Implementation of Arbitrage Pricing Theory in Python

Modern Portfolio Theory

In 1959, Harry Markowitz pioneered a new theory of portfolio diversification, which emphasized the consideration of risk versus returns for an investor's portfolio[1]. Prior to him, John Burr Williams promoted the belief that an investor must engage in the value maximization of future security returns[2], which Markoitz changed by intriducing the idea that increased risk should discount higher future returns. Indeed, Markowitz argued, the expected returns for a portfolio should be normalized to account for future risk. The major assumption of the Markowitz model was that risk and future expected returns had a negative linear relationship. But, for some investors, the lower-risk, lower-return portfolio was desireable as it would decrease the chances of catastrophc loss.

To choose the best portfolio from a number of possible portfolios, each with different return and risk, two separate decisions are to be made, detailed in the below sections:

1. Determination of a set of efficient portfolios.
2. Selection of the best portfolio out of the efficient set.

A portfolio that gives maximum return for a given risk, or minimum risk for given return is an efficient portfolio. Thus, portfolios are selected as follows:

(a) From the portfolios that have the same return, the investor will prefer the portfolio with lower risk, and [3]

(b) From the portfolios that have the same risk level, an investor will prefer the portfolio with higher rate of return.

As the investor is rational, they would like to have higher return. And as they are risk averse, they want to have lower risk.[3] In figure 1 below, the shaded area PVWP includes all the possible securities an investor can invest in. The efficient portfolios are the ones that lie on the boundary of PQVW. For example, at risk level x2, there are three portfolios S, T, U. But portfolio S is called the efficient portfolio as it has the highest return, y2, compared to T and U. All the portfolios that lie on the boundary of PQVW are efficient portfolios for a given risk level.

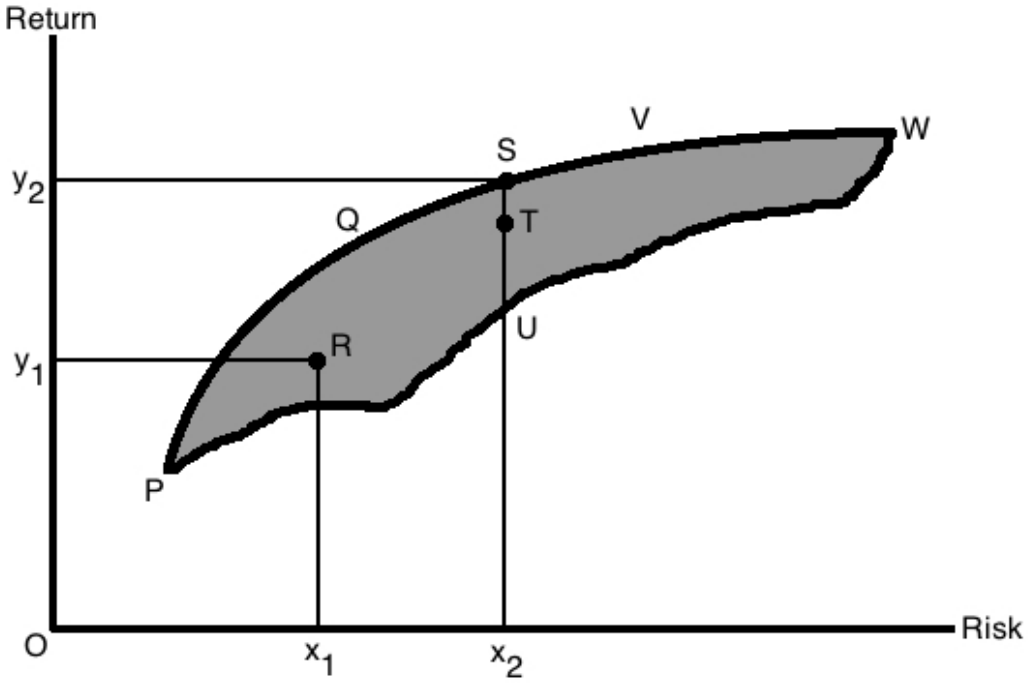


Figure 1. (credit:Parthiv.ravindran on Wikipedia)

The boundary PQVW is called the **Efficient Frontier**. All portfolios that lie below the Efficient Frontier are not good enough because the return would be lower for the given risk. Portfolios that lie to the right of the Efficient Frontier would not be good enough, as there is higher risk for a given rate of return. All portfolios lying on the boundary of PQVW are called Efficient Portfolios. The Efficient Frontier is the same for all investors, as all investors want maximum return with the lowest possible risk and they are risk averse.

Selecting the Optimal Portfolio

For an individual investor to choose the best portfolio, they must consider their indifference cureves with respect to risk and returns. Then, the point at which an indifference curve lies exactly tangent with the efficient frontier is the optimal portfolio for that individual. In the figure below, that is R.

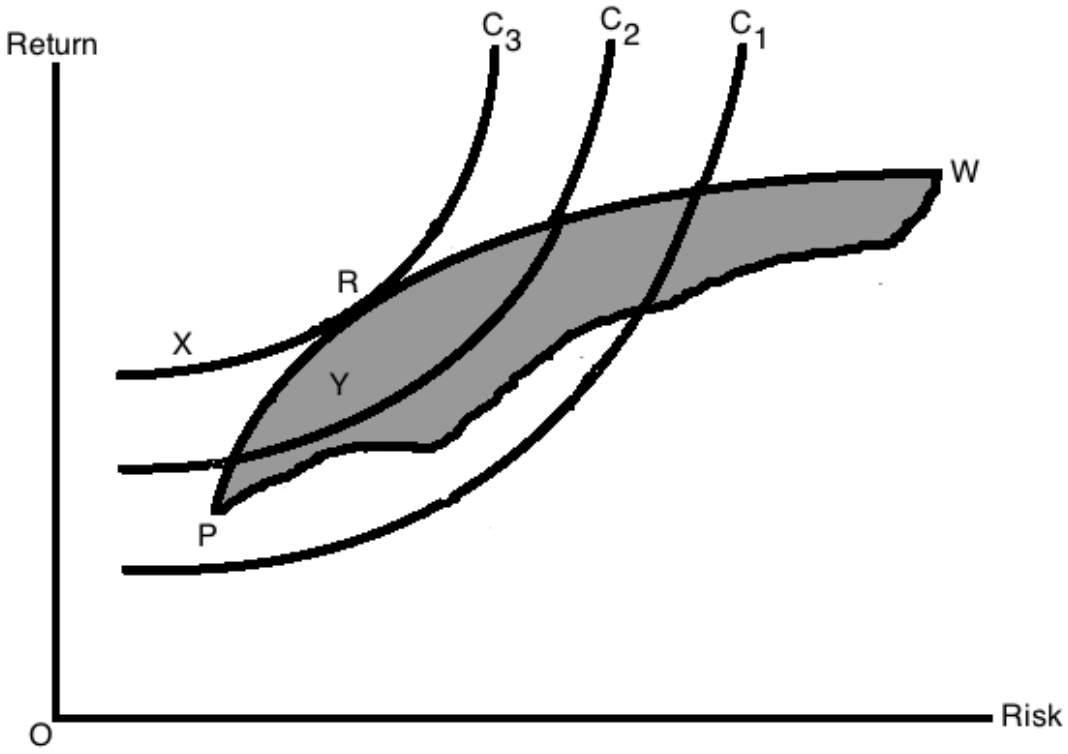


Figure 2. (Credit: Parthiv.ravindran on Wikipedia)

Capital Asset Pricing Model

In the 1960s, MPT was added to concurrently by William Sharpe, John Litner, and Fischer Black[4]. The focus of each of their works was to determine which parts of a securities risk could be eliminated through diversification. In theory, an investor would only have to analyze the aggregate risk changes brought by adding a security to a portfolio. A single addition, in a well-diversified portfolio, would increase risk infinitessimaly. The risk that was unavoidable was called **systemic risk**, which is also called *beta* and is represented by β. β refers to the risk given from the entire stock market, and is essentially unavoidable unless one buys government tresuries. The remaining risk on a given stock is *unsystematic risk*, which is specific to a given company, sector, etc.

Sysytemic Risk

β is unique for each security in a portfolio. It is a weight that describes the sensetivity of that security to market changes. β is generally found using past market data[4]. In fact, a β for a given security i can be found by taking the ratio of the covariance between the returns from that security and the market returns, and the variance of the market returns. This will quantify relative volatility of a security:

$$\beta_i = \frac{Cov(r_i,r_m)}{Var(r_m)}$$

It turns out that beta is just a simple linear regression model comparing the stock market change to an individual stock change. The general calculation for any regression β is :

$$\beta_i = \frac{Cov(x,y)}{Var(x)}$$

The entire market, then, has a β of 1, because the market will always change on a 1-to-1 basis with itself. This makes comparisons to individual securities very easy. A security with a β of 1.5 will respond 50% more drastically to market changes. A 10% change in the market, up or down, would cause a 15% change in that security.

