# Predicting Customer Purchase Behavior Using Machine Learning

## About Data set

Context: This project applies and compares different machine learning algorithms—including Naive Bayes—to analyze customer behavior and predict purchasing decisions. The objective is to identify the key factors influencing whether a customer will purchase a product or not.

Content: The dataset used in this project is a randomly generated dataset representing information about 400 customers, including attributes such as user ID, gender, age, and estimated salary. The target variable, Purchased, indicates whether a customer decided to buy the product. This dataset is designed for educational and analytical purposes to demonstrate supervised learning techniques for binary classification problems.

```
# import requirement libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
import itertools

# for solve problem of show plotly plots
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

# optional
import warnings
warnings.filterwarnings('ignore')
plt.style.use('_mpl-gallery')
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

```
# import dataset
data = pd.read_csv('../input/customer-behaviour/Customer_Behaviour.csv')
print(f"shape: {data.shape}")
data.head()
```

```
shape: (400, 5)
```

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| 0 | 15624510 | Male | 19 | 19000 | 0 |
| 1 | 15810944 | Male | 35 | 20000 | 0 |
| 2 | 15668575 | Female | 26 | 43000 | 0 |
| 3 | 15603246 | Female | 27 | 57000 | 0 |
| 4 | 15804002 | Male | 19 | 76000 | 0 |

## 1. Overview Of Dataset

```
df = pd.DataFrame(data)
df
```

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| **0** | 15624510 | Male | 19 | 19000 | 0 |
| **1** | 15810944 | Male | 35 | 20000 | 0 |
| **2** | 15668575 | Female | 26 | 43000 | 0 |
| **3** | 15603246 | Female | 27 | 57000 | 0 |
| **4** | 15804002 | Male | 19 | 76000 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **395** | 15691863 | Female | 46 | 41000 | 1 |
| **396** | 15706071 | Male | 51 | 23000 | 1 |
| **397** | 15654296 | Female | 50 | 20000 | 1 |
| **398** | 15755018 | Male | 36 | 33000 | 0 |
| **399** | 15594041 | Female | 49 | 36000 | 1 |

400 rows × 5 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

Based on the above information:

The dataset contains 400 entries and 5 columns: User ID, Gender, Age, EstimatedSalary, and Purchased.

There are no missing values in any of the columns, indicating a clean dataset ready for analysis.

The target variable is Purchased, which indicates whether a customer bought the product (1 = Yes, 0 = No).

The feature variables include User ID, Gender, Age, and EstimatedSalary.

Among these, Gender is the only categorical variable, while the others (User ID, Age, EstimatedSalary, Purchased) are numerical.

The dataset is balanced and well-structured, suitable for applying and evaluating classification algorithms such as Naive Bayes.`

## 2. Preparing Dataset

```
# Check missing value
df.isnull().sum().to_frame('NaN value').T
```

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| **NaN value** | 0 | 0 | 0 | 0 | 0 |

Check each column for any syntax errors or invalid values.

```
# check count of unique values in each columns
for col in df:
    print(f"{col}: {df[col].nunique()}")
```

```
User ID: 400
Gender: 2
Age: 43
EstimatedSalary: 117
Purchased: 2
```

```
# more details
df.describe(include=[np.number]).T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **User ID** | 400.0 | 1.569154e+07 | 71658.321581 | 15566689.0 | 15626763.75 | 15694341.5 | 15750363.0 | 15815236.0 |
| **Age** | 400.0 | 3.765500e+01 | 10.482877 | 18.0 | 29.75 | 37.0 | 46.0 | 60.0 |
| **EstimatedSalary** | 400.0 | 6.974250e+04 | 34096.960282 | 15000.0 | 43000.00 | 70000.0 | 88000.0 | 150000.0 |
| **Purchased** | 400.0 | 3.575000e-01 | 0.479864 | 0.0 | 0.00 | 0.0 | 1.0 | 1.0 |

User ID: 400 unique IDs ranging from 15,566,689 to 15,815,236

Age: Ranges from 18 to 60 years, with a mean of ~37 years

EstimatedSalary: Ranges from 15,000 to 150,000, with a mean of ~69,742

Purchased: Binary target variable (0 or 1), with 35.75% positive cases

```
df.describe(include=[object]).T
```

| | count | unique | top | freq |
|---|---|---|---|---|
| **Gender** | 400 | 2 | Female | 204 |

Key Observations

The age range is between 18 and 60 years.

The target variable (Purchased) has 2 classes: 0 and 1.

The gender distribution is nearly balanced.

The feature scales differ significantly (e.g., Age vs. EstimatedSalary), indicating a need for standardization before applying most machine learning algorithms.

We do not need the user ID column to build the predictive model, so we drop it

```
# Drop User ID columns
df.drop('User ID', axis=1, inplace=True)
df
```

| | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| **0** | Male | 19 | 19000 | 0 |
| **1** | Male | 35 | 20000 | 0 |
| **2** | Female | 26 | 43000 | 0 |
| **3** | Female | 27 | 57000 | 0 |
| **4** | Male | 19 | 76000 | 0 |
| **...** | ... | ... | ... | ... |
| **395** | Female | 46 | 41000 | 1 |
| **396** | Male | 51 | 23000 | 1 |
| **397** | Female | 50 | 20000 | 1 |
| **398** | Male | 36 | 33000 | 0 |
| **399** | Female | 49 | 36000 | 1 |

400 rows × 4 columns

```
# convert categoriacl feature to numerical:
# only Gender is categorical
df['Gender'] = df['Gender'].replace(['Male', 'Female'], [0, 1])
df
```

| | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| **0** | 0 | 19 | 19000 | 0 |
| **1** | 0 | 35 | 20000 | 0 |
| **2** | 1 | 26 | 43000 | 0 |
| **3** | 1 | 27 | 57000 | 0 |
| **4** | 0 | 19 | 76000 | 0 |
| **...** | ... | ... | ... | ... |
| **395** | 1 | 46 | 41000 | 1 |
| **396** | 0 | 51 | 23000 | 1 |
| **397** | 1 | 50 | 20000 | 1 |
| **398** | 0 | 36 | 33000 | 0 |
| **399** | 1 | 49 | 36000 | 1 |

400 rows × 4 columns

Now check dataset for the last time

```
df.info()
```

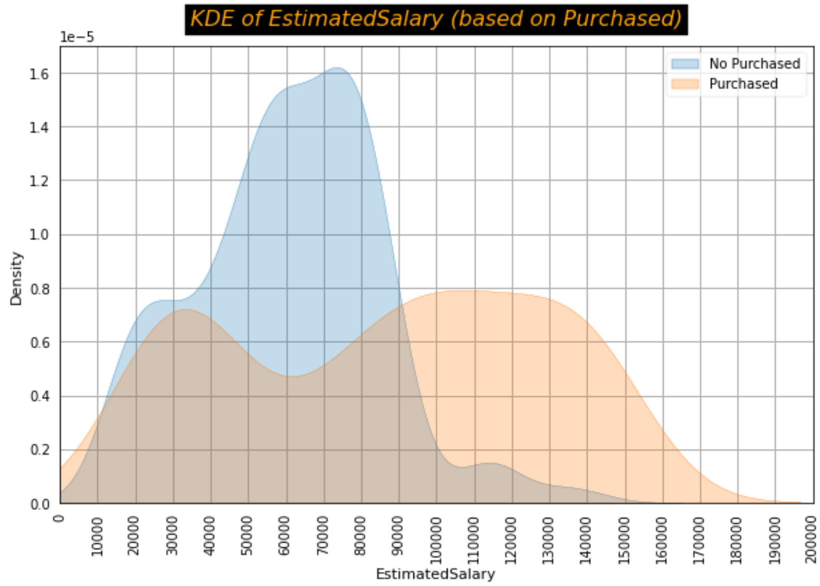
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    int64
 1   Age              400 non-null    int64
 2   EstimatedSalary  400 non-null    int64
 3   Purchased        400 non-null    int64
dtypes: int64(4)
memory usage: 12.6 KB
```

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

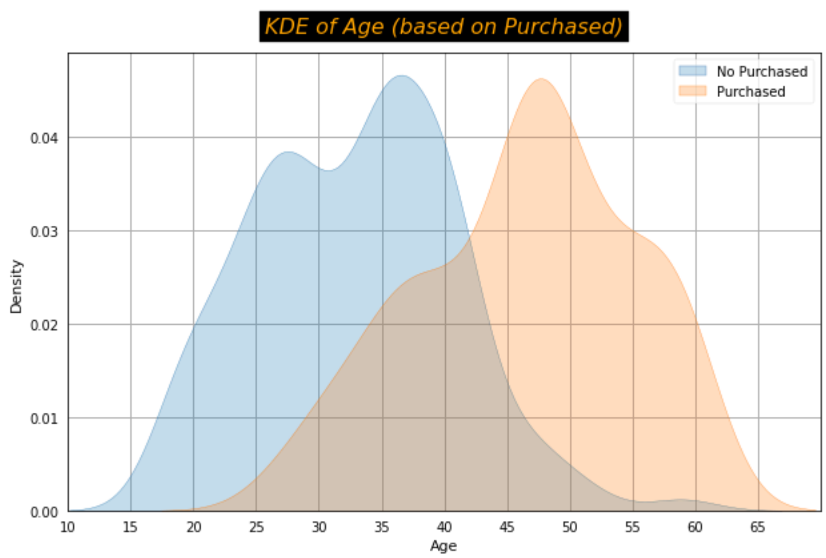
```
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

## 3. Data Analysis

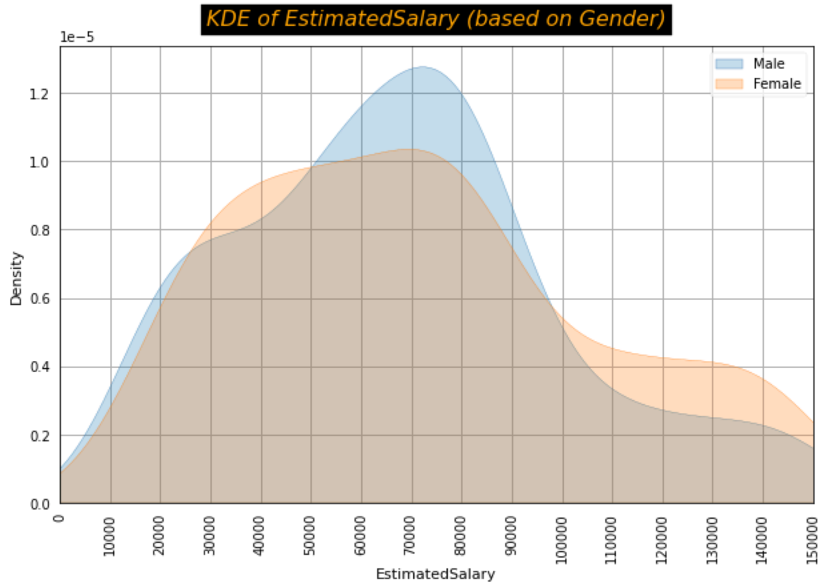
```
# check distribution of EstimatedSalary (based on Purchased)
font = {'fontsize':16, 'fontstyle':'italic', 'backgroundcolor':'black', 'color':'orange'}
%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Purchased'] == 0, 'EstimatedSalary'], label='No Purchased', shade=True)
sns.kdeplot(df.loc[df['Purchased'] == 1, 'EstimatedSalary'], label='Purchased', shade=True)
plt.title('KDE of EstimatedSalary (based on Purchased)', fontdict=font, pad=15)
plt.xticks(np.arange(0,200001,10000), rotation=90)
plt.xlim([0,200001])
plt.legend()
plt.show()
```



