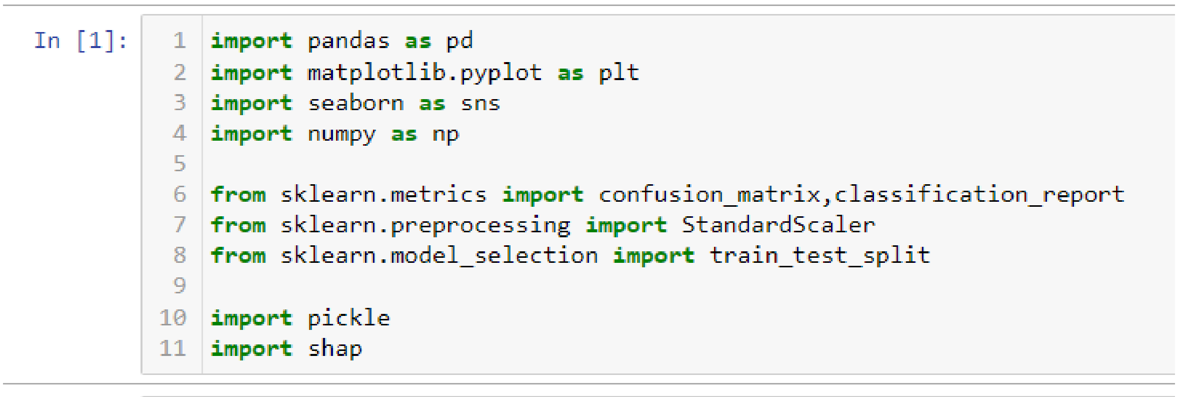
Project Design Phase-I
Solution Architecture

Date	9th October 2023
Team ID	PNT2023TMID592830
Project Name	Project – Travel Insurance Prediction
Maximum Marks	4 Marks

Solution Architecture:

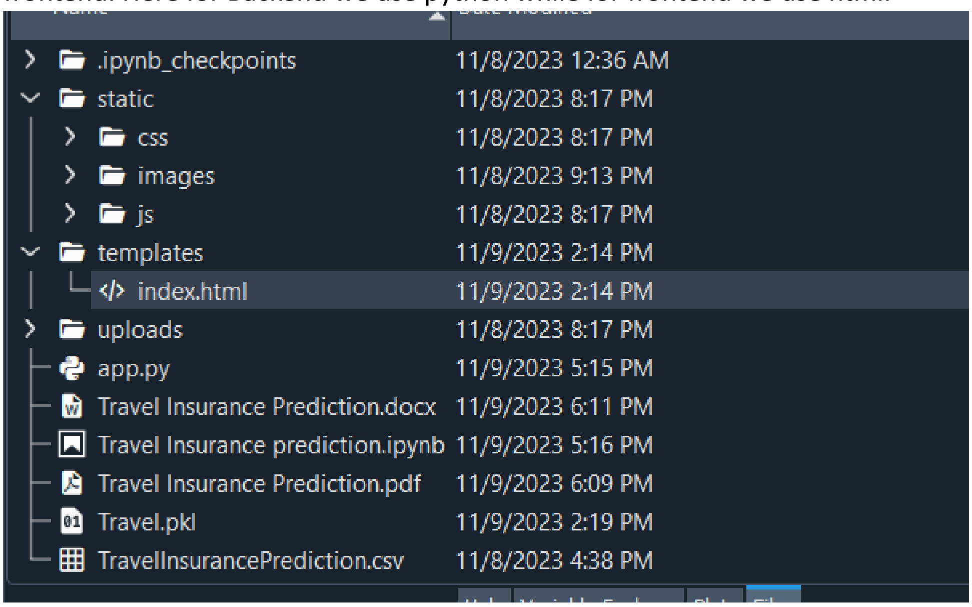
Tech Solution:-
We use the jupyter to create the prediction part of this topic and we import the following



In [1]:

