Group Member:
Siming Wu
Aiguo Shao
Yi Zheng

# Melbourne Climate Group Project Report

## BACKGROUND

The project Melbourne Climate is aimed to investigate the long-term climate of Melbourne, in order to generate unique phenomenons, find regularity between features which benefits future climate forecasts. After attaining clean data, different types of models are applied to implement goals. The clean data are resampled into different frequencies such as monthly or daily, also resample by sum or mean, enabling comparison with the climatological data from Bureau of Meteorology. In the end, developing  models to predict rainfall based on previous records is one of the goals of the project.

## Dataset

These five features are for create timestamp:

• Year

• Month

• Date

• Hour

• Minute

These eight features for predicting:

• Dry bulb temperature (degrees C): the standard temperature measurement.

• Dew point temperature (degrees C): a measure of the moisture content of the air.

• Apparent temperature (degrees C): a measure of how hot it feels, which takes into account dry-bulb temperature, humidity and wind-speed.

• Relative humidity (%): the percentage of moisture in the air, relative to the saturation level. • Wind direction (NNW, etc).

• Average wind-speed (km/h).

• Wind-gust (km/h): maximum wind-speed.

• Mean sea level pressure (hPa): the pressure interpolated to mean sea level.

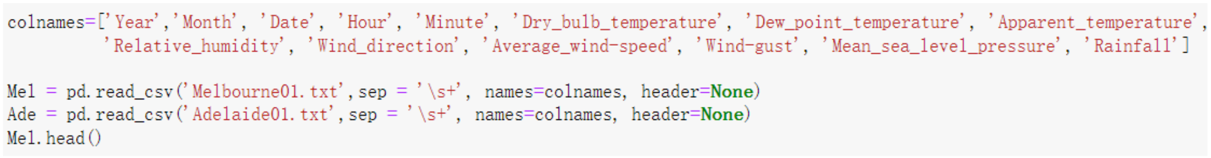
Rainfall is the target value in the model:

• Rainfall (mm): rainfall since 9 am.( includes all forms of precipitation that reach the ground, such as rain, drizzle, hail and snow. )

(To separate the level of Rainfall, we have Light Rain < 2.5 mm, Moderate Rain between 2.5 mm — 7.6 mm, Heavy Rain >7.6 mm)

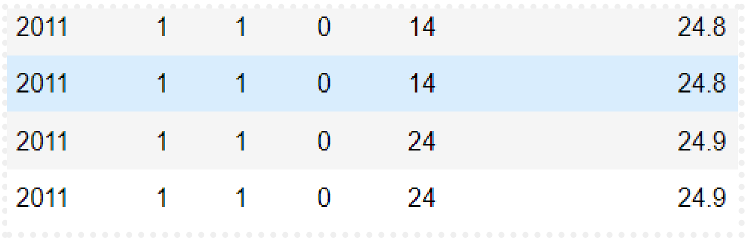
## CLEAN DATA

Read the text file and give proper column names for our features.

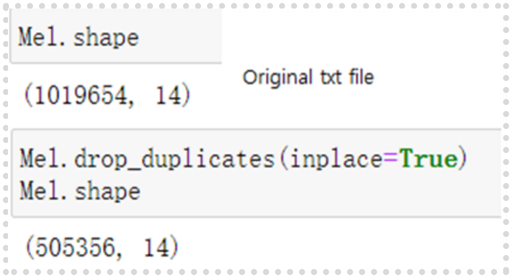


```
colnames=['Year','Month', 'Date', 'Hour', 'Minute', 'Dry_bulb_temperature', 'Dew_point_temperature', 'Apparent_temperature',
          'Relative_humidity', 'Wind_direction', 'Average_wind-speed', 'Wind-gust', 'Mean_sea_level_pressure', 'Rainfall']

Mel = pd.read_csv('Melbourne01.txt',sep = '\s+', names=colnames, header=None)
Ade = pd.read_csv('Adelaide01.txt',sep = '\s+', names=colnames, header=None)
Mel.head()
```

In the dataset, we found that there are many repeated data in our dataset.



| | | | | | |
|---|---|---|---|---|---|
| 2011 | 1 | 1 | 0 | 14 | 24.8 |
| 2011 | 1 | 1 | 0 | 14 | 24.8 |
| 2011 | 1 | 1 | 0 | 24 | 24.9 |
| 2011 | 1 | 1 | 0 | 24 | 24.9 |



```
Mel.shape
```

```
(1019654, 14)
```

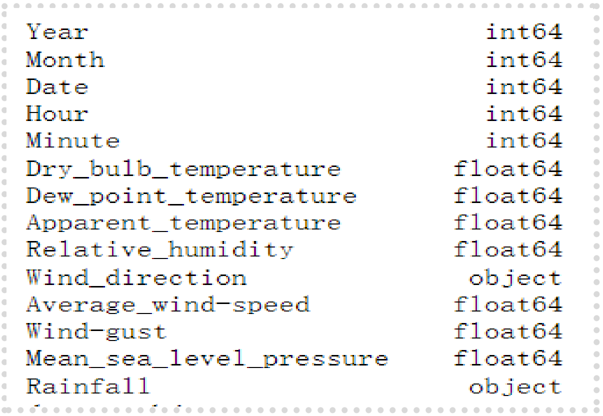
Original txt file

```
Mel.drop_duplicates(inplace=True)
Mel.shape
```

```
(505356, 14)
```

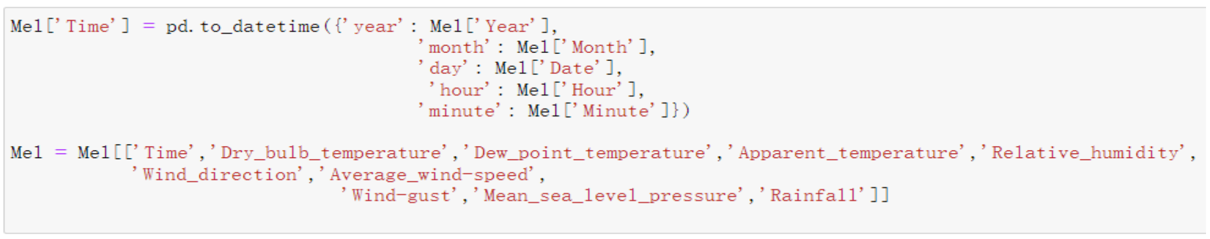
By implementing the correct codes, we drop the duplicates, and only half of the dataset remains.

Then the following step is to check the data type, convert all object data to numeric. And we got two objects in our dataset.

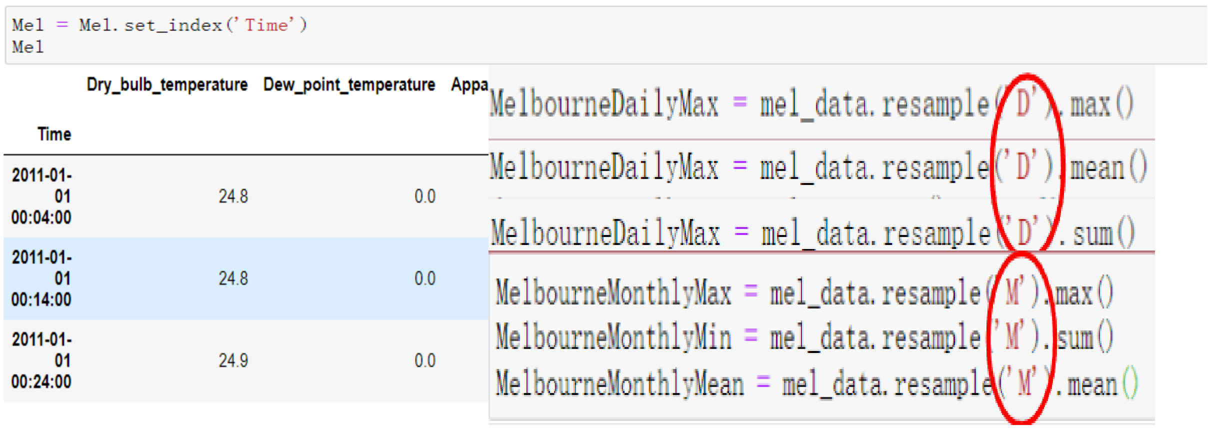


```
Year                          int64
Month                         int64
Date                          int64
Hour                          int64
Minute                        int64
Dry_bulb_temperature        float64
Dew_point_temperature       float64
Apparent_temperature        float64
Relative_humidity           float64
Wind_direction               object
Average_wind-speed          float64
Wind-gust                   float64
Mean_sea_level_pressure     float64
Rainfall                     object
```

Finally, the four features( Year, Month,Date,Hour ) are used to construct a timestamp called Time, which is to put Time as the index.

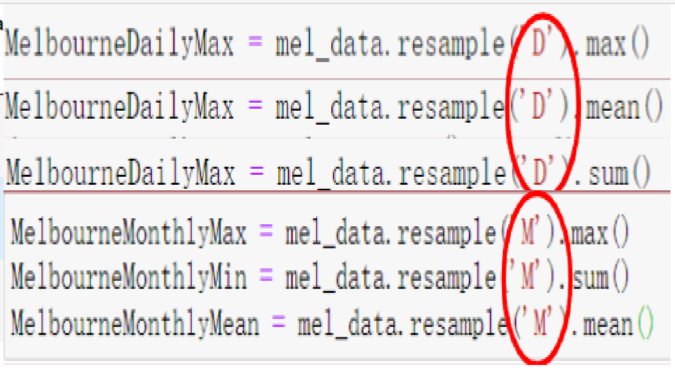


```
Mel['Time'] = pd.to_datetime({'year': Mel['Year'],
                              'month': Mel['Month'],
                              'day': Mel['Date'],
                               'hour': Mel['Hour'],
                              'minute': Mel['Minute']})

Mel = Mel[['Time','Dry_bulb_temperature','Dew_point_temperature','Apparent_temperature','Relative_humidity',
           'Wind_direction','Average_wind-speed',
                              'Wind-gust','Mean_sea_level_pressure','Rainfall']]
```



```
Mel = Mel.set_index('Time')
Mel
```

