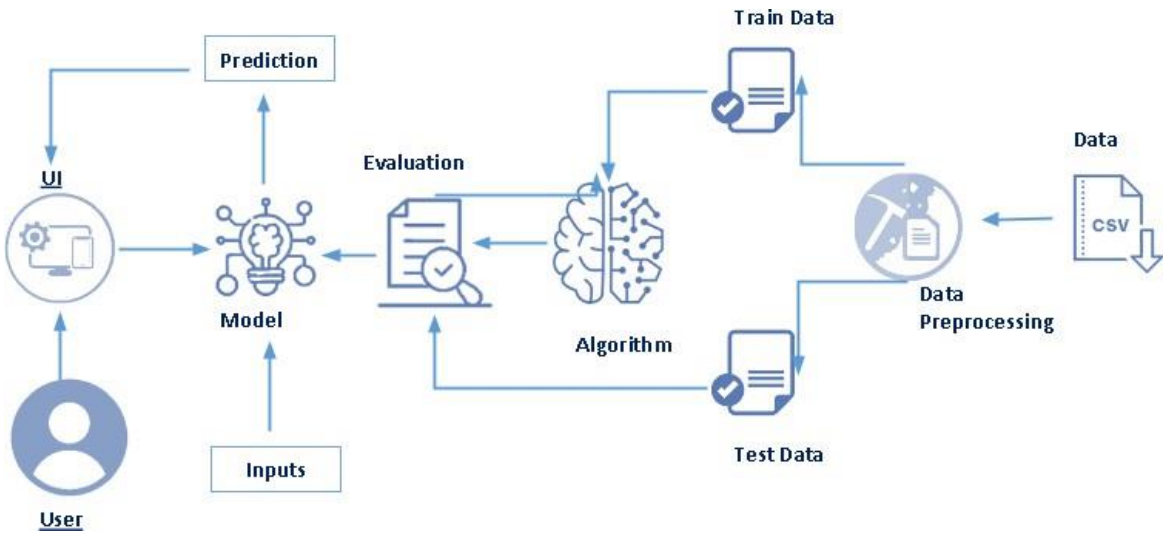
Online Shoppers Intentions Using IBM Watson

Project Overview:

Online shopping is the activity or action of buying products or services over the Internet. It means going online, landing on a seller's website, selecting something, and arranging for its delivery. The buyer either pays for the good or service online with a credit or debit card or upon delivery. The term does not only include buying things online but also searching for them online. In other words, I may have been engaged in online shopping but did not buy anything.
We are going to predict whether the customer will buy the product or just go window shopping. Here, We will be using classification algorithms such as Logistic Regression, Random forest, & Clustering algorithm K-Means. We will train and test the data with these algorithms. From this, the best model is selected and saved in pkl format.

Architecture:



Prerequisites

To complete this project, you must required following software's, concepts and packages

- Anaconda navigator and pycharm:
    - Refer the link below to download anaconda navigator
- Python packages:
    - Open anaconda prompt as administrator
    - Type "pip install numpy" and click enter.
    - Type "pip install pandas" and click enter.
    - Type "pip install scikit-learn" and click enter.
    - Type "pip install matplotlib" and click enter.
    - Type "pip install pickle-mixin" and click enter.
    - Type "pip install seaborn" and click enter.
    - Type "pip install Flask" and click enter.

Prior Knowledge

You must have prior knowledge of following topics to complete this project.

- ML Concepts : checkout the links
    - Supervised learning
    - Unsupervised learning
    - K-means
    - Random forest
    - Logistic regression
    - Evaluation metrics

Project Objectives

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.
- Understand the difference between Unsupervised and supervised ML.

Project Workflow

- 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,

- **Data collection**
    - Collect the dataset or create the dataset

- **Visualizing and analyzing data**

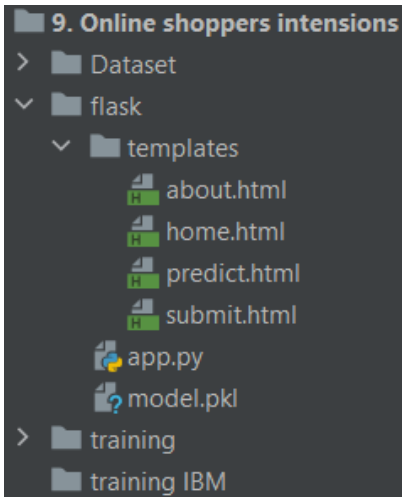
    - Import the required libraries
    - Univariate analysis
    - Bivariate analysis
    - Multivariate analysis
    - Descriptive analysis
- **Data pre-processing**

    - Checking for null values
    - Handling categorical data
    - Dropping unwanted features
    - Scaling features
- **Model building**

    - **Unsupervised ML**
        - Elbow method
        - Initializing the model
        - Dimensionality reduction
    - **Supervised ML**
        - Splitting dataset
        - Initializing the model
        - Evaluating performance of model
        - Save the model
- **Application Building**
    - Create an HTML file
    - Build python code

Project Structure

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and training_ibm folder contains IBM deployment files.

Dataset Collection

Data Collection

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.

Dataset

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

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

Link

Visualizing And Analysing The Data

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.

Importing The Libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as fivethirtyeight.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
import pickle
```

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. And head() function is used to visualize the first 5 rows of the dataset.

```
df = pd.read_csv(r'D:\TheSmartBridge\Projects\9. Online shoppers intensions\Dataset\online_shoppers_intention.csv')
df.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0

- To find the shape of our data, df.shape method is used. To find the data type, memory usage and non-null values df.info() function is used.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

```
df.shape
```

```
(12330, 18)
```

Univariate Analysis

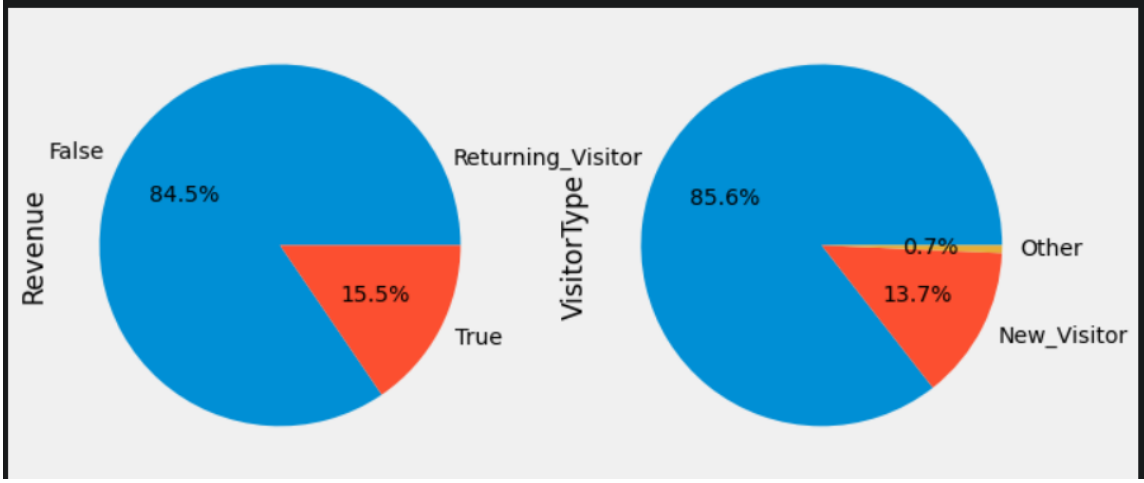
In simple words, univariate analysis is understanding the data with single feature. Here we have used pieplot.

Note: Different approaches can be used by the developer to analyze the data.

- The pie plot is plotted with categorical features value counts. With a pie plot, the importance of categories in categorical features is analyzed in form of a percentage. To perform this plot, plot() function is used. From the below image we can visualize most of the customers are return visitors. The probability of buying product chance is high if any discounts are allotted to products.

```
plt.figure(figsize=(10,10))
plt.subplot(121)
df['Revenue'].value_counts().plot(kind='pie',autopct='%.1f%%')
plt.subplot(122)
df['VisitorType'].value_counts().plot(kind='pie',autopct='%.1f%%')

"""From the dataset, 84% of customers have not done shopping. And 86% of customers are returning visitors."""
```

```
'From the dataset, 84% of customers have not done shopping. And 86% of customers are returning visitors.'
```



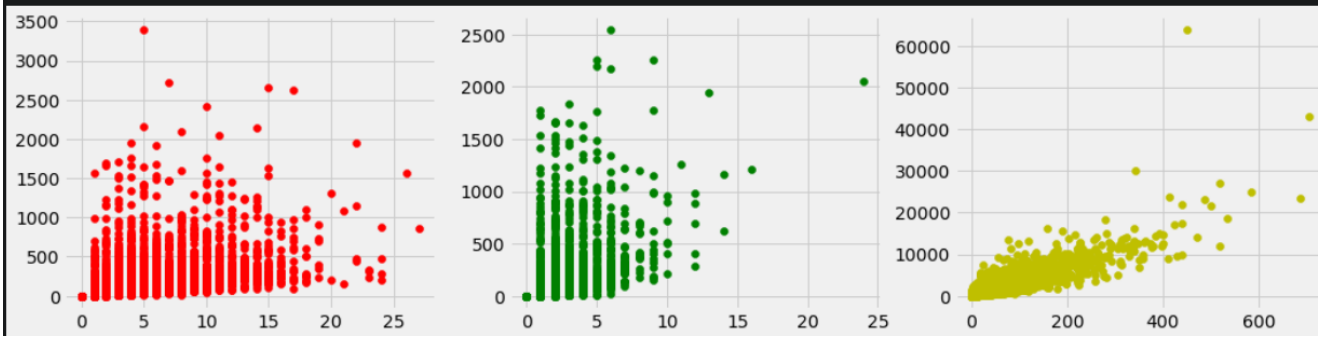
Bivariate Analysis

To find the relation between two features we use bivariate analysis.

- "Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product Related Duration" represents the number of different types of pages visited by the visitor and total time spent in each of these page categories. To find the relationship between two features scatterplot() function is used. (Refer below image) Product related page and product related duration has high positive linearity.
-

```
"""Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product
Related Duration" represent the number of different types of pages visited by the visitor and total time spent in each of
these page categories. """

plt.figure(figsize=(16,4))
plt.subplot(131)
plt.scatter(df['Administrative'],df['Administrative_Duration'],color='r')
plt.subplot(132)
plt.scatter(df['Informational'],df['Informational_Duration'],color='g')
plt.subplot(133)
plt.scatter(df['ProductRelated'],df['ProductRelated_Duration'],color='y')

"""Most of the visitors spending time on product related page"""
```

```
'Most of the visitors spending time on product related page'
```



To compare the sale on special days and normal days crosstab() method is used. From the below image we found not much difference in the sale of normal and special days.

```
"""Not much difference in sale rate on normal and special days"""

pd.crosstab(df['Revenue'],df['SpecialDay'])
```

SpecialDay	0.0	0.2	0.4	0.6	0.8	1.0
Revenue						
False	9248	164	230	322	314	144
True	1831	14	13	29	11	10

Multivariate Analysis

In simple words, multivariate analysis is to find the relation between multiple features.

- To find the no. of sales based on month and visitor type crosstab() method is used. (refer below image) November month has the highest sales.

```
"""Based on month and visitor type features, count of revenue is analysed."""

pd.crosstab([df['Month'],df['VisitorType']],df['Revenue'])
```

June	Other	1	0
	Returning_Visitor	235	22
Mar	New_Visitor	196	36
	Returning_Visitor	1519	156
May	New_Visitor	231	88
	Returning_Visitor	2768	277
Nov	New_Visitor	291	128
	Other	19	3
	Returning_Visitor	1928	629
Oct	New_Visitor	96	28
	Returning_Visitor	338	87
Sep	New_Visitor	80	28
	Returning_Visitor	282	58

Descriptive 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.

```
df.describe(include='all')
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	P
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	123
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
mean	2.315166	80.818611	0.503569	34.472398	31.731468	1194.746220	0.022191	0.043073	
std	3.321784	176.779107	1.270156	140.749294	44.475503	1913.669288	0.048488	0.048597	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	7.000000	184.137500	0.000000	0.014286	
50%	1.000000	7.500000	0.000000	0.000000	18.000000	598.936905	0.003112	0.025156	
75%	4.000000	93.256250	0.000000	0.000000	38.000000	1464.157214	0.016813	0.050000	
max	27.000000	3398.750000	24.000000	2549.375000	705.000000	63973.522230	0.200000	0.200000	3

Data Pre-Processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

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.

Checking For Null Values

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling of missing values step

```
df.isnull().sum()
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using label encoder.

```
# handling categorical feature

le = LabelEncoder()
df['Month'] = le.fit_transform(df['Month'])
df['VisitorType'] = le.fit_transform(df['VisitorType'])
df['Weekend'] = le.fit_transform(df['Weekend'])
df['Revenue'] = le.fit_transform(df['Revenue'])
```

Dropping Unwanted Features

Now let's create our 1st model with unsupervised ML algorithm- KMeans clustering. For implementing this algorithm target feature should be removed. With drop() method revenue column is dropped.

Scaling The Features

Normalization technique Minmax scaler() is used on dfKmeans and the array values are converted into dataframe.