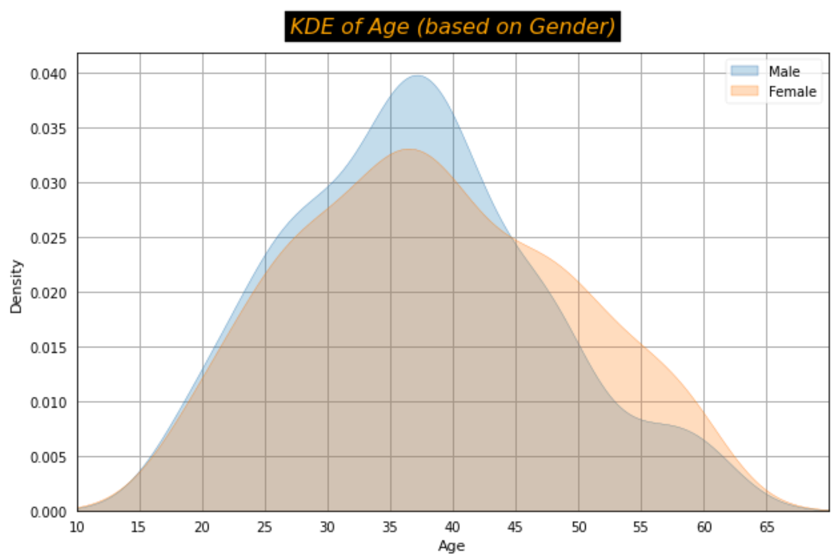

```
# check distribution of Purchased (based on Purchased)
%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Purchased'] == 0, 'Age'], label='No Purchased', shade=True)
sns.kdeplot(df.loc[df['Purchased'] == 1, 'Age'], label='Purchased', shade=True)
plt.title('KDE of Age (based on Purchased)', fontdict=font, pad=15)
plt.xticks(np.arange(0,70,5))
plt.xlim([10,70])
plt.legend()
plt.show()
```




```
# check distribution of EstimatedSalary (based on Gender)
%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Gender'] == 0, 'EstimatedSalary'], label='Male', shade=True)
sns.kdeplot(df.loc[df['Gender'] == 1, 'EstimatedSalary'], label='Female', shade=True)
plt.title('KDE of EstimatedSalary (based on Gender)', fontdict=font, pad=15)
plt.xticks(np.arange(0,150001,10000), rotation=90)
plt.xlim([0,150001])
plt.legend()
plt.show()
```




```
# check distribution of Age (based on Gender)
%matplotlib inline
plt.style.use('seaborn-notebook')
sns.kdeplot(df.loc[df['Gender'] == 0, 'Age'], label='Male', shade=True)
sns.kdeplot(df.loc[df['Gender'] == 1, 'Age'], label='Female', shade=True)
plt.title('KDE of Age (based on Gender)', fontdict=font, pad=15)
plt.xticks(np.arange(0,70,5))
plt.xlim([10,70])
plt.legend()
plt.show()
```




EstimatedSalary vs Purchased

Most customers with income between 40,000 and 90,000 tend not to purchase the product.

Customers who decide to purchase generally have higher salaries compared to those who don’t.

Age vs Purchased

Customers who purchase a product are generally older than those who do not.

People over the age of 43 are more likely to make a purchase.

EstimatedSalary vs Gender

The salary distribution is similar for males and females, with no significant differences observed.

Age vs Gender

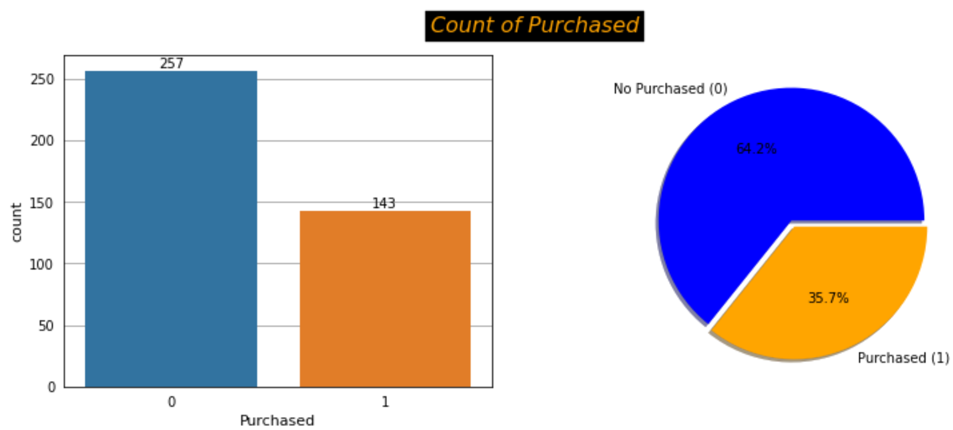
The age distribution is almost identical for males and females.

## 4. Univariate Analysis

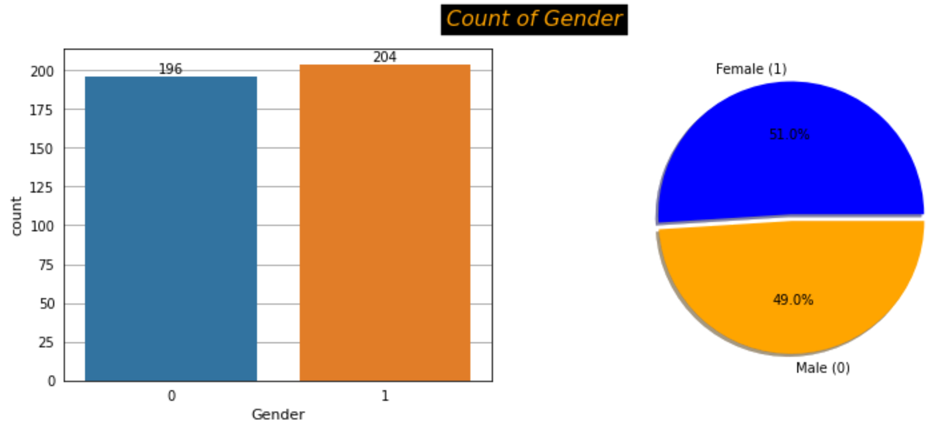
```
df.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Gender** | 400.0 | 0.5100 | 0.500526 | 0.0 | 0.00 | 1.0 | 1.0 | 1.0 |
| **Age** | 400.0 | 37.6550 | 10.482877 | 18.0 | 29.75 | 37.0 | 46.0 | 60.0 |
| **EstimatedSalary** | 400.0 | 69742.5000 | 34096.960282 | 15000.0 | 43000.00 | 70000.0 | 88000.0 | 150000.0 |
| **Purchased** | 400.0 | 0.3575 | 0.479864 | 0.0 | 0.00 | 0.0 | 1.0 | 1.0 |

