

Volatility Forecasting and Linear Factor Models in Equity Markets

Economics 139 (Intermediate Financial Economics) Final Project

By Richard Jin, Anh-Tu Lu, Atul Madhugiri

Economics 139, Professor Stephen Bianchi

December 18, 2020

Overview

There are many important applications of predicting stock market volatility: risk management, asset pricing, portfolio selection, and market timing decisions. Because of this, the topic of modeling and forecasting stock market volatility is often discussed and researched. In this project, we will be looking at several popular volatility forecasting models to determine their effectiveness and accuracy in predicting individual stock and portfolio returns in relation to the market.

Preface






In order to analyze the historical financial data provided, we used the Jupyter Notebook computational environment. Specifically, we used the R and Python programming languages in order to conduct statistical analyses on the provided time series data and to create plots and visualizations. The specific libraries primarily used were `pandas` for storing data in Series and DataFrames, `numpy` in order to do computation, and `statsmodel` in order to do rolling computations and regressions.

```
# Imports for the various libraries used in this assignment
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.regression.rolling import RollingOLS
from statsmodels.regression.linear_model import OLS
from statsmodels.api import add_constant
```






Data

We were provided with a time-series dataset that contained financial data (returns, market cap, etc) for the time period between January 2, 2004 and December 31, 2009 (spanning 1511 trading days). Included in this dataset were the daily returns of 1,877 of the largest securities in the US equity markets. In addition, time-series data on overall market returns, Fama-French size factor returns, Fama-French value factor returns, and risk-free rate returns were included. The data was provided as whitespace-separated text files which we parsed into our computational environment.

Securities Data (1877 Securities, 1511 Trading Days)

 Ticker Number	 Returns	 Market Capitalizations	 0	 Ticker
1	<u>-0.015048</u>	13713091.20	1	A
1	<u>0.026042</u>	14070202.95	1	A
1	<u>0.031472</u>	14513021.52	1	A
1	<u>0.012795</u>	14698719.63	1	A
1	<u>0.045675</u>	15370089.72	1	A
...
1877	<u>0.000000</u>	580951.00	1877	ZRAN
1877	<u>-0.008079</u>	576257.50	1877	ZRAN
1877	<u>0.000000</u>	576257.50	1877	ZRAN
1877	<u>0.008145</u>	580951.00	1877	ZRAN
1877	<u>-0.008079</u>	576257.50	1877	ZRAN

Fama-French Factors Data

 Date Index	 Market Returns	 Returns - FF Size Factor	 Returns - FF Value Factor	 Risk Free Rate
0	<u>-0.0008</u>	0.0083	0.0041	0.00004
1	<u>0.0122</u>	0.0035	0.0002	0.00004
2	<u>0.0019</u>	0.0012	0.0018	0.00004

# Date Index	Aa Market Returns	Returns - FF Size Factor	Returns - FF Value Factor	Risk Free Rate
3	<u>0.0027</u>	0.0050	-0.0007	0.00004
4	<u>0.0048</u>	0.0033	0.0062	0.00004

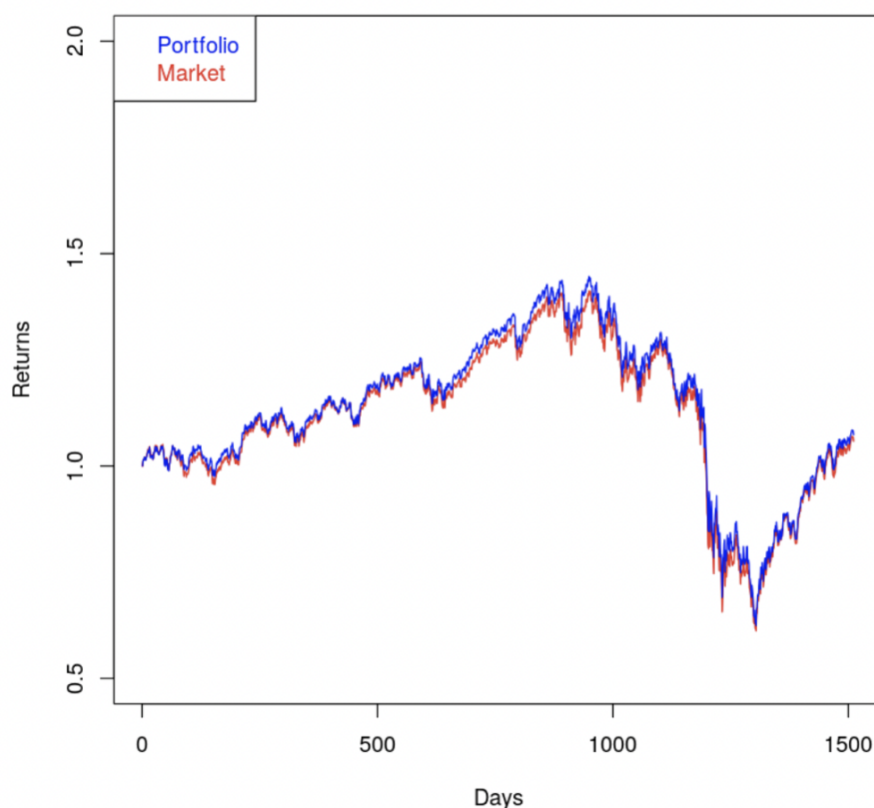
1506	<u>0.0052</u>	-0.0008	0.0016	0.00000
1507	<u>0.0010</u>	-0.0015	-0.0030	0.00000
1508	<u>-0.0009</u>	0.0008	-0.0008	0.00000
1509	<u>-0.0007</u>	-0.0004	-0.0018	0.00000
1510	<u>-0.0088</u>	-0.0013	0.0015	0.00000

Part 1

1A. Plotting cumulative returns to the market

We computed cumulative stock returns by dividing the total market capitalization at time t by the total market capitalization at time $t - 1$, giving us a daily return that could be compounded to get the cumulative return.

Cumulative Return of the 1877 Stocks vs. Cumulative Market Return



```
ylabel = "Returns"
xlabel = "Days"
title = "Cumulative Return of the 1877 Stocks vs Cumulative Market Return"
cumulative_returns(return_data['all']).plot(xlabel=xlabel, ylabel=ylabel)
cumulative_returns(ff_ret['mkt_ret']).plot(title=title)
```

From the plot above, we can observe that the cumulative returns of the `market` and `portfolio` are overlapping and following each other quite closely. This is a visual indication that the 1,877 stocks chosen are a representative of market returns.

