

# Configuration Manual

MSc Research Project  
Data Analytics

Prachi Pradeep Patil  
Student ID: x23109980

School of Computing  
National College of Ireland

Supervisor: Teerath Kumar Menghwar

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Prachi Pradeep Patil
<b>Student ID:</b>	x23109980
<b>Programme:</b>	Data Analytics
<b>Year:</b>	2024
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Teerath Kumar Menghwar
<b>Submission Due Date:</b>	12/08/2024
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	997
<b>Page Count:</b>	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

<b>Signature:</b>	Prachi Pradeep Patil
<b>Date:</b>	15th September 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	✓
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	✓
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Prachi Pradeep Patil  
x23109980

## 1 Introduction

### 1.1 Project Brief

In today's highly competitive retail and e-commerce landscape, understanding customer behavior is important for formulating effective marketing strategies. This research focuses on Customer Segmentation and Customer Lifetime Value (CLTV) prediction, essential techniques for classifying customers into segments and forecasting their future value to the business. By applying data analytics, retailers can customize their marketing efforts to maximize customer retention and profitability. The aim is to improve the predictive power of RFM analysis by implementing clustering techniques and supervised machine learning models.

### 1.2 Objective of Configuration Manual

A configuration manual is a detailed document outlining the necessary system, software, and library requirements for successfully running the project. Objective of this manual is to ensure that users can replicate the environment and execute the code without issues, providing detailed instructions on setup and installation. This manual aims for smooth implementation and operation of the project for other researchers and practitioners.

## 2 System Configuration

This section discusses the Hardware and Software specifications of the used machine in detail for smooth execution of the project.

### 2.1 Hardware Specification

Specification	Details
Operating System	Windows 11, 64-bit operating system (x64-based processor)
RAM	16.0 GB (15.7 GB usable)
Processor	12th Gen Intel(R) Core(TM) i5-1240P, 1.70 GHz

Table 1: System Specifications

Specification	Details
Programming Language	Python (Python Version - 3.11.5)
IDE	Jupyter Notebook
Platform	Anaconda Navigator (Anaconda Version - 23.7.4)
Web Browser	Google Chrome

Table 2: Software Specifications

## 2.2 Software Specification

## 2.3 Installation and Setup Guide

Step by step guide to install and setup the required softwares and libraries.

- Download the Anaconda Navigator from the official website <sup>1</sup>. Follow the installation instructions specific to operating system being used.

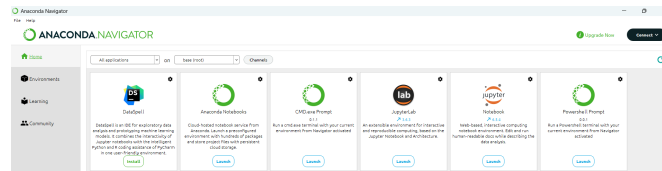


Figure 1: Anaconda Navigator

- Launch the Jupyter Notebook in the Anaconda Navigator as shown in Figure 1.

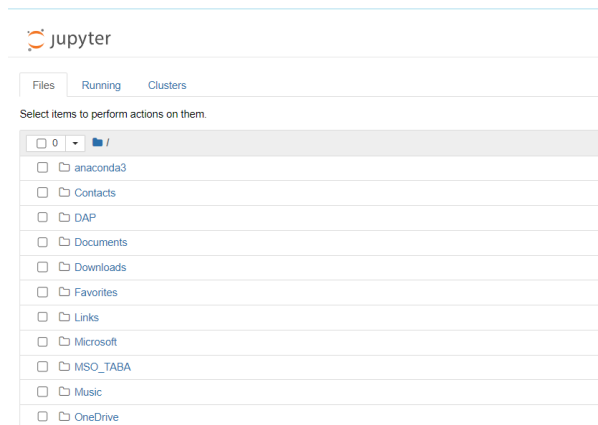


Figure 2: Jupyter Homepage

- New Tab with the Jupyter Notebook Home Page will open in Chrome Browser as shown in Figure 2.
- To begin coding, create a new .ipynb file.

