final notebook

April 16, 2023

0.1 Import Data and EDA

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas_ta as ta

df = pd.read_csv('spy.csv')
```

[ ]:
```
df.head()
```

[ ]:
```
         Date       Open       High        Low      Close   Volume  Day  \
0  1993-01-29  25.236158  25.236158  25.110605  25.218222  1003200   29
1  1993-02-01  25.236146  25.397572  25.236146  25.397572   480500    1
2  1993-02-02  25.379673  25.469354  25.325865  25.451418   201300    2
3  1993-02-03  25.487270  25.738376  25.469334  25.720440   529400    3
4  1993-02-04  25.810132  25.881876  25.523153  25.828068   531500    4

   Weekday  Week  Month  Year
0        4     4      1  1993
1        0     5      2  1993
2        1     5      2  1993
3        2     5      2  1993
4        3     5      2  1993
```

[ ]:
```
df['RSI'] = ta.rsi(df.Close, length=15)
df['EMAF'] = ta.ema(df.Close, length=20)
df['EMAM'] = ta.ema(df.Close, length=100)
df['EMAS'] = ta.ema(df.Close, length=150)

df['avg'] = (df['High'] + df['Low']) / 2
df['Tmrw_avg'] = df['avg'].shift(-1)

df.dropna(inplace=True)
df.reset_index(drop=True, inplace=True)
```

[ ]:
```
import datetime
```

```
def str_to_datetime(s):
    split = s.split('-')
    year, month, day = int(split[0]), int(split[1]), int(split[2])
    return datetime.datetime(year=year, month=month, day=day)

df['Date'] = df['Date'].apply(str_to_datetime)

df.head()
```

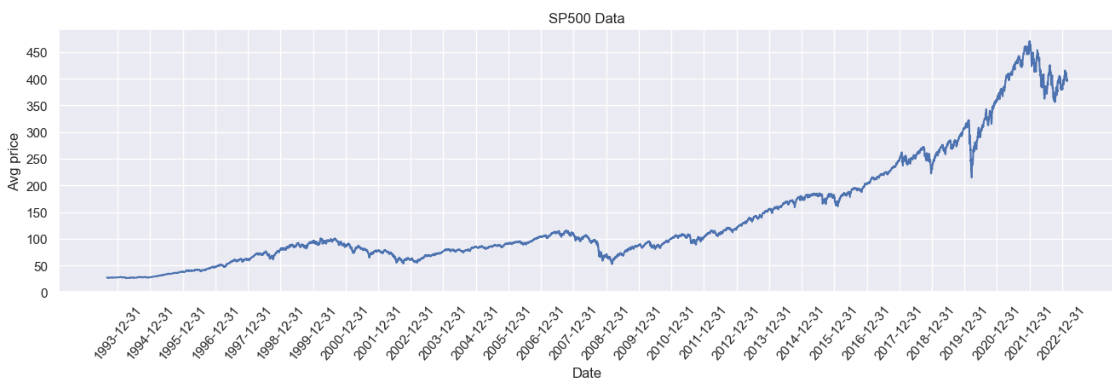
[ ]:
```
        Date       Open       High        Low      Close  Volume  Day  \
0 1993-09-01  26.950641  27.059533  26.950641  27.005087  136500    1
1 1993-09-02  27.023232  27.059530  26.896192  26.914341  472400    2
2 1993-09-03  26.896179  26.968773  26.859882  26.932476  630500    3
3 1993-09-07  26.932487  26.968784  26.714704  26.751001  196400    7
4 1993-09-08  26.751000  26.751000  26.478772  26.660257  269900    8

   Weekday  Week  Month  Year        RSI       EMAF       EMAM       EMAS  \
0        2    35      9  1993  73.860313  26.625405  26.106154  25.902142
1        3    35      9  1993  67.949468  26.652923  26.122157  25.915548
2        4    35      9  1993  68.489414  26.679547  26.138203  25.929018
3        1    36      9  1993  58.011208  26.686352  26.150338  25.939905
4        2    36      9  1993  53.616527  26.683867  26.160435  25.949446

         avg   Tmrw_avg
0  27.005087  26.977861
1  26.977861  26.914327
2  26.914327  26.841744
3  26.841744  26.614886
4  26.614886  26.642101
```

[ ]:
```
df['pct'] = df['Tmrw_avg']/df['avg']
df['pct_diff'] = (df['pct'] - 1) * 100

df.head()
```

[ ]:
```
        Date       Open       High        Low      Close  Volume  Day  \
0 1993-09-01  26.950641  27.059533  26.950641  27.005087  136500    1
1 1993-09-02  27.023232  27.059530  26.896192  26.914341  472400    2
2 1993-09-03  26.896179  26.968773  26.859882  26.932476  630500    3
3 1993-09-07  26.932487  26.968784  26.714704  26.751001  196400    7
4 1993-09-08  26.751000  26.751000  26.478772  26.660257  269900    8

   Weekday  Week  Month  Year        RSI       EMAF       EMAM       EMAS  \
0        2    35      9  1993  73.860313  26.625405  26.106154  25.902142
1        3    35      9  1993  67.949468  26.652923  26.122157  25.915548
2        4    35      9  1993  68.489414  26.679547  26.138203  25.929018
3        1    36      9  1993  58.011208  26.686352  26.150338  25.939905
```

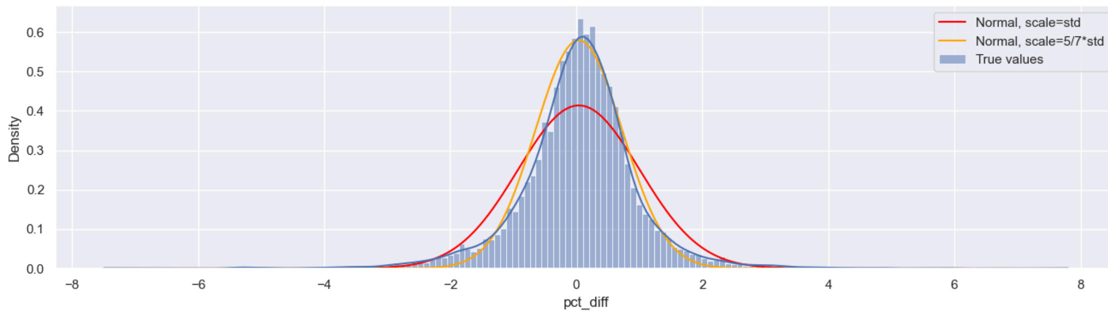
```
4         2    36       9   1993   53.616527   26.683867   26.160435   25.949446

          avg    Tmrw_avg        pct  pct_diff
0   27.005087   26.977861   0.998992 -0.100818
1   26.977861   26.914327   0.997645 -0.235502
2   26.914327   26.841744   0.997303 -0.269683
3   26.841744   26.614886   0.991548 -0.845170
4   26.614886   26.642101   1.001023  0.102254
```

[ ]:
```
import seaborn as sns
from matplotlib.dates import DateFormatter

date_form = DateFormatter('%Y-%m-%d')

sns.set(rc={'figure.figsize':(16, 4)})
ax = sns.lineplot(x=df['Date'], y=df['avg'])
ax.set_yticks(range(0, 500, 50))
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(df['Date'].min(), df['Date'].max(), freq='Y'),␣
↪rotation=50)
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



[ ]:
```
from scipy import stats

mean = df['pct_diff'].mean()
std = df['pct_diff'].std()

x = np.linspace(df['pct_diff'].min(), df['pct_diff'].max(), len(df))
pdf = stats.norm.pdf(x, loc=mean, scale=std)
pdf2 = stats.norm.pdf(x, loc=mean, scale=std*5/7)
```

```
sns.lineplot(x=x, y=pdf, color='red', label='Normal, scale=std')
sns.lineplot(x=x, y=pdf2, color='orange', label='Normal, scale=5/7*std')
sns.histplot(df['pct_diff'], stat='density', kde=True, label='True values')
plt.legend()
plt.show()
```



0.2 Forecasting predicted averages

0.2.1 LSTM

[ ]:
```
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, LSTM, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
```

[ ]:
```
def rmse_calc(y_true, y_pred):
    rmse = np.sqrt(np.mean((y_true - y_pred)**2))
    return rmse

def mape_calc(y_true, y_pred):
    y_pred = np.array(y_pred)
    y_true = np.array(y_true)
    mape = np.mean(np.abs((y_true - y_pred) / y_true))
    return mape
```

[ ]:
```
split = int(len(df)*.8)

x = df.drop(['Tmrw_avg', 'pct', 'pct_diff', 'RSI', 'EMAF', 'EMAM', 'EMAS'],␣
 ↪axis=1)
y = df['Tmrw_avg']
y = y.values.reshape(-1,1)
```

```
x_train, x_test = x[:split], x[split:]
y_train, y_test = y[:split], y[split:]

train_dates = x_train['Date']
test_dates = x_test['Date']
x_test = x_test.drop('Date', axis=1)
x_train = x_train.drop('Date', axis=1)
```

[ ]:
```
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()
x_train_scaled = scaler_x.fit_transform(x_train)
x_test_scaled = scaler_x.transform(x_test)
y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)
```

[ ]:
```
x_train_lstm = []
x_test_lstm = []
timesteps = 20

for i in range(x_train_scaled[0].size):
    x_train_lstm.append([])
    x_test_lstm.append([])
    for j in range(timesteps, x_train_scaled.shape[0]):
        x_train_lstm[i].append(x_train_scaled[j-timesteps:j, i])
    for j in range(timesteps, x_test_scaled.shape[0]):
        x_test_lstm[i].append(x_test_scaled[j-timesteps:j, i])

x_train_lstm = np.moveaxis(x_train_lstm, [0], [2])
x_test_lstm = np.moveaxis(x_test_lstm, [0], [2])

y_train_lstm = np.array(y_train_scaled[timesteps:,-1])
y_test_lstm = np.array(y_test_scaled[timesteps:,-1])

y_train_lstm = y_train_lstm.reshape(len(y_train_lstm),1)
y_test_lstm = y_test_lstm.reshape(len(y_test_lstm),1)

train_dates_lstm = np.array(train_dates[timesteps:])
test_dates_lstm = np.array(test_dates[timesteps:])
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\3268294454.py:23:
FutureWarning: The behavior of `series[i:j]` with an integer-dtype index is
deprecated. In a future version, this will be treated as *label-based* indexing,
consistent with e.g. `series[i]` lookups. To retain the old behavior, use
`series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  test_dates_lstm = np.array(test_dates[timesteps:])
```

```
[ ]: early_stop = EarlyStopping(monitor='val_loss', patience=15, mode='min')
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',␣
 ↪save_best_only=True)
model = Sequential()
model.add(LSTM(64, input_shape=(x_train_lstm.shape[1], x_train_lstm.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
history = model.fit(x_train_lstm, y_train_lstm, epochs=1000, batch_size=50,␣
 ↪validation_data=(x_test_lstm, y_test_lstm), shuffle=False, verbose=2,␣
 ↪callbacks=[early_stop, checkpoint])
```

```
Epoch 1/1000
119/119 - 3s - loss: 0.0060 - val_loss: 0.2464 - 3s/epoch - 21ms/step
Epoch 2/1000
119/119 - 1s - loss: 0.0050 - val_loss: 0.0978 - 678ms/epoch - 6ms/step
Epoch 3/1000
119/119 - 1s - loss: 0.0018 - val_loss: 0.0238 - 702ms/epoch - 6ms/step
Epoch 4/1000
119/119 - 1s - loss: 0.0019 - val_loss: 0.0442 - 672ms/epoch - 6ms/step
Epoch 5/1000
119/119 - 1s - loss: 0.0013 - val_loss: 0.0523 - 1s/epoch - 9ms/step
Epoch 6/1000
119/119 - 2s - loss: 0.0012 - val_loss: 0.0558 - 2s/epoch - 18ms/step
Epoch 7/1000
119/119 - 2s - loss: 0.0010 - val_loss: 0.0278 - 2s/epoch - 18ms/step
Epoch 8/1000
119/119 - 2s - loss: 0.0012 - val_loss: 0.0547 - 2s/epoch - 14ms/step
Epoch 9/1000
119/119 - 1s - loss: 0.0011 - val_loss: 0.0605 - 717ms/epoch - 6ms/step
Epoch 10/1000
119/119 - 1s - loss: 9.2061e-04 - val_loss: 0.0439 - 726ms/epoch - 6ms/step
Epoch 11/1000
119/119 - 1s - loss: 9.0096e-04 - val_loss: 0.0483 - 780ms/epoch - 7ms/step
Epoch 12/1000
119/119 - 1s - loss: 9.2830e-04 - val_loss: 0.0402 - 738ms/epoch - 6ms/step
Epoch 13/1000
119/119 - 1s - loss: 9.5713e-04 - val_loss: 0.0465 - 700ms/epoch - 6ms/step
Epoch 14/1000
119/119 - 1s - loss: 9.0187e-04 - val_loss: 0.0386 - 658ms/epoch - 6ms/step
Epoch 15/1000
119/119 - 1s - loss: 0.0012 - val_loss: 0.0496 - 642ms/epoch - 5ms/step
Epoch 16/1000
119/119 - 1s - loss: 9.7780e-04 - val_loss: 0.0403 - 685ms/epoch - 6ms/step
Epoch 17/1000
119/119 - 1s - loss: 9.0681e-04 - val_loss: 0.0675 - 719ms/epoch - 6ms/step
Epoch 18/1000
119/119 - 1s - loss: 8.4959e-04 - val_loss: 0.0771 - 740ms/epoch - 6ms/step
```

[ ]:
```
best_model = load_model('best_model1.h5')
y_pred = best_model.predict(x_test_lstm)
```

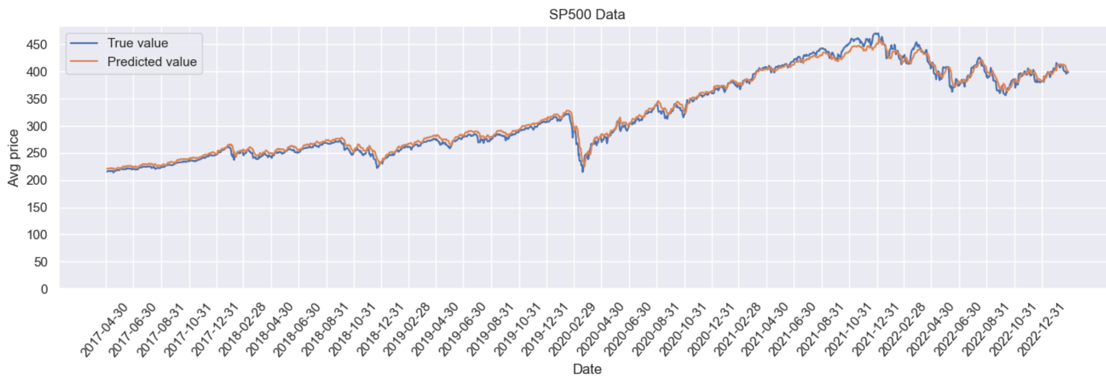
```
46/46 [==============================] - 0s 2ms/step
```

[ ]:
```
y_pred_inv = scaler_y.inverse_transform(y_pred)
y_test_inv = scaler_y.inverse_transform(y_test_lstm)

mse = np.mean((y_pred_inv - y_test_inv)**2)

print('LSTM Scores')
print(f'MSE:  {mse:.3f}')
print(f'MAE:  {mean_absolute_error(y_test_inv, y_pred_inv):.3f}')
print(f'RMSE: {rmse_calc(y_test_inv, y_pred_inv):.3f}')
print(f'MAPE: {mape_calc(y_test_inv, y_pred_inv):.3f}')
```

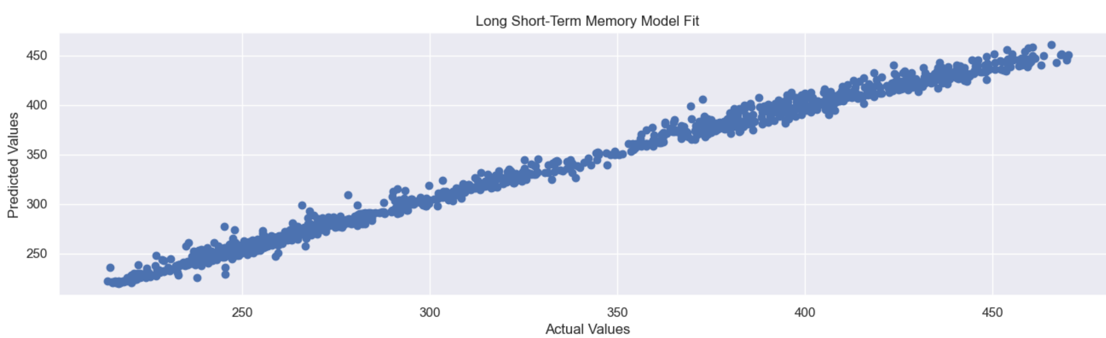
```
LSTM Scores
MSE:  58.719
MAE:  6.094
RMSE: 7.663
MAPE: 0.020
```

[ ]:
```
perf = [{'Model': 'LSTM', 'Target': 'Price', 'MSE': mse, 'MAE':␣
↪mean_absolute_error(y_test_inv, y_pred_inv), 'MAPE': mape_calc(y_test_inv,␣
↪y_pred_inv), 'RMSE': rmse_calc(y_test_inv, y_pred_inv), 'R2': np.nan}]

perf_df = pd.DataFrame(perf)
```