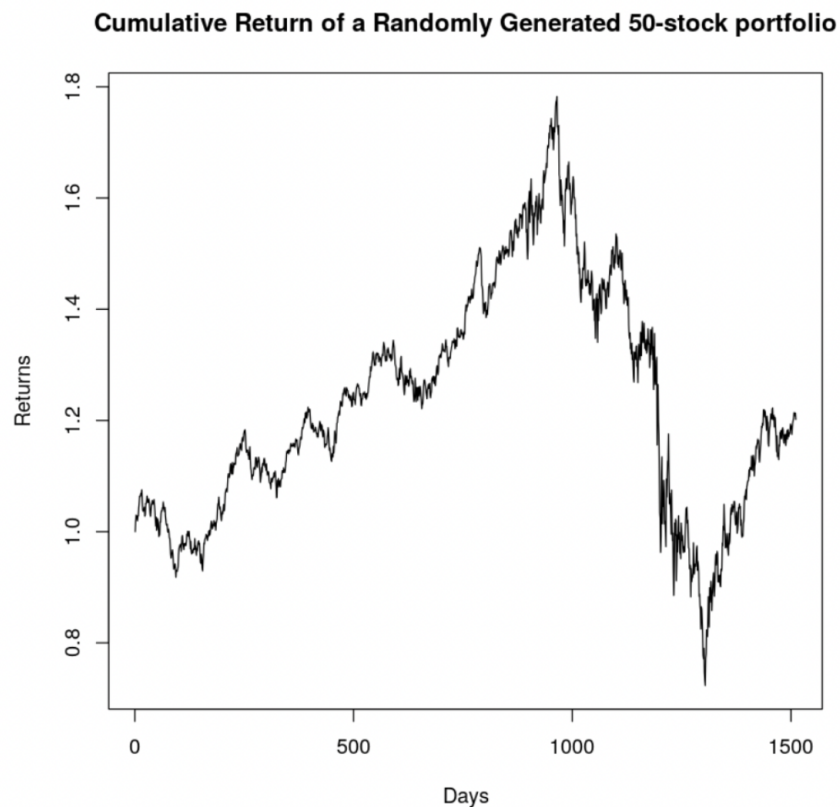
1B. Pick 50 stocks and plot cumulative return to a capitalization weighted portfolio

We picked 50 stocks at random from the given data set. We used randomly selected tickers to index into the market cap and daily return `DataFrames` and plot the data.

```
# Securities were selected at random using sample().
selected = tickers.sample(n=50)

# Market caps are summed to determine security weights
mkt_cap = mktcap_data[selected].copy()
mkt_cap_sum = cloned[selected].sum(axis=1)
mkt_cap[selected] = mkt_cap[selected].div(mkt_cap_sum,axis=0)

# The returns are then weighted in a new frame.
weighted_returns = pd.DataFrame()
for each in selected:
    weighted_returns[each] = mkt_cap[each] * returns[each]
```



Calculate the RMSE over the entire sample period.

```
In [112]: (sum((rnd_stock_net_returns - ff_returns)^2)/1511)^(1/2)
0.00629367147361637
```

Can you reduce the RMSE by choosing a different portfolio of 50 stocks?

The RMSE can be reduced by selected a different portfolio of 50 securities. To demonstrate this, we can select another 50 securities with another random sample. We obtain the following:

```
In [108]: (sum((rnd_stock_net_returns_2 - ff_returns)^2)/1511)^(1/2)
0.00591688424400269
```

We see here that the RMSE has decreased from 0.00629 to 0.00592. This supports the conclusion that a different sample of 50 securities can reduce RMSE.

1C. Build 25 n-stock portfolios of randomly chosen stocks

Calculate the mean RMSE over the 25 portfolios.

```
retval = 0
for _ in range(25):

    selected = tickers.sample(n=10)
    mkt = mktcap[selected].copy()
    mkt_sum = mkt[selected].sum(axis=1)
    mkt = mkt.div(mkt_sum.values, axis=0)
    weighted_returns = pd.DataFrame()

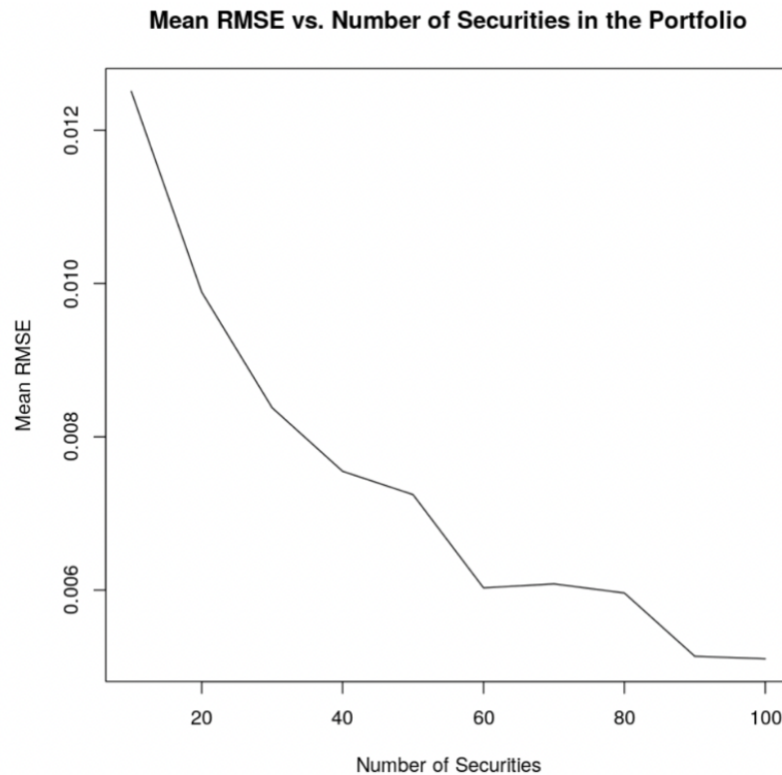
    for each in selected:
        weighted_returns[each] = mkt[each] * returns[each]

    weighted_returns["sum"] = weighted_returns.sum(axis=1)
    retval += ((weighted_returns["sum"]-returns) ** 2)

return retval
```

> 0.028128808394006888

Repeat for n = 10,20, ... , 100, and plot the mean RMSE as a function of the number of securities in the portfolios.



