```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
import yfinance as yf
```

```
org = yf.Ticker('ORA.PA')
with open('org.csv', "w") as f:
  org.history(period="max").to_csv(f)

df = pd.read_csv('org.csv')
df.head()
```

| | Date | Open | High | Low | Close | Volume | Dividends | Stock Splits |
|---|---|---|---|---|---|---|---|---|
| **0** | 2000-01-03 00:00:00+01:00 | 32.255717 | 32.982197 | 31.601885 | 31.722965 | 1551490 | 0.0 | 0.0 |
| **1** | 2000-01-04 00:00:00+01:00 | 31.577658 | 31.601876 | 29.301356 | 30.027836 | 1988788 | 0.0 | 0.0 |
| **2** | 2000-01-05 00:00:00+01:00 | 28.429598 | 29.495103 | 28.308518 | 28.453814 | 2593952 | 0.0 | 0.0 |
| **3** | 2000-01-06 00:00:00+01:00 | 28.574891 | 29.204506 | 26.903986 | 28.090570 | 2847297 | 0.0 | 0.0 |
| **4** | 2000-01-07 00:00:00+01:00 | 27.388306 | 29.204507 | 27.243011 | 28.938131 | 1593258 | 0.0 | 0.0 |

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6590 entries, 0 to 6589
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Date          6590 non-null   object 
 1   Open          6590 non-null   float64
 2   High          6590 non-null   float64
 3   Low           6590 non-null   float64
 4   Close         6590 non-null   float64
 5   Volume        6590 non-null   int64  
 6   Dividends     6590 non-null   float64
 7   Stock Splits  6590 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 412.0+ KB
```

```
close_price = df["Close"]
scaler = MinMaxScaler()
scaled_price_reshaped = scaler.fit_transform(close_price.values.reshape(-1, 1))
```

```
def create_window(df, lookback_window):
  X=[]
  y=[]
  for i in range(len(df)-lookback_window):
    X.append(df[i: i+lookback_window])
    y.append(df[i+lookback_window])
  return np.array(X), np.array(y)

X,y = create_window(scaled_price_reshaped, 60)

X.shape, y.shape
```

```
((6530, 60, 1), (6530, 1))
```

```
X_train, y_train = X[:6001], y[:6001]
X_test, y_test = X[6001:6401], y[6001:6401]
X_val, y_val = X[6401:], y[6401:]
X_train.shape, y_train.shape, X_test.shape, y_test.shape, X_val.shape, y_val.shape
```

```
((6001, 60, 1), (6001, 1), (400, 60, 1), (400, 1), (129, 60, 1), (129, 1))
```

```
model = Sequential()
model.add(LSTM(units=128, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.1))
model.add(LSTM(units=64, return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(units=32))
model.add(Dropout(0.1))
model.add(Dense(units=1))
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
Model: "sequential_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ lstm_7 (LSTM)                   │ (None, 60, 128)        │        66,560 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_5 (Dropout)             │ (None, 60, 128)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_8 (LSTM)                   │ (None, 60, 64)         │        49,408 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_6 (Dropout)             │ (None, 60, 64)         │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_9 (LSTM)                   │ (None, 32)             │        12,416 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_7 (Dropout)             │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            33 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 128,417 (501.63 KB)
 Trainable params: 128,417 (501.63 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss=MeanSquaredError(), metrics=[RootMeanSquaredError()])
early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
```

