PROJECT FOR DATA SCIENCE AND BUSINESS

TASK NO 4: Predicting Customer Churn in a Telecom Company

STUDENT NAME AND ENROLLMENT NO:NUNNA SOMANADH ,24CS003440.

SEC :G

Data loading

Subtask:

Load the dataset "somuprodata1.csv" into a pandas DataFrame.

- CODE:
- import pandas as pd

```
df = pd.read_csv('somuprodata1.csv')

display(df.head())
```

Data Preprocessing: Clean the data, handle missing values, and encodecategorical
features.

Data exploration

Subtask:

Explore the loaded dataset df.

Reasoning: Explore the dataset by finding its shape, data types, descriptive statistics, target variable distribution, missing values, and correlation with the target variable.

CODE:

```
# 1. Shape of the data
print("Shape of the data:", df.shape)

# 2. Data types of each column
print("\nData types of each column:\n", df.dtypes)

# 3. Descriptive statistics for numerical features
print("\nDescriptive statistics for numerical features:\n", df.describe())

# 4. Distribution of the target variable (churn)
print("\nDistribution of the target variable (churn):\n", df['Churn'].value_counts())
```

```
# 5. Missing values in each column
print("\nMissing values in each column:\n", df.isnull().sum())
```

Data cleaning

Subtask:

Clean the data in the DataFrame df.

Reasoning: Identify and handle missing values, convert 'TotalCharges' to numeric, and handle duplicates.

CODE:

```
# Handle missing values
df.dropna(inplace=True)

# Convert 'TotalCharges' to numeric
df['TotalCharges']=pd.to_numeric(df['TotalCharges'],
errors='coerce')
df.dropna(subset=['TotalCharges'], inplace=True)

# Handle duplicate rows
df.drop_duplicates(inplace=True)
```

Data wrangling

Subtask:

Encode categorical features in the DataFrame df.

Reasoning: Identify categorical features and encode them using one-hot encoding or label encoding based on the number of unique values and the presence of a natural order.

CODE:

```
from sklearn.preprocessing import LabelEncoder

# Identify categorical features
categorical_features =
df.select_dtypes(include=['object']).columns.tolist()

# Encode categorical features
for feature in categorical_features:
```

```
  if df[feature].nunique() <= 10:  # Consider one-hot encoding for features with fewer unique values
    df = pd.get_dummies(df, columns=[feature], prefix=[feature])
  else:  # Consider label encoding for features with many unique values
    le = LabelEncoder()
    df[feature] = le.fit_transform(df[feature])
```

EDA: Visualize relationships between features and churn using barcharts,histograms,and correlation matrices.

Data visualization

Subtask:

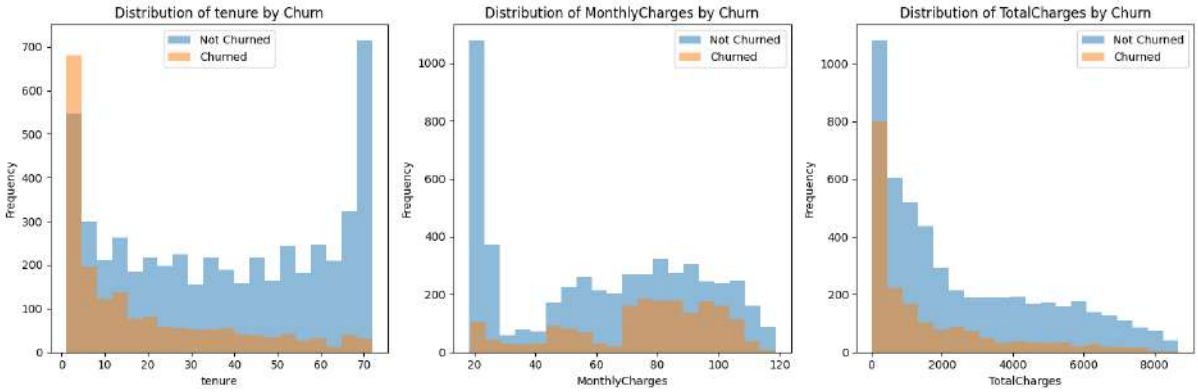
Visualize the relationships between features and churn using histograms, bar charts, and correlation matrices.