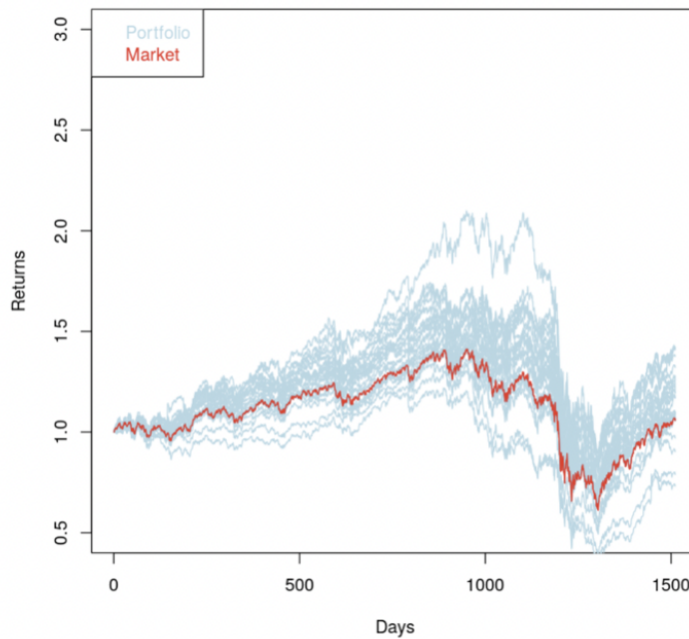
What did you find?

We observe that the mean RMSE steadily decreases as the number of securities increase. This is within expectation as more securities means a larger sample size and lower resulting variance.

Choose an n , and plot the cumulative returns to all 25 portfolios, and to the market.

We selected $n = 100$ and plotted the cumulative returns of all 25 portfolios and market below:

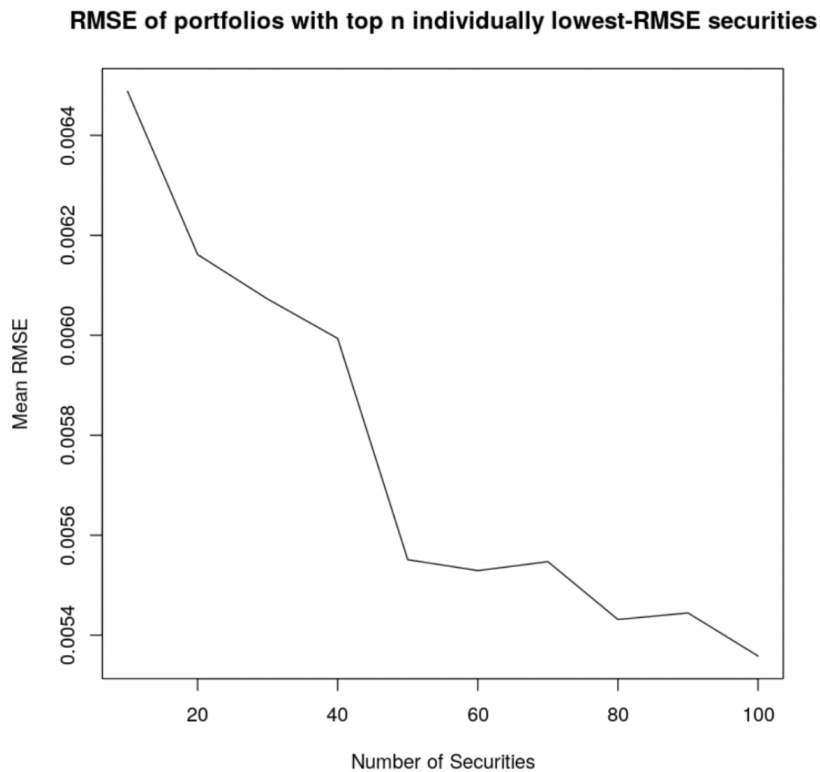
Cumulative Returns of 25 100-stock portfolios vs. Cumulative Market Ret



1D. Heuristic approach to choosing stocks

One heuristic that we can use is comparing each individual stock's return with the market return and picking the n stocks with the lowest RMSE for our portfolio. This approach results in the following relationship of RMSEs as a function of the number of securities:

Build several n -stock portfolios, where $n = 10, 20, \dots, 100$, and plot the RMSE as a function of the number of stocks in the portfolio.



How do your results compare to results using randomized portfolios?

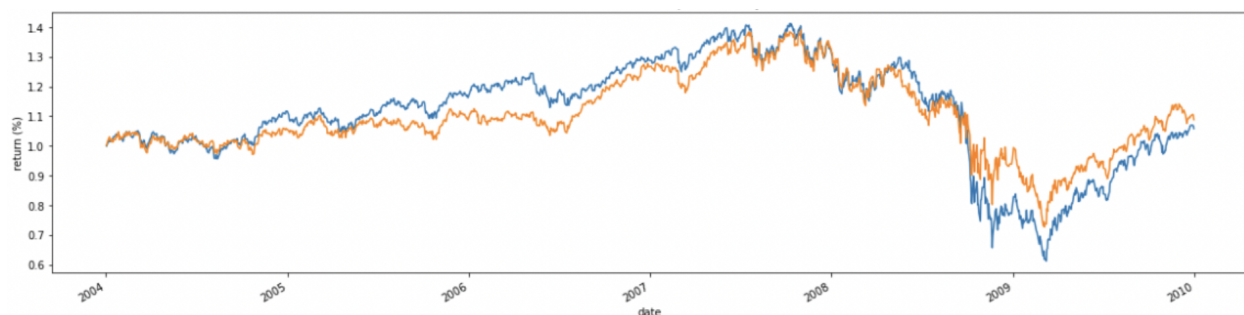
We observe that by constructing portfolios with n securities that have the n -lowest individual RMSE, we were able to drastically improve our RMSE for portfolios with lower numbers of securities. We were able to achieve an RMSE of 0.0065 with only 10 securities using this new method of security selection.

When we were using the method of random security selection, the mean RMSE of 10 securities was **0.0125**. Hence, this new method has cut the RMSE of portfolios with 10 securities in half.

For larger number of securities, we see that the RMSE of this new method is pretty much the same as the RMSE of the old method where we randomly select securities. The value of this heuristic lies in the fact that we can minimize the number of securities that we put into our portfolio (to 10) without sacrificing as much RMSE as before.

Choose an n , and plot cumulative returns to your portfolio and to the market.

$n=10$:



Part 2: MPT

For this section, we used the modern portfolio theory (MPT) approach to estimate volatility. MPT theorizes that risk-averse investors can maximize their expected returns by constructing specialized portfolios based on a chosen level of risk.

