Appendix

Data Processing

This script imports and updates all raw portfolio returns, Fama-French, BMG, Momentum, daily,and monthly data from assignment and Kenneth R. French sources.

```
# This completes the data processing for the BMG Empirical Assignment

# Imports important python packages
import numpy as np # arithmetic operations
import pandas as pd # data analysis package
import csv as csv # read and write csvs
import random as rd # random functionality
import saspy as sas # Use saspy functionality in python
import matplotlib.pyplot as plt # Use MatLab functionality for plotting
import seaborn as sb # Imports seaborn library for use
import wrds as wrds# Wharton Research Data Services API
import pydatastream as pds # Thomas Reuters Datastream API
import yfinance as yf # Yahoo Finance API
import datetime as dt # Manipulate datetime values
import statsmodels.api as sm # Create Stats functionalities
import sklearn as sl # ML functionality
from stargazer.stargazer import Stargazer
import finance_byu as fin # Python Package for Fama-MacBeth Regressions


# Creates dataframes to convert data
# Daily
pd_df = pd.read_excel('data.xlsx', sheet_name = 'portfolio_daily')
ffd_df = pd.read_excel('data.xlsx', sheet_name = 'fama_french_daily')
bmgd_df = pd.read_excel('data.xlsx', sheet_name = 'bmg_daily')
da_df = pd.read_excel('data.xlsx', sheet_name = 'daily_all')

# Monthly
pm_df = pd.read_excel('data.xlsx', sheet_name = 'portfolio_monthly')
ffm_df = pd.read_excel('data.xlsx', sheet_name = 'fama_french_monthly')
bmgm_df = pd.read_excel('data.xlsx', sheet_name = 'bmg_monthly')
ma_df = pd.read_excel('data.xlsx', sheet_name = 'monthly_all')

# Converts fama-french factors from percentages to fractions
# Daily
ffd_df['mktrf'] = ffd_df['mktrf']/100
ffd_df['smb'] = ffd_df['smb']/100
ffd_df['hml'] = ffd_df['hml']/100
ffd_df['rf'] = ffd_df['rf']/100
ffd_df['umd'] = ffd_df['umd']/100

# Monthly
ffm_df['rf'] = ffm_df['rf']/100

# Convert data columns from timestamps to datatime to enable matching
pd_df['date'] = pd.to_datetime(pd_df['date'], unit='s')
ffd_df['date'] = pd.to_datetime(ffd_df['date'], unit='s')
bmgd_df['date'] = pd.to_datetime(bmgd_df['date'], unit='s')
da_df['date'] = pd.to_datetime(da_df['date'], unit='s')

# Creates to dataframes
```

```
uda_df = da_df.copy()
uma_df = ma_df.copy()

# Daily adjustments and additions
# Updates umd and bmg prior to adding new additions
for index, row in uda_df.iterrows():
    date = uda_df.at[index,'date']
    try:
        # Gets the factors
        factors_df = ffd_df.loc[ffd_df['date']== date]
        # print(factors_df.head())
        bmg_df = bmgd_df.loc[bmgd_df['date']== date]
        # print(bmg_df.head())
        # Changes the value
        uda_df.at[index,'umd']= factors_df.iloc[0]['umd']
        uda_df.at[index,'bmg'] = bmg_df.iloc[0]['bmg']
    except:
        print("Error updating the umd and bmg foactor (2010 - 2016)")

# Add the portofilio returns data to the updated dataframes
# Sets sequence for portfolio returns
portfolios = list(range(1,31))
for index, row in pd_df.iterrows():
    # Set the time period information
    year = pd_df.at[index,'year']
    month = pd_df.at[index,'month']
    day = pd_df.at[index,'day']
    date = pd_df.at[index,'date']
    # Set the factor elements based on dates with index matching from
    dataframes
    # Locates the factors at the required date
    # Try statement to skip entries when portfolio, factor and bmg
    dates don't align.
