Name	**Zunaira Hameed**
Tasks	**Intermediate Level**
Domain	**Data Analysis**
Company	**Codveda**

Overview:

This project involves performing three intermediate-level data analysis tasks using Python:

1. Linear Regression Analysis
2. Time Series Trend and Seasonality Detection
3. K-Means Clustering for Pattern Discovery

Tools used: pandas, numpy, scikit-learn, matplotlib, seaborn, statsmodels

Dataset: Stock prices (CSV file)

Task 1: Regression Analysis

Goal:

Predict the **closing price** of a stock based on other features such as date, volume, and symbol using **Linear Regression**.

Step 1: Import Dependencies

Basic data manipulation, visualization, and modeling libraries are imported. Warnings are also suppressed to avoid clutter.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.preprocessing import LabelEncoder

from warnings import filterwarnings

filterwarnings('ignore')
```

Step 2: Mounting Google drive

To access the dataset stored in Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

Step 3: Load and Read Dataset

Reading the stock prices CSV file using pandas.

```
data = pd.read_csv('/content/drive/MyDrive/Stock Prices Data Set.csv')
data.head()
```

Step 4: Handling missing and duplicate values

Checking for nulls.

Filling missing values using forward fill method.

Checking for duplicate records.

```
data.fillna(method='ffill', inplace=True)
```

Step 5: Encoding categorical variables

The symbol column is encoded to numeric values using LabelEncoder.

```
le = LabelEncoder()
```

```
data['symbol'] = le.fit_transform(data['symbol'])
```

Step 6: Data transformation

The date column is converted to ordinal format to use in numerical modeling.

```
data['date'] = pd.to_datetime(data['date'])
```

```
data['date_ordinal'] = data['date'].apply(lambda date: date.toordinal())
```

Step 7: Preparing features and labels

Independent variables are selected by dropping close and date. The target variable is close.

Step 8: Splitting Dataset

The data is split into training and testing sets (80% train, 20% test).

```
x_test, x_train, y_test, y_train = train_test_split(x, y, test_size=0.2, random_state=42)
```

Step 9: Training Linear Regression

Using scikit-learn’s LinearRegression to fit the model on training data.

```
model = LinearRegression()
```

model.fit(x_train, y_train)

Step 10: Evaluating the model

Intercept and coefficients are printed.

R-squared and Mean Squared Error (MSE) are used for performance evaluation.

r2 = r2_score(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

Task 2: Time Series Analysis

Goal:

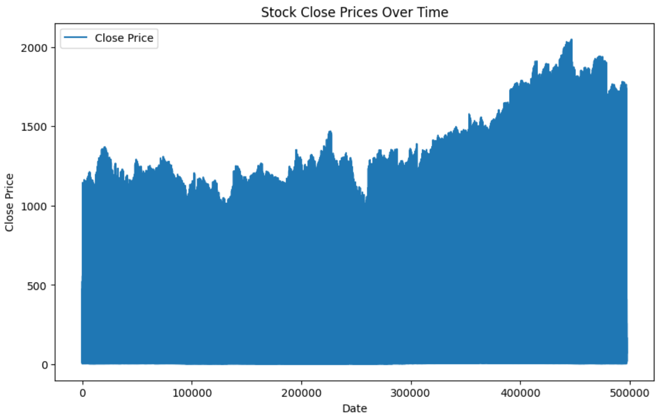
Detect **trend**, **seasonality**, and **residuals** in the stock closing prices and apply **moving average smoothing**.

Step 1: Plotting Time Series

Visualizing the closing prices over time to identify patterns.

```
# Plotting the close price over time
plt.figure(figsize=(10, 6))
plt.plot(data['close'], label="Close Price")
plt.title("Stock Close Prices Over Time")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.legend()
plt.show()
```

Stock close prices over time



Step 2: Time series decomposition

Using seasonal_decompose from statsmodels to split the time series into:

Trend

Seasonality

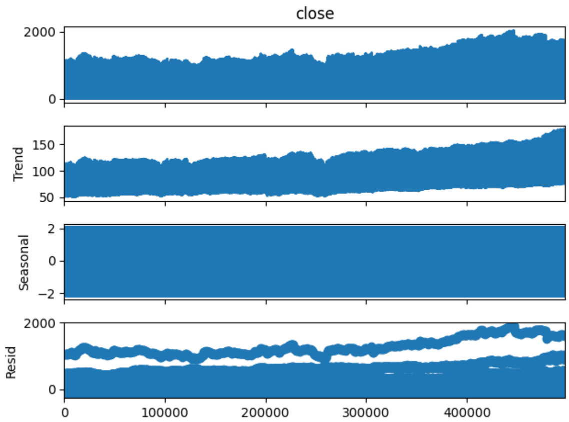
Residual (Noise)

