Predicting Stock Prices with Linear Regression

To answer the question if we can accurately predict stock prices over time, I'm going to use Stock Price history data from the Quadl API and apply a regression analysis method.

Check out the Data

Import Libraries

In [69]:

```
#import warnings
#warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import quandl
import datetime

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-darkgrid')
plt.rc('figure', figsize=(16,10))
plt.rc('lines', markersize=4)
```

Configure Quandl

In [ ]:

```
# Import API key from file
import API_config
```

In [ ]:

```
# Quandl API Auth
quandl.ApiConfig.api_key = API_config.API_KEY
```

Get the Data

In [36]:

```
# Set start and end date for stock prices
start_date = datetime.date(2009, 3,8)
end_date = datetime.date.today()
# Load data from Quandl
data = quandl.get('FSE/SAP_X', start_date=start_date, end_date=end_date)
# Save data to CSV file
data.to_csv('data/sap_stock.csv')
```

In [37]:

```
data.head()
```

Out[37]:

	Open	High	Low	Close	Change	Traded Volume	Turnover	Last Price of the Day	Daily Traded Units	Daily Turnover
Date										
2009-03-09	25.16	25.82	24.48	25.59	NaN	5749357.0	145200289.0	None	None	NaN
2009-03-10	25.68	26.95	25.68	26.87	NaN	7507770.0	198480965.0	None	None	NaN
2009-03-11	26.50	26.95	26.26	26.64	NaN	5855095.0	155815439.0	None	None	NaN
2009-03-12	26.15	26.47	25.82	26.18	NaN	6294955.0	164489409.0	None	None	NaN
2009-03-13	26.01	26.24	25.65	25.73	NaN	6814568.0	176228331.0	None	None	NaN

In [38]:

```
# Check data types in columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2623 entries, 2009-03-09 to 2019-06-25
Data columns (total 10 columns):
Open                     2242 non-null float64
High                     2616 non-null float64
Low                      2616 non-null float64
Close                    2623 non-null float64
Change                   11 non-null float64
Traded Volume            2577 non-null float64
Turnover                 2570 non-null float64
Last Price of the Day    0 non-null object
Daily Traded Units       0 non-null object
Daily Turnover           7 non-null float64
dtypes: float64(8), object(2)
memory usage: 225.4+ KB
```

In [39]:

```
# Get descriptive statistics summary of data set
data.describe()
```

Out[39]:

	Open	High	Low	Close	Change	Traded Volume	Tu
count	2242.000000	2616.000000	2616.000000	2623.000000	11.000000	2.577000e+03	2.5700
mean	56.686896	62.881705	61.829606	62.305957	-0.070000	3.277231e+06	1.8568
std	18.320821	22.322180	22.039678	22.227715	0.709761	1.991769e+06	9.7098
min	25.160000	25.820000	24.480000	25.590000	-0.740000	0.000000e+00	1.7673
25%	41.500000	43.815000	42.917500	43.340000	-0.500000	2.124596e+06	1.3072
50%	56.560000	58.990000	58.045000	58.430000	-0.290000	2.811753e+06	1.6478
75%	67.732500	80.900000	79.802500	80.405000	0.085000	3.848936e+06	2.1347
max	100.100000	119.740000	118.320000	118.820000	1.250000	3.645671e+07	1.3694

In [40]:

```
# Display features in data set
data.columns
```

Out[40]:

```
Index(['Open', 'High', 'Low', 'Close', 'Change', 'Traded Volume', 'Turnove
r',
       'Last Price of the Day', 'Daily Traded Units', 'Daily Turnover'],
      dtype='object')
```

Select Subset with relevant features

We use the daily closing price **Close** as the value to predict, so we can discard the other features.

- 'Close' column has numerical data type
- The 'Date' is the index column and contains datetime values

In [41]:

```
# Create a new DataFrame with only closing price and date
df = pd.DataFrame(data, columns=['Close'])

# Reset index column so that we have integers to represent time for later analysis
df = df.reset_index()
```

In [42]:

```
df.head()
```

Out[42]:

	Date	Close
0	2009-03-09	25.59
1	2009-03-10	26.87
2	2009-03-11	26.64
3	2009-03-12	26.18
4	2009-03-13	25.73

In [43]:

```
# Check data types in columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2623 entries, 0 to 2622
Data columns (total 2 columns):
Date     2623 non-null datetime64[ns]
Close    2623 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 41.1 KB
```

In [44]:

```
# Check for missing values in the columns
df.isna().values.any()
```

Out[44]:

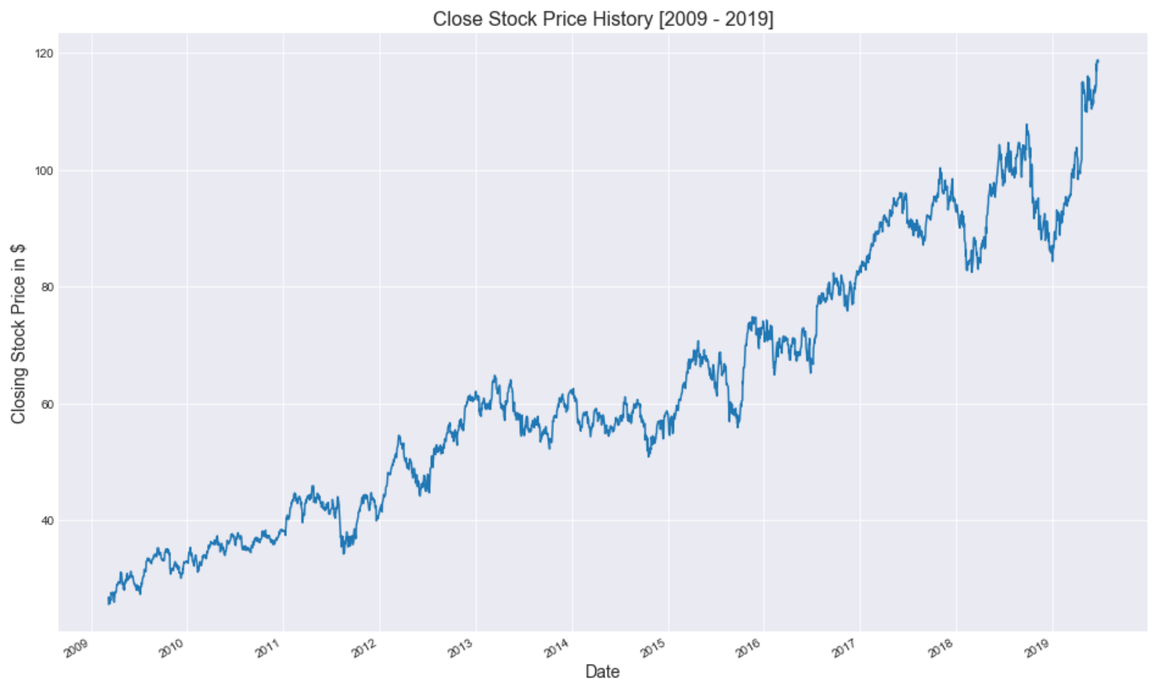
```
False
```

Explore the Data

When we take a look at the price movement over time by simply plotting the *Closing price* vs *Time*, we can already see, that the price continously increases over time and we can also estimate that trend could be linear.

In [45]:

```
# Import matplotlib package for date plots
import matplotlib.dates as mdates

years = mdates.YearLocator() # Get every year
yearsFmt = mdates.DateFormatter('%Y') # Set year format

# Create subplots to plot graph and control axes
fig, ax = plt.subplots()
ax.plot(df['Date'], df['Close'])

# Format the ticks
ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(yearsFmt)

# Set figure title
plt.title('Close Stock Price History [2009 - 2019]', fontsize=16)
# Set x label
plt.xlabel('Date', fontsize=14)
# Set y label
plt.ylabel('Closing Stock Price in $', fontsize=14)

# Rotate and align the x labels
fig.autofmt_xdate()

# Show plot
plt.show()
```



Linear Regression

Our data contains only one **independent variable (X)** which represents the *date* and the **dependent variable (Y)** we are trying to predict is the *Stock Price*. To fit a line to the data points, which then represents an estimated relationship between X and Y, we can use a **Simple Linear Regression**.

The best fit line can be described with

$$Y = \beta_0 + \beta_1 X$$

where

- Y is the predicted value of the dependent variable
- β_0 is the y-intercept
- β_1 is the slope
- X is the value of the independent variable

The goal is to find such coefficients β_0 and β_1 that the **Sum of Squared Errors**, which represents the difference between each point in the dataset with it's corresponding predicted value outputted by the model, is minimal.

Training a Linear Regression Model

Train Test Split

In [46]:

```
# Import package for splitting data set
from sklearn.model_selection import train_test_split
```

In [47]:

```
# Split data into train and test set: 80% / 20%
train, test = train_test_split(df, test_size=0.20)
```

Create and Train the Model

In [48]:

```
# Import package for linear model
from sklearn.linear_model import LinearRegression
```

In [49]:

```
# Reshape index column to 2D array for .fit() method
X_train = np.array(train.index).reshape(-1, 1)
y_train = train['Close']
```

In [50]:

```
# Create LinearRegression Object
model = LinearRegression()
# Fit linear model using the train data set
model.fit(X_train, y_train)
```

Out[50]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

Model Evaluation

In [51]:

```
# The coefficient
print('Slope: ', np.asscalar(np.squeeze(model.coef_)))
# The Intercept
print('Intercept: ', model.intercept_)
```

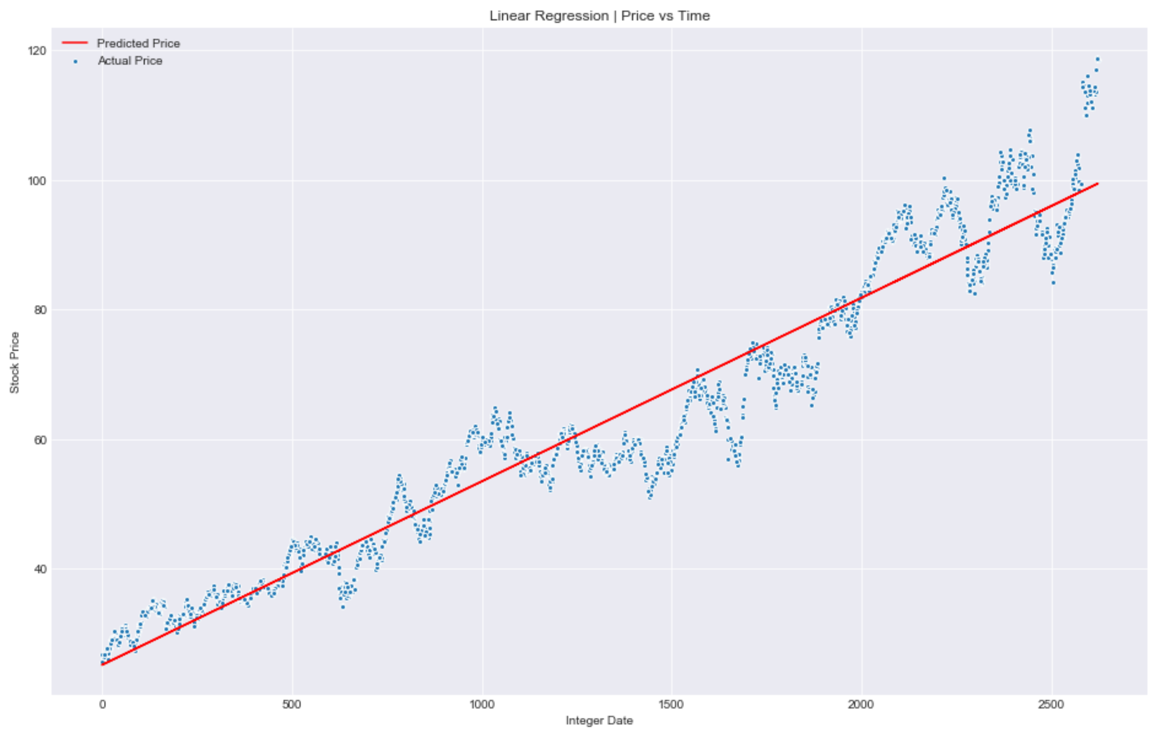
```
Slope:  0.028327707465384017
Intercept:  25.156203076940393
```

Interpreting the coefficients:

- The **slope** coefficient tells us that with a 1 unit increase in **date** the **closing price** increases by 0.0276 $
- The **intercept** coefficient is the price at wich the **closing price** measurement started, the stock price value at date zero

In [52]:

```
# Train set graph
plt.figure(1, figsize=(16,10))
plt.title('Linear Regression | Price vs Time')
plt.scatter(X_train, y_train, edgecolor='w', label='Actual Price')
plt.plot(X_train, model.predict(X_train), color='r', label='Predicted Price')
plt.xlabel('Integer Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Prediction from our Model

In [53]:

```
# Create test arrays
X_test = np.array(test.index).reshape(-1, 1)
y_test = test['Close']
```

In [54]:

```
# Generate array with predicted values
y_pred = model.predict(X_test)
```

Regression Evaluation

Let's have a look at how the predicted values compare with the actual value on random sample from our data set.

In [55]:

```
# Get number of rows in data set for random sample
df.shape
```

Out[55]:

```
(2623, 2)
```

In [56]:

```
# Generate 25 random numbers
randints = np.random.randint(2550, size=25)

# Select row numbers == random numbers
df_sample = df[df.index.isin(randints)]
```

In [57]:

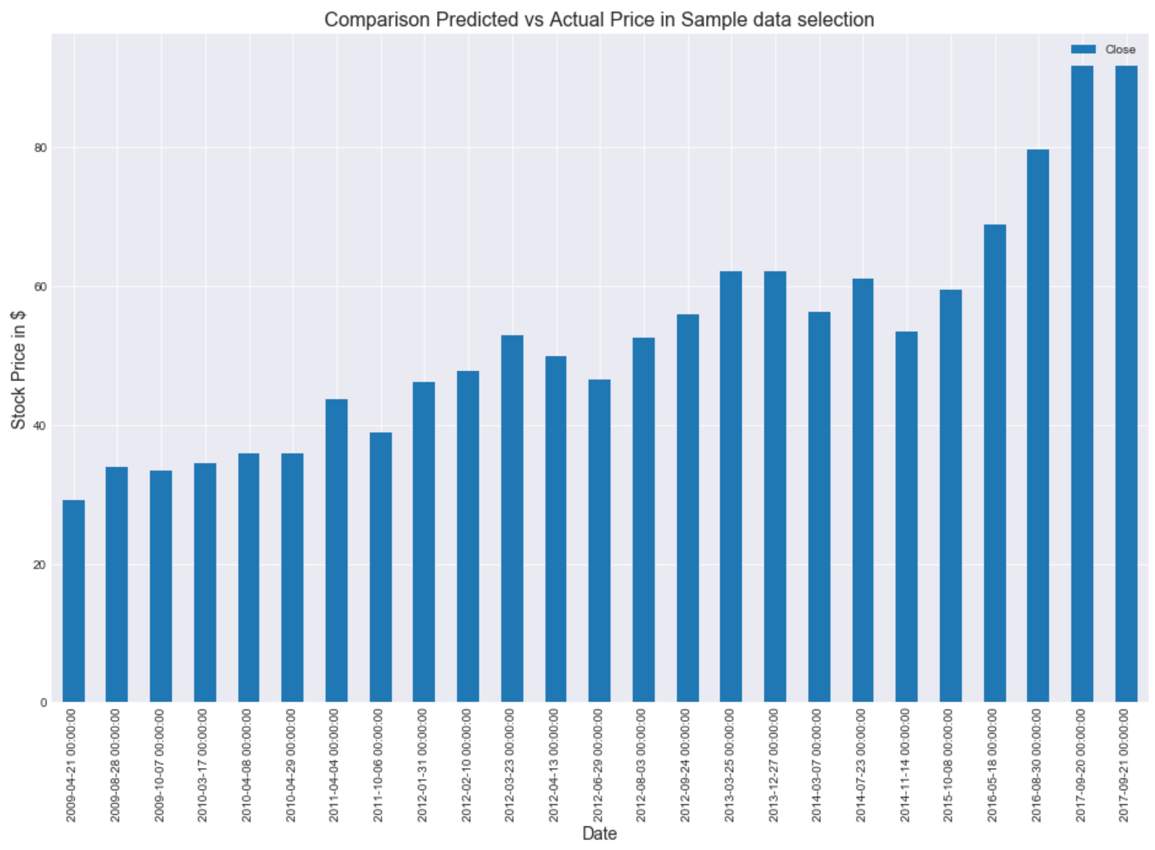
```
df_sample.head()
```

Out[57]:

	Date	Close
31	2009-04-21	29.23
124	2009-08-28	34.02
152	2009-10-07	33.40
267	2010-03-17	34.50
281	2010-04-08	35.94

In [71]:

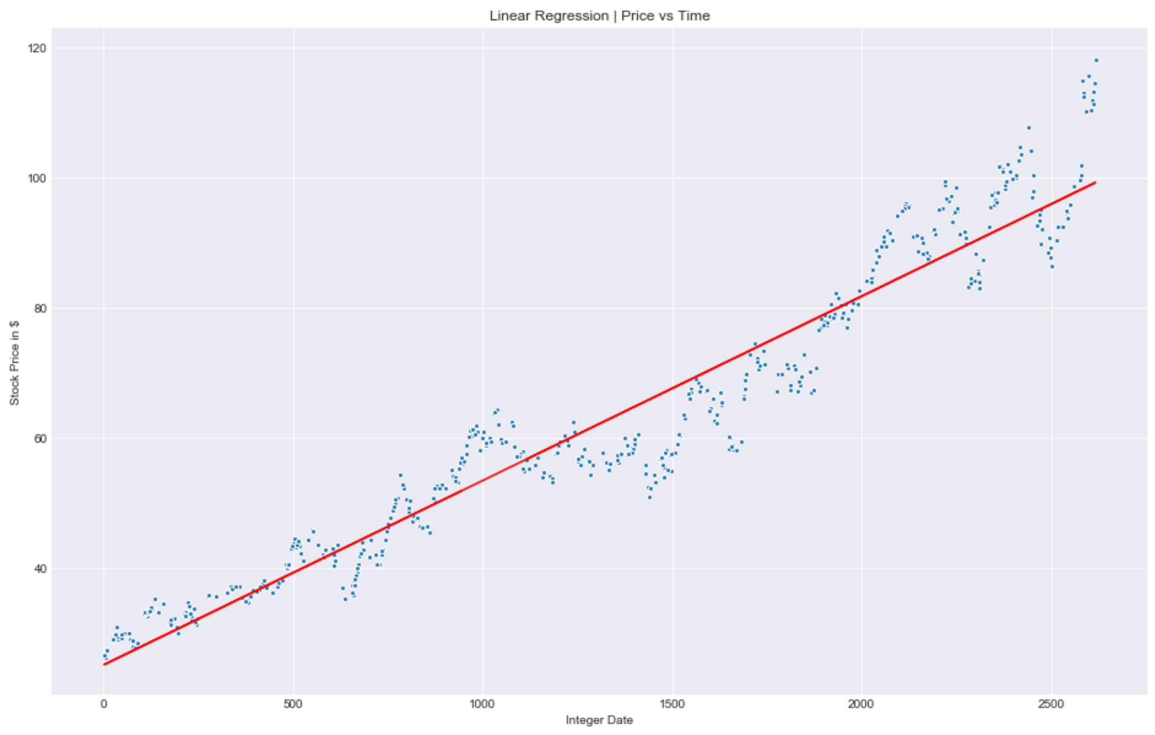
```
# Create subplots to plot graph and control axes
fig, ax = plt.subplots()

df_sample.plot(x='Date', y=['Close'], kind='bar', ax=ax)

# Set figure title
plt.title('Comparison Predicted vs Actual Price in Sample data selection', fontsize=16)

# Set x label
plt.xlabel('Date', fontsize=14)

# Set y label
plt.ylabel('Stock Price in $', fontsize=14)

# Show plot
plt.show()
```



We can see some larger variations between predicted and actual values in the random sample. Let's see how the model performed over the whole test data set.

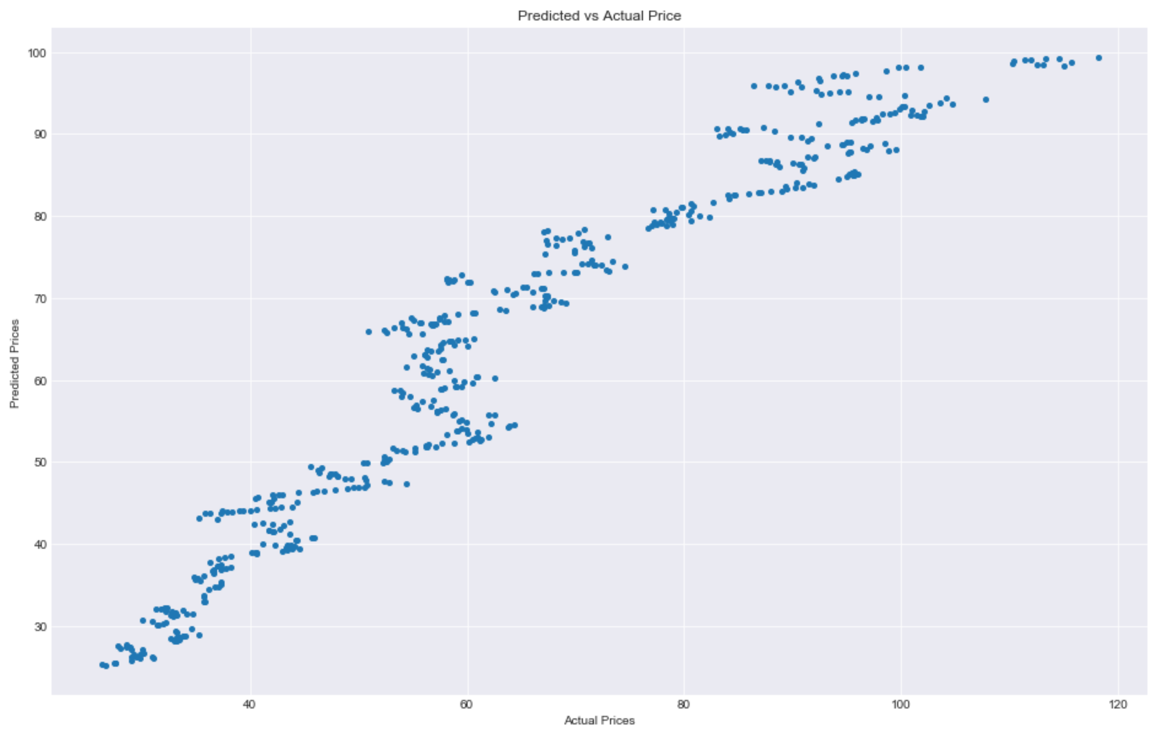
In [59]:

```
# Plot fitted line, y test
plt.figure(1, figsize=(16,10))
plt.title('Linear Regression | Price vs Time')
plt.plot(X_test, model.predict(X_test), color='r', label='Predicted Price')
plt.scatter(X_test, y_test, edgecolor='w', label='Actual Price')

plt.xlabel('Integer Date')
plt.ylabel('Stock Price in $')

plt.show()
```



In [60]:

```
# Plot predicted vs actual prices
plt.scatter(y_test, y_pred)

plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')

plt.title('Predicted vs Actual Price')

plt.show()
```



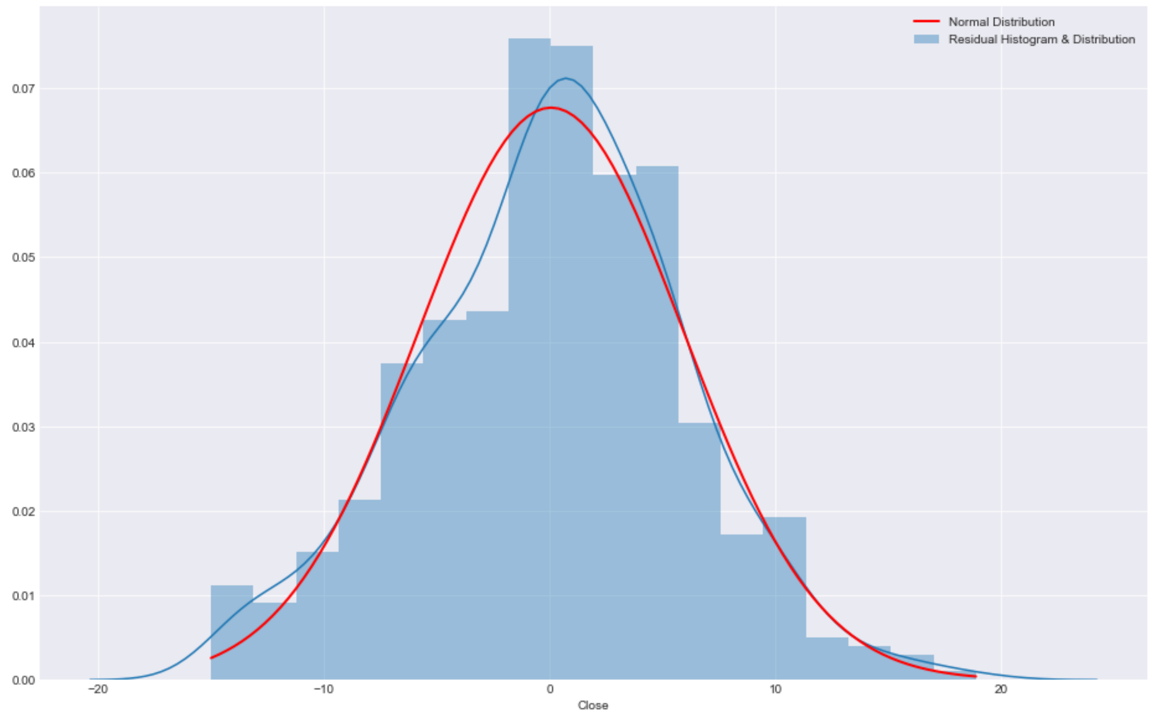
The data points are mostly close to a diagonal, which indicates, that the predicted values are close to the actual value and the model's performance is largerly quite good.
Yet there are some areas, around 55 to 65, the model seems to be quite random and shows no relationship between the predicted and actual value.
Also in the area around 85 - 110 the data point are spread out quite heavily and the predictions don't cover the values above 100.

Residual Histogram

The residuals are nearly normally distributed around zero, with a slight skewedness to the right.

In [61]:

```
# Import norm package to plot normal distribution
from scipy.stats import norm

# Fit a normal distribution to the data:
mu, std = norm.fit(y_test - y_pred)

ax = sns.distplot((y_test - y_pred), label='Residual Histogram & Distribution')

# Calculate the pdf over a range of values
x = np.linspace(min(y_test - y_pred), max(y_test - y_pred), 100)
p = norm.pdf(x, mu, std)

# And plot on the same axes that seaborn put the histogram
ax.plot(x, p, 'r', lw=2, label='Normal Distribution')

plt.legend()
plt.show()
```



In [62]:

```
# Add new column for predictions to df
df['Prediction'] = model.predict(np.array(df.index).reshape(-1, 1))
```

In [63]:

```
df.head()
```

Out[63]:

	Date	Close	Prediction
0	2009-03-09	25.59	25.156203
1	2009-03-10	26.87	25.184531
2	2009-03-11	26.64	25.212858
3	2009-03-12	26.18	25.241186
4	2009-03-13	25.73	25.269514

Error Evaluation Metrics

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

All of these are **cost functions** we want to minimize.

In [64]:

```
# Import metrics package from sklearn for statistical analysis
from sklearn import metrics
```

In [65]:

```
# Statistical summary of test data
df['Close'].describe()
```

Out[65]:

```
count    2623.000000
mean       62.305957
std        22.227715
min        25.590000
25%        43.340000
50%        58.430000
75%        80.405000
max       118.820000
Name: Close, dtype: float64
```

In [66]:

```
# Calculate and print values of MAE, MSE, RMSE
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4.622975319138565
Mean Squared Error: 34.73723135215753
Root Mean Squared Error: 5.893829939195525
```

- The MAE is 3% (of minimum) and 6% (of maximum) of the Closing Price.
- The other two errors are larger, because the errors are squared and have therefore a greater influence on the result.

Accuracy Evaluation Metrics

To see how accurate our model is, we can calculate the **Coefficient of determination**, which describes the ratio between the total error and the error, that is explained by our model. It's value is between 0 and 1, with 1 meaning 100% of the error is acoounted for by the model.

Coefficient of determination

$$R^2 = 1 - \frac{RSS}{TSS}$$

with

Residual Sum of Squares (RSS)

$$RSS = \sum_{i=1}^{N} \epsilon_i^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Total Sum of Squares (TSS)

$$TSS = \sum_{i=1}^{N} (y_i - \bar{y}_i)^2$$

In [67]:

```
print('R2: ', metrics.r2_score(y_test, y_pred))
```

```
R2:  0.931518054893835
```

In [68]:

```
from sklearn.metrics import explained_variance_score
explained_variance_score(y_test, y_pred)
```

Out[68]:

```
0.9315252948228882
```

The value of R^2 shows that are model accounts for nearly 94% of the differences between the actual stock prices and the predicted prices.

In [ ]: