Australian Stock Prices Prediction

Author: Phuc An Nguyen

August 2023

1. Introduction

Stock market analysis and prediction is an important part of the stock trading and investing process. The benefits from performing this analysis include: Making smart investment decisions, Investment orientation, Understand the business, Risk forecast, Investment strategy orientation and Monitor investment performance.

This project will use Descriptive analysis to identify patterns in time series data, like trends, cycles, or seasonal variation as well as highlights the main characteristics of the time series data. Futhermore, I will use the Forecasting model for predicting future data based on the historical data with different types. Then I will evaluate and compare the performance of these models against each other and choose the best model. Finally, I will answer the inspiration questions raised at the beginning of the project.

1.1. Data description

This dataset contains historical share price data from the top 100 companies (with 100 different CSV files) listed on the Australian Securities Exchange, each data file contains:

- **Date**: date
- **Open**: opening price
- **High**: high price
- **Low**: low price
- **Close**: closing price
- **Adj Close**: adjusted closing price (including dividends)
- **Volume**: trading volume

1.2. Inspiration questions

1. Which stocks have high value, high trading volume and low volatility?
2. What periods of the year are stocks most volatile?
3. Investors can profit by making smart decisions about buying, selling or holding stocks by predicting stock trends.

1.3. Import libraries

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import seaborn as sns
import math
import os

# ARIMA
# from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm

# LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from keras.models import Sequential
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error, r2_score
from keras.layers import LSTM
from keras.layers import Dense
from keras.utils.vis_utils import plot_model

# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score

import warnings
warnings.filterwarnings('ignore')
```

2. Read multiple CSV file

In [2]:
```
# Set the path to the folder containing the CSV files
folder_path = 'dataset'

# Create an empty list to store the data from all the CSV files
data_frames = []

# Create an empty list to store the names of the files
file_names = []

# Loop through the files in the folder, check if they are CSV files
for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):
        file_path = os.path.join(folder_path, file_name)
        df = pd.read_csv(file_path)
        # Add the name of the file as a new column to the DataFrame
        df['Company'] = file_name
        data_frames.append(df)
        # Store the name of the file in the file_names list
        file_names.append(file_name)

# After the loop, data_frames will contain all the DataFrames from the CSV files in the folder
# file_names will contain the names of all the CSV files in the folder
# To combine all the DataFrames into a single DataFrame
combined_df = pd.concat(data_frames, ignore_index=True)
```

3. Exploratory Data Analysis (EDA)

3.1. Data Exploration

In [3]:
```
combined_df.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume	Company
0	2015-03-31	0.555	0.595	0.530	0.565	0.565	4816294.0	A2M.csv
1	2015-04-01	0.575	0.580	0.555	0.565	0.565	4376660.0	A2M.csv
2	2015-04-02	0.560	0.565	0.535	0.555	0.555	2779640.0	A2M.csv
3	2015-04-07	0.545	0.550	0.540	0.545	0.545	392179.0	A2M.csv
4	2015-04-08	0.545	0.545	0.530	0.540	0.540	668446.0	A2M.csv

In [4]:
```
# Check dataframe shape
combined_df.shape
```

Out[4]:
```
(432888, 8)
```

In [5]:
```
# Check null
combined_df.isnull().sum()
```

Out[5]:
```
Date            0
Open         1527
High         1527
Low          1527
Close        1527
Adj Close    1527
Volume       1527
Company         0
dtype: int64
```

In [6]:
```
# Check information dataframe
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 432888 entries, 0 to 432887
Data columns (total 8 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Date       432888 non-null  object
 1   Open       431361 non-null  float64
 2   High       431361 non-null  float64
 3   Low        431361 non-null  float64
 4   Close      431361 non-null  float64
 5   Adj Close  431361 non-null  float64
 6   Volume     431361 non-null  float64
 7   Company    432888 non-null  object
dtypes: float64(6), object(2)
memory usage: 26.4+ MB
```

In [7]:
```
# summarizes dataframe
combined_df.describe()
```

Out[7]:

	Open	High	Low	Close	Adj Close	Volume
count	431361.000000	431361.000000	431361.000000	431361.000000	431361.000000	4.313610e+05
mean	13.628159	13.765587	13.486251	13.626580	10.388578	3.646548e+06
std	19.325604	19.506007	19.140934	19.325726	17.343674	7.011357e+06
min	0.004000	0.004000	0.004000	0.004000	0.002449	0.000000e+00
25%	3.490000	3.530730	3.450000	3.490000	2.229591	5.879090e+05
50%	7.200000	7.275480	7.110000	7.196750	5.001390	1.799148e+06
75%	15.888700	16.040001	15.698000	15.866700	11.451574	4.207143e+06
max	339.420013	342.750000	337.029999	341.000000	339.376190	9.930183e+08

In [8]:
```
# Check negative or zero value
(combined_df[['Open','High','Low','Close','Adj Close']].values <= 0).any()
```

Out[8]:
```
False
```

3.2. Data Wrangling

Removing Rows with Null Values

In [9]:
```
# Removing all Null rows in dataframe
df = combined_df.dropna()
```

Change type Date column

In [10]:
```
# Change the type of Date column from string to datetime
df['Date'] = pd.to_datetime(df['Date'])
```

Change Company value

In [11]:
```
# Change the Company name without .csv
df['Company'] = df['Company'].str.replace('.csv', '')
```

Change Date into Weekday and Quarter

In [12]:
```
df['Day_of_week'] = df['Date'].dt.day_name()
df['Quarter'] = df['Date'].dt.to_period('Q')
```

In [13]:
```
df.head()
```

Out[13]:

	Date	Open	High	Low	Close	Adj Close	Volume	Company	Day_of_week	Quarter
0	2015-03-31	0.555	0.595	0.530	0.565	0.565	4816294.0	A2M	Tuesday	2015Q1
1	2015-04-01	0.575	0.580	0.555	0.565	0.565	4376660.0	A2M	Wednesday	2015Q2
2	2015-04-02	0.560	0.565	0.535	0.555	0.555	2779640.0	A2M	Thursday	2015Q2
3	2015-04-07	0.545	0.550	0.540	0.545	0.545	392179.0	A2M	Tuesday	2015Q2
4	2015-04-08	0.545	0.545	0.530	0.540	0.540	668446.0	A2M	Wednesday	2015Q2

Get top 5 companies with the highest volume

In [14]:
```
# Calculate sum of volume and get top 5 companies
top_volume_df = df.groupby(['Company'])['Volume'].agg('sum').reset_index()
top_volume_df = top_volume_df.sort_values(by="Volume", ascending=False).head(5)
```

In [15]:
```
# Filter the dataframe keep top 5 companies
company_list = top_volume_df['Company'].tolist()
df = df[df['Company'].isin(company_list)]
```

In [16]:
```
df.shape
```

Out[16]:
```
(25716, 10)
```

3.3. Data Visualization

In [17]:
```
company_list
```

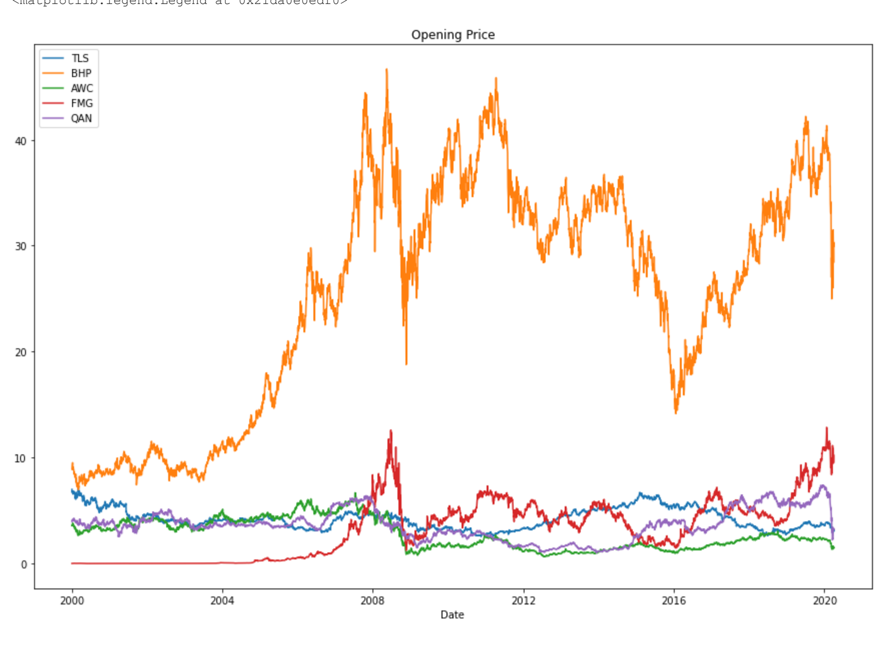
Out[17]:
```
['TLS', 'BHP', 'AWC', 'FMG', 'QAN']
```

In [18]:
```
# Split big dataframe into different company dataframes
TLS_df = df.loc[df['Company'] == 'TLS']
BHP_df = df.loc[df['Company'] == 'BHP']
AWC_df = df.loc[df['Company'] == 'AWC']
FMG_df = df.loc[df['Company'] == 'FMG']
QAN_df = df.loc[df['Company'] == 'QAN']
```

Opening price

In [19]:
```
fig,ax = plt.subplots(figsize=(15,10))
ax.plot(TLS_df['Date'], TLS_df['Open'], label="TLS")
ax.plot(BHP_df['Date'], BHP_df['Open'], label="BHP")
ax.plot(AWC_df['Date'], AWC_df['Open'], label="AWC")
ax.plot(FMG_df['Date'], FMG_df['Open'], label="FMG")
ax.plot(QAN_df['Date'], QAN_df['Open'], label="QAN")
ax.set_xlabel('Date')
ax.set_title('Opening Price')
plt.legend()
```

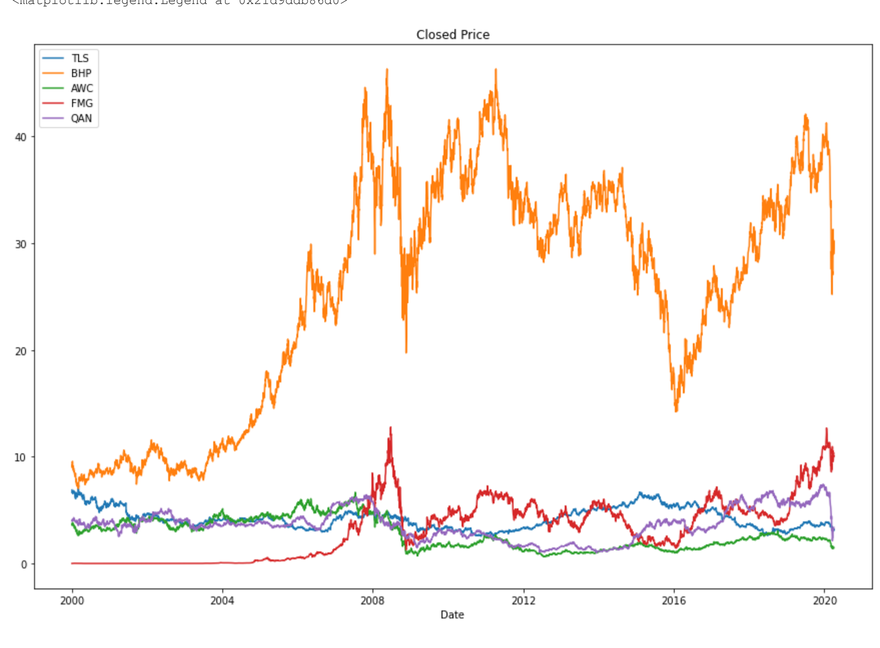
Out[19]:
```
<matplotlib.legend.Legend at 0x21da0e0edf0>
```



Closed Price

In [20]:
```
fig,ax = plt.subplots(figsize=(15,10))
ax.plot(TLS_df['Date'], TLS_df['Close'], label="TLS")
ax.plot(BHP_df['Date'], BHP_df['Close'], label="BHP")
ax.plot(AWC_df['Date'], AWC_df['Close'], label="AWC")
ax.plot(FMG_df['Date'], FMG_df['Close'], label="FMG")
ax.plot(QAN_df['Date'], QAN_df['Close'], label="QAN")
ax.set_xlabel('Date')
ax.set_title('Closed Price')
plt.legend()
```

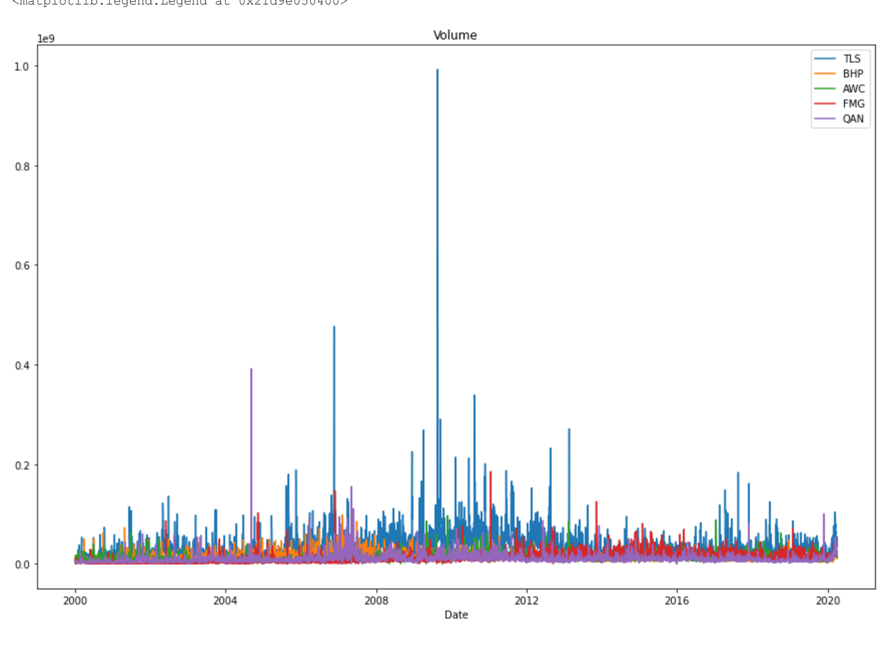
Out[20]:
```
<matplotlib.legend.Legend at 0x21d9ddb86d0>
```



Volume

In [21]:
```
fig,ax = plt.subplots(figsize=(15,10))
ax.plot(TLS_df['Date'], TLS_df['Volume'], label="TLS")
ax.plot(BHP_df['Date'], BHP_df['Volume'], label="BHP")
ax.plot(AWC_df['Date'], AWC_df['Volume'], label="AWC")
ax.plot(FMG_df['Date'], FMG_df['Volume'], label="FMG")
ax.plot(QAN_df['Date'], QAN_df['Volume'], label="QAN")
ax.set_xlabel('Date')
ax.set_title('Volume')
plt.legend()
```

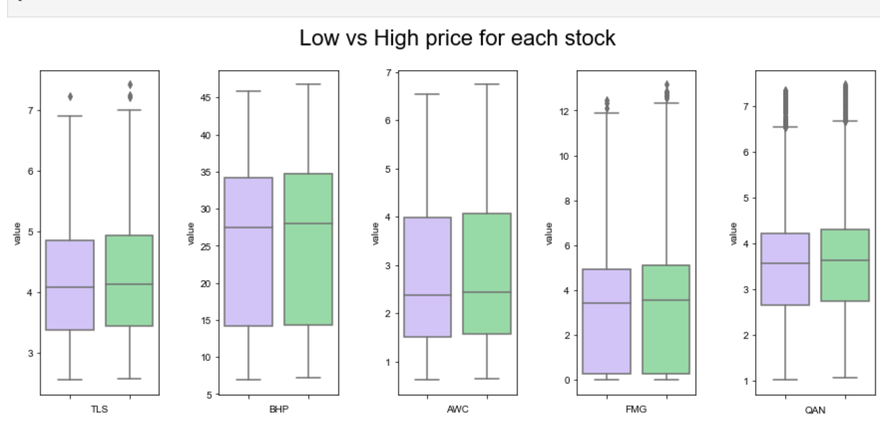
Out[21]:
```
<matplotlib.legend.Legend at 0x21d9e050400>
```



Distribution of Low vs High price

In [22]:
```
TLS_df_melted = pd.melt(TLS_df[['Low', 'High']])
BHP_df_melted = pd.melt(BHP_df[['Low', 'High']])
AWC_df_melted = pd.melt(AWC_df[['Low', 'High']])
FMG_df_melted = pd.melt(FMG_df[['Low', 'High']])
QAN_df_melted = pd.melt(QAN_df[['Low', 'High']])
```

In [23]:
```
fig, ax = plt.subplots(1, 5, figsize=(15, 6))
sns.set_theme(style="ticks", palette="pastel")

# add padding between the subplots
plt.subplots_adjust(wspace=0.5)

# draw boxplot for age in the 1st subplot
sns.boxplot(x='variable', y='value', data=TLS_df_melted,
            palette=["m", "g"], ax=ax[0])
ax[0].set_xlabel('TLS')

# draw boxplot for station_distance in the 2nd subplot
sns.boxplot(x='variable', y='value', data=BHP_df_melted,
            palette=["m", "g"], ax=ax[1])
ax[1].set_xlabel('BHP')

# # draw boxplot for stores_count in the 3rd subplot
sns.boxplot(x='variable', y='value', data=AWC_df_melted,
            palette=["m", "g"], ax=ax[2])
ax[2].set_title('Low vs High price for each stock', fontdict={'size': 22}, pad=25)
ax[2].set_xlabel('AWC')

# # finally draw boxplot for unit_price in the 4th subplot
sns.boxplot(x='variable', y='value', data=FMG_df_melted,
            palette=["m", "g"], ax=ax[3])
ax[3].set_xlabel('FMG')

# # finally draw boxplot for unit_price in the 4th subplot
sns.boxplot(x='variable', y='value', data=QAN_df_melted,
            palette=["m", "g"], ax=ax[4])
ax[4].set_xlabel('QAN')

# by default, you'll see x-tick label set to 0 in each subplot
# remove it by setting it to empty list
for subplot in ax:
    subplot.set_xticklabels([])

plt.show()
```

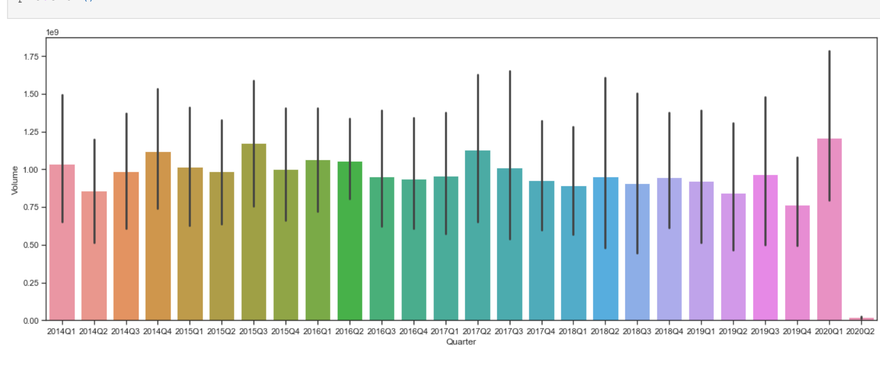


Total trading volume by quarter

In [24]:
```
# Group by dataframe follow the company and quarter and get sum of volume
quarter_df = df.groupby(['Company','Quarter'])['Volume'].sum().reset_index()
```

In [25]:
```
# Get the quarter from 2014 t0 2020
quarter_df = quarter_df[~quarter_df['Quarter'].astype(str).str.startswith('200')]
quarter_df = quarter_df[~quarter_df['Quarter'].astype(str).str.startswith('2010')]
quarter_df = quarter_df[~quarter_df['Quarter'].astype(str).str.startswith('2011')]
quarter_df = quarter_df[~quarter_df['Quarter'].astype(str).str.startswith('2012')]
quarter_df = quarter_df[~quarter_df['Quarter'].astype(str).str.startswith('2013')]
```

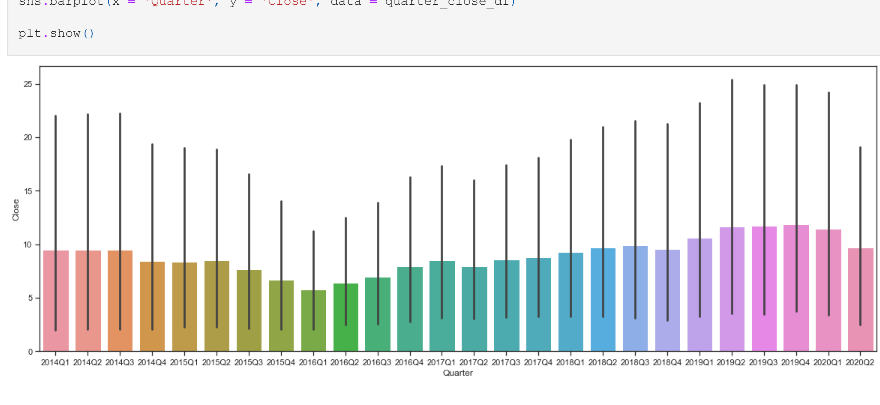
In [26]:
```
fig = plt.figure(figsize =(20, 7))
sns.barplot(x = 'Quarter', y = 'Volume', data = quarter_df)

plt.show()
```



Average Closed price by quarter

In [27]:
```
# Group by dataframe follow the company and quarter and get average of close price
quarter_close_df = df.groupby(['Company','Quarter'])['Close'].mean().reset_index()

# Get the quarter from 2014 t0 2020
quarter_close_df = quarter_close_df[~quarter_close_df['Quarter'].astype(str).str.startswith('200')]
quarter_close_df = quarter_close_df[~quarter_close_df['Quarter'].astype(str).str.startswith('2010')]
quarter_close_df = quarter_close_df[~quarter_close_df['Quarter'].astype(str).str.startswith('2011')]
quarter_close_df = quarter_close_df[~quarter_close_df['Quarter'].astype(str).str.startswith('2012')]
quarter_close_df = quarter_close_df[~quarter_close_df['Quarter'].astype(str).str.startswith('2013')]
```

In [28]:
```
fig = plt.figure(figsize =(20, 7))
sns.barplot(x = 'Quarter', y = 'Close', data = quarter_close_df)

plt.show()
```



EDA Summary

In this section, I had a better insight into the data set and at the same time provided specific results for the selection and creation of suitable models for this data. Important discoveries found in this section include:

- Data processing steps include removing rows with null values, change type Date column, change company value, change Date into Weekday and Quarter.
- After performing data wrangling, we shorten the data (from 432888 to 25716 rows and 10 factors with no null value) to further analyze the top 5 stocks with the highest total trading volume.
- Data types in the dataset include datetime, float, and object.
- Use multiple line charts to show the opening price, closing price, and volume of the 5 analyzed stocks from 2000 to 2020.
- Use box plot to see the division of Low versus High price for each stock.
- Use bar chart to show the Total trading and Average Closed price by quarter for all stocks from 2014 to 2020.