- Minmax scaler is initialized and fit_transform() method is used to scale the values. Now, those array values are created as a dataframe as shown in below image.

```
# Scalling values

scaler = MinMaxScaler()
scaled_df = scaler.fit_transform(dfKmeans)

dfKmeans = pd.DataFrame(scaled_df,columns=dfKmeans.columns)
dfKmeans.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValue
0	0.0	0.0	0.0	0.0	0.001418	0.000000	1.00	1.00	0.
1	0.0	0.0	0.0	0.0	0.002837	0.001000	0.00	0.50	0.
2	0.0	0.0	0.0	0.0	0.001418	0.000000	1.00	1.00	0.
3	0.0	0.0	0.0	0.0	0.002837	0.000042	0.25	0.70	0.
4	0.0	0.0	0.0	0.0	0.014184	0.009809	0.10	0.25	0.

Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project unsupervised ML- KMeans and supervised ML- Logistic regression and Random forest classifier is used. The best model is selected.

Elbow Method

To find the optimal number of clusters elbow method is used. The plot will be plotted with sse (sum of squared error). Inertia is used from kmeans algorithm to calculate sse.

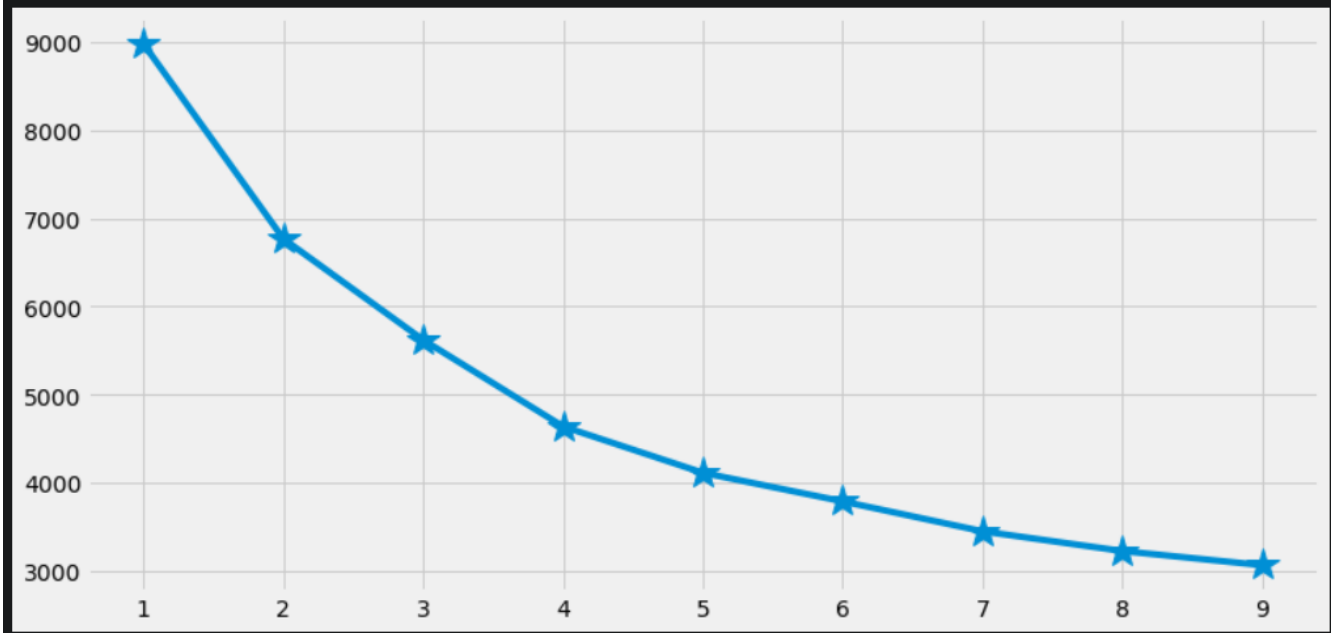
```
# Finding N clusters by elbow method.

n_cluster = range(1,10,1)
sse = []
for i in n_cluster:
    k = KMeans(n_clusters=i)
    ypred = k.fit(scaled_df)
    sse.append(k.inertia_)
```

```
sse
```

```
[8977.629530570664,
 6768.01393714652,
 5619.938835283883,
 4638.301369734072,
 4112.527344033329,
 3788.079018522391,
 3448.4291813087816,
 3224.6785631183748,
 3063.896789860229]
```

```
plt.figure(figsize=(12,6))
plt.plot(n_cluster,sse,marker='*',markersize=20)
```

```
[<matplotlib.lines.Line2D at 0x29a758d0d90>]
```



Initialize The Model

From the previous diagram, we found our optimal number of clusters is 4. The model is initialized in a new variable km. For fitting and predicting the values fit_predict() method is used.

```
km = KMeans(n_clusters=4)
ypred = km.fit_predict(dfKmeans)
```

Dimensionality Reduction

PCA- Principal Component Analysis is used to reduce the dataset dimension. Scatterplot is used to visualize the clusters.

- PCA() is initialized to pca variable. We need 2 columns. So, let's pass 2 as a parameter. To transform the dataset into 2 columns fit_transform() is used.

```
#dfKmeans['Cluster']=ypred

pca = PCA(n_components=2)
dfPCA = pca.fit_transform(dfKmeans)
dfPCA
```

```
array([[-1197.63555578,   -40.8241322 ],
       [-1133.68455771,   -43.2912114 ],
       [-1197.63603661,   -40.82414532],
       ...,
       [-1013.48454092,   -47.94070163],
       [ -849.24642902,    18.23938657],
       [-1176.36824437,   -41.65359505]])
```

- New dataframe is created. Previous step(2d array) are passed as the parameters. To pass the cluster values, a new column is created as shown in the below figure.

