```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)


df = pd.read_csv('Weather_Data.csv')
df.head()
```

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	
0	2/1/2008	19.5	22.4	15.6	6.2	0.0	W	41	S	
1	2/2/2008	19.5	25.6	6.0	3.4	2.7	W	41	W	
2	2/3/2008	21.6	24.5	6.6	2.4	0.1	W	41	ESE	
3	2/4/2008	20.2	22.8	18.8	2.2	0.0	W	41	NNE	
4	2/5/2008	19.7	25.7	77.4	4.8	0.0	W	41	NNE	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3271 entries, 0 to 3270
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           3271 non-null   object 
 1   MinTemp        3271 non-null   float64
 2   MaxTemp        3271 non-null   float64
 3   Rainfall       3271 non-null   float64
 4   Evaporation    3271 non-null   float64
 5   Sunshine       3271 non-null   float64
 6   WindGustDir    3271 non-null   object 
 7   WindGustSpeed  3271 non-null   int64  
 8   WindDir9am     3271 non-null   object 
 9   WindDir3pm     3271 non-null   object 
 10  WindSpeed9am   3271 non-null   int64  
 11  WindSpeed3pm   3271 non-null   int64  
 12  Humidity9am    3271 non-null   int64  
 13  Humidity3pm    3271 non-null   int64  
 14  Pressure9am    3271 non-null   float64
 15  Pressure3pm    3271 non-null   float64
 16  Cloud9am       3271 non-null   int64  
 17  Cloud3pm       3271 non-null   int64  
 18  Temp9am        3271 non-null   float64
 19  Temp3pm        3271 non-null   float64
 20  RainToday      3271 non-null   object 
 21  RainTomorrow   3271 non-null   object 
dtypes: float64(9), int64(7), object(6)
memory usage: 562.3+ KB
```

```
df['Date'] = df['Date'].astype('datetime64[ns]')
df.head()
```

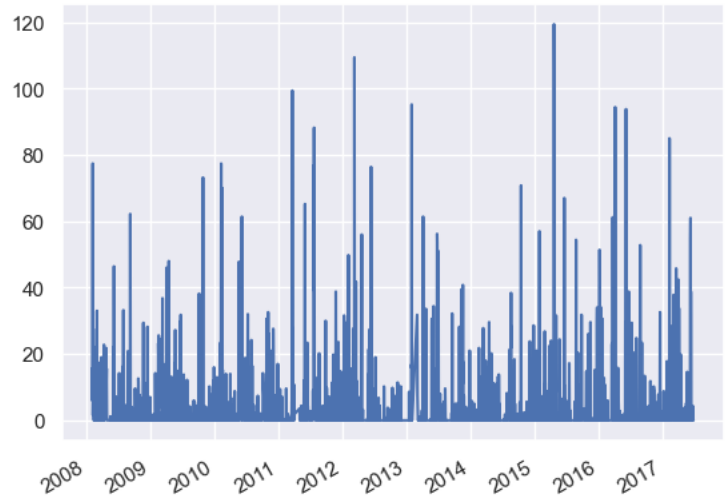
	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	Wir
0	2008-02-01	19.5	22.4	15.6	6.2	0.0	W	41	S	
1	2008-02-02	19.5	25.6	6.0	3.4	2.7	W	41	W	
2	2008-02-03	21.6	24.5	6.6	2.4	0.1	W	41	ESE	
3	2008-02-04	20.2	22.8	18.8	2.2	0.0	W	41	NNE	
4	2008-02-05	19.7	25.7	77.4	4.8	0.0	W	41	NNE	

Exploartory Data Analysis

```
x = df['Date']
y = df['Rainfall']

# plot
plt.plot(x,y)
# beautify the x-labels
plt.gcf().autofmt_xdate()

plt.show()
```

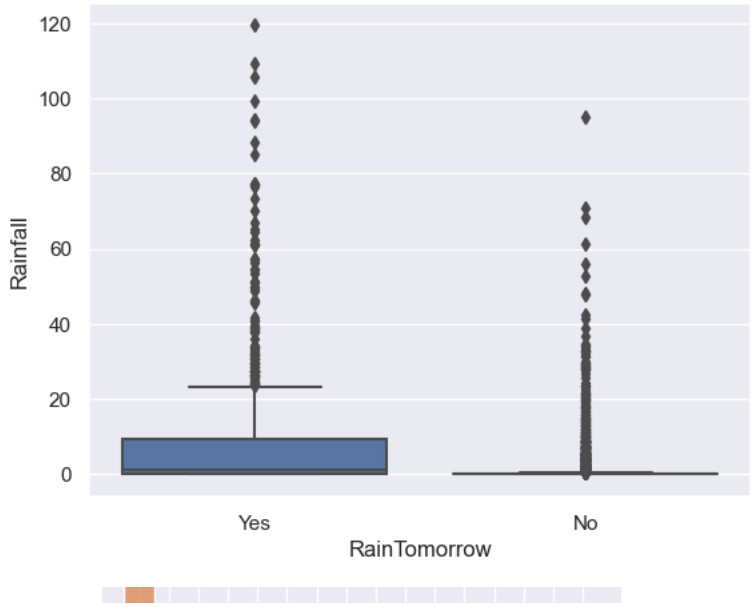



```
sns.displot(x='WindGustDir', hue='RainTomorrow', data=df, multiple='stack')
plt.xticks (rotation='vertical')
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
 [Text(0, 0, 'W'),
  Text(1, 0, 'NNW'),
  Text(2, 0, 'WNW'),
  Text(3, 0, 'ENE'),
  Text(4, 0, 'NNE'),
```

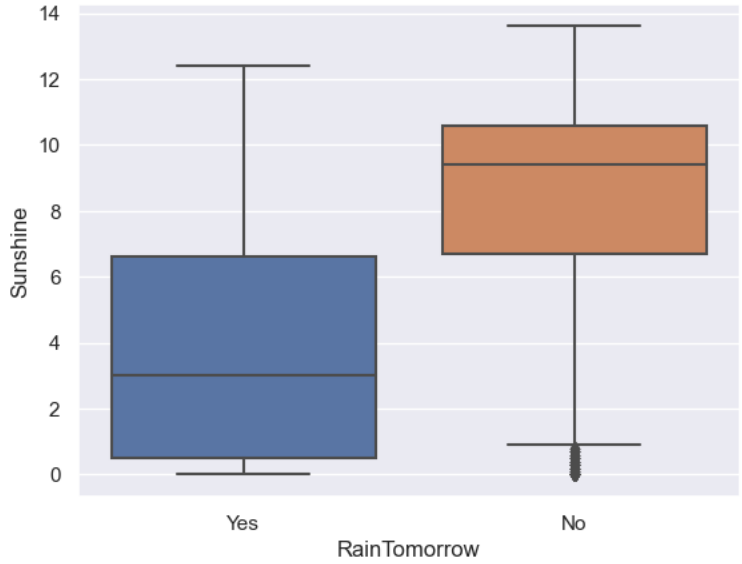
```
sns.boxplot(data=df, x="RainTomorrow", y="Rainfall")
```

```
<AxesSubplot:xlabel='RainTomorrow', ylabel='Rainfall'>
```



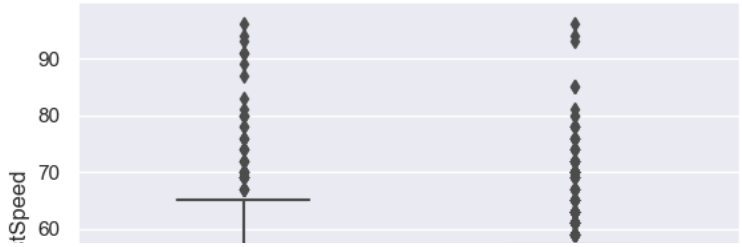

