DATA3001 - Group Report

Eu Shaun Lim, Rachel Ha

August 11, 2019

I declare that this assessment item is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere. I acknowledge that the assessor of this item may, for the purpose of assessing this item reproduce this assessment item and provide a copy to another member of the University; and/or communicate a copy of this assessment item to a plagiarism checking service (which may then retain a copy of the assessment item on its database for the purpose of future plagiarism checking). I certify that I have read and understood the University Rules in respect of Student Academic Misconduct.

Name / Student Number: . . . Eu Shaun Lim / z5156345 . . .

Signature, Date: . . . . , 11 / 08 / 2019

Name / Student Number: . . . Rachel Ha / z5161472 . . .

Signature, Date: . . . . , 11 / 08 / 2019

Table of Contents

Introduction

Pakistan is located on a great landmass north of the Tropic of Cancer. Four seasons happen throughout the year, which are the cool, dry winter from December to February to hot, dry spring from March to May, the rainy season or the Southwest Monsoon period which occurs in the summer from June to September and the retreating monsoon period happening from October to November. The heavy rainfall that occurs during the Southwest Monsoon can cause extreme flooding and this leads to drowning or ingestion of toxic water from the flood, which in turn causes deadly diseases, such as jaundice, gastro intestinal infections like typhoid and cholera (Pakistan Today 2018). One such massive flood happened in 2010, where nearly all of Pakistan was caught in heavy monsoon rain, affecting approximately 18 million people with a death toll of nearly 2,000 (Scott 2011). Therefore, the objective of this project is to improve the given rainfall forecast models in Pakistan, with the hope that flood detection will be improved and thus more lives will be saved.

Literature Review

Many studies have been done on rainfall forecasting using various methodologies. One such study focuses on using the global sea-surface temperature (SST) and mean sea-level pressure (SLP) to predict the summer rainfall in Pakistan using Multiple Linear Regression (MLR) and Principal Component Regression (PCR) (Adnan et al. 2017). Observation data from 1961 to 2004 were used as the training set, while data from 2005 to 2014 were the testing sets. Using MLR, the initial predictors for the model are the regions in Pakistan over the sea as the ocean has a much higher heat capacity than land. Stepwise regression is used while selecting predictors to ensure that multicollinearity does not happen and only predictors that are 5% significant were chosen in the final model. For PCR, the predictors are transformed using Principal Component Analysis (PCA) which removes multicollinearity in the predictors. PCA is combined with regression models using the training data to create a PCR model. It was found that both SLP and SST are significant in predicting rainfall in Pakistan, and that both MLR and PCR accurately captured the rainfall patterns. However, PCR outperformed MLR in the various evaluation methods such as correlation coefficient, root mean square error (RMSE), mean absolute error (MAE) and mean bias. More importantly, MLR made prediction errors for anomaly rainfalls which PCR predicted fairly well.

Chai et al. (2017) utilises Artificial Neural Networks (ANN) to predict the rainfall in Kuching, Malaysia. The observed data is collected from 2009 to 2013 and precipitation is classified into light, moderate, heavy and very heavy. Back Propagation Neural Network (BPNN) and Radial Basis Function Neural Network (RBFN) were used to forecast rainfall. BPNN produced inconsistent results compared to RBFN, hence more training and validation are required for BPNN in order to generate a more accurate result. Chai et al. (2017) suggests that the inconsistent results observed in BPNN are due to random assignment of the weights in the network as compared to RBFN which involves the adjustment of “spread” of the network in the single-hidden-layer experiment. BPNN was more sensitive with the different number of hidden neurons used compared to the RBFN model. BPNN achieved a higher accuracy than RBFN with the use of 10 hidden neurons, but RBFN model can be trained faster than BPN model. By comparing the mean squared error (MSE) for both models, BPNN with 10 hidden neurons has a lower MSE than RBFN, indicating that BPNN performs better than RBFN.

Nair et al. (2017) also used ANN to predict the summer monsoon rainfall over the Indian region, but instead of looking at observed data, they used forecasted data from Global Climate Models (GCM). Predictors for the ANN are the precipitation variables from the GCMs, and the target is the observed rainfall. BPNN is used with one hidden layer in the structure, and double-cross-validation is employed to find the optimal number of hidden neurons and prevent overfitting. Pre-processing was done by randomizing the data to avoid selection bias before applying Min-Max transformation. During each iteration of development, RMSE and MAE is evaluated while increasing hidden neurons. Nair et al. (2017) noticed that after a certain point, RMSE and MAE in the testing data increases while in the training data the errors are unchanged. The best ANN model is chosen based on the smallest RMSE and MAE values. It is then evaluated by comparing against the ensemble mean of each member of the GCMs. It was found that the ANN model was very efficient and predicted rainfall anomalies quite accurately. In addition, it also predicted the rainfall spread close to the observed data during the monsoon months.

On the other hand, Shabib et al. (2018) used five different data mining techniques for rainfall prediction in Lahore, capital of Punjab province in Pakistan. The aim of the research was to analyze the performance of data mining techniques such as Support Vector Machine (SVM), Naïve Bayes, k Nearest Neighbor, Decision Tree and Multilayer Perceptron, on rainfall prediction in Lahore city using a classification framework. Weka is used in this study for classification and performance analysis. It is developed in Java language at the University of Waikato, New Zealand, which is one of the extensively used data mining software. The dataset that they used was obtained from the weather forecasting website from the 1st December 2005 to 31st November 2017, which the following attributes: Temperature, Atmospheric Pressure (weather station), Atmospheric Pressure (sea level), Pressure Tendency, Relative Humidity, Mean Wind Speed, Minimum Temperature, Maximum Temperature, Visibility, Dew Point Temperature. To evaluate how well the data mining techniques predict rainfall in Lahore city, three evaluation parameters used for this study were Precision, aims to evaluate the true positive entities with respect to false positive entities, Recall, which aids to evaluate the true positive entities with respect to the non- classified false negative entities, and F-measure, which provides the average of precision and recall. After the comparative analysis, the data mining techniques used showed good outcomes for no-rain in all accuracy measures but these techniques did not perform well for rain class. The reasons behind this may include the absence of important climatic attributes in dataset and the overall lowe rate of rainfall in the city.

Kamath and Kamath (2018) used Autoregressive Integrated Moving Average (ARIMA), Artificial Neural Network (ANN) and Exponential Smoothing State Space (ETS) to simulate monthly rainfall in Indukki district of Kerala. The duration of the datasets was from January 2006 to December 2016. They did a time series exploratory analysis, to analyze the trends before building time-series model. The time series data was visualized by plotting time series plot and correlation plot. ARIMA model is fit by adding seasonal component. Before implementing ARIMA model, Augmented DIckey- Fuller Test was used to reveal if the time series is stationary enough to do time-series modelling. The correlation plots such as Auto Correlation Function (ACF) and Partial Correlation Function (PACF) plots are plotted to help identifying the type of stationary series. On the other hand, ANN model is being done through 20 networks with three hidden nodes, by using “caret” package and “nnetr” function in R. The ETS model is designed by using “ets” function in R. All the models were used to predict rainfall for the next three years. To evaluate the performance of the models on how well the models have predicted rainfall, they used Root Mean Squared Error (RMSE) and model fit. The comparative analysis revealed that ARIMA model accurately forecasts the rainfall with less error.

Olatayo and Taiwo (2014) utilized autoregressive integrated moving average (ARIMA), the emerging fuzzy time series (FST) model and the non-parametric method (Theil’s regression) to model and predict the behavioural rainfall pattern in Nigeria. This study used 31 years of annual rainfall data from year 1982 to 2012 of Ibadan South West, Nigeria. The fuzzy time series model divided the dataset into 13 intervals and the interval with the largest number of rainfall data is being divided into 4 subintervals of equal length again. Three rules were used to determine if the forecast value under FST is upward 0.75 point, middle or downward 0.25 point. ARIMA (1,2,1) was used to derive the weights and the regression coefficients whereas theil’s regression was used to fit linear model. This study used mean squared forecast error (MAE), root mean square forecast error (RMSE) and coefficient of determination to evaluate the performance of the models. They concluded that FTS model outperformed all the models.

Material and Methods

The software that we intend to use are R/RStudio and Python/Jupyter Notebook. The data that we were given consists of the observed rainfall data and the forecasted rainfall data from various forecasting models. The observed data spans from January 2019 to June 2019 while the forecasted data contains data from the last 4 months (February 2019 - June 2019). There are 84 weather stations across Pakistan that record these data independently at 3 hour intervals. There is a dictionary that contains the latitude, longitude and height above sea level for each of the stations. For standardization purposes, the date and time are recorded in Coordinated Universal Time (UTC). The observation data is stored in daily and hourly formats, where the hourly data is stored at 3 hour intervals starting at 0000 UTC (5am PKT).

For our analysis, we will be using the hourly datasets, specifically from the weather stations in Punjab. After data cleaning for the weather stations is completed, we will aggregate the weather stations by calculating the average values for the weather stations in Punjab. Hence, the analysis and results that we obtain from this report is representative of the entire region of Punjab.

