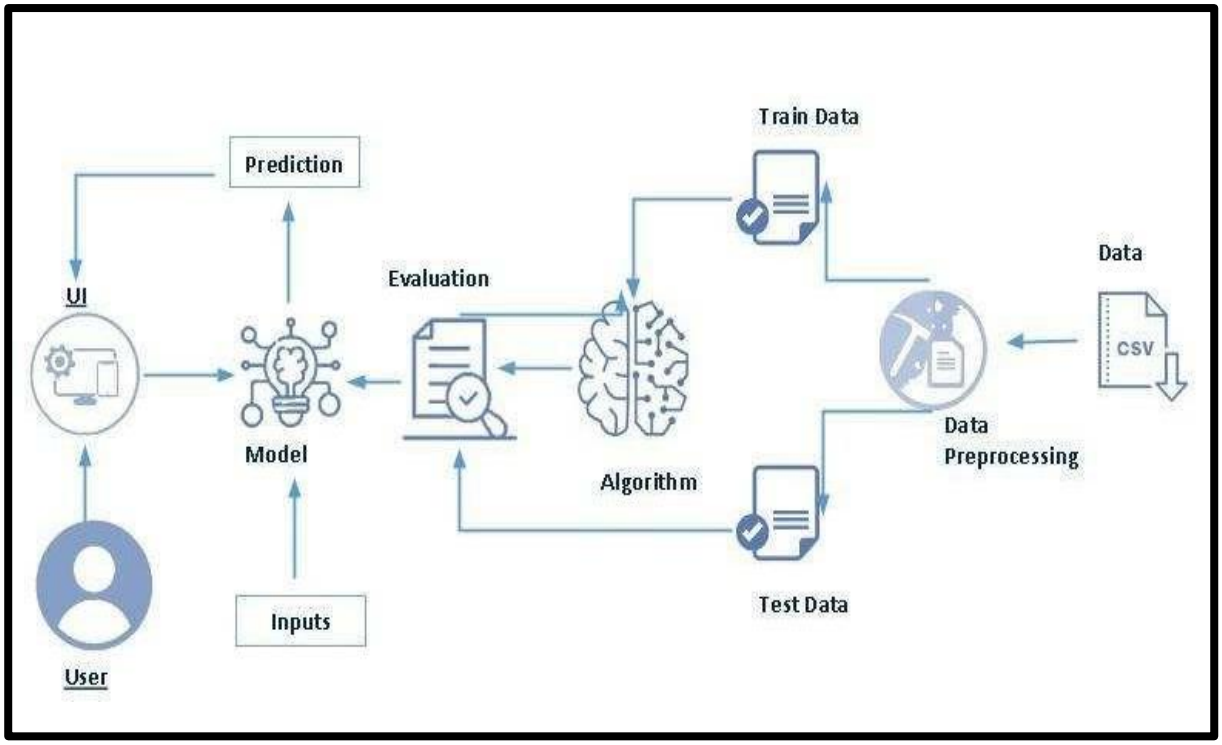
Wholesale Customer Segmentation Analysis Using ML

Project Description:

- This project aims to analyse the spending behaviour of wholesale customers and identify opportunities for growth. The data set consists of annual spending (in monetary units) on various product categories, including fresh, milk, grocery, frozen, detergents and paper, and delicatessen products. Additionally, the data includes information on the channel (hotel/restaurant/cafe or retail) and region (Lisbon, Oporto, or other) of the customer.
- By identifying customer segments with distinct spending behaviours, the project aims to provide insights on how wholesale businesses can tailor their marketing strategies and product offerings to better serve each customer segment.
- The outcomes of this project can have several applications. Business can utilize the predictive model to identify customers with distinct spending behaviours.

Technical Architecture:



Pre requisites:

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

- **Anaconda navigator and pycharm:**

- **Python packages:**
  - Numpy: It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations.
  - Scikit-learn: It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, etc. and also supports NumPy.
  - Pandas: This is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
  - Matplotlib: Matplotlib is a 2D plotting library for Python that produces high-quality charts and figures. It provides a wide variety of plots, charts, and visualizations for data analysis and presentation.
  - Scipy: SciPy is an open-source library used for scientific and technical computing. It builds on NumPy and provides additional functionality for optimization, integration, interpolation, eigenvalue problems, signal and image processing, and more.
  - Pickle-mixin: The pickle-mixin is not a standard library but is likely referring to a module or class that provides a mixin for pickling in Python. Pickling is the process of converting a Python object into a byte stream, and the pickle module is commonly used for this purpose.
  - Flask: Flask is a lightweight web framework for Python. It simplifies the process of building web applications by providing tools and libraries for handling routing, templates, and interactions with databases.

