Project Title : **Flight Delay Prediction**

Aim :

The aim of the project is to predict if the flight is Delayed or not

Team Members:

Pragadeeshwar N E0119003

Aswin G E0119062

Lohit Arun S E0119042

Importing libraries

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingCla
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import auc

import tensorflow as tf
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

Importing the dataset

```
flights = pd.read_csv('/content/drive/MyDrive/flights.csv')
flights
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_
0	2015	1	1	4	AS	98	N407AS	
1	2015	1	1	4	AA	2336	N3KUAA	
2	2015	1	1	4	US	840	N171US	
3	2015	1	1	4	AA	258	N3HYAA	
4	2015	1	1	4	AS	135	N527AS	
...	...	...	...	...	...	...	...	
5819074	2015	12	31	4	B6	688	N657JB	
5819075	2015	12	31	4	B6	745	N828JB	
5819076	2015	12	31	4	B6	1503	N913JB	
5819077	2015	12	31	4	B6	333	N527JB	
5819078	2015	12	31	4	B6	839	N534JB	

5819079 rows × 31 columns

Data Preprocessing

```
flights_needed_data = flights[0:100000]
flights_needed_data
```

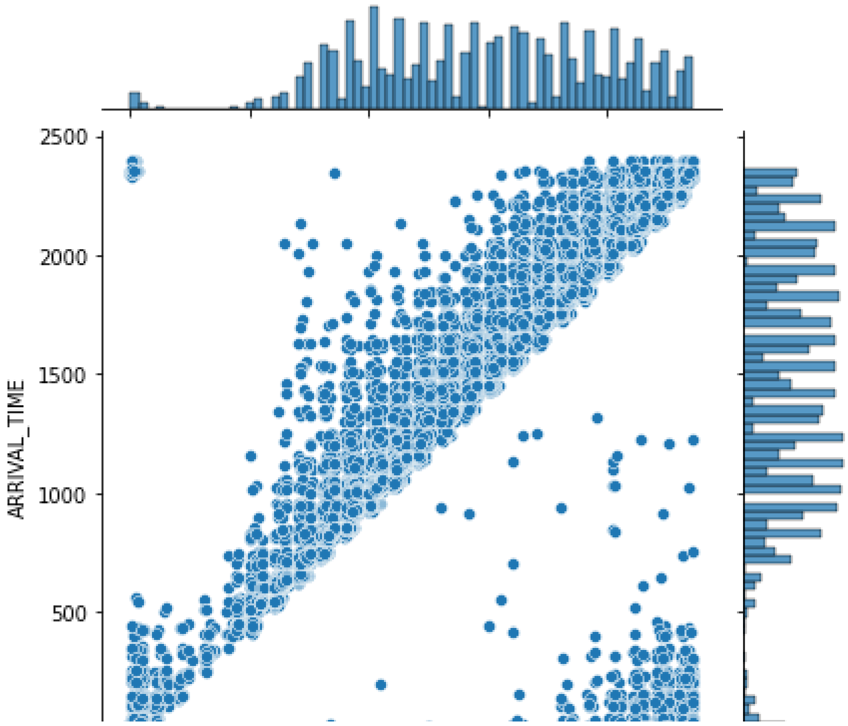
	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AI
0	2015	1	1	4	AS	98	N407AS	
1	2015	1	1	4	AA	2336	N3KUAA	
2	2015	1	1	4	US	840	N171US	
3	2015	1	1	4	AA	258	N3HYAA	

```
flights_needed_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 31 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   YEAR                 100000 non-null  int64  
 1   MONTH                100000 non-null  int64  
 2   DAY                  100000 non-null  int64  
 3   DAY_OF_WEEK          100000 non-null  int64  
 4   AIRLINE              100000 non-null  object 
 5   FLIGHT_NUMBER        100000 non-null  int64  
 6   TAIL_NUMBER          99833 non-null   object 
 7   ORIGIN_AIRPORT       100000 non-null  object 
 8   DESTINATION_AIRPORT  100000 non-null  object 
 9   SCHEDULED_DEPARTURE  100000 non-null  int64  
 10  DEPARTURE_TIME       97702 non-null   float64
 11  DEPARTURE_DELAY      97702 non-null   float64
 12  TAXI_OUT             97629 non-null   float64
 13  WHEELS_OFF           97629 non-null   float64
 14  SCHEDULED_TIME       100000 non-null  float64
 15  ELAPSED_TIME         97387 non-null   float64
 16  AIR_TIME             97387 non-null   float64
 17  DISTANCE             100000 non-null  int64  
 18  WHEELS_ON            97560 non-null   float64
 19  TAXI_IN              97560 non-null   float64
 20  SCHEDULED_ARRIVAL    100000 non-null  int64  
 21  ARRIVAL_TIME         97560 non-null   float64
 22  ARRIVAL_DELAY        97387 non-null   float64
 23  DIVERTED             100000 non-null  int64  
 24  CANCELLED            100000 non-null  int64  
 25  CANCELLATION_REASON  2389 non-null    object 
 26  AIR_SYSTEM_DELAY     34625 non-null   float64
 27  SECURITY_DELAY       34625 non-null   float64
 28  AIRLINE_DELAY        34625 non-null   float64
 29  LATE_AIRCRAFT_DELAY  34625 non-null   float64
 30  WEATHER_DELAY        34625 non-null   float64
dtypes: float64(16), int64(10), object(5)
memory usage: 23.7+ MB
```

```
sns.jointplot(data=flights_needed_data, x="SCHEDULED_ARRIVAL", y="ARRIVAL_TIME")
```

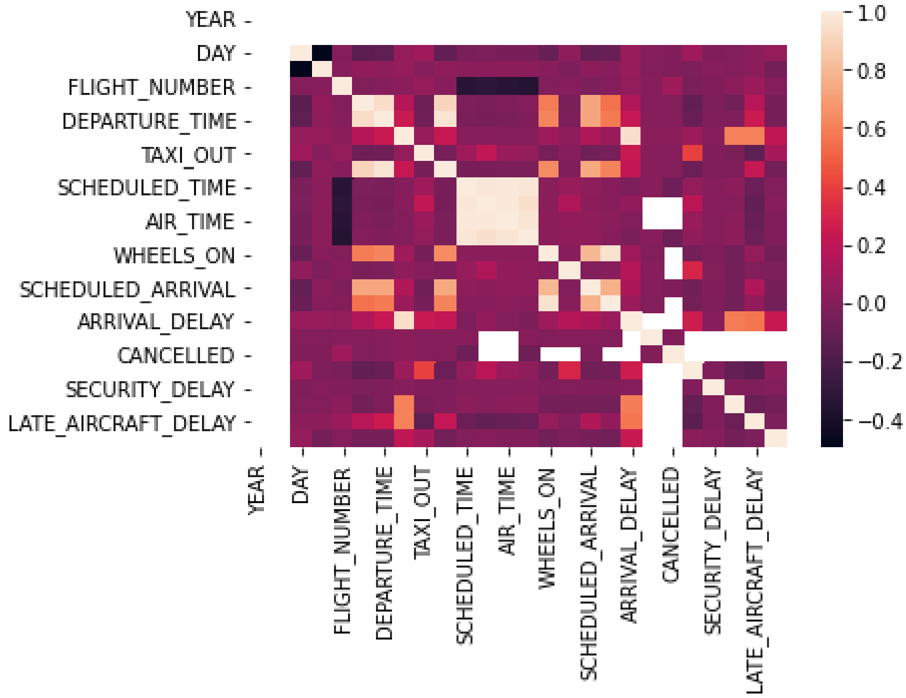
<seaborn.axisgrid.JointGrid at 0x7f8d617cf390>