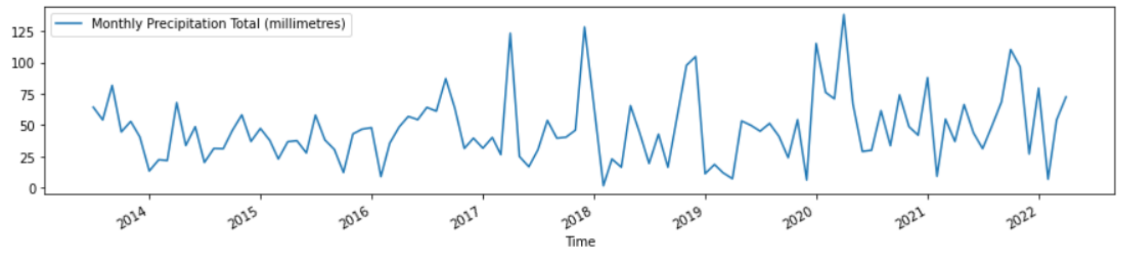
| | Dry_bulb_temperature | Dew_point_temperature | Appa |
|---|---|---|---|
| **Time** | | | |
| **2011-01-01 00:04:00** | 24.8 | 0.0 | |
| **2011-01-01 00:14:00** | 24.8 | 0.0 | |
| **2011-01-01 00:24:00** | 24.9 | 0.0 | |

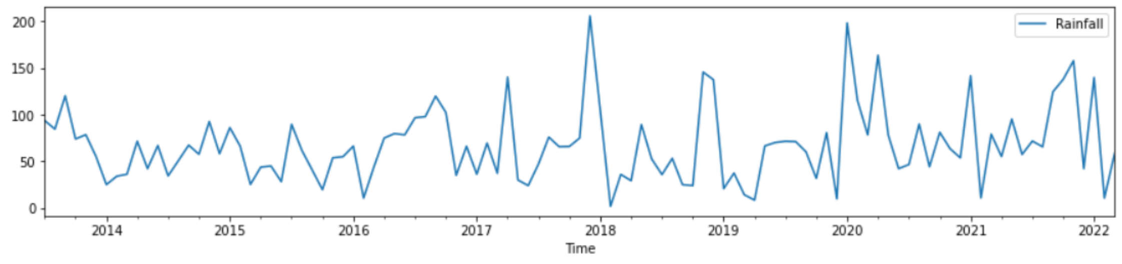


```
MelbourneDailyMax = mel_data.resample('D').max()
MelbourneDailyMax = mel_data.resample('D').mean()
MelbourneDailyMax = mel_data.resample('D').sum()
MelbourneMonthlyMax = mel_data.resample('M').max()
MelbourneMonthlyMin = mel_data.resample('M').sum()
MelbourneMonthlyMean = mel_data.resample('M').mean()
```

After timestamp, we resampled our dataset by daily and Monthly in three different ways, which are maximum, mean and sum.

Compared with the Government's data, using max data is much similar to the government's data, so it's better and safer to use daily max data in proper models.





## ARIMA/ SARIMA

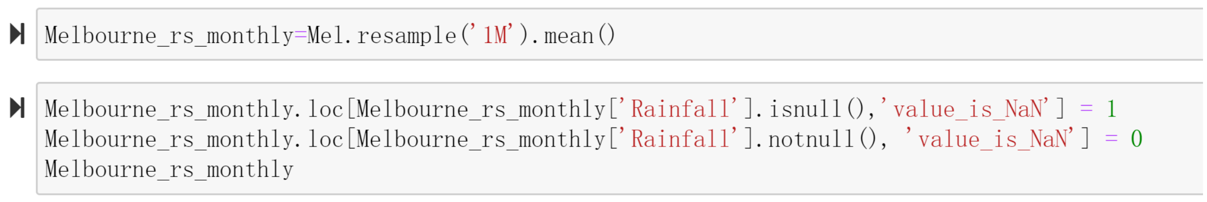
### Introduction of ARIMA

ARIMA(Autoregressive Integrated Moving Average) is a model that fits to time series data either to better understand the data or to predict future points in the series(forecasting).

ARIMA models are applied in some cases where data show evidence of non-stationarity in the sense of mean(but not variance), where an initial differencing step can be applied one or more times to eliminate the non-stationarity of the mean function.  SARIMA is seasonal ARIMA which targets the dataset with obvious seasonal patterns.The dataset is from the Bureau of Meteorology since 2011.The purpose of using SARIMA is for forecasting future trends and maximising its accuracy and minimising the errors. SARIMA is more preferable than ARIMA is that it allows stationarity data(which don’t satisfy either trend or seasonal pattern at the same time), whereas ARIMA only allows for non-stationary data. After data cleaning there exists 490,000 rows of daily data.(Figure 1.1.2). Rainfall is the key feature that is expected to be explored the most. Before implementing the code it is hypothesised that it is supposed to be seasonal and predictable, if not it is necessary to use any climatic phenomenon to explain the irregularity. Rolling forecast enables increasement of accuracy of the predicted model with historical records such that it allows for continuous planning with a constant number of periods.

(Data Quality)

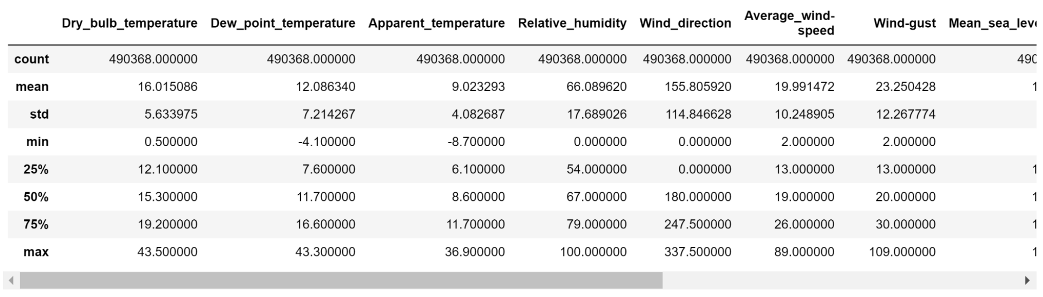
Since the dataset covers from 2011 to 2022, so resample data to monthly is better for observation and remove NaN(Figure 1.1.1), then it can conclude that the dataset is stationary since it doesn’t satisfy either trend or seasonal at the same time.(Figure 1.1.3). We can see that the plot shows seasonality of rainfall over 10 years, also that the scatter of residual evenly distributed on both sides of 0, which demonstrates the resample dataset is well-performed. The second step is to train the model and testify its accuracy with different measurements. The process of splitting train and test by date, the testing dataset is based on previous 9 years from 2011 to 2020.(Figure 1.1.4).Before applying the SARIMA model, we must testify that the dataset Rainfall is stationary. The input of the SARIMA model is the feature rainfall from monthly resample dataset, and we expect that the output demonstrates the output of prediction and its accuracy based on RMSE and MAPE. The p value of the adfuller test is quite low and less than 0.05 which demonstrates it is a stationary function. After this we can apply the dataset to model SARIMA.(Figure 1.1.5)



```
Melbourne_rs_monthly=Mel.resample('1M').mean()
```

```
Melbourne_rs_monthly.loc[Melbourne_rs_monthly['Rainfall'].isnull(),'value_is_NaN'] = 1
Melbourne_rs_monthly.loc[Melbourne_rs_monthly['Rainfall'].notnull(), 'value_is_NaN'] = 0
Melbourne_rs_monthly
```

Figure 1.1.1



| | Dry_bulb_temperature | Dew_point_temperature | Apparent_temperature | Relative_humidity | Wind_direction | Average_wind-speed | Wind-gust | Mean_sea_lev |
|---|---|---|---|---|---|---|---|---|
| **count** | 490368.000000 | 490368.000000 | 490368.000000 | 490368.000000 | 490368.000000 | 490368.000000 | 490368.000000 | 490 |
| **mean** | 16.015086 | 12.086340 | 9.023293 | 66.089620 | 155.805920 | 19.991472 | 23.250428 | 1 |
| **std** | 5.633975 | 7.214267 | 4.082687 | 17.689026 | 114.846628 | 10.248905 | 12.267774 | |
| **min** | 0.500000 | -4.100000 | -8.700000 | 0.000000 | 0.000000 | 2.000000 | 2.000000 | |
| **25%** | 12.100000 | 7.600000 | 6.100000 | 54.000000 | 0.000000 | 13.000000 | 13.000000 | 1 |
| **50%** | 15.300000 | 11.700000 | 8.600000 | 67.000000 | 180.000000 | 19.000000 | 20.000000 | 1 |
| **75%** | 19.200000 | 16.600000 | 11.700000 | 79.000000 | 247.500000 | 26.000000 | 30.000000 | 1 |
| **max** | 43.500000 | 43.300000 | 36.900000 | 100.000000 | 337.500000 | 89.000000 | 109.000000 | 1 |

Figure 1.1.2

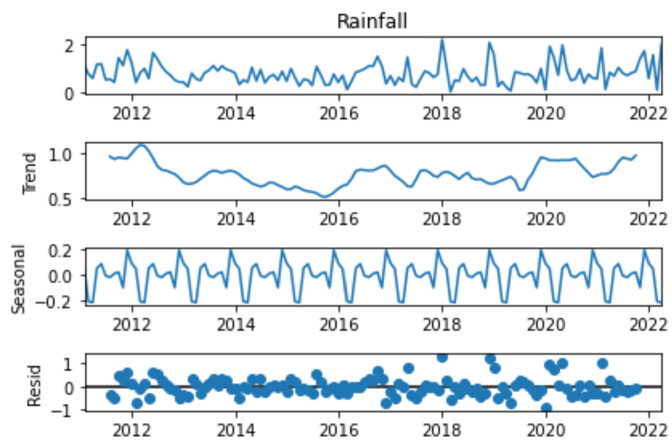
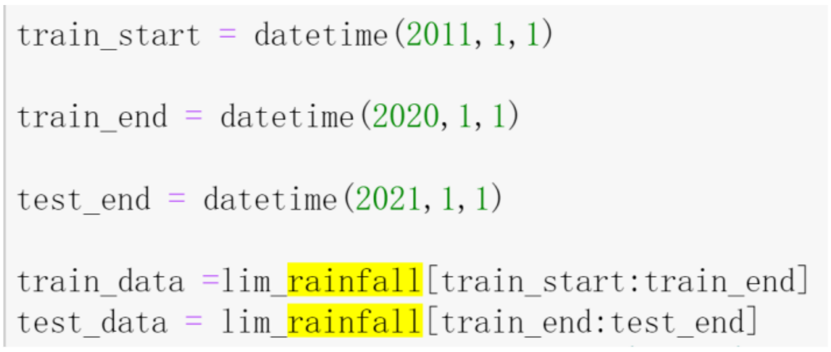
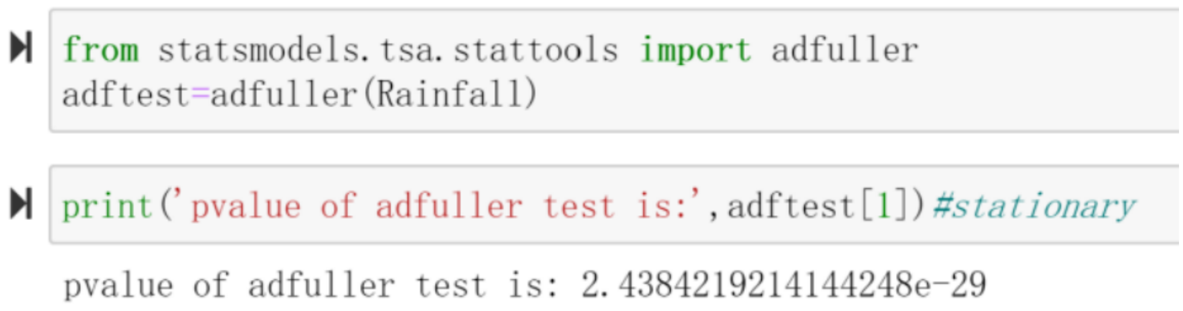


Figure 1.1.3



```
train_start = datetime(2011, 1, 1)

train_end = datetime(2020, 1, 1)

test_end = datetime(2021, 1, 1)

train_data =lim_rainfall[train_start:train_end]
test_data = lim_rainfall[train_end:test_end]
```

Figure 1.1.4



```
from statsmodels.tsa.stattools import adfuller
adftest=adfuller(Rainfall)
```

```
print('pvalue of adfuller test is:',adftest[1])#stationary
```

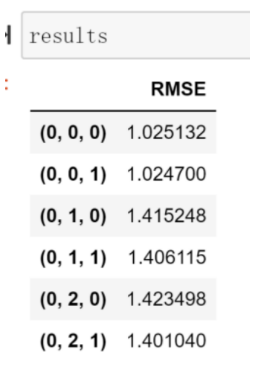
```
pvalue of adfuller test is: 2.4384219214144248e-29
```

Train test split

Figure 1.1.5

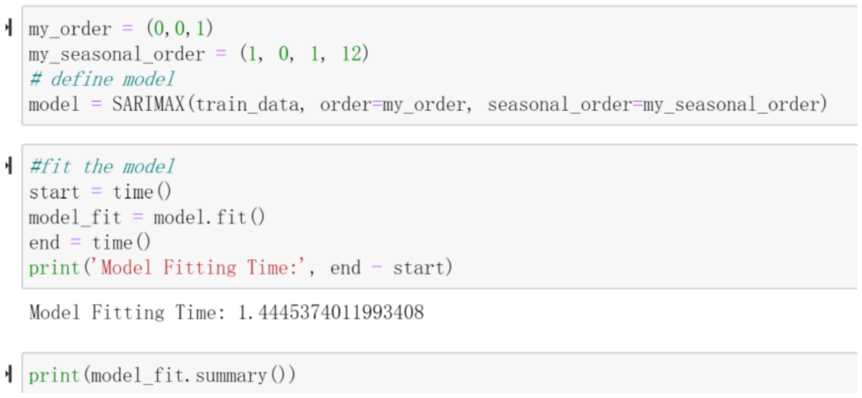
(Model Development)

The parameters P,D,Q, as one of the inputs of the model, the choice of these parameters will interfere with the result of prediction, so that determination of P, D, Q is a very important step of SARIMA. **Also, series Rainfall is also an input. The next step is to identify the parameters P,Q,D which relate to AR(autoregressive), I(integrated) and MA(moving average) respectively. ACF is related to p and PACF is related to q. the ACF should show a sharp drop after a certain q number of lags while PACF should show a geometric or gradual decreasing trend**.For example, ARIMA(1, 0, 12) means that if you are describing some response variable (Y) by combining a 1st order Auto-Regressive model and a 12th order Moving Average model. **There are 3 ways to determine the parameters, but iteration is the best approach to view all RMSE one by one. (0,0,1) provides the lowest RMSE and we apply order=(0,0,1) to the model.(Figure 1.2.1 and Figure 1.2.2) We observe the fit model by summary and plot of residuals and discover the model fit is not very ideal. Also the residuals of the first 5 months are above the 0. (Figure 1.2.3,1.2.4)** Mean Absolute Percent Error and Root Mean Square Error is quite high as well.(Figure 1.2.5)



```
results
```

|  | RMSE |
|---|---|
| **(0, 0, 0)** | 1.025132 |
| **(0, 0, 1)** | 1.024700 |
| **(0, 1, 0)** | 1.415248 |
| **(0, 1, 1)** | 1.406115 |
| **(0, 2, 0)** | 1.423498 |
| **(0, 2, 1)** | 1.401040 |



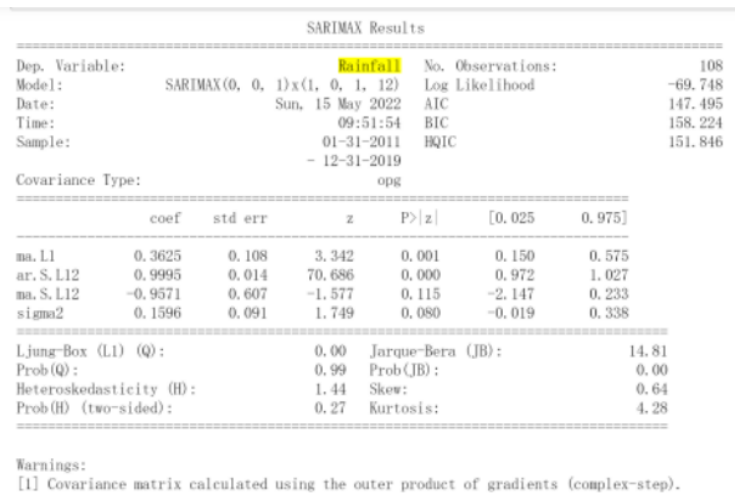
```
my_order = (0,0,1)
my_seasonal_order = (1, 0, 1, 12)
# define model
model = SARIMAX(train_data, order=my_order, seasonal_order=my_seasonal_order)
```

```
#fit the model
start = time()
model_fit = model.fit()
end = time()
print('Model Fitting Time:', end - start)
```

```
Model Fitting Time: 1.4445374011993408
```

```
print(model_fit.summary())
```

**Figure 1.2.1** **Figure 1.2.2**



```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                         Rainfall   No. Observations:                  108
Model:             SARIMAX(0, 0, 1)x(1, 0, 1, 12)   Log Likelihood                 -69.748
Date:                          Sun, 15 May 2022   AIC                            147.495
Time:                                  09:51:54   BIC                            158.224
Sample:                              01-31-2011   HQIC                           151.846
                                   - 12-31-2019
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          0.3625      0.108      3.342      0.001       0.150       0.575
ar.S.L12       0.9995      0.014     70.686      0.000       0.972       1.027
ma.S.L12      -0.9571      0.607     -1.577      0.115      -2.147       0.233
sigma2         0.1596      0.091      1.749      0.080      -0.019       0.338
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                14.81
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               1.44   Skew:                             0.64
Prob(H) (two-sided):                  0.27   Kurtosis:                         4.28
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Figure 1.2.3

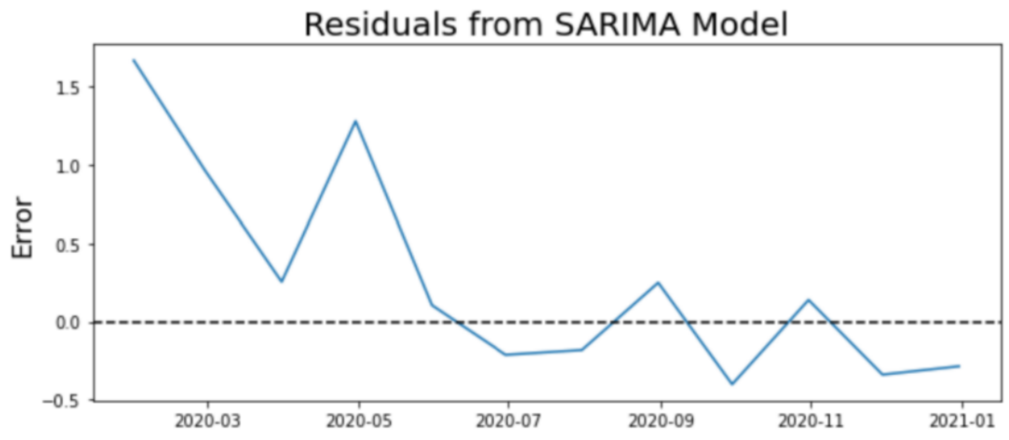
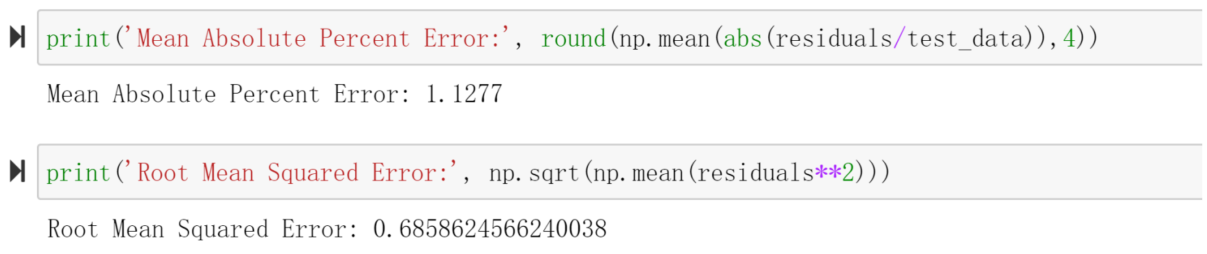


Figure 1.2.4

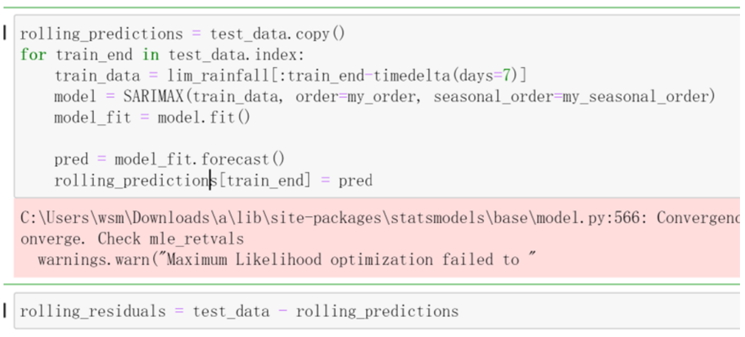


