DR. D. Y. PATIL SCHOOL OF SCIENCE AND TECHNOLOGY

TATHAWADE, PUNE

A Foundation of Data Science-Report on S&P-500 Index Price Prediction.

SUBMITTED BY:

NAME OF STUDENT	ROLL NUMBER
1.TEJAS BHAVSAR	BTAI-204
2. ROHAN MASHERE	BTAI-228
3.PRATHEMESH NANGARE	BTAI-230

GUIDED BY:

Mrs. Mily Lal.

Model Building and Evaluation.

I. Model Selection and Training:

Train/Test data split

- Time series data can be sliced randomly. So, I take sliced first 85% data to train the model and remaining 15% to test the models.

```
import statsmodels.api as sm

# Assuming '14d_future_Close_pct' is your target variable and the rest are features
# You should adjust these based on your specific use case
target = data_500['14d_future_Close_pct']
features = data_500[['Adj Close', 'ma14', 'ma200', 'rsi14', 'rsi200', 'ema14', 'ema200']]

# Add a constant to the features
linear_features = sm.add_constant(features)

# Split the data into train and test sets
train_size = int(0.85 * target.shape[0])
train_features = linear_features[:train_size]
train_targets = target[:train_size]
test_features = linear_features[train_size:]
test_targets = target[train_size:]
```

Buliding models

1. Decision Tree.

```
# Taking first 4000 days of data as training dataset
train_features=train_features[4000:]
train_targets=train_targets[4000:]
# Taking 200 days worth of data for testing
test_features=test_features[200:]
test_targets=test_targets[200:]
```

```
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor(max_depth=5)
decision_tree.fit(train_features, train_targets)

```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5)
```

```
print(decision_tree.score(train_features, train_targets))
```

```
0.7712255860537994
```

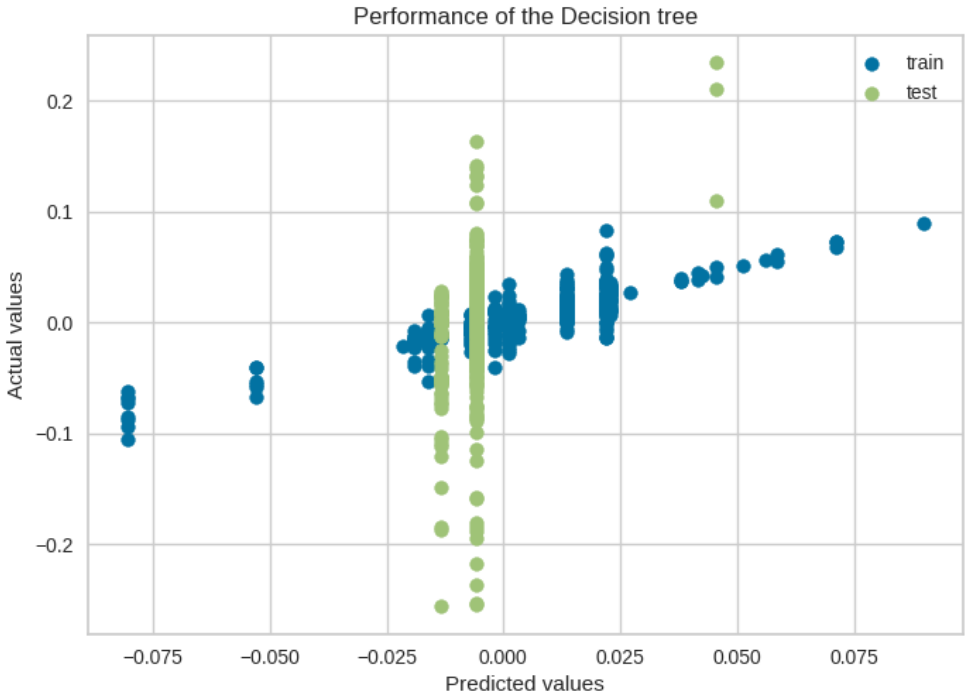
Accuracy on the training dataset is 0.771 which is excellent, so let's check how the decision tree performed on test dataset.