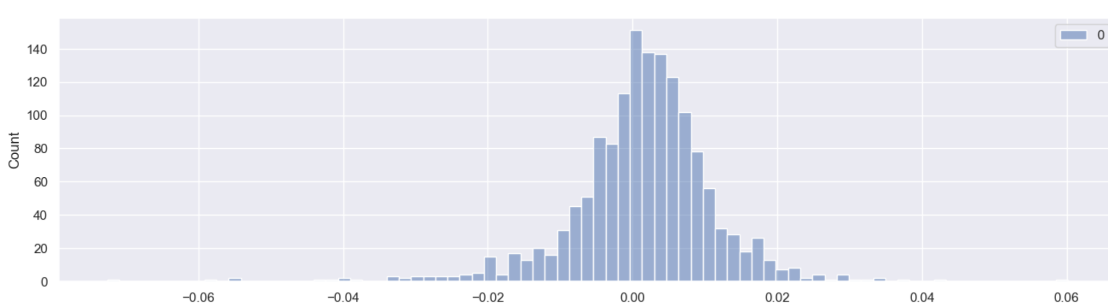
[ ]:
```
sns.set(rc={'figure.figsize':(16, 4)})
sns.lineplot(x=test_dates_lstm, y=y_test_inv.flatten(), label=f'True value')
sns.lineplot(x=test_dates_lstm, y=y_pred_inv.flatten(), label=f'Predicted␣
↪value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
↪rotation=50)
plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



[ ]:
```python
plt.scatter(y_test_inv, y_pred_inv)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Long Short-Term Memory Model Fit')
plt.show()
```



[ ]:
```python
error = y_test_inv - y_pred_inv

sns.histplot(error)

plt.show()
```

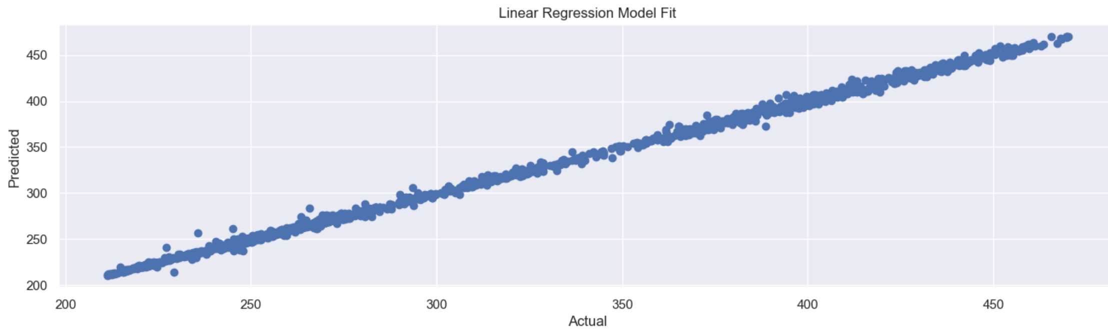


0.2.2 Linear Regression

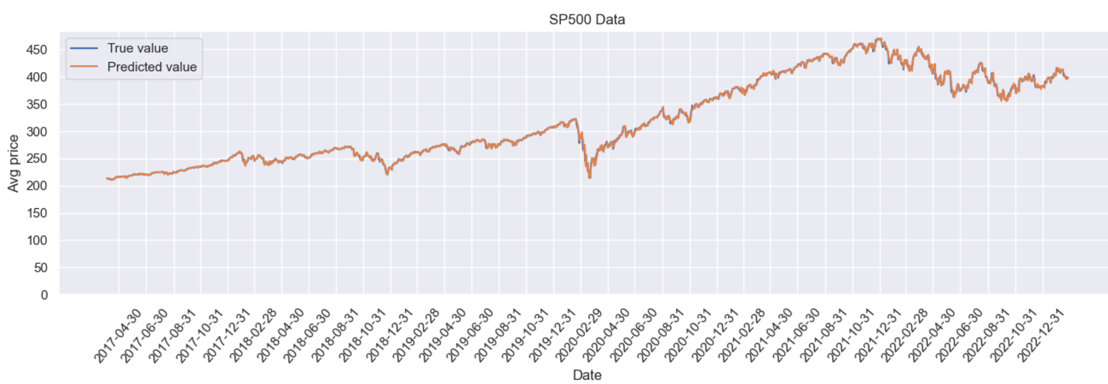
[ ]:
```
# Linear Regression - Import library
from sklearn.linear_model import LinearRegression

# Linear Regression
lr_model = LinearRegression()
lr_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
 ↪y_train_scaled)
lr_pred = lr_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
lr_pred = scaler_y.inverse_transform(lr_pred)

lr_mse = np.mean((lr_pred - y_test)**2)
lr_mae = mean_absolute_error(y_test, lr_pred)

# Calculate R-squared
lr_r2 = r2_score(y_test, lr_pred)

# Print Linear Regression Mean Squared Error, Mean Absolute Error, and R-squared
print('Linear regression scores')
print(f'R-squared: {lr_r2:.3f}')
print(f'MSE: {lr_mse:.3f}')
print(f'MAE: {lr_mae:.3f}')
print(f'RMSE: {rmse_calc(y_test, lr_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, lr_pred):.3f}')

perf = [{'Model': 'LR', 'Target': 'Price', 'MSE': lr_mse, 'MAE': lr_mae, 'MAPE':
 ↪ mape_calc(y_test, lr_pred), 'RMSE': rmse_calc(y_test, lr_pred), 'R2':␣
 ↪lr_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, lr_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression Model Fit')
plt.show()
```

```
Linear regression scores
R-squared: 0.999
MSE: 7.866
MAE: 1.845
RMSE: 2.805
MAPE: 0.006
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\4059208779.py:25:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```



[ ]:
```python
sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=lr_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
  ↪rotation=50)
plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```

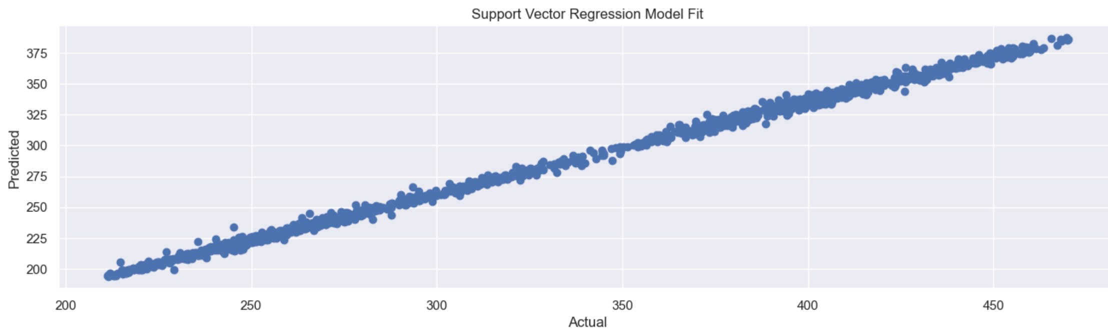


0.2.3 SVM

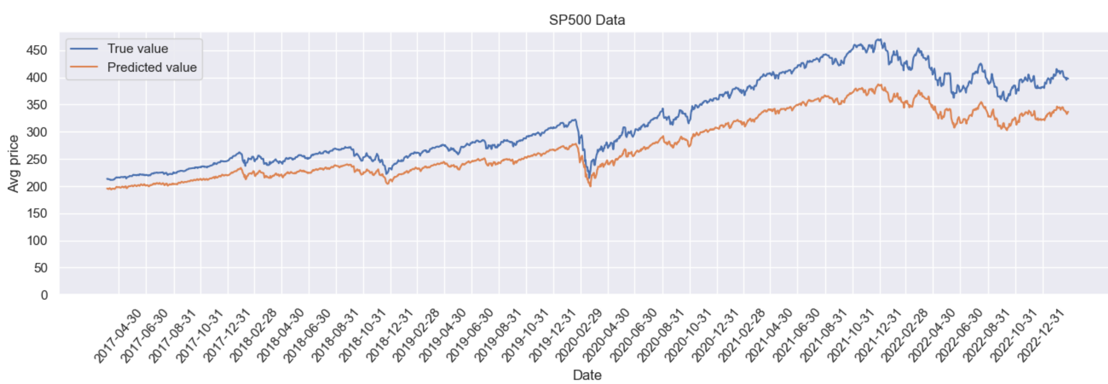
[ ]:
```python
# Support Vector Regression - Import library
from sklearn.svm import SVR

# Support Vector Regression
svm_model = SVR(kernel='linear')
svm_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
 ↪y_train_scaled.ravel())
svm_pred = svm_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
svm_pred = scaler_y.inverse_transform(svm_pred.reshape(-1, 1))
svm_mse = np.mean((svm_pred - y_test)**2)

# Calculate R-squared and mean absolute error
svm_r2 = r2_score(y_test, svm_pred)
svm_mae = mean_absolute_error(y_test, svm_pred)

# Print SVM Mean Squared Error, Mean Absolute Error and R-squared
print('SVM Regression scores')
print('R-squared:', svm_r2)
print('MSE:', svm_mse)
print('MAE:', svm_mae)
print(f'RMSE: {rmse_calc(y_test, svm_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, svm_pred):.3f}')

perf = [{'Model': 'SVM', 'Target': 'Price', 'MSE': svm_mse, 'MAE': svm_mae,␣
 ↪'MAPE': mape_calc(y_test, svm_pred), 'RMSE': rmse_calc(y_test, svm_pred),␣
 ↪'R2': svm_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, svm_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Support Vector Regression Model Fit')
plt.show()
```