If you are using anaconda navigator, do below steps to download required packages:

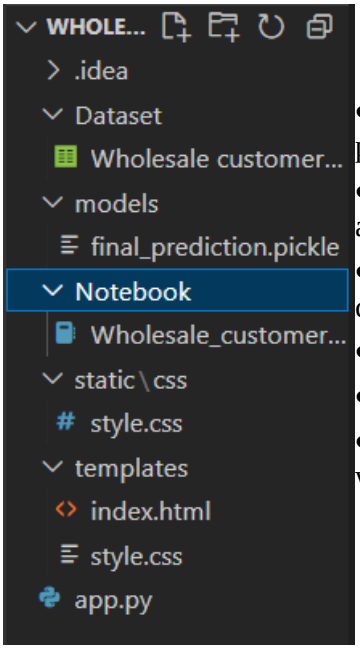
- Open anaconda prompt as administrator
- Type "pip install numpy" and click enter.
- Type "pip install pandas" and click enter.
- Type "pip install scikit-learn" and click enter.
- Type "pip install matplotlib" and click enter.
- Type "pip install scipy" and click enter.
- Type "pip install pickle-mixin" and click enter.
- Type "pip install seaborn" and click enter.
- Type "pip install Flask" and click enter.

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.

Project Structure:



- The dataset folder contains the images of different conditions of potatoes in their respective folders.
- The flask folder has all necessary files to build the flask application.
- The static folder contains the images and the style sheets needed to customize the web page.
- The templates folder has the HTML pages.
- App.py is the python script for server-side computing.
- Wholesale_customer_segmentation.ipynb is the notebook on which the code is executed.

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- final_prediction.pkl is our saved model. Further we will use this model for flask integration.

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,

- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical data
  - Splitting data into train and test
- Model building
  - Import the model building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating performance of model
  - Save the model

- Application Building
    - Create an HTML file
    - Build python code

Milestone 1: 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.

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 The Wholesale Customer Segmentation data. This data is downloaded from kaggle.com. Please refer the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/binovi/wholesale-customeís-data-set

Milestone 2: 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.

Activity 1: Importing the libraries

```
#Data handling Imports
import pandas as pd
import numpy as np
#Notebook arrange Imports
import warnings
warnings.filterwarnings('ignore')
#Visualisation Imports
import seaborn as sns
import pylab
pylab.style.use('seaborn-pastel')
import matplotlib.pyplot as plt
%matplotlib inline
#Normalization & Scaler Imports
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
#Sampling Imports
from sklearn.model_selection import KFold
#Modeling Imports
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
#Accuracy Validation Imports
from sklearn import metrics
from sklearn.metrics import auc, roc_curve, f1_score, accuracy_score,precision_recall_curve,\
confusion_matrix, classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import cross_val_score
```

Activity 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.

```
#Read the dataset
df = pd.read_csv("/content/Wholesale customers data.csv")
df
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185
...	...	...	...	...	...	...	...	...
435	1	3	29703	12051	16027	13135	182	2204
436	1	3	39228	1431	764	4510	93	2346
437	2	3	14531	15488	30243	437	14841	1867
438	1	3	10290	1981	2232	1038	168	2125
439	1	3	2787	1698	2510	65	477	52

440 rows × 8 columns

Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

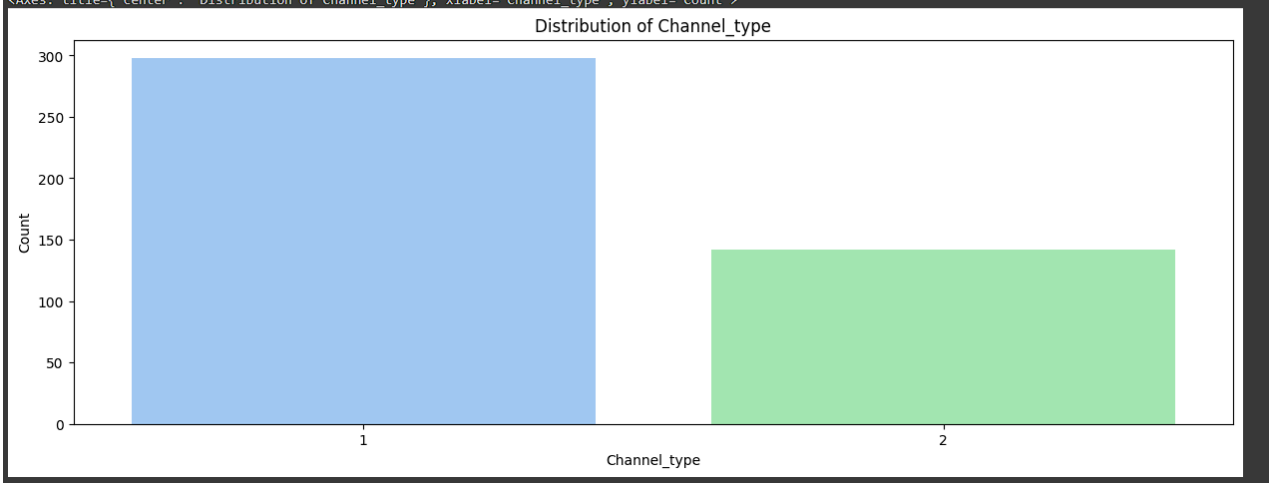
- Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
Channel_df=df['Channel'].value_counts().reset_index()
Channel_df.rename(columns={'index': 'Channel_type'}, inplace=True)
Channel_df.rename(columns={'Channel': 'Count'}, inplace=True)
Channel_df.head()
```