<sup>1</sup>Anaconda Navigator :<https://docs.anaconda.com/anaconda/install/windows/>

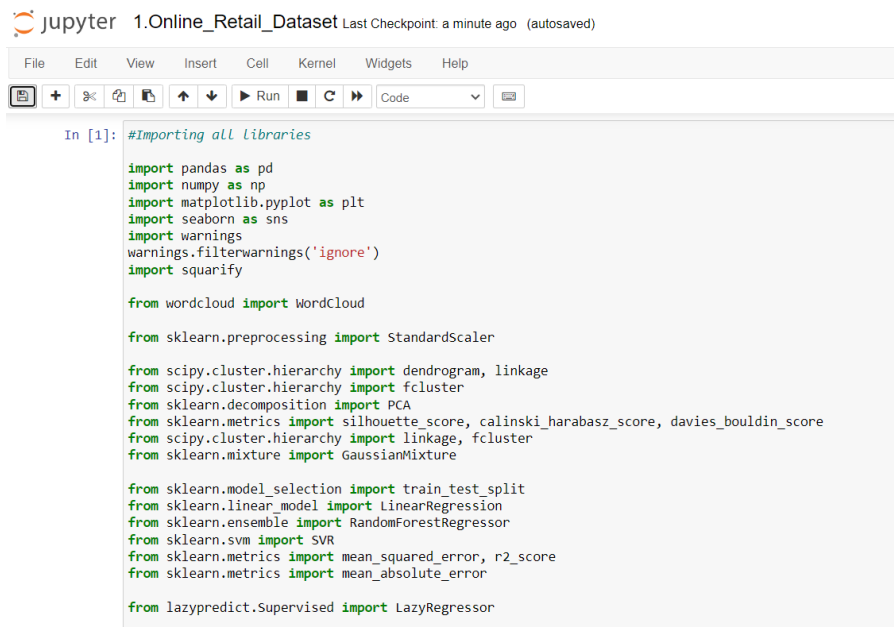
## 3 Data Selection

Two datasets used are Online Retail <sup>2</sup> and Retail Insights: A Comprehensive Sales Dataset <sup>3</sup>, obtained from open source ethical platform UCI machine learning repository and Kaggle respectively. Both datasets are available in CSV format.

## 4 Implementation

### 4.1 Importing all libraries and dataset

To begin with the implementation of code, all the required libraries have been installed and imported in jupyter notebook for Customer Segmentation and Customer Lifetime Value Prediction as shown in Figure 3.



```
In [1]: #Importing all libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import squarify

from wordcloud import WordCloud

from sklearn.preprocessing import StandardScaler

from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.mixture import GaussianMixture

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error

from lazypredict.Supervised import LazyRegressor
```

Figure 3: Importing all libraries

Used libraries collectively support data preprocessing, data manipulation, visualization, model development, model training and model evaluation.

### 4.2 Data Preprocessing

- In this section, the data processing steps such as - checking null values, duplicates, removing unnecessary features and many other steps should be performed as shown in Figure 4 on both datasets.

<sup>2</sup>Online Retail: <https://archive.ics.uci.edu/dataset/352/online+retail>

<sup>3</sup>RetailInsights:<https://www.kaggle.com/datasets/rajneesh231/retail-insights-a-comprehensive-sales-da>

```

# Check for duplicate rows
duplicates = df.duplicated().sum()
print("Number of duplicate rows:", duplicates)

Number of duplicate rows: 0

# Convert InvoiceDate to datetime
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%d-%m-%Y')

# Remove Unnecessary Columns
drop_columns = ['Address']
df = df.drop(columns=drop_columns)

# Convert Currency Columns
# Remove currency symbols and convert to numeric
currency_columns = ['Cost Price', 'Retail Price', 'Profit Margin', 'Sub Total', 'Discount $', 'Order Total', 'Shipping Cost', 'Total']
for col in currency_columns:
    df[col] = df[col].replace(['$', ''], regex=True).astype(float)

# Round off all prices to two decimal places
df[currency_columns] = df[currency_columns].round()

```

Figure 4: Data Processing

- The code as shown in Figure 5 is used for feature engineering to create new columns that shows additional information from the existing InvoiceDate and transaction details (Quantity and UnitPrice).

```

: # Data Preparation / Feature Engineering

df['TotalPrice'] = df['Quantity'] * df['UnitPrice']
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month
df['Day'] = df['InvoiceDate'].dt.day
df['Hour'] = df['InvoiceDate'].dt.hour
df['DayOfWeek'] = df['InvoiceDate'].dt.dayofweek
df['YearMonth'] = df['InvoiceDate'].dt.to_period('M')
df

```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalPrice	Year	Month	Day	Hour	DayOfWeek	YearMonth
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	15.30	2010	12	1	8	2	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	20.34	2010	12	1	8	2	2010-12

Figure 5: Feature Engineering

- These new features (TotalPrice, Year, Month, Day, Hour, DayOfWeek, YearMonth) can be useful for further analysis or for training machine learning models, as they provide detail information about the transactions.

