Using Machine Learning to Predict Customers' Next Purchase Day

Machine Learning model to predict whether customers will make their next purchase after a certain period.

Evans Doe Ocansey Apr 30 · 12 min read ★

Image by Mediamodifier and can be accessed here.

Introduction

If there is one major lesson that those in the retail business have learnt from the SARS-CoV-2 pandemic, it is the demand to switch to doing business via the Internet, i.e., e-commerce. The idea of e-commerce assists those in managerial positions to make decisions for the progress of their companies. Undoubtedly, most of these decisions are influenced by the results derived from studying the purchasing behavioural data of online customers by experts in data analysis, data science, and machine learning.

Suppose the managerial team of an online retail shop approaches you, a data scientist, with the dataset wanting to know whether customers will make their next purchase 90 days from the day they made their last purchase. Your answer to their inquiry will help them identify which customers their marketing team need to have a focus on with regard to the next promotional offers they will be rolling out.

In this article, my goal as a data scientist is to build a model that will provide a suitable answer to the question posed by the firm's managers. More precisely, using the given dataset, I build a machine learning model that predicts whether an online customer of a retail shop will make their next purchase 90 days from the day they made their last purchase.

It is worth mentioning that Barış Karaman[1] has done a similar work that answers a different question and not this exact forecast the question we have, seeks to preempt.

Some Information about the Dataset

Before proceeding to answer the question of interest, I will first present some general and useful information about the dataset.

The dataset recorded 5942 online customers from 43 different countries. Among the online customers of the retail shop, 90.8% of them were living in the United Kingdom.

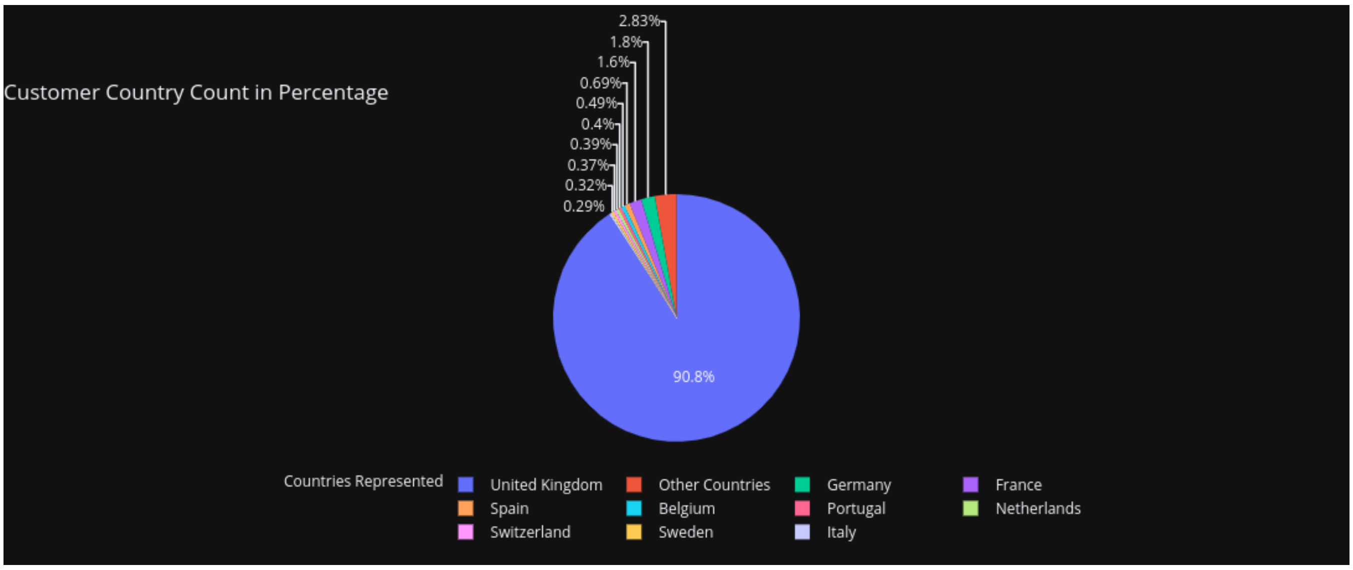


Figure 1: Customer Country Count in Percentage — Image by Author

With such a huge customer base in the United Kingdom, it is not surprising that 83% of the company's revenue came from the United Kingdom.

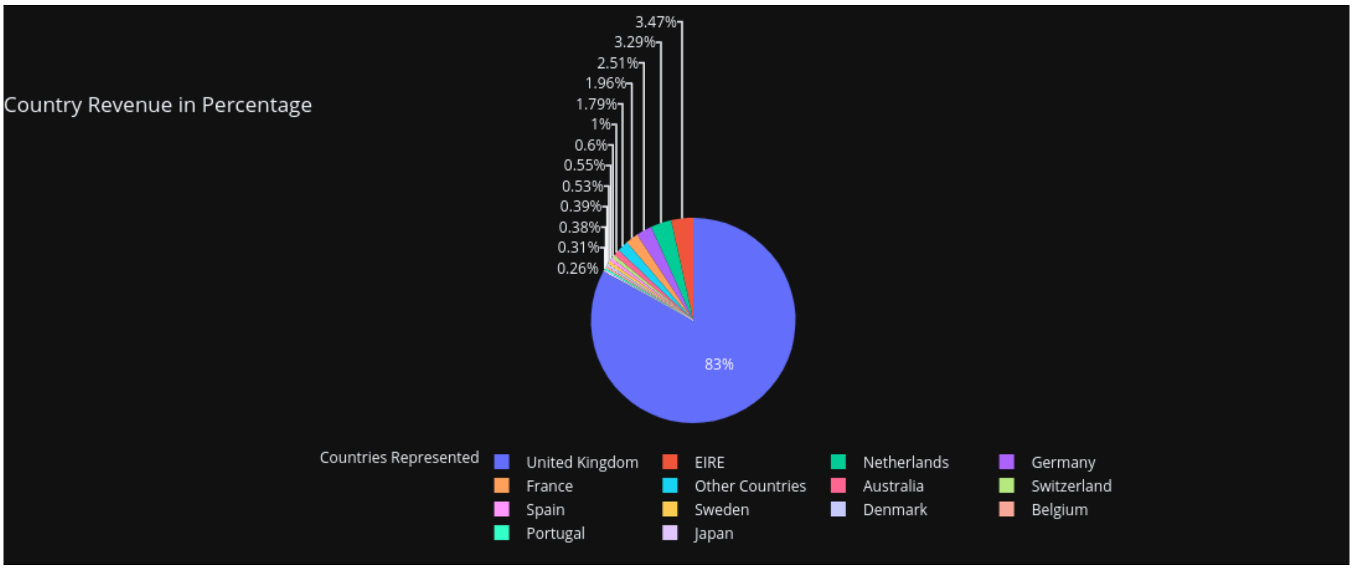


Figure 2: Revenue of Countries in Percentage — Image by Author

Figure 3 below gives a visual representation of the monthly revenue earned by the online retail company.

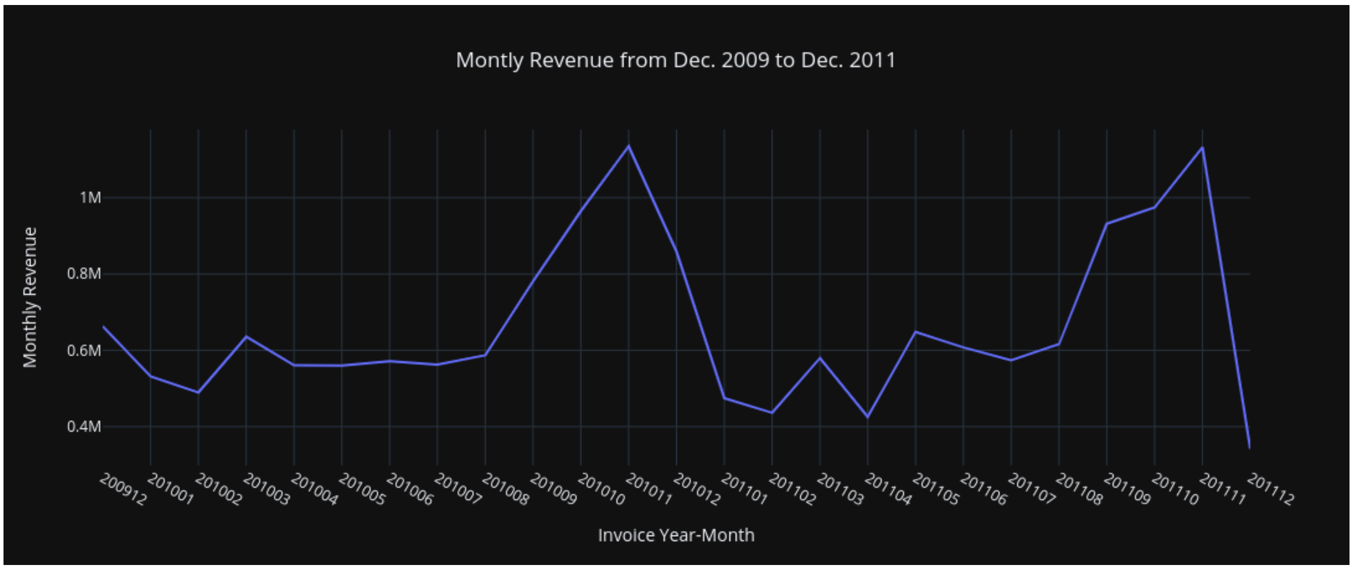


Figure 3: Monthly Revenue from December 2009 until December 2011 — Image by Author.

Here, one can observe that the company recorded its highest revenue in the month of November 2010, followed by November 2011. In addition, there is a rise in monthly revenue after August.

From the analysis made in this section, there is the advice one can give to the managers for consideration. In the company's bid to increase its customer base in other countries than the United Kingdom, what could be a possible advice a data scientist can suggest to the managerial team? In an answer to this, I would say that…

Since the company has a solid customer base in the United Kingdom, it could capitalise on that and roll out a "win-win promotion". In implementing this rollout, specifically, for any product that an existing customer buys, he/she gets the opportunity to invite a new customer outside the United Kingdom via a web link. If the new customer buys something from the online shop using the web link he/she received from the already existing customer, the company gives both the existing customer and the new customer a cash voucher that can be used by both parties in their next purchase. By so doing we see that the company, the existing or earlier customer, and the new customer all receive a level of satisfaction in the transaction made.

