TABLE OF CONTENTS

ABSTRACT

Businesses now have access to vast consumer datasets, because of big-data so to control the decision-making in marketing and sales using large point-of-sale (POS) customer data in retail settings, a new approach has been proposed to enhance market segmentation and predict segment purchase probabilities. By representing different market segments and then predicting their future likelihood of purchasing the advertised products through target sales and marketing which increases the sales and business of the organization, which is accomplished by using the 3 properties of products like Recency, Value, Frequency. Source of the Dataset is Kaggle "Online Retail" with no. of Observations being 541910 rows with Invoice ID, Invoice Date, Stock Code, Description, Quantity, Unit Price, Customer ID, Country Methodologies include using various models like logistic regression, decision trees, and clustering techniques such as K-means. Tools used python, R and sci-kit. Outcome will be a better a model that can predict the future purchases and segmentation of the market which results in better sales and marketing resource utilization while increasing the profits of the business.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. MARKET SEGMENTATION AND PURCHASE PREDICTION

Every year lakhs of amount of money is being wasted since the advertisements do not reach the target audience and customers, so previous models established for this problem only is accurate around 78% but this model discussed below has accuracy of around 88%.Using this model businesses can better predict the customers purchases and improve upon their existing advertising models.

The aim of this study is to analyze market segmentation through machine learning techniques, identifying distinct consumer groups based on purchasing behaviors, and develop predictive models to forecast future purchases

1.2. STATISTICAL INFORMATION

Large consumer data sets have become available to businesses in recent years, which presents possibilities for decision-making in the areas of marketing and sales. A deep understanding of customers may be made possible by large customer data sets. We use a large point-of-sale (POS) customer data in retail setting to propose a new technique to market segmentation and the detection of relative segment purchase probabilities in order to address this issue. In order to identify segments in the consumer data set, stage one of our technique uses supervised and unsupervised learning algorithms that analyse three aspects of purchases (Recency, Frequency, and Monetary value, or RFM) and product properties.Market basket analysis (MBA) is used in stage two of our approach to ascertain the likelihood of buying behaviours for each segment.

1.3 METHODS

The aim of this study is to analyze market segmentation through machine learning techniques, identifying distinct consumer groups based on purchasing behaviors, and develop predictive models to forecast future purchases which is accomplished by using the 3 properties of products like Recency, Value, Frequency. Value(revenue) is obtained by multiplying Quantity and Unit Price After this , we can apply K-means clustering to each factor of the data, Frequency is obtained by counting the number of times they buy from the online retail shop, the total number of orders made by each customer. and finally the Recency is obtained by finding the most recent purchase date of each customer and see how many days they have been inactive. Afterwards, we can apply K-means clustering to assign customers a recency score through this customer segmentation is done. Purchase prediction is done by using recency column the variable “next purchase day” = min purchase day- max purchase day , and using this we set if next purchase day is > 90 then the customer will probably not buy the next quarter too, feed this information into the prediction algorithms like RF or XGBoost these will then result in purchase prediction.

2. LITERATURE REVIEW

This paper deals with segmentation using various algorithms like k-means and purchase prediction is done with [1] with A Machine-Learnt Approach to Market Segmentation and Purchase Prediction Using Point-Of-Sale (POS) Data SVM, RFC,Naïve Bayes SVM - 75%RFC-77%Naïve Bayes-79% Less accuracy percentage hence my model gives better results.

This paper[2] Customer’s Purchase Prediction Using Customer Segmentation Approach for Clustering of Categorical Data. (2023, July 26). Management and Production Engineering Review. Sometimes cause consumers to be assigned to one cluster, but in a re-analysis, they are assigned to another cluster new algorithms can be used because the degrees of accuracy achieved are not yet entirely satisfactory

Paper [3] Sales Prediction and Product Recommendation Model Through User Behavior Analytics uses XGBoost, RF, 5-Fold Cross Validation 77.82% Accuracy Less accuracy percentage The cold start problem and Shilling attack

Paper [4] Customer Segmentation Using Machine Learning usues only K-Means,The value of silhouette index is 0.442 Low silhouette index Better model can be used

3. OBJECTIVE

The aim of this study is to analyze market segmentation through machine learning techniques, identifying distinct consumer groups based on purchasing behaviors, and develop predictive models to forecast future purchases.

To produce a better model with better accuracy for the given Dataset.

