In [1]:
```
import time
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sb
import numpy as np
import random

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import SimpleExpSmoothing, ExponentialS
moothing


from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

import pickle
import yfinance as yf
from sklearn.metrics import mean_absolute_error, mean_squared_error

from sklearn.linear_model import LinearRegression, LogisticRegression

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM


from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
from tensorflow import random
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.p
y:7: FutureWarning: pandas.Int64Index is deprecated and will be removed
from pandas in a future version. Use pandas.Index with the appropriate
dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.p
y:7: FutureWarning: pandas.Float64Index is deprecated and will be remov
ed from pandas in a future version. Use pandas.Index with the appropria
te dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

In [2]:
```python
ticker = '^GSPC'
period1 = int(time.mktime(datetime.datetime(1927, 12, 29, 23, 59).timetuple()))
period2 = int(time.mktime(datetime.datetime(2023, 1, 16, 23, 59).timetuple()))
interval = '1d'

query_string = f'https://query1.finance.yahoo.com/v7/finance/download/{ticker}?period1={period1}&period2={period2}&interval={interval}&events=history&includeAdjustedClose=true'

df = pd.read_csv(query_string)

csv_data=df.to_csv('SNP.csv', header=True, index = False)

snpdf = pd.read_csv("SNP.csv")
snpdf.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1927-12-30	17.660000	17.660000	17.660000	17.660000	17.660000	0
1	1928-01-03	17.760000	17.760000	17.760000	17.760000	17.760000	0
2	1928-01-04	17.719999	17.719999	17.719999	17.719999	17.719999	0
3	1928-01-05	17.549999	17.549999	17.549999	17.549999	17.549999	0
4	1928-01-06	17.660000	17.660000	17.660000	17.660000	17.660000	0

In [3]:
```python
snpdf.shape
```

Out[3]:
```
(23874, 7)
```

In [4]:
```python
snpdf.describe()
```

Out[4]:

	Open	High	Low	Close	Adj Close	Volume
count	23874.000000	23874.000000	23874.000000	23874.000000	23874.000000	2.387400e+04
mean	551.068917	574.621012	567.544959	571.304355	571.304355	8.563583e+08
std	916.289815	910.203806	899.249453	905.060532	905.060532	1.579850e+09
min	0.000000	4.400000	4.400000	4.400000	4.400000	0.000000e+00
25%	9.480000	24.350000	24.350000	24.350000	24.350000	1.420000e+06
50%	39.459999	101.910000	100.349998	101.079998	101.079998	1.887000e+07
75%	942.289978	950.774994	932.690002	942.297485	942.297485	7.614500e+08
max	4804.509766	4818.620117	4780.040039	4796.560059	4796.560059	1.145623e+10

In [5]:
```
snpdf.info()
```

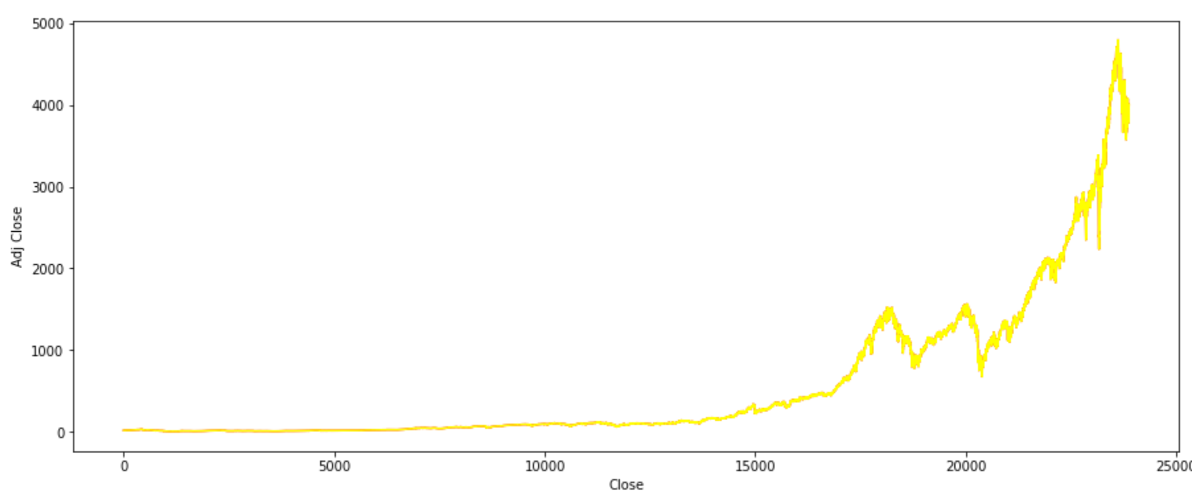
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23874 entries, 0 to 23873
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       23874 non-null  object 
 1   Open       23874 non-null  float64
 2   High       23874 non-null  float64
 3   Low        23874 non-null  float64
 4   Close      23874 non-null  float64
 5   Adj Close  23874 non-null  float64
 6   Volume     23874 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 1.3+ MB
```

In [6]:
```
snpdf.isnull().sum()
```

Out[6]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

There are no null values in the data set.

In [7]:
```
plt.figure(figsize=(15,6))
plt.plot(snpdf['Close'],'red')
plt.plot(snpdf['Adj Close'],'yellow')
plt.xlabel('Close')
plt.ylabel('Adj Close')
plt.show()
```



In [8]:
```
snpdf['Close'].equals(snpdf['Adj Close'])
```
Out[8]:
```
True
```

From both of the tests above, we can see that values of Close and Adj Close are same and hence we can drop one of the columns.

In [9]:
```
del snpdf['Adj Close']
snpdf.head()
```
Out[9]:

	Date	Open	High	Low	Close	Volume
0	1927-12-30	17.660000	17.660000	17.660000	17.660000	0
1	1928-01-03	17.760000	17.760000	17.760000	17.760000	0
2	1928-01-04	17.719999	17.719999	17.719999	17.719999	0
3	1928-01-05	17.549999	17.549999	17.549999	17.549999	0
4	1928-01-06	17.660000	17.660000	17.660000	17.660000	0

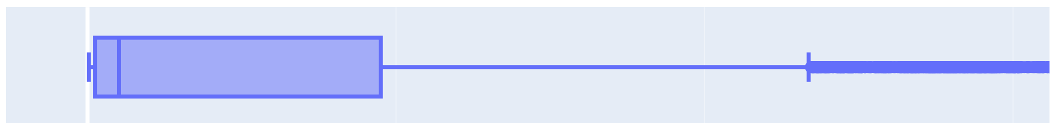
In [10]:
```
snpdf['Date']=pd.to_datetime(snpdf['Date'])
```

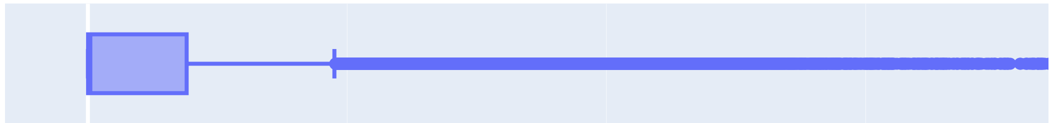
In [11]:
```
fig = px.line(snpdf, x=snpdf['Date'].dt.year, y=snpdf['Close'])
fig.update_layout(title='Closing Price of S&P500 over years',
                  xaxis_title='Year',
                  yaxis_title='Closing Price')
fig.show()
```

Reason behind dip or growth in S&P 500 throughout years

- Rise in 1999: Investors invested a lot of money into interent based startups known as dot-com bubble
- Drop in 2002: Fallout from frenzied investments in internet tech companies and implosion of dot-com bubble
- Drop in 2008: Widespread debt defaults created distrust in stock investment along with Great Recession
- Drop in 2020: The COVID-19 pandemic affected stock market sending the world into recession
- Rise in 2021: Continued federal support, low interest rates, a healthy job market, and massive growth in the largest sector of the U.S. economy - technology

In [12]:
```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