    try:
        factors_df = ffd_df.loc[ffd_df['date']== date]
        bmg_df = bmgd_df.loc[bmgd_df['date']== date]
        mktrf = factors_df.iloc[0]['mktrf']
        smb = factors_df.iloc[0]['smb']
        hml = factors_df.iloc[0]['hml']
        rf = factors_df.iloc[0]['rf']
        umd = factors_df.iloc[0]['umd']
        bmg = bmg_df.iloc[0]['bmg']
        # Add the portfolio components
        for portfolio in portfolios:
            ret = pd_df.at[index,portfolio]
            # Creates dataframe to append
            d = {'ind': [portfolio] , 'ret': [ret],'year': [year],'
    month': [month],'day': [day],'date': [date],'mktrf': [mktrf],'smb':
    [smb],'hml': [hml],'rf': [rf],'umd': [umd],'bmg': [bmg]}
            row_df = pd.DataFrame(data=d)
            # Append to dataframe (use assignment)
            uda_df = uda_df.append(row_df,ignore_index= True)
    except:
        # Documents date omissions
        print("Warning - Error")
        line = date.strftime("%m/%d/%Y, %H:%M:%S")
        with open('omissions-daily.txt', 'a+') as f:
            f.seek(0)
```

```
            data = f.read(100)
            if len(data) > 0 :
                f.write("\n")
                # Append text at the end of file
                f.write(line)
# Create new csv file
uda_df.to_csv('updated_daily_all.csv')

# Monthly adjustments and additions
# Add the portofilio returns data to the updated dataframes
# Sets sequence for portfolio returns
portfolios = list(range(1,31))
for index, row in pm_df.iterrows():
    # Set the time period imformation
    year = pm_df.at[index,'year']
    month = pm_df.at[index,'month']
    day = pm_df.at[index,'day']
    date = pm_df.at[index,'date']
    eom = pm_df.at[index,'eom']
    # Set the factor elements based on dates with index matching from dataframes
    # Locates the factors at the required date
    # print("index: ",index)
    # print("date: ",date)
    # Try statement to skip entries when portfolio, factor and bmg dates don't align.
    try:
        factors_df = ffm_df.loc[ffm_df['eom']== eom]
        rf = factors_df.iloc[0]['rf']
        # Add the portfolio components
        for portfolio in portfolios:
            ret = pm_df.at[index,portfolio]
            # Creates dataframe to append
            d = {'year': [year],'month': [month],'day': [day],'date': [date], 'eom': [eom], 'ind': [portfolio] , 'ret': [ret], 'rf':[rf]}
            row_df = pd.DataFrame(data=d)
            # Append to dataframe (use assignment)
            uma_df = uma_df.append(row_df,ignore_index= True)
    except:
        # Documents date omissions
        print("Warning - Error")
        line = date.strftime("%m/%d/%Y, %H:%M:%S")
        with open('omissions-monthly.txt', 'a+') as f:
            f.seek(0)
            data = f.read(100)
            if len(data) > 0 :
                f.write("\n")
                # Append text at the end of file
                f.write(line)
# Create new csv file
uma_df.to_csv('updated_monthly_all.csv')
uma_df.to_excel('updated_monthly_all.xlsx')
```

Data Analysis

This script implements all data analysis performed in the assignment.