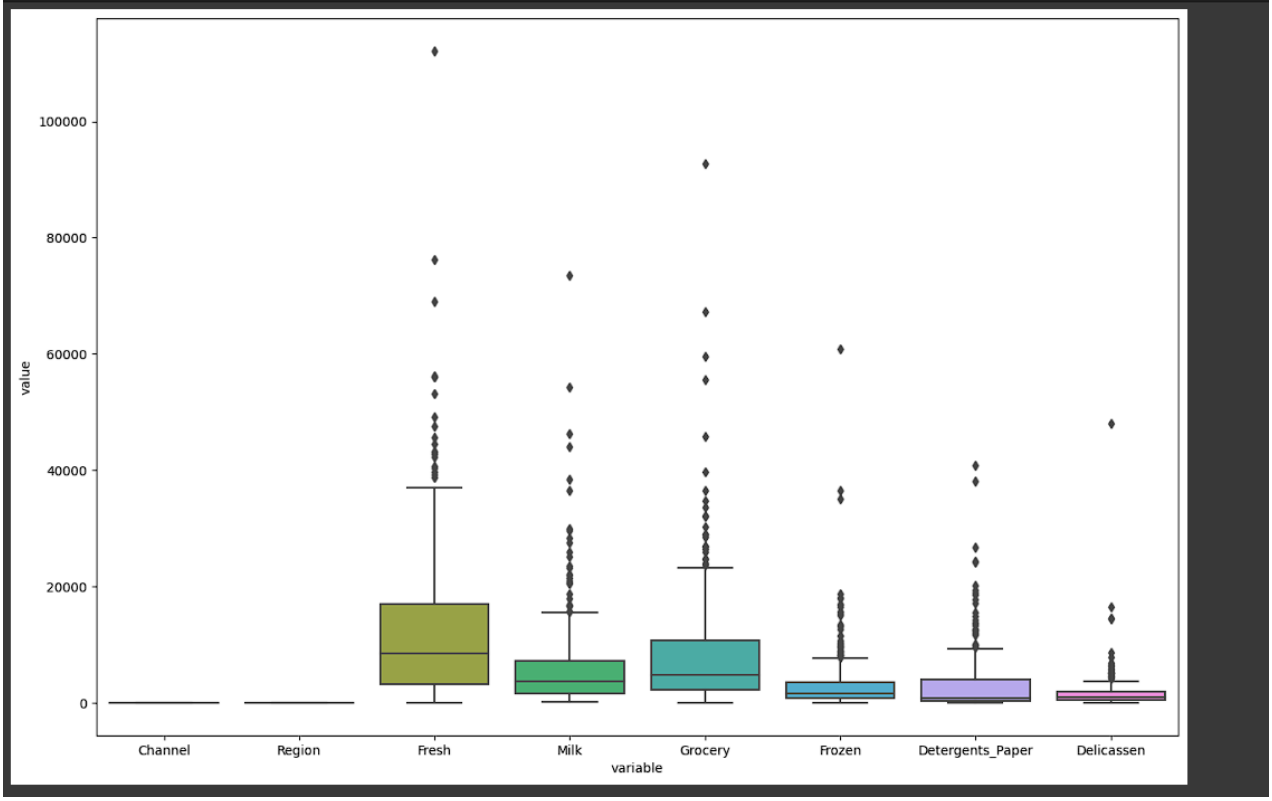
	Channel_type	Count
0	1	298
1	2	142

```
plt.figure(figsize=(15,5))
plt.title('Distribution of Channel_type')
sns.barplot(x='Channel_type',y='Count',data=Channel_df)
```

```
<Axes: title={'center': 'Distribution of Channel_type'}, xlabel='Channel_type', ylabel='Count'>
```



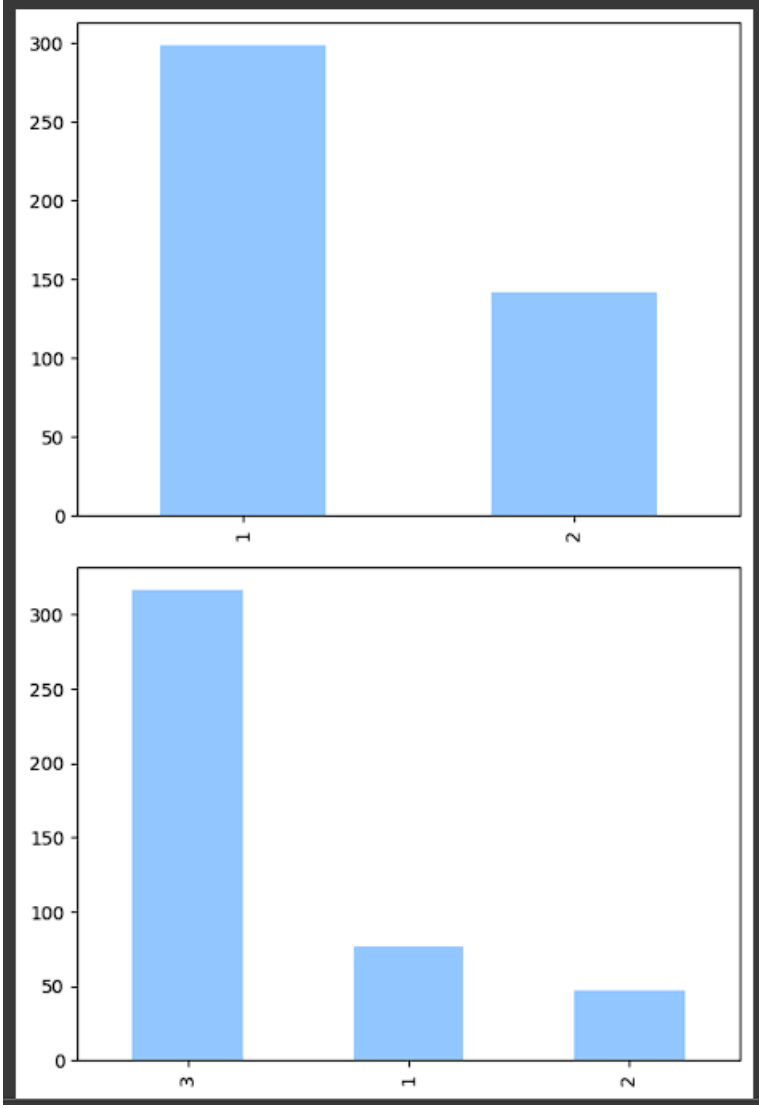

```
#Boxplot of each column
plt.figure(figsize=(15,10))
sns.boxplot(x='variable', y='value', data=df.melt())

plt.show()
```