```
# count based on Purchased (countplot)
fig, axes = plt.subplots(1,2,figsize=(10,4))

sns.countplot(data=df, x='Purchased', ax=axes[0])
for container in axes[0].containers:
    axes[0].bar_label(container)

# count based on Purchased (pie chart)
slices = df.Purchased.value_counts().values
activities = ['No Purchased (0)', 'Purchased (1)']
axes[1].pie(slices, labels=activities, colors=['blue','orange'], shadow=True, explode=[0,0.05], autopct='%1.1f%%')

plt.suptitle('Count of Purchased', y=1.09, **font)
plt.show()
```




```
# count based on Gender (countplot)
fig, axes = plt.subplots(1,2,figsize=(10,4))

sns.countplot(data=df, x='Gender', ax=axes[0])
for container in axes[0].containers:
    axes[0].bar_label(container)

# count based on Gender (pie chart)
slices = df.Gender.value_counts().values
activities = ['Female (1)', 'Male (0)']
axes[1].pie(slices, labels=activities, colors=['blue','orange'], shadow=True, explode=[0,0.05], autopct='%1.1f%%')

plt.suptitle('Count of Gender', y=1.09, **font)
plt.show()
```




Univariate Analysis Insights

Gender Distribution:

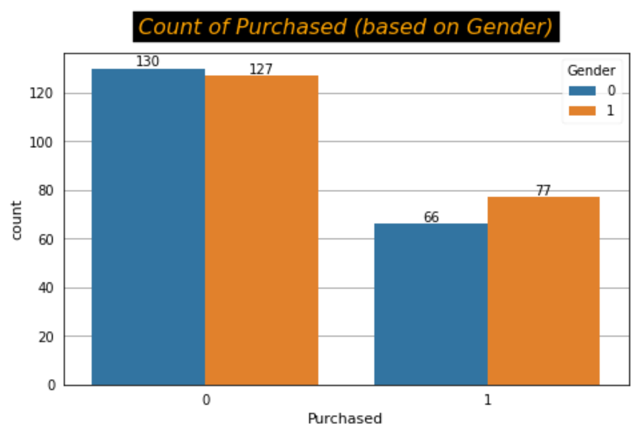
The dataset has almost an equal number of males and females, indicating a balanced representation.

Target Variable (Purchased) Distribution:

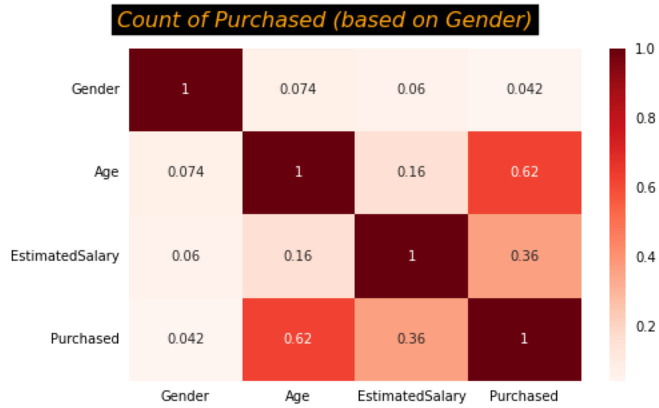
The number of people who decided to purchase a product is lower than the number of people who did not, showing that most customers did not make a purchase.

## 5. Bivariate Analysis

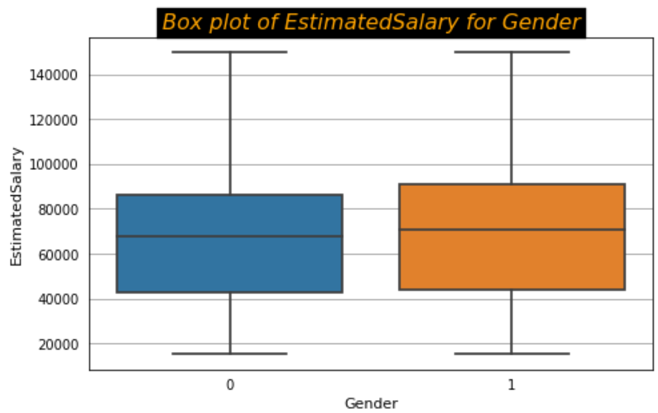
```
# count of purchased based on Gender
%matplotlib inline
ax = sns.countplot(data=df, x='Purchased', hue='Gender')
for container in ax.containers:
    ax.bar_label(container)
plt.title('Count of Purchased (based on Gender)', fontdict=font, pad=15)
plt.show()
```

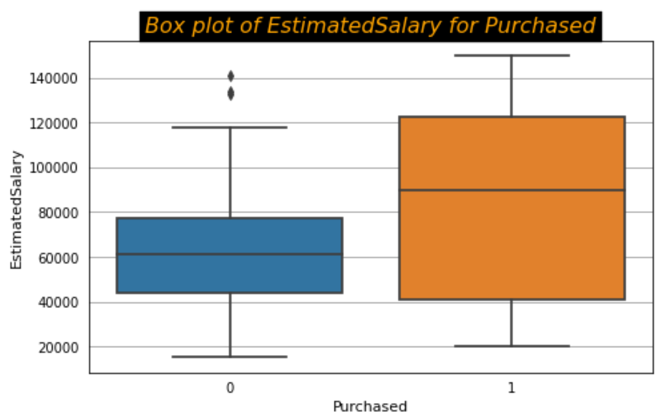



```
# heatmap
sns.heatmap(df.corr(), cmap='Reds', annot=True)
plt.suptitle('Count of Purchased (based on Gender)', y=1.09, x=0.35, **font)
plt.show()
```