```
[ ]  1 print(decision_tree.score(test_features, test_targets))
```

```
0.01861861519886554
```

- Accuracy of the decision tree is 0.186. That means the tree didn't perform well for the test dataset. Decision tree probably overfit for the training dataset.

```
1
2 train_predictions = decision_tree.predict(train_features)
3 test_predictions = decision_tree.predict(test_features)
4 plt.scatter(train_predictions, train_targets, label='train')
5 plt.scatter(test_predictions, test_targets, label='test')
6 plt.xlabel('Predicted values')
7 plt.ylabel('Actual values')
8 plt.title('Performance of the Decision tree')
9 plt.legend()
10 plt.show()
```



2. Random Forest.

```
[ ]  1 from sklearn.ensemble import RandomForestRegressor
     2 random_forest = RandomForestRegressor(n_estimators=200,
     3                                       max_depth=5,
     4                                       max_features=5,
     5                                       random_state=42)
     6 random_forest.fit(train_features, train_targets)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=5, max_features=5, n_estimators=200,
                      random_state=42)
```

```
[ ]  1 print(random_forest.score(train_features, train_targets))
```

```
0.8507707102224015
```

```
[ ]  1 print(random_forest.score(test_features, test_targets))
```

```
0.0558471457666575
```

- The accuracy is not great. That's because I used random hyperparameters to begin with. So, let's figure out what are the best hyperparameters for Random Forest regressor.

```
[ ]  1 from sklearn.model_selection import ParameterGrid
     2 grid = {'n_estimators': [200], 'max_depth':[3, 4,5], 'max_features': [1,2,3,4]}
     3 from pprint import pprint
     4 pprint(list(ParameterGrid(grid)))
```

```
[{'max_depth': 3, 'max_features': 1, 'n_estimators': 200},
 {'max_depth': 3, 'max_features': 2, 'n_estimators': 200},
 {'max_depth': 3, 'max_features': 3, 'n_estimators': 200},
 {'max_depth': 3, 'max_features': 4, 'n_estimators': 200},
 {'max_depth': 4, 'max_features': 1, 'n_estimators': 200},
 {'max_depth': 4, 'max_features': 2, 'n_estimators': 200},
 {'max_depth': 4, 'max_features': 3, 'n_estimators': 200},
 {'max_depth': 4, 'max_features': 4, 'n_estimators': 200},
 {'max_depth': 5, 'max_features': 1, 'n_estimators': 200},
 {'max_depth': 5, 'max_features': 2, 'n_estimators': 200},
 {'max_depth': 5, 'max_features': 3, 'n_estimators': 200},
 {'max_depth': 5, 'max_features': 4, 'n_estimators': 200}]
```

```
     1 test_scores = []
     2 for g in ParameterGrid(grid):
     3     random_forest.set_params(**g)
     4     random_forest.fit(train_features, train_targets)
     5     test_scores.append(random_forest.score(test_features, test_targets))
     6 best_idx = np.argmax(test_scores)
     7 print(test_scores[best_idx])
     8 print(ParameterGrid(grid)[best_idx])
```

```
0.05026954945863071
{'n_estimators': 200, 'max_features': 4, 'max_depth': 5}
```

Boosted models are general class of machine learning algorithms. These work by iteratively fitting models such as decision trees to the data. They work by taking residual error of the first model to the next model and so on.

3. Gradient boosting

```
[ ]  1 # Build the gradient boosting algorithm with some hyperparameters
     2 from sklearn.ensemble import GradientBoostingRegressor
     3 gb = GradientBoostingRegressor(max_features=4,
     4                                learning_rate=0.01,
     5                                n_estimators=200,
     6                                subsample=0.6,
     7                                random_state=42)
     8 gb.fit(train_features, train_targets)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.01, max_features=4, n_estimators=200,
                          random_state=42, subsample=0.6)
```

