Data Loading

```
1 import pandas as pd
```

```
1 btc = pd.read_csv('https://github.com/JesperDramsch/skillshare-financial-prediction/raw
2 btc['Timestamp'] = pd.to_datetime(btc.Timestamp)
3 btc.set_index('Timestamp', inplace=True)
4 btc.head()
5
```

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	W
2011-12-31	4.465000	4.482500	4.465000	4.482500	23.829470	106.330084	
2012-01-01	4.806667	4.806667	4.806667	4.806667	7.200667	35.259720	
2012-01-02	5.000000	5.000000	5.000000	5.000000	19.048000	95.240000	
2012-01-							

Describe Data

```
1 btc.head()
```

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	W
2011-12-31	4.465000	4.482500	4.465000	4.482500	23.829470	106.330084	
2012-01-01	4.806667	4.806667	4.806667	4.806667	7.200667	35.259720	
2012-01-02	5.000000	5.000000	5.000000	5.000000	19.048000	95.240000	
2012-01-							

```
1 btc.tail()
2
```

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volu
2021-03-27	55193.240643	55219.665031	55168.757372	55195.415367	1.823877	
2021-03-28	55833.608471	55857.735342	55810.425126	55835.012863	1.447939	

Columns/features in data

```
1 btc.columns
2
```

```
Index(['Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)',
       'Weighted_Price'],
      dtype='object')
```

Data information

```
1 btc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3379 entries, 2011-12-31 to 2021-03-31
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Open               3376 non-null   float64
 1   High               3376 non-null   float64
 2   Low                3376 non-null   float64
 3   Close              3376 non-null   float64
 4   Volume_(BTC)       3376 non-null   float64
 5   Volume_(Currency)  3376 non-null   float64
 6   Weighted_Price     3376 non-null   float64
dtypes: float64(7)
memory usage: 211.2 KB
```

Is there any missing values?

```
1 btc.isnull().values.any()
2
```

```
True
```

```
1 btc[btc.isnull().any(axis=1)].head()
2
```

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2015-01-06	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-01-	.. ..	.. ..	.. ..	.. ..	.. ..	.. ..	.. ..

```
1 btc = btc.dropna()
2 btc
3
```

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volu
2011-12-31	4.465000	4.482500	4.465000	4.482500	23.829470	
2012-01-01	4.806667	4.806667	4.806667	4.806667	7.200667	
2012-01-02	5.000000	5.000000	5.000000	5.000000	19.048000	
2012-01-03	5.252500	5.252500	5.252500	5.252500	11.004660	
2012-01-04	5.200000	5.223333	5.200000	5.223333	11.914807	
...	...	...	...	...	...	
2021-03-27	55193.240643	55219.665031	55168.757372	55195.415367	1.823877	
2021-03-28	55833.608471	55857.735342	55810.425126	55835.012863	1.447939	

```
1 btc.isnull().values.any()
```

```
False
```

Data Description

```
1 btc.describe()
```

	Open	High	Low	Close	Volume_(BTC)	Volume_(
count	3376.000000	3376.000000	3376.000000	3376.000000	3376.000000	33
mean	4605.644798	4608.916329	4602.220278	4605.640430	10.355675	317
std	8207.258774	8213.768034	8200.562238	8207.368264	8.897358	627
min	4.331667	4.331667	4.331667	4.331667	0.250000	