This method is considered one of the most convenient ways to measure a stock or portfolio's volatility. To test the theory and estimate volatility, we choose the first 100 securities in alphabetical order to comprise our portfolio. Each step below was completed using rolling windows of 504, 252, 216, and 63 days.

2A. MPT: Portfolio of 100 securities.

(We will show steps with rolling window of size 504. We'll then show a summary of the results for all windows at the end of this section.)

We begin by selecting the first 100 tickers in alphabetical order to create our portfolio.

```
[18]: # Select hundred securities to be used throughout.
hundred_securities = tickers[:100]
return_data[hundred_securities].head()
```

	A	AA	AAI	AAON	AAP	AAPL	ABAX	ABC	ABCB	ABI
date										
2004-01-02	-0.015048	-0.011842	0.030252	-0.024729	0.000246	-0.004212	-0.072416	-0.038290	0.011801	-0.0045
2004-01-05	0.026042	0.032756	0.008157	0.004754	-0.009089	0.041823	-0.072706	0.033889	0.016574	0.0037
2004-01-06	0.031472	-0.007478	0.062298	0.014196	0.017848	-0.003608	0.097044	-0.004120	0.006099	0.0022
2004-01-07	0.012795	-0.007534	0.022848	-0.005184	0.031417	0.022635	0.062683	0.009712	0.012544	-0.0014
2004-01-08	0.045675	0.012042	-0.067014	0.001042	-0.026446	0.034086	0.006064	-0.004809	0.007706	-0.024

5 rows x 100 columns

Estimate a sample covariance matrix (100 x 100) of the returns to your securities.

We then estimate a covariance matrix using the `.rolling()` functionality of pandas to calculate the covariance for every 504-day time window within our time-series. We will have empty values for the top rows of the matrix because an entire time-window is not able to fit before it. We use the `.dropna()` function to get rid of these rows. We also get rid of the last row to avoid out-of-bounds issues when it is time for the one-day ahead returns.

```
[19]: # Calculate sample covariance matrix (100 x 100) of returns to securities.
# Drop NAs and trail row (see one-day ahead portion).

cov_504 = return_data[hundred_securities].rolling(504).cov()
cov_504.dropna(inplace=True)
cov_504 = cov_504.drop(cov_504.tail(100).index)
cov_504.head()
```

```
[19]:
```

		A	AA	AAI	AAON	AAP	AAPL	ABAX	ABC	ABCB	ABCO	...	AMED	AM
	date													
2005-12-30	A	0.000438	0.000085	0.000203	0.000117	0.000071	0.000140	0.000142	0.000058	0.000109	0.000071	...	0.000090	0.000005
	AA	0.000085	0.000255	0.000123	0.000068	0.000062	0.000122	0.000086	0.000053	0.000103	0.000056	...	0.000067	0.000001
	AAI	0.000203	0.000123	0.000910	0.000135	0.000131	0.000171	0.000280	0.000100	0.000219	0.000118	...	0.000065	0.000011
	AAON	0.000117	0.000068	0.000135	0.000482	0.000065	0.000090	0.000122	0.000040	0.000112	0.000053	...	0.000098	0.000000
	AAP	0.000071	0.000062	0.000131	0.000065	0.000316	0.000070	0.000099	0.000007	0.000089	0.000046	...	0.000053	0.000001

5 rows x 100 columns

Using the market capitalization weights (from last day in the rolling window), estimate the standard deviation of the portfolio $\hat{\sigma}_p$.

```
[20]: wgt_504 = mktcap_data[hundred_securities].copy()
wgt_504['sum'] = wgt_504[hundred_securities].sum(axis=1)
wgt_504[hundred_securities] = wgt_504[hundred_securities].div(wgt_504['sum'].values,axis=0)
wgt_504.drop(wgt_504.tail(1).index, inplace=True)
```

```
[31]: sd_504 = []
cov504_np = cov_504.to_numpy()
arr504_weight = wgt_504[hundred_securities].tail(1007).to_numpy()

for i in range(0, len(cov504_np), 100):
    idx = 0
    sd_504.append(sd_portfolio(cov504_np[i:i+100], arr504_weight[idx]))
    idx += 1

sd_504_np = np.array(sd_504)
sd_504_np
```

```
[31]: array([0.00820984, 0.00824556, 0.00823351, ..., 0.03161844, 0.03161853,
        0.03161824])
```

In the top cell, we determine the weights for each security in the portfolio by normalizing and determining what proportion of the total portfolio market cap each security comprises.

In the subsequent cell, we use the weights and covariances we found to estimate the overall standard deviation of the portfolio σ_p .

Using the market capitalization weights and returns (from the day following the last day in the rolling window) of your securities, calculate the one-day ahead return of the portfolio \tilde{r}_p .

```
[34]: dayahead_504 = []
      ret_copy = return_data[hundred_securities].copy()
      ret_copy.drop(ret_copy.tail(1).index, inplace=True)
      dayahead_504_ret = ret_copy.tail(1007).to_numpy()
      dayahead_504_wgt = wgt_504[hundred_securities].tail(1007).to_numpy()

      for i in range(0, len(dayahead_504_wgt)):
          dayahead_504.append(np.multiply(dayahead_504_ret[i], dayahead_504_wgt[i]))

      dayahead_504_ret_np = np.sum(np.array(dayahead_504), axis=1)
      dayahead_504_ret_np
```

```
[34]: array([-0.00618081,  0.01826034,  0.00564215, ...,  0.00519065,
           -0.00414611,  0.00514476])
```

We construct the one-day ahead return of the portfolio by multiplying together the day ahead's return and weights and computing a sum across the securities of the portfolio.

Calculate the standardized outcome \tilde{z}_p , where

$$\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$$

where we make the simplifying assumption that $E[\tilde{r}_p] = 0$.

```
[37]: std_outcomes = dayahead_505_ret_np / sd_504_np
std_504 = pd.DataFrame(std_outcomes)
std_504.index += 504
std_504.rename(columns={0: "Standardized Outcome"}, inplace=True)
std_504.head()
```

```
[37]:
```

	Standardized Outcome
504	-0.752854
505	2.214566
506	0.685266
507	-0.066878
508	1.336027

We compute the standard outcome by dividing the one-day ahead rate by the portfolio's standard deviation. By then calling `numpy.std` on the standard outcome, we arrive at the bias statistic. In the case of a rolling window of 504, the bias statistic is **1.169**. The bias statistics for the remaining window sizes—along with analysis—can be found in the next section. The code to calculate the values for the other window sizes is negligibly different, but can be reviewed completely in the appendix.