```
sns.boxplot(data=df, x="RainTomorrow", y="Sunshine")
```

```
<AxesSubplot:xlabel='RainTomorrow', ylabel='Sunshine'>
```



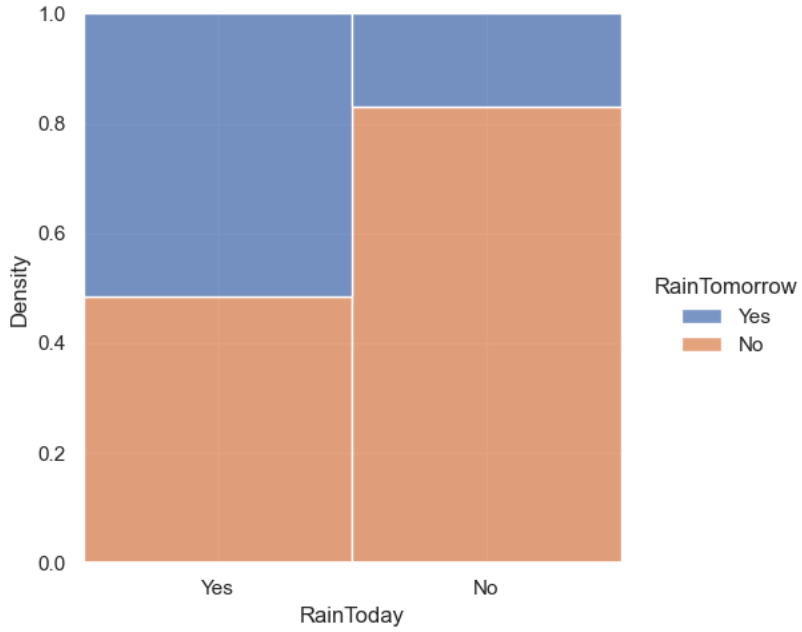

```
sns.boxplot(data=df, x="RainTomorrow", y="WindGustSpeed")
```

```
<AxesSubplot:xlabel='RainTomorrow', ylabel='WindGustSpeed'>
```



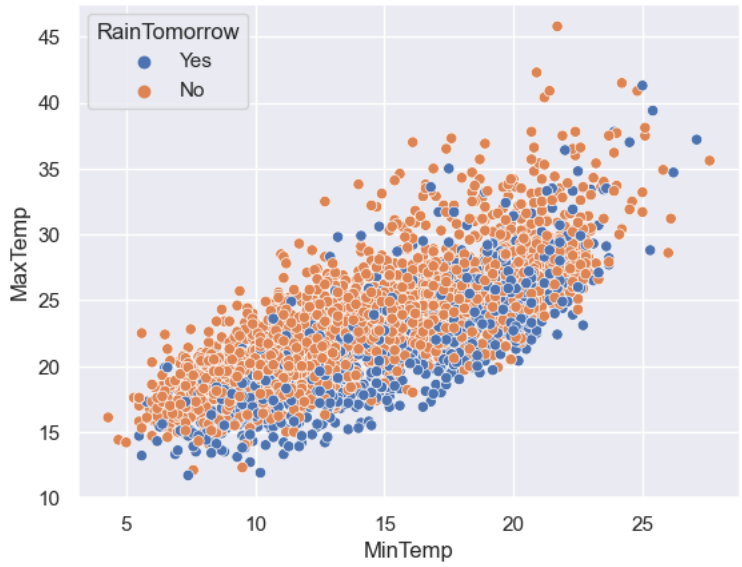

```
sns.displot(x='RainToday', hue='RainTomorrow', data=df, multiple='fill', stat="density")
```

```
<seaborn.axisgrid.FacetGrid at 0x1ed318d9040>
```



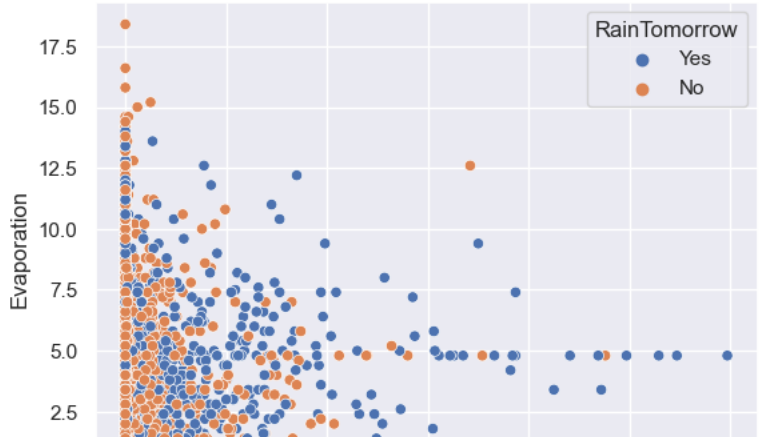

```
sns.scatterplot(data=df, x="MinTemp", y="MaxTemp", hue="RainTomorrow")
```

```
<AxesSubplot:xlabel='MinTemp', ylabel='MaxTemp'>
```




```
sns.scatterplot(data=df, x="Rainfall", y="Evaporation", hue="RainTomorrow")
```