The main insight brought on by CAPM was that taking on increased risk would, in the long run, result in higher expected returns for securities and portfolios alike[4]. Systemic risk could not be eliminated unless an investor accepted only the returns guaranteed from a risk-free security. Unsystematic risk -defined below- could be eliminated by simply having a large number of securities in a portfolio.

The risk that is reduced by diversification is *unsystematic risk*. This would be caused by employee strikes, scandals, big new contracts, discovery of new resources, etc.

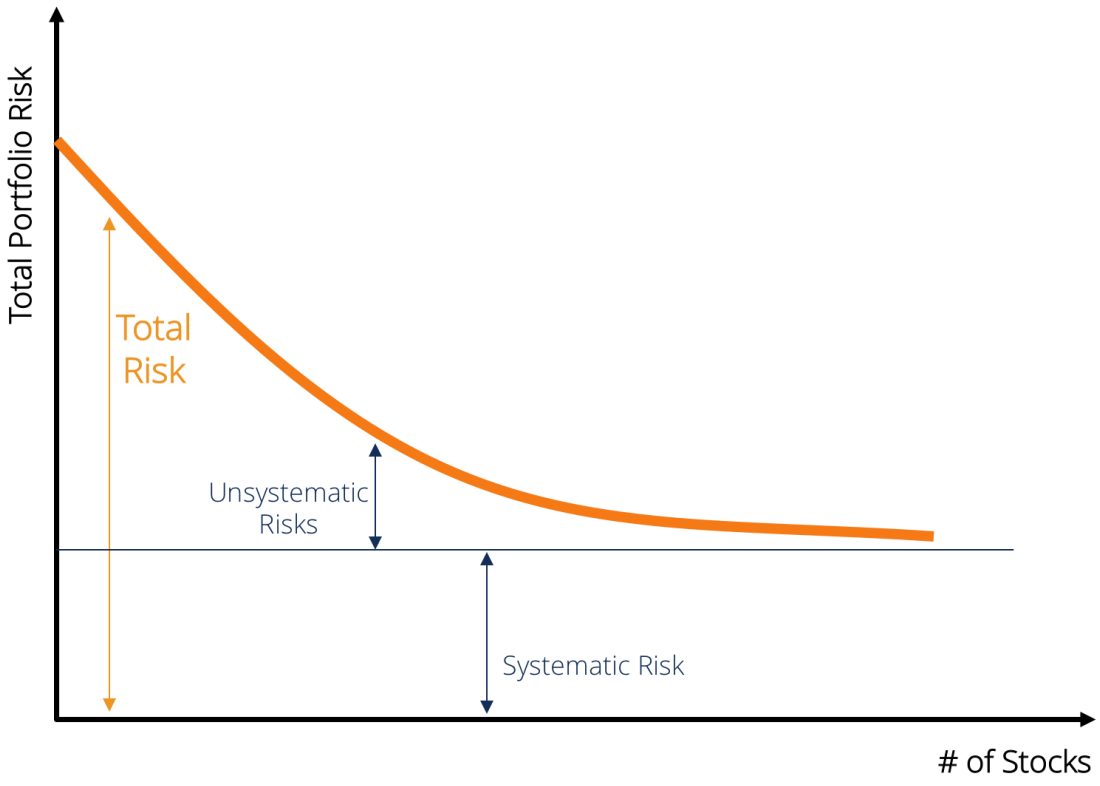


Figure 3. Unsystematic risk and diversification.
Credit:https://corporatefinanceinstitute.com/resources/knowledge/finance/systematic-risk/ [5]

The asymptotic horizontal line shown in this graph is β. As we can see, the level of risk in a given portfolio will eventually come down to just be β if the portfolio has a large number of securities. Hence, investors should primarily consider the overall β of their portfolio when making investment decisions.

The investor, then, must be compensated for taking on higher levels of risk(β), shown in the CAPM model.

The Model

The model presents the relationship between expected return for an individual security and level of systematic risk. The *Security Market Line*(SML) is the linearly changing value(also referred to as a line in some economic circles) that delineates *undervalued stocks* from *Overvaued stocks*. The line's y-intercept is the expected return from a "risk-free" security r_f, usually a government treasury. This security, then, has a β of 0. As risk(β) increases, the expected return value for an individual stock at which the stock becomes 'undervalued' increases as risk must be considered. This consideration is referred to as the *risk premium*, and is the difference between the expected return for a given stock and the risk-free return.

Risk premium = $[E(r_i) - r_f]$.

The total market also has a risk premium, referred to as the market risk premium:

Market risk premium = $[E(r_m)) - r_f]$

In fact, this value is referred to as the Treynor Ratio, and is the slope of the SML. It weights the β of a given security, and shows that increased risk must be compensated for as an investor. We can take the concepts layed out thus far to obtain the CAPM model.

The Capital Asset Pricing Model:

$$\mathbb{E}(r_i) = r_f + \beta_i * [\mathbb{E}(r_m) - r_f]$$

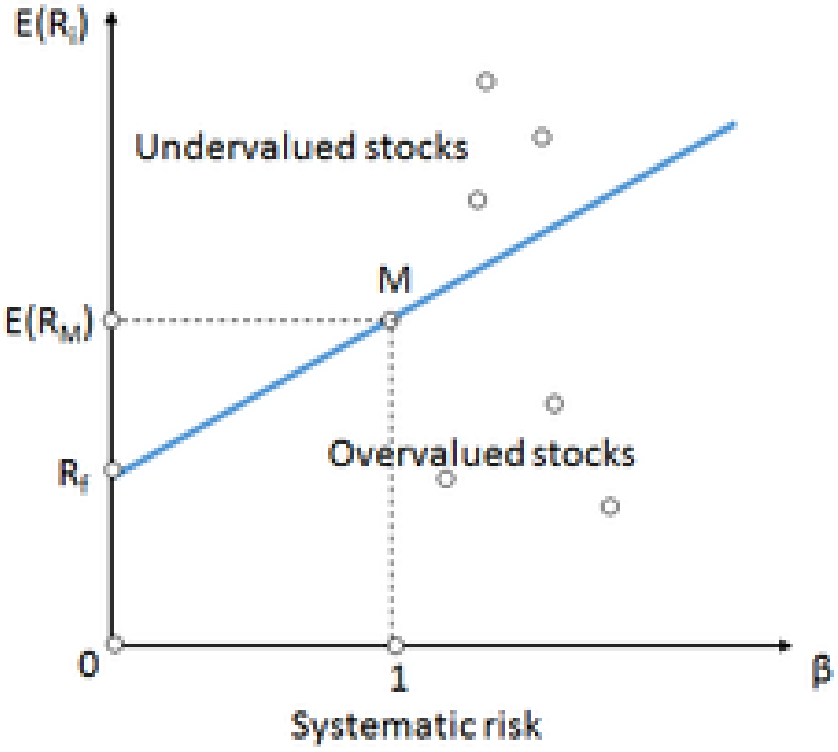


Figure 4. SML Line. Credit: user Lamro on Wikipedia.

Here, the blue line is the SML and the point M corresponds to the overall market level of risk.

Using this model, an investor can find the expected return for a security $\mathbb{E}(R_i)$.

Thsi model differs from its predecessor MPT in that it weights systematic risk, β.

In theory, an investor could now expect higher returns by simply raising their total portfolio risk, β_p. The portfolio total beta is:

$\beta_p = \sum_{i=1}^{n} w_i \beta_i$

where w_i is an individual securities proportion of the portfolio.

As a warm-up, I implement Python code that allows a user to find the monthly β for any individual stock since 1970 or in a specified time frame. IN addition, the user of the method can specify what frequency they want the beta to be analyzed at. The default is monthly, but could be analyzed daily, weekly, or yearly as well:

In [1]:
```
import pandas as pd
import pandas_datareader as web
import datetime as dt
import scipy.stats as stats
import numpy as np
```

In [2]:
```
sp500 = pd.read_csv("SP500", delimiter="        ")
```

In [3]:
```
def get_beta(ticker,
             start=dt.datetime(1970, 1, 1),
             end=dt.datetime.today(),
             freq="M"):

    total_market_data = web.DataReader(
        '^GSPC', 'yahoo', start,
        end)['Adj Close'].pct_change(freq="M").dropna()
    market_variance = np.var(total_market_data)

    # pull specified ticker data
    security_data = web.DataReader(
        ticker, 'yahoo', start,
        end)['Adj Close'].pct_change(freq="M").dropna()

    # get covariance for market and our stock
    market_security_covariance = np.cov(
        total_market_data.reindex_like(security_data), security_data)

    # calculate beta
    beta = (market_security_covariance / market_variance)

    # np.cov() returns a covariance matrix. We only want the covariance between our two
    return beta[0, 1]
```

Tesla has probably been a rather volatile stock. Lets see what its beta is.