```
dfPCA = pd.DataFrame(dfPCA,columns=['PCA 1','PCA 2'])
dfPCA.head()
```

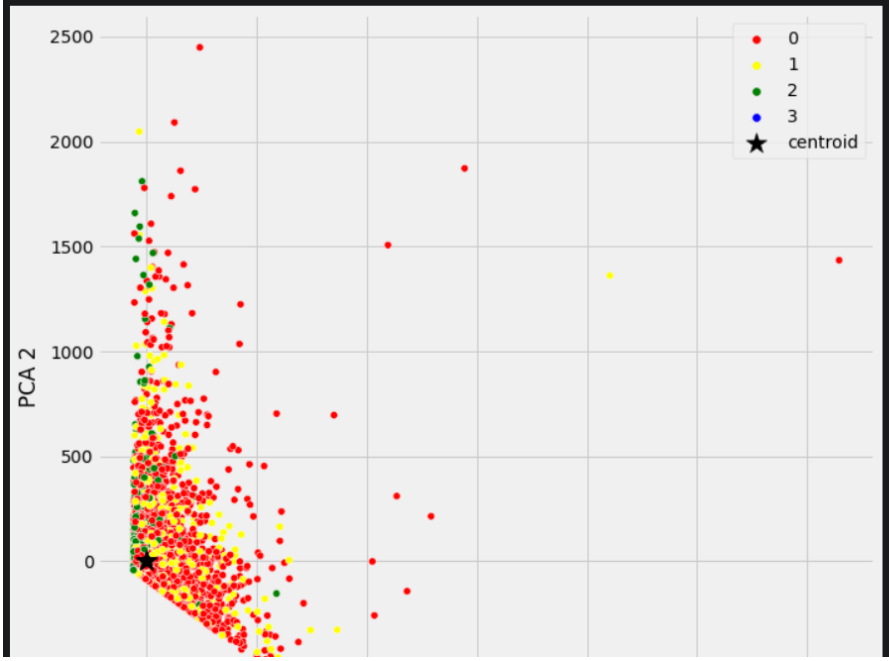
	PCA 1	PCA 2
0	-1197.635556	-40.824132
1	-1133.684558	-43.291211
2	-1197.636037	-40.824145
3	-1194.952046	-40.933714
4	-570.631539	-65.005398

```
dfPCA['Cluster'] = ypred
```

- Scatterplot() from seaborn is used to visualize the data points based on clusters. Refer the below image.

```
plt.figure(figsize=(10,10))
sns.scatterplot(dfPCA['PCA 1'], dfPCA['PCA 2'],hue = dfPCA['Cluster'],palette=['red','yellow','green','blue'])
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='black',s=300,marker='*',label='centroid')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x29a75938820>
```



Supervised ML

Here we build our supervised ml models.

Splitting The Dataset

First split the dataset into x and y. Independent features on x and dependent feature on y.

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
# Splitting dataset

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

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=10)
```

Logistic Regression Model

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

```
def logisticReg(x_train, x_test, y_train, y_test):
    lr = LogisticRegression()
    lr.fit(x_train,y_train)
    yPred = lr.predict(x_test)
    print('***LogisticRegression***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Random Forest Model

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

```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Now let's see the performance of all the models and save the best model

Compare The Model

For comparing the above two models compareModel function is defined.

```
def compareModel(x_train, x_test, y_train, y_test):
    logisticReg(x_train, x_test, y_train, y_test)
    print('-'*100)
    randomForest(x_train, x_test, y_train, y_test)
```

```
compareModel(x_train, x_test, y_train, y_test)
```

```
***LogisticRegression***
Confusion matrix
[[3033   82]
 [ 370  214]]
Classification report
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      3115
           1       0.72      0.37      0.49       584

    accuracy                           0.88      3699
   macro avg       0.81      0.67      0.71      3699
weighted avg       0.86      0.88      0.86      3699

----------------------------------------------------------------------------------------------------
***RandomForestClassifier***
Confusion matrix
[[3011  104]
 [ 257  327]]
Classification report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      3115
           1       0.76      0.56      0.64       584

    accuracy                           0.90      3699
   macro avg       0.84      0.76      0.79      3699
weighted avg       0.90      0.90      0.90      3699
```

Evaluating Performance Of The Model And Saving The Model

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Note: To understand cross validation, refer this link.

```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
yPred = rf.predict(x_test)
```

```
cv = cross_val_score(rf,x,y,cv=5)
np.mean(cv)
```

```
0.8944849959448499
```

```
pickle.dump(rf,open('model.pkl','wb'))
```

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

Building Html Pages

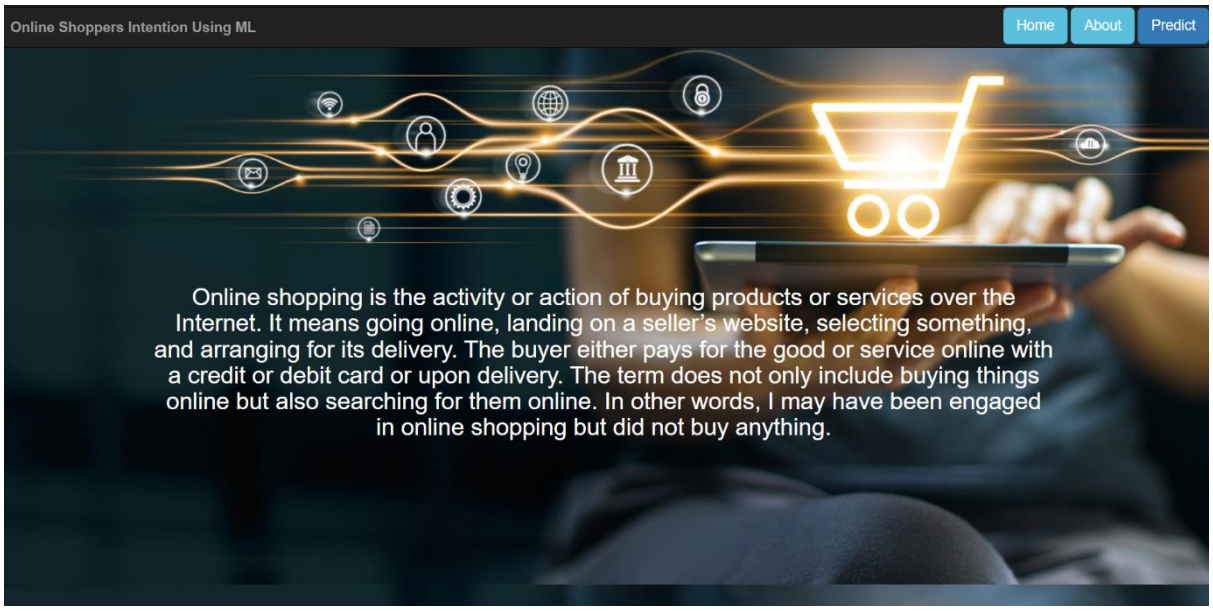
For this project create three HTML files namely

- home.html
- about.html
- predict.html
- submit.html

and save them in templates folder.

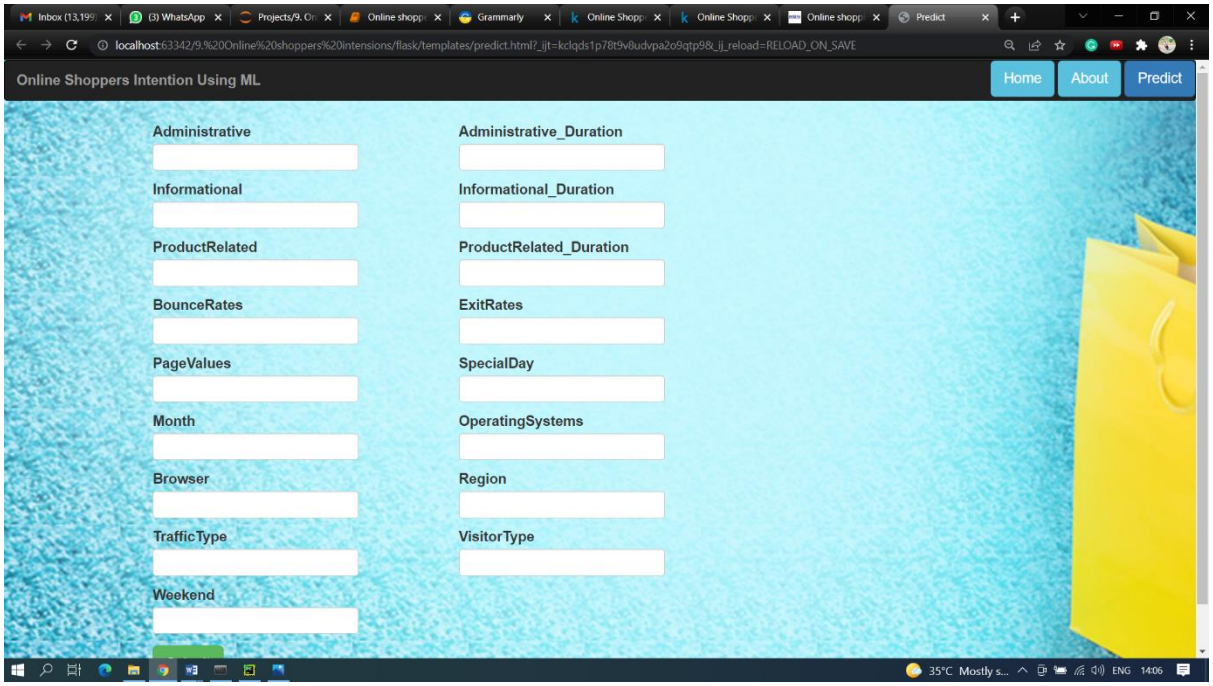
Let's see how our home.html page looks like:

Now when you click on about button from top right corner you will get redirected to about.html



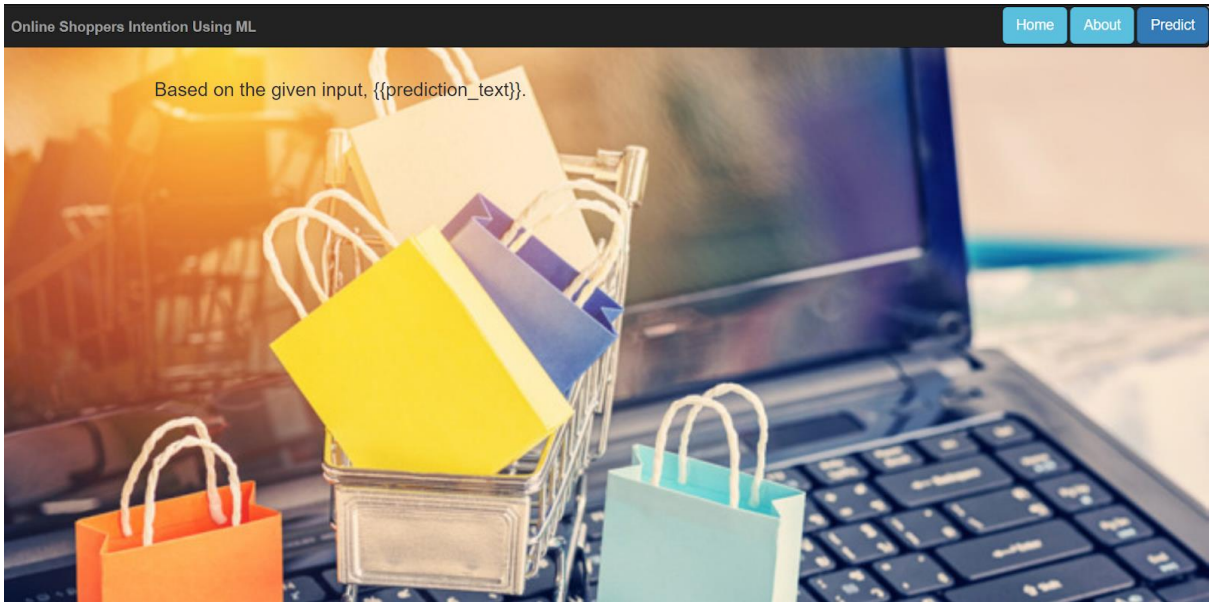
Now when you click on predict button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:



Now when you click on submit button from left bottom corner you will get redirected to submit.html

Lets look how our submit.html file looks like:



Build Python Code

Import the libraries

```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
model = pickle.load(open('model.pkl', 'rb'))
app = Flask(__name__)
```

Render HTML page:

```
@app.route("/home")
def home():
    return render_template('home.html')
```

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:

```python
@app.route("/pred", methods=['POST'])
def predict():
    Administrative = request.form['Administrative']
    Administrative_Duration = request.form['Administrative_Duration']
    Informational = request.form['Informational']
    Informational_Duration = request.form['Informational_Duration']
    ProductRelated = request.form['ProductRelated']
    ProductRelated_Duration = request.form['ProductRelated_Duration']
    BounceRates = request.form['BounceRates']
    ExitRates = request.form['ExitRates']
    PageValues = request.form['PageValues']
    SpecialDay = request.form['SpecialDay']
    Month = request.form['Month']
    OperatingSystems = request.form['OperatingSystems']
    Browser = request.form['Browser']
    Region = request.form['Region']
    TrafficType = request.form['TrafficType']
    VisitorType = request.form['VisitorType']
    Weekend = request.form['Weekend']
    total = [[int(Administrative), float(Administrative_Duration), int(Informational), float(Informational_Duration)
              int(ProductRelated), float(ProductRelated_Duration), float(BounceRates), float(ExitRates),
              float(PageValues), float(SpecialDay), int(Month), int(OperatingSystems), int(Browser), int(Region),
              int(TrafficType), int(VisitorType), int(Weekend)]]
    print(total)
    prediction = model.predict(total)
    print(prediction)
    if prediction == 0:
        text = 'The visitor is not interested in buying products.'
    else:
        text = 'The visitor is interested in buying products'
    return render_template('submit.html', prediction_text=text)
```

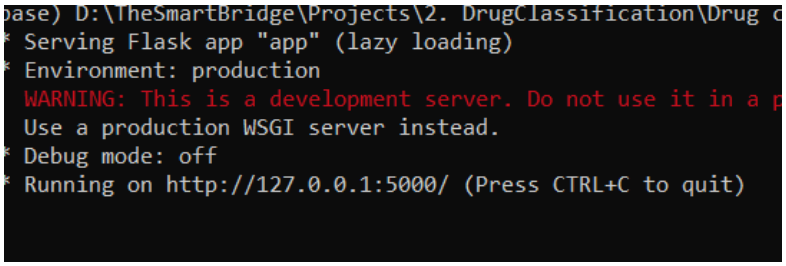
Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text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

Main Function:

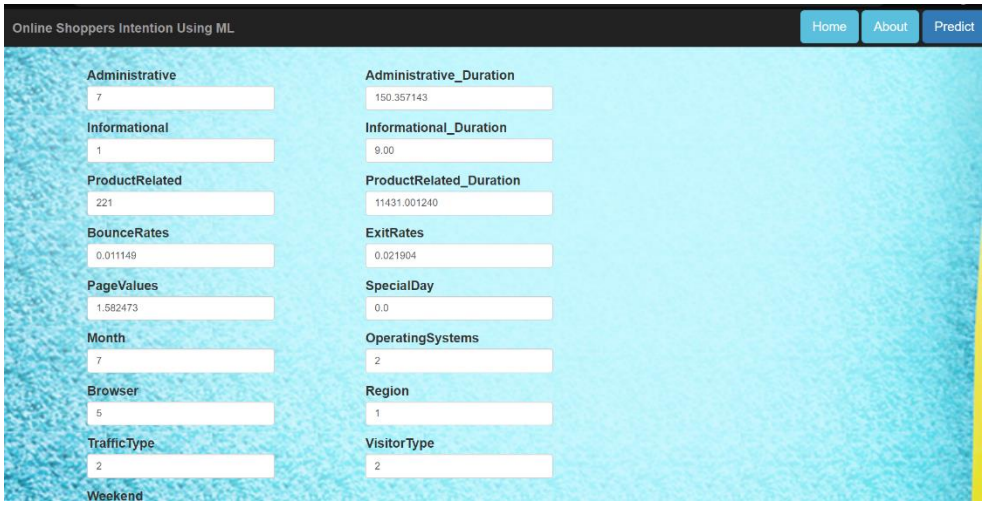
```python
if __name__ == "__main__":
    app.run(debug=False)
```

Run The Application

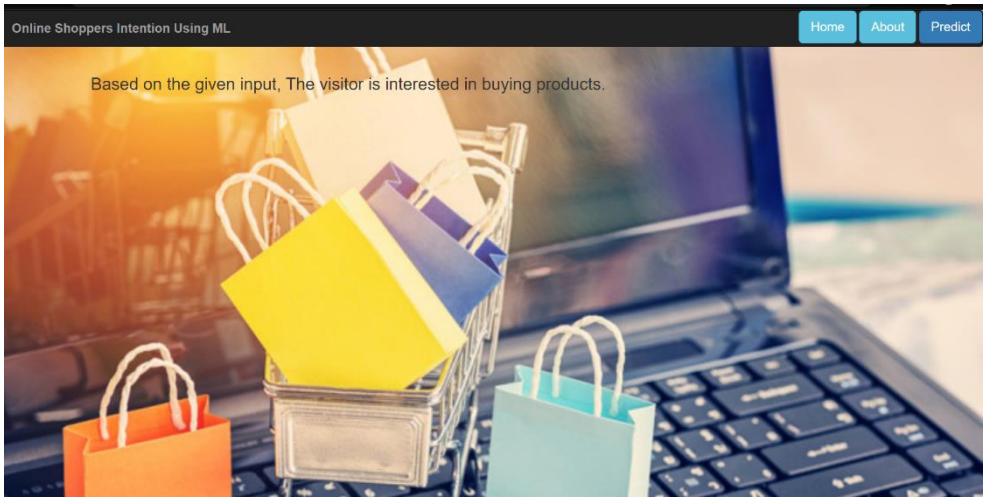
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



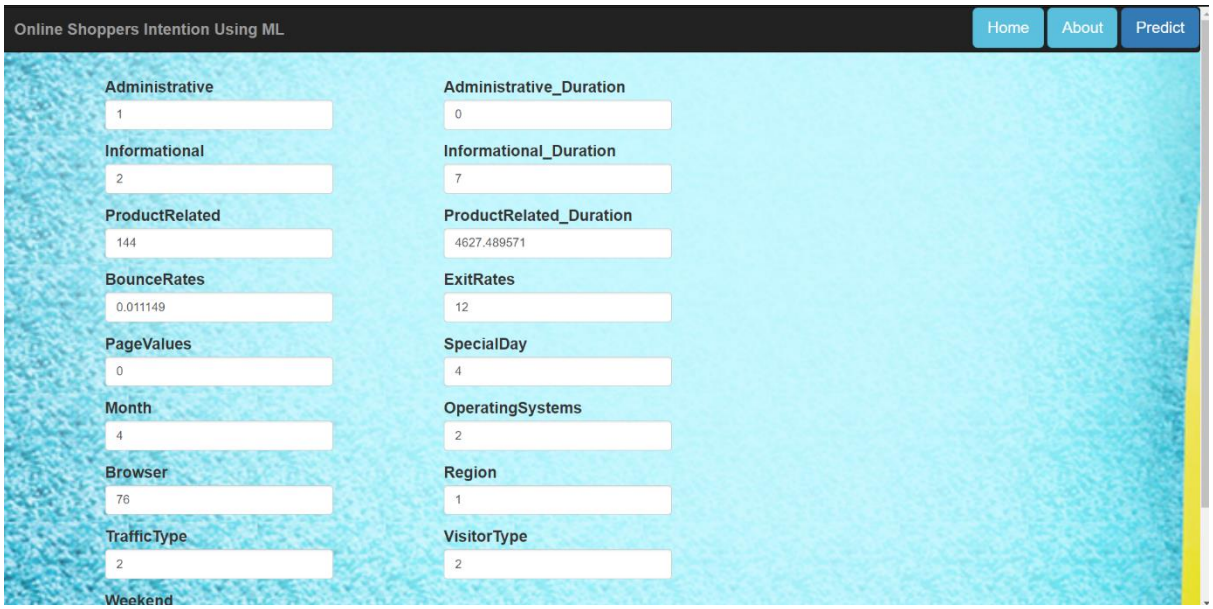
Input 1:



Output 1:



Input 2:



Output 2: