Data Decription

This dataset contains a survey on air passenger satisfaction. The following classification problem is set:

It is necessary to predict which of the two levels of satisfaction with the airline the passenger belongs to:

1-Satisfaction.

2-Neutral or dissatisfied.

Import Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
%matplotlib inline
from sklearn.preprocessing import StandardScaler,OneHotEncoder,LabelEncoder,OrdinalEncoder
from sklearn.metrics import confusion_matrix , roc_auc_score, roc_curve
```

```
from sklearn.model_selection import GridSearchCV , train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.tree import plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.naive_bayes import GaussianNB
```

Read Data File

```
df = pd.read_csv('Airline.csv')
```

```
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Depart time
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	

5 rows × 25 columns

Check Null

First, i want to check if there is null values in my data

```
df.isna().sum()
```

```
Unnamed: 0                            0
id                                    0
Gender                                0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Inflight wifi service                 0
Departure/Arrival time convenient     0
Ease of Online booking                0
Gate location                         0
Food and drink                        0
Online boarding                       0
Seat comfort                          0
Inflight entertainment                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Inflight service                      0
Cleanliness                           0
Departure Delay in Minutes            0
Arrival Delay in Minutes            310
satisfaction                          0
dtype: int64
```

Handle Null

Here we can see that the feature arrival delay in minutes have 310 null values which need to be handled. first, i will visualize the distribution of this feature

```
sns.histplot(data=df,x= 'Arrival Delay in Minutes')
plt.xlim(0,40)
```

(0.0, 40.0)

Here we can see that most of the values are 0 , so we will replace this feature with the median

```
df['Arrival Delay in Minutes'].fillna(value=df['Arrival Delay in Minutes'].median(),inplace=T
```

```
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Depart time
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	

5 rows × 25 columns

Data Cleaning

I will drop the Unnamed:0 columns because it represents the index of the data

```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

I will drop the id column because it doesnt make sense to affect the model

```
df.drop('id',axis=1,inplace=True)
```

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

```
array(['Male', 'Female'], dtype=object)
```

EDA

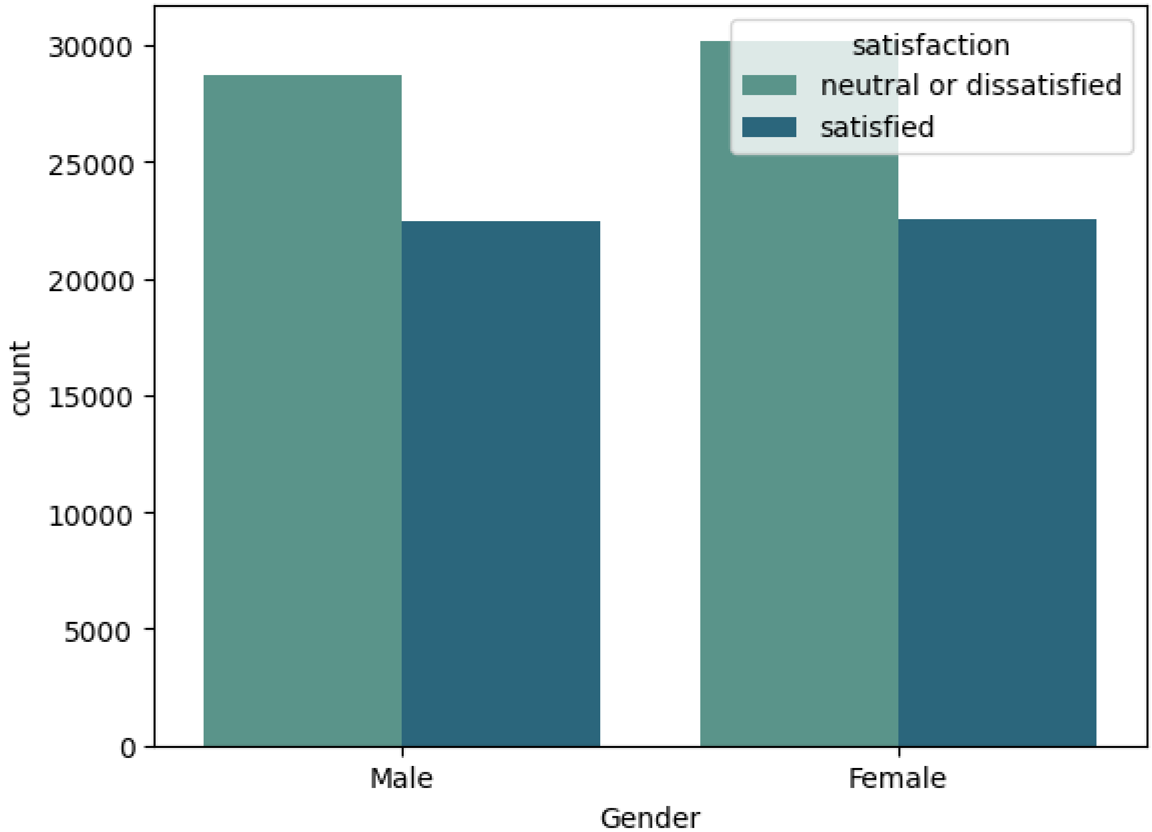
I want to visualise how many males and females are in my data

```
sns.countplot(data=df,x=df['Gender'],palette='crest')
```

```
<Axes: xlabel='Gender', ylabel='count'>
```

```
sns.countplot(df, x = "Gender", hue = "satisfaction", palette = "crest", linewidth = .5)
```

```
<Axes: xlabel='Gender', ylabel='count'>
```



Here we can see that the amount of male satisfied and unsatisfied are almost the same as female satisfied and unsatisfied

```
df['Gender'].value_counts()
```

```
Gender
Female    52727
Male      51177
Name: count, dtype: int64
```

As we can see, the number of females in our data is almost the same as the number of males

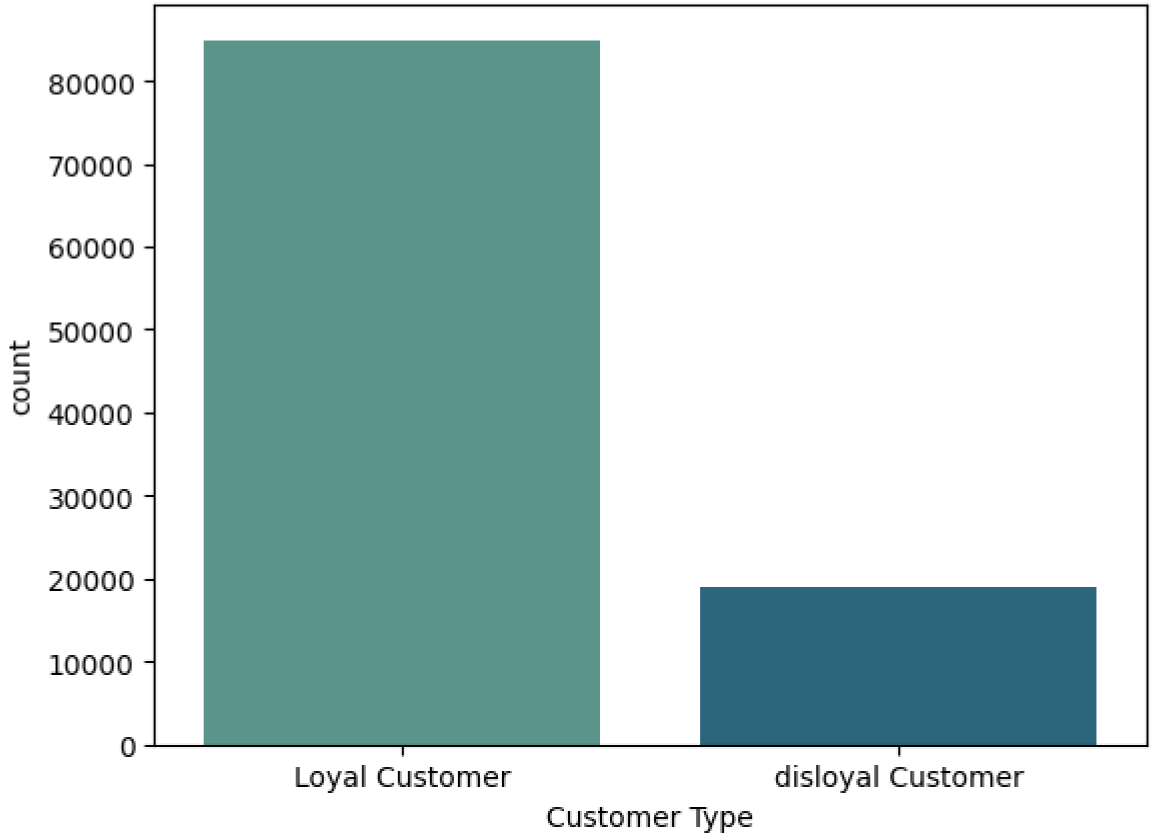
```
df['Customer Type'].unique()
```

```
array(['Loyal Customer', 'disloyal Customer'], dtype=object)
```

There are two types of customers: loyal and unloyal customers

```
sns.countplot(data=df,x= df['Customer Type'],palette='crest')
```

```
<Axes: xlabel='Customer Type', ylabel='count'>
```



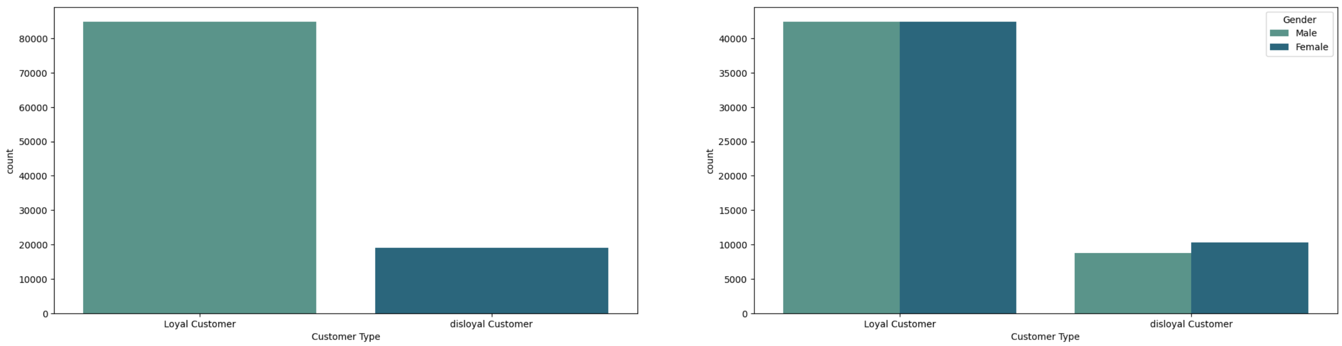

```
sns.countplot(df, x = "Customer Type", hue = "satisfaction", palette = "crest", linewidth = .
```

```
<Axes: xlabel='Customer Type', ylabel='count'>
```

Here we can see that the amount of satisfied loyal customers are thee same as unsatisfied loyal customers, While for unloyal customers tend to be unsatisfied or neutral rather than satisfied

```
f, axes = plt.subplots(1, 2)
f.set_size_inches(25, 6)
sns.countplot(x=df['Customer Type'],data=df,ax=axes[0],palette = "crest")
sns.countplot(data=df,x= df['Customer Type'],hue='Gender',ax=axes[1],palette='crest')
```

```
<Axes: xlabel='Customer Type', ylabel='count'>
```