In [4]:
```
get_beta("TSLA")
```

Out[4]:
```
1.669942666547994
```

This result makes sense. What about a blue-chip stock like Microsoft?

In [5]:
```
get_beta('MSFT')
```

Out[5]:
```
1.2081785249279309
```

Normally, we wouldnt think of Microsoft as being more volatile than the rest of the market. The reason it's volatility appears higher than the markets is because of the stock's early days as a young IPO in the dot-com bubble, at least that is the hypothesis. To test this, we get MSFT's beta from only the previous 10 years:

In [6]:
```
new_start = dt.datetime(2011, 11, 20)
get_beta('MSFT', start=new_start)
```

Out[6]:
```
0.8989416122480802
```

Ah ha! So the hypothesis may be correct. MSFT has, more recently, been a less volatile stock when compared to the whole market. This demostrates that the time period for which we analyze beta matters a great deal and will certainly change the beta value significantly, so we must keep that in mind and consider an investors time horizon.

Testing the Model

The obvious question after the invention of beta was: does beta actually predict returns in the real world? Could a savvy investor simply take on more risk and reap more reward?

In a study punlished in 1992, Eugene Fama and Kenneth French divided all stocks traded in the United States and put them into decitiles with the first decitile containing the 10% of stocks with the lowest betas and the last decitile containing the 10% of stocks with the highest betas. They then compared a stocks decitile ranking with its actual return. The result? Basically no correlation[6].

I will now attempt to reproduce these findings, to create a clearer picture of the efficacy of beta as a pricing model in the modern age.

I will use the stocks in the S&P500 and find their monthly β since 2010, to provide a clearly modern analysis of beta for an investor with a medium-long time horizon.

In [7]:
```
sp500
```

Out[7]:

	#	Company	Symbol	Weight	Price	Chg	% Chg
0	1	Microsoft Corporation	MSFT	6.424937	343.60	0.49	(0.14%)
1	2	Apple Inc.	AAPL	6.210814	160.99	0.44	(0.27%)
2	3	Amazon.com Inc.	AMZN	4.032845	3,672.16	-4.41	(-0.12%)
3	4	Alphabet Inc. Class A	GOOGL	2.260380	2,978.75	0.22	(0.01%)
4	5	Tesla Inc	TSLA	2.228760	1,140.00	2.94	(0.26%)
...	...	...	...	...	...	...	...
501	502	Under Armour Inc. Class A	UAA	0.012803	27.11	0.15	(0.56%)
502	503	Under Armour Inc. Class C	UA	0.011192	22.63	0.00	(0.00%)
503	504	Discovery Inc. Class A	DISCA	0.010978	25.31	0.05	(0.20%)
504	505	News Corporation Class B	NWS	0.007236	22.32	0.00	(0.00%)
505	506	Orion Office REIT Inc.	ONL	0.000071	17.27	-0.37	(-2.10%)

506 rows × 7 columns

In [8]:
```
# this dataset contains tickers for all SP500 stocks
ticker_list = sp500['Symbol']
```

In [9]:
```
ticker_list.head()
```

Out[9]:
```
0     MSFT
1     AAPL
2     AMZN
3    GOOGL
```

```
4     TSLA
Name: Symbol, dtype: object
```

The "Weight" column gives the percentage of the index the company takes up. So, later on, the aggregate beta values and percentage returns will have to be weighted according to this column

I now devise an algorithm to create a dataframe storing the ticker for each stock, along with its corresponding beta and percent return values from 2010-2021.

In [10]:

```
# create empty list to store betas
#beta_vs_returns_list = pd.DataFrame(
#    columns=["Ticker", "Beta", "Percent Return", "Weight"])

# timeframe is from 2010 to 2021
start = dt.datetime(2010, 1, 1)
end = dt.datetime(2021, 1, 1)

# pull market data once
total_market_data = web.DataReader('^GSPC', 'yahoo', start,
                                   end)['Adj Close'].dropna()
market_pct_changes = total_market_data.pct_change(freq="M").dropna()
market_pct_gain = (total_market_data[len(total_market_data) - 1] -
                   total_market_data[0]) / total_market_data[0] * 100

# market variance
market_var = np.var(market_pct_changes)

# loop through each ticker and store beta value
#for i in range(len(sp500)):
#    try:
        # pull data since 2010
#        stock_data = web.DataReader(sp500["Symbol"][i], 'yahoo', start,
#                                    end)['Adj Close'].dropna()

        # calculate return since 2010
#        stock_percent_returns = (
#            (stock_data[len(stock_data) - 1] - stock_data[0]) /
#            stock_data[0]) * 100

 #       stock_pct_change_data = stock_data.pct_change(freq="M").dropna()

 #       beta = np.cov(stock_pct_change_data,
 #                     market_pct_changes.reindex_like(
 #                         stock_pct_change_data)) / market_var

        # in next row of dataframe, store [ticker, beta of ticker, % return, weight]
 #       beta_vs_returns_list.loc[i] = [
 #           sp500["Symbol"][i], beta[0, 1], stock_percent_returns,
 ##           (sp500["Weight"][i] / 100)
 #       ]
#    except KeyError:
 #       beta_vs_returns_list.loc[len(beta_vs_returns_list)] = [
 #           "", np.nan, np.nan, np.nan
 #       ]
```

In [11]:

```
# saved as CSV for future use
beta_vs_returns_list = pd.read_csv("beta_vs_percent_with_weights_returns_list.csv")
```

One of the stocks was unable to be analyzed due to IPO'ing very recently. It can be dropped with no significant impact on our results.

In [12]:
```
beta_vs_returns_list = beta_vs_returns_list.dropna()
```

In [13]:
```
beta_vs_returns_list
```

Out[13]:

	Unnamed: 0	Unnamed: 0.1	Ticker	Beta	Percent Return	Weight	Weighted Returns
0	0	0	MSFT	0.885122	822.699112	0.064249	52.857900
1	1	1	AAPL	1.205686	1912.460095	0.062108	118.779339
2	2	2	AMZN	1.046028	2332.360030	0.040328	94.060465
3	3	3	GOOGL	1.130341	458.719542	0.022604	10.368805
4	4	4	TSLA	1.890632	14669.150292	0.022288	326.940154
...	...	...	...	...	...	...	...
497	497	500	GPS	1.537348	37.756871	0.000129	0.004863
498	498	501	UAA	1.140698	389.173793	0.000128	0.049826
499	499	502	UA	1.759662	-63.890075	0.000112	-0.007151
500	500	503	DISCA	1.444682	89.955258	0.000110	0.009875
501	501	504	NWS	1.692217	24.310671	0.000072	0.001759

502 rows × 7 columns

This list took my HP Pavillion a long time to complete, but here we have each stock on the S&P500's Ticker, almost-12-year monthly beta value, and the almost-12-year proportional return. I have saved it as a CSV for future use.

record rolling beta over time

Total market percent gain since 2010:

In [14]:
```
mkt_gain = (total_market_data[len(total_market_data)-1] - total_market_data[0]) / total
print(mkt_gain*100, '%')
```

```
231.51838063303444 %
```

In [15]:
```
beta_vs_returns_list["Weighted Returns"] = beta_vs_returns_list["Percent Return"] * bet
```

In [16]:
```
beta_vs_returns_list
```

Out[16]:

	Unnamed: 0	Unnamed: 0.1	Ticker	Beta	Percent Return	Weight	Weighted Returns
0	0	0	MSFT	0.885122	822.699112	0.064249	52.857900
1	1	1	AAPL	1.205686	1912.460095	0.062108	118.779339

	Unnamed: 0	Unnamed: 0.1	Ticker	Beta	Percent Return	Weight	Weighted Returns
2	2	2	AMZN	1.046028	2332.360030	0.040328	94.060465
3	3	3	GOOGL	1.130341	458.719542	0.022604	10.368805
4	4	4	TSLA	1.890632	14669.150292	0.022288	326.940154
...	...	...	...	...	...	...	...
497	497	500	GPS	1.537348	37.756871	0.000129	0.004863
498	498	501	UAA	1.140698	389.173793	0.000128	0.049826
499	499	502	UA	1.759662	-63.890075	0.000112	-0.007151
500	500	503	DISCA	1.444682	89.955258	0.000110	0.009875
501	501	504	NWS	1.692217	24.310671	0.000072	0.001759

502 rows × 7 columns

Average weighted return(should be very close to total market return to verify our data):

In [17]:
```
print(np.mean(beta_vs_returns_list["Weighted Returns"])*100, '%')
```

```
225.4391987101456 %
```

Due to the dropping of a few stocks that couldnt be analyzed, the data we have accurately represents the sp500 with around 95% accuracy, which, for our purposes, will suffice.