```
print('Mean Absolute Percent Error:', round(np.mean(abs(residuals/test_data)),4))

Mean Absolute Percent Error: 1.1277

print('Root Mean Squared Error:', np.sqrt(np.mean(residuals**2)))

Root Mean Squared Error: 0.6858624566240038
```

Figure 1.2.5

(Model Development)

To enhance this, rolling forecast is a good way to enhance its accuracy. Rolling forecasts allow for continuous planning with a constant number of periods. For example, if the forecast period lasts for 12 months as each month ends another month will be added, then it means the model is always forecasting 12 months in the future.(Figure 1.3.1) About accuracy, rolling forecasts allow you to make quick tweaks along the way rather than letting mistakes count up and only giving one shot to make those changes. The residual Predictions are shown below. The rolling forecast has better accuracy than the previous one since its MAPE and RMSE is less than the previous one. (Figure 1.3.2, 1.3.3)

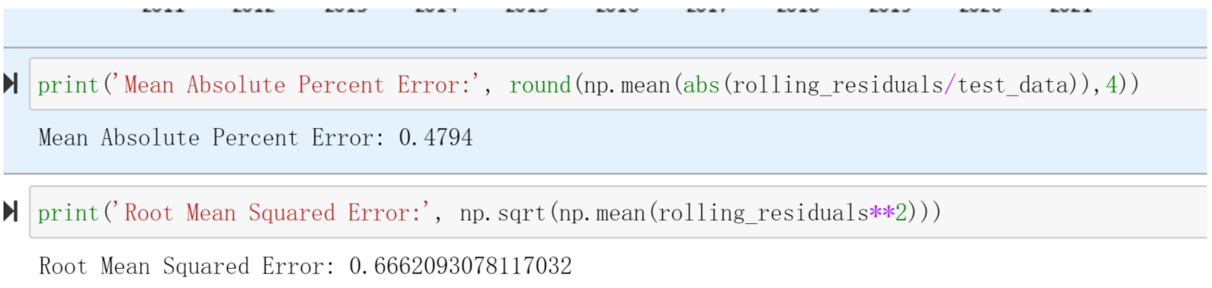


```
rolling_predictions = test_data.copy()
for train_end in test_data.index:
    train_data = lim_rainfall[:train_end-timedelta(days=7)]
    model = SARIMAX(train_data, order=my_order, seasonal_order=my_seasonal_order)
    model_fit = model.fit()

    pred = model_fit.forecast()
    rolling_predictions[train_end] = pred
```

```
C:\Users\wsm\Downloads\a\lib\site-packages\statsmodels\base\model.py:566: Convergenc
onverge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
rolling_residuals = test_data - rolling_predictions
```

Figure 1.3.1



```
print('Mean Absolute Percent Error:', round(np.mean(abs(rolling_residuals/test_data)),4))
```

```
Mean Absolute Percent Error: 0.4794
```

```
print('Root Mean Squared Error:', np.sqrt(np.mean(rolling_residuals**2)))
```

```
Root Mean Squared Error: 0.6662093078117032
```

Figure 1.3.2

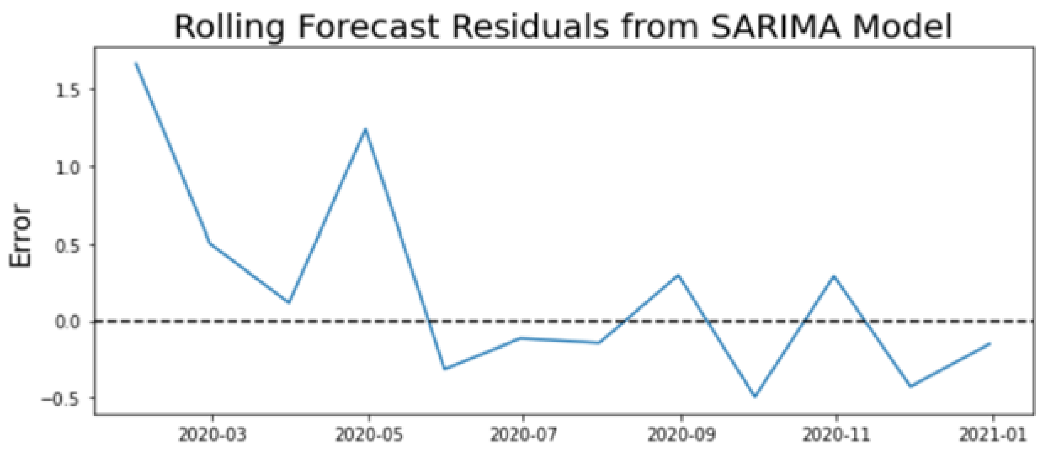
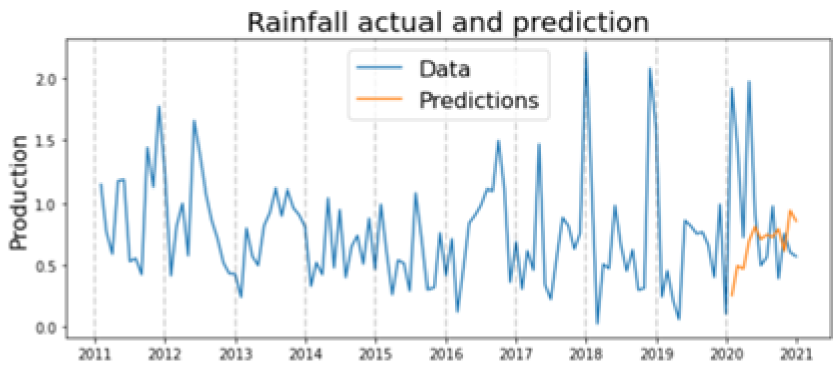


Figure 1.3.3

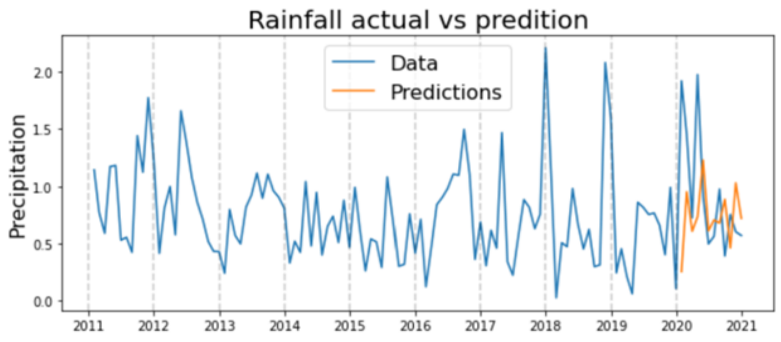
(Result)

Figure 1.4.1 and Figure 1.4.2 are the comparison prediction with the difference of rolling forecast. Figure 1.4.3 illustrates the plots of monthly rainfall from Bureau Of Meteorology and resample dataset, the resample data is lower than the official forecast all the way. In original data, if resample by sum within daily rainfall average, because the frequency of data is too high  the cumulative monthly rainfall is very high. The resample data is derived from dataset *30 to gain monthly cumulative rainfall. Because time 30 to daily rainfall dataset and not every month has exact 30 days, there exist some biases.



Before rolling forecast

Figure 1.4.1



After rolling forecast

Figure 1.4.2

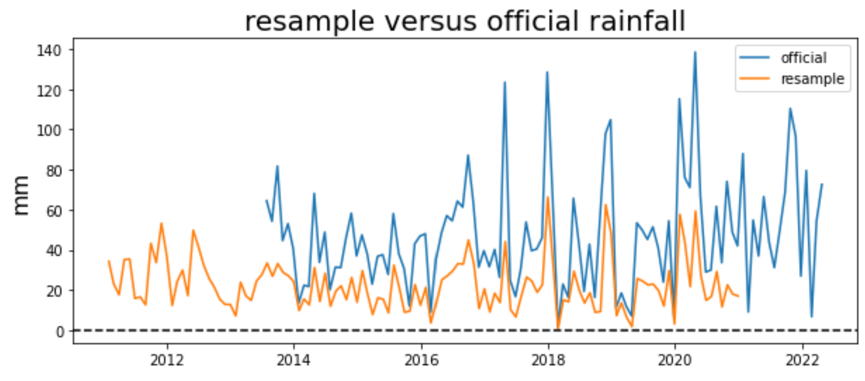


Figure 1.4.3

Conclusion

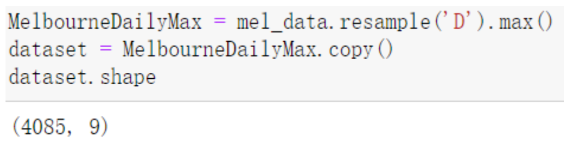
1. Rolling forecast enables SARIMA models to attain a higher accuracy.
2. Comparing resample dataset and official dataset allows checking for the validation of resample dataset.
3. SARIMA is just a type of model with one single feature, it processes the previous training set to predict the testing set, it couldn't explain correlation between multiple features at one time.

## RANDOM FOREST

- **What is Random Forest**

Random forest is an integration algorithm that integrates the modelling results of all models by building multiple models on the data. Random forest is the representative model of bagging. The core idea of the bagging method is to build several independent evaluators. All its base estimators are decision trees. The forest composed of classification trees is called random forest classifier, and the forest integrated by regression trees is called random forest regressor.

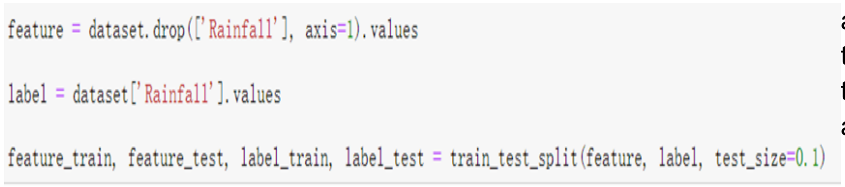
- **Random Forest Regression**



```
MelbourneDailyMax = mel_data.resample('D').max()
dataset = MelbourneDailyMax.copy()
dataset.shape