```
corr = flights_needed_data.corr(method='pearson')
```

```
sns.heatmap(corr)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f8d57d6da50>




```
corr
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	FLIGHT_NUMBER	SCHEDULI
YEAR	NaN	NaN	NaN	NaN	NaN	
MONTH	NaN	NaN	NaN	NaN	NaN	
DAY	NaN	NaN	1.000000	-0.497084	0.004412	
DAY_OF_WEEK	NaN	NaN	-0.497084	1.000000	0.010955	
FLIGHT_NUMBER	NaN	NaN	0.004412	0.010955	1.000000	
SCHEDULED_DEPARTURE	NaN	NaN	-0.138130	0.046914	-0.003027	
DEPARTURE_TIME	NaN	NaN	-0.124369	0.045182	0.010140	
DEPARTURE_DELAY	NaN	NaN	0.060064	0.055632	0.034863	
TAXI_OUT	NaN	NaN	0.093451	0.007291	0.061010	
WHEELS_OFF	NaN	NaN	-0.119781	0.044150	0.016377	
SCHEDULED_TIME	NaN	NaN	-0.026285	0.019755	-0.337801	
ELAPSED_TIME	NaN	NaN	-0.018470	0.029025	-0.318819	
AIR_TIME	NaN	NaN	-0.036330	0.030678	-0.339135	
DISTANCE	NaN	NaN	-0.035208	0.024666	-0.356196	
WHEELS_ON	NaN	NaN	-0.095731	0.013749	-0.003670	
TAXI_IN	NaN	NaN	0.037407	-0.017789	0.014464	
SCHEDULED_ARRIVAL	NaN	NaN	-0.110820	0.031725	-0.018891	
ARRIVAL_TIME	NaN	NaN	-0.091687	0.011477	0.000753	
ARRIVAL_DELAY	NaN	NaN	0.070770	0.067520	0.056163	
DIVERTED	NaN	NaN	0.004847	-0.000709	0.007155	
CANCELLED	NaN	NaN	-0.006000	-0.006409	0.090008	
AIR_SYSTEM_DELAY	NaN	NaN	0.097693	-0.019626	-0.032564	
SECURITY_DELAY	NaN	NaN	-0.010550	0.008156	-0.007260	
AIRLINE_DELAY	NaN	NaN	-0.001603	0.003648	0.023770	
LATE_AIRCRAFT_DELAY	NaN	NaN	0.033213	0.033729	0.076581	

```
# filtering out unnecessary columns
flights_needed_data=flights_needed_data.drop(['YEAR','FLIGHT_NUMBER','AIRLINE','DISTANCE',
                                              'SCHEDULED_TIME','DEPARTURE_TIME','WHEELS_OF
                                              'AIR_TIME','WHEELS_ON','DAY_OF_WEEK','TAXI_I
                                             axis=1)
flights_needed_data
```

	MONTH	DAY	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTU
0	1	1	ANC	SEA	5	
1	1	1	LAX	PBI	10	
2	1	1	SFO	CLT	20	
3	1	1	LAX	MIA	20	
4	1	1	SEA	ANC	25	
...	...	...	...	...	...	
99995	1	7	ATL	BQK	1108	
99996	1	7	LAS	PHL	1108	
99997	1	7	SFO	BFL	1108	
99998	1	7	ORD	MCO	1109	
99999	1	7	HOU	DFW	1109	

100000 rows × 16 columns

```
# replacing all NaN values with the mean of the attribute in which they are present
flights_needed_data=flights_needed_data.fillna(flights_needed_data.mean())
flights_needed_data
```

	MONTH	DAY	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTU
0	1	1	ANC	SEA	5	
1	1	1	LAX	PBI	10	
2	1	1	SFO	CLT	20	
3	1	1	LAX	MIA	20	
4	1	1	SEA	ANC	25	
...	...	...	...	...	...	
99995	1	7	ATL	BQK	1108	
99996	1	7	LAS	PHL	1108	
99997	1	7	SFO	BFL	1108	
99998	1	7	ORD	MCO	1109	
99999	1	7	HOU	DFW	1109	

100000 rows × 16 columns

