge_catchement_57007['timestamp']
})
print('-----------------------------------------------------------------------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57007 (Taff at Fiddlers Elbow- parent of 57015 and other tributeries and child 57005)**')
print('-----------------------------------------------------------------------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57007)
print('-----------------------------------------------------------------------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57007 (Taff at Fiddlers Elbow- parent of 57015 and other tributeries and child of 5
print('-----------------------------------------------------------------------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57007)


bold('**structure summary of dataframe 57007**')
print('------------------------')
print('|| Rainfall at 57007 ||')
print('------------------------')
print(df_daily_rainfall_catchement_57007.info())
print('------------------------')
print('|| Daily flow at 57007 ||')
print('------------------------')
print(df_daily_gauge_catchement_57007.info())


bold('**Description summary of dataframe 57007**')
print('------------------------')
print('|| Rainfall at 57007 ||')
print('------------------------')
print(df_daily_rainfall_catchement_57007.describe())
print('------------------------')
print('|| Rainfall at 57007 ||')
print('------------------------')
print(df_daily_gauge_catchement_57007.describe())


bold('**Merge DataFrames 57007 on \'time\'column using inner join**')
merged_df_57007 = pd.merge(df_daily_rainfall_catchement_57007, df_daily_gauge_catchement_57007, on='date', how='inner')
print("Inner Join Result:")
merged_df_57007['gauge'] = 57007
print(merged_df_57007)


# converting type of attritute
merged_df_57007['date'] = pd.to_datetime(merged_df_57007['date'])
merged_df_57007['cdr'] = pd.to_numeric(merged_df_57007['cdr'])
```

```
merged_df_57007['gdf'] = pd.to_numeric(merged_df_57007['gdf'])
merged_df_57007.info()


print('--------------------------------')
bold('**|| Numerical Summary of 57007 ||**')
print('--------------------------------')
merged_df_57007.iloc[0:,1:3].describe()


print('--------------------------------')
bold('**|| Numerical Summary of 57007 ||**')
print('--------------------------------')
missing_values_57007 = merged_df_57007.isna().sum()
missing_values_57007


print('**********************************')
bold('**|| Checking duplicacy in data 57007 ||**')
print('**********************************')
duplicate_values_count_57007 = merged_df_57007.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57007}')
```

3.1.6 Gauge station (57015 - Taff at Merthyr Tydfil)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57015 (which is tributaries river)**')
dir_57015_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57015 (which is tributaries river)**')
dir_57015_gdf.head(30)


daily_rainfall_catchement_57015 = dir_57015_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57015 = pd.DataFrame({
    'date': daily_rainfall_catchement_57015['file'],
    'cdr': daily_rainfall_catchement_57015['timestamp']
})
daily_gauge_catchement_57015 = dir_57015_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57015 = pd.DataFrame({
    'date': daily_gauge_catchement_57015['file'],
    'gdf': daily_gauge_catchement_57015['timestamp']
})
print('-------------------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57015 (Taff at Merthyr Tydfil-parent of 57001,57002)**')
print('-------------------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57015)
print('-------------------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57015 (Taff at Merthyr Tydfil- parent of 57001, 57002**')
print('-------------------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57015)


bold('**structure summary of dataframe 57015**')
print('------------------------')
print('|| Rainfall at 57015 ||')
print('------------------------')

print(df_daily_rainfall_catchement_57015.info())
print('------------------------')
print('|| Daily flow at 57015 ||')
print('------------------------')
print(df_daily_gauge_catchement_57015.info())


bold('**Description summary of dataframe 57015**')
print('------------------------')
print('|| Rainfall at 57015 ||')
print('------------------------')
print(df_daily_rainfall_catchement_57015.describe())
print('------------------------')
print('|| Daily flow at 57015 ||')
print('------------------------')
print(df_daily_gauge_catchement_57015.describe())


bold('**Merge DataFrames 57015 on \'time\'column using inner join**')
merged_df_57015 = pd.merge(df_daily_rainfall_catchement_57015, df_daily_gauge_catchement_57015, on='date', how='inner')
print("Inner Join Result:")
merged_df_57015['gauge'] = 57015
print(merged_df_57015)
```

```
# converting type of attritute
merged_df_57015['date'] = pd.to_datetime(merged_df_57015['date'])
merged_df_57015['cdr'] = pd.to_numeric(merged_df_57015['cdr'])
merged_df_57015['gdf'] = pd.to_numeric(merged_df_57015['gdf'])
merged_df_57015.info()


print('--------------------------------')
bold('**|| Numerical Summary of 570015||**')
print('--------------------------------')
merged_df_57015.iloc[0:,1:3].describe()


print('**********************************')
bold('**|| Checking missing value in 57015 ||**')
print('**********************************')
missing_values_57015 = merged_df_57015.isna().sum()
missing_values_57015


print('**********************************')
bold('**|| Checking duplicacy in data 57015 ||**')
print('**********************************')
duplicate_values_count_57015 = merged_df_57015.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57015}')
```

3.1.7 Gauge station (57002 - Taf Fawr at Llwynon Reservoir)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57002 (which is tributaries river)**')
dir_57002_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57002 (which is tributaries river)**')
dir_57002_gdf.head(30)


daily_rainfall_catchement_57002 = dir_57002_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57002 = pd.DataFrame({
    'date': daily_rainfall_catchement_57002['file'],
    'cdr': daily_rainfall_catchement_57002['timestamp']
})
daily_gauge_catchement_57002 = dir_57002_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57002 = pd.DataFrame({
    'date': daily_gauge_catchement_57002['file'],
    'gdf': daily_gauge_catchement_57002['timestamp']
})
print('-------------------------------------------------------------------------')
bold('**measurement of rainfall (mm) for station 57002 (Taf Fawr at Llwynon Reservoir-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57002)
print('-------------------------------------------------------------------------')
bold('**measurement of daily flow (m3/s) for station 57002 (Taf Fawr at Llwynon Reservoir-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57002)


bold('**structure summary of dataframe 57002**')
print('-----------------------')
print('|| Rainfall at 57002 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57002.info())
print('-----------------------')
print('|| Daily flow at 57002 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57002.info())


bold('**Description summary of dataframe 57002**')
print('-----------------------')
print('|| Rainfall at 57002 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57002.describe())
print('-----------------------')
print('|| Daily flow at 57002 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57002.describe())


bold('**Merge DataFrames 57002 on \'time\'column using inner join**')
merged_df_57002 = pd.merge(df_daily_rainfall_catchement_57002, df_daily_gauge_catchement_57002, on='date', how='inner')
print("Inner Join Result:")
```

```
merged_df_57002['gauge'] = 57002
print(merged_df_57002)


# converting type of attritute
merged_df_57002['date'] = pd.to_datetime(merged_df_57002['date'])
merged_df_57002['cdr'] = pd.to_numeric(merged_df_57002['cdr'])
merged_df_57002['gdf'] = pd.to_numeric(merged_df_57002['gdf'])
merged_df_57002.info()


print('--------------------------------')
bold('**|| Numerical Summary of 57002 ||**')
print('--------------------------------')
merged_df_57002.iloc[0:,1:3].describe()


print('**********************************')
bold('**|| Checking missing value in 57002 ||**')
print('**********************************')
missing_values_57002 = merged_df_57002.isna().sum()
missing_values_57002


print('**********************************')
bold('**|| Checking duplicacy in data 57002 ||**')
print('**********************************')
duplicate_values_count_57002 = merged_df_57002.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57002}')
```

3.1.8 Gauge station (57001 - Taf Fechan at Taf Fechan Reservoir)