(4085, 9)
```

The Bureau of Meteorology data is based on daily max, we focus primarily on the daily max dataset, which we get as the daily max as shown and its shape is (4085,9).

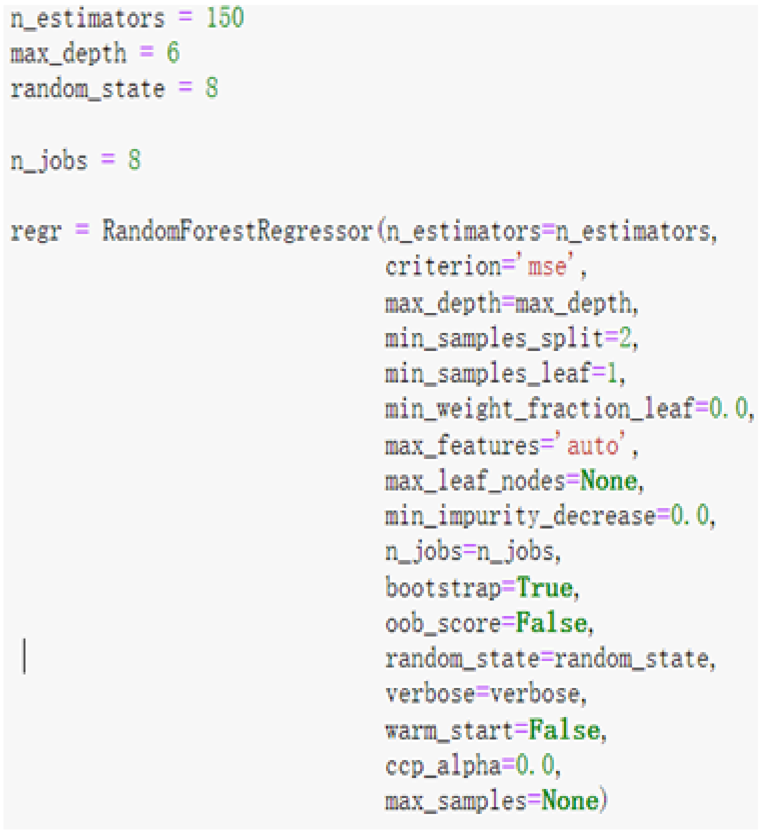
Our goal is to analyse what factors in the daily max dataset are related to rainfall, and whether this regression can predict the value of future rainfall. Get the desired data set, we should start to extract feature variables and the target variable – Rainfall by returning all values in the daily max dataset as a list. Use train_ test_ split () function for random division. This is because the data set is calculated on a daily basis, but it is not very temporal, so there is no need to divide the data set according to the time series.



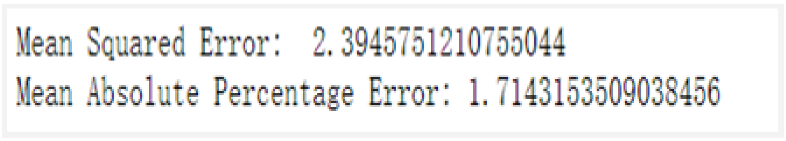
```
feature = dataset.drop(['Rainfall'], axis=1).values

label = dataset['Rainfall'].values

feature_train, feature_test, label_train, label_test = train_test_split(feature, label, test_size=0.1)
```

- Important Parameter Setting

n_ estimators is the number of trees in the forest. The larger the n_ estimators, the better the effect of the model in normal However, for this set of data sets, after several modifications of n_ estimators, their results are verified and compared. I found that when n_ The regression model has the best effect when estimators =150. The maximum depth of a tree in a random forest is defined as the longest path between the root node and the leaf node. Due to this regression model having a large number of samples and features, I limit the growth depth of each tree in the random forest to 6. As for random_ state equal 8 is to make the results consistent and eliminate the interference of random components. The parameter n_jobs means the number of parallel jobs and is set as 8.



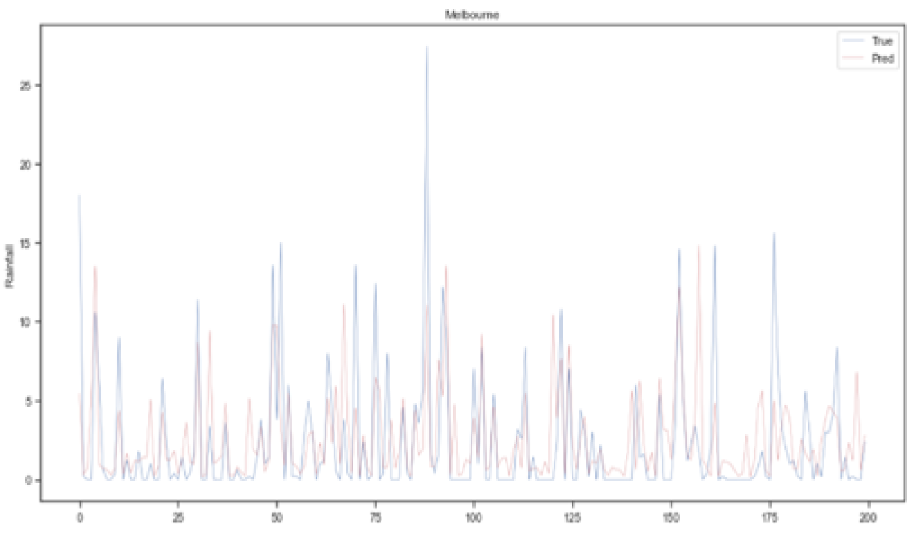
```
n_estimators = 150
max_depth = 6
random_state = 8

n_jobs = 8

regr = RandomForestRegressor(n_estimators=n_estimators,
                             criterion='mse',
                             max_depth=max_depth,
                             min_samples_split=2,
                             min_samples_leaf=1,
                             min_weight_fraction_leaf=0.0,
                             max_features='auto',
                             max_leaf_nodes=None,
                             min_impurity_decrease=0.0,
                             n_jobs=n_jobs,
                             bootstrap=True,
                             oob_score=False,
                             random_state=random_state,
                             verbose=verbose,
                             warm_start=False,
                             ccp_alpha=0.0,
                             max_samples=None)
```



```
Mean Squared Error:  2.3945751210755044
Mean Absolute Percentage Error: 1.7143153509038456
```

After basic parameter setting and fitting data, the accuracy of this model should be checked. Here, I select MSE and MAPE as accuracy indicators to check. This statistical parameter 2.4 is the expected value of the square of the error of the corresponding point between the predicted data and the original data. And MAPE is 1.71, it means that the predicted result deviates from the true result by an average of 1.71

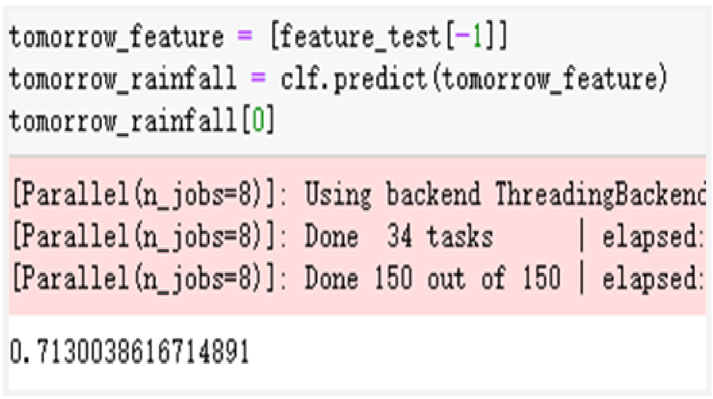
- Comparison Figure



In this expected and actual comparison chart, we can see that when the rainfall is below nearly 12.5mm, the predicted value and the actual value are highly coincident. And there are still some sharp parts, because the maximum daily rainfall in the daily max dataset is greater than 12.5mm which is only 186 days. And that is the reason why the non-overlapping part of the results of this comparison chart.

- Predict Tomorrow Maximum Rainfall

After the actual and predicted comparison charts are obtained, we can start to test the rainfall the next day. Because using the random forest regression, we can get a more specific value as shown.
After the successful establishment of this random forest regression model, the maximum rainfall of any day can be predicted according to the corresponding feature value. In other words, because the equation already exists, as long as you input the value of X, you can get the value of Y.



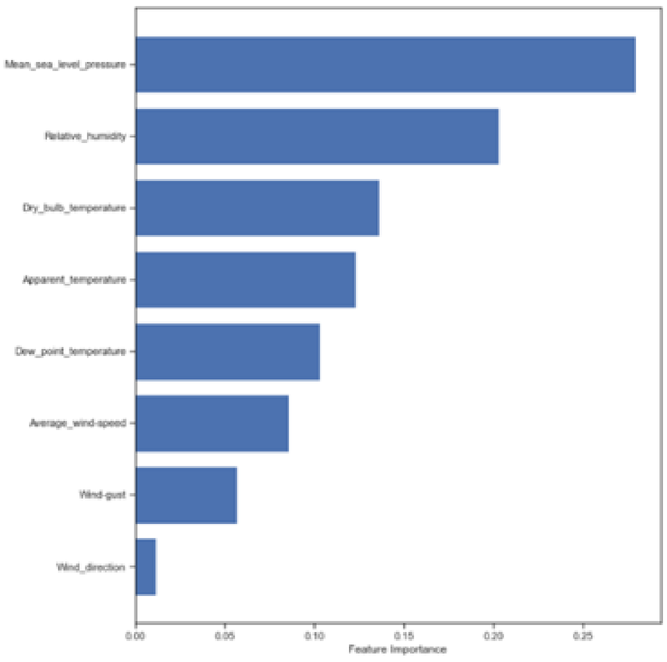
```
tomorrow_feature = [feature_test[-1]]
tomorrow_rainfall = clf.predict(tomorrow_feature)
tomorrow_rainfall[0]
```

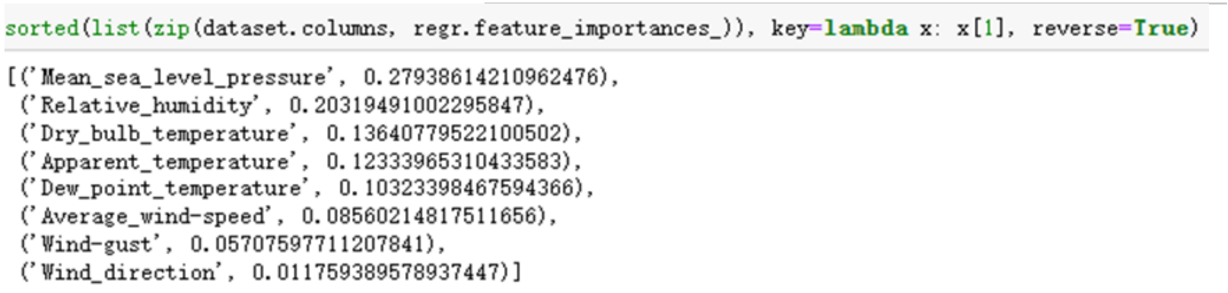
```
[Parallel(n_jobs=8)]: Using backend ThreadingBacken
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:
[Parallel(n_jobs=8)]: Done 150 out of 150 | elapsed:
```