```
SVM Regression scores
R-squared: 0.5872422668443428
MSE: 2345.7899558459226
MAE: 44.34346207640726
RMSE: 48.433
MAPE: 0.132
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\1081433625.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=svm_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'), rotation=50)
plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



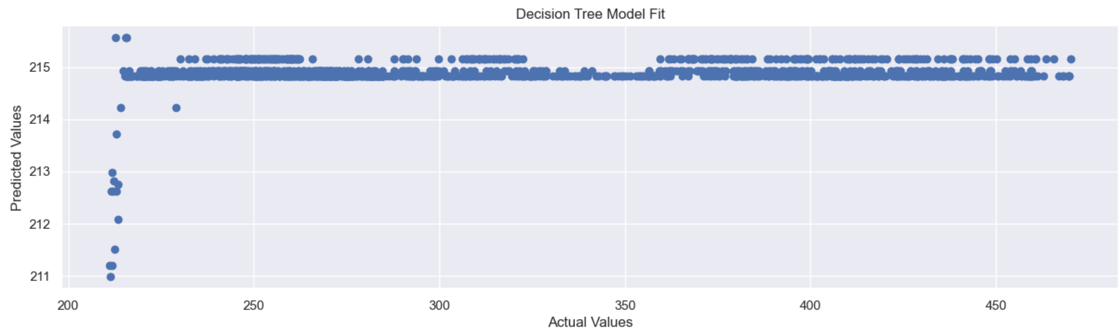
0.2.4 Decision Tree

```
[ ]: # Decision Tree Regression - Import library
from sklearn.tree import DecisionTreeRegressor

# Decision Tree Regression
dt_model = DecisionTreeRegressor()
dt_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1), y_train_scaled)
```

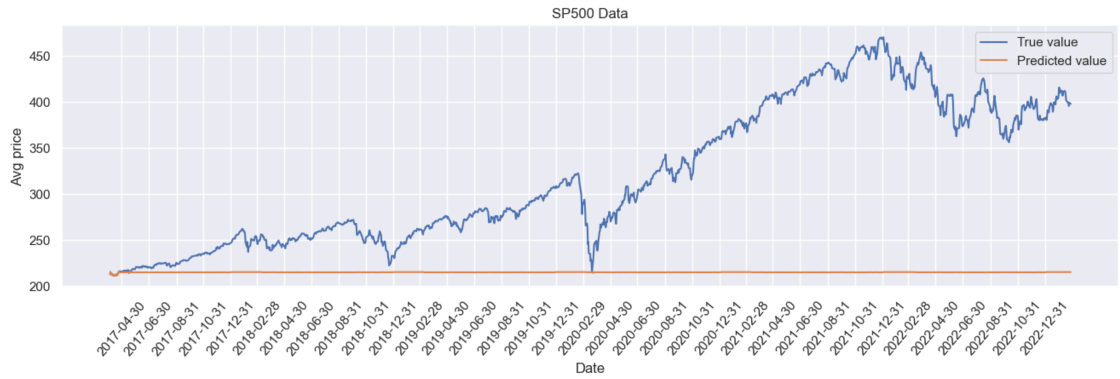
```python
dt_pred = dt_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
dt_pred = scaler_y.inverse_transform(dt_pred.reshape(-1, 1))
dt_mse = np.mean((dt_pred - y_test)**2)

# Calculate R-squared and mean absolute error
dt_r2 = r2_score(y_test, dt_pred)
dt_mae = mean_absolute_error(y_test, dt_pred)

# Print Decision Tree Mean Squared Error, Mean Absolute Error, and R-squared
print('Decision Tree regression scores')
print('R-squared:', dt_r2)
print('MSE:', dt_mse)
print('MAE:', dt_mae)
print(f'RMSE: {rmse_calc(y_test, dt_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, dt_pred):.3f}')

perf = [{'Model': 'DT', 'Target': 'Price', 'MSE': dt_mse, 'MAE': dt_mae, 'MAPE':
 ↪ mape_calc(y_test, dt_pred), 'RMSE': rmse_calc(y_test, dt_pred), 'R2':␣
 ↪dt_r2}]
perf_df = perf_df.append(perf, ignore_index=True)


# Create scatterplot with model fit
plt.scatter(y_test, dt_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Decision Tree Model Fit')
plt.show()
```

```
Decision Tree regression scores
R-squared: -1.9445349417248172
MSE: 16734.41812495516
MAE: 105.16252408859516
RMSE: 129.362
MAPE: 0.291
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\2505954022.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
     sns.lineplot(x=test_dates, y=dt_pred.flatten(), label=f'Predicted value')
     plt.title('SP500 Data')
     plt.xlabel('Date')
     plt.ylabel('Avg price')
     plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
      ↪rotation=50)
     #plt.yticks(range(0, 500, 50))
     plt.gca().xaxis.set_major_formatter(date_form)
     plt.show()
```



0.2.5 Gradient Boosting

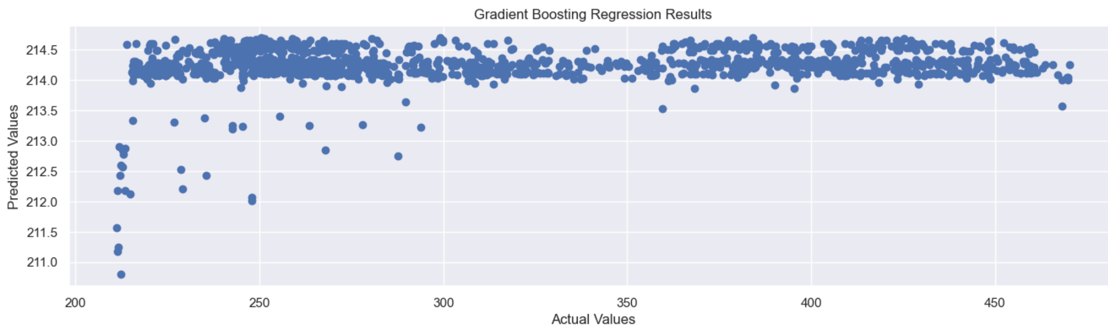
```
[ ]: # Gradient Boosting Regression - Import library
     from sklearn.ensemble import GradientBoostingRegressor

     # Gradient Boosting Regression
     gb_model = GradientBoostingRegressor(n_estimators=100)
     gb_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
      ↪y_train_scaled)
```

```python
gb_pred = gb_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
gb_pred = scaler_y.inverse_transform(gb_pred.reshape(-1, 1))
gb_mse = np.mean((gb_pred - y_test)**2)

# Calculate R-squared and mean absolute error
gb_r2 = r2_score(y_test, gb_pred)
gb_mae = mean_absolute_error(y_test, gb_pred)

# Print Gradient Boosting Mean Squared Error, Mean Absolute Error, and R-squared
print('Grabient Boosting regression scores')
print('MSE:', gb_mse)
print('MAE:', gb_mae)
print('R-squared:', gb_r2)
print(f'RMSE: {rmse_calc(y_test, gb_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, gb_pred):.3f}')

perf = [{'Model': 'GB', 'Target': 'Price', 'MSE': gb_mse, 'MAE': gb_mae, 'MAPE':
 ↪ mape_calc(y_test, gb_pred), 'RMSE': rmse_calc(y_test, gb_pred), 'R2':␣
 ↪gb_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, gb_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Gradient Boosting Regression Results')
plt.show()
```

```
c:\Users\Reed Oken\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\ensemble\_gb.py:437: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Grabient Boosting regression scores
MSE: 16868.664058528288
MAE: 105.79906622394478
R-squared: -1.9681564288442925
RMSE: 129.879
MAPE: 0.293
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\288731510.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
     sns.lineplot(x=test_dates, y=gb_pred.flatten(), label=f'Predicted value')
     plt.title('SP500 Data')
     plt.xlabel('Date')
     plt.ylabel('Avg price')
     plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
      ↪rotation=50)
     #plt.yticks(range(0, 500, 50))
     plt.gca().xaxis.set_major_formatter(date_form)
     plt.show()
```

