Machine Learning Engineer Nanodegree

Capstone Proposal
Shankar Cherla
21st May 2019
Capstone Project
24th May 2019

Predicting if it rains tomorrow in Albury, Australia.

I. Definition

Project Overview

Weather Forecasting is a branch of earth science. Traditionally, it is done through physical simulations of the atmosphere. For example, weather balloons are used for weather forecasting. Its current state is sampled, and the future state is calculated using fluid dynamics and thermodynamics equations. Weather forecasting through physical simulations demands understanding of complex physical processes. However, this physical model is subject to disturbances and uncertainties in the measurements.

Thus, machine learning represents a viable alternative in the weather forecast. It is comparatively robust to disturbances, does not require a complete understanding of physical processes and the predictions are trustworthy as the statistical model developed is completely data driven.

Source:

- The weather forecast dataset is taken from Kaggle.
- The kernel which I used for reference is also from Kaggle.
- This binary classification paper is used for reference

Problem Statement

My intention of this project is to predict if it rains tomorrow in the weather station Albury, Australia utilizing ML. To serve the same purpose I will utilize supervised learning for developing a model. As our target variable 'RainTomorrow' is simply a binary class i.e., it contains either Yes or no, I opt to apply classification algorithms. Initially I will preprocess the data which include dropping irrelevant features, removing null values, removing outliers and I will create models by applying classification algorithms and choose the model with greatest performance. I will try to improve the performance of the chosen model by tuning parameters, try to fit it on reduced data and calculate respective scores in each of these scenarios. In this way after all the steps I will choose the best model suitable with best performance and later this model can be used to predict if it rains or not for new/unseen data.

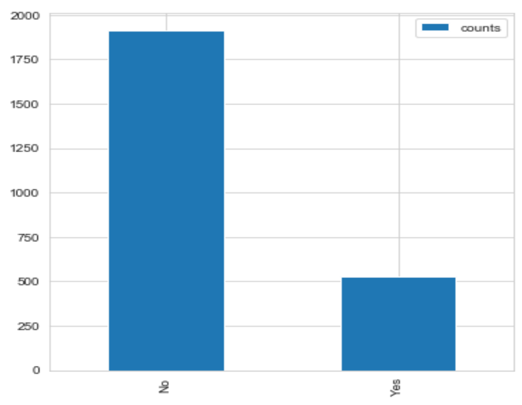
Evaluation Metrics

For Classification problems there are many types of performance evaluation metrics namely Accuracy, F-score etc. Our data has 1913 No and 527 Yes class labels for the predictor variable 'RainTomorrow'. This shows our data is skewed. For classification problems that are skewed in their classification distributions like in our case, Accuracy is not dependable measure for performance evaluation and hence precision, recall are very useful. These two metrics can be combined to get the F-beta score, which is weighted average of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible score. In particular, when beta = 0.5, more emphasis is placed on precision. This is called the F-0.5 score or F-score for simplicity.

The general formula of F-score for positive real β is:

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

The following is a plot of counts of individual classes of predictor variable i.e., 'RainTomorrow'. The plot clearly shows that our data is skewed.



Source

There are many articles giving detailed explanation for choosing the right performance evaluation metric. I am linking the article which I referred to.

II. Analysis

Data Exploration Datasets and Inputs

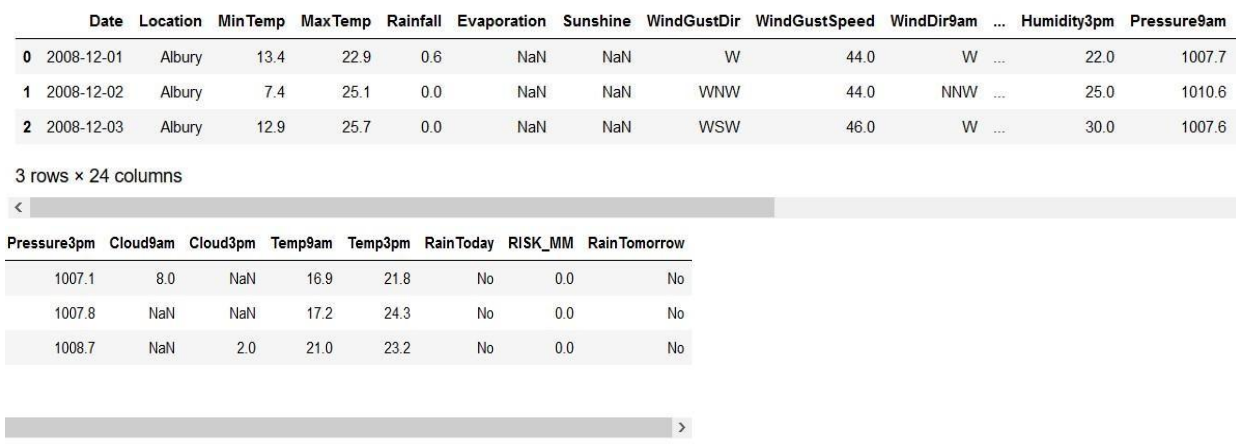
This dataset contains daily weather observations from numerous Australian weather stations among which I considered Albury weather station and the initial shape of the dataset is (3011, 24). After removing all the null values both column wise and row wise the final shape of dataset changes and is (2440,17). Our data has 1913 No and 527 Yes class labels for the predictor variable 'RainTomorrow'. This shows our data is skewed. To reduce the dimension of data from (3011, 24) to (2440,17) various preprocessing steps like removing null values, dropping irrelevant columns, removing outliers and other preprocessing steps are implemented which are clearly discussed below and also in Methodology section in this report. The daily observations are available at URL Copyright
Commonwealth of Australia 2010, Bureau of Meteorology. The weather forecast dataset is taken from Kaggle.

Input information:

1. 'Date': The date of observation
2. 'Location': The common name of the location of the weather station
3. 'MinTemp': The minimum temperature in degrees celsius
4. 'MaxTemp': The maximum temperature in degrees celsius

5. ‘Rainfall’: The amount of rainfall recorded for the day in mm
6. ‘Evaporation’:The so-called Class A pan evaporation (mm) in the 24 hours to 9am
7. ‘Sunshine’: The number of hours of bright sunshine in the day.
8. ‘WindGustDir’: The direction of the strongest wind gust in the 24 hours to midnight.
9. ‘WindGustSpeed’: The speed (km/h) of the strongest wind gust in the 24 hours to midnight.
10. ‘WindDir9am’: Direction of the wind at 9am.
11. ‘WindDir3pm’: Direction of the wind at 3pm.
12. ‘WindSpeed9am’: Wind speed (km/hr) averaged over 10 minutes prior to 9am WindSpeed3pm.
13. ‘Wind speed (km/hr) averaged over 10 minutes prior to 3pm
14. ‘Humidity9am: Humidity (percent) at 9am
15. ‘Humidity3pm’: Humidity (percent) at 3pm
16. ‘Pressure9am’: Atmospheric pressure (hpa) reduced to mean sea level at 9am
17. ‘Pressure3pm’: Atmospheric pressure (hpa) reduced to mean sea level at 3pm
18. ‘Cloud9am’: Fraction of sky obscured by cloud at 9am. This is measured in "oktas", which are a unit of eigths. It records how many eigths of the sky are obscured by cloud. A 0 measure indicates completely clear sky whilst an 8 indicates that it is completely overcast.
19. ‘Cloud3pm’: Fraction of sky obscured by cloud (in "oktas": eighths) at 3pm.
20. ‘Temp9am’: Temperature (degrees C) at 9am.
21. ‘Temp3pm’: Temperature (degrees C) at 3pm.
22. ‘RainToday’: Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0.
23. ‘RISK_MM’: The amount of next day rain in mm. Used to create response variable.
24. ‘RainTomorrow’: The target variable. Will it rain tomorrow?

We can observe top 3 rows of our dataframe below. We can clearly observe that it has 24 columns out of which some are numerical and some are categorical.

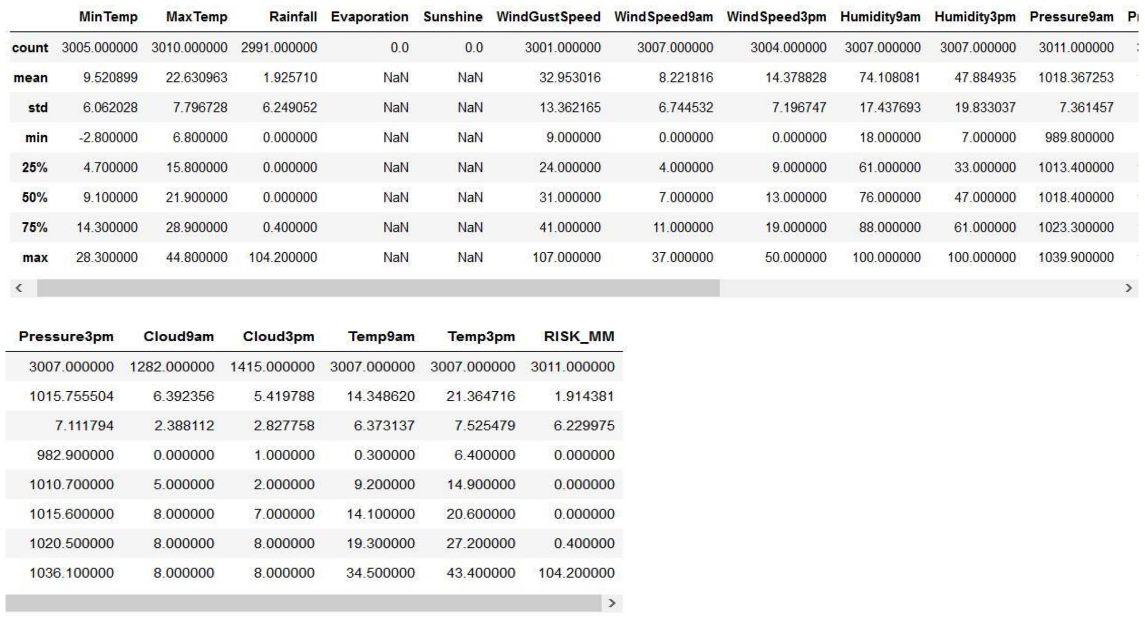


	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	1007.7
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	1010.6
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	1007.6

3 rows × 24 columns

Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
1007.1	8.0	NaN	16.9	21.8	No	0.0	No
1007.8	NaN	NaN	17.2	24.3	No	0.0	No
1008.7	NaN	2.0	21.0	23.2	No	0.0	No