```
0.7130038616714891
```

- Analyse Feature Importance of Feature Variables

After the regression image is drawn, we should check the feature importance ranking of this group of data. Therefore, a chart is drawn, which is arranged in descending order according to the importance of the features, and then the specific value of the proportion of each feature is listed in detail. According to the figures above, mean_sea_level_pressure, relative_humidity, dry_bulb_ temperature and approximate_ temperature have a great impact on rainfall.



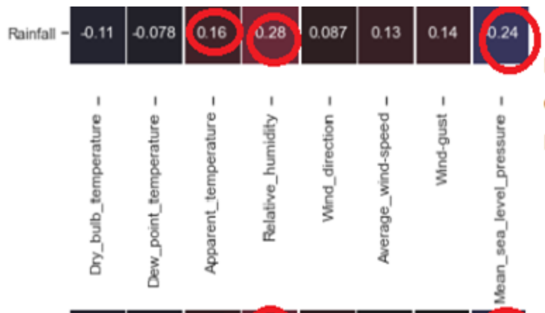


```
sorted(list(zip(dataset.columns, regr.feature_importances_)), key=lambda x: x[1], reverse=True)
```

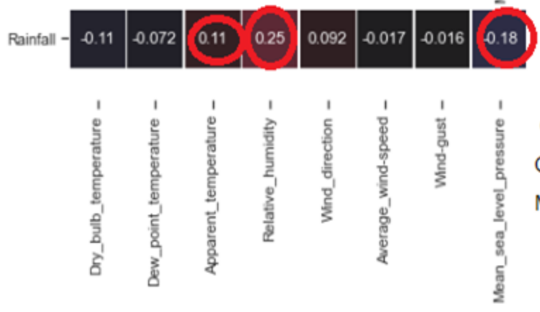
```
[('Mean_sea_level_pressure', 0.27938614210962476),
 ('Relative_humidity', 0.20319491002295847),
 ('Dry_bulb_temperature', 0.13640779522100502),
 ('Apparent_temperature', 0.12333965310433583),
 ('Dew_point_temperature', 0.10323398467594366),
 ('Average_wind-speed', 0.08560214817511656),
 ('Wind-gust', 0.05707597711207841),
 ('Wind_direction', 0.011759389578937447)]
```

- <u>Checking and Comparing the Correlation Tables</u>

In order to testify our conclusion of the influencing factors, we can review and compare the numerical changes of the original dataset and current dataset by their correlation tables.



Daily Max Correlation Matrix

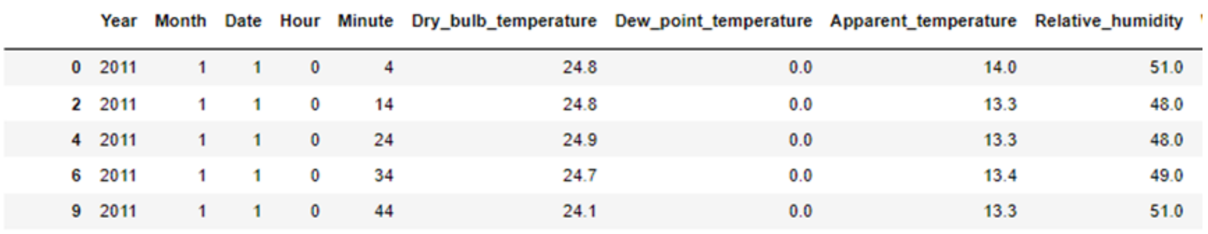


Original Correlation Matrix

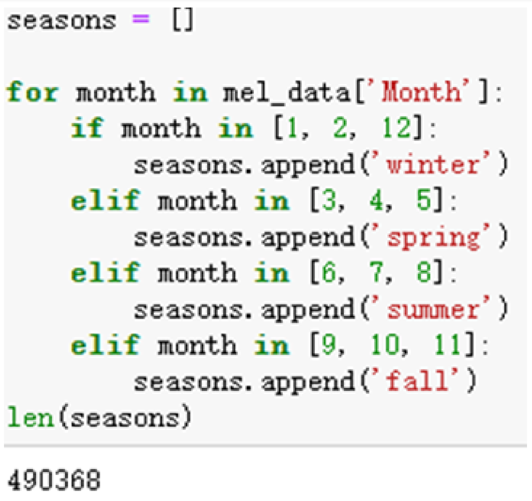
Except for dry_bulb_ temperature, the correlation between these three features (mean_sea_level_pressure, relative_humidity, approximate_ temperature) and rainfall has been significantly improved.

- <u>Seasonally Dataset</u>

After that, I did another clean data for the initial dataset without adding a timestamp.



| | Year | Month | Date | Hour | Minute | Dry_bulb_temperature | Dew_point_temperature | Apparent_temperature | Relative_humidity |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011 | 1 | 1 | 0 | 4 | 24.8 | 0.0 | 14.0 | 51.0 |
| 2 | 2011 | 1 | 1 | 0 | 14 | 24.8 | 0.0 | 13.3 | 48.0 |
| 4 | 2011 | 1 | 1 | 0 | 24 | 24.9 | 0.0 | 13.3 | 48.0 |
| 6 | 2011 | 1 | 1 | 0 | 34 | 24.7 | 0.0 | 13.4 | 49.0 |
| 9 | 2011 | 1 | 1 | 0 | 44 | 24.1 | 0.0 | 13.3 | 51.0 |



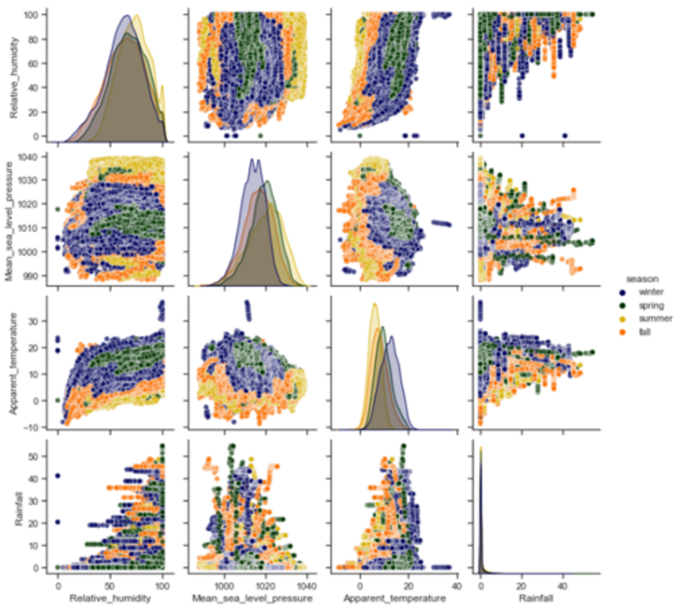
```
seasons = []

for month in mel_data['Month']:
    if month in [1, 2, 12]:
        seasons.append('winter')
    elif month in [3, 4, 5]:
        seasons.append('spring')
    elif month in [6, 7, 8]:
        seasons.append('summer')
    elif month in [9, 10, 11]:
        seasons.append('fall')
len(seasons)
```

```
490368
```

This is to ensure the existence of the Month, perform regression and fitting data according to the previous settings for random forest regression. Then, divide the dataset into seasonally.

Interestingly, the closeness of these three features and rainfall can also be reflected through images in the seasonally dataset.

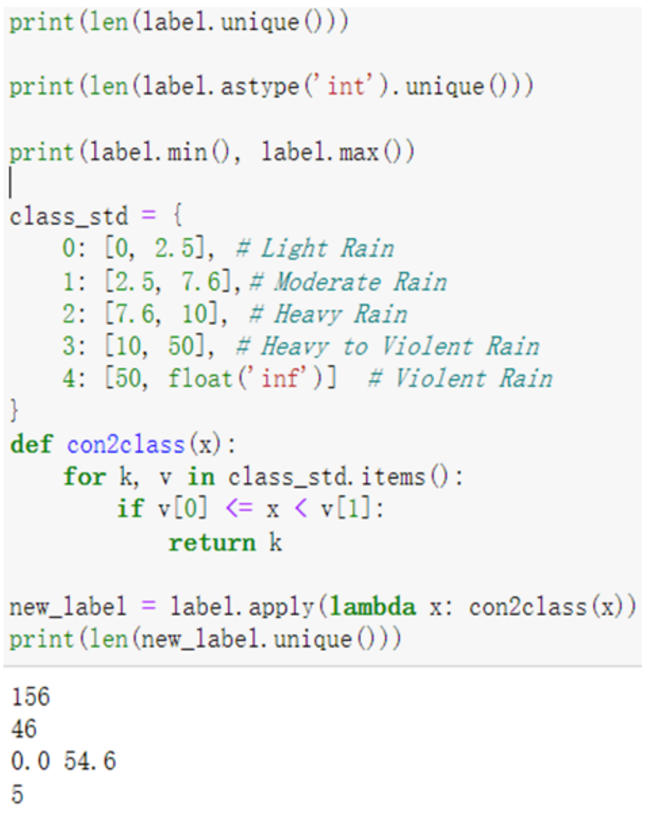
The seasonally output results show that both x-axis and y-axis are Relative_humidity, Mean_sea_level_pressure, Apparent_temperature, Rainfall. Points of different colours represent different seasons. On the main diagonal, both the x-axis and y-axis use the same characteristics to represent their numerical distribution in different seasons. In other locations, scatter plots are used to represent the relationship between the two characteristics, such as relative in the lower left corner Relative_humidity and Rainfall show a strong correlation.



- <u>Conclusion of Random Forest Regression</u>

The above parts have achieved the two objectives required. It can analyse the factors (mean_sea_level_pressure, relative_humidity, approximate_ temperature) related to rainfall of the dataset through Random Forest regression, and the Random Forest regression can predict the exact value of future rainfall by certain feature value.

- **Random Forest Classification**



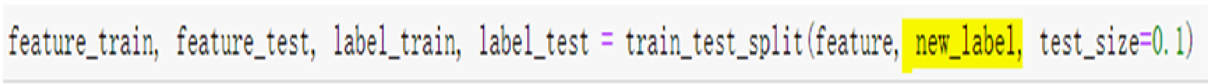
```
print(len(label.unique()))

print(len(label.astype('int').unique()))

print(label.min(), label.max())

class_std = {
    0: [0, 2.5], # Light Rain
    1: [2.5, 7.6], # Moderate Rain
    2: [7.6, 10], # Heavy Rain
    3: [10, 50], # Heavy to Violent Rain
    4: [50, float('inf')]  # Violent Rain
}
def con2class(x):
    for k, v in class_std.items():
        if v[0] <= x < v[1]:
            return k

new_label = label.apply(lambda x: con2class(x))
print(len(new_label.unique()))
```