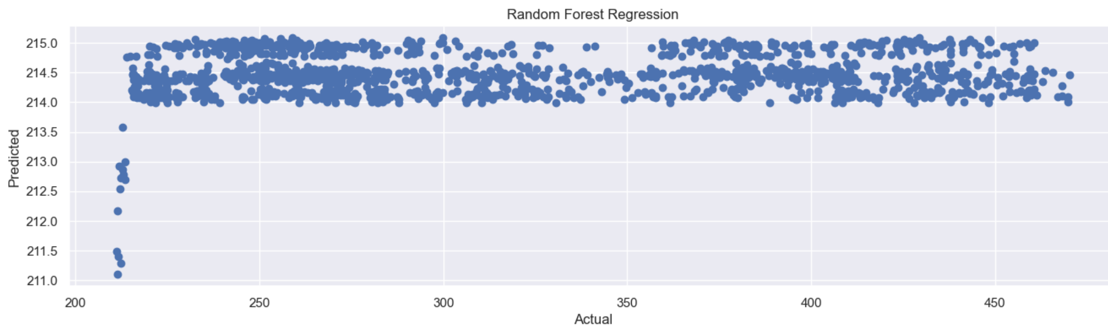
0.2.6 Random forest

```
[ ]: # Random Forest - Import library
     from sklearn.ensemble import RandomForestRegressor

     # Random Forest Regression
     rf_model = RandomForestRegressor(n_estimators=100)
     rf_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
      ↪y_train_scaled.ravel())
```

```
rf_pred = rf_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
rf_pred = scaler_y.inverse_transform(rf_pred.reshape(-1, 1))
rf_mse = np.mean((rf_pred - y_test)**2)

# Calculate R-squared and Mean Absolute Error
rf_r2 = r2_score(y_test, rf_pred)
rf_mae = mean_absolute_error(y_test, rf_pred)

# Print Random Forest Mean Squared Error, Mean Absolute Error, and R-squared
print('Random Forest regression scores')
print('R-squared:', rf_r2)
print('MSE:', rf_mse)
print('MAE:', rf_mae)
print(f'RMSE: {rmse_calc(y_test, rf_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, rf_pred):.3f}')

perf = [{'Model': 'RF', 'Target': 'Price', 'MSE': rf_mse, 'MAE': rf_mae, 'MAPE':
 ↪ mape_calc(y_test, rf_pred), 'RMSE': rmse_calc(y_test, rf_pred), 'R2':␣
 ↪rf_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, rf_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Random Forest Regression")
plt.show()
```

```
Random Forest regression scores
R-squared: -1.9603604308096898
MSE: 16824.357744154597
MAE: 105.57970997371218
RMSE: 129.709
MAPE: 0.293
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\3399960131.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
     sns.lineplot(x=test_dates, y=rf_pred.flatten(), label=f'Predicted value')
     plt.title('SP500 Data')
     plt.xlabel('Date')
     plt.ylabel('Avg price')
     plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
      ↪rotation=50)
     #plt.yticks(range(0, 500, 50))
     plt.gca().xaxis.set_major_formatter(date_form)
     plt.show()
```

0.3 Predicting percent change

0.3.1 adjusting testing and training datasets

```
[ ]: split = int(len(df)*.8)

     x = df.drop(['Tmrw_avg', 'pct', 'pct_diff'], axis=1)
     y = df['pct']
     y = y.values.reshape(-1,1)
```

```
x_train, x_test = x[:split], x[split:]
y_train, y_test = y[:split], y[split:]

train_dates = x_train['Date']
test_dates = x_test['Date']
x_test = x_test.drop('Date', axis=1)
x_train = x_train.drop('Date', axis=1)
```

[ ]:
```
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()
x_train_scaled = scaler_x.fit_transform(x_train)
x_test_scaled = scaler_x.transform(x_test)
y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)
```

[ ]:
```
x_train_lstm = []
x_test_lstm = []
timesteps = 20

for i in range(x_train_scaled[0].size):
    x_train_lstm.append([])
    x_test_lstm.append([])
    for j in range(timesteps, x_train_scaled.shape[0]):
        x_train_lstm[i].append(x_train_scaled[j-timesteps:j, i])
    for j in range(timesteps, x_test_scaled.shape[0]):
        x_test_lstm[i].append(x_test_scaled[j-timesteps:j, i])

x_train_lstm = np.moveaxis(x_train_lstm, [0], [2])
x_test_lstm = np.moveaxis(x_test_lstm, [0], [2])

y_train_lstm = np.array(y_train_scaled[timesteps:,-1])
y_test_lstm = np.array(y_test_scaled[timesteps:,-1])

y_train_lstm = y_train_lstm.reshape(len(y_train_lstm),1)
y_test_lstm = y_test_lstm.reshape(len(y_test_lstm),1)

train_dates_lstm = np.array(train_dates[timesteps:])
test_dates_lstm = np.array(test_dates[timesteps:])
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\3268294454.py:23:
FutureWarning: The behavior of `series[i:j]` with an integer-dtype index is
deprecated. In a future version, this will be treated as *label-based* indexing,
consistent with e.g. `series[i]` lookups. To retain the old behavior, use
`series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  test_dates_lstm = np.array(test_dates[timesteps:])
```

0.3.2 LSTM

[ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=15, mode='min')
checkpoint = ModelCheckpoint('best_model_pct.h5', monitor='val_loss',␣
↪save_best_only=True)
model = Sequential()
model.add(LSTM(64, input_shape=(x_train_lstm.shape[1], x_train_lstm.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
history = model.fit(x_train_lstm, y_train_lstm, epochs=1000, batch_size=50,␣
↪validation_data=(x_test_lstm, y_test_lstm), shuffle=False, verbose=2,␣
↪callbacks=[early_stop, checkpoint])
```

```
Epoch 1/1000
119/119 - 2s - loss: 0.0131 - val_loss: 0.0088 - 2s/epoch - 21ms/step
Epoch 2/1000
119/119 - 1s - loss: 0.0081 - val_loss: 0.0050 - 704ms/epoch - 6ms/step
Epoch 3/1000
119/119 - 1s - loss: 0.0059 - val_loss: 0.0043 - 696ms/epoch - 6ms/step
Epoch 4/1000
119/119 - 1s - loss: 0.0059 - val_loss: 0.0052 - 689ms/epoch - 6ms/step
Epoch 5/1000
119/119 - 1s - loss: 0.0054 - val_loss: 0.0049 - 694ms/epoch - 6ms/step
Epoch 6/1000
119/119 - 1s - loss: 0.0054 - val_loss: 0.0073 - 698ms/epoch - 6ms/step
Epoch 7/1000
119/119 - 1s - loss: 0.0052 - val_loss: 0.0079 - 703ms/epoch - 6ms/step
Epoch 8/1000
119/119 - 1s - loss: 0.0052 - val_loss: 0.0071 - 691ms/epoch - 6ms/step
Epoch 9/1000
119/119 - 1s - loss: 0.0052 - val_loss: 0.0084 - 690ms/epoch - 6ms/step
Epoch 10/1000
119/119 - 1s - loss: 0.0049 - val_loss: 0.0058 - 697ms/epoch - 6ms/step
Epoch 11/1000
119/119 - 1s - loss: 0.0052 - val_loss: 0.0073 - 695ms/epoch - 6ms/step
Epoch 12/1000
119/119 - 1s - loss: 0.0049 - val_loss: 0.0065 - 747ms/epoch - 6ms/step
Epoch 13/1000
119/119 - 1s - loss: 0.0048 - val_loss: 0.0062 - 690ms/epoch - 6ms/step
Epoch 14/1000
119/119 - 1s - loss: 0.0049 - val_loss: 0.0072 - 688ms/epoch - 6ms/step
Epoch 15/1000
119/119 - 1s - loss: 0.0048 - val_loss: 0.0065 - 671ms/epoch - 6ms/step
Epoch 16/1000
119/119 - 1s - loss: 0.0047 - val_loss: 0.0061 - 689ms/epoch - 6ms/step
Epoch 17/1000
119/119 - 1s - loss: 0.0046 - val_loss: 0.0067 - 680ms/epoch - 6ms/step
```

```
Epoch 18/1000
119/119 - 1s - loss: 0.0046 - val_loss: 0.0068 - 681ms/epoch - 6ms/step
```

[ ]:
```
best_model = load_model('best_model_pct.h5')
y_pred = best_model.predict(x_test_lstm)
```

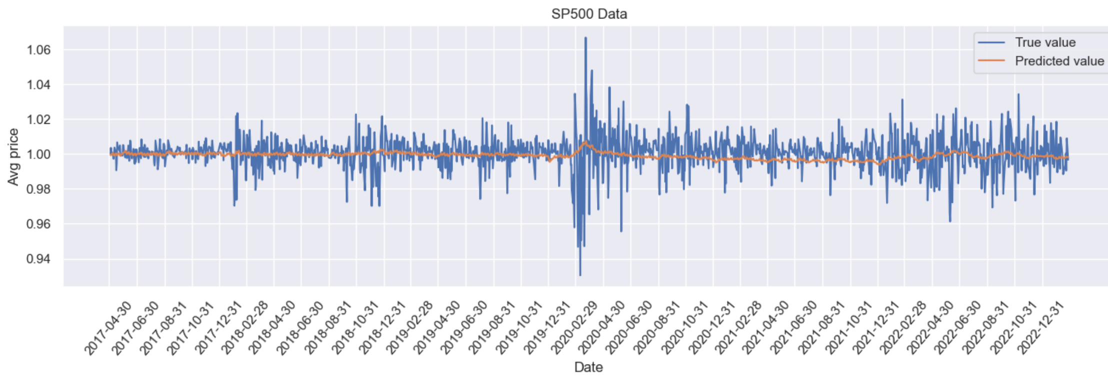
```
46/46 [==============================] - 0s 2ms/step
```

[ ]:
```
y_pred_inv = scaler_y.inverse_transform(y_pred)
y_test_inv = scaler_y.inverse_transform(y_test_lstm)

mse = np.mean((y_pred_inv - y_test_inv)**2)

print('LSTM Scores')
print(f'MSE:  {mse:.3f}')
print(f'MAE:  {mean_absolute_error(y_test_inv, y_pred_inv):.3f}')
print(f'RMSE: {rmse_calc(y_test_inv, y_pred_inv):.3f}')
print(f'MAPE: {mape_calc(y_test_inv, y_pred_inv):.3f}')

perf = [{'Model': 'LSTM', 'Target': 'Pct', 'MSE': mse, 'MAE':␣
 ↪mean_absolute_error(y_test_inv, y_pred_inv), 'MAPE': mape_calc(y_test_inv,␣
 ↪y_pred_inv), 'RMSE': rmse_calc(y_test_inv, y_pred_inv), 'R2': np.nan}]
perf_df = perf_df.append(perf, ignore_index=True)
```