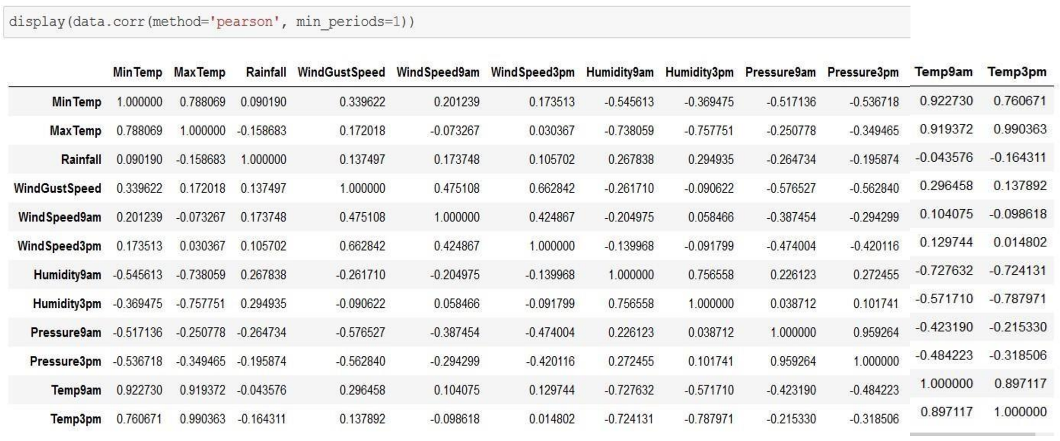
The basic statistical description for all the numerical data of our dataset is given below. The basic statistical descriptions for each column include count of instances, mean, standard deviation, minimum value, maximum value and quartile ranges. This can be known by using describe() function of Pandas. Pandas is quite popular in Data Science and other fields which is an open source tool widely used for data analysis and visualization for python programming language.



	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am
count	3005.000000	3010.000000	2991.000000	0.0	0.0	3001.000000	3007.000000	3004.000000	3007.000000	3007.000000	3011.000000
mean	9.520899	22.630963	1.925710	NaN	NaN	32.953016	8.221816	14.378828	74.108081	47.884935	1018.367253
std	6.062028	7.796728	6.249052	NaN	NaN	13.362165	6.744532	7.196747	17.437693	19.833037	7.361457
min	-2.800000	6.800000	0.000000	NaN	NaN	9.000000	0.000000	0.000000	18.000000	7.000000	989.800000
25%	4.700000	15.800000	0.000000	NaN	NaN	24.000000	4.000000	9.000000	61.000000	33.000000	1013.400000
50%	9.100000	21.900000	0.000000	NaN	NaN	31.000000	7.000000	13.000000	76.000000	47.000000	1018.400000
75%	14.300000	28.900000	0.400000	NaN	NaN	41.000000	11.000000	19.000000	88.000000	61.000000	1023.300000
max	28.300000	44.800000	104.200000	NaN	NaN	107.000000	37.000000	50.000000	100.000000	100.000000	1039.900000

Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RISK_MM
3007.000000	1282.000000	1415.000000	3007.000000	3007.000000	3011.000000
1015.755504	6.392356	5.419788	14.348620	21.364716	1.914381
7.111794	2.388112	2.827758	6.373137	7.525479	6.229975
982.900000	0.000000	1.000000	0.300000	6.400000	0.000000
1010.700000	5.000000	2.000000	9.200000	14.900000	0.000000
1015.600000	8.000000	7.000000	14.100000	20.600000	0.000000
1020.500000	8.000000	8.000000	19.300000	27.200000	0.400000
1036.100000	8.000000	8.000000	34.500000	43.400000	104.200000

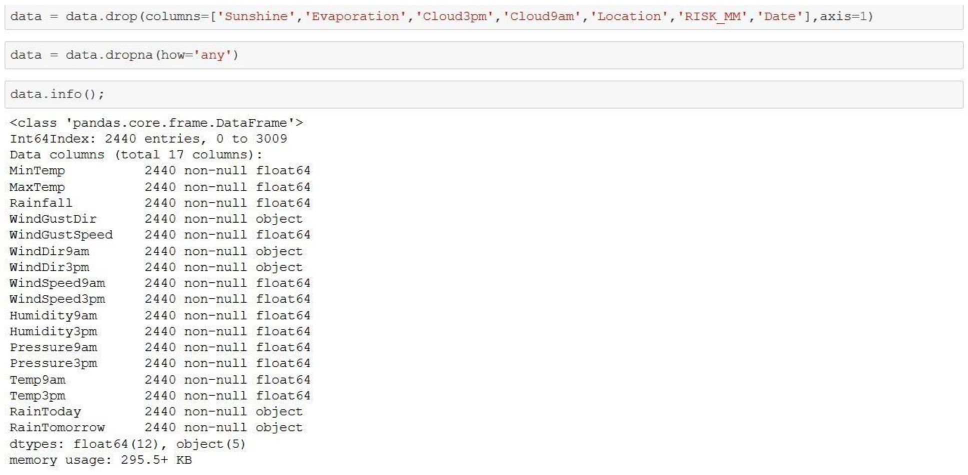
Correlation between two numerical variables is defined as the percentage of variance in one variable explained by the other variable. The correlation between all the numerical variables of our data can be observed below



```
display(data.corr(method='pearson', min_periods=1))
```

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Temp9am	Temp3pm
MinTemp	1.000000	0.788069	0.090190	0.339622	0.201239	0.173513	-0.545613	-0.369475	-0.517136	-0.536718	0.922730	0.760671
MaxTemp	0.788069	1.000000	-0.158683	0.172018	-0.073267	0.030367	-0.738059	-0.757751	-0.250778	-0.349465	0.919372	0.990363
Rainfall	0.090190	-0.158683	1.000000	0.137497	0.173748	0.105702	0.267838	0.294935	-0.264734	-0.195874	-0.043576	-0.164311
WindGustSpeed	0.339622	0.172018	0.137497	1.000000	0.475108	0.662842	-0.261710	-0.090622	-0.576527	-0.562840	0.296458	0.137892
WindSpeed9am	0.201239	-0.073267	0.173748	0.475108	1.000000	0.424867	-0.204975	0.058466	-0.387454	-0.294299	0.104075	-0.098618
WindSpeed3pm	0.173513	0.030367	0.105702	0.662842	0.424867	1.000000	-0.139968	-0.091799	-0.474004	-0.420116	0.129744	0.014802
Humidity9am	-0.545613	-0.738059	0.267838	-0.261710	-0.204975	-0.139968	1.000000	0.756558	0.226123	0.272455	-0.727632	-0.724131
Humidity3pm	-0.369475	-0.757751	0.294935	-0.090622	0.058466	-0.091799	0.756558	1.000000	0.038712	0.101741	-0.571710	-0.787971
Pressure9am	-0.517136	-0.250778	-0.264734	-0.576527	-0.387454	-0.474004	0.226123	0.038712	1.000000	0.959264	-0.423190	-0.215330
Pressure3pm	-0.536718	-0.349465	-0.195874	-0.562840	-0.294299	-0.420116	0.272455	0.101741	0.959264	1.000000	-0.484223	-0.318506
Temp9am	0.922730	0.919372	-0.043576	0.296458	0.104075	0.129744	-0.727632	-0.571710	-0.423190	-0.484223	1.000000	0.897117
Temp3pm	0.760671	0.990363	-0.164311	0.137892	-0.098618	0.014802	-0.724131	-0.787971	-0.215330	-0.318506	0.897117	1.000000

The columns ‘Sunshine’, ’Evaporation’, ’Cloud3pm’, Cloud9am’, ’Location’, ’RISK_MM’, ’Date’ constitute to less than 60% of whole data and hence they can be neglected. We are using drop() of pandas with axis =1 indicating to drop each respective feature column wise. Null values if any also needs to be dropped as they lead to unnecessary complexities and might mislead us to wrong modelling. This can be achieved using dropna() of pandas with parameter how=’any’ indicating to remove null values both column and row wise i.e., if null value is present drop both that row and column. I chose to drop instead of replacing with certain values to avoid complexities in this project.



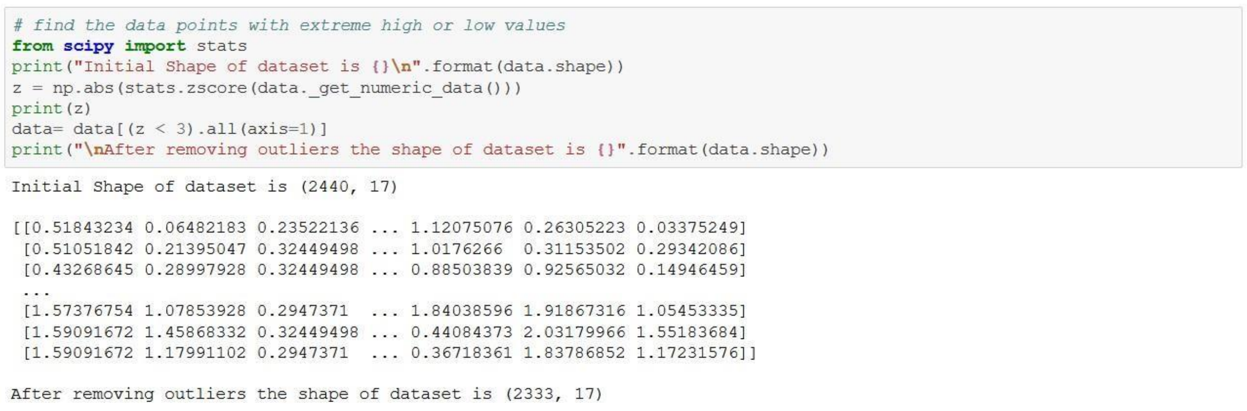
```
data = data.drop(columns=['Sunshine','Evaporation','Cloud3pm','Cloud9am','Location','RISK_MM','Date'],axis=1)
```

```
data = data.dropna(how='any')
```

```
data.info();
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2440 entries, 0 to 3009
Data columns (total 17 columns):
MinTemp          2440 non-null float64
MaxTemp          2440 non-null float64
Rainfall         2440 non-null float64
WindGustDir      2440 non-null object
WindGustSpeed    2440 non-null float64
WindDir9am       2440 non-null object
WindDir3pm       2440 non-null object
WindSpeed9am     2440 non-null float64
WindSpeed3pm     2440 non-null float64
Humidity9am      2440 non-null float64
Humidity3pm      2440 non-null float64
Pressure9am      2440 non-null float64
Pressure3pm      2440 non-null float64
Temp9am          2440 non-null float64
Temp3pm          2440 non-null float64
RainToday        2440 non-null object
RainTomorrow     2440 non-null object
dtypes: float64(12), object(5)
memory usage: 295.5+ KB
```

