In [2]:
```
import datetime
import numpy as np
import pandas as pd
%matplotlib inline
from matplotlib import pyplot as plt
import pandas_datareader.data as web
import fix_yahoo_finance as yf
import seaborn as sns
```

In [3]:
```
#download stock price for HMC, RACE, TM, GM, F from 1 Jan 2017 to 1 Oc
start_date = datetime.datetime(2017, 1, 1)
end_date = datetime.datetime(2022, 10, 1)
```

Datasets

Honda Motors(HMC)

In [4]:
```
#download stock prices
hmc = yf.download('HMC', start_date, end_date)
hmc.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[4]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03 00:00:00-05:00	29.480000	29.610001	29.420000	29.610001	25.722599	864500
2017-01-04 00:00:00-05:00	30.209999	30.670000	30.209999	30.660000	26.634747	705500
2017-01-05 00:00:00-05:00	30.620001	30.780001	30.580000	30.660000	26.634747	482600
2017-01-06 00:00:00-05:00	30.350000	30.580000	30.240000	30.469999	26.469692	493600
2017-01-09 00:00:00-05:00	30.370001	30.500000	30.299999	30.430000	26.434942	585200

In [5]:
```
#add a column called 'Log_return' which is the log of the price close
hmc['Log_return'] = np.log(hmc['Close']/hmc['Open'])
hmc.reset_index(inplace = True)
hmc.head()
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume	Log_return
0	2017-01-03 00:00:00-05:00	29.480000	29.610001	29.420000	29.610001	25.722599	864500	0.004400
1	2017-01-04 00:00:00-05:00	30.209999	30.670000	30.209999	30.660000	26.634747	705500	0.014786
2	2017-01-05 00:00:00-05:00	30.620001	30.780001	30.580000	30.660000	26.634747	482600	0.001305
3	2017-01-06 00:00:00-05:00	30.350000	30.580000	30.240000	30.469999	26.469692	493600	0.003946
4	2017-01-09 00:00:00-05:00	30.370001	30.500000	30.299999	30.430000	26.434942	585200	0.001974

Toyota Motors(TM)

In [6]:
```
#same passages of HMC dataset done here
tm = yf.download('TM', start_date, end_date)
tm.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[6]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2017-01-03 00:00:00-05:00	118.169998	118.669998	117.830002	118.550003	118.550003	204000
2017-01-04 00:00:00-05:00	120.269997	121.290001	120.139999	121.190002	121.190002	250600
2017-01-05 00:00:00-05:00	121.190002	121.389999	120.320000	120.440002	120.440002	525900
2017-01-06 00:00:00-05:00	119.839996	120.230003	119.410004	120.129997	120.129997	171600
2017-01-09 00:00:00-05:00	119.480003	119.959999	119.470001	119.739998	119.739998	135800

In [7]:
```python
#same passages of HMC dataset done here
tm['Log_return'] = np.log(tm['Close']/tm['Open'])
tm.reset_index(inplace = True)
tm.head()
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume	Log_return
0	2017-01-03 00:00:00-05:00	118.169998	118.669998	117.830002	118.550003	118.550003	204000	0.003211
1	2017-01-04 00:00:00-05:00	120.269997	121.290001	120.139999	121.190002	121.190002	250600	0.007620
2	2017-01-05 00:00:00-05:00	121.190002	121.389999	120.320000	120.440002	120.440002	525900	-0.006208
3	2017-01-06 00:00:00-05:00	119.839996	120.230003	119.410004	120.129997	120.129997	171600	0.002417
4	2017-01-09 00:00:00-05:00	119.480003	119.959999	119.470001	119.739998	119.739998	135800	0.002174

Ferrari(RACE)