```
LSTM Scores
MSE:  0.000
MAE:  0.007
RMSE: 0.010
MAPE: 0.007
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\777414301.py:13:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```

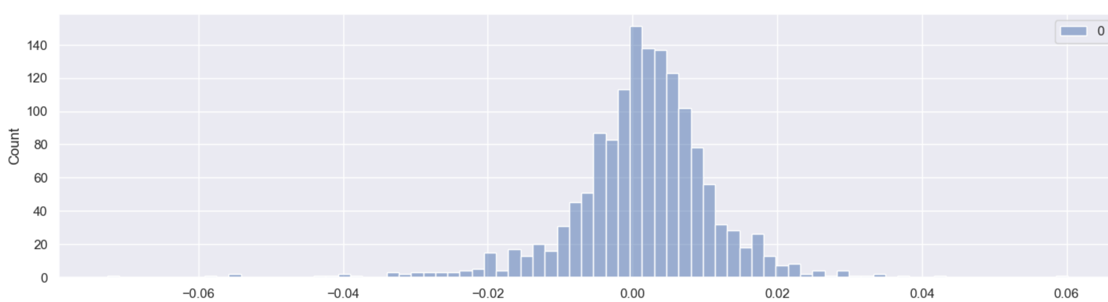
[ ]:
```
sns.set(rc={'figure.figsize':(16, 4)})
sns.lineplot(x=test_dates_lstm, y=y_test_inv.flatten(), label=f'True value')
sns.lineplot(x=test_dates_lstm, y=y_pred_inv.flatten(), label=f'Predicted␣
 ↪value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
 ↪rotation=50)
#plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



[ ]:
```
plt.scatter(y_test_inv, y_pred_inv)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Long Short-Term Memory Model Fit')
plt.show()
```

[ ]:
```
error = y_test_inv - y_pred_inv

sns.histplot(error)
```

[ ]: <AxesSubplot: ylabel='Count'>



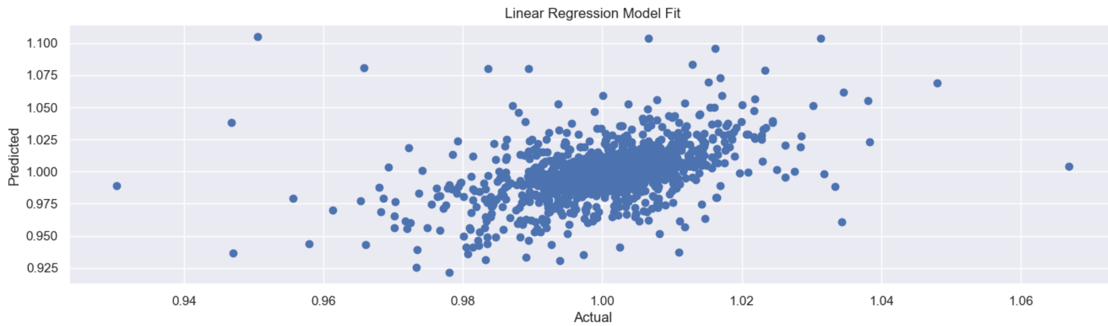
0.3.3 Linear Regression

[ ]:
```
# Linear Regression - Import library
from sklearn.linear_model import LinearRegression

# Linear Regression
lr_model = LinearRegression()
lr_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
 ↪y_train_scaled)
lr_pred = lr_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
lr_pred = scaler_y.inverse_transform(lr_pred)

lr_mse = np.mean((lr_pred - y_test)**2)
lr_mae = mean_absolute_error(y_test, lr_pred)

# Calculate R-squared
lr_r2 = r2_score(y_test, lr_pred)

# Print Linear Regression Mean Squared Error, Mean Absolute Error, and R-squared
print('Linear regression scores')
print(f'R-squared: {lr_r2:.3f}')
print(f'MSE: {lr_mse:.3f}')
print(f'MAE: {lr_mae:.3f}')
print(f'RMSE: {rmse_calc(y_test, lr_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, lr_pred):.3f}')

perf = [{'Model': 'LR', 'Target': 'Pct', 'MSE': lr_mse, 'MAE': lr_mae, 'MAPE':␣
 ↪mape_calc(y_test, lr_pred), 'RMSE': rmse_calc(y_test, lr_pred), 'R2': lr_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, lr_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression Model Fit')
plt.show()
```

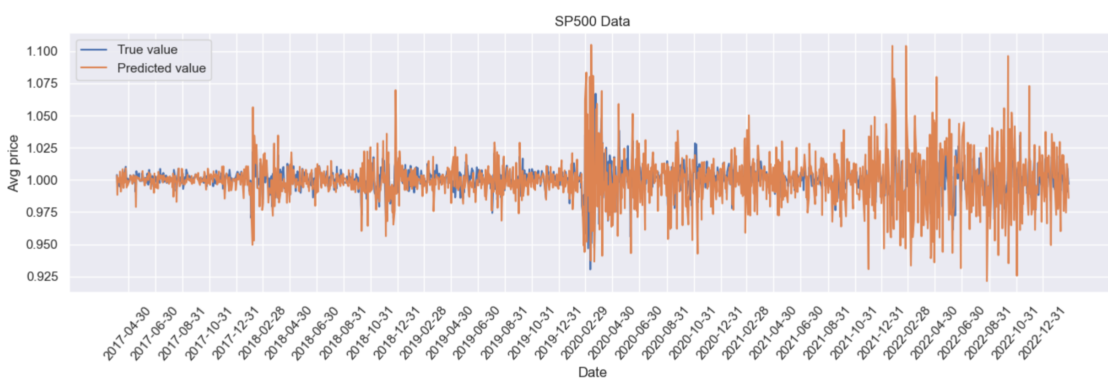
```
Linear regression scores
R-squared: -2.140
MSE: 0.000
MAE: 0.011
RMSE: 0.017
MAPE: 0.011
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\1335242769.py:25:
```

```
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
     sns.lineplot(x=test_dates, y=lr_pred.flatten(), label=f'Predicted value')
     plt.title('SP500 Data')
     plt.xlabel('Date')
     plt.ylabel('Avg price')
     plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
       ↪rotation=50)
     #plt.yticks(range(0, 500, 50))
     plt.gca().xaxis.set_major_formatter(date_form)
     plt.show()
```



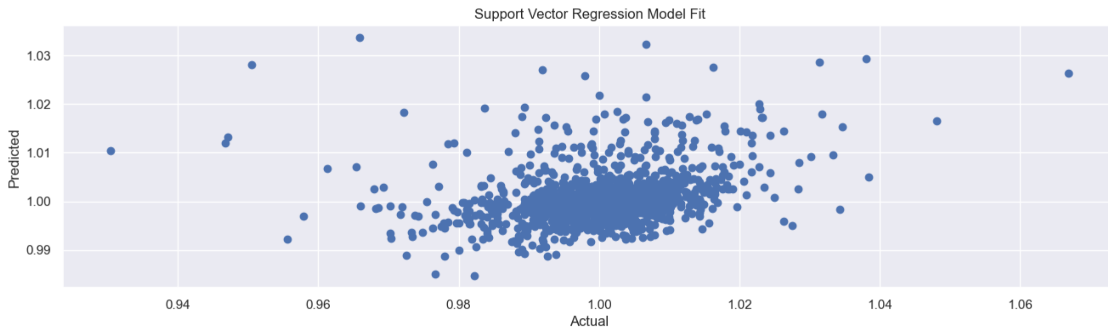
0.3.4 SVM

```
[ ]: # Support Vector Regression - Import library
     from sklearn.svm import SVR

     # Support Vector Regression
```

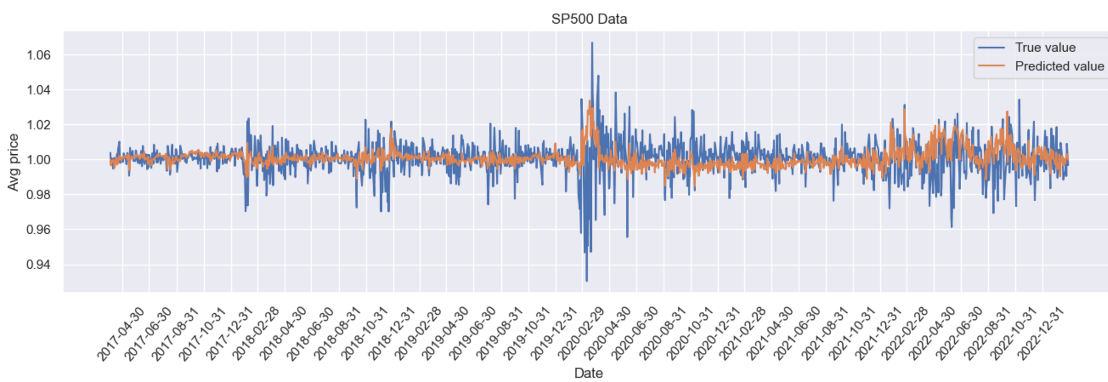
```python
svm_model = SVR(kernel='linear')
svm_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
↪y_train_scaled.ravel())
svm_pred = svm_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
svm_pred = scaler_y.inverse_transform(svm_pred.reshape(-1, 1))
svm_mse = np.mean((svm_pred - y_test)**2)

# Calculate R-squared and mean absolute error
svm_r2 = r2_score(y_test, svm_pred)
svm_mae = mean_absolute_error(y_test, svm_pred)

# Print SVM Mean Squared Error, Mean Absolute Error and R-squared
print('SVM Regression scores')
print('R-squared:', svm_r2)
print('MSE:', svm_mse)
print('MAE:', svm_mae)
print(f'RMSE: {rmse_calc(y_test, svm_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, svm_pred):.3f}')

perf = [{'Model': 'SVM', 'Target': 'Pct', 'MSE': svm_mse, 'MAE': svm_mae,␣
↪'MAPE': mape_calc(y_test, svm_pred), 'RMSE': rmse_calc(y_test, svm_pred),␣
↪'R2': svm_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, svm_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Support Vector Regression Model Fit')
plt.show()
```

```
SVM Regression scores
R-squared: -0.002779893554113455
MSE: 9.725886005147249e-05
MAE: 0.006564372349675416
RMSE: 0.010
MAPE: 0.007
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\2654001317.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```



[ ]:
```
sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=svm_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
 ↪rotation=50)
#plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



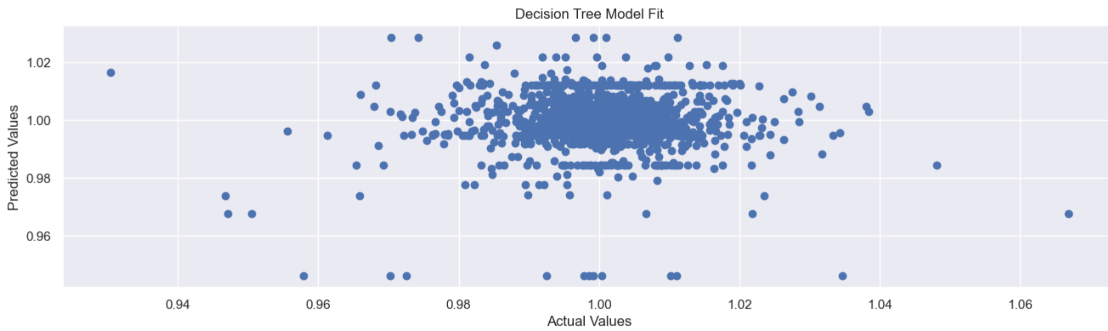
0.3.5 Decision Tree

[ ]:
```
# Decision Tree Regression - Import library
from sklearn.tree import DecisionTreeRegressor

# Decision Tree Regression
dt_model = DecisionTreeRegressor()
dt_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
 ↪y_train_scaled)
```

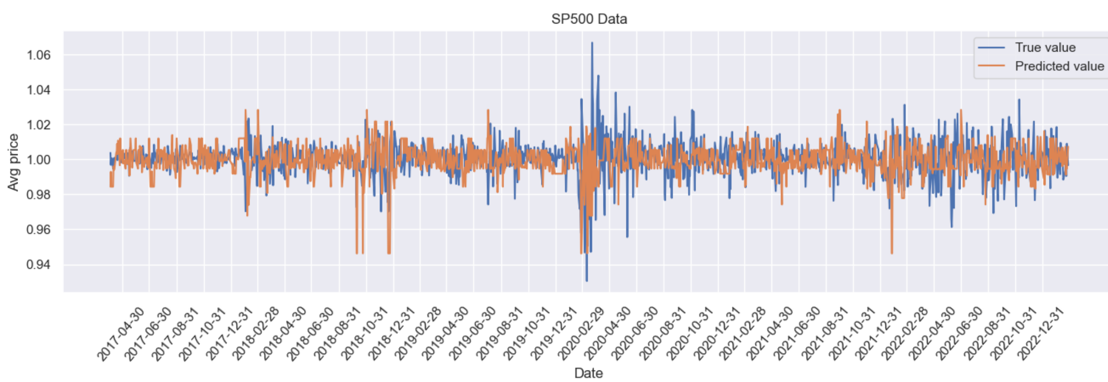
```python
dt_pred = dt_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
dt_pred = scaler_y.inverse_transform(dt_pred.reshape(-1, 1))
dt_mse = np.mean((dt_pred - y_test)**2)

# Calculate R-squared and mean absolute error
dt_r2 = r2_score(y_test, dt_pred)
dt_mae = mean_absolute_error(y_test, dt_pred)

# Print Decision Tree Mean Squared Error, Mean Absolute Error, and R-squared
print('Decision Tree regression scores')
print('R-squared:', dt_r2)
print('MSE:', dt_mse)
print('MAE:', dt_mae)
print(f'RMSE: {rmse_calc(y_test, dt_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, dt_pred):.3f}')

perf = [{'Model': 'DT', 'Target': 'Pct', 'MSE': dt_mse, 'MAE': dt_mae, 'MAPE':␣
↪mape_calc(y_test, dt_pred), 'RMSE': rmse_calc(y_test, dt_pred), 'R2': dt_r2}]
perf_df = perf_df.append(perf, ignore_index=True)


# Create scatterplot with model fit
plt.scatter(y_test, dt_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Decision Tree Model Fit')
plt.show()
```

```
Decision Tree regression scores
R-squared: -0.8296180883150333
MSE: 0.00017745326840208712
MAE: 0.00942555039700151
RMSE: 0.013
MAPE: 0.009
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\2956157721.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```




```
[ ]: sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=dt_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'), rotation=50)
#plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



0.3.6 Gradient Boosting

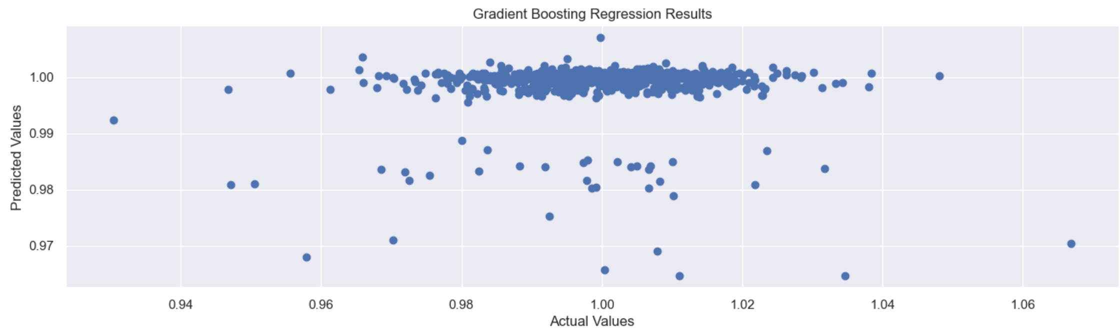
```
[ ]: # Gradient Boosting Regression - Import library
from sklearn.ensemble import GradientBoostingRegressor

# Gradient Boosting Regression
gb_model = GradientBoostingRegressor(n_estimators=100)
gb_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1), y_train_scaled)
```

```python
gb_pred = gb_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
gb_pred = scaler_y.inverse_transform(gb_pred.reshape(-1, 1))
gb_mse = np.mean((gb_pred - y_test)**2)

# Calculate R-squared and mean absolute error
gb_r2 = r2_score(y_test, gb_pred)
gb_mae = mean_absolute_error(y_test, gb_pred)