The data that we will be using is a time series data, thus we will be fitting our data with an autoregressive integrated moving average (ARIMA) model and subsequently use the final model to forecast 7 days of rainfall in Punjab. Multiple graphical methods will be used to aid the model building process such as time series plots, pairwise scatterplots, histograms and the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots. A brief description of the variables in the hourly dataset is provided below:

- STATNO: Weather station corresponding to the data
- YYYY, MM, DD, HH: Year, month, day and time for each observation
- TEMPERATR (Temperature): A physical quantity measuring hot and cold at a given time
- WET TEMP (Wet-bulb temperature): the temperature read by a thermometer covered in a water-soaked cloth over which air is passed
- DEWPOINT: Atmospheric temperature below which water droplets begin to condense and dew can form
- RELHUMIDY (Relative humidity): The ratio of the partial pressure of water vapor to the equilibrium vapor pressure of water at a given temperature
- SEALVLPRS (Sea-level pressure): The atmospheric pressure at sea level
- SURFPRESS (Surface pressure): The atmospheric pressure at a location on Earth's surface
- WINDSPEED: A fundamental atmospheric quantity caused by air moving from high to low pressure
- DIRECTION (Wind direction): Direction of the wind from which it originated
- RAIN 1: Amount of rain observed in the last `x` hours, where `x` is the value associated in RPeriod 1
- RAIN 2: Amount of rain observed in the last `x` hours, where `x` is the value associated in RPeriod 2
- RPeriodi 1 (Rain Period 1): Period which the amount of rain is observed in RAIN 1
- RPeriod 2 (Rain Period 2): Period which the amount of rain is observed in RAIN 2
- TLTCLOUD (Total cloud coverage): The fraction of the sky covered by all visible clouds
- LOW CLOUD (Low lying clouds): Low level clouds consisting of Stratus, Stratocumulus and Nimbostratus clouds
- VISIBLITY: A measure of the distance at which an object and light can be clearly discerned

Analyses and Results

```
# import libraries
from datetime import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn; seaborn.set()
from statsmodels.tsa.arima_model import ARMA
```

Initial Exploratory Data Analysis

```
import glob

# combining all of the hourly data
df_all = pd.concat([pd.read_csv(f) for f in glob.glob("pmd_data/*hour.csv")],
                    ignore_index = True)
df_all.rename(columns={' RAIN 1  ': 'RAIN1',
                       ' RAIN 2   ': 'RAIN2',
                       'RPeriodi 1': 'RPeriod1',
                       'RPeriod 2': 'RPeriod2',
                       'TEMPERATR': 'TEMP'}, inplace = True)
print(df_all.RPeriod1.value_counts())
```

```
## -9999.0    62930
##  12.0      23429
##  6.0       11279
##  18.0      10646
##  0.0          88
##  3.0          54
##  24.0          6
##  15.0          3
##  1.0           3
##  2.0           3
## Name: RPeriod1, dtype: int64
```

```
print(df_all.RPeriod2.value_counts())
```

```
##  3.0       86606
## -9999.0    21648
##  0.0          88
##  12.0         32
##  6.0          31
##  18.0         16
##  15.0         11
##  2.0           4
##  1.0           3
##  9.0           1
##  24.0          1
## Name: RPeriod2, dtype: int64
```

We will first look at the hourly dataset as a whole by combining all the stations. Initial exploratory data analysis shows that most of the observation in RAIN2 is from the last 3 hours, while RAIN1 values are mostly spread out between 6, 12 and 18 hours. We will have to standardize the rain observations into one time period to reduce any bias for that given time. Instead of finding a method to combine RAIN1 and RAIN2, we could just use RAIN2 with RPeriod2 of 3.0 since most of the observations are already standardized into 3 hour rainfall. But before we do that, we first need to check how many among the 86,606 observations are null to ensure that we do not lose any more data.

```
df_test = df_all[df_all.RPeriod2 == 3.0]
print("Frequency of null values: ", (df_test.RAIN2 == -9999).sum(), df_test.shape)
```

```
## Frequency of null values:  4558 (86606, 21)
```

```
print("Frequency of zero values: ", (df_test.RAIN2 == 0).sum())
```

```
## Frequency of zero values:  77361
```

There are 4,558 null observations out of 86,606 observations. Additionally, 77,361 observations are zero value, which is inline with the fact that rainfall is zero-inflated. Since there is only a 5% loss in data, we choose to only use the RAIN2 observation data with RPeriod2 of 3.0.

```
df_all.drop(['RAIN1', 'RPeriod1'], axis = 1, inplace = True)
```

Data Cleaning

We will now analyze only one station data and clean it, with the assumption that the data in all the weather stations have a similar pattern of missing or misclassified data. Therefore, we will create a function that will clean the other stations' data in a similar manner as this one station.

```
df = pd.read_csv('pmd_data/OBS_n41504_hour.csv')
df.head()
```

```
##    RECDNO  STATNO  YYYY  MM  ...  RPeriod 2  TLTCLOUD   LOW CLOUD  VISIBLITY
## 0       2   41504  2019   1  ...        3.0       0.0     -9999.0     2000.0
## 1       0   41504     0   0  ...        0.0       0.0         0.0        0.0
## 2       0   41504     0   0  ...        0.0       0.0         0.0        0.0
## 3       5   41504  2019   1  ...    -9999.0   -9999.0     -9999.0    -9999.0
## 4       6   41504  2019   1  ...    -9999.0   -9999.0     -9999.0    -9999.0
##
## [5 rows x 21 columns]
```

Initial impressions of the data shows that we need to clean up the data by:

- Renaming the columns
- Removing whitespaces from the column names
- Replace -9999 values with proper NaN values
- Remove any invalid date rows
- Remove any rows where all values are null
- Merge 'YYYY', 'MM', 'DD', 'HH' into a pandas datetime format for easier handling

```
# rename columns and remove whitespaces
df.columns = df.columns.str.strip()
df.rename(columns={'RAIN 1': 'RAIN1',
               'RAIN 2': 'RAIN2',
               'RPeriodi 1': 'RPeriod1',
               'RPeriod 2': 'RPeriod2',
               'TEMPERATR': 'TEMP',
               'RELHUMIDY': 'RELHUMIDITY',
               'VISIBLITY': 'VISIBILITY',}, inplace = True)

# replace -9999 with NaN values
df.replace(-9999.0, np.nan, inplace = True)

# remove rows where date is invalid
df.drop(df[(df.YYYY == 0) | (df.MM == 0) | (df.DD == 0)].index, inplace = True)

# remove rows where all values are null
df.dropna(axis=0, subset=['TEMP'], inplace = True)

# convert date to datetime format
df['TIME'] = df['YYYY'].map(str) + '-' + df['MM'].map(str) + '-' + df['DD'].map(str) + ':' + df['HH'].ma
df['TIME'] = pd.to_datetime(df['TIME'], format = '%Y-%m-%d:%H')
df.drop(['YYYY','MM','DD','HH','RECDNO'], axis = 1, inplace = True)

# move time to the first column
cols = list(df.columns.values)
cols = cols[-1:] + cols[:-1]
df = df.loc[:,cols]
```

```
# remove RAIN1 and RPeriod1 (explanation made below)
df.drop(['RAIN1', 'RPeriod1'], axis = 1, inplace = True)

df.head()
```

```
##                    TIME  STATNO  TEMP  ...  TLTCLOUD  LOW CLOUD  VISIBILITY
## 0  2019-01-01 00:00:00   41504  -8.0  ...       0.0        NaN      2000.0
## 8  2019-01-02 00:00:00   41504  -6.0  ...     100.0        NaN      2000.0
## 9  2019-01-02 03:00:00   41504  -4.5  ...     100.0        NaN      4000.0
## 11 2019-01-02 09:00:00   41504   5.0  ...      63.0        NaN      4000.0
## 12 2019-01-02 12:00:00   41504   2.0  ...     100.0        NaN      2000.0
##
## [5 rows x 15 columns]
```

In addition, we will also:

- Use only RAIN2 as our target variable
- Shift the rain values one row up
- Impute any missing values by linear interpolation

Our justification for using only RAIN2 and not RAIN1 was explained in the section above. For shifting the rain observations, the current rainfall values that we have now are observations for the last 3 hours, which is in the past, while the other variables are values at the specific time which is the present. Essentially, we would be using be using our data at a given time to predict rainfall for the past 3 hours from that time. Hence, we need to shift the rainfall values back by 3 hours so that we are using the current data to predict rainfall for the next 3 hours. After shifting the rain observations to their correct timestamp, there will be rows where there are only rain observations and no other variables recorded and vice versa. For these rows, we will need a method to fill in the missing data. Linear interpolation estimates the missing values by fitting a straight line between the previous and next non-missing data. Imputing data by linear interpolation ensures that a smooth curve and no outrageous values will be generated.