```
     1 print(gb.score(train_features, train_targets))
```

```
0.7437536334098643
```

```
[ ]  1 print(gb.score(test_features, test_targets))
```

```
0.024578882898274257
```

Interpretation: Gradient boosting gave us slightly better performance (0.024) than previous models but its still not enough to predict the stock prices accurately.

4 - Neural networks

- Standardization

```
[ ]  1 from sklearn.preprocessing import StandardScaler
     2 sc = StandardScaler()
     3 scaled_train_features = sc.fit_transform(train_features)
     4 scaled_test_features = sc.transform(test_features)
     5
```

```
     1 from keras.models import Sequential
     2 from keras.layers import Dense
     3 #Creating the model
     4 model = Sequential()
     5
     6 #Lets add layers
     7 #First layer with 50 nodes, specify the input dim as scaled features from training se
     8 model.add(Dense(50, input_dim=scaled_train_features.shape[1], activation='relu'))
     9
    10 #Another layer with 10 nodes
    11 model.add(Dense(10, activation='relu'))
    12
    13 #Output layer and using linear for regression
    14 model.add(Dense(1, activation='linear'))
    15
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

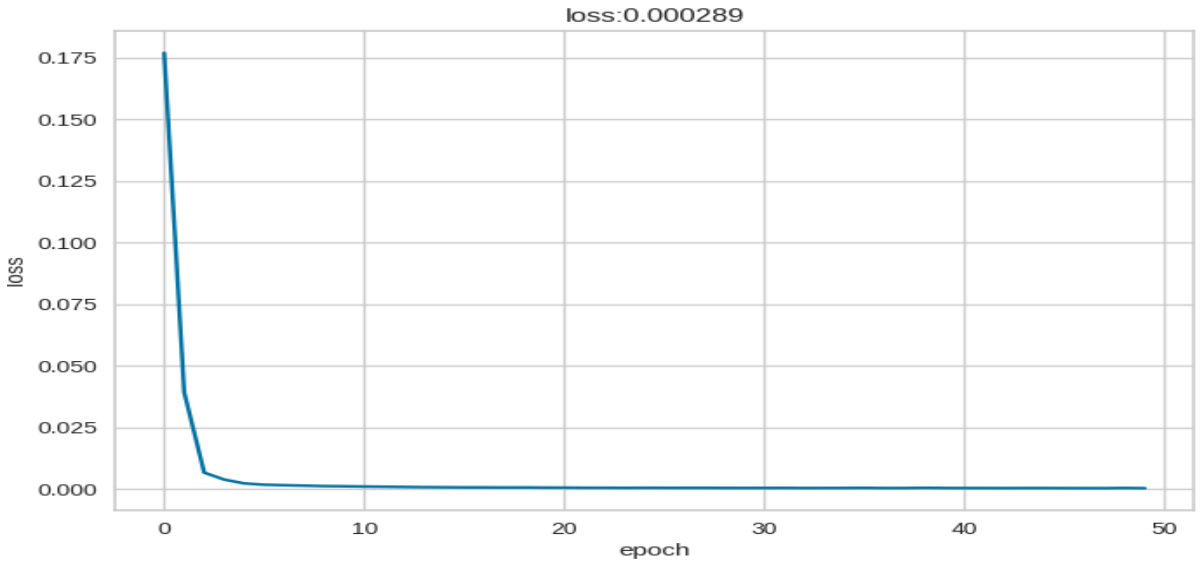
```
[ ]  1 model.compile(optimizer='adam', loss='mse')
     2 history = model.fit(scaled_train_features, train_targets, epochs=50)
```