```
<AxesSubplot:xlabel='Rainfall', ylabel='Evaporation'>
```



Data Preprocessing

Rainfall

```
df.dtypes
```

```
Date             datetime64[ns]
MinTemp                 float64
MaxTemp                 float64
Rainfall                float64
Evaporation             float64
Sunshine                float64
WindGustDir              object
WindGustSpeed             int64
WindDir9am               object
WindDir3pm               object
WindSpeed9am              int64
WindSpeed3pm              int64
Humidity9am               int64
Humidity3pm               int64
Pressure9am             float64
Pressure3pm             float64
Cloud9am                  int64
Cloud3pm                  int64
Temp9am                 float64
Temp3pm                 float64
RainToday                object
RainTomorrow             object
dtype: object
```

```
df['WindGustDir'].unique()
```

```
array(['W', 'NNW', 'WNW', 'ENE', 'NNE', 'NW', 'SSE', 'NE', 'ESE', 'WSW',
       'SE', 'SW', 'N', 'E', 'SSW', 'S'], dtype=object)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['WindGustDir']= label_encoder.fit_transform(df['WindGustDir'])
df['WindGustDir'].unique()
```

```
array([13,  6, 14,  1,  5,  7, 10,  4,  2, 15,  9, 12,  3,  0, 11,  8])
```

```
df['WindDir9am'].unique()
```

```
array(['S', 'W', 'ESE', 'NNE', 'SSW', 'WNW', 'N', 'SW', 'SE', 'SSE',
       'WSW', 'E', 'ENE', 'NW', 'NNW', 'NE'], dtype=object)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['WindDir9am']= label_encoder.fit_transform(df['WindDir9am'])
df['WindDir9am'].unique()
```

```
array([ 8, 13,  2,  5, 11, 14,  3, 12,  9, 10, 15,  0,  1,  7,  6,  4])
```

```
df['WindDir3pm'].unique()
```

```
array(['SSW', 'E', 'ESE', 'W', 'ENE', 'S', 'SE', 'SSE', 'NE', 'NNE',
       'NNW', 'NW', 'WNW', 'N', 'WSW', 'SW'], dtype=object)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['WindDir3pm']= label_encoder.fit_transform(df['WindDir3pm'])
df['WindDir3pm'].unique()
```

```
array([11,  0,  2, 13,  1,  8,  9, 10,  4,  5,  6,  7, 14,  3, 15, 12])
```

```
df['RainToday'].unique()
```

```
array(['Yes', 'No'], dtype=object)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['RainToday']= label_encoder.fit_transform(df['RainToday'])
df['RainToday'].unique()
```

```
array([1, 0])
```

```
df['RainTomorrow'].unique()
```

```
array(['Yes', 'No'], dtype=object)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['RainTomorrow']= label_encoder.fit_transform(df['RainTomorrow'])
df['RainTomorrow'].unique()
```

```
array([1, 0])
```

```
df.dtypes
```

```
Date             datetime64[ns]
MinTemp                 float64
MaxTemp                 float64
Rainfall                float64
Evaporation             float64
Sunshine                float64
WindGustDir               int32
WindGustSpeed             int64
WindDir9am                int32
WindDir3pm                int32
WindSpeed9am              int64
WindSpeed3pm              int64
Humidity9am               int64
Humidity3pm               int64
Pressure9am             float64
Pressure3pm             float64
Cloud9am                  int64
Cloud3pm                  int64
Temp9am                 float64
Temp3pm                 float64
RainToday                 int32
RainTomorrow              int32
dtype: object
```

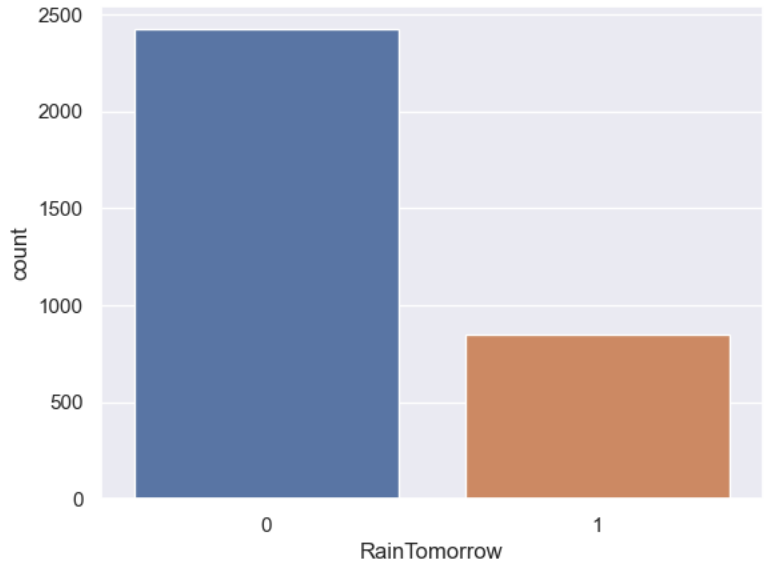
```
df.head()
```

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	Wir
1	2008-	19.5	25.6	6.0	3.4	2.7	13	41	13	

Check The Class Value if its Balanced or Not