for i, col in enumerate(features):

    duration_box = px.box(
        snpdf,
        x=snpdf[col],
        height=200
    )
    duration_box.show()
```



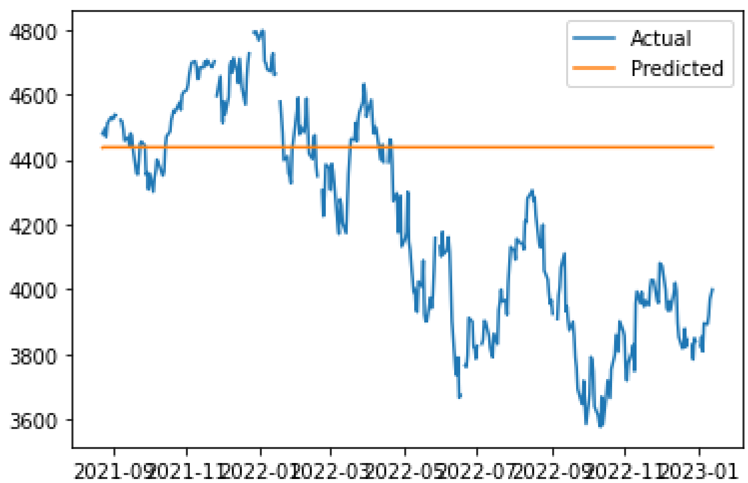


From above boxplots, we can see that only Volume column has outliers values. The rest of the columns doesn't have significant number of outliers.

ARIMA

In [13]:
```
arpdf = snpdf.copy()
arpdf['Date'] = pd.to_datetime(arpdf['Date'])
arpdf.set_index('Date', inplace=True)
arpdf = arpdf.asfreq('B')
train = arpdf.iloc[:-365]
test = arpdf.iloc[-365:]
arimamodel = ARIMA(train['Close'], order=(1,1,1))
model_fit = arimamodel.fit()
predictions = model_fit.predict(start=len(train), end=len(train)+len(tes
t)-1, typ='levels')
plt.plot(test['Close'], label='Actual')
plt.plot(predictions, label='Predicted')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.p
y:574: FutureWarning:

is_monotonic is deprecated and will be removed in a future version. Use
is_monotonic_increasing instead.
```



Linear Regression

In [14]:
```
lrdf=snpdf

# 1 for increasing trend, 0 for same or decreasing trend, as compared to
previous day
lrdf["Trend"] = lrdf["Close"].diff().apply(lambda x:1 if x>0 else 0)
lrdf.dropna(inplace=True)

feature = ["Open", "Close", "High", "Low"]
X = lrdf[feature]
y = lrdf["Trend"]

X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X, y, te
st_size=0.2, shuffle=False)
```

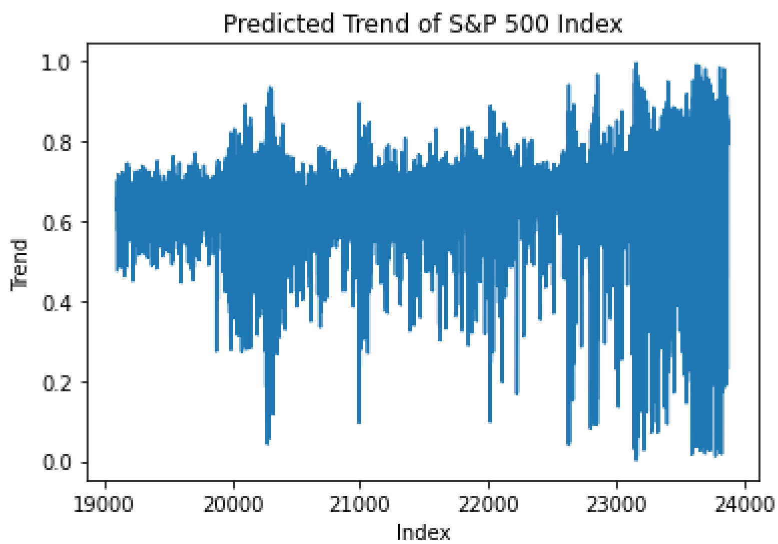
In [15]:
```
lrmodel = LinearRegression()
lrmodel.fit(X_train_lr, y_train_lr)
```

Out[15]: LinearRegression()

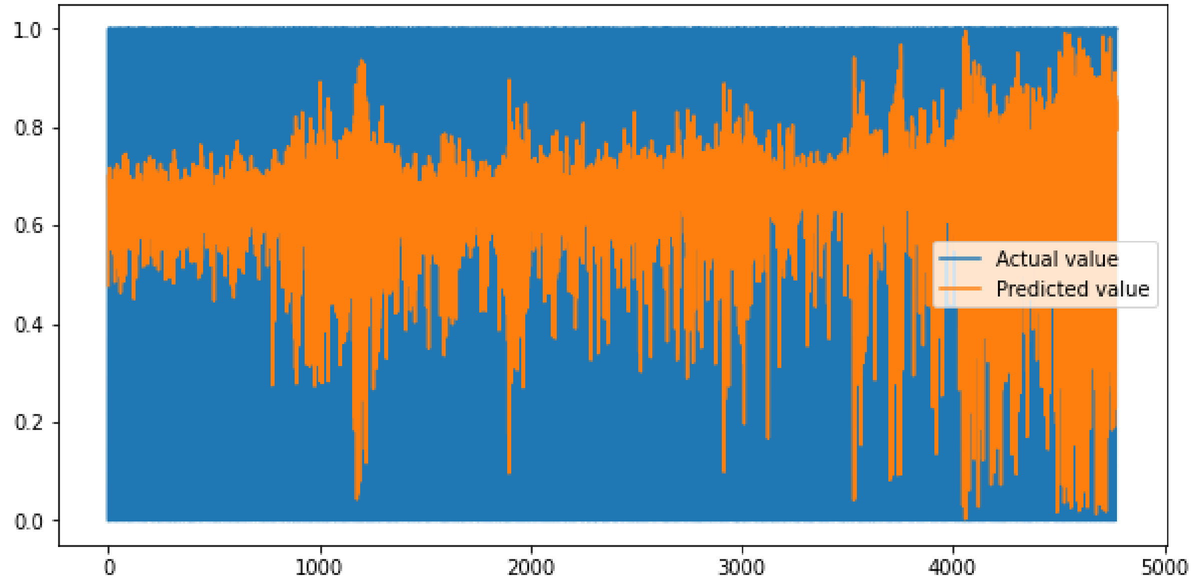
In [16]:
```
y_pred_lr = lrmodel.predict(X_test_lr)
y_pred_lr = np.exp(y_pred_lr) / (1 + np.exp(y_pred_lr)) #Transforming pr
edicted value between 0 and 1
MSE = mean_squared_error(y_test_lr, y_pred_lr)
print("Mean Square Error:", MSE)
```

```
Mean Square Error: 0.19593024156229216
```

In [17]:
```
dates = X_test_lr.index
pred_lrdf = pd.DataFrame({"Date": dates, "Trend": y_pred_lr})

# Plot the predicted trend values
plt.plot(pred_lrdf["Date"], pred_lrdf["Trend"])
plt.title("Predicted Trend of S&P 500 Index")
plt.xlabel("Index")
plt.ylabel("Trend")
plt.show()
```



In [18]:
```
plt.figure(figsize=(10,5))
plt.plot(y_test_lr.values, label="Actual value")
plt.plot(y_pred_lr, label="Predicted value")
plt.legend()
plt.show()
```



kNN

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(snpdf[['Open', 'Lo
w', 'High', 'Volume']], snpdf['Close'], test_size=0.2, random_state=10,
shuffle=False)

# Initialize the regressor and set the number of neighbors (k)
k = 5
knn = KNeighborsRegressor(n_neighbors=k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

mse_knn1 = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: ", mse_knn1)
```

```
Mean Squared Error:  2348137.4819751726
```

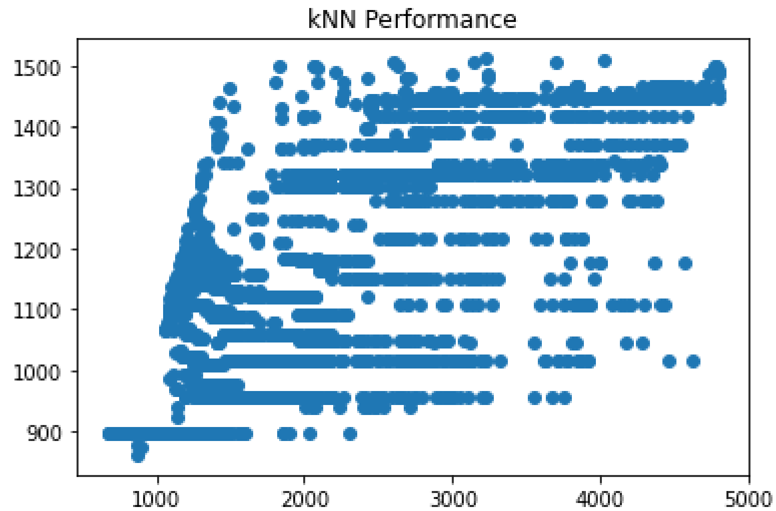
In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(snpdf[['Open', 'Lo
w', 'High', 'Volume']], snpdf['Close'], test_size=0.2, random_state=10,
shuffle=False)

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

k = 5
knn = KNeighborsRegressor(n_neighbors=k)

knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

mse_knn2 = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: ", mse_knn2)
```

```
Mean Squared Error:  1546639.2560510621
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(snpdf[['Open', 'Lo
w', 'High', 'Volume']], snpdf['Close'], test_size=0.2, random_state=10,
shuffle=False)

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    "n_neighbors": [3, 5, 7, 9],
    "weights": ["uniform", "distance"],
    "p": [1, 2]
}

knn = KNeighborsRegressor()

grid_search = GridSearchCV(knn, param_grid, cv=5)

grid_search.fit(X_train_scaled, y_train)

print("Best hyperparameters for kNN:", grid_search.best_params_)

# Use the best hyperparameters to initialize a new KNN regressor and fit
it on the scaled training data
best_knn = KNeighborsRegressor(**grid_search.best_params_)
best_knn.fit(X_train_scaled, y_train)

y_pred = best_knn.predict(X_test_scaled)

mse_knn3 = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: ", mse_knn3)
```

```
Best hyperparameters for kNN: {'n_neighbors': 9, 'p': 2, 'weights': 'un
iform'}
Mean Squared Error:  1565827.4598722989
```

In [22]:
```
plt.scatter(y_test, y_pred)
plt.title('kNN Performance')
plt.show()
```



Lasso Regression

In [23]:
```
lsdf = snpdf
X_train, X_test, y_train, y_test = train_test_split(lsdf[['Open', 'Clos
e', 'Low', 'High']], lsdf['Volume'], test_size=0.3, random_state=42, shu
ffle=False)
```

In [24]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [25]:
```
lasso = Lasso(alpha=0.5, max_iter=100000) # set the regularization param
eter alpha to 0.1
lasso.fit(X_train, y_train)
```

```
/home/nbgrader/spring22/student-accounts/jkovach2/.local/lib/python3.8/
site-packages/sklearn/linear_model/_coordinate_descent.py:647: Converge
nceWarning:

Objective did not converge. You might want to increase the number of it
erations, check the scale of the features or consider increasing regula
risation. Duality gap: 2.330e+18, tolerance: 7.517e+15
```

Out[25]:
```
Lasso(alpha=0.5, max_iter=100000)
```

In [26]:
```
y_pred = lasso.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error: ', mse)
```

```
Mean Squared Error:  5.41005210477516e+18
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(lsdf[['Open', 'Low', 'High', 'Volume']], lsdf['Close'], test_size=0.3, random_state=42, shuffle=False)
```

In [28]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [29]:
```
lasso2 = Lasso(alpha=0.1, max_iter=10000) # set the regularization parameter alpha to 0.1
lasso2.fit(X_train, y_train)
```

Out[29]:
```
Lasso(alpha=0.1, max_iter=10000)
```

In [30]:
```
y_pred = lasso2.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error: ', mse)
```

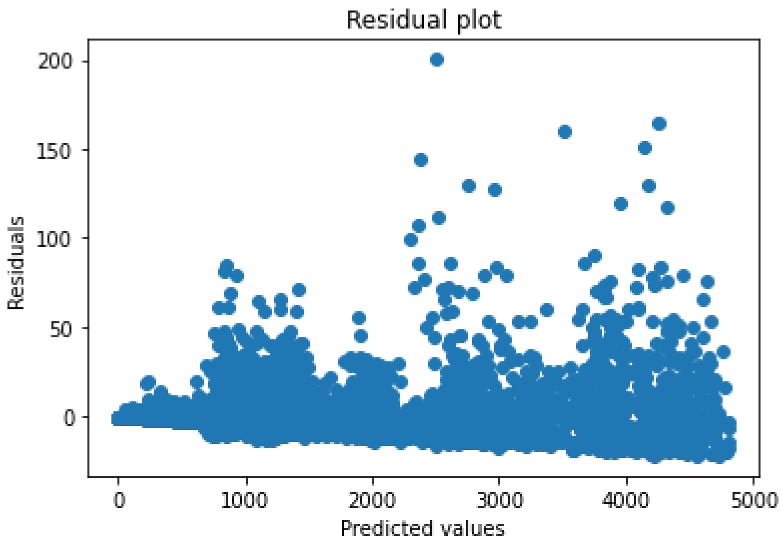
```
Mean Squared Error:  243.26719080932938
```

In [31]:
```
X = snpdf[['Low', 'High', 'Open', 'Volume']].values
y = snpdf['Close'].values


lasso_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('lasso', Lasso())
])

lasso_pipe.named_steps['lasso'].shuffle = False
param_grid = {'lasso__alpha': np.logspace(-5, 1, 100)}

kf = KFold(n_splits=5, shuffle=False)
grid_search = GridSearchCV(lasso_pipe, param_grid, cv=kf, scoring='neg_mean_squared_error')
grid_search.fit(X, y)

print("Best alpha:", grid_search.best_params_['lasso__alpha'])
print("Mean squared error:", -grid_search.best_score_)
```