```
# draw box plot of Estimated salary for male(0) or female(1) Gender
# draw box plot of Estimated salary for no purchased(0) or purchased(1)
for col in ['Gender', 'Purchased']:
    sns.boxplot(data=df, x=col, y='EstimatedSalary')
    plt.title(f'Box plot of EstimatedSalary for {col}', fontdict=font)
    plt.show()
```




Bivariate Analysis Insights

Purchased vs Gender:

Among the people who decide to purchase a product, there are more females than males.

Among the people who do not purchase, there are more males than females.

Purchased vs Age:

There is a strong positive correlation (0.62) between Age and Purchased, indicating that older customers are more likely to make a purchase.

Purchased vs EstimatedSalary:

The average EstimatedSalary of customers who purchase a product is higher than those who do not.

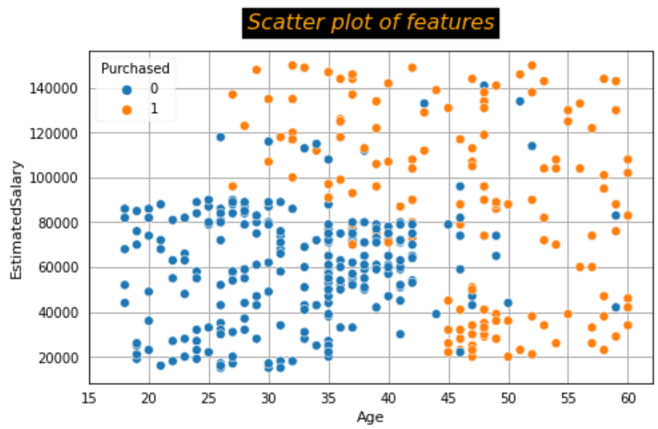
Gender vs EstimatedSalary:

The average salary of males and females is almost the same, showing no significant difference between genders.

## 6. Multivariate Analysis

We use scatter plots to examine data that is numerical in nature.

```
%matplotlib inline
# check feature correlation
sns.scatterplot(data=df,x='Age', y='EstimatedSalary', hue='Purchased',)
plt.title('Scatter plot of features', y=1.04, fontdict=font)
plt.xticks(np.arange(15,65,5))
plt.show()
```



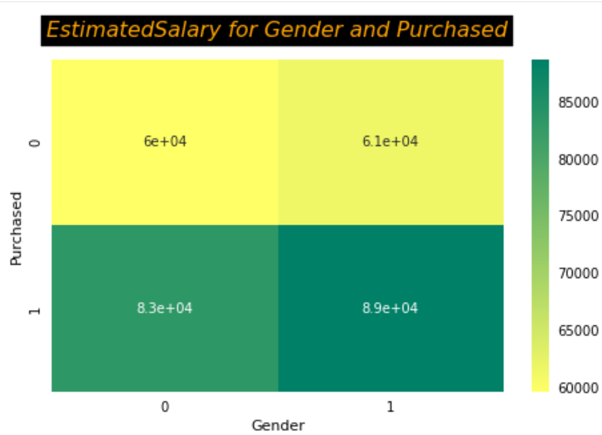

For a better view, we draw a 3D scatter plot

```
fig = px.scatter_3d(
        data_frame=df,
        x='Age',
        y='EstimatedSalary',
        z='Gender',
        color='Purchased',
        template='ggplot2',
        opacity=0.6,
        height=700,
        title=f'3d scatter based on Age, EstimatedSalary, Gender and Purchased'
)

pio.show(fig)
```

```
# check mean of EstimatedSalary based on Gender and Purchased
results = pd.pivot_table(data=df, index='Purchased', columns='Gender', values='EstimatedSalary')
results.style.background_gradient(cmap='summer_r')
```

| Gender | 0 | 1 |
|---|---|---|
| **Purchased** | | |
| **0** | 59630.769231 | 61480.314961 |
| **1** | 83424.242424 | 88714.285714 |

```
# show result in heatmap
sns.heatmap(results, cmap='summer_r', annot=True)
plt.suptitle('EstimatedSalary for Gender and Purchased', y=1.09, x=0.4, **font)
plt.show()
```




```
# check mean of Age based on Gender and Purchased
results = pd.pivot_table(data=df, index='Purchased', columns='Gender', values='Age')
results.style.background_gradient(cmap='summer_r')
```

| Gender | 0 | 1 |
|---|---|---|
| Purchased | | |
| 0 | 32.484615 | 33.110236 |
| 1 | 45.500000 | 47.155844 |

```
# show result in heatmap
sns.heatmap(results, cmap='summer_r', annot=True)
plt.suptitle('Age for Gender and Purchased', y=1.09, x=0.4, **font)
plt.show()
```


According to above plots:

- People with young age and low EstimatedSalary often do not have a decision to purchase product.
- People with a EstimatedSalary of more than 100000, regardless of their Age, often decide to purchase product.
- People over the age of 45, regardless of their EstimatedSalary, are more likely to pruchase a product.

These plots also confirm the previous results

## 7. Model

```
# Our dataset we use for modeling
df
```

| | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| 0 | 0 | 19 | 19000 | 0 |
| 1 | 0 | 35 | 20000 | 0 |
| 2 | 1 | 26 | 43000 | 0 |
| 3 | 1 | 27 | 57000 | 0 |
| 4 | 0 | 19 | 76000 | 0 |
| ... | ... | ... | ... | ... |
| 395 | 1 | 46 | 41000 | 1 |
| 396 | 0 | 51 | 23000 | 1 |
| 397 | 1 | 50 | 20000 | 1 |
| 398 | 0 | 36 | 33000 | 0 |
| 399 | 1 | 49 | 36000 | 1 |

400 rows × 4 columns

Since the dataset contains both discrete and continuous features, we choose to use Multinomial Naive Bayes among the Naive Bayes variants. Additionally, because the feature ranges differ significantly, it is important to standardize the features before applying the model to ensure better performance and convergence.