Reasoning: Generate histograms for numerical features, comparing churned and non-churned customers.

CODE:

```
import matplotlib.pyplot as plt

# Histograms for numerical features, comparing churned and non-churned customers
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']

plt.figure(figsize=(15, 5))

for i, feature in enumerate(numerical_features):
  plt.subplot(1, 3, i + 1)
    plt.hist(df[df['Churn'] == 0][feature], bins=20, alpha=0.5, label='Not Churned')
    plt.hist(df[df['Churn'] == 1][feature], bins=20, alpha=0.5, label='Churned')
  plt.xlabel(feature)
  plt.ylabel('Frequency')
  plt.title(f'Distribution of {feature} by Churn')
  plt.legend()
```

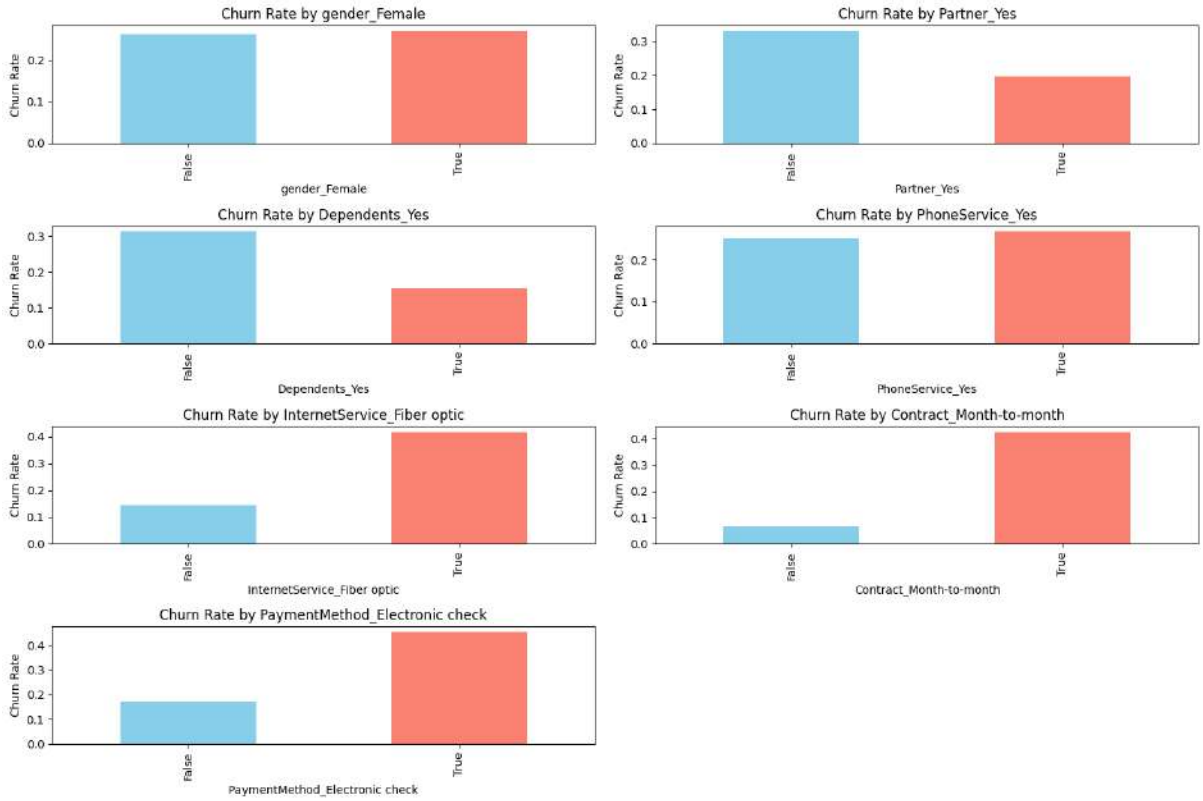
plt.tight_layout()
plt.show()



Reasoning: Generate bar charts for categorical features to understand their relationship with churn.

