```
import pandas

filename = "normalized_2.xlsx"

df = pandas.read_excel(filename, index_col=0)  # read an Excel spreadsheet

# shift by specified # days and join to original dataframe
#deniz changed num_days to 0 here
num_days = 0
df1 = 0
dfout = df
for n in range(1,num_days+1):
    df1 = df.shift(periods = n).add_suffix(str(n))
    dfout = dfout.join(df1)
    df1 = 0

# drop rows that now have NA values
dfout.drop(dfout.index[range(num_days)], axis=0, inplace=True)

# create target column
t = 28 # number days ahead to predict
dfout['Target'] = dfout['Close'].shift(periods = -t)
dfout.reset_index(inplace=True)
dfout.drop(['index'], axis=1, inplace=True)

# again drop rows that now have NA values
dfout.drop(dfout.index[range(-1,(-t-1),-1)], axis=0, inplace=True)
#print(dfout)

dfout.to_excel("shifted_new.xlsx")


#linear regression and decision tree regressor models will be created in the followin

#df_lin holds the dataset we will work on
df_lin=pandas.read_excel('shifted_new.xlsx')


#checking for any NaN at the end
df_lin.tail(20)
```

	Unnamed: 0	Open	High	Low	Close	Adj Close	Volume	Close_Gc
1212	1212	0.682891	0.637323	0.640907	0.652987	0.662807	0.348066	0.774
1213	1213	0.643552	0.604656	0.630920	0.629827	0.640782	0.131233	0.759
1214	1214	0.634779	0.625373	0.649041	0.665144	0.674367	0.201411	0.776
1215	1215	0.665946	0.652919	0.675660	0.682822	0.691179	0.154790	0.811
1216	1216	0.684894	0.645587	0.674291	0.655570	0.665263	0.292183	0.808
1217	1217	0.674358	0.634103	0.657418	0.653504	0.663298	0.115152	0.817
1218	1218	0.664463	0.626693	0.649203	0.635031	0.645731	0.102668	0.820
1219	1219	0.677362	0.658894	0.669942	0.672414	0.681281	0.310074	0.838
1220	1220	0.668229	0.690630	0.683433	0.728746	0.741019	0.242295	0.795
1221	1221	0.752233	0.814239	0.766350	0.857739	0.864176	0.558223	0.809
1222	1222	0.872491	0.869137	0.886316	0.913197	0.917125	0.471823	0.858
1223	1223	0.902295	0.870883	0.855469	0.902431	0.906846	0.365581	0.834
1224	1224	0.904819	0.861377	0.876852	0.887732	0.892812	0.281345	0.849
1225	1225	0.911789	0.930630	0.911646	0.962736	0.964422	0.340026	0.884
1226	1226	0.985018	1.000000	1.000000	1.000000	1.000000	0.510732	0.917

```
# few arrangements on the dataset
df_lin.set_index('Date', inplace=True)
```

```
df_lin = df_lin.drop(['Unnamed: 0'],1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: I
  """Entry point for launching an IPython kernel.
```

```
df_lin
```

Date	Open	High	Low	Close	Adj Close	Volume	Close_Gold	Ta
2017-04-24	0.089933	0.064714	0.096448	0.091014	0.058994	0.195251	0.107316	0.12
2017-04-25	0.086528	0.064209	0.095643	0.087001	0.055460	0.307911	0.095705	0.1
2017-04-26	0.093218	0.065684	0.078206	0.086723	0.055215	0.358317	0.092323	0.1(
2017-04-27	0.089853	0.064985	0.095764	0.088074	0.056404	0.183590	0.094240	0.1

```
df_lin.shape
```

```
(1232, 8)
```

2022-03-08	1.000000	0.949447	0.904357	0.885825	0.890991	0.521641	0.970578	0.95

```
# visualizing the dataset in terms of days and closing prices
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
# visualizing closing prices for each day in dataset
plt.figure(figsize=(16,8))
plt.title('stock market')
plt.xlabel('Days')
plt.ylabel('Close Price')
plt.plot(df_lin['Close'])
plt.show()
```


```
#creating the feauture set and target set
X = np.array(df_lin.drop(['Target'],1))
y = np.array(df_lin['Target'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: I
```

```
X
```

```
array([[0.08993309, 0.06471384, 0.09644812, ..., 0.05899394, 0.19525097,
        0.10731597],
       [0.08652808, 0.06420946, 0.09564272, ..., 0.05545961, 0.30791113,
        0.09570511],
       [0.09321798, 0.06568377, 0.07820553, ..., 0.05521468, 0.35831668,
        0.0923233 ],
       ...,
       [0.89716775, 0.89253148, 0.87874511, ..., 0.87354391, 0.28529446,
        0.92221846],
       [0.88591111, 0.84108624, 0.8539787 , ..., 0.84202474, 0.22252263,
        0.90485853],
       [0.8523414 , 0.81311343, 0.83601804, ..., 0.86144459, 0.16085577,
        0.87757863]])
```

```
#split data to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
```

```
#lr is the linear regression model
lr = LinearRegression().fit(X_train,y_train)
```

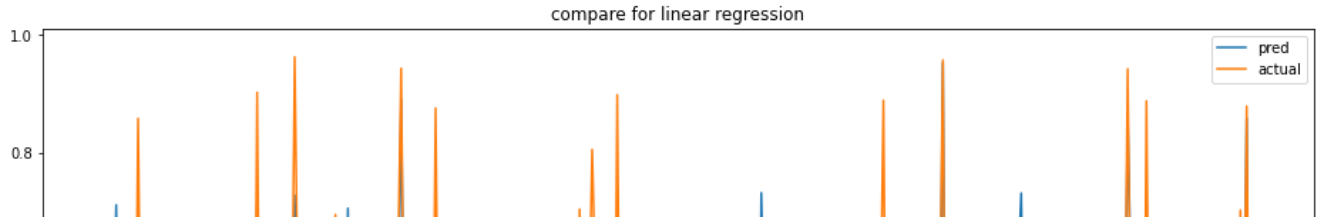
```
#evaluate linear regressions performnace
predictions = lr.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

```
0.08975055732840652
```

```
#creating this dataframe so that it will be useful for graph creation in the followin
df1 = pandas.DataFrame({"predictions": predictions, "actual": y_test})
df1
```

	predictions	actual
0	0.555881	0.581360
1	0.395161	0.463134
2	0.430880	0.304942
3	0.338516	0.259932
4	0.483400	0.562411
...	...	...
365	0.374605	0.393175
366	0.857806	0.878635
367	0.173374	0.216073
368	0.292155	0.319641

```
#comparing actual values with estimated ones by help of a graph
plt.figure(figsize=(16,8))
plt.title('compare for linear regression')
plt.xlabel('days')
plt.ylabel('close price')

plt.plot(df1[['predictions', 'actual']])
plt.legend(['pred','actual'])
plt.show()
```




```
#decision tree model
from sklearn.tree import DecisionTreeRegressor

#split data to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state

tree = DecisionTreeRegressor().fit(X_train, y_train)
```

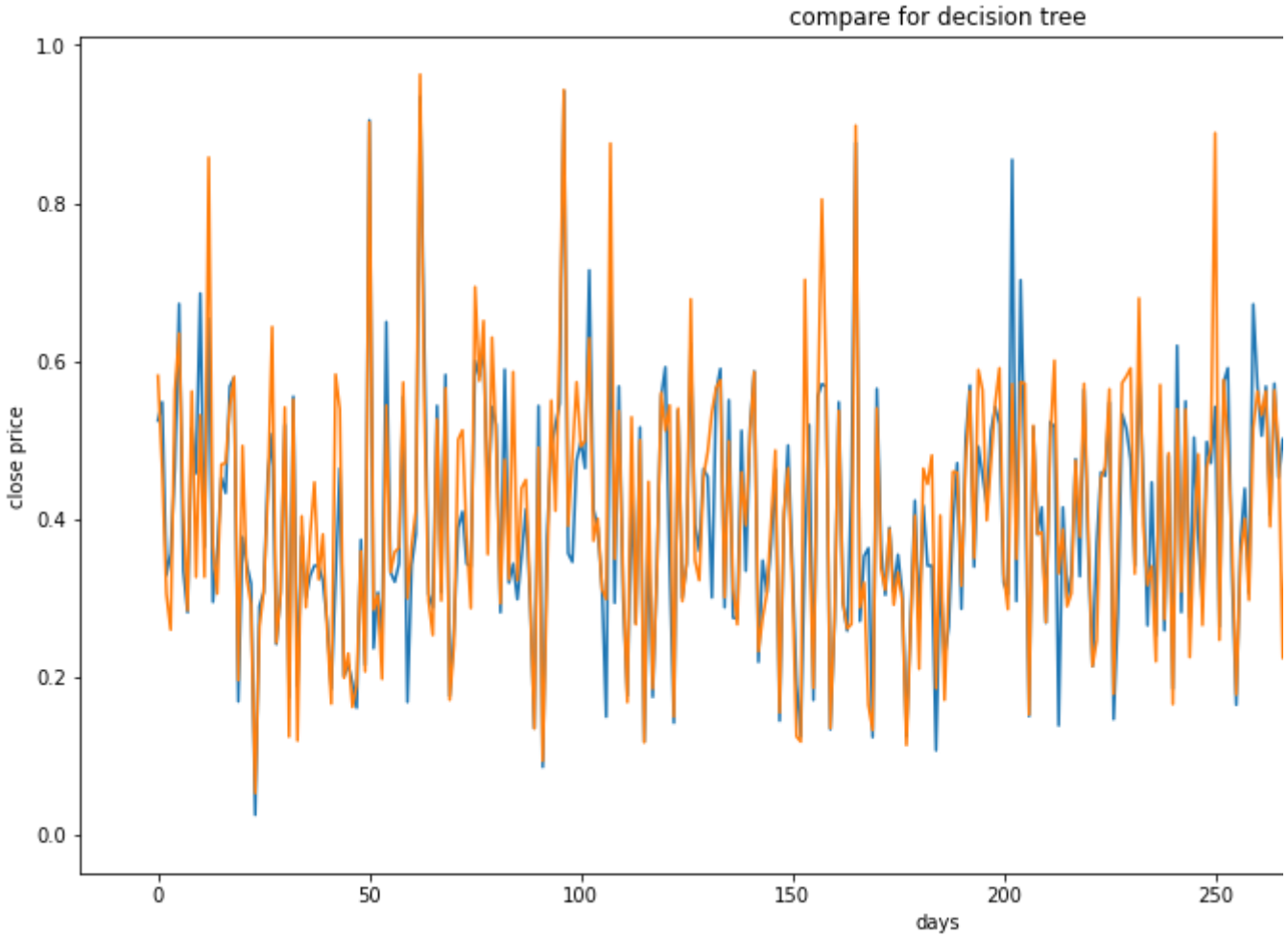
```
prediction_tree = tree.predict(X_test)


df2 = pandas.DataFrame({"predictions": prediction_tree, "actual": y_test})
df2
```

	predictions	actual
0	0.524670	0.581360
1	0.548030	0.463134
2	0.329016	0.304942
3	0.356746	0.259932
4	0.464087	0.562411
...	...	...
365	0.393374	0.393175
366	0.862943	0.878635
367	0.219450	0.216073
368	0.337995	0.319641
369	0.507985	0.604878

370 rows × 2 columns

```
#comparing actual values with estimated ones by help of a graph
plt.figure(figsize=(16,8))
plt.title('compare for decision tree')
plt.xlabel('days')
plt.ylabel('close price')
```

```
plt.plot(df2[['predictions', 'actual']])
plt.legend(['pred','actual'])
plt.show()
```



✓ 0 sn. tamamlanma zamanı: 16:49