```
# creating a new column; it will tell if the flight was delayed or not
```

```
result=[]

for row in flights_needed_data['ARRIVAL_DELAY']:
  if row > 15:
    result.append(1)
  else:
    result.append(0)


flights_needed_data['result'] = result


flights_needed_data
```

	MONTH	DAY	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTU
0	1	1	ANC	SEA	5	
1	1	1	LAX	PBI	10	
2	1	1	SFO	CLT	20	
3	1	1	LAX	MIA	20	
4	1	1	SEA	ANC	25	
...	...	...	...	...	...	
99995	1	7	ATL	BQK	1108	
99996	1	7	LAS	PHL	1108	
99997	1	7	SFO	BFL	1108	
99998	1	7	ORD	MCO	1109	
99999	1	7	HOU	DFW	1109	

100000 rows × 17 columns

```
flights_needed_data.value_counts('result')
```

```
result
0    63779
1    36221
dtype: int64
```

```
# removing some more columns
flights_needed_data=flights_needed_data.drop(['ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', 'AR
flights_needed_data
```

	MONTH	DAY	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	DIVERTED
0	1	1	5	-11.0	430	0
1	1	1	10	-8.0	750	0
2	1	1	20	-2.0	806	0
3	1	1	20	-5.0	805	0
4	1	1	25	-1.0	320	0
...	...	...	...	...	...	...
99995	1	7	1108	-6.0	1219	0
99996	1	7	1108	9.0	1842	0
99997	1	7	1108	-7.0	1225	0
99998	1	7	1109	7.0	1454	0
99999	1	7	1109	-9.0	1220	0

Splitting the dataset into Train and Test data

```
data = flights_needed_data.values
X, y = data[:,:-1], data[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=202
```

```
# define a function to train and test the models
def train_model(model, X_train, y_train):
    model.fit(X_train, y_train)
    return model

def test_model(model,X_test,y_test):
  predictions = model.predict(X_test)
  accuracy = model.score(X_test, y_test)
  metrics_report = classification_report(y_test, predictions)
  precision, recall, fscore, train_support = score(y_test, predictions, average='weighted'
  return predictions, accuracy, metrics_report, (precision, recall, fscore)
```

Feature Scaling : Standardisation

```
scaled_features = StandardScaler()
X_train = scaled_features.fit_transform(X_train)
X_test = scaled_features.transform(X_test)
```

CO1 : Base Learners

Decision Tree :

```
# define the model
dt_model = DecisionTreeClassifier(random_state=42)

# fit the model
dt_model = train_model(dt_model, X_train, y_train)
print(dt_model)
```

```
DecisionTreeClassifier(random_state=42)
```

Logistic Regression:

```
# define the model
log_model = LogisticRegression(penalty='l2', max_iter=500)

# fit the model
log_model= train_model(log_model, X_train,y_train)
print(log_model)
```

```
LogisticRegression(max_iter=500)
```

Support Vector Machine:

```
svm_model= SVC(kernel = 'linear', random_state = 0)
svm_model = train_model(svm_model,X_train,y_train)
print(svm_model)
```

```
SVC(kernel='linear', random_state=0)
```

CO2 :Ensemble Learning :

Hard Voting

```
estimators = [('Decision Tree',dt_model),('Logistic Regression',log_model),('Support Vecto

ensemble_model = VotingClassifier(estimators=estimators,voting='hard')
ensemble_model = train_model(ensemble_model,X_train,y_train)
```

Gradient Boosting

```
# define the model
gb_model = GradientBoostingClassifier(n_estimators=50, max_depth=10, random_state=2)
```

```
# fit the model
gb_model = train_model(gb_model, X_train,y_train)
print(gb_model)
```

```
GradientBoostingClassifier(max_depth=10, n_estimators=50, random_state=2)
```

CO3 : Dimensionality Reduction

Principal Component Analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components=9)
X1_train = pca.fit_transform(X_train)
X1_test = pca.transform(X_test)
```

```
# define the model
log_model_pca = LogisticRegression(penalty='l2', max_iter=500)

# fit the model
log_model_pca= train_model(log_model_pca, X1_train,y_train)
print(log_model_pca)
```

```
LogisticRegression(max_iter=500)
```

CO4 : Artificial Neural Network

```
ann = tf.keras.models.Sequential()
```

```
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
```

```
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
```