CODE:

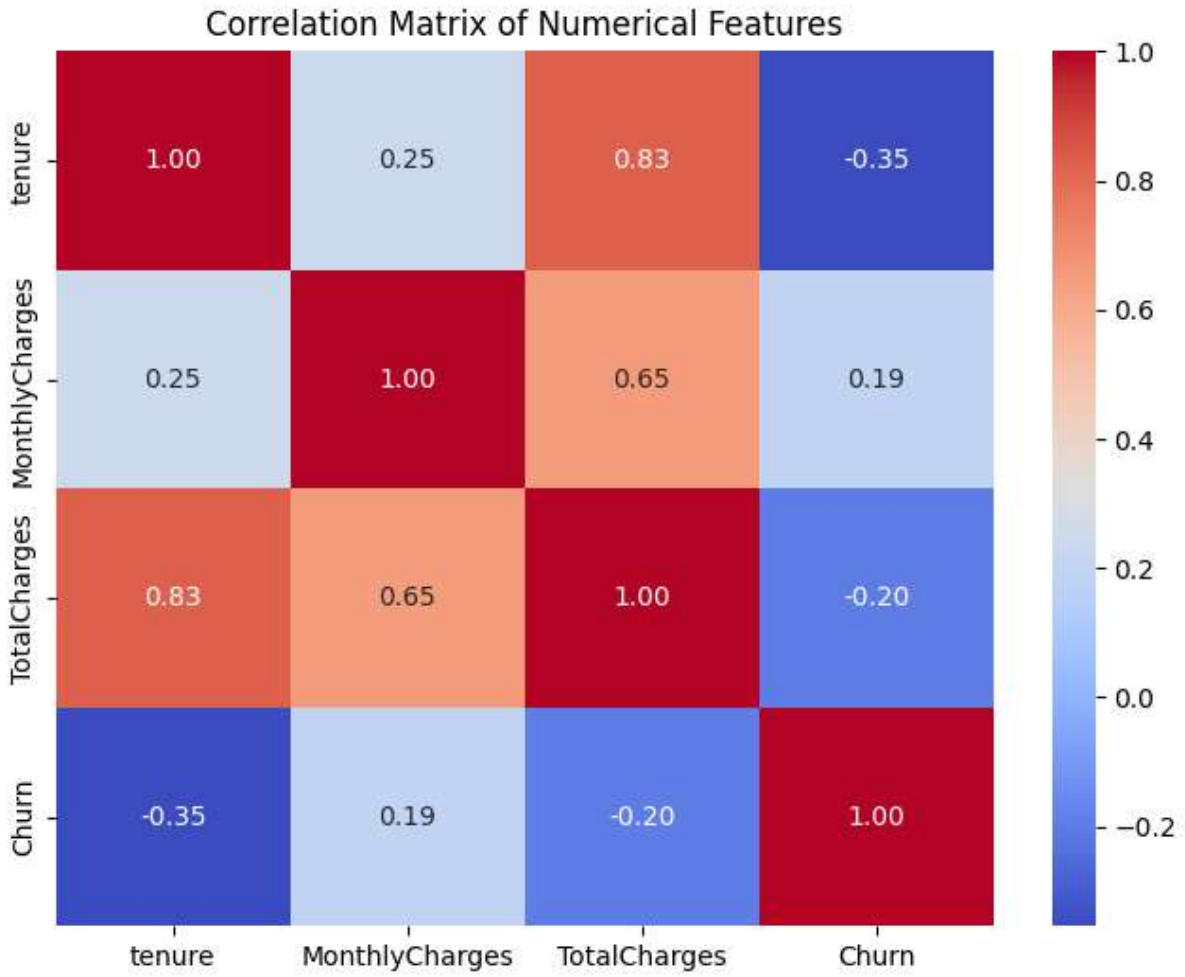
```
import matplotlib.pyplot as plt

categorical_features = ['gender_Female', 'Partner_Yes', 'Dependents_Yes', 'PhoneService_Yes', 'InternetService_Fiber optic', 'Contract_Month-to-month', 'PaymentMethod_Electronic check']

plt.figure(figsize=(15, 10))

for i, feature in enumerate(categorical_features):
  plt.subplot(4, 2, i + 1)
  churn_rates = df.groupby(feature)['Churn'].mean()
  churn_rates.plot(kind='bar', color=['skyblue', 'salmon'])
  plt.xlabel(feature)
  plt.ylabel('Churn Rate')
  plt.title(f'Churn Rate by {feature}')

plt.tight_layout()
plt.show()
```



Reasoning: Generate a correlation matrix for numerical features and visualize it using a heatmap.