4. PROPOSED METHODOLOGY

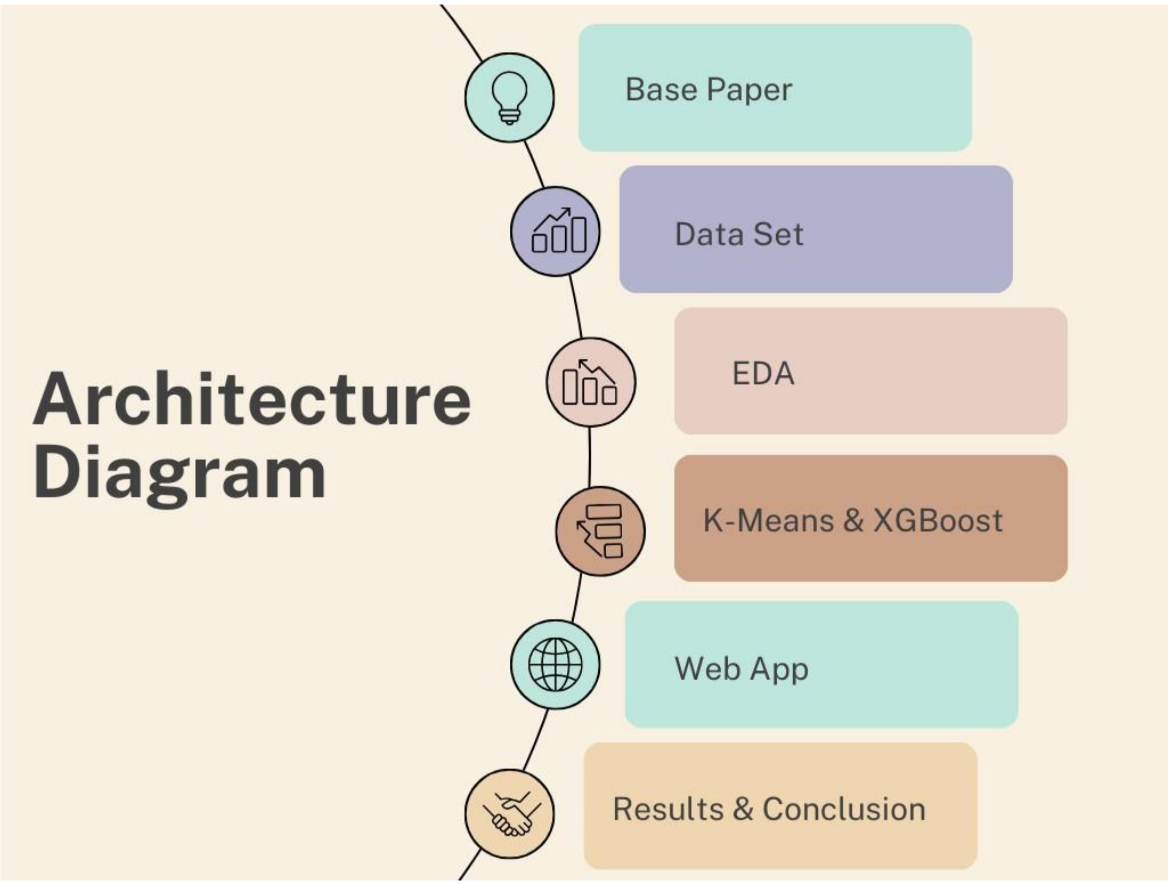

Fig 4.1 Methodology

K-Means clustering is used for market segmentation it is a popular clustering algorithm used in machine learning and data mining. Its primary goal is to partition a dataset into K clusters, where each data point belongs to the cluster with the nearest mean (centroid), clustering is done based on Revenue, Frequency and Recency.

Random Forest is a versatile and powerful ensemble learning technique used for both classification and regression tasks in machine learning. It operates by constructing multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees since Random Forest is an ensemble method composed of multiple decision trees improve performance. Combining prediction

ns from various trees helps to reduce overfitting that might be present in individual trees, leading to a more generalized model.

XGBoost

Speed: It's highly optimized for performance, often significantly faster than traditional gradient boosting implementations. Parallel processing and tree pruning techniques contribute to its speed.

Scalability: Handles large datasets efficiently due to its scalability and ability to work with a vast number of observations and features.

Missing Values Handling: Can handle missing data internally during training, reducing the need for extensive preprocessing.

Categorical Variable Support: Handles categorical variables well without requiring one-hot encoding, simplifying preprocessing steps

XGBoost has been widely used and proven effective in winning solutions in machine learning competitions like Kaggle. Its performance in such competitive environments showcases its capabilities.

Robustness: It's robust to outliers and noise in the data, thanks to its capacity to handle irregularities and patterns effectively.

Customizable: Offers a wide range of hyperparameters for fine-tuning the model according to specific dataset requirements.

All these algorithms take accuracy, recall and precision of train and test dataset as baselines for comparison

Source of the Dataset is Kaggle "Online Retail" with no. of Observations being 541910 rows with Invoice ID, Invoice Date, Stock Code, Description, Quantity, Unit Price, Customer ID, Country

Data validity and reliability is done by removing null values and removing outliers and analysis will be discussed later

5. TOOLS AND TECHNIQUES

There are several tools commonly used for deep learning classification, depending on the specific application and the level of expertise of the practitioner. Here are some popular tools used for deep learning classification:

Tools used python, R and sci-kit. Outcome will be a better a model that can predict the future purchases and segmentation of the market which results in better sales and marketing resource utilization while increasing the profits of the business.

Scikit-learn: Scikit-learn is a popular machine learning library for Python that includes many algorithms for classification tasks, including deep learning algorithms such as multi-layer perceptrons (MLPs).

6 IMPLEMENTATION

6.1 Market Segmentation and Purchase Prediction

6.1.1. ABOUT THE DATASET:

The Data is hand collected from various websites with each and every labels verified and sourced Kaggle.

Dataset

- Source of the Dataset

  Kaggle

- No. of Observations

  541910

- Column/Feature Details

  Invoice ID*(discrete)*, Invoice Date*(date/time)*, Stock Code(*discrete*), Description(*categorical*), Quantity(*discrete*), Unit Price(*continuous*), Customer ID(*categorical*), Country(*categorical*)

6.1.2. PREPROCESSING

Pre-processing is just as important in deep learning as it is in other areas of machine learning and data analysis. In fact, it can be argued that it is even more critical in deep learning due to the complexity of the models and the large amount of data typically involved.

.

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

     InvoiceNo         0
     StockCode         0
     Description      45
     Quantity          1
     InvoiceDate       1
     UnitPrice         1
     CustomerID     3506
     Country           1
     dtype: int64
```

```
[ ]  data=dataset.dropna()

[ ]  data.isnull().sum()

     InvoiceNo      0
     StockCode      0
     Description    0
     Quantity       0
     InvoiceDate    0
     UnitPrice      0
     CustomerID     0
     Country        0
     dtype: int64
```

Fig 6.1 EDA

6.1.3. MODELS

K-Means clustering is used for market segmentation it is a popular clustering algorithm used in machine learning and data mining. Its primary goal is to partition a dataset into K clusters, where each data point belongs to the cluster with the nearest mean (centroid), clustering is done based on Revenue, Frequency and Recency.

Random Forest is a versatile and powerful ensemble learning technique used for both classification and regression tasks in machine learning. It operates by constructing multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees since Random Forest is an ensemble method composed of multiple decision trees improve performance. Combining predictions from various trees helps to reduce overfitting that might be present in individual trees, leading to a more generalized model.

XGBoost

- Speed: It's highly optimized for performance, often significantly faster than traditional gradient boosting implementations. Parallel processing and tree pruning techniques contribute to its speed.
- Scalability: Handles large datasets efficiently due to its scalability and ability to work with a vast number of observations and features.
- Missing Values Handling: Can handle missing data internally during training, reducing the need for extensive preprocessing.
- Categorical Variable Support: Handles categorical variables well without requiring one-hot encoding, simplifying preprocessing steps
- XGBoost has been widely used and proven effective in winning solutions in machine learning competitions like Kaggle. Its performance in such competitive environments showcases its capabilities.
- Robustness: It's robust to outliers and noise in the data, thanks to its capacity to handle irregularities and patterns effectively.
- Customizable: Offers a wide range of hyperparameters for fine-tuning the model according to specific dataset requirements.

Cross-validation Results