In [8]:
```python
#same passages of HMC dataset done here
race = yf.download('RACE', start_date, end_date)
race.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[8]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03 00:00:00-05:00	59.160000	59.259998	58.349998	58.939999	55.978436	546700
2017-01-04 00:00:00-05:00	58.840000	59.480000	58.790001	59.410000	56.424824	373000
2017-01-05 00:00:00-05:00	59.439999	59.880001	59.341000	59.360001	56.377335	304800
2017-01-06 00:00:00-05:00	58.970001	59.160000	58.810001	58.939999	55.978436	280500
2017-01-09 00:00:00-05:00	57.770000	58.500000	57.560001	58.279999	55.351601	409300

In [9]:
```
#same passages of HMC dataset done here
race['Log_return'] = np.log(race['Close']/race['Open'])
race.reset_index(inplace = True)
race.head()
```

Out[9]:

	Date	Open	High	Low	Close	Adj Close	Volume	Log_return
0	2017-01-03 00:00:00-05:00	59.160000	59.259998	58.349998	58.939999	55.978436	546700	-0.003726
1	2017-01-04 00:00:00-05:00	58.840000	59.480000	58.790001	59.410000	56.424824	373000	0.009641
2	2017-01-05 00:00:00-05:00	59.439999	59.880001	59.341000	59.360001	56.377335	304800	-0.001347
3	2017-01-06 00:00:00-05:00	58.970001	59.160000	58.810001	58.939999	55.978436	280500	-0.000509
4	2017-01-09 00:00:00-05:00	57.770000	58.500000	57.560001	58.279999	55.351601	409300	0.008789

General Motors (GM)

In [10]:
```
#same passages of HMC dataset done here
gm = yf.download('GM', start_date, end_date)
gm.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[10]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2017-01-03 00:00:00-05:00	34.980000	35.570000	34.840000	35.150002	30.369265	10904900
2017-01-04 00:00:00-05:00	35.599998	37.240002	35.470001	37.090000	32.045403	23388500
2017-01-05 00:00:00-05:00	37.009998	37.049999	36.070000	36.389999	31.440619	15636700
2017-01-06 00:00:00-05:00	36.410000	36.549999	35.930000	35.990002	31.095022	13240100
2017-01-09 00:00:00-05:00	36.119999	36.529999	35.860001	36.009998	31.112295	15204500

In [11]:

```
#same passages of HMC dataset done here
gm['Log_return'] = np.log(gm['Close']/gm['Open'])
gm.reset_index(inplace = True)
gm.head()
```

Out[11]:

	Date	Open	High	Low	Close	Adj Close	Volume	Log_return
0	2017-01-03 00:00:00-05:00	34.980000	35.570000	34.840000	35.150002	30.369265	10904900	0.004848
1	2017-01-04 00:00:00-05:00	35.599998	37.240002	35.470001	37.090000	32.045403	23388500	0.041002
2	2017-01-05 00:00:00-05:00	37.009998	37.049999	36.070000	36.389999	31.440619	15636700	-0.016894
3	2017-01-06 00:00:00-05:00	36.410000	36.549999	35.930000	35.990002	31.095022	13240100	-0.011602
4	2017-01-09 00:00:00-05:00	36.119999	36.529999	35.860001	36.009998	31.112295	15204500	-0.003050

Ford Motors(F)

In [12]:

```
#same passages of HMC dataset done here
f = yf.download('F', start_date, end_date)
f.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[12]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2017-01-03 00:00:00-05:00	12.20	12.60	12.13	12.59	8.943367	40510800
2017-01-04 00:00:00-05:00	12.77	13.27	12.74	13.17	9.355372	77638100
2017-01-05 00:00:00-05:00	13.21	13.22	12.63	12.77	9.071233	75628400
2017-01-06 00:00:00-05:00	12.80	12.84	12.64	12.76	9.064129	40315900
2017-01-09 00:00:00-05:00	12.79	12.86	12.63	12.63	8.971783	39438400

In [13]:
```
#same passages of HMC dataset done here
f['Log_return'] = np.log(f['Close']/f['Open'])
f.reset_index(inplace = True)
f.head()
```

Out[13]:

	Date	Open	High	Low	Close	Adj Close	Volume	Log_return
0	2017-01-03 00:00:00-05:00	12.20	12.60	12.13	12.59	8.943367	40510800	0.031467
1	2017-01-04 00:00:00-05:00	12.77	13.27	12.74	13.17	9.355372	77638100	0.030843
2	2017-01-05 00:00:00-05:00	13.21	13.22	12.63	12.77	9.071233	75628400	-0.033875
3	2017-01-06 00:00:00-05:00	12.80	12.84	12.64	12.76	9.064129	40315900	-0.003130
4	2017-01-09 00:00:00-05:00	12.79	12.86	12.63	12.63	8.971783	39438400	-0.012589

Functions

This section contains all the methods used to write the report and that animates the Dashboard in the next section.

All the methods will be called by the Dashboard

Mean and Standard Deviation Log Returns

In [14]:
```
#calculat mean and standard deviation of the Log_return column for eac
print('-'*30)
print("HMC:")
stats = hmc['Log_return'].agg(['mean', 'std'])
print(stats)
print('-'*30)
print("TM:")
stats = tm['Log_return'].agg(['mean', 'std'])
print(stats)
print('-'*30)
print("RACE:")
stats = race['Log_return'].agg(['mean', 'std'])
print(stats)
print('-'*30)
print("GM:")
stats = gm['Log_return'].agg(['mean', 'std'])
print(stats)
print('-'*30)
print("F:")
stats = f['Log_return'].agg(['mean', 'std'])
print(stats)
print('-'*30)
```