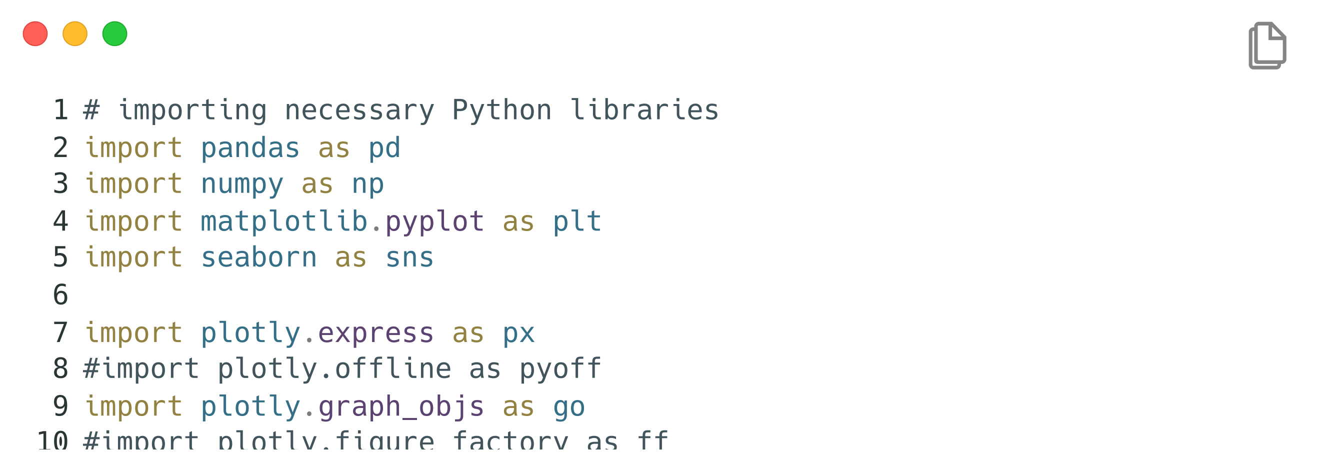
Predicting Customers' Next Purchase

In this section, I focus on the methods that I deployed to solve the problem of interest. That is, *to build a machine learning model that will predict whether an online customer of a retail shop will make their next purchase 90 days from the day they made their last purchase.*

The major steps included the following:

- **Data Wrangling**
- **Feature Engineering**
- **Building Machine Learning Models**
- **Selecting Model**

I begin by importing the necessary Python packages, download the dataset and then load it into my Python environment. The code snippet below summarises this step.

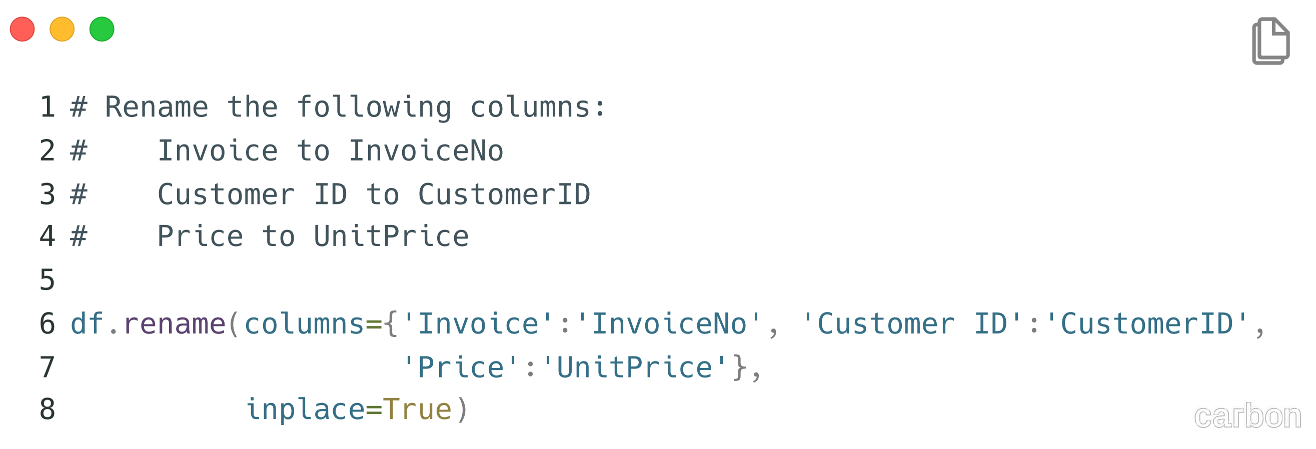


```
# importing necessary Python libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import plotly.express as px
#import plotly.offline as pyoff
import plotly.graph_objs as go
#import plotly.figure_factory as ff
```

```

# avoid displaying warnings
import warnings
warnings.filterwarnings("ignore")
```

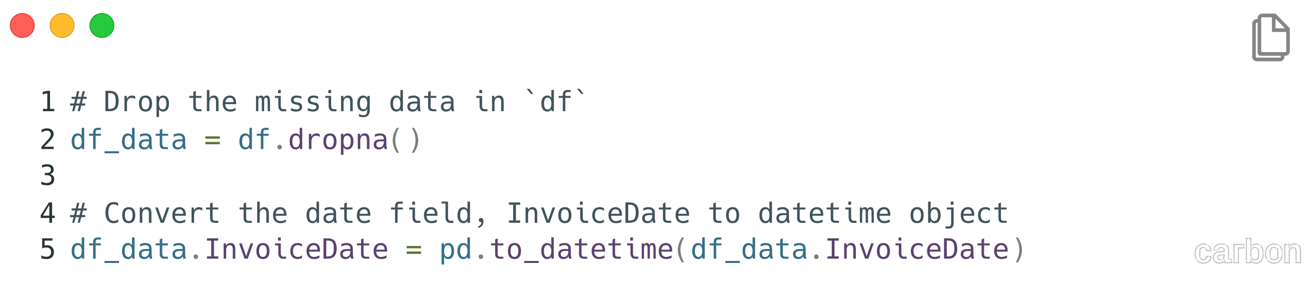
Data Wrangling

I then wrangle with the dataset to put it into good shape so as to introduce new `X` features.



```
# Rename the following columns:
#    Invoice to InvoiceNo
#    Customer ID to CustomerID
#    Price to UnitPrice

df.rename(columns={'Invoice':'InvoiceNo', 'Customer ID':'CustomerID',
                   'Price':'UnitPrice'},
          inplace=True)
```

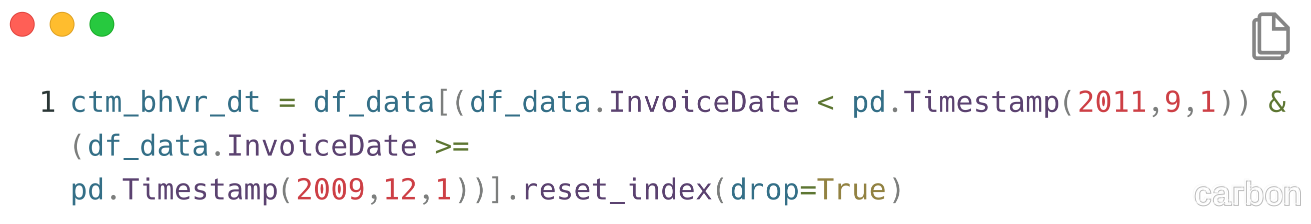
The `CustomerID` column of the given dataset has `243007` missing data. That represents `22.77%` of the entire online customers. Moreover, the `Description` column has `4382` missing data. How do I deal with these missing data? After talking to the company leaders, they suggested that any item that has missing `CustomerID` should be dropped.



```
# Drop the missing data in `df`
df_data = df.dropna()

# Convert the date field, InvoiceDate to datetime object
df_data.InvoiceDate = pd.to_datetime(df_data.InvoiceDate)
```

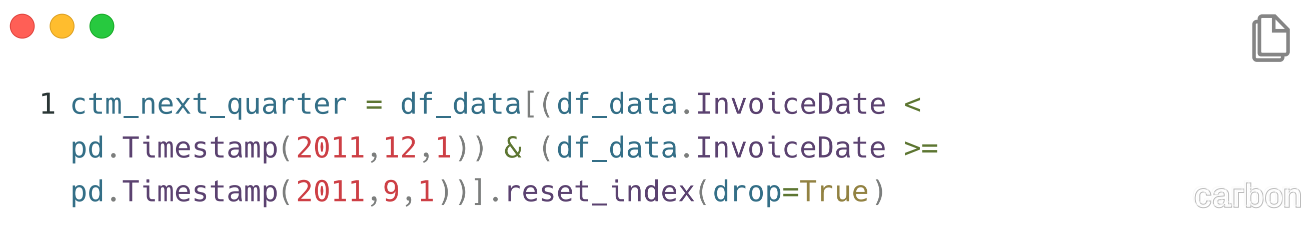
The dataframe `df_data` is split into two pandas data frames. Namely,

- The first sub-dataframe, `ctm_bhvr_dt`, contains purchases made by customers from 01–12–2009 to 30–08–2011. From this dataset, I get the last purchase date of all online customers.



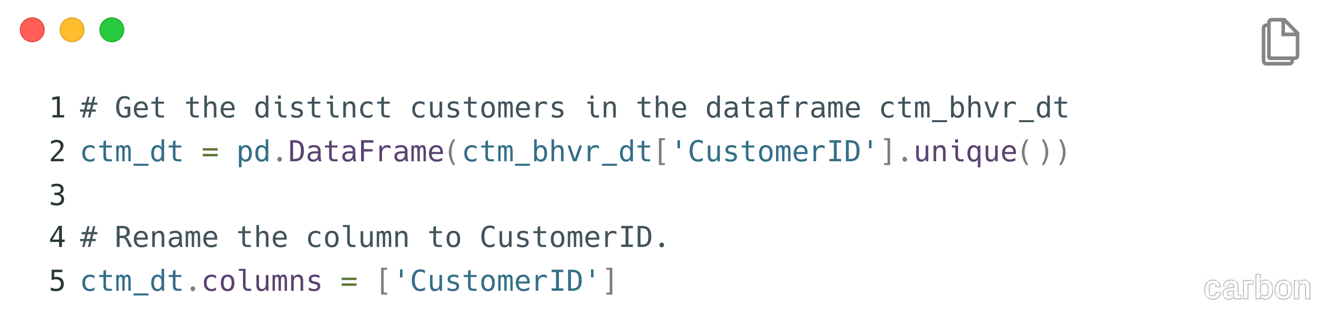
```
ctm_bhvr_dt = df_data[(df_data.InvoiceDate < pd.Timestamp(2011,9,1)) &
(df_data.InvoiceDate >=
pd.Timestamp(2009,12,1))].reset_index(drop=True)
```

- The second sub-dataframe `ctm_next_quarter` is used to get the first purchase date of the customers from 01–09–2011 to 30–11–2011.