```
# This completes the data analysis for the BMG Empirical Assignment
# Note: Data is processed using the finance-761-data-processing script

# Imports important python packages and data from data prcocessing
import numpy as np # arithmetic operations
import pandas as pd # data analysis package
import csv as csv # read and write csvs
import random as rd # random functionality
import saspy as sas # Use saspy functionality in python
import matplotlib.pyplot as plt # Use MatLab functionality for plotting
import seaborn as sb # Imports seaborn library for use
import wrds as wrds# Wharton Research Data Services API
import pydatastream as pds # Thomas Reuters Datastream API
import yfinance as yf # Yahoo Finance API
import datetime as dt # Manipulate datetime values
import statsmodels.api as sm # Create Stats functionalities
import linearmodels as lp # Ability to use PooledOLS
import sklearn as sl # ML functionality
from stargazer.stargazer import Stargazer
import finance_byu as fin # Python Package for Fama-MacBeth Regressions
from statsmodels.regression.rolling import RollingOLS # Use factor
    loadings
from stargazer.stargazer import Stargazer
import sympy as sy # convert latex code
import scipy as sc # Scipy packages
import tabulate as tb # Create tables in python

# Establishes plotting setting for rolling regressions
sb.set_style('darkgrid')
pd.plotting.register_matplotlib_converters()

# Reads data csvs as dataframes
daily_all_df = pd.read_csv("updated_daily_all.csv")
monthly_all_df = pd.read_csv("updated_monthly_all.csv")


# Creates excess return variable for both daily and monthly
daily_all_df['eret'] = daily_all_df['ret']/100 - daily_all_df['rf'] #
    Check if this step is necessary
monthly_all_df['eret'] = monthly_all_df['ret']/100 - monthly_all_df['rf
    '] # Check if this step is necessary

# Creates month variables to both the daily and monthly sets
daily_all_df['m'] = daily_all_df['month'] + daily_all_df['year']*12
monthly_all_df['m'] = monthly_all_df['month'] + monthly_all_df['year'
    ]*12

daily_all_df.to_excel('excel/daily_all_df_excel_check.xlsx')
monthly_all_df.to_excel('excel/monthly_all_df_excel_check.xlsx')

# Correctly sorts the columns
daily_all_df.sort_values(by=['ind','year','month'],ascending=True,
    inplace=True)

# Creates a unique list of month values
```

```
m_list = sorted(np.unique(daily_all_df['m']))
ind_list = sorted(np.unique(daily_all_df['ind']))

# Shifted for Stage 2 of the Fama MacBeth Regression
# monthly_all_df['eret'] = (monthly_all_df.sort_values(by=['m'], ascending=True)
#                         .groupby(['ind'])['eret'].shift(-1))

# Drop NaN Datas
monthly_all_df = monthly_all_df.dropna(axis=0, how = 'any')

# Start Fama-Macbeth Regressions

# This is Stage 1 of the Fama-Macbeth Regression - Estimate Factor Loadings (Crossed Checked)
# https://en.wikipedia.org/wiki/Fama%E2%80%93MacBeth_regression
# Create a new dataframe with every possible combination of month and index combinations available
factor_df = pd.DataFrame(columns=['ind','m', 'mktrf','smb','hml','umd', 'bmg'])
for i in ind_list:
    for j in m_list:
        # Loops over factor dataframe to get the desired values
        # Get slice of dataframe based on multiple columns
        index_df = daily_all_df[daily_all_df['ind'] == i]
        slice_df = index_df[index_df['m'] == j]
        # Perform the OLS Regressions on the sliced dataframne
        y = slice_df['eret']
        x = slice_df[['mktrf','smb','hml','umd','bmg']]
        x = sm.add_constant(x)
        model =sm.OLS(y,x).fit()
        # Save model parameters to column dataframe
        new_row = {'ind':i,'m':j, 'mktrf':model.params[1],'smb':model.params[2],'hml':model.params[3],'umd':model.params[4],'bmg':model.params[5]}
        # Append factor loading to the factors to the dataframe
        factor_df = factor_df.append(new_row, ignore_index = True)

# Produce average industry beta
average_ffb_df = pd.DataFrame(columns=['m','bmg'])
for m in m_list:
    cross_sectional_df = factor_df[factor_df['m'] == m]
    new_row = {'m':m,'bmg':cross_sectional_df['bmg'].mean()}
    print(new_row)
    average_ffb_df = average_ffb_df.append(new_row, ignore_index= True)


# PLot average ffb
average_ffb_df.plot(x='m', y=['bmg'], kind='line', figsize=(28,18))
plt.title('Average Industry BMG Time Series', fontsize = 30)
plt.ylabel('BMG', fontsize = 24)
