PROJECT REPORT

- **Submitted By:** Anurag Gupta
- **Institution:** Kanpur Institute Of Technology
- **Program:** B.Tech in Artificial Intelligence and Machine Learning

Project Title: Customer Segmentation and Stock Price Prediction using Machine Learning

Date: 07/05/2024

Author: Anurag Gupta

Table of Contents:

Abstract: This project aims to segment customers based on their tenure, monthly charges, and total charges using machine learning techniques. The report covers the data collection process, exploratory data analysis, feature engineering steps, model building process, and the evaluation of the model's performance. The results and insights gained from this analysis can help businesses better understand their customers and tailor their marketing strategies accordingly.

Keywords: Customer Segmentation, Machine Learning, Exploratory Data Analysis, Feature Engineering, Random Forest, Churn Prediction.

Acknowledgements: We would like to thank "**Yhills platform**" for providing the dataset used in this analysis.

Project Report: Customer Segmentation

1. Introduction

Customer segmentation is a crucial task in marketing and business strategy. By dividing customers into groups based on common characteristics, businesses can tailor their marketing efforts and improve customer satisfaction. In this project, we perform customer segmentation using machine learning techniques.

2. Data Collection and Preprocessing

- We collected the data from the 'Customer_Segmentation.csv' file.
- The dataset contains information about customers, including their tenure, monthly charges, total charges, and churn status ('Yes' or 'No').
- We checked for missing values and outliers in the data and performed necessary preprocessing steps, such as filling missing values and standardizing numerical features.

3. Exploratory Data Analysis (EDA)

- We visualized the relationship between tenure and monthly charges using a scatter plot.
- We created a pie chart to show the distribution of tenure for a subset of customers.
- We plotted a line graph showing the relationship between tenure and total charges for a subset of customers.

4. Feature Engineering

- We standardized the numerical features (tenure, monthly charges, total charges) using Standard Scaler.
- We normalized the numerical features using Min Max Scaler.

5. Model Building

- We split the standardized features and the target variable ('Churn') into training and test sets.
- We built a Random Forest classifier model using the standardized features.
- We trained the model on the training set and evaluated its performance on the test set.

6. Results

- The Random Forest model achieved an accuracy of [0.7774] on the test set.

7. Conclusion

- Customer segmentation is a critical task for businesses to understand their customer base better.
- Machine learning models can help in automating the segmentation process and improving marketing strategies.
- Further analysis and experimentation can be done to improve the model's performance and explore other segmentation techniques.

Source Code :

```
            # CUSTOMER SEGMENTATION
import os
import pandas as pd
import numpy as np

from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression



import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

                        # Data Collection


data_path = 'Customer_Segmentation.csv'
if os.path.exists(data_path):
    Data = pd.read_csv(data_path)
    print(Data.head())
    print(Data.tail())
    print("Shape of Datasheet:", Data.shape)
```

```
    print("Columns of Datasheet:", Data.columns)
    print(Data.index)
    print("\n")
    print(Data.loc[[1, 2]])
else:
    print(f"File {data_path} not found.")

                        # Data Preprocessing
print(Data.isna())
print(Data.isna().sum())
print(Data.head().dropna())
print("\n")
print(Data.tail().sort_values)

                        # Data Analysis
# Scatter Plot
ploting = Data.plot(kind="scatter",x= "tenure", y= "MonthlyCharges")
plt.title("Customer Segmentation")
plt.xlabel("Tenure")
plt.ylabel("MonthlyCharges")
plt.show()

# Pie plot
df = Data.head()["tenure"].plot(kind= "pie")
```

```
plt.show()


# Line plot

Data['MonthlyCharges'] = pd.to_numeric(Data['MonthlyCharges'], errors='coerce')

Data['TotalCharges'] = pd.to_numeric(Data['TotalCharges'], errors='coerce')

Data.head().plot(kind="line", x="tenure", y="TotalCharges")

plt.xlabel("tenure")

plt.ylabel("Total Charges")

plt.title("Tenure vs Total Charges")

plt.show()
```

Exploratory Data Analysis (EDA) - (Missing Values And Outlier)

```
print(Data["MonthlyCharges"].isnull().sum())                # Before Filling

avg = Data['MonthlyCharges'].mean()

print("Average of MonthlyCharges columns = ", avg,"\n")            # filling missing value with
avg

Data['MonthlyCharges'] = Data['MonthlyCharges'].fillna(avg)

print(Data["MonthlyCharges"].isnull().sum())                # After Filling




# Outlier

sns.boxplot(y = Data["tenure"])
```

```
plt.show()

sns.boxenplot(x =Data["TotalCharges"])

plt.show()


                              # Feature Engineering


from sklearn.preprocessing import StandardScaler, MinMaxScaler


# Standardization

scaler_standard = StandardScaler()

Data_standardized = scaler_standard.fit_transform(Data[['tenure', 'MonthlyCharges',
'TotalCharges']])

Data_standardized = pd.DataFrame(Data_standardized, columns=['tenure', 'MonthlyCharges',
'TotalCharges'])


# Normalization

scaler_minmax = MinMaxScaler()

Data_normalized = scaler_minmax.fit_transform(Data[['tenure', 'MonthlyCharges',
'TotalCharges']])

Data_normalized = pd.DataFrame(Data_normalized, columns=['tenure', 'MonthlyCharges',
'TotalCharges'])


# Plotting the standardized and normalized data


plt.subplot(2, 3, 1)
```

```
sns.histplot(Data_standardized['tenure'])
plt.title('Standardized Tenure')

plt.subplot(2, 3, 2)
sns.histplot(Data_standardized['MonthlyCharges'])
plt.title('Standardized MonthlyCharges')

plt.subplot(2, 3, 3)
sns.histplot(Data_standardized['TotalCharges'])
plt.title('Standardized TotalCharges')


plt.tight_layout()
plt.show()

# Model Building


from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report


Data['Churn'] = Data['Churn'].map({'Yes': 1, 'No': 0})
```

```
# Selecting features and target variable
X = Data_standardized[['TotalCharges', 'MonthlyCharges']]
y = Data['Churn']



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

# Building a Random Forest model using standardized features
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)




# Evaluating the models
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Random Forest model using standardized  {0:0.4f}".format(accuracy))
```