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
ann.compile(optimizer = 'adam',loss = 'binary_crossentropy' ,metrics = ['accuracy'])
```

```
ann.fit(X1_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 73/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0039 - accuracy
Epoch 74/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0032 - accuracy
Epoch 75/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0032 - accuracy
Epoch 76/100
```

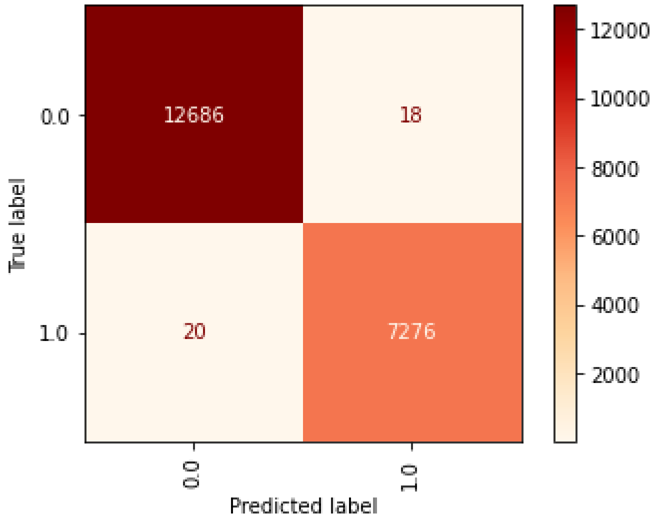
```
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0041 - accuracy
Epoch 77/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0027 - accuracy
Epoch 78/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0037 - accuracy
Epoch 79/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0028 - accuracy
Epoch 80/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0034 - accuracy
Epoch 81/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0031 - accuracy
Epoch 82/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0037 - accuracy
Epoch 83/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0028 - accuracy
Epoch 84/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0029 - accuracy
Epoch 85/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0039 - accuracy
Epoch 86/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0021 - accuracy
Epoch 87/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0035 - accuracy
Epoch 88/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0025 - accuracy
Epoch 89/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0036 - accuracy
Epoch 90/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0032 - accuracy
Epoch 91/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0041 - accuracy
Epoch 92/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0040 - accuracy
Epoch 93/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0039 - accuracy
Epoch 94/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0032 - accuracy
Epoch 95/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0033 - accuracy
Epoch 96/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0026 - accuracy
Epoch 97/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0041 - accuracy
Epoch 98/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0020 - accuracy
Epoch 99/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0043 - accuracy
Epoch 100/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.0020 - accuracy
<keras.callbacks.History at 0x7f8d40569650>
```

CO5: Comparison of Models

Decision Tree Prediction:

```
# Predicting the Test set results
from sklearn.metrics import accuracy_score
y_pred = dt_model.predict(X_test)
dt_accuracy = accuracy_score(y_pred,y_test)
metrics_report = classification_report(y_test, y_pred)
precision, recall, fscore, train_support = score(y_test, y_pred, average='weighted')
dt_prf =(precision, recall, fscore)
print('accuracy: {}'.format(dt_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(dt_model,X_test,y_test,xticks_rotation='vertical', cmap="OrRd")
```

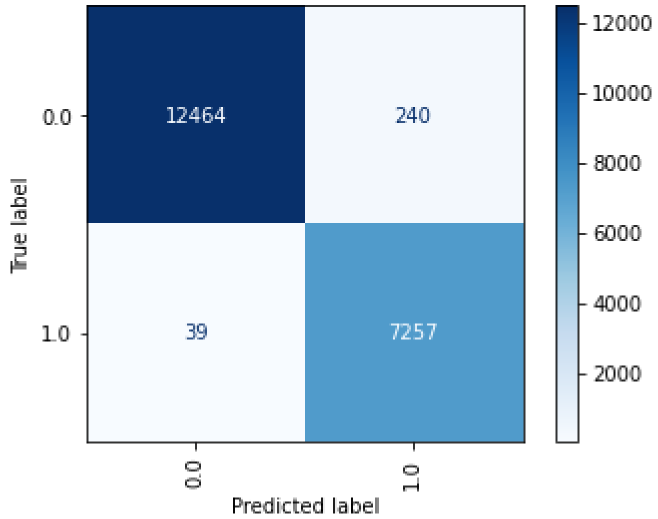
```
accuracy: 0.9981
=====================================================================================
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     12704
         1.0       1.00      1.00      1.00      7296