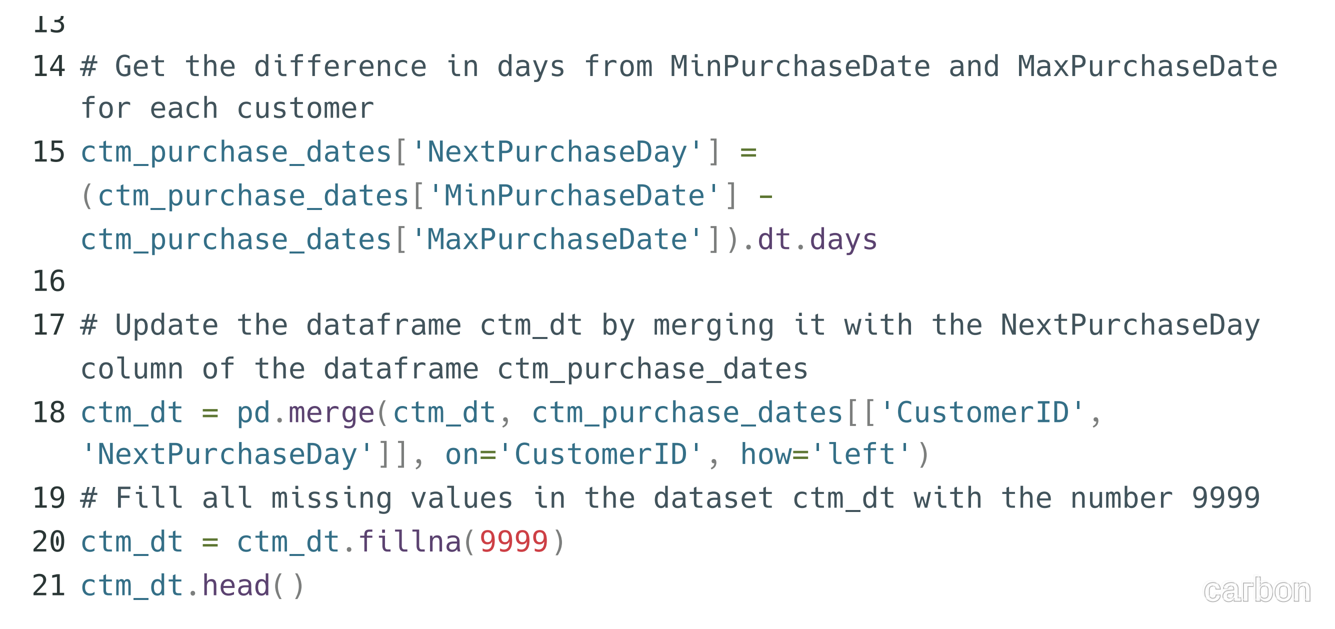
```
ctm_next_quarter = df_data[(df_data.InvoiceDate <
pd.Timestamp(2011,12,1)) & (df_data.InvoiceDate >=
pd.Timestamp(2011,9,1))].reset_index(drop=True)
```

Next, I create a pandas dataframe that contains a set of features of each customer for us to build our prediction model. I begin by creating a dataset that contains the distinct customers in the dataframe `ctm_bhvr_dt`.



```python
# Get the distinct customers in the dataframe ctm_bhvr_dt
ctm_dt = pd.DataFrame(ctm_bhvr_dt['CustomerID'].unique())

# Rename the column to CustomerID.
ctm_dt.columns = ['CustomerID']
```

I then add a new label, **`NextPurchaseDay`** to the dataframe `ctm_dt`. This new label will be the number of days between the last purchase date of a customer in the dataframe the customer who has the most frequently purchased item that is with missing CustomerID the following procedure to deal with the missing values in the CustomerID column. turns out that the `ctm_bhvr_dt` and his/her first purchase date in the dataframe `ctm_next_quarter`.



```python
# Get the difference in days from MinPurchaseDate and MaxPurchaseDate for each customer
ctm_purchase_dates['NextPurchaseDay'] = (ctm_purchase_dates['MinPurchaseDate'] - ctm_purchase_dates['MaxPurchaseDate']).dt.days

# Update the dataframe ctm_dt by merging it with the NextPurchaseDay column of the dataframe ctm_purchase_dates
ctm_dt = pd.merge(ctm_dt, ctm_purchase_dates[['CustomerID', 'NextPurchaseDay']], on='CustomerID', how='left')
# Fill all missing values in the dataset ctm_dt with the number 9999
ctm_dt = ctm_dt.fillna(9999)
ctm_dt.head()
```

Figure 5 below is the output of the code snippet above. It shows the first 5 entries of the dataframe object, `ctm_dt`.

Search this file…

	CustomerID	NextPurchaseDay
0	13085.0	9999.0

1	13078.0	13.0
2	15362.0	9999.0
3	18102.0	27.0
4	12682.0	15.0

ctm_dt_head.csv hosted with ❤ by **GitHub** view raw

Figure 5: Customers' next purchase day data

In the next section, I introduce some features and add them to the dataframe `ctm_dt` to build our machine learning model.

Feature Engineering

I introduce features into our dataframe `ctm_dt` that segments customers into groups based on their value to the company. In executing this I use the RFM segmentation method. RFM stands for

- **Recency**: indicating how recent a customer made a purchase.
- **Frequency**: How often or the number of times a customer purchases.
- **Monetary Value/Revenue**: The amount of money a customer spends when making a purchase at a point in time.

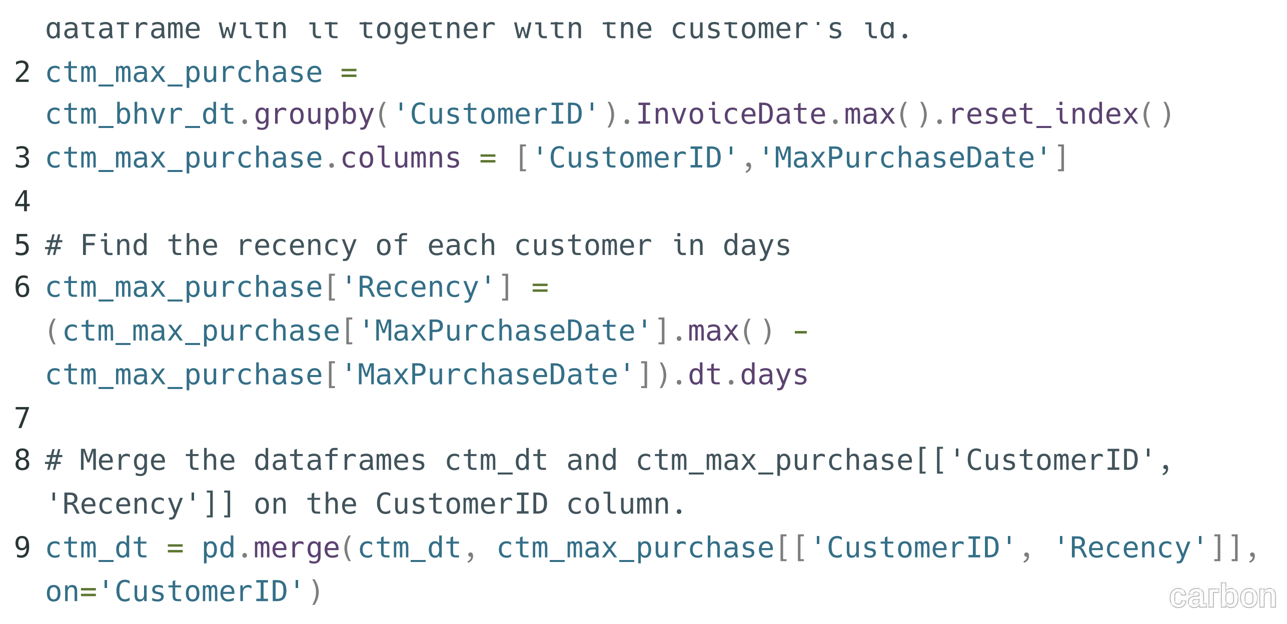
Using these three features being recency, frequency, and monetary value/revenue, I create an RFM score system to group the customers. Essentially, the RFM score derived is what helps to give an insight into what a customer would probably do regarding purchase decisions in the future.

After calculating the RFM score, I then apply *unsupervised machine learning* to identify different groups (clusters) for each score and add them to the dataframe `ct_dt` . Finally, I apply the pandas dataframe method `get_dummies` to `ctm_dt` to deal with the categorical features in the dataframe. I now make a move into coding to fish out the computation of the RFM scores and the clustering.

Recency

In getting to know who is likely to make a current purchase, I use the recency feature to work this out. Factoring in the length of time a customer has taken off after his or her last purchase, the recency characteristic comes in handy here. I use this feature to know which customer will be coming in for a transaction. It is relevant to also note that the sales transaction of a recent purchasing customer is of far more worth than the customer who has not bought in a while.

Let us get into the coding here below.



```
dataframe with it together with the customer's id.
ctm_max_purchase = ctm_bhvr_dt.groupby('CustomerID').InvoiceDate.max().reset_index()
ctm_max_purchase.columns = ['CustomerID','MaxPurchaseDate']

# Find the recency of each customer in days
ctm_max_purchase['Recency'] = (ctm_max_purchase['MaxPurchaseDate'].max() - ctm_max_purchase['MaxPurchaseDate']).dt.days

# Merge the dataframes ctm_dt and ctm_max_purchase[['CustomerID', 'Recency']] on the CustomerID column.
ctm_dt = pd.merge(ctm_dt, ctm_max_purchase[['CustomerID', 'Recency']], on='CustomerID')
```

Figure 6 below gives a visual presentation of the recency data of the online customers.

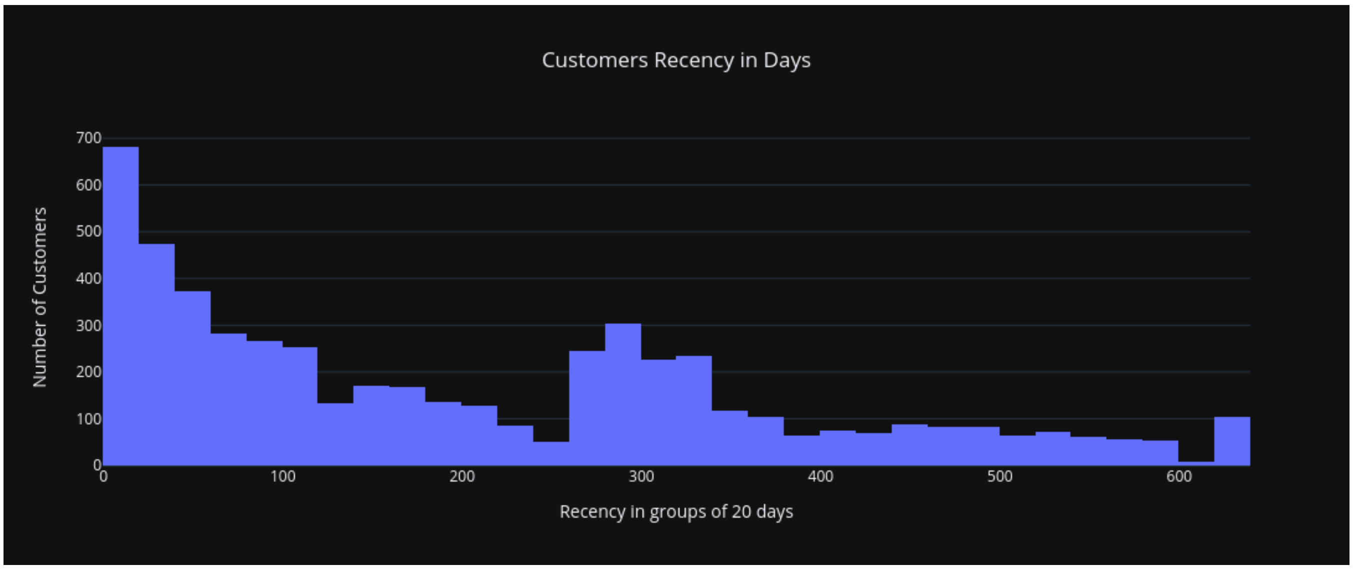
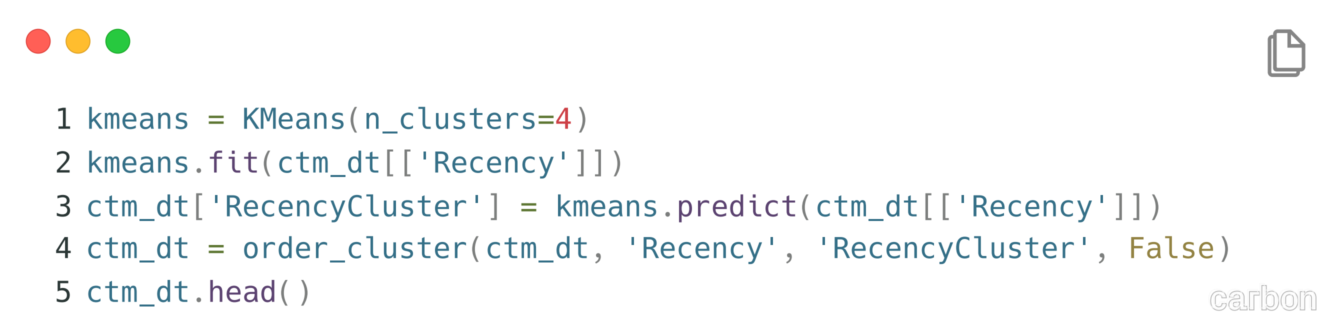


Figure 6: Histogram of the recency data of customers — Image by Author.

The code used to generate Figure 6 above can be accessed in the Jupyter notebook here.

Next, I need to assign a recency score for the recency values. This can be achieved by applying the *K-means clustering algorithm*. However, we need to know the number of clusters before using the algorithm. Applying the *Elbow Method*, one can determine the number of clusters needed for a given data. In our case, given the recency values as our data, the number of clusters computed is 4. The code used to compute the number of clusters is available in the Jupyter notebook here.

I can now build 4-clusters using the **`Recency`** column in the dataframe `ctm_dt` and create a new column **`RecencyCluster`** in `ctm_dt` whose values are the cluster value predicted by the unsupervised machine learning algorithm `kmeans`. Using the user-defined Python function `order_cluster` accessible here, I sort the dataframe `ctm_dt` in decreasing order of the values in **`RecencyCluster`**. The code snippet below outputs Figure 7 below.



```python
kmeans = KMeans(n_clusters=4)
kmeans.fit(ctm_dt[['Recency']])
ctm_dt['RecencyCluster'] = kmeans.predict(ctm_dt[['Recency']])
ctm_dt = order_cluster(ctm_dt, 'Recency', 'RecencyCluster', False)
ctm_dt.head()
```

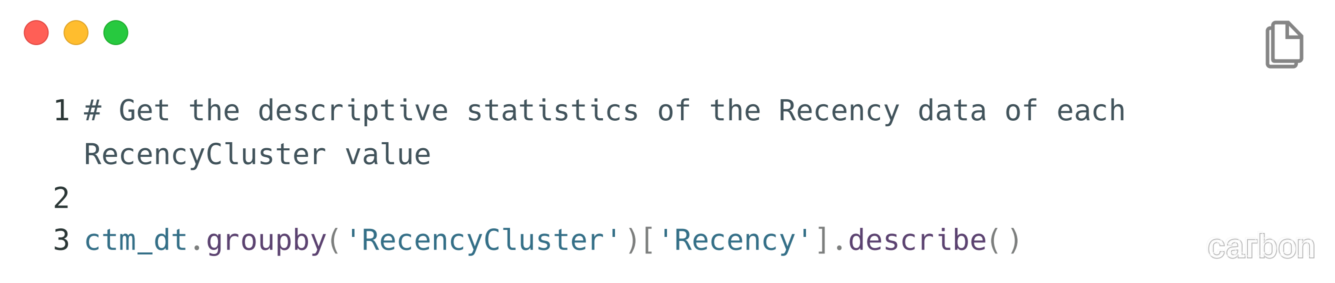
Search this file…

	CustomerID	NextPurchaseDay	Recency	RecencyCluster
0	13085.0	9999.0	57	1
1	13078.0	13.0	0	1
2	15362.0	9999.0	348	0
3	18102.0	27.0	26	1
4	12682.0	15.0	0	1

ctm_dt_recency.csv hosted with ❤ by GitHub view raw

Figure 7: Customer Recency Cluster Data

Let us group the dataframe `ctm_dt` by the cluster values in the column labelled **`RecencyCluster`** and fetch out the statistical description of the **`Recency`** data of each of these clusters



```python
# Get the descriptive statistics of the Recency data of each
  RecencyCluster value

ctm_dt.groupby('RecencyCluster')['Recency'].describe()
```

Search this file…

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	750.0	520.509333	65.218473	418.0	465.0	517.0	571.5	638.0
1	1427.0	314.360196	41.129298	236.0	281.5	308.0	338.0	416.0
2	1121.0	154.556646	39.686997	96.0	117.0	152.0	185.0	234.0
3	2016.0	37.407738	27.264651	0.0	13.0	33.0	58.0	94.0

recency_cluster_stats.csv hosted with ❤ by GitHub view raw

Figure 8: Statistical Summary of Recency Data against RecencyCluster

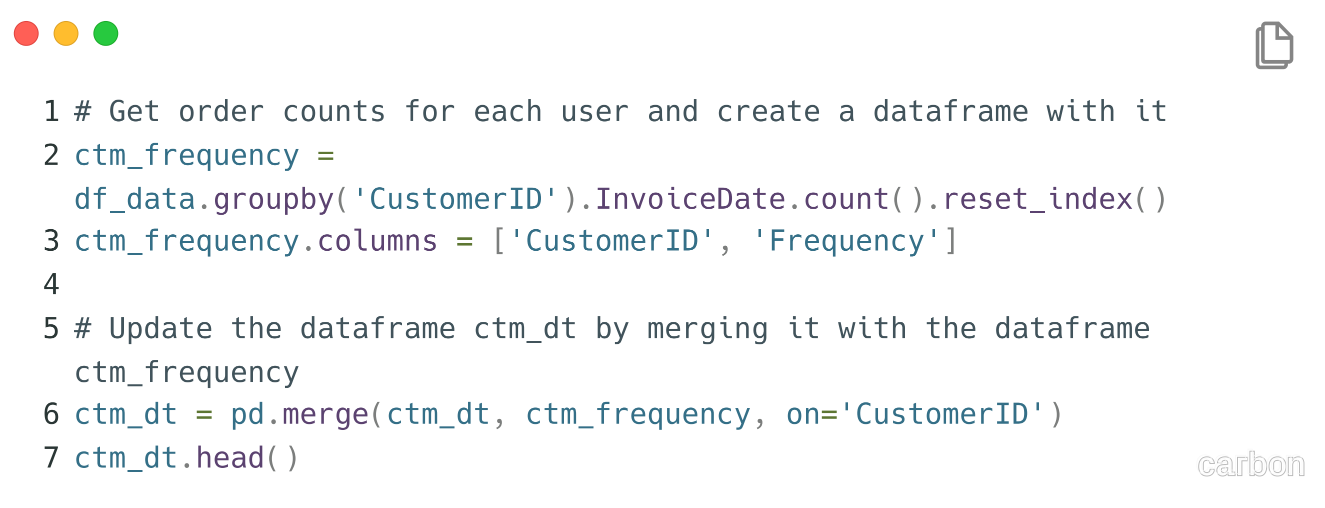
From Figure 8 above, it can be observed that cluster value 3 covers the most recent customers whereas 0 has the most inactive customers.

