

BOĞAZİÇİ ÜNİVERSİTESİ

STATISTICAL LEARNING FOR DATA MINING
IE582

STOCK PRICE FORECASTING PROJECT

Group 11
Ecesu Bilgin
2023708012

Introduction

In the golden age of technology and internet, the economy is involved in any second of our lives. Currently, rather than only considering their own money usage and invest based on traditional methods, everyone can invest their money on stock market. It is an interesting area which becomes more popular day by day, people follow the updates daily and read the predictions of popular economy magazines. While this method is riskier than the traditional methods, it can make one person gain much more money if one follows good steps. However, this method is also dependent on many factors like others; such as politics, natural disasters, wars, pandemics, basically all factors that affect humans. These factors are hard to predict, yet, machine learning can be utilized for predictions based on the past data and trends.

In this project, ARIMA method is used for making daily predictions for BIST 30 companies. Turkey's stock market history traces back to the days of Ottoman Empire. BIST30 contains 30 biggest companies based on Borsa Istanbul's index. These companies have the highest market share and their stocks can present the overall economic situation of Turkey. BIST 30 group's stock values has greatly evolved, proportionate to the world trends. Below there is a figure on SP 500 annual stock returns and a figure of volume and price of BIST30 group of Turkey.

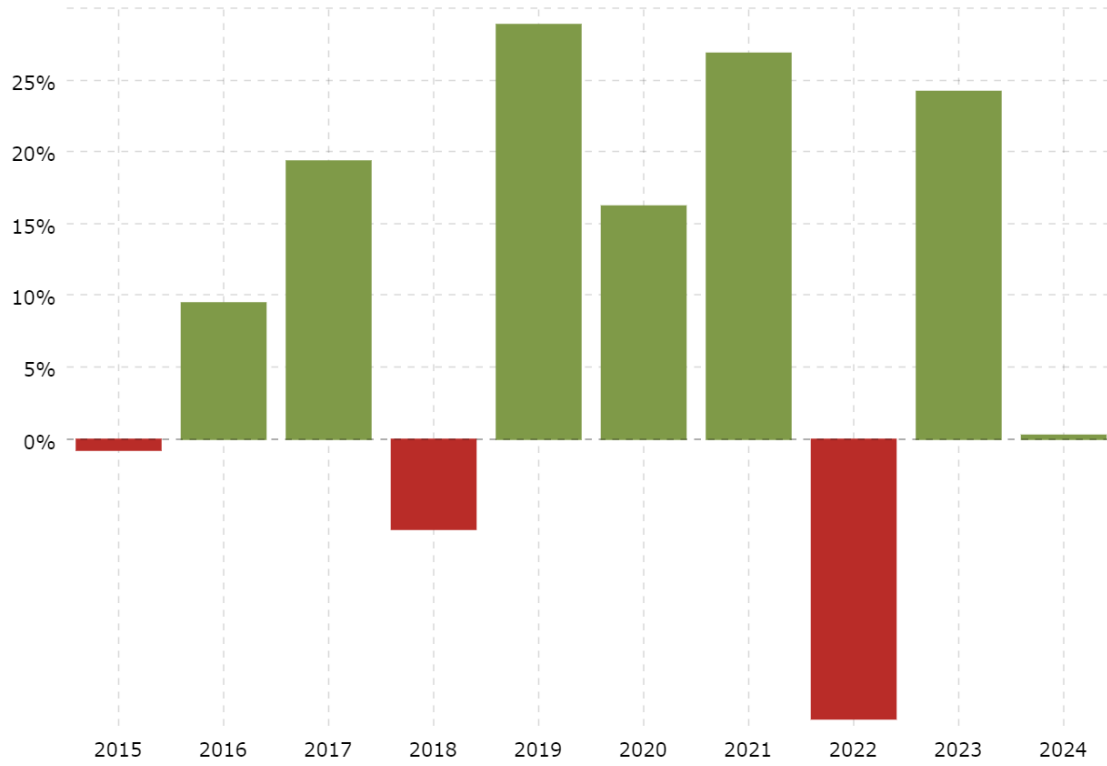


Figure 1 (S&P 500 historical annual returns, MacroTrends,2023)

Published on Investing.com, 16/Jan/2024 - 11:35:20 GMT, Powered by TradingView.
BIST 30, Türkiye, İstanbul:XU030, M