```
bold('**First 30 instance of rainfall\'s raw data of gauge 57001 (which is tributaries river)**')
dir_57001_cdr.head(30)


bold('**First 30 instance of daily flow\'s raw data of gauge 57001 (which is tributaries river)**')
dir_57001_gdf.head(30)


daily_rainfall_catchement_57001 = dir_57001_cdr.iloc[19:,0:2]
df_daily_rainfall_catchement_57001 = pd.DataFrame({
    'date': daily_rainfall_catchement_57001['file'],
    'cdr': daily_rainfall_catchement_57001['timestamp']
})
daily_gauge_catchement_57001 = dir_57001_gdf.iloc[19:,0:2]
df_daily_gauge_catchement_57001 = pd.DataFrame({
    'date': daily_gauge_catchement_57001['file'],
    'gdf': daily_gauge_catchement_57001['timestamp']
})
print('-------------------------------------------------------------------------')
bold('**Measurement of rainfall (mm) for station 57001  (Taf Fechan at Taf Fechan Reservoir-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_rainfall_catchement_57001)
print('-------------------------------------------------------------------------')
bold('**Measurement of daily flow (m3/s) for station 57001(Taf Fechan at Taf Fechan Reservoir-child)**')
print('-------------------------------------------------------------------------')
print(df_daily_gauge_catchement_57001)


bold('**structure summary of dataframe 57001**')
print('-----------------------')
print('|| Rainfall at 57001 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57001.info())
print('-----------------------')
print('|| Daily flow at 57001 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57001.info())


bold('**Description summary of dataframe 57001**')
print('-----------------------')
print('|| Rainfall at 57001 ||')
print('-----------------------')
print(df_daily_rainfall_catchement_57001.describe())
print('-----------------------')
print('|| Daily flow at 57001 ||')
print('-----------------------')
print(df_daily_gauge_catchement_57001.describe())
```

```
bold('**Merge DataFrames 57001 on \'time\'column using inner join**')
merged_df_57001 = pd.merge(df_daily_rainfall_catchement_57001, df_daily_gauge_catchement_57001, on='date', how='inner')
print("Inner Join Result:")
merged_df_57001['gauge'] = 57001
print(merged_df_57001)


# converting type of attritute
merged_df_57001['date'] = pd.to_datetime(merged_df_57001['date'])
merged_df_57001['cdr'] = pd.to_numeric(merged_df_57001['cdr'])
merged_df_57001['gdf'] = pd.to_numeric(merged_df_57001['gdf'])
merged_df_57001.info()


print('--------------------------------')
bold('**|| Numerical Summary of 57001 ||**')
print('--------------------------------')
merged_df_57001.iloc[0:,1:3].describe()


print('**********************************')
bold('**|| Checking missing value in 57001 ||**')
print('**********************************')
missing_values_57001 = merged_df_57001.isna().sum()
missing_values_57001


print('**********************************')
bold('**|| Checking duplicacy in data 57001 ||**')
print('**********************************')
duplicate_values_count_57001 = merged_df_57001.duplicated().sum()
print(f'duplicate values : {duplicate_values_count_57001}')


merged_df_57001
```

3.2 Graphical Representation of data

3.2.1 Graphical representation to understand recording data gap at gauages

```
all_data_frame = [
    merged_df_57005,
    merged_df_57006,
    merged_df_57017,
    merged_df_57004,
    merged_df_57007,
    merged_df_57015,
    merged_df_57002,
    merged_df_57001
]

def graphical_inspection_missing_values(all_data_frame):
    for gauge in all_data_frame:
        plt.figure(figsize=(15, 5))
        plt.plot(
            gauge['date'],
            gauge['gdf'],
            label='Daily Flow Measurements',
            color='red')
        plt.title(
            f'Representation of missing values at daily flow data (gdf: {gauge["gauge"][0]})',
            fontsize = 15,
            fontweight='bold'
        )
        plt.xticks(
            fontsize=15,
            fontfamily='serif',
            fontweight='bold'
        )
        plt.yticks(
            fontsize=15,
            fontfamily='serif',
            fontweight='bold'
        )
        plt.gca().set_facecolor('white')
        plt.legend(fontsize=15)
        plt.xlabel('years', fontsize = 15, fontweight='bold')
        plt.ylabel('Gauge Reading (m³/s)', fontsize = 15, fontweight='bold')
        plt.legend(prop={'size': 15, 'weight': 'bold'})
        plt.show()
```

```
graphical_inspection_missing_values(all_data_frame)
```

3.2.2 Comparison of combined representation rainfall and daily flow to Understand recording gap of data for all gauges

```
# Combine all data frames into one
combined_gauge_data = pd.concat(all_data_frame)

def plot_gdf_and_cdr_data_gaps(
    combined_gauge_df,
    date_column='date',
    value_column='gdf',
    rainfall_column='cdr',
    gauge_column='gauge'
):
    plt.figure(figsize=(15, 7))
    #date column is datetime
    combined_gauge_df[date_column] = pd.to_datetime(combined_gauge_df[date_column])

    # Identify unique gauges
    gauges = combined_gauge_df[gauge_column].unique()

    # Define a color map
    cmap = plt.get_cmap('tab10')
    colors = cmap(
        np.linspace(0, 1, len(gauges))
    )
    light_cmap = plt.get_cmap('tab20c')
    light_colors = light_cmap(
        np.linspace(0, 1, len(gauges))
    )

    # Create a plot for each gauge
    for i, gauge in enumerate(gauges):
        gauge_data = combined_gauge_df[combined_gauge_df[gauge_column] == gauge].copy()

        # Sort data by date
        gauge_data = gauge_data.sort_values(by=date_column)

        # Create segments where there are no missing values for gdf
        gdf_segments = []
        current_segment = []
        for j in range(len(gauge_data)):
            if not np.isnan(gauge_data.iloc[j][value_column]):
                current_segment.append((gauge_data.iloc[j][date_column], 2 * i + 0.2))  # Offset gdf line
            else:
                if current_segment:
                    gdf_segments.append(current_segment)
                    current_segment = []
        if current_segment:
            gdf_segments.append(current_segment)

        # Plot the segments for gdf with a unique color for each gauge
        for segment in gdf_segments:
            dates, y = zip(*segment)
            plt.plot(dates, y, 'o-', label=f'Gauge {gauge} (gdf)' if segment == gdf_segments[0] else "", color=colors[i], alpha=0.7)

        # Create segments where there are no missing values for cdr
        cdr_segments = []
        current_segment = []
        for j in range(len(gauge_data)):
            if not np.isnan(gauge_data.iloc[j][rainfall_column]):
                current_segment.append((gauge_data.iloc[j][date_column], 2 * i - 0.2))  # Offset cdr line
            else:
                if current_segment:
                    cdr_segments.append(current_segment)
                    current_segment = []
        if current_segment:
            cdr_segments.append(current_segment)

        # Plot the segments for cdr with a dashed line and the same color for each gauge
        for segment in cdr_segments:
            dates, y = zip(*segment)
            plt.plot(dates, y, 'o--', label=f'Gauge {gauge} (cdr)' if segment == cdr_segments[0] else "", color=light_colors[i], alpha=(