In the next subsections, I apply the method we have discussed in this subsection for the Frequency and Revenue features.

Frequency

So as earlier mentioned, in a particular frame of time, if we consider the number of times a customer has engaged in a purchasing transaction, frequency comes into play. Now, this frequency characteristic is that which helps us know a customers alliance to a specific company or trading brand. In view of this, it gives the company insight into the marketing strategies to relay and at what points in time, in order to reach out to such customers in particular.

Here, I conduct a similar procedure of analysis as I did in the previous subsection (**Recency**).



```python
# Get order counts for each user and create a dataframe with it
ctm_frequency = df_data.groupby('CustomerID').InvoiceDate.count().reset_index()
ctm_frequency.columns = ['CustomerID', 'Frequency']

# Update the dataframe ctm_dt by merging it with the dataframe ctm_frequency
ctm_dt = pd.merge(ctm_dt, ctm_frequency, on='CustomerID')
ctm_dt.head()
```

Search this file…

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency
0	13085.0	9999.0	57	3	92
1	13078.0	13.0	0	3	855
2	18102.0	27.0	26	3	1068
3	12682.0	15.0	0	3	1042
4	18087.0	46.0	44	3	95

ctm_dt_frequency.csv hosted with ❤ by GitHub view raw

Figure 9: First 5 entries of the main dataset with Frequency

Figure 10 below illustrates the histogram of customers whose purchase frequency is less than 1200.

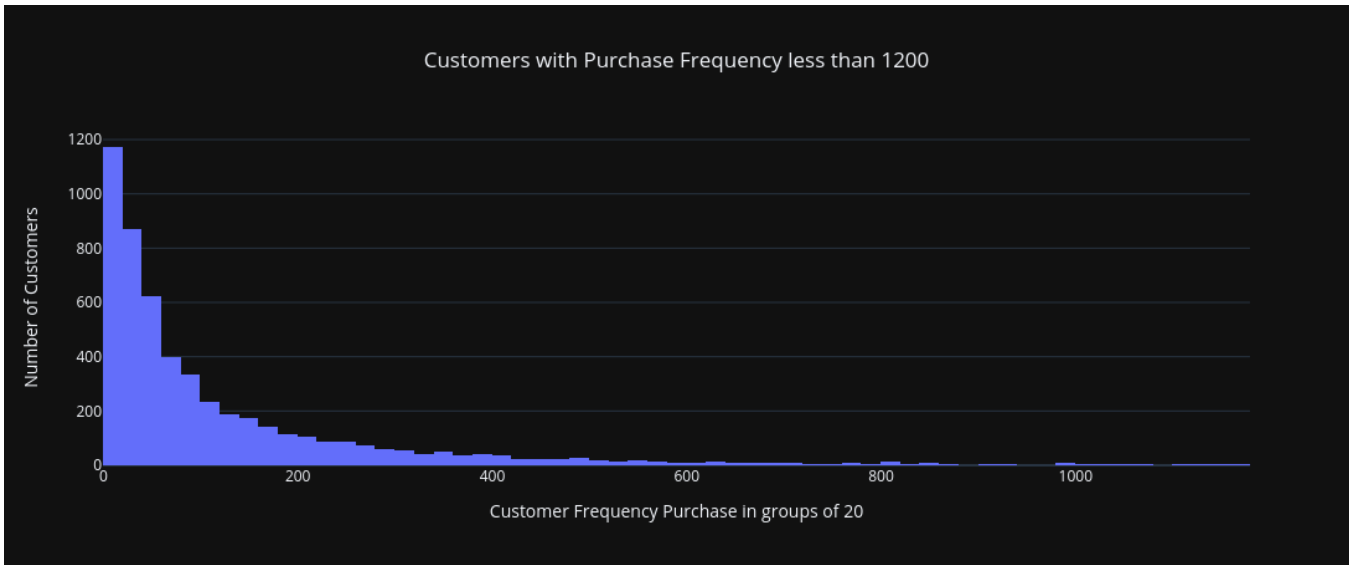
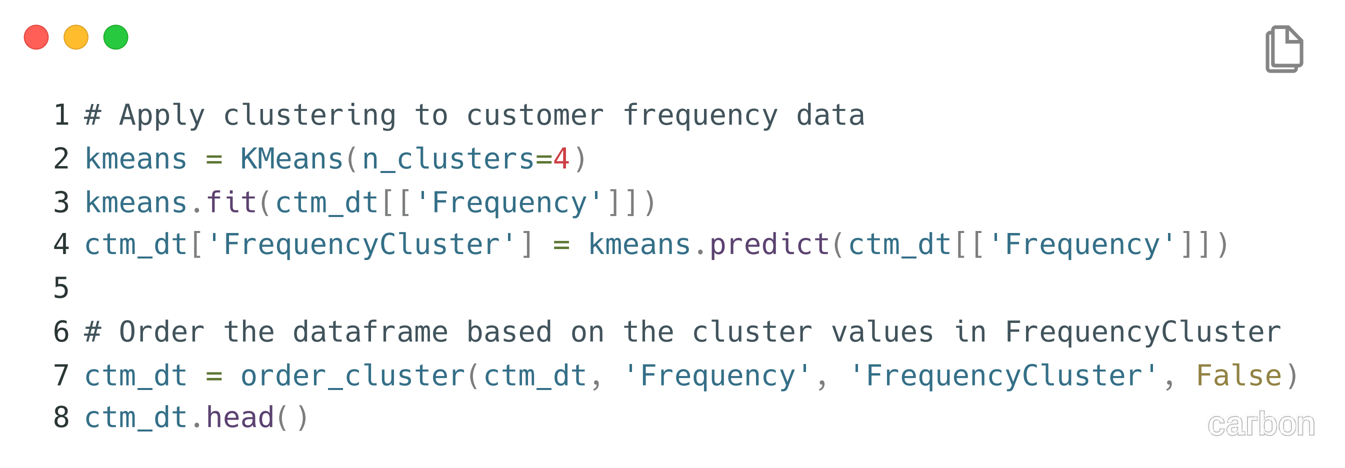


Figure 10: Histogram of customers with purchase frequency less than 1200 — Image by Author.

The code snippet below assigns a cluster value for the purchase frequency of each customer and sorts the cluster values in decreasing order.



```python
# Apply clustering to customer frequency data
kmeans = KMeans(n_clusters=4)
kmeans.fit(ctm_dt[['Frequency']])
ctm_dt['FrequencyCluster'] = kmeans.predict(ctm_dt[['Frequency']])

# Order the dataframe based on the cluster values in FrequencyCluster
ctm_dt = order_cluster(ctm_dt, 'Frequency', 'FrequencyCluster', False)
ctm_dt.head()
```

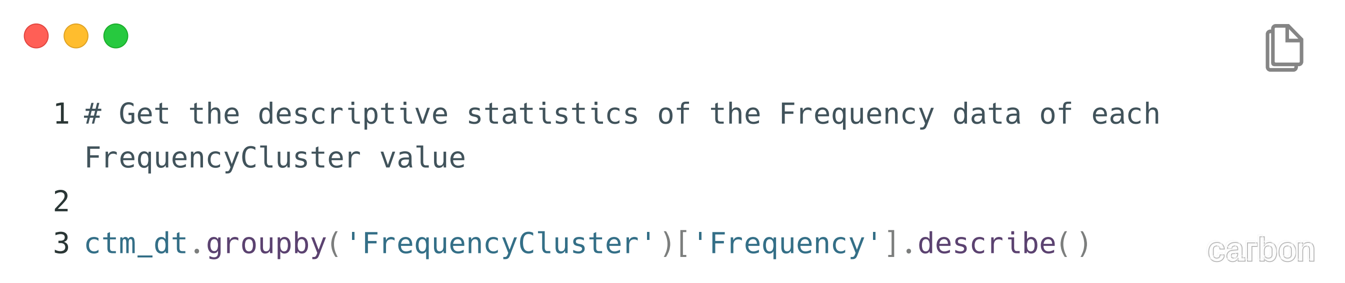
Search this file…

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster
0	13085.0	9999.0	57	3	92	3
1	18087.0	46.0	44	3	95	3
2	17519.0	116.0	33	3	224	3
3	12362.0	40.0	12	3	275	3
4	15712.0	38.0	9	3	167	3

frequency_cluster.csv hosted with ❤ by GitHub view raw

Figure 11: First five entries of the main dataset with FrequencyCluster

The code snippet below groups the dataframe `ctm_dt` by the cluster values recorded in the column labelled **`FrequencyCluster`** and fetches out the statistical description of the **`Frequency`** data of each of these **`FrequencyCluster`** values.



```python
# Get the descriptive statistics of the Frequency data of each
FrequencyCluster value

ctm_dt.groupby('FrequencyCluster')['Frequency'].describe()
```

Search this file…

FrequencyCluster	count	mean	std	min	25%	50%	75%	max
0	2.0	12355.00	1049.35	11613.00	11984.00	12355.00	12726.00	13097.00
1	14.0	3921.00	1541.06	2430.00	2829.50	3346.50	4570.25	7307.00
2	404.0	716.73	318.76	403.00	482.75	604.00	844.50	2134.00
3	4894.0	86.70	91.18	1.0	20.00	51.00	123.00	402.00

frequency_cluster_stats.csv hosted with ❤ by GitHub view raw

Figure 12: Statistical Summary of Frequency Data against FrequencyCluster

As it was for the case of the Recency, customers with a higher frequency cluster value are better customers. In other words, they patronise the products of the retail shop very often than those with a lower frequency cluster value.

Monetary Value/Revenue

To give a little more detail to Monetary Value or revenue, it centres more on the money a customer spends when in for a purchase at any point in time. So here it helps to ascertain how much money a customer is likely to let out when making a purchase. Even though this feature of revenue does not expose one to predict when next there will be a purchase from the customer, knowing how much could come in when the customer comes through for a transaction is worth knowing.