```
------------------------------
HMC:
mean   -0.000436
std     0.007819
Name: Log_return, dtype: float64
------------------------------
TM:
mean   -0.000441
std     0.007016
Name: Log_return, dtype: float64
------------------------------
RACE:
mean    0.000022
std     0.012751
Name: Log_return, dtype: float64
------------------------------
GM:
mean   -0.000860
std     0.018137
Name: Log_return, dtype: float64
------------------------------
F:
mean   -0.001121
std     0.018446
Name: Log_return, dtype: float64
------------------------------
```

Confidence Intervals

In [15]:
```
import math
import statistics
from scipy.stats import norm

def create_interval_mean(alpha, dataset, Type):
    #this function receives an alpha, a dataset of a stock and a Type
    #and gives back the confidence interval for the mean of a particul
    sample_mean = dataset.mean()
    percentile='' #it refers to alpha percentile
    n = len(dataset)

    #3 cases must be analyzed, because the computatation of the alpha

    if Type == 'Two sided':
        percentile = (100 - alpha/2)/100
        little_t = norm.ppf(percentile) #alpha percentile;
        #theoretically we should have used the t-student distribution,
        #so using a normal distribution, the computation will be more
        s = math.sqrt(statistics.variance(dataset)) #standard deviatic

        return (round(sample_mean - little_t*s/math.sqrt(n), 10), rour

    if Type == 'Lower one sided':
        #in this case we want an upper bound for the mean
        percentile = (100 - alpha)/100
        little_t = norm.ppf(percentile)
        s = math.sqrt(statistics.variance(dataset))
        return ('-∞', round(sample_mean + little_t*s/math.sqrt(n),10))

    if Type == 'Upper one sided':
        #in this case we want a lower bound for the mean
        percentile = alpha/100
        little_t = norm.ppf(percentile)
        s = math.sqrt(statistics.variance(dataset))
        #print(f'little t: {little_t}')
        return (round(sample_mean + little_t*s/math.sqrt(n), 10), '+∞'

    return ('','') #we will never be here
```

In [16]:

```
from scipy.stats import chi2

#this function receives an alpha, a dataset of a stock and a Type of i
#and gives back the confidence interval for the variance of a particul

def create_interval_variance(alpha, dataset, Type):
    n = len(dataset)
    df=n-1 #degree of freedom
    s_squared = statistics.variance(dataset) #estimation sample varian

    if Type == 'Two sided':
        percentile_low = (alpha/2)/100
        percentile_high = (100 - alpha/2)/100
        p_lower = chi2.ppf(percentile_low, df)
        p_higher = chi2.ppf(percentile_high, df)
        return (round(df*s_squared/p_higher, 10), round(df*s_squared/p

    if Type == 'Lower one sided':
        #in this case we want an upper bound for the variance
        percentile_high = (100 - alpha)/100
        p_higher = chi2.ppf(percentile_high, df)
        return ('-∞', round(df*s_squared/p_higher, 10)) #confidence in

    if Type == 'Upper one sided':
        #in this case we want a lower bound for the variance
        percentile_low = alpha/100
        p_lower = chi2.ppf(percentile_low, df)
        return (round(df*s_squared/p_lower, 10), '+∞')

    return ('','') #we will never be here
```

In [17]:
```
import math
import statistics
from scipy.stats import t
from scipy.stats import chi2

#this function calls in a proper manner the create_interval_mean(alpha
#create_interval_variance(alpha, dataset, Type) functions, and properl

def confidence_intervals(Ticker, Confidence_level, Type):
    alpha = 100 - Confidence_level
    if alpha == 0:
        s1 = f"{Ticker} mean interval at {Confidence_level}% level of
        s2 = f"{Ticker} variance interval at {Confidence_level}% level
        return s1, s2

    if Ticker == 'GM':
        dataset = gm['Log_return']

    if Ticker == 'F':
        dataset = f['Log_return']

    if Ticker == 'HMC':
        dataset = hmc['Log_return']

    if Ticker == 'TM':
        dataset = tm['Log_return']

    if Ticker == 'RACE':
        dataset = race['Log_return']

    mean_interval = create_interval_mean(alpha, dataset, Type)
    var_interval = create_interval_variance(alpha, dataset, Type)
    s1 = f"{Ticker} mean interval at {Confidence_level}% level of conf
    s2 = f"{Ticker} variance interval at {Confidence_level}% level of
    return s1, s2
```

Equal mean Test

In [18]:
```
from scipy.stats import norm

#this function tests the h0 hypotesis that mean sample 1 is equal to m
#it receives:
# - dataset sample 1
# - dataset sample 2
# - an alpha
# - h1: mu1!=mu2 or mu1>mu2 or mu1<mu2

def test_equal_mean(dataset1, dataset2, alpha, h1):
    #calculate the sample means
    mean1 = dataset1.mean()
```

```python
    mean2 = dataset2.mean()

    #estiate the variances
    s1_squared = statistics.variance(dataset1)
    s2_squared = statistics.variance(dataset2)

    #compute the lenght of each sample
    n = len(dataset1)
    m = len(dataset2)

    #calculate the t-statistic --> no assumption of equal variance
    t_stat = (mean1 - mean2)/math.sqrt(s1_squared/n + s2_squared/m)

    #different cases if test bilateral or unilateral
    if h1 == 'mu1 != mu2':
        p_value = norm.sf(abs(t_stat))*2
        perc = norm.ppf((alpha/2)/100) #little z percentile, n and m v
        p_value_perc = norm.sf(abs(perc))*2
        if p_value<p_value_perc: #Rejection Region
            return [False, t_stat, perc, -perc, p_value] #we return al
        else:
            return [True, t_stat, perc, -perc, p_value] #Acceptance re

    if h1 == 'mu1 > mu2':
        p_value = norm.sf(t_stat)
        perc = -norm.ppf(alpha/100) #in this case we don't divide alph
        p_value_perc = norm.sf(abs(perc))
        if p_value < p_value_perc: #if the p value of our test statist
            return [False, t_stat, perc, perc, p_value]
        else: #else we can NOT reject h0
            return [True, t_stat, perc, perc, p_value]

    if h1 == 'mu1 < mu2':
        p_value = 1 - norm.sf(t_stat)
        perc = norm.ppf(alpha/100)
        p_value_perc = norm.sf(abs(perc))
        if p_value<p_value_perc:
            return [False, t_stat, perc, perc, p_value]
        else:
            return [True, t_stat, perc, perc, p_value]

    return False #we will never be here
```

In [19]:

```python
#this function calls in a proper manner the test_equal_mean(dataset1,
#and properly format the output, creating some graphs

def mean_test(Ticker1, Ticker2, h1, Confidence_level):
    alpha = 100-Confidence_level
    if Ticker1 == 'GM':
        dataset1 = gm['Log_return']

    if Ticker1 == 'F':
        dataset1 = f['Log_return']
```

```
    if Ticker1 == 'HMC':
        dataset1 = hmc['Log_return']

    if Ticker1 == 'TM':
        dataset1 = tm['Log_return']

    if Ticker1 == 'RACE':
        dataset1 = race['Log_return']

    if Ticker2 == 'GM':
        dataset2 = gm['Log_return']

    if Ticker2 == 'F':
        dataset2 = f['Log_return']

    if Ticker2 == 'HMC':
        dataset2 = hmc['Log_return']

    if Ticker2 == 'TM':
        dataset2 = tm['Log_return']

    if Ticker2 == 'RACE':
        dataset2 = race['Log_return']

    result = test_equal_mean(dataset1, dataset2, alpha, h1)

    x_axis = np.arange(-3, 3, 0.01)
    plt.plot(x_axis, norm.pdf(x_axis, 0, 1))
    f1 = plt.plot(result[1], 0, linewidth=0, marker='o', color='b', la
    f2 = plt.plot(result[2], 0, linewidth=0, marker='x', color='r', la
    f3 = plt.plot(result[3], 0, linewidth=0, marker='x', color='r')
    s1 = "P-value: " + str(round(result[4]*100,2)) +"%" #test statisti
    sns.despine()
    plt.legend()
    if result[0]: #if we cannot reject h0
        s2 = "At "+ str(Confidence_level)+"% of confidence, we cannot
        return s1, s2
    else: #if we can reject h0
        s2 = "At "+ str(Confidence_level)+"% of confidence, we reject
        return s1, s2
```

Regression between Two Log Returns

In [20]:

```
from scipy import stats

#this function receives 2 tickers and create a log return linear regre
# and the dependent one is Y_Ticker
```

```
def log_return_regression(X_Ticker, Y_Ticker):
    x=''
    y=''
    if X_Ticker == 'GM':
        x = gm['Log_return']

    if X_Ticker == 'F':
        x = f['Log_return']

    if X_Ticker == 'HMC':
        x = hmc['Log_return']

    if X_Ticker == 'TM':
        x = tm['Log_return']

    if X_Ticker == 'RACE':
        x = race['Log_return']

    if Y_Ticker == 'GM':
        y = gm['Log_return']

    if Y_Ticker == 'F':
        y = f['Log_return']

    if Y_Ticker == 'HMC':
        y = hmc['Log_return']

    if Y_Ticker == 'TM':
        y = tm['Log_return']

    if Y_Ticker == 'RACE':
        y = race['Log_return']

    slope, intercept, r, p, std_err = stats.linregress(x, y)

    def myfunc(x):
      return slope * x + intercept

    #draw the graph


    mymodel = list(map(myfunc, x))
    plt.figure(figsize=(10,5))
    plt.scatter(x, y, s=15)
    plt.plot(x, mymodel, color = 'red')
    x0, xmax = plt.xlim()
    y0, ymax = plt.ylim()
    data_width = xmax - x0
    data_height = ymax - y0
    title = f'Linear regression of {X_Ticker} log returns (X variable)
    plt.title(title, fontsize=20)
    plt.ylabel(f'{Y_Ticker} log returns', fontsize=15)
    plt.xlabel(f'{X_Ticker} log returns', fontsize=15)
```

```
    plt.text(x0 + data_width*1.06, y0 + data_height * 0.5, f'$LogRetur
    plt.text(x0 + data_width * 1.06, y0 + data_height * 0.4, f'$R^2 =
    plt.show()
```

Normal returns and probability plot

In [21]:
```
import statsmodels.graphics.gofplots as sm

#this function receives as input the Ticker and generates 2 graphs:
# - one containing the stock returns histogram and a normal PDF to see
# - a normal probability plot of the stock returns
def generate_graph(Ticker):
    x_values = None
    if Ticker == 'GM':
        x_values = gm['Log_return']

    if Ticker == 'F':
        x_values = f['Log_return']

    if Ticker == 'HMC':
        x_values = hmc['Log_return']

    if Ticker == 'TM':
        x_values = tm['Log_return']

    if Ticker == 'RACE':
        x_values = race['Log_return']

    plt.figure(figsize=(10,5))
    #Plot stock return Histogram
    plt.hist(x_values, bins=100, density=True, alpha=0.6, color='b')

    # Plot the normal PDF
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, x_values.mean(), x_values.std())
    plt.plot(x, p, 'k', linewidth=2)
    title = f"{Ticker} Returns vs Normal Returns"
    plt.title(title)

    #Plot the Normal Probability plot
    plt.figure(figsize=(10,5))
    sm.ProbPlot(x_values).qqplot(line='s')
    title=f"{Ticker} Normal Probability plot"
    plt.title(title)
    sns.despine()
```

Linear regression on time

In [22]:
```
import statsmodels.formula.api as smf

#this function creates the regression of a Ticker over time, to try tc
#stock return and if it can be used as a prediction of future returns
#it also plots the linear regression
def create_regression(Ticker):
    if Ticker == 'GM':
        dataset = gm.copy()

    if Ticker == 'F':
        dataset = f.copy()

    if Ticker == 'HMC':
        dataset = hmc.copy()

    if Ticker == 'TM':
        dataset = tm.copy()

    if Ticker == 'RACE':
        dataset = race.copy()

    dataset['date'] = np.arange(len(dataset))
    reg_dataset = smf.ols('Log_return ~ date', data=dataset) #ordinary
    reg_result_dataset = reg_dataset.fit()
    plt.figure(figsize=(14, 8))
    plt.plot(dataset['Date'], dataset.Log_return, linewidth=0, marker=
    x_points = dataset['Date']
    b0_dataset = reg_result_dataset.params[0]
    b1_dataset = reg_result_dataset.params[1]
    y_points = b0_dataset + b1_dataset*dataset['date'] #regression lir
    plt.plot(x_points, y_points, linewidth=3)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    x0, xmax = plt.xlim()
    y0, ymax = plt.ylim()
    data_width = xmax - x0
    data_height = ymax - y0
    plt.text(x0 + data_width*1.06, y0 + data_height * 0.5,f'$y = {rour
    plt.text(x0 + data_width*1.06, y0 + data_height * 0.4, f'$R^2 = {r
    plt.xlabel('Date', fontsize=20)
    plt.ylabel('Log_Return', fontsize=20)
    sns.despine()
    reg_result_dataset.summary()
```

Dashboard

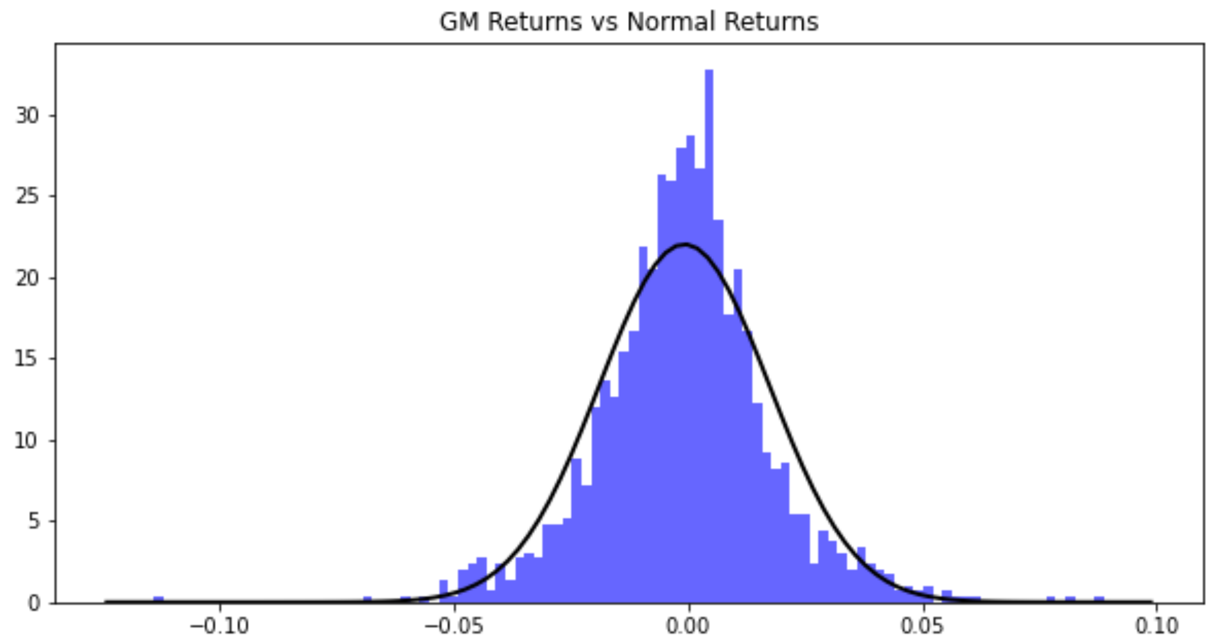
Normal returns and probability plot

In [23]:
```python
from ipywidgets import interact, FloatSlider
```

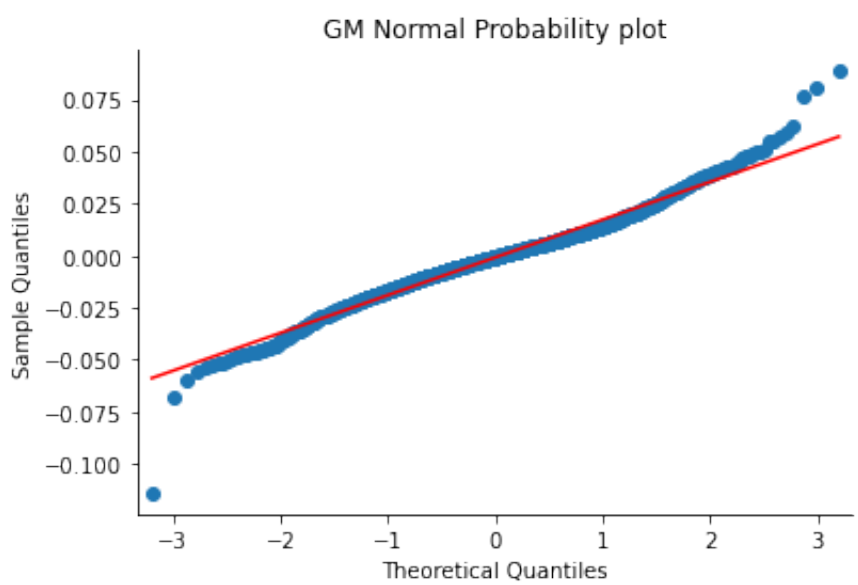
Choose a ticker to see if its log return over time can be aproximated by a normal and to see its normal probability plot

In [24]:
```python
interact(generate_graph, Ticker = ['GM','F','HMC','TM', 'RACE'])
```

Ticker GM