4. Methodology

4.1. Modelling methods

The time series analysis in this project uses statistical methods and machine learning to analyze time data and make predictions or inferences about the future. There will be developed with 3 models builts, including:

- **Time Series Models**: ARIMA (AutoRegressive Integrated Moving Average)
- **Deep Learning Models**: LSTM (Long Short-Term Memory)
- **Ensemble Models**: Random Forest Classifier

4.2. Model evaluation methods

Data science places a lot of importance on model evaluation. It makes it simple for you to communicate your model to others and lets you see how well your model is performing and how closely your forecast matches the actual value. \ The assessment measures that will ultimately be used to assess and choose the model with the greatest performance are described below. Each of them has unique strengths and limitations, and they were chosen to serve as the foundation for an assessment that is as objective as possible.

- **Mean Absolute Percentage Error (MAPE)**: $$ MAPE = \frac{1}{N} \sum^{N}_{i=1}{|\frac{y_{i} - \hat{y_{i}}}{y_{i}}|} $$
- **Root Mean Squared Error (RMSE)**: $$ RMSE = \sqrt{\frac{1}{N} \sum^{N}_{i=1}{(y_{i} - \hat{y_{i}})^2}} $$
- **Mean Absolute Error (MAE)**: $$ MAE = \frac{1}{N} \sum^{N}_{i=1}{|y_{i} - \hat{y_{i}}|} $$

4.3. Data preparation

Setting up the target

In [29]:
```
# Remove the extra columns that we don't need
TLS_df = TLS_df.drop(columns=["Adj Close", "Company", "Day_of_week", "Quarter"])

# Make Date column to index
TLS_df.set_index('Date', inplace=True)
TLS_df.head()
```

Out[29]:

Date	Open	High	Low	Close	Volume
2000-01-04	7.00044	7.00044	6.88222	6.89066	8203436.0
2000-01-05	6.75555	6.81466	6.69644	6.75555	9600494.0
2000-01-06	6.79778	6.84844	6.71333	6.71333	9683533.0
2000-01-07	6.70066	6.73866	6.58666	6.62044	13613336.0
2000-01-10	6.75555	6.75555	6.69644	6.71333	6067734.0

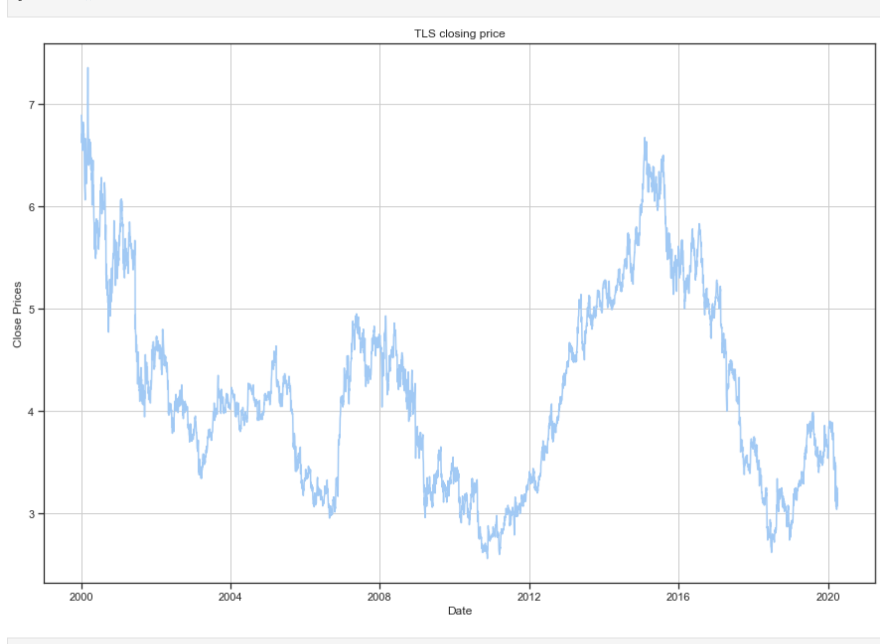
5. Model Development

5.1. SARIMA (Seasonal AutoRegressive Integrated Moving Average)

ARIMA Models are specified by three order parameters: (p, d, q), where,

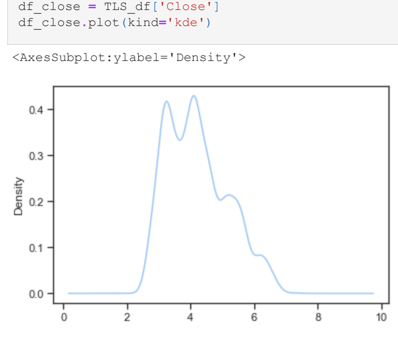
- p is the order of the AR (Auto Regressive) term
- q is the order of the MA (Moving Average) term
- d is the number of differencing required to make the time series stationary

In [30]:
```
# Plot the data want to predict
plt.figure(figsize=(15,10))
plt.grid(True)
plt.xlabel('Date')
plt.ylabel('Close Prices')
plt.plot(TLS_df['Close'])
plt.title('TLS closing price')
plt.show()
```



In [31]:
```
# Distribution of the dataset
df_close = TLS_df['Close']
df_close.plot(kind='kde')
```

Out[31]:
```
<AxesSubplot:ylabel='Density'>
```

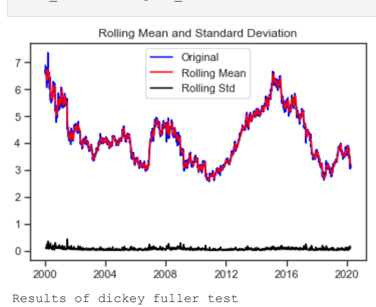


Choosing d value:

d can have a value of 0 or 1. For stationary data, we should use 0, while for seasonal data, we should use 1.

Now, let test the stationarity and seasonality

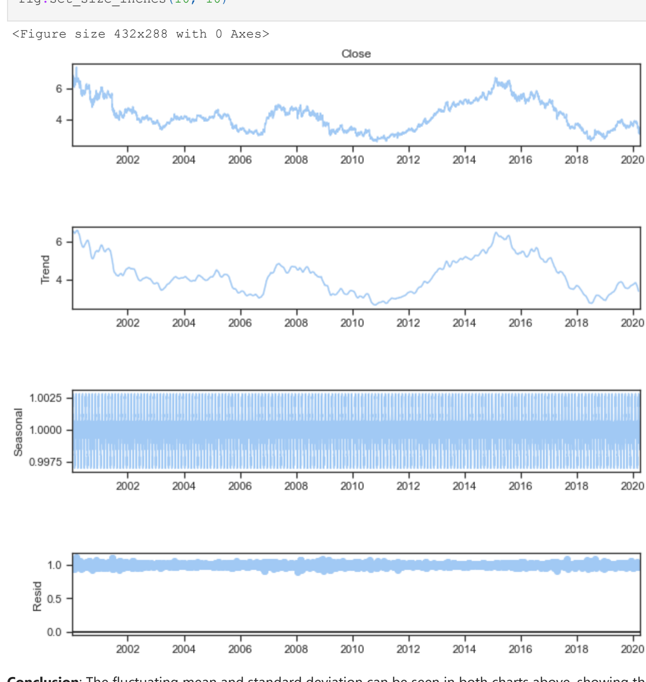
In [32]:
```
#Test for staionarity
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)
    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')

    # output for dft will give us without defining what the values are.
    # hence we manually write what values does it explains using a for loop
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)

test_stationarity(df_close)
```




```
Results of dickey fuller test
Test Statistics                  -2.439774
p-value                           0.130824
No. of lags used                 33.000000
Number of observations used    5122.000000
critical value (1%)              -3.431627
critical value (5%)              -2.862104
critical value (10%)             -2.567070
dtype: float64
```