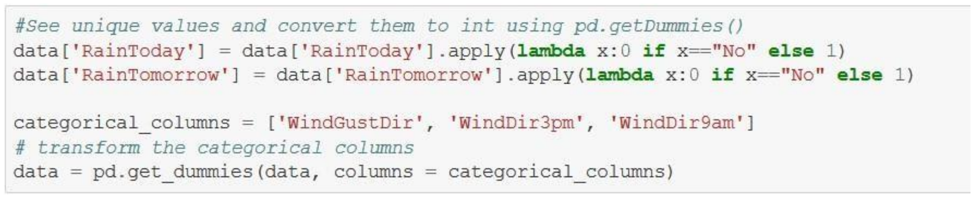
Outliers cause serious deviations in statistical analysis. They lead to wrong statistical outcomes in analysis. A datapoint which is significantly different from other datapoints is termed as an Outlier. While plotting datapoints in a graph we can observe outliers plotted at a farther distance from rest of datapoints. In this project we remove outliers for better statistical analysis. In this project I utilized z-score to remove datapoints with extreme low or high values i.e., outliers. We can observe that the dimensions of data reduced from (2440,17) to (2333,17). We can study more about outliers from this link.



```
# find the data points with extreme high or low values
from scipy import stats
print("Initial Shape of dataset is {}\n".format(data.shape))
z = np.abs(stats.zscore(data._get_numeric_data()))
print(z)
data= data[(z < 3).all(axis=1)]
print("\nAfter removing outliers the shape of dataset is {}".format(data.shape))
```

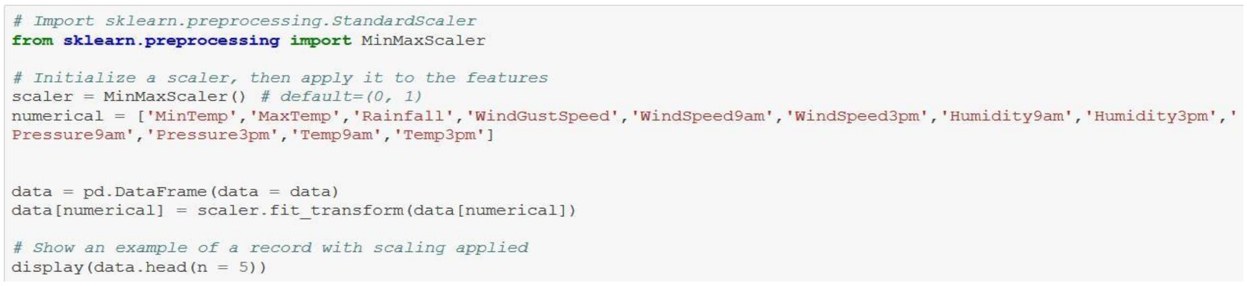
```
Initial Shape of dataset is (2440, 17)

[[0.51843234 0.06482183 0.23522136 ... 1.12075076 0.26305223 0.03375249]
 [0.51051842 0.21395047 0.32449498 ... 1.0176266  0.31153502 0.29342086]
 [0.43268645 0.28997928 0.32449498 ... 0.88503839 0.92565032 0.14946459]
 ...
 [1.57376754 1.07853928 0.2947371  ... 1.84038596 1.91867316 1.05453335]
 [1.59091672 1.45868332 0.32449498 ... 0.44084373 2.03179966 1.55183684]
 [1.59091672 1.17991102 0.2947371  ... 0.36718361 1.83786852 1.17231576]]

After removing outliers the shape of dataset is (2333, 17)
```

We need to convert the categorical data into numerical data. We can find that ‘WindGustDir’, ’WindDir3pm’, ’WindDir9am’, ’RainToday’, ’RainTomorrow’ are categorical columns/variables. Out of these columns ‘RainToday’ and ‘RainTomorrow’ are binary class variables. So, a simple lambda function is applied using apply() where the class outcome ‘No’ is represented by numerical value zero and ‘Yes’ is represented by ‘Yes’. For rest of categorical columns one-hot encoding is applied using get_dummies() of pandas which creates binary columns in our data.



```
#See unique values and convert them to int using pd.getDummies()
data['RainToday'] = data['RainToday'].apply(lambda x:0 if x=="No" else 1)
data['RainTomorrow'] = data['RainTomorrow'].apply(lambda x:0 if x=="No" else 1)

categorical_columns = ['WindGustDir', 'WindDir3pm', 'WindDir9am']
# transform the categorical columns
data = pd.get_dummies(data, columns = categorical_columns)
```

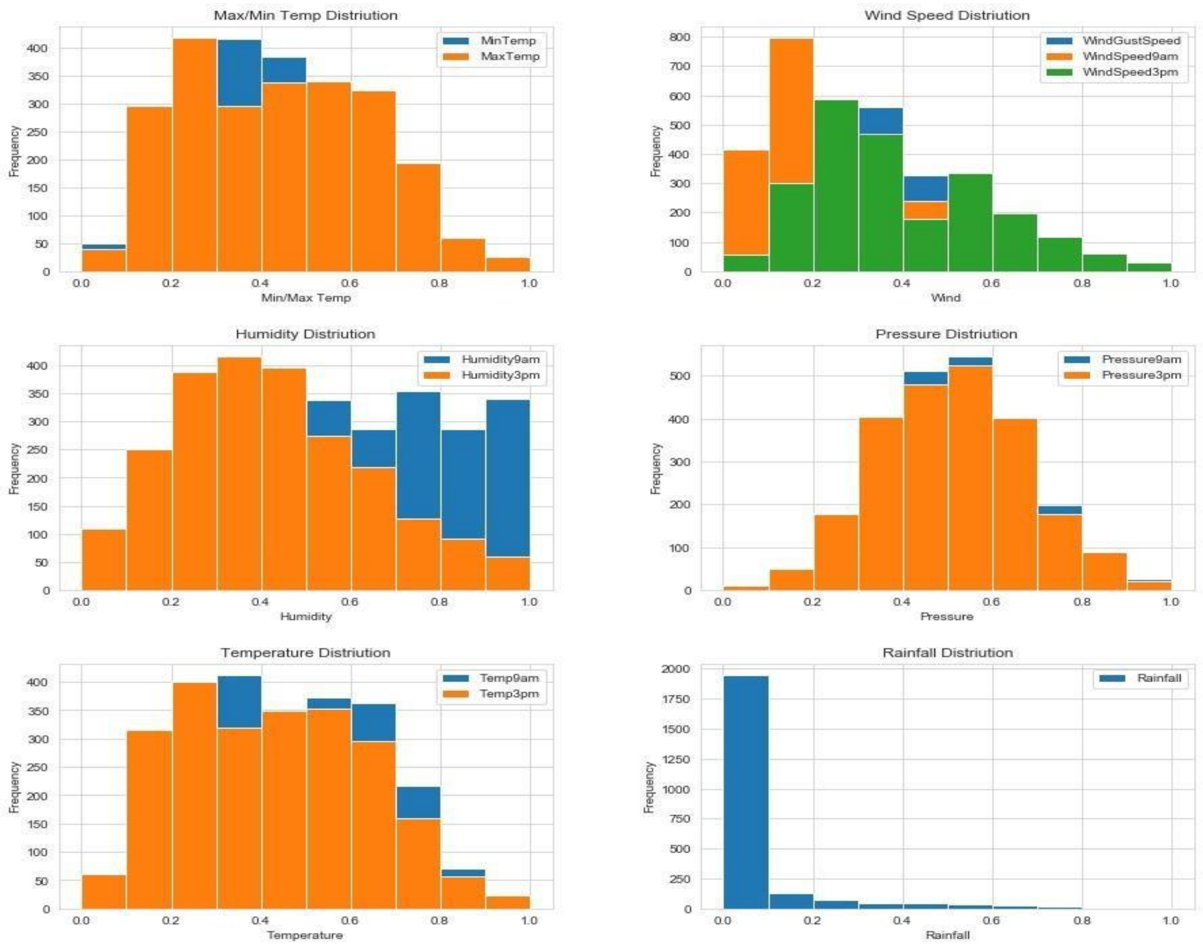
Generally, features with high numerical value have more impact on outcome/predictions when compared with features having low numerical values. To avoid this kind of scenario we apply MinMaxScaler .This restricts numerical values of all features of data between a range. This also prevents the impact of a particular feature on the end result due to its high numerical values. I applied MinMaxScaler to the features ‘MinTemp’, ‘MaxTemp’, ‘Rainfall’, ‘WindGustSpeed’, ‘WindSpeed9am’, ‘WindSpeed3pm’,’Humidity9am’, ‘Humidity3pm’, ‘Pressure9am’, ‘Pressure3pm’, ‘Temp9am’, ‘Temp3pm’.



```
# Import sklearn.preprocessing.StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)
numerical = ['MinTemp','MaxTemp','Rainfall','WindGustSpeed','WindSpeed9am','WindSpeed3pm','Humidity9am','Humidity3pm','
Pressure9am','Pressure3pm','Temp9am','Temp3pm']


data = pd.DataFrame(data = data)
data[numerical] = scaler.fit_transform(data[numerical])

# Show an example of a record with scaling applied
display(data.head(n = 5))
```