I again follow a similar procedure to obtain a revenue score for each customer and assign cluster values for each customer based on their revenue score.

```python
# Create a new label, Revenue of each item bought
df_data['Revenue'] = df_data.UnitPrice * df_data.Quantity

# Get the revenue from each customer and sum them.
ctm_revenue =
  df_data.groupby('CustomerID').Revenue.sum().reset_index()

# Merge the dataframe ctm_revenue with ctm_dt
ctm_dt = pd.merge(ctm_dt, ctm_revenue, on='CustomerID')
```

```python
ctm_dt.head()
```

carbon

Search this file…

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue
0	13085.0	9999.0	57	3	92	3	1459.46
1	18087.0	46.0	44	3	95	3	14411.62
2	17519.0	116.0	33	3	224	3	5102.80
3	12362.0	40.0	12	3	275	3	5284.58
4	15712.0	38.0	9	3	167	3	3467.46

ctm_dt_revenue.csv hosted with ❤ by GitHub view raw

Figure 13: First five entries of the main dataset with Revenue

The figure below illustrates a visual representation of customers whose revenue is below £10,000.

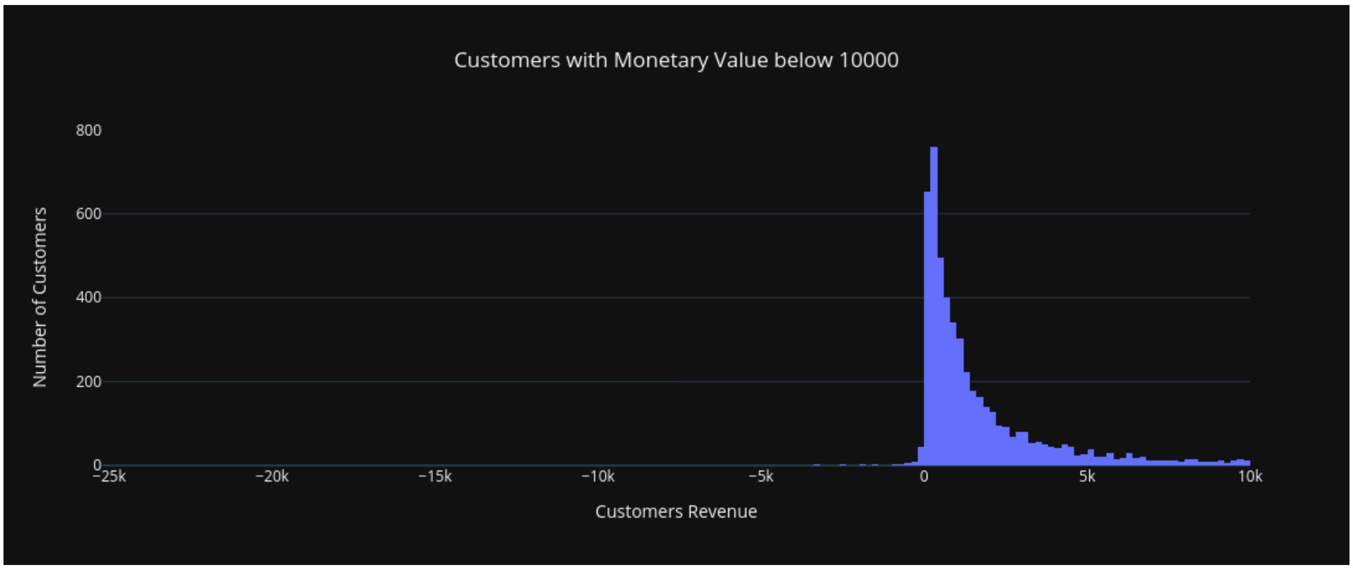


Figure 14: Histogram of customers with a monetary value below £10000 — Image by Author.

The code snippet below assigns a cluster value for the revenue of each customer and sorts the cluster values in ascending order.

```python
# Apply clustering to customer revenue data
kmeans = KMeans(n_clusters=number_of_clusters)
kmeans.fit(ctm_dt[['Revenue']])
ctm_dt['RevenueCluster'] = kmeans.predict(ctm_dt[['Revenue']])

# Order the dataframe based on the cluster values in RevenueCluster
```

```python
ctm_dt = order_cluster(ctm_dt, 'Revenue', 'RevenueCluster', True)

# Group the dataframe ctm_dt by RevenueCluster and get the statistical
description of the Revenue data of each of cluster value
ctm_dt.groupby('RevenueCluster')['Revenue'].describe()
```

carbon

Search this file…

RevenueCluster	count	mean	std	min	25%	50%	75%	max
0	5239.0	2029.77	3092.14	-25111.09	344.67	924.36	2327.09	23090.4
1	65.0	44586.32	22598.24	23350.74	26960.02	35866.63	55703.13	113513.0
2	8.0	198033.39	62010.60	136391.48	142827.53	181355.75	242746.68	296564.
3	2.0	560778.65	52943.31	523342.07	542060.36	560778.65	579496.93	598215.2

revenue_cluster.csv hosted with ❤ by **GitHub** view raw

Figure 15: Statistical Summary of Revenue Data against RevenueCluster

Overall Score

In the code snippet below, I add a new column **`OverallScore`** to the dataframe `ctm_dt` with values as the sum of the cluster values obtained for the Recency, Frequency and Revenue.

```python
# Calculate overall score
ctm_dt['OverallScore'] = ctm_dt['RecencyCluster'] +
ctm_dt['FrequencyCluster'] + ctm_dt['RevenueCluster']

# Get mean of the components of OverallScore
ctm_dt.groupby('OverallScore')['Recency', 'Frequency',
'Revenue'].mean()
```

carbon

Search this file…

OverallScore	Recency	Frequency	Revenue
3	514.61	39.93	435.11
4	309.11	94.88	1203.62
5	126.55	259.68	3779.93

6	39.60	148.76	3586.26
7	15.00	614.71	95521.63
8	26.00	1068.00	598215.22

overall_score.csv hosted with ❤ by GitHub view raw

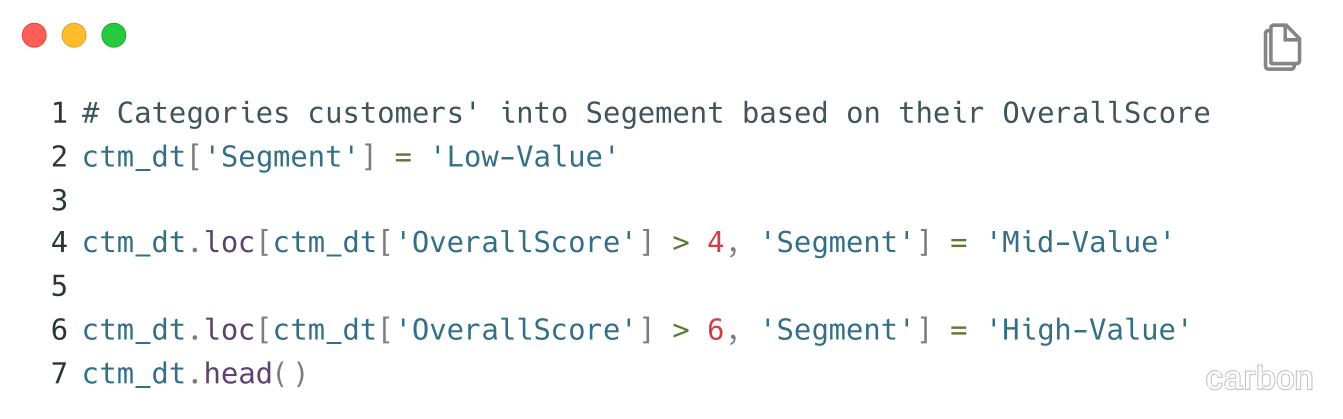
Figure 16: Mean of the Recency, Frequency and Revenue grouped against OverallScore value

The scoring above clearly shows us that customers with an overall score of 8 are the positively outstanding customers who bring much value to the company whereas those assigned a score of 3 are supposedly unreliable and merely wandering.

As a follow-up, I group the customers into the segments based on their overall score as follows:

- 3 to 4: Low Value
- 5 to 6: Mid Value
- 7 to 8: High Value

The code snippet is as follows:



```python
# Categories customers' into Segement based on their OverallScore
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_dt.head()
```

Search this file…

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	R
0	13085.0	9999.0	57	3	92	3	1459.46	0
1	18087.0	46.0	44	3	95	3	14411.62	0
2	17519.0	116.0	33	3	224	3	5102.80	0
3	12362.0	40.0	12	3	275	3	5284.58	0
4	15712.0	38.0	9	3	167	3	2467.46	0

4	15712.0	38.0	9	3	167	3	3467.46	0

segments.csv hosted with ❤ by GitHub view raw

Figure 17: First five entries of the main dataset with Segment column

I then create a copy of the dataset `ctm_dt` and apply the method `get_dummies` to it so as to convert all categorical column **`Segment`** to indicator variables.

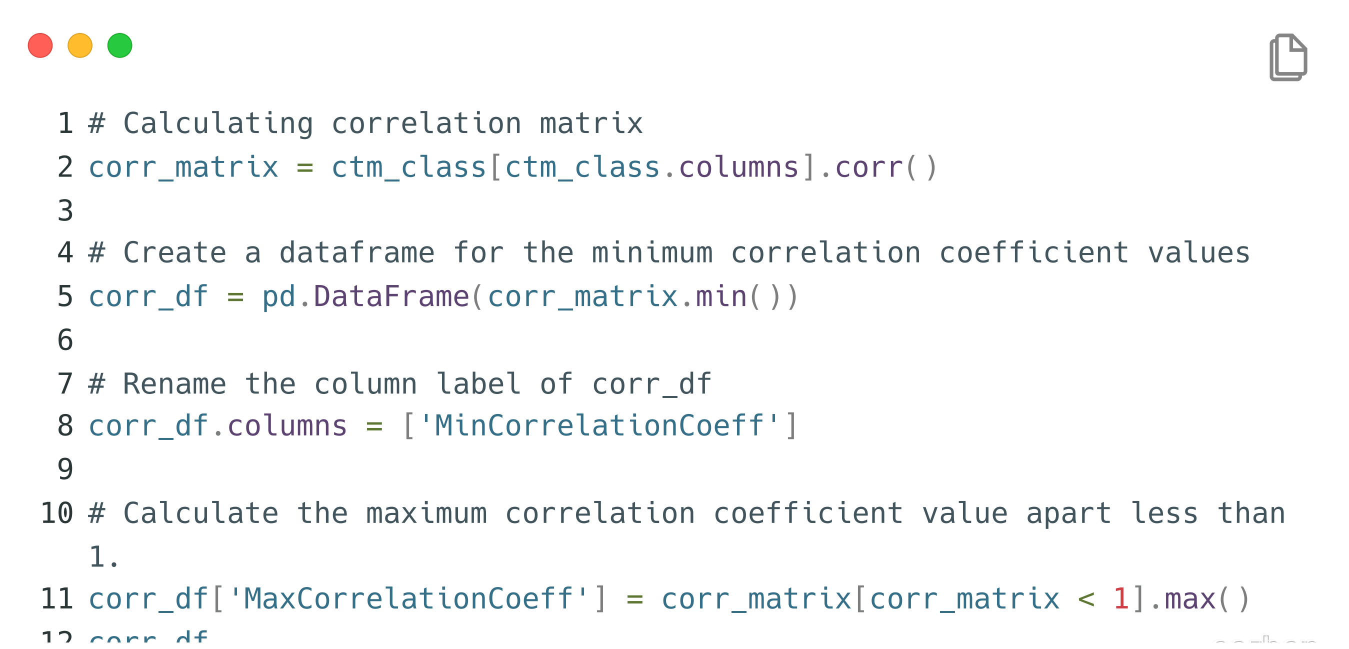
```
# Create a copy the dataframe ctm_dt
ctm_class = ctm_dt.copy()

# Convert categorical data to numerical data using get_dummies
ctm_class = pd.get_dummies(ctm_class)
```