2B. Bias Statistic

MPT Model Bias Statistics

<u>Aa</u> Rolling Window	<u>#</u> Bias Statistic
<u>504</u>	1.169
<u>252</u>	1.037
<u>126</u>	0.969
<u>63</u>	0.947

We observe that the bias statistics decreases with the size of the rolling windows. These results make sense because the market direction is constantly changing. The constantly changing directions causes older data to become obsolete and have less predictive power compared to current data. In many cases, the older data may even become a hindrance to the current data because it represents completely different trends. This is why we see rolling windows with smaller sizes, which are less affected by older data, perform better than rolling windows with larger sizes, which are more affected by older

data. **Rolling window 126 has the best bias statistic, since it is closest to 1.** This may be due to it being the perfect window between having enough windows to reduce market risk while also having low enough windows to reduce idiosyncratic risk. Part of the reason is also the volatility of our data in particular as we observe major ups and downs in returns.

Part 3: CAPM

For this part, we utilized the market model approach to estimate volatility. Beta measures volatility relative to the stock market, and it can be used to evaluate the relative risks of stocks or determine the diversification benefits of other asset classes. A beta of 1 means the security has volatility that mirrors the degree and direction of the market as a whole. This can be illustrated in the example of if the S&P sharply falls, then portfolios with betas closest to 1 are likely to follow suit and fall by a similar amount.

Each step below was completed using rolling windows of 504, 252, 216, and 63 days. For calculations, we used the same portfolio as we did in Part 2.

3A. Computing Statistics using Rolling Windows

The following code generates our betas for the rolling windows of 504. We utilized linear regressions to find the coefficient of correlation between the market and equity risk premiums.

The regression formula that we used, which is the capital asset pricing model (CAPM), is:

$$\tilde{r}_i - r_f = \alpha_i + \beta_i(\tilde{r}_M - r_f) + \tilde{\epsilon}_i$$

```
betas_q3_504 = np.zeros(shape=(1007,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_beta = []
    for i in range(1007):
        model = OLS(ri_minus_rf[i:i+503], add_constant(rm_minus_rf[i:i+503]))
        res = model.fit()
        beta = res.params[1]
        col_beta.append(beta)
```

```
betas_q3_504 = np.c_[betas_q3_504, col_beta]
pd.DataFrame(betas_q3_504).drop(0, axis = 1)
```

```
In [117]: betas_df_q3_504 = pd.DataFrame(betas_q3_504).drop(0, axis = 1)
betas_df_q3_504
```

```
Out[117]:
```

	1	2	3	4	5	6	7	8	9	10	...	91	92	93	94	95
0	1.586132	1.400397	2.201703	1.143659	0.984847	1.567119	1.805154	0.771245	1.559725	0.864650	...	0.922545	1.163089	1.070224	2.305267	1.51065
1	1.585057	1.398955	2.201317	1.139775	0.979738	1.564692	1.801294	0.769244	1.564287	0.864417	...	0.922761	1.163957	1.070103	2.303939	1.50857
2	1.568528	1.383266	2.199448	1.143749	0.980552	1.562290	1.839123	0.762197	1.543885	0.861147	...	0.907057	1.158397	1.077900	2.256219	1.52920
3	1.565407	1.383734	2.198504	1.141111	0.979743	1.561133	1.843936	0.760247	1.543489	0.863790	...	0.911997	1.158007	1.075599	2.253799	1.52714
4	1.563436	1.384451	2.197515	1.141732	0.976663	1.560020	1.838144	0.760024	1.542143	0.865187	...	0.912934	1.157517	1.075112	2.250480	1.52657
...
1002	1.049304	1.718971	1.326889	1.149744	0.822715	0.948279	1.031868	0.537028	1.998436	0.721880	...	0.883524	1.774767	0.598938	1.442129	1.07598
1003	1.049207	1.719024	1.326736	1.149722	0.822565	0.948359	1.033234	0.536986	1.998578	0.722371	...	0.883948	1.774756	0.599008	1.442054	1.07385
1004	1.049402	1.719990	1.325560	1.148348	0.822522	0.949578	1.031047	0.537483	1.995442	0.720507	...	0.885010	1.774997	0.599195	1.440668	1.07280
1005	1.049348	1.719963	1.325524	1.148276	0.822531	0.949600	1.031096	0.537497	1.995745	0.720484	...	0.885050	1.775171	0.599277	1.440606	1.07282
1006	1.048995	1.720058	1.325445	1.147740	0.822563	0.949548	1.030476	0.537204	1.996368	0.719834	...	0.884997	1.775964	0.599013	1.440062	1.07311

1007 rows x 100 columns

These betas give the measure of the volatilities of the security on a given day compared to the market as a whole. Through this, we generated a matrix of $(1511 - rw) \times 100$ betas for each stock in the random portfolio.

Next, we estimated the variance of the market, $\hat{\sigma}_M^2 = Var(\tilde{r}_M)$, and idiosyncratic variances, $\hat{\sigma}_{\epsilon_i}^2 = Var(\tilde{\epsilon}_i)$, of each security. To do this, we created rolling windows and calculating the variance for each window period and used the residuals function within our model to find idiosyncratic variances.

```
#variance of market returns
var_rm_q3_504 = list(ffdata['Market Returns'].rolling(504).var())
```

```
#idiosyncratic variances of each security
var_e_q3_504 = np.zeros(shape=(1007,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_vars = []
    for i in range(1007):
        model = OLS(ri_minus_rf[i:i+503], add_constant(rm_minus_rf[i:i+503]))
        res = model.fit()
        varis = res.resid #getting the residuals
        col_vars.append(np.var(varis))
    var_e_q3_504 = np.c_[var_e_q3_504, col_vars]
```

Then, we estimated the variance and standard deviation of our portfolio, using the formula:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T \hat{\beta} \hat{\sigma}_M^2 \hat{\beta}^T w + w^T \hat{\Delta} w$$

where w is a (100×1) vector of the security capitalization weights (normalized to sum to 1), $\hat{\beta}$ is a (100×1) vector of the security betas with respect to the market from part (i) and $\hat{\Delta}$ is a (100×100) diagonal matrix with the security idiosyncratic variances on the diagonal.

```
# Define function to get variance of portfolio
def var_portfolio(w, b, m, d):
    return np.dot(np.dot(np.dot(np.dot(np.transpose(w), b), m),
                                np.transpose(b)), w) + np.dot(np.dot(np.transpose(w), d), w)
```

Afterwards, similarly to part 2, we computed the standardized outcomes, $\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$, for rolling window and attained its standard deviations to get the bias statistics for our next section.

```
In [147]: standardized_outcomes_504_q3 = dayahead504_port_ret_q3_arr / arr_sd_q3_504
std_outcomes_504_q3 = pd.DataFrame(standardized_outcomes_504_q3)
std_outcomes_504_q3.index += 504
std_outcomes_504_q3.rename(columns={0: "Standardized Outcome"}, inplace = True)
std_outcomes_504_q3
```

Out[147]:

Standardized Outcome	
504	2.241673
505	0.687769
506	-0.067248
507	1.343726
508	0.453598
...	...