### 4.3 RFM Ananlysis

- Calculate the Recency, Frequency, and Monetary (RFM) values for each customer by processing their transaction history.

```

#Calculate Recency,Frequency,Monetary
most_recent_date = df['InvoiceDate'].max()
# Group by 'CustomerID' and get the most recent 'InvoiceDate' for each customer
recency_df = df.groupby('CustomerID')['InvoiceDate'].max().reset_index()
# Calculate the recency in days
recency_df['Recency'] = (most_recent_date - recency_df['InvoiceDate']).dt.days
recency_df.head(10)
#Group by 'CustomerID' and calculate the frequency of purchases for each customer
frequency_df = df.groupby('CustomerID').size().reset_index(name='Frequency')
frequency_df.head(10)
#Group by 'CustomerID' and calculate the total monetary value of purchases for each customer
monetary_df = df.groupby('CustomerID')['TotalPrice'].sum().reset_index()
monetary_df.columns = ['CustomerID', 'Monetary']
monetary_df.head(10)
#Merging all RFM values in one dataframe
RFM = recency_df.merge(frequency_df, on = 'CustomerID').merge(monetary_df, on = 'CustomerID')
RFM

```

Figure 6: Calculating RFM Values

Assign scores to customers based on their Recency, Frequency, and Monetary values, and combines them into a single RFM score.

```

# Assign scores for Recency
RFM['RecencyScore'] = pd.qcut(RFM['Recency'], 5, labels=[5, 4, 3, 2, 1])
# Assign scores for Frequency
RFM['FrequencyScore'] = pd.qcut(RFM['Frequency'].rank(method='first'), 5, labels=[1, 2, 3, 4, 5])
# Assign scores for Monetary
RFM['MonetaryScore'] = pd.qcut(RFM['Monetary'], 5, labels=[1, 2, 3, 4, 5])
# Combine the scores into a single RFM score
RFM['RFMScore'] = RFM['RecencyScore'].astype(str) + RFM['FrequencyScore'].astype(str) + RFM['MonetaryScore'].astype(str)

```

Figure 7: Assigning RFM Score

- Categorize the customers in distinct segments like 'Champions', 'Loyal Customers', 'Potential Loyalists', 'Needs Attention', 'At Risk'.

```

# Define RFM segments
def rfm_segment(x):
    r, f, m = int(x[0]), int(x[1]), int(x[2])
    if r >= 4 and f >= 4 and m >= 4:
        return 'Champions'
    elif r >= 4 and f >= 3 and m >= 3:
        return 'Loyal Customers'
    elif r >= 3 and f >= 2 and m >= 2:
        return 'Potential Loyalists'
    elif r >= 2 and f >= 1 and m >= 1:
        return 'Needs Attention'
    else:
        return 'At Risk'

RFM['Segment'] = RFM['RFMScore'].apply(rfm_segment)

```

Figure 8: Segmenting RFM Customers

## 4.4 Hierarchical Clustering

- Using Ward's method in Hierarchical clustering to minimize the variance within clusters.

```
# Perform hierarchical/agglomerative clustering
Z = linkage(RFM_scaled, method='ward')

# Decide the number of clusters based on the dendrogram and assign cluster labels
max_d = 40 # Adjust this value based on the dendrogram
clusters = fcluster(Z, max_d, criterion='distance')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z, truncate_mode='lastp', p=12, show_leaf_counts=False, leaf_rotation=90, leaf_font_size=12, show_contracted=True)
plt.title('Customer Dendrogram')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.axhline(y=max_d, color='r', linestyle='--')
plt.grid()
plt.show()

# Add cluster labels to the RFM dataframe
Clean_RFM['H_Cluster'] = clusters
```

