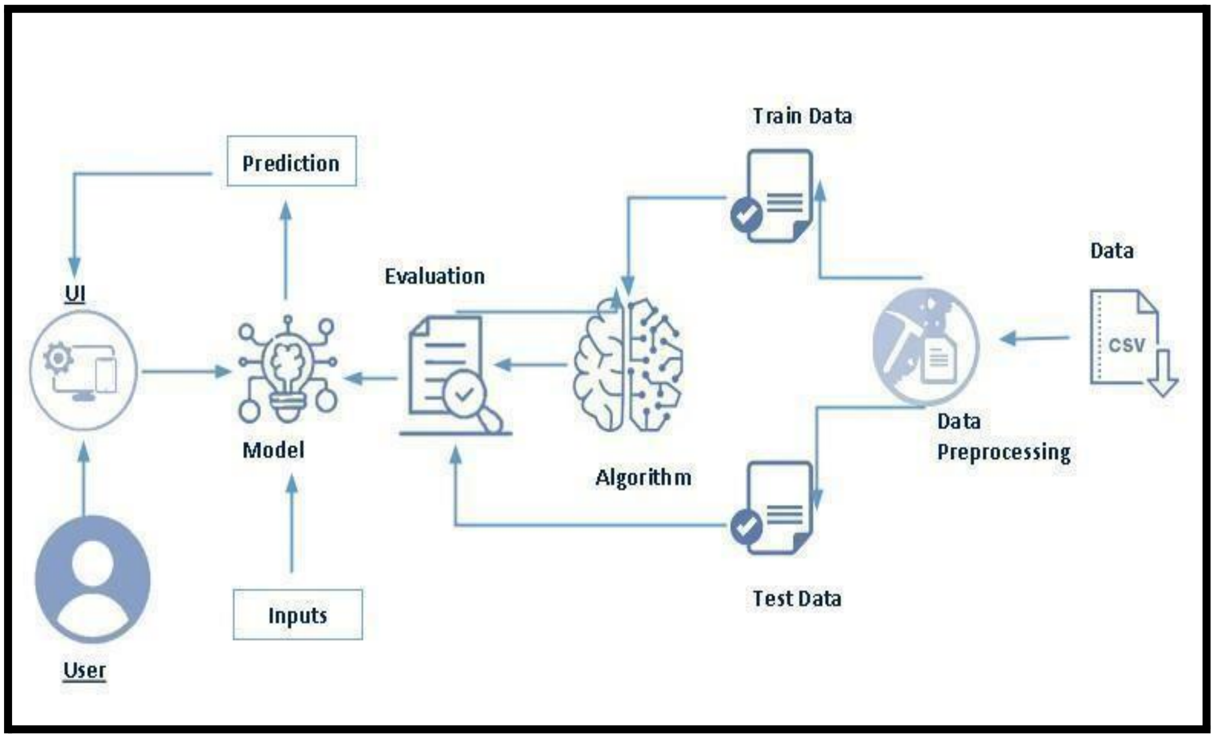
Car purchase Prediction Using ML

Project Description:

Developed an innovative ML solution to predict car purchases based on customer data. Leveraged features such as age, income, and historical purchase patterns for accurate forecasts. Achieved high predictive accuracy using advanced algorithms and thorough data preprocessing. The model assists potential buyers by estimating their likelihood to make a purchase, guiding decision-making.
Seamlessly integrated the model into a user-friendly interface, enabling easy predictions for users. This project revolutionizes the automotive industry by offering insights for tailored marketing strategies. Enhances customer experiences by facilitating informed choices and dealership targeting.
A groundbreaking application of ML driving data-powered decisions.

Through meticulous training and feature engineering, the model attains a high accuracy rate, ensuring dependable predictions. Seamlessly integrated into a user-friendly interface, users input their demographics and receive precise purchase likelihoods. This innovation revolutionizes marketing strategies by enabling targeted customer engagement and resource optimization. By harnessing data insights, the project empowers automotive businesses to tailor their approaches, enhancing overall efficiency.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

1) Define Problem / Problem Understanding

- Specify the business problem
- Business requirement
- Literature Survey
- Social or Business Impact.

2) Data Collection & Preparation

- Collect the dataset

3) Exploratory Data Analysis

- Descriptive statistical
- Visual Analysis
- Feature Selection
- Scaling the data
- Checking if the dataset is balanced or not

4) Model Building

- Splitting data into train and test
- Training the model in multiple algorithms
- Testing the model

5) Performance Testing

- Create data frame of model performance
- Bar plot for model performance

6) Model Deployment & Application Building

1. Save the best model
2. Integrate with Web Framework

Project Demonstration & Documentation

○ Record explanation Video for project end to end solution

○ Project Documentation-Step by step project development procedure

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

A car dealer seeks to improve on their sales approach, whereby it predicts which consumers will purchase a vehicle soon. It has a database which entails client's details such as age, previous purchases, online interactions as well as other traits. The aim is to construct a machine learning algorithm which will be able to forecast the probability of a car purchase by a certain consumer in the near future.

Activity 2: Business requirements

Here are some potential business requirements for Share price predictor.

Data Collection and Integration: Gather and integrate a comprehensive dataset that includes customer age and income information, ensuring data accuracy and consistency.

Integration with Marketing Strategies: The model should seamlessly integrate with the company's marketing strategies, enabling targeted campaigns towards potential car buyers.

Scalability: Design the model to handle a growing number of customer data points while maintaining prediction accuracy and response times.

Resource Optimization: Assist in optimizing marketing resources by directing efforts towards customers with a higher probability of purchasing a car, leading to reduced costs and improved ROI.

Customization: Consider the ability to customize the model for specific market segments or product lines, enabling fine-tuned predictions and targeted strategies.

Activity 3: Literature Survey (Student Will Write)

C. Jin, "Price Prediction of Used Cars Using Machine Learning," *2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT)*, Chongqing, China, 2021, pp. 223-230, doi: 10.1109/ICESIT53460.2021.9696839

K. Samruddhi and R. Ashok Kumar, "Used Car Price Prediction using K-Nearest Neighbor Based Model", International Journal of Innovative Research in Applied Sciences and Engineering, vol. 4, no. 3, pp. 686-689, 2020.

Cumhur. E. & Senturk, I. (2009), "A Hedonic Analysis of Used Car Prices in Turkey", International Journal of Economic Perspectives, 3(2), 141-149.

O. Celik and U. O. Osmanoglu, "Prediction of The Prices of Second-Hand Cars", Avrupa Bilim ve Teknoloji Dergisi, no. 16, pp. 77-83, Aug. 2019.

Activity 4: Social and Business Impact.

The Anime Recommendation project can have both social and business impacts.

Social Impact:

The introduction of the car purchase prediction ML model brings notable social implications. Customers benefit from a more personalized experience as their purchasing likelihood is assessed based on individual characteristics, reducing irrelevant marketing outreach. This enhances user satisfaction and trust, fostering positive brand-consumer relationships. Moreover, the model encourages responsible consumption by aligning customers with suitable car options, considering their financial circumstances. As data-driven decisions become more prevalent, it encourages an informed approach to car purchases, promoting financial prudence among consumers.

Business Impact:

On the business front, the car purchase prediction ML model yields substantial impact. Marketing efforts become laser-focused, targeting individuals with a higher chance of conversion, resulting in enhanced efficiency and cost reduction. The model facilitates improved resource allocation, optimizing budget allocation for campaigns that promise the highest return on investment. Sales teams can prioritize leads, streamlining the conversion process and potentially increasing sales volume. Over time, the model's accurate predictions contribute to higher customer satisfaction rates and improved brand reputation. As data-driven strategies gain prominence, the business stays ahead in a competitive market, poised for growth and innovation.

Milestone 2: Data Collection and Preparation

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used car_data.csv data. This data is downloaded from kaggle.com. Please refer the link given below to download the dataset.

Link: https://www.kaggle.com/code/vishesh1412/car-purchase-decision-eda-and-decision-tree/input

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```python
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

```python
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```python
# fitting logistic regresion to the training set
from sklearn.linear_model import LogisticRegression
classi = LogisticRegression()
classi.fit(X_train, y_train)
```

```python
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```python
# visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred = classi.predict(X1X2_array_t)
X1X2_pred_reshape = X1X2_pred.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```python
# fitting classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classi1 = KNeighborsClassifier(n_neighbors = 5, p=2, metric = 'minkowski')
classi1.fit(X_train, y_train)
```

```
# fitting classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classi2 = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classi2.fit(X_train, y_train)
```

```
# fitting random forest classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classi3 = RandomForestClassifier(criterion = 'entropy', n_estimators = 10, random_state = 0)
classi3.fit(X_train, y_train)
```

```
# fitting SVM to the training set
from sklearn.svm import SVC
classi4 = SVC(kernel='rbf', random_state = 0)
classi4.fit(X_train, y_train)
```

```
# fitting Naive Bayes to the training set
from sklearn.naive_bayes import GaussianNB
classi6 = GaussianNB()
classi6.fit(X_train, y_train)
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

```
# Importing the dataset
dataset = pd.read_csv('car_data.csv')
```

```
dataset.head()
```

	User ID	Gender	Age	AnnualSalary	Purchased
0	385	Male	35	20000	0
1	681	Male	40	43500	0
2	353	Male	49	74000	0
3	895	Male	40	107500	1
4	661	Male	25	79000	0

```
dataset.shape
```

```
(1000, 5)
```

```
X = dataset.iloc[:, [2, 3]]
y = dataset.iloc[:, 4]
```

```
dataset.describe()
```

	User ID	Age	AnnualSalary	Purchased
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	40.106000	72689.000000	0.402000
std	288.819436	10.707073	34488.341867	0.490547
min	1.000000	18.000000	15000.000000	0.000000
25%	250.750000	32.000000	46375.000000	0.000000
50%	500.500000	40.000000	72000.000000	0.000000
75%	750.250000	48.000000	90000.000000	1.000000
max	1000.000000	63.000000	152500.000000	1.000000

```
dataset["Age"].isnull().sum()
```

```
0
```

```
dataset["Purchased"].value_counts()
```

```
Purchased
0    598
1    402
Name: count, dtype: int64
```

Activity 2: Data Preparation

As we have understood how the data is, let us pre-process the collected data. The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

Handling Missing Values
- Encoding data
- Handling Imbalance Data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Handling Missing Values:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   User ID       1000 non-null   int64 
 1   Gender        1000 non-null   object
 2   Age           1000 non-null   int64 
 3   AnnualSalary  1000 non-null   int64 
 4   Purchased     1000 non-null   int64 
dtypes: int64(4), object(1)
memory usage: 39.2+ KB
```

```
dataset["Age"].isnull().any()
```

```
False
```

```
dataset["Age"].isnull().sum()
```

```
0
```

There are no missing values in the dataset. That is why we can skip this step

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features

```
dataset.describe()
```

	User ID	Age	AnnualSalary	Purchased
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	40.106000	72689.000000	0.402000
std	288.819436	10.707073	34488.341867	0.490547
min	1.000000	18.000000	15000.000000	0.000000
25%	250.750000	32.000000	46375.000000	0.000000
50%	500.500000	40.000000	72000.000000	0.000000
75%	750.250000	48.000000	90000.000000	1.000000
max	1000.000000	63.000000	152500.000000	1.000000

```
dataset.nunique()
```

```
User ID         1000
Gender             2
Age               46
AnnualSalary     247
Purchased          2
dtype: int64
```