```
#Counting 1 and 0 Value in Response column
sns.countplot(df['RainTomorrow'])
df['RainTomorrow'].value_counts()
```

```
D:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
  warnings.warn(
0    2422
1     849
Name: RainTomorrow, dtype: int64
```




```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df[(df['RainTomorrow']==0)]
df_minority = df[(df['RainTomorrow']==1)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,    # sample with replacement
                                 n_samples= 2422, # to match majority class
                                 random_state=0)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])


sns.countplot(df_upsampled['RainTomorrow'])
df_upsampled['RainTomorrow'].value_counts()
```

```
D:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
  warnings.warn(
1    2422
0    2422
Name: RainTomorrow, dtype: int64
```

Remove the Outlier using Z-Score

```
df_upsampled.drop(columns='Date', inplace=True)
df_upsampled.head()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDi
2656	15.0	24.3	0.0	5.6	6.4	13	41	14	
2166	14.3	20.7	2.4	1.0	7.5	5	31	14	
2451	20.0	31.4	0.0	5.2	8.0	10	72	13	
736	22.1	26.4	10.0	2.0	1.4	13	41	13	
3217	14.9	18.4	1.4	5.0	3.4	8	43	8	

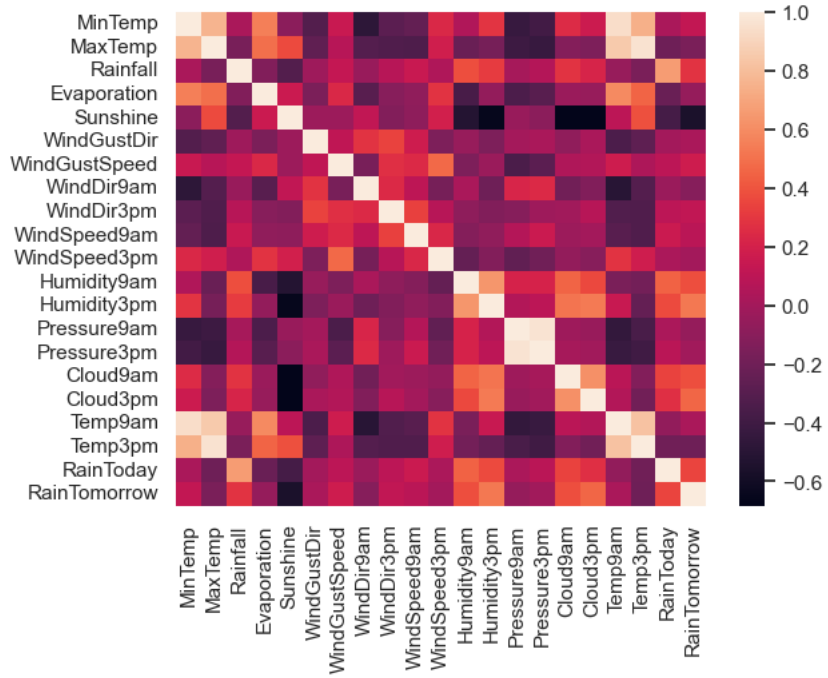
```
#Remove Outlier using Z-Score Method
import scipy.stats as stats
z = np.abs(stats.zscore(df_upsampled))
data_clean = df_upsampled[(z<3).all(axis = 1)]
data_clean.shape
```

```
(4556, 21)
```

Check the Correlation

```
sns.heatmap(data_clean.corr(), fmt='.2g')
```

```
<AxesSubplot:>
```