plt.xlabel('Time (Date)', fontsize = 24)
plt.legend(loc=2, prop={'size': 18})
plt.xticks(size = 18)
plt.yticks(size = 18)
# Add caption to below plot python
plt.savefig('plots/bmg-time-series-premium.png')

```

```
raw_fama_macbeth_df = pd.merge(monthly_all_df, factor_df[['ind','m','mktrf','smb','hml','umd','bmg']].copy(),  how='right', left_on=['ind','m'], right_on = ['ind','m'])
raw_fama_macbeth_df.to_excel('excel/raw_fama_macbeth_df.xlsx')


# # Shift factors forward by one value (month)
factor_df['mktrf'] = (factor_df.sort_values(by=['m'], ascending=True)
                        .groupby(['ind'])['mktrf'].shift(1))
factor_df['smb'] = (factor_df.sort_values(by=['m'], ascending=True)
                        .groupby(['ind'])['smb'].shift(1))
factor_df['hml'] = (factor_df.sort_values(by=['m'], ascending=True)
                        .groupby(['ind'])['hml'].shift(1))
factor_df['umd'] = (factor_df.sort_values(by=['m'], ascending=True)
                        .groupby(['ind'])['umd'].shift(1))
factor_df['bmg'] = (factor_df.sort_values(by=['m'], ascending=True)
                        .groupby(['ind'])['bmg'].shift(1))

# Drop NaN Datas
factor_df = factor_df.dropna(axis=0, how = 'any')
# Merge the regression co-efficients with monthly dataset
fama_macbeth_df = pd.merge(monthly_all_df, factor_df[['ind','m','mktrf','smb','hml','umd','bmg']].copy(),  how='right', left_on=['ind','m'], right_on = ['ind','m'])
# Drops rows with NaN
fama_macbeth_df = fama_macbeth_df.dropna(axis=0, how = 'any')
# Saves Fama-Macbeth Regression Results to Excell
fama_macbeth_df.to_excel('excel/processed_fama_macbeth_df.xlsx')


time_series_df = pd.DataFrame(columns=['ind','value','alpha', 'mktrf','smb','hml','umd','bmg'])
# Work out the Fama-French Time series
for i in ind_list:
    # Gets section of dataframe for monthly date (i.e. days for ind x )
    cross_sectional_df = daily_all_df[daily_all_df['ind'] == i]
    # Performs regression using the cross section to get factor price
    y = cross_sectional_df['eret']
    x = cross_sectional_df[['mktrf','smb','hml','umd','bmg']]
    # print(cross_sectional_df.head(n=30))
    # input("Press Enter to continue...")
    x = sm.add_constant(x)
    model =RollingOLS(y,x).fit()
    model =sm.OLS(y,x).fit()
    # Save model parameters to column dataframe
    new_row_coef = {'ind':i,'value':'co-efficient','alpha': model.params[0],'mktrf':model.params[1],'smb':model.params[2],'hml':model.params[3],'umd':model.params[4],'bmg':model.params[5]}
    # Save model pvalues
    new_row_pvalue = {'ind':i,'value':'pvalue','alpha': model.pvalues[0],'mktrf':model.pvalues[1],'smb':model.pvalues[2],'hml':model.pvalues[3],'umd':model.pvalues[4],'bmg':model.pvalues[5]}
    # new_row_se = {'m':i,'alpha': model.std_errors[0],'mktrf':model.std_errors[1],'smb':model.std_errors[2],'hml':model.std_errors[3],'umd':model.std_errors[4],'bmg':model.std_errors[5]}
    # Append the new row to the dataframe
    time_series_df = time_series_df.append(new_row_coef, ignore_index = True)
```

```
    time_series_df = time_series_df.append(new_row_pvalue, ignore_index = True)
    # factor_price_se_df = factor_price_se_df.append(new_row_se, ignore_index = True)
    # Append models to list
    stargazer = Stargazer([model])
    stargazer.custom_columns('FF Time-series-'+ str(i))
    expr = stargazer.render_latex()
    sy.preview(expr, viewer='file', filename='time-series/' + str(i)+'-regression.png')

# Prints the time-series (all months) accross industries
time_series_df.to_excel('excel/time-series-regression-table.xlsx')


# This is Stage 2 of the Fama-Macbeth Regression - Estimate Factor Prices from Monthly Data
# Create factor pricing dataframe
factor_price_df = pd.DataFrame(columns=['m', 'alpha', 'mktrf','smb','hml','umd','bmg'])
factor_price_se_df = pd.DataFrame(columns=['m', 'alpha', 'mktrf','smb','hml','umd','bmg'])

# Update the m list as excludes the first month
m_list = sorted(np.unique(fama_macbeth_df['m']))

# Run cross-sectional regression for each time period using monthly data
for i in m_list:
    # Gets section of dataframe for monthly date (i.e. ind 1-30 for month x )
    cross_sectional_df = fama_macbeth_df[fama_macbeth_df['m'] == i]
    # Performs regression using the cross section to get factor price
    y = cross_sectional_df['eret']
    x = cross_sectional_df[['mktrf','smb','hml','umd','bmg']]