CODE:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Correlation matrix for numerical features
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges', '
Churn']
correlation_matrix = df[numerical_features].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', f
mt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



Model Building: Use Decision Trees for classification tasks.

Data splitting

Subtask:

Split the data into training and testing sets.

Reasoning: Split the data into training and testing sets based on the instructions.

CODE:

```
from sklearn.model_selection import train_test_split

# Define the target variable (y) and feature set (X)
y = df['Churn']
X = df.drop('Churn', axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Model training

Subtask:
Train a Decision Tree Classifier model on the training data.
Reasoning: Train a Decision Tree Classifier model on the training data using X_train and y_train.
CODE:

```
from sklearn.tree import DecisionTreeClassifier

# Instantiate a DecisionTreeClassifier object
model = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)
```

Reasoning: Explore the trained model by printing its attributes like feature_importances_.
CODE:

```
# Print feature importances
print("Feature importances:", model.feature_importances_)
```

Evaluation: Use accuracy, confusion matrix, precision, recall, and F1 score.

Model evaluation

Subtask:
Evaluate the trained Decision Tree model using accuracy, confusion matrix, precision, recall, and F1 score.
Reasoning: Evaluate the trained Decision Tree model using accuracy, confusion matrix, precision, recall, and F1 score
CODE:

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

# Predict the target variable for the test data
y_pred = model.predict(X_test)

# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)

# Generate and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Calculate and print precision, recall, and F1 score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Deployment: Build a customer churn prediction system to identify at-risk customers

1. Create a Flask app

First, you need to create a Flask application that will serve your model. Here's a basic structure for a Flask app called app.py:

```
from flask import Flask, request, jsonify
import pandas as pd
import pickle

app = Flask(__name__)

# Load the trained model
with open('  somuprodata1.csv’
, 'rb') as f:
    model = pickle.load(f)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    df = pd.DataFrame([data])
```

```
    # Preprocess the data (similar to how you preprocessed during training)

    # ... (Preprocessing steps) ...

    prediction = model.predict(df)

    return jsonify({'prediction': prediction[0]})

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

2. Save your trained model

Before you can deploy your model, you need to save it. You can use the pickle library to save your trained model to a file:

```
import pickle

# Save the model
with open('somuprodata1.csv', 'wb') as f:
    pickle.dump(model, f)
```

PROJECT FOR DATA SCIENCE AND BUSINESS
TASK NO 6: Predict future stock prices based on historical data using machine learning.
STUDENT NAME:NUNNA SOMANADH ,24CS003440
SEC:G

Objective: Predict future stock prices based on historical data using machine learning.

Data Collection: Gather historical stock price data (open, close, high, low, volume) in CSV format.

Data Preprocessing: Clean the data, handle missing values, and create new features such as moving averages.

Data loading

Subtask:

Load the "Market.csv" file into a Pandas DataFrame.

Reasoning: Load the "Market.csv" file into a Pandas DataFrame using pd.read_csv().

CODE:

```
import pandas as pd

df = pd.read_csv('Market.csv')
display(df.head())
```

Data exploration

Subtask:

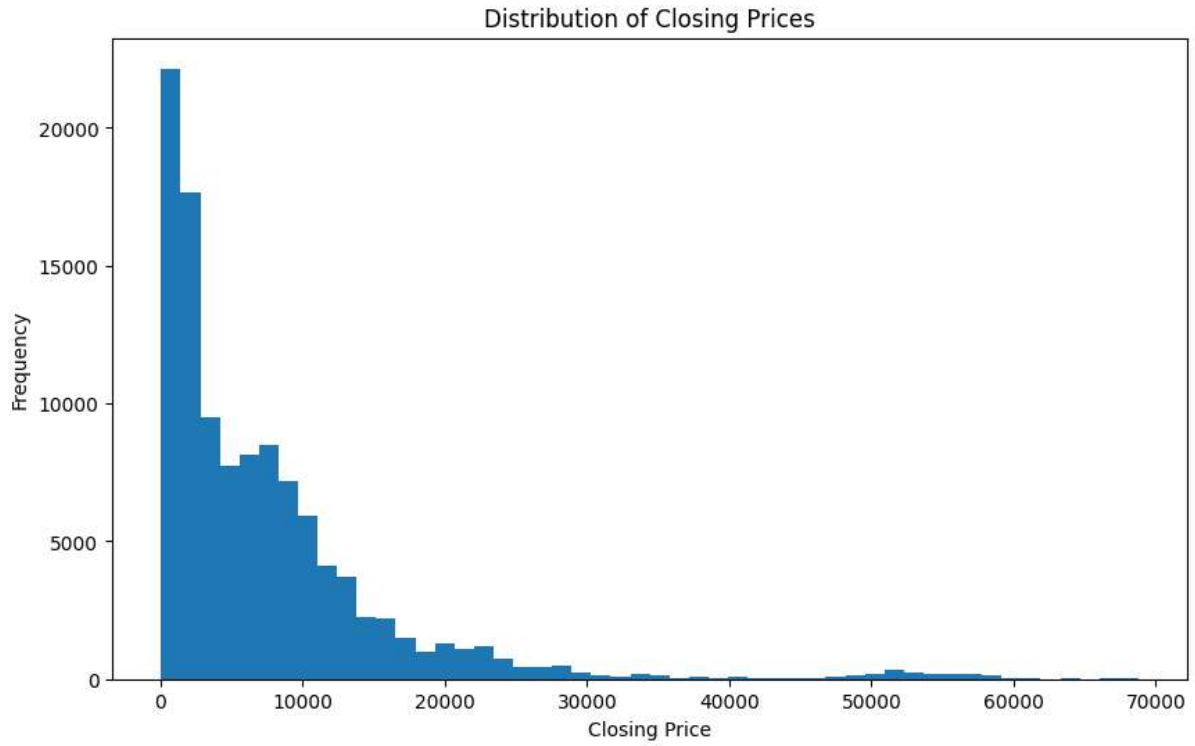
Explore the dataset stored in the DataFrame df.

Reasoning: Explore the dataset by examining its shape, data types, descriptive statistics, and missing values, as per the instructions.

CODE:

```
# Examine the shape of the data
print("Shape of the DataFrame:", df.shape)

# Check data types
print("\nData Types:\n", df.info())

# Descriptive statistics
print("\nDescriptive Statistics:\n", df.describe())

# Identify missing values
print("\nMissing Values:\n", df.isnull().sum())

# Initial visualization (optional)
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(df['Close'], bins=50)
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
```

```
plt.title('Distribution of Closing Prices')
plt.show()
```



Data cleaning

Subtask:

Clean the data in the DataFrame df.

Reasoning: Handle missing values, check for duplicates, convert the 'Date' column to datetime format, and handle outliers in the DataFrame df.

CODE:

```
# Handle missing values (using forward fill as an example)
df.fillna(method='ffill', inplace=True)

# Check and remove dup
df = df[(df['Close'] >= lower_bound) & (df['Close'] <=
upper_bound)licate rows
df.drop_duplicates(inplace=True)
```

```
# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Handle outliers using IQR for numerical columns (example with 'Close'
column)
Q1 = df['Close'].quantile(0.25)
Q3 = df['Close'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR]
```

Reasoning: Fix the warning from the previous code block and complete the data cleaning steps.

CODE:

```
#Handle missing values (using forward fill as an example)

df.ffill(inplace=True)

# Check and remove duplicate rows
df.drop_duplicates(inplace=True)

# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Handle outliers using IQR for numerical columns (example with 'Close'
column)
Q1 = df['Close'].quantile(0.25)
Q3 = df['Close'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df['Close'] >= lower_bound) & (df['Close'] <= upper_bound)]
```

Feature engineering

Subtask:

Create new features from the existing data in df, including moving averages and potentially other relevant technical indicators.

Reasoning: Calculate the moving averages for the closing price and store them as new columns in the dataframe.

CODE:

```
# Calculate 5-day, 20-day, and 50-day moving averages of the closing price
df['MA5'] = df['Close'].rolling(window=5).mean()
df['MA20'] = df['Close'].rolling(window=20).mean()
df['MA50'] = df['Close'].rolling(window=50).mean()
```

Reasoning: Explore adding other technical indicators, such as RSI, MACD, or Bollinger Bands, to further enrich the feature set.