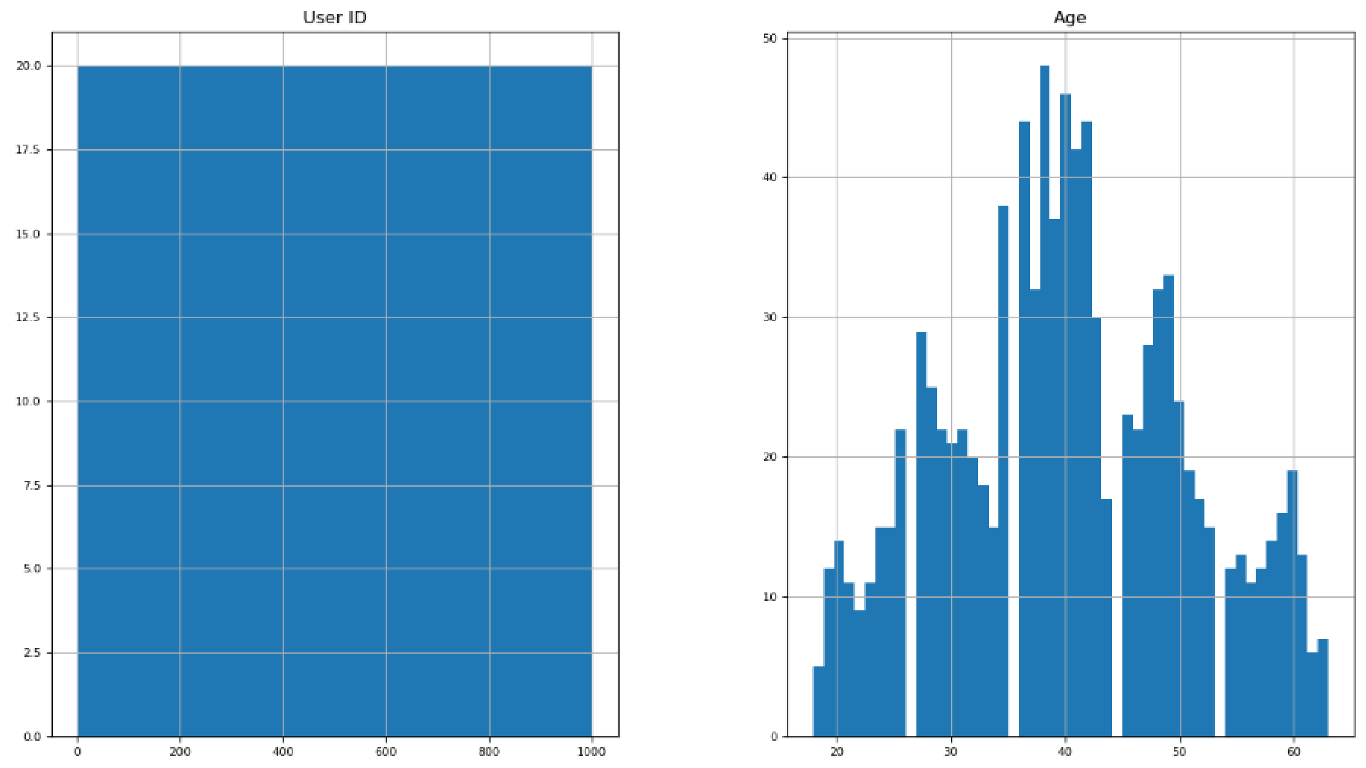
Activity 2: Visual analysis

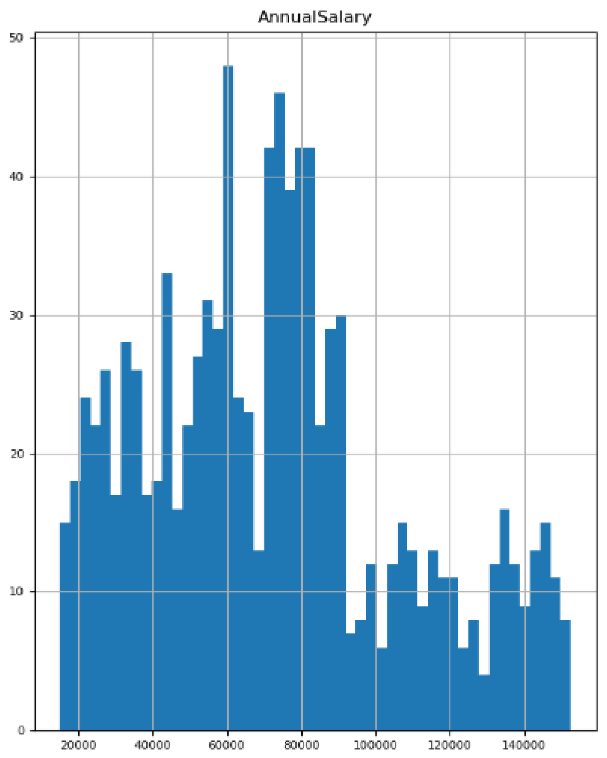
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

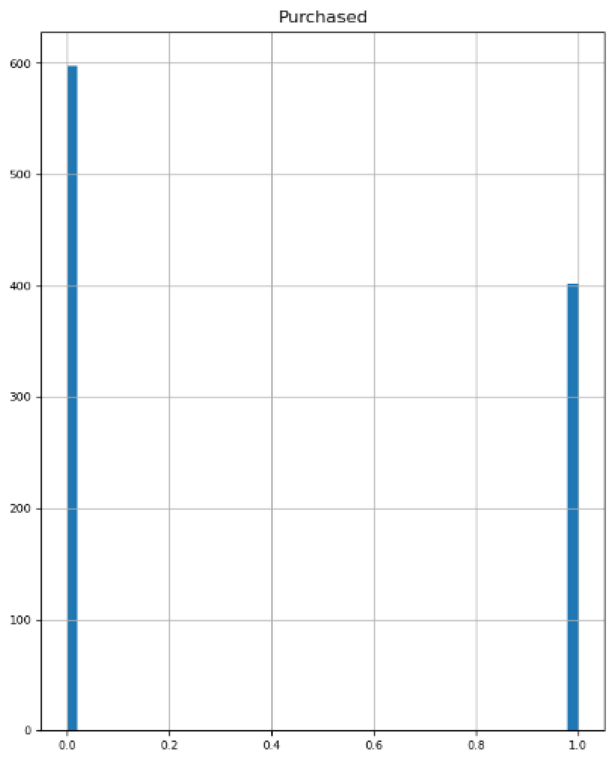
Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed different graphs such as histogram and boxplot. The Seaborn and matplotlib package provides a wonderful functions histogram and boxplot. With the help of histogram and boxplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
dataset.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8); # ; avoid having the matplotlib verbose informations
```

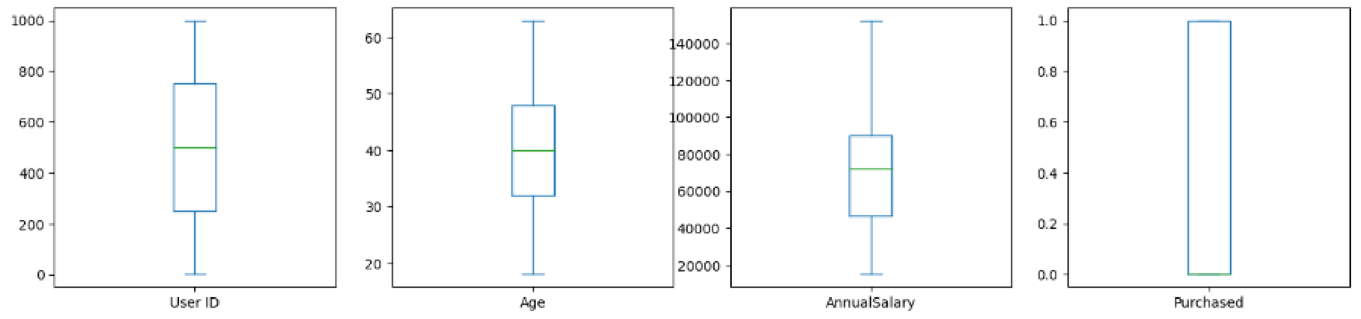






```
dataset.plot(kind="box", figsize=(17,17), layout=(4,4), sharex=False, subplots=True)
```

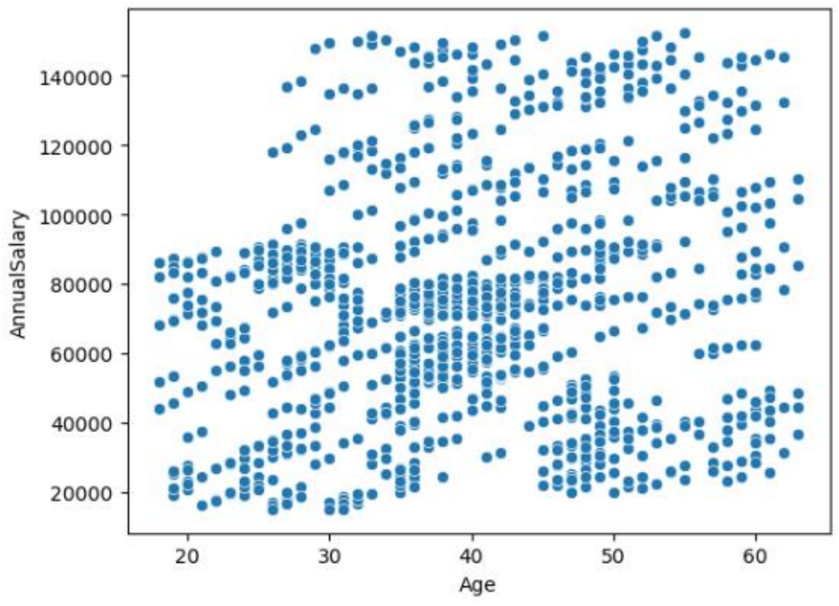
```
User ID            Axes(0.125,0.712609;0.168478x0.167391)
Age            Axes(0.327174,0.712609;0.168478x0.167391)
AnnualSalary   Axes(0.529348,0.712609;0.168478x0.167391)
Purchased      Axes(0.731522,0.712609;0.168478x0.167391)
dtype: object
```



Bivariate Analysis:

```
sns.scatterplot(x=dataset["Age"], y=dataset["AnnualSalary"])
```

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



Activity 1: Splitting data into train and test

Splitting the Dataset and Feature Scaling ¶

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
X_train.shape
```

```
(750, 2)
```

```
X_test.shape
```

```
(250, 2)
```

```
y_train.shape
```

```
(750,)
```

```
y_test.shape
```

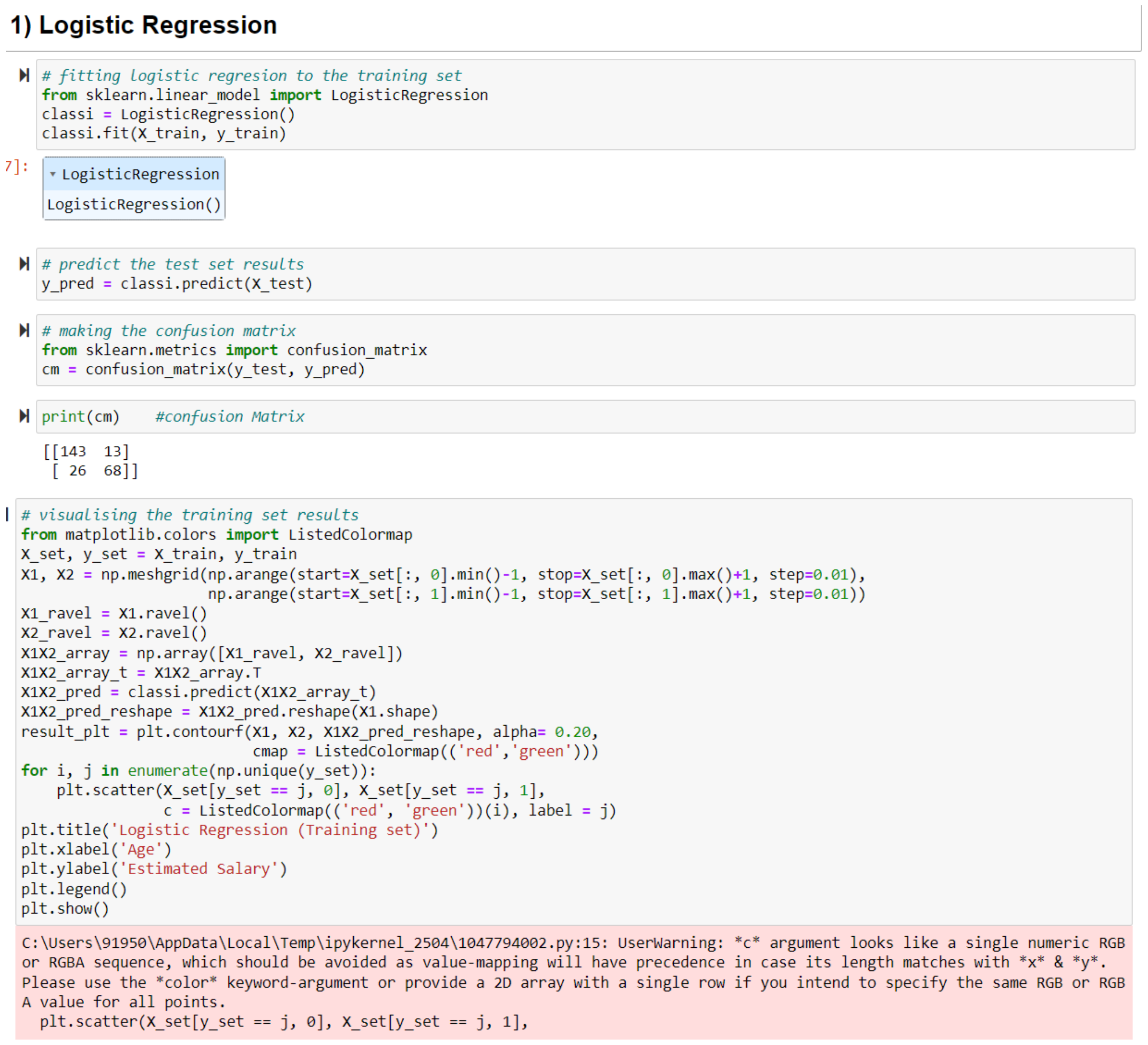
```
(250,)
```

Milestone 4: Model Building

Activity 4: Training the model in multiple algorithms

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Activity 4.1: Logistic Regression



1) Logistic Regression

```
# fitting logistic regresion to the training set
from sklearn.linear_model import LogisticRegression
classi = LogisticRegression()
classi.fit(X_train, y_train)
```

```
7]: ▾ LogisticRegression
    LogisticRegression()
```

```
# predict the test set results
y_pred = classi.predict(X_test)
```

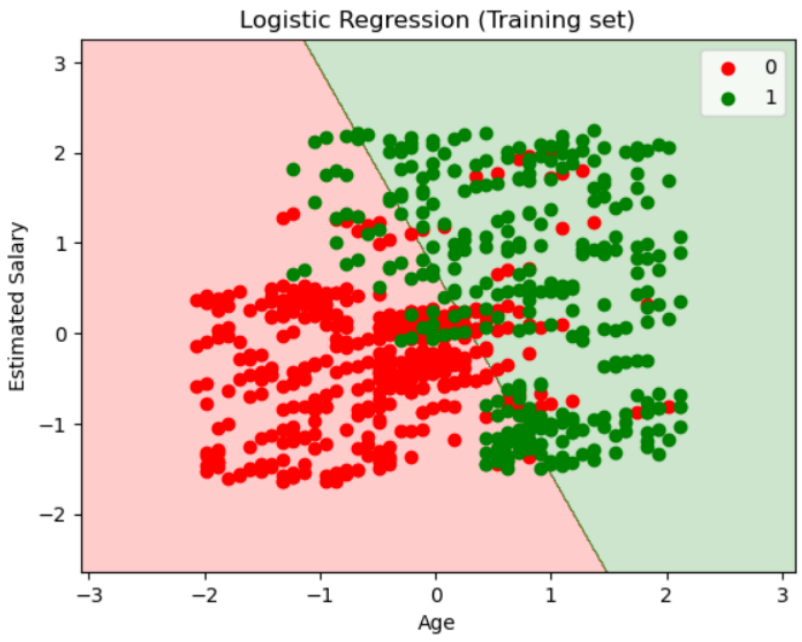
```
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)    #confusion Matrix
```

```
[[143  13]
 [ 26  68]]
```

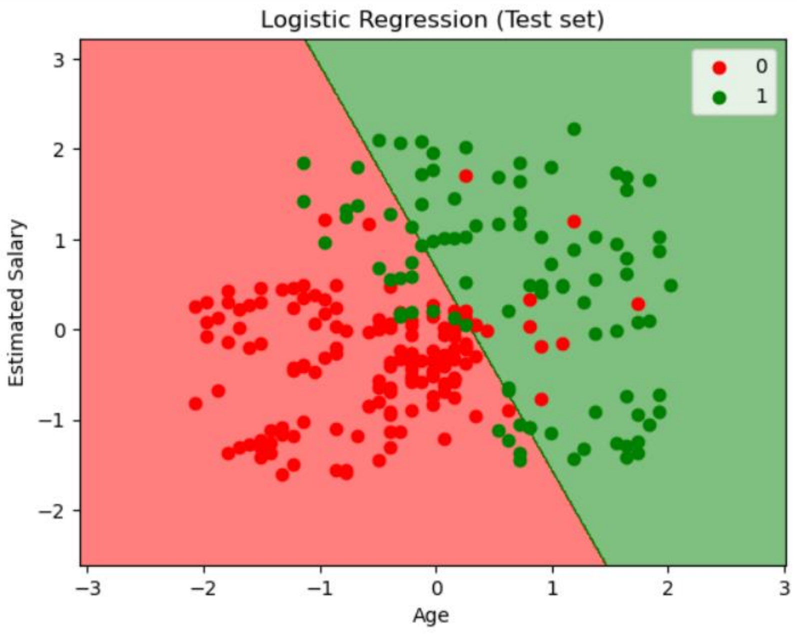
```
# visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred = classi.predict(X1X2_array_t)
X1X2_pred_reshape = X1X2_pred.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_2504\1047794002.py:15: UserWarning: *c* argument looks like a single numeric RGB
or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.
Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGB
A value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



```
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred = classi.predict(X1X2_array_t)
X1X2_pred_reshape = X1X2_pred.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape, alpha= 0.50,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_2504\2343604338.py:15: UserWarning: *c* argument looks like a single numeric RGB
or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.
Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGB
A value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



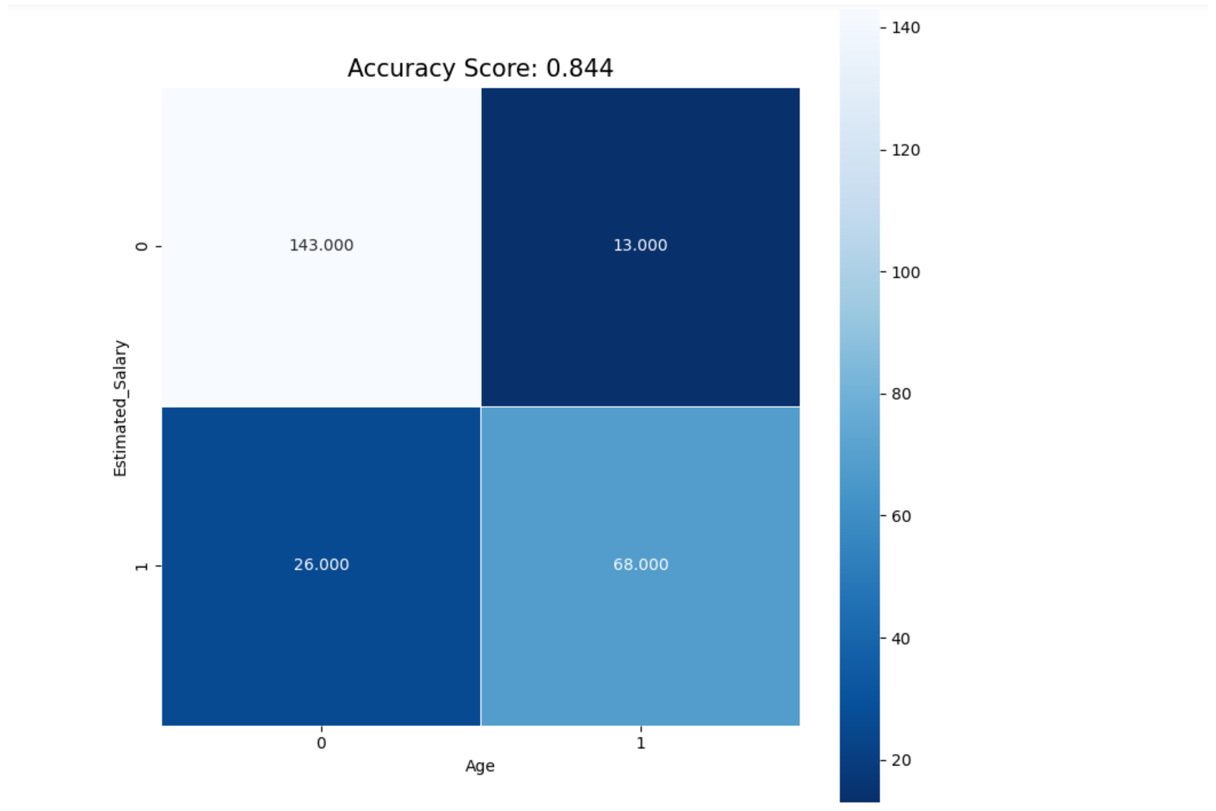
```
print(y_pred)           #Test Set result Prediction using Logistic Regression
```

```
[0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0
 1 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0 0 1 1 0 1 1 0
 0 1 0 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 1 0 0 0 0
 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0
 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0]
```

```
# Use score method to get accuracy of model
score = classi.score(X_test, y_test)
print("Accuracy using Logistic Regression: ",score)
```

```
Accuracy using Logistic Regression:  0.844
```

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Estimated_Salary');
plt.xlabel('Age');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



Activity 4.2: K-Nearest Neighbour[KNN]

2) K-Nearest Neighbour[KNN]

```
# fitting classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classi1 = KNeighborsClassifier(n_neighbors = 5, p=2, metric = 'minkowski')
classi1.fit(X_train, y_train)
```

```
8]:
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
#predict the test set results
y_pred1 = classi1.predict(X_test)
```

```
print(y_pred1)
```

```
[0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1
 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0
 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0
 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0
 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 0
 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0
 0 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1]
```

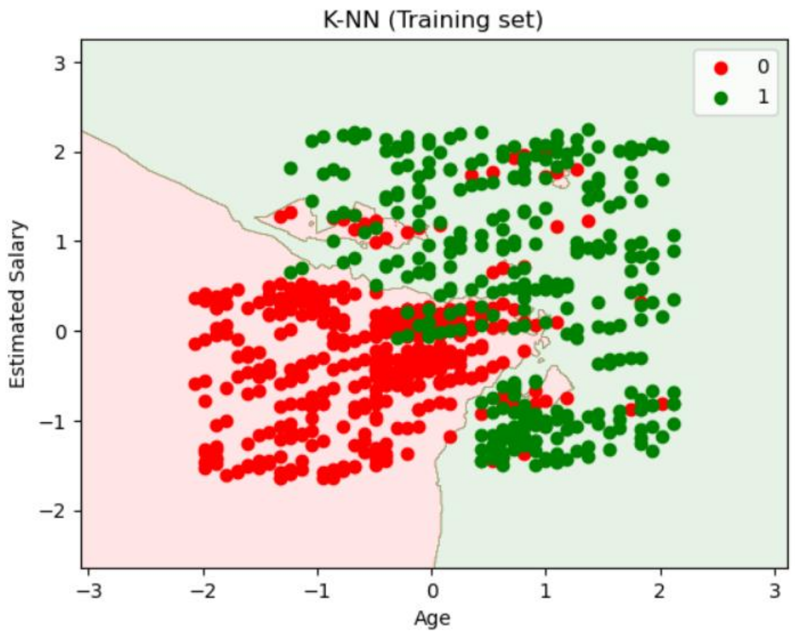
```
# Use score method to get accuracy of model
score = classi1.score(X_test, y_test)
print("Accuracy using KNN: ",score)
```

```
Accuracy using KNN:  0.924
```

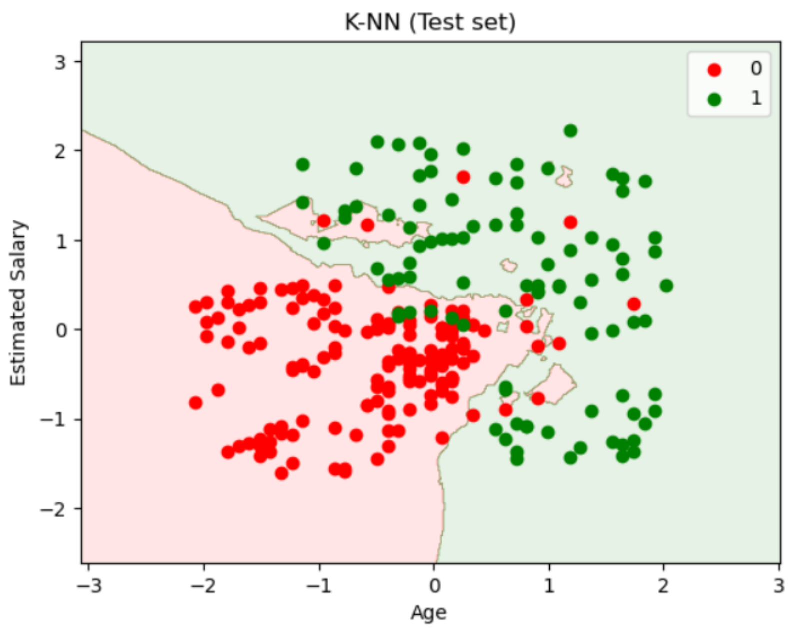
```python
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_pred1)
print(cm1)
```

```
[[147   9]
 [ 10  84]]
```