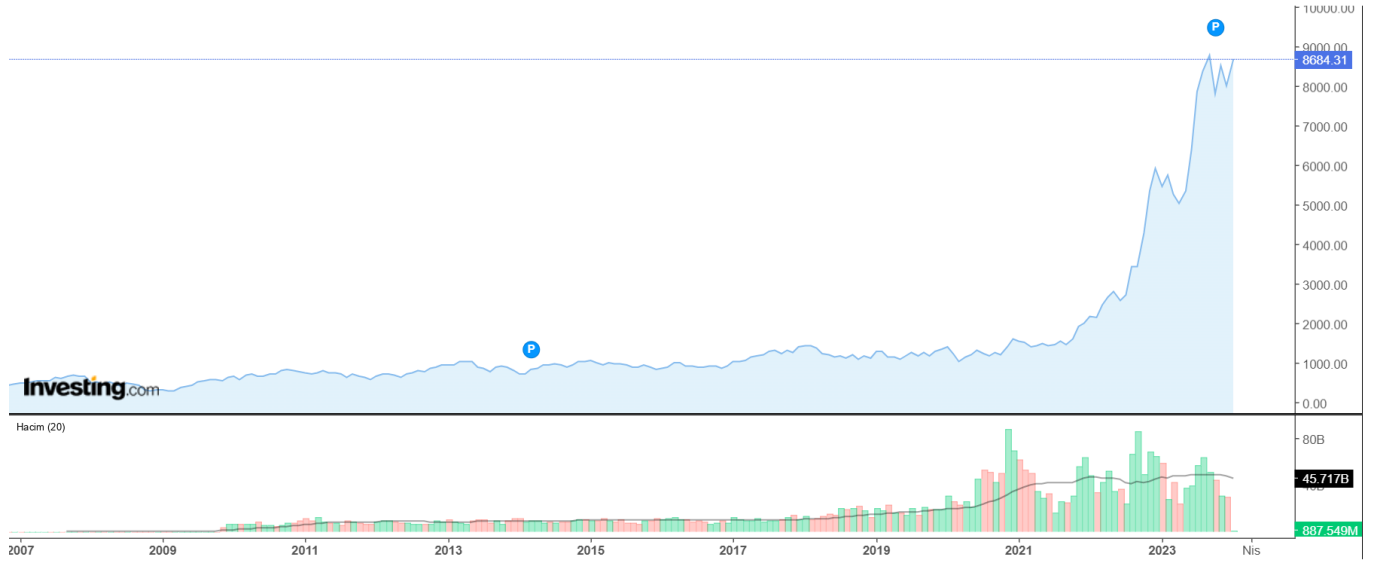


Figure 2 (BİST 30 Endeksi investing.com, 2023)

Literature survey

Many machine learning methods were utilized for stock prediction from a time series data. A literature survey highlights several methods like Support Vector Machines(SVM), Random forest and decision trees, linear regression, Artificial Neural Network (ANN) and Autoregressive Integrated Moving Average (ARIMA) (Vijh et al., 2023). In this project, three models are considered.

- 1- **Linear regression:** It is the most basic statistical model, thus, a good starting point and ideal for comparing purposes. In some studies, it gave a better result than SVM model. (Panwar et al, 2021)
- 2- **Random forest:** It is a simple method for application and can work well with both regression and classification, in some studies it performs comparably with ANN method in stock prediction, yet ANN method was slightly better. (Vijh et al., 2023)
- 3- **ARIMA:** It is widely used for prediction with time prediction and it outperformed ANN and similar neural network methods which are known as the best in some studies (Ariyo et al. 2014).

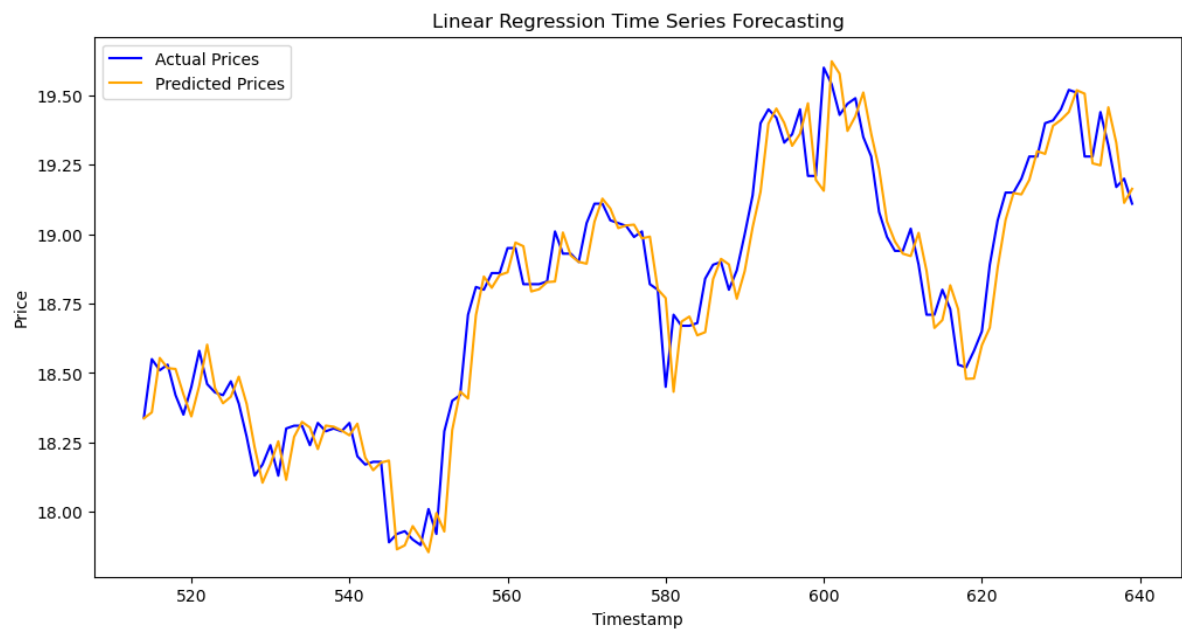
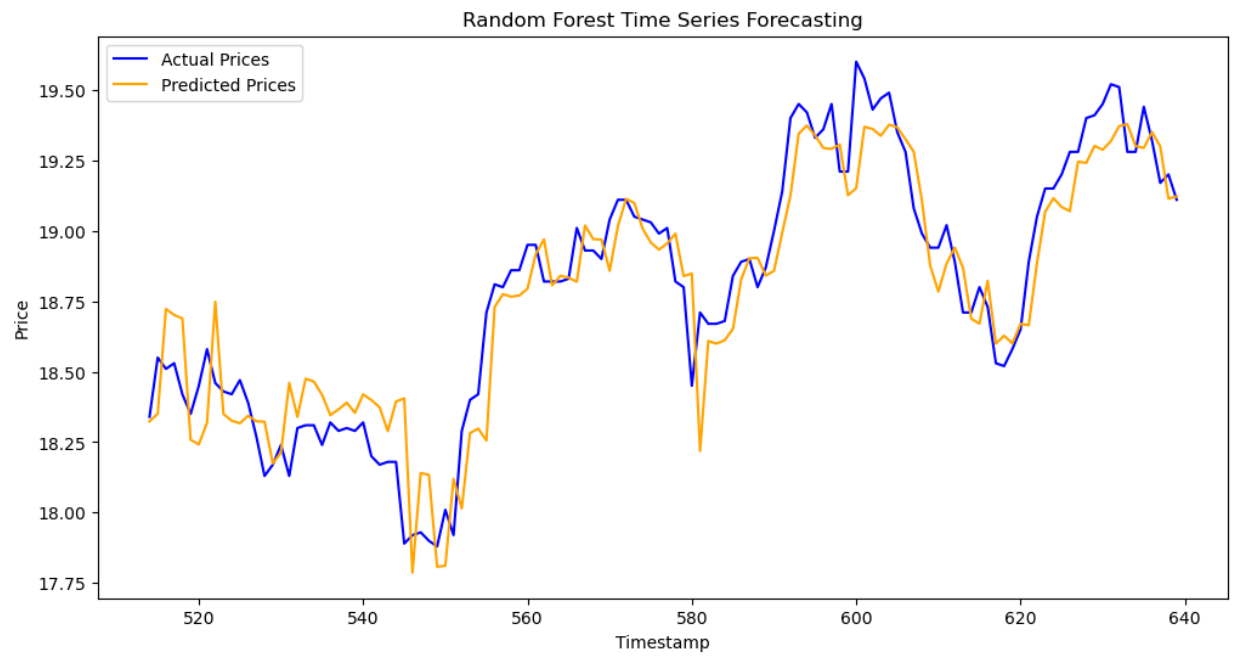
What is ARIMA?