    # print(cross_sectional_df.head(n=30))
    # input("Press Enter to continue...")
    x = sm.add_constant(x)
    # model = sm.OLS(endog = y,exog = x).fit()
    model =sm.OLS(y,x,).fit()
    # Save model parameters to column dataframe
    new_row_price = {'m':i,'alpha': model.params[0],'mktrf':model.params[1],'smb':model.params[2],'hml':model.params[3],'umd':model.params[4],'bmg':model.params[5]}
    # new_row_se = {'m':i,'alpha': model.std_errors[0],'mktrf':model.std_errors[1],'smb':model.std_errors[2],'hml':model.std_errors[3],'umd':model.std_errors[4],'bmg':model.std_errors[5]}
    # Append the new row to the dataframe
    factor_price_df = factor_price_df.append(new_row_price, ignore_index = True)
    # factor_price_se_df = factor_price_se_df.append(new_row_se, ignore_index = True)
    # Append models to list
    stargazer = Stargazer([model])
    expr = stargazer.render_latex()
    sy.preview(expr, viewer='file', filename='statistical-tables/' + str(i)+'-regression.png')

```

```
# Converts Factor Prices to Excel
factor_price_df.to_excel('excel/ffb_stage_2_df.xlsx')

# Plot the dataframe
factor_price_df.plot(x='m', y=['bmg'], kind='line', figsize=(28,18))
plt.title('Average Industry BMG Risk Premium Time Series')
plt.ylabel('BMG Risk Premium', fontsize = 24)
plt.xlabel('Time (Date)', fontsize = 24)
plt.legend(loc=2, prop={'size': 18})
plt.xticks(size = 18)
plt.yticks(size = 18)
# Add caption to below plot python
plt.savefig('plots/bmg-risk-premium.png')

# This is Stage 3 of the Fama-Macbeth Regression - Estimate average factor pricing and error ()
# Calculate estimated factor prices across all time periods
factor_prices_average_dict = {'alpha': factor_price_df['alpha'].mean(), 'mktrf':factor_price_df['mktrf'].mean(),'smb':factor_price_df['smb'].mean(),'hml':factor_price_df['hml'].mean(),'umd':factor_price_df['umd'].mean(),'bmg':factor_price_df['bmg'].mean()}
# factor_prices_se_average_dict = {'alpha': factor_price_se_df['alpha'].mean(),'mktrf':factor_price_se_df['mktrf'].mean(),'smb':factor_price_se_df['smb'].mean(),'hml':factor_price_se_df['hml'].mean(),'umd':factor_price_se_df['umd'].mean(),'bmg':factor_price_se_df['bmg'].mean()}

# Calculates unbiased standard error of the mean over requested axis
# https://www.geeksforgeeks.org/python-pandas-dataframe-sem/
factor_prices_sem_average_dict = {'alpha': factor_price_df['alpha'].sem(),'mktrf':factor_price_df['mktrf'].sem(),'smb':factor_price_df['smb'].sem(),'hml':factor_price_df['hml'].sem(),'umd':factor_price_df['umd'].sem(),'bmg':factor_price_df['bmg'].sem()}

# Print dictionaries to display factor prices and standard errors
alpha_mean = factor_price_df['alpha'].mean()
mktrf_mean = factor_price_df['mktrf'].mean()
smb_mean = factor_price_df['smb'].mean()
hml_mean = factor_price_df['hml'].mean()
umd_mean = factor_price_df['umd'].mean()
bmg_mean = factor_price_df['bmg'].mean()

# Performs one sample ttest on all variables
bmg_tstat,bmg_pvalue = sc.stats.ttest_1samp(a=factor_price_df['bmg'], popmean=factor_price_df['bmg'].mean())
mktrf_tstat,mktrf_pvalue = sc.stats.ttest_1samp(a=factor_price_df['mktrf'], popmean=factor_price_df['mktrf'].mean())