```
LogisticRegression: 0.8571428571428571, 0.8712797619047619
GaussianNB: 0.8147321428571429, 0.8444940476190477
RandomForestClassifier: 0.8683035714285714, 0.8668154761904762
SVC: 0.7938988095238095, 0.8058035714285714
DecisionTreeClassifier: 0.8162202380952381, 0.8333333333333334
xgb.XGBClassifier: 0.8430059523809523, 0.8586309523809523
KNeighborsClassifier: 0.7834821428571429, 0.7886904761904762
```

Random Forest Classifier Metrics

Accuracy on Training Set: 1.00

Accuracy on Test Set: 0.86

Classification Report:

```
precision    recall  f1-score   support

           0       0.89      0.91      0.90       477
           1       0.77      0.73      0.75       195

    accuracy                           0.86       672
   macro avg       0.83      0.82      0.83       672
weighted avg       0.86      0.86      0.86       672
```

Fig 6.2 Ensemble and Random Forest

Refined XGBoost Classifier Metrics

Accuracy on Training Set: 0.91

Accuracy on Test Set: 0.88

Fig 6.3 Refined XGBoost

7. RESULTS AND DISCUSSIONS

MODEL	ACCURACY
Refined XGBoost	**89%**
Random Forest	87%
XGBoost	86%