        # Label long gaps in gdf data with start and end dates
        for j in range(1, len(gauge_data)):
            if np.isnan(gauge_data.iloc[j][value_column]) and not np.isnan(gauge_data.iloc[j-1][value_column]):
```

```
                start_date = gauge_data.iloc[j-1][date_column]
            if not np.isnan(gauge_data.iloc[j][value_column]) and np.isnan(gauge_data.iloc[j-1][value_column]):
                end_date = gauge_data.iloc[j][date_column]
                if (end_date - start_date).days > 365:  #threshold for 365
                    mid_date = start_date + (end_date - start_date) / 2
                    plt.text(
                        mid_date,
                        2 * i + 0.2,
                        f'{start_date.strftime("%Y-%m-%d")} to {end_date.strftime("%Y-%m-%d")}',
                        color='black',
                        ha='center',
                        va='center'
                    )

    plt.title(
        'Comparison of Gauge and Rainfall Recording data Gaps',
        fontsize = 15,
        fontweight='bold'
    )
    plt.xlabel(
        'years',
        fontsize = 15,
        fontweight='bold'
    )
    plt.ylabel(
        'Gauge station',
        fontsize = 15,
        fontweight='bold'
    )
    ytick_labels = []
    for gauge in gauges:
        ytick_labels.append(
            f'Gauge {gauge} (gdf)'
        )
        ytick_labels.append(
            f'Gauge {gauge} (cdr)'
        )
    plt.yticks(
        range(0, 2 * len(gauges), 2),
        [f'Gauge {g}' for g in gauges],
        fontsize = 15,
        fontweight='bold'
    )
    plt.xticks(
        fontsize = 15,
        fontweight='bold'
    )
    plt.legend(
        prop={'size': 10, 'weight': 'bold'}
    )
    plt.grid(True)
    plt.show()

plot_gdf_and_cdr_data_gaps(combined_gauge_data)
```

Note: With the observation of above graph comparsion between all gauges of daily flow and rainfall, we will used data from 26-05-2006 because there is long period missing data approx 1.5 year, So we will taken base of gauge : 57002 to arranged data of all gauge for further analysis

3.3 Arranging data from 2006-05-26 for all gauge by taken base of gdf 57002

```
df = pd.DataFrame()
start_date = '2006-05-26'
arranged_df_57005 = pd.DataFrame(merged_df_57005[merged_df_57005['date']>= start_date]).reset_index(drop=True)
arranged_df_57006 = pd.DataFrame(merged_df_57006[merged_df_57006['date']>= start_date]).reset_index(drop=True)
arranged_df_57004 = pd.DataFrame(merged_df_57004[merged_df_57004['date']>= start_date]).reset_index(drop=True)
arranged_df_57007 = pd.DataFrame(merged_df_57007[merged_df_57007['date']>= start_date]).reset_index(drop=True)
arranged_df_57015 = pd.DataFrame(merged_df_57015[merged_df_57015['date']>= start_date]).reset_index(drop=True)
arranged_df_57002 = pd.DataFrame(merged_df_57002[merged_df_57002['date']>= start_date]).reset_index(drop=True)
arranged_df_57001 = pd.DataFrame(merged_df_57001[merged_df_57001['date']>= start_date]).reset_index(drop=True)

print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57005')
print('------------------------------------------------------------')
print(arranged_df_57005)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57006')
print('------------------------------------------------------------')
```

```
print(arranged_df_57006)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57004')
print('------------------------------------------------------------')
print(arranged_df_57004)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 570057')
print('------------------------------------------------------------')
print(arranged_df_57007)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57015')
print('------------------------------------------------------------')
print(arranged_df_57015)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57002')
print('------------------------------------------------------------')
print(arranged_df_57002)
print('------------------------------------------------------------')
print('Arranged dataframe of gauge 57001')
print('------------------------------------------------------------')
print(arranged_df_57001)


print('------------------------------------------------------------')
bold('**checking missing value after Arrangind data with same date**')
print('------------------------------------------------------------')
df_list = [arranged_df_57005, arranged_df_57006, arranged_df_57004, arranged_df_57007, arranged_df_57015, arranged_df_57002, arranged_df
for i in df_list:
    bold('**gauge ' +str(i['gauge'][0])+'**')
    print(i.isna().sum())
```

handling Missing value at gauge 57002 of gdf

```
#Rest of value is handling by backward and forward fill
arranged_df_57002 = arranged_df_57002.fillna(method='ffill')

# Perform backward fill
arranged_df_57002 = arranged_df_57002.fillna(method='bfill')
bold('**After handling missing value by backward and forward fill method**')
print(arranged_df_57002.isna().sum())


arranged_df_57002.reset_index(drop=True, inplace=True)
arranged_df_57002
```

3.4 Monthly Average of daily data

```
arranged_df_list =  [arranged_df_57005, arranged_df_57006, arranged_df_57004, arranged_df_57007, arranged_df_57015, arranged_df_57002, a

for arng_gauge in arranged_df_list:
    if (arng_gauge['cdr'] < 0).any().any() or (arng_gauge['gdf'] < 0).any().any():
        print(f"cdr or gdf contains negative values in gauge : {arng_gauge['gauge'][0]}.")
    else:
        print(f"cdr or gdf does not contain negative values in gauge : {arng_gauge['gauge'][0]}.")


arranged_df_57001['date'] = pd.to_datetime(arranged_df_57001['date'])
# Set 'date' as the index
arranged_df_57001.set_index('date', inplace=True)
# Resample to monthly frequency and calculate the mean
monthly_avg_57001 = arranged_df_57001.resample('M').mean().reset_index()
#rename the 'date' column to 'month' for clarity
monthly_avg_57001.rename(columns={'date': 'month'}, inplace=True)
# Display the first few rows of the resulting DataFrame
print(monthly_avg_57001)


monthly_avg_57001['gauge'] = monthly_avg_57001['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57001['month'] = monthly_avg_57001['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57001**')
print(monthly_avg_57001)


arranged_df_57002['date'] = pd.to_datetime(arranged_df_57002['date'])
```

```
# Set 'date' as the index
arranged_df_57002.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57002 = arranged_df_57002.resample('M').mean().reset_index()

#rename the 'date' column to 'month' for clarity
monthly_avg_57002.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57002)



monthly_avg_57002['gauge'] = monthly_avg_57002['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57002['month'] = monthly_avg_57002['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57002**')
print(monthly_avg_57002)


arranged_df_57015['date'] = pd.to_datetime(arranged_df_57015['date'])

# Set 'date' as the index
arranged_df_57015.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57015 = arranged_df_57015.resample('M').mean().reset_index()

# Rename the 'date' column to 'month' for clarity
monthly_avg_57015.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57015)


monthly_avg_57015['gauge'] = monthly_avg_57015['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57015['month'] = monthly_avg_57015['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57015**')
print(monthly_avg_57015)


arranged_df_57007['date'] = pd.to_datetime(arranged_df_57007['date'])

# Set 'date' as the index
arranged_df_57007.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57007 = arranged_df_57007.resample('M').mean().reset_index()

# Rename the 'date' column to 'month' for clarity
monthly_avg_57007.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57007)


monthly_avg_57007['gauge'] = monthly_avg_57007['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57007['month'] = monthly_avg_57007['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57007**')
print(monthly_avg_57007)


arranged_df_57004['date'] = pd.to_datetime(arranged_df_57004['date'])