Activity 4: Bivariate analysis

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

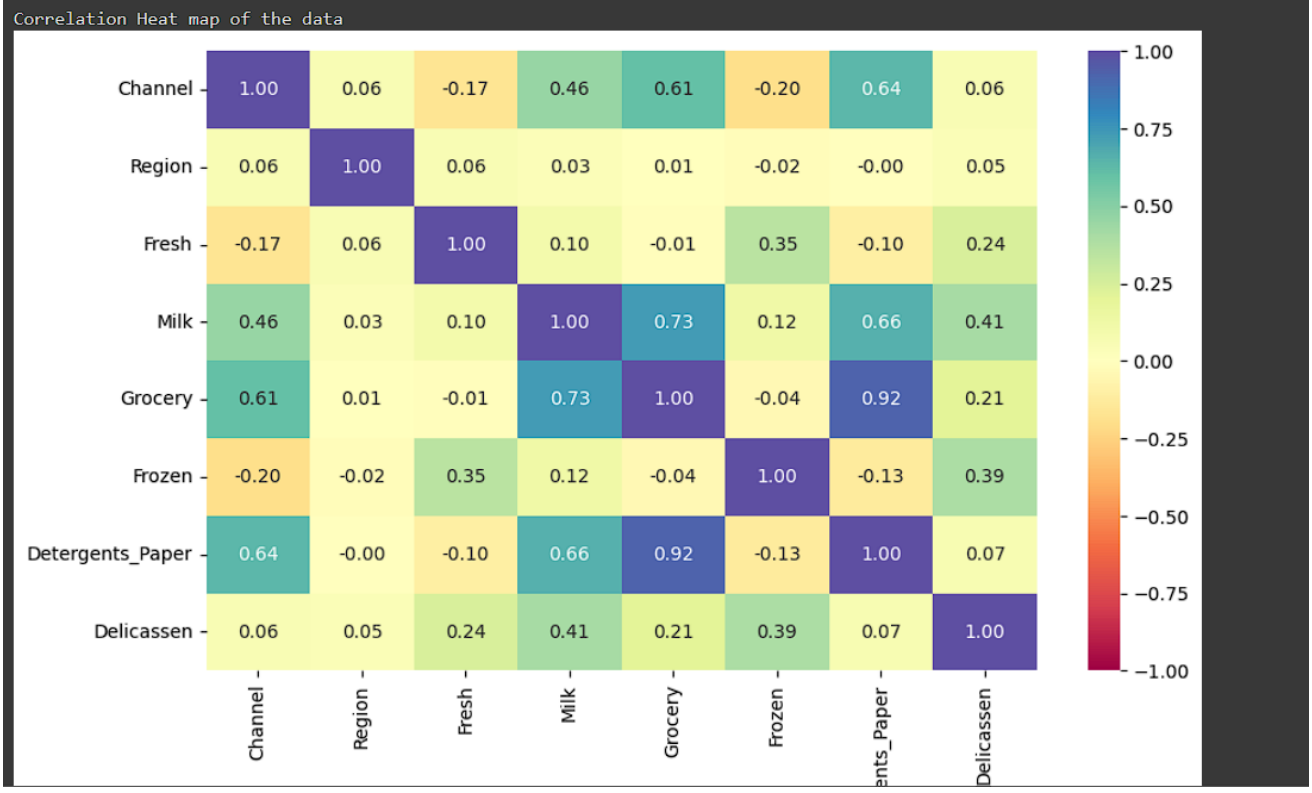
```
def categorical_data(i):
    df[i].value_counts().plot(kind='bar')

j_1 = ['Channel','Region']

for k in j_1:
    categorical_data(i=k)
    plt.show()
```




```
df.corr()
```