smb_tstat,smb_pvalue = sc.stats.ttest_1samp(a=factor_price_df['smb'], popmean=factor_price_df['smb'].mean())
hml_tstat,hml_pvalue = sc.stats.ttest_1samp(a=factor_price_df['hml'], popmean=factor_price_df['hml'].mean())
umd_tstat,umd_pvalue = sc.stats.ttest_1samp(a=factor_price_df['umd'], popmean=factor_price_df['umd'].mean())
alpha_tstat,alpha_pvalue = sc.stats.ttest_1samp(a=factor_price_df['alpha'], popmean=factor_price_df['alpha'].mean())

# Create dataframe
head = ['name', 'mean', 'tstat','pvalue']
```

```
names = ['alpha', 'mktrf', 'smb','hml','umd','bmg']
means = [alpha_mean, mktrf_mean, smb_mean, hml_mean, umd_mean, bmg_mean]
tstats = [alpha_tstat, mktrf_tstat, smb_tstat, hml_tstat, umd_tstat, bmg_tstat]
pvalues = [alpha_pvalue, mktrf_pvalue, smb_pvalue, hml_pvalue, umd_pvalue, bmg_pvalue]

ffb_statistics = pd.DataFrame(columns=[head[0], head[1], head[2],head[3]])
# For loop to create
for i in range(len(names)):
    new_row = {head[0]:names[i],head[1]:means[i], head[2]:tstats[i], head[3]:pvalues[i]}
    ffb_statistics = ffb_statistics.append(new_row, ignore_index = True)
# Save to CSV
ffb_statistics.to_excel('excel/ffb_statistics.xlsx')

# Additional Analysis 1: Rolling Regression
# Implements Rolling Regressions for each industry (1-30) rolling through months in regressing
# https://www.statsmodels.org/dev/examples/notebooks/generated/rolling_ls.html
exog_vars = ['mktrf','smb','hml','umd','bmg']
for i in ind_list:
    cross_sectional_df = daily_all_df[daily_all_df['ind'] == i]
    # Create eret dataframe
    eret_df = cross_sectional_df[['date','eret']].copy()
    exog = sm.add_constant(cross_sectional_df[exog_vars])
    rols = RollingOLS(eret_df['eret'], exog, window=len(m_list))
    rres = rols.fit()
    fig = rres.plot_recursive_coefficient(variables=exog_vars, figsize=(14,18))
    path = "rolling-regressions/" + str(i) + "-rolling-regression.png"
    plt.savefig(path)

# Additional Analysis 2: Hedging Positions
# Implements Hedging Portfolio based on BMG rankings on the first date (Monthly)
# Imports S&P 500 Data
sp500_df = pd.read_excel('sp500.xlsx', sheet_name = 'sp500')

ranking_df = pd.DataFrame(columns=['ind','mean'])
# Sets bmg ranking from fama_macbeth
for i in ind_list:
    # This is an index
    index_rank_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
    new_row = {'ind':i, 'mean': index_rank_df['bmg'].mean()}
    ranking_df = ranking_df.append(new_row,ignore_index = True)
bmg_good_df = ranking_df.sort_values(by ='mean',ascending = True).head(5)
bmg_bad_df = ranking_df.sort_values(by = 'mean',ascending = True).tail(5)
# Create green list (Top 5, Green Stocks)
bmg_good = bmg_good_df['ind'].to_list()
# Create brown list (Bottom 5, Brown Stocks)
bmg_bad = bmg_bad_df['ind'].to_list()
```

```
# Create new dateframes with Python
good_returns = pd.DataFrame(columns=['m',str(bmg_good[0]),str(bmg_good[1]),str(bmg_good[2]),str(bmg_good[3]),str(bmg_good[4])])
bad_returns = pd.DataFrame(columns=['m',str(bmg_bad[0]),str(bmg_bad[1]),str(bmg_bad[2]),str(bmg_bad[3]),str(bmg_bad[4])])

# Starts cumulative returns calculations
# Top 5 Green Stocks
for j in m_list:
    # Empty list to append to