# Set 'date' as the index
arranged_df_57004.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57004 = arranged_df_57004.resample('M').mean().reset_index()

# Rename the 'date' column to 'month' for clarity
monthly_avg_57004.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57004)


monthly_avg_57004['gauge'] = monthly_avg_57004['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57004['month'] = monthly_avg_57004['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57004**')
print(monthly_avg_57004)


arranged_df_57006['date'] = pd.to_datetime(arranged_df_57006['date'])
```

```
# Set 'date' as the index
arranged_df_57006.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57006 = arranged_df_57006.resample('M').mean().reset_index()

# Rename the 'date' column to 'month' for clarity
monthly_avg_57006.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57006)
```

```
monthly_avg_57006['gauge'] = monthly_avg_57006['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57006['month'] = monthly_avg_57006['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57006**')
print(monthly_avg_57006)
```

```
arranged_df_57005['date'] = pd.to_datetime(arranged_df_57005['date'])

# Set 'date' as the index
arranged_df_57005.set_index('date', inplace=True)

# Resample to monthly frequency and calculate the mean
monthly_avg_57005 = arranged_df_57005.resample('M').mean().reset_index()

# Rename the 'date' column to 'month' for clarity
monthly_avg_57005.rename(columns={'date': 'month'}, inplace=True)

# Display the first few rows of the resulting DataFrame
print(monthly_avg_57005)
```

```
monthly_avg_57005['gauge'] = monthly_avg_57005['gauge'].astype(int)  # Convert gauge to integer
monthly_avg_57005['month'] = monthly_avg_57005['month'].dt.to_period('M').astype(str) # Format the 'month' column to year and month
bold('**Monthly average of gauge 57005**')
print(monthly_avg_57005)
```

3.5 Outlier Checking

```
def box_plot_daily_data(arranged_df_list):
    for gauge in arranged_df_list:
        # Box plot for gdf
        plt.figure(figsize=(10, 2))
        plt.boxplot(
            gauge['gdf'].dropna(),
            vert=False,
            patch_artist=True,
            boxprops=dict(facecolor="lightblue")
        )
        plt.title(f"Box Plot for GDF: {gauge['gauge'][0]}")
        plt.xlabel('GDF', fontsize='10', fontweight = 'bold')
        plt.xticks(fontsize='10', fontweight = 'bold')
        plt.grid(True)
        plt.show()

        # Box plot for cdr
        plt.figure(figsize=(10, 2))
        plt.boxplot(
            gauge['cdr'].dropna(),
            vert=False,
            patch_artist=True,
            boxprops=dict(facecolor="lightgreen")
        )
        plt.title(f"Box Plot for CDR:  {gauge['gauge'][0]}")
        plt.xlabel('CDR', fontsize='10', fontweight = 'bold')
        plt.xticks(fontsize='10', fontweight = 'bold')
        plt.grid(True)
        plt.show()
box_plot_daily_data(arranged_df_list)
```

Note After investigating outlier by plotting box in above representation, it seem outlier exist, those points outside whiskers are considered outliers.

1. Extreme value of rainfall(cdr) can be natural phenomena.
2. Higher rainfall are common event may not be outlier. For rainfall event need to investigate to check at particular season or trend.
3. Due to Higher rainfall flow can be higher need to investigate further before handle all these point outside the whisker.

4. also box plot showing it right skewed

4. Data Exploration

```
from functools import reduce

# Define DataFrames
dfs = [
    arranged_df_57015,
    arranged_df_57002,
    arranged_df_57006,
    arranged_df_57007,
    arranged_df_57004,
    arranged_df_57005,
    arranged_df_57001
]

# Define the gauge numbers
gauges = ['57015', '57002', '57006', '57007', '57004', '57005', '57001']

# Process each DataFrame
for df, gauge in zip(dfs, gauges):
    df.rename(
        columns={'cdr': f'cdr_{gauge}','gdf': f'gdf_{gauge}'},
        inplace=True
    )
    df.drop(
        columns=['gauge'],
        inplace=True
    )

# Combine all data frames into one using reduce to merge them iteratively on 'date'
merge_allfilter_df = reduce(lambda left, right: pd.merge(left, right, on='date', how='inner'), dfs)



merge_allfilter_df


print('------------------------------------------------------------')
print('no missing value after merge all dataframe in single dataframe')
print('------------------------------------------------------------')
print(merge_allfilter_df.isna().sum())
```

4.1 Relation of daily flow and daily rainfall at gauges

```
def relation_cdr_gdf_daily_data(merge_allfilter_df):
    #merge_allfilter_df is already loaded
    flow_columns = [col for col in merge_allfilter_df.columns if col.startswith('gdf_')]
    rainfall_columns = [col for col in merge_allfilter_df.columns if col.startswith('cdr_')]

    # Loop over each gdf column (flow_columns) to create separate figures
    for flow_col in flow_columns:
        # Define the number of columns for the subplot grid
        ncols = 3  # Two plots per row
        nrows = int(len(rainfall_columns) / ncols) + (len(rainfall_columns) % ncols > 0)

        # Creating a new figure for each gdf column
        fig, axes = plt.subplots(
            nrows=nrows,
            ncols=ncols,
            figsize=(5 * ncols, 5 * nrows)
        )

        # Flatten the axes array for easy iteration
        axes = axes.flatten()

        # Plot each cdr column against the current gdf column
        for j, rain_col in enumerate(rainfall_columns):
            sns.scatterplot(x=merge_allfilter_df[rain_col], y=merge_allfilter_df[flow_col], ax=axes[j])
            axes[j].set_title(f'{flow_col} vs. {rain_col}', fontsize=12, fontweight='bold')
            axes[j].set_xlabel(flow_col, fontsize=10, fontweight='bold')
            axes[j].set_ylabel(rain_col, fontsize=10, fontweight='bold')
            axes[j].tick_params(axis='x')  # Rotate x-axis labels for better readability
            plt.setp(axes[j].get_xticklabels(), fontweight='bold')
            plt.setp(axes[j].get_yticklabels(), fontweight='bold')
```

```
        for k in range(len(rainfall_columns), len(axes)):
            fig.delaxes(axes[k])
        plt.suptitle(
            f"Relation Between {flow_col} with daily rainfall at all gauge ",
            fontsize=20, fontweight='bold'
        )
        plt.tight_layout()
        plt.show()
relation_cdr_gdf_daily_data(merge_allfilter_df)
```

4.2 Relation of daily flow at all gauges

```
def relationship_daily_data_flow_vs_flow(merge_allfilter_df):
    # Extract flow columns
    gdf_columns = [col for col in merge_allfilter_df.columns if col.startswith('gdf_')]
    flow_df = merge_allfilter_df[gdf_columns]

    # Create a pairplot
    g = sns.pairplot(flow_df)

    # Removed the upper triangle to avoid redundancy
    for i, j in zip(*np.triu_indices_from(g.axes, 1)):
        g.axes[i, j].set_visible(False)