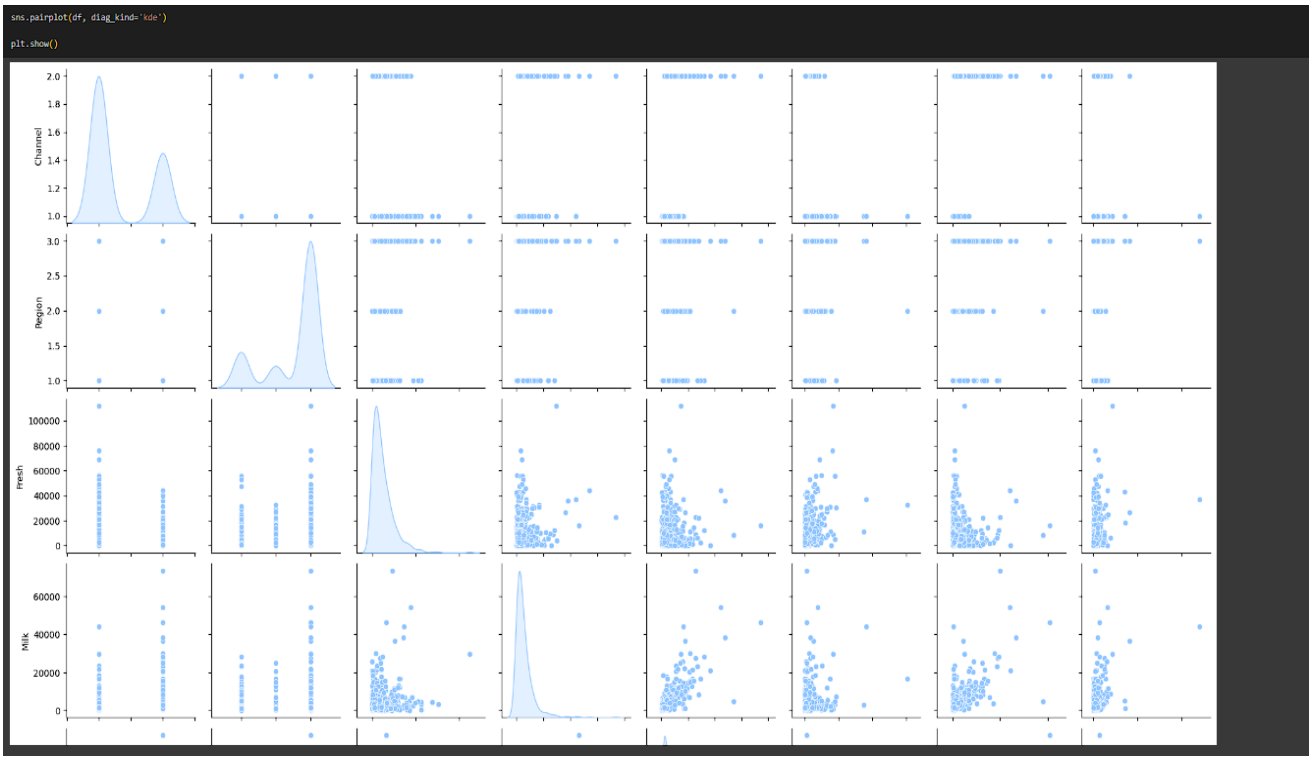
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Channel	1.000000	0.062028	-0.169172	0.460720	0.608792	-0.202046	0.636026	0.056011
Region	0.062028	1.000000	0.055287	0.032288	0.007696	-0.021044	-0.001483	0.045212
Fresh	-0.169172	0.055287	1.000000	0.100510	-0.011854	0.345881	-0.101953	0.244690
Milk	0.460720	0.032288	0.100510	1.000000	0.728335	0.123994	0.661816	0.406368
Grocery	0.608792	0.007696	-0.011854	0.728335	1.000000	-0.040193	0.924641	0.205497
Frozen	-0.202046	-0.021044	0.345881	0.123994	-0.040193	1.000000	-0.131525	0.390947
Detergents_Paper	0.636026	-0.001483	-0.101953	0.661816	0.924641	-0.131525	1.000000	0.069291
Delicassen	0.056011	0.045212	0.244690	0.406368	0.205497	0.390947	0.069291	1.000000

```
print('Correlation Heat map of the data')
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(),annot=True,fmt='.2f',vmin=-1,vmax=1,cmap='Spectral')
plt.show()
```



Activity 5: Multivariate analysis

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



Activity 6: 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.

Descriptive Analysis

```
[173] df.describe()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Milestone 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 Missing Values

```
#Checking for missing values
df.isnull().sum()
```

```
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

As we can see, there are no missing values.

- Handling categorical data

```
[ ]  df.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

- Handling outliers

```
[ ]  #defining a function to replace outliers with median
     def replace_outliers_with_median(df):
         Q1 = df.quantile(0.25)
         Q3 = df.quantile(0.75)
         IQR = Q3 - Q1
         lower_bound = Q1 - 1.5 * IQR
         upper_bound = Q3 + 1.5 * IQR
         return df.apply(lambda x: x if (x >= lower_bound and x <= upper_bound) else df.median())
```

```
[ ]  numerical_features = df.select_dtypes(include=[np.number])
     df[numerical_features.columns] = df[numerical_features.columns].apply(replace_outliers_with_median)
```

```
[ ]  df.describe()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	10033.079545	4281.829545	6218.863636	1896.656818	1806.022727	1084.296591
std	0.468052	0.774272	8450.546022	3402.380107	5339.702492	1674.915834	2257.986747	840.423787
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8489.500000	3623.500000	4754.750000	1521.500000	814.750000	964.750000
75%	2.000000	3.000000	14792.000000	6202.000000	8860.750000	2534.000000	2688.000000	1525.000000
max	2.000000	3.000000	37036.000000	15488.000000	23127.000000	7683.000000	9265.000000	3637.000000