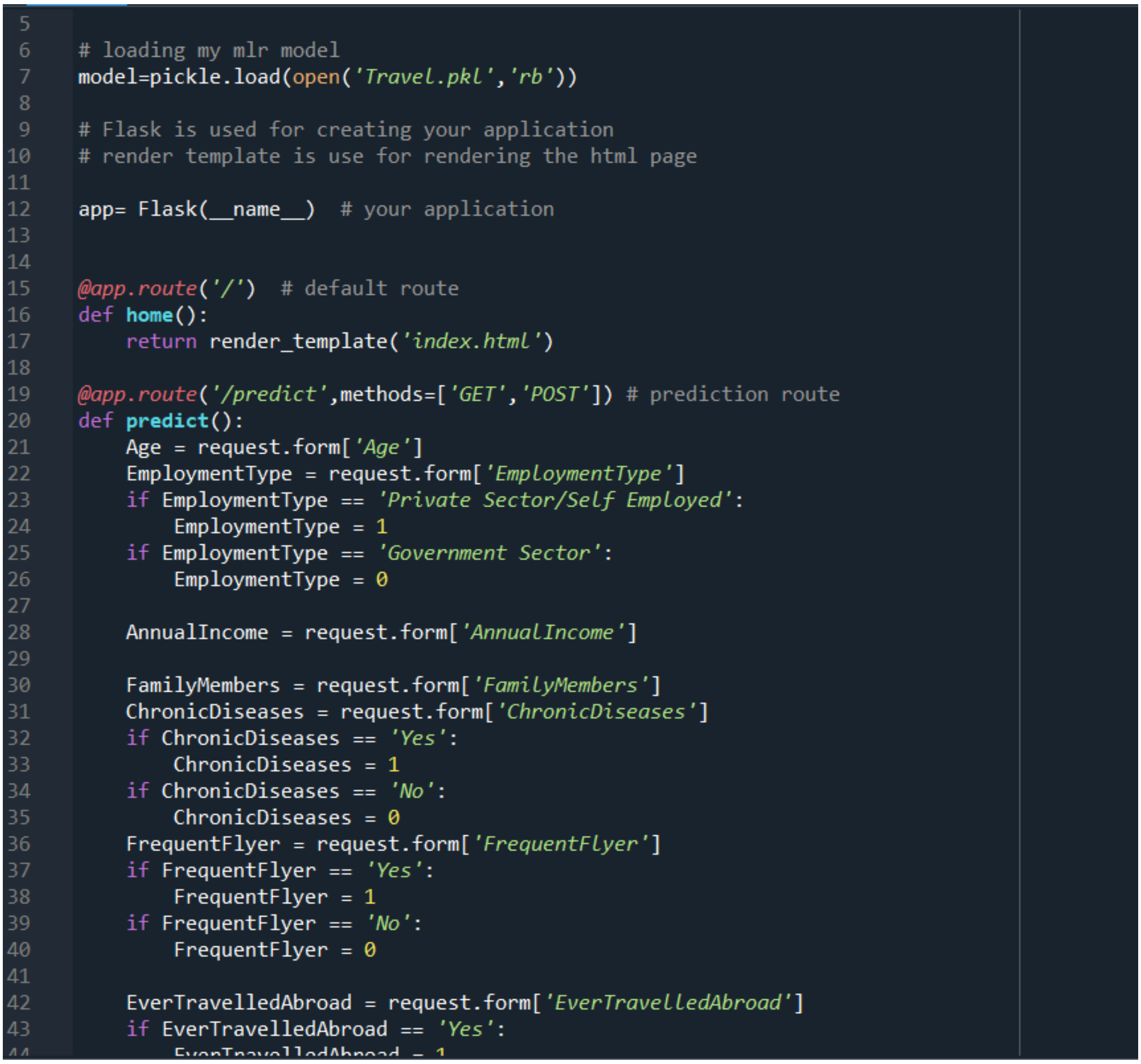
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import pickle
import shap
```

we use the above to get the required output.
After this we use spyder to create the webapp where we have 2 phases the backend and the frontend. Here for Backend we use python while for frontend we use html.



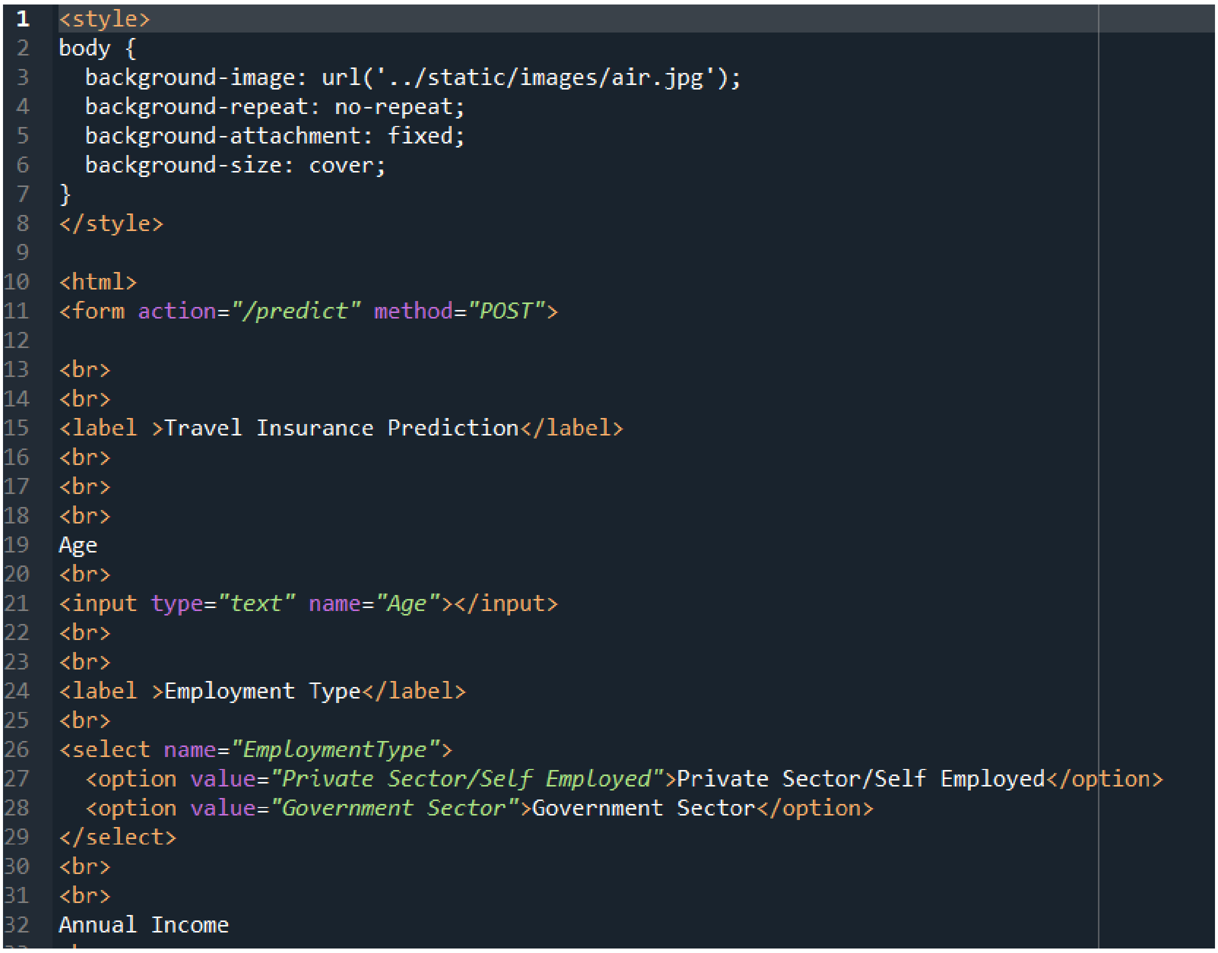
.ipynb_checkpoints	11/8/2023 12:36 AM
static	11/8/2023 8:17 PM
css	11/8/2023 8:17 PM
