Appendix 1

Figure 8

Python Script – Feature Scaling/Normalisation

```
In [5]: from sklearn.preprocessing import MinMaxScaler
        sc = MinMaxScaler(feature_range=(0,1))
        training_set_scaled = sc.fit_transform(training_set)

In [6]: print(training_set_scaled)

        [[0.08581368]
         [0.09701243]
         [0.09433366]
         ...
         [0.95725128]
         [0.93796041]
         [0.93688146]]
```

Source: authors’ own work.

Figure 9

Python Script – Creating a Data Structure with 60 Time Steps and One Outcome

```
In [7]: X_train = []
        y_train = []
        for i in range(60, 1257):
            X_train.append(training_set_scaled[i-60:i, 0])
            y_train.append(training_set_scaled[i+1, 0])
        X_train, y_train = np.array(X_train), np.array(y_train)

In [8]: print(X_train)

        [[0.08581368 0.09701243 0.09433366 ... 0.07846566 0.08034452 0.08497656]
         [0.09701243 0.09433366 0.09156187 ... 0.08034452 0.08497656 0.08627874]
         [0.09433366 0.09156187 0.07984225 ... 0.08497656 0.08627874 0.08471612]
         ...
         [0.92493861 0.92106928 0.92438053 ... 0.96123223 0.95475854 0.95204256]
         [0.92106928 0.92438053 0.93048218 ... 0.95475854 0.95204256 0.95163331]
         [0.92438053 0.93048218 0.9299055  ... 0.95204256 0.95163331 0.95725128]]

In [9]: pd.DataFrame(y_train)
```

Out[9]:

	0
0	0.084716
1	0.074541
2	0.078838
3	0.072383
4	0.066634
...	...
1192	0.952043
1193	0.951633
1194	0.957251
1195	0.937960
1196	0.936881

1197 rows × 1 columns

Source: authors’ own work.

Figure 10

Python Script – Data Transformation Step

```
In [10]: X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

In [11]: print(X_train)

[[[0.08581368]
  [0.09701243]
  [0.09433366]
  ...
  [0.07846566]
  [0.08034452]
  [0.08497656]]

 [[0.09701243]
  [0.09433366]
  [0.09156187]
  ...
  [0.08034452]
  [0.08497656]
  [0.08627874]]

 [[0.09433366]
  [0.09156187]
  [0.07984225]
  ...
  [0.08497656]
  [0.08627874]
  [0.08471612]]

 ...
```

Source: authors' own work.

Figure 11

Python Scripts – Part 2. Building an RNN

```
In [12]: from keras.models import Sequential
         from keras.layers import Dense
         from keras.layers import LSTM
         from keras.layers import Dropout

In [13]: regressor = Sequential()
```

Adding the first LSTM layer and some Dropout regularisation

```
In [14]: regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
         regressor.add(Dropout(0.2))
```

Adding 4 LSTM layers and Dropout regularisations

```
In [15]: regressor.add(LSTM(units = 50, return_sequences = True))
         regressor.add(Dropout(0.2))

In [16]: regressor.add(LSTM(units = 50, return_sequences = True))
         regressor.add(Dropout(0.2))

In [17]: regressor.add(LSTM(units = 50))
         regressor.add(Dropout(0.2))
```

Adding the output layer

```
In [18]: regressor.add(Dense(units = 1))
```

Compiling the RNN

```
In [19]: regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Source: authors' own work.

Figure 12

Python Script – Fitting an RNN to the Training Set

```
In [20]: regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)

Epoch 98/100
38/38 [==============================] - 2s 42ms/step - loss: 0.0019
Epoch 99/100
38/38 [==============================] - 2s 42ms/step - loss: 0.0021
Epoch 100/100
38/38 [==============================] - 2s 42ms/step - loss: 0.0018
Out[20]: <keras.callbacks.History at 0x7fe8db6474c0>
```

Source: authors' own work.

Figure 13

Python Scripts – Part 3. Forecasting and Data Visualisation

Getting the real stock price of 2017

```
In [21]: dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
         real_stock_price = dataset_test.iloc[:, 1:2].values

In [22]: print(dataset_test)

         Date    Open    High     Low   Close     Volume
0    1/3/2017  778.81  789.63  775.80  786.14  1,657,300
1    1/4/2017  788.36  791.34  783.16  786.90  1,073,000
2    1/5/2017  786.08  794.48  785.02  794.02  1,335,200
3    1/6/2017  795.26  807.90  792.20  806.15  1,640,200
4    1/9/2017  806.40  809.97  802.83  806.65  1,272,400
5   1/10/2017  807.86  809.13  803.51  804.79  1,176,800
6   1/11/2017  805.00  808.15  801.37  807.91  1,065,900
7   1/12/2017  807.14  807.39  799.17  806.36  1,353,100
8   1/13/2017  807.48  811.22  806.69  807.88  1,099,200
9   1/17/2017  807.08  807.14  800.37  804.61  1,362,100
10  1/18/2017  805.81  806.21  800.99  806.07  1,294,400
11  1/19/2017  805.12  809.48  801.80  802.17    919,300
12  1/20/2017  806.91  806.91  801.69  805.02  1,670,000
13  1/23/2017  807.25  820.87  803.74  819.31  1,963,600
14  1/24/2017  822.30  825.90  817.82  823.87  1,474,000
15  1/25/2017  829.62  835.77  825.06  835.67  1,494,500
16  1/26/2017  837.81  838.00  827.01  832.15  2,973,900
17  1/27/2017  834.71  841.95  820.44  823.31  2,965,800
18  1/30/2017  814.66  815.84  799.80  802.32  3,246,600
19  1/31/2017  796.86  801.25  790.52  796.79  2,160,600

In [23]: print(real_stock_price)

[[778.81]
 [788.36]
 [786.08]
 [795.26]
 [806.4 ]
 [807.86]
 [805.  ]
 [807.14]
 [807.48]
 [807.08]
 [805.81]
 [805.12]
 [806.91]
 [807.25]
 [822.3 ]
 [829.62]
 [837.81]
 [834.71]
 [814.66]
 [796.86]]
```

Source: authors' own work.

Figure 14

Python Script – Retrieving the Forecasted Stock Price from 2017

```
In [24]: dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis=0)
         inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
         inputs = inputs.reshape(-1,1)
         inputs = sc.transform(inputs)

         X_test = []
         for i in range(60, 80):
             X_test.append(inputs[i-60:i, 0])

         X_test = np.array(X_test)
         X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

         predicted_stock_price = regressor.predict(X_test)
         predicted_stock_price = sc.inverse_transform(predicted_stock_price)

         1/1 [==============================] - 1s 649ms/step

In [25]: print(predicted_stock_price)

         [[769.5995 ]
          [767.2304 ]
          [766.9906 ]
          [767.723  ]
          [770.09265]
          [774.6596 ]
          [779.22723]
          [781.5972 ]
          [782.45874]
          [782.6622 ]
          [782.66956]
          [782.54407]
          [782.4236 ]
          [782.84283]
          [783.6622 ]
          [787.353  ]
          [793.1025 ]
          [799.58813]
          [803.7723 ]
          [801.68805]]
```

Source: authors' own work.