- Scaling Techniques

```
[ ]  #Using standard scalaer let's Transform & Normalize the data and visulize it.
     sc=StandardScaler()
     scaled_data=sc.fit_transform(df)

     norm_data=normalize(df)

     df=pd.DataFrame(scaled_data,columns=df.columns)
     df_SN=pd.DataFrame(norm_data,columns=df.columns)
     df_SN.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286
2	0.000143	0.000214	0.453712	0.629040	0.548768	0.171758	0.251102	0.068953
3	0.000065	0.000194	0.856837	0.077254	0.272650	0.413659	0.032749	0.115494
4	0.000081	0.000121	0.914206	0.218698	0.290977	0.158263	0.071835	0.039030

- Splitting dataset into training and test set

```
[ ]  X = df.drop('Channel', axis=1)
     Y = df['Channel']
```

```
     #using 5 folds split the data.
     skf = KFold(n_splits=5)
     for train_index,test_index in skf.split(X, Y):
       X_train, X_test = X.loc[train_index],X.loc[test_index]
       Y_train, Y_test = Y.loc[train_index],Y.loc[test_index]


     print('{:<15} {:<15} {:<15}'.format('DataSet','Features','Label'))
     print('{:<15} {:<15} {:<15}'.format('Train',X_train.shape[0],Y_train.shape[0]))
     print('{:<15} {:<15} {:<15}'.format('Test',X_test.shape[0],Y_test.shape[0]))
```

```
DataSet         Features        Label
Train           352             352
Test            88              88
```

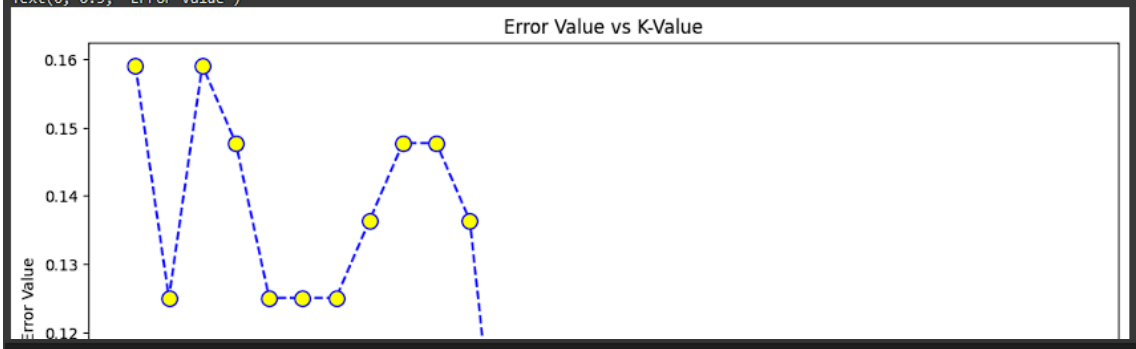
Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying Two classification algorithms KNN, Naive-Bayes algorithm. The best model is saved based on its performance.

1. K-Neighbors Classifier

```
#Choosing K value
error_rate = []

for i in range(1,30):
    KNN_class = KNeighborsClassifier(n_neighbors = i)
    KNN_class.fit(X_train, Y_train)
    Y_predict = KNN_class.predict(X_test)
    #appending error rate that is not equal to test dataset
    error_rate.append(np.mean(Y_predict != Y_test))

#Plotting the plot to check the K value from graph
plt.figure(figsize = (12,6))
plt.plot(range(1,30), error_rate, color = 'blue', linestyle = 'dashed', marker = 'o', markerfacecolor = 'yellow', markersize = 10)
plt.title('Error Value vs K-Value')
plt.xlabel('K-Value')
plt.ylabel('Error Value')
```

```
Text(0, 0.5, 'Error Value')
```




```
#As per above graph it seems neighbors 12 gives min error
KNN_class = KNeighborsClassifier(n_neighbors=12)

KNN_class.fit(X_train, Y_train)

Y_predict = KNN_class.predict(X_train)

print("Train Data Model Accuracy: {0:.4f}".format(metrics.accuracy_score(Y_train, Y_predict)*100))

Y_predict = KNN_class.predict(X_test)
```

```
Train Data Model Accuracy: 84.9432
```

2. Naive-Bayes algorithms

```
GNB = GaussianNB()

GNB.fit(X_train, Y_train)

Y_predict = GNB.predict(X_train)

print("Train Data Model Accuracy: {0:.3f}".format(metrics.accuracy_score(Y_train, Y_predict)*100))