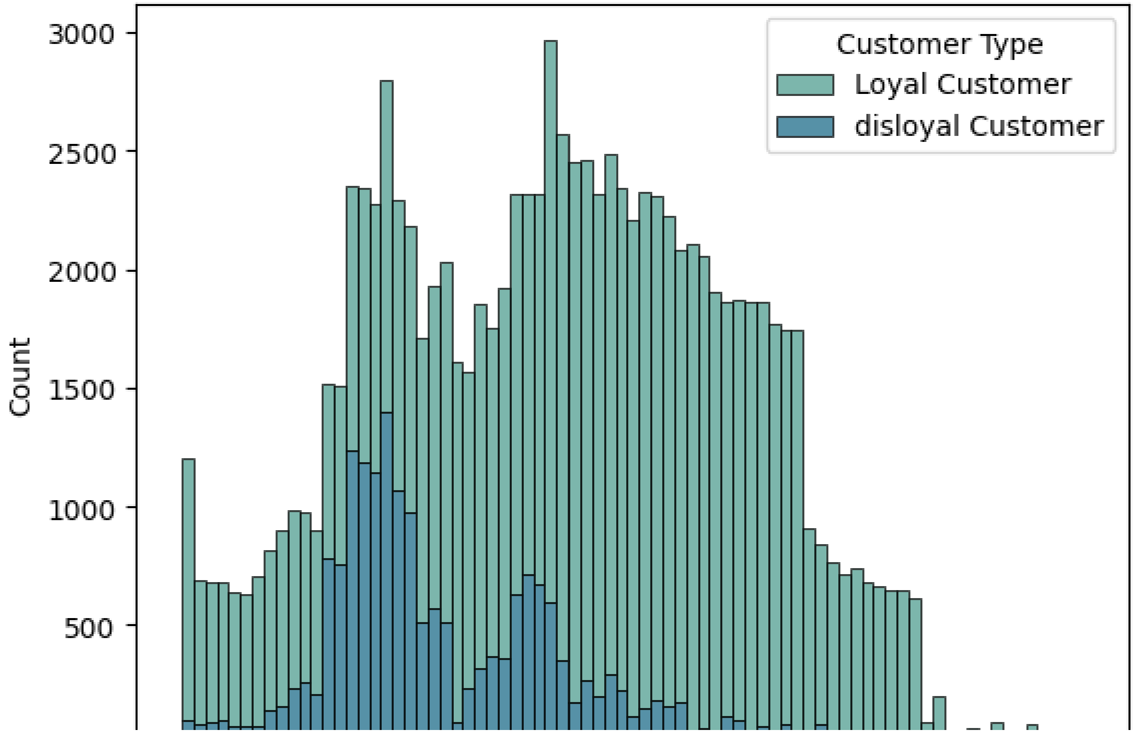


Most of the Customers Are Loyal

The visualization shows that the amount of male and female loyal customers are equal

```
sns.histplot(df, x = "Age", hue = "Customer Type", palette = "crest",multiple = "stack", line
```

```
<Axes: xlabel='Age', ylabel='Count'>
```



From this graph, We can see that most loyal customers occur from age 30 to 50, While most of unloyal customers age from 20 to 30

```
sns.histplot(df, x = "Age", hue = "Gender", palette = "crest",multiple = "stack", linewidth =
```

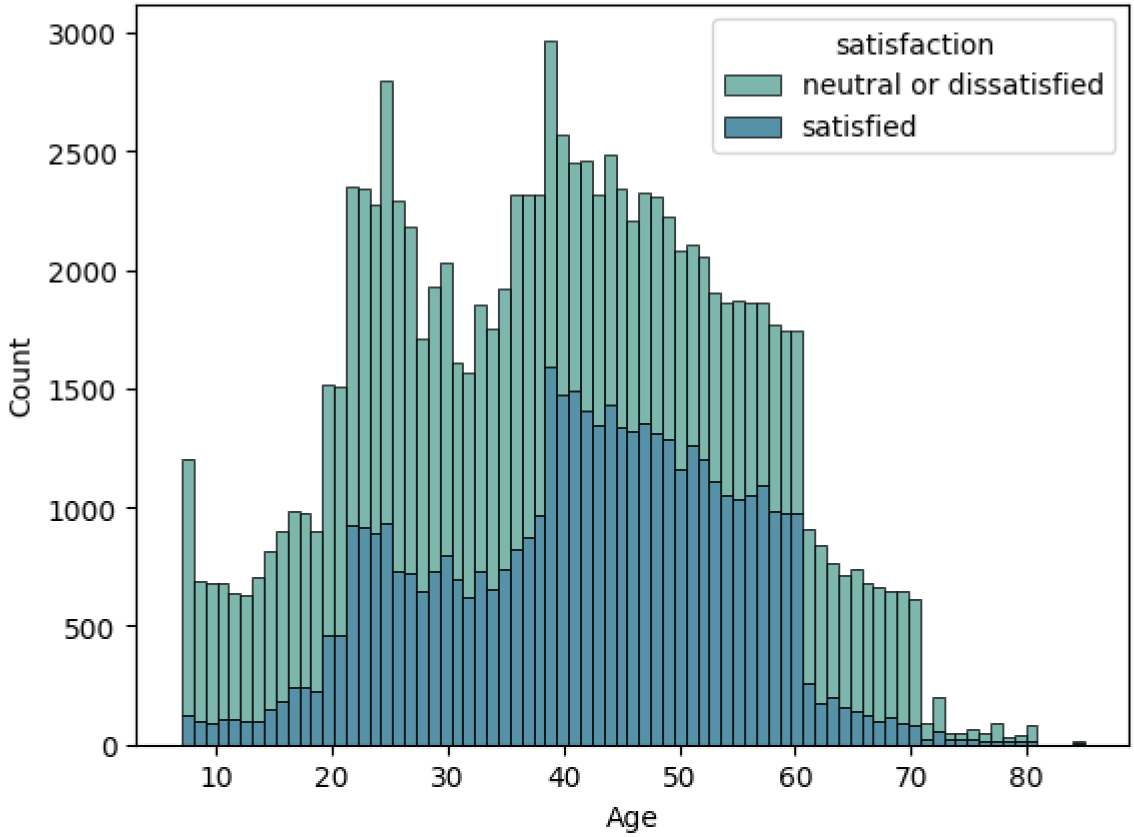
```
<Axes: xlabel='Age', ylabel='Count'>
```

Most of the Data in this model are concentrated between 20 and 60 years old almost equally distributed between male and female

```
#plt.pie(df.satisfaction.value_counts(), labels = ["Neutral or dissatisfied", "Satisfied"], c
```

```
sns.histplot(df, x = "Age", hue = "satisfaction", palette = "crest",multiple = "stack", linew
```

```
<Axes: xlabel='Age', ylabel='Count'>
```



Here we can visualize that neutral or unsatisfied people whose age are less than 25, are more than the satisfied

from age 25 to 40, the number of satisfied people are equal to the number of unsatisfied people

from the age 40 to 60, people tend to be more satisfied than neutral or unsatisfied

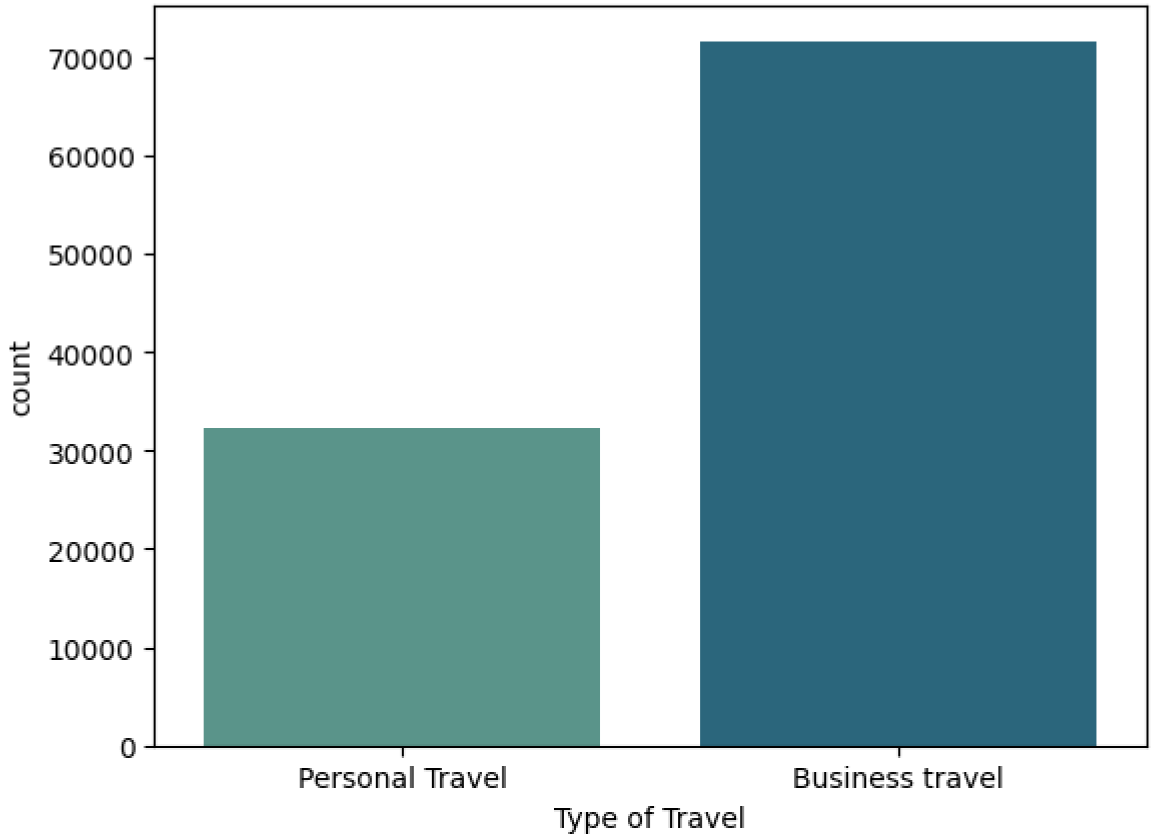
from above the age 60, people again tend to be more unsatified than people satisfied

```
df['Type of Travel'].unique()
```

```
array(['Personal Travel', 'Business travel'], dtype=object)
```

```
sns.countplot(x=df['Type of Travel'],data=df,palette = "crest")
```

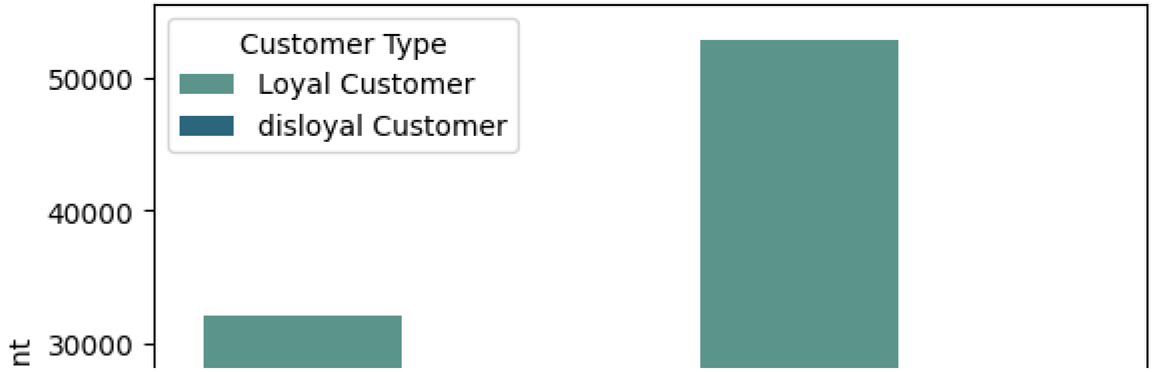
```
<Axes: xlabel='Type of Travel', ylabel='count'>
```



This graph shows that the amount of people using this airline for business travel more than people using it for perosnal travel

```
sns.countplot(x=df['Type of Travel'],hue='Customer Type',data=df,palette = "crest")
```

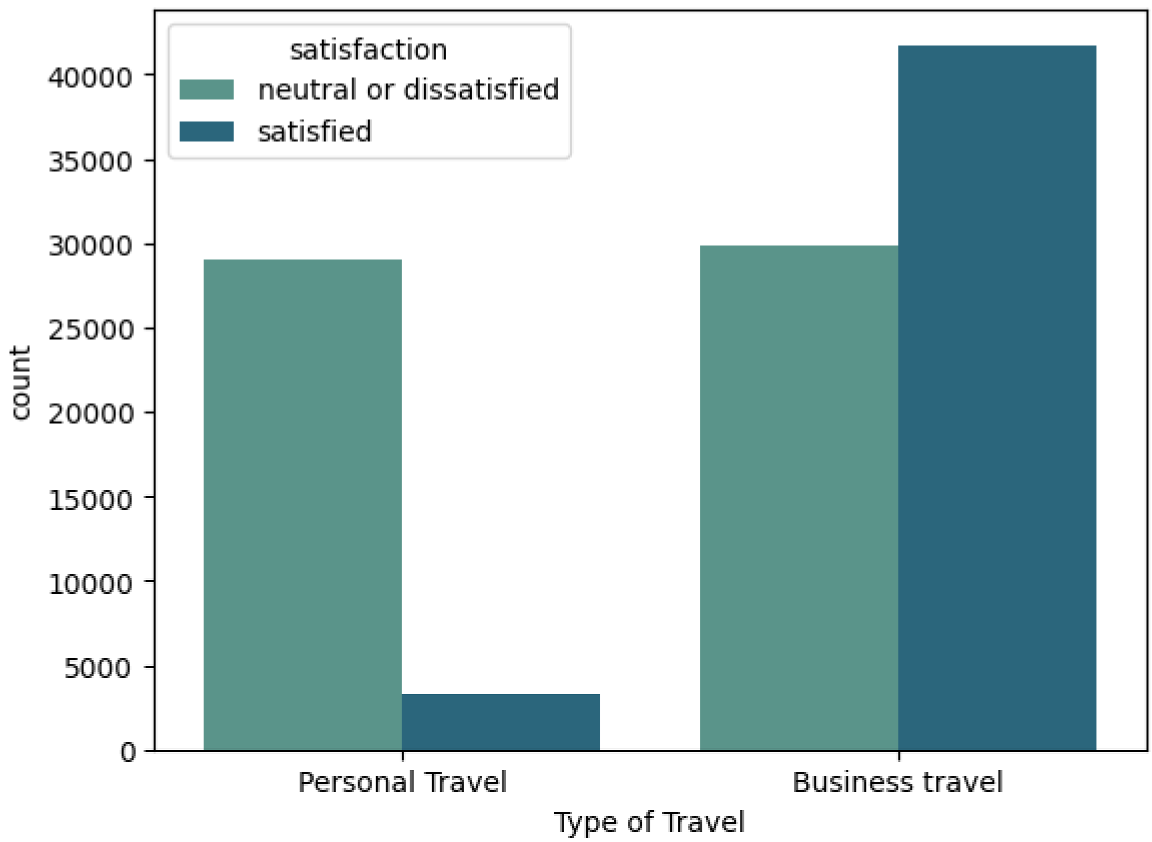
<Axes: xlabel='Type of Travel', ylabel='count'>



Here we can see that almost no disloyal customer is using this airline for personal travel. all of the disloyal customers using this plane use it for business travel

```
sns.countplot(x=df['Type of Travel'],hue='satisfaction',data=df,palette = "crest")
```

<Axes: xlabel='Type of Travel', ylabel='count'>



This plot shows that the number of dissatisfied peple using personal travel are much more than the satisfied people

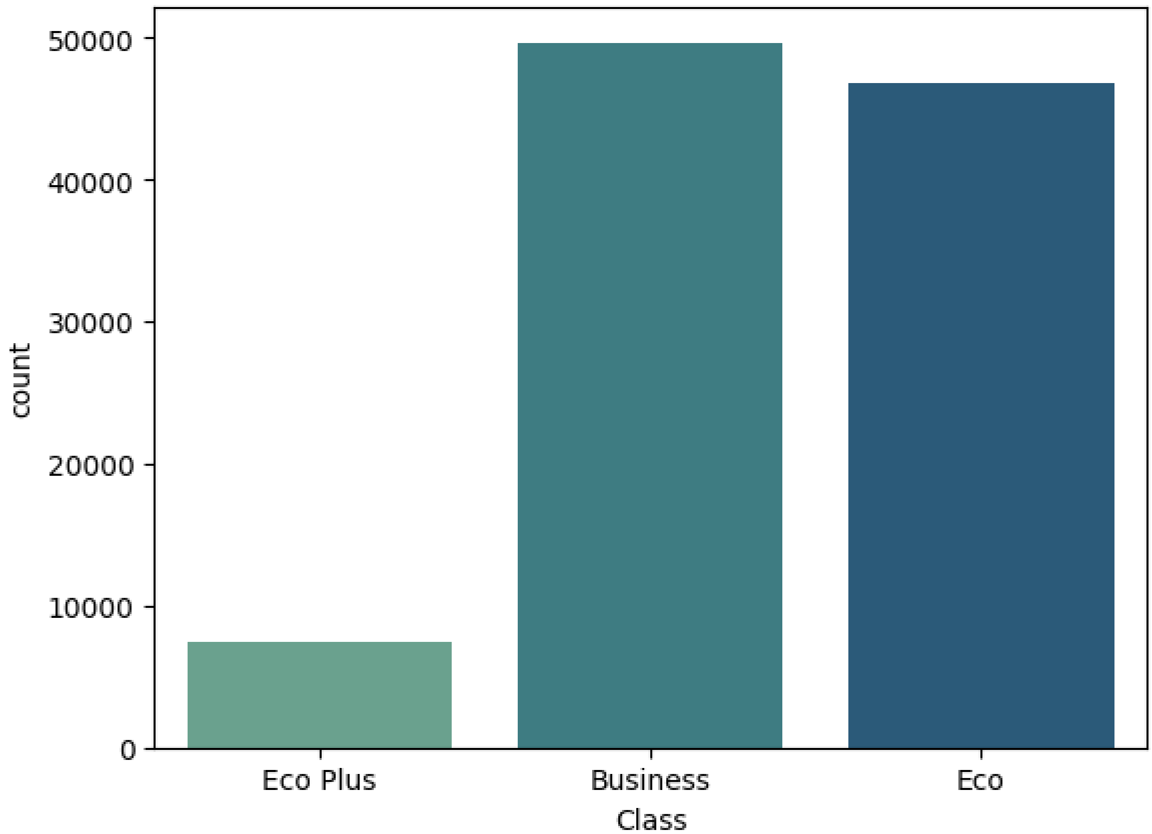
The plot also shows that people using this airline for business travel tend to be more satisfied

```
df['Class'].value_counts()
```

```
Class
Business    49665
Eco         46745
Eco Plus     7494
Name: count, dtype: int64
```

```
sns.countplot(x=df['Class'],data=df,palette = "crest")
```

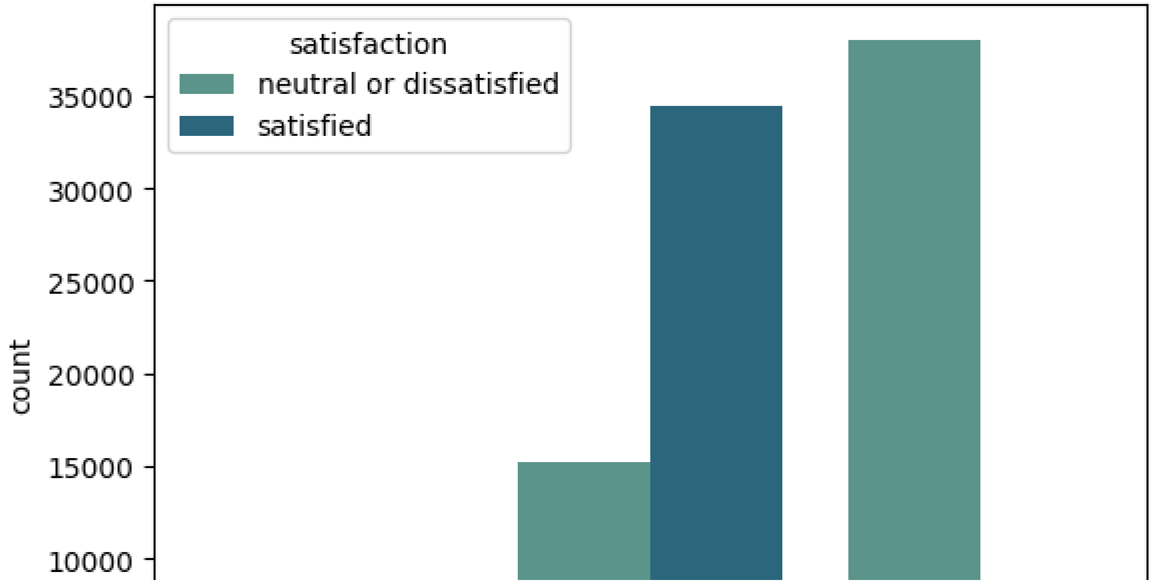
```
<Axes: xlabel='Class', ylabel='count'>
```



Most of the people use the business class or eco rather than eco plus which makes sense because the amount of eco plus seats are limited

```
sns.countplot(x=df['Class'],hue='satisfaction',data=df,palette = "crest")
```

```
<Axes: xlabel='Class', ylabel='count'>
```

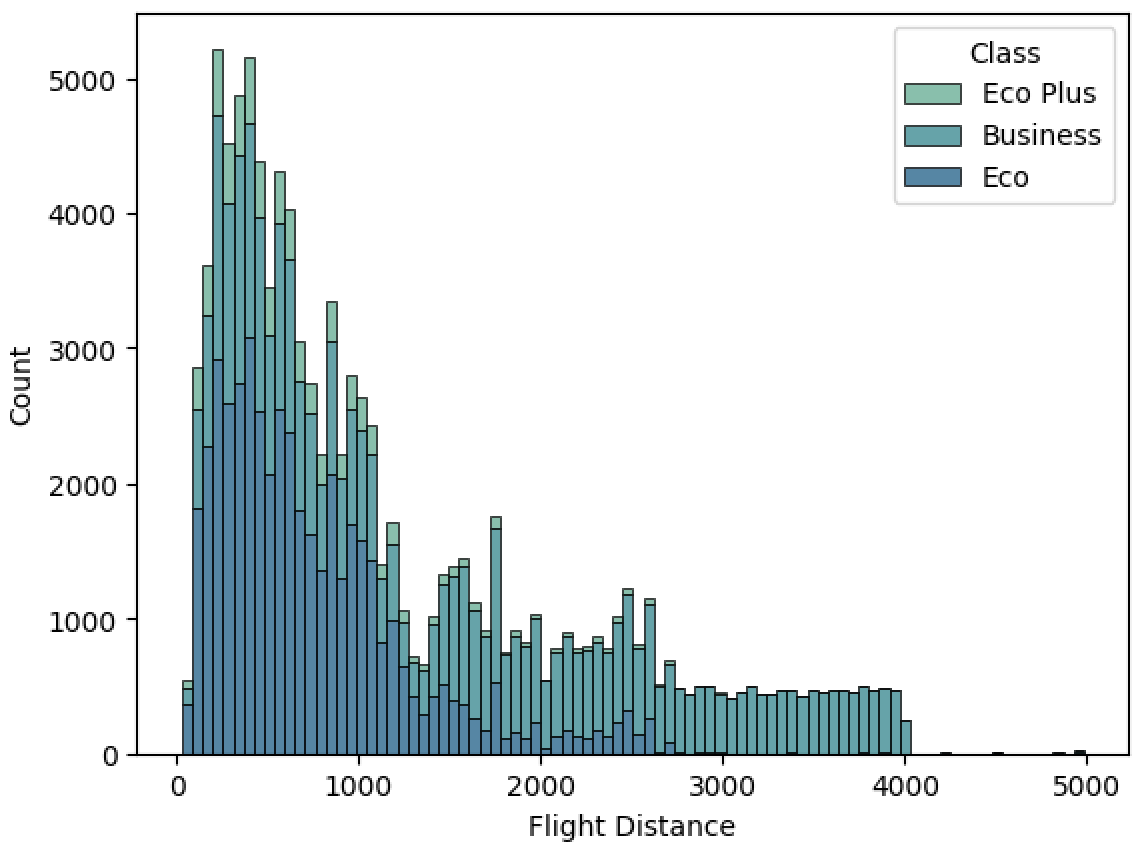


Here we can see that satisfied people from the business class are almost the double