Training and Test Data

```
X = data_clean.drop('RainTomorrow', axis=1)
y = data_clean['RainTomorrow']
```

```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression(random_state=0)
```

```
y_pred = lr.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 77.85 %
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.7785087719298246
Precision Score :  0.7785087719298246
Recall Score :  0.7785087719298246
Jaccard Score :  0.6373429084380611
Log Loss :  7.650127181913046
```

K - Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = neigh.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 84.32 %
D:\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8432017543859649
Precision Score :  0.8432017543859649
Recall Score :  0.8432017543859649
Jaccard Score :  0.728909952606635
Log Loss :  5.415710726540535
```

Support Vector Machine

```
from sklearn import svm
support = svm.SVC()
support.fit(X_train, y_train)
```

```
SVC()
```

```
y_pred = support.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 74.12 %
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.7412280701754387
Precision Score :  0.7412280701754386
Recall Score :  0.7412280701754386
Jaccard Score :  0.5888501742160279
Log Loss :  8.937753496646865
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=0)
```

```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 89.04 %
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8903508771929824
Precision Score :  0.8903508771929824
Recall Score :  0.8903508771929824
Jaccard Score :  0.8023715415019763
Log Loss :  3.787210537394159
```

Confusion Matrix for All of Algorithms

```
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

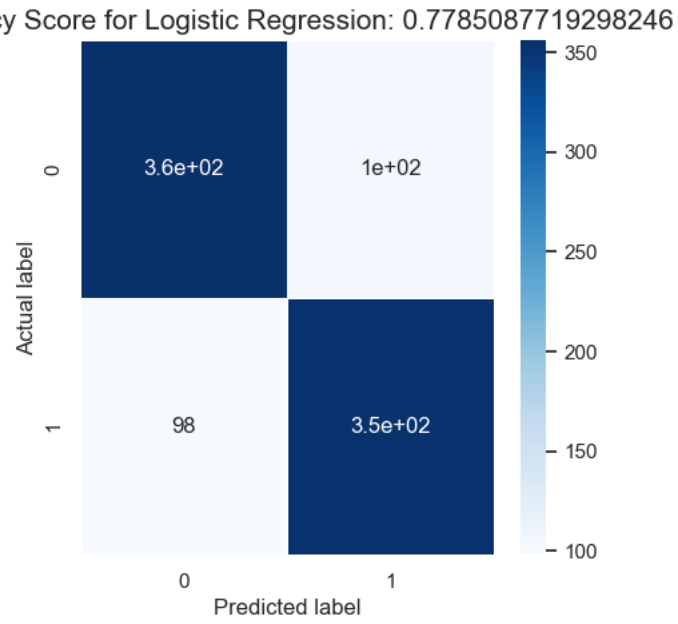
```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression(random_state=0)
```

```
y_pred = lr.predict(X_test)
```

+ Code + Text

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Logistic Regression: {0}'.format(lr.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

```
Text(0.5, 1.0, 'Accuracy Score for Logistic Regression: 0.7785087719298246')
```




```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
```

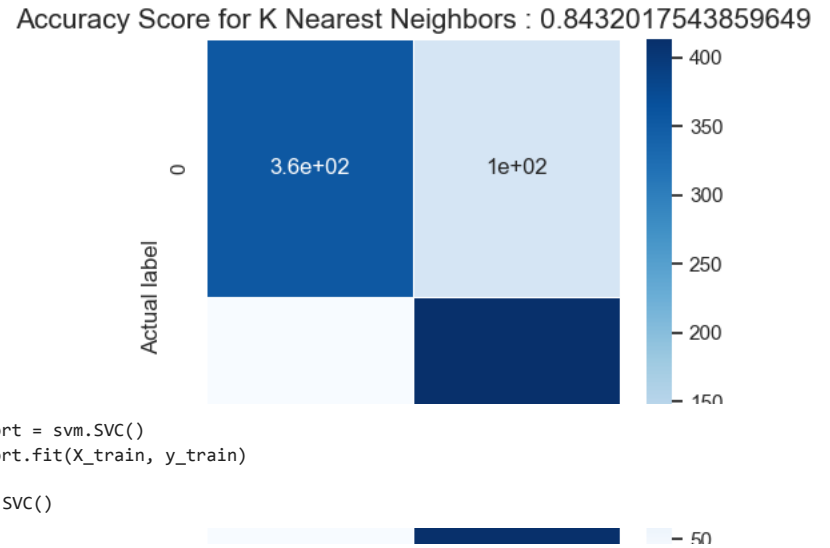
```
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = neigh.predict(X_test)
```

```
D:\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for K Nearest Neighbors : {0}'.format(neigh.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