```
# standardize EstimatedSalary and Age with StandardScaler
df2 = df.copy()
```

```
scaler = MinMaxScaler(feature_range=(18,60)).fit(df[['EstimatedSalary']])
df2['EstimatedSalary'] = scaler.transform(df2['EstimatedSalary'].values.reshape(-1,1))
df2
```

| | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| **0** | 0 | 19 | 19.244444 | 0 |
| **1** | 0 | 35 | 19.555556 | 0 |
| **2** | 1 | 26 | 26.711111 | 0 |
| **3** | 1 | 27 | 31.066667 | 0 |
| **4** | 0 | 19 | 36.977778 | 0 |
| **...** | ... | ... | ... | ... |
| **395** | 1 | 46 | 26.088889 | 1 |
| **396** | 0 | 51 | 20.488889 | 1 |
| **397** | 1 | 50 | 19.555556 | 1 |
| **398** | 0 | 36 | 23.600000 | 0 |
| **399** | 1 | 49 | 24.533333 | 1 |

400 rows × 4 columns

```
# define x (features) and y (target)
x = np.asanyarray(df2.drop('Purchased', axis=1))
y = df2.Purchased.values.reshape(-1,1)
```

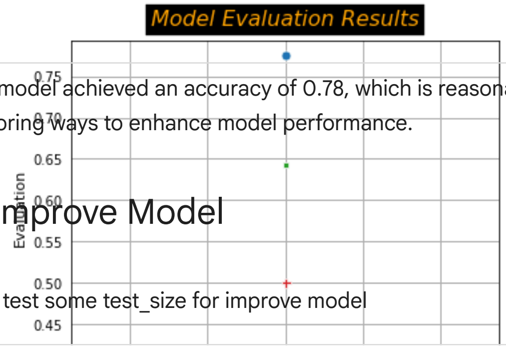
```
FPR1 = []
TPR1 = []
FPR0 = []
TPR0 = []
ACC_test = []
ACC_train = []
Recall = []
Precision = []
F1 = []

def plot_confusion_matrix2(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function plots the confusion matrix.
        cm(array): confusion matrix
        classes(dictionary): classes in our target
    """
    plt.figure(figsize=(10,7))
    plt.grid(False)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt ='d'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")


    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.tight_layout()
    plt.show()

# ---------------------------------------------------------------------------------------

def Perform_cross_val(model, k, x, y, scoring):
    """
    perform cross validation
```

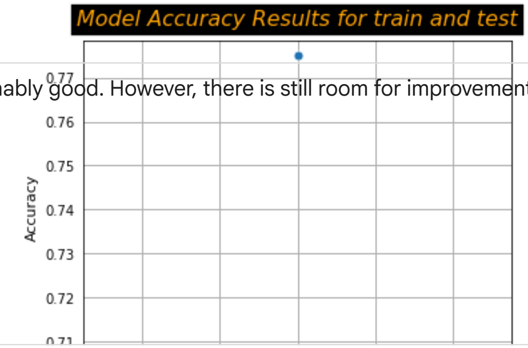
```
        model: logistic model
        k(scaler): the value for n_splits in KFold()
        x(DataFrame or array):  x_train
        y(DataFrame or array): y_train
        scoring(string): an approach for evaluation in cross validation
    """

    kf = KFold(n_splits=k)
    cv_results = cross_val_score(model, x, y, cv=kf, scoring=scoring)
    cv_mean = np.mean(cv_results)
    print('-'*20, f"CV for k={k}, scoring={scoring}", '-'*20)
    print(f"CV mean: {cv_mean}")
    print(f"CV results: {cv_results}\n")

# --------------------------------------------------------------------------------------

def find_fold_index(k, x):
    """
    Find fold index in kfold
        k(scaler): the value used for n_splits in KFold()
        x(DataFrame or array): x_train
    """

    my_fold_index = []
    j=1
    for _ , test in KFold(k).split(x):

        my_fold_index = []
        for i in test:
            my_fold_index.append(i)
        print(f"fold {j}: [{my_fold_index[0]},{my_fold_index[-1]}]")
        print(20*'-')
        j += 1

# --------------------------------------------------------------------------------------

def plot_results(FPR0, TPR0, FPR1, TPR1, ACC_test, ACC_train, Recall, Precision, F1):
    """
    draw ROC curve and plot of Recall, precision, f1 score etc.
        FPR0(list): list of False Positive Rate for class 0
        TPR0(list): list of True Positive Rate for class 0
        FPR1(list): list of Flase Positive Rate for class 1
        TPR1(list): list of True Positive Rate for class 1
        ACC(list): list of accuracy of models
        Recall(list): list of recall score of models
        Precision(list): list of Precision score of models
        F1(list): list of F1 score of models
    """
    fig, ax = plt.subplots(1,2,figsize=(10,4))
    # plot model evaluation
    ax[0].set_title('Model Evaluation Results', fontdict=font, y=1.02)
    sns.lineplot(data=pd.DataFrame({'accoracy': ACC_test, 'Recall': Recall,
                                'Precision': Precision, 'F1 score': F1}),
                                markers=True, ax=ax[0])
    ax[0].set_xlabel('M')
    ax[0].set_ylabel('Evaluation')
    ax[0].legend(loc='upper center', bbox_to_anchor=(0.5, -0.12),
          fancybox=True, shadow=True)

    # plot model evaluation
    ax[1].set_title('Model Accuracy Results for train and test', fontdict=font, y=1.02)
    sns.lineplot(data=pd.DataFrame({'test accuracy': ACC_test, 'train accuracy': ACC_train}),
                                markers=True, ax=ax[1])
    ax[1].set_xlabel('M')
    ax[1].set_ylabel('Accuracy')
    ax[1].legend(loc='upper center', bbox_to_anchor=(0.5, -0.12),
          fancybox=True, shadow=True)
    plt.show()

    # plot ROC curve for class 1
    fig, ax = plt.subplots(1,2,figsize=(10,4))
    i=1
    ax[0].set_title('ROC Curve of Class 1', fontdict=font, y=1.02)
    for fpr , tpr in zip(FPR1, TPR1):
        ax[0].plot(fpr, tpr, label=f"ROC curve of model{i} (AUC = {round(metrics.auc(fpr, tpr),3)})")
        i += 1
        ax[0].set_xlabel('FPR')
```

```
        ax[0].set_ylabel('TPR')
    ax[0].legend(loc='upper center', bbox_to_anchor=(0.5, -0.12),
          fancybox=True, shadow=True)

    # plot ROC curve for class zero
    i=1
    ax[1].set_title('ROC Curve of Class 0', fontdict=font, y=1.02)
    for fpr , tpr in zip(FPR0, TPR0):
        ax[1].plot(fpr, tpr, '--', label=f"ROC curve of model{i} (AUC = {round(metrics.auc(fpr, tpr),3)})")
        i += 1
        ax[1].set_xlabel('FPR')
        ax[1].set_ylabel('TPR')
    ax[1].legend(loc='upper center', bbox_to_anchor=(0.5, -0.12),
        fancybox=True, shadow=True)

    plt.show()