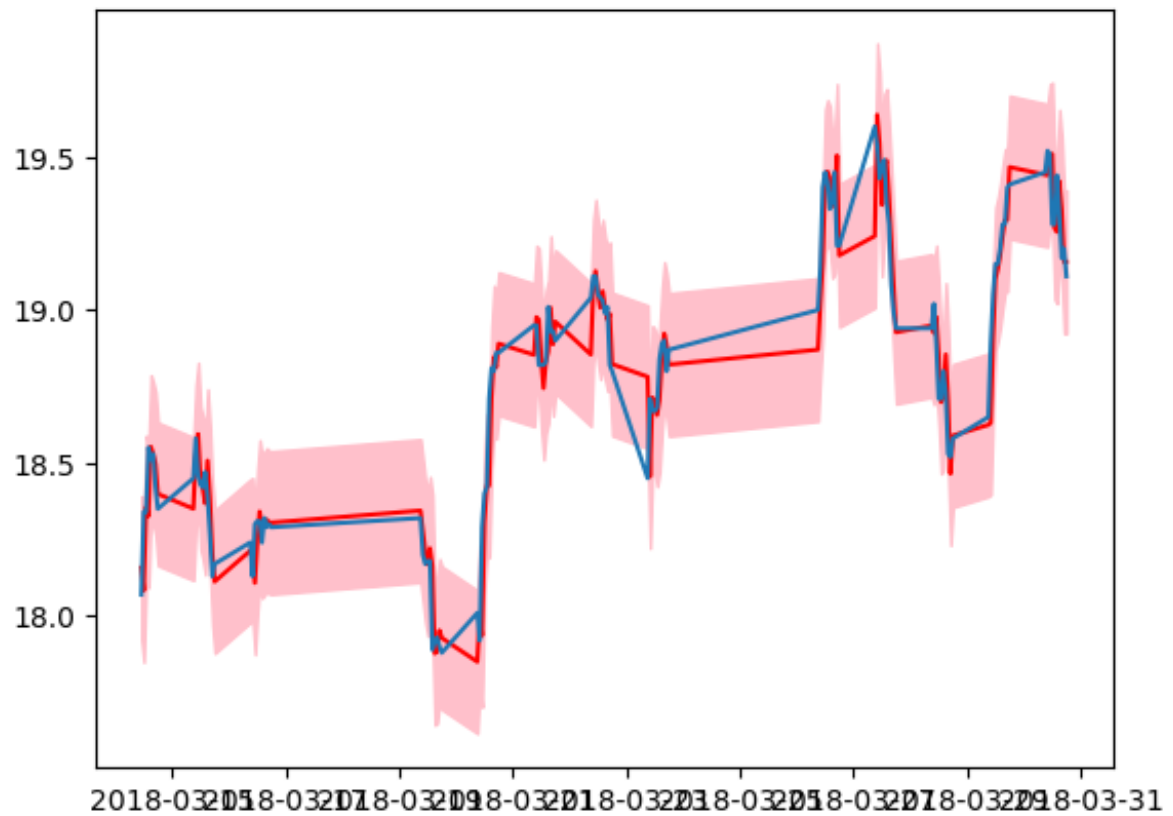
ARIMA has 3 parts and parameters such as AR(p), I(d) and MA(q). AR stands for autoregression, meaning that this model uses its past data to predict for future values. I part is for replacing the data with its differentiated form. If the data values aren't stationary (independent from time) already, then data is differentiated to get rid of time dependence. MA stands for moving average, this method approaches the data with moving window approach. It takes account that the last q values' error terms to calculate the next values (Kotu & Deshpande, 2019).

Model Selection & Parameter tuning

By the light of the information gathered from the literature survey, it was expected that ARIMA would demonstrate a better performance. Thus, I firstly took time and tried to evaluate the best parameters for ARIMA, in order to make a good comparison of ARIMA method with others. Firstly, the model is applied on the first company in the data set, THYAO. A time series cross validation is applied. The model is trained until a timestamp and the error between actual values and the predictions is calculated. For error calculation, both RMSE and MWAPE errors and considered in the following parts of the project; yet, in this part only RMSE is used since they would give similar results. Many parts of the complete past data is taken and different parameters are tried to determine which parameter set works well with all of the parts, i.e. different scenarios, to make the model more robust. Only 1 differentiation was enough to make the model stationary so “d” value is kept at 1. Choosing “p,d,q” values as 2,1,1 and 30,1,1 gave the overall best results; 2,1,1 model worked much faster than the other, but 30,1,1 is found to better if the data is supposed to be changing fastly. The 2,1,1 model would give more linear predictions and could underpredict in some cases.

After selecting the parameters; linear regression, RF and ARIMA models are applied to predict the first datasets last 80% values. Again, THYAO data is taken only. The results are given below.

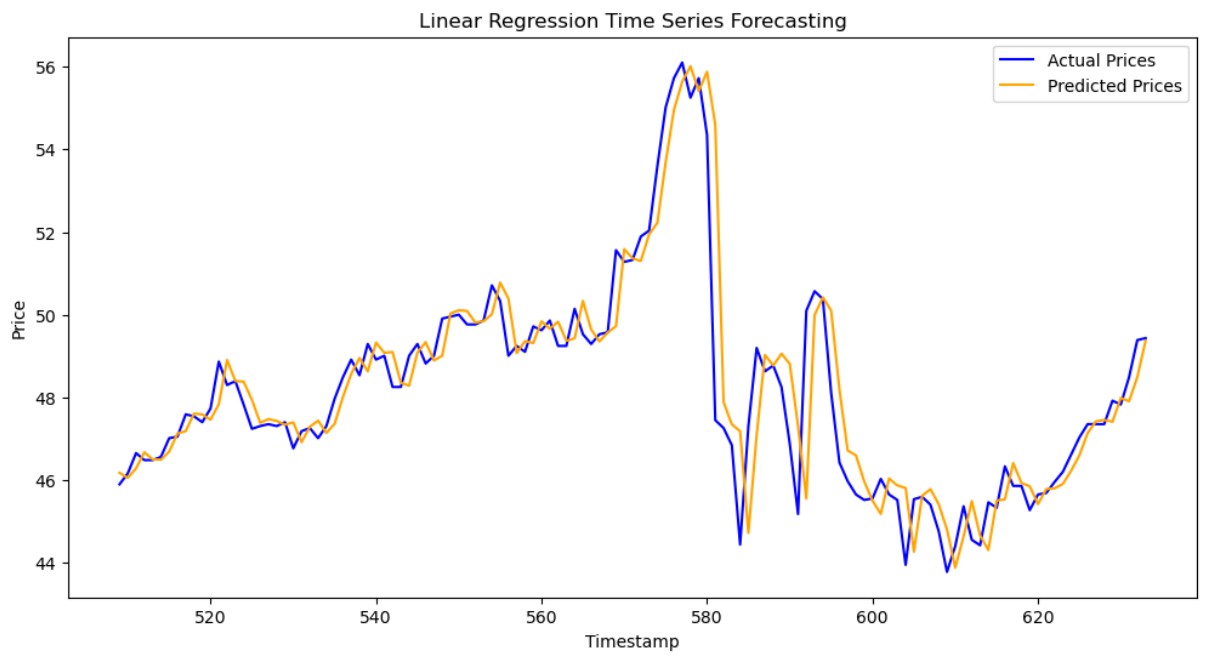
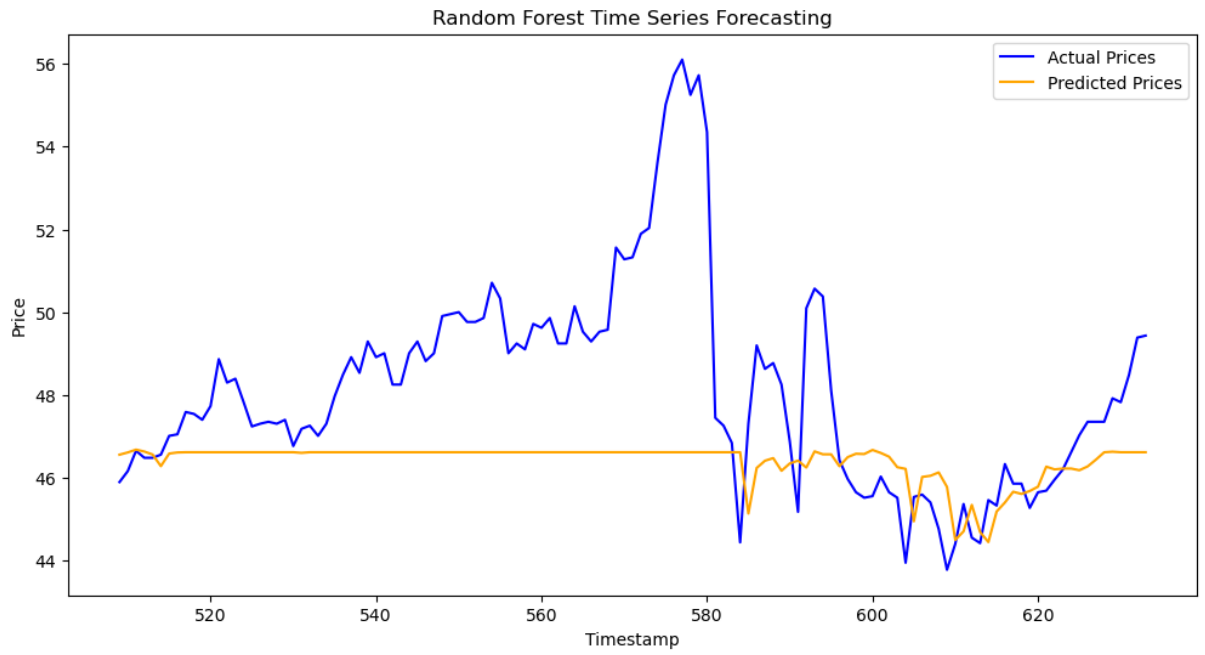


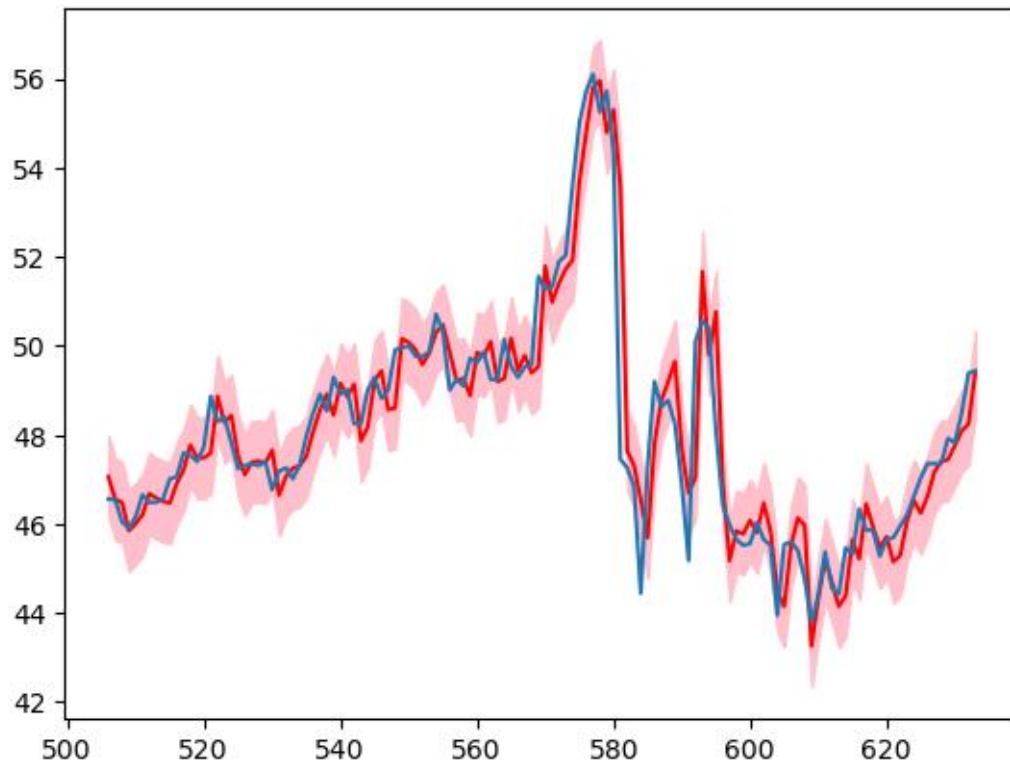


	RF	Linear Reg.	ARIMA
RMSE	0.1599	0.1208	0.1162
MWAPE	0.0067	0.0047	0.0045

As it can be seen, ARIMA model worked the best. Then, the models are tested on ARCLK and with data between 20210927 – 20211226.

The results are as follows.

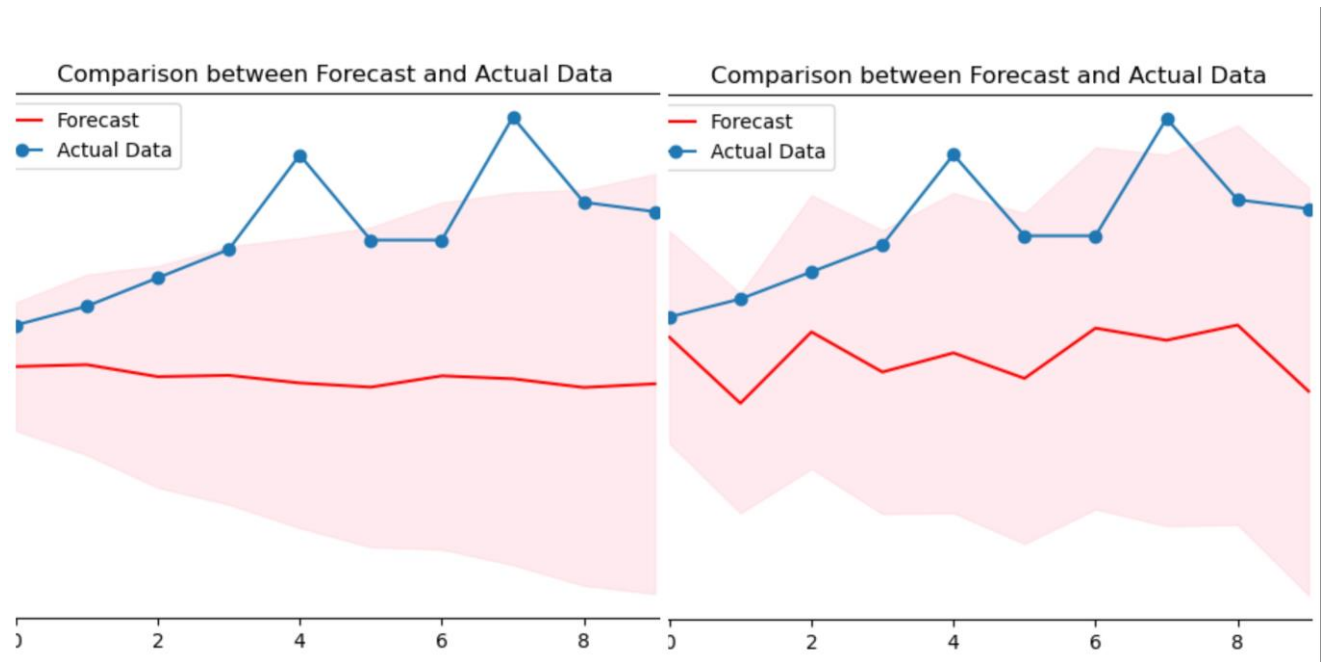




	RF	Linear Reg.	ARIMA
RMSE	2.8936	1.0828	0.9260
MWAPE	0.0412	0.0131	0.0119

ARIMA method consistently worked better with a different company and different timestamps. With these results, I decided to work with this model.