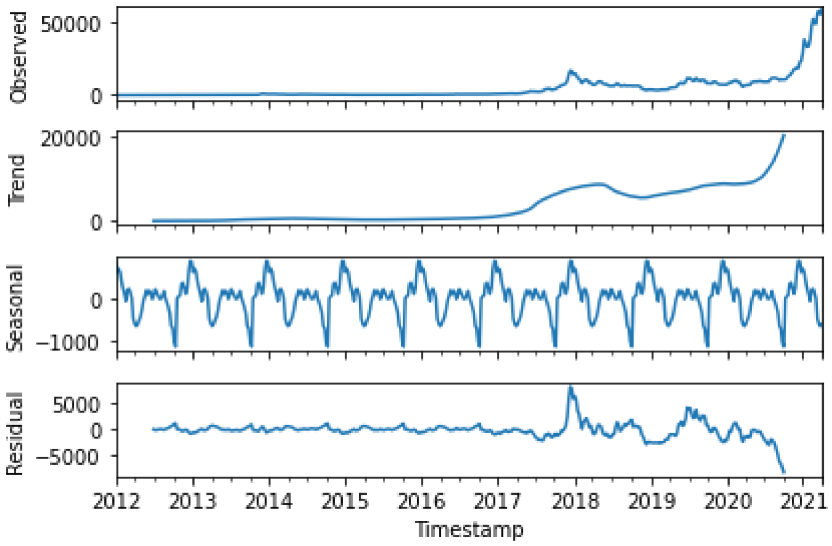
Statistical Test

```
1 import statsmodels.api as sm
2 import matplotlib.pyplot as plt
3 import numpy as np
4
```

```
1 btc["High_log"] = np.log1p(btc.High)
```

```
1 plt.figure(figsize=[20,10])
2 sm.tsa.seasonal_decompose(btc.resample("W").median().High).plot()
3 plt.show()
4
```

```
<Figure size 1440x720 with 0 Axes>
```




```
1 dickey = sm.tsa.stattools.adfuller(btc.resample("W").mean().High)
2
3
4 print(f'ADF Statistic: {dickey[0]:.2f}')
5 print(f'p-value: {dickey[1]:.2f}')
6 print('Critical Values:')
7 for key, value in dickey[4].items():
8         print(f'\t {key}: {value:.3f}')
9
```

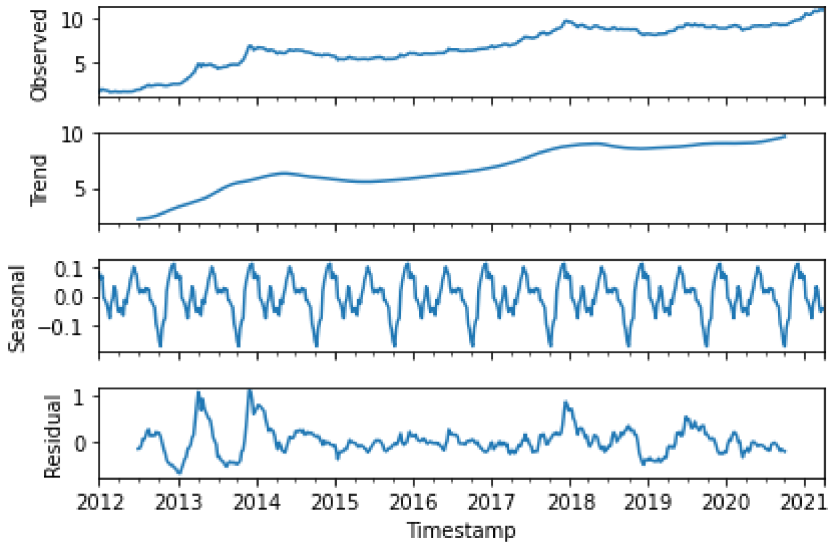
```
ADF Statistic: 2.52
```

```
    p-value: 1.00
    Critical Values:
             1%: -3.444
             5%: -2.868
             10%: -2.570
```

```
plt.figure(figsize=[20,10])
sm.tsa.seasonal_decompose(btc.resample("W").median().High_log).plot()
plt.show()

```

```
    <Figure size 1440x720 with 0 Axes>
```




```
dickey = sm.tsa.stattools.adfuller(btc.resample("W").mean().High_log)


print(f'ADF Statistic: {dickey[0]:.2f}')
print(f'p-value: {dickey[1]:.2f}')
print('Critical Values:')
for key, value in dickey[4].items():
        print(f'\t {key}: {value:.3f}')

