0_FF3_analysis_ver2.1

March 24, 2021

[1]:
```python
from datetime import datetime

import pandas as pd
import numpy as np
from scipy.stats import moment, mode

# Plotting related
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'figure.figsize':(20, 7.5)})
```

1 Volatility-Managed Portfolios

1.1 Part 0:

A. Read the data

1. Daily Fama-French 3 factor data from: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_l
2. Monthly Fama-French 3 factor data also from: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french
3. Monthly NBER based recession Indicators for the United States from: https://fred.stlouisfed.org/series/USREC
4. Cross validate the months across all the dataframes
5. Put the NBER recession indicator together with the monthy FF3 dataframe

B. Primary investigation of the data

1. Visualization (series and distribution)

C. Statistical analysis of the daily data clustered on monthly basis

1. Calculate min, max, median, mean, std, variance, 3-moment, cube-root of 3-moment, 4-moment, quad-root of 4-moment, 5-moment, pent-root of 5-moment, skewness and kurtosis for each month
2. Visualization of these Note: In this stage, I considered only the absolute values (need to be discussed)

D. Expanatory strength analysis of each monthly statistics (lagged) towards predicting next month's (current) statistics and return

1. Visualization of current month's return and statistics(median, mean, std, variance, 3-moment, cube-root of 3-moment, 4-moment, quad-root of 4-moment, 5-moment, pent-root of 5-moment, skewness and kurtosis) against previous month's statistics

E. Calculate the scaled monthly return

1. Scale the monthly returns using all the statistics found in step C and add to the monthly dataframe as separate columns
2. Visualization of these scaled returns

F. Bagging: Splitting the monthly dataframes with respect to the lagged statistic

1. Visualize the intuition behind the Moreira paper
2. Compare with the result they provided
3. For a detailed analysis, number of bags are not restricted to 5. I consider [5,15] bags.

G. Save the monthly preprocessed factors

1.1.1 A. Read the data

A.1.a. Reading the daily 3FF data

[3]:
```
daily_dataframe = pd.read_csv('../../data/raw/F-F_Research_Data_Factors_daily.
 ↪CSV', skiprows=4, index_col=0,\
                              parse_dates=True, skipfooter=1, engine='python')
daily_dataframe.index.names = ['Date']
```

A.1.b. Getting the dates and months from daily dataframe

[4]:
```
tmp_dates_from_daily = list(daily_dataframe.index)
dates_from_daily = [date.strftime('%Y-%m-%d') for date in tmp_dates_from_daily]
dates_from_daily = list(dict.fromkeys(dates_from_daily))
months_from_daily = [date.strftime('%Y-%m') for date in tmp_dates_from_daily]
months_from_daily = list(dict.fromkeys(months_from_daily))
```

A.2.a. Reading the monthly 3FF data

[7]:
```
monthly_dataframe = pd.read_csv('../../data/raw/F-F_Research_Data_Factors.CSV',␣
 ↪skiprows=3, index_col=0,\
                                skipfooter=97, engine='python')
```

A.2.b. Cross checking the months (Daily vs Monthly dataframe)

[8]:
```
months_from_monthly = list(monthly_dataframe.index)
months_from_monthly = [(lambda month: str(month)[:4]+"-"+str(month)[-2:])\
                       (month) for month in months_from_monthly]
```

```
if set(months_from_daily)==set(months_from_monthly):
    monthly_dataframe.index=pd.Series(months_from_monthly)
    monthly_dataframe.index.names = ['Month']
else:
    raise Exception("ERROR!! Months from daily don't match those from monthly!!!
 ↪")
```

A.3.a. Reading the monthly recession prediction from NBER

[9]:
```
monthly_recession_dataframe = pd.read_csv('../../data/raw/USREC.csv',␣
 ↪skipfooter=1, index_col=0,\
                                          engine='python')
```

A.3.b. Cross checking the months (Daily vs NBER recession prediction dataframe)

[10]:
```
months_from_nber= [datetime.strptime(date, "%Y-%m-%d").strftime("%Y-%m")\
                   for date in monthly_recession_dataframe.index]
if set(months_from_daily)==set(months_from_nber):
    monthly_recession_dataframe.index=pd.Series(months_from_nber)
    monthly_recession_dataframe.index.names = ['Month']
else:
    raise Exception("ERROR!! Months from daily don't match the months from NBER!!
 ↪!")
```