Then, I took the last dataset as test data and the previous ones as train data. When the model is applied to all the companies (with 30,1,1 parameters), it seem to be not working really good with some companies. To solve this problem, Yahoo Finance data is also utilized. The data provided was at timestamps at hours, and Yahoo Finance data was at half hours. When these datas are combined, the dataset is doubled and the time interval between data points is decreased. In some cases, it improved the prediction performance. For example, GARAN had a RMSE value of 0.8448 and the addition of Yahoo Finance data lowered this to 0.7789. Before and after graphs are given below.



If the results are still not satisfying, the parameters are adjusted. Usually, if the model keeps underpredicting, changing the parameters to 30,1,3 improved the results. Changing the p (AR) parameter didn't have a much affect at this scenario, yet, increasing q (Moving Average) part improved the model since it takes a higher number of prediction errors into account. Also, in some companies 30,1,1 model gave good results in the training stage but in the submission stage they had high errors and the model seemed to be overpredicting. In these cases, the model is adjusted to 2,1,1.

Conclusion

ARIMA is a promising method on the topic of time series stock price forecasting. Yet, economy and stock prices may fluctuate highly in Turkey. Daily prediction is based on the previous data, yet, if a political or another important news occurs in that day, stock prices get extremely affected. Those kind of news can be used to adjust the predictions with building another model

or manually. Still, machine learning methods are really beneficial for giving an insight.

References:

Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014, March). Stock price prediction using the ARIMA model. In 2014 UKSim-AMSS 16th international conference on computer modelling and simulation (pp. 106-112). IEEE.

BİST 30 Endeksi (XU030): BİST 30 hisse - investing.com. Investing.com Türkiye. (n.d.). <https://tr.investing.com/indices/ise-30>

Kotu, V., & Deshpande, B. (2019). Time series forecasting. Data Science; Elsevier: Amsterdam, The Netherlands, 395-445.

Panwar, B., Dhuriya, G., Johri, P., Yadav, S. S., & Gaur, N. (2021, March). Stock market prediction using linear regression and SVM. In 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE) (pp. 629-631). IEEE.

S&P 500 historical annual returns. MacroTrends,2023. <https://www.macrotrends.net/2526/sp-500-historical-annual-returns>

Vijh, M., Chandola, D., Tikkiwal, V. A., & Kumar, A. (2020). Stock closing price prediction using machine learning techniques. Procedia computer science, 167, 599-606.

APPENDIX:

Code: <https://github.com/BU-IE-582/fall-23-ecesubilgin/blob/main/files/projectIE582-EcesuBilgin.ipynb>

For ARIMA codes, I have used DataCamp's guide to ARIMA.

<https://campus.datacamp.com/courses/arima-models-in-python/>

ChatGPT answers

--my data has an index of timestamp and a single column of 'price' . can you write me a code with random forest

ChatGPT

Certainly! Below is an example Python code using the scikit-learn library to implement a Random Forest model for time series forecasting with your data. Make sure you have the required libraries installed by running `pip install pandas scikit-learn` in your terminal.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt

# Load your time series data
# Assuming 'timestamp' is the index and 'price' is the target variable
# Replace 'your_data.csv' with your actual file or data loading method
data = pd.read_csv('your_data.csv', parse_dates=['timestamp'], index_col='timestamp')

# Feature engineering: Adding lag features
for i in range(1, 11):
    data[f'lag_{i}'] = data['price'].shift(i)

# Drop rows with NaN values due to lag features
data = data.dropna()

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]

# Separate features and target variable
X_train, y_train = train_data.drop('price', axis=1), train_data['price']
X_test, y_test = test_data.drop('price', axis=1), test_data['price']

# Create a Random Forest Regressor
rf_model = Pipeline([
```

```

('scaler', StandardScaler()), # Standardize the data
('random_forest', RandomForestRegressor(n_estimators=100, random_state=42))
])

```

```

# Train the model
rf_model.fit(X_train, y_train)

```

```

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

```

```

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

```

```

# Plotting actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, y_test, label='Actual Prices', color='blue')
plt.plot(test_data.index, y_pred, label='Predicted Prices', color='orange')
plt.title('Random Forest Time Series Forecasting')
plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.legend()
plt.show()

```

can you write me a linear regression model

ChatGPT

Certainly! Here's an example of a simple linear regression model using the same lagged features for predicting the next day's stock prices:

```

python
Copy code
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

```

```

# Load your time series data
# Assuming 'timestamp' is the index and 'price' is the target variable
# Replace 'your_data.csv' with your actual file or data loading method
data = pd.read_csv('your_data.csv', parse_dates=['timestamp'], index_col='timestamp')

```

```

# Feature engineering: Adding lag features
for i in range(1, 11):
    data[f'lag_{i}'] = data['price'].shift(i)

```

```

# Drop rows with NaN values due to lag features
data = data.dropna()

```

```

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]

# Separate features and target variable
X_train, y_train = train_data.drop('price', axis=1), train_data['price']
X_test, y_test = test_data.drop('price', axis=1), test_data['price']