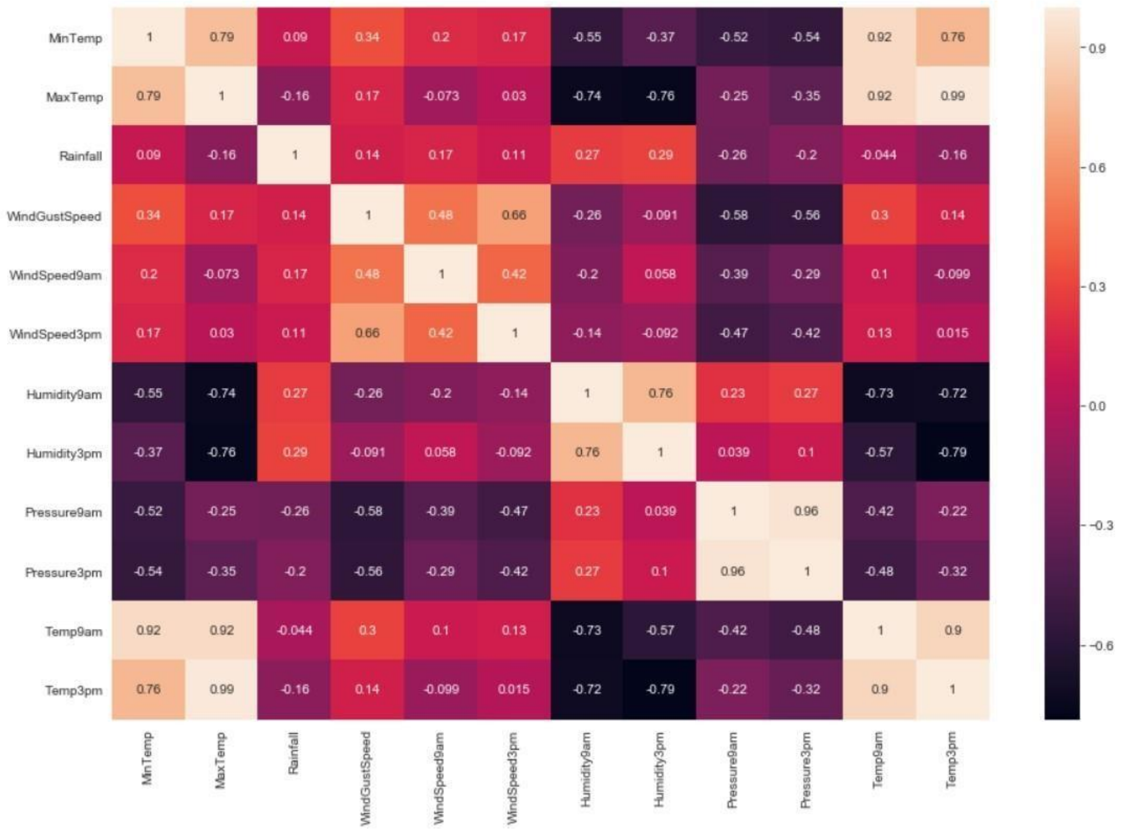
Exploratory Visualization

All the relevant features of our data are plotted to aid better analysis. It becomes easy to analyze our data based on visualization rather than analyzing from statistical measures. We can also gain hidden insights of data via visualization of our data. This is a famous step used by Data Scientists and is termed as Exploratory data analysis.

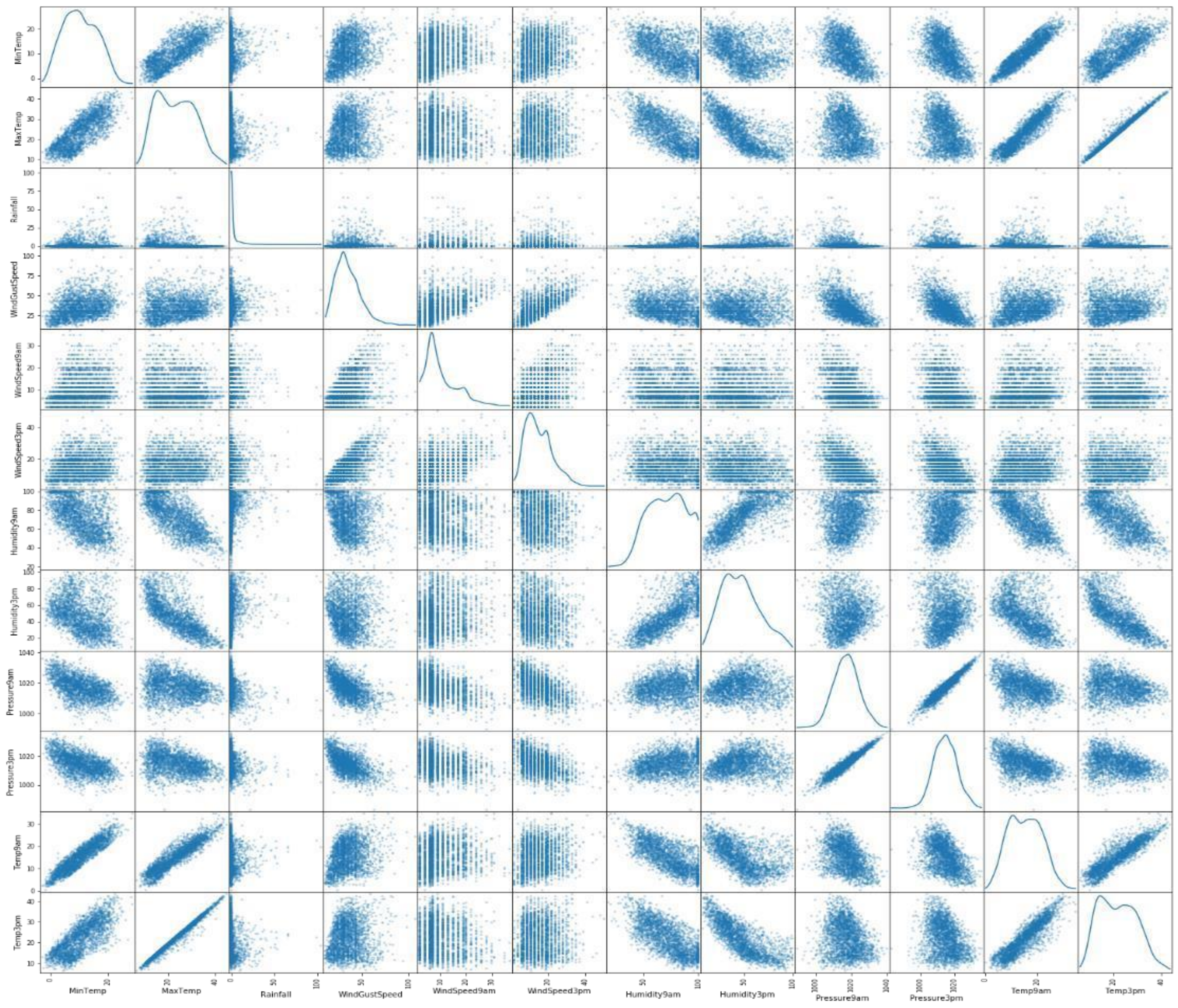


I used the above plot only for relevant features to check if they are normally distributed. We could observe that majority of relevant features are normally distributed. If they are not normally distributed, we need to normalize our data as better analysis can be done on normalized data.

The following is a correlation heatmap. I used it to visualize the degree of correlation between a pair of relevant features located in two discrete dimensions for which it uses colored cells in a monochromatic scale. I used heatmap also because it is a simple way to visualize correlation matrix. For example, consider ‘Temp3pm’ and ‘MinTemp’, we can observe that a high degree of correlation exists between these two features. The degree of correlation is a numerical value which ranges from 0 to 1 with 0 being lowest degree of correlation and 1 being highest degree of correlation between two variables. Similarly, we can find extent of correlation between other pairs of relevant features.



The below plot is called pairplot from seaborn module. I used this to visualize correlation between two relevant features. It also has two dimensions just like heatmap. While heatmap uses colored cells, pairplot utilizes scatterplots to visualize correlation trend between a pair of relevant features.For example consider two features MinTemp and MaxTemp, we can find positive trend of correlation between these two features.



Algorithms and Techniques

Decision Trees:

It is very simple machine learning algorithm used for both Classification and Regression. Decision Tree algorithms can be applied to estimate both Numeric and Non-Numeric output variable. Like any algorithm, Decision tree too has its share of pros and cons. Let us try to list a few of them out:

Pros:

- Decision Trees are white boxes - which means the results can be interpreted very clearly. The outcome is transparent.
- Decision Trees can be applied for both Numerical and Categorical independent variables.
- Missing values in attributes can be efficiently handled.

- Decision trees help with the probability & payoffs of decisions - Information Gain.
- Factors/Feature selection is a byproduct of Decision Trees. ● Decision Trees can provide a visual output of the model.
- These models are fast.
- Linearity between independent and dependent variable is not a constraint for prediction.

Cons:

- The hierarchical structure of the tree causes results to be very unstable if data is altered slightly.
- Size of the Tree may grow without any control.
- Decision Trees split the data based on a single attribute, it may not be helpful.
- Information gain for variables with a high number of subclasses gives a biased response for said attributes.
- Suffers from the problem of overfitting where the tree becomes more specific to data on which it is trained.

Real world application:

Predicting housing prices in real estate business.

References:

https://www.quora.com/What-are-the-advantages-of-using-a-decision-tree-for-classi fication

Logistic Regression:

As the name suggests Logistic Regression is not used for regression instead used for binary classification. It gives you a discrete binary outcome between 0 and 1. Logistic Regression measures the relationship between the dependent variable and the independent variable(s), by estimating probabilities using its underlying logistic function. These probabilities must then be transformed into binary values in order to actually make a prediction.

Pros:

- It is incredibly easy to implement and very efficient to train.
- Logistic Regression is also a good baseline that you can use to measure the performance of other more complex Algorithms.
- Conditional probabilities are calculated while implementing Logistic Regression, which can be very useful in certain applications.
- This algorithm has low variance.
- Feature engineering can be applied to convert most of non-linear features into linear ones.
- Most widely used algorithm.
- It is very robust to noise.

Cons:

- We can't solve non-linear problems with logistic regression since it's decision surface is linear.
- This algorithm can suffer the problem of high bias • Cannot handle categorical variables well.

Real world application:

- Breast Cancer detection.

References:

https://www.quora.com/What-are-the-pros-and-cons-of-using-logistic-regression-with-onebinary-outcome-and-several-binary-predictors

KNN:

KNN stands for k-nearest neighbors. This algorithm can be used for both classification and regression problems. It is a simple machine learning algorithm that categorizes an input by using its *k* nearest neighbors.

Pros:

- K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.
- K-NN is a non-parametric algorithm which means there are no assumptions to be met to implement K-NN unlike parametric models like linear regression.
- K-NN does not explicitly build any model. New data entry would be tagged with majority class in the nearest neighbor.
- Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
- One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.
- While matching a new datapoint to a class K-NN uses the distance measure. It gives user the flexibility to choose distance while building K-NN model. By default, K-NN classifier uses Minkowski Distance

1. Euclidean Distance
2. Hamming Distance
3. Manhattan Distance 4. Minkowski Distance

Cons:

- As dataset grows speed of algorithm declines very fast as K-NN searches through entire dataset.
- As the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
- One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
- k-NN doesn’t perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A. This might result in getting the less common class B wrongly classified.

- K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
- K-NN has no capability of dealing with missing value problem.