    # Make x and y labels and tick labels bold
    for ax in g.axes.flatten():
        if ax is not None:
            ax.set_xlabel(ax.get_xlabel(), fontsize=12, fontweight='bold')
            ax.set_ylabel(ax.get_ylabel(), fontsize=12, fontweight='bold')
            ax.tick_params(axis='both', which='major', labelsize=10)
            ax.tick_params(axis='x')  # Rotate x-axis labels for better readability
            plt.setp(ax.get_xticklabels(), fontweight='bold')
            plt.setp(ax.get_yticklabels(), fontweight='bold')

    #main title
    plt.suptitle(
        "Relation Between Daily Flow vs. daily Flow ",
        fontsize=25, fontweight='bold'
    )

    # Adjust the position of the main title
    plt.subplots_adjust(top=0.95)

    # Show the plot
    plt.show()

relationship_daily_data_flow_vs_flow(merge_allfilter_df)
```

4.3 Monthly Average Relation of daily flow for all gauges

```
def monthly_avg_plot_flow():
    # List of gauge data
    gauges = [
        ('57005', monthly_avg_57005),
        ('57006', monthly_avg_57006),
        ('57004', monthly_avg_57004),
        ('57007', monthly_avg_57007),
        ('57015', monthly_avg_57015),
        ('57001', monthly_avg_57001),
        ('57002', monthly_avg_57002)
    ]

    # Create a set to track plotted combinations
    plotted_pairs = set()

    # Loop over each gauge and create a subplot for each of its relationships
    for gauge1_id, gauge1_data in gauges:
        # Find pairs involving the current gauge that have not yet been plotted
        pairs_to_plot = [(gauge2_id, gauge1_data, gauge2_data) for gauge2_id, gauge2_data in gauges
                         if gauge1_id != gauge2_id and (gauge1_id, gauge2_id) not in plotted_pairs
                         and (gauge2_id, gauge1_id) not in plotted_pairs]

        if not pairs_to_plot:
            continue  # Skip if there are no new pairs to plot for this gauge

        num_plots = len(pairs_to_plot)
```

```
        ncols = 3  # Number of columns in the subplot grid
        nrows = (num_plots + ncols - 1) // ncols  # Calculate rows needed based on the number of plots

        fig, axes = plt.subplots(
            nrows=nrows,
            ncols=ncols,
            figsize=(15, 5 * nrows)
        )
        fig.suptitle(
            f'Monthly average Relations Between Gauge {gauge1_id} with other gauge flow',
            fontsize=16, fontweight='bold', x=0.5, y=0.98
        )

        for ax, (gauge2_id, gauge1_data, gauge2_data) in zip(axes.flat, pairs_to_plot):
            sns.scatterplot(x=gauge1_data['gdf'], y=gauge2_data['gdf'], ax=ax)
            ax.set_title(f'Gauge {gauge1_id} vs Gauge {gauge2_id}', fontweight='bold')
            ax.set_xlabel(f'GDF at Gauge {gauge1_id}', fontweight='bold')
            ax.set_ylabel(f'GDF at Gauge {gauge2_id}', fontweight='bold')
            plt.setp(ax.get_xticklabels(), fontweight='bold')
            plt.setp(ax.get_yticklabels(), fontweight='bold')
            # Add the plotted pair to the set to avoid duplication
            plotted_pairs.add((gauge1_id, gauge2_id))

        # Hide any unused subplots
        if num_plots < nrows * ncols:
            for i in range(num_plots, nrows * ncols):
                fig.delaxes(axes.flat[i])

        plt.tight_layout(rect=[0, 0, 1, 0.96])
        plt.show()
monthly_avg_plot_flow()
```

4.4 Correlations Analysis on daily data

```
def correlation_daily_data(merge_allfilter_df):
    # Calculate the correlation matrix
    correlation_matrix = merge_allfilter_df.corr()

    # Reorder the columns to group flows and rainfall together
    columns_order = sorted(correlation_matrix.columns, key=lambda x: ('gdf' in x, x))
    correlation_matrix = correlation_matrix.loc[columns_order, columns_order]

    # Create a mask for the upper triangle
    mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

    # Set up the matplotlib figure
    plt.figure(figsize=(8, 5))

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(correlation_matrix, mask=mask, annot=True, fmt=".2f", cmap='coolwarm', center=0, square=True, linewidths=.5)

    plt.title('Correlation Between daily flow (gdf) and daily Rainfall (cdr)')
    plt.show()
correlation_daily_data(merge_allfilter_df)
```

Note: With correlation heatmap observation, Strong Correlations between gauges(GDF) which important factor to predict pattern and less Correlations between daily flow and rainfall

4.5 Distribution at Daily Mean Data

```
def dist_plot_daily_data(merge_allfilter_df):
    # List of gauge numbers
    gauges = ['57005', '57006', '57004', '57007', '57015', '57001', '57002']

    # Loop over each gauge to create the plots
    for gauge in gauges:
        plt.figure(figsize=(8, 4))

        # Plot CDR histogram with its own scale
        plt.subplot(1, 2, 1)
        sns.histplot(merge_allfilter_df[f'cdr_{gauge}'], bins=20, kde=True, color='blue')
        plt.title(f'Distribution of CDR at Gauge {gauge}', fontsize=10, fontweight='bold')
        plt.xlabel('CDR', fontsize=10, fontweight='bold')
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')
```

```
        plt.ylim(0, 3500)  # Adjust y-axis to a common scale

        # Plot GDF histogram with its own scale
        plt.subplot(1, 2, 2)
        sns.histplot(merge_allfilter_df[f'gdf_{gauge}'], bins=20, kde=True, color='green')
        plt.title(f'Distribution of GDF at Gauge {gauge}', fontsize=10, fontweight='bold')
        plt.xlabel('GDF', fontsize=10, fontweight='bold')
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')
        plt.ylim(0, 3500)  # Adjust y-axis to a common scale

        # Adjust layout and show the plot
        plt.tight_layout()
        plt.show()
dist_plot_daily_data(merge_allfilter_df)
```

4.6 Distribution at monthly Average Data

```
def monthly_avg_dist_plot():
    # List of gauge data dictionaries and their identifiers
    m_gauges = [
        ('57005', monthly_avg_57005),
        ('57006', monthly_avg_57006),
        ('57004', monthly_avg_57004),
        ('57007', monthly_avg_57007),
        ('57015', monthly_avg_57015),
        ('57001', monthly_avg_57001),
        ('57002', monthly_avg_57002)
    ]

    # Loop over each gauge to create the plots
    for gauge_id, gauge_data in m_gauges:
        plt.figure(figsize=(8, 4))

        # Plot CDR histogram
        plt.subplot(1, 2, 1)
        sns.histplot(gauge_data['cdr'], bins=20, kde=True, color='blue')
        plt.title(f'Monthly Average Distribution of CDR at Gauge {gauge_id}', fontsize=9, fontweight='bold')
        plt.xlabel('CDR', fontsize=10, fontweight='bold')
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')
        plt.ylim(0, 70)

        # Plot GDF histogram
        plt.subplot(1, 2, 2)
        sns.histplot(gauge_data['gdf'], bins=20, kde=True, color='green')
        plt.title(f'Monthly Average Distribution of GDF at Gauge {gauge_id}', fontsize=9, fontweight='bold')
        plt.xlabel('GDF', fontsize=10, fontweight='bold')
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')
        plt.ylim(0, 70)
        # Adjust layout and show the plot
        plt.tight_layout()
        plt.show()
monthly_avg_dist_plot()
```

4.7 Comparsion of mean with variance at 30 rolling window