# Create a Linear Regression model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = linear_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plotting actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, y_test, label='Actual Prices', color='blue')
plt.plot(test_data.index, y_pred, label='Predicted Prices', color='orange')
plt.title('Linear Regression Time Series Forecasting')
plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.legend()
plt.show()

```

This code uses scikit-learn's LinearRegression model to create a linear regression model. Just like with the Random Forest example, X_train and X_test represent the lagged values of the 'price' column, and y_train and y_test are the corresponding target values.

Feel free to adjust the lag range and other parameters based on your specific needs and dataset characteristics.

User

in my data column has no name, can you fix the places where you wrote "price". it doesnt identify it

ChatGPT

Certainly! If your data column doesn't have a specific name, you can modify the code to handle the target column dynamically. Here's the updated code:

```

python
Copy code
import pandas as pd
from sklearn.linear_model import LinearRegression

```

```

from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load your time series data
# Assuming 'timestamp' is the index and your target column is unnamed
# Replace 'your_data.csv' with your actual file or data loading method
data = pd.read_csv('your_data.csv', parse_dates=['timestamp'], index_col='timestamp')

# Identify the target column dynamically
target_column = data.columns[0] # Assumes the target column is the first column; adjust as
needed

# Feature engineering: Adding lag features
for i in range(1, 11):
    data[f'lag_{i}'] = data[target_column].shift(i)

# Drop rows with NaN values due to lag features
data = data.dropna()

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]

# Separate features and target variable
X_train, y_train = train_data.drop(target_column, axis=1), train_data[target_column]
X_test, y_test = test_data.drop(target_column, axis=1), test_data[target_column]

# Create a Linear Regression model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = linear_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plotting actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, y_test, label='Actual Prices', color='blue')
plt.plot(test_data.index, y_pred, label='Predicted Prices', color='orange')
plt.title('Linear Regression Time Series Forecasting')
plt.xlabel('Timestamp')
plt.ylabel(target_column) # Use the target column name for the y-axis label
plt.legend()

```

```
plt.show()
```

In this modified code, the target column is identified dynamically using `data.columns[0]`, assuming the target column is the first column in your dataset. Adjust this line if your target column is located elsewhere.

User

AttributeError: 'Series' object has no attribute 'columns'

ChatGPT

I apologize for the confusion. The error you're encountering indicates that the object is a Series, not a DataFrame, and hence, it doesn't have a 'columns' attribute. Let's make the necessary adjustments:

python

Copy code

```
import pandas as pd
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error
```

```
import matplotlib.pyplot as plt
```

```
# Load your time series data
```

```
# Assuming 'timestamp' is the index and your target column is unnamed
```

```
# Replace 'your_data.csv' with your actual file or data loading method
```

```
data = pd.read_csv('your_data.csv', parse_dates=['timestamp'], index_col='timestamp')
```

```
# Identify the target column dynamically
```

```
target_column = data.columns[0] # Assumes the target column is the first column; adjust as needed
```

```
# Convert the target column to numeric
```

```
data[target_column] = pd.to_numeric(data[target_column], errors='coerce')
```

```
# Feature engineering: Adding lag features
```

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

```
    data[f'lag_{i}'] = data[target_column].shift(i)
```

```
# Drop rows with NaN values due to lag features
```

```
data = data.dropna()
```

```
# Split the data into training and testing sets
```

```
train_size = int(len(data) * 0.8)
```

```
train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]
```

```
# Separate features and target variable
```

```
X_train, y_train = train_data.drop(target_column, axis=1), train_data[target_column]
```

```
X_test, y_test = test_data.drop(target_column, axis=1), test_data[target_column]
```

```
# Create a Linear Regression model
```

```
linear_model = LinearRegression()
```



```

# Train the model
linear_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = linear_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plotting actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, y_test, label='Actual Prices', color='blue')
plt.plot(test_data.index, y_pred, label='Predicted Prices', color='orange')
plt.title('Linear Regression Time Series Forecasting')
plt.xlabel('Timestamp')
plt.ylabel(target_column) # Use the target column name for the y-axis label
plt.legend()
plt.show()
This adjusted code should work for a Series as well.

```

User

it gives error in this line `target_column = data.columns[0]` ; `AttributeError: 'Series' object has no attribute 'columns'`

ChatGPT

I apologize for the oversight. If your data is a Series instead of a DataFrame, you can modify the code accordingly. In the case of a Series, there are no columns, so you can directly use the name of the Series as the target variable. Here's the adjusted code:

```

python
Copy code
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load your time series data
# Assuming 'timestamp' is the index and your target column is unnamed
# Replace 'your_data.csv' with your actual file or data loading method
data = pd.read_csv('your_data.csv', parse_dates=['timestamp'],
index_col='timestamp')['price']

# Convert the target column to numeric
data = pd.to_numeric(data, errors='coerce')

# Feature engineering: Adding lag features
for i in range(1, 11):

```

```

data[f'lag_{i}'] = data.shift(i)

# Drop rows with NaN values due to lag features
data = data.dropna()

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]

# Separate features and target variable
X_train, y_train = train_data.drop('price', axis=1), train_data['price']
X_test, y_test = test_data.drop('price', axis=1), test_data['price']

# Create a Linear Regression model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train.values.reshape(-1, 1), y_train) # Reshape X_train for compatibility
with Linear Regression

# Make predictions on the test set
y_pred = linear_model.predict(X_test.values.reshape(-1, 1)) # Reshape X_test for
compatibility with Linear Regression