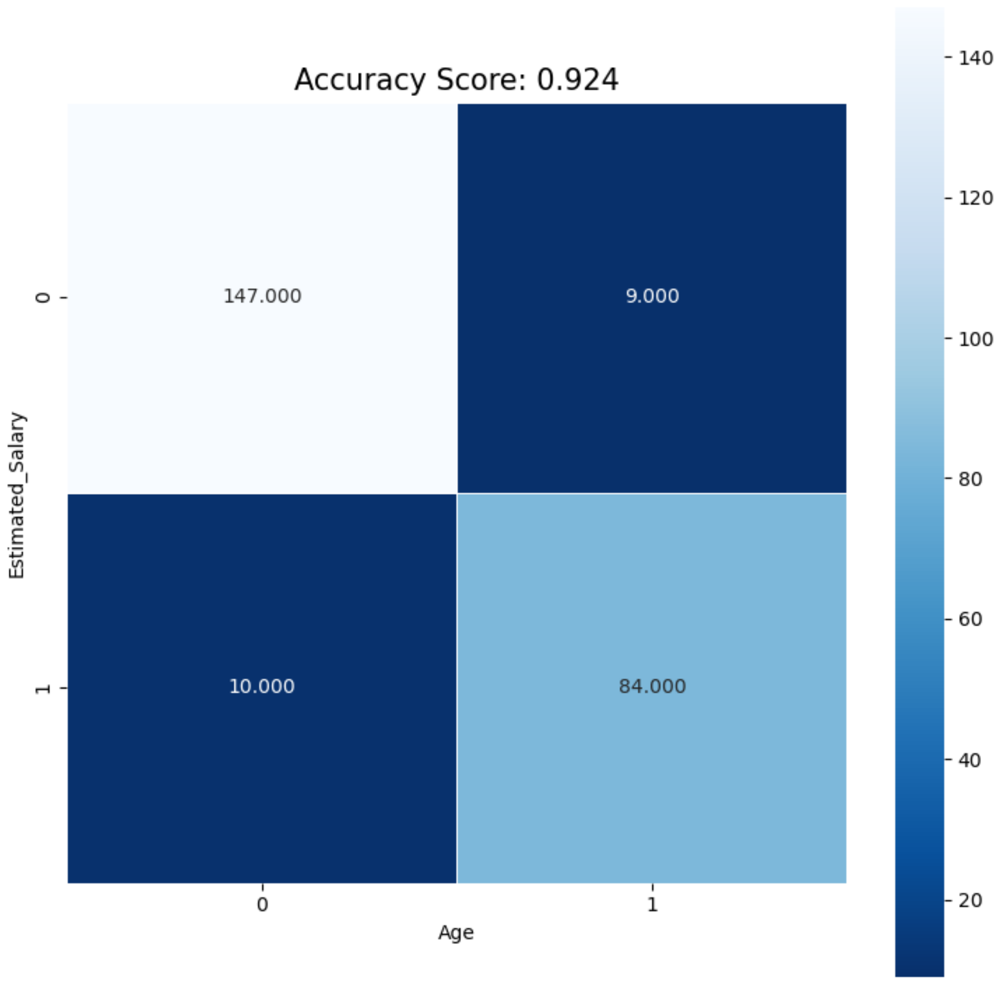
```python
#visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred = classi1.predict(X1X2_array_t)
X1X2_pred_reshape = X1X2_pred.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape, alpha= 0.10,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```python
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred = classi1.predict(X1X2_array_t)
X1X2_pred_reshape = X1X2_pred.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape, alpha= 0.10,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```python
#heatmap
plt.figure(figsize=(9,9))
sns.heatmap(cm1, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Estimated_Salary');
plt.xlabel('Age');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```


Accuracy Score: 0.924
147.000
9.000
10.000
84.000
Estimated_Salary
Age
0
1
0
1
140
120
100
80
60
40
20

Activity 4.3: Decision Tree

3) Decision tree

```
# fitting classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classi2 = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classi2.fit(X_train, y_train)
```

6]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
#predict the test set results
y_pred2 = classi2.predict(X_test)
```

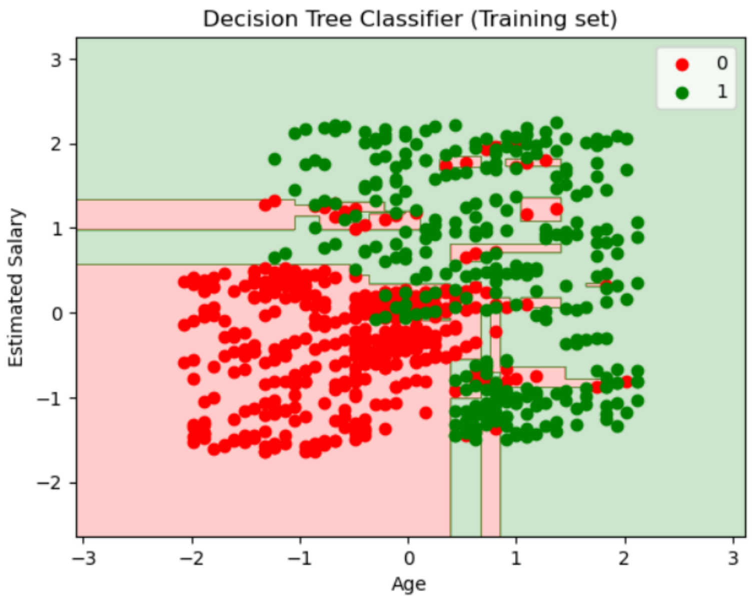
```
print(y_pred2)
```

```
[0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1
 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0
 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1
 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0
 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 0
 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0
 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]
```

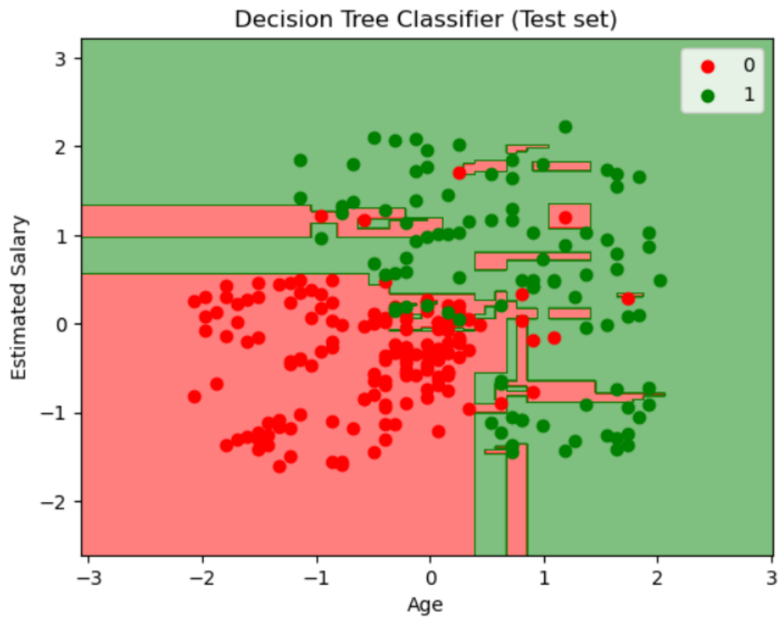
```
# Use score method to get accuracy of model
score = classi2.score(X_test, y_test)
print("Accuracy using Decision tree: ",score)
```

```
Accuracy using Decision tree:  0.888
```

```
#visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred1 = classi2.predict(X1X2_array_t)
X1X2_pred_reshape1 = X1X2_pred1.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape1, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

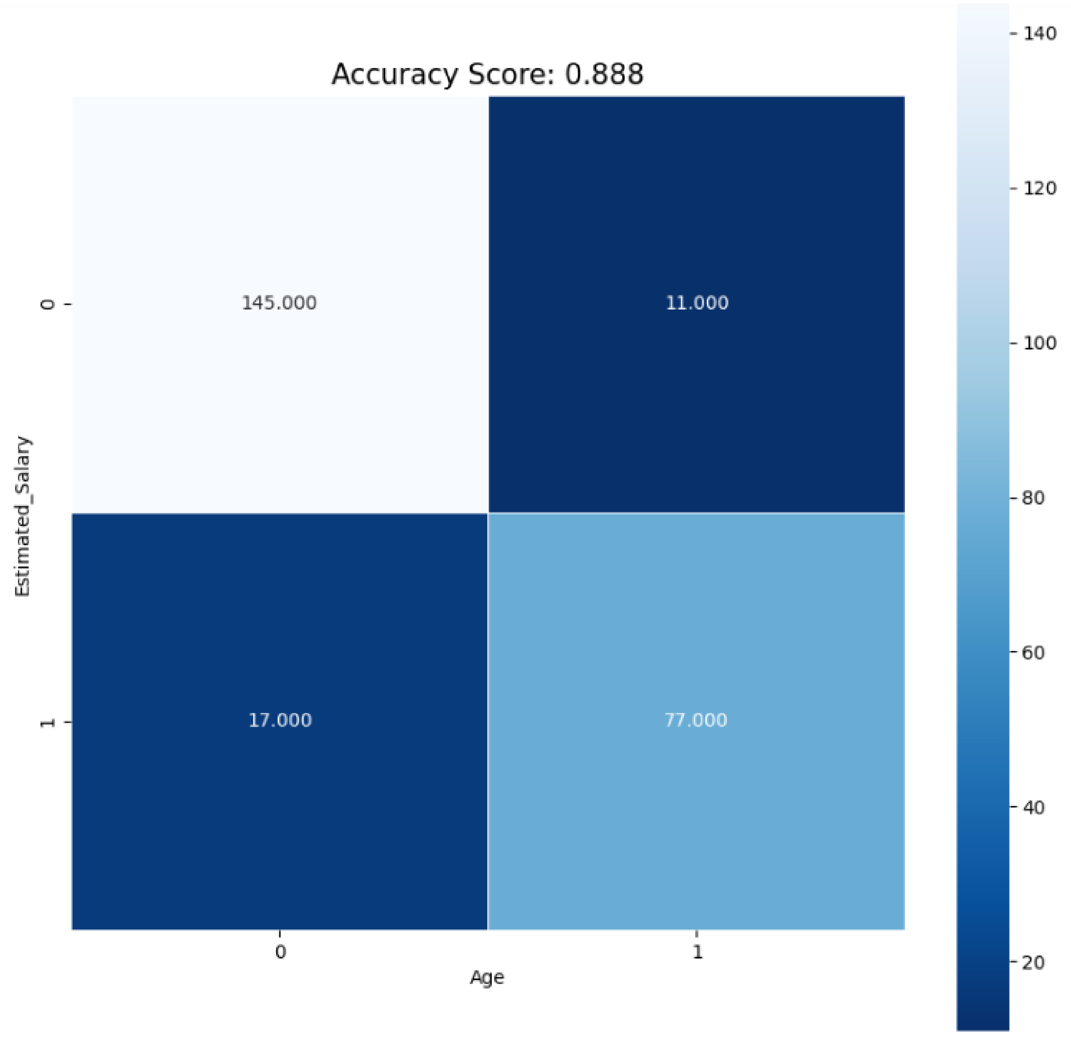


```
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred1 = classi2.predict(X1X2_array_t)
X1X2_pred_reshape1 = X1X2_pred1.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape1, alpha= 0.50,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```
cm2 = confusion_matrix(y_test, y_pred2)
```

```
#heatmap
plt.figure(figsize=(10,10))
sns.heatmap(cm2, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Estimated_Salary');
plt.xlabel('Age');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



Activity 4.4: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done

4) random forest classification

```
# fitting random forest classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classi3 = RandomForestClassifier(criterion = 'entropy', n_estimators = 10, random_state = 0)
classi3.fit(X_train, y_train)
```

```
1]:
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
#predict the test set results
y_pred3 = classi3.predict(X_test)
```

```
print(y_pred3)
```

