Stock-RNN-Deep-Learning-TechIndicators

September 29, 2021

1 Recurrent Neural Network - LSTM - Technical Indicators

1.0.1 Importing Libraries

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import RobustScaler
plt.style.use("bmh")
import ta
from datetime import timedelta

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
```

```
Using TensorFlow backend.
```

1.0.2 Loading the Data

[2]:
```
df = pd.read_csv("SPY.csv")
```

1.1 Preprocessing Data

1.1.1 Datetime Conversion

[3]:
```
# Datetime conversion
df['Date'] = pd.to_datetime(df.Date)

# Setting the index
df.set_index('Date', inplace=True)

# Dropping any NaNs
df.dropna(inplace=True)
```

1.1.2 Technical Indicators

[4]:
```
# Adding all the indicators
df = ta.add_all_ta_features(df, open="Open", high="High", low="Low",␣
↪close="Close", volume="Volume", fillna=True)

# Dropping everything else besides 'Close' and the Indicators
df.drop(['Open', 'High', 'Low', 'Adj Close', 'Volume'], axis=1, inplace=True)
```

```
/anaconda3/lib/python3.7/site-packages/ta/trend.py:543: RuntimeWarning: invalid
value encountered in double_scalars
  dip[i] = 100 * (self._dip[i]/self._trs[i])
/anaconda3/lib/python3.7/site-packages/ta/trend.py:547: RuntimeWarning: invalid
value encountered in double_scalars
  din[i] = 100 * (self._din[i]/self._trs[i])
```

[5]:
```
# Checking the new df with indicators
print(df.shape)

df.tail()
```

```
(1259, 69)
```

[5]:
```
                 Close   volume_adi  volume_obv  volume_cmf     volume_fi  \
Date
2020-04-27  287.049988  9.602766e+09    45969900    0.052734  1.688365e+08
2020-04-28  285.730011  9.509076e+09   -59300100   -0.030747  1.248664e+08
2020-04-29  293.209991  9.539094e+09    59445500    0.020667  2.339162e+08
2020-04-30  290.480011  9.514410e+09   -63456200    0.060298  1.525683e+08
2020-05-01  282.790009  9.424102e+09  -188520100   -0.027931 -6.618860e+06

            volume_em  volume_sma_em    volume_vpt   volume_nvi  \
Date
2020-04-27  25.044953       9.208920  2.310252e+06  4561.065425
2020-04-28  11.142787       5.555074  6.390718e+05  4561.065425
2020-04-29  15.979709       6.364424  2.624505e+06  4561.065425
2020-04-30  -6.504168       4.241465  1.964284e+06  4561.065425
2020-05-01 -35.554799       3.172063 -4.455167e+06  4561.065425

            volatility_atr  …  momentum_uo  momentum_stoch  \
Date                        …
2020-04-27        8.512108  …    51.904928       94.781855
2020-04-28        8.260898  …    44.684545       78.317460
2020-04-29        8.349807  …    51.605502       92.884474
2020-04-30        7.987828  …    56.008303       81.252689
2020-05-01        8.103046  …    47.679786       47.112915

            momentum_stoch_signal  momentum_wr  momentum_ao  momentum_kama  \
```

```
Date
2020-04-27              88.060659    -5.218145    17.850146     275.541443
2020-04-28              87.344555   -21.682540    20.306793     275.626071
2020-04-29              88.661263    -7.115526    22.567293     276.905559
2020-04-30              84.151541   -18.747311    23.495617     277.510981
2020-05-01              73.750026   -52.887085    23.726262     277.579561

            momentum_roc  others_dr  others_dlr  others_cr
Date
2020-04-27      4.751301   1.441844    1.431549  35.836636
2020-04-28      2.706685  -0.459842   -0.460903  35.212001
2020-04-29      6.366534   2.617849    2.584170  38.751647
2020-04-30      2.357378  -0.931066   -0.935428  37.459777
2020-05-01      1.810915  -2.647343   -2.683016  33.820746