Project Report: Stock Price Prediction

1. Introduction

Stock price prediction is a challenging yet essential task in the financial industry. By forecasting stock prices, investors and traders can make informed decisions and maximize their returns. In this project, we aim to predict stock prices using machine learning techniques.

2. Data Collection and Preprocessing

- **Data Source:** The data was collected from the 'Stock_Price_Prediction.csv' file.
- **Dataset Description:** The dataset contains stock price information, including open, high, low, close prices, and volume.
- **Preprocessing Steps:**
- Checked for missing values and outliers.
- Filled missing values with appropriate methods (e.g., median, mean).
- Standardized numerical features using Standard Scaler.
- Normalized numerical features using Min Max Scaler.

3. Exploratory Data Analysis (EDA)

- **Visualization 1:** Line plot showing the relationship between high and low prices over time.
- **Visualization 2:** Pie chart displaying the distribution of volume for a subset of data.
- **Visualization 3:** Scatter plot illustrating the relationship between high and low prices.

4. Feature Engineering

- Standardized the numerical features (open, high, low, close prices) using Standard Scaler.
- Normalized the numerical features using Min Max Scaler.

5. Model Building

- Split the standardized features and the target variable ('Close price') into training and test sets.
- Built a Random Forest regression model using the standardized features.
- Trained the model on the training set and evaluated its performance on the test set.

6. Results

- The Random Forest regression model achieved a mean squared error of [26.1003] on the test set.

7. Conclusion

- Stock price prediction is a critical task for investors and traders.
- Machine learning models can help in predicting stock prices and making informed investment decisions.
- Further analysis and experimentation can be done to improve the model's performance and explore other prediction techniques

Source Code :

```
# STOCK PRICE PREDICTION
import os
import pandas as pd
import numpy as np

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns


# Data Collection and Preprocessing

data_path = 'Stock_Price_Prediction.csv'
if os.path.exists(data_path):
    Data = pd.read_csv(data_path)
    print(Data.head())
    print(Data.tail())
    print(Data.loc[[1, 2]])
    print(Data.index)
    print("Shape of Datasheet:", Data.shape)
    print("Columns of Datasheet:", Data.columns)
else:
```

```
    print(f"File {data_path} not found.")


print(Data.isna())

print(Data.isna().sum())

print(Data.head().dropna())

print(Data.tail().sort_values)


# Data Analysis


# Line plot

Data['High'] = pd.to_numeric(Data['High'])

Data['Low'] = pd.to_numeric(Data['Low'])

Data.head(144).plot(kind="line", x="High", y="Low")

plt.xlabel("High")

plt.ylabel("Low")

plt.title("High vs Low")

plt.show()



# Pie plot

df = Data.head(14)["Volume"].plot(kind= "pie")

plt.show()


# Scatter Plot
```

```python
ploting = Data.plot(kind="scatter",x= "High", y= "Low")
plt.title("Stock price Prediction")
plt.xlabel("High")
plt.ylabel("Low")
plt.show()


# Exploratory Data Analysis (EDA)
print("--------------------------------------------------------------------------")
print(Data["Adj Close"].isnull().sum())                              # Before Filling
Median = Data['Adj Close'].std()
print("Median of  columns = ", Median,"\n")             # filling missing value with avg
Data['Adj Close'] = Data['Close'].fillna(Median)
print(Data["Adj Close"].isnull().sum())                       # After Filling
print("--------------------------------------------------------------------------")




# Outlier
sns.boxenplot(x =Data["Volume"])
plt.show()


sns.boxplot(y = Data["Close"])
plt.show()
```

```python
# Feature Engineering


from sklearn import preprocessing

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import StandardScaler, MinMaxScaler


# Standardization

scaler_standard = StandardScaler()

Data_standardized = scaler_standard.fit_transform(Data[['Open', 'High', 'Close']])

Data_standardized = pd.DataFrame(Data_standardized, columns=['Open', 'High', 'Close'])


# Normalization

scaler_minmax = MinMaxScaler()

Data_normalized = scaler_minmax.fit_transform(Data[['Open', 'High', 'Close']])

Data_normalized = pd.DataFrame(Data_normalized, columns=['Open', 'High', 'Close'])


# Plotting the standardized and normalized data


plt.subplot(2, 3, 1)

sns.histplot(Data_normalized['Open'])

plt.title('Normalized Data')


plt.subplot(2, 3, 5)

sns.histplot(Data_standardized['High'])
```

```
plt.title('Standardized Data')


plt.subplot(2, 3, 3)
sns.histplot(Data_normalized['Close'])
plt.title('Normalized Data')



plt.tight_layout()
plt.show()


# Model Building and It's Accuracy


from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


# Selecting features and target variable
X = Data[['Open', 'High', 'Low', 'Volume']]
y = Data['Adj Close']


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


# Building a Random Forest regression model
```

```
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```