However, we can also see that most people from the economy class are unsatisfied

```
sns.histplot(df, x = "Flight Distance", hue = "Class", multiple = "stack", palette = "crest",
```

```
<Axes: xlabel='Flight Distance', ylabel='Count'>
```



Encoding

Most people travelling short distance use economy class but as the distance increases people tend more to use business class

Now, I will make a function for the first type of encoding which is called One Hot Encoder. I will use it to classify binary categorical features

```
def ohe_transform(df,*args):
    ohe = OneHotEncoder()
    transf=ohe.fit_transform(df[[*args]])
    df_encoded = pd.DataFrame(transf.toarray(), columns=ohe.get_feature_names_out([*args]))
    print(df_encoded.shape)
    df = pd.concat([df, df_encoded], axis=1)

    return df
```

```
df = ohe_transform(df,'Type of Travel','Gender','Customer Type')
```

```
(103904, 6)
```

Now, After Assigning the encoding to new columns, I will delete the old columns

```
df.drop('Type of Travel',axis=1,inplace=True)
df.drop('Gender',axis=1,inplace=True)
df.drop('Customer Type',axis=1,inplace=True)
```

I will encode the class feature using ordinal encoding to give advantage to the business over the eco plus and eco class

```
def Class_transform(df,Class):
    oe = OrdinalEncoder(categories=[['Eco', 'Eco Plus', 'Business']])
    df[[Class]] = oe.fit_transform(df[[Class]])
    return df
```

```
df = Class_transform(df,'Class')
```

```
df['Class'].value_counts()
```

```
Class
2.0    49665
0.0    46745
1.0     7494
Name: count, dtype: int64
```

I will encode the satisfaction using the comprehensive list to make sure the output will be in one column only

```
df["satisfaction"]=[1 if i=="satisfied" else 0 for i in df["satisfaction"]]
df["satisfaction"].unique()
```

```
array([0, 1], dtype=int64)
```

Remove Outliers

I will remove the outliers from this dataset

```
for column in df.columns:

    if df[column].nunique() > 100:

        q1 = df[column].quantile(0.25)
        q3 = df[column].quantile(0.75)
        iqr = q3 - q1
        lower_whisker = q1 - 1.5 * iqr
        upper_whisker = q3 + 1.5 * iqr
        if lower_whisker < 0:
            lower_whisker = 0
        filt_lower = df[column] < lower_whisker
        filt_upper = df[column] > upper_whisker
        filt = filt_lower | filt_upper
        df = df.drop(df[filt].index, axis = 0)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 75372 entries, 1 to 103903
Data columns (total 26 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Age                                75372 non-null  int64
 1   Class                              75372 non-null  float64
 2   Flight Distance                    75372 non-null  int64
 3   Inflight wifi service              75372 non-null  int64
 4   Departure/Arrival time convenient  75372 non-null  int64
```

```
 5   Ease of Online booking             75372 non-null  int64
 6   Gate location                      75372 non-null  int64
 7   Food and drink                     75372 non-null  int64
 8   Online boarding                    75372 non-null  int64
 9   Seat comfort                       75372 non-null  int64
 10  Inflight entertainment             75372 non-null  int64
 11  On-board service                   75372 non-null  int64
 12  Leg room service                   75372 non-null  int64
 13  Baggage handling                   75372 non-null  int64
 14  Checkin service                    75372 non-null  int64
 15  Inflight service                   75372 non-null  int64
 16  Cleanliness                        75372 non-null  int64
 17  Departure Delay in Minutes         75372 non-null  int64
 18  Arrival Delay in Minutes           75372 non-null  float64
 19  satisfaction                       75372 non-null  int64
 20  Type of Travel_Business travel     75372 non-null  float64
 21  Type of Travel_Personal Travel     75372 non-null  float64
 22  Gender_Female                      75372 non-null  float64
 23  Gender_Male                        75372 non-null  float64
 24  Customer Type_Loyal Customer       75372 non-null  float64
 25  Customer Type_disloyal Customer    75372 non-null  float64
dtypes: float64(8), int64(18)
memory usage: 15.5 MB
```

Check Balanced Data

```
df['satisfaction'].value_counts()
```

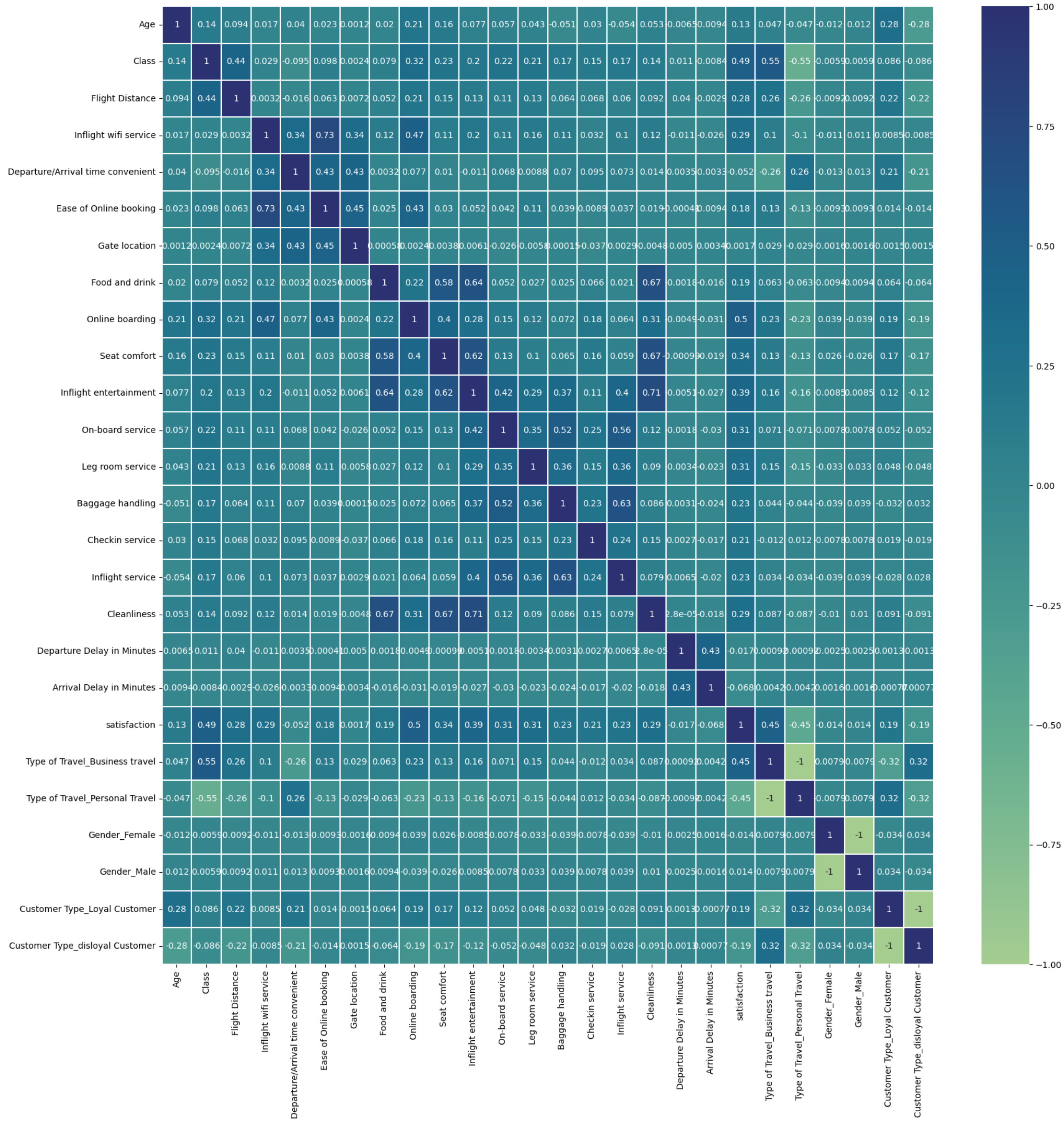
```
satisfaction
0    41306
1    34066
Name: count, dtype: int64
```

The data is almost balanced so no need to rebalance the data

Now, I will try to apply some feature selections techniques to extract the best features in the data

```
corr=df.corr()
plt.subplots(figsize=(20,20))
sns.heatmap(corr,cmap="crest",annot=True,linewidths=0.1)
```

<Axes: >



Splitting the Data

Now, I will split my model to x and y

```
x = df.drop('satisfaction',axis = 1)
y = df['satisfaction']
```

I will split my data to training, validation and test set

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_sta
```

Scaling the data

```
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test=scaler.transform(x_test)
```

Feature Selection

Since some of the features are numerical and other are categorical, I will use Anova to get the best features for my model

```
fsm=SelectKBest(f_classif,k=10)
fsm.fit(x_train,y_train)

x_train_selected=fsm.transform(x_train)
x_val_selected = fsm.transform(x_val)
x_test_selected=fsm.transform(x_test)
```

```
mask=fsm.get_support()
mask
```

```
array([False,  True, False,  True, False, False, False, False,  True,
        True,  True,  True,  True, False, False, False,  True, False,
       False,  True,  True, False, False, False, False])
```

```
selected_features_index=pd.DataFrame(x_train).columns[mask]
selected_features_index
```

```
Index([1, 3, 8, 9, 10, 11, 12, 16, 19, 20], dtype='int64')
```

```
df.drop('satisfaction',axis=1).columns[selected_features_index]
```

```
Index(['Class', 'Inflight wifi service', 'Online boarding', 'Seat comfort',
       'Inflight entertainment', 'On-board service', 'Leg room service',
       'Cleanliness', 'Type of Travel_Business travel',
       'Type of Travel_Personal Travel'],
      dtype='object')
```

Logistic Regression

Now, I will apply logistic regression model

```
LR_model = LogisticRegression()
LR_model.fit(x_train_selected,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Check Overfitting

```
y_val_predict = LR_model.predict(x_val_selected)
```

```
print(accuracy_score(y_val_predict,y_val))
```

```
0.8479270315091211
```

Evaluation

```
y_pred = LR_model.predict(x_test_selected)
LR_score = accuracy_score(y_pred,y_test)
LR_recall = recall_score(y_pred,y_test)
LR_precision = precision_score(y_pred,y_test)
LR_f1 = f1_score(y_pred,y_test)
```

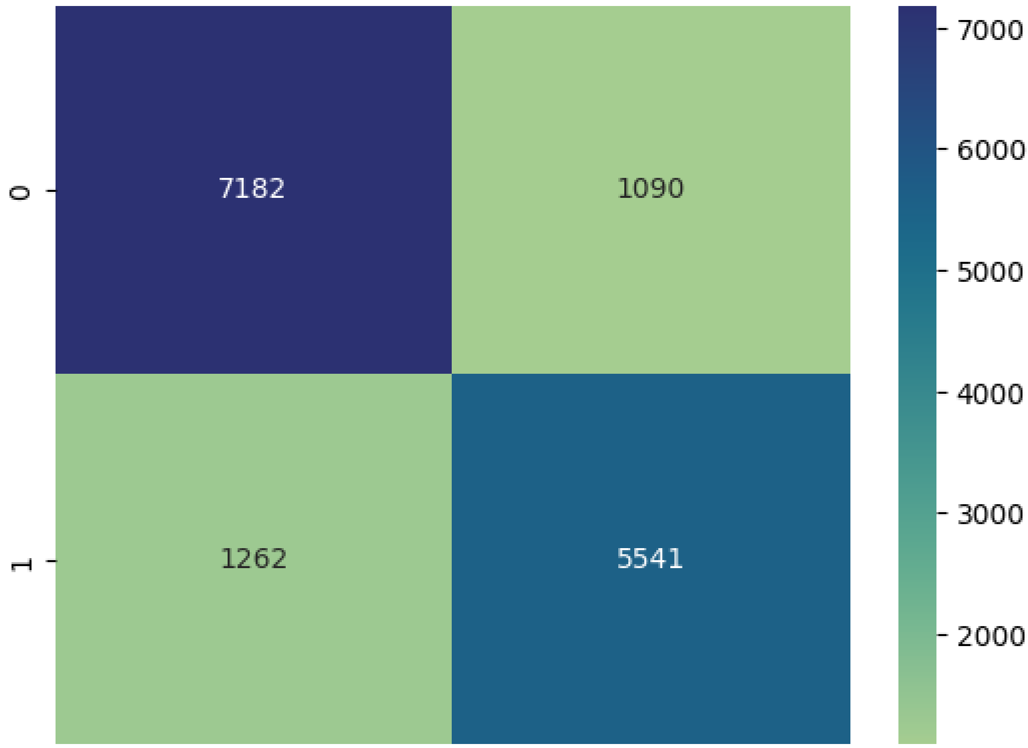
```
print(f'acurracy = {LR_score}')
print(f'recall = {LR_recall}')
print(f'precision = {LR_precision}')
print(f'f1 = {LR_f1}')
```

```
acurracy = 0.8439800995024875
recall = 0.8356205700497662
precision = 0.8144936057621638
f1 = 0.8249218401071907
```

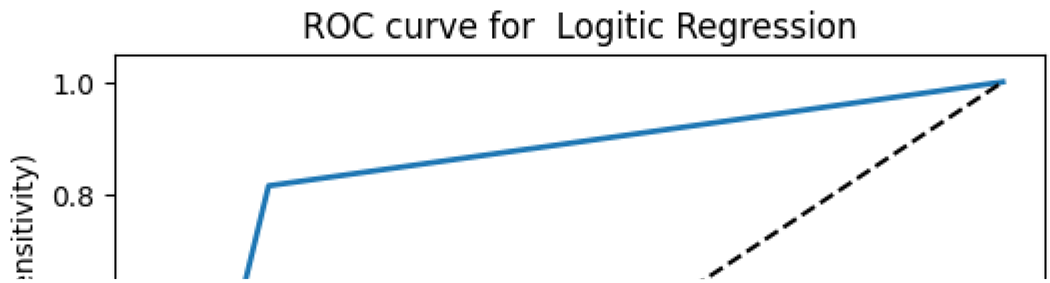
looks like no overfitting so we are all good

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```




```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.title('ROC curve for  Logitic Regression')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



Ummmmm, Not the best score. I think other complex models will be better

Decision Tree Classifier

```
DTC = DecisionTreeClassifier()
```

Grid Search

Let's do grid search to find the best hyper parameters

```
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
```

```
grid = GridSearchCV(
    estimator=DTC,
    param_grid=params,
    cv= 5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid.fit(x_train_selected,y_train)
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
grid.best_estimator_
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_leaf_nodes=99, min_samples_split=3)
```

```
DTC= DecisionTreeClassifier(max_leaf_nodes=99)
```

```
DTC.fit(x_train_selected,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_leaf_nodes=99)
```

Check Overfitting

```
y_val_predict = DTC.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.937893864013267
```

```
y_pred = DTC.predict(x_test_selected)
```

Evaluation

```
DTC_score = accuracy_score(y_pred,y_test)
DTC_recall = recall_score(y_pred,y_test)
DTC_precision = precision_score(y_pred,y_test)
DTC_f1 = f1_score(y_pred,y_test)
```

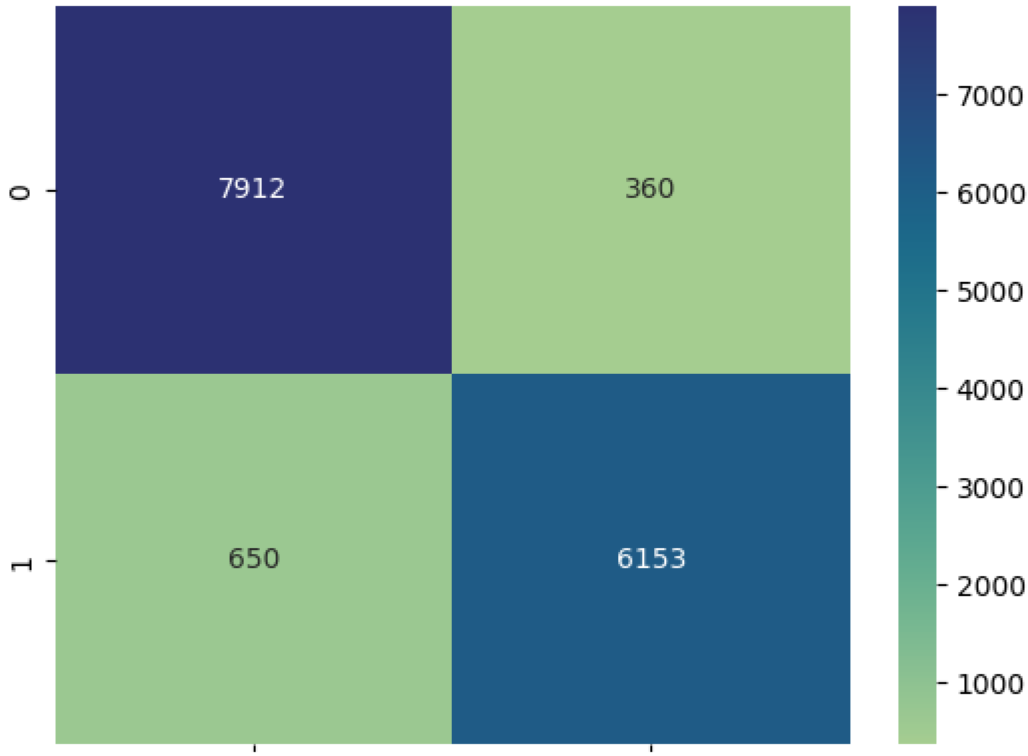
```
print(f'acurracy = {DTC_score}')
print(f'recall = {DTC_recall}')
print(f'precision = {DTC_precision}')
print(f'f1 = {DTC_f1}')
```

```
acurracy = 0.9330016583747927
recall = 0.9447259327498848
precision = 0.9044539173893871
f1 = 0.9241513968158608
```

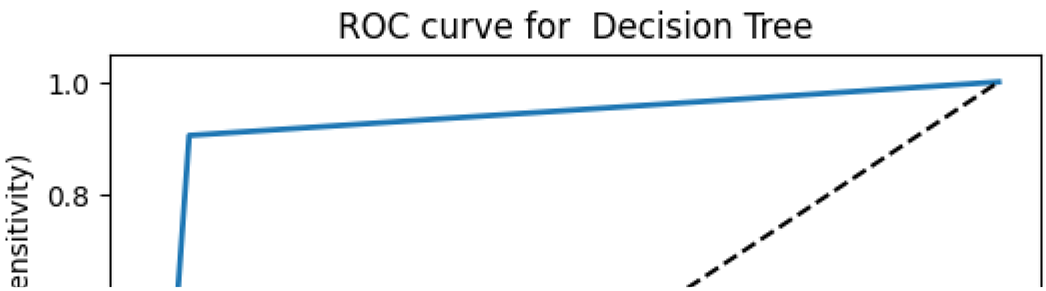
no overfitting

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```



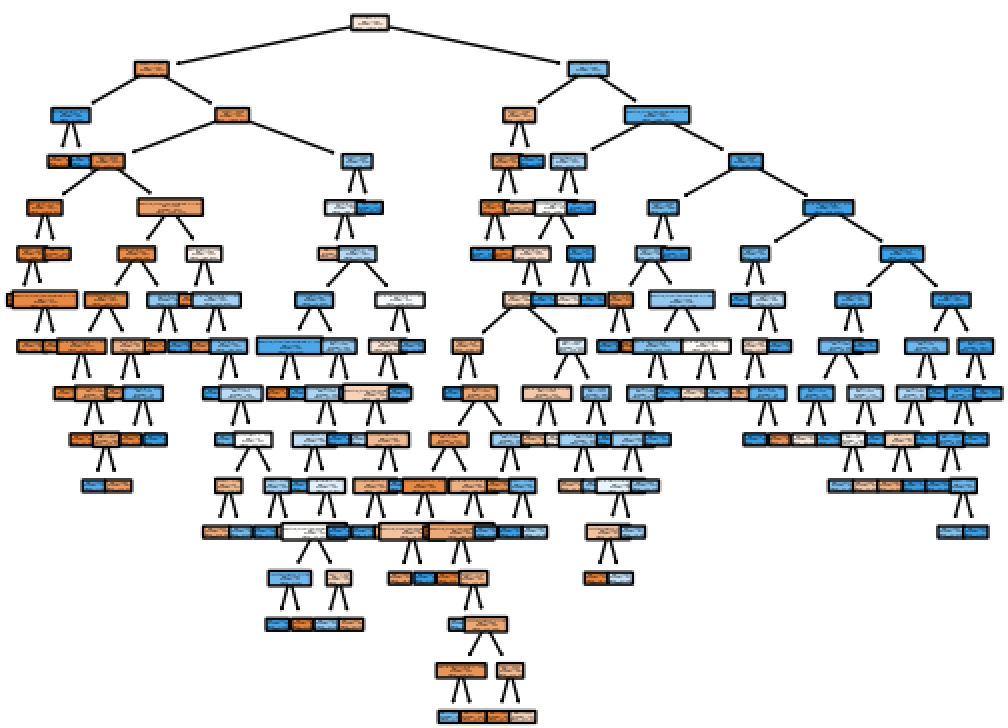

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.title('ROC curve for  Decision Tree')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



Much better choice but lets visualize the tree first

```
feature_name = [str(i) for i in x.columns]
class_name = df['satisfaction'].unique().astype(str).tolist()
```

```
plot_tree(
    DTC,
    feature_names=feature_name ,
    class_names= class_name,
    filled=True,
    rounded=True
)
plt.savefig('tree.png')
```



Uhhhhhh, Not cool. Right?

K Nearest Neighbour

```
KNN = KNeighborsClassifier()
params = {'n_neighbors': list(range(1,31))}
```

Grid Search

to find best hyperparameters

```
grid = GridSearchCV(
    estimator= KNN,
    param_grid=params,
    scoring='accuracy',
    n_jobs= -1,
    cv = 5
)
```

```
grid.fit(x_train_selected,y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
grid.best_params_
```

```
{'n_neighbors': 5}
```

```
KNN = KNeighborsClassifier(n_neighbors=5)
KNN.fit(x_train_selected,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Check Overfitting

```
y_val_predict = KNN.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9275290215588723
```

Evaluation

```
y_pred = KNN.predict(x_test_selected)
```

```
KNN_score = accuracy_score(y_pred,y_test)
KNN_recall = recall_score(y_pred,y_test)
KNN_precision = precision_score(y_pred,y_test)
KNN_f1 = f1_score(y_pred,y_test)
```

```
print(f'acurracy = {KNN_score}')
print(f'recall = {KNN_recall}')
print(f'precision = {KNN_precision}')
print(f'f1 = {KNN_f1}')
```

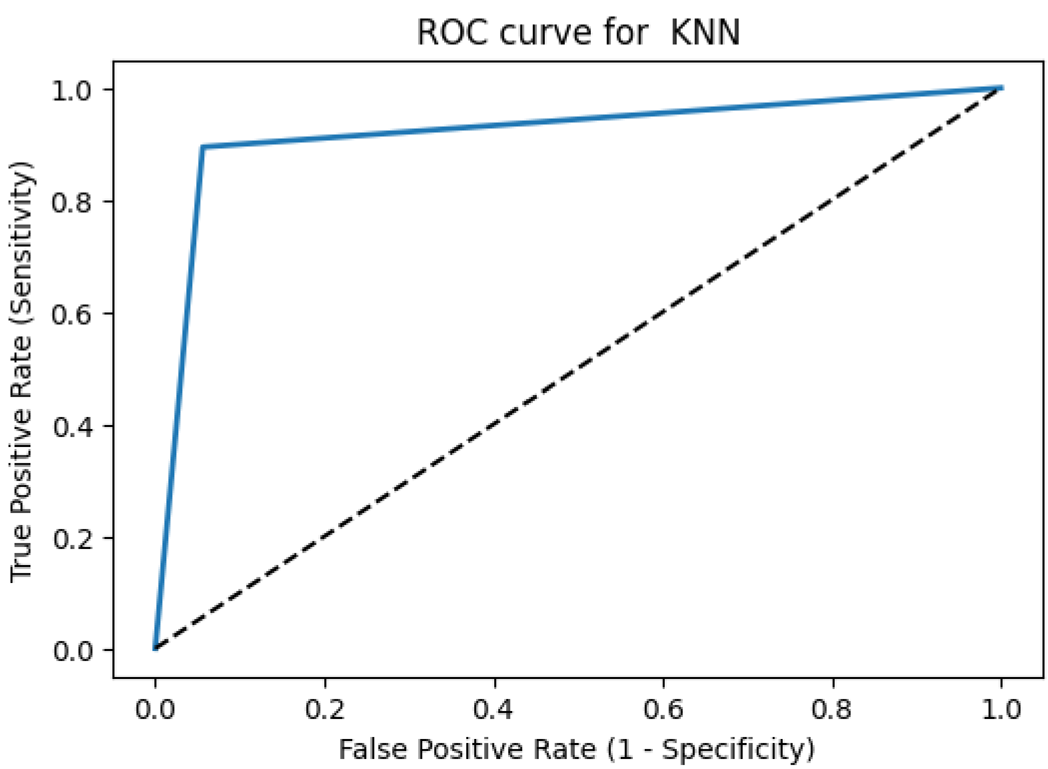
```
acurracy = 0.9215257048092869
recall = 0.9290076335877863
precision = 0.8944583272085844
f1 = 0.9114056766269752
```

no overfitting

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```


```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.title('ROC curve for  KNN')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



Support Vector Classifier

```
model = SVC()
```

Grid Search

```
param_distributions = {
  'C':  [0.1, 1, 10],
  'kernel': ['linear', 'rbf'],
  'gamma': ['scale', 'auto']
}
```

```
grid = GridSearchCV(
    estimator=model,
    param_grid=param_distributions,
    cv= 5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid.fit(x_train_selected,y_train)
```

```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

```
grid.best_estimator_
```

```
▾ SVC
SVC(C=10, gamma='auto')
```

```
model = SVC(C=10,gamma='auto')
```

```
model.fit(x_train_selected,y_train)
```

```
▾ SVC
SVC(C=10, gamma='auto')
```

Check Overfitting

```
y_val_predict = model.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.936318407960199
```

Evaluation

```
y_pred = model.predict(x_test_selected)
SVC_score = accuracy_score(y_pred,y_test)
SVC_recall = recall_score(y_pred,y_test)
SVC_precision = precision_score(y_pred,y_test)
SVC_f1 = f1_score(y_pred,y_test)


print(f'acurracy = {SVC_score}')
print(f'recall = {SVC_recall}')
print(f'precision = {SVC_precision}')
print(f'f1 = {SVC_f1}')
```

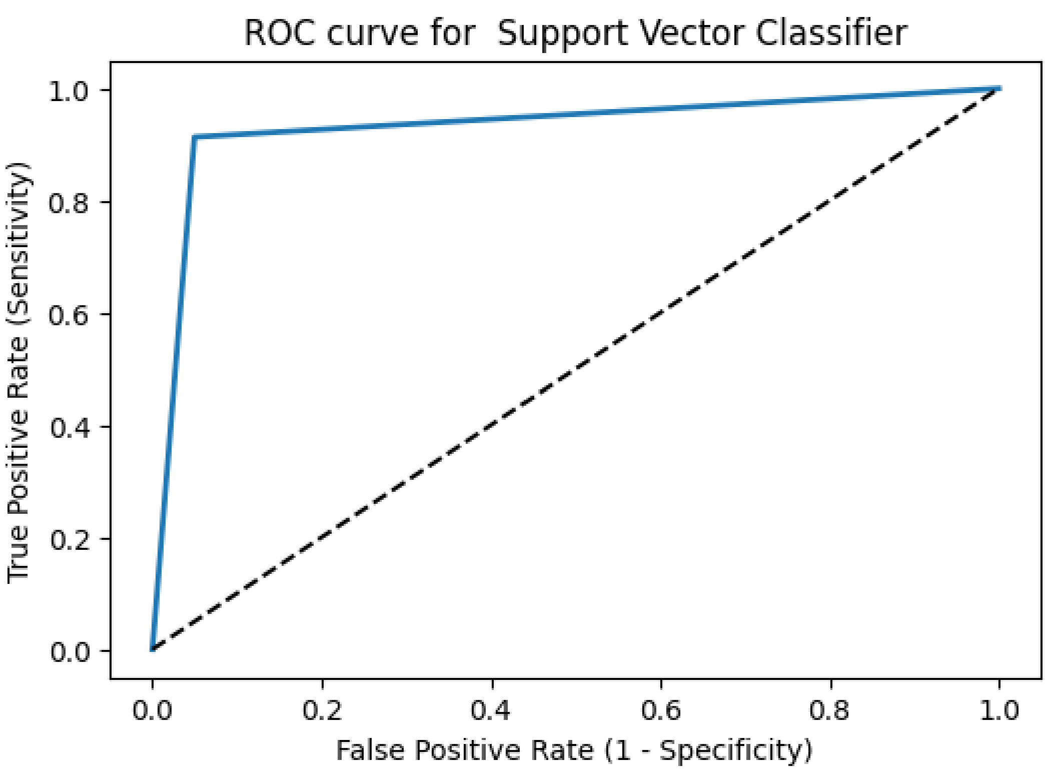
```
acurracy = 0.9335986733001659
recall = 0.9379528985507246
precision = 0.9132735557842129
f1 = 0.9254487227228718
```

no overfitting

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```


```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.title('ROC curve for  Support Vector Classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



Bagging

Now lets try the bagging classifier method with decision tree

```
BagClf = BaggingClassifier()
```

Grid Search

```
params = { 'n_estimators':[20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]}
```

```
grid = GridSearchCV(
    estimator=BagClf,
    param_grid=params,
    cv= 5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid.fit(x_train_selected,y_train)
```

```
GridSearchCV
estimator: BaggingClassifier
BaggingClassifier
```

```
grid.best_estimator_
```

```
BaggingClassifier
BaggingClassifier(n_estimators=600)
```

```
bag_class=BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=600,
    bootstrap=True,
    n_jobs=-1
)
```

```
bag_class.fit(x_train_selected,y_train)
```

```
C:\Users\Youssef\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8
```

Check overfitting

```
▸ DecisionTreeClassifier
```

```
y_val_predict = bag_class.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9371475953565506
```

Evaluation

```
y_pred = bag_class.predict(x_test_selected)
Bag1_score = accuracy_score(y_pred,y_test)
Bag1_recall = recall_score(y_pred,y_test)
Bag1_precision = precision_score(y_pred,y_test)
Bag1_f1 = f1_score(y_pred,y_test)


print(f'acurracy = {Bag1_score}')
print(f'recall = {Bag1_recall}')
print(f'precision = {Bag1_precision}')
print(f'f1 = {Bag1_f1}')
```

```
acurracy = 0.9337313432835821
recall = 0.931460005947071
precision = 0.9209172423930618
f1 = 0.9261586222189371
```

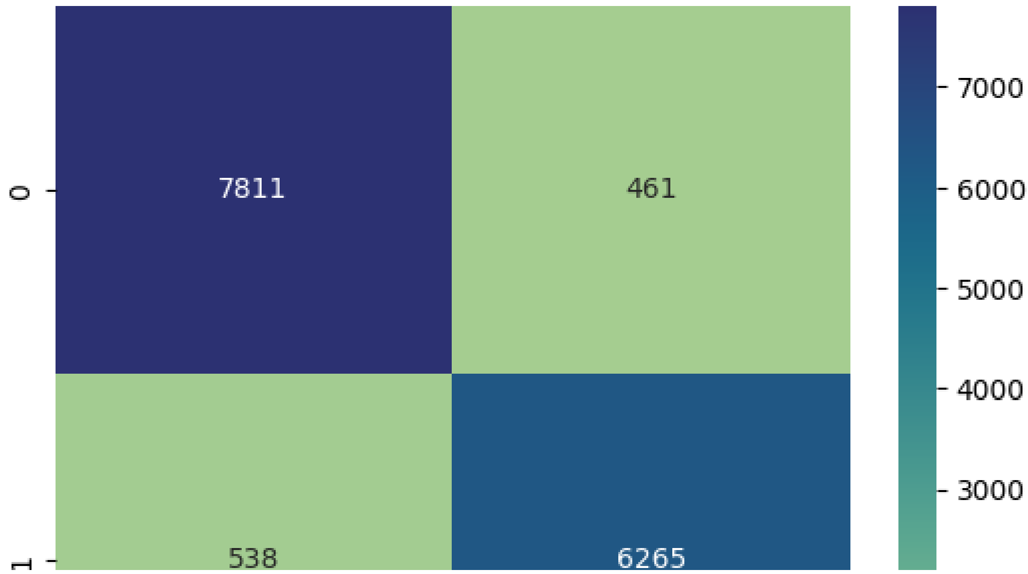
no overfitting

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[7811,  461],
       [ 538, 6265]], dtype=int64)
```

```
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```



Now we will try bagging but using different models

```
estimator=[('lr',LogisticRegression())   ,    ('tree',DecisionTreeClassifier())]

voting_clf=VotingClassifier(estimators=estimator)

bagging_2=BaggingClassifier(
    base_estimator=voting_clf,
    n_estimators=10
                  )


grid = GridSearchCV(
    estimator=bagging_2,
    param_grid=params,
    cv= 5,
    scoring='accuracy',
    n_jobs=-1
)


grid.fit(x_train_selected,y_train)
```

```
C:\Users\Youssef\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8
  warnings.warn(
```

```
grid.best_estimator_
```

```
BaggingClassifier
base_estimator: VotingClassifier
lr                        tree
LogisticRegression        DecisionTreeClassifier
```

```
bagging_2=BaggingClassifier(
    base_estimator=voting_clf,
    n_estimators=10
              )
```

```
bagging_2.fit(x_train_selected,y_train)
```

```
C:\Users\Youssef\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8
  warnings.warn(
```

```
BaggingClassifier
base_estimator: VotingClassifier
lr                        tree
LogisticRegression        DecisionTreeClassifier
```

Check Overfitting

```
y_val_pred=bagging_2.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9371475953565506
```

Evaluation

```
y_pred = bagging_2.predict(x_test_selected)
acc_bag2=accuracy_score(y_pred,y_test)
recall_bag2=recall_score(y_pred,y_test)
precision_bag2=precision_score(y_pred,y_test)
f1_bag2=f1_score(y_pred,y_test)


print(f'accuracy : {acc_bag2}')
print(f'recall =  {recall_bag2}')
print(f'precision =  {precision_bag2}')
print(f'f1 =  {f1_bag2}')
```

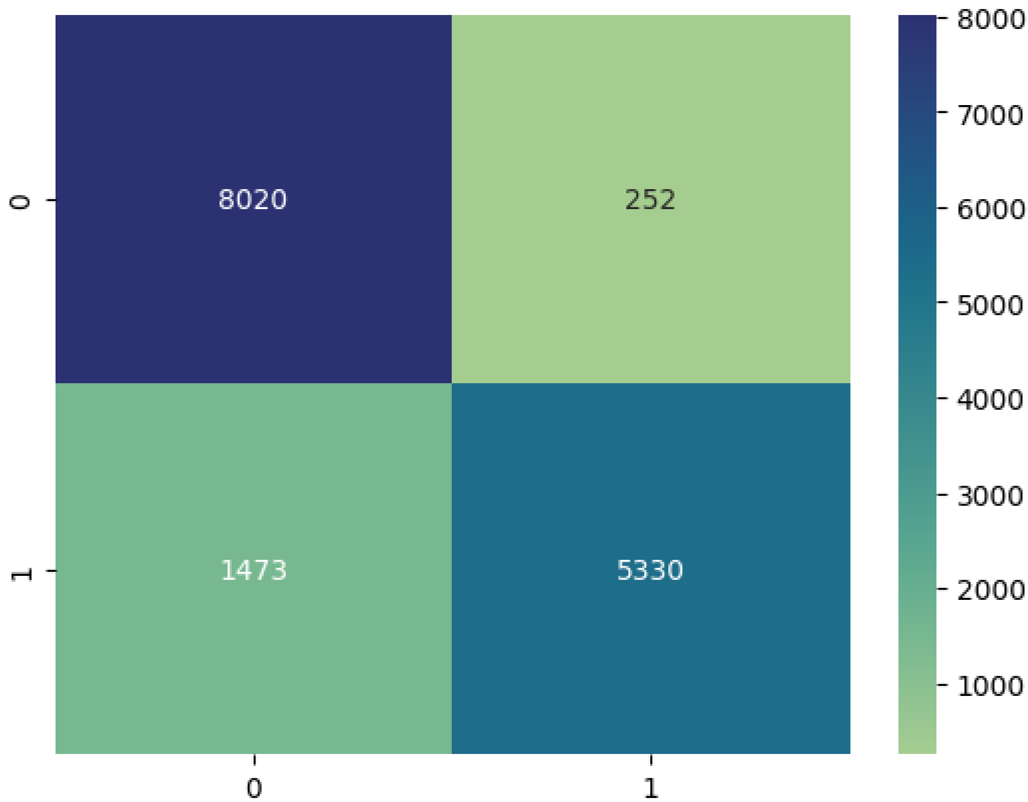
```
accuracy : 0.8855721393034826
recall =  0.9548548907201719
precision =  0.7834778774070263
f1 =  0.860718611223254
```

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[8020,  252],
       [1473, 5330]], dtype=int64)
```

```
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```



Random Forest Classifier

```
clf=RandomForestClassifier()
```

Grid search

```
params = {
    'n_estimators': [100,200,300,400,500,600,700,800]
}
```

```
grid = GridSearchCV(
    estimator=clf,
    param_grid=params,
    cv= 5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid.fit(x_train_selected,y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
grid.best_estimator_
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=400)
```

```
clf=RandomForestClassifier(
    n_estimators=400,random_state=42
)
clf.fit(x_train_selected,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=400, random_state=42)
```

Check Overfitting

```
y_val_pred=clf.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9371475953565506
```

```
y_pred=clf.predict(x_test_selected)
acc_clf=accuracy_score(y_pred,y_test)
recall_clf=recall_score(y_pred,y_test)
precision_clf=precision_score(y_pred,y_test)
f1_clf=f1_score(y_pred,y_test)
```

```
print(f'accuracy : {acc_clf}')
print(f'recall =  {recall_clf}')
print(f'precision =  {precision_clf}')
print(f'f1 =  {f1_clf}')
```

```
accuracy : 0.9346600331674959
recall =  0.934049537451507
precision =  0.9201822725268264
f1 =  0.9270640503517216
```

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[7830,  442],
       [ 543, 6260]], dtype=int64)
```

```
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

<Axes: >

Boosting method

Adaptive boosting

```
ada_model= AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=7),
    n_estimators=200,
    learning_rate=.5,
    random_state=42
)
```

```
ada_model.fit(x_train_selected,y_train)
```

```
C:\Users\Youssef\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8
  warnings.warn(
```

```
▸ AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

Check overfitting

```
y_val_pred=ada_model.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9371475953565506
```

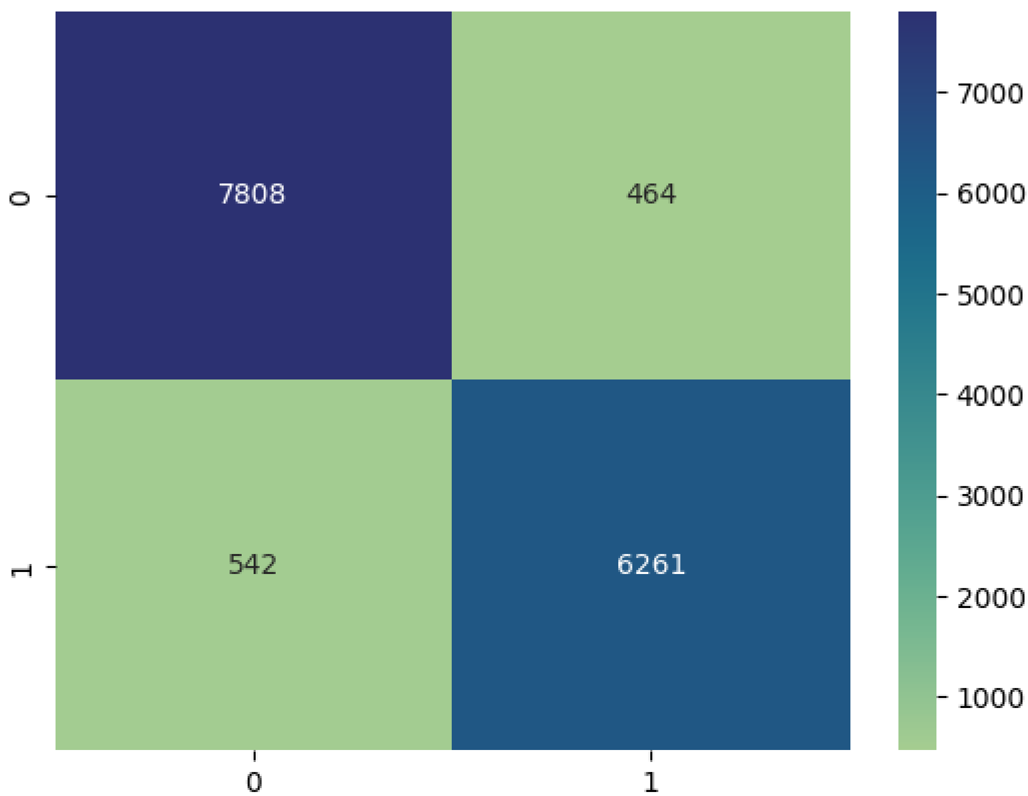
Evaluation

```
y_pred=ada_model.predict(x_test_selected)
acc_ada=accuracy_score(y_pred,y_test)
recall_ada=recall_score(y_pred,y_test)
precision_ada=precision_score(y_pred,y_test)
f1_ada=f1_score(y_pred,y_test)


print(f'accuracy : {acc_ada}')
print(f'recall =  {recall_ada}')
print(f'precision =  {precision_ada}')
print(f'f1 =  {f1_ada}')
```

```
accuracy : 0.9332669983416252
recall =  0.931003717472119
precision =  0.9203292665000735
f1 =  0.9256357185097577
```

```
cm=confusion_matrix(y_test,y_pred)


sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```



Gradient Boost

```
GB=GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=.5,
    random_state=40
)


GB.fit(x_train_selected,y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.5, n_estimators=200, random_state=40)
```

Check overfitting

```
y_val_pred=GB.predict(x_val_selected)
print(accuracy_score(y_val_predict,y_val))
```

```
0.9371475953565506
```

Evaluation

```
y_pred=GB.predict(x_test_selected)
acc_GB=accuracy_score(y_pred,y_test)
recall_GB=recall_score(y_pred,y_test)
precision_GB=precision_score(y_pred,y_test)
f1_GB=f1_score(y_pred,y_test)


print(f'accuracy : {acc_GB}')
print(f'recall =  {recall_GB}')
print(f'precision =  {precision_GB}')
print(f'f1 =  {f1_GB}')
```

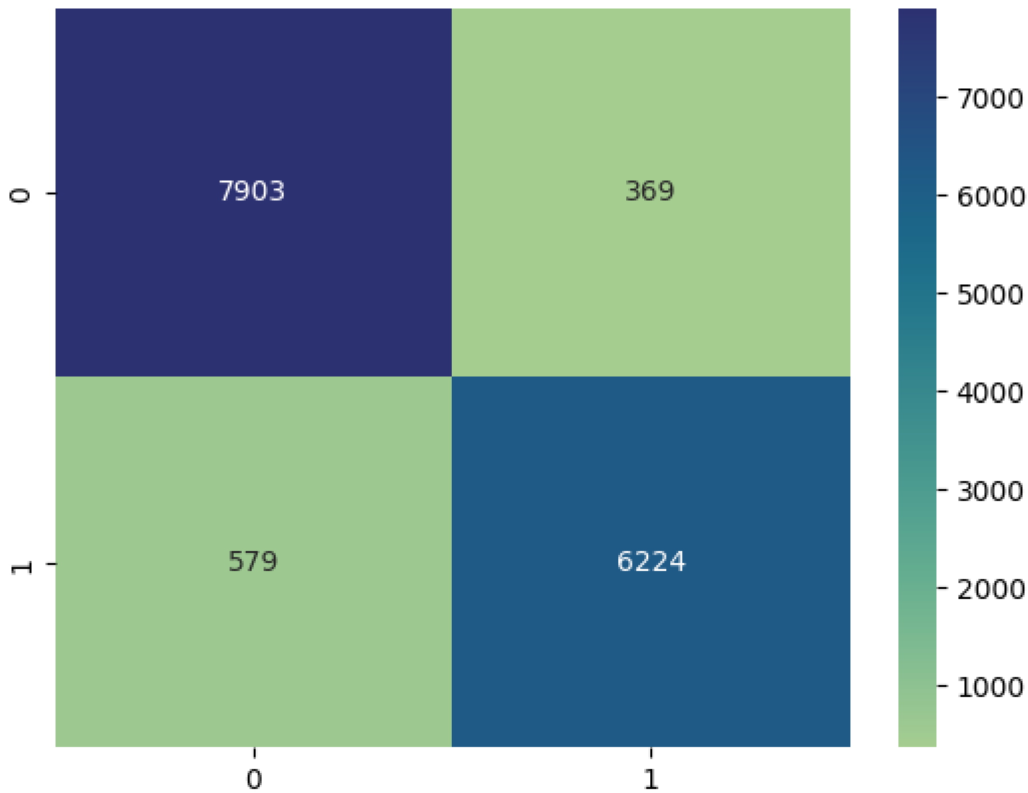
```
accuracy : 0.9371144278606965
recall =  0.9440315486121644
precision =  0.9148904894899309
f1 =  0.929232606748283
```

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[7903,  369],
       [ 579, 6224]], dtype=int64)
```

```
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

<Axes: >



Xtreme Gradient Boost