```
def mean_var_comparsion_plot(merge_allfilter_df):
    # Calculate the rolling variance and mean with a window of 30 days
    monthly_variance = merge_allfilter_df.rolling(window=30, min_periods=1).var()
    monthly_means = merge_allfilter_df.rolling(window=30, min_periods=1).mean()

    # Create a plot for each column
    for column in merge_allfilter_df.columns:
        plt.figure(figsize=(15, 4))
        plt.plot(monthly_variance.index, monthly_variance[column], label=f'Variance of {column}', color='blue')
        plt.plot(monthly_means.index, monthly_means[column], label=f'Mean of {column}', color='red')
        plt.title(f' Variance and Mean Over Time for {column} before Log Transformation at 30 days window', fontsize=10, fontweight='bo
        plt.xlabel('Year', fontsize=10, fontweight='bold')
        plt.ylabel('Value', fontsize=10, fontweight='bold')
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')
        plt.legend(prop={'size': 10, 'weight': 'bold'})
        plt.grid(True)
```

```
        plt.show()
mean_var_comparsion_plot(merge_allfilter_df)
```

Variance is not stable there is no pattern found over months or sudden fluction over the year.
means is approximately constant over time but variance is highly volatile

5. Transformation

5.1 Applying Log Transformation

```
# Apply log transformation
log_transformed_daily_df = merge_allfilter_df.copy()
gdf_columns = [col for col in merge_allfilter_df.columns if col.startswith('gdf')]
cdr_columns = [col for col in merge_allfilter_df.columns if col.startswith('cdr')]
# Apply log transformation to gdf columns
log_transformed_daily_df[gdf_columns] = np.log(merge_allfilter_df[gdf_columns])
# Apply log(1 + x) transformation to cdr columns
log_transformed_daily_df[cdr_columns] = np.log1p(merge_allfilter_df[cdr_columns])
print(log_transformed_daily_df)
```

```
# Check for negative values
negative_values_exist = (log_transformed_daily_df < 0).any()

# Check for zero values
zero_values_exist = (log_transformed_daily_df == 0).any()

# Print the results
print("Negative values exist:", negative_values_exist)
print("Zero values exist:", zero_values_exist)
```

5.1.1 Distribution plot at log transformed data

```
# List of gauge IDs
gauges = ['57005', '57006', '57004', '57007', '57015', '57001', '57002']

# Loop over each gauge to create the plots
for gauge in gauges:
    plt.figure(figsize=(12, 4))

    # Plot CDR histogram
    plt.subplot(1, 2, 1)
    sns.histplot(log_transformed_daily_df[f'cdr_{gauge}'], bins=20, kde=True, color='blue')
    plt.title(f'Distribution of CDR at Gauge {gauge} after log transformation', fontsize=10, fontweight='bold')
    plt.xlabel('CDR', fontsize=10, fontweight='bold')
    plt.xticks(fontsize=10, fontweight='bold')
    plt.yticks(fontsize=10, fontweight='bold')

    # Plot GDF histogram
    plt.subplot(1, 2, 2)
    sns.histplot(log_transformed_daily_df[f'gdf_{gauge}'], bins=20, kde=True, color='green')
    plt.title(f'Distribution of GDF at Gauge {gauge} after log transformation', fontsize=10, fontweight='bold')
    plt.xlabel('GDF', fontsize=10, fontweight='bold')
    plt.xticks(fontsize=10, fontweight='bold')
    plt.yticks(fontsize=10, fontweight='bold')

    # Adjust layout and show the plot
    plt.tight_layout()
    plt.show()
```

Note: 1. Above distribution plot showing that after applied log transformation it appear more normalize which stabilize variance 2. For CDR most of days have approximately very low rainfall tend to zero.

5.1.2 Box plot at log transformed data

```
for column in log_transformed_daily_df.columns:
    plt.figure(figsize=(10, 2))
```

```
    plt.boxplot(
        log_transformed_daily_df[column].dropna(),
        vert=False,
        patch_artist=True,
        boxprops=dict(facecolor="Red")
    )
    plt.title(f'Box Plot for {column} after Log transformed')
    #plt.xlabel('CDR')
    plt.grid(True)
    plt.show()
```

5.1.3 Comparsion between Variance and Mean to check stablity of variance with mean after applied log Tranformation

```
# Calculate the rolling variance and mean with a window of 30 days on the transformed data
log_transform_monthly_variance = log_transformed_daily_df.rolling(window=30, min_periods=1).var()
log_transform_monthly_means = log_transformed_daily_df.rolling(window=30, min_periods=1).mean()

# Create a plot for each column
for column in log_transformed_daily_df.columns:
    plt.figure(figsize=(15, 4))
    plt.plot(log_transform_monthly_variance.index, log_transform_monthly_variance[column], label=f'Variance of {column}', color='blue')
    plt.plot(log_transform_monthly_means.index, log_transform_monthly_means[column], label=f'Mean of {column}', color='red')
    plt.title(f'Monthly Variance and Mean Over Time for {column} After Log Transformation at 30 days window', fontsize=10, fontweight='b
    plt.xlabel('Years', fontsize=10, fontweight='bold')
    plt.ylabel('Value', fontsize=10, fontweight='bold')
    plt.xticks(fontsize=10, fontweight='bold')
    plt.yticks(fontsize=10, fontweight='bold')
    plt.legend()
    plt.grid(True)
    plt.show()
```

5.1.4 Moving Average to Identify Underlying Trend at 7MA, double 7MA, 30MA, weighted MA

```
def ma_find_trend(log_transformed_daily_df):
    # List of column types and their corresponding labels
    column_types = [
        ('gdf', [col for col in log_transformed_daily_df.columns if 'gdf' in col]),
        ('cdr', [col for col in log_transformed_daily_df.columns if 'cdr' in col])
    ]

    # Loop through each column type ('gdf' and 'cdr') and process accordingly
    for label, columns in column_types:
        # Calculate moving averages for the columns
        seven_MA = log_transformed_daily_df[columns].rolling(7).mean()  # 7MA
        double_seven_MA = seven_MA.rolling(7).mean()  # Double 7MA
        thirty_MA = log_transformed_daily_df[columns].rolling(30).mean()  # 30MA
        Wma = log_transformed_daily_df[columns].ewm(halflife=7).mean()  # Weighted MA

        # Plot moving averages for each column
        for col in columns:
            plt.figure(figsize=(20, 6))
            plt.plot(log_transformed_daily_df[col], label=f'Actual Data ({col})', color='grey')
            plt.plot(seven_MA[col], label='7 MA', color='green')
            plt.plot(double_seven_MA[col], label='Double 7 MA', color='orange')
            plt.plot(thirty_MA[col], label='30 MA', color='red')
            plt.plot(Wma[col], label='Weighted MA', color='blue')

            # Adding labels and title
            plt.xlabel('years', fontsize=15, fontweight='bold')
            plt.ylabel('Value', fontsize=15, fontweight='bold')
            plt.title(f'Comparison of Actual Data, SMA, and DMA for {col}', fontsize=15, fontweight='bold')
            plt.xticks(fontsize=15, fontweight='bold')
            plt.yticks(fontsize=15, fontweight='bold')