```
D:\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other re
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
Text(0.5, 1.0, 'Accuracy Score for K Nearest Neighbors : 0.8432017543859649')
```



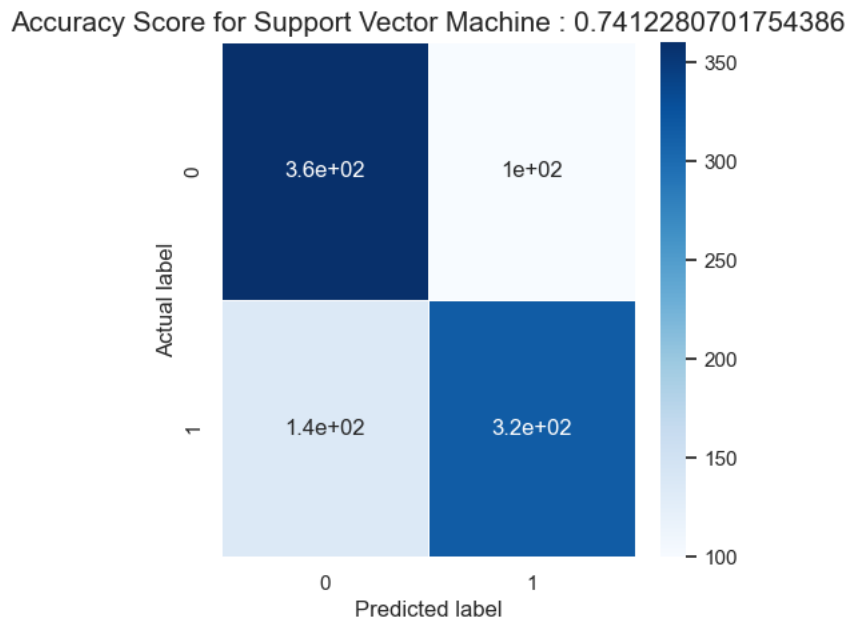

```
support = svm.SVC()
support.fit(X_train, y_train)
```

```
SVC()
```

```
y_pred = support.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Support Vector Machine : {0}'.format(support.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

```
Text(0.5, 1.0, 'Accuracy Score for Support Vector Machine : 0.7412280701754386')
```




```
dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train, y_train)
```

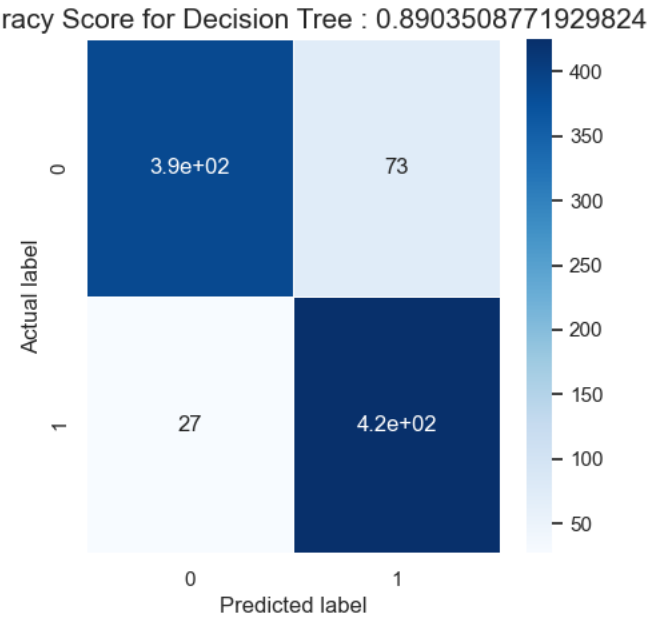
```
DecisionTreeClassifier(random_state=0)
```

```
y_pred = dtree.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
```

```
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree : {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree : 0.8903508771929824')
```



Feature Importance for Decision Tree

```
#Feature Importance
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)
fi
```

	Feature Name	Importance
4	Sunshine	0.315243

```
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```