To shift the rain observation into their correct timestamp, we will create a new DataFrame with a time period of 3 hours and merge the rain observations.

```
# Separating rainfall observation into another dataframe
rain = df[['TIME','RAIN2','RPeriod2']]
df.drop(['RAIN2','RPeriod2'], axis=1, inplace=True)
rain.head()
```

```
##                    TIME  RAIN2  RPeriod2
## 0  2019-01-01 00:00:00    0.0       3.0
## 8  2019-01-02 00:00:00    NaN       3.0
## 9  2019-01-02 03:00:00    0.0       3.0
## 11 2019-01-02 09:00:00    0.0       3.0
## 12 2019-01-02 12:00:00    NaN       3.0
```

```
# create new date range for every 3 hours
start = rain.TIME.min()
end = rain.TIME.max()
date_range = pd.date_range(start=start, end=end, freq='3H')
date_range = pd.to_datetime(date_range)

# create new dataframe of just time and rain
new_df = pd.DataFrame(date_range, columns=['TIME'])
rain = pd.merge(new_df, rain[['TIME', 'RAIN2']], how='left', on='TIME')
```

```
# shift rainfall values up one row
rain[['RAIN2']] = rain.RAIN2.shift(-1)
rain.head()
```

```
##                  TIME  RAIN2
## 0 2019-01-01 00:00:00    NaN
## 1 2019-01-01 03:00:00    NaN
## 2 2019-01-01 06:00:00    NaN
## 3 2019-01-01 09:00:00    NaN
## 4 2019-01-01 12:00:00    NaN
```

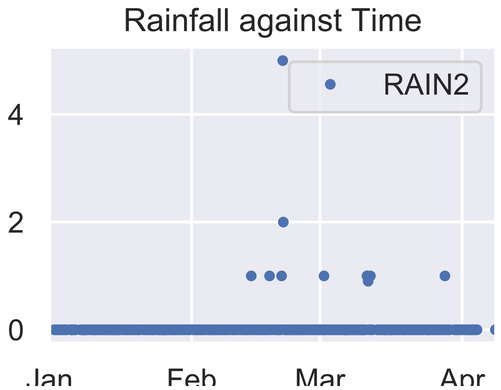
Now that we have the rain observation at their correct time stamps, we will merge it back into the original dataset.

```
# merge rain back into original dataframe
df2 = pd.merge(rain, df, how='left', on='TIME')
df2.head()
```

```
##                  TIME  RAIN2   STATNO  ...  TLTCLOUD  LOW CLOUD  VISIBILITY
## 0 2019-01-01 00:00:00    NaN  41504.0  ...       0.0        NaN      2000.0
## 1 2019-01-01 03:00:00    NaN      NaN  ...       NaN        NaN         NaN
## 2 2019-01-01 06:00:00    NaN      NaN  ...       NaN        NaN         NaN
## 3 2019-01-01 09:00:00    NaN      NaN  ...       NaN        NaN         NaN
## 4 2019-01-01 12:00:00    NaN      NaN  ...       NaN        NaN         NaN
##
## [5 rows x 14 columns]
```

Plotting rain againt time shows that rainfall for this particular station is not very occurent. We may need to look at other stations that has frequent rain.

```
# plotting rainfall against time
rain.plot(x='TIME', y='RAIN2', style='.', figsize=(3,2), title= "Rainfall against Time")
plt.show()
```



We now fill in any missing values in the explanatory variables by interpolating them using the linear method. This draws a straight line between any available data and and chooses the value from the line. This happens when we shift the rain observation to a time where no data was recorded as we would only have the rain observation for that time and no variables to explain it.

```
df2 = df2.interpolate(method='linear')
df2.dropna(axis=1, how='all', inplace = True)
df2.isnull().sum()
```

```
## TIME           0
## RAIN2          8
## STATNO         0
## TEMP           0
## DEWPOINT       0
## RELHUMIDITY    0
## WINDSPEED      0
## DIRECTION      0
## TLTCLOUD       0
## VISIBILITY     0
## dtype: int64
```

It looks like we still have some null values for our RAIN2 variable and they are at the first few rows of the dataset. This is because interpolation method only fills in null values between two non-null values, and since the remaining null values are at the start of our data, it cannot be interpolated. For these null values, we will simply replace them with zeros, since rain has a zero-inflated distribution, most of the rain observation will be zeros and thus by replacing them with zeros there is a low chance of having false information in our data.

```
# fill remaining missing RAIN2 values with 0
df2.RAIN2.fillna(0, inplace= True)
df2.head()
```

```
##                  TIME  RAIN2   STATNO  ...  DIRECTION  TLTCLOUD  VISIBILITY
## 0 2019-01-01 00:00:00    0.0  41504.0  ...       0.00       0.0      2000.0
## 1 2019-01-01 03:00:00    0.0  41504.0  ...      28.75      12.5      2000.0
## 2 2019-01-01 06:00:00    0.0  41504.0  ...      57.50      25.0      2000.0
## 3 2019-01-01 09:00:00    0.0  41504.0  ...      86.25      37.5      2000.0
## 4 2019-01-01 12:00:00    0.0  41504.0  ...     115.00      50.0      2000.0
##
## [5 rows x 10 columns]
```

Next, we will classify rainfall into two categories, “heavy rainfall” and “little to no rainfall”. According to the US Geological Survey (https://water.usgs.gov/edu/activity-howmuchrain-metric.html), we define any rainfall above moderate intensity to be greater than 0.5 mm per hour, which for our dataset is equivalent to 1.5 mm per 3 hours.

```
df2['RAIN'] = [1 if x >= 1.5 else 0 for x in df2['RAIN2']]
df2[df2.RAIN == 1]
```

```
##                    TIME  RAIN2   STATNO  ...  TLTCLOUD  VISIBILITY  RAIN
## 406 2019-02-20 18:00:00    3.0  41504.0  ...     100.0      2000.0     1
## 407 2019-02-20 21:00:00    5.0  41504.0  ...     100.0      2000.0     1
## 408 2019-02-21 00:00:00    2.0  41504.0  ...     100.0      2000.0     1
##
## [3 rows x 11 columns]
```

This particular station is not very useful in predicting heavy rain since there is not much heavy rain observation.

Selecting Stations

Now, we need to decide which station data to use to create our model. For our analysis, we will be using all the station data across the Punjab region by getting the average value for all the stations in Punjab region. This means that any results we obtain will be representative of the whole Punjab region.

```
import re
# station numbers that are located in the Punjab region
stations = [41571,41573,41577,41592,41594,41597,41598,41599,41600,41630,
            41633,41634,41635,41636,41638,41639,41640,41641,41642,41646,
            41652,41669,41670,41672,41675,41676,41678,41679,41680,41700,
            41701,41716,41718]

# create empty DataFrame
start = '2019-01-01 00:00:00'
end = '2019-06-19 21:00:00'
date_range = pd.date_range(start=start, end=end, freq='3H')
date_range = pd.to_datetime(date_range)

df = pd.DataFrame(columns=['RAIN2','TEMP','WET TEMP','DEWPOINT','RELHUMIDITY',
                           'SEALVLPRS','SURFPRESS','WINDSPEED','DIRECTION',
                           'TLTCLOUD', 'LOW CLOUD','VISIBILITY'],
                  index = date_range)
df.fillna(0, inplace = True)

i = 0
for f in glob.glob("pmd_data/*hour.csv"):
    statno = int(re.findall('\d+',f)[0])
    if int(statno) in stations:
        # read and clean data
        x = pd.read_csv(f)
        x = clean_obs_before(x)
        x = shift_rain(x)
        x.set_index('TIME', inplace = True)
        # sum up all values in each variable for each station in Punjab
        if len(x) == 1360:
            i = i + 1
            x.fillna(0, inplace = True)
            x = x.apply(pd.to_numeric)
            df = df + x

# average values over all stations
df = df/i

# remove NA columns
df.drop(['WET TEMP','SURFPRESS','LOW CLOUD'], axis = 1, inplace = True)

# classify rain to be 1 if heavy rain and 0 if not
df['RAIN'] = [1 if x >= 1.5 else 0 for x in df['RAIN2']]

# export to csv
# df.to_csv('stations_punjab_hour.csv')
```