            # Adding legend with bold font
            plt.legend(prop={'weight': 'bold'})

            # Showing plot
            plt.grid(True)
            plt.show()

ma_find_trend(log_transformed_daily_df)
```

5.1.5 Time Series Decomposition to find seasonality or trend (STL Decomposition)

```
# Function to perform STL decomposition and plot the components
def plot_stl_decomposition(data, column, seasonal=31):
    stl = STL(data[column], seasonal=seasonal)
    result = stl.fit()

    fig = result.plot()
    plt.figure(figsize=(15, 8))
    plt.suptitle(f'STL Decomposition of {column}', y=1.02)
    plt.show()

# Apply STL decomposition to each column in log_transformed
for column in log_transformed_daily_df.columns:
    plot_stl_decomposition(log_transformed_daily_df, column, seasonal=31)
```

5.1.6 Seasoanl plots for last 3 year to yearly seasonality

```
seasonal_plot_lg_transform_df = log_transformed_daily_df.copy()

# Ensure the index is a datetime index if it's not already
seasonal_plot_lg_transform_df.index = pd.to_datetime(log_transformed_daily_df.index)

# Extract year and day of year
seasonal_plot_lg_transform_df['Year'] = log_transformed_daily_df.index.year
seasonal_plot_lg_transform_df['DayOfYear'] = log_transformed_daily_df.index.dayofyear

# Filter the DataFrame to include only the last 5 years
last_5_years = seasonal_plot_lg_transform_df['Year'].unique()[-3:]
filtered_df = seasonal_plot_lg_transform_df[seasonal_plot_lg_transform_df['Year'].isin(last_5_years)]

# Function to plot seasonal patterns for each column
def plot_seasonal(data, column):
    plt.figure(figsize=(25, 8))

    colors = cm.get_cmap('tab20', len(data['Year'].unique()))

    for i, year in enumerate(data['Year'].unique()):
        yearly_data = data[data['Year'] == year]
        plt.plot(yearly_data['DayOfYear'], yearly_data[column], label=str(year), color=colors(i))

    # Customize the plot
    plt.title(f'Seasonal Plot of {column} (Last 3 Years)', fontsize=25, fontweight='bold')
    plt.xlabel('Day of Year', fontsize=20, fontweight='bold')
    plt.ylabel(column, fontsize=20, fontweight='bold')
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1), prop={'weight': 'bold', 'size': 15})
    plt.grid(True)

    # Set x-axis to show months
    months = pd.date_range('2022-01-01', periods=12, freq='MS').strftime('%b')
    plt.xticks(ticks=pd.date_range('2022-01-01', periods=12, freq='MS').dayofyear, labels=months, fontsize=20, fontweight='bold')

    plt.tight_layout()
    plt.show()

# List of columns to plot
columns_to_plot = filtered_df.columns.difference(['Year', 'DayOfYear', 'Day', 'Month'])
print(columns_to_plot)
# Plot seasonal patterns for all columns
for column in columns_to_plot:
    plot_seasonal(filtered_df, column)
```

5.1.7 Correlation after applied log transformation

```
def correlation_transformed_data(log_transformed_daily_df):
    # Calculate the correlation matrix
    correlation_matrix = log_transformed_daily_df.corr()

    # Reorder the columns to group flows and rainfall together
    columns_order = sorted(correlation_matrix.columns, key=lambda x: ('gdf' in x, x))
    correlation_matrix = correlation_matrix.loc[columns_order, columns_order]

    # Create a mask for the upper triangle
    mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

    # Set up the matplotlib figure
```

```
    plt.figure(figsize=(8, 5))

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(correlation_matrix, mask=mask, annot=True, fmt=".2f", cmap='coolwarm', center=0, square=True, linewidths=.5)

    plt.title('Correlation at log transformed datat between daily flow (gdf) and daily Rainfall (cdr)', fontweight = 'bold')
    plt.show()
correlation_transformed_data(log_transformed_daily_df)
```

5.2 Removing Seasonal Component From Data (Deaseasonalised)

```
log_transformed_deseasonalised_df = pd.DataFrame()
seasonal_component = pd.DataFrame()

def remove_seasonality(data, column, seasonal=7):
    stl = STL(data[column], seasonal=seasonal)
    result = stl.fit()
    #deseasonalized = result.trend+ result.resid
    deseasonalized = data[column]-result.seasonal
    if column == 'gdf_57005':
        seasonal_component[column] = result.seasonal

    return deseasonalized

# Apply STL decomposition and remove seasonality for each column
for column in log_transformed_daily_df.columns:
    log_transformed_deseasonalised_df[column] = remove_seasonality(log_transformed_daily_df, column, seasonal=31)
log_transformed_deseasonalised_df
```

5.3 Applied Box-cox Transformation over original cleaned data

```
boxcox_transformed_df = merge_allfilter_df.copy()
boxcox_lambdas = {}
for column in merge_allfilter_df.columns:
    # Shift the data to be strictly positive if necessary
    data = merge_allfilter_df[column]
    if any(data <= 0):
        shift = abs(min(data)) + 1  # Ensure all data is positive
        data = data + shift

    # Apply Box-Cox transformation and store the lambda value
    boxcox_transformed, lam = boxcox(data)
    boxcox_transformed_df[column] = boxcox_transformed
    boxcox_lambdas[column] = lam  # Store the lambda value for this column

print(boxcox_transformed_df)

# Function to reverse Box-Cox transformation after final prediction
def inverse_boxcox(y, lam):
    if lam == 0:
        return np.exp(y)
    else:
        return np.exp(np.log(lam * y + 1) / lam)


# Check for negative values
negative_values_exist = (boxcox_transformed_df < 0).any()

# Check for zero values
zero_values_exist = (boxcox_transformed_df == 0).any()

# Print the results
print("Negative values exist:", negative_values_exist)
print("Zero values exist:", zero_values_exist)
```

5.3.1 Box Plot at after applied box-cox Transformation over original cleaned data

```
for column in boxcox_transformed_df.columns:
    plt.figure(figsize=(10, 2))
    plt.boxplot(
        boxcox_transformed_df[column].dropna(),
        vert=False,
        patch_artist=True,
```

```
        boxprops=dict(facecolor="Red")
    )
    plt.title(f'Box Plot for {column} after Applied Box-cox transformation over Log transformed data', fontsize ='10', fontweight ='bold
    #plt.xlabel('CDR')
    plt.grid(True)
    plt.show()
```

5.3.2 Distribution plot after applied box-cox transformation over original cleaned data

```
# List of gauge IDs
gauges = ['57005', '57006', '57004', '57007', '57015', '57001', '57002']

# Loop over each gauge to create the plots
for gauge in gauges:
    plt.figure(figsize=(12, 4))

    # Plot CDR histogram
    plt.subplot(1, 2, 1)
    sns.histplot(boxcox_transformed_df[f'cdr_{gauge}'], bins=20, kde=True, color='blue')
    plt.title(f'Distribution of CDR at Gauge {gauge} after applied box-cox transformation', fontsize ='10', fontweight ='bold')
    plt.xlabel('CDR', fontsize ='10', fontweight ='bold')
    plt.xticks(fontsize ='10', fontweight ='bold')
    plt.yticks(fontsize ='10', fontweight ='bold')