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d4046c950>
```



Support Vector Machine Prediction:

```
# Predicting the Test set results
from sklearn.metrics import accuracy_score
y_pred = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_pred,y_test)
metrics_report = classification_report(y_test, y_pred)
precision, recall, fscore, train_support = score(y_test, y_pred, average='weighted')
svm_prf =(precision, recall, fscore)
print('accuracy: {}'.format(svm_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(svm_model,X_test,y_test,xticks_rotation='vertical', cmap="Blues")
```

```
accuracy: 0.98605
====================================================================================================
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99     12704
         1.0       0.97      0.99      0.98      7296

    accuracy                           0.99     20000
   macro avg       0.98      0.99      0.99     20000
weighted avg       0.99      0.99      0.99     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d55f0e5d0>
```



Logistic Regression Prediction

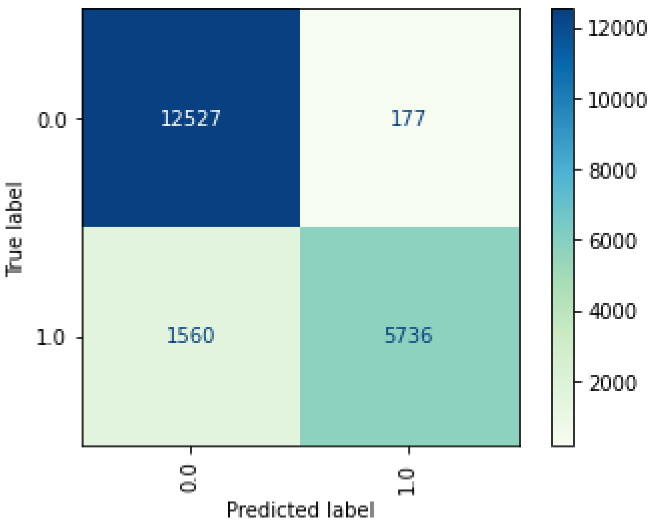
```
# Predicting the Test set results
from sklearn.metrics import accuracy_score
y_pred = log_model.predict(X_test)
log_accuracy = accuracy_score(y_pred,y_test)
metrics_report = classification_report(y_test, y_pred)
precision, recall, fscore, train_support = score(y_test, y_pred, average='weighted')
log_prf =(precision, recall, fscore)
print('accuracy: {}'.format(log_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(log_model,X_test,y_test,xticks_rotation='vertical', cmap="GnBu")
```

```
accuracy: 0.9848
====================================================================================================
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99     12704
         1.0       0.97      0.99      0.98      7296

    accuracy                           0.98     20000
   macro avg       0.98      0.99      0.98     20000
weighted avg       0.99      0.98      0.98     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d402c3310>
```

Logistic Regression after using PCA

```
# Predicting the Test set results
from sklearn.metrics import accuracy_score
y_pred = log_model_pca.predict(X1_test)
log_pca_accuracy = accuracy_score(y_pred,y_test)
metrics_report = classification_report(y_test, y_pred)
precision, recall, fscore, train_support = score(y_test, y_pred, average='weighted')
log_pca_prf =(precision, recall, fscore)
print('accuracy: {}'.format(log_pca_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(log_model_pca,X1_test,y_test,xticks_rotation='vertical', cmap="GnBu"
```

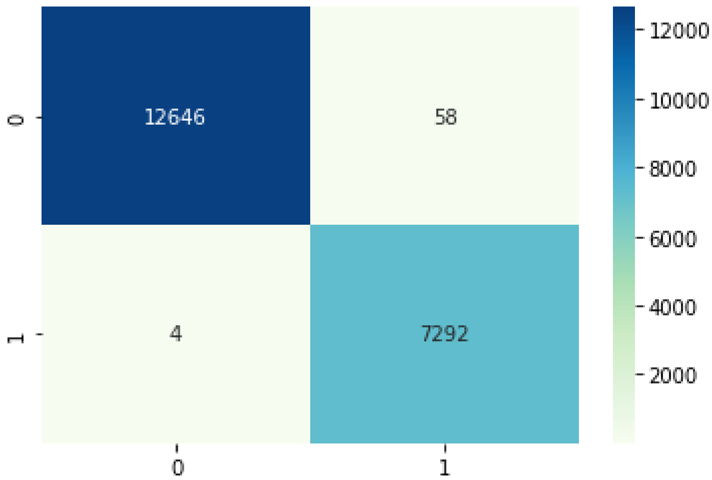
```
accuracy: 0.91315
====================================================================================================
              precision    recall  f1-score   support