```
156
46
0.0 54.6
5
```

The creation of the random forest classifier in this dataset is similar to the creation of the random forest regressor, but there are still some differences, which need more details.

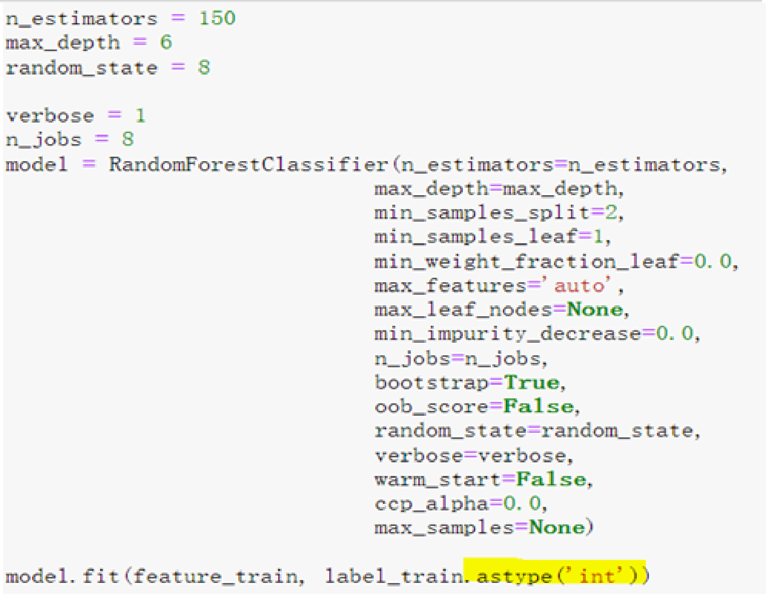
After using the train_ test_ split () function for random division, the label Rainfall of this daily max needs to be discretized.

Check the number of unique values of the original label continuous values, and 156 when viewing the output After converting it to int type, there are 46. In other words, it is converted to int as the label. This model has 46 categories, so there are too many and too miscellaneous categories, which is not conducive to subsequent data analysis. Therefore, the maximum and minimum values of the data are checked. According to the daily rainfall standard, the label is classified into five categories. The numbers 0, 1, 2, 3 and 4 represent light rain, moderate rain, heavy rain, heavy rain to violent rain and violent rain. And the value between each label will join in the certain label by def con2class(x) defined. That is, discretization.

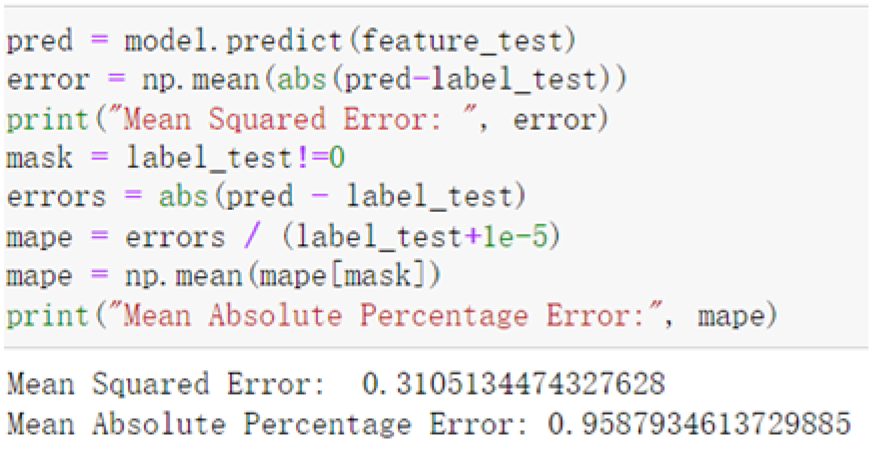


```
feature_train, feature_test, label_train, label_test = train_test_split(feature, new_label, test_size=0.1)
```

Then, do train_ test_ split () function again to replace the label with the new_label. Set the classifier with the same settings as regression. Set the classifier with the same settings as regression, but when fitting data, I need to set the data type of label_train to be converted to integers, because the classifier cannot get an accurate value like regression, and it can only get the yes or no such result as the final answer.



```
n_estimators = 150
max_depth = 6
random_state = 8

verbose = 1
n_jobs = 8
model = RandomForestClassifier(n_estimators=n_estimators,
                               max_depth=max_depth,
                               min_samples_split=2,
                               min_samples_leaf=1,
                               min_weight_fraction_leaf=0.0,
                               max_features='auto',
                               max_leaf_nodes=None,
                               min_impurity_decrease=0.0,
                               n_jobs=n_jobs,
                               bootstrap=True,
                               oob_score=False,
                               random_state=random_state,
                               verbose=verbose,
                               warm_start=False,
                               ccp_alpha=0.0,
                               max_samples=None)

model.fit(feature_train, label_train.astype('int'))
```

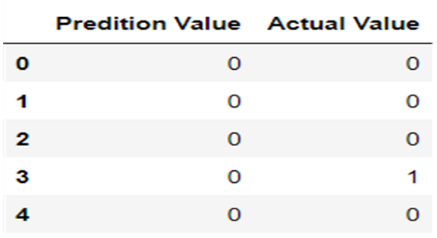


```
pred = model.predict(feature_test)
error = np.mean(abs(pred-label_test))
print("Mean Squared Error: ", error)
mask = label_test!=0
errors = abs(pred - label_test)
mape = errors / (label_test+1e-5)
mape = np.mean(mape[mask])
print("Mean Absolute Percentage Error:", mape)
```

```
Mean Squared Error:  0.3105134474327628
Mean Absolute Percentage Error: 0.9587934613729885
```

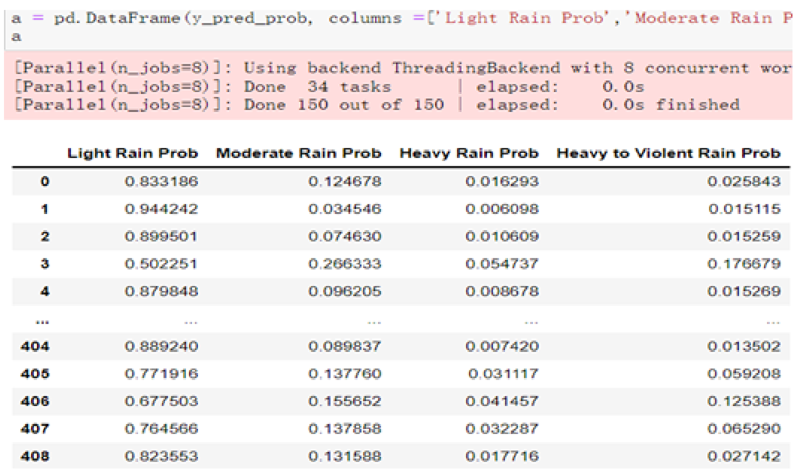
Therefore, using MSE and MAPE as accuracy indicators to check. This Low expectation of error squared 0.31, and MAPE is 0.96, it means that the predicted result deviates from the true result by an average of 0.96, which this classification model has high accuracy.

- Forecast the next day's rainfall



| | Predition Value | Actual Value |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 0 |
| **2** | 0 | 0 |
| **3** | 0 | 1 |
| **4** | 0 | 0 |

The next step is the prediction of rainfall conditions. Before that, we have been divided into different labels and represented here by numbers instead of names. The figure on the right is the precondition dataframe.head(), and the



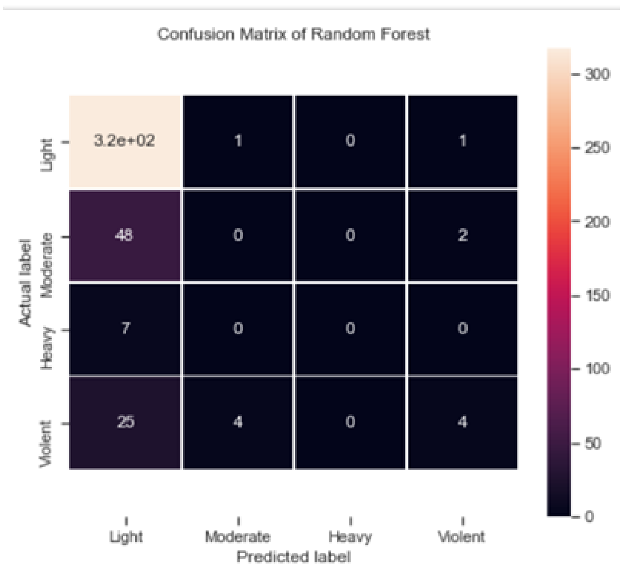
```
a = pd.DataFrame(y_pred_prob, columns =['Light Rain Prob','Moderate Rain P
a
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent wor
[Parallel(n_jobs=8)]: Done  34 tasks       | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 150 out of 150 | elapsed:    0.0s finished
```

| | Light Rain Prob | Moderate Rain Prob | Heavy Rain Prob | Heavy to Violent Rain Prob |
|---|---|---|---|---|
| **0** | 0.833186 | 0.124678 | 0.016293 | 0.025843 |
| **1** | 0.944242 | 0.034546 | 0.006098 | 0.015115 |
| **2** | 0.899501 | 0.074630 | 0.010609 | 0.015259 |
| **3** | 0.502251 | 0.266333 | 0.054737 | 0.176679 |
| **4** | 0.879848 | 0.096205 | 0.008678 | 0.015269 |
| **...** | ... | ... | ... | ... |
| **404** | 0.889240 | 0.089837 | 0.007420 | 0.013502 |
| **405** | 0.771916 | 0.137760 | 0.031117 | 0.059208 |
| **406** | 0.677503 | 0.155652 | 0.041457 | 0.125388 |
| **407** | 0.764566 | 0.137858 | 0.032287 | 0.065290 |
| **408** | 0.823553 | 0.131588 | 0.017716 | 0.027142 |

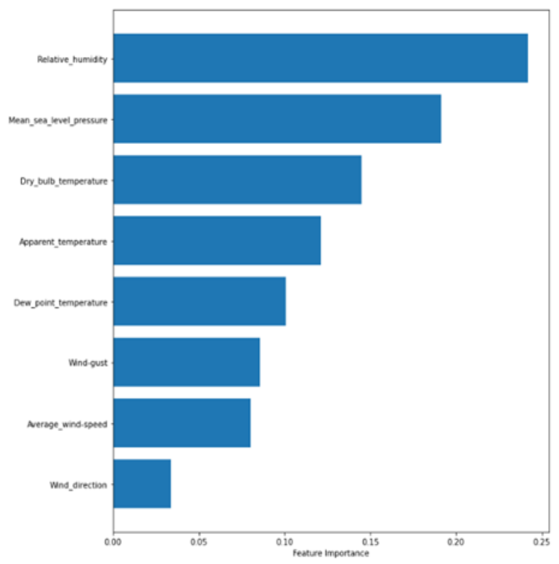
following figure shows the occurrence probability of each label displayed after the label is attached with their name.

The diagonal value of the confusion matrix can be seen to be on the high side, indicating that many predictions are correct. It's a reflection of high probability to predict the correct result.

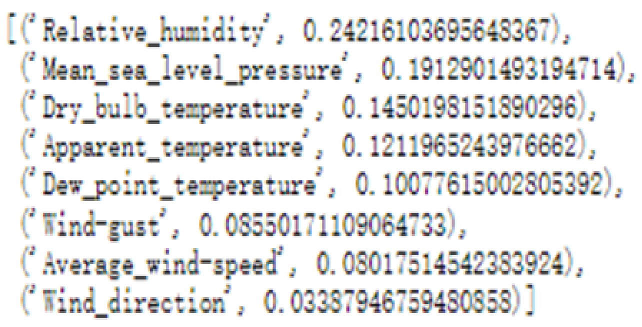


| | Light | Moderate | Heavy | Violent |
|---|---|---|---|---|
| Light | 3.2e+02 | 1 | 0 | 1 |
| Moderate | 48 | 0 | 0 | 2 |
| Heavy | 7 | 0 | 0 | 0 |
| Violent | 25 | 4 | 0 | 4 |

- <u>Analyse Feature Importance of Feature Variables</u>



Subsequently, we should naturally go to the step of checking the importance of features based on this daily max random forest classifier.