In [18]:
```
beta_vs_returns_list.to_csv(r"C:\Users\jzach\ECON411\beta_vs_percent_with_weights_retur
```

I now create labels for each row(i.e. each stock) to assign them to a given β decitile:

In [19]:
```
beta_quantiles = []
for i in range(1, 10, 1):
    beta_quantiles.append(beta_vs_returns_list["Beta"].quantile(q=i / 10))

beta_vs_returns_list["Beta Decitile"] = pd.qcut(beta_vs_returns_list["Beta"],
                                                10,
                                                labels=False)
```

In [20]:
```
beta_quantiles
```

Out[20]:
```
[0.5324972854905223,
 0.7078484057522195,
 0.8725625297441758,
 0.9702092402245285,
 1.0910629905155211,
 1.2008149708597897,
 1.3240239143786428,
 1.523736080549829,
 1.7461130440495123]
```

Now, I need to find the average return for each stock in each beta decitile, to look for correlation. After much deliberating, I settled on accomplishing this in the least efficient way possible, shown

below:

In [21]:
```
Decitile1 = []
Decitile2 = []
Decitile3 = []
Decitile4 = []
Decitile5 = []
Decitile6 = []
Decitile7 = []
Decitile8 = []
Decitile9 = []

for i in range(len(beta_vs_returns_list)):
    try:
        actual_return = beta_vs_returns_list["Weighted Returns"][i]
        if beta_vs_returns_list["Beta Decitile"][i] == 1:
            Decitile1.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 2:
            Decitile2.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 3:
            Decitile3.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 4:
            Decitile4.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 5:
            Decitile5.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 6:
            Decitile6.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 7:
            Decitile7.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 8:
            Decitile8.append(actual_return)
        elif beta_vs_returns_list["Beta Decitile"][i] == 9:
            Decitile9.append(actual_return)
    except KeyError:
        continue
```

Define "plot_data" to be the averages of the returns for stocks in each beta decitile:

In [22]:
```
plot_data = [
    np.mean(Decitile1),
    np.mean(Decitile2),
    np.mean(Decitile3),
    np.mean(Decitile4),
    np.mean(Decitile5),
    np.mean(Decitile6),
    np.mean(Decitile7),
    np.mean(Decitile8),
    np.mean(Decitile9)
]
```

Create new DataFrame for plotting and column-adding convenience:

In [23]:
```
plot_data_df = pd.DataFrame(plot_data, columns=["Average Weighted Return for Beta Decit
```

In [24]:
```
c = np.arange(0,11)
```

```
c
```

Out[24]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Add new column that labels the averages to their corresponding β Decitiles:

In [25]:
```
plot_data_df["Beta Decitile"] = [1, 2, 3, 4, 5, 6, 7, 8, 9]
plot_data_df["Beta Decitile Average Beta"] = beta_quantiles
plot_data_df
```

Out[25]:

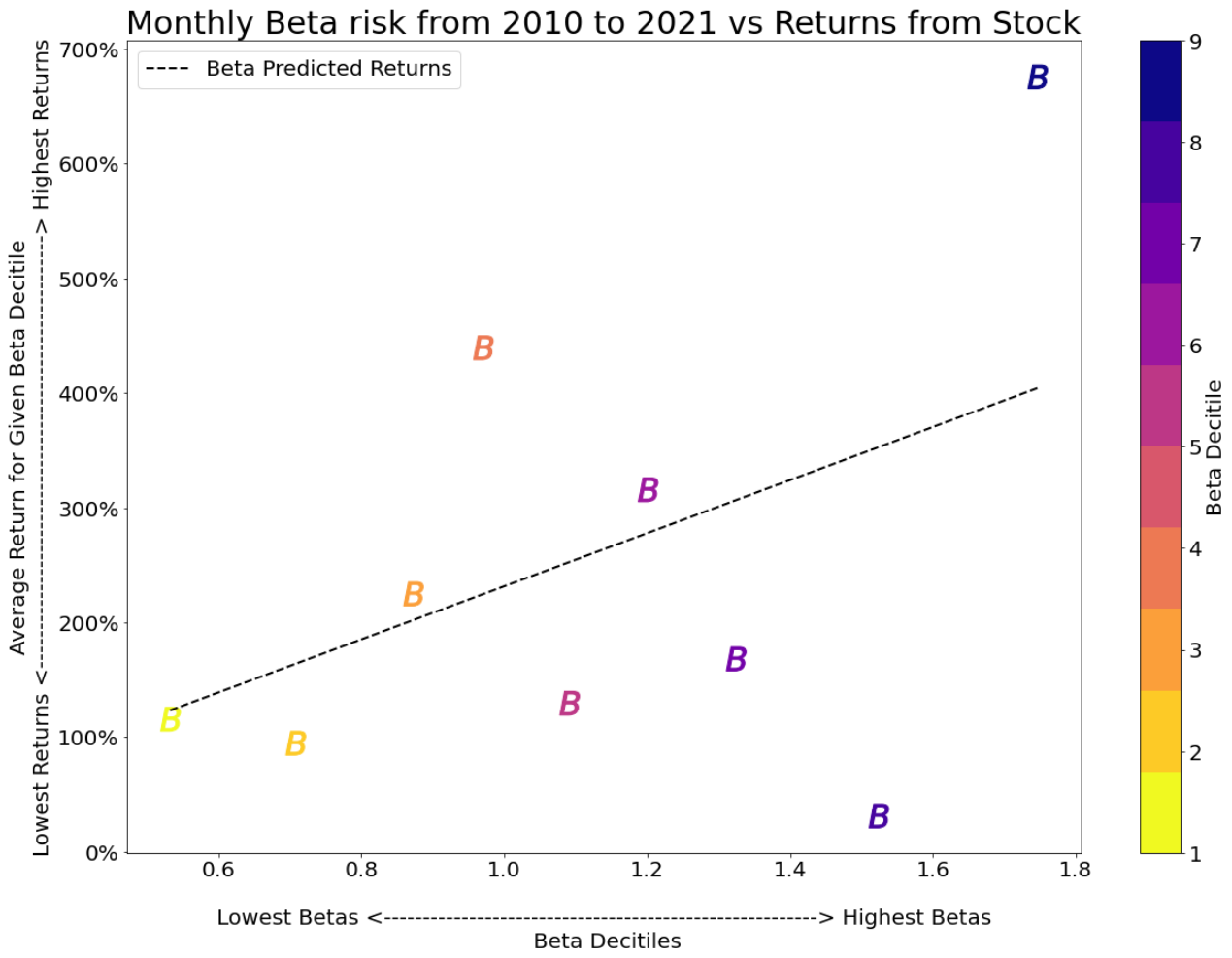
	Average Weighted Return for Beta Decitile	Beta Decitile	Beta Decitile Average Beta
0	1.156391	1	0.532497
1	0.946630	2	0.707848
2	2.248493	3	0.872563
3	4.394463	4	0.970209
4	1.295151	5	1.091063
5	3.156918	6	1.200815
6	1.681598	7	1.324024
7	0.312280	8	1.523736
8	6.754715	9	1.746113

And finally, plot the DataFrame to visually analyze correlation:

In [26]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.ticker as mtick

plt.rcParams.update({"font.size": 20})
fig, ax = plt.subplots(figsize=(20, 14))

#create color bar to show Beta level increasing
norm = cm.colors.Normalize()
cmap = cm.get_cmap('plasma_r', 10)
plt.cm.ScalarMappable(cmap=cmap, norm=norm)

plot_data_df.plot.scatter(x="Beta Decitile Average Beta",
                          y="Average Weighted Return for Beta Decitile",
                          c="Beta Decitile",
                          cmap=cmap,
                          norm=norm,
                          s=500,
                          marker='$B$',
                          ax=ax)
ax.set_xlabel(
    "\nLowest Betas <----------------------------------------------------------> Highest
)
ax.set_ylabel(
    "Average Return for Given Beta Decitile\nLowest Returns <-------------------------
)
```

```
x_coor = plot_data_df["Beta Decitile Average Beta"]
y_coor_for_line = mkt_gain * x_coor
ax.plot(x_coor,
        y_coor_for_line,
        '--',
        linewidth=2,
        markersize=12,
        c='k',
        label="Beta Predicted Returns")
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1))
ax.legend(loc='best')
ax.set_title("Monthly Beta risk from 2010 to 2021 vs Returns from Stock",
             fontsize=30)
```

Out[26]: Text(0.5, 1.0, 'Monthly Beta risk from 2010 to 2021 vs Returns from Stock')



It looks like the correlation is minimal, if any. This seems to confirm the findings of Fama and French. We can also run regression to see of the t-stat fails to reject our null hypothesis that there is no correlation:

In [27]:

```
import statsmodels.api as sm

X = plot_data_df["Beta Decitile"]
Y = plot_data_df["Average Weighted Return for Beta Decitile"]
X = sm.add_constant(X)
results = sm.OLS(Y, X).fit()
print(results.summary())
```

```
                                  OLS Regression Results
==============================================================================
=============
Dep. Variable:     Average Weighted Return for Beta Decitile   R-squared:
0.164
Model:                                                 OLS   Adj. R-squared:
0.045
Method:                                      Least Squares   F-statistic:
1.377
Date:                                     Tue, 23 Nov 2021   Prob (F-statistic):
0.279
Time:                                             09:11:59   Log-Likelihood:
-17.847
No. Observations:                                        9   AIC:
39.69
Df Residuals:                                            7   BIC:
40.09
Df Model:                                                1
Covariance Type:                                 nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const              0.9286      1.448      0.641      0.542      -2.495       4.352
Beta Decitile      0.3020      0.257      1.174      0.279      -0.306       0.910
==============================================================================
Omnibus:                        0.173   Durbin-Watson:                   2.241
Prob(Omnibus):                  0.917   Jarque-Bera (JB):                0.178
Skew:                           0.196   Prob(JB):                        0.915
Kurtosis:                       2.435   Cond. No.                         12.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

```
C:\Users\jzach\anaconda3\lib\site-packages\scipy\stats\stats.py:1541: UserWarning: kurto
sistest only valid for n>=20 ... continuing anyway, n=9
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Small sample size notwithstanding, the p value is large, .270. So there is almost no chance that the beta is positively correlated with real returns. We choose to fail to reject out null that they are uncorrelated.

We can also run regression using the whole dataset of each stock's beta vs return:

In [28]:
```
X = beta_vs_returns_list["Beta"]
Y = beta_vs_returns_list["Percent Return"]
X = sm.add_constant(X)
results2 = sm.OLS(Y, X).fit()
print(results2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Percent Return   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.3229
Date:                Tue, 23 Nov 2021   Prob (F-statistic):              0.570
Time:                        09:11:59   Log-Likelihood:                -4140.8
No. Observations:                 502   AIC:                             8286.
Df Residuals:                     500   BIC:                             8294.
Df Model:                           1
```

```
Covariance Type:            nonrobust
================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const        591.2062     82.214      7.191      0.000     429.680     752.733
Beta         -35.1978     61.940     -0.568      0.570    -156.892      86.496
==============================================================================
Omnibus:                      784.145   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           277065.481
Skew:                           8.665   Prob(JB):                         0.00
Kurtosis:                     116.780   Cond. No.                         3.88
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

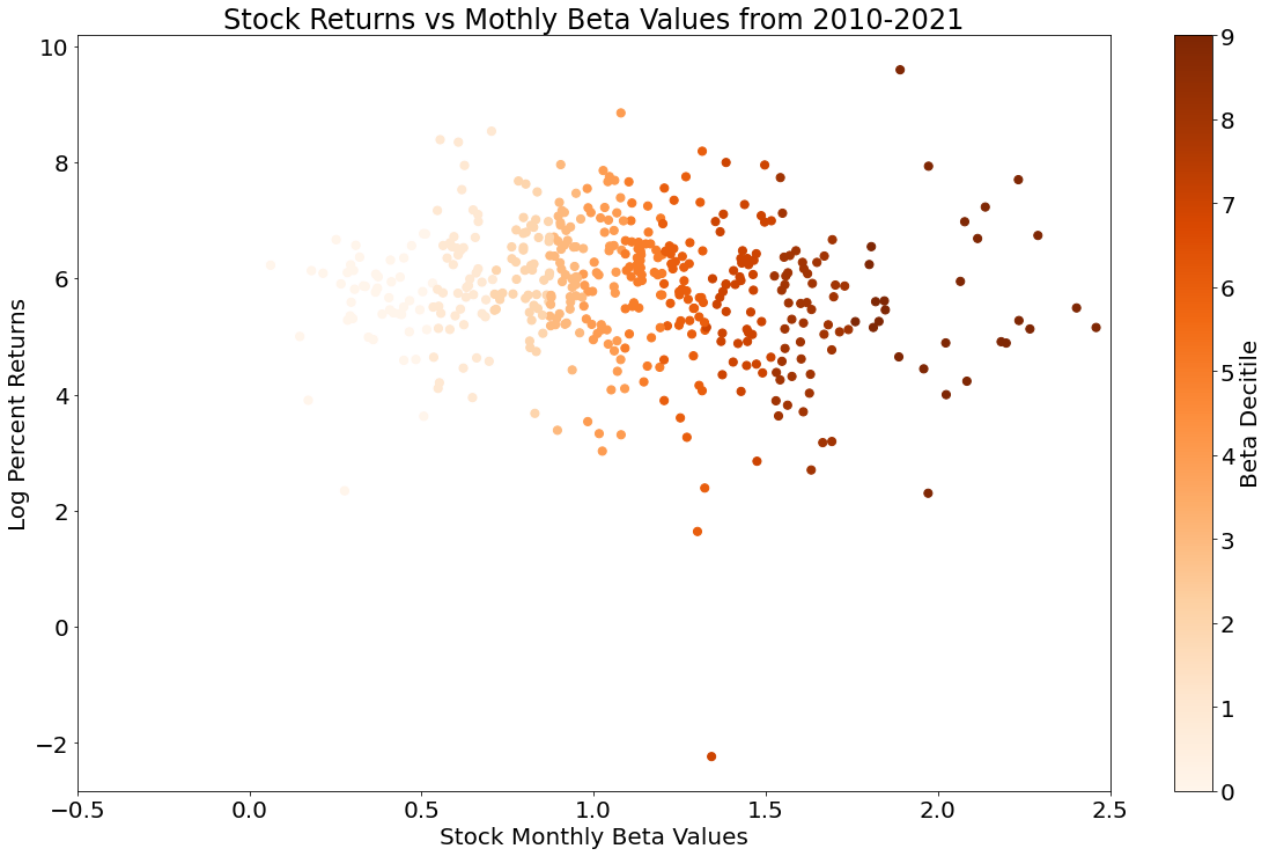
The R values are impressively low. Non-adjusted of .001, and adjusted of -.001 means there is absolutely zero correlation. With a p-value of 0.578, we easily fail to reject our null.

A scatterplot of beta versus actual returns for all values gives an idea of the strength of correlation. Note that the actual returns have been logged to normalize changes.

In [29]:
```
fig, ax = plt.subplots(figsize=(20, 12))
beta_vs_returns_list["Log Percent Returns"] = np.log(
    beta_vs_returns_list["Percent Return"])
#create color bar to show Beta level increasing
norm = cm.colors.Normalize()
cmap = cm.get_cmap('Oranges')
plt.cm.ScalarMappable(cmap=cmap, norm=norm)

beta_vs_returns_list.plot.scatter(x="Beta",
                                  y="Log Percent Returns",
                                  c="Beta Decitile",
                                  cmap=cmap,
                                  norm=norm,
                                  s=50,
                                  ax=ax)
# change xlim to get rid of some outliers from the graph
ax.set_xbound(lower=-.5, upper=2.5)

plt.title("Stock Returns vs Mothly Beta Values from 2010-2021")
plt.xlabel("Stock Monthly Beta Values")
```

```
C:\Users\jzach\anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning:
invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[29]:
```
Text(0.5, 0, 'Stock Monthly Beta Values')
```



As shown visually in this plot, the correlation is essentially none, which was confirmed by our OLS.

Based off of this analysis, I conclude that the CAPM Beta value of a stock does not predict future earnings with any accuracy. One should not take on excess risk just for the sake of taking on risk, if they believe it will increase future returns. That is not to say that Beta doesn't have any value. It still does indicate higher volatility, by definition. The past beta of a stock tells us a lot about its history. Almost every investor would favor a portfolio with lower beta over one with higher beta. So, investors want a lower beta, *ceteris paribus*. But, not all of the cetersises are paribus. We need a model with more explanatory power. In the second part of this paper, I analyze the Arbitrage Pricing Theory.

Arbitrage Pricing Theory

An arbitrage opportunity is an investment strategy that gurantees a positive payoff in some contingency with no possibility of a negative payoff and with no net investment[9].

First introduced by Stephen Ross in 1976[7], the Arbitrage Pricing Theory is an extension of the CAPM. The APT uses fewer assumptions but is more comlex than the CAPM. Ross developed the APT on the basis that the prices of securities are driven by multiple factors, which could be grouped into macroeconomic or company-specific factors. Ross[7] saw the limitations of the CAPM model as being its use of a single factor, β, to explain a linear relationship.

From Ross' 1976 paper[7]:

> "The restrictiveness of the assumptions that underlie the [CAPM] have, however, long been recognized, but its tractability and the evident appeal of the linear relation between return, $E(R_i)$, and risk, β embodied in [the CAPM model] have ensured its popularity."

Unlike the CAPM, the APT does not indicate the identity or even the number of risk factors. Instead, for any multifactor model assumed to generate returns, which follows a return-generating process, the theory gives the associated expression for the asset's expected return. While the CAPM formula requires the input of the expected market return, the APT formula uses an asset's expected rate of return and the risk premium of multiple macroeconomic factors.

The general model

The APT states that if asset returns follow a multifactor model then the following relation exists between expected returns and the factor sensitivities:

$$\mathbb{E}\left(r_i\right) = r_f + \beta_{i1}\delta_1 + \beta_{i2}\delta_2 + \cdots + \beta_{in}\delta_n + \epsilon_i$$

- $E(r_i)$ is the expected return for the ith asset
- r_f is the risk-free return rate
- δ_n is the nth factor's risk premium
- β_{in} is the *factor loading* specific to the nth factor and ith asset
- ϵ_i is a random error term which follows a normal distribution centered at 0, so a large enough number of assets will neglect this term, identical to the unsystemic risk factor mentioned earlier.

As we can see, the APT has mutiple factors, each of which have an associated beta, whereas the CAPM had only one beta, and one factor, which was the market risk premium.

In the CAPM, each asset i got a single beta: $\beta_i = \frac{Cov(r_i,r_m)}{Var(r_m)}$.

In the APT, each asset i can be analyzed using n factors each with a risk premium of $\delta_n = [E(\delta_n) - r_f]$, each of which have a β_{in}, which is the propensity of the ith asset to respond to changes in the nth factor, δ_n:

$$\beta_{in} = \frac{Cov(r_i,\delta_n)}{Var(\delta_n)}$$

My interest is in analyzing the predictive power of the APT, as I did with the CAPM model. In addition, I can find which macroeconomic factors affect market prices the most.

The factors

In a 1986 analysis by Chen, Nai-fu, and Ross[8], they theorized the most relevant macroeconomic factors to be incorporated into an APT model, using this logic:

Stock prices can be written as expected discounted dividends:

- $p = \frac{E(c)}{k}$,

where c is the dividend stream and k is the discount rate. This implies that actual returns in any period are given by:

- $\frac{dp}{p} + \frac{c}{p} = \frac{d[E(c)]}{E(c)} - \frac{dk}{k} + \frac{c}{p}$

It follows that the systematic forces that influence returns are those that change discount factors, k, and expected cash flows, E(c).[8]

For instance, some factors that could be used for analysis due to effects on c and k could be:

- Rate of inflation
- Growth rate in industrial production
- Fed Funds rate
- Long Term Treasury Maturity Minus short Term Treasury Maturity

I will start with these and then branch off to analyze other factors. I will use the S&P500 to represent the market as I did earlier. I will stick to the timeframe of 2010 to 2021.

the FRED will be my source for macro-factors.

The APT deals with returns as its input. And, since some of the macro-data is given quarterly, I will analyze all data on a quarterly basis to avoid NaNs.

In [30]:
```
total_market_data = web.DataReader('^GSPC', 'yahoo', start, end)
```

In [64]:
```
market_quarterly_change = total_market_data['Adj Close'].resample('Q').first().pct_chan
```

In [67]:
```
market_quarterly_change.describe()
```

Out[67]:
```
count    43.000000
mean      0.029617
std       0.087652
min      -0.241678
25%       0.007028
50%       0.041214
75%       0.067059
max       0.261227
Name: Adj Close, dtype: float64
```

Now we use FRED to pull our macro variables:

In [78]:
```
inflation = web.DataReader('CPIAUCSL', 'fred', start,
                           end).resample('Q').first().pct_change().dropna()
```

```
industrial_production = web.DataReader(
    'INDPRO', 'fred', start, end).resample('Q').first().pct_change().dropna()
fed_funds = web.DataReader('FEDFUNDS', 'fred', start,
                           end).resample('Q').first().pct_change().dropna()
long_minus_short = web.DataReader('T10Y3M', 'fred', start,
                           end).resample('Q').first().pct_change().dropna()

macro_vars = pd.DataFrame(columns=["Inflation", "Industrial Production", "Fed Funds Rat
```

In [92]:

```
macro_vars["Inflation"] = inflation["CPIAUCSL"]
macro_vars["Industrial Production"] = industrial_production["INDPRO"]
macro_vars["Fed Funds Rate"] = fed_funds["FEDFUNDS"]
macro_vars["Long Short Difference"] = long_minus_short["T10Y3M"]
```

In [94]:

```
macro_vars
```

Out[94]:

DATE	Inflation	Industrial Production	Fed Funds Rate	Long Short Difference
2010-06-30	-0.000391	0.014500	0.818182	-0.010610
2010-09-30	0.000929	0.020181	-0.100000	-0.252011
2010-12-31	0.006572	0.003407	0.055556	-0.146953
2011-03-31	0.009825	0.008255	-0.105263	0.348739
2011-06-30	0.013138	0.002400	-0.411765	0.056075
2011-09-30	0.005810	0.009466	-0.300000	-0.056047
2011-12-31	0.006012	0.011652	0.000000	-0.443750
2012-03-31	0.004816	0.011360	0.142857	0.095506
2012-06-30	0.005903	0.005546	0.750000	0.097436
2012-09-30	-0.002605	0.003748	0.142857	-0.294393
2012-12-31	0.013334	-0.001476	0.000000	0.026490
2013-03-31	0.000177	0.006480	-0.125000	0.148387
2013-06-30	0.000509	0.008417	0.071429	0.000000
2013-09-30	0.004758	-0.000730	-0.400000	0.382022
2013-12-31	0.003302	0.010073	0.000000	0.073171
2014-03-31	0.006929	0.001213	-0.222222	0.109848
2014-06-30	0.005015	0.017950	0.285714	-0.068259
2014-09-30	0.004356	0.009959	0.000000	-0.062271
2014-12-31	-0.000286	0.001392	0.000000	-0.062500
2015-03-31	-0.011300	-0.001390	0.222222	-0.125000
2015-06-30	0.006283	-0.015187	0.090909	-0.123810
2015-09-30	0.007671	-0.001015	0.083333	0.315217

DATE	Inflation	Industrial Production	Fed Funds Rate	Long Short Difference
2015-12-31	-0.001265	-0.010123	-0.076923	-0.152893
2016-03-31	-0.000341	-0.006780	1.833333	-0.014634
2016-06-30	0.005638	-0.010177	0.088235	-0.227723
2016-09-30	0.004640	0.004386	0.054054	-0.243590
2016-12-31	0.006830	-0.002314	0.025641	0.110169
2017-03-31	0.007773	0.000753	0.625000	0.465649
2017-06-30	0.002685	0.013081	0.384615	-0.187500
2017-09-30	0.000025	0.001728	0.277778	-0.173077
2017-12-31	0.009731	0.007772	0.000000	0.031008
2018-03-31	0.008368	0.003645	0.226087	-0.233083
2018-06-30	0.006248	0.020259	0.198582	-0.058824
2018-09-30	0.004275	0.001334	0.130178	-0.072917
2018-12-31	0.006183	0.004249	0.146597	-0.033708
2019-03-31	-0.001811	-0.006588	0.095890	-0.720930
2019-06-30	0.011428	-0.010145	0.008333	-0.750000
2019-09-30	0.002346	-0.001931	-0.008264	-4.000000
2019-12-31	0.005713	-0.005804	-0.237500	-0.055556
2020-03-31	0.005051	-0.003673	-0.153005	-3.000000
2020-06-30	-0.009645	-0.167076	-0.967742	0.558824
2020-09-30	0.009415	0.126920	0.800000	0.037736
2020-12-31	0.007185	0.018513	0.000000	0.072727
2021-03-31	0.006792	0.028581	0.000000	0.000000

The data looks good, lets check for correlation between the variables to see if we will have issues with multicollinearity

In [100…

```
macro_vars.corr()
```

Out[100…

	Inflation	Industrial Production	Fed Funds Rate	Long Short Difference
Inflation	1.000000	0.429913	-0.003105	0.040753
Industrial Production	0.429913	1.000000	0.449148	-0.034119
Fed Funds Rate	-0.003105	0.449148	1.000000	0.045173
Long Short Difference	0.040753	-0.034119	0.045173	1.000000

In [101…
```
import seaborn as sns
fig, ax = plt.subplots(figsize = (12, 8))
sns.heatmap(macro_vars.corr(), annot=True,  ax=ax)
```

Out[101… <AxesSubplot:>

The correlation data looks good as well. We could run regression to see if any signficant correlation exists as well.

In [130…
```
clist = ["Inflation", "Industrial Production", "Fed Funds Rate", "Long Short Difference
for i in range(4):
    for j in range(4):

        results = sm.OLS(macro_vars[clist[i]], macro_vars[clist[j]]).fit()
        if (clist[i] != clist[j]) and (j > i):
            print(clist[i]+','+clist[j]+':', results.rsquared_adj, results.pvalues)
```

```
Inflation,Industrial Production: 0.12742205504734605 Industrial Production    0.009263
dtype: float64
Inflation,Fed Funds Rate: 0.001438512046171847 Fed Funds Rate    0.30821
dtype: float64
Inflation,Long Short Difference: -0.005538286189670227 Long Short Difference    0.388896
dtype: float64
Industrial Production,Fed Funds Rate: 0.18955424576583402 Fed Funds Rate    0.001642
dtype: float64
Industrial Production,Long Short Difference: -0.02010963761017459 Long Short Difference
0.717517
dtype: float64
```

```
Fed Funds Rate,Long Short Difference: -0.022951995559674643 Long Short Difference    0.9
10549
dtype: float64
```

These results suggest that inflation and industrial production, and industrial production and the fed funds rate are barely correlated. However, the R^2 value for each is under 0.2. I will move forward with the analysis but one should note the slight correlation between these two sets of variables.

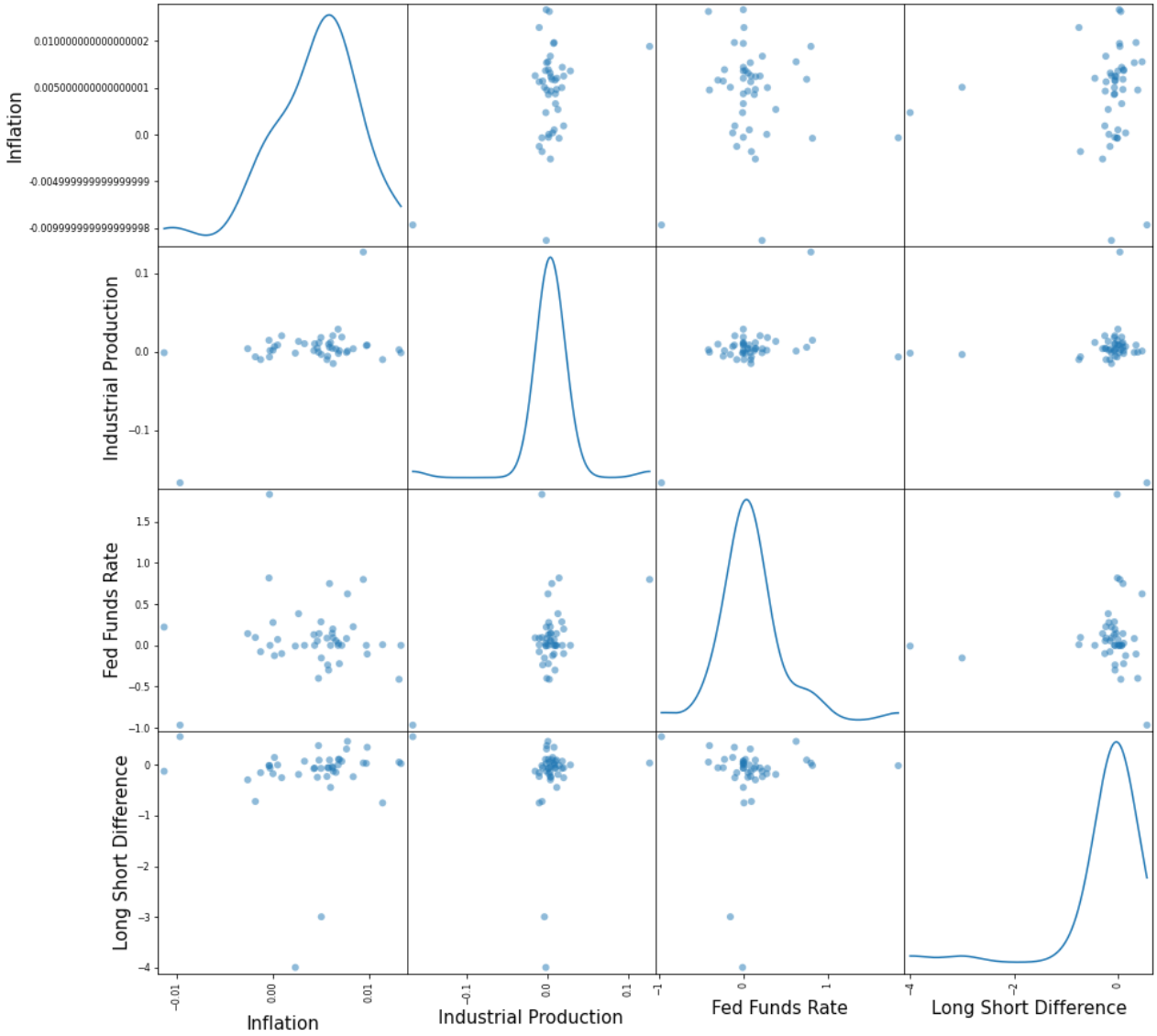
In [162…

```
fig, ax = plt.subplots(figsize = (15, 15))
plt.rcParams.update({'font.size':15})
pd.plotting.scatter_matrix(macro_vars, diagonal='kde', marker='0', ax = ax)
```

```
<ipython-input-162-096b708258e1>:3: UserWarning: To output multiple subplots, the figure
containing the passed axes is being cleared
  pd.plotting.scatter_matrix(macro_vars, diagonal='kde', marker='0', ax = ax)
```

Out[162…

```
array([[<AxesSubplot:xlabel='Inflation', ylabel='Inflation'>,
        <AxesSubplot:xlabel='Industrial Production', ylabel='Inflation'>,
        <AxesSubplot:xlabel='Fed Funds Rate', ylabel='Inflation'>,
        <AxesSubplot:xlabel='Long Short Difference', ylabel='Inflation'>],
       [<AxesSubplot:xlabel='Inflation', ylabel='Industrial Production'>,
        <AxesSubplot:xlabel='Industrial Production', ylabel='Industrial Production'>,
        <AxesSubplot:xlabel='Fed Funds Rate', ylabel='Industrial Production'>,
        <AxesSubplot:xlabel='Long Short Difference', ylabel='Industrial Production'>],
       [<AxesSubplot:xlabel='Inflation', ylabel='Fed Funds Rate'>,
        <AxesSubplot:xlabel='Industrial Production', ylabel='Fed Funds Rate'>,
        <AxesSubplot:xlabel='Fed Funds Rate', ylabel='Fed Funds Rate'>,
        <AxesSubplot:xlabel='Long Short Difference', ylabel='Fed Funds Rate'>],
       [<AxesSubplot:xlabel='Inflation', ylabel='Long Short Difference'>,
        <AxesSubplot:xlabel='Industrial Production', ylabel='Long Short Difference'>,
        <AxesSubplot:xlabel='Fed Funds Rate', ylabel='Long Short Difference'>,
        <AxesSubplot:xlabel='Long Short Difference', ylabel='Long Short Difference'>]],
      dtype=object)
```



The scatterplot matrix also verifies that the correlation between our independent variables is minimal.

I now build the model:

In [195…
```
macro_vars = macro_vars.iloc[:-1]
```

In [210…
```
macro_vars[(macro_vars.columns)].shift(-1).dropna().index
```

Out[210…
```
DatetimeIndex(['2010-06-30', '2010-09-30', '2010-12-31', '2011-03-31',
               '2011-06-30', '2011-09-30', '2011-12-31', '2012-03-31',
               '2012-06-30', '2012-09-30', '2012-12-31', '2013-03-31',
               '2013-06-30', '2013-09-30', '2013-12-31', '2014-03-31',
               '2014-06-30', '2014-09-30', '2014-12-31', '2015-03-31',
               '2015-06-30', '2015-09-30', '2015-12-31', '2016-03-31',
               '2016-06-30', '2016-09-30', '2016-12-31', '2017-03-31',
               '2017-06-30', '2017-09-30', '2017-12-31', '2018-03-31',
               '2018-06-30', '2018-09-30', '2018-12-31', '2019-03-31',
               '2019-06-30', '2019-09-30', '2019-12-31', '2020-03-31',
               '2020-06-30', '2020-09-30'],
              dtype='datetime64[ns]', name='DATE', freq='Q-DEC')
```

In [214…
```
X = macro_vars[(macro_vars.columns)]
Y = market_quarterly_change
X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
```

In [215…
```
print(model.summary())
```

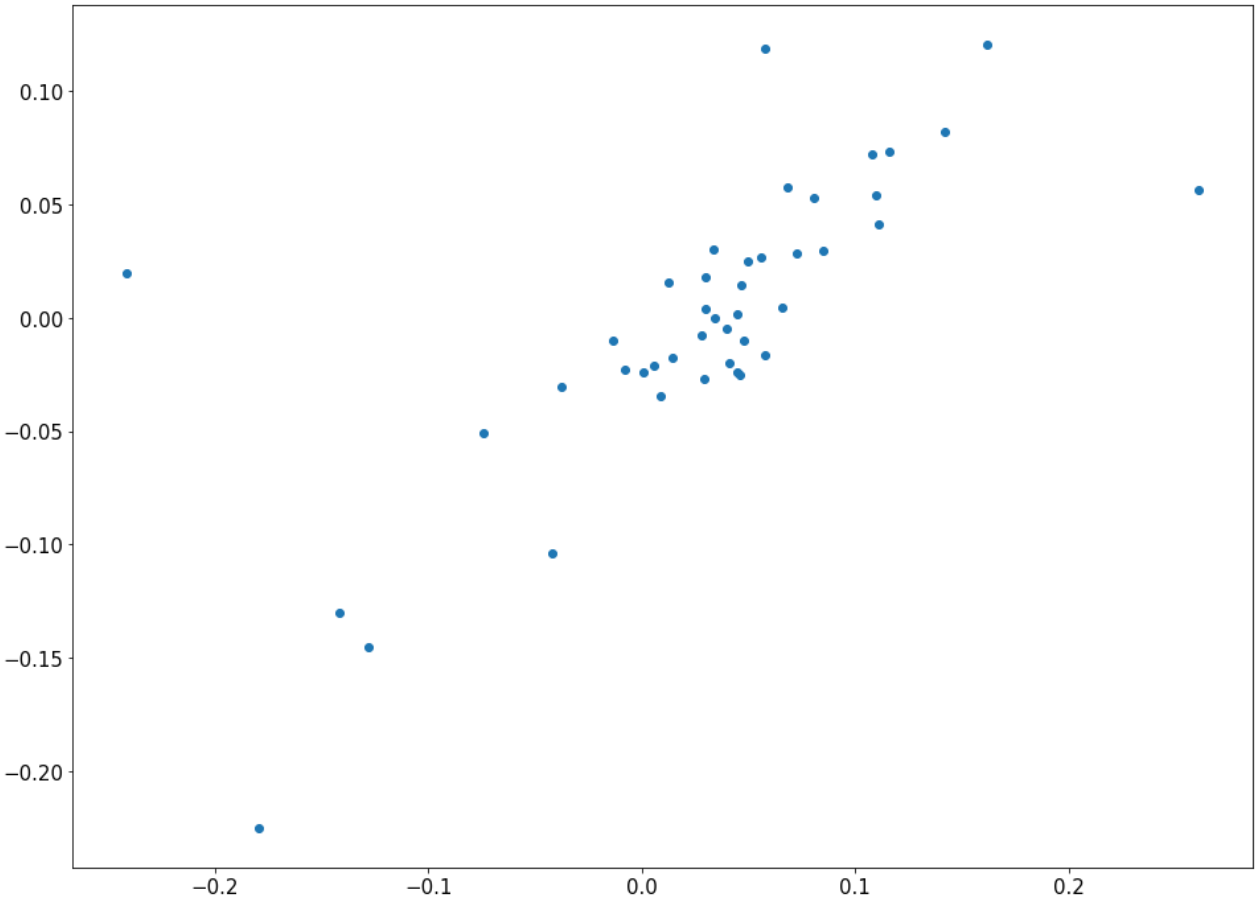
```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Adj Close   R-squared:                       0.460
Model:                            OLS   Adj. R-squared:                  0.403
Method:                 Least Squares   F-statistic:                     8.099
Date:                Tue, 23 Nov 2021   Prob (F-statistic):           7.97e-05
Time:                        21:18:39   Log-Likelihood:                 57.426
No. Observations:                  43   AIC:                            -104.9
Df Residuals:                      38   BIC:                            -96.05
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
                               coef    std err          t      P>|t|      [0.025      0.97
5]
--------------------------------------------------------------------------------
-
const                     -0.0037      0.015     -0.242      0.810      -0.034       0.02
7
Inflation                  5.9514      2.359      2.523      0.016       1.176      10.72
7
Industrial Production      0.9151      0.401      2.284      0.028       0.104       1.72
6
Fed Funds Rate             0.0453      0.029      1.551      0.129      -0.014       0.10
5
Long Short Difference     -0.0065      0.013     -0.486      0.630      -0.034       0.02
1
==============================================================================
Omnibus:                       15.851   Durbin-Watson:                   2.419
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               21.647
Skew:                          -1.124   Prob(JB):                     1.99e-05
Kurtosis:                       5.651   Cond. No.                         241.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