# ---------------------------------------------------------------------------------------

def modeling(x, y, test_size, classes, is_add=1 ):

    # split data to train and test
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=0)
    print(20*'-', 'Shape', 20*'-')
    print(f"x_train: {x_train.shape}")
    print(f"y_train: {y_train.shape}")
    print(f"x_test: {x_test.shape}")
    print(f"y_test: {y_test.shape}")



    # define model and fit model
    clf = MultinomialNB()
    clf.fit(x_train, y_train.ravel())

    # prediction and results
    y_pred_train = clf.predict(x_train)
    y_pred_test = clf.predict(x_test)
    y_proba_train = clf.predict_proba(x_train)
    y_proba_test = clf.predict_proba(x_test)

    cm = metrics.confusion_matrix(y_test, y_pred_test)
    fpr1, tpr1, _ = metrics.roc_curve(y_test, y_proba_test[:,1])
    fpr0, tpr0, _ = metrics.roc_curve(y_test, y_proba_test[:,0])
    acc_test = metrics.accuracy_score(y_test, y_pred_test)
    acc_train = metrics.accuracy_score(y_train, y_pred_train)
    rec = metrics.recall_score(y_test, y_pred_test)
    pre = metrics.precision_score(y_test, y_pred_test)
    f1 = metrics.f1_score(y_test, y_pred_test)

    # append results
    if is_add == 1:
        FPR0.append(fpr0)
        TPR0.append(tpr0)
        FPR1.append(fpr1)
        TPR1.append(tpr1)
        ACC_test.append(acc_test)
        ACC_train.append(acc_train)
        Recall.append(rec)
        Precision.append(pre)
        F1.append(f1)

    plot_results(FPR0, TPR0, FPR1, TPR1, ACC_test, ACC_train, Recall, Precision, F1)

    # Evaluation model
    print('-'*20 , 'Confusion Matrix', '-'*20)
    print(cm)
    plot_confusion_matrix2(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues)
    # or use plot_confusion_matrix from sklearn.metrics
    print('-'*20 , 'Classification Report', '-'*20)
    print(metrics.classification_report(y_test, y_pred_test, target_names=classes), '\n')
    print(f"Jaccard Score: {metrics.jaccard_score(y_test, y_pred_test)}", '\n')

    # print other result about predicted data
    return clf, acc_test, acc_train
```

```
# Now create first model
clf1, acc_test1, acc_train1 = modeling(x, y, 0.2, ['No Purchased=0', 'Purchased=1'])
```

```
-------------------- Shape --------------------
x_train: (320, 3)
y_train: (320, 1)
x_test: (80, 3)
y_test: (80, 1)
```



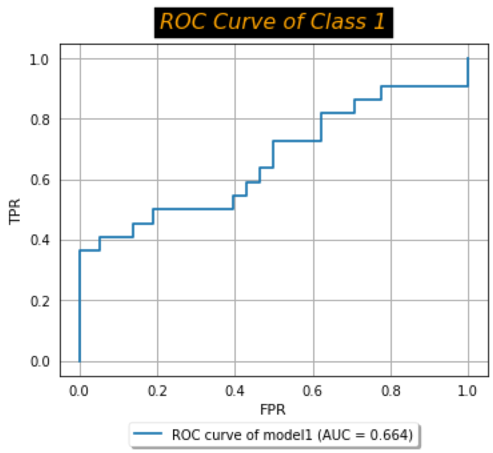



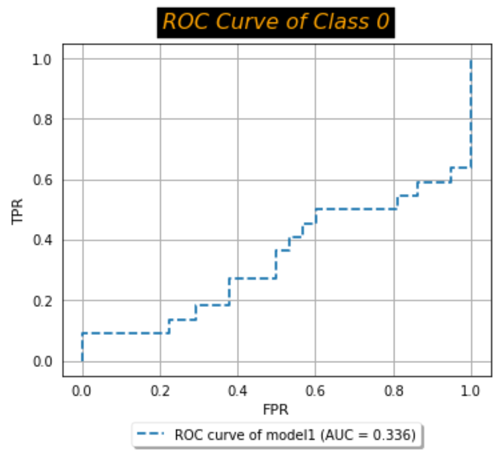

The model achieved an accuracy of 0.78, which is reasonably good. However, there is still room for improvement, so we will continue exploring ways to enhance model performance.

## 8. Improve Model

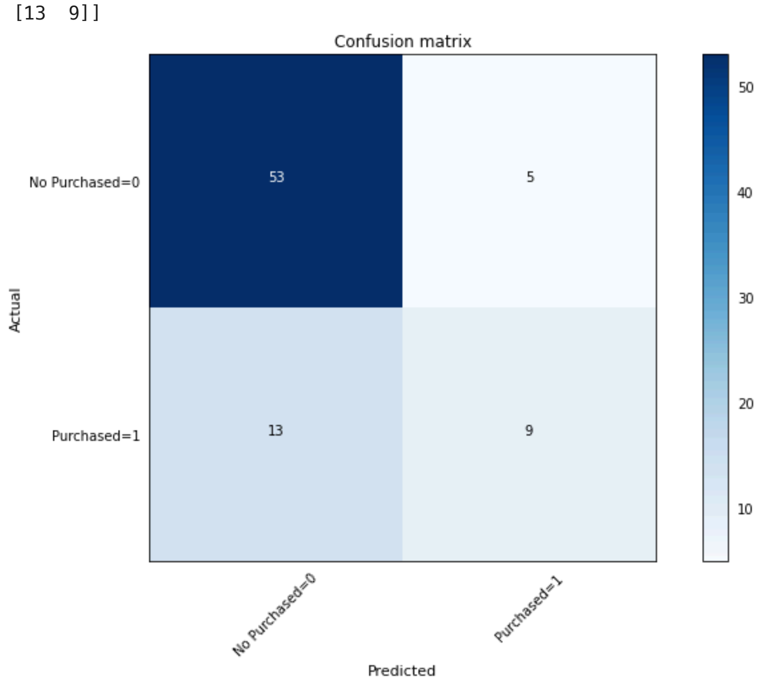
First test some test_size for improve model