```
[('Relative_humidity', 0.24216103695648367),
 ('Mean_sea_level_pressure', 0.19129014931947l4),
 ('Dry_bulb_temperature', 0.145019815l890296),
 ('Apparent_temperature', 0.1211965243976662),
 ('Dew_point_temperature', 0.10077615002805392),
 ('Wind-gust', 0.08550171109064733),
 ('Average_wind-speed', 0.08017514542383924),
 ('Wind_direction', 0.03387946759480858)]
```

It is obvious from the figures that the top 4 importance features are the same as the random forest regression top 4 importance features, but the position of mean_ sea_ level_ pressure and the position of relative_humidity has been replaced.

- <u>Conclusion of Random Forest Classification</u>

The above Random Forest Classification parts have achieved our 2 goals that one is analyse the factors (mean_sea_level_pressure, relative_humidity, approximate_ temperature) with high correlation with Rainfall based on this dataset through Random Forest Classification, and the

Random Forest regression can predict the type of future rainfall by certain feature value, which is light raining, moderate raining, heavy raining and such so on.

- Random Forest Summary

Whether it is a regressor or a classifier, we can finally get the prediction result of the next day's rainfall. As long as the value of the feature on the day you want to predict is given, you can get the prediction of rainfall, the regression will return an exact value, and the classification can get the type of predicted rainfall.

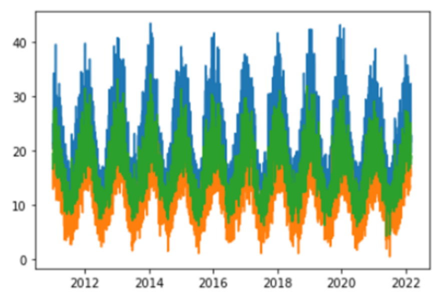
Compare the feature importance obtained by these two methods by looking at this dataset, and the dataset seasonally as a validation of the regression method. Finally, the top 2 features with the highest correlation with rainfall are mean_ sea_ level_ pressure and relative_humidity.

## LOGISTICS REGRESSION

For the modelling, we show the logistic regression classifier model to predict the climate in the future. And we decide to use the daily max data in our model. To visualise the max daily data by plotting.

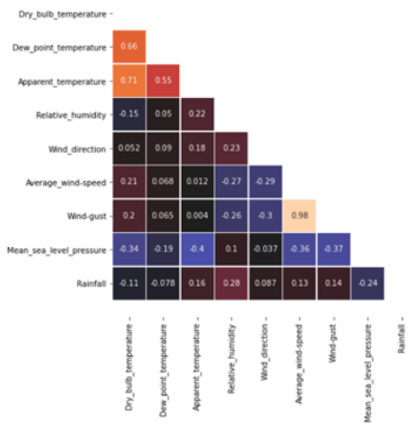
```
import matplotlib.pyplot as plt
plt.plot(MelbourneDailyMax.Dry_bulb_temperature)
plt.plot(MelbourneDailyMin.Dry_bulb_temperature)
plt.plot(MelbourneDailyMean.Dry_bulb_temperature)
```

```
[<matplotlib.lines.Line2D at 0x1fa29f2d130>]
```



The blue line is the max daily data, by using this data, our prediction will have a higher probability of rain. And here is the heat map displays the correlations for different variables on Rainfall, which means the importance in predicting rainfall

.



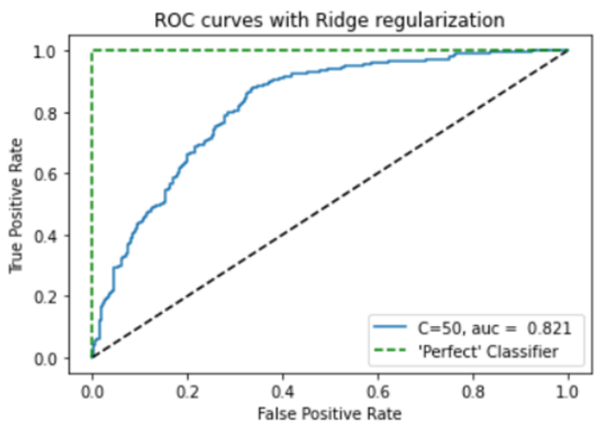
In logistic regression classifier model we need to have a binary data, so we need to create four binary data which are different levels of rainfall, they are ‘NoRain’ , ‘LightRain’, ‘ModerateRain’ and ‘HeavyRain’, which demonstrate is it raining today, if rain, return 1, if no, return 0. To have a new binary column, we need to compute the code ‘ *MelbourneDailyMax1['RainTom'] = (MelbourneDailyMax1.shift(1)['Rainfall'] > 0).astype(int)’* . So far, all the preparation is ready for our logistic model.

## Split Data

The first step to create a model is to set train data and test data. We have features other than 'Rainfall', 'NoRain' , 'LightRain', 'ModerateRain', 'HeavyRain' and ' RainTom' to predict, because they are actually one column of data split into 3 data.

```
XX, Y = MelbourneDailyMax1.drop(['Rainfall','RainTom','NoRain','LightRain','ModerateRain','HeavyRain'],axis=1),
X_train,X_test,y_train,y_test=train_test_split(XX,Y,train_size=0.8,random_state=0)
```

## Create Models



Then to create a model of No Rain.

The image above shows a 0.821 auc score, and 0.821 is greater than 0.5, so it means this model is useful. Also, our accuracy score is 0.733, which is a pretty high score, because it means the number of correct scores divided by the number of samples is 73.3%.

## Checking for Underfitting and Overfitting

```
Train Data Score: 0.7452541334966319
Test Data Score: 0.7331701346389229
```

The next step is to Check for Underfitting and Overfitting. The accuracy Score of training and testing data is comparable and almost equal. So, there is no question of underfitting and overfitting. And the model is generalising well for new unseen data.

## Cross-validation to improve the Model

Finally, the last step is to check whether model performance can be

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(logreg1, X_train, y_train, cv = 5, scoring='accuracy')
print('Cross-validation scores:{}'.format(scores))
print('Average cross-validation score: {}'.format(scores.mean()))
```

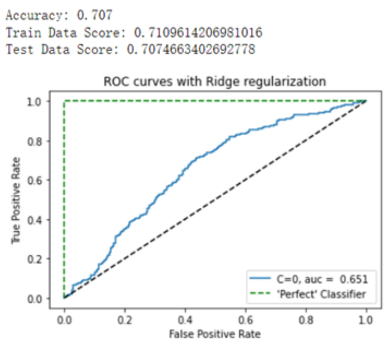
```
Cross-validation scores:[0.74006116 0.7381317  0.75803982 0.75191424 0.7136294 ]
Average cross-validation score: 0.740355264575167
```

improved using Cross-Validation Score.

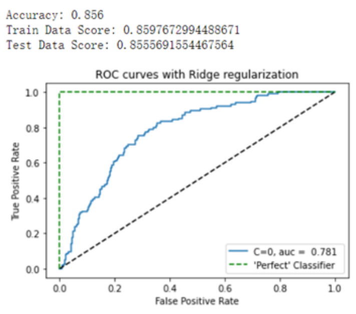
After doing cross validation, we found that the mean accuracy score of cross-validation is almost the same as the original model accuracy score which is 0.733. So, the accuracy of the model may not be improved using Cross-validation.

## Other Predictions

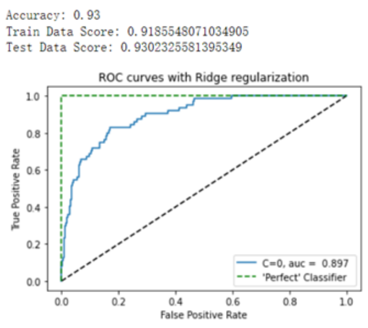
And we repeated the same steps for other models which predict Light Rain, Moderate Rain and Heavy Rain in order. They are our final model.



```
Accuracy: 0.707
Train Data Score: 0.7109614206981016
Test Data Score: 0.7074663402692778
```



```
Accuracy: 0.856
Train Data Score: 0.8597672994488671
Test Data Score: 0.8555691554467564
```



```
Accuracy: 0.93
Train Data Score: 0.9185548071034905
Test Data Score: 0.9302325581395349
```

| Light Rain model | Moderate Rain Model | Heavy Rain Model |
|---|---|---|
| Accuracy: 0.707 | Accuracy : 0.856 | Accuracy: 0.93 |
| Train Data Score: 0.7109 | Train Data Score: 0.8597 | Train Data Score: 0.9185 |
| Test Data Score: 0.7074 | Test Data Score: 0.8555 | Test Data Score: 0.9302 |

## Conclusion for Logistic Regression

- The average accuracy score for the four logistic Regression models is 0.831. So, these models can be using in predicting,
- The models do not have Underfitting or Overfitting issues. This means the models generalise well for unseen data.
- The mean accuracy score of cross-validation is similar to the original model's accuracy score**.** Therefore, we cannot use Cross-validation to improve our accuracy of the model.