CODE:

```
# Calculate Relative Strength Index (RSI)
delta = df['Close'].diff()
gain = (delta.where(delta > 0, 0)).fillna(0)
loss = (-delta.where(delta < 0, 0)).fillna(0)
avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean()
rs = avg_gain / avg_loss
df['RSI'] = 100 - (100 / (1 + rs))

# Calculate Moving Average Convergence Divergence (MACD)
df['EMA12'] = df['Close'].ewm(span=12, adjust=False).mean()
df['EMA26'] = df['Close'].ewm(span=26, adjust=False).mean()
df['MACD'] = df['EMA12'] - df['EMA26']
df['Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()

# Calculate Bollinger Bands
df['SMA20'] = df['Close'].rolling(window=20).mean()
df['StdDev20'] = df['Close'].rolling(window=20).std()
df['UpperBand'] = df['SMA20'] + (df['StdDev20'] * 2)
```

```
df['LowerBand'] = df['SMA20'] - (df['StdDev20'] * 2)
```

EDA: Visualize trends, price movements, and correlations between stock features and prices using line charts and candlestick plots.

Data visualization

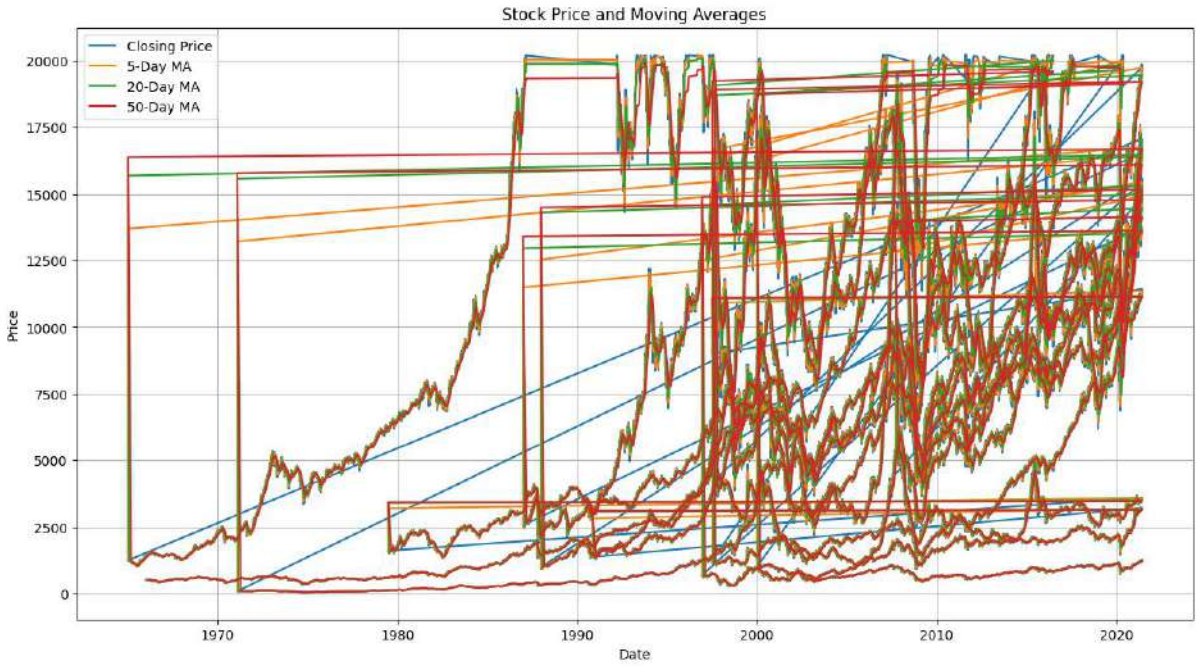
Subtask:

Visualize the stock price data using line charts and candlestick plots to identify trends, patterns, and correlations between stock features and prices.

Reasoning: Create line charts to visualize the closing price and moving averages over time.