Logistic Reg	86%
Decision Tree	82%
KNN	79%
SVC	78%

Tab 7.1 Model and Accuracy

It is clear that Random forest and XGBoost gives the highest accuracy

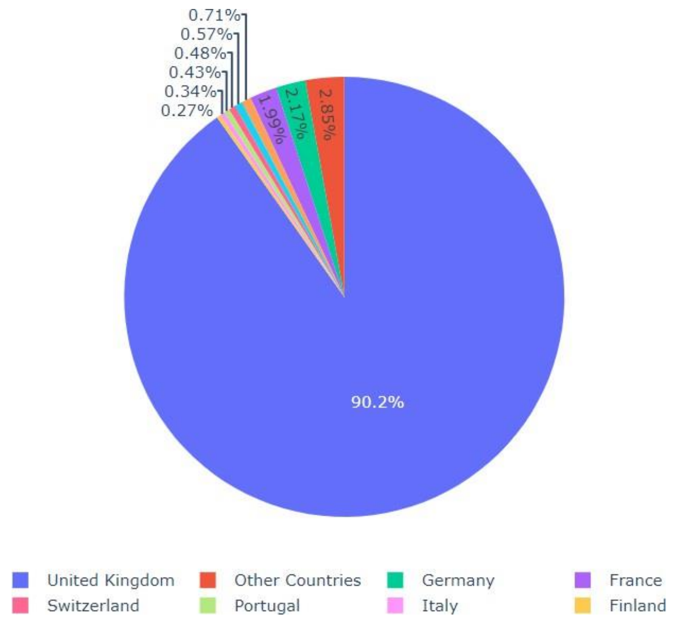

Fig 7.2 Country vs Quantity

Country vs Quantity:

From the chart it is clear that UK has highest count of 90% and other countries percentages of counts are also given.

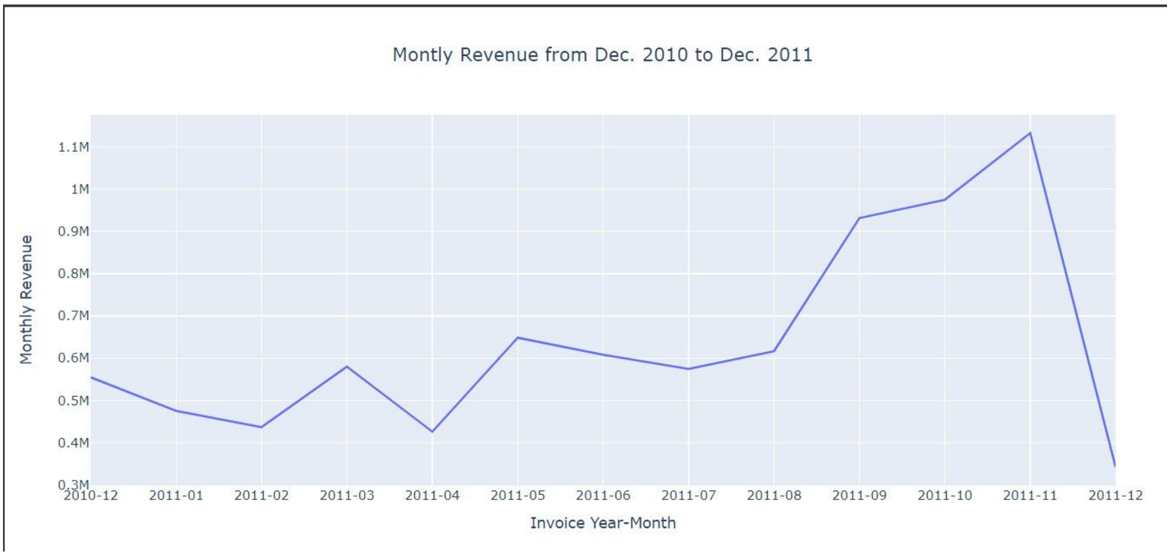

Fig 7.3 Date/Time vs Revenue

Revenue from 2010 to 2011 December
In the month of November the revenue is higher than others from 2010 -2011

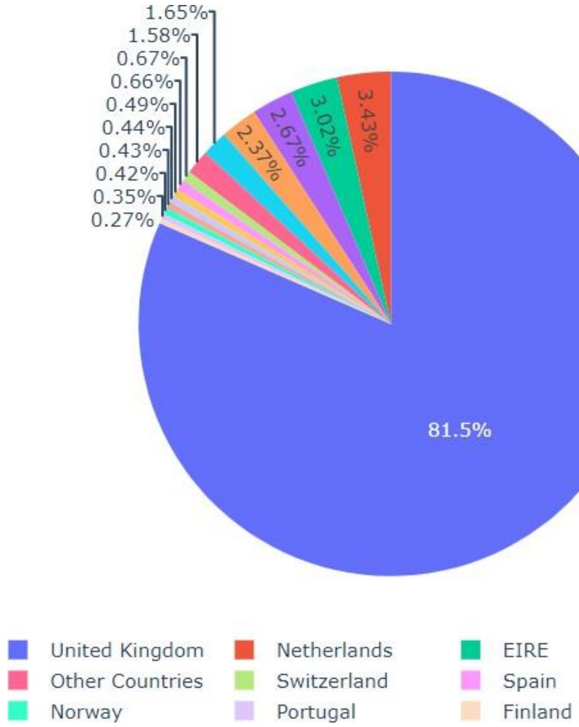

Fig 7.4 Country vs Revenue

Country vs Revenue:

From the chart it is clear that UK has highest revenue of 81.5% and other countries percentages of revenue are also given.

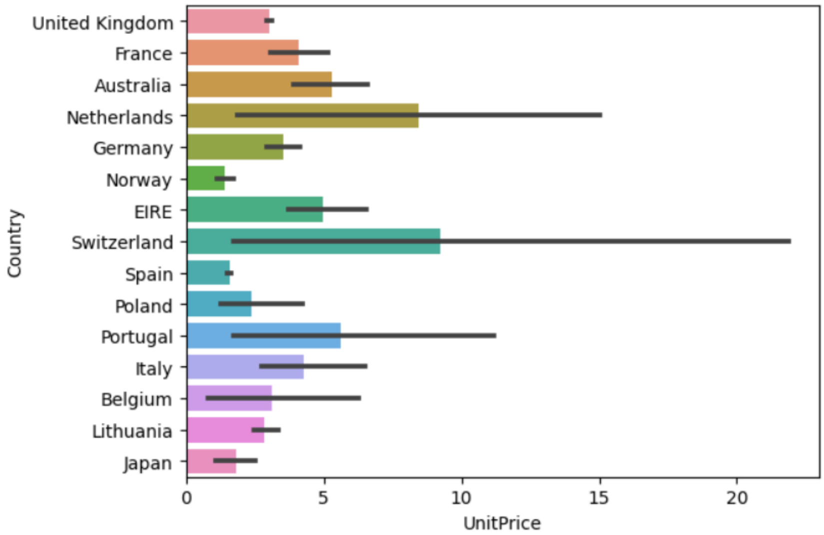

Fig 7.5 Country vs Unit price

This chart shows the Switzerland buys highest UnitPrice per item followed by Netherlands.

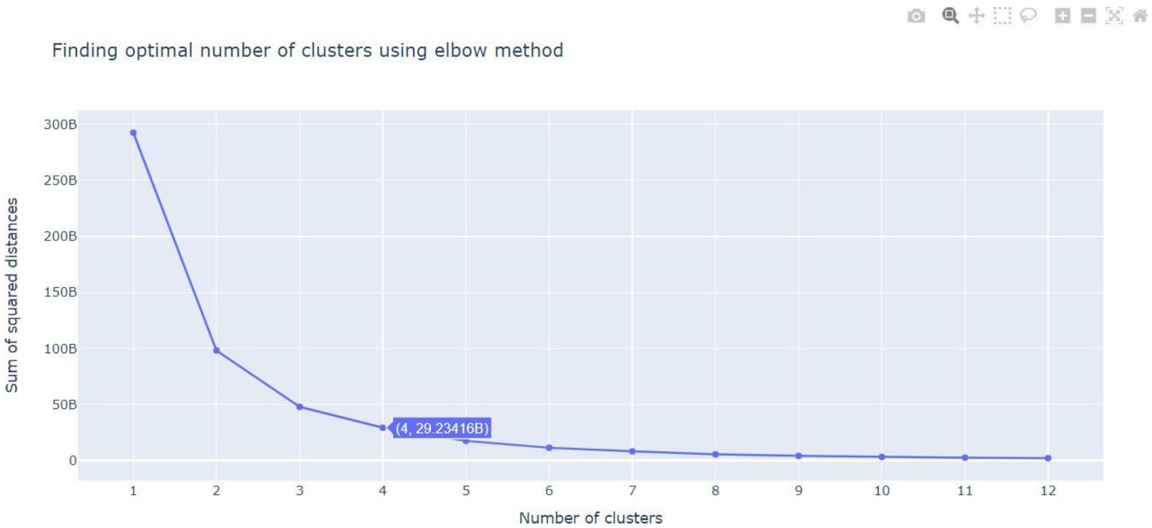

Fig 7.6 Clusters

This shows that we can take around 4 clusters

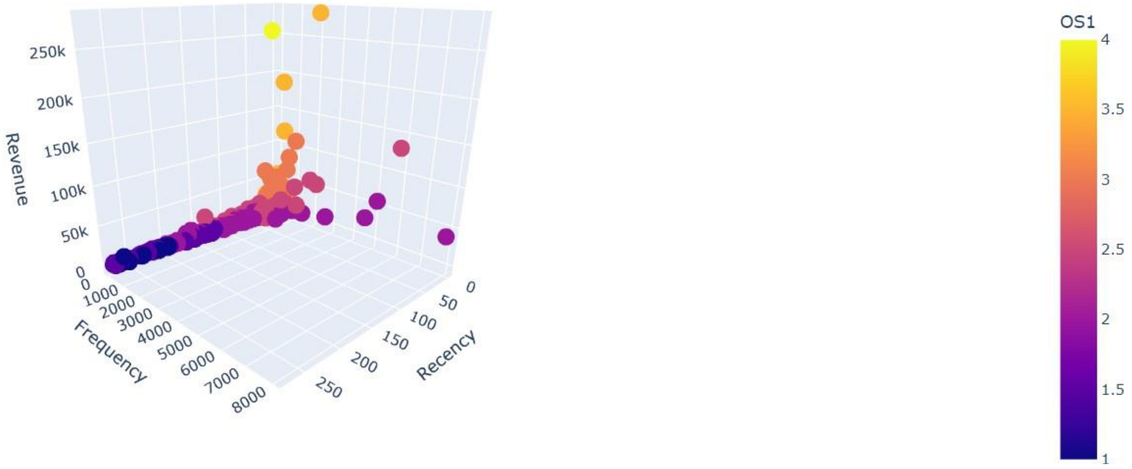

Fig 7.7 Segmentation of the customers