    # Plot GDF histogram
    plt.subplot(1, 2, 2)
    sns.histplot(boxcox_transformed_df[f'gdf_{gauge}'], bins=20, kde=True, color='green')
    plt.title(f'Distribution of GDF at Gauge {gauge} after applied box-cox transformation', fontsize ='10', fontweight ='bold')
    plt.xlabel('GDF', fontsize ='10', fontweight ='bold')
    plt.xticks(fontsize ='10', fontweight ='bold')
    plt.yticks(fontsize ='10', fontweight ='bold')

    # Adjust layout and show the plot
    plt.tight_layout()
    plt.show()
```

5.4 Autocorrelation Function and Partial Auto-correlation function at log transformed data

```
# Function to plot ACF and PACF for a given gauge as subplots
def plot_acf_pacf_cdr(data, gauge_ids, lags=15):
    for gauge_id in gauge_ids:
        fig, ax = plt.subplots(1, 2, figsize=(10, 4))

        # Plot ACF
        plot_acf(data[f'cdr_{gauge_id}'], lags=lags, ax=ax[0])
        ax[0].set_xlabel('Lag', fontsize=12, fontweight='bold')
        ax[0].set_ylabel('Autocorrelation', fontsize=12, fontweight='bold')
        ax[0].set_title(f'ACF of rainfall data for {gauge_id}', fontsize=15, fontweight='bold')

        # Plot PACF
        plot_pacf(data[f'cdr_{gauge_id}'], lags=lags, ax=ax[1])
        ax[1].set_xlabel('Lag', fontsize=12, fontweight='bold')
        ax[1].set_ylabel('Partial Autocorrelation', fontsize=12, fontweight='bold')
        ax[1].set_title(f'PACF of rainfall data for {gauge_id}', fontsize=15, fontweight='bold')
        for axis in ax:
            plt.setp(axis.get_xticklabels(), fontweight='bold')
            plt.setp(axis.get_yticklabels(), fontweight='bold')
            for line in axis.get_lines():
                line.set_linewidth(2)
                line.set_markersize(6)
        plt.tight_layout()
        plt.show()

def plot_acf_pacf_gdf(data, gauge_ids, lags=15):
    for gauge_id in gauge_ids:
        fig, ax = plt.subplots(1, 2, figsize=(10, 4))

        # Plot ACF
        plot_acf(data[f'gdf_{gauge_id}'], lags=lags, ax=ax[0])
        ax[0].set_xlabel('Lag', fontsize=12, fontweight='bold')
        ax[0].set_ylabel('Autocorrelation', fontsize=12, fontweight='bold')
        ax[0].set_title(f'ACF of daily flow data for {gauge_id}', fontsize=15, fontweight='bold')

        # Plot PACF
        plot_pacf(data[f'gdf_{gauge_id}'], lags=lags, ax=ax[1])
```

```
        ax[1].set_xlabel('Lag', fontsize=12, fontweight='bold')
        ax[1].set_ylabel('Partial Autocorrelation', fontsize=12, fontweight='bold')
        ax[1].set_title(f'PACF of daily flow data for {gauge_id}', fontsize=15, fontweight='bold')
        for axis in ax:
            plt.setp(axis.get_xticklabels(), fontweight='bold')
            plt.setp(axis.get_yticklabels(), fontweight='bold')
            for line in axis.get_lines():
                line.set_linewidth(2)
                line.set_markersize(6)
        plt.tight_layout()
        plt.show()

# List of gauge IDs
gauge_ids = ['57005', '57006', '57004', '57007', '57015', '57001', '57002']

# Plot ACF and PACF for each gauge as subplots

plot_acf_pacf_cdr(log_transformed_daily_df, gauge_ids)
plot_acf_pacf_gdf(log_transformed_daily_df, gauge_ids)
```

6. Model Building and Evaluation and optimization

6.0 Train Test Split data

```
# Function to create lagged features with different lags for each column
def lagged_datas(df, lags_dict, target_var):
    lagged = pd.DataFrame()
    for col, lags in lags_dict.items():
        if col != target_var:
            for lag in range(1, lags + 1):
                lagged[f'{col}_lag_{lag}'] = df[col].shift(lag)
    return lagged

# lags_dict which consider by pacf plot
lags_dict = {
    'cdr_57002': 4,
    'gdf_57002': 5,
    'cdr_57015': 4,
    'gdf_57015': 5,
    'cdr_57006': 4,
    'gdf_57006': 5,
    'cdr_57007': 4,
    'gdf_57007': 5,
    'cdr_57004': 4,
    'gdf_57004': 5,
    'cdr_57001': 4,
    'gdf_57001': 5,
    'cdr_57005': 4,
}

target_col = 'gdf_57005'

# creating lagged data log_transformed_daily_df is your dataframe
lag_features = lagged_datas(log_transformed_daily_df, lags_dict, target_col)

# Merge the lagged features with the target variable
data_with_lags = pd.concat([log_transformed_daily_df[target_col], lag_features], axis=1).dropna()

np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
torch.manual_seed(42)
X = data_with_lags.drop(columns=[target_col])
y = data_with_lags[target_col]

# Split the data into train and test sets
X_train_ss, X_test_ss, y_train_ss, y_test_ss = train_test_split(X, y, test_size=0.3, random_state=42)
```

6.1 model building at log transformed data

6.1.0 Automatic Stepwise selection of linear model by using p-value

```
# Stepwise selection function
```

```
# Stepwise selection function
def stepwise_selection(X, y, initial_features=[], threshold_in=0.05, threshold_out=0.05, verbose=True):
    included = list(initial_features)
    while True:
        changed = False
        # Forward step
        excluded = list(set(X.columns) - set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included + [new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed = True
            if verbose:
                print(f'Add {best_feature} with p-value {best_pval}')

        # Backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # Use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max()
        if worst_pval > threshold_out:
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            changed = True
            if verbose:
                print(f'Drop {worst_feature} with p-value {worst_pval}')

        if not changed:
            break

    return included



# Cross-Validation with the selected features
def cross_validate_selected_features(X_train, y_train, selected_features, cv_folds=5):
    # Create KFold object for cross-validation
    kf = KFold(n_splits=cv_folds, shuffle=True, random_state=42)

    cv_mse_scores = []
    cv_r2_scores = []
    cv_mape_scores = []

    # Iterate through each fold
    for train_index, val_index in kf.split(X_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index][selected_features], X_train.iloc[val_index][selected_features]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        # Fit the model on the training fold
        model = sm.OLS(y_train_fold, sm.add_constant(X_train_fold)).fit()

        # Predict on the validation fold
        y_val_pred = model.predict(sm.add_constant(X_val_fold))

        # Calculate metrics
        mse = mean_squared_error(y_val_fold, y_val_pred)
        r2 = r2_score(y_val_fold, y_val_pred)
        mape = mean_absolute_percentage_error(y_val_fold, y_val_pred)

        # Store the scores
        cv_mse_scores.append(mse)
        cv_r2_scores.append(r2)
        cv_mape_scores.append(mape)

    # Return the average scores across all folds
    return {
        'CV MSE': np.mean(cv_mse_scores),
        'CV MSE Std': np.std(cv_mse_scores),
        'CV R-squared': np.mean(cv_r2_scores),
        'CV R-squared Std': np.std(cv_r2_scores),
        'CV MAPE': np.mean(cv_mape_scores),
        'CV MAPE Std': np.std(cv_mape_scores),
    }

# Perform stepwise selection on the complete data
selected_features = stepwise_selection(X, y)
print('Selected features:', selected_features)
```