```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(data['close'], model='additive', period=30)

decomposition.plot()
```

Decompose Close price

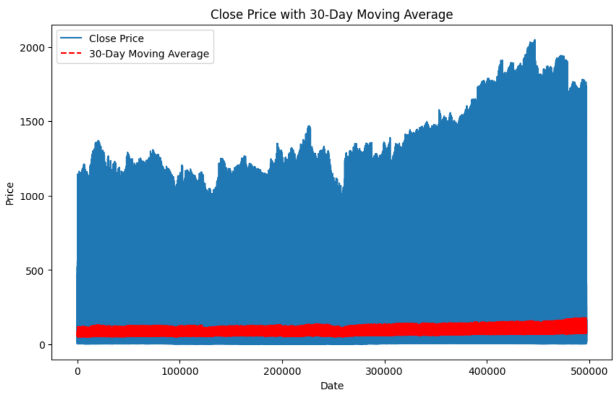


Step 3: Moving Average smoothing

A 30-day simple moving average is calculated and plotted to observe trend smoothing.

data['moving_avg'] = data['close'].rolling(window=30).mean()

Close Price with 30 Day Moving AVG



Task 3: Clustering Analysis (K-means)

Goal:

Group similar stock records using **K-Means Clustering** based on numerical features.

Step 1: Standardize the data

Only numeric columns are selected.

Missing values are handled using SimpleImputer.

Features are scaled using StandardScaler.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer # Import SimpleImputer

# Select only numeric features for scaling
numeric_features = data.select_dtypes(include=['number']).columns
numeric_data = data[numeric_features]

# Impute missing values using the mean strategy
imputer = SimpleImputer(strategy='mean') # Create an imputer instance
numeric_data = imputer.fit_transform(numeric_data) # Impute missing values

# Standardize the features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_data)
```

Step 2: Elbow method for optimal clusters

The elbow method is applied by plotting **inertia** for k=1 to 10 to decide the best number of clusters.

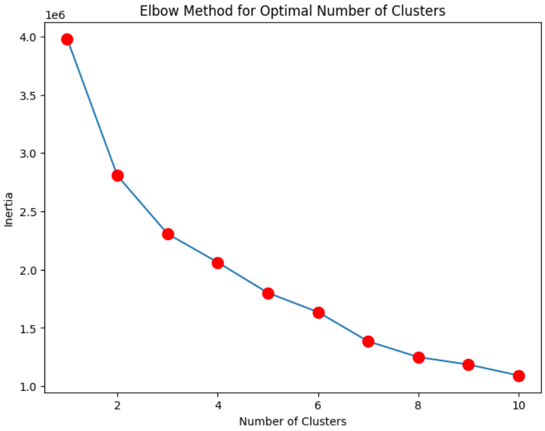
```
for k in range(1, 11):

    kmeans = KMeans(n_clusters=k, random_state=42)

    kmeans.fit(scaled_data)

    inertia.append(kmeans.inertia_)
```

Elbow Graph



Step 3: Applying K-means

Once optimal clusters are found (e.g., k=3), KMeans is applied and cluster labels are added to the original data.

kmeans = KMeans(n_clusters=3, random_state=42)

clusters = kmeans.fit_predict(scaled_data)

data['Cluster'] = clusters

Step 4: Visualizing clusters in 2D

Using **PCA** to reduce dimensions and plot clusters in 2D space for clear visualization.

from sklearn.decomposition import PCA

reduced_data = pca.fit_transform(scaled_data)

sns.scatterplot(x=reduced_data[:, 0], y=reduced_data[:, 1], hue=data['Cluster'])

K-means clusters