References: https://brilliant.org/wiki/k-nearest-neighbors/

Real world application:

K-NN is used for pattern recognition

GridSearchCV:

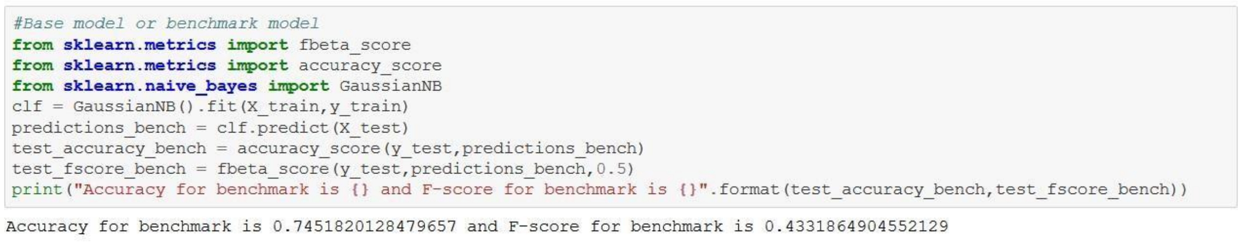
Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

class sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, *scoring=None*, *n_jobs=None*, *iid='warn'*, *refit=True*, *cv='warn'*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score='raise-deprecating'*, *return_train_score=False*) **References:**

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Benchmark

For the benchmark model, I used Gaussian Naïve Bayes algorithm from scikit learn module. I used Accuracy and F-score as evaluation metrics. As the data is skewed, I preferred to use Fscore for evaluation of performance. As F-score is performance evaluation metric, this score of Benchmark model will be compared with the respective scores of the new models. The purpose of this comparison of scores is to check how better the new models are good at prediction when compared with benchmark model.



```
#Base model or benchmark model
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB().fit(X_train,y_train)
predictions_bench = clf.predict(X_test)
test_accuracy_bench = accuracy_score(y_test,predictions_bench)
test_fscore_bench = fbeta_score(y_test,predictions_bench,0.5)
print("Accuracy for benchmark is {} and F-score for benchmark is {}".format(test_accuracy_bench,test_fscore_bench))
```

```
Accuracy for benchmark is 0.7451820128479657 and F-score for benchmark is 0.4331864904552129
```

III. Methodology

Data Preprocessing

In this step data is refined and prepared as follows:

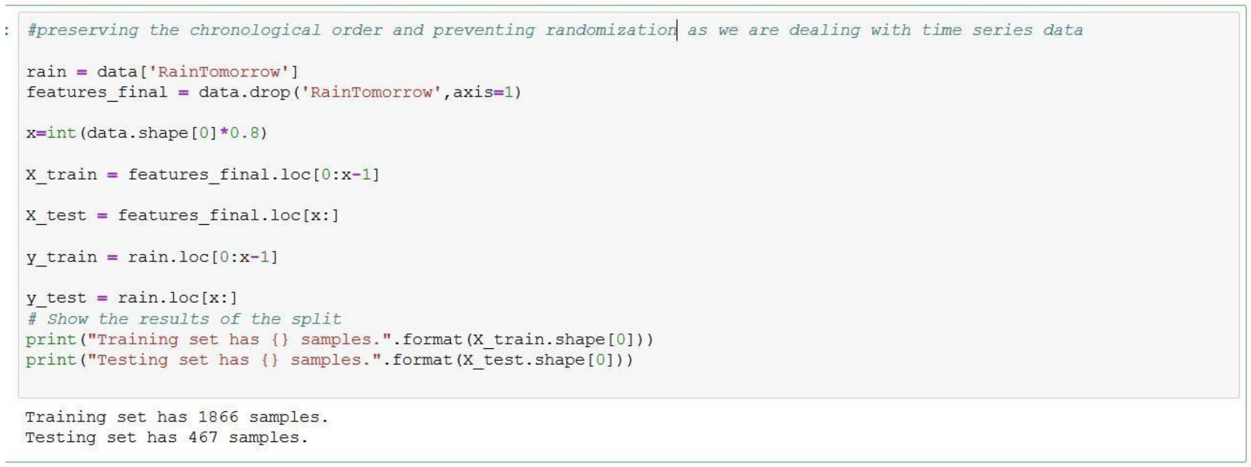
- Initially, I've downloaded the weather dataset from Kaggle and I chose the subset from the entire dataset on which I wanted to apply ML and predict the outcome.
- Loading the data into pandas DataFrame, I checked how the data is and came to know the basic statistical insights of data I'm dealing with.
- I then removed certain features of the dataset as they constitute less than 60% of total data. They will have no impact on the outcome predicted i.e., they are not relevant features.
- I used dropna() of pandas to drop null values if any existed both column wise and row wise i.e., if there is null value remove both row and column. Later I checked all the relevant features for null values and none existed as expected.
- As this is a classification problem, I need to choose the performance evaluation metric for the models I would construct later on. For this purpose, I plotted the counts of individual classes of predictor variable 'RainTomorrow' and found that data is imbalanced and I decided to use F-score by referring to online articles.
- I plotted all the relevant features as grouped based on their similarity and found that data is almost normally distributed.
- Out of curiosity I wanted to check the correlation between each pair of features and hence I printed the correlation matrix. I visualized the correlations using heatmap and pairplot.
- Outliers affect the accuracy of prediction and hence I removed outliers of the data using Z-score.
- Some of the algorithms which I wanted to use later on for modelling cannot deal with categorical data. So, I converted categorical data into numerical data i.e., some features using onehot encoding and some features by applying lambda functions. The categorical columns 'RainToday' and 'RainTomorrow' had only two possible outcomes and hence I utilized apply() and lambda functions to assign 0 for 'No' and 1 for 'Yes' whereas the columns 'WindGustDir', 'WindDir3pm', 'WindDir9am' have more possible outcomes and hence I utilized one hot encoding for these features.
- As we are dealing with numerical data some features might have high numerical values ad others features might have low numerical values. The features with high numerical values tend to have more impact on outcomes predicted. So, to avoid this bias and to restrict the values in between a range I utilized MinMaxScaler.

Implementation:

Train/Test Split

- Now we split our data into independent variables and dependent variable. We could do this step before preprocessing step and apply preprocessing individually for both variables. But I have done the preprocessing step collectively for whole data. Our independent variables is all the features except 'RainTomorrow' whereas dependent variable is 'RainTomorrow'.

- In Supervised Learning, there are two phases namely training phase and testing phase. In training phase our model learns from training data and in testing phase our model predicts on testing data.
- Generally, a train test split of 80:20 or 70:30 is followed worldwide. This signifies that a model spends majority of its time in learning or training. As we are dealing with time series data, I did not choose to randomly pick training and testing data. I preserved the chronological order of data while picking 80% of data as training data and 30% of data as testing data.
- We can observe that our data is split into 1866 training samples and 467 testing samples. This means that the model learns from 1866 samples and tests it on 467 samples.
- Our model learns from training data and applies what it learnt on testing data. Training data is associated with class labels, which the model is expected to learn as it is and it tests what it learnt by predicting labels for testing data. To measure the performance of model i.e., proportion of correctness of what it learnt from training data an evaluation metric (F-score in our case) is used which computes score by comparing the predicted values and the actual outcomes of the testing data. Higher the F-score higher the performance which signifies that our model learnt well from training data and predicted more correctly on testing data.

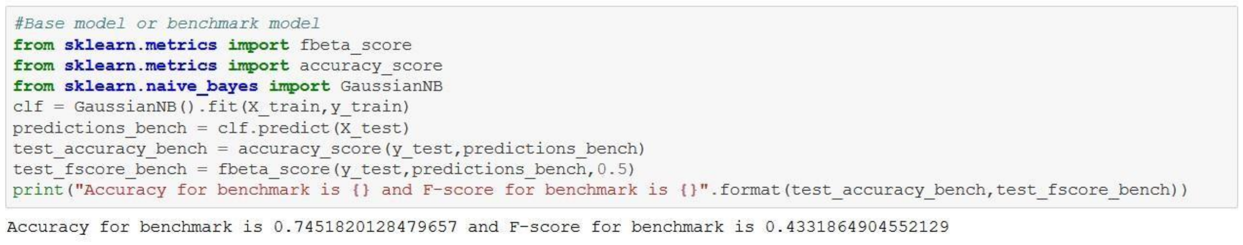


```
#preserving the chronological order and preventing randomization as we are dealing with time series data

rain = data['RainTomorrow']
features_final = data.drop('RainTomorrow',axis=1)

x=int(data.shape[0]*0.8)

X_train = features_final.loc[0:x-1]

X_test = features_final.loc[x:]

y_train = rain.loc[0:x-1]

y_test = rain.loc[x:]
# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 1866 samples.
Testing set has 467 samples.
```

Benchmark model

For the benchmark model, I used Gaussian Naïve Bayes algorithm from scikit learn module. I used Accuracy and F-score as evaluation metrics. As the data is skewed, I preferred to use Fscore for evaluation of performance. As F-score is performance evaluation metric, this score of Benchmark model will be compared with the respective scores of the new models. The purpose of this comparison of scores is to check how better the new models are good at prediction when compared with benchmark model. The Accuracy and F-score for benchmark model is 0.7430 and 0.44096 respectively. As explained these values will be used for comparison with other models.