    emp = []
    for i in bmg_good:
        #Gets the slicesd row
        index_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
        slice_df = index_df[index_df['m'] == j]
        idx = slice_df.loc[slice_df['ind'] == i].index
        emp.append(slice_df.at[idx[0],'ret'])
        # emp.append(slice_df.at[idx[0],'ret'] - slice_df.at[idx[0],'rf'])*100)
    # Append new row of returns data
    new_row = {'m':j,str(bmg_good[0]): emp[0],str(bmg_good[1]): emp[1], str(bmg_good[2]): emp[2], str(bmg_good[3]): emp[3],str(bmg_good[4]): emp[4]}
    good_returns = good_returns.append(new_row, ignore_index = True)

# Create equally weighting returns from the columns
good_returns['ret'] = ((good_returns[str(bmg_good[0])] + good_returns[str(bmg_good[1])] + good_returns[str(bmg_good[2])] + good_returns[str(bmg_good[3])] + good_returns[str(bmg_good[4])])/len(bmg_good))/100

# Calculate cumulative returns dataframe
for index, row in good_returns.iterrows():
    if index == 0:
        good_returns.at[index,'cr'] = 0
    if index > 0:
        good_returns.at[index,'cr'] = ((1+good_returns.at[index,'ret'])*(1+good_returns.at[index-1,'cr']))-1


good_returns.to_excel('excel/bmg_green.xlsx')
# Top 5 Brown Stocks
for j in m_list:
    # Empty list to append to
    emp = []
    for i in bmg_bad:
        #Gets the slicesd row
        index_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
        slice_df = index_df[index_df['m'] == j]
        idx = slice_df.loc[slice_df['ind'] == i].index
        emp.append(slice_df.at[idx[0],'ret'])
        # emp.append(slice_df.at[idx[0],'ret'] - slice_df.at[idx[0],'rf'])*100)
    # Append new row of returns data
    new_row = {'m':j,str(bmg_bad[0]): emp[0],str(bmg_bad[1]): emp[1], str(bmg_bad[2]): emp[2], str(bmg_bad[3]): emp[3],str(bmg_bad[4]): emp[4]}
    bad_returns = bad_returns.append(new_row, ignore_index = True)
```

```
325
326 # Create equally weighting returns from the columns
327 bad_returns['ret'] = ((bad_returns[str(bmg_bad[0])] + bad_returns[str(
    bmg_bad[1])] + bad_returns[str(bmg_bad[2])] + bad_returns[str(
    bmg_bad[3])] + bad_returns[str(bmg_bad[4])])/len(bmg_bad))/100
328
329 # Calculate cumulative returns dataframe
330 for index, row in bad_returns.iterrows():
331     if index == 0:
332         bad_returns.at[index,'cr'] = 0
333     if index > 0:
334         bad_returns.at[index,'cr'] = ((1+bad_returns.at[index,'ret'])
    *(1+bad_returns.at[index-1,'cr']))-1
335
336 # Saves bad_returns to excel
337 bad_returns.to_excel('excel/bmg_brown.xlsx')
338 # Create the hedge cumulative returns
339 hedge_ret_df = good_returns['m'].copy()
340 hedge_ret_df = pd.merge(hedge_ret_df, good_returns[['m','ret']],  how='
    left', left_on=['m'], right_on = ['m'])
341 hedge_ret_df = hedge_ret_df.rename(columns = {'ret':'bmg_green_ret'})
342 hedge_ret_df = pd.merge(hedge_ret_df, bad_returns[['m','ret']],  how='
    left', left_on=['m'], right_on = ['m'])
343 hedge_ret_df = hedge_ret_df.rename(columns = {'ret':'bmg_brown_ret'})
344 hedge_ret_df['hedge_ret'] = hedge_ret_df['bmg_green_ret'] -
    hedge_ret_df['bmg_brown_ret']
345 for index, row in hedge_ret_df.iterrows():