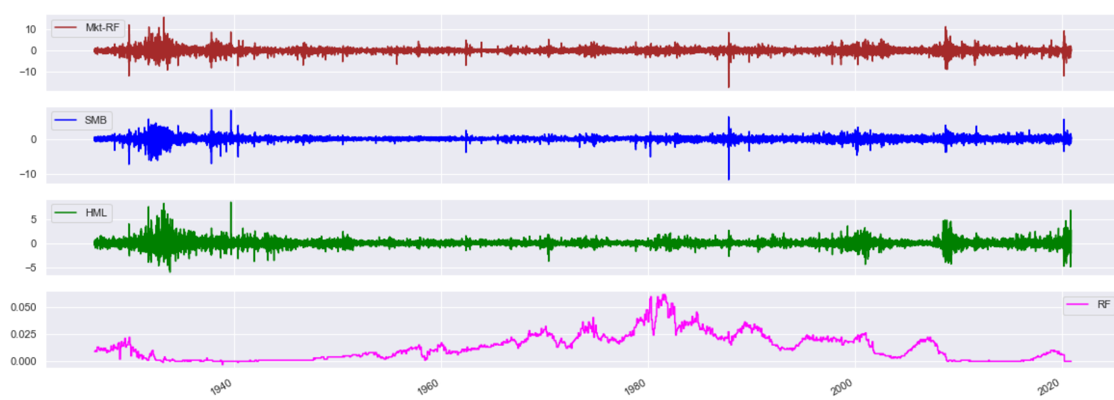
A.4. NBER recession indicator together with the monthy FF3 dataframe

[11]:
```
monthly_dataframe["USREC"] = monthly_recession_dataframe["USREC"]
```

1.1.2 B. Primary graphical investigation of the data

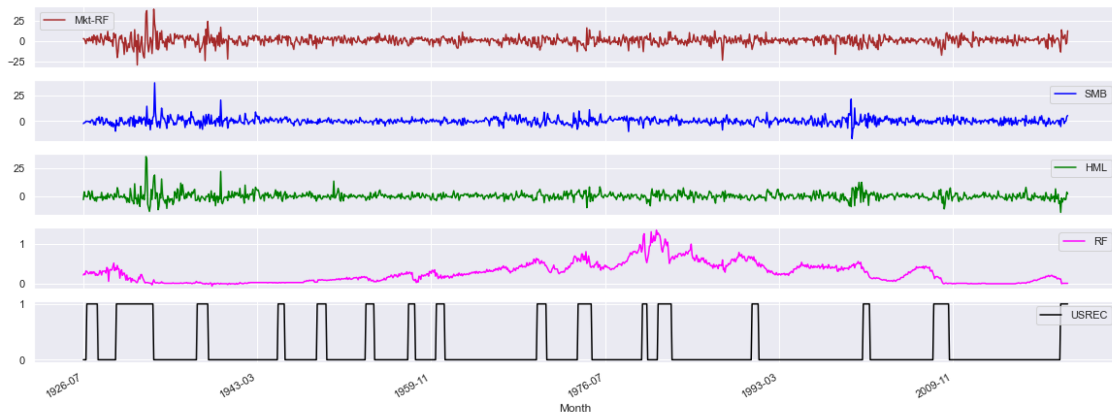
B.1.a Daily FF3 data series from 1926 till 2020

[12]:
```
daily_dataframe.plot(subplots=True, y=["Mkt-RF","SMB","HML","RF"],␣
 ↪color=["Brown","Blue","Green","Magenta"])
plt.show()
```