         0.0       0.89      0.99      0.94     12704
         1.0       0.97      0.79      0.87      7296

    accuracy                           0.91     20000
   macro avg       0.93      0.89      0.90     20000
weighted avg       0.92      0.91      0.91     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d404d0dd0>
```



ANN Prediction

```
# Predicting the Test set results
from sklearn.metrics import accuracy_score
y_pred = ann.predict(X1_test)
y_pred = (y_pred > 0.5)
ann_accuracy = accuracy_score(y_pred,y_test)
metrics_report = classification_report(y_test, y_pred)
precision, recall, fscore, train_support = score(y_test, y_pred, average='weighted')
ann_prf =(precision, recall, fscore)
print('accuracy: {}'.format(ann_accuracy))
print('='*100)
print(metrics_report)
cm= confusion_matrix(y_test,y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="GnBu")
```

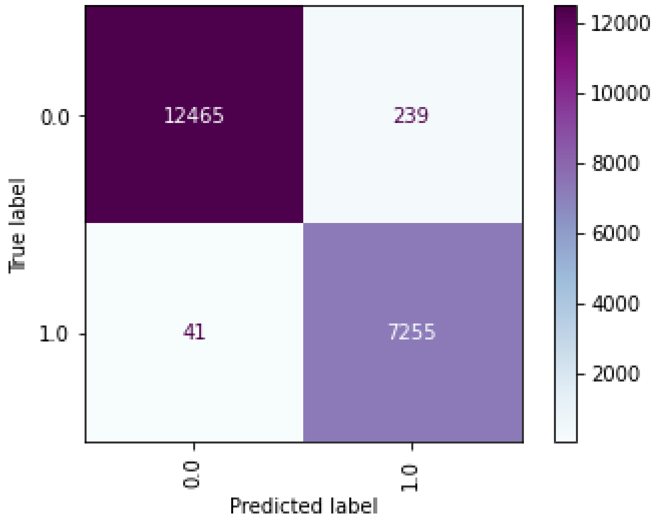
```
accuracy: 0.9969
==========================================================================================
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     12704
         1.0       0.99      1.00      1.00      7296

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

<matplotlib.axes._subplots.AxesSubplot at 0x7f8d4050bed0>
```



Hard Voting Classifier

```
predictions, hard_accuracy, metrics_report, hard_prf = test_model(ensemble_model, X_test,

print('accuracy: {}'.format(hard_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(ensemble_model, X_test,y_test, xticks_rotation='vertical', cmap="BuP
```

```
accuracy: 0.986
====================================================================================================
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99     12704
         1.0       0.97      0.99      0.98      7296