Show hidden output

Epoch 50/50

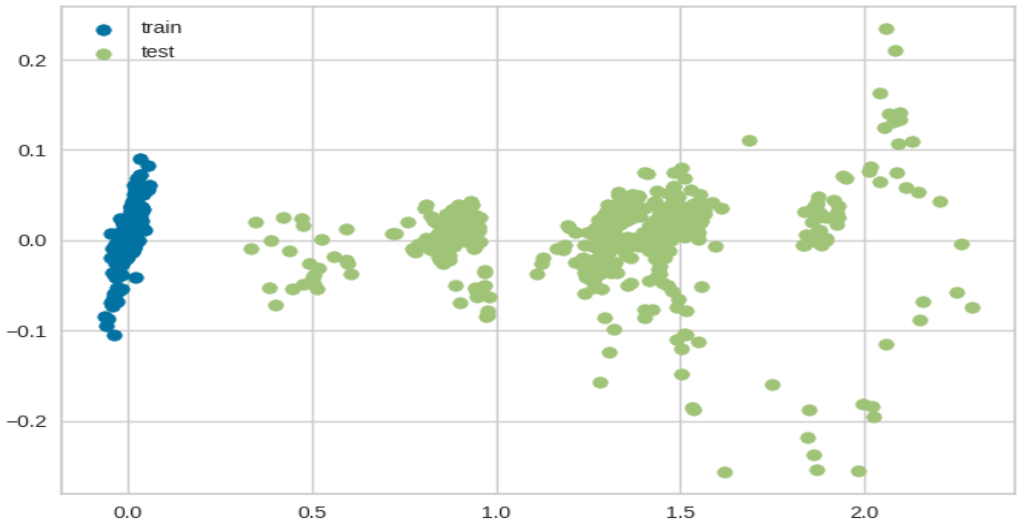
12/12 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 4ms/step - loss: 2.8024e-04.

```
1 plt.plot(history.history['loss'])
2 plt.title('loss:' + str(round(history.history['loss'][-1], 6)))
3 plt.xlabel('epoch')
4 plt.ylabel('loss')
5 plt.show()
```



```python
from sklearn.metrics import r2_score

train_preds = model.predict(scaled_train_features)
test_preds = model.predict(scaled_test_features)
print(r2_score(train_targets, train_preds))
print(r2_score(test_targets, test_preds))

plt.scatter(train_preds, train_targets, label='train')
plt.scatter(test_preds, test_targets, label='test')
plt.legend()
plt.show()
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
0.6107976932199127
-714.705827690308
```



Interpretation:

Neural network performed poorly with the dataset. That's because I didn't tune the model well, there's a lot of scope for improvement. Like creating custom loss functions and using different activation functions etc, I could have tried other options but due to lack of time I'm going to try one more model.

5. Facebook's prophet.

```python
# Defining proper training and test data sets
train_size = int(data_500.shape[0]*0.85)
train_df = data_500.iloc[:train_size]
test_df = data_500.iloc[train_size+1:]
```