B.1.b. Monthly FF3 data series in the same time range

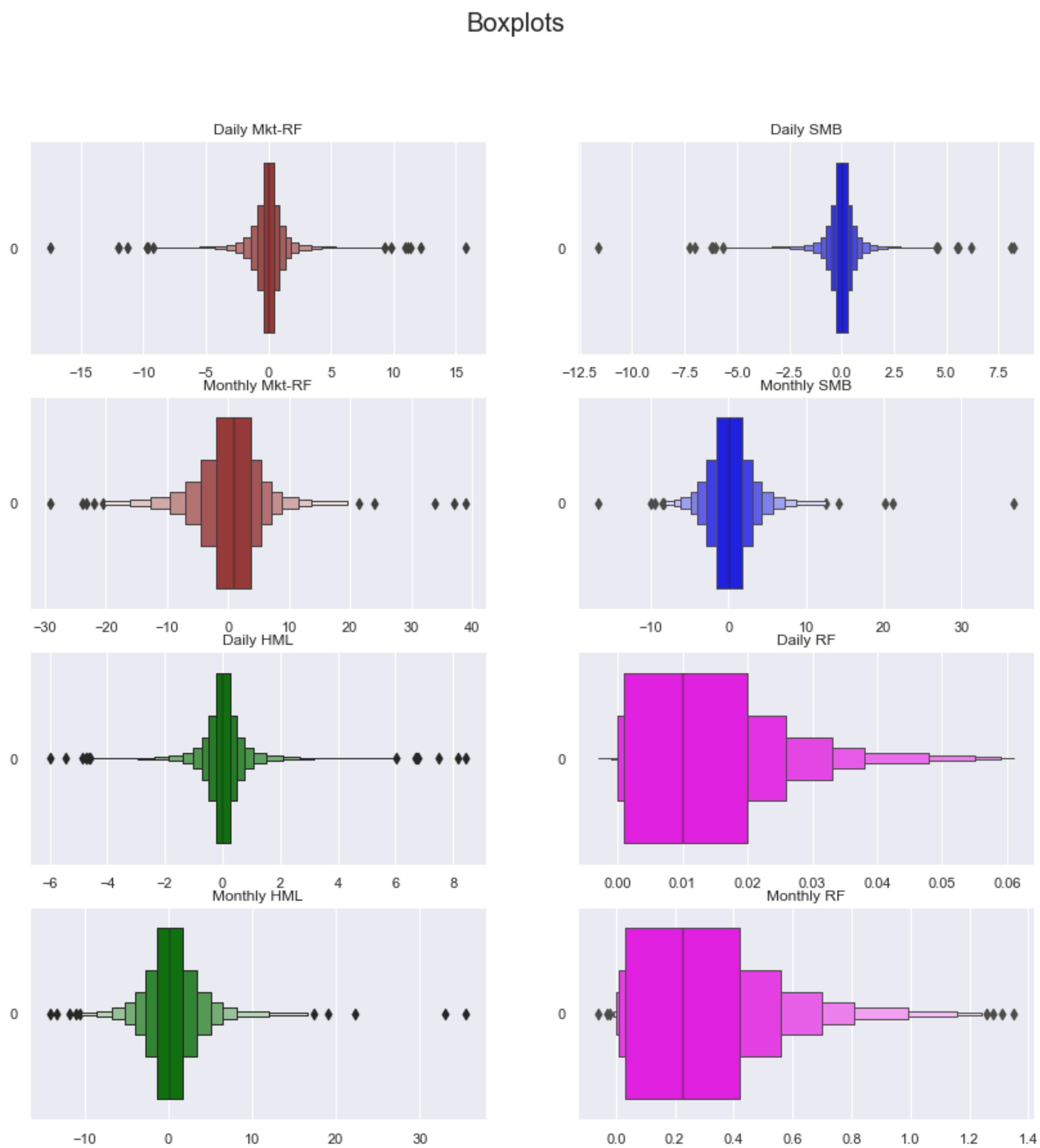
[13]:
```
monthly_dataframe.plot(subplots=True, y=["Mkt-RF","SMB","HML","RF","USREC"],\
                       color=["Brown","Blue","Green","Magenta","Black"])
plt.show()
```



B.2.a Daily and monthly box plot

[14]:
```
fig, axes = plt.subplots(4,2,figsize=(14, 14))
fig.suptitle('Boxplots', fontsize=20)

sns.boxenplot(ax=axes[0,0], data=daily_dataframe["Mkt-RF"], color="Brown",␣
 ↪orient="h")
axes[0,0].set_title("Daily Mkt-RF")
sns.boxenplot(ax=axes[1,0], data=monthly_dataframe["Mkt-RF"], color="Brown",␣
 ↪orient="h")
axes[1,0].set_title("Monthly Mkt-RF")

sns.boxenplot(ax=axes[0,1], data=daily_dataframe["SMB"], color="Blue",␣
 ↪orient="h")
axes[0,1].set_title("Daily SMB")
sns.boxenplot(ax=axes[1,1], data=monthly_dataframe["SMB"], color="Blue",␣
 ↪orient="h")
axes[1,1].set_title("Monthly SMB")

sns.boxenplot(ax=axes[2,0], data=daily_dataframe["HML"], color="Green",␣
 ↪orient="h")
axes[2,0].set_title("Daily HML")
sns.boxenplot(ax=axes[3,0], data=monthly_dataframe["HML"], color="Green",␣
 ↪orient="h")
```

```python
axes[3,0].set_title("Monthly HML")

sns.boxenplot(ax=axes[2,1], data=daily_dataframe["RF"], color="Magenta", orient="h")
axes[2,1].set_title("Daily RF")
sns.boxenplot(ax=axes[3,1], data=monthly_dataframe["RF"], color="Magenta", orient="h")
axes[3,1].set_title("Monthly RF")

plt.show()
```