    accuracy                           0.99     20000
   macro avg       0.98      0.99      0.98     20000
weighted avg       0.99      0.99      0.99     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d4002b8d0>
```

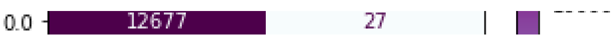


Gradient Boosting Classifier:

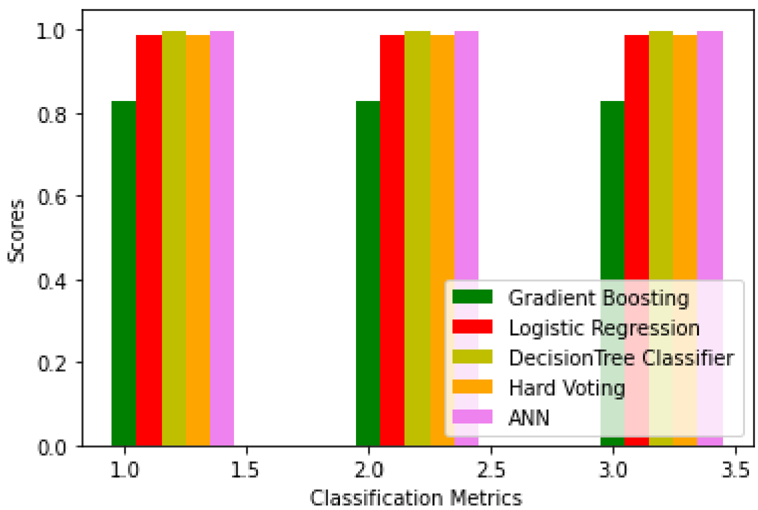
```
predictions, gb_accuracy, metrics_report, gb_prf = test_model(gb_model, X_test, y_test)

print('accuracy: {}'.format(gb_accuracy))
print('='*100)
print(metrics_report)
plot_confusion_matrix(gb_model, X_test, y_test, xticks_rotation='vertical', cmap="BuPu")
```

```
accuracy: 0.99855
==========================================================================================
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     12704
         1.0       1.00      1.00      1.00      7296

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8d401a4290>
```

CO5 : Model Evaluation




```
# classification matric indices
xx = np.array([1,2,3])
width = 0.1
gb_prf = np.array([0.83, 0.83, 0.83])
ax = plt.subplot(111)
ax.bar(xx ,height=np.array(gb_prf), width=width, color='g', align='center', label='Gradien
ax.bar(xx + 1*width, height=np.array(log_prf), width=width, color='r', align='center', lab
ax.bar(xx + 2*width, height=np.array(dt_prf), width=width, color='y', align='center', labe
ax.bar(xx + 3*width, height=np.array(hard_prf), width=width, color='orange', align='center
ax.bar(xx + 4*width, height=np.array(ann_prf), width=width, color='violet', align='center'


plt.xlabel('Classification Metrics')
plt.ylabel('Scores')
plt.legend(loc='lower right')
plt.show()
```




```
cl_metric = pd.DataFrame(data = {"Models":[],"Precision":[],"recall":[],"f1-score":[],"acc
prfs=[("Logistic Regression",log_prf),("Logistic Regression (PCA)",log_pca_prf),("Support 
names=[]
pre=[]
rec=[]
f1=[]
ac=[log_accuracy,log_pca_accuracy,svm_accuracy,dt_accuracy,ann_accuracy,hard_accuracy,gb_a
```

```
for name,(p,r,f) in prfs:
  names.append(name)
  pre.append(p)
  rec.append(r)
  f1.append(f)
cl_metric["Models"]=names
cl_metric["Precision"]=pre
cl_metric["recall"]=rec
cl_metric["f1-score"]=f1
cl_metric["accuracy_score"]=ac


print(cl_metric.to_string(index=False))
```

```
                 Models  Precision   recall  f1-score  accuracy_score
    Logistic Regression   0.985110  0.98480  0.984844         0.98480
 Logistic Regression (PCA)  0.918738  0.91315  0.910845         0.91315
 Support Vector Machine   0.986340  0.98605  0.986089         0.98605
          Decision Tree   0.998100  0.99810  0.998100         0.99810
                    ANN   0.996920  0.99690  0.996902         0.99690
            Hard Voting   0.986283  0.98600  0.986039         0.98600
      Gradient Boosting   0.830000  0.83000  0.830000         0.99855
```

Inference: Since we are convinced with the accuracy received from the Boosting model (Gradient) it was not necessary to use the Bagging and Stacking techniques for the prediction

So, from the above predictions it clearly indicates that the Gradient Boosting Model is much reliable in comparison with other models

0s completed at 1:35 PM