# Print Gradient Boosting Mean Squared Error, Mean Absolute Error, and R-squared
print('Grabient Boosting regression scores')
print('MSE:', gb_mse)
print('MAE:', gb_mae)
print('R-squared:', gb_r2)
print(f'RMSE: {rmse_calc(y_test, gb_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, gb_pred):.3f}')

perf = [{'Model': 'GB', 'Target': 'Pct', 'MSE': gb_mse, 'MAE': gb_mae, 'MAPE':␣
↪mape_calc(y_test, gb_pred), 'RMSE': rmse_calc(y_test, gb_pred), 'R2': gb_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, gb_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Gradient Boosting Regression Results')
plt.show()
```

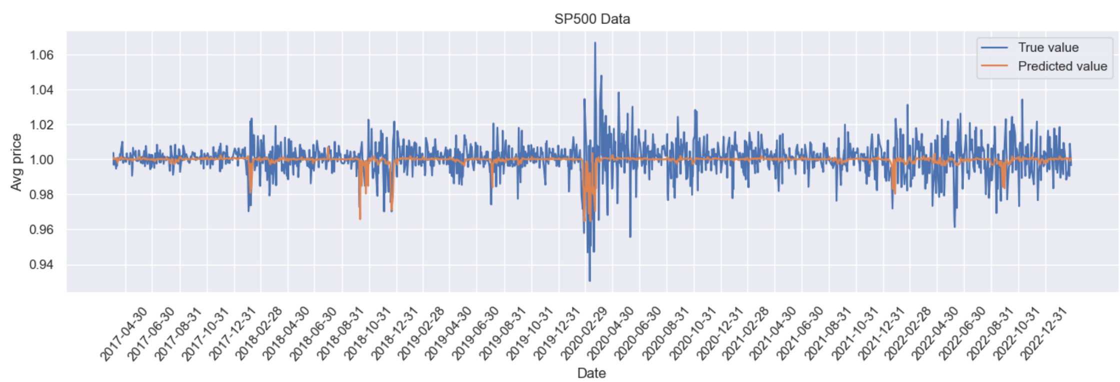
```
c:\Users\Reed Oken\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\ensemble\_gb.py:437: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Grabient Boosting regression scores
MSE: 0.00010512533737048393
MAE: 0.0068140602603072205
R-squared: -0.08388659462411763
RMSE: 0.010
MAPE: 0.007
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\444773503.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```



[ ]:
```
sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=gb_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
 ↪rotation=50)
#plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



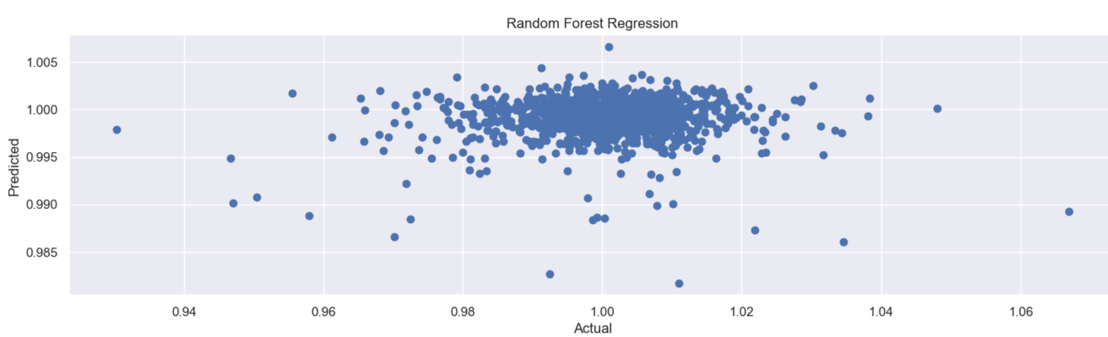
0.3.7 Random forest

[ ]:
```
# Random Forest - Import library
from sklearn.ensemble import RandomForestRegressor

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(x_train_scaled.reshape(x_train_scaled.shape[0], -1),␣
 ↪y_train_scaled.ravel())
```

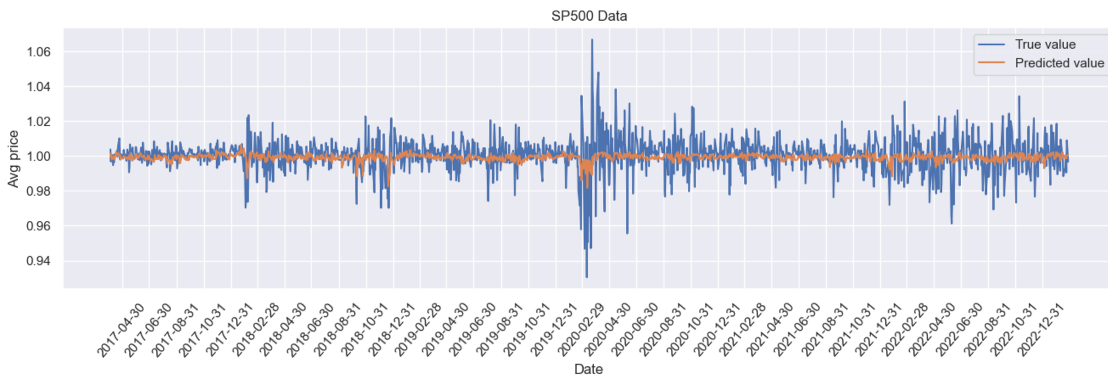
```python
rf_pred = rf_model.predict(x_test_scaled.reshape(x_test_scaled.shape[0], -1))
rf_pred = scaler_y.inverse_transform(rf_pred.reshape(-1, 1))
rf_mse = np.mean((rf_pred - y_test)**2)

# Calculate R-squared and Mean Absolute Error
rf_r2 = r2_score(y_test, rf_pred)
rf_mae = mean_absolute_error(y_test, rf_pred)

# Print Random Forest Mean Squared Error, Mean Absolute Error, and R-squared
print('Random Forest regression scores')
print('R-squared:', rf_r2)
print('MSE:', rf_mse)
print('MAE:', rf_mae)
print(f'RMSE: {rmse_calc(y_test, rf_pred):.3f}')
print(f'MAPE: {mape_calc(y_test, rf_pred):.3f}')

perf = [{'Model': 'RF', 'Target': 'Pct', 'MSE': rf_mse, 'MAE': rf_mae, 'MAPE':␣
↪mape_calc(y_test, rf_pred), 'RMSE': rmse_calc(y_test, rf_pred), 'R2': rf_r2}]
perf_df = perf_df.append(perf, ignore_index=True)

# Create scatterplot with model fit
plt.scatter(y_test, rf_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Random Forest Regression")
plt.show()
```

```
Random Forest regression scores
R-squared: -0.032133443263056005
MSE: 0.00010010583853748651
MAE: 0.0069460207575218775
RMSE: 0.010
MAPE: 0.007
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_10420\1661125558.py:24:
FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  perf_df = perf_df.append(perf, ignore_index=True)
```



[ ]:
```
sns.lineplot(x=test_dates, y=y_test.flatten(), label=f'True value')
sns.lineplot(x=test_dates, y=rf_pred.flatten(), label=f'Predicted value')
plt.title('SP500 Data')
plt.xlabel('Date')
plt.ylabel('Avg price')
plt.xticks(pd.date_range(test_dates.min(), df['Date'].max(), freq='2M'),␣
 ↪rotation=50)
#plt.yticks(range(0, 500, 50))
plt.gca().xaxis.set_major_formatter(date_form)
plt.show()
```



[ ]:
```
perf_df.head(20)
```

[ ]:

	Model	Target	MSE	MAE	MAPE	RMSE	R2
0	LSTM	Price	58.718952	6.094461	0.019569	7.662829	NaN
1	LR	Price	7.866433	1.845412	0.005730	2.804716	0.998616
2	SVM	Price	2345.789956	44.343462	0.132029	48.433356	0.587242
3	DT	Price	16734.418125	105.162524	0.291265	129.361579	-1.944535
4	GB	Price	16868.664059	105.799066	0.293362	129.879421	-1.968156
5	RF	Price	16824.357744	105.579710	0.292627	129.708742	-1.960360
6	LSTM	Pct	0.000102	0.007081	0.007084	0.010084	NaN
7	LR	Pct	0.000305	0.011236	0.011247	0.017451	-2.139990
8	SVM	Pct	0.000097	0.006564	0.006584	0.009862	-0.002780
9	DT	Pct	0.000177	0.009426	0.009424	0.013321	-0.829618
10	GB	Pct	0.000105	0.006814	0.006811	0.010253	-0.083887
11	RF	Pct	0.000100	0.006946	0.006946	0.010005	-0.032133

[ ]:
```
#perf_df = perf_df.drop('R2', axis=1)
perf_df = perf_df.drop('Target', axis=1)

perf_df.head(20)
```

[ ]:

	Model	MSE	MAE	MAPE	RMSE
0	LSTM	58.718952	6.094461	0.019569	7.662829
1	LR	7.866433	1.845412	0.005730	2.804716
2	SVM	2345.789956	44.343462	0.132029	48.433356
3	DT	16734.418125	105.162524	0.291265	129.361579
4	GB	16868.664059	105.799066	0.293362	129.879421
5	RF	16824.357744	105.579710	0.292627	129.708742

Percent difference =

$\frac{P_{n+1}-P_n}{P_n}$