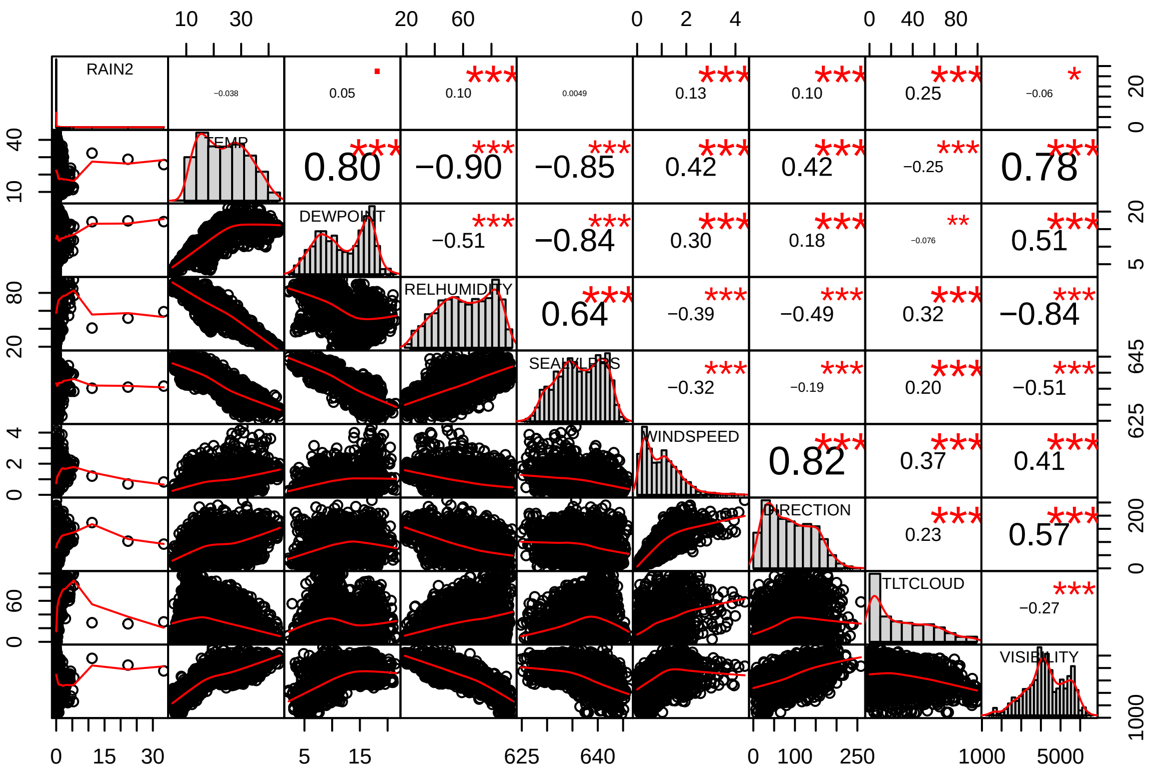
Time Series Analysis of Rainfall

Our dataset, stations_punjab_hour.csv, consists of the average values of all variable calculated at a specific time stamp for all 27 stations in Punjab. The dataset consists of our response variable RAIN2 and our explanatory variables TEMP, DEWPOINT, RELHUMIDITY, SEALVLPRS, WINDSPEED, DIRECTION, TLTCLOUD and VISIBILITY. It also has a binary variable RAIN which classifies RAIN2 into heavy rainfall and little or no rainfall. 1 represents heavy rainfall; 0 represents little or no rainfall.

```r
setwd("C:/group-report-rainfall/")
new_df <- read.csv("stations_punjab_hour.csv", header=TRUE)
summary(new_df)
```

```
##                     X             RAIN2               TEMP
##  2019-01-01 00:00:00:   1   Min.   : 0.00000   Min.   : 4.931
##  2019-01-01 03:00:00:   1   1st Qu.: 0.00000   1st Qu.:13.305
##  2019-01-01 06:00:00:   1   Median : 0.00000   Median :21.139
##  2019-01-01 09:00:00:   1   Mean   : 0.26201   Mean   :21.717
##  2019-01-01 12:00:00:   1   3rd Qu.: 0.06759   3rd Qu.:29.185
##  2019-01-01 15:00:00:   1   Max.   :33.38889   Max.   :44.148
##  (Other)            :1354
##     DEWPOINT       RELHUMIDITY      SEALVLPRS       WINDSPEED
##  Min.   : 2.137   Min.   :18.82   Min.   :624.8   Min.   :0.0000
##  1st Qu.: 8.315   1st Qu.:44.88   1st Qu.:633.0   1st Qu.:0.3852
##  Median :12.485   Median :61.03   Median :636.7   Median :0.8722
##  Mean   :12.306   Mean   :60.39   Mean   :636.6   Mean   :0.9872
##  3rd Qu.:16.583   3rd Qu.:77.79   3rd Qu.:640.6   3rd Qu.:1.4144
##  Max.   :21.544   Max.   :94.73   Max.   :646.1   Max.   :4.3778
##
##    DIRECTION         TLTCLOUD       VISIBILITY        RAIN
##  Min.   :  0.00   Min.   : 0.00   Min.   :1192   Min.   :0.00000
##  1st Qu.: 41.76   1st Qu.: 5.63   1st Qu.:3628   1st Qu.:0.00000
##  Median : 82.59   Median :23.15   Median :4241   Median :0.00000
##  Mean   : 89.02   Mean   :29.91   Mean   :4274   Mean   :0.04412
##  3rd Qu.:131.85   3rd Qu.:50.63   3rd Qu.:5111   3rd Qu.:0.00000
##  Max.   :258.52   Max.   :99.56   Max.   :6667   Max.   :1.00000
##
```

```r
# install.packages("PerformanceAnalytics")
library("PerformanceAnalytics", quietly = TRUE)

rm_new_df_in <- new_df[,c(-1,-11)] #exclude x

#Did not change the structure of Rain, leave it as int instead of factor
chart.Correlation(rm_new_df_in, histogram=TRUE, pch=19)
```



The diagram above provides a brief overview of our variables. This helps us to get an initial feel for our data by having a look at their distributions and relationships with other variables. It has a histogram to visualize the distribution of a variable and a scatter plot which visualizes the relationship between two variables. The values in the upper triangular area are the correlation coefficient, which is a measurement of both the strength and direction of the linear relationship between two continuous variables. The greater the absolute value of the correlation coefficient, the stronger the relationship and vice versa. The red stars in the plot represents the significance level of the hypothesis test for the correlation relationship between two continuous variables. Each significance level is associated to a symbol. Three stars indicates that it has p value of 0.001; two stars represents p value of 0.01; one star shows that it has p value of 0.1; no stars indicates the p value of 1.

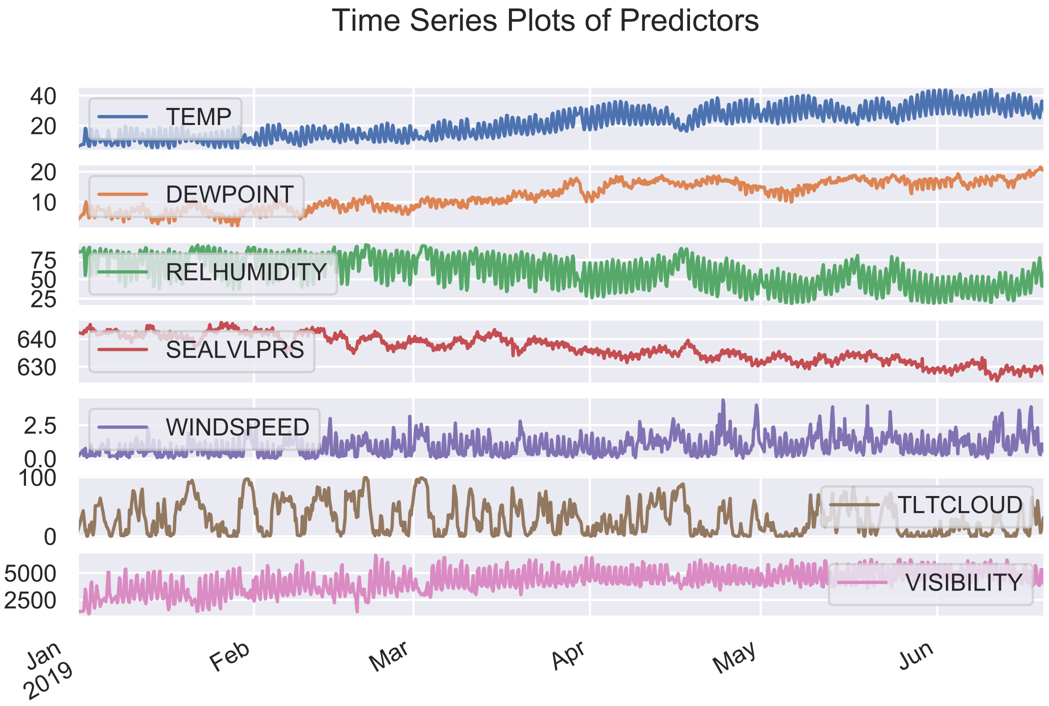
Our response, RAIN2, has a very right-skewed histogram with a very large bar at the left end of the histogram and nothing much everywhere else. This shows that the values for RAIN2 are zero-inflated indicating not much rain has happened in our dataset. The first row of the diagram shows the correlation coefficient and significance level between RAIN2 and our explanatory variables. In particular, RELHUMIDITY, WINDSPEED, DIRECTION and TLTCLOUD shows significance with our responce, RAIN2.

The scatter plots show no multicollinearity between our explanatory variables, however there is a very strong relationship between TEMP and the other explanatory variables DEWPOINT, RELHUMIDITY, SEALVLPRS and VISIBILITY, and the p-values indicate significance at a 5% level for TEMP with all other explanatory variables. In turn, TEMP has no relationship nor is it significant with our response variable RAIN2 at all. Other explanatory variables that has a strong relationship with each other are DEWPOINT and SEALVLPRS, RELHUMIDITY and VISIBILITY and lastly WINDSPEED and DIRECTION.

Early analysis from the diagram above shows that the final model might not include TEMP but will have RELHUMIDITY, WINDSPEED, DIRECTION and TLTCLOUD as the predictors.