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plotting actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, y_test, label='Actual Prices', color='blue')
plt.plot(test_data.index, y_pred, label='Predicted Prices', color='orange')
plt.title('Linear Regression Time Series Forecasting')
plt.xlabel('Timestamp')
plt.ylabel('Price') # Use the target column name for the y-axis label
plt.legend()
plt.show()
This modification directly extracts the 'price' Series and treats it as the target variable. The
reshaping of input data is necessary because scikit-learn expects 2D input arrays for X.

```

User

how can i take the values of my data's first column

ChatGPT

If you want to extract the values of the first column from your pandas DataFrame or Series, you can use the `.values` attribute. Here's an example:

If you have a DataFrame:

python

Copy code

```
import pandas as pd
```

```
# Assuming you have a DataFrame named 'df'
# Replace 'your_data.csv' with your actual file or data loading method
df = pd.read_csv('your_data.csv')
```

```
# Extract values from the first column
first_column_values = df.iloc[:, 0].values
If you have a Series:
```

python

Copy code

```
import pandas as pd
```

```
# Assuming you have a Series named 'series'
# Replace 'your_data.csv' with your actual file or data loading method
series = pd.read_csv('your_data.csv', squeeze=True, usecols=[0])
```

```
# Extract values from the Series
series_values = series.values
```

User

my model is here

```
"model=ARIMA(company_1_data, order=(20,1,10))
results= model.fit()
print(results.summary())". when i wrote a code to predict the last 25 rows with this model
"forecast=results.get_prediction(start=-25)
mean_forecast=forecast.predicted_mean
confidence_intervals=forecast.conf_int()
print(confidence_intervals)
lower_limits = confidence_intervals.loc[:,'lower price']
upper_limits = confidence_intervals.loc[:,'upper price']" it worked well. can you write me a
code so that i predict the next 25 steps
```

ChatGPT

Absolutely! To forecast the next 25 steps using your ARIMA model, you can use the `get_prediction()` function again, specifying the end parameter to define the end of the forecast period.

Here's an example of how you can predict the next 25 steps:

python

Copy code

```
forecast = results.get_prediction(start=len(company_1_data), end=len(company_1_data) +
24)
mean_forecast = forecast.predicted_mean
```

```
confidence_intervals = forecast.conf_int()
lower_limits = confidence_intervals.loc[:, 'lower price']
upper_limits = confidence_intervals.loc[:, 'upper price']
```

```
print(mean_forecast)
print(confidence_intervals)
```

This code assumes that `company_1_data` is a time series containing your historical data, and the results variable holds the fitted ARIMA model.

`start=len(company_1_data)` sets the starting point of the forecast to be right after the last known data point.

`end=len(company_1_data) + 24` specifies that you want to forecast the next 25 steps (adding 24 to include 25 steps total) after the last known data point.

Adjust the code according to your variable names and data structure. This code should provide you with predictions for the next 25 steps using your ARIMA model.

User

"

```
plt.figure()
```

```
plt.plot(time, mean_forecast.values, color='red', label='forecast')
```

```
plt.fill_between(time, lower_limits, upper_limits, color='pink')
```

```
plt.plot(time, company_1_data, label='Actual Data') # Replace with your actual data
```

`plt.show`" i want to plot my data, then also i want to see the predictions next to them. but they dont have the time index, they only have price values. can you modify the code i wrote above for me

ChatGPT

Certainly! To plot the actual data along with the predicted values without a time index, you can create a simple numerical index for the predicted values and use it for plotting.

Here's an example of how you can modify the code to plot the actual data alongside the predicted values:

python

Copy code

```
# Assuming 'time' represents the time index of your actual data
# Assuming 'company_1_data' contains your actual data values
# 'mean_forecast', 'lower_limits', 'upper_limits' contain the predicted values and confidence intervals
```

```
# Create a numerical index for the predicted values
```

```
num_predicted_points = len(mean_forecast)
```

```
numerical_index = range(len(company_1_data), len(company_1_data) +
num_predicted_points)
```

```
plt.figure()

# Plotting the actual data
plt.plot(time, company_1_data, label='Actual Data', color='blue')

# Plotting the forecasted values using the numerical index
plt.plot(numerical_index, mean_forecast.values, color='red', label='Forecast')

# Fill between the confidence intervals
plt.fill_between(numerical_index, lower_limits, upper_limits, color='pink')

plt.xlabel('Time') # Update with appropriate label
plt.ylabel('Price') # Update with appropriate label
plt.title('Actual Data and Forecast')
plt.legend()
plt.show()

```

This code creates a numerical index (numerical_index) starting right after the last known data point and plots the actual data along with the forecasted values using this numerical index. Adjust the labels and titles according to your requirements.

User

they have different indexes so there is a problem in plotting.

ChatGPT

You're right; the discrepancy in indexes could cause issues in plotting directly. To plot two series with different indexes, you can align them by their values rather than indexes for comparison.

Here's an approach to plot mean_forecast.values against the first 25 points of company_1_data2:

python

Copy code

```
import matplotlib.pyplot as plt
```

```
# Assuming 'mean_forecast' contains your forecasted values (25 points)
# Assuming 'lower_limits' and 'upper_limits' contain the corresponding confidence intervals
# Assuming 'company_1_data2' contains your second dataset
```

```
plt.figure()
```

```
# Plotting the forecasted values and confidence intervals
plt.plot(mean_forecast.values, color='red', label='Forecast')
plt.fill_between(range(len(mean_forecast)), lower_limits, upper_limits, color='pink',
alpha=0.3)
```

```
# Plotting the first 25 points of company_1_data2 as 'Actual Data'
```

```
plt.plot(range(25), company_1_data2.iloc[:25], label='Actual Data', marker='o') # Adjust the  
slicing as needed
```

```
plt.xlabel('Index') # Update with appropriate label  
plt.ylabel('Values') # Update with appropriate label  
plt.title('Comparison between Forecast and Actual Data')  
plt.legend()  
plt.show()
```