[5 rows x 69 columns]
```

[6]:
```python
# Only using the last 1000 days of data to get a more accurate representation␣
 ↪of the current climate
df = df.tail(1000)
```

1.1.3 Scaling

[7]:
```python
# Scale fitting the close prices separately for inverse_transformations␣
 ↪purposes later
close_scaler = RobustScaler()

close_scaler.fit(df[['Close']])
```

[7]:
```
RobustScaler(copy=True, quantile_range=(25.0, 75.0), with_centering=True,
       with_scaling=True)
```

[8]:
```python
# Normalizing/Scaling the Data
scaler = RobustScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns, index=df.index)

df.tail(10)
```

[8]:
```
               Close  volume_adi  volume_obv  volume_cmf  volume_fi  \
Date
2020-04-20  0.301717    0.703077   -0.097597    0.020887   4.246393
2020-04-21  0.122697    0.679715   -0.184741    0.016249   0.906682
2020-04-22  0.249581    0.681569   -0.120255   -0.211177   2.180444
2020-04-23  0.249162    0.656015   -0.192453   -0.260639   1.843892
2020-04-24  0.330611    0.673141   -0.133731   -0.453540   2.392542
2020-04-27  0.416038    0.680362   -0.080020   -0.229975   2.828966
```

2020-04-28	0.388400	0.654166	-0.152605	-0.604168	2.056130
2020-04-29	0.545016	0.662559	-0.070729	-0.373708	3.972834
2020-04-30	0.487856	0.655657	-0.155471	-0.196068	2.543029
2020-05-01	0.326843	0.630407	-0.241703	-0.591542	-0.254911

	volume_em	volume_sma_em	volume_vpt	volume_nvi	volatility_atr \
Date					
2020-04-20	-1.085101	7.190792	2.523279	1.901140	4.254813
2020-04-21	-10.350689	4.513933	-6.823563	1.901140	4.233225
2020-04-22	4.023301	8.963496	-2.222827	1.988861	4.126294
2020-04-23	2.753898	10.129771	2.366538	1.988861	3.879511
2020-04-24	-0.430511	9.844314	1.301711	2.045174	3.657951
2020-04-27	5.904110	6.703276	2.656553	2.104238	3.463980
2020-04-28	2.586016	3.943019	0.654017	2.104238	3.327451
2020-04-29	3.740467	4.554434	3.033115	2.104238	3.375772
2020-04-30	-1.625864	2.950669	2.241988	2.104238	3.179041
2020-05-01	-8.559511	2.142801	-5.450292	2.104238	3.241661

	…	momentum_uo	momentum_stoch	momentum_stoch_signal \
Date	…			
2020-04-20	…	-0.024662	0.212624	0.291212
2020-04-21	…	-0.393401	-0.207080	0.154458
2020-04-22	…	-0.154273	0.083892	0.025250
2020-04-23	…	-0.721260	0.076755	-0.021363
2020-04-24	…	-0.318395	0.257172	0.137907
2020-04-27	…	-0.402651	0.381749	0.240091
2020-04-28	…	-0.895522	0.030986	0.224390
2020-04-29	…	-0.423090	0.341327	0.253260
2020-04-30	…	-0.122550	0.093519	0.154378
2020-05-01	…	-0.691064	-0.633806	-0.073689

	momentum_wr	momentum_ao	momentum_kama	momentum_roc	others_dr \
Date					
2020-04-20	0.212624	2.151083	0.146169	5.171074	-2.451698
2020-04-21	-0.207080	2.129633	0.145297	2.881552	-4.173474
2020-04-22	0.083892	2.244571	0.149277	4.426382	2.926631
2020-04-23	0.076755	2.464789	0.150657	1.711584	-0.081350
2020-04-24	0.257172	2.407330	0.153165	2.232642	1.811339
2020-04-27	0.381749	2.447829	0.163056	1.474955	1.876147
2020-04-28	0.030986	2.844634	0.164805	0.691541	-0.692868
2020-04-29	0.341327	3.209757	0.191245	2.093847	3.464828
2020-04-30	0.093519	3.359702	0.203757	0.557701	-1.329451
2020-05-01	-0.633806	3.396957	0.205174	0.348319	-3.647993

	others_dlr	others_cr
Date		
2020-04-20	-2.475376	0.301717

```
2020-04-21   -4.241268   0.122697
2020-04-22    2.896765   0.249581
2020-04-23   -0.081413   0.249162
2020-04-24    1.800160   0.330611
2020-04-27    1.864126   0.416038
2020-04-28   -0.694978   0.388400
2020-04-29    3.422780   0.545016
2020-04-30   -1.336664   0.487856
2020-05-01   -3.699874   0.326843