3B. Bias Statistics for Market Model

Market Model Bias Statistics

[Aa](#) Rolling Window <#> Bias Statistic

<u>Aa</u> Rolling Window	<u>#</u> Bias Statistic
<u>504</u>	1.312
<u>252</u>	1.132
<u>126</u>	1.075
<u>63</u>	1.058

From this, we can see that the best bias statistic for the market model is 63. The motivation for a smaller window size is that it has increased sensitivity to changes in the underlying process from which we are sampling. A smaller rolling window reduces the idiosyncratic risk of individual securities that could affect their performance, deviating their returns away from market returns. This shows Compared to the MPT approach, we find that the bias statistics are overall a bit higher as MPT tries to maximize returns given a certain level of risk.

Part 4: Fama-French

For this part, we're utilizing the Fama-French model approach to estimate volatility. The model uses three different factors: market risk, the outperformance of small versus big companies (FF size factor), and the outperformance of large market capitalization versus small market capitalization companies (FF value factor).

Each step below was completed using rolling windows of 504, 252, 216, and 63 days. For calculations, we used the same portfolio as we did in Part 2.

4A. Computing Statistics using Rolling Windows

We used the following code to generate our betas for 504 rolling windows. We used the following regression equation to estimate our three betas:

$$\tilde{r}_i - r_f = \alpha_i + \beta_{i,M}(\tilde{r}_M - r_f) + \beta_{i,SMB}\tilde{r}_{SMB} + \beta_{i,HML}\tilde{r}_{HML} + \tilde{\epsilon}_i$$

```
In [467]: beta_storage <- matrix(rep(0,30210), nrow = 30, ncol = 1007) #For storing betas. Every three rows represents the rolling betas for a given security
for (j in 0:99) {
  ri_minus_rf <- stock_returns[,rnd_stocks_ff][,(j+1)] - fdata[,4]
  rm_minus_rf <- fdata[,1] - fdata[,4]
  SMB <- fdata[,2]
  HML <- fdata[,3]
  beta_M <- c(rep(0,1007))
  beta_SMB <- c(rep(0,1007))
  beta_HML <- c(rep(0,1007))
  for (i in 1:1007) { #iterate through each rolling window
    ols <- lm(ri_minus_rf[i:(i+503)] ~ rm_minus_rf[i:(i+503)] + SMB[i:(i+503)] + HML[i:(i+503)]) #OLS to find betas at each rolling window
    beta_M[i] <- ols$coefficients[2] # assign coefficients
    beta_SMB[i] <- ols$coefficients[3]
    beta_HML[i] <- ols$coefficients[4]
  }
  beta_storage [(3*j+1),] <- beta_M # assign coefficients to broader beta_storage
  beta_storage [(3*j+2),] <- beta_SMB
  beta_storage [(3*j+3),] <- beta_HML
}
```

Through this code we were able to generate a matrix to store all of our beta values. Every three rows represent a set of betas for a given security, which is why there are 300 rows. For each row, there are 1007 columns, representing the 1007 rolling windows.

Next, we generated our variance matrices for this rolling windows. We stored each matrix in a list for easy retrieval. In this section we also have our code for generating idiosyncratic variances, similar to our process to part 3. Here, we directly called on the residuals of our OLS to give us our error terms and took the variance of them. This gave us a data set of 100 rows, each row representing a security, and their error variances across all 1007 rolling windows, hence the 1007 columns.

```
In [470]: cov_matrix_storage <- list() #this part is for building fama-french covariance matrices
for (i in 1:1007) {
  A <- matrix(beta_storage[,i], nrow=100, ncol = 3, byrow = TRUE)
  cov_matrix_storage[i] <- cov(A)
}

In [472]: idio_var_storage <- matrix(rep(0,100700), nrow = 100, ncol = 1007) #For storing idiosyncratic variances
for (j in 0:99) {
  ri_minus_rf <- stock_returns[,rnd_stocks_ff][,(j+1)] - fdata[,4]
  rm_minus_rf <- fdata[,1] - fdata[,4]
  SMB <- fdata[,2]
  HML <- fdata[,3]
  var_residuals <- c(rep(0,1007))
  for (i in 1:1007) {
    ols <- lm(ri_minus_rf[i:(i+503)] ~ rm_minus_rf[i:(i+503)] + SMB[i:(i+503)] + HML[i:(i+503)]) #perform OLS to obtain residuals
    var_residuals[i] <- var(ols$residuals) #find variance of residuals
  }
  idio_var_storage[j+1,] <- var_residuals
}
```

Next, we found the variances for all of the rolling windows by implementing the equation from part 3:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T \hat{\beta} \hat{\sigma}_M^2 \hat{\beta}^T w + w^T \hat{\Delta} w$$

```
In [473]: variance_storage <- c(rep(0,1007)) #obtain the variances for all rolling windows
for (i in 1:1007) {
  beta <- matrix(beta_storage[,i], nrow=100, ncol = 3, byrow = TRUE)
  w <- matrix(market_cap[(i+503),rnd_stocks_ff] / sum(market_cap[(i+503),rnd_stocks_ff]))
  delta <- diag(idio_var_storage[,i])
  variance_storage [i] <- t(w) %*% delta %*% w + t(w) %*% beta %*% cov_matrix_storage[[i]] %*% t(beta) %*% w
}
```

The 1007 variances are stored in the vector `variance_storage`.

We then used the following code to obtain the one-day ahead returns. We did so by normalizing the weights in the one-day ahead period so that they sum to one. The values were stored in `one_day_ahead_storage`.

```
In [463]: one_day_ahead_storage <- c(rep(0,1007)) #obtain one-day-ahead returns
for (i in 1:1007) {
  w <- market_cap[(i+504),rnd_stocks_ff] / sum(market_cap[(i+504),rnd_stocks_ff])
  stock_returns_ahead <- stock_returns[(i+504),rnd_stocks_ff]
  weighted_returns <- 0
  for (j in 1:100) {
    weighted_returns <- weighted_returns + w[j]* stock_returns_ahead[j]
  }
  one_day_ahead_storage[i] <- weighted_returns
}
```

Finally, we computed the bias statistics for the rolling window by finding the standard deviation of the standardized outcomes to get the bias statistics.

```
In [594]: standardized_outcome <- c(rep(0,1007))
for (i in 1:1007) {
  standardized_outcome[i] <- one_day_ahead_storage[i] / (variance_storage[i])^(1/2)
}
```

```
In [595]: var(standardized_outcome)^(1/2)
2.16216814281339
```

4B. Bias Statistics for Rolling Windows

Fama-French Model Bias Statistics

<u>Aa</u> Rolling Window	<u>#</u> Bias Statistic
<u>504</u>	2.162
<u>252</u>	1.811
<u>126</u>	1.574
<u>63</u>	1.456

We observe that the best bias statistics occurs at rolling window 63. This aligns with what we found with other methods, where the smallest rolling window yielded the lowest bias statistics. To reiterate, we believe that this is because the direction of

market returns is constantly changing, which makes historical data less accurate and obsolete and a worse predictor than current data. We also see that the bias statistics associated with the Fama-French model are noticeably higher than the biased statistics associated with MPT and market models. Among these three approaches, the one we personally prefer the most is the Fama-French model. We prefer the Fama-French model because it accounts for more factors compared to the other two models. Even though the Fama-French model has a higher bias statistics, we believe that its overall flexibility will benefit us more.

Part 5: Visual Test of the CAPM versus the Fama-French Model

To wrap everything up, we build 50 100-names portfolios of randomly chosen stocks and calculated statistics for each of the three dates 12/30/2005, 12/31/2007, and 12/31/2009.

Using our estimation results based on the market model, the Fama-French model, and the best rolling window in each case, we estimated the market beta and betas respective to the Fama-French factors of each of our portfolios.

```
Date: 12/30/2005 (idx loc = 503)

In [304]: betas63_q3_2005 = []
          for i in range(50):
              beta_m = betas_df_q3_63.iloc[63]
              weights = df_weightz[list(stocks.iloc[i])].iloc[63]
              betas_market = market_beta(weights, beta_m)
              betas63_q3_2005.append(betas_market)
          betas63_q3_2005[0:5]

Out[304]: [0.052476585476291554,
           0.060517796382025806,
           0.05266254554281719,
           0.04484296288982689,
           0.05008550819633388]

In [560]: weighted_market_cap_1005 <- market_cap[1005,random_stocks[1,]]/sum(market_cap[1005,random_stocks[1,]])

In [562]: weighted_market_cap_503 <- market_cap[503,random_stocks[1,]]/sum(market_cap[503,random_stocks[1,]])

In [561]: weighted_market_cap_1510 <- market_cap[1510,random_stocks[1,]]/sum(market_cap[1510,random_stocks[1,]])
```

We then used these betas statistics to estimate the mean excess return of each of our portfolios and each of the factors from the market model and the Fama-French model. The formulas are:

$$\bar{r}_p = \frac{1}{rw} \sum_{t=T-rw+1}^T (r_{p,t} - r_{f,t})$$

$$\bar{r}_M = \frac{1}{rw} \sum_{t=T-rw+1}^T (r_{M,t} - r_{f,t})$$

$$\bar{r}_{SMB} = \frac{1}{rw} \sum_{t=T-rw+1}^T r_{SMB,t}$$

$$\bar{r}_{HML} = \frac{1}{rw} \sum_{t=T-rw+1}^T r_{HML,t}$$

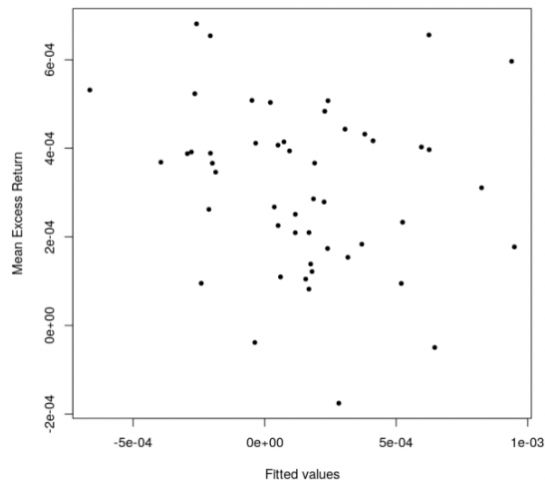
```
In [604]: beta_storage <- matrix(rep(0), nrow = 30, ncol = 1448) #For storing betas. Every three rows represents the rolling betas for
a given security
for (j in 0:9) {
  ri_minus_rf <- stock_returns[,random_stocks][,(j+1)] - fdata[,4]
  rm_minus_rf <- fdata[,1] - fdata[,4]
  SMB <- fdata[,2]
  HML <- fdata[,3]
  beta_M <- c(rep(0,1448))
  beta_SMB <- c(rep(0,1448))
  beta_HML <- c(rep(0,1448))
  for (i in 1:1448) { #iterate through each rolling window
    ols <- lm(ri_minus_rf[i:(i+62)] ~ rm_minus_rf[i:(i+62)] + SMB[i:(i+62)] + HML[i:(i+62)]) #OLS to find betas at each ro
lling window
    beta_M[i] <- ols$coefficients[2] # assign coefficients
    beta_SMB[i] <- ols$coefficients[3]
    beta_HML[i] <-ols$coefficients[4]
  }
  beta_storage [(3*j+1),] <- beta_M # assign coefficients to broader beta_storage
  beta_storage [(3*j+2),] <- beta_SMB
  beta_storage [(3*j+3),] <- beta_HML}
```

```
In [290]: rm_2005 = 1/63*np.sum((np.array(fdata['Market Returns'])[441:504] - np.array(fdata['Risk-free rate'])[441:504]))
rm_2005
```

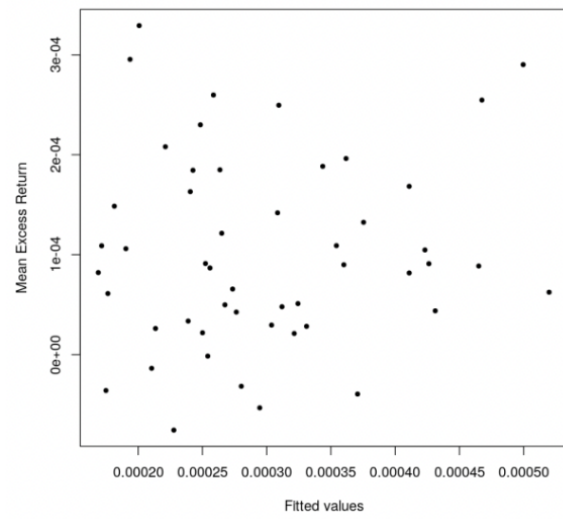
```
Out[290]: 8.365079365079375e-05
```

With those statistics, we plotted the mean excess returns of our portfolios against their fitted values from the CAPM model in one figure and the mean excess returns of your portfolios against their fitted values from the Fama-French model.

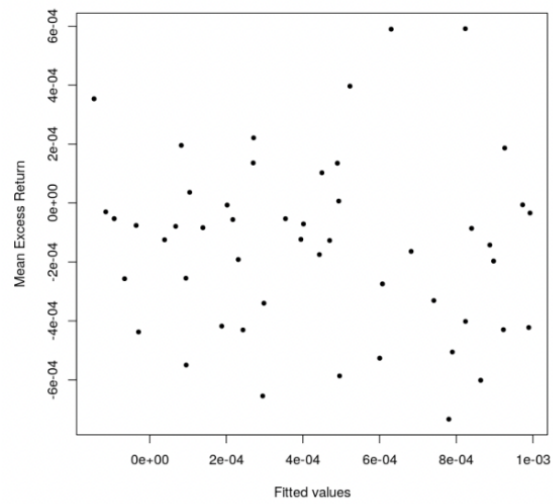
Market Model 12/31/2009 Fitted Values vs. Mean Excess Return



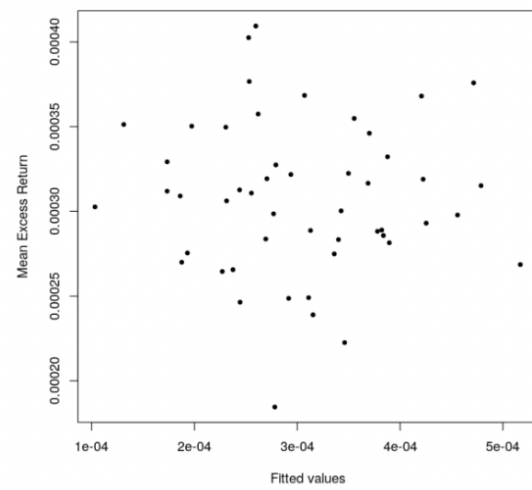
Fama Fench 12/31/2009 Fitted Values vs. Mean Excess Return

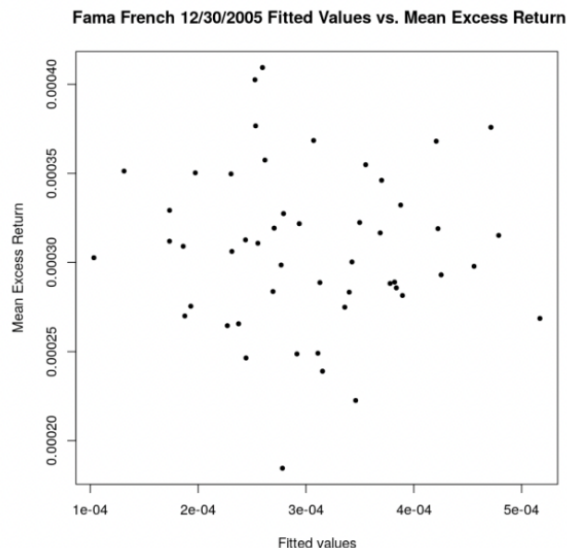
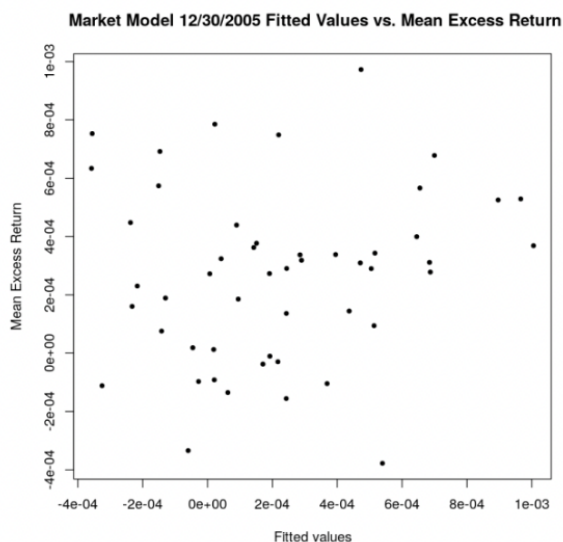


Market Model 12/31/2007 Fitted Values vs. Mean Excess Return



Fama French 12/30/2005 Fitted Values vs. Mean Excess Return





Upon scrutinizing the data, we realize two common characteristics. The Fama-French's mean excess return tends to deviate further from 0 compared to the market model, and the Fama-French model also has lower variance compared to the market model.

With regards to deviating from 0, we observe that the Fama-French model tends to deviate positively (i.e. the Fama-French model on average has more positive mean excess returns compared to the model). This is likely due to the fact that the Fama-French model is able to capture unrealized gains by factoring in impacts that are size related and value related. However, the positive mean excess return does not hold for every one of these charts.

We see that in the 2007 chart, the Fama-French had more negative excess returns compared to the market model. We theorize that this is likely due to the volatility in the market that occurred around 2007 due to the financial crisis. The lower variance of the Fama-French model, on the other hand, is fairly consistent across all of the time periods. Similarly, the high variance of the market model is also fairly consistent across all of the time periods. We theorize that this is because the Fama-French model has more parameters, which enable a more precise fit compared to the market model.

We use the same reason to explain why the Fama-French tends to deviate more than the market model. Due to bias-variance tradeoff, since Fama-French has less variance,

its bias is higher. In addition to being apparent in graphs, this high bias is also apparent from what we see in the bias statistics.

Overall, we prefer the Fama-French model to the market model for the following reasons:


1. The Fama-French model has more parameters which enables it to account for more factors that the market model cannot. This can lead to more precisely fitted value, as we see in the graphs.
2. While the Fama-French model is more biased, we believe that bias can be beneficial especially when the market is on the rise.
3. As more risk-seeking investors we are willing to take a risk with the more precise but more biased Fama-French rather than the less precise but less biased capital market.

Appendix

All data and code utilized in this project can be found on our GitHub repository:

anhtulu12/econ139-f20

Final Project. Contribute to anhtulu12/econ139-f20 development by creating an account on GitHub.

 <https://github.com/anhtulu12/econ139-f20>

All of our code can also be found below.