```
refined_xgb_model = xgb.XGBClassifier(eval_metric='logloss',
                                      max_depth=list(p_grid_search.best_params_.values())[0]-1,
                                      min_child_weight=list(p_grid_search.best_params_.values())[-1]+4
                                     ).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'.format(refined_xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'.format(refined_xgb_model.score(X_test[X_train.columns], y_test)))

Accuracy of XGB classifier on training set: 0.90
Accuracy of XGB classifier on test set: 0.89
```

Fig 7.8 Refined XGBooost

This shows that this refined XGBoost gives the highest accuracy amongst all other algorithms.

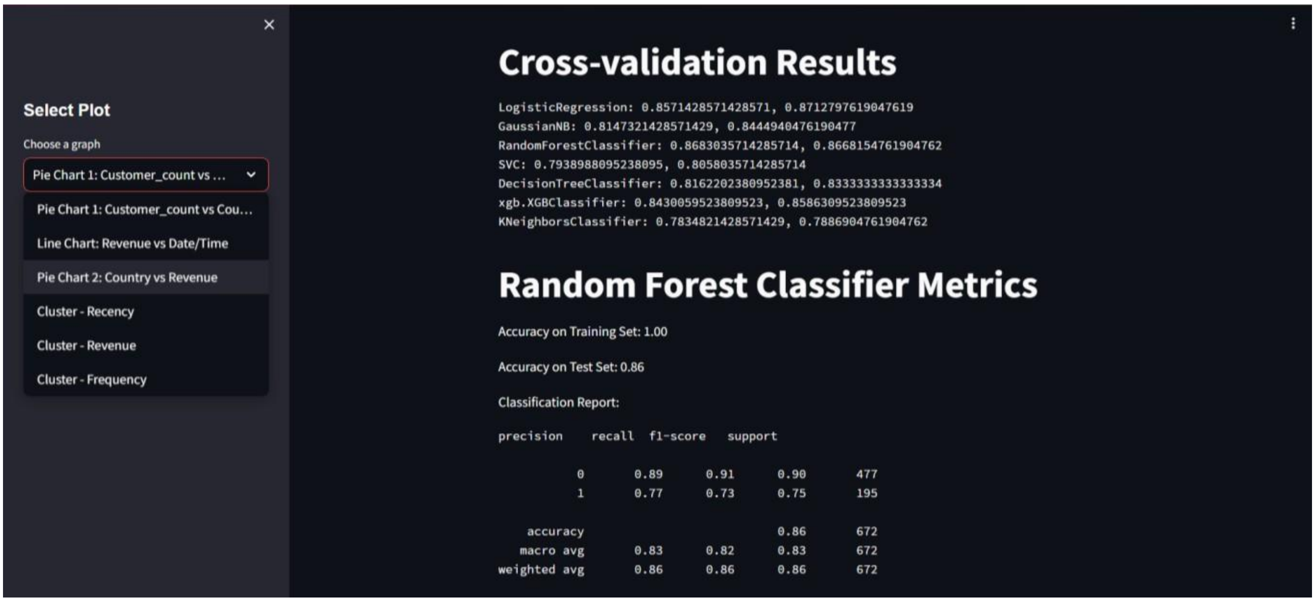

Fig 7.9 Web App

8. CONCLUSION

- Represented different market segments and then predicted their future likelihood for purchasing the products through target sales and marketing which increase the sales and business of the organization using Random Forest and XGBoost with accuracy of 89%.
- This reduces the amount spend on advertising and sales eventually resulting in better business model.

9. FUTURE ENHANCEMENT

Increase the accuracy of the model and so better understanding and prediction can be performed which will benefit the business while also improving the efficiency of the model by saving lakhs of money spent on advertising , sales and marketing.

10. APPENDICIES

10.1 FULL CODE

```
!pip install streamlit
!pip install streamlit pyngrok
!pip install pyngrok
```

```
%%writefile app.py
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import seaborn as sns
import streamlit as st
import plotly.express as px
#import plotly.offline as pyoff
import plotly.graph_objs as go
#import plotly.figure_factory as ff
#AUC, confusion matrix


from sklearn.svm import SVC
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score,
train_test_split, GridSearchCV, cross_validate
from sklearn.metrics import accuracy_score, f1_score,
precision_score, recall_score, confusion_matrix
from sklearn.cluster import KMeans
import xgboost as xgb
import time
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold, cross_val_score,
train_test_split, GridSearchCV, cross_validate
from sklearn.metrics import accuracy_score, f1_score,
precision_score, recall_score, confusion_matrix

#-----------------------------------------------------------------------------------------------------------------
```

```
dataset=pd.read_excel("Online_Retail.xlsx")
print(dataset)
data=dataset.dropna()
data.InvoiceDate = pd.to_datetime(data.InvoiceDate)

data.head()

print('From the dataset, the online retail shop has {} customers \
from {} different countries.'.format(len(data.CustomerID.unique()),
len(data.Country.unique())))
ctm_cntry_df = data.groupby(['CustomerID',
'Country']).count().reset_index()

ctm_cntry_df =
ctm_cntry_df.groupby('Country')['CustomerID'].count().reset_index().s
ort_values(
    by=['CustomerID'], ascending=False)

ctm_cntry_df['Percentage']= np.round(ctm_cntry_df.CustomerID /
ctm_cntry_df.CustomerID.sum() * 100, 2)

ctm_cntry_df.head(10)
ctm_2_cntry = {}


for idx, cid in enumerate(data.CustomerID.unique()):


    cntry = data[data.CustomerID == cid].Country.unique()


    if len(cntry) > 1:


        ctm_2_cntry[cid] = cntry

pd.DataFrame(ctm_2_cntry) # Create a pandas dataframe using
ctm_2_cntry
percent_margin = 0.25


ctm_cntry_df['CountryCategory'] = ctm_cntry_df.Country



ctm_cntry_df.loc[ctm_cntry_df.Percentage <= percent_margin,
'CountryCategory'] = 'Other Countries'

ctm_cntry_df.head(11)
```

```
############################################
pie_fig = px.pie(ctm_cntry_df,
                    names="CountryCategory",
                    values="Percentage",
                    title="Customer Country Count in Percentage"
                    )

pie_fig.update_layout(title_x=0,
                          legend_title="Countries Represented",
                          legend=dict(orientation="h")
                        )

pie_fig.show(config={'displaylogo': False})
#############################################
data['InvoiceYearMonth'] = data['InvoiceDate'].map(lambda date:
100*date.year + date.month)
data.head()
data['Revenue'] = data.UnitPrice * data.Quantity
data.head()
ctm_revenue =
data.groupby('InvoiceYearMonth').Revenue.sum().reset_index()
ctm_revenue.head()
ctm_revenue['InvoiceYearMonth'] =
ctm_revenue['InvoiceYearMonth'].apply(lambda x: f"{x // 100}-{x %
100:02}")
line_fig = px.line(ctm_revenue,
                   x = "InvoiceYearMonth",
                   y = "Revenue",
                   title = "Montly Revenue from Dec. 2010 to Dec.
2011"
                  )

line_fig.update_layout(title_x=0.5,
                       showlegend=False,
                       xaxis={"type": "category"},
                       xaxis_title="Invoice Year-Month",
                       yaxis_title="Monthly Revenue"
                      )

line_fig.show(config={'displaylogo': False})
cntry_revenue_df =
data.groupby(['Country']).Revenue.sum().reset_index().sort_values(by=
['Revenue'],

                  ascending=False)

cntry_revenue_df['Percentage'] = np.round(cntry_revenue_df.Revenue /
cntry_revenue_df.Revenue.sum() * 100, 2)
```