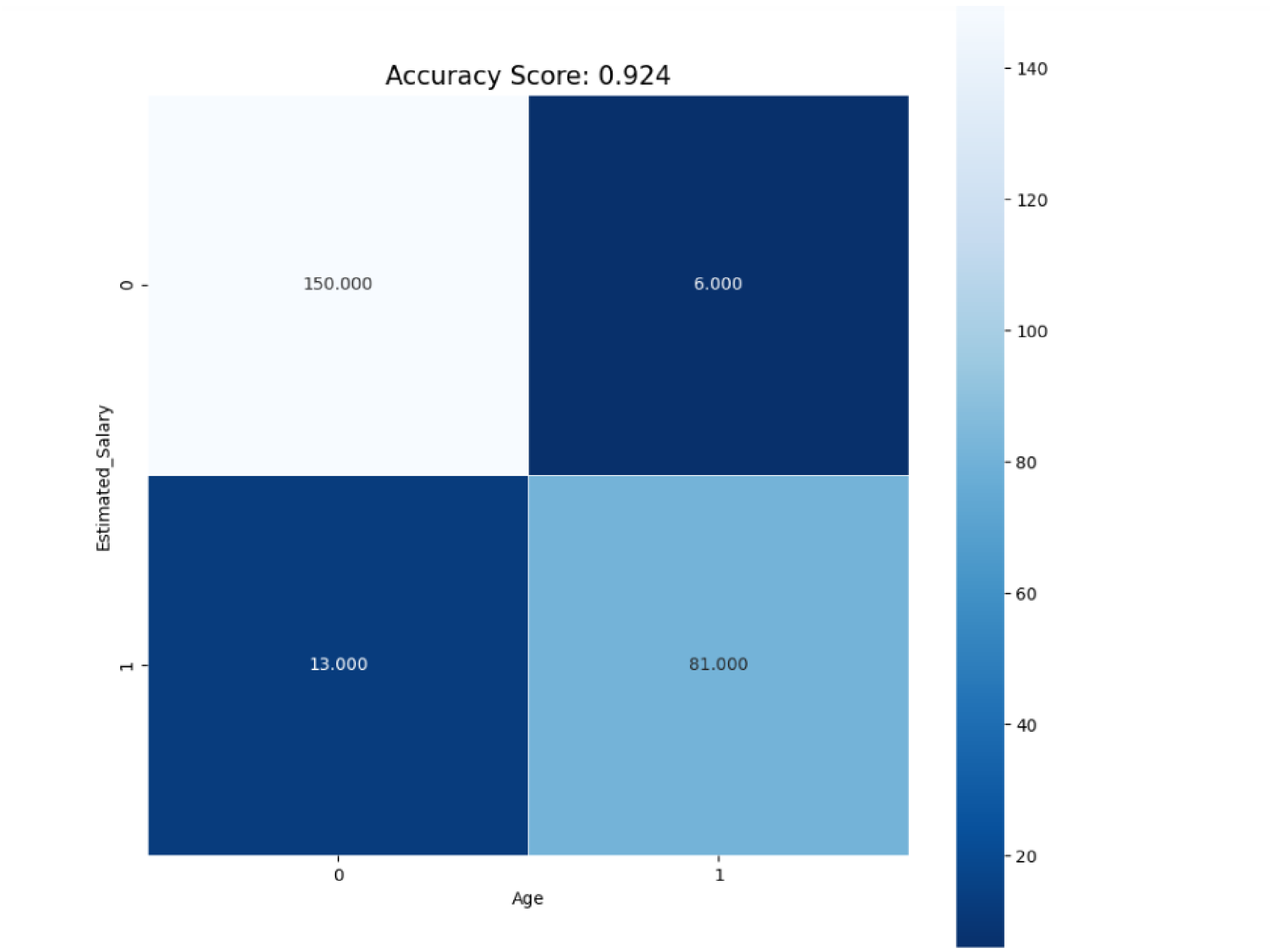
```
[0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1
 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0
 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0
 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0
 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 0
 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0]
```

```
# Use score method to get accuracy of model
score = classi3.score(X_test, y_test)
print("Accuracy using Random Forest: ",score)
```

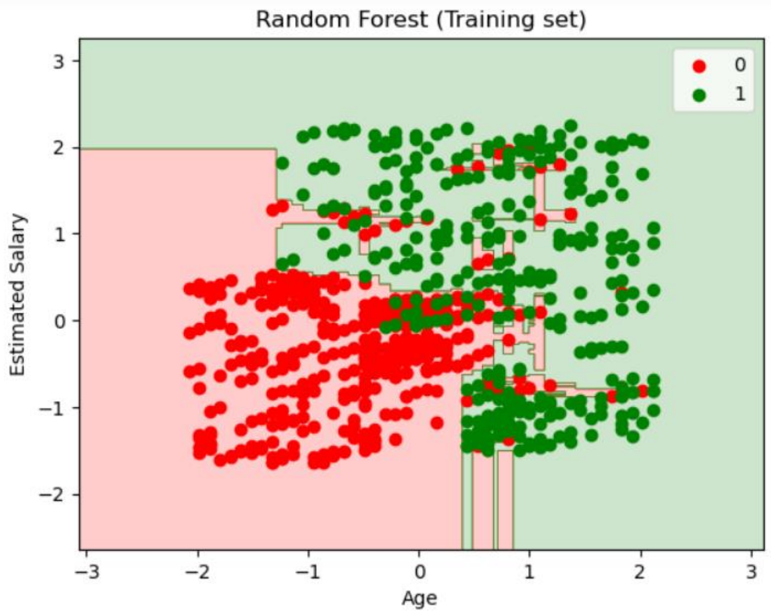
```
Accuracy using Random Forest:  0.924
```

```
cm3 = confusion_matrix(y_test, y_pred3)
```

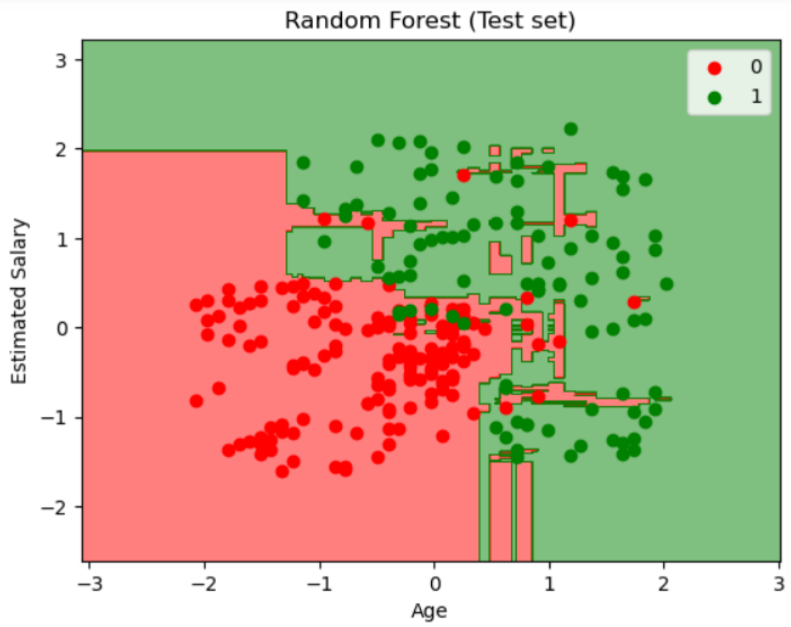
```
#heatmap
plt.figure(figsize=(10,10))
sns.heatmap(cm3, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Estimated_Salary');
plt.xlabel('Age');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```


Accuracy Score: 0.924
150.000
6.000
13.000
81.000
Estimated_Salary
Age
0
1
0
1
140
120
100
80
60
40
20

```python
#visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred2 = classi3.predict(X1X2_array_t)
X1X2_pred_reshape2 = X1X2_pred2.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape2, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```python
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred2 = classi3.predict(X1X2_array_t)
X1X2_pred_reshape2 = X1X2_pred2.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape2, alpha= 0.50,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Activity 4.5: Support Vector Machine (SVM)

5) Support Vector Machine (SVM)

```python
# fitting SVM to the training set
from sklearn.svm import SVC
classi4 = SVC(kernel='rbf', random_state = 0)
classi4.fit(X_train, y_train)
```

```
52]:
SVC
SVC(random_state=0)
```

```python
#predict the test set results
y_pred4 = classi4.predict(X_test)
```

```python
print(y_pred4)
```

```
[0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1
 0 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0
 1 0 1 0 1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0
 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 0
 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0
 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0]
```

```python
# making the confusion matrix
from sklearn.metrics import confusion_matrix
cm4 = confusion_matrix(y_test, y_pred4)
```

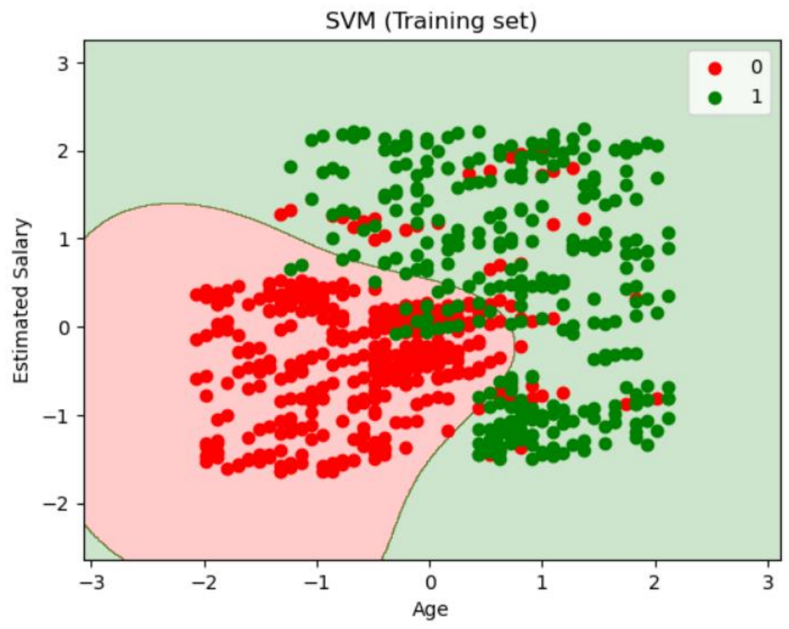
```python
print(cm4)
```

```
[[145  11]
 [ 11  83]]
```