```
Best alpha: 1.232846739442066
Mean squared error: 51.95472246634172
```

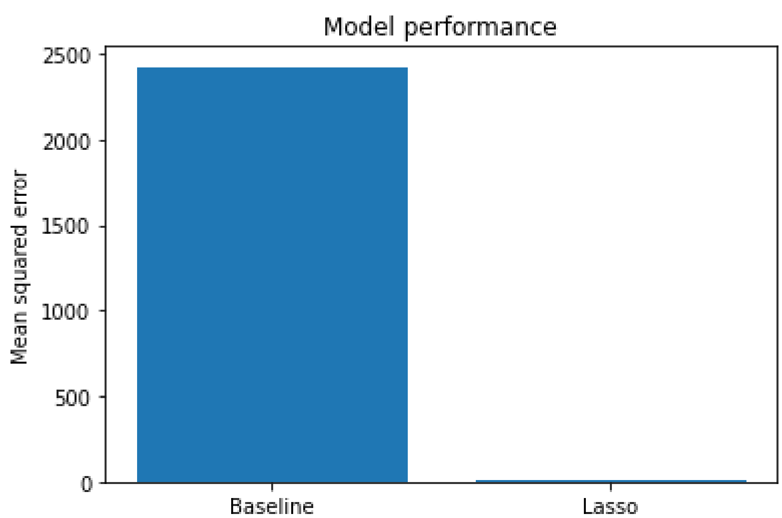
In [32]:
```python
y_pred = grid_search.predict(X)
residuals = y - y_pred

plt.scatter(y_pred, residuals)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.title('Residual plot')
```

Out[32]: Text(0.5, 1.0, 'Residual plot')



In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0, shuffle = False)


y_pred = np.roll(y_test, 1)


mse_baseline = mean_squared_error(y_test, y_pred)


grid_search.fit(X_train, y_train)


y_pred = grid_search.predict(X_test)


mse_lasso = mean_squared_error(y_test, y_pred)


plt.bar(['Baseline', 'Lasso'], [mse_baseline, -grid_search.best_score_])
plt.ylabel('Mean squared error')
plt.title('Model performance')
```

Out[33]:
```
Text(0.5, 1.0, 'Model performance')
```



In [34]:
```
import joblib
joblib.dump(grid_search.best_estimator_, 'lasso_model.pkl')
```

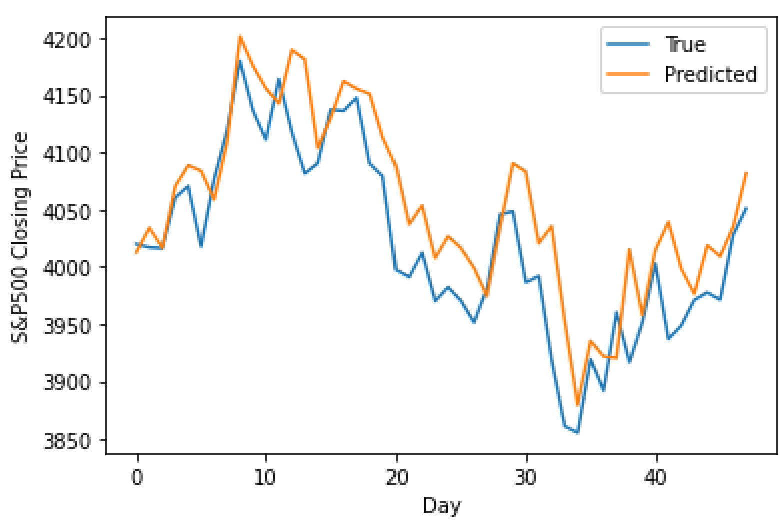
Out[34]:
```
['lasso_model.pkl']
```

In [35]:
```
lasso_model = joblib.load('lasso_model.pkl')
```

In [36]:
```
sp500 = yf.download('^GSPC', start='2023-01-23', end='2023-03-31')

del sp500['Adj Close']


X_new = sp500.drop(['Close'], axis=1)
y_pred = lasso_model.predict(X_new)


y_true = sp500['Close'].values
mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))


print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
```

```
[*********************100%***********************]  1 of 1 completed
MAE: 38.44
RMSE: 49.04

/home/nbgrader/spring22/student-accounts/jkovach2/.local/lib/python3.8/
site-packages/sklearn/base.py:443: UserWarning:

X has feature names, but StandardScaler was fitted without feature name
s
```

In [37]:
```
plt.plot(y_true, label='True')
plt.plot(y_pred, label='Predicted')


plt.xlabel('Day')
plt.ylabel('S&P500 Closing Price')
plt.legend()


plt.show()
```



Random forest

In [38]:
```
X=snpdf.drop(['Close', 'Date'], axis=1)
y=snpdf['Close']

#Splitting into training(80%) and testing data
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X, y, te
st_size=0.2, random_state=np.random.seed(40), shuffle=False)
```

In [39]:
```
#Creating random forest model
rfmodel = RandomForestRegressor(n_estimators=100, random_state=np.rando
m.seed(40))
```

In [40]:
```
#Training the model
rfmodel.fit(X_train_rf, y_train_rf)
```

Out[40]:
```
RandomForestRegressor()
```

In [41]:
```
predictions_rf = rfmodel.predict(X_test_rf)
```

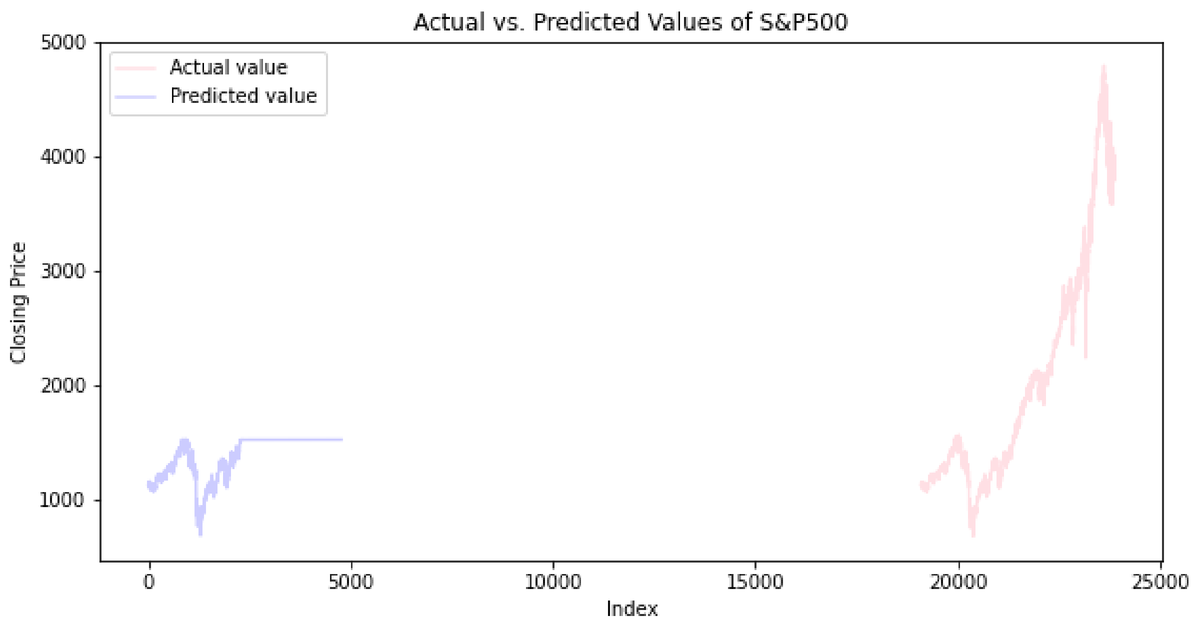
In [42]:
```
MSE_rf = mean_squared_error(y_test_rf, predictions_rf)
print('Mean squared error:', MSE_rf)
```

```
Mean squared error: 1194173.6016176166
```

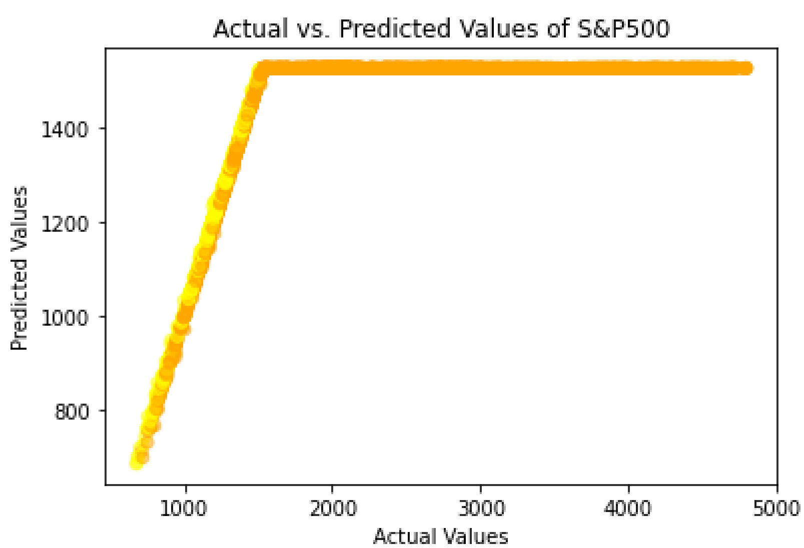
In [43]:
```
r2_rf = r2_score(y_test_rf, predictions_rf)
print('R squared:', r2_rf)
```

```
R squared: -0.20686592248754931
```

In [44]:
```
plt.figure(figsize=(10,5))
plt.plot(y_test_rf,color='pink', label='Actual value', alpha=0.5)
plt.plot(predictions_rf,color='blue', label='Predicted value', alpha=0.
2)
plt.xlabel('Index')
plt.ylabel('Closing Price')
plt.title('Actual vs. Predicted Values of S&P500')
plt.legend()
plt.show()
```



In [45]:
```
plt.scatter(y_test_rf,predictions_rf,  c=['yellow' if x < y else 'orang
e' for x, y in zip(y_test_rf, predictions_rf)], alpha=0.5 )
#yellow-predicted
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values of S&P500')
plt.show()
```



In [46]:
```
param_dist = {'n_estimators': randint(50, 500),
              'max_features': ['auto', 'sqrt', 'log2'],
              'max_depth': [10, 20, 30, 40, None],
              'min_samples_split': randint(2, 20),
              'min_samples_leaf': randint(1, 10)}
random_search = RandomizedSearchCV(rfmodel, param_distributions=param_di
st, n_iter=50,
                                   n_jobs=-1, cv=5, random_state=np.rand
om.seed(40))
```

In [47]:
```
random_search.fit(X_train_rf, y_train_rf)
print('Best hyperparameters:', random_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 40, 'max_features': 'log2', 'min_sa
mples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 107}
```

In [48]:
```
#Random forest model using best parameters
#New shuffle
rfmodel_best = RandomForestRegressor(n_estimators=230, max_features='sqr
t', max_depth=30,
                                     min_samples_split=10, min_samples_leaf=
2, random_state=np.random.seed(40))
```

In [49]:
```
rfmodel_best.fit(X_train_rf, y_train_rf)
```

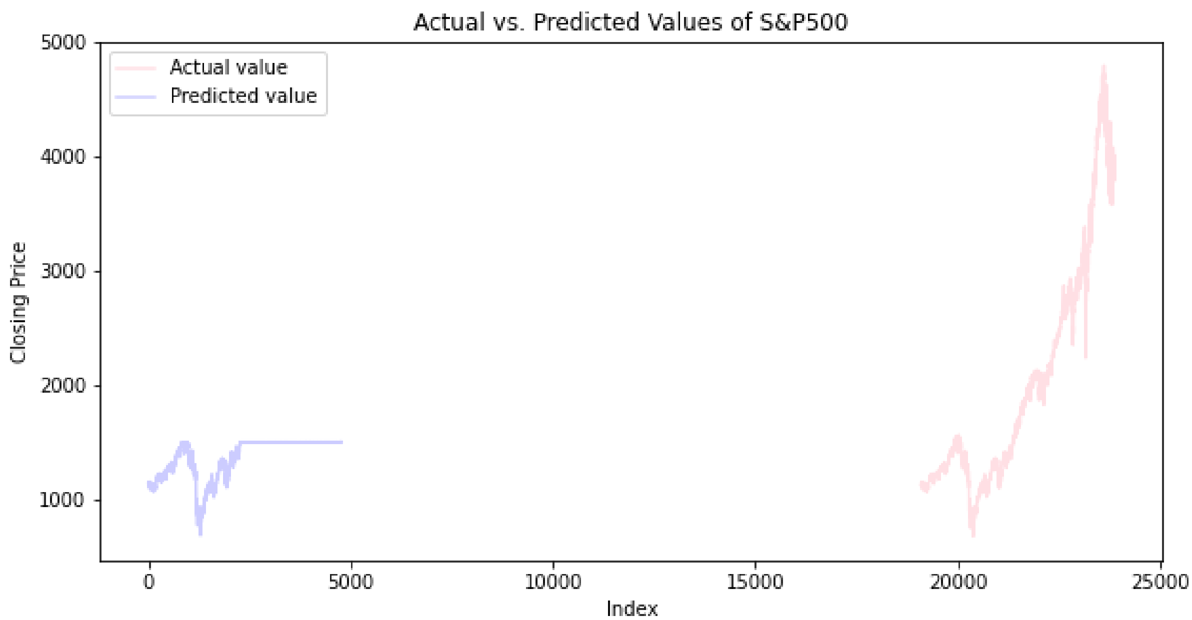
Out[49]:
```
RandomForestRegressor(max_depth=30, max_features='sqrt', min_samples_le
af=2,
                      min_samples_split=10, n_estimators=230)
```

In [50]:
```
predictions_rfbest = rfmodel_best.predict(X_test_rf)
```

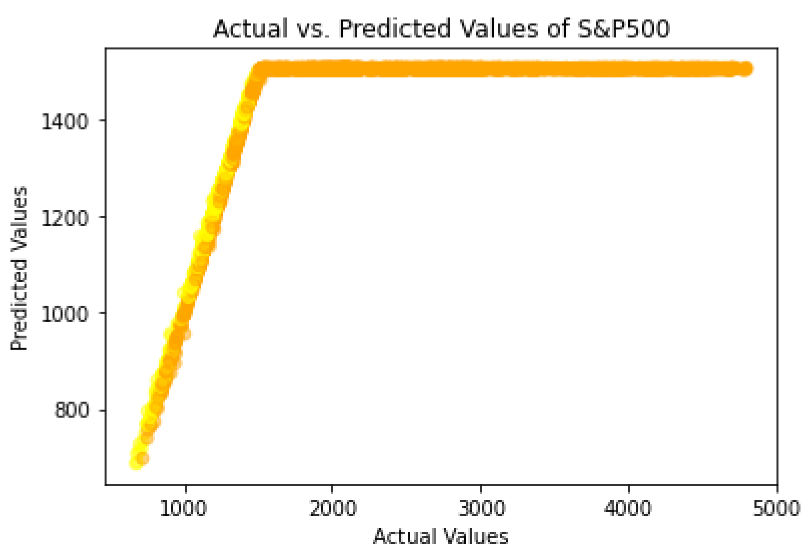
In [51]:
```
MSE_rfbest = mean_squared_error(y_test_rf, predictions_rfbest)
print('Mean squared error:', MSE_rfbest)

r2_rfbest = r2_score(y_test_rf, predictions_rfbest)
print('R squared:', r2_rfbest)
```

```
Mean squared error: 1220410.749592023
R squared: -0.23338193301623744
```

In [52]:
```
plt.figure(figsize=(10,5))
plt.plot(y_test_rf,color='pink', label='Actual value', alpha=0.5)
plt.plot(predictions_rfbest,color='blue', label='Predicted value', alpha
=0.2)
plt.xlabel('Index')
plt.ylabel('Closing Price')
plt.title('Actual vs. Predicted Values of S&P500')
plt.legend()
plt.show()
```



In [53]:
```
plt.scatter(y_test_rf,predictions_rfbest,  c=['yellow' if x < y else 'or
ange' for x, y in zip(y_test_rf, predictions_rfbest)], alpha=0.5 )
#yellow-predicted
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values of S&P500')
plt.show()
```



LSTM-Long Short Term Memory Neural Network Model

In [54]:
```
lstmdf = snpdf

lstmdf['Date'] = pd.to_datetime(lstmdf['Date'])

lstmdf['Year'] = lstmdf['Date'].dt.year
lstmdf['Month'] = lstmdf['Date'].dt.month
lstmdf['Day'] = lstmdf['Date'].dt.day

lstmdf = lstmdf.drop(columns=['Date'])

lstmdf.head()
```

Out[54]:

	Open	High	Low	Close	Volume	Trend	Year	Month	Day
0	17.660000	17.660000	17.660000	17.660000	0	0	1927	12	30
1	17.760000	17.760000	17.760000	17.760000	0	1	1928	1	3
2	17.719999	17.719999	17.719999	17.719999	0	0	1928	1	4
3	17.549999	17.549999	17.549999	17.549999	0	0	1928	1	5
4	17.660000	17.660000	17.660000	17.660000	0	1	1928	1	6

In [55]:
```
#Scaling data
scaler = MinMaxScaler()
scaling_data = scaler.fit_transform(lstmdf)
```

In [56]:
```
#Splitting into training(80%) and testing data
train_size = int(len(scaling_data) * 0.8)
train_data = scaling_data[:train_size, :]
test_data = scaling_data[train_size:, :]
```

In [57]:
```
#Creating sequence

def create_sequence(data, length):
    X = []
    y = []
    for i in range(len(data)-length):
        X.append(data[i:i+length])
        y.append(data[i+length])
    return np.array(X), np.array(y)

length = 10

X_train, y_train = create_sequence(train_data, length)
X_test, y_test = create_sequence(test_data,length)
```

In [58]:

```
#LSTM model
random.set_seed(452)
lstmmodel = Sequential()

lstmmodel.add(LSTM(50,input_shape=(length, lstmdf.shape[1])))
lstmmodel.add(Dense(9)) #Dense(8)
lstmmodel.compile(loss='mean_squared_error', optimizer='adam')
```

In [59]:

```
#Training the model
history = lstmmodel.fit(X_train, y_train, epochs=15, batch_size=2, valid
ation_split=0.1, verbose=1)
```

```
Epoch 1/15
8590/8590 [==============================] - 50s 6ms/step - loss: 0.032
6 - val_loss: 0.0320
Epoch 2/15
8590/8590 [==============================] - 49s 6ms/step - loss: 0.030
9 - val_loss: 0.0317
Epoch 3/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.030
5 - val_loss: 0.0331
Epoch 4/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.030
4 - val_loss: 0.0352
Epoch 5/15
8590/8590 [==============================] - 49s 6ms/step - loss: 0.030
3 - val_loss: 0.0324
Epoch 6/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.030
1 - val_loss: 0.0323
Epoch 7/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.030
1 - val_loss: 0.0340
Epoch 8/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.029
9 - val_loss: 0.0328
Epoch 9/15
8590/8590 [==============================] - 49s 6ms/step - loss: 0.029
9 - val_loss: 0.0330
Epoch 10/15
8590/8590 [==============================] - 49s 6ms/step - loss: 0.029
8 - val_loss: 0.0336
Epoch 11/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.029
8 - val_loss: 0.0317
Epoch 12/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.029
7 - val_loss: 0.0325
Epoch 13/15
8590/8590 [==============================] - 49s 6ms/step - loss: 0.029
6 - val_loss: 0.0320
Epoch 14/15
8590/8590 [==============================] - 50s 6ms/step - loss: 0.029
4 - val_loss: 0.0333
Epoch 15/15
8590/8590 [==============================] - 48s 6ms/step - loss: 0.029
4 - val_loss: 0.0327
```

In [60]:
```
score = lstmmodel.evaluate(X_test, y_test)
print("Error in testing:",score)
print("Accuracy of LSTM model",(1-score)*100)
```

```
149/149 [==============================] - 0s 3ms/step - loss: 0.0471
Error in testing: 0.047074880450963974
Accuracy of LSTM model 95.2925119549036
```

In [61]:
```
predictions = lstmmodel.predict(X_test)
# predictions = scaler.inverse_transform(predictions)
# y_test = scaler.inverse_transform(y_test)
```

In [62]:
```
MSE = mean_squared_error(y_test, predictions)
print("Mean Square Error:", MSE)
```

```
Mean Square Error: 0.047074876936049476
```

In [63]:
```
plt.figure(figsize=(10,5))
plt.plot(y_test,color='yellow', label='Actual value', alpha=0.5)
plt.plot(predictions,color='black', label='Predicted value', alpha=0.5)
plt.xlabel('Year')
plt.ylabel('S&P500 Index')
plt.legend()
plt.show()
```