```
cntry_revenue_df.head(5)
percent_margin = 0.25


cntry_revenue_df['CountryCategory'] = cntry_revenue_df.Country



cntry_revenue_df.loc[cntry_revenue_df.Percentage <= percent_margin,
'CountryCategory'] = 'Other Countries'

cntry_revenue_df.head(11)
pie_fig1 = px.pie(cntry_revenue_df,
                  names="CountryCategory",
                  values="Percentage",
                  title="Country Revenue in Percentage"
                 )

pie_fig1.update_layout(title_x=0,
                       legend_title="Countries Represented",
                       legend=dict(orientation="h")
                      )

pie_fig1.show(config={'displaylogo': False})
ctm_bhvr_dt = data[(data.InvoiceDate < pd.Timestamp(2011,9,1)) &
      (data.InvoiceDate >=
pd.Timestamp(2010,12,1))].reset_index(drop=True)


ctm_next_quarter = data[(data.InvoiceDate < pd.Timestamp(2011,12,1))
&
      (data.InvoiceDate >=
pd.Timestamp(2011,9,1))].reset_index(drop=True)
ctm_dt = pd.DataFrame(ctm_bhvr_dt['CustomerID'].unique())


ctm_dt.columns = ['CustomerID']
ctm_1st_purchase_in_next_quarter =
ctm_next_quarter.groupby('CustomerID').InvoiceDate.min().reset_index(
)
ctm_1st_purchase_in_next_quarter.columns =
['CustomerID','MinPurchaseDate']
ctm_last_purchase_bhvr_dt =
ctm_bhvr_dt.groupby('CustomerID').InvoiceDate.max().reset_index()
ctm_last_purchase_bhvr_dt.columns = ['CustomerID','MaxPurchaseDate']
ctm_purchase_dates = pd.merge(ctm_last_purchase_bhvr_dt,
ctm_1st_purchase_in_next_quarter, on='CustomerID',
                              how='left')
```

```
ctm_purchase_dates['NextPurchaseDay'] =
(ctm_purchase_dates['MinPurchaseDate'] -
ctm_purchase_dates['MaxPurchaseDate']).dt.days
ctm_dt = pd.merge(ctm_dt,
ctm_purchase_dates[['CustomerID','NextPurchaseDay']],
on='CustomerID', how='left')
ctm_dt = ctm_dt.fillna(9999)
ctm_max_purchase =
ctm_bhvr_dt.groupby('CustomerID').InvoiceDate.max().reset_index()
ctm_max_purchase.columns = ['CustomerID','MaxPurchaseDate']
ctm_max_purchase['Recency'] =
(ctm_max_purchase['MaxPurchaseDate'].max() -
ctm_max_purchase['MaxPurchaseDate']).dt.days


ctm_dt = pd.merge(ctm_dt, ctm_max_purchase[['CustomerID',
'Recency']], on='CustomerID')
hist_fig = px.histogram(ctm_dt,
                        x="Recency",
                        title="Customers Recency in
Days"                       )

hist_fig.update_layout(title_x=0.5,
                       xaxis_title="Recency in groups of 20 days",
                       yaxis_title="Number of Customers"
                      )

hist_fig.show(config={'displaylogo': False})
######################
my_dict={}
ctm_recency = ctm_dt[['Recency']]
for idx in range(1, 10):
    kmeans = KMeans(n_clusters=idx, max_iter=1000).fit(ctm_recency)
    ctm_recency["clusters"] = kmeans.labels_
    my_dict[idx] = kmeans.inertia_

line_fig1 = px.line(x=list(my_dict.keys()),
                   y=list(my_dict.values()),
                   template="plotly_dark"
                  )

line_fig1.update_layout(title_x=0,
                       xaxis_title="Number of cluster",
                       yaxis_title="Recency"
                      )

line_fig1.show(config={'displaylogo':  False})
```

```
number_of_clusters = 4
kmeans = KMeans(n_clusters=number_of_clusters)
kmeans.fit(ctm_dt[['Recency']])
ctm_dt['RecencyCluster'] = kmeans.predict(ctm_dt[['Recency']])
def order_cluster(df, target_field_name, cluster_field_name,
ascending):
    """
    INPUT:
        - df                    - pandas DataFrame
        - target_field_name     - str - A column in the pandas
DataFrame df
        - cluster_field_name  - str - Expected to be a column in the
pandas DataFrame df
        - ascending             - Boolean

    OUTPUT:
        - df_final              - pandas DataFrame with
target_field_name and cluster_field_name as columns

    """

    new_cluster_field_name = "new_" + cluster_field_name


    df_new =
df.groupby(cluster_field_name)[target_field_name].mean().reset_index(
)


    df_new = df_new.sort_values(by=target_field_name,
ascending=ascending).reset_index(drop=True)


    df_new["index"] = df_new.index


    df_final = pd.merge(df, df_new[[cluster_field_name, "index"]],
on=cluster_field_name)

    df_final = df_final.drop([cluster_field_name], axis=1)

    df_final = df_final.rename(columns={"index": cluster_field_name})

    return df_final
ctm_dt = order_cluster(ctm_dt, 'Recency', 'RecencyCluster', False)

ctm_frequency =
data.groupby('CustomerID').InvoiceDate.count().reset_index()
ctm_frequency.columns = ['CustomerID','Frequency']
```