Figure 9: Hierarchical Clustering

- Calculate cluster labels based on a specified distance threshold. Visualizes the clusters with a dendrogram, and adds the cluster labels to the RFM DataFrame for further analysis.
- Apply PCA technique to reduce the dimensionality to 2 components(2D).

```
: # Scatter Plot using PCA
pca = PCA(n_components=2)
RFM_pca = pca.fit_transform(Clean_RFM[['Recency', 'Frequency', 'Monetary']])

plt.figure(figsize=(10, 7))
sns.scatterplot(x=RFM_pca[:, 0], y=RFM_pca[:, 1], hue=Clean_RFM['H_Cluster'], palette='tab10')
plt.title('Customer Segments (PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='cluster')
plt.grid()
plt.show()
```

Figure 10: Customer Segments (PCA)

- Visualize the customer segments in a scatter plot, highlighting the clusters identified through hierarchical clustering.



## 4.5 Gaussian Mixture Model

- Apply Gaussian Mixture Modeling (GMM) to cluster customers based on their RFM values, and then visualizes the clusters in a 2D scatter plot using PCA for dimensionality reduction.

```
: # Fit the GMM Model
gmm = GaussianMixture(n_components=4, random_state=42) # Adjust the number of components if needed
gmm.fit(RFM_scaled)

# Assign Clusters
Clean_RFM['GMM_Cluster'] = gmm.predict(RFM_scaled)

# Add GMM cluster labels to the RFM dataframe
gmm_clusters = Clean_RFM['GMM_Cluster']

# Scatter Plot using PCA
pca = PCA(n_components=2)
RFM_pca = pca.fit_transform(RFM_scaled)

plt.figure(figsize=(10, 7))
sns.scatterplot(x=RFM_pca[:, 0], y=RFM_pca[:, 1], hue=Clean_RFM['GMM_Cluster'], palette='tab10')
plt.title('Customer Segments (GMM, PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='GMM Cluster')
plt.grid()
plt.show()
```

Figure 11: Gaussian Clustering

- Evaluate the performance of the Gaussian Mixture Model (GMM) clustering using metrics like the Silhouette Score, Calinski-Harabasz Index, Davies-Bouldin Index, and model fit criteria (BIC and AIC).

```
#Calculate evaluation metrics for GMM
Silhouette_Score_GMM = silhouette_score(RFM_scaled, gmm_clusters)
Calinski_Harabasz_Index_GMM = calinski_harabasz_score(RFM_scaled, gmm_clusters)
Davies_Bouldin_Index_GMM = davies_bouldin_score(RFM_scaled, gmm_clusters)
# BIC and AIC for GMM
BIC_gmm = gmm.bic(RFM_scaled)
AIC_gmm = gmm.aic(RFM_scaled)

print("\nGaussian Mixture Model Evaluation Metrics:")
print(f"Silhouette Score: {Silhouette_Score_GMM}")
print(f"Calinski-Harabasz Index: {Calinski_Harabasz_Index_GMM}")
print(f"Davies-Bouldin Index: {Davies_Bouldin_Index_GMM}")
print(f"BIC: {BIC_gmm}")
print(f"AIC: {AIC_gmm}")
```

Figure 12: Evaluation of GMM

## 4.6 Regression Models

### 4.6.1 Linear Regression

- Split the dataset into training and testing sets and standardizes the features, preparing the data for machine learning model training and evaluation.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 13: Splitting the data

- Train a linear regression model on the standardized training data and then uses the trained model to make predictions on the test set.

```
: # Initialize and train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test_scaled)

# Evaluate the model
MSE_LR = mean_squared_error(y_test, y_pred)
R2_LR = r2_score(y_test, y_pred)
MAE_LR = mean_absolute_error(y_test, y_pred)
RMSE_LR = np.sqrt(MSE_LR)
MAPE_LR = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print("\nLinear Regression Model Evaluation Metrics:")
print(f'Linear Regression Mean Squared Error: {MSE_LR}')
print(f'Linear Regression R^2 Score: {R2_LR}')
print(f'Linear Regression Mean Absolute Error (MAE): {MAE_LR}')
print(f'Linear Regression Root Mean Squared Error (RMSE): {RMSE_LR}')
print(f'Linear Regression Mean Absolute Percentage Error (MAPE): {MAPE_LR}%')
```

Figure 14: Linear Regression

- Evaluate the linear regression model's performance using metrics like MSE,  $R^2$ , MAE, RMSE, and MAPE, to gain insights into the model's accuracy and fit.

```

: # Apply Log transformation to features and target variable
X_log = np.log1p(Clean_RFM[['Recency', 'Frequency']])
y_log = np.log1p(Clean_RFM['Monetary'])

: # Split the Log-transformed data into training and testing sets
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(X_log, y_log, test_size=0.2, random_state=42)

# Standardize the Log-transformed features
scaler_log = StandardScaler()
X_train_log_scaled = scaler_log.fit_transform(X_train_log)
X_test_log_scaled = scaler_log.transform(X_test_log)

: # Initialize and train the Linear Regression model on Log-transformed data
lr_log = LinearRegression()
lr_log.fit(X_train_log_scaled, y_train_log)

# Make predictions on the Log-transformed test set
y_pred_log = lr_log.predict(X_test_log_scaled)

```

Figure 15: Linear Regression with Log Transformation

- Apply a log transformation to stabilize variance and normalize the data, then train a linear regression model on the transformed and standardized features, and finally makes predictions on the log-transformed test set.

#### 4.6.2 Random Forest Regression

- Initialize and train a Random Forest Regressor on standardized training data and then uses the trained model to make predictions on the test set.

```

# Initialize the Random Forest model
rf = RandomForestRegressor(random_state=42)

# Fit the model to the training data
rf.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_rf = rf.predict(X_test_scaled)

# Evaluate the model
MSE_RF = mean_squared_error(y_test, y_pred_rf)
R2_RF = r2_score(y_test, y_pred_rf)
MAE_RF = mean_absolute_error(y_test, y_pred_rf)
RMSE_RF = np.sqrt(MSE_RF)
MAPE_RF = np.mean(np.abs((y_test - y_pred_rf) / y_test)) * 100

print("\nRandom Forest Model Evaluation Metrics:")
print(f'Random Forest Mean Squared Error: {MSE_RF}')
print(f'Random Forest R^2 Score: {R2_RF}')
print(f'Random Forest Mean Absolute Error (MAE): {MAE_RF}')
print(f'Random Forest Root Mean Squared Error (RMSE): {RMSE_RF}')
print(f'Random Forest Mean Absolute Percentage Error (MAPE): {MAPE_RF}%')

```

Figure 16: Random Forest Regression

- Evaluate the performance of the Random Forest model using metrics such as MSE,  $R^2$ , MAE, RMSE, and MAPE, to gain insights into the model's accuracy and fit.

```

# Initialize and train the Random Forest model on log-transformed data
rf_log = RandomForestRegressor(n_estimators=100, random_state=42)
rf_log.fit(X_train_log_scaled, y_train_log)

# Make predictions on the log-transformed test set
y_pred_log_rf = rf_log.predict(X_test_log_scaled)

# Evaluate the Random Forest model on log-transformed data
MSE_RF_log = mean_squared_error(y_test_log, y_pred_log_rf)
R2_RF_log = r2_score(y_test_log, y_pred_log_rf)
MAE_RF_log = mean_absolute_error(y_test_log, y_pred_log_rf)
RMSE_RF_log = np.sqrt(MSE_RF_log)
MAPE_RF_log = np.mean(np.abs((y_test_log - y_pred_log_rf) / y_test_log)) * 100

print("\nRandom Forest Model Evaluation Metrics (Log-Transformed Data):")
print(f'Random Forest Mean Squared Error: {MSE_RF_log}')
print(f'Random Forest R^2 Score: {R2_RF_log}')
print(f'Random Forest Mean Absolute Error (MAE): {MAE_RF_log}')
print(f'Random Forest Root Mean Squared Error (RMSE): {RMSE_RF_log}')
print(f'Random Forest Mean Absolute Percentage Error (MAPE): {MAPE_RF_log}%')

```

Figure 17: Random Forest with Log Transformation

- Train a Random Forest Regressor on log-transformed data, makes predictions on the log-transformed test set, and evaluate the model's performance using metrics like MSE,  $R^2$ , MAE, RMSE, and MAPE.

#### 4.6.3 Support Vector Regression

- Train a Support Vector Regressor (SVR) with an RBF kernel on standardized training data and then makes predictions on the standardized test set.

```

# Initialize and train the SVR model
svr = SVR(kernel='rbf')
svr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_svr = svr.predict(X_test_scaled)

# Evaluate the model
MSE_SVR = mean_squared_error(y_test, y_pred_svr)
R2_SVR = r2_score(y_test, y_pred_svr)
MAE_SVR = mean_absolute_error(y_test, y_pred_svr)
RMSE_SVR = np.sqrt(MSE_SVR)
MAPE_SVR = np.mean(np.abs((y_test - y_pred_svr) / y_test)) * 100

print(f'SVR Mean Squared Error (MSE): {MSE_SVR}')
print(f'SVR R^2 Score: {R2_SVR}')
print(f'SVR Mean Absolute Error (MAE): {MAE_SVR}')
print(f'SVR Root Mean Squared Error (RMSE): {RMSE_SVR}')
print(f'SVR Mean Absolute Percentage Error (MAPE): {MAPE_SVR}%')

```

Figure 18: Support Vector Regression

- Evaluate the performance of the Support Vector Regressor (SVR) using metrics like MSE,  $R^2$ , MAE, RMSE, and MAPE, to gain insights into the model's accuracy and fit.

```

# Initialize and train the SVR model on Log-transformed data
svr_log = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
svr_log.fit(X_train_log_scaled, y_train_log)

# Make predictions on the log-transformed test set
y_pred_log_svr = svr_log.predict(X_test_log_scaled)

# Evaluate the SVR model on Log-transformed data
MSE_SVR_log = mean_squared_error(y_test_log, y_pred_log_svr)
R2_SVR_log = r2_score(y_test_log, y_pred_log_svr)
MAE_SVR_log = mean_absolute_error(y_test_log, y_pred_log_svr)
RMSE_SVR_log = np.sqrt(MSE_SVR_log)
MAPE_SVR_log = np.mean(np.abs((y_test_log - y_pred_log_svr) / y_test_log)) * 100

print("\nSupport Vector Regression Model Evaluation Metrics (Log-Transformed Data):")
print(f'Support Vector Regression Mean Squared Error: {MSE_SVR_log}')
print(f'Support Vector Regression R^2 Score: {R2_SVR_log}')
print(f'Support Vector Regression Mean Absolute Error (MAE): {MAE_SVR_log}')
print(f'Support Vector Regression Root Mean Squared Error (RMSE): {RMSE_SVR_log}')
print(f'Support Vector Regression Mean Absolute Percentage Error (MAPE): {MAPE_SVR_log}%')

```

Figure 19: SVR using Log Transformation

- Train a Support Vector Regressor (SVR) with an RBF kernel on log-transformed data, to make predictions on the log-transformed test set, and evaluate the model's performance using various metrics.

#### 4.6.4 By LazyPredict Library

- Use LazyRegressor to automatically fit and evaluate multiple regression models on standardized training and test data.

```

# Initialize LazyRegressor
reg = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)

# Fit the models and get the results
models, predictions = reg.fit(X_train_scaled, X_test_scaled, y_train, y_test)

# Display the results
print(models)

```

Figure 20: Evaluating models using LazyPredict Library

- Apply log transformation and standardization to the features and target variable, then use LazyRegressor to fit multiple regression models on the transformed data.

```

# Apply Log transformation to features and target variable
X_log = np.log1p(Clean_RFH[['Recency', 'Frequency']])
y_log = np.log1p(Clean_RFH['Monetary'])

# Split the Log-transformed data into training and testing sets
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(X_log, y_log, test_size=0.2, random_state=42)

# Standardize the Log-transformed features
scaler_log = StandardScaler()
X_train_log_scaled = scaler_log.fit_transform(X_train_log)
X_test_log_scaled = scaler_log.transform(X_test_log)

# Initialize LazyRegressor for Log-transformed data
reg_log = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)

# Fit the models and get the results on Log-transformed data
models_log, predictions_log = reg_log.fit(X_train_log_scaled, X_test_log_scaled, y_train_log, y_test_log)

# Display the results for Log-transformed data
print(models_log)

```

Figure 21: Evaluating models using LazyPredict with Log Transformation