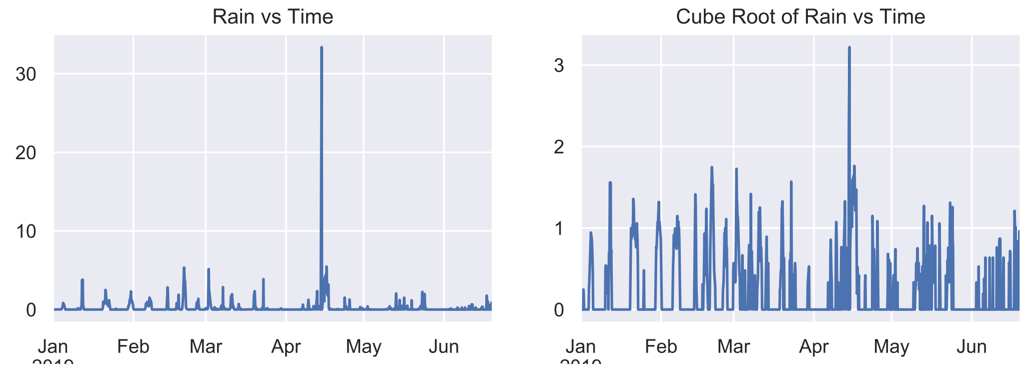
```
df = pd.read_csv('stations_punjab_hour.csv', index_col = 0)
df.set_index(pd.to_datetime(df.index), inplace = True)
```

```
_ = df[['TEMP','DEWPOINT','RELHUMIDITY','SEALVLPRS','WINDSPEED','TLTCLOUD','VISIBILITY']].plot(subplots
_ = plt.suptitle('Time Series Plots of Predictors')
plt.show()
```



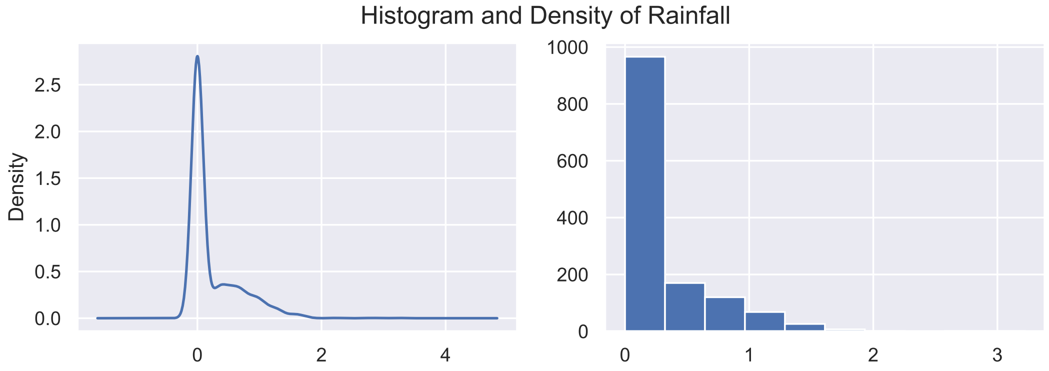
The main assumption for an ARIMA model is that our data have to be stationary. To achieve stationarity, all variables must have a constant mean and variance with no trends and seasonality. A time series plot of our explanatory variables show that trends can be seen in TEMP, DEWPOINT, RELHUMIDITY and SEALVLPRS, which means that they are not stationary. To investigate further, we have to run the Augmented Dickey-Fuller test which checks at a 5% significance level if our variables are stationary.

```
_ = plt.figure(figsize=(10,3))
_ = plt.subplot(121)
_ = df.RAIN2.plot(title = "Rain vs Time")
_ = plt.subplot(122)
df[['RAIN2']] = np.cbrt(df[['RAIN2']])
_ = df.RAIN2.plot(title = "Cube Root of Rain vs Time")
plt.show()
```



A time series plot of our response variable, RAIN2, shows a very large spike during the middle of April, while normal rainfall ranges from 0 mm to 5 mm. We will apply a cube root transformation to our response variable to reduce the very large spike of rain in April.

```
_ = plt.figure(figsize=(10,3))
_ = plt.subplot(121)
_ = df.RAIN2.plot(kind='kde', ax=plt.gca())
_ = plt.subplot(122)
_ = df.RAIN2.hist(ax=plt.gca())
_ = plt.suptitle("Histogram and Density of Rainfall")
plt.show()
```



Plotting a histogram for our response variable shows that rainfall has a zero-inflated distribution, that is the most common amount of rainfall is no rain at all.

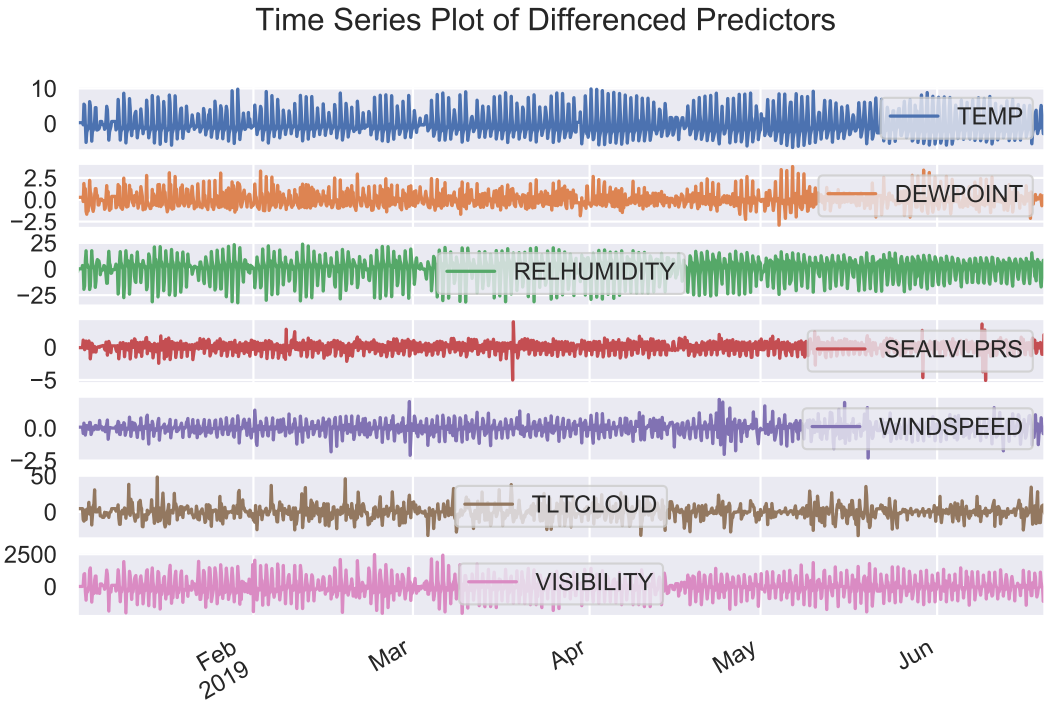
```
# checking if data is stationary
from statsmodels.tsa.stattools import adfuller

for name, values in df.iteritems():
    result = adfuller(values)
    print('p-value for %s: %4f' % (name,result[1]))
```

```
## p-value for RAIN2: 0.000000
## p-value for TEMP: 0.673552
## p-value for DEWPOINT: 0.647437
## p-value for RELHUMIDITY: 0.106152
## p-value for SEALVLPRS: 0.545265
## p-value for WINDSPEED: 0.000000
## p-value for DIRECTION: 0.000000
## p-value for TLTCLOUD: 0.000000
## p-value for VISIBILITY: 0.017265
## p-value for RAIN: 0.000000
```

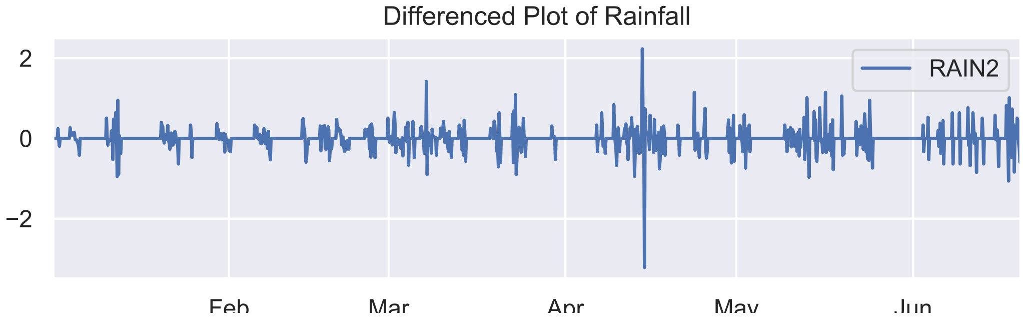
A statistical test we can carry out to confirm stationarity in our variables is the Augmented Dickey-Fuller test, where the null hypothesis is there exists a unit root in our variables. From the Augmented Dickey-Fuller test, we can confirm our suspicions from the time series plots that TEMP, DEWPOINT, RELHUMIDITY and SEALVLPRS are indeed non-stationary as shown by their respecting p-values being greater than 0.05. Therefore, for these variables we accept the null hypothesis that there is a unit root. The test also shows that our rain obsertion is stationary. However, intuition suggests that whenever there is any heavy rain i.e. a big spike in the time series plot, the mean and variance will change according to the heavy rain. This will not be stationary as the mean and variance will not be constant. To ensure model stability, we will apply differencing to all our variables.

```
# differencing our dataset
diff = pd.DataFrame(index = df.index)
for col in df.columns:
    diff[col] = df[col].diff()

diff = diff.iloc[1:]
```

```
_ = diff[['TEMP','DEWPOINT','RELHUMIDITY','SEALVLPRS','WINDSPEED','TLTCLOUD','VISIBILITY']].plot(subplo
_ = plt.suptitle("Time Series Plot of Differenced Predictors")
plt.show()
```



```
_ = diff[['RAIN2']].plot(figsize=(8,2), title = "Differenced Plot of Rainfall")
plt.show()
```