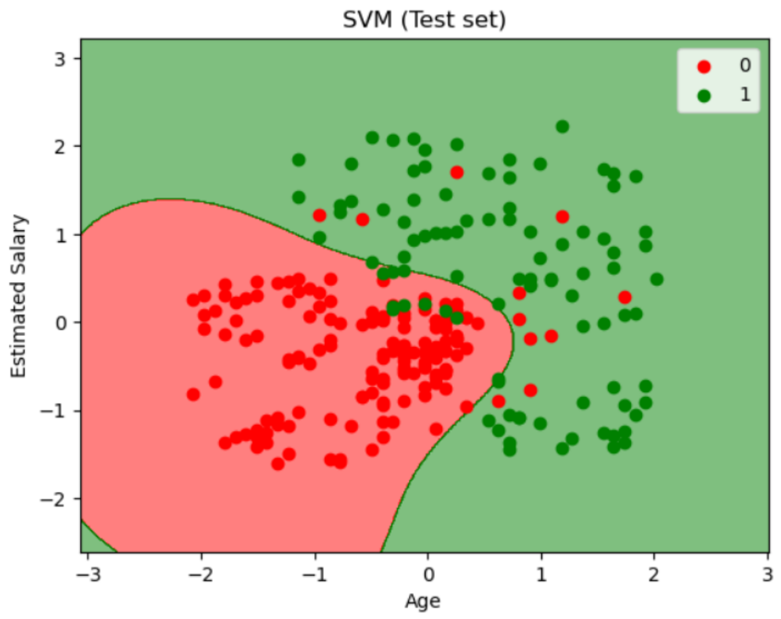
```python
# Use score method to get accuracy of model
scoreee = classi4.score(X_test, y_test)
print("Accuracy using SVM : ",scoreee)
```

```
Accuracy using SVM :  0.912
```

```
#visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred3 = classi4.predict(X1X2_array_t)
X1X2_pred_reshape3 = X1X2_pred3.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape3, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred3 = classi4.predict(X1X2_array_t)
X1X2_pred_reshape3 = X1X2_pred3.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape3, alpha= 0.50,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Activity 4.6: Naïve bayes

6) Naive bayes

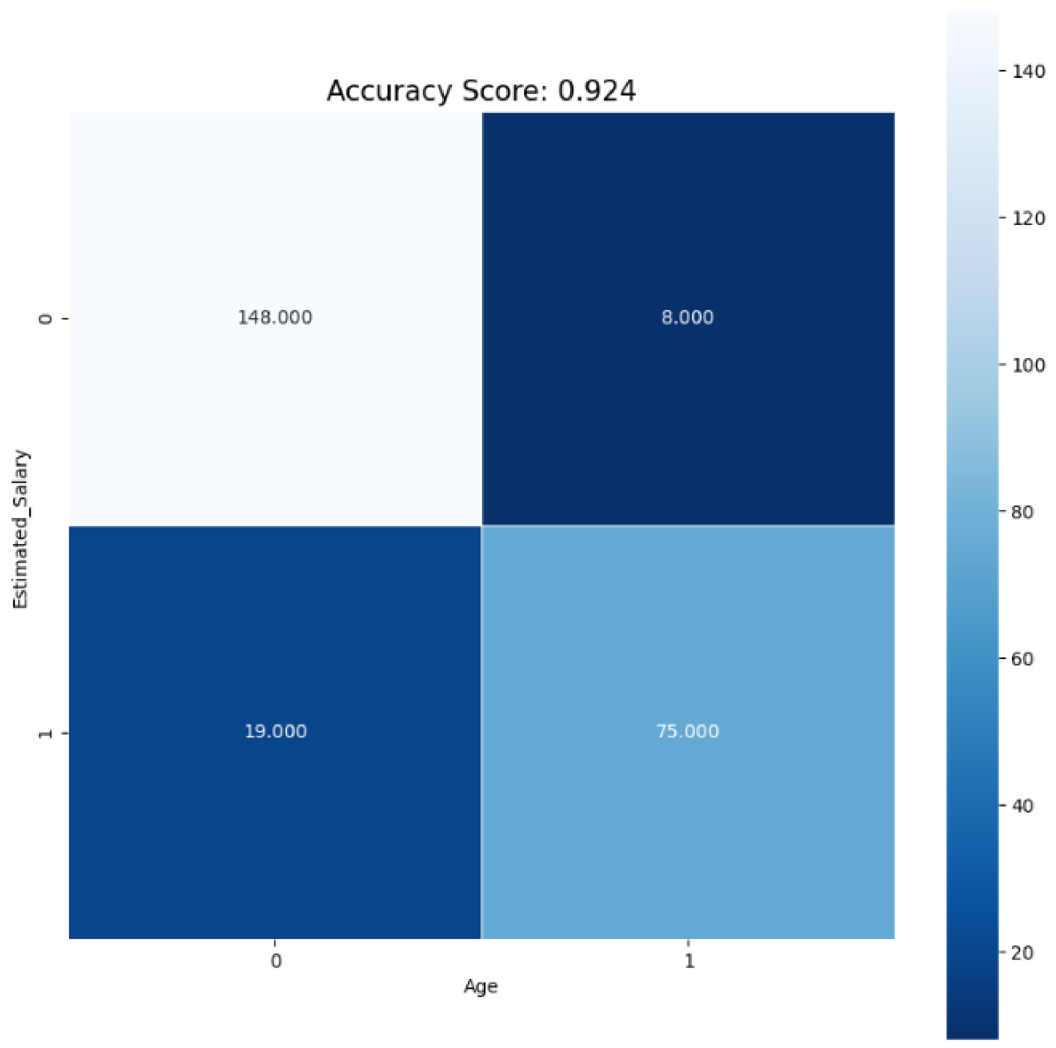
```
# fitting Naive Bayes to the training set
from sklearn.naive_bayes import GaussianNB
classi6 = GaussianNB()
classi6.fit(X_train, y_train)
```

```
0]:  ▾ GaussianNB
     GaussianNB()
```

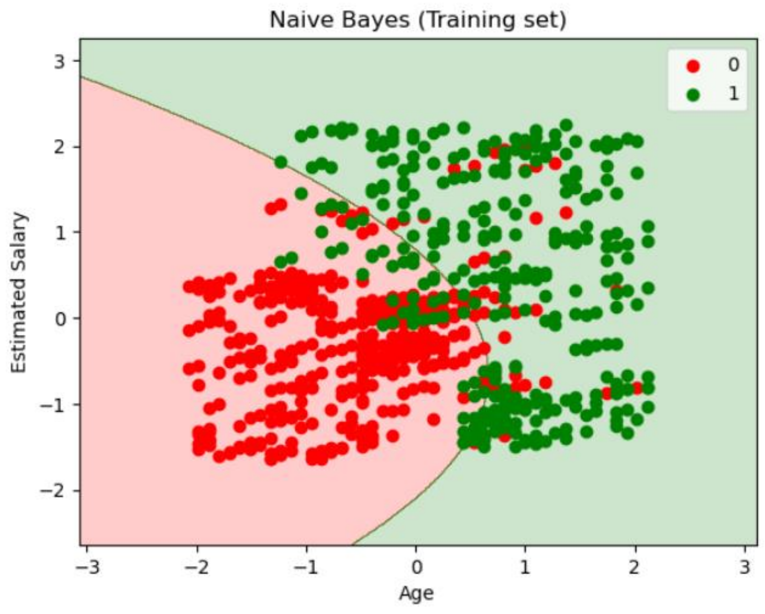
```
#predict the test set results
y_pred6 = classi6.predict(X_test)
```

```
cm6 = confusion_matrix(y_test, y_pred6)
```

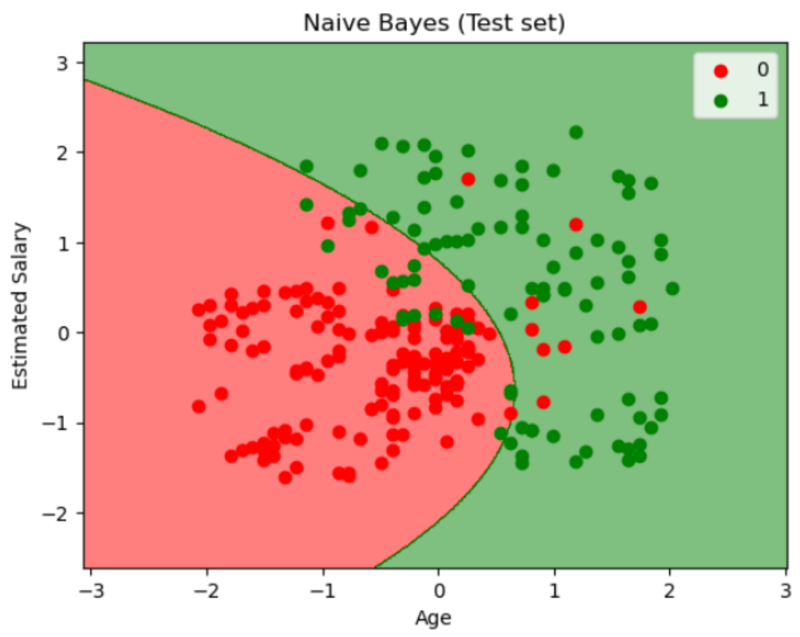
```
#heatmap
plt.figure(figsize=(10,10))
sns.heatmap(cm6, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Estimated_Salary');
plt.xlabel('Age');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```


Accuracy Score: 0.924
148.000
8.000
19.000
75.000
Estimated_Salary
Age
0
1
140
120
100
80
60
40
20

```python
#visualising the training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred6 = classi6.predict(X1X2_array_t)
X1X2_pred_reshape6 = X1X2_pred6.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape6, alpha= 0.20,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```python
#visualising the test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min()-1, stop=X_set[:, 0].max()+1, step=0.01),
                     np.arange(start=X_set[:, 1].min()-1, stop=X_set[:, 1].max()+1, step=0.01))
X1_ravel = X1.ravel()
X2_ravel = X2.ravel()
X1X2_array = np.array([X1_ravel, X2_ravel])
X1X2_array_t = X1X2_array.T
X1X2_pred6 = classi6.predict(X1X2_array_t)
X1X2_pred_reshape6 = X1X2_pred6.reshape(X1.shape)
result_plt = plt.contourf(X1, X2, X1X2_pred_reshape6, alpha= 0.50,
                          cmap = ListedColormap(('red','green')))
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



```python
# Use score method to get accuracy of model
scoree = classi6.score(X_test, y_test)
print("Accuracy using Naive Bayes: ",scoree)
```

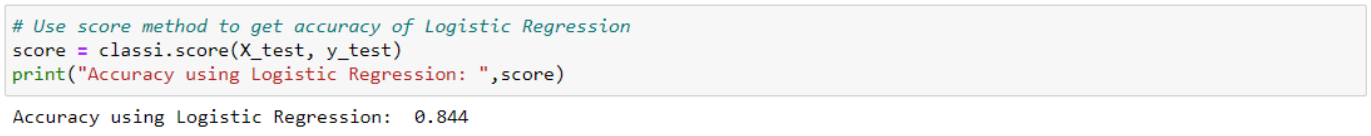
```
Accuracy using Naive Bayes:  0.892
```

Activity 5: Testing the model

```python
# User input for Age and Estimated Salary
user_age = float(input("Enter the age of the person: "))
user_salary = float(input("Enter the estimated salary of the person: "))

# Create a numpy array with the user input
user_data = np.array([[user_age, user_salary]])

# Standardize the user input using the same scaler used for training data
user_data_scaled = sc_X.transform(user_data)

# Predictions using all six classifiers

predictions_logistic_regression = classi.predict(user_data_scaled)
predictions_knn = classi1.predict(user_data_scaled)
predictions_decision_tree = classi2.predict(user_data_scaled)
predictions_random_forest = classi3.predict(user_data_scaled)
predictions_svm = classi4.predict(user_data_scaled)
predictions_naive_bayes = classi6.predict(user_data_scaled)

# Displaying the predictions
print(f"Logistic Regression Prediction: {predictions_logistic_regression}")
print(f"KNN Prediction: {predictions_knn}")
print(f"Decision Tree Prediction: {predictions_decision_tree}")
print(f"Random Forest Prediction: {predictions_random_forest}")
print(f"SVM Prediction: {predictions_svm}")
print(f"Naive Bayes Prediction: {predictions_naive_bayes}")
```

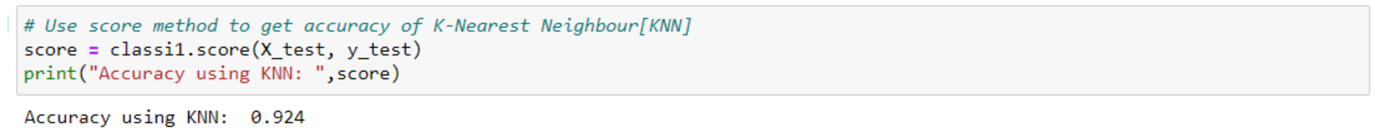
```
Enter the age of the person: 30
Enter the estimated salary of the person: 1000
Logistic Regression Prediction: [0]
KNN Prediction: [0]
Decision Tree Prediction: [0]
Random Forest Prediction: [0]
SVM Prediction: [0]
Naive Bayes Prediction: [0]
```

Milestone 5: Performance Testing



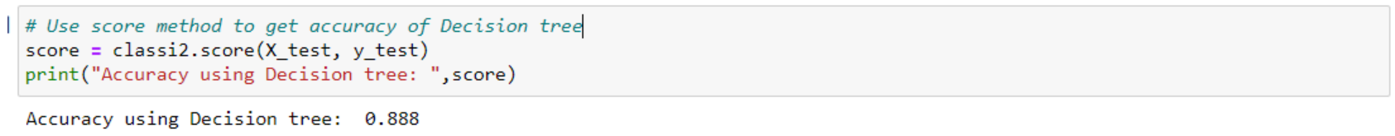
```
# Use score method to get accuracy of Logistic Regression
score = classi.score(X_test, y_test)
print("Accuracy using Logistic Regression: ",score)
```

```
Accuracy using Logistic Regression:  0.844
```



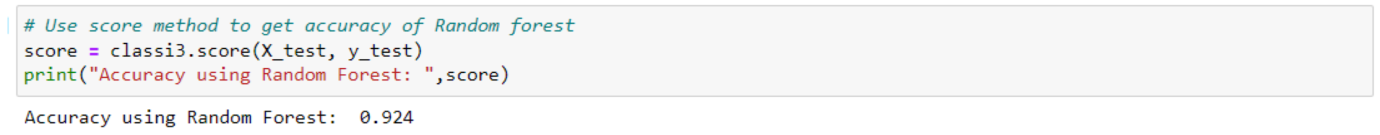
```
# Use score method to get accuracy of K-Nearest Neighbour[KNN]
score = classi1.score(X_test, y_test)
print("Accuracy using KNN: ",score)
```