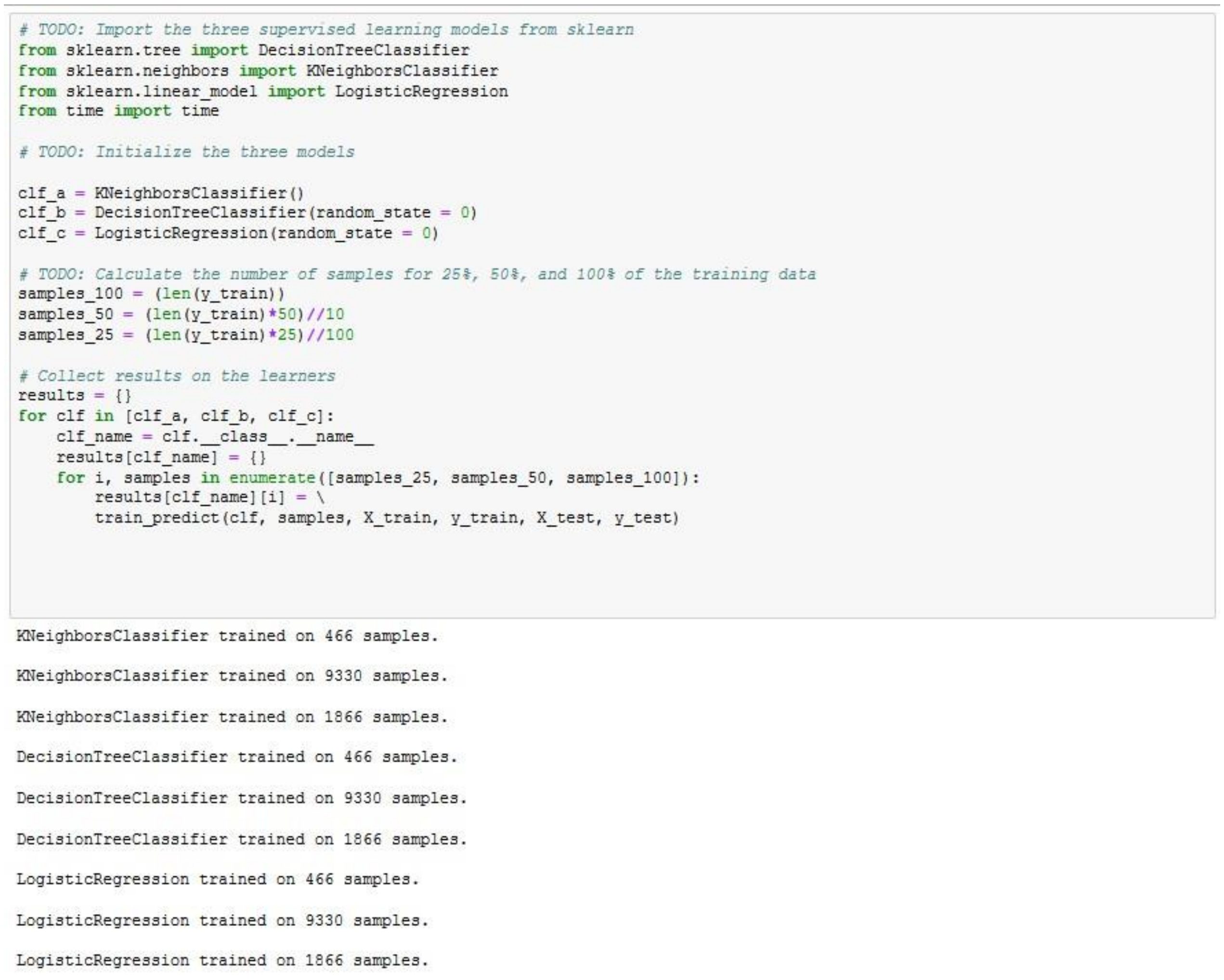


```
#Base model or benchmark model
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB().fit(X_train,y_train)
predictions_bench = clf.predict(X_test)
test_accuracy_bench = accuracy_score(y_test,predictions_bench)
test_fscore_bench = fbeta_score(y_test,predictions_bench,0.5)
print("Accuracy for benchmark is {} and F-score for benchmark is {}".format(test_accuracy_bench,test_fscore_bench))
```

```
Accuracy for benchmark is 0.7451820128479657 and F-score for benchmark is 0.4331864904552129
```

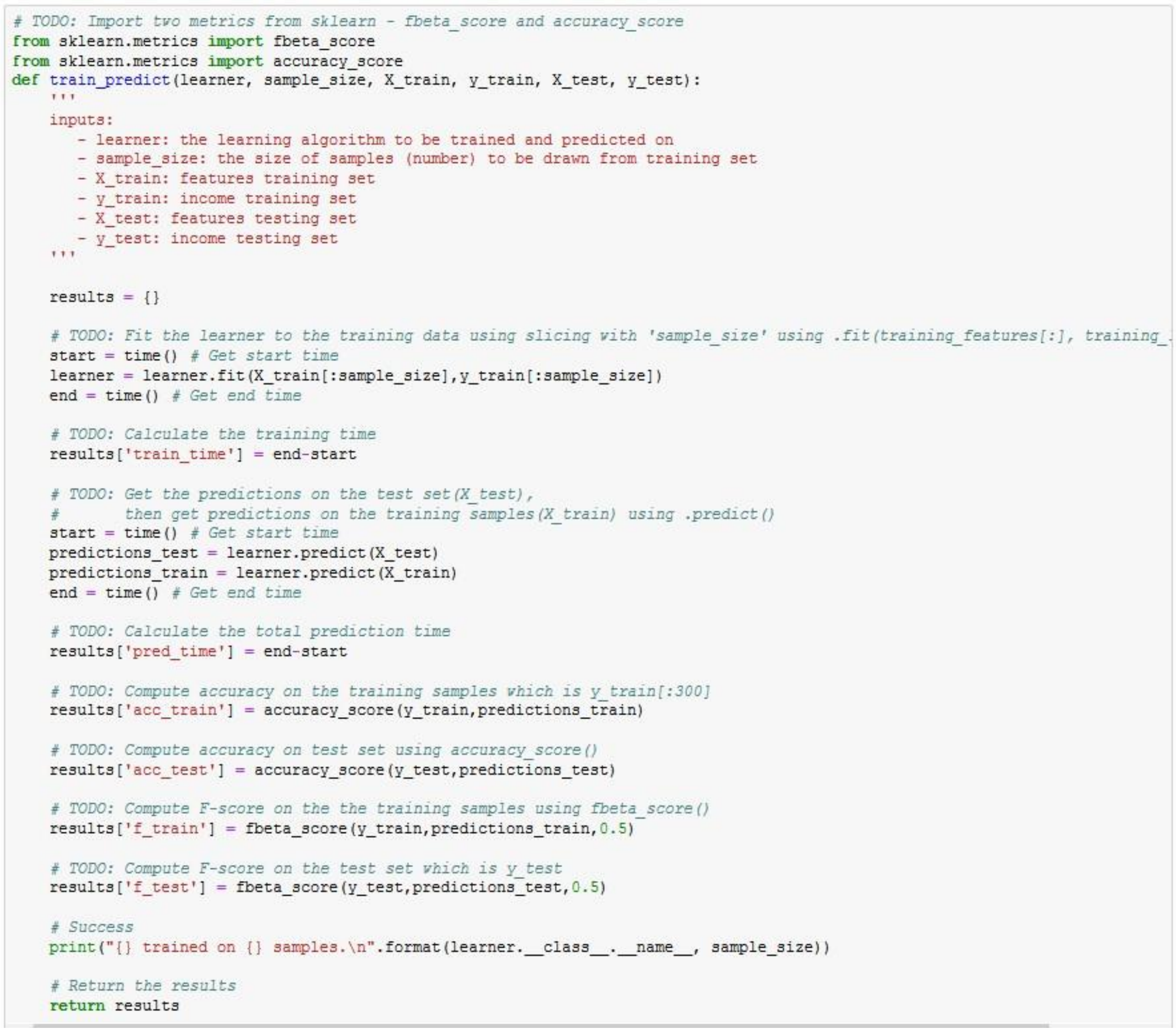
Algorithms

In pursuit of creating a better classification model I have chosen a few Supervised Learning algorithms and finally got settled with three of them. They are **K-NN, DecisionTreeClassifier** and **Logistic Regression**. A detailed explanation of all the algorithms is given in Algorithms and techniques section. I created separate models for each Supervised Learning algorithm that I chose for 25%,50% and 100% of training data respectively. I fit models on different sample sizes just out of curiosity. For each sample size of training data, I calculated training time, prediction time, training accuracy, testing accuracy, F-score for training data and F-score for testing data respectively. For this purpose I wrote train_predict() which stores entire results in a dictionary named result. I created a visual for this entire scenario using the result dictionary I computed previously. Sometimes, to get high performance model we need to compromise with learning time i.e., models with high performance might take more time in learning phase. This was not the case with me because I found that all the three models took same time to train on 100% training data. With the help of visuals, it became easy for me to choose better model with high performance and it turned out to be Logistic Regression with an F-score of 0.7064

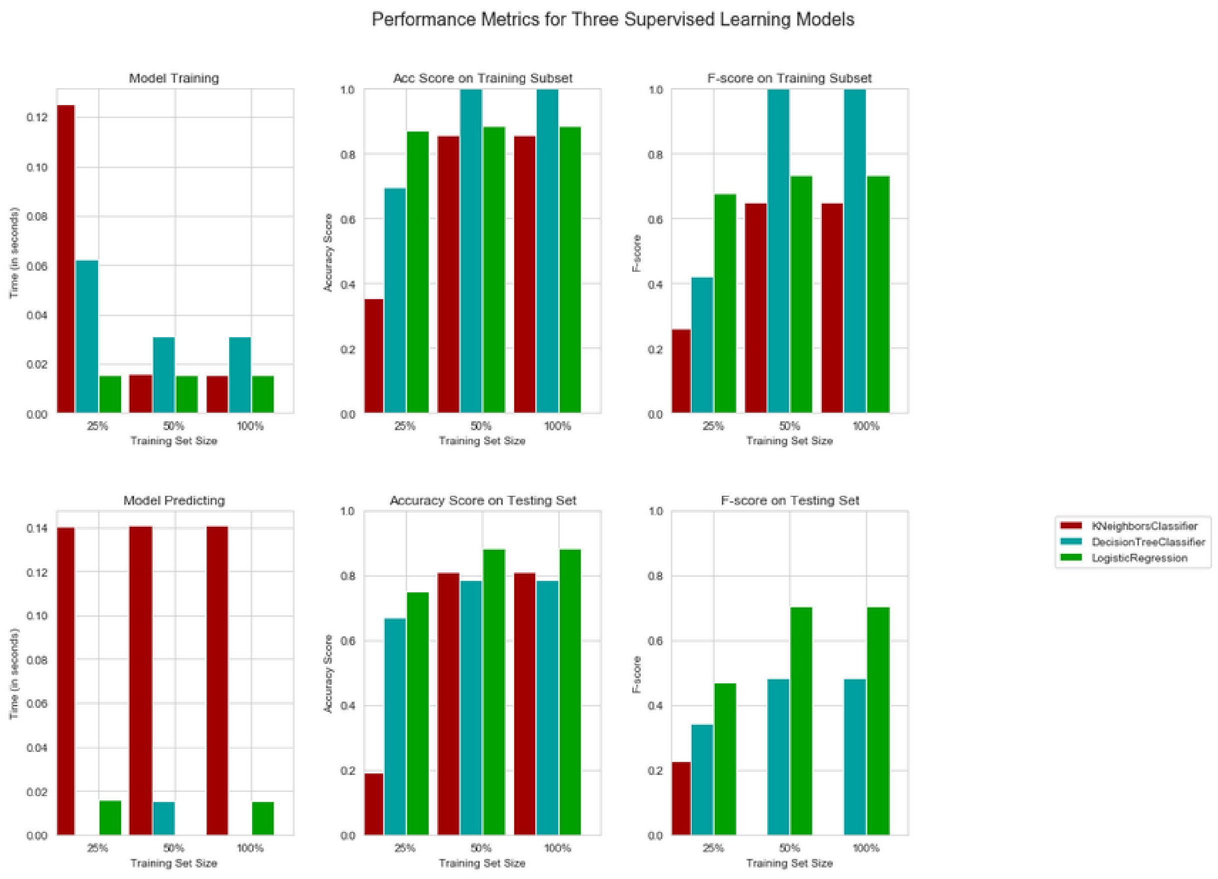


```python
# TODO: Import the three supervised learning models from sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from time import time

# TODO: Initialize the three models

clf_a = KNeighborsClassifier()
clf_b = DecisionTreeClassifier(random_state = 0)
clf_c = LogisticRegression(random_state = 0)

# TODO: Calculate the number of samples for 25%, 50%, and 100% of the training data
samples_100 = (len(y_train))
samples_50 = (len(y_train)*50)//10
samples_25 = (len(y_train)*25)//100

# Collect results on the learners
results = {}
for clf in [clf_a, clf_b, clf_c]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_25, samples_50, samples_100]):
        results[clf_name][i] = \
        train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

```
KNeighborsClassifier trained on 466 samples.