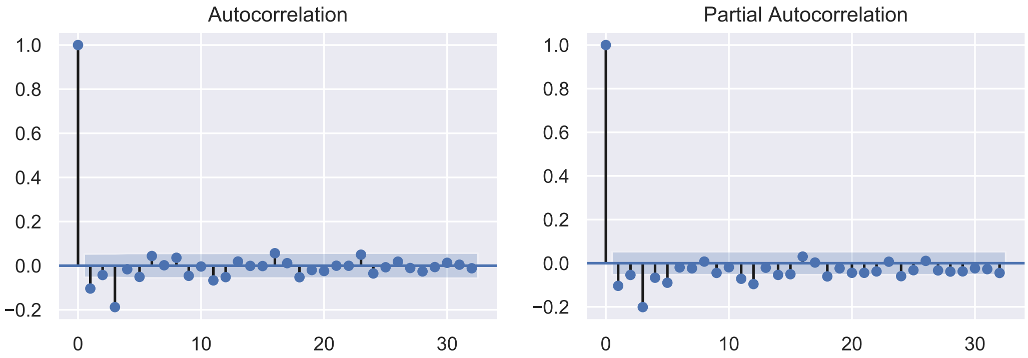


```
for name, values in diff.iteritems():
    result = adfuller(values)
    print('p-value for %s: %4f' % (name,result[1]))
```

```
## p-value for RAIN2: 0.000000
## p-value for TEMP: 0.000000
## p-value for DEWPOINT: 0.000000
## p-value for RELHUMIDITY: 0.000000
## p-value for SEALVLPRS: 0.000000
## p-value for WINDSPEED: 0.000000
## p-value for DIRECTION: 0.000000
## p-value for TLTCLOUD: 0.000000
## p-value for VISIBILITY: 0.000000
## p-value for RAIN: 0.000000
```

The plot for our first order differenced dataset shows that there are no longer any trends in our explanatory variables. Checking the Augmented Dickey-Fuller test on our differenced dataset shows that all our variables are significant at a 5% level (p-value < 0.05), therefore we reject the null hypothesis for all our variables, that is there does not exist a unit root in our variables. Hence, our dataset is now stationary.

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

_ = plt.figure(figsize=(10,3))
_ = plt.subplot(121)
_ = plot_acf(diff.RAIN2, ax=plt.gca())
_ = plt.subplot(122)
_ = plot_pacf(diff.RAIN2, ax=plt.gca())
plt.show()
```



To fit an ARIMA model, we need to determine the order of auto-regressive (AR) and moving average (MA) series by choosing the optimal values for p and q. AR is a process where present values can be estimated by the previoud values of the same time series, while MA is a process where present values can be defined as a linear combination of past errors, where the errors are independent and normally distributed (Salvi 2019).

To achieve this we will plot the auto-correlation function (ACF) and partial auto-correlation function (PACF). The ACF plot provides values of auto-correlation in a given series with its lagged values and the PACF plot finds the correlation of the residuals with the next lag value. Through both these plots we can determine a starting value for p and q in our model.

A pattern to look for when determining the p value is if the ACF trails off after a value and has a hard cut-off in the PACF after said value. This value is taken as the value for p. For the q value, we look at the value where the PACF trails off and has a hard cut-off in the ACF.

Looking at the ACF and PACF plots we can see that both plots trail off after the value of 3. Hence, we can build our initial model with p = 3 and q = 3.

```
# keep 7 days of data for validation
train = diff['2019-01-01':'2019-04-14']
test = diff['2019-04-15':'2019-04-21']
```

In order to fit and test our model, we will split our data into a training and testing set. Our testing set will have 7 days of data for us to do 7 days of forecasting. We have to be careful in selecting our testing set since time series dataset is dependent on time, thus the testing set must not contain observations that precede the training set. For the testing set, we also want observations where heavy rain occurs. A quick check from our dataset helps us to determine our testing set to be from 15 April 2019 to 21 April 2019. Hence, the training set will be everything before 15 April, which is 1 Jan 2019 to 14 April 2019.

```
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(endog = train[['RAIN2']],
           order=(3,3), freq='3H')
res = mod.fit(disp = 0)
print(res.summary())
```

```
##                               ARMA Model Results
## ==============================================================================
## Dep. Variable:                  RAIN2   No. Observations:                  831
## Model:                     ARMA(3, 3)   Log Likelihood                  36.457
## Method:                       css-mle   S.D. of innovations              0.232
## Date:                Sun, 11 Aug 2019   AIC                            -56.915
## Time:                        22:49:37   BIC                            -19.133
## Sample:                    01-01-2019   HQIC                           -42.427
##                          - 04-14-2019
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## const          0.0005      0.006      0.078      0.938      -0.012       0.013
## ar.L1.RAIN2    0.3993      0.125      3.196      0.001       0.154       0.644
## ar.L2.RAIN2    0.1403      0.138      1.017      0.309      -0.130       0.411
## ar.L3.RAIN2   -0.7100      0.146     -4.861      0.000      -0.996      -0.424
## ma.L1.RAIN2   -0.4282      0.146     -2.928      0.004      -0.715      -0.142
## ma.L2.RAIN2   -0.1092      0.150     -0.729      0.466      -0.403       0.184
## ma.L3.RAIN2    0.4776      0.183      2.607      0.009       0.119       0.837
##                                     Roots
## =============================================================================
##                   Real          Imaginary           Modulus         Frequency
## -----------------------------------------------------------------------------
## AR.1            0.7071           -0.8111j            1.0760           -0.1359
## AR.2            0.7071           +0.8111j            1.0760            0.1359
## AR.3           -1.2164           -0.0000j            1.2164           -0.5000
## MA.1            0.8279           -0.8840j            1.2112           -0.1302
## MA.2            0.8279           +0.8840j            1.2112            0.1302
## MA.3           -1.4273           -0.0000j            1.4273           -0.5000
## -----------------------------------------------------------------------------
```

Since our data is already stationary, we will use the ARMA model instead of the ARIMA model. Running an ARMA model without any explanatory variables and p = 3 and q = 3 gives an AIC value of -56.915.

The next step is to select which variables to add to our model. By doing stepwise regression using both forward and backwards selection and finding the minimum AIC value as our evaluation method, we find that a model with DEWPOINT, RELHUMIDITY, SEALVLPRS, WINDSPEED and TLTCLOUD as the explanatory variables gives the smallest AIC value of -142.80. All the explanatory variables are significant at a 5% level as well.

```python
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(endog = train[['RAIN2']], exog = train[['DEWPOINT','RELHUMIDITY','SEALVLPRS',
                                                   'WINDSPEED','TLTCLOUD']],
           order=(3,3), freq='3H')
res = mod.fit(trend="nc", disp = 0)
print(res.summary())
```

```
##                               ARMA Model Results
## ==============================================================================
## Dep. Variable:                  RAIN2   No. Observations:                  831
## Model:                     ARMA(3, 3)   Log Likelihood                  83.400
## Method:                       css-mle   S.D. of innovations              0.219
## Date:                Sun, 11 Aug 2019   AIC                           -142.800
## Time:                        22:49:46   BIC                            -86.129
## Sample:                    01-01-2019   HQIC                          -121.069
##                          - 04-14-2019
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## DEWPOINT       0.0348      0.010      3.586      0.000       0.016       0.054
## RELHUMIDITY    0.0042      0.001      3.777      0.000       0.002       0.006
## SEALVLPRS     -0.0313      0.009     -3.455      0.001      -0.049      -0.014
## WINDSPEED       0.0852      0.025      3.404      0.001       0.036       0.134
## TLTCLOUD        0.0042      0.001      5.520      0.000       0.003       0.006
## ar.L1.RAIN2     0.6217      0.190      3.280      0.001       0.250       0.993
## ar.L2.RAIN2    -0.3028      0.253     -1.197      0.232      -0.798       0.193
## ar.L3.RAIN2     0.1476      0.144      1.026      0.305      -0.134       0.430
## ma.L1.RAIN2    -0.8100      0.178     -4.544      0.000      -1.159      -0.461
## ma.L2.RAIN2     0.2921      0.264      1.107      0.269      -0.225       0.809
## ma.L3.RAIN2    -0.4505      0.162     -2.784      0.005      -0.768      -0.133
##                                     Roots
## =============================================================================
##                   Real          Imaginary           Modulus         Frequency
## -----------------------------------------------------------------------------
## AR.1            1.8010           -0.0000j            1.8010           -0.0000
## AR.2            0.1250           -1.9354j            1.9394           -0.2397
## AR.3            0.1250           +1.9354j            1.9394            0.2397
## MA.1            1.0197           -0.0000j            1.0197           -0.0000
## MA.2           -0.1857           -1.4636j            1.4754           -0.2701
## MA.3           -0.1857           +1.4636j            1.4754            0.2701
## -----------------------------------------------------------------------------
```

Checking the residuals of our model involves fitting our model and comparing the actual rain observation from the training set with our model's predicted values. Since our data was already differenced when we put it into our model, the predicted values are also differenced. Therefore, we have to apply inverse differencing to revert first order differencing done on our dataset.

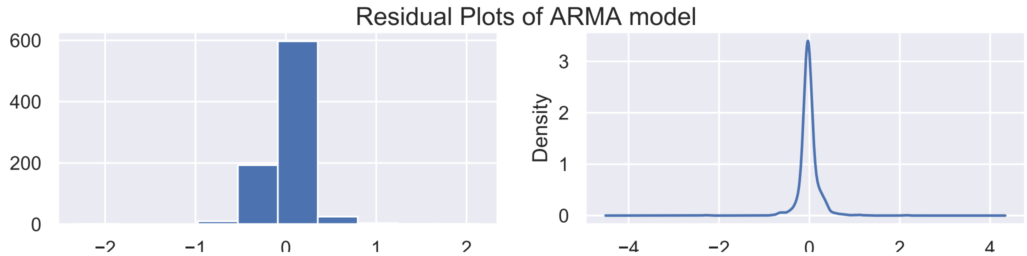
```
# inverse differencing
def inv_diff(df, diff):
    # prepare DataFrame
    inv_df = pd.DataFrame()
    inv_df['actual_rain'] = df
    inv_df['pred_rain'] = np.append(np.nan, diff)