```
Accuracy using KNN:  0.924
```



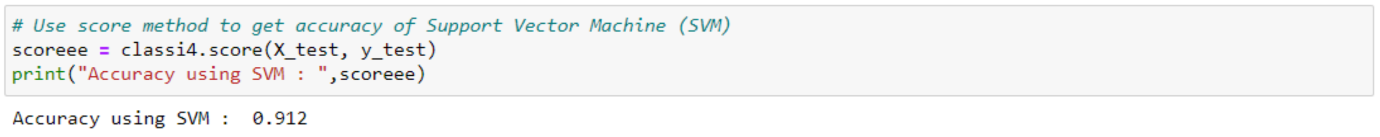
```
# Use score method to get accuracy of Decision tree
score = classi2.score(X_test, y_test)
print("Accuracy using Decision tree: ",score)
```

```
Accuracy using Decision tree:  0.888
```



```
# Use score method to get accuracy of Random forest
score = classi3.score(X_test, y_test)
print("Accuracy using Random Forest: ",score)
```

```
Accuracy using Random Forest:  0.924
```



```
# Use score method to get accuracy of Support Vector Machine (SVM)
scoreee = classi4.score(X_test, y_test)
print("Accuracy using SVM : ",scoreee)
```

```
Accuracy using SVM :  0.912
```

Milestone 6: Model Deployment & Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

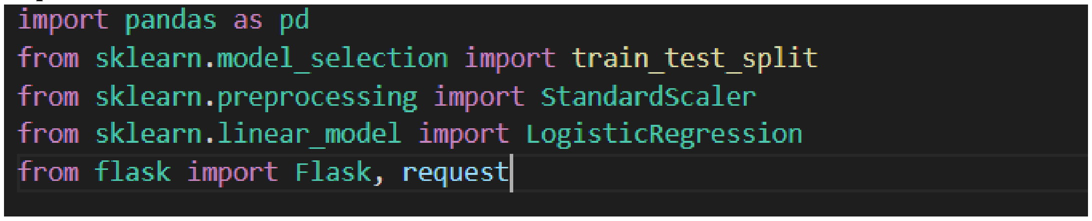
Activity1: Building Html Pages:

For this project create three HTML files namely

- home.html
- result.html

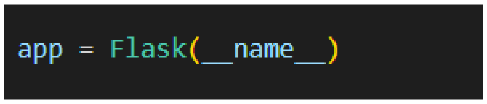
Activity 2: Build Python code:

Import the libraries



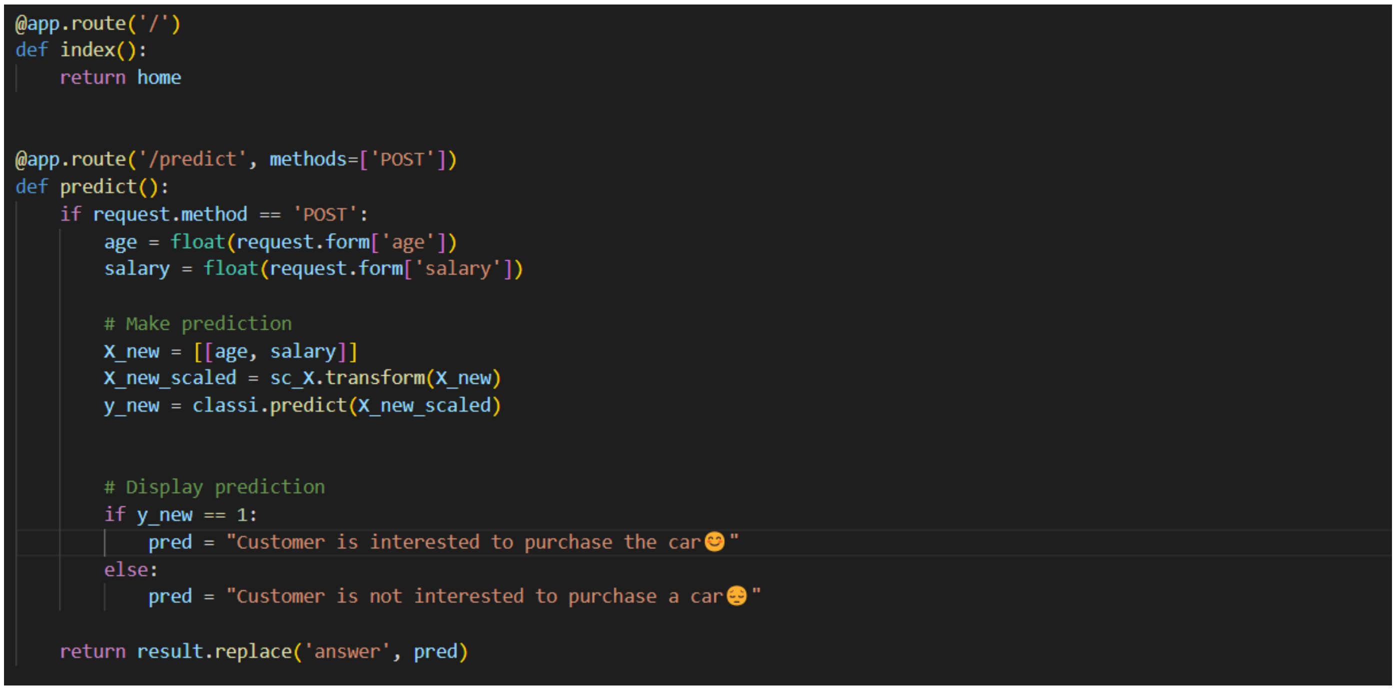
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from flask import Flask, request
```

Importing flask module in the project is mandatory. Flask constructor takes the name of the current module (__name__) as argument.



```
app = Flask(__name__)
```

Render HTML page:

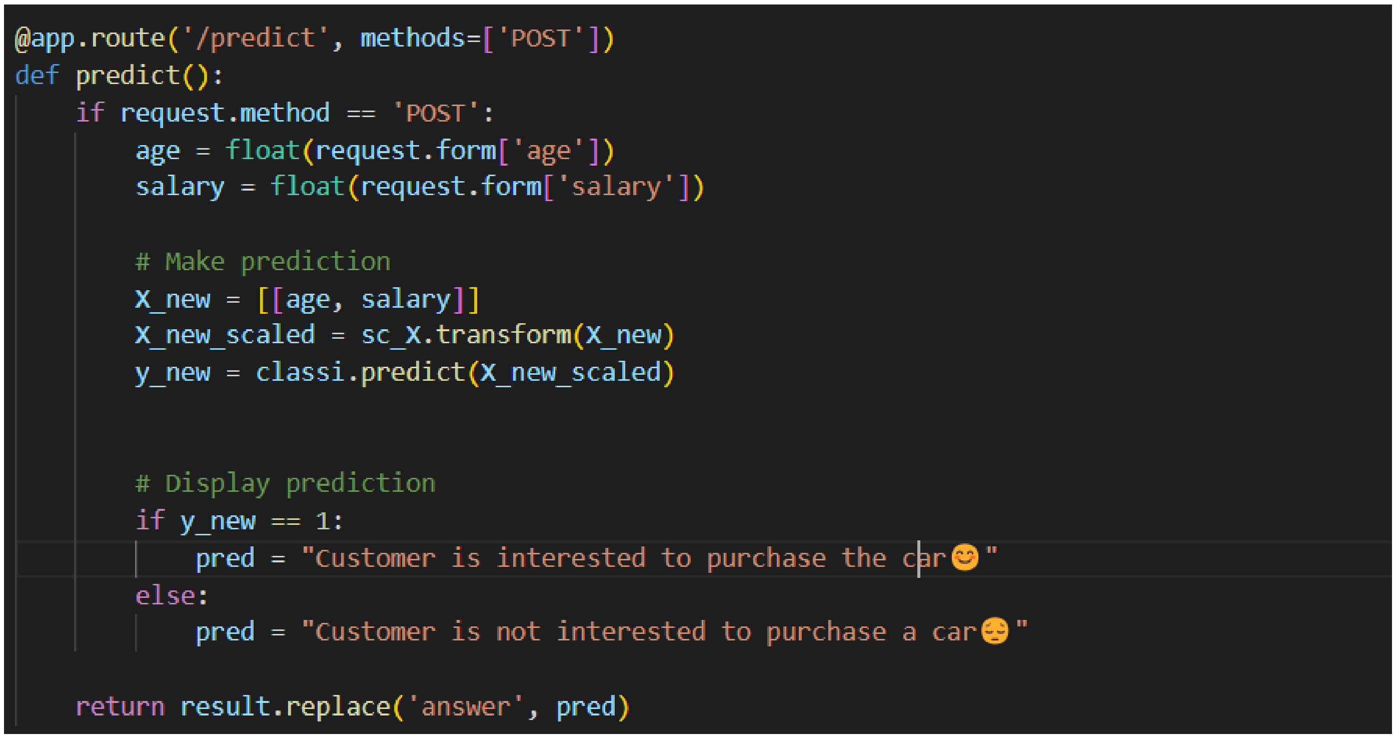


```
@app.route('/')
def index():
    return home


@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        age = float(request.form['age'])
        salary = float(request.form['salary'])

        # Make prediction
        X_new = [[age, salary]]
        X_new_scaled = sc_X.transform(X_new)
        y_new = classi.predict(X_new_scaled)


        # Display prediction
        if y_new == 1:
            pred = "Customer is interested to purchase the car😊"
        else:
            pred = "Customer is not interested to purchase a car😔"

    return result.replace('answer', pred)
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

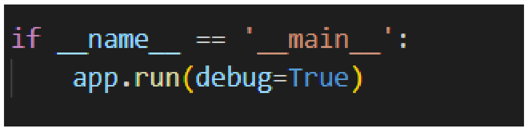
Retrieves the value from UI:



```
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        age = float(request.form['age'])
        salary = float(request.form['salary'])

        # Make prediction
        X_new = [[age, salary]]
        X_new_scaled = sc_X.transform(X_new)
        y_new = classi.predict(X_new_scaled)


        # Display prediction
        if y_new == 1:
            pred = "Customer is interested to purchase the car😊"
        else:
            pred = "Customer is not interested to purchase a car😔"

    return result.replace('answer', pred)
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the carprediction.html page earlier.

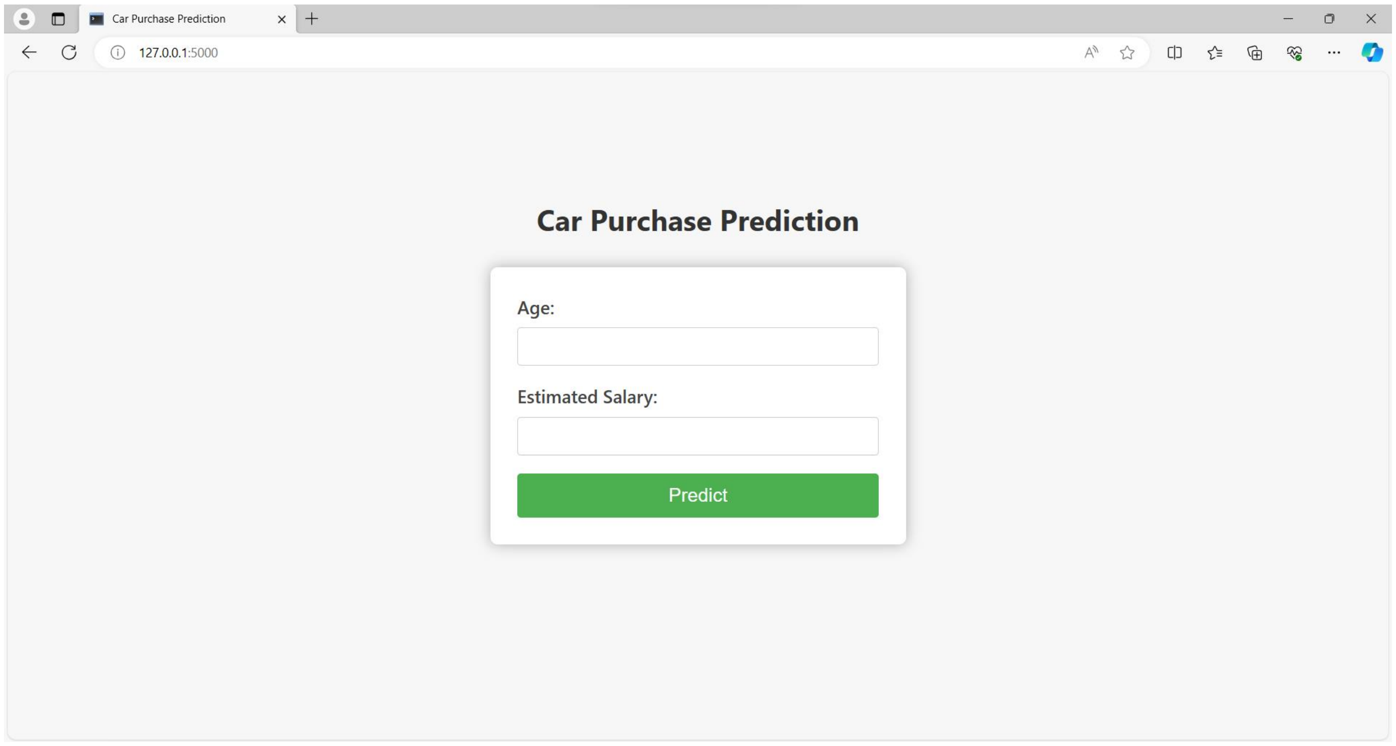
Main Function:



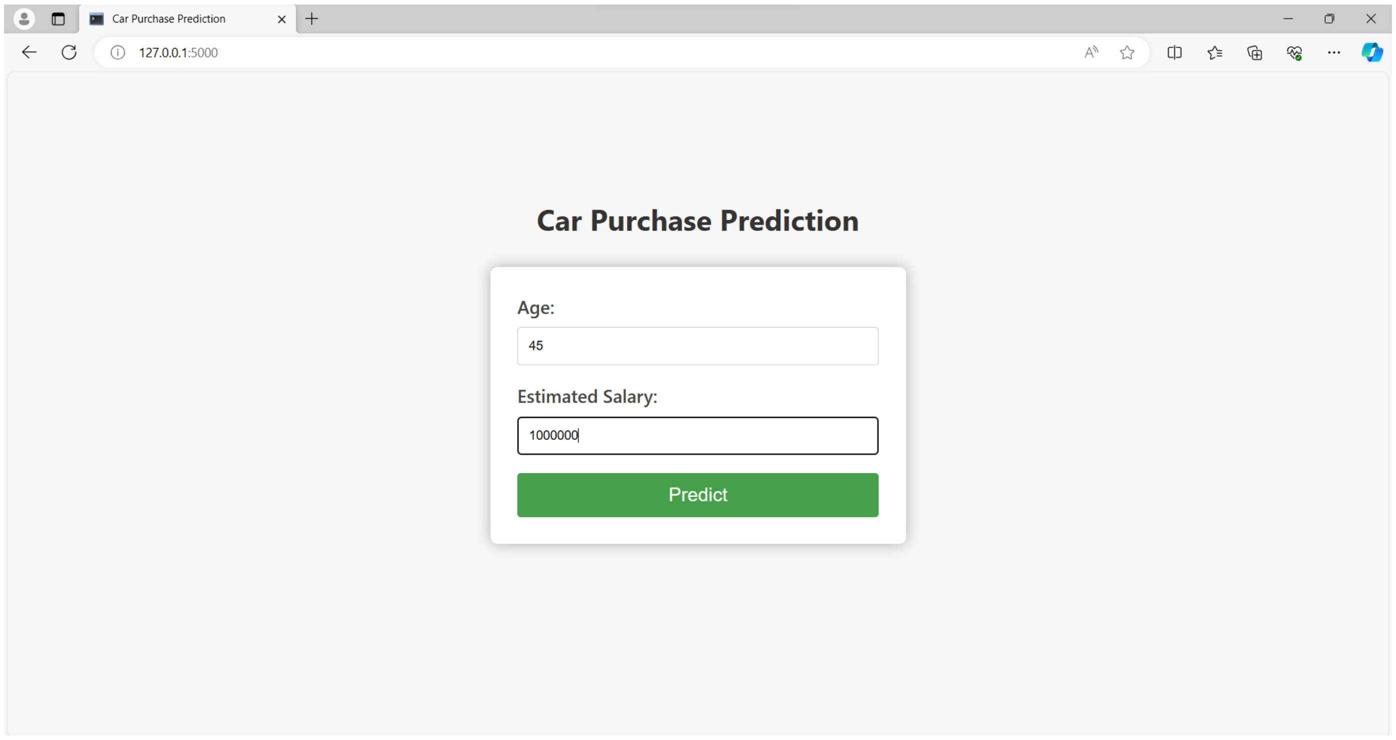
```
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3: Run the application

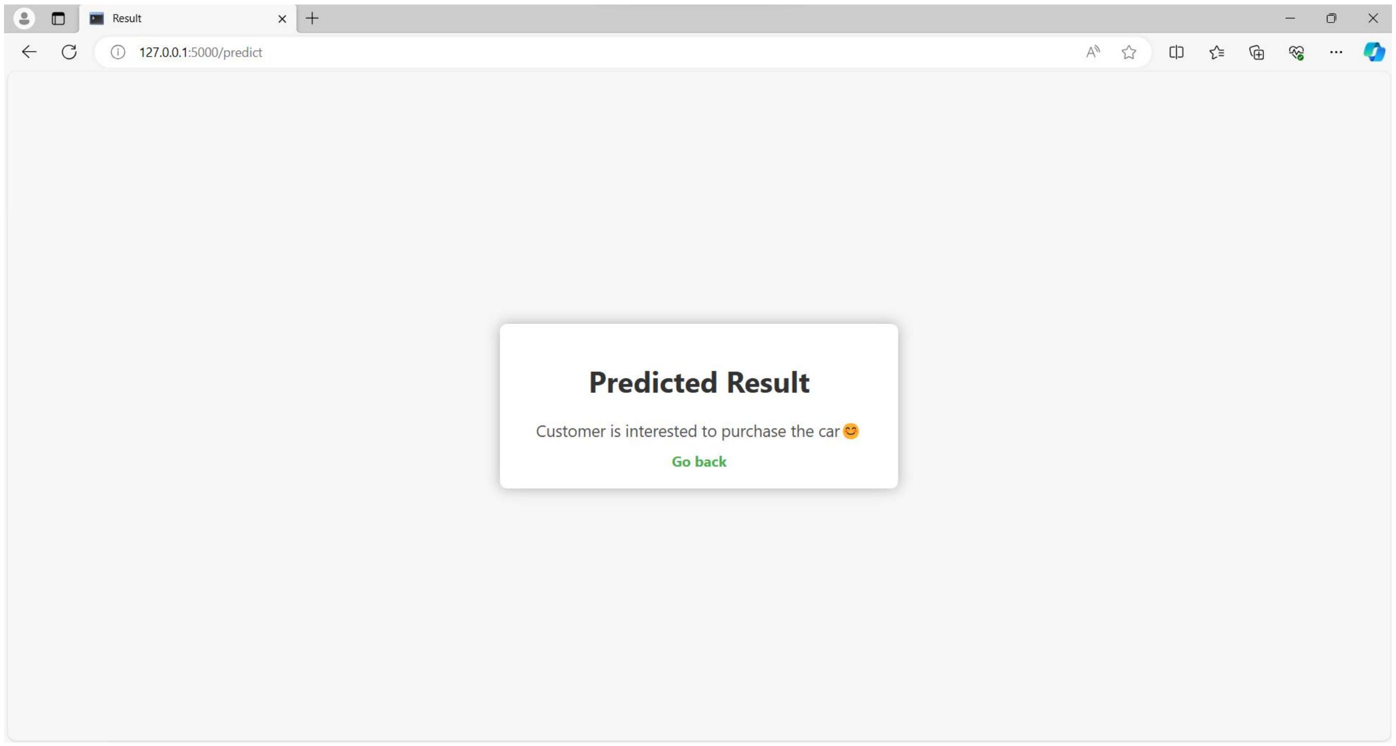
Open Visual studio code and Import all the project folders.



When you run the app.py file and click on the server url in terminal, you will redirected to home page. The home page will looks like:



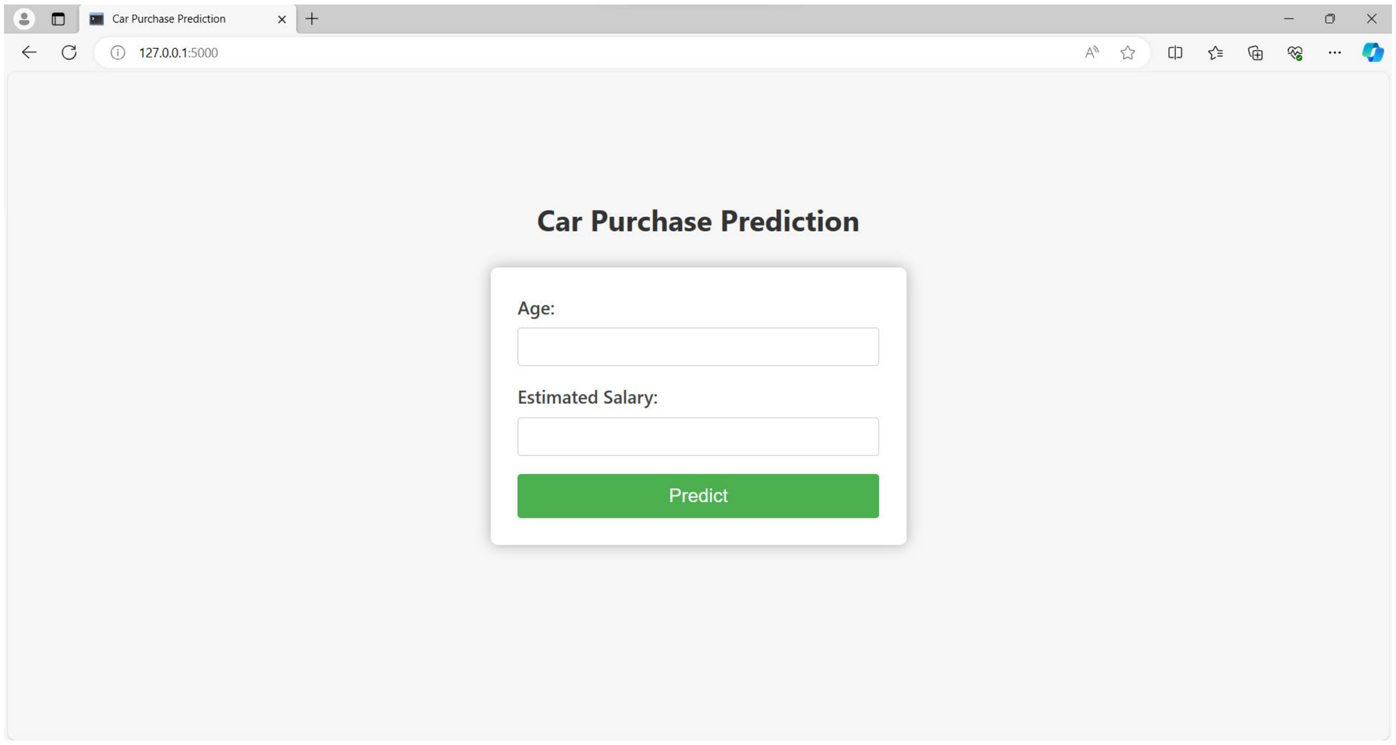
Enter age and salary of the customer and click on the predict button. You can see new Result.html page where your predicted result will be there, You can also see GOBACK link which takes you to the home page. And you can again enter the customer information there.



Predicted Result

Customer is interested to purchase the car😊

Go back



Conclusion:

In conclusion, the developed customer car purchase prediction model proves its effectiveness in aiding decision-making within the automotive industry. Leveraging customer age and salary as predictive features, the model offers valuable insights into purchase behaviors. Its accurate predictions empower businesses to tailor marketing strategies for targeted customer segments, optimizing resource allocation. By enabling personalized interactions, the model enhances user experience and engagement. Its integration into a user-friendly web interface simplifies access, making it an invaluable tool for both customers and dealerships. The model not only streamlines sales efforts but also drives customer satisfaction through informed choices. As a pivotal advancement in the automotive landscape, this model marks a significant step toward data-driven success.