[10 rows x 69 columns]
```

1.1.4 Plotting

[9]:
```
# Plotting the Closing Prices
df['Close'].plot(figsize=(16,5))
plt.title("Closing Prices")
plt.ylabel("Price (Scaled)")
plt.show()
```

1.1.5 Functions to prepare the data for LSTM

[10]:
```
def split_sequence(seq, n_steps_in, n_steps_out):
    """
    Splits the multivariate time sequence
    """

    # Creating a list for both variables
    X, y = [], []

    for i in range(len(seq)):

        # Finding the end of the current sequence
```

```
        end = i + n_steps_in
        out_end = end + n_steps_out

        # Breaking out of the loop if we have exceeded the dataset's length
        if out_end > len(seq):
            break

        # Splitting the sequences into: x = past prices and indicators, y =␣
↪prices ahead
        seq_x, seq_y = seq[i:end, :], seq[end:out_end, 0]

        X.append(seq_x)
        y.append(seq_y)

    return np.array(X), np.array(y)
```

[11]:
```
def visualize_training_results(results):
    """
    Plots the loss and accuracy for the training and testing data
    """
    history = results.history
    plt.figure(figsize=(16,5))
    plt.plot(history['val_loss'])
    plt.plot(history['loss'])
    plt.legend(['val_loss', 'loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()

    plt.figure(figsize=(16,5))
    plt.plot(history['val_accuracy'])
    plt.plot(history['accuracy'])
    plt.legend(['val_accuracy', 'accuracy'])
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.show()
```

[12]:
```
def layer_maker(n_layers, n_nodes, activation, drop=None, d_rate=.5):
    """
    Creates a specified number of hidden layers for an RNN
    Optional: Adds regularization option - the dropout layer to prevent␣
↪potential overfitting (if necessary)
    """
```

```python
    # Creating the specified number of hidden layers with the specified number␣
↪of nodes
    for x in range(1,n_layers+1):
        model.add(LSTM(n_nodes, activation=activation, return_sequences=True))

        # Adds a Dropout layer after every Nth hidden layer (the 'drop'␣
↪variable)
        try:
            if x % drop == 0:
                model.add(Dropout(d_rate))
        except:
            pass
```

[13]:
```python
def validater(n_per_in, n_per_out):
    """
    Runs a 'For' loop to iterate through the length of the DF and create␣
↪predicted values for every stated interval
    Returns a DF containing the predicted values for the model with the␣
↪corresponding index values based on a business day frequency
    """

    # Creating an empty DF to store the predictions
    predictions = pd.DataFrame(index=df.index, columns=[df.columns[0]])

    for i in range(1, len(df)-n_per_in, n_per_out):
        # Creating rolling intervals to predict off of
        x = df[-i - n_per_in:-i]

        # Predicting using rolling intervals
        yhat = model.predict(np.array(x).reshape(1, n_per_in, n_features))

        # Transforming values back to their normal prices
        yhat = close_scaler.inverse_transform(yhat)[0]

        # DF to store the values and append later, frequency uses business days
        pred_df = pd.DataFrame(yhat,
                               index=pd.date_range(start=x.
↪index[-1]+timedelta(days=1),
                                                   periods=len(yhat),
                                                   freq="B"),
                               columns=[x.columns[0]])

        # Updating the predictions DF
        predictions.update(pred_df)

    return predictions
```

```
[14]: def val_rmse(df1, df2):
    """
    Calculates the root mean square error between the two Dataframes
    """
    df = df1.copy()

    # Adding a new column with the closing prices from the second DF
    df['close2'] = df2.Close

    # Dropping the NaN values
    df.dropna(inplace=True)

    # Adding another column containing the difference between the two DFs'␣
 ↪closing prices
    df['diff'] = df.Close - df.close2

    # Squaring the difference and getting the mean
    rms = (df[['diff']]**2).mean()

    # Returning the sqaure root of the root mean square
    return float(np.sqrt(rms))
```

1.1.6 Splitting the Data

```
[15]: # How many periods looking back to learn
n_per_in  = 90

# How many periods to predict
n_per_out = 30

# Features
n_features = df.shape[1]

# Splitting the data into appropriate sequences
X, y = split_sequence(df.to_numpy(), n_per_in, n_per_out)
```

1.2 Modeling - LSTM (RNN)

1.2.1 Creating the Neural Network