From our model, the S&P500 price change is significantly correlated with changes in inflation, and changes in industrial production. It is not correlated with the Fed Funds rate, or the difference between the long and short term treasury appreciation rate.

The fact that the S&Ps price is correlated with changes in inflation and industrial production are no suprise. Inflation is literally an increase in prices. Industrial production is exactly the type of variable which would incentivise investors to invest- increased production.

In [224…
```
fig, ax = plt.subplots( figsize=(16, 12))
plt.scatter(market_quarterly_change, model.resid)
```

Out[224…
```
<matplotlib.collections.PathCollection at 0x2710f65a970>
```

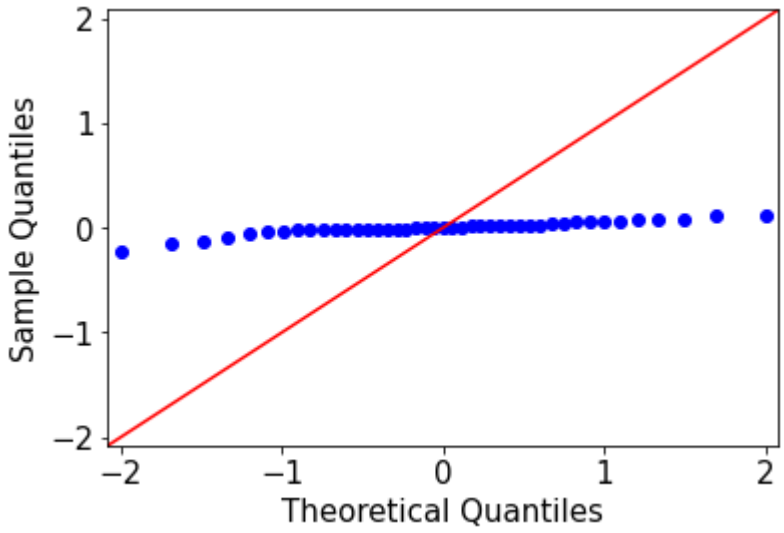


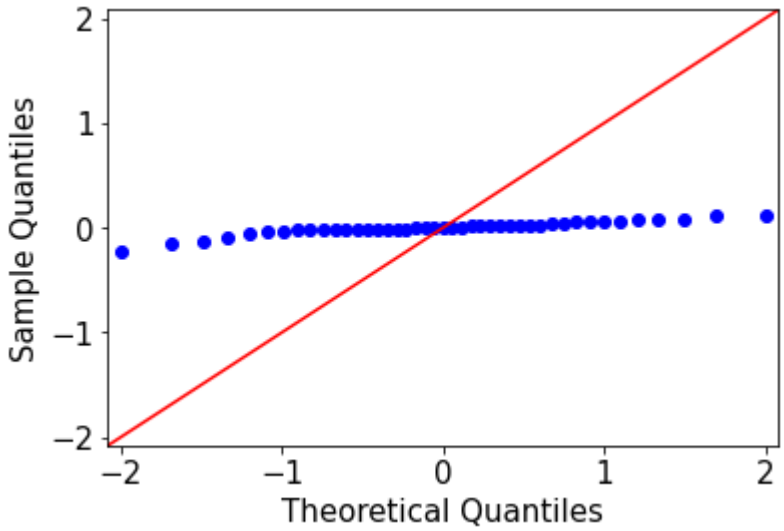
In [226…

```
sm.qqplot(model.resid, line="45")
```

```
C:\Users\jzach\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWar
ning: marker is redundantly defined by the 'marker' keyword argument and the fmt string
"bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```

Out[226…





Rough draft done. Need to sort out this model to normalize residuals.

References

1. Markowitz, Harry M.. Portfolio Selection: Efficient Diversification of Investments, New Haven: Yale University Press, 1968. https://cowles.yale.edu/sites/default/files/files/pub/mon/m16-all.pdf

2. Graham, Benjamin. Journal of Political Economy, vol. 47, no. 2, University of Chicago Press, 1939, http://www.jstor.org/stable/1826645.

3. Rustagi, R.P. (September 2010). Financial Management. India: Taxmann Publications (P.) Ltd. ISBN 978-81-7194-786-7.

4. Malkiel, Burton Gordon. A Random Walk down Wall Street : the Time-Tested Strategy for Successful Investing. New York :W.W. Norton, 2003, pp.220-221.

5. "Systematic Risk." Corporate Finance Institute, 15 Sept. 2021, https://corporatefinanceinstitute.com/resources/knowledge/finance/systematic-risk/.

6. FAMA, E.F. and FRENCH, K.R. (1992), The Cross-Section of Expected Stock Returns. The Journal of Finance, 47: 427-465. https://doi.org/10.1111/j.1540-6261.1992.tb04398.x

7. Ross, S. "The arbitrage pricing theory of capital asset pricing model." Journal of finance (1976).

8. Chen, Nai-Fu, Richard Roll, and Stephen A. Ross. "Economic forces and the stock market." Journal of business (1986): 383-403.

9. Dybvig, Philip H., and Stephen A. Ross. "Arbitrage." Finance. Palgrave Macmillan, London, 1989. 57-71.