In [33]:
```
result = seasonal_decompose(df_close, model='multiplicative', period = 30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(10, 10)
```

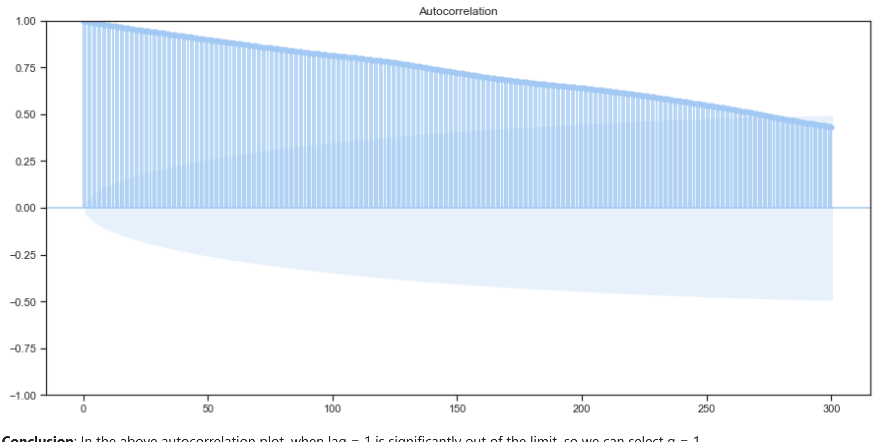
```
<Figure size 432x288 with 0 Axes>
```



Conclusion: The fluctuating mean and standard deviation can be seen in both charts above, showing that our series is not stationary. We should use 1 as the d value because our data is seasonal.

Choosing q value: using ACF

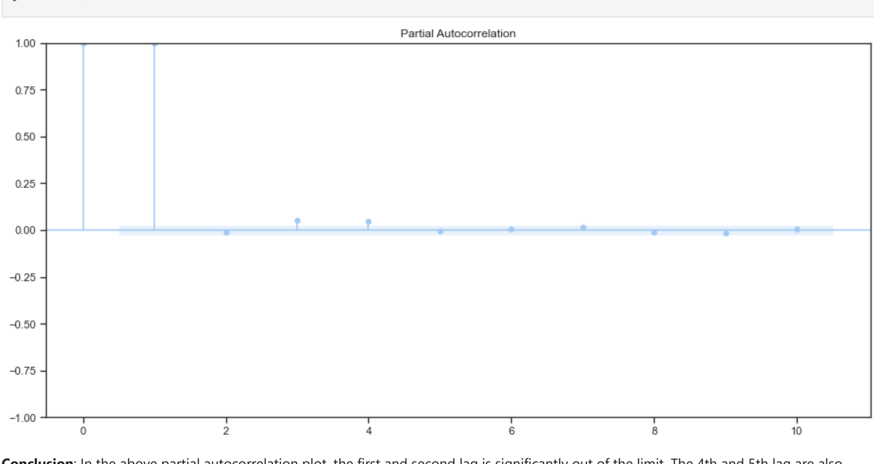
In [34]:
```
fig, ax = plt.subplots(figsize=(15, 7))
plot_acf(df_close, lags = 300, ax=ax)
plt.show()
```



Conclusion: In the above autocorrelation plot, when lag = 1 is significantly out of the limit, so we can select q = 1

Choosing p value: using PACF

In [35]:
```python
fig, ax = plt.subplots(figsize=(15, 7))
plot_pacf(df_close, lags = 10, ax=ax)
plt.show()
```



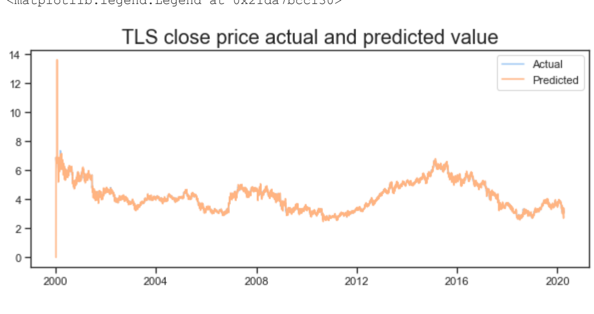
Conclusion: In the above partial autocorrelation plot, the first and second lag is significantly out of the limit. The 4th and 5th lag are also out of the significant limit but it is not that far, so we can select the order of the p as 3 or 4.

SARIMA

In [36]:
```python
SARIMA_model = SARIMAX(df_close, order=(3, 1, 1), seasonal_order=(2, 2, 0, 12))
predictions = SARIMA_model.fit().predict()
```

In [37]:
```python
plt.figure(figsize=(10,4))
plt.plot(df_close, label="Actual")
plt.plot(predictions, label="Predicted")
plt.title('TLS close price actual and predicted value', fontsize=20)
plt.legend()
```

Out[37]:
```
<matplotlib.legend.Legend at 0x21da7bcc130>
```



Evaluating model

In [38]:
```python
# report performance
mse = mean_squared_error(df_close, predictions)
print('MSE: '+str(mse))
mae = mean_absolute_error(df_close, predictions)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(df_close, predictions))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(predictions - df_close)/np.abs(df_close))
print('MAPE: '+str(mape))
```

```
MSE: 0.02652820283487824
MAE: 0.06761036500427316
RMSE: 0.16287480724433182
MAPE: 0.015919325385689756
```

5.2. Long Short-Term Memory (LSTM)

Setting the Target Variable and Selecting the Features

In [39]:
```python
#Set Target Variable
output_var = pd.DataFrame(TLS_df['Close'])

#Selecting the Features
features = ['Open', 'High', 'Low', 'Volume']
```

Scale

We will scale the stock values to values between 0 and 1 in order to reduce the processing cost of the data in the table. As a consequence, the total amount of data in huge numbers is minimised, which lowers memory use. Additionally, because the data is not dispersed across large numbers, scaling down allows for higher precision.

In [40]:
```python
# Scaling
scaler = MinMaxScaler()
feature_transform = scaler.fit_transform(TLS_df[features])
feature_transform = pd.DataFrame(columns=features, data=feature_transform, index=TLS_df.index)
feature_transform.head()
```

Out[40]:

Date	Open	High	Low	Volume
2000-01-04	0.939397	0.914583	0.925997	0.008261
2000-01-05	0.887706	0.876232	0.886287	0.009668
2000-01-06	0.896620	0.883205	0.889897	0.009752
2000-01-07	0.876120	0.860543	0.862822	0.013709
2000-01-10	0.887706	0.864030	0.886287	0.006110

In [41]:
```python
# Splitting to Training set and Test set
timesplit = TimeSeriesSplit(n_splits=10)
for train_index, test_index in timesplit.split(feature_transform):
    X_train, X_test = feature_transform[:len(train_index)], feature_transform[len(train_index): (len(train_inde
    y_train, y_test = output_var[:len(train_index)].values.ravel(), output_var[len(train_index): (len(train_inc
```

Data Processing For LSTM

The data from the training and test sets must be transformed into a form that the LSTM model can understand. We first convert the training and test data to NumPy arrays and then reorganise them to comply with the format since the LSTM requires that the data be supplied in the 3D form (Number of Samples, 1, Number of Features). Likewise, the test set is reshaped.

In [42]:
```python
# Process the data for LSTM
trainX = np.array(X_train)
testX = np.array(X_test)
X_train = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = testX.reshape(X_test.shape[0], 1, X_test.shape[1])
```

Building the LSTM Model

In [43]:
```python
#Building the LSTM Model
lstm = Sequential()
lstm.add(LSTM(32, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=False))
lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error', optimizer='adam')
```

In [44]:
```python
history = lstm.fit(X_train, y_train, epochs=100, batch_size=8, verbose=1, shuffle=False)
```

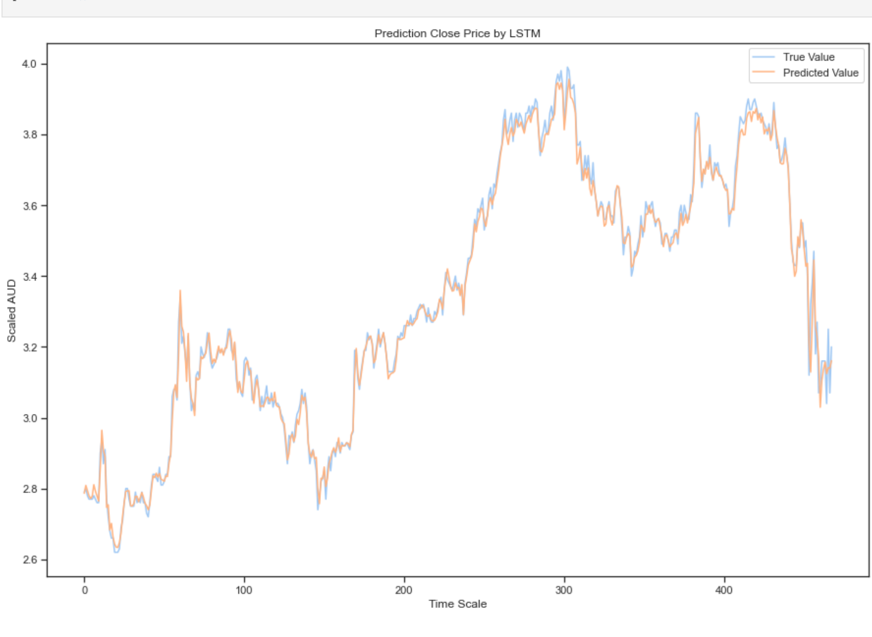
```
Epoch 1/100
586/586 [==============================] - 1s 793us/step - loss: 8.4755
Epoch 2/100
586/586 [==============================] - 0s 779us/step - loss: 0.1253
Epoch 3/100
586/586 [==============================] - 0s 769us/step - loss: 0.0599
Epoch 4/100
586/586 [==============================] - 0s 769us/step - loss: 0.0308
Epoch 5/100
586/586 [==============================] - 0s 744us/step - loss: 0.0193
Epoch 6/100
586/586 [==============================] - 0s 760us/step - loss: 0.0127
Epoch 7/100
586/586 [==============================] - 0s 759us/step - loss: 0.0089
Epoch 8/100
586/586 [==============================] - 0s 760us/step - loss: 0.0066
Epoch 9/100
586/586 [==============================] - 0s 786us/step - loss: 0.0049
Epoch 10/100
586/586 [==============================] - 0s 800us/step - loss: 0.0036
Epoch 11/100
586/586 [==============================] - 0s 763us/step - loss: 0.0029
Epoch 12/100
586/586 [==============================] - 0s 752us/step - loss: 0.0024
Epoch 13/100
586/586 [==============================] - 0s 764us/step - loss: 0.0021
Epoch 14/100
586/586 [==============================] - 0s 788us/step - loss: 0.0019
Epoch 15/100
586/586 [==============================] - 0s 780us/step - loss: 0.0018
Epoch 16/100
586/586 [==============================] - 0s 761us/step - loss: 0.0017
Epoch 17/100
586/586 [==============================] - 0s 741us/step - loss: 0.0016
Epoch 18/100
586/586 [==============================] - 0s 770us/step - loss: 0.0016
Epoch 19/100
586/586 [==============================] - 0s 741us/step - loss: 0.0015
Epoch 20/100
586/586 [==============================] - 0s 779us/step - loss: 0.0015
Epoch 21/100
586/586 [==============================] - 0s 740us/step - loss: 0.0014
Epoch 22/100
586/586 [==============================] - 0s 775us/step - loss: 0.0014
Epoch 23/100
586/586 [==============================] - 0s 744us/step - loss: 0.0014
Epoch 24/100
586/586 [==============================] - 0s 757us/step - loss: 0.0013
Epoch 25/100
586/586 [==============================] - 0s 742us/step - loss: 0.0013
Epoch 26/100
586/586 [==============================] - 0s 737us/step - loss: 0.0012
Epoch 27/100
586/586 [==============================] - 0s 751us/step - loss: 0.0012
Epoch 28/100
586/586 [==============================] - 0s 752us/step - loss: 0.0012
Epoch 29/100
586/586 [==============================] - 0s 774us/step - loss: 0.0011
Epoch 30/100
586/586 [==============================] - 0s 774us/step - loss: 0.0011
Epoch 31/100
586/586 [==============================] - 0s 767us/step - loss: 0.0011
Epoch 32/100
586/586 [==============================] - 0s 769us/step - loss: 0.0011
Epoch 33/100
586/586 [==============================] - 0s 758us/step - loss: 0.0010
Epoch 34/100
586/586 [==============================] - 0s 787us/step - loss: 0.0010
Epoch 35/100
586/586 [==============================] - 0s 795us/step - loss: 0.0010
Epoch 36/100
586/586 [==============================] - 0s 774us/step - loss: 9.8884e-04
Epoch 37/100
586/586 [==============================] - 0s 798us/step - loss: 9.7581e-04
Epoch 38/100
586/586 [==============================] - 0s 766us/step - loss: 9.6679e-04
Epoch 39/100
586/586 [==============================] - 0s 755us/step - loss: 9.5686e-04
Epoch 40/100
586/586 [==============================] - 0s 769us/step - loss: 9.5294e-04
Epoch 41/100
586/586 [==============================] - 0s 768us/step - loss: 9.5437e-04
Epoch 42/100
586/586 [==============================] - 0s 777us/step - loss: 9.5952e-04
Epoch 43/100
586/586 [==============================] - 0s 772us/step - loss: 9.6961e-04
Epoch 44/100
586/586 [==============================] - 0s 764us/step - loss: 9.7899e-04
Epoch 45/100
586/586 [==============================] - 0s 767us/step - loss: 9.9593e-04
Epoch 46/100
586/586 [==============================] - 0s 757us/step - loss: 0.0010
Epoch 47/100
586/586 [==============================] - 0s 785us/step - loss: 0.0010
Epoch 48/100
586/586 [==============================] - 0s 783us/step - loss: 0.0010
Epoch 49/100
586/586 [==============================] - 0s 773us/step - loss: 0.0010
Epoch 50/100
586/586 [==============================] - 0s 753us/step - loss: 0.0010
Epoch 51/100
586/586 [==============================] - 0s 765us/step - loss: 0.0010
Epoch 52/100
586/586 [==============================] - 0s 743us/step - loss: 0.0010
Epoch 53/100
586/586 [==============================] - 0s 769us/step - loss: 0.0010
Epoch 54/100
586/586 [==============================] - 0s 761us/step - loss: 0.0010
Epoch 55/100
586/586 [==============================] - 0s 752us/step - loss: 0.0010
Epoch 56/100
586/586 [==============================] - 0s 752us/step - loss: 0.0010
Epoch 57/100
586/586 [==============================] - 0s 775us/step - loss: 0.0010
Epoch 58/100
586/586 [==============================] - 0s 766us/step - loss: 9.9904e-04
Epoch 59/100
586/586 [==============================] - 0s 760us/step - loss: 9.8863e-04
Epoch 60/100
586/586 [==============================] - 0s 777us/step - loss: 9.8498e-04
Epoch 61/100
586/586 [==============================] - 0s 798us/step - loss: 9.8394e-04
Epoch 62/100
586/586 [==============================] - 0s 807us/step - loss: 9.8189e-04
Epoch 63/100
586/586 [==============================] - 0s 753us/step - loss: 9.7629e-04
Epoch 64/100
586/586 [==============================] - 0s 787us/step - loss: 9.7068e-04
Epoch 65/100
586/586 [==============================] - 0s 766us/step - loss: 9.6199e-04
Epoch 66/100
586/586 [==============================] - 0s 763us/step - loss: 9.5826e-04
Epoch 67/100
586/586 [==============================] - 0s 821us/step - loss: 9.5034e-04
Epoch 68/100
586/586 [==============================] - 0s 845us/step - loss: 9.4376e-04
Epoch 69/100
586/586 [==============================] - 0s 835us/step - loss: 9.3850e-04
Epoch 70/100
586/586 [==============================] - 0s 761us/step - loss: 9.3317e-04
Epoch 71/100
586/586 [==============================] - 0s 761us/step - loss: 9.2798e-04
Epoch 72/100
586/586 [==============================] - 1s 879us/step - loss: 9.2993e-04
Epoch 73/100
586/586 [==============================] - 0s 793us/step - loss: 9.2566e-04
Epoch 74/100
586/586 [==============================] - 0s 780us/step - loss: 9.1699e-04
Epoch 75/100
586/586 [==============================] - 0s 750us/step - loss: 9.1095e-04
Epoch 76/100
586/586 [==============================] - 0s 730us/step - loss: 8.9865e-04
Epoch 77/100
586/586 [==============================] - 1s 914us/step - loss: 8.9673e-04
Epoch 78/100
586/586 [==============================] - 1s 905us/step - loss: 8.9684e-04
Epoch 79/100
586/586 [==============================] - 0s 845us/step - loss: 8.9075e-04
Epoch 80/100
586/586 [==============================] - 0s 800us/step - loss: 8.8339e-04
Epoch 81/100
586/586 [==============================] - 0s 838us/step - loss: 8.6588e-04
Epoch 82/100
586/586 [==============================] - 0s 769us/step - loss: 8.5491e-04
Epoch 83/100
586/586 [==============================] - 0s 819us/step - loss: 8.5722e-04
Epoch 84/100
586/586 [==============================] - 0s 801us/step - loss: 8.6085e-04
Epoch 85/100
586/586 [==============================] - 0s 774us/step - loss: 8.4575e-04
Epoch 86/100
586/586 [==============================] - 0s 778us/step - loss: 8.5540e-04
Epoch 87/100
586/586 [==============================] - 0s 754us/step - loss: 8.6150e-04
Epoch 88/100
586/586 [==============================] - 0s 768us/step - loss: 8.5534e-04
Epoch 89/100
586/586 [==============================] - 0s 751us/step - loss: 8.5716e-04
Epoch 90/100
586/586 [==============================] - 0s 766us/step - loss: 8.5281e-04
Epoch 91/100
586/586 [==============================] - 0s 742us/step - loss: 8.4558e-04
Epoch 92/100
586/586 [==============================] - 0s 776us/step - loss: 8.4835e-04
Epoch 93/100
586/586 [==============================] - 0s 725us/step - loss: 8.4542e-04
Epoch 94/100
586/586 [==============================] - 0s 817us/step - loss: 8.4124e-04
Epoch 95/100
586/586 [==============================] - 0s 766us/step - loss: 8.3678e-04
Epoch 96/100
586/586 [==============================] - 0s 759us/step - loss: 8.3343e-04
Epoch 97/100
586/586 [==============================] - 0s 764us/step - loss: 8.3111e-04
Epoch 98/100
586/586 [==============================] - 0s 774us/step - loss: 8.2960e-04
Epoch 99/100
586/586 [==============================] - 0s 777us/step - loss: 8.1198e-04
Epoch 100/100
586/586 [==============================] - 0s 797us/step - loss: 8.2433e-04
```

Making the LSTM Prediction

In [45]:
```python
#LSTM Prediction
y_pred = lstm.predict(X_test)
```

```
15/15 [==============================] - 0s 968us/step
```

In [46]:
```python
# Predicted vs True Adj Close Value – LSTM
fig,ax = plt.subplots(figsize=(15,10))
plt.plot(y_test, label='True Value')
plt.plot(y_pred, label='Predicted Value')
plt.title("Prediction Close Price by LSTM")
plt.xlabel('Time Scale')
plt.ylabel('Scaled AUD')
plt.legend()
plt.show()
```



Evaluating model

In [47]:
```python
mse = mean_squared_error(y_test, y_pred)
print('MSE: '+ str(mse))
mae = mean_absolute_error(y_test, y_pred)
print('MAE: '+ str(mae))
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print('RMSE: '+ str(rmse))
mape = np.mean(np.abs(y_pred - y_test)/np.abs(y_test))
print('MAPE: '+ str(mape))
```

```
MSE: 0.0006879845274795911
MAE: 0.018529243408105314
RMSE: 0.026229459153394512
MAPE: 0.12575917917569046
```

5.3. Random Forest Classifier (RFC)

In [48]:
```python
# Shift the close price, which mean the next day to be the tomorrow's price
TLS_df['Tomorrow'] = TLS_df['Close'].shift(-1)
TLS_df.head()
```

Out[48]:

Date	Open	High	Low	Close	Volume	Tomorrow
2000-01-04	7.00044	7.00044	6.88222	6.89066	8203436.0	6.75555
2000-01-05	6.75555	6.81466	6.69644	6.75555	9600494.0	6.71333
2000-01-06	6.79778	6.84844	6.71333	6.71333	9683533.0	6.62044
2000-01-07	6.70066	6.73866	6.58666	6.62044	13613336.0	6.71333
2000-01-10	6.75555	6.75555	6.69644	6.71333	6067734.0	6.62044

In [49]:
```python
# Set the target that we try to predict called 'Target'
# return the boolean value, 1 if tomorrow > today's price, otherwise 0
TLS_df['Target'] = (TLS_df['Tomorrow'] > TLS_df['Close']).astype(int)
TLS_df.head()
```

Out[49]:

Date	Open	High	Low	Close	Volume	Tomorrow	Target
2000-01-04	7.00044	7.00044	6.88222	6.89066	8203436.0	6.75555	0
2000-01-05	6.75555	6.81466	6.69644	6.75555	9600494.0	6.71333	0
2000-01-06	6.79778	6.84844	6.71333	6.71333	9683533.0	6.62044	0
2000-01-07	6.70066	6.73866	6.58666	6.62044	13613336.0	6.71333	1
2000-01-10	6.75555	6.75555	6.69644	6.71333	6067734.0	6.62044	0

In [50]:
```python
TLS_df.shape
```

Out[50]:
```
(5156, 7)
```

Train the initial model

Parameters for the RFC model:

- **n_estimators**: the number of individual decision trees
- **min_samples_split**: this help us protect against overfitting
- **random_state**: if we run the same model twice, the random numbers that generated will be in a predictable sequence, which means we'll get the same result

In [51]:
```python
# Training the initial model
RFC_model = RandomForestClassifier(n_estimators = 100,
                                   min_samples_split = 100,
                                   random_state = 1)

# Split train set and test set
train = TLS_df.iloc[:-100]
test = TLS_df.iloc[-100:]

predictors = ["Close", "Volume", "Open", "High", "Low"]
RFC_model.fit(train[predictors], train["Target"])
```

Out[51]:
```
RandomForestClassifier
RandomForestClassifier(min_samples_split=100, random_state=1)
```

In [52]:
```python
# Make prediction
preds = RFC_model.predict(test[predictors])

# Show prediction
preds = pd.Series(preds, index = test.index)
preds
```

Out[52]:
```
Date
2019-11-08    0
2019-11-11    1
2019-11-12    1
2019-11-13    0
2019-11-14    0
             ..
2020-03-26    0
2020-03-27    0
2020-03-30    0
2020-03-31    0
2020-04-01    0
Length: 100, dtype: int32
```

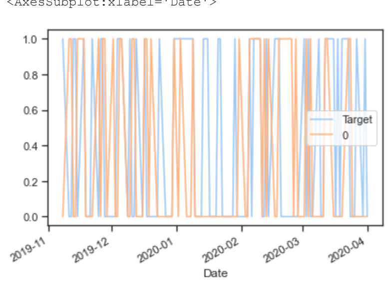
In [53]:
```python
# Calculate the precision score
precision_score(test["Target"], preds)
```

Out[53]:
```
0.40540540540540543
```

Plot prediction

In [54]:
```python
combined = pd.concat([test["Target"], preds], axis=1)
combined.plot()
```

Out[54]:
```
<AxesSubplot:xlabel='Date'>
```



Build backtesting system

In [55]:
```python
# Create function to predict
def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict(test[predictors])
    preds = pd.Series(preds, index = test.index, name="Predictions")
    combined = pd.concat([test["Target"], preds], axis=1)
    return combined
```

In [56]:
```python
# take the first 10-year of data and predict values for the 11th year
# then take the first 11-year of data and predict values for the 12th year
# and so on
def backtest(data, model, predictors, start=2500, step=250):
    all_predictions = []

    for i in range(start, data.shape[0], step):
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train, test, predictors, model)
        all_predictions.append(predictions)
    return pd.concat(all_predictions)
```

In [57]:
```python
# Predict
predictions = backtest(TLS_df, RFC_model, predictors)
```

In [58]:
```python
# Count how many price go up and go down
predictions["Predictions"].value_counts()
```

Out[58]:
```
0    1634
1    1022
Name: Predictions, dtype: int64
```

In [59]:
```python
# Calculate the precision score
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[59]:
```
0.4794520547945205
```

In [60]:
```python
predictions["Target"].value_counts() / predictions.shape[0]
```

Out[60]:
```
0    0.528991
1    0.471009
Name: Target, dtype: float64
```

=> this stock will go down 52.8% and go up 47.1%

Adding additional predictors for the model

In [61]:
```python
horizons = [2,5,60,250,1000]
new_predictors = []

for horizon in horizons:
    rolling_averages = TLS_df.rolling(horizon).mean()

    # The ratio between today's close and the average close in the last 2 (5,60,250,1000) days
    ratio_column = f"Close_Ratio_{horizon}"
    TLS_df[ratio_column] = TLS_df["Close"] / rolling_averages["Close"]

    # The sum of days in the past 2 (5,60,250,1000) days is that the stock price actually went up
    trend_column = f"Trend_{horizon}"
    TLS_df[trend_column] = TLS_df.shift(1).rolling(horizon).sum()["Target"]

    new_predictors += [ratio_column, trend_column]
```

In [62]:
```python
TLS_df = TLS_df.dropna()
```

Improving RFC model

In [63]:
```python
RFC_model = RandomForestClassifier(n_estimators = 200,
                                   min_samples_split = 50,
                                   random_state = 1)
```

In [64]:
```python
def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict_proba(test[predictors])[:,1]

    # Set the custom threshold
    preds[preds >= .6] = 1
    preds[preds < .6] = 0
    preds = pd.Series(preds, index = test.index, name="Predictions")
    combined = pd.concat([test["Target"], preds], axis=1)
    return combined
```

In [65]:
```python
# Make prediction with new model
predictions = backtest(TLS_df, RFC_model, new_predictors)
```

In [66]:
```python
predictions["Predictions"].value_counts()
```

Out[66]:
```
0.0    1635
1.0      20
Name: Predictions, dtype: int64
```

Evaluating model

In [67]:
```python
# Calculate the precision score
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[67]:
```
0.55
```

In [68]:
```python
mse = mean_squared_error(predictions["Target"], predictions["Predictions"])
print('MSE: '+ str(mse))
mae = mean_absolute_error(predictions["Target"], predictions["Predictions"])
print('MAE: '+ str(mae))
rmse = math.sqrt(mean_squared_error(predictions["Target"], predictions["Predictions"]))
print('RMSE: '+ str(rmse))
```

```
MSE: 0.46404833836858006
MAE: 0.46404833836858006
RMSE: 0.6812109352972691
```

6. Conclusion

In the descriptive analysis, I came up with the following findings:

```
1. The opening and closing prices of BHP shares were the highest, outperforming other stocks in 20
years.

2. However, the trading volume of TLS stock is considered larger than that of the remaining stocks.

3. The gap between the highest and lowest prices is not much. Although there are a few outliers, they do
not have a significant impact. It is noticeable that because BHP shares are highly valued and highly
volatile, the split in the highest and lowest prices is wider (about 40 compared to 4-12 for the
remaining stocks).

4. For all stocks, the total number of shares traded does not have a hard rule of how to increase or
decrease in a particular quarter. For example, stocks may be heavily traded in the third quarter of
2015, but in 2017 the second quarter recorded the highest number of transactions of the year. On the
other hand, for 2020, Q1 was recorded as the quarter with the largest total transactions.

5. The same goes for the average closing price of all stocks, which is not known to increase or decrease
in any given quarter of the year. Because the price of a stock can change daily and is affected by many
different factors.
```

In the predictive analysis, I came up with the following conclusions:

- For **SARIMA**, Although using SARIMA gives good parameters, specifically very small error indices, we need to consider whether overfitting occurs or not. Besides, instead of determining parameters p, d, q using adfuller, acf chart or pacf chart, we can use another method like autoARIMA to automatically run and determine the best parameters for the data, from there can improve model performance.
- For **LSTM**, in order to implement this model you need to clearly understand how it works, some more complex transformations such as scaling, data processing into 3D form.
- For **RandomForestClassifier**, although the precision score is not too high (0.55), it is acceptable. The recommendation is that you can try to change/add parameters (max_depth, min_sample_leaf, etc.) or additional predictors to improve the model accuracy. Furthermore, we can also try increasing the resolution, instead of looking at daily data, try to look at hourly or minute by minute data.

Summary: When comparing the error indices (MSE, MAE, RMSE, MAPE) of all three models above, we see that **LSTM** produces significantly smaller errors compared to the other two models. So in this project, to predict other values (Open, High, Low and Volume) for all the stocks in the dataset, I will use LSTM.

References

The dataset is collected from source: https://www.kaggle.com/datasets/ashbellett/australian-historical-stock-prices?resource=download

PETER ABRAHAMSSON, NIKLAS AHLQVIST. (2022). Evaluation of Machine Learning Methods for Time Series Forecasting on E-Commerce Data. Available: https://www.diva-portal.org/smash/get/diva2:1719774/FULLTEXT01.pdf