```
<Figure size 720x360 with 0 Axes>
```



Out[24]:
```
<function __main__.generate_graph(Ticker)>
```

Confidence Intervals

Choose a Ticker to see its mean and variance intervals given a confidence level

In [25]:
```
interact(confidence_intervals, Ticker = ['GM','F','HMC','TM', 'RACE'],
```

Ticker GM

Confidence... 90.00

Type Two sided

```
('GM mean interval at 90.0% level of confidence: [-0.0016443988 --
-7.58523e-05]',
 'GM variance interval at 90.0% level of confidence: [0.0003097742
-- 0.0003501023]')
```

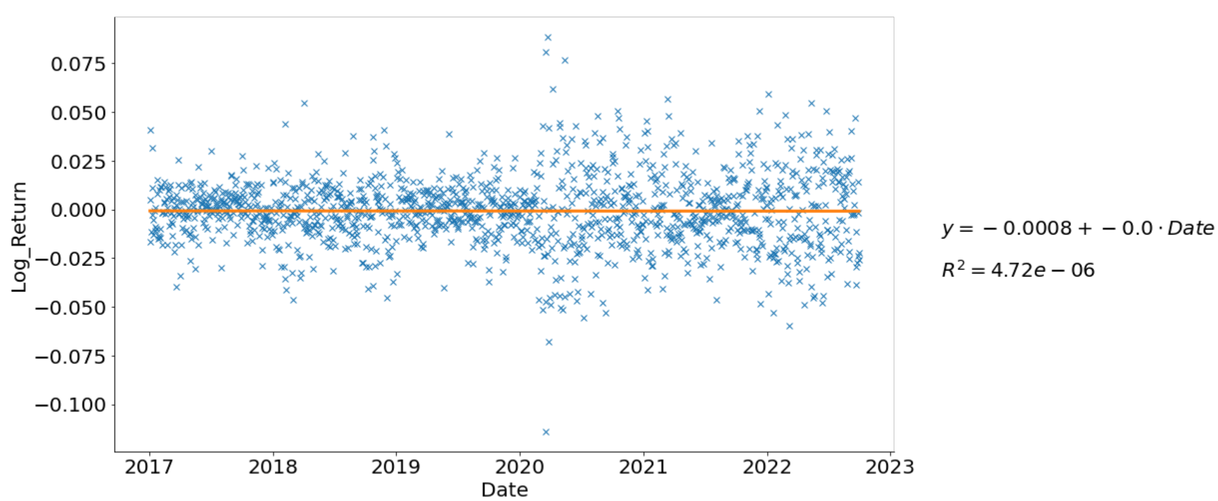
Out[25]:
```
<function __main__.confidence_intervals(Ticker, Confidence_level, Typ
e)>
```

Linear Regression over time

Choose a Ticker to run a linear regression of its log returns over time

In [26]:
```
interact(create_regression, Ticker = ['GM','F','HMC','TM', 'RACE'])
```

Ticker GM



Out[26]:
```
<function __main__.create_regression(Ticker)>
```

Equal mean Test

Choose two tickers and a proper alternative hypostesis h1 versus h0: mu1=mu2

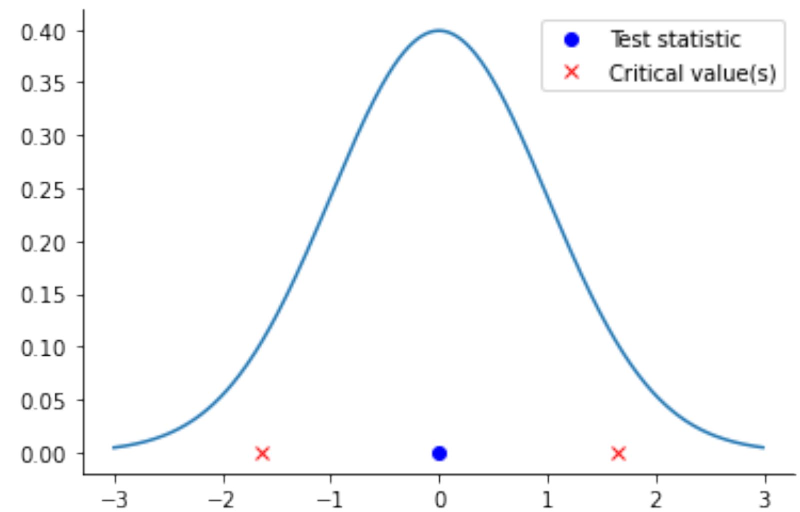
In [27]:
```
interact(mean_test, Ticker1 = ['GM','F','HMC','TM', 'RACE'], Ticker2 =
```

Ticker1 GM

Ticker2 GM

h1 mu1 != mu2

Confidence... 90.00




```
('P-value: 100.0%',
 'At 90.0% of confidence, we cannot reject the null hypotesis h0 th
at mean of GM is equal to mean of GM')
```

Out[27]:
```
<function __main__.mean_test(Ticker1, Ticker2, h1, Confidence_level)>
```

Regression between Two Log Returns

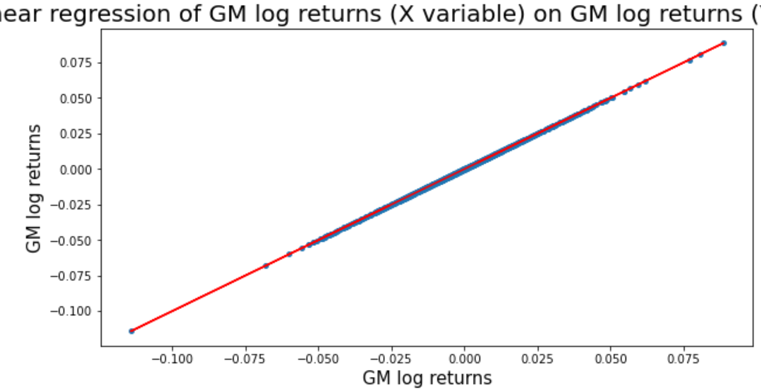
Choose 2 Tickers to run a linear regression of the log returns of one over the other ones

```
In [28]: interact(log_return_regression, X_Ticker = ['GM','F','HMC','TM', 'RACE
```

X_Ticker GM

Y_Ticker GM

Linear regression of GM log returns (X variable) on GM log returns (Y variable)



$LogReturn(GM) = 0.0 + 1.0 \cdot LogReturn(GM)$
$R^2 = 1.0$

```
Out[28]: <function __main__.log_return_regression(X_Ticker, Y_Ticker)>
```