```
# test test_size= 0.3
clf2, acc_test2, acc_train2 = modeling(x, y, 0.3, ['No Purchased=0', 'Purchased=1'])
```

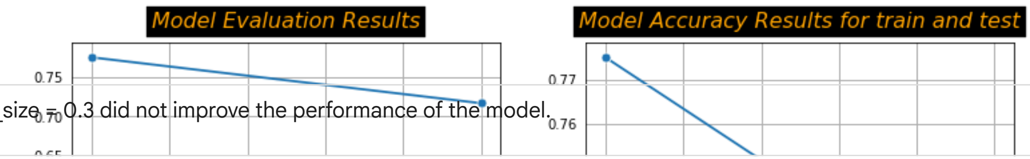

```
-------------------- Confusion Matrix --------------------
[[53  5]
 [13  9]]
```




```
-------------------- Classification Report --------------------
                precision    recall  f1-score   support

No Purchased=0       0.80      0.91      0.85        58
   Purchased=1       0.64      0.41      0.50        22
```

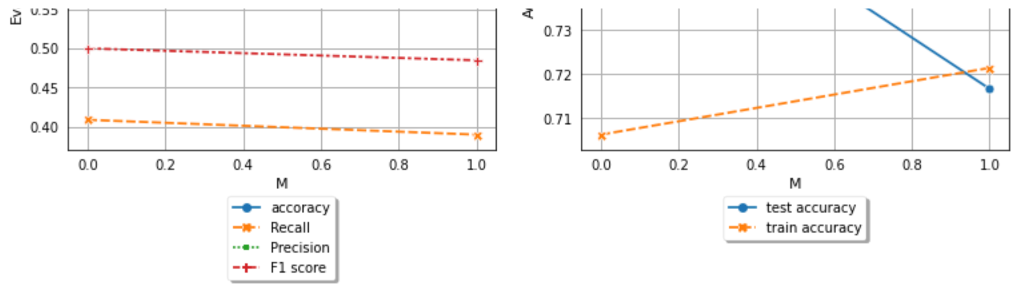
```
    accuracy                           0.78        80
   macro avg       0.72      0.66      0.68        80
weighted avg       0.76      0.78      0.76        80


Jaccard Score: 0.3333333333333333
```

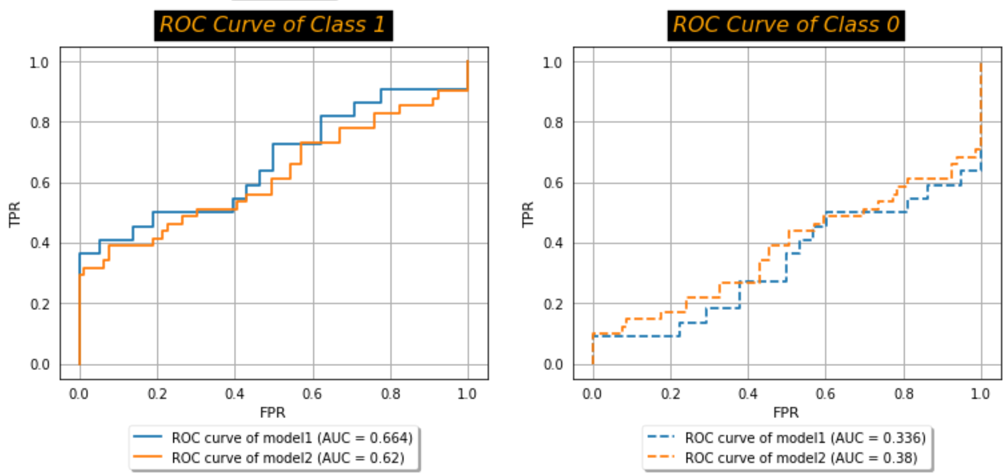
```
-------------------- Shape --------------------
x_train: (280, 3)
y_train: (280, 1)
x_test: (120, 3)
y_test: (120, 1)
```



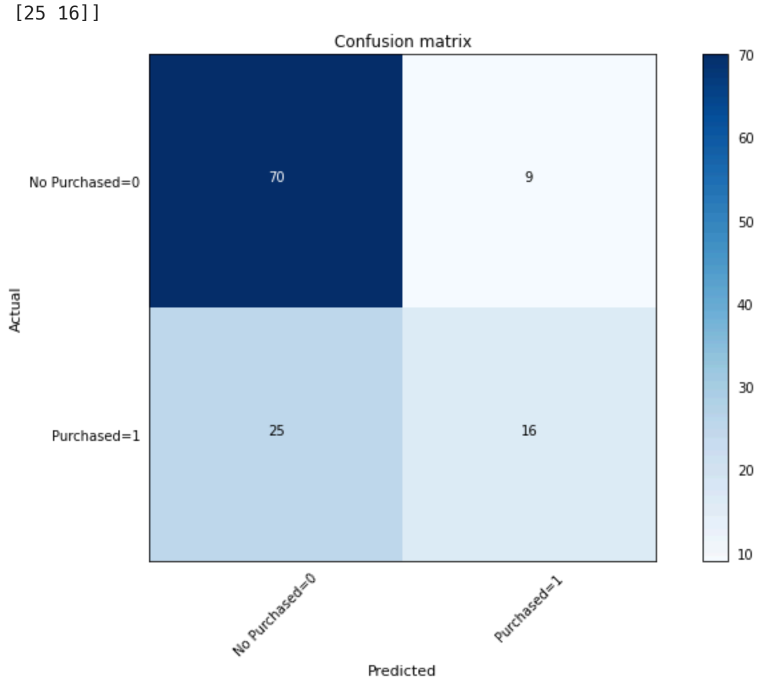

test_size = 0.3 did not improve the performance of the model.

```
# test test_size= 0.15
clf3, acc_test3, acc_train3 = modeling(x, y, 0.15, ['No Purchased=0', 'Purchased=1'])
```




```
-------------------- Confusion Matrix --------------------
[[70  9]
 [25 16]]
```




```
-------------------- Classification Report --------------------
                precision    recall  f1-score   support

No Purchased=0       0.74      0.89      0.80        79
   Purchased=1       0.64      0.39      0.48        41
```

```
    accuracy                           0.72       120
   macro avg       0.69      0.64      0.64       120
weighted avg       0.70      0.72      0.70       120


Jaccard Score: 0.32
```