346     if index == 0:
347         hedge_ret_df.at[index,'cr'] = 0
348     if index > 0:
349         hedge_ret_df.at[index,'cr'] = ((1+hedge_ret_df.at[index,'
    hedge_ret'])*(1+hedge_ret_df.at[index-1,'cr']))-1
350
351 # Plot the cumulartive returns
352 bad_returns.plot(x ='m', y='cr', kind = 'line')
353 plt.savefig('plots/bad_bmg_returns.png')
354
355 # Sets cumulative returns calculation
356 green_cr = good_returns[['m','cr']].copy()
357 green_cr = green_cr.rename(columns = {'cr':'bmg_green'})
358 brown_cr = bad_returns[['m','cr']].copy()
359 brown_cr = brown_cr.rename(columns = {'cr':'bmg_brown'})
360 sp500_cr = sp500_df[['m','cr']].copy()
361 sp500_cr = sp500_cr.rename(columns = {'cr':'sp500'})
362 hedge_cr = hedge_ret_df[['m','cr']].copy()
363 hedge_cr = hedge_cr.rename(columns = {'cr':'hedge'})
364
365 # hedge = pd.DataFrame(columns=['m','bmg_good','bmg_bad','sp500'])
366 hedge_df = good_returns['m'].copy()
367
368 # Merge dataframes
369 hedge_df = pd.merge(hedge_df, green_cr,  how='left', left_on=['m'],
    right_on = ['m'])
370 hedge_df = pd.merge(hedge_df, brown_cr,  how='left', left_on=['m'],
    right_on = ['m'])
371 hedge_df = pd.merge(hedge_df, sp500_cr,  how='left', left_on=['m'],
    right_on = ['m'])
```

```
hedge_df = pd.merge(hedge_df, hedge_cr,  how='left', left_on=['m'], right_on = ['m'])
hedge_df.to_excel('excel/hedge_df.xlsx')

# Plot the dataframe
hedge_df.plot(x='m', y=['bmg_brown', 'bmg_green', 'hedge','sp500'], kind='line', figsize=(28,14))
plt.title('Hedging - BMG Green, BMG Brown, S&P 500', fontsize = 30)
plt.ylabel('Cumulative Return', fontsize = 24)
plt.xlabel('Time (m)', fontsize = 24)
plt.xticks(size = 18)
plt.yticks(size = 18)
plt.legend(loc=2, prop={'size': 18})
# Add caption to below plot python
plt.savefig('plots/hedging.png')

# Additional Analysis 3: Perform PooledOLS for event study
# Paris Agreement (24192) and Trump Election (24203)
# Sort the dataframe into the needed sections
pre_paris_df = daily_all_df[daily_all_df['m'] < 24192]
print(pre_paris_df.tail())
post_paris_df = daily_all_df[daily_all_df['m'] >= 24192]
print(post_paris_df.head())
pre_trump_df = daily_all_df[daily_all_df['m'] < 24203]
print(pre_trump_df.tail())
post_trump_df = daily_all_df[daily_all_df['m'] >= 24203]
print(post_trump_df.head())

# Reindex dataframes for PooledOLS
pre_paris_df = pre_paris_df.set_index(['m','ind'])
post_paris_df = post_paris_df.set_index(['m','ind'])
pre_trump_df = pre_trump_df.set_index(['m','ind'])
post_trump_df = post_trump_df.set_index(['m','ind'])

# Events
events = [pre_paris_df, post_paris_df, pre_trump_df, post_trump_df]
names = ['Pre-Paris-Agreement-(before-Dec-2015)', 'Post-Paris-Agreement-(Dec-2015-onwards)', 'Pre-Trump-Election-(before-Nov-2016)', 'Post-Trump-Election-(Nov-2016-onwards)']
name_count = 0
# Runs PooledOLS for
for ev in events:
    # Performs PooledOLS
    endo = ev['eret']
    exog = ev[['mktrf','smb','hml','umd','bmg']]
    exog = sm.add_constant(exog)
    model = lp.PooledOLS(endo, exog).fit(cov_type='clustered', cluster_entity=True)
    print(model, file=open("event-study/"+ names[name_count]+"pooledOLS.txt", "w"))
    name_count = name_count + 1
```