```
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])
```

```
188/188 ━━━━━━━━━━━━━━━━━━━━ 40s 176ms/step - loss: 1.4814e-04 - root_mean_squared_error: 0.0121 - val_loss: 1.7491e-05 - val_root
Epoch 23/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 41s 175ms/step - loss: 1.1214e-04 - root_mean_squared_error: 0.0106 - val_loss: 1.3547e-05 - val_root
Epoch 24/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 41s 176ms/step - loss: 1.2297e-04 - root_mean_squared_error: 0.0111 - val_loss: 1.4836e-05 - val_root
Epoch 25/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 35s 184ms/step - loss: 1.1353e-04 - root_mean_squared_error: 0.0106 - val_loss: 5.8852e-06 - val_root
```

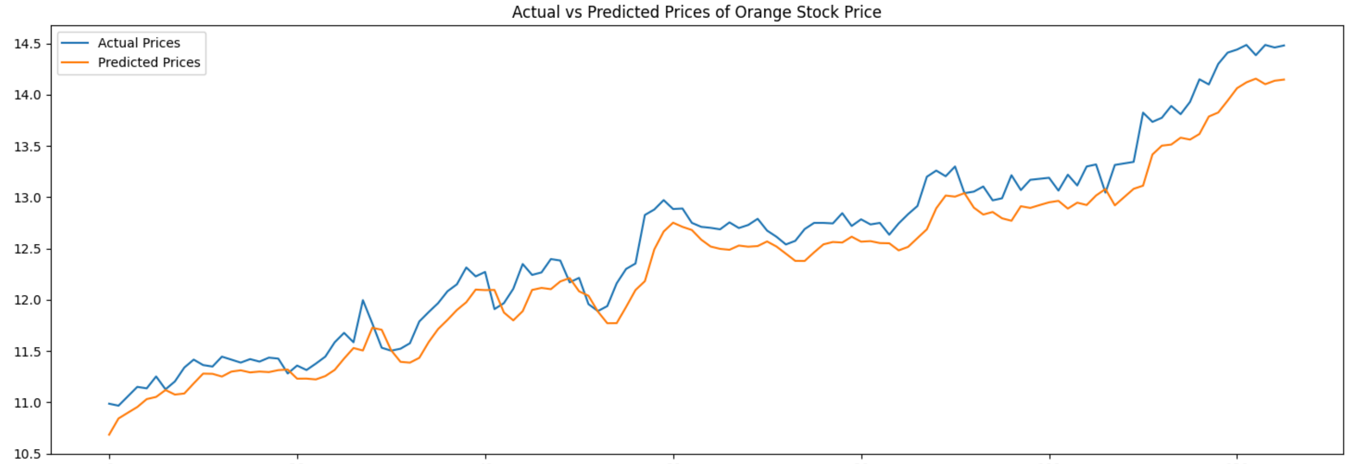
```
188/188                      35s 186ms/step - loss: 8.5360e-05 - root_mean_squared_error: 0.0093 - val_loss: 4.4304e-06 - val_root
```

```
188/188 ━━━━━━━━━━━━━━━━━━━━ 35s 186ms/step - loss: 8.5360e-05 - root_mean_squared_error: 0.0092 - val_loss: 4.4204e-06 - val_root
Epoch 48/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 41s 186ms/step - loss: 1.0033e-04 - root_mean_squared_error: 0.0100 - val_loss: 4.7235e-05 - val_root
Epoch 49/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 41s 186ms/step - loss: 1.2981e-04 - root_mean_squared_error: 0.0113 - val_loss: 1.1717e-05 - val_root
Epoch 50/50
188/188 ━━━━━━━━━━━━━━━━━━━━ 41s 185ms/step - loss: 1.6454e-04 - root_mean_squared_error: 0.0127 - val_loss: 2.9013e-05 - val_root
<keras.src.callbacks.history.History at 0x79668c16e8a0>
```

```python
predicted = model.predict(X_val)
predicted = scaler.inverse_transform(predicted)
actual_price = scaler.inverse_transform(y_val.reshape(-1,1))

mse_score = MeanSquaredError()(actual_price, predicted).numpy()
rmse_score = RootMeanSquaredError()(actual_price, predicted).numpy()

print(f"MSE: {mse_score}")
print(f"RMSE: {rmse_score}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 282ms/step
MSE: 0.06792736798524857
RMSE: 0.2606287896633148
```

```python
num_of_days_in_6_months = 6*21
if num_of_days_in_6_months > len(y_val):
  num_of_days_in_6_months = len(y_val)

plt.figure(figsize=(18,6))
plt.plot(actual_price[-num_of_days_in_6_months:], label='Actual Prices')
plt.plot(predicted[-num_of_days_in_6_months:], label='Predicted Prices')
plt.legend()
plt.title('Actual vs Predicted Prices of Orange Stock Price')
plt.show()
```



Start coding or generate with AI.