```
XG=xgb.XGBClassifier(objective='binary:logistic',random_state=42)
```

```
XG.fit(x_train_selected,y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=42, ...)
```

Check Overfitting

```
y_val_pred=XG.predict(x_val_selected)
```

Evaluation

```
y_pred=XG.predict(x_test_selected)
acc_XG=accuracy_score(y_pred,y_test)
recall_XG=recall_score(y_pred,y_test)
precision_XG=precision_score(y_pred,y_test)
f1_XG=f1_score(y_pred,y_test)
```

```
print(f'accuracy : {acc_XG}')
print(f'recall =  {recall_XG}')
print(f'precision =  {precision_XG}')
print(f'f1 =  {f1_XG}')
```

```
accuracy : 0.9397678275290215
recall =  0.9442351168048229
precision =  0.9209172423930618
f1 =  0.93243042119363
```

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[7902,  370],
       [ 538, 6265]], dtype=int64)
```

```
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```

Stacking Method

```
base_models = [
    ('dt', DecisionTreeClassifier()),
    ('lr', LogisticRegression()),
    ('rf', RandomForestClassifier())
]

model_stack=StackingClassifier(estimators=base_models,final_estimator=LogisticRegression())

model_stack.fit(x_train_selected,y_train)
```

```
StackingClassifier
dt                        lr                    rf
DecisionTreeClassifier    LogisticRegression    RandomForestClassifier
final_estimator
LogisticRegression
```

Check Overfitting

```
y_val_pred=model_stack.predict(x_val_selected)
```

Evaluation

```
y_pred=model_stack.predict(x_test_selected)
acc_stack=accuracy_score(y_pred,y_test)
recall_stack=recall_score(y_pred,y_test)
precision_stack=precision_score(y_pred,y_test)
f1_stack=f1_score(y_pred,y_test)
```

```
print(f'accuracy : {acc_stack}')
print(f'recall =  {recall_stack}')
print(f'precision =  {precision_stack}')
print(f'f1 =  {f1_stack}')
```

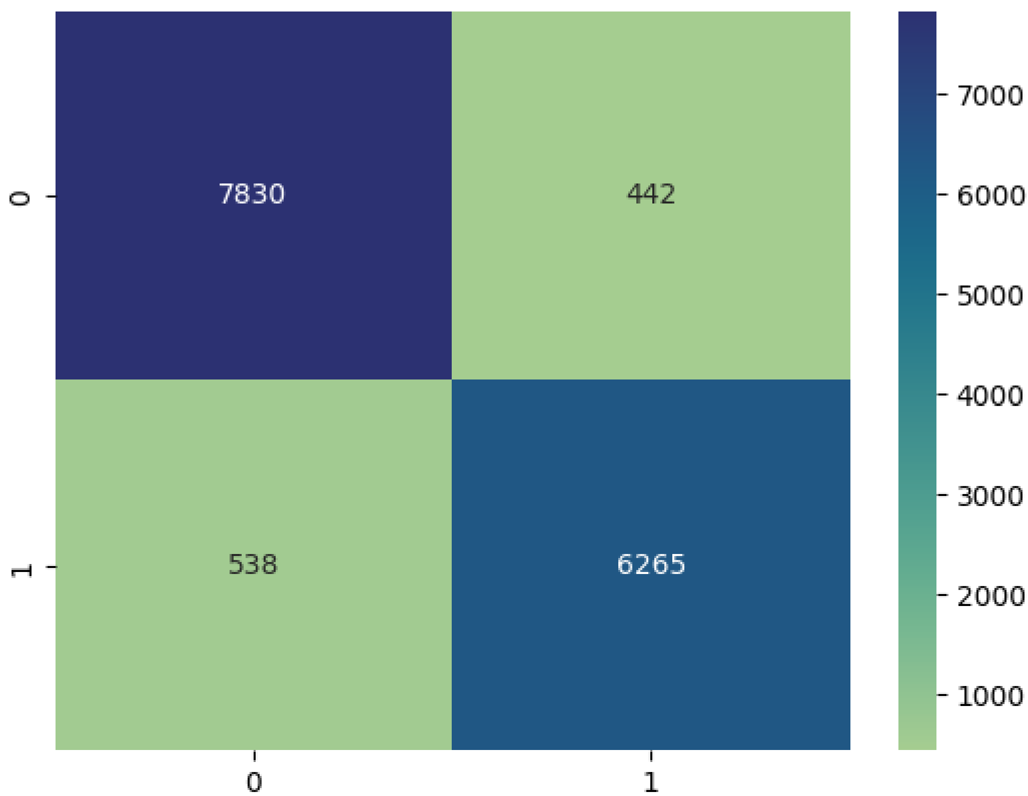
```
accuracy : 0.9349917081260365
recall =  0.934098702847771
precision =  0.9209172423930618
f1 =  0.927461139896373
```

```
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[7830,  442],
       [ 538, 6265]], dtype=int64)
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```



Naive Bayes

```
NB = GaussianNB()
NB.fit(x_train_selected, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

Check Overfitting

```
y_val_pred = NB.predict(x_val_selected)
print(accuracy_score(y_val_pred,y_val))
```

```
0.8359038142620232
```

Evaluation

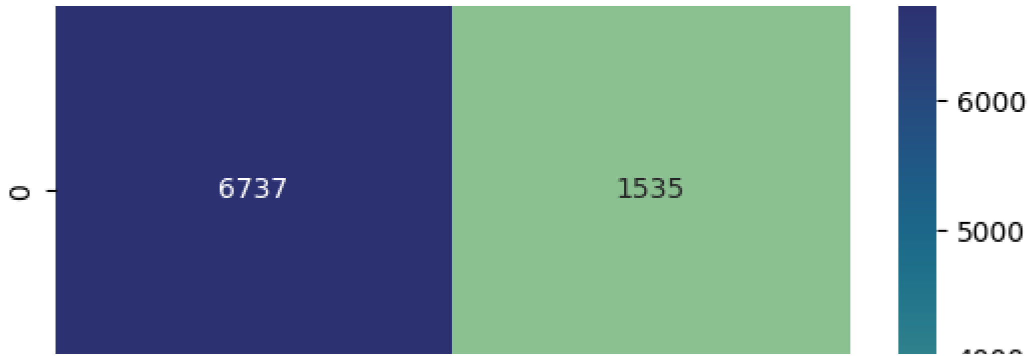
```
y_pred = NB.predict(x_test_selected)
acc_NB=accuracy_score(y_pred,y_test)
recall_NB=recall_score(y_pred,y_test)
precision_NB=precision_score(y_pred,y_test)
f1_NB=f1_score(y_pred,y_test)
```

```
print(f'accuracy : {acc_NB}')
print(f'recall =  {recall_NB}')
print(f'precision =  {precision_NB}')
print(f'f1 =  {f1_NB}')
```

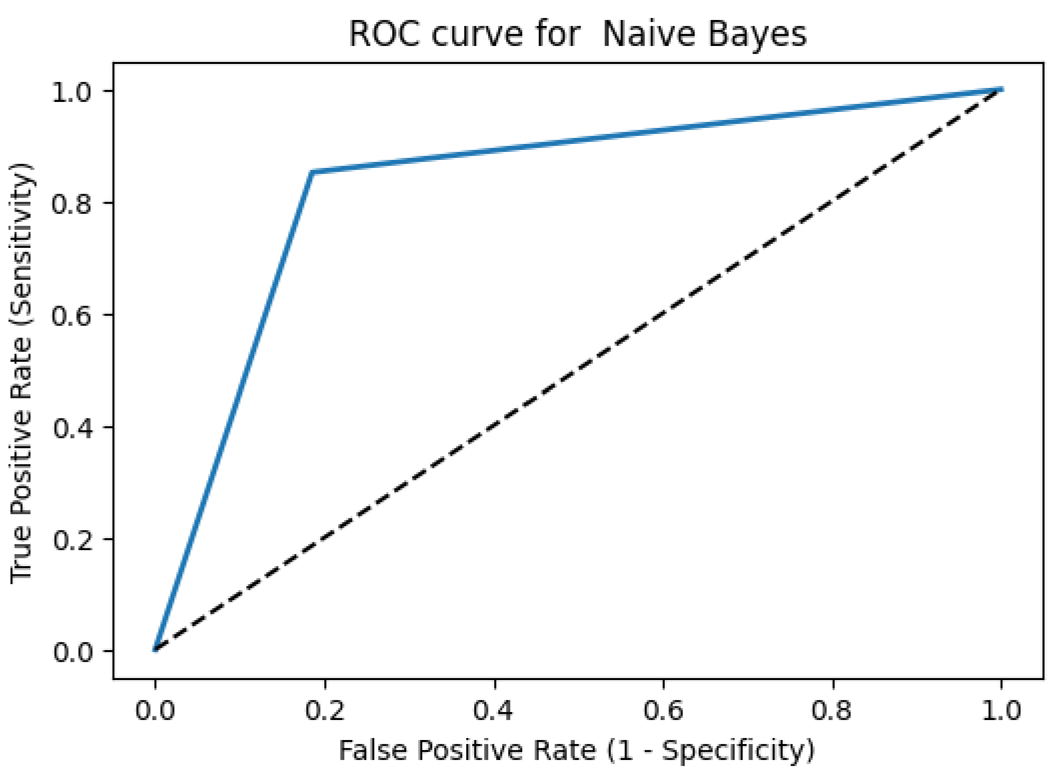
```
accuracy : 0.8312437810945273
recall =  0.7905580570337017
precision =  0.8516830809936793
f1 =  0.8199830172657798
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,cmap='crest',annot=True,fmt='d')
```

```
<Axes: >
```




```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.title('ROC curve for  Naive Bayes')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



Evaluation between all methods used

```
models_names=["Logistic Regression","TREE","KNN","SVC","Bagging","bagging2","Random_forest","
models_scores=[LR_score,DTC_score,KNN_score,SVC_score,Bag1_score,acc_bag2,acc_clf,acc_ada,acc

plt.figure(figsize=(15, 8))
sns.barplot(x=models_names, y=models_scores, data=df,palette='crest')
plt.title('comprasion of classification models')
plt.xlabel('classification models')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```