```

```
    ADF Statistic: -0.97
    p-value: 0.76
    Critical Values:
             1%: -3.444
             5%: -2.868
             10%: -2.570
```

```
btc.columns
```

```
    Index(['Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)',
           'Weighted_Price', 'High_log'],
          dtype='object')
```

```
btc.drop(['Open', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)',
       'Weighted_Price'], axis='columns', inplace=True)
```

Prepare Training data

Classic Machine Learning

```
from sklearn.preprocessing import MinMaxScaler
```

```
split = int(btc.shape[0]*0.8)
df_train = btc[:split]
df_test = btc[split:]

```

```
def create_dataset(df, n, feature=0):
    x = []
    y = []

    for i in range(n, df.shape[0]):
        x.append(df[i-n:i, feature])
        y.append(df[i, feature])
    x = np.expand_dims(np.array(x), -1)
    y = np.expand_dims(np.array(y), -1)
    return x, y
```

```
scaler = MinMaxScaler(feature_range=(0,1))
dataset_train = scaler.fit_transform(df_train)
dataset_train

```

```
array([[7.85580963e-06, 3.40656059e-03],
       [2.47392900e-05, 1.04212680e-02],
       [3.48086151e-05, 1.44207334e-02],
       ...,
       [4.01240116e-01, 8.88539676e-01],
       [4.15818387e-01, 8.92894669e-01],
       [4.19060213e-01, 8.93842352e-01]])
```

```
dataset_test = scaler.transform(df_test)

```

```
samples, feature = 10, 1
X_train, y_train = create_dataset(dataset_train,samples, feature)
X_test, y_test = create_dataset(dataset_test,samples, feature)

```

```
X_train.shape
```

```
(2690, 10, 1)
```

```
y_train.shape
```

```
(2690, 1)
```

Evaluation function

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

```
def evaluate_model(model, model_name="Model", test_data=X_test, target_data=y_test):
    y_pred_test = model.predict(test_data)

    try:
        y_pred_test = y_pred_test.yhat
    except:
        pass

    test_rs = r2_score(target_data,y_pred_test)
    print('R Squared : ', round(test_rs,5))

    test_MAE = mean_absolute_error(target_data, y_pred_test)
    print('Mean Absolute Error: ', round(test_MAE, 5))

    plt.figure(figsize=(20,10))
    plt.plot(y_pred_test, color='green', marker='o', linestyle='dashed',label='Predicte
    plt.plot(target_data, color='red', label='Actual Price')
    plt.title('Comparison of actual and predicted stock prices for ' + model_name)
    plt.xlabel('Day')
    plt.ylabel('Prices')
    plt.legend()
    plt.show()

    return test_rs, test_MAE
```

LSTM

```
from tensorflow.keras.layers import LSTM, Activation, Dense, Dropout, Conv1D, MaxPoolir
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import MSE
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.activations import relu
```

```
LSTM_Model = Sequential()

```

```
LSTM_Model.add(LSTM(units=96,
                    return_sequences=True,
                    input_shape=(X_train.shape[1], 1)))
```

```
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(LSTM(units=96,
                    return_sequences=True))
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(LSTM(units=96,
                    return_sequences=True))
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(LSTM(units=96))
LSTM_Model.add(Dropout(0.2))
LSTM_Model.add(Dense(units=1))
```

```
LSTM_Model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 10, 96)            37632

 dropout_4 (Dropout)         (None, 10, 96)            0

 lstm_5 (LSTM)               (None, 10, 96)            74112

 dropout_5 (Dropout)         (None, 10, 96)            0

 lstm_6 (LSTM)               (None, 10, 96)            74112

 dropout_6 (Dropout)         (None, 10, 96)            0

 lstm_7 (LSTM)               (None, 96)                74112

 dropout_7 (Dropout)         (None, 96)                0

 dense_1 (Dense)             (None, 1)                 97

=================================================================
Total params: 260,065
Trainable params: 260,065
Non-trainable params: 0
_________________________________________________________________
```

```
LSTM_Model.compile(loss=MSE, optimizer=Adam())

```

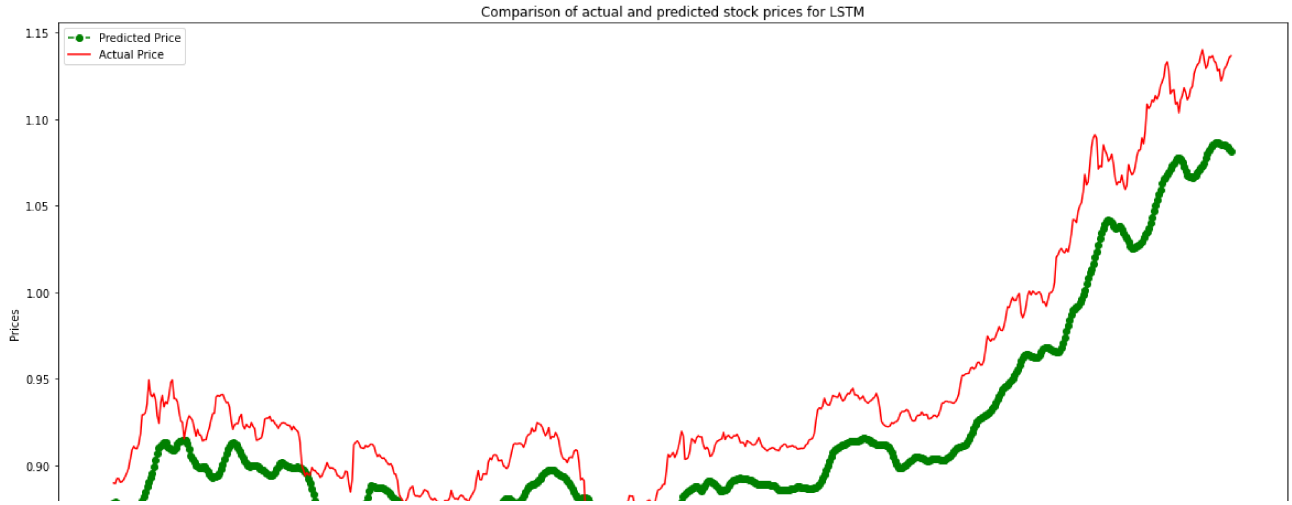
```
LSTM_Model.fit(X_train,y_train,batch_size=32,epochs=20,verbose=1,validation_split=0.05)
```

```
Epoch 1/20
80/80 [==============================] - 11s 42ms/step - loss: 0.0219 - val_loss: 3.
Epoch 2/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0023 - val_loss: 1.0
Epoch 3/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0019 - val_loss: 3.8
Epoch 4/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0017 - val_loss: 2.1
Epoch 5/20
```

```
80/80 [==============================] - 2s 20ms/step - loss: 0.0020 - val_loss: 1.7
Epoch 6/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0019 - val_loss: 3.0
Epoch 7/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0016 - val_loss: 1.0
Epoch 8/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0014 - val_loss: 1.0
Epoch 9/20
80/80 [==============================] - 2s 21ms/step - loss: 0.0014 - val_loss: 4.3
Epoch 10/20
80/80 [==============================] - 2s 22ms/step - loss: 0.0016 - val_loss: 1.3
Epoch 11/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0015 - val_loss: 4.4
Epoch 12/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0016 - val_loss: 9.6
Epoch 13/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0014 - val_loss: 9.2
Epoch 14/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0014 - val_loss: 0.0
Epoch 15/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0015 - val_loss: 0.0
Epoch 16/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0013 - val_loss: 1.5
Epoch 17/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0013 - val_loss: 3.4
Epoch 18/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0013 - val_loss: 4.6
Epoch 19/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0013 - val_loss: 1.2
Epoch 20/20
80/80 [==============================] - 2s 20ms/step - loss: 0.0013 - val_loss: 6.2
<keras.callbacks.History at 0x7fbcb87cab90>
```

```
1 evaluate_model(LSTM_Model, "LSTM")
2
```

```
R Squared :  0.78899
Mean Absolute Error:  0.02982
```



Unseen Data Test

```
1 btc_new = pd.read_csv('BTC-USD.csv')
```

```
1 btc_new['Timestamp'] = pd.to_datetime(btc_new.Date)
2 btc_new.set_index('Timestamp', inplace=True)
3 btc_new.drop(["Date", "Open", "Low", "Close", "Adj Close", "Volume"], axis='columns', i
4 btc_new.head()
```

Timestamp	High
2021-06-14	40978.363281
2021-06-15	41295.269531
2021-06-16	40516.777344
2021-06-17	39513.671875
2021-06-18	38187.261719

```
1 btc_new['High_log'] = np.log1p(btc_new.High)
2 btc_new.tail()
```

```
                        High   High_log
```

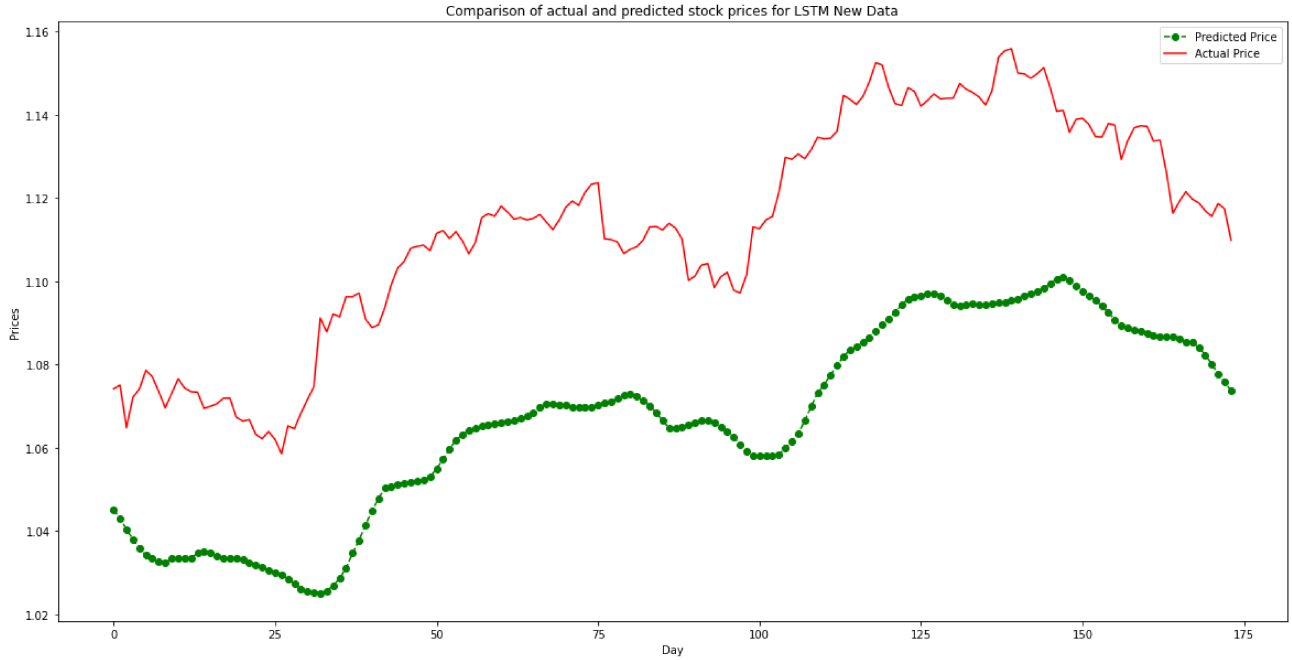
```
1 dataset_test_2 = scaler.transform(btc_new)
```

```
     2021-12-10   50015.253906   10.820103
```

```
1 X_test_2, y_test_2 = create_dataset(dataset_test_2, samples, feature)
```

```
1 evaluate_model(LSTM_Model, "LSTM New Data", X_test_2, y_test_2)
```

```
    R Squared :  -2.29029
    Mean Absolute Error:  0.04651
```




```
    (-2.290288716546341, 0.046513833091563546)
```

```
1
```


0s
completed at 10:09