    # add actual value with difference
    shifted = df.shift(1)
    inv_df['inv_pred_rain'] = shifted + inv_df['pred_rain']
    # first row is null, change it to the first value of actual_rain
    inv_df.inv_pred_rain.fillna(df.iloc[0], inplace = True)
    # round up to 2 decimal places
    inv_df['inv_pred_rain'] = inv_df.inv_pred_rain.round(2)

    return inv_df
```

```
pred_rain = res.predict()
actual_rain = df['2019-01-01':'2019-04-14'].iloc[:,0]
inverted_df = inv_diff(actual_rain, res.predict())
inverted_df.head()
```

```
##                      actual_rain  pred_rain  inv_pred_rain
## 2019-01-01 00:00:00          0.0        NaN           0.00
## 2019-01-01 03:00:00          0.0   0.024356           0.02
## 2019-01-01 06:00:00          0.0   0.038574           0.04
## 2019-01-01 09:00:00          0.0   0.039732           0.04
## 2019-01-01 12:00:00          0.0   0.046243           0.05
```

After inverting the values back we can plot our residuals.

```
# errors
residuals = inverted_df['actual_rain'] - inverted_df['inv_pred_rain']
# plot
_ = plt.figure(figsize=(10,2))
_ = plt.subplot(121)
_ = residuals.hist(ax=plt.gca());
_ = plt.subplot(122)
_ = residuals.plot(kind='kde', ax=plt.gca())
_ = plt.suptitle('Residual Plots of ARMA model')
plt.show()
```



The plot shows that the errors are normally distributed with a mean of zero, which is one of the assumptions for an ARMA model.

Grid Search

The next step is to search for the optimal p and q values for our ARMA model. We can achieve this by applying grid search which fits an ARMA model with every possible combination of p and q values, and find the combination that has the lowest RMSE.

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
import warnings

# evaluate an ARIMA model for a given order (p,d,q) and return RMSE
def evaluate_arima_model(df, train, arma_order):
    mod = ARMA(endog = train[['RAIN2']], exog = train[['DEWPOINT','RELHUMIDITY','SEALVLPRS',
                                                       'WINDSPEED','TLTCLOUD']],
           order=arma_order, freq='3H')
    res = mod.fit(trend="nc", disp = 0)
    pred_rain = res.predict()
    actual_rain = df['2019-01-01':'2019-04-14'].iloc[:,0]
    inverted_df = inv_diff(actual_rain, res.predict())
    mse = mean_squared_error(inverted_df.actual_rain, inverted_df.inv_pred_rain)
    rmse = sqrt(mse)
    return rmse

# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(df, train, p_range, q_range):
    best_score, best_cfg = float("inf"), None
    for p in p_range:
        for q in q_range:
            order = (p,q)
            try:
                mse = round(evaluate_arima_model(df, train, order), 3)
                # print('ARMA%s RMSE=%.3f' % (order,mse))
                if mse < best_score:
                    best_score, best_cfg = mse, order
            except:
                continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

# read data
df = pd.read_csv('stations_punjab_hour.csv', index_col = 0)
df.set_index(pd.to_datetime(df.index), inplace = True)

# apply cube root transformation to response
df[['RAIN2']] = np.cbrt(df[['RAIN2']])

# differencing our dataset
diff = pd.DataFrame(index = df.index)
for col in df.columns:
    diff[col] = df[col].diff()
diff = diff.iloc[1:]

# keep last 7 days of data for validation
train = diff['2019-01-01':'2019-04-14']
test = diff['2019-04-15':'2019-04-21']
```

```python
# grid search to find optimal p and q values
p_range = range(0,7)
q_range = range(0,7)
warnings.filterwarnings("ignore")
evaluate_models(df, train, p_range, q_range)
```

```
## Best ARIMA(6, 2) RMSE=0.217
```

Checking all possible combinations of p and q from 0 to 6 shows that there are a few combination that has the least RMSE of 0.217. we will pick p = 6 and q = 2 as our optimal parameters. Hence, our final model is an ARMA model with explanatory variables DEWPOINT, RELHUMIDITY, SEALVLPRS, WINDSPEED, and TLTCLOUD with parameters p = 6, q = 2.

```python
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(endog = train[['RAIN2']], exog = train[['DEWPOINT','RELHUMIDITY','SEALVLPRS',
                                                   'WINDSPEED','TLTCLOUD']],
           order=(6,2), freq='3H')
res = mod.fit(trend="nc", disp = 0)
print(res.summary())
```

```
##                               ARMA Model Results
## ==============================================================================
## Dep. Variable:                  RAIN2   No. Observations:                  831
## Model:                     ARMA(6, 2)   Log Likelihood                  87.211
## Method:                       css-mle   S.D. of innovations              0.218
## Date:                Sun, 11 Aug 2019   AIC                           -146.422
## Time:                        22:55:05   BIC                            -80.305
## Sample:                    01-01-2019   HQIC                          -121.069
##                          - 04-14-2019
## ===============================================================================
##                   coef    std err          z      P>|z|      [0.025      0.975]
## -------------------------------------------------------------------------------
## DEWPOINT        0.0363      0.010      3.622      0.000       0.017       0.056
## RELHUMIDITY     0.0040      0.001      3.543      0.000       0.002       0.006
## SEALVLPRS      -0.0318      0.009     -3.530      0.000      -0.049      -0.014
## WINDSPEED       0.0828      0.025      3.340      0.001       0.034       0.131
## TLTCLOUD        0.0040      0.001      5.298      0.000       0.003       0.006
## ar.L1.RAIN2     0.1373      0.206      0.666      0.506      -0.267       0.541
## ar.L2.RAIN2     0.5501      0.167      3.299      0.001       0.223       0.877
## ar.L3.RAIN2    -0.2277      0.045     -5.024      0.000      -0.316      -0.139
## ar.L4.RAIN2    -0.0932      0.069     -1.357      0.175      -0.228       0.041
## ar.L5.RAIN2     0.0847      0.049      1.731      0.084      -0.011       0.181
## ar.L6.RAIN2     0.1104      0.042      2.609      0.009       0.027       0.193
## ma.L1.RAIN2    -0.3290      0.205     -1.607      0.108      -0.730       0.072
## ma.L2.RAIN2    -0.6461      0.201     -3.210      0.001      -1.041      -0.252
##                                     Roots
## =============================================================================
##                   Real          Imaginary           Modulus         Frequency
## -----------------------------------------------------------------------------
## AR.1            1.2358           -0.0000j            1.2358           -0.0000
## AR.2            0.8716           -1.0532j            1.3671           -0.1400
## AR.3            0.8716           +1.0532j            1.3671            0.1400
## AR.4           -1.2227           -0.0000j            1.2227           -0.5000
## AR.5           -1.2616           -1.2709j            1.7908           -0.3744
## AR.6           -1.2616           +1.2709j            1.7908            0.3744
## MA.1            1.0152           +0.0000j            1.0152            0.0000
## MA.2           -1.5245           +0.0000j            1.5245            0.5000
## -----------------------------------------------------------------------------
```