Y_predict = GNB.predict(X_test)
```

```
Train Data Model Accuracy: 84.659
```

Model Evaluation:

We need to evaluate both the models to find which model we should use while predicting for the application.

	Model	Train Accuracy	Test Accuracy	F1-Score	Recall	Precision	AUC	Best_Cross_Val_Score	False Negatives	False Positives	True Negatives	True Positives
0	KNN	84.94	90.91	0.79	0.88	0.71	10.11	0.84 (+/- 0.05)	2	6	65	15
1	Naive Bayes	84.66	86.36	0.68	0.76	0.62	17.40	0.84 (+/- 0.03)	4	8	63	13

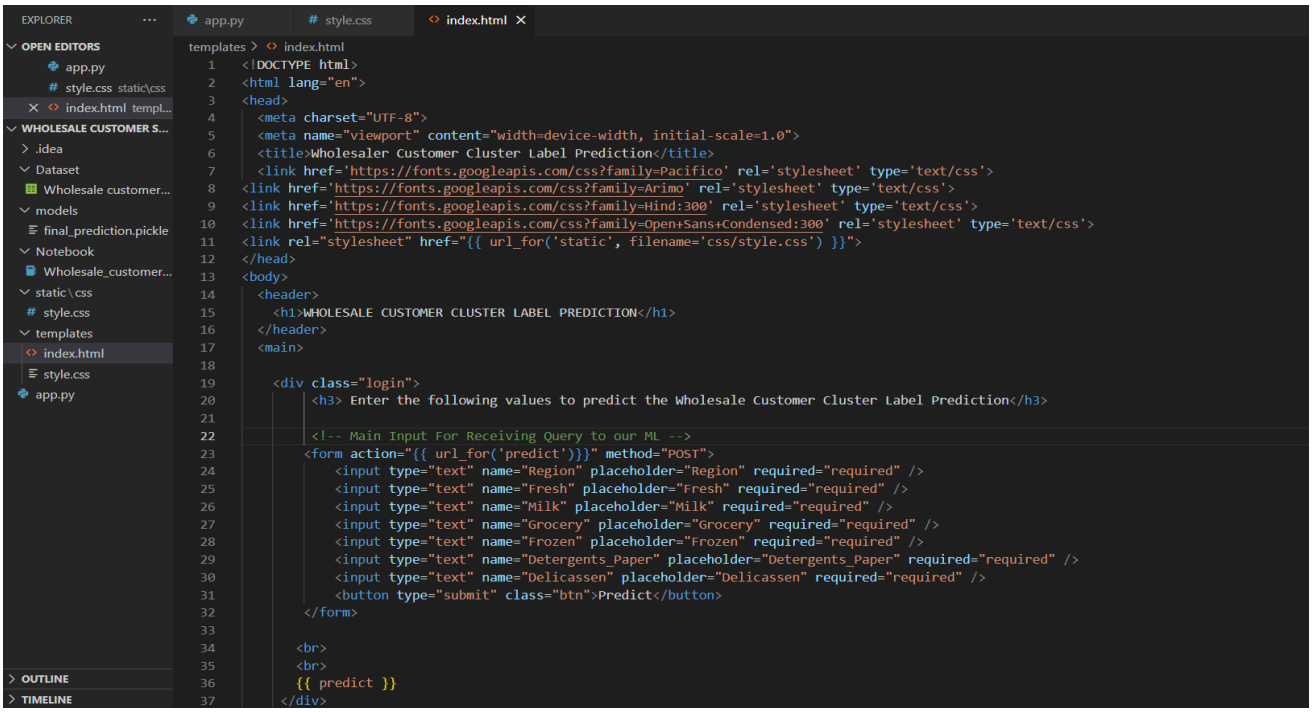
Milestone 5: Application Building

After the model is built, we will be integrating it to a web application so that normal users can also use it. The UI is based on taking input for various parameters and a predict button which will use the ML model to predict which cluster label the given input parameters correspond to.

Activity1: Building Html Pages:

For this project create an HTML file and a CSS file for styling namely

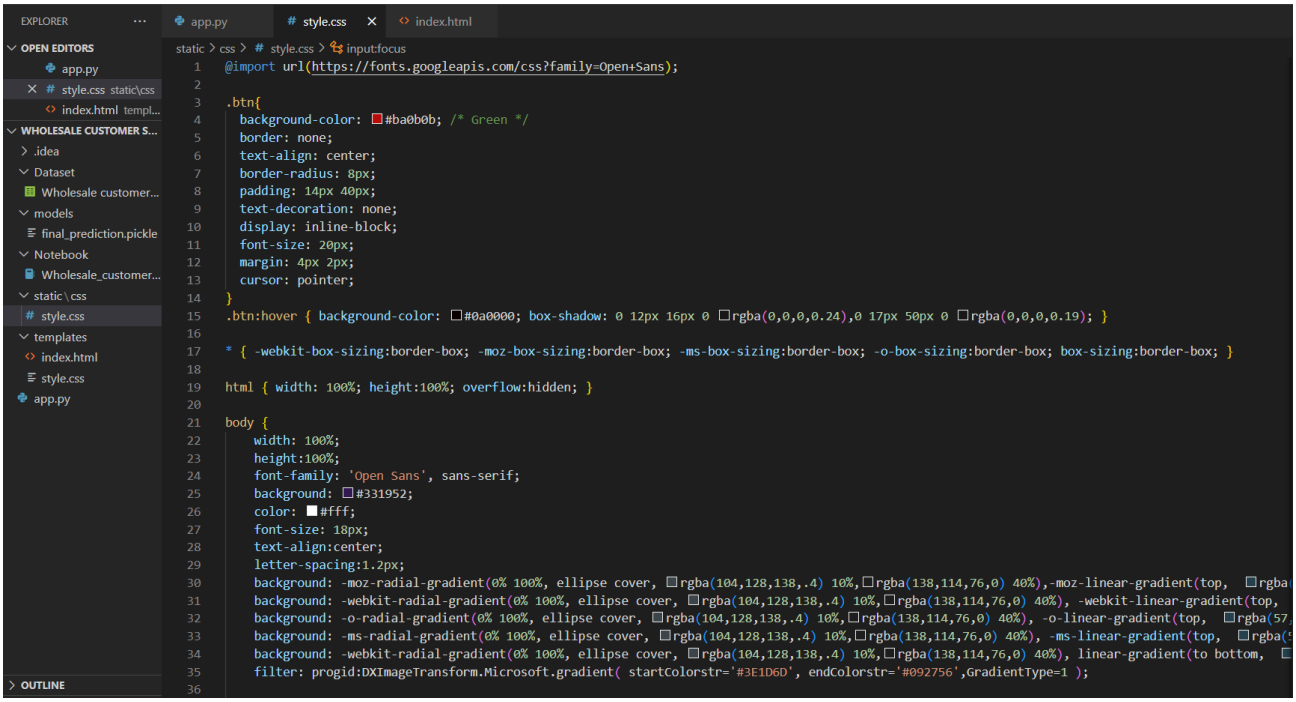
- Index.html
- Style.css


EXPLORER
app.py
style.css
index.html
templates > index.html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Wholesaler Customer Cluster Label Prediction</title>
<link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
<header>
<h1>WHOLESALE CUSTOMER CLUSTER LABEL PREDICTION</h1>
</header>
<main>

<h3> Enter the following values to predict the Wholesale Customer Cluster Label Prediction</h3>
<!-- Main Input For Receiving Query to our ML -->
<form action="{{ url_for('predict')}}" method="POST">
<input type="text" name="Region" placeholder="Region" required="required" />
<input type="text" name="Fresh" placeholder="Fresh" required="required" />
<input type="text" name="Milk" placeholder="Milk" required="required" />
<input type="text" name="Grocery" placeholder="Grocery" required="required" />
<input type="text" name="Frozen" placeholder="Frozen" required="required" />
<input type="text" name="Detergents_Paper" placeholder="Detergents_Paper" required="required" />
<input type="text" name="Delicassen" placeholder="Delicassen" required="required" />
<button type="submit" class="btn">Predict</button>
</form>

{{ predict }}

OUTLINE
TIMELINE


EXPLORER
app.py
style.css
index.html
static > css > style.css > input:focus
@import url(https://fonts.googleapis.com/css?family=Open+Sans);
.btn{
background-color: #ba0b0b; /* Green */
border: none;
text-align: center;
border-radius: 8px;
padding: 14px 40px;
text-decoration: none;
display: inline-block;
font-size: 20px;
margin: 4px 2px;
cursor: pointer;
}
.btn:hover { background-color: #0a0000; box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24),0 17px 50px 0 rgba(0,0,0,0.19); }
* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }
html { width: 100%; height:100%; overflow:hidden; }
body {
width: 100%;
height:100%;
font-family: 'Open Sans', sans-serif;
background: #331952;
color: #fff;
font-size: 18px;
text-align:center;
letter-spacing:1.2px;
background: -moz-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%),-moz-linear-gradient(top,
background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -webkit-linear-gradient(top,
background: -o-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -o-linear-gradient(top,
background: -ms-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -ms-linear-gradient(top,
background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), linear-gradient(to bottom,
filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1);
OUTLINE

Activity 2: Build Python code:

Step 1: Import the libraries

