1. Introduction

The Stock Prices Predictor project is designed to analyze historical stock prices using time series analysis techniques and predict future stock prices. The project leverages advanced statistical methods, including ARIMA modeling, to identify trends, seasonal patterns, and other characteristics in the data to make accurate predictions.

2. Prerequisites

Before running the script, ensure you have the following software and libraries installed:

Python 3.6+: The code is written in Python and requires a version of Python 3.6 or higher.
Libraries:
- numpy
- pandas
- matplotlib
- statsmodels
- pmdarima
- scikitlearn

You can install the required Python libraries using pip:

```
bash
pip install numpy pandas matplotlib statsmodels pmdarima scikitlearn
```

3. Data

The project requires a CSV file containing historical stock prices. The CSV file should have at least the following columns:

Date: The date of the stock price record.
Close: The closing price of the stock on that date.

Ensure that the file is named aapl.csv and placed in the same directory as the script, or adjust the file path in the script accordingly.

4. Project Structure

The project is organized into the following steps:

1. Importing Libraries: All necessary Python libraries are imported.
2. Loading and Preprocessing Data: The CSV file is loaded into a DataFrame, and preprocessing steps are applied to handle missing data and set the date as the index.
3. Stationarity Check: The DickeyFuller test is used to determine if the time series is stationary, which is a key assumption for ARIMA modeling.
4. Time Series Decomposition: The series is decomposed into its trend, seasonal, and residual components to understand its underlying structure.
5. Auto ARIMA Model: The auto_arima function automatically identifies the best ARIMA model parameters (p, d, q) based on statistical criteria.
6. Model Training and Forecasting: The model is trained on the historical data, and predictions are made for the test set and future dates.

7. Evaluation and Visualization: The predictions are evaluated using error metrics, and various plots are generated to visualize the results.

5. Code Explanation

5.1 Importing Libraries

```
python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

numpy: Used for numerical computations.
pandas: Provides data manipulation capabilities.
matplotlib: Used for data visualization.
statsmodels: Contains statistical models and tests.
pmdarima: Provides automated ARIMA model selection.
sklearn.metrics: Used for evaluating prediction accuracy.

5.2 Loading and Preprocessing Data

```
python
df = pd.read_csv(aapl.csv, parse_dates=[Date])
df = df[[Date, Close]]
df.set_index(Date, inplace=True)
df.sort_index(inplace=True)
df[Close].fillna(method=ffill, inplace=True)
```

The CSV file is loaded into a DataFrame, and the Date column is parsed as a date object.
Only the Date and Close columns are retained for analysis.
The Date column is set as the index, and the DataFrame is sorted by date.
Missing values in the Close column are filled using the forward fill method.

5.3 Stationarity Check

```
python
def test_stationarity(timeseries):
    result = adfuller(timeseries)
    print(ADF Statistic: %f % result[0])
    print(pvalue: %f % result[1])
    print(Critical Values:)
    for key, value in result[4].items():
        print(\t%s: %.3f % (key, value))

    return result[1]
```

```
p_value = test_stationarity(df[Close])
```

The test_stationarity function performs the Augmented DickeyFuller test to check if the series is stationary.
If the pvalue is greater than 0.05, the series is considered nonstationary, and differencing is applied.

5.4 Time Series Decomposition

```
python
decomposition = seasonal_decompose(df[Close], model=multiplicative, period=30)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

The series is decomposed into trend, seasonal, and residual components using multiplicative decomposition.
This helps in understanding the underlying structure of the time series.

5.5 Auto ARIMA Model

```
python
model = auto_arima(df[Close],
                   start_p=1, start_q=1,
                   test=adf,   Use ADF test to find optimal d
                   max_p=3, max_q=3,
                   m=1,              Frequency of series
                   d=None,           Let model determine d
                   seasonal=False,   No Seasonality
                   trace=True,       Print status
                   error_action=ignore,
                   suppress_warnings=True,
                   stepwise=True)
```

The auto_arima function selects the best ARIMA model based on statistical criteria like AIC.
The function tests various combinations of p, d, q parameters and selects the one with the lowest AIC.

5.6 Model Training and Forecasting

```
python
train_size = int(len(df) * 0.8)
train, test = df.iloc[:train_size], df.iloc[train_size:]

model.fit(train[Close])
forecast, conf_int = model.predict(n_periods=len(test), return_conf_int=True)
```

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

The ARIMA model is trained on the training set and used to predict the test set.
Forecasted values and confidence intervals are generated.

5.7 Evaluation and Visualization

```
python
plt.figure(figsize=(10, 6))
plt.plot(train[Close], label=Training Data)
plt.plot(test[Close], label=Test Data)
plt.plot(forecast_series, label=Predicted Data)
plt.fill_between(lower_series.index, lower_series, upper_series, color=k, alpha=.15)
plt.title(Apple Stock Price Prediction)
plt.xlabel(Date)
plt.ylabel(Close Price)
plt.legend()
plt.show()

mae = mean_absolute_error(test[Close], forecast_series)
rmse = np.sqrt(mean_squared_error(test[Close], forecast_series))
```

Predictions and confidence intervals are plotted along with the actual data.
The Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are calculated to evaluate model performance.

5.8 Future Forecasting

```
python
future_forecast, future_conf_int = model.predict(n_periods=30, return_conf_int=True)
future_forecast_series = pd.Series(future_forecast, index=pd.date_range(start=test.index[1] +
pd.Timedelta(days=1), periods=30, freq=B))
```

The model forecasts future stock prices for the next 30 business days.
The future forecast and confidence intervals are plotted to visualize the predictions.

6. Usage

1. Prepare the Data: Ensure the CSV file with historical stock prices is correctly formatted and placed in the script directory.
2. Run the Script: Execute the script in a Python environment. The script will automatically load the data, preprocess it, perform stationarity tests, decompose the time series, fit the ARIMA model, and generate forecasts.
3. View the Output: The script will produce various plots, including the historical stock prices, decomposed components, predicted prices, and future forecasts. Error metrics will also be displayed to evaluate the models performance.

7. Conclusion

This Stock Prices Predictor project provides a comprehensive framework for time series analysis and forecasting using historical stock data. By leveraging advanced techniques such as ARIMA modeling and decomposition, the project offers accurate and insightful predictions that can be used for various financial analysis tasks.

For further customization or improvement, you can experiment with different models, adjust the preprocessing steps, or incorporate additional data sources.