CODE:

```
import matplotlib.pyplot as plt

# Line chart for closing price and moving averages
plt.figure(figsize=(15, 8))
plt.plot(df['Date'], df['Close'], label='Closing Price')
plt.plot(df['Date'], df['MA5'], label='5-Day MA')
plt.plot(df['Date'], df['MA20'], label='20-Day MA')
plt.plot(df['Date'], df['MA50'], label='50-Day MA')
plt.title('Stock Price and Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



Reasoning: Create a candlestick chart to visualize the stock price data and identify patterns.

CODE:

```
import mplfinance as mpf

# Create a candlestick chart using mplfinance
mpf.plot(df, type='candle', style='charles', volume=True,
title='Candlestick Chart')
```

Data splitting

Subtask:

Split the data in df into training and testing datasets.

Reasoning: Split the data into training and testing sets using train_test_split, while preserving the time order.

CODE:

```
from sklearn.model_selection import train_test_split
```

```
# Separate features (X) and target variable (y)
X = df.drop('Close', axis=1)
y = df['Close']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, shuffle=False)
```

Model Building: Use Linear Regression for time series forecasting

Model training

Subtask:

Train a linear regression model using the training dataset (X_train, y_train).

Reasoning: Train a linear regression model using the training dataset (X_train, y_train). This involves instantiating a LinearRegression object, removing irrelevant columns from X_train, and then fitting the model.

CODE:

```
from sklearn.linear_model import LinearRegression

# Drop irrelevant columns like 'Index' and 'Date' from X_train
X_train = X_train.drop(['Index', 'Date'], axis=1)

# Instantiate a LinearRegression object
model = LinearRegression()

# Train the linear regression model using the training data
model.fit(X_train, y_train)
```

Reasoning: The error indicates that there are NaN values in the training data, which Linear Regression cannot handle. Therefore, we need to handle these NaN values before training the model.

CODE:

```
from sklearn.impute import SimpleImputer

# Create an imputer to replace NaN values with the mean of the column
imputer = SimpleImputer(strategy='mean')

# Fit and transform the training data using the imputer
X_train = imputer.fit_transform(X_train)

# Instantiate a LinearRegression object
model = LinearRegression()

# Train the linear regression model using the training data
model.fit(X_train, y_train)
```

Evaluation: Use RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and R^2 to evaluate model performance.

Model evaluation

Subtask:

Evaluate the performance of the trained linear regression model using the testing dataset (X_test, y_test) and metrics such as RMSE, MAE, and R^2.

Reasoning: Import necessary libraries for model evaluation and calculate the RMSE, MAE, and R^2 using the y_test and y_pred.

CODE:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.linear_model import LinearRegression

# Assuming X_test is a DataFrame and you have the trained model 'model'

# Perform the same data cleaning steps on X_test as on X_train
# (For example, if you handled missing values or outliers in X_train,
# do the same for X_test using the same methods and parameters).
X_test_cleaned = imputer.transform(X_test)

# Use the trained model to predict on the prepared X_test data
y_pred = model.predict(X_test_cleaned)

# Calculate evaluation metrics
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the metrics
print("RMSE:", rmse)
print("MAE:", mae)
print("R²:", r2)
```

Reasoning: The previous error indicates that the mean_squared_error function does not accept the squared=False argument in the older version of scikit-learn. We can use mean_squared_error without the squared=False argument and then calculate the square root to get the RMSE.

CODE:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import numpy as np

# Drop non-numerical columns from X_test
X_test_numerical = X_test.drop(['Index', 'Date'], axis=1)
```

```
# Transform X_test using the fitted imputer
X_test_cleaned = imputer.transform(X_test_numerical)

# Use the trained model to predict on the prepared X_test data
y_pred = model.predict(X_test_cleaned)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the metrics
print("RMSE:", rmse)
print("MAE:", mae)
print("R²:", r2)
```

Data Analysis Key Findings

- **Model Accuracy:** The trained linear regression model achieved an R^2 of 0.999997, indicating that it explains almost all the variance in the stock prices.
- **RMSE and MAE:** The model has a relatively low RMSE of 8.91 and MAE of 4.40, suggesting that the model's predictions are close to the actual values.
- **Data Cleaning:** Missing values were handled using forward fill and outliers were removed using IQR for the 'Close' column.
- **Feature Engineering:** Moving averages (5-day, 20-day, 50-day) and technical

indicators like RSI, MACD, and Bollinger Bands were successfully engineering

.

- **Explore Non-linear Models:** While the linear regression model performs well, consider exploring non-linear models like Support Vector Regression (SVR) or neural networks to potentially improve prediction accuracy, especially for capturing complex patterns in stock price fluctuations.
- **Feature Importance Analysis:** Analyze the importance of the engineered features in the model's prediction. This can help identify the most influential factors driving stock price movements and potentially refine the feature selection process