In pursuance of my goal to estimate whether a customer will make a purchase in the next quarter, I create a new column **`NextPurchaseDayRange`** with values as either 1 or 0 defined as follows:

- If the value is 1, then it indicates that the customer will buy something in the next quarter, i.e., 90 days from his or her last purchase.
- The value 0 indicates that the customer will buy something in more than 90 days from his or her last purchase.

```
# Set the column 'NextPurchaseDayRange' to 1. Note that this
# captures customers with NextPurchaseDay value is less than 90 days

ctm_class['NextPurchaseDayRange'] = 1

# Set the value of NextPurchaseDayRange to 0 for customers whose
  NextPurchaseDay is more than 90 days.
ctm_class.loc[ctm_class.NextPurchaseDay>90, 'NextPurchaseDayRange'] =
0
```

I conclude this section by computing the correlation between our features and label. I achieve this by applying the `corr` method to the dataframe `ctm_class`.



```python
# Calculating correlation matrix
corr_matrix = ctm_class[ctm_class.columns].corr()

# Create a dataframe for the minimum correlation coefficient values
corr_df = pd.DataFrame(corr_matrix.min())

# Rename the column label of corr_df
corr_df.columns = ['MinCorrelationCoeff']

# Calculate the maximum correlation coefficient value apart less than 1.
corr_df['MaxCorrelationCoeff'] = corr_matrix[corr_matrix < 1].max()
corr_df
```

Search this file…

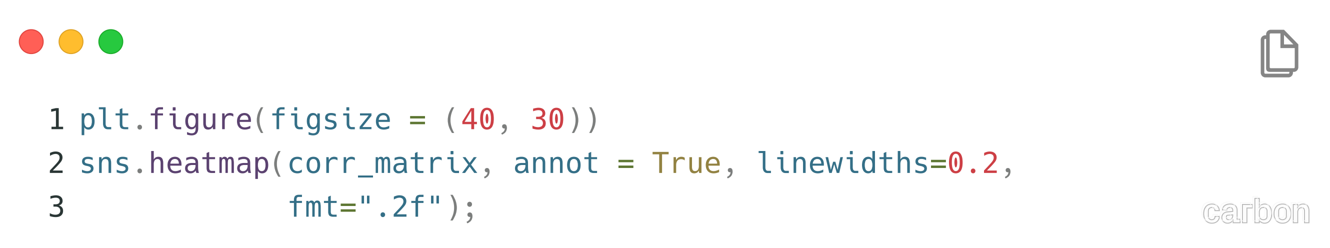
	MinCorrelationCoeff	MaxCorrelationCoeff
CustomerID	-0.0428	0.0413
NextPurchaseDay	-0.5695	0.4514
Recency	-0.9645	0.8574
RecencyCluster	-0.9645	0.9681
Frequency	-0.7039	0.4767
FrequencyCluster	-0.7039	0.2724
Revenue	-0.3777	0.7998
RevenueCluster	-0.3604	0.7998
OverallScore	-0.9326	0.9681
Segment_High-Value	-0.0813	0.6507
Segment_Low-Value	-0.9904	0.8574
Segment_Mid-Value	-0.9904	0.8794
NextPurchaseDayRange	-0.5695	0.5204

corr_matrix.csv hosted with ❤️ by GitHub view raw

Figure 18: Minimum and Maximum Correlation Coefficient

From Figure 18 above, it can be seen that **`OverallScore`** has the highest positive correlation of 0.97 with **`RecencyCluster`** and **`Segment_Low-Value`** has the highest negative of -0.99 with `Segment_Mid-Value`.

In Figure 19 below, I present a good visualisation of the coefficient matrix. The code snippet is below.



```
plt.figure(figsize = (40, 30))
sns.heatmap(corr_matrix, annot = True, linewidths=0.2,
            fmt=".2f");
```

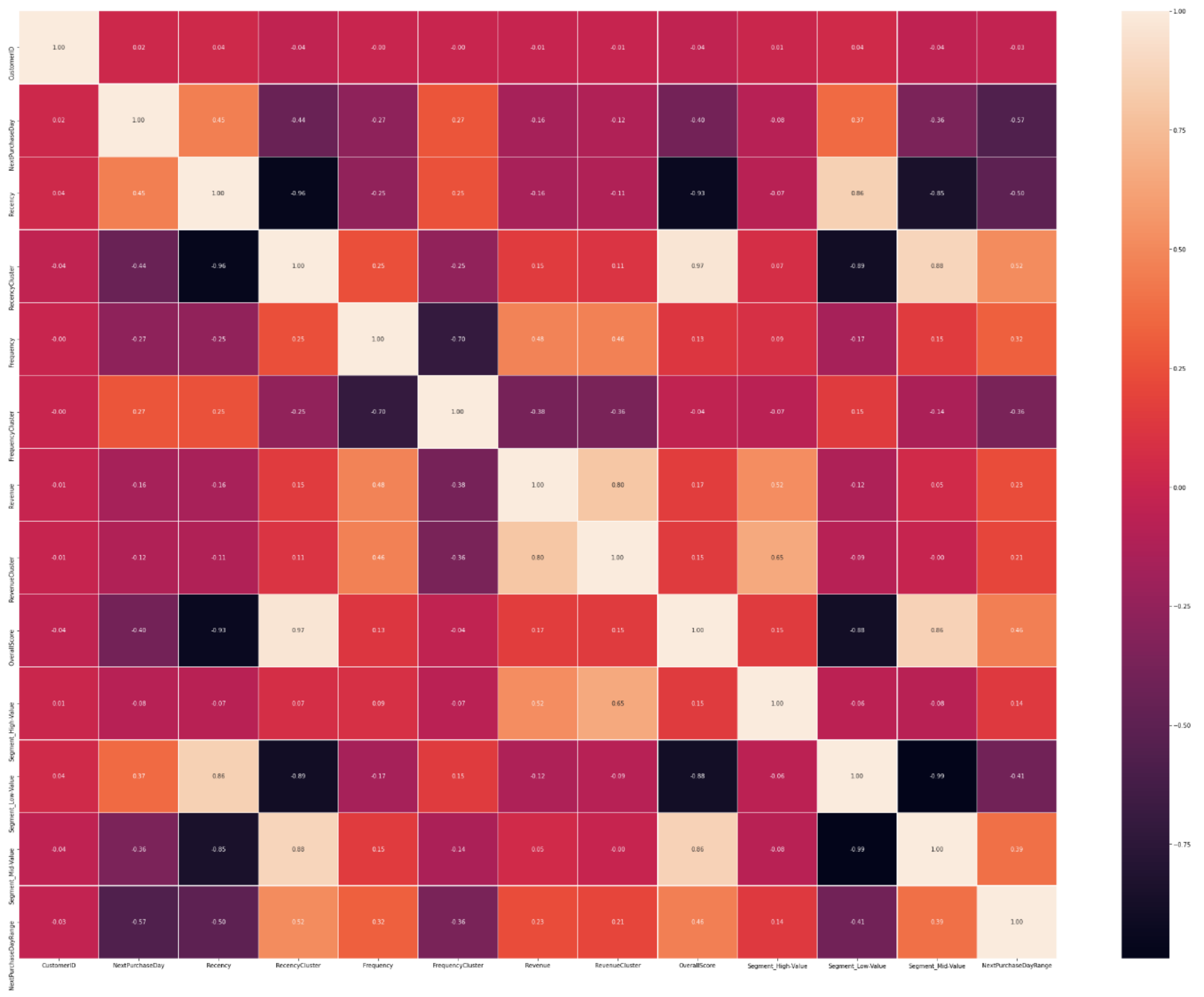


Figure 19: Correlation Matrix — Image by Author.

Building Machine Learning Models

In this section, I have what it takes with regard to the necessary prerequisites to build the machine learning model. The code snippet below separates the dataframe `ctm_class` into `X` features and the target variable `y`. Afterwards, I split `X` and `y` to get the training and test datasets and then measure the accuracy, F1-score, recall, and precision of the different models.

```python
ctm_class = ctm_class.drop('NextPurchaseDay', axis=1)
X, y = ctm_class.drop('NextPurchaseDayRange', axis=1), ctm_class.NextPurchaseDayRange

# Split feature data X and target data y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=None, shuffle=True)

# Create an array of models
models = []
models.append(("LogisticRegression", LogisticRegression()))
models.append(("GaussianNB", GaussianNB()))
models.append(("RandomForestClassifier", RandomForestClassifier()))
```

Search this file…

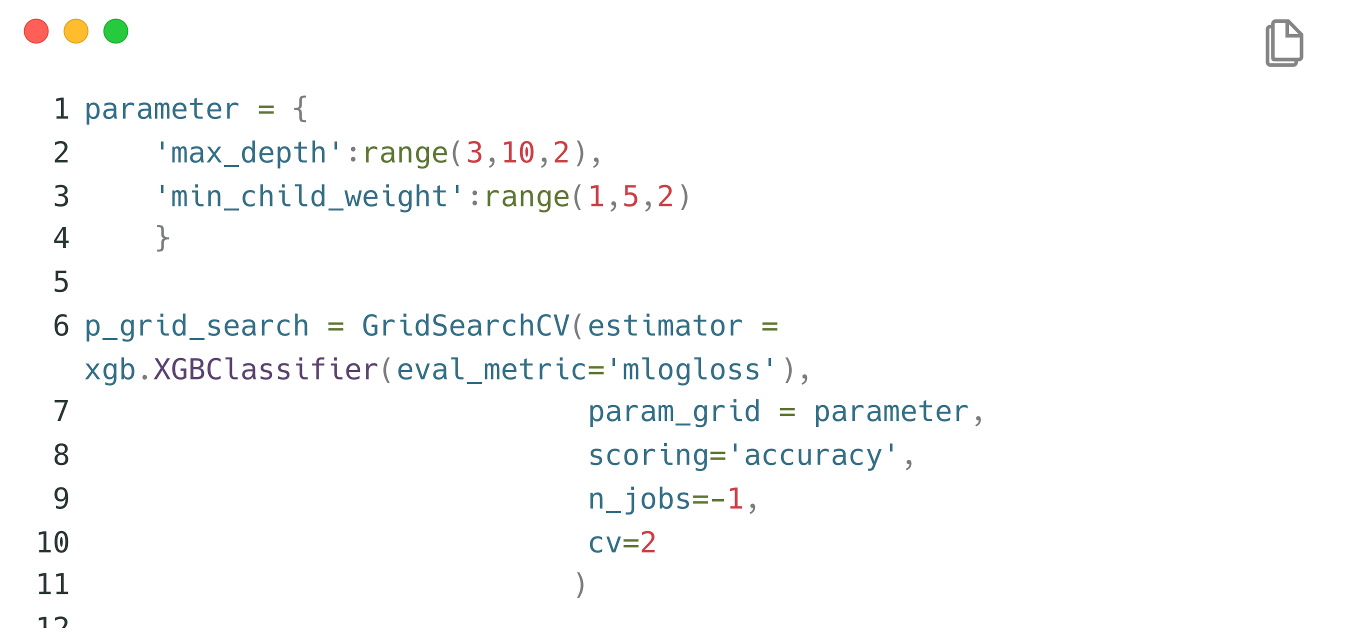
model_name	accuracy	f1_score	recall	precision	time
LogisticRegression	0.9040	0.8454	0.8474	0.8435	0.1087
RandomForestClassifier	0.8870	0.8325	0.8639	0.8215	0.5348
GaussianNB	0.8807	0.8297	0.8758	0.8140	0.0357
xgb.XGBClassifier	0.8772	0.7949	0.8058	0.8116	0.4897
SVC	0.8767	0.7905	0.8062	0.8161	0.3397
DecisionTreeClassifier	0.8747	0.7894	0.8024	0.8096	0.0415
KNeighborsClassifier	0.8721	0.7852	0.7926	0.8020	0.2305

model_metrics.csv hosted with ❤ by GitHub view raw

Figure 20: Metric of all models

From the results in Figure 20 above, we see that the `LogisticRegression` model is the best in terms of the metrics accuracy and F1-score.

Let's see how an improvement can be made for the existing model `XGB Classifier` which ranks fourth in Figure 20 above, by finding suitable parameters to control the learning process of the model. This process is called *hyperparameter tuning*. I then verify using the computation below to know if the improved `XGB Classifier` model outperforms the `LogisticRegression` model.

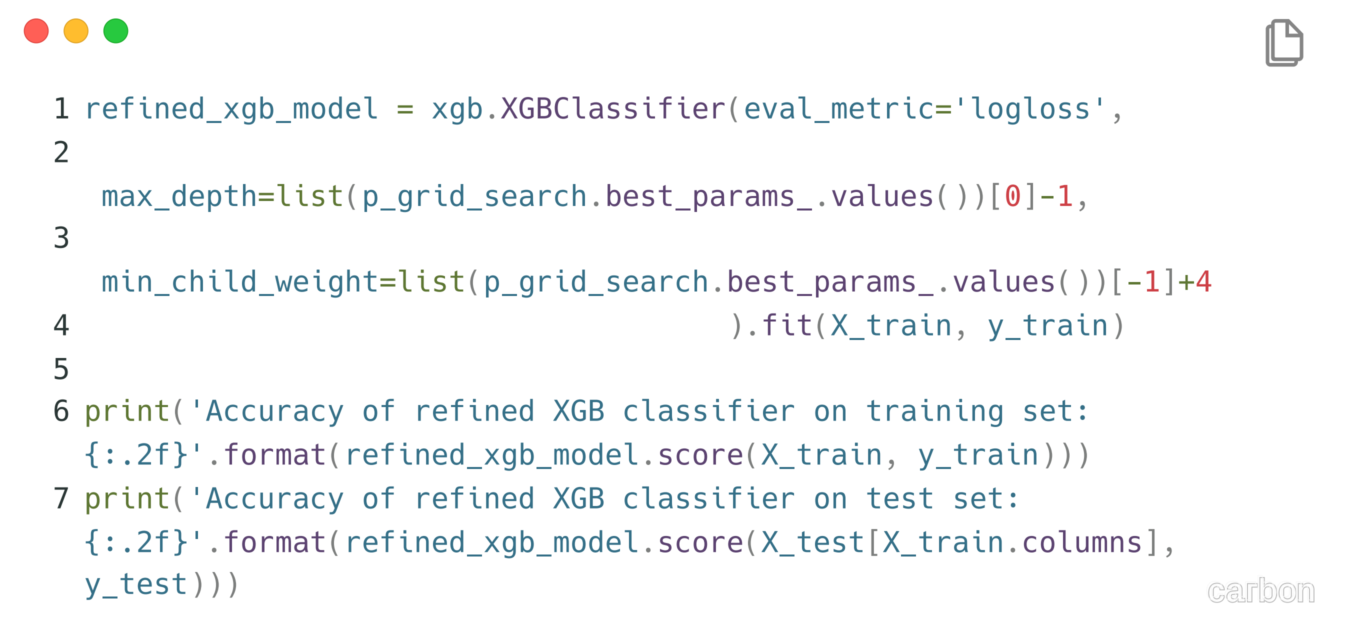


```python
parameter = {
    'max_depth':range(3,10,2),
    'min_child_weight':range(1,5,2)
    }

p_grid_search = GridSearchCV(estimator = xgb.XGBClassifier(eval_metric='mlogloss'),
                             param_grid = parameter,
                             scoring='accuracy',
                             n_jobs=-1,
                             cv=2
                            )
```

```
({'max_depth': 3, 'min_child_weight': 3}, 0.8993187980742626)
```

hyper_parameter_tunning.py hosted with ❤ by GitHub   view raw

Figure 21: XGB Classifier Model — Hyperparameter Tuning



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

```
Accuracy of refined XGB classifier on training set: 0.93
Accuracy of refined XGB classifier on test set: 0.92
```

refined_xgb_classifier_accuracy.txt hosted with ❤ by GitHub view raw

Figure 22: Refined XGB Classifier Accuracy Score

Selecting Model

Comparing the accuracy of the `LogisticsRegression` in Figures 20 above and that of the refined `XGB classifier` in Figure 22 above, it is obvious that the refined `XGB classifier` model is accurate than the `LogisticRegression` model by a margin of `0.1`. How about the other metrics?

```
# Predict the target values for the X_test with the refined XGB-Classifier
ref_xgb_pred_y = refined_xgb_model.predict(X_test)

# Predict the target values for the X_test with the Logistic Regression-Classifier
log_reg_pred_y = LogisticRegression().fit(X_train, y_train).predict(X_test)


# A dictionary of model names with the various metrics
ref_xgb_log_reg_dict = {"model_name" : ["xgb.XGBClassifier",
```

Search this file…

	model_name	accuracy	f1_score	recall	precision
2	xgb.XGBClassifier	0.9200	0.7848	0.7949	0.7750
3	LogisticRegression	0.9087	0.7569	0.7744	0.74020

ref_xgb_log_reg_scores.csv hosted with ❤ by GitHub view raw

Figure 23: Metric scores of XGB and LogisticRegression Classifiers

It is obvious from the output in Figure 23 above that for each metric, `accuracy`, `F1-score`, `recall`, and `precision`, the refined `XGB classifier` model outperforms the `LogisticRegression` model.

In forecasting the expectancy of a customer to make another purchase at the online retail shop after 90 days of one's last purchase, there is the need to be accurate in our submission. As a result, I am interested in the model which gives the highest accuracy possible in making this pre-emption. Thus it is the best of options to make a choice to use the improved `XGB classifier` model over the `LogisticRegression` model.

Conclusion

From the dataset, I highlight the fact that the strong customer base of the online shop centred in the United Kingdom is a major reason for the high revenue the company profits from the United Kingdom as a region.

I also give a detailed demonstration of how to build a machine learning model to predict whether an online customer of the retail shop will make their next purchase 90 days from the day they made their last purchase. Among the models that I used, I had to further improve on the `XGB classifier` model by the process of hyperparameter tuning to outperform the `LogisticRegression` model. The initial metrics after the hyperparameter tuning of the `XGB classifier` model `max_depth` and `min_child_weight` both set to 3, did not outperform that of the `LogisticRegression`. So I had to further tweak these values heuristically in order to get the `XGB classifier` model to outperform the `LogisticRegression` model.

The above notwithstanding, it will be interesting to investigate with further work how one can again improve the model's accuracy and F1-score metrics. I suggest improving the dataset by introducing the "right" `X` features so as to avoid the usage of a hyperparameter tuning process. So then my question now stands that

> What `X`– features would be appropriate to introduce into the dataset to reach or increase the model's hightened accuracy and F_1-score metrics without hyperparameter tuning?

The Jupyter notebook used for this article is available here.

Reference

[1] Barış Karaman. (Accessed on April 28, 2021) Predicting Next Purchase Day

Predictive Analytics Data Analysis Customer Engagement Deep Dives Business Intelligence