```
ctm_dt = pd.merge(ctm_dt, ctm_frequency, on='CustomerID')
kmeans = KMeans(n_clusters=number_of_clusters)
kmeans.fit(ctm_dt[['Frequency']])
ctm_dt['FrequencyCluster'] = kmeans.predict(ctm_dt[['Frequency']])
ctm_dt = order_cluster(ctm_dt, 'Frequency', 'FrequencyCluster',
False)
ctm_revenue = data.groupby('CustomerID').Revenue.sum().reset_index()
ctm_dt = pd.merge(ctm_dt, ctm_revenue, on='CustomerID')
kmeans = KMeans(n_clusters=number_of_clusters)
kmeans.fit(ctm_dt[['Revenue']])
ctm_dt['RevenueCluster'] = kmeans.predict(ctm_dt[['Revenue']])
ctm_dt = order_cluster(ctm_dt, 'Revenue', 'RevenueCluster', True)
ctm_dt['OverallScore'] = ctm_dt['RecencyCluster'] +
ctm_dt['FrequencyCluster'] + ctm_dt['RevenueCluster']
ctm_dt.groupby('OverallScore')['Recency','Frequency','Revenue'].mean(
)
ctm_dt['Segment'] = 'Low-Value'
ctm_dt.loc[ctm_dt['OverallScore'] > 4, 'Segment'] = 'Mid-Value'
ctm_dt.loc[ctm_dt['OverallScore'] > 6, 'Segment'] = 'High-Value'
ctm_class = ctm_dt.copy()
ctm_class = pd.get_dummies(ctm_class)
ctm_class['NextPurchaseDayRange'] = 1  ## less than 3 months
ctm_class.loc[ctm_class.NextPurchaseDay>90,'NextPurchaseDayRange'] =
0 # more than 3 months
corr_matrix = ctm_class[ctm_class.columns].corr()
corr_df = pd.DataFrame(corr_matrix.min())
corr_df.columns = ['MinCorrelationCoeff']
corr_df['MaxCorrelationCoeff'] = corr_matrix[corr_matrix < 1].max()

plt.figure(figsize = (40, 30))
sns.heatmap(corr_matrix, annot = True, linewidths=0.2, fmt=".2f");
ctm_class = ctm_class.drop('NextPurchaseDay', axis=1)
X, y = ctm_class.drop('NextPurchaseDayRange', axis=1),
ctm_class.NextPurchaseDayRange
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=None, shuffle=True)


models = []
models.append(("LogisticRegression", LogisticRegression()))
models.append(("GaussianNB", GaussianNB()))
models.append(("RandomForestClassifier", RandomForestClassifier()))
models.append(("SVC", SVC()))
models.append(("DecisionTreeClassifier", DecisionTreeClassifier()))
models.append(("xgb.XGBClassifier",
xgb.XGBClassifier(eval_metric='mlogloss')))
models.append(("KNeighborsClassifier", KNeighborsClassifier()))
######################################################################
############
```

```
output_string = ""
for name, model in models:
    kfold = KFold(n_splits=2)
    cv_result = cross_val_score(model, X_train, y_train, cv=kfold,
scoring="accuracy")

    result_text = f"{name}: {', '.join([str(score) for score in
cv_result])}\n"
    output_string += result_text

st.title("Cross-validation Results")
st.text(output_string)


rf_model = RandomForestClassifier().fit(X_train, y_train)

print('Accuracy of Random Forest classifier on training set: {:.2f}'
       .format(rf_model.score(X_train, y_train)))
print('Accuracy of Random Forest classifier on test set: {:.2f}'
       .format(rf_model.score(X_test[X_train.columns], y_test)))
y_pred = rf_model.predict(X_test)
print(classification_report(y_test, y_pred))

############################################
train_accuracy = rf_model.score(X_train, y_train)
test_accuracy = rf_model.score(X_test[X_train.columns], y_test)


y_pred = rf_model.predict(X_test)

report = classification_report(y_test, y_pred)

st.title("Random Forest Classifier Metrics")
st.write(f"Accuracy on Training Set: {train_accuracy:.2f}")
st.write(f"Accuracy on Test Set: {test_accuracy:.2f}")
st.write("Classification Report:")
st.text(report)
#####################################
xgb_model = xgb.XGBClassifier().fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'
       .format(xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'
       .format(xgb_model.score(X_test[X_train.columns],  y_test)))

y_pred = xgb_model.predict(X_test)
print(classification_report(y_test, y_pred))

##################################################
```

```
train_accuracy = xgb_model.score(X_train, y_train)
test_accuracy = xgb_model.score(X_test[X_train.columns], y_test)

y_pred = xgb_model.predict(X_test)

report = classification_report(y_test, y_pred)


st.title("XGBoost Classifier Metrics")
st.write(f"Accuracy on Training Set: {train_accuracy:.2f}")
st.write(f"Accuracy on Test Set: {test_accuracy:.2f}")
st.write("Classification Report:")
st.text(report)
##################################################
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

parameter = {
    'max_depth': range(3, 10, 2),
    'min_child_weight': range(1, 5, 2)
}


cv_values = [2,3,4,5,6,7,8,9,10,15,20,]  # You can add more values if
needed

best_accuracy = 0
best_cv = None

for cv_val in cv_values:
    p_grid_search = GridSearchCV(
        estimator=xgb.XGBClassifier(eval_metric='mlogloss'),
        param_grid=parameter,
        scoring='accuracy',
        n_jobs=-1,
        cv=cv_val
    )

    p_grid_search.fit(X_train, y_train)


    best_params = p_grid_search.best_params_
    best_score = p_grid_search.best_score_

    if best_score > best_accuracy:
        best_accuracy = best_score
        best_cv = cv_val
```

```
print(f"The highest accuracy of {best_accuracy:.4f} is achieved with
cv={best_cv}.")

parameter = {
    'max_depth':range(3,10,2),
    'min_child_weight':range(1,5,2)
    }

p_grid_search = GridSearchCV(estimator =
xgb.XGBClassifier(eval_metric='mlogloss'),
                             param_grid = parameter,
                             scoring='accuracy',
                             n_jobs=-1,
                             #iid=False,
                             cv=6
                            )

p_grid_search.fit(X_train, y_train)

refined_xgb_model = xgb.XGBClassifier(eval_metric='logloss',
                                      max_depth=list(p_grid_search.be
st_params_.values())[0]-1,
                                      min_child_weight=list(p_grid_se
arch.best_params_.values())[-1]+4
                                     ).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set:
{:.2f}'.format(refined_xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set:
{:.2f}'.format(refined_xgb_model.score(X_test[X_train.columns],
y_test)))
ref_xgb_pred_y = refined_xgb_model.predict(X_test)
ref_xgb_pred_y = refined_xgb_model.predict(X)
################################################
refined_xgb_model1 = xgb.XGBClassifier(
    eval_metric='logloss',
    max_depth=list(p_grid_search.best_params_.values())[0] - 1,
    min_child_weight=list(p_grid_search.best_params_.values())[-1] +
4
).fit(X_train, y_train)


train_accuracy = refined_xgb_model1.score(X_train, y_train)
test_accuracy = refined_xgb_model1.score(X_test[X_train.columns],
y_test)


st.title("Refined XGBoost Classifier Metrics")
st.write(f"Accuracy on Training Set: {train_accuracy:.2f}")
```