KNeighborsClassifier trained on 9330 samples.

KNeighborsClassifier trained on 1866 samples.

DecisionTreeClassifier trained on 466 samples.

DecisionTreeClassifier trained on 9330 samples.

DecisionTreeClassifier trained on 1866 samples.

LogisticRegression trained on 466 samples.

LogisticRegression trained on 9330 samples.

LogisticRegression trained on 1866 samples.
```

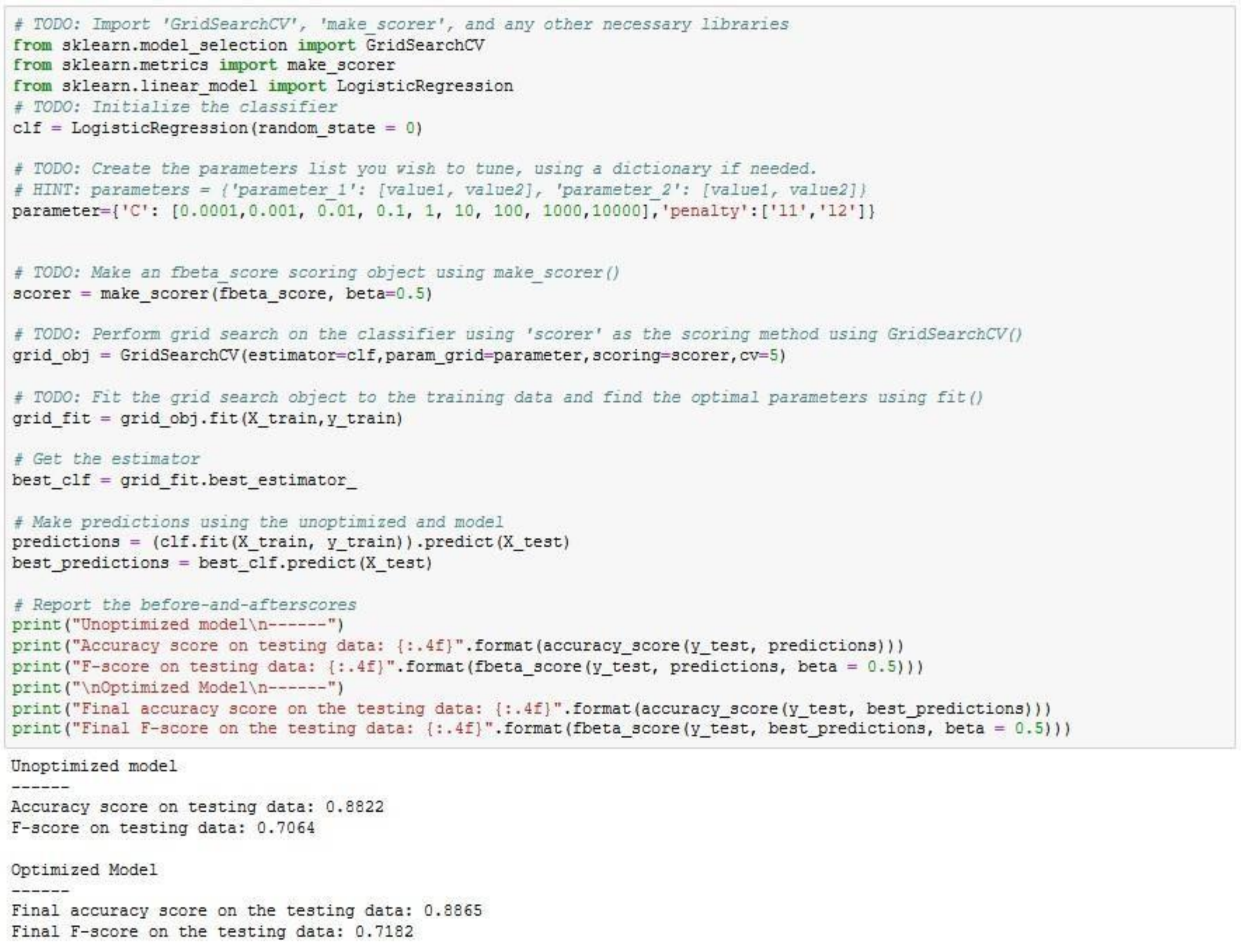


```python
# TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: income training set
       - X_test: features testing set
       - y_test: income testing set
    '''

    results = {}

    # TODO: Fit the learner to the training data using slicing with 'sample_size' using .fit(training_features[:], training_
    start = time() # Get start time
    learner = learner.fit(X_train[:sample_size],y_train[:sample_size])
    end = time() # Get end time

    # TODO: Calculate the training time
    results['train_time'] = end-start

    # TODO: Get the predictions on the test set(X_test),
    #       then get predictions on the training samples(X_train) using .predict()
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train)
    end = time() # Get end time

    # TODO: Calculate the total prediction time
    results['pred_time'] = end-start

    # TODO: Compute accuracy on the training samples which is y_train[:300]
    results['acc_train'] = accuracy_score(y_train,predictions_train)

    # TODO: Compute accuracy on test set using accuracy_score()
    results['acc_test'] = accuracy_score(y_test,predictions_test)

    # TODO: Compute F-score on the the training samples using fbeta_score()
    results['f_train'] = fbeta_score(y_train,predictions_train,0.5)

    # TODO: Compute F-score on the test set which is y_test
    results['f_test'] = fbeta_score(y_test,predictions_test,0.5)

    # Success
    print("{} trained on {} samples.\n".format(learner.__class__.__name__, sample_size))

    # Return the results
    return results
```



Refinement

Out of the three classification models created using the respective supervised learning algorithms, the model created using Logistic Regression turned out to be better model with an Fscore of 0.7064. I wanted to further improve the performance of this model by tuning its respective parameters. This can be achieved using GridSearchCV from sklearn.model_selection. For Logistic Regression two parameters namely "penalty" and "c" can be tuned. The possible values for two parameters are passed using the dictionary "parameter". For "c" parameter I have given the values 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000 and for "penalty" two possibilities exist namely "l1" and "l2". GridSearchCV finds the best possible parameter values combination therefore tuning the performance. We can find a significant increase in Accuracy from 0.8822 to 0.8865 and F-score from 0.7064 to 0.7182 resulting in an optimized model.



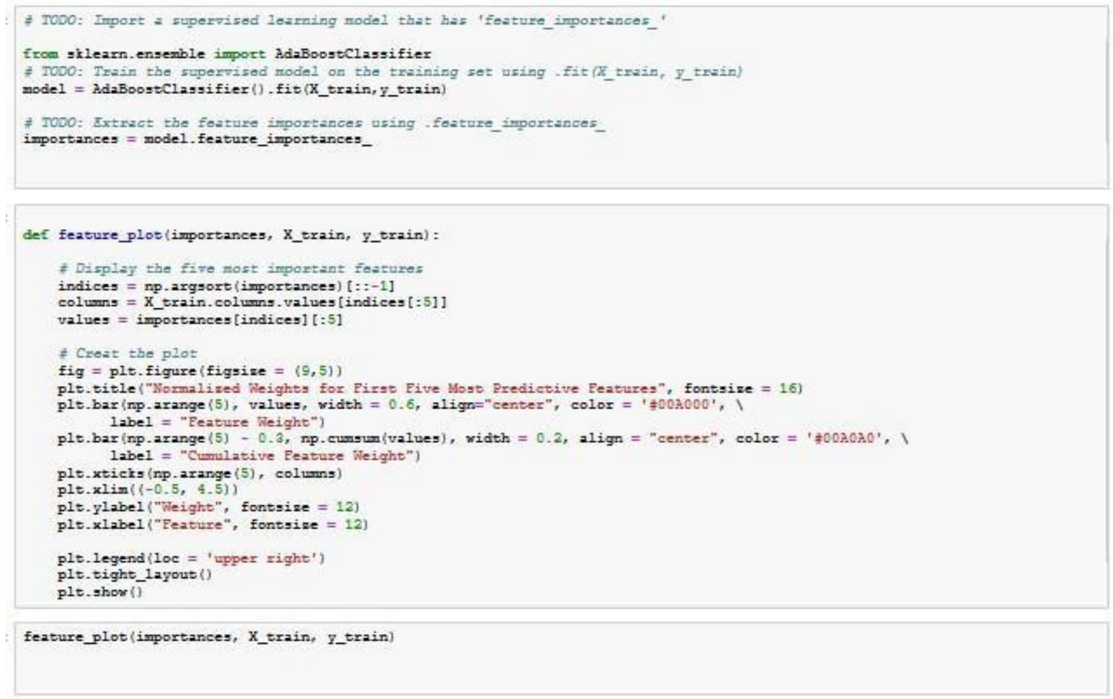
```
# TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.linear_model import LogisticRegression
# TODO: Initialize the classifier
clf = LogisticRegression(random_state = 0)

# TODO: Create the parameters list you wish to tune, using a dictionary if needed.
# HINT: parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1, value2]}
parameter={'C': [0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000,10000],'penalty':['l1','l2']}


# TODO: Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score, beta=0.5)

# TODO: Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
grid_obj = GridSearchCV(estimator=clf,param_grid=parameter,scoring=scorer,cv=5)

# TODO: Fit the grid search object to the training data and find the optimal parameters using fit()
grid_fit = grid_obj.fit(X_train,y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)

# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
```

```
Unoptimized model
------
Accuracy score on testing data: 0.8822
F-score on testing data: 0.7064