images	11/8/2023 9:13 PM
js	11/8/2023 8:17 PM
templates	11/9/2023 2:14 PM
index.html	11/9/2023 2:14 PM
uploads	11/8/2023 8:17 PM
app.py	11/9/2023 5:15 PM
Travel Insurance Prediction.docx	11/9/2023 6:11 PM
Travel Insurance prediction.ipynb	11/9/2023 5:16 PM
Travel Insurance Prediction.pdf	11/9/2023 6:09 PM
Travel.pkl	11/9/2023 2:19 PM
TravelInsurancePrediction.csv	11/8/2023 4:38 PM

Backend



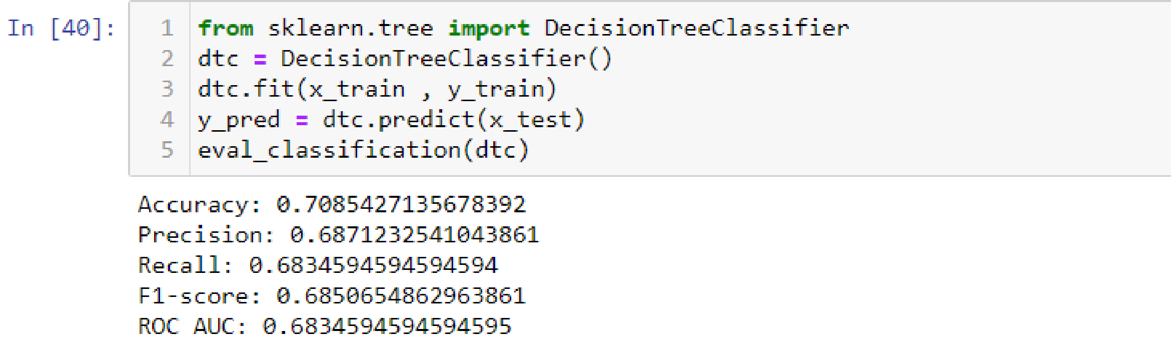
```python

# loading my mlr model
model=pickle.load(open('Travel.pkl','rb'))

# Flask is used for creating your application
# render template is use for rendering the html page

app= Flask(__name__)  # your application


@app.route('/')  # default route
def home():
    return render_template('index.html')

@app.route('/predict',methods=['GET','POST']) # prediction route
def predict():
    Age = request.form['Age']
    EmploymentType = request.form['EmploymentType']
    if EmploymentType == 'Private Sector/Self Employed':
        EmploymentType = 1
    if EmploymentType == 'Government Sector':
        EmploymentType = 0

    AnnualIncome = request.form['AnnualIncome']

    FamilyMembers = request.form['FamilyMembers']
    ChronicDiseases = request.form['ChronicDiseases']
    if ChronicDiseases == 'Yes':
        ChronicDiseases = 1
    if ChronicDiseases == 'No':
        ChronicDiseases = 0
    FrequentFlyer = request.form['FrequentFlyer']
    if FrequentFlyer == 'Yes':
        FrequentFlyer = 1
    if FrequentFlyer == 'No':
        FrequentFlyer = 0

    EverTravelledAbroad = request.form['EverTravelledAbroad']
    if EverTravelledAbroad == 'Yes':
        EverTravelledAbroad = 1
```

Frontend



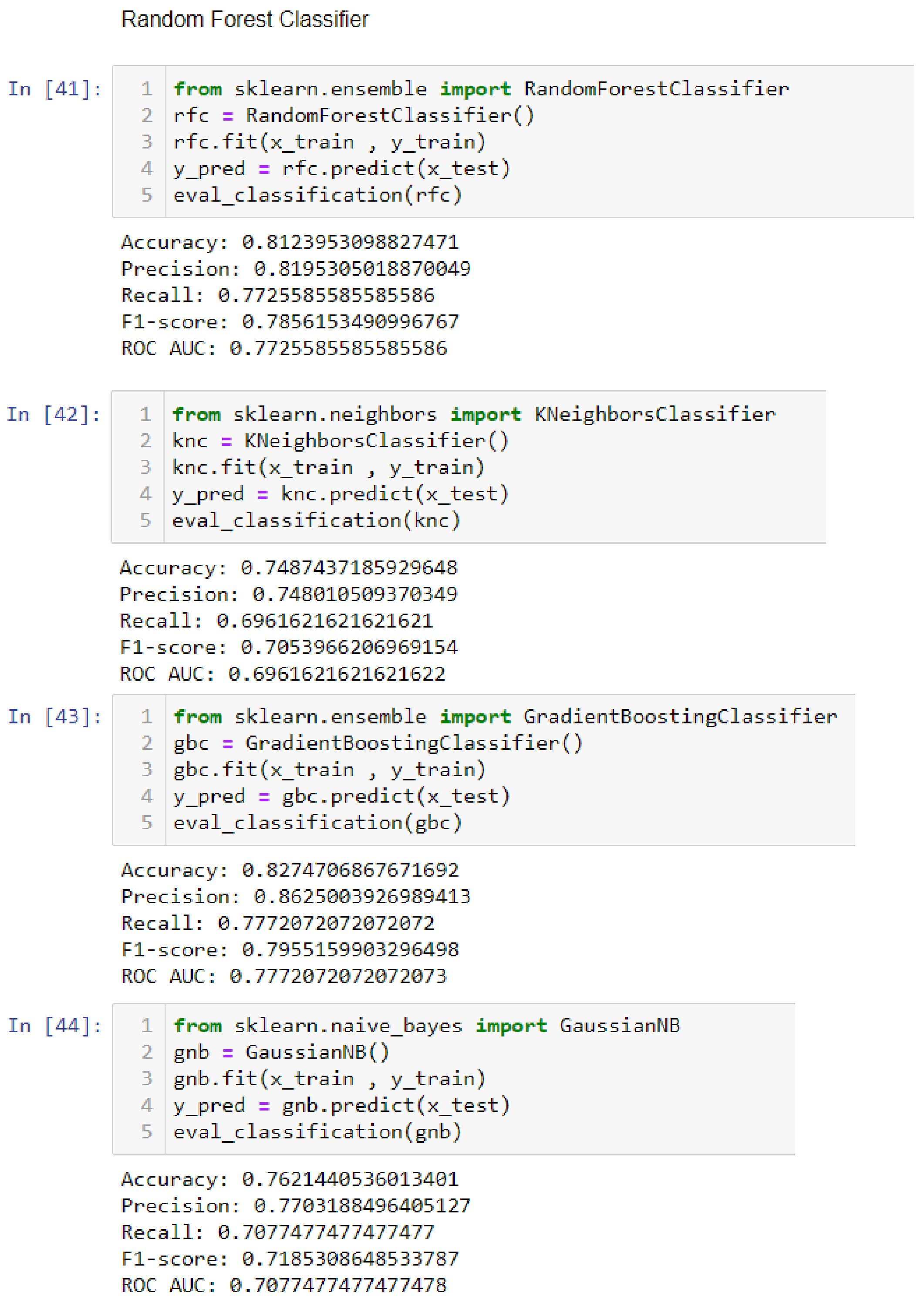
```
<style>
body {
  background-image: url('../static/images/air.jpg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
}
</style>

<html>
<form action="/predict" method="POST">

<br>
<br>
<label >Travel Insurance Prediction</label>
<br>
<br>
<br>
Age
<br>
<input type="text" name="Age"></input>
<br>
<br>
<label >Employment Type</label>
<br>
<select name="EmploymentType">
  <option value="Private Sector/Self Employed">Private Sector/Self Employed</option>
  <option value="Government Sector">Government Sector</option>
</select>
<br>
<br>
Annual Income
```

The accuracy of the prediction is pretty good since we have used around 5 different methods they are:-



```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train , y_train)
y_pred = dtc.predict(x_test)
eval_classification(dtc)
```

```
Accuracy: 0.7085427135678392
Precision: 0.6871232541043861
Recall: 0.6834594594594594
F1-score: 0.6850654862963861
ROC AUC: 0.6834594594594595
```



Random Forest Classifier

In [41]:

```python
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train , y_train)
y_pred = rfc.predict(x_test)
eval_classification(rfc)
```

```
Accuracy: 0.8123953098827471
Precision: 0.8195305018870049
Recall: 0.7725585585585586
F1-score: 0.7856153490996767
ROC AUC: 0.7725585585585586
```

In [42]:

```python
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier()
knc.fit(x_train , y_train)
y_pred = knc.predict(x_test)
eval_classification(knc)
```

```
Accuracy: 0.7487437185929648
Precision: 0.748010509370349
Recall: 0.6961621621621621
F1-score: 0.7053966206969154
ROC AUC: 0.6961621621621622
```

In [43]:

```python
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(x_train , y_train)
y_pred = gbc.predict(x_test)
eval_classification(gbc)
```

```
Accuracy: 0.8274706867671692
Precision: 0.8625003926989413
Recall: 0.7772072072072072
F1-score: 0.7955159903296498
ROC AUC: 0.7772072072072073
```

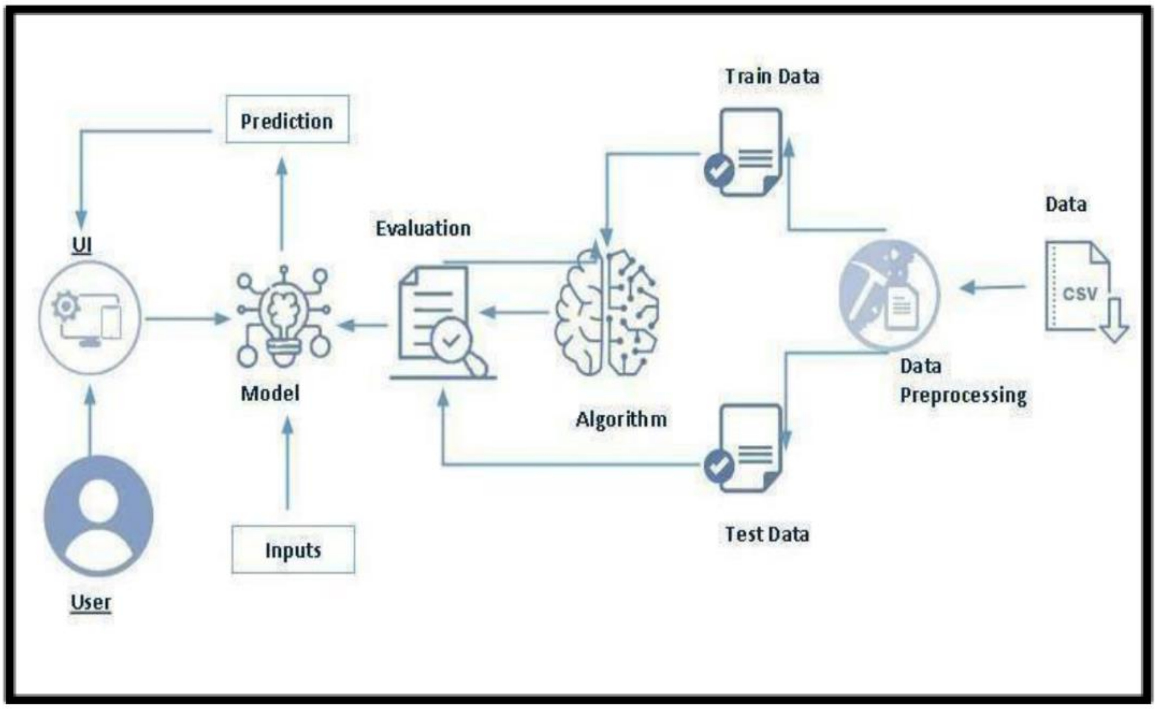
In [44]:

```python
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train , y_train)
y_pred = gnb.predict(x_test)
eval_classification(gnb)
```

```
Accuracy: 0.7621440536013401
Precision: 0.7703188496405127
Recall: 0.7077477477477477
F1-score: 0.7185308648533787
ROC AUC: 0.7077477477477478
```

By using the above methods we have managed to get a good prediction of whether this would be useful to the customer or not.

Solution Architecture Diagram:


Train Data
Prediction
UI
Evaluation
Data
CSV
Model
Algorithm
Data
Preprocessing
Test Data
Inputs
User