Project Documentation: Stock Prices Predictor Using Time Series Analysis

1. Introduction

Time series analysis is a statistical method used to analyze timeordered data points. In the context of stock prices, it allows us to model, understand, and predict future prices based on historical data. This project aims to implement a stock price predictor using various time series forecasting methods, including Moving Average, Exponential Smoothing, and ARIMA (Autoregressive Integrated Moving Average).

2. Objectives

To understand and apply time series analysis techniques for stock price prediction.
To compare the accuracy of different time series models.
To predict future stock prices based on historical data.
To explore the strengths and limitations of each forecasting model.

3. Prerequisites

Programming Language: Python
Libraries:
- pandas for data manipulation
- numpy for numerical operations
- matplotlib/seaborn for data visualization
- statsmodels for statistical models (including ARIMA)
- scikitlearn for model evaluation metrics
- yfinance or alpha_vantage for fetching stock data

4. Dataset

The dataset will consist of historical stock prices, typically including the following fields:
Date: The date of the trading day.
Open: The price of the stock at the opening of the trading day.
High: The highest price of the stock during the trading day.
Low: The lowest price of the stock during the trading day.
Close: The price of the stock at the close of the trading day.
Volume: The number of shares traded during the trading day.

Data Source: Stock price data can be obtained using the yfinance library or any other financial data provider like Alpha Vantage.

5. Methodology

5.1. Data Collection

Use the yfinance library to fetch historical stock data.
Store the data in a CSV file or directly manipulate it using pandas.

python
import yfinance as yf

Example: Fetching historical data for Apple Inc. (AAPL)
data = yf.download('AAPL', start='20100101', end='20230101')

5.2. Data Preprocessing
Handle missing data by imputing or dropping them.
Convert the 'Date' column to a datetime format and set it as the index.
Perform any necessary scaling or normalization.

python
import pandas as pd

Handling missing values
data = data.dropna()

Set Date as index
data['Date'] = pd.to_datetime(data.index)
data.set_index('Date', inplace=True)

5.3. Exploratory Data Analysis (EDA)
Visualize the stock price trends over time.
Analyze seasonality, trends, and patterns.

python
import matplotlib.pyplot as plt

Plotting the Closing Price
plt.figure(figsize=(14,7))
plt.plot(data['Close'], label='Close Price')
plt.title('Stock Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

5.4. Time Series Forecasting Models

5.4.1. Moving Average
Calculate the moving average of the stock prices over a specified window.

python
data['Moving_Avg'] = data['Close'].rolling(window=10).mean()

Plotting Moving Average
plt.plot(data['Close'], label='Close Price')
plt.plot(data['Moving_Avg'], label='Moving Average')

```
plt.legend()
plt.show()
```

5.4.2. Exponential Smoothing
Apply exponential smoothing to the data to give more weight to recent observations.

```
python
from statsmodels.tsa.holtwinters import ExponentialSmoothing

model = ExponentialSmoothing(data['Close'], trend='add', seasonal=None)
fit = model.fit()
data['Exp_Smoothing'] = fit.fittedvalues

 Plotting Exponential Smoothing
plt.plot(data['Close'], label='Close Price')
plt.plot(data['Exp_Smoothing'], label='Exponential Smoothing')
plt.legend()
plt.show()
```

5.4.3. ARIMA Model
Fit an ARIMA model to the data for more sophisticated forecasting.

```
python
from statsmodels.tsa.arima.model import ARIMA

 Differencing to make the series stationary
data['Diff'] = data['Close'].diff().dropna()

 ARIMA Model
model = ARIMA(data['Diff'].dropna(), order=(1, 1, 1))
fit = model.fit()
data['ARIMA_Forecast'] = fit.predict(start=100, end=len(data)1)

 Plotting ARIMA forecast
plt.plot(data['Close'], label='Close Price')
plt.plot(data['ARIMA_Forecast'], label='ARIMA Forecast')
plt.legend()
plt.show()
```

5.5. Model Evaluation
Evaluate the models using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

```
python
from sklearn.metrics import mean_absolute_error, mean_squared_error

 Calculate MAE, MSE, RMSE for each model
mae = mean_absolute_error(data['Close'].iloc[1:], data['ARIMA_Forecast'].iloc[1:])
mse = mean_squared_error(data['Close'].iloc[1:], data['ARIMA_Forecast'].iloc[1:])
```

```
rmse = mse0.5

print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')
```

6. Results and Discussion
Present the forecasting results with visualizations and numerical evaluations.
Discuss the accuracy and applicability of each model.
Compare and contrast the performance of Moving Average, Exponential Smoothing, and ARIMA.

7. Conclusion
Summarize the key findings from the analysis.
Suggest possible improvements and extensions, such as integrating machine learning models or incorporating additional features (e.g., macroeconomic indicators).