Optimized Model
------
Final accuracy score on the testing data: 0.8865
Final F-score on the testing data: 0.7182
```

Feature Importances:

Out of curiosity I wanted to know the five most important features of our data and I wanted to apply the optimized model to it to check if the model performance increased more due to data reduction. To know the five most important features of our data I used AdaBoostClassifier from sklearn.ensemble. This contains an attribute called feature_importances_ which serves our purpose. I fit the AdaBoostClassifier on our entire data and I found the five most important features. You can find below the plot visualizing the five most important features of our data.

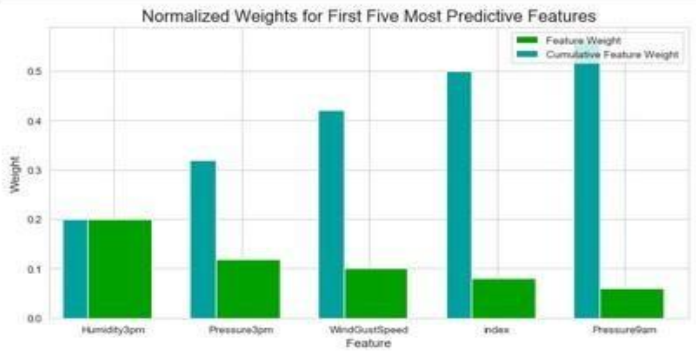
Later, I have applied the optimized model on this reduced data i.e., considering only 5 features of entire data and the F-score for this model is 0.6891. It is clear that the optimized model had high performance on full data when compared with reduced data. Hence, it is better to prefer the optimized model on full data as it has high performance.



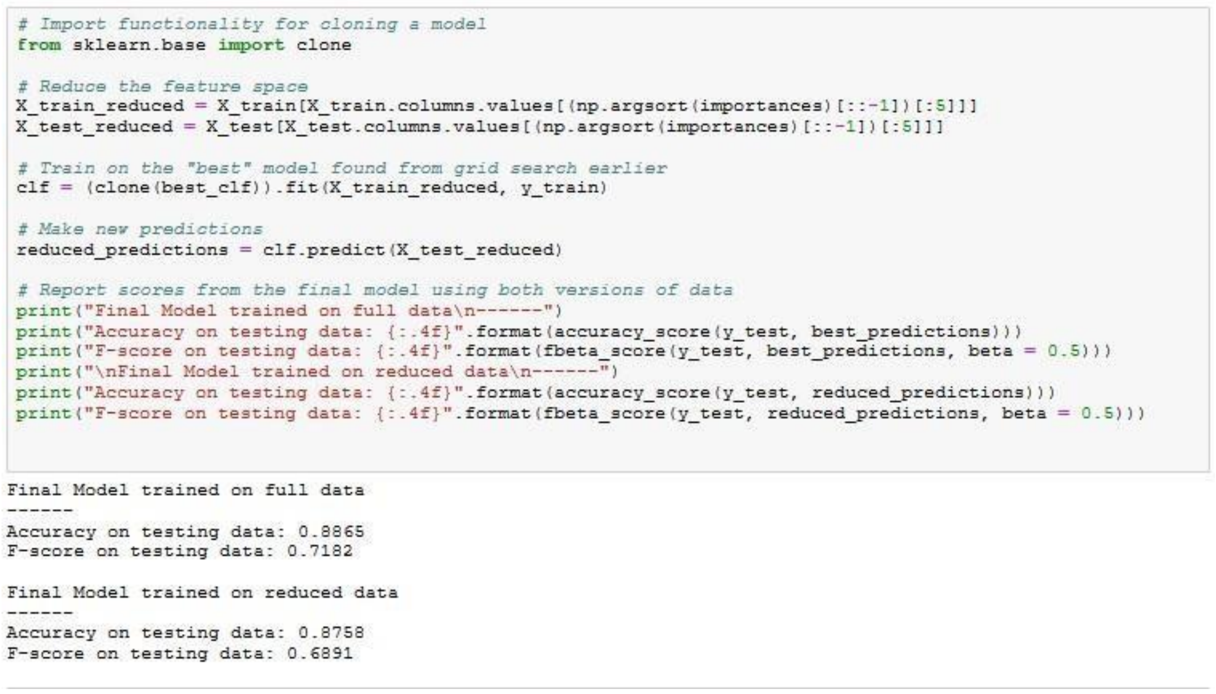
```
# TODO: Import a supervised learning model that has 'feature_importances_'

from sklearn.ensemble import AdaBoostClassifier
# TODO: Train the supervised model on the training set using .fit(X_train, y_train)
model = AdaBoostClassifier().fit(X_train,y_train)

# TODO: Extract the feature importances using .feature_importances_
importances = model.feature_importances_
```

```
def feature_plot(importances, X_train, y_train):

    # Display the five most important features
    indices = np.argsort(importances)[::-1]
    columns = X_train.columns.values[indices[:5]]
    values = importances[indices][:5]

    # Creat the plot
    fig = plt.figure(figsize = (9,5))
    plt.title("Normalized Weights for First Five Most Predictive Features", fontsize = 16)
    plt.bar(np.arange(5), values, width = 0.6, align="center", color = '#00A000', \
          label = "Feature Weight")
    plt.bar(np.arange(5) - 0.3, np.cumsum(values), width = 0.2, align = "center", color = '#00A0A0', \
          label = "Cumulative Feature Weight")
    plt.xticks(np.arange(5), columns)
    plt.xlim((-0.5, 4.5))
    plt.ylabel("Weight", fontsize = 12)
    plt.xlabel("Feature", fontsize = 12)

    plt.legend(loc = 'upper right')
    plt.tight_layout()
    plt.show()
```

```
feature_plot(importances, X_train, y_train)
```



With the help of above plot we can observe that ‘Pressure9am’ has highest cumulative frequency and ‘Humidity3pm’ has lowest cumulative frequency when compared with other features.



```
# Import functionality for cloning a model
from sklearn.base import clone

# Reduce the feature space
X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:5]]]
X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:5]]]

# Train on the "best" model found from grid search earlier
clf = (clone(best_clf)).fit(X_train_reduced, y_train)

# Make new predictions
reduced_predictions = clf.predict(X_test_reduced)

# Report scores from the final model using both versions of data
print("Final Model trained on full data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
print("\nFinal Model trained on reduced data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_predictions, beta = 0.5)))
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8865
F-score on testing data: 0.7182

Final Model trained on reduced data
------
Accuracy on testing data: 0.8758
F-score on testing data: 0.6891
```

From the above we can observe that the model which is fit on reduced data i.e., data with only 5 most important features we found that the performance reduced from 0.7182 to 0.6891.We should always opt for the model with high performance and hence here the model trained on full data is considered.

IV. Results

The 4 models namely the benchmark model, unoptimized model, optimized model on full data, optimized model on reduced data are tabulated. The final model is to be chosen based on the evaluation metric F-score but not based on Accuracy as we are dealing with skewed data. Among all these models the optimized model on full data has highest performance and is the best model in our case.

S No	Supervised Model for Classification	F-score
1.	Gaussian Naïve Bayees (Benchmark model)	0.4331
2.	Logistic Regression (Unoptimized model)	0.7064
3.	Logistic Regression (Optimized model after parameter tuning)	0.7182
4.	Logistic Regression (Optimized model on reduced data)	0.6891

- Above table clearly shows the scores of models. Our entire project can be summarized with the help of above table.
- Initially I created a benchmark model as I felt that it is better to create the same instead of random guessing. The F-score of the benchmark model is 0.4331.
- By applying 3 classification algorithms I finally figured the supervised model that best fits our data when compared with other models and it turned out to be Logistic Regression with 0.7064 F-score.
- I tuned the parameters of Logistic Regression resulting in an optimized model. I was able to create an optimized model from an unoptimized model with its performance score increasing from 0.7064 to 0.7182.
- I tried to find out if the performance of the optimized model would increase by fitting it on reduced data. For this purpose I found out 5 most important features using AdaBoostClassifier. After fitting optimized model on the 5 most important features I found the score reduced from 0.7182 to 0.6891.
- Finally, I decided the final model to be optimized model which is fit on full data (Logistic Regression after parameter tuning) as it has high performance score.

Justification

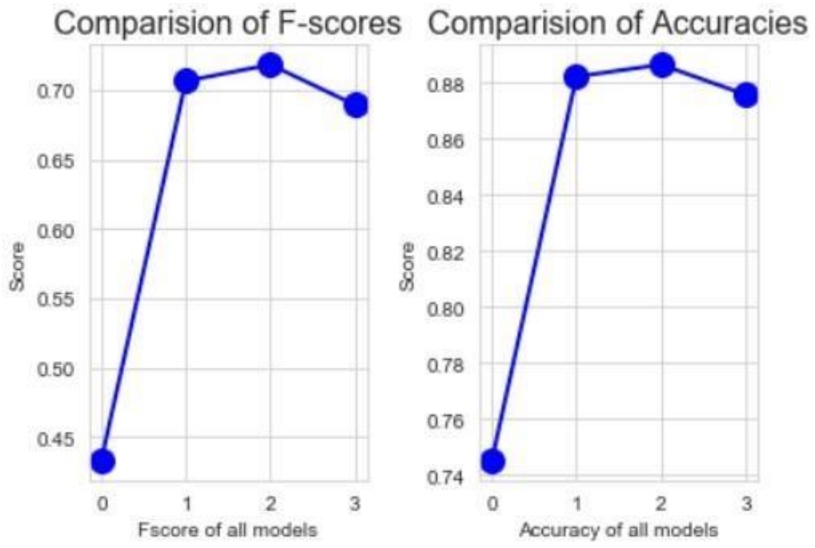
I successfully completed my project and I can clearly justify the following from this project

- The final model has performance greater than the benchmark model as expected.
- I clearly discussed and analyzed my complete project.
- I could successfully arrive at the solution as I have proposed.

- The final solution is robust with good performance and can generalize well on new/unseen data.
- The final solution can predict if it rains tomorrow in the weather station Albury, Australia for new/unseen data.

V. Conclusion

Free-Form Visualization



- The above visual clearly visualizes the scores of models that I used in this project to arrive at the solution.
- The final model has its score greater than benchmark model as expected.
- We can clearly find how the performance of final model is greater than all the other models.

Reflection:

- I have learned that data cleaning plays a crucial part in Exploratory Data Analysis (EDA).
- Removing the data features which are not necessary for evaluating a model is most important skill.
- I have Learned that reducing outliers is also very important techniques to get better results.

- During my process of doing this project, I learned how to visualize data and understand the data from visualizations very clearly.
- I have come to know how to use the best algorithm in different conditions for the dataset using appropriate techniques. ’sklearn’ helped me about respective algorithms and their parameters.
- I came to know how to evaluate our models, I used F-score as evaluation metric in this project.
- I have learned about how to use GridSearchCV for parameter tuning to get better results.
- Along with all the above points, I have learnt how to fetch dataset from open source machine learning repositories and apply techniques on it.
- I referred open communities like ‘Stackoverflow’ and ‘Quora’ in online to clear my doubts.
- Finally, I can say that I gained confidence and felt happy that I can solve a realworld problem and acquire a solution using machine learning.

Improvement

- In Future I want to replace null values with some other value like mean.
- I want to further improve the accuracy of classification and create better classification models using neural networks.