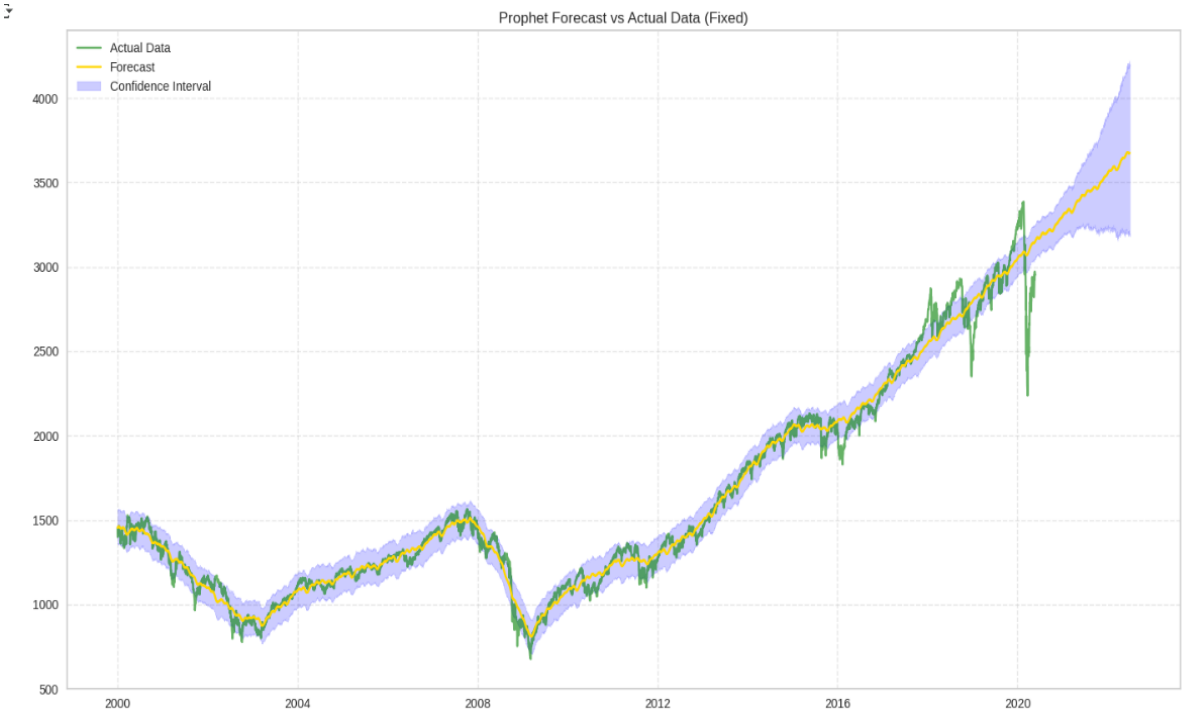
```python
from prophet import Prophet

prophet_df = data_500[['Date','Adj Close']]

prophet_df.rename(columns={'Date':'ds','Adj Close':'y'},inplace=True)
prophet_df.ds = pd.to_datetime(prophet_df.ds)

prophet_df = prophet_df[['ds','y']]

pro_model = Prophet()
pro_model.fit(prophet_df)
```

Show hidden output

```python
from prophet import Prophet

pro_model = Prophet() # prophet module is imported above so no need to write prophet.Prophet()
pro_model.fit(prophet_df)
#Create future dataframe
test_dates = pro_model.make_future_dataframe(periods=test_df.shape[0])
#Forecast the data using the model
forecast_df = pro_model.predict(test_dates)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

```python
# Filter actual data to match forecast's date range
prophet_df_filtered = prophet_df[prophet_df['ds'] >= forecast_df['ds'].min()]

fig, ax = plt.subplots(figsize=(14, 8))

# Plot cleaned actual data (aligned)
ax.plot(prophet_df_filtered['ds'], prophet_df_filtered['y'], c='green', label='Actual Data', alpha=0.6)

# Forecast
ax.plot(forecast_df['ds'], forecast_df['yhat'], c='gold', label='Forecast')

# Confidence interval
ax.fill_between(forecast_df['ds'], forecast_df['yhat_lower'], forecast_df['yhat_upper'],
                color='blue', alpha=0.2, label='Confidence Interval')

ax.legend()
ax.grid(True, linestyle='--', alpha=0.5)
ax.set_title("Prophet Forecast vs Actual Data (Fixed)")

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



```
[ ]  1 metric_df = forecast_df.set_index('ds')[['yhat']].join(prophet_df.set_index('ds').y).reset_index()
     2 metric_df.dropna(inplace=True)
```

```
[ ]  1 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
     2 r2_score(metric_df.y, metric_df.yhat)
```

```
0.9834319129894793
```

R-square value is 0.98 which means the model is a great fit. Let's find out if its an overfit using Mean squared error and Mean absolute error.

```
[ ]  1 mean_squared_error(metric_df.y, metric_df.yhat)
```

```
6214.925578482243
```

Mean square error is decent so the model is a good fit. Let's check mean absolute error as well.

```
[ ]  1 mean_absolute_error(metric_df.y, metric_df.yhat)
```

```
51.24331594626781
```

Mean absolute error is 51 which is less. It means the predicted value can be 51 basis points away from the actual value either side of the curve at the maximum.

Interpretition:

From the above plot and R-square, mean square error and mean absolute error. I think Prophet predicts the stock prices with decent accuracy.

END!!