Out-of-Sample Forecasting

Now that we have decided on a final model, we can forecast 7 days of rainfall in Punjab using the testing dataset.

```
# read data
df = pd.read_csv('stations_punjab_hour.csv', index_col = 0)
df.set_index(pd.to_datetime(df.index), inplace = True)

# apply cube root transformation to response
df[['RAIN2']] = np.cbrt(df[['RAIN2']])

# differencing our dataset
diff = pd.DataFrame(index = df.index)
for col in df.columns:
    diff[col] = df[col].diff()
diff = diff.iloc[1:]

# keep last 7 days of data for validation
train = diff['2019-01-01':'2019-04-14']
test = diff['2019-04-15':'2019-04-21']
```

```
# ARIMA model
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(endog = train[['RAIN2']], exog = train[['DEWPOINT','RELHUMIDITY','SEALVLPRS',
                                                    'WINDSPEED','TLTCLOUD']],
           order=(6,2), freq='3H')
res = mod.fit(trend="nc", disp = 0)

# forecast 7 days of rainfall
# since our data is 3 hourly it is 7 * 8 rows of observation to forecast
pred_rain = res.forecast(steps=56,exog=test[['DEWPOINT','RELHUMIDITY','SEALVLPRS',
                                              'WINDSPEED','TLTCLOUD']])[0]
actual_rain = df['2019-04-14 21:00:00':'2019-04-21 21:00:00'].iloc[:,0]
inverted_df = inv_diff(actual_rain, pred_rain)
inverted_df.head()
```

```
##                        actual_rain  pred_rain  inv_pred_rain
## 2019-04-14 21:00:00       0.000000        NaN           0.00
## 2019-04-15 00:00:00       0.735667  -0.363584          -0.36
## 2019-04-15 03:00:00       0.926883  -0.276248           0.46
## 2019-04-15 06:00:00       1.061016   0.852583           1.78
## 2019-04-15 09:00:00       1.167799   0.477700           1.54
```

The inverse differenced forecasted rain values show that the model has mostly predicted close to 0mm of rainfall. Since heavy rainfall is so scarce, it is hard to feed in good data to forecast heavy rains using our current model. It also seems that our model has predicted some negative values as well, which is isn't ideal.

To check how much heavy rainfall our model has predicted, we will classify the predicted values into a binary class where heavy rain = 1 and not heavy rain = 0. After that, we will check our model accuracy by looking at the proportion of accurate heavy rain our model predicted correctly. In other words, we will look at our model accuracy given it is heavy rain.

```
# classify rain to be 1 if heavy rain and 0 if not
accuracy_df = pd.DataFrame()
accuracy_df['TRUE_RAIN'] = [1 if x >= 1.5 else 0 for x in inverted_df['actual_rain']]
accuracy_df['PRED_RAIN'] = [1 if x >= 1.5 else 0 for x in inverted_df['inv_pred_rain']]
accuracy_df = accuracy_df[accuracy_df.TRUE_RAIN == 1]
accuracy_df
```

```
##     TRUE_RAIN  PRED_RAIN
## 8           1          0
## 9           1          1
## 10          1          1
## 11          1          1
## 12          1          1
## 14          1          0
## 15          1          1
```

```
from sklearn.metrics import accuracy_score
print('Model accuracy for predicting heavy rainfall:',
    accuracy_score(accuracy_df.TRUE_RAIN, accuracy_df.PRED_RAIN, normalize = True))
```

```
## Model accuracy for predicting heavy rainfall: 0.7142857142857143
```

```
print('Number of correctly classified samples:',
    accuracy_score(accuracy_df.TRUE_RAIN, accuracy_df.PRED_RAIN, normalize = False))
```

```
## Number of correctly classified samples: 5
```

Checking our model accuracy given that TRUE_RAIN == 1 gives a 0.71 accuracy for our model, or 5 out of 7 observations. In other words, given that heavy rain has occured, our model predicted heavy rain with a 71% accuracy.

Conclusion

In conclusion, we have managed to build a time series model which is able to decently forecast 7 days of future rainfall. That being said, there are many areas in which the model can be improved upon. Having just a few months worth of data with limited heavy rainfall ended up with a smaller cut-off point for classifying heavy rain in our data. This was necessary to increase the frequency of heavy rain in the data to train our model. This meant that the "heavy rain" that our model predicted is not really heavy rain, but rather moderate rainfall. Having a few years worth of rainfall data with the monsoon rain period would have helped in this regard and our model would also be able to analyze the seasonality of monsoon rain better. Other techniques could have been employed which would have resulted in a better model, however due to time constraints and a limited knowledge in the area of time series analysis, these methods were not tested. For example, applying random forest with a rolling mean method could have been used to compare with the ARIMA model that we built. A walk forward validation method would have provided a more thorough evaluation method instead of a simple train and test split. Better understanding of the ARIMA model and time series forecasting in general would have resulted in a better model and parameter selection. More complicated deep learning models such as Artificial Neural Networks or Gradient Boosting models were not considered in our analysis due to a lack of interpretability of the model but can be considered as the next step forward.

References

Adnan, M, Rehman, N, Ali, S, Mehmood, S, Mir, KA, Khan, AA and Khalid, B 2017, ‘Prediction of summer rainfall in Pakistan from global sea-surface temperature and sea-level pressure’, Weather, vol. 72, no. 3, pp. 76-84.

Aftab, S, Ahmad, M, Hameed, N, Bashir, M, Ali, and Nawaz, Z, 2018, ‘Rainfall Prediction in Lahore City using Data Mining Techniques’, *International Journal of Advanced Computer Science and Applications*, vol. 9, no4, 2018, pp. 254- 260.

Chai, S, Wong, W and Goh, K 2017, ‘Rainfall Classification for Flood Prediction Using Meteorology Data of Kuching, Sarawak, Malaysia: Backpropagation vs Radial Basis Function Neural Network’, *International Journal of Environmental Science and Development*, vol. 8, no. 5, pp. 385-388.

Kamath, R and Kamat, R 2018, ‘Time-series Analysis and Forecasting of Rainfall at Idukki district, Kerala: Machine Learning Approach’, vol. 11 2018, pp 27-33.

Nair, A, Singh, G, and Mohanty, UC 2017, ‘Prediction of Monthly Summer Monsoon Rainfall Using Global Climate Models Through Artificial Neural Network Technique’, *Pure and Applied Geophysics*, vol. 175, no. 2018, pp. 403-419.

Pakistan Today 2018, ‘With the arrival of monsoon, threat of diseases increases’, viewed 20 June 2019, https://www.pakistantoday.com.pk/2018/07/15/with-the-arrival-of-monsoon-threat-of-diseases-increases/.

Salvi J 2019, ‘Significance of ACF and PACF Plots In Time Series Analysis’, viewed 11 August 2019, *Towards Data Science*, https://towardsdatascience.com/significance-of-acf-and-pacf-plots-in-time-series-analysis-2fa11a5d10a8

Appendix

```
# function to clean observation dataset before shifting the rainfall 3 hours back
def clean_obs_before(df):

    # rename columns and remove whitespaces
    df.columns = df.columns.str.strip()
    df.rename(columns={'RAIN 1': 'RAIN1',
                       'RAIN 2': 'RAIN2',
                       'RPeriodi 1': 'RPeriod1',
                       'RPeriod 2': 'RPeriod2',
                       'TEMPERATR': 'TEMP',
                       'RELHUMIDY': 'RELHUMIDITY',
                       'VISIBLITY': 'VISIBILITY',}, inplace = True)


    # replace -9999 with NaN values
    df.replace(-9999.0, np.nan, inplace = True)

    # remove rows where date is invalid
    df.drop(df[(df.YYYY == 0) | (df.MM == 0) | (df.DD == 0)].index, inplace = True)

    # remove rows where all values are null
    df.dropna(axis=0, subset=['TEMP'], inplace = True)

    # convert date to datetime format
    df['TIME'] = df['YYYY'].map(str) + '-' + df['MM'].map(str) + '-' + df['DD'].map(str) + ':' + df['HH
    df['TIME'] = pd.to_datetime(df['TIME'], format = '%Y-%m-%d:%H')
    df.drop(['YYYY','MM','DD','HH','RECDNO'], axis = 1, inplace = True)

    # move time to the first column
    cols = list(df.columns.values)
    cols = cols[-1:] + cols[:-1]
    df = df.loc[:,cols]

    # remove RAIN1 and RPeriod1 (explanation made below)
    df.drop(['RAIN1', 'RPeriod1'], axis = 1, inplace = True)

    return df

# function to shift rain observation back 3 hours
def shift_rain(df):
    # Separating rainfall observation into another dataframe
    rain = df[['TIME','RAIN2','RPeriod2']]
    df.drop(['RAIN2','RPeriod2'], axis=1, inplace=True)

    # create new date range for every 3 hours
    start = rain.TIME.min()
    end = rain.TIME.max()
    date_range = pd.date_range(start=start, end=end, freq='3H')
    date_range = pd.to_datetime(date_range)

    # create new dataframe of just time and rain
    new_df = pd.DataFrame(date_range, columns=['TIME'])
    rain = pd.merge(new_df, rain[['TIME', 'RAIN2']], how='left', on='TIME')

    # shift rainfall values up one row
    rain[['RAIN2']] = rain.RAIN2.shift(-1)

    # merge rain back into original dataframe
    df = pd.merge(rain, df, how='left', on='TIME')

    # impute missing values by linear interpolation
    df = df.interpolate(method='linear')

    # fill remaining missing RAIN2 values with 0
    df.RAIN2.fillna(0, inplace= True)

    df.drop(['STATNO'], axis = 1, inplace = True)

    return df
```