```
from flask import Flask, render_template, url_for, request
import pickle as p
import pickle
from flask import Flask, request,jsonify,render_template
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

```

Step 2: Initialize the flask app and load the model

```
modelfile = 'models/final_prediction.pickle'
model = p.load(open(modelfile, 'rb'))
app = Flask(__name__)
```

Step 3: Render the HTML pages.

```
@app.route('/')
def welcome():
    return render_template('index.html')
```

Step 4: Build the predict function which will take inputs from the UI and then use it to predict the cluster label and give the prediction to the submit in HTML page.

```python
@app.route('/predict', methods =['GET','POST'])
def predict():
    Region = float(request.form['Region'])
    Fresh = float(request.form['Fresh'])
    Milk = float(request.form['Milk'])
    Grocery = float(request.form['Grocery'])
    Frozen = float(request.form['Frozen'])
    Detergents_Paper = float(request.form['Detergents_Paper'])
    Delicassen = float(request.form['Delicassen'])

    total = [[Region,Fresh,Milk,Grocery,Frozen,Detergents_Paper,Delicassen]]
    prediction = model.predict(total)
    prediction = int(prediction[0])


    if prediction==1:
        return render_template('index.html',predict="Customer belongs to Cluster Label 1")
    else:
        return render_template('index.html',predict="Customer belongs to Cluster Label 2")
```

Main Function:

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

Activity 3: Run the application

Run the flask application using the run method. By default, the flask runs on port 5000. If the port is to be changed, an argument can be passed and the port can be modified.

- Run the application using the anaconda command prompt or the powershell terminal in Virtual Studio Code if required libraries are installed.
- We need to move to the folder where the project is stored in the terminal and then type the command “python app.py” to run the flask application.

```
PS C:\Users\Asus Tuf A15\Wholesale Customer segmentation> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 577-256-571
```

Final Output :