```
[16]: # Instatiating the model
model = Sequential()

# Activation
activ = "tanh"

# Input layer
```

```
model.add(LSTM(90,
               activation=activ,
               return_sequences=True,
               input_shape=(n_per_in, n_features)))

# Hidden layers
layer_maker(n_layers=2,
            n_nodes=30,
            activation=activ,
            drop=1,
            d_rate=.1)

# Final Hidden layer
model.add(LSTM(90, activation=activ))

# Output layer
model.add(Dense(n_per_out))

# Model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 90, 90)            57600
_________________________________________________________________
lstm_2 (LSTM)                (None, 90, 30)            14520
_________________________________________________________________
dropout_1 (Dropout)          (None, 90, 30)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 90, 30)            7320
_________________________________________________________________
dropout_2 (Dropout)          (None, 90, 30)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 90)                43560
_________________________________________________________________
dense_1 (Dense)              (None, 30)                2730
=================================================================
Total params: 125,730
Trainable params: 125,730
Non-trainable params: 0
_________________________________________________________________
```

[17]:
```
# Compiling the data with selected specifications
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
```

1.2.2 Fitting and Training the RNN

[18]:
```
res = model.fit(X, y, epochs=100, batch_size=32, validation_split=0.1)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 792 samples, validate on 89 samples
Epoch 1/100
792/792 [==============================] - 6s 8ms/step - loss: 0.1144 -
accuracy: 0.0341 - val_loss: 0.5180 - val_accuracy: 0.0225
Epoch 2/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0236 -
accuracy: 0.0530 - val_loss: 0.5590 - val_accuracy: 0.0225
Epoch 3/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0129 -
accuracy: 0.0909 - val_loss: 0.4665 - val_accuracy: 0.0225
Epoch 4/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0097 -
accuracy: 0.1111 - val_loss: 0.4344 - val_accuracy: 0.0449
Epoch 5/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0077 -
accuracy: 0.1048 - val_loss: 0.4176 - val_accuracy: 0.0337
Epoch 6/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0067 -
accuracy: 0.1275 - val_loss: 0.4202 - val_accuracy: 0.0337
Epoch 7/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0059 -
accuracy: 0.1250 - val_loss: 0.4269 - val_accuracy: 0.0112
Epoch 8/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0053 -
accuracy: 0.1326 - val_loss: 0.4234 - val_accuracy: 0.1011
Epoch 9/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0049 -
accuracy: 0.1301 - val_loss: 0.4322 - val_accuracy: 0.0674
Epoch 10/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0046 -
accuracy: 0.1465 - val_loss: 0.4372 - val_accuracy: 0.0337
Epoch 11/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0043 -
accuracy: 0.1149 - val_loss: 0.4237 - val_accuracy: 0.0674
Epoch 12/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0040 -
accuracy: 0.1237 - val_loss: 0.4361 - val_accuracy: 0.0337
Epoch 13/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0039 -
accuracy: 0.1338 - val_loss: 0.4373 - val_accuracy: 0.0562
```

```
Epoch 14/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0038 -
accuracy: 0.1326 - val_loss: 0.4520 - val_accuracy: 0.0449
Epoch 15/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0035 -
accuracy: 0.1439 - val_loss: 0.4412 - val_accuracy: 0.0337
Epoch 16/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0035 -
accuracy: 0.1465 - val_loss: 0.4322 - val_accuracy: 0.0674
Epoch 17/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0033 -
accuracy: 0.1225 - val_loss: 0.4314 - val_accuracy: 0.1124
Epoch 18/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0032 -
accuracy: 0.1439 - val_loss: 0.4203 - val_accuracy: 0.0674
Epoch 19/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0034 -
accuracy: 0.1629 - val_loss: 0.4289 - val_accuracy: 0.0112
Epoch 20/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0031 -
accuracy: 0.1402 - val_loss: 0.4403 - val_accuracy: 0.0674
Epoch 21/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0031 -
accuracy: 0.1540 - val_loss: 0.4417 - val_accuracy: 0.0899
Epoch 22/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0033 -
accuracy: 0.1591 - val_loss: 0.4391 - val_accuracy: 0.0787
Epoch 23/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0031 -
accuracy: 0.1856 - val_loss: 0.4338 - val_accuracy: 0.0674
Epoch 24/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0028 -
accuracy: 0.1932 - val_loss: 0.4422 - val_accuracy: 0.0899
Epoch 25/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0028 -
accuracy: 0.1730 - val_loss: 0.4376 - val_accuracy: 0.1236
Epoch 26/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0027 -
accuracy: 0.1932 - val_loss: 0.4439 - val_accuracy: 0.1236
Epoch 27/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0027 -
accuracy: 0.2071 - val_loss: 0.4413 - val_accuracy: 0.0899
Epoch 28/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0028 -
accuracy: 0.1793 - val_loss: 0.4473 - val_accuracy: 0.0899
Epoch 29/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0026 -
accuracy: 0.1919 - val_loss: 0.4424 - val_accuracy: 0.1124
```

```
Epoch 30/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0027 -
accuracy: 0.2058 - val_loss: 0.4429 - val_accuracy: 0.1124
Epoch 31/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0026 -
accuracy: 0.1894 - val_loss: 0.4352 - val_accuracy: 0.1348
Epoch 32/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0025 -
accuracy: 0.1995 - val_loss: 0.4439 - val_accuracy: 0.1685
Epoch 33/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0024 -
accuracy: 0.1843 - val_loss: 0.4468 - val_accuracy: 0.1124
Epoch 34/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0025 -
accuracy: 0.2096 - val_loss: 0.4498 - val_accuracy: 0.1124
Epoch 35/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0024 -
accuracy: 0.2096 - val_loss: 0.4455 - val_accuracy: 0.1685
Epoch 36/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0027 -
accuracy: 0.1932 - val_loss: 0.4502 - val_accuracy: 0.1348
Epoch 37/100
792/792 [==============================] - 4s 6ms/step - loss: 0.0025 -
accuracy: 0.2020 - val_loss: 0.4495 - val_accuracy: 0.1798
Epoch 38/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0025 -
accuracy: 0.2210 - val_loss: 0.4566 - val_accuracy: 0.1685
Epoch 39/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0023 -
accuracy: 0.2071 - val_loss: 0.4524 - val_accuracy: 0.1573
Epoch 40/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0023 -
accuracy: 0.2083 - val_loss: 0.4562 - val_accuracy: 0.1685
Epoch 41/100
792/792 [==============================] - 4s 6ms/step - loss: 0.0023 -
accuracy: 0.2260 - val_loss: 0.4543 - val_accuracy: 0.1685
Epoch 42/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0024 -
accuracy: 0.2083 - val_loss: 0.4587 - val_accuracy: 0.1573
Epoch 43/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2235 - val_loss: 0.4554 - val_accuracy: 0.1685
Epoch 44/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0023 -
accuracy: 0.1982 - val_loss: 0.4559 - val_accuracy: 0.1573
Epoch 45/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2172 - val_loss: 0.4631 - val_accuracy: 0.1573
```

```
Epoch 46/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2121 - val_loss: 0.4465 - val_accuracy: 0.1573
Epoch 47/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2235 - val_loss: 0.4708 - val_accuracy: 0.1798
Epoch 48/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2134 - val_loss: 0.4489 - val_accuracy: 0.1461
Epoch 49/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2336 - val_loss: 0.4667 - val_accuracy: 0.1573
Epoch 50/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0021 -
accuracy: 0.2222 - val_loss: 0.4611 - val_accuracy: 0.1685
Epoch 51/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2285 - val_loss: 0.4724 - val_accuracy: 0.1461
Epoch 52/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0023 -
accuracy: 0.2083 - val_loss: 0.4552 - val_accuracy: 0.1685
Epoch 53/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0021 -
accuracy: 0.2361 - val_loss: 0.4681 - val_accuracy: 0.1573
Epoch 54/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2298 - val_loss: 0.4666 - val_accuracy: 0.1685
Epoch 55/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2184 - val_loss: 0.4594 - val_accuracy: 0.1685
Epoch 56/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2374 - val_loss: 0.4651 - val_accuracy: 0.1685
Epoch 57/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2197 - val_loss: 0.4610 - val_accuracy: 0.1685
Epoch 58/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0019 -
accuracy: 0.2311 - val_loss: 0.4452 - val_accuracy: 0.1685
Epoch 59/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2298 - val_loss: 0.4683 - val_accuracy: 0.1685
Epoch 60/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0024 -
accuracy: 0.2412 - val_loss: 0.4555 - val_accuracy: 0.1685
Epoch 61/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0022 -
accuracy: 0.2298 - val_loss: 0.4584 - val_accuracy: 0.1685
```

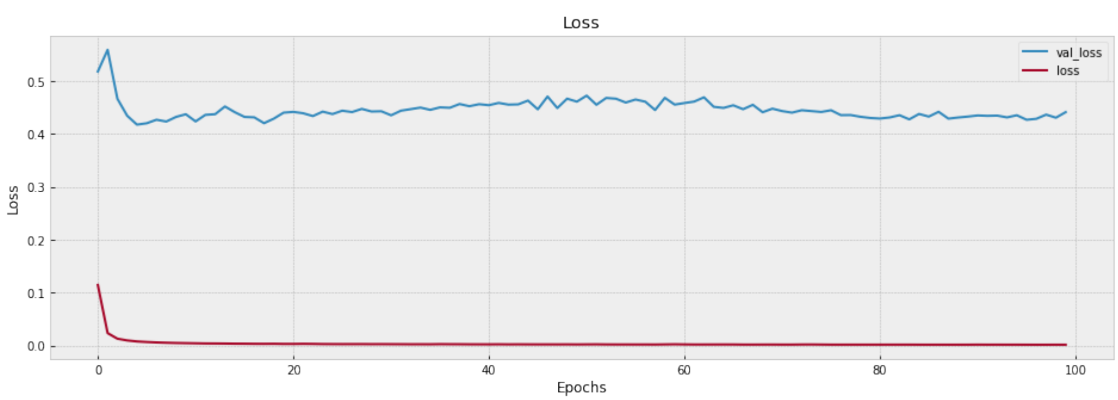
```
Epoch 62/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2348 - val_loss: 0.4612 - val_accuracy: 0.1685
Epoch 63/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2210 - val_loss: 0.4692 - val_accuracy: 0.1798
Epoch 64/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2323 - val_loss: 0.4513 - val_accuracy: 0.1685
Epoch 65/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2247 - val_loss: 0.4492 - val_accuracy: 0.1685
Epoch 66/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2348 - val_loss: 0.4542 - val_accuracy: 0.1685
Epoch 67/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0018 -
accuracy: 0.2323 - val_loss: 0.4467 - val_accuracy: 0.1685
Epoch 68/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0018 -
accuracy: 0.2336 - val_loss: 0.4551 - val_accuracy: 0.1461
Epoch 69/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0019 -
accuracy: 0.2361 - val_loss: 0.4411 - val_accuracy: 0.1685
Epoch 70/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0019 -
accuracy: 0.2235 - val_loss: 0.4478 - val_accuracy: 0.1685
Epoch 71/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0018 -
accuracy: 0.2449 - val_loss: 0.4432 - val_accuracy: 0.1461
Epoch 72/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0018 -
accuracy: 0.2424 - val_loss: 0.4403 - val_accuracy: 0.1685
Epoch 73/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2361 - val_loss: 0.4448 - val_accuracy: 0.1685
Epoch 74/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0020 -
accuracy: 0.2399 - val_loss: 0.4432 - val_accuracy: 0.1685
Epoch 75/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0019 -
accuracy: 0.2285 - val_loss: 0.4415 - val_accuracy: 0.1685
Epoch 76/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2500 - val_loss: 0.4448 - val_accuracy: 0.1685
Epoch 77/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2538 - val_loss: 0.4356 - val_accuracy: 0.1685
```

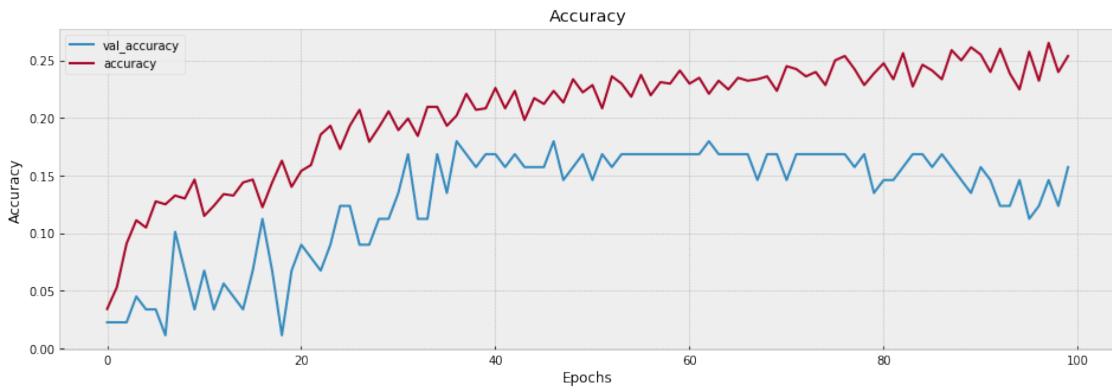
```
Epoch 78/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2424 - val_loss: 0.4357 - val_accuracy: 0.1573
Epoch 79/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2285 - val_loss: 0.4325 - val_accuracy: 0.1685
Epoch 80/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2386 - val_loss: 0.4302 - val_accuracy: 0.1348
Epoch 81/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2475 - val_loss: 0.4291 - val_accuracy: 0.1461
Epoch 82/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2336 - val_loss: 0.4312 - val_accuracy: 0.1461
Epoch 83/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2563 - val_loss: 0.4354 - val_accuracy: 0.1573
Epoch 84/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2273 - val_loss: 0.4276 - val_accuracy: 0.1685
Epoch 85/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2462 - val_loss: 0.4378 - val_accuracy: 0.1685
Epoch 86/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2412 - val_loss: 0.4328 - val_accuracy: 0.1573
Epoch 87/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2336 - val_loss: 0.4419 - val_accuracy: 0.1685
Epoch 88/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2588 - val_loss: 0.4289 - val_accuracy: 0.1573
Epoch 89/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2500 - val_loss: 0.4311 - val_accuracy: 0.1461
Epoch 90/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2614 - val_loss: 0.4328 - val_accuracy: 0.1348
Epoch 91/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2551 - val_loss: 0.4349 - val_accuracy: 0.1573
Epoch 92/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2399 - val_loss: 0.4342 - val_accuracy: 0.1461
Epoch 93/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0017 -
accuracy: 0.2601 - val_loss: 0.4346 - val_accuracy: 0.1236
```

```
Epoch 94/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2386 - val_loss: 0.4313 - val_accuracy: 0.1236
Epoch 95/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2247 - val_loss: 0.4353 - val_accuracy: 0.1461
Epoch 96/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2576 - val_loss: 0.4268 - val_accuracy: 0.1124
Epoch 97/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0015 -
accuracy: 0.2323 - val_loss: 0.4286 - val_accuracy: 0.1236
Epoch 98/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0015 -
accuracy: 0.2652 - val_loss: 0.4365 - val_accuracy: 0.1461
Epoch 99/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0016 -
accuracy: 0.2399 - val_loss: 0.4307 - val_accuracy: 0.1236
Epoch 100/100
792/792 [==============================] - 5s 6ms/step - loss: 0.0015 -
accuracy: 0.2538 - val_loss: 0.4413 - val_accuracy: 0.1573
```

1.2.3 Plotting the Accuracy and Loss

[19]:
```
visualize_training_results(res)
```





1.3 Visualizing the Predictions

1.3.1 Validating the Model

Plotting the difference between the Actual closing prices and the Predicted prices

[20]:
```
# Transforming the actual values to their original price
actual = pd.DataFrame(close_scaler.inverse_transform(df[["Close"]]),
                      index=df.index,
                      columns=[df.columns[0]])

# Getting a DF of the predicted values to validate against
predictions = validater(n_per_in, n_per_out)

# Printing the RMSE
print("RMSE:", val_rmse(actual, predictions))

# Plotting
plt.figure(figsize=(16,6))

# Plotting those predictions
plt.plot(predictions, label='Predicted')

# Plotting the actual values
plt.plot(actual, label='Actual')

plt.title(f"Predicted vs Actual Closing Prices")
plt.ylabel("Price")
plt.legend()
plt.xlim('2018-05', '2020-05')
plt.show()
```

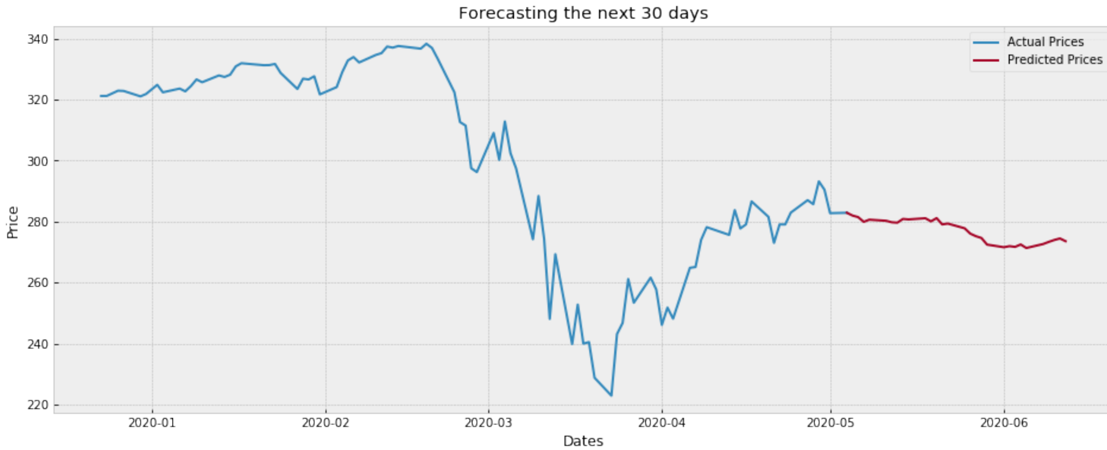
```
RMSE: 10.109217273920311
```

1.3.2 Predicting/Forecasting the future prices

[21]:
```
# Predicting off of the most recent days from the original DF
yhat = model.predict(np.array(df.tail(n_per_in)).reshape(1, n_per_in,␣
↪n_features))

# Transforming the predicted values back to their original format
yhat = close_scaler.inverse_transform(yhat)[0]

# Creating a DF of the predicted prices
preds = pd.DataFrame(yhat,
                     index=pd.date_range(start=df.index[-1]+timedelta(days=1),
                                         periods=len(yhat),
                                         freq="B"),
                     columns=[df.columns[0]])

# Number of periods back to plot the actual values
pers = n_per_in

# Transforming the actual values to their original price
actual = pd.DataFrame(close_scaler.inverse_transform(df[["Close"]].tail(pers)),
                      index=df.Close.tail(pers).index,
                      columns=[df.columns[0]]).append(preds.head(1))

# Printing the predicted prices
print(preds)

# Plotting
plt.figure(figsize=(16,6))
plt.plot(actual, label="Actual Prices")
plt.plot(preds, label="Predicted Prices")
```

```python
plt.ylabel("Price")
plt.xlabel("Dates")
plt.title(f"Forecasting the next {len(yhat)} days")
plt.legend()
plt.show()
```

```
                 Close
2020-05-04  282.918671
2020-05-05  282.004364
2020-05-06  281.497040
2020-05-07  279.946838
2020-05-08  280.647980
2020-05-11  280.283142
2020-05-12  279.801086
2020-05-13  279.665558
2020-05-14  280.898163
2020-05-15  280.740723
2020-05-18  281.119965
2020-05-19  280.099854
2020-05-20  281.166199
2020-05-21  279.094482
2020-05-22  279.372681
2020-05-25  277.824799
2020-05-26  276.083313
2020-05-27  275.239197
2020-05-28  274.644531
2020-05-29  272.477295
2020-06-01  271.616821
2020-06-02  271.966614
2020-06-03  271.730042
2020-06-04  272.520721
2020-06-05  271.362488
2020-06-08  272.663757
2020-06-09  273.363983
2020-06-10  273.996521
2020-06-11  274.488281
2020-06-12  273.598694
```


Forecasting the next 30 days
Actual Prices
Predicted Prices
Price
340
320
300
280
260
240
220
2020-01
2020-02
2020-03
2020-04
2020-05
2020-06
Dates

```
[ ]:
```