```
st.write(f"Accuracy on Test Set: {test_accuracy:.2f}")
##################################################
ctm_class['predictions'] = ref_xgb_pred_y

ctm_class_final = ctm_class[['CustomerID', 'predictions']]

customer_country = data[['CustomerID', 'Country']]

ctm_class_final = pd.merge(ctm_class_final[['CustomerID',
'predictions']], customer_country, on='CustomerID', how='left')
ctm_class_final.drop_duplicates(subset='CustomerID', keep='last',
inplace=True)
# Reset the index starting from 1
ctm_class_final.reset_index(drop=True, inplace=True)
import streamlit as st
#####################################################################
#############################################
st.title("The Final Result:")
ctm_class_final

##########################
kmeans_model = KMeans(init='k-means++',  max_iter=400,
random_state=42)
kmeans_model.fit(ctm_dt[['Recency','Frequency','Revenue']])
def try_different_clusters(K, data):

    cluster_values = list(range(1, K+1))
    inertias=[]

    for c in cluster_values:
        model = KMeans(n_clusters = c,init='k-
means++',max_iter=400,random_state=42)
        model.fit(data)
        inertias.append(model.inertia_)

    return inertias
outputs = try_different_clusters(12,
ctm_dt[['Recency','Frequency','Revenue']])
distances = pd.DataFrame({"clusters": list(range(1, 13)),"sum of
squared distances": outputs})
figure = go.Figure()
figure.add_trace(go.Scatter(x=distances["clusters"], y=distances["sum
of squared distances"]))

figure.update_layout(xaxis = dict(tick0 = 1,dtick = 1,tickmode =
'linear'),
                  xaxis_title="Number of clusters",
                  yaxis_title="Sum of squared distances",
```

```
                  title_text="Finding optimal number of clusters
using elbow method")
figure.show()
ctm_dt['OS1']=ctm_dt['OverallScore'].div(2).round(3)
figure = px.scatter_3d(ctm_dt,
                    color='OS1',
                    x="Recency",
                    y="Frequency",
                    z="Revenue",
                    category_orders = {"clusters": ["0", "1", "2",
"3", "4"]}
                    )
figure.update_layout()
figure.show()


##
my_dict1={}
ctm_revenue = ctm_dt[['Revenue']]
for idx in range(1, 10):
    kmeans = KMeans(n_clusters=idx, max_iter=1000).fit(ctm_revenue)
    ctm_revenue["clusters"] = kmeans.labels_
    my_dict1[idx] = kmeans.inertia_

line_fig2 = px.line(x=list(my_dict1.keys()),
                   y=list(my_dict1.values()),
                   template="plotly_dark"
                  )

line_fig2.update_layout(title_x=0,
                       xaxis_title="Number of cluster",
                       yaxis_title="Revenue"
                      )

line_fig2.show(config={'displaylogo': False})


my_dict2={}
ctm_frequency = ctm_dt[['Frequency']]
for idx in range(1, 10):
    kmeans = KMeans(n_clusters=idx, max_iter=1000).fit(ctm_frequency)
    ctm_frequency["clusters"] = kmeans.labels_
    my_dict2[idx] = kmeans.inertia_

line_fig3 = px.line(x=list(my_dict2.keys()),
                   y=list(my_dict2.values()),
                   template="plotly_dark"
                  )
```

```
line_fig3.update_layout(title_x=0,
                        xaxis_title="Number of cluster",
                        yaxis_title="Frequency"
                       )

line_fig3.show(config={'displaylogo': False})
##############




st.sidebar.header("Select Plot")
plot_choice = st.sidebar.selectbox("Choose a graph", ("Pie Chart 1:
Customer_count vs Country",
                     "Line Chart: Revenue vs Date/Time","Pie Chart 2:
Country vs Revenue","Cluster - Recency","Cluster - Revenue","Cluster
- Frequency"))

st.header("Selected Plot")

if plot_choice == "Pie Chart 1: Customer_count vs Country ":
    st.plotly_chart(pie_fig)  # Display pie chart when selected
elif plot_choice == "Line Chart: Revenue vs Date/Time ":
    st.plotly_chart(line_fig)  # Display line chart when selected
elif plot_choice == "Pie Chart 2: Country vs Revenue":
    st.plotly_chart(pie_fig1)  # Display pie chart when selected

elif plot_choice == "Cluster - Recency":
    st.plotly_chart(line_fig1)  # Display pie chart when selected

elif plot_choice == "Cluster - Revenue":
    st.plotly_chart(line_fig2)  # Display pie chart when selected

elif plot_choice == "Cluster - Frequency":
    st.plotly_chart(line_fig3)  # Display pie chart when selected
```

```
!wget -q -O - ipv4.icanhazip.com
```

```
!streamlit run app.py & npx localtunnel --port 8501
```

11. REFERENCES

[1] Paranavithana, I. R., Rupasinghe, T., & Prior, D. D. (2022, September 28). *A Machine-Learnt Approach to Market Segmentation and Purchase Prediction Using Point-Of-Sale (POS) Data*. https://doi.org/10.1007/978-981-19-3579-4_4

[2] Customer's Purchase Prediction Using Customer Segmentation Approach for Clustering of Categorical Data. (2023, July 26). *Management and Production Engineering Review*. https://doi.org/10.24425/mper.2021.137678

[3] Aazmaan Ahmed Khan, R. A. (2023, September 11). Customer Segmentation using Machine Learning Techniques. *Tuijin Jishu/Journal of Propulsion Technology*, *44*(3), 2051–2061. https://doi.org/10.52783/tjjpt.v44.i3.653

[4] Zohdi, M., Rafiee, M., Kayvanfar, V., & Salamiraad, A. (2022, February 12). Demand forecasting based machine learning algorithms on customer information: an applied approach. *International Journal of Information Technology*, *14*(4), 1937–1947. https://doi.org/10.1007/s41870-022-00875-3

Web References:

1. Streamlit and ngrok: https://www.youtube.com/watch?v=NEhrkeF2o_M

2. Streamlit documentation: https://docs.streamlit.io/

3. Kaggle dataset: https://www.kaggle.com/datasets/lakshmi25npathi/online-retail-dataset

4. K-Means for segmentation: https://neptune.ai/blog/customer-segmentation-using-machine-learning

5. XGboost and other machine learning algorithms:
https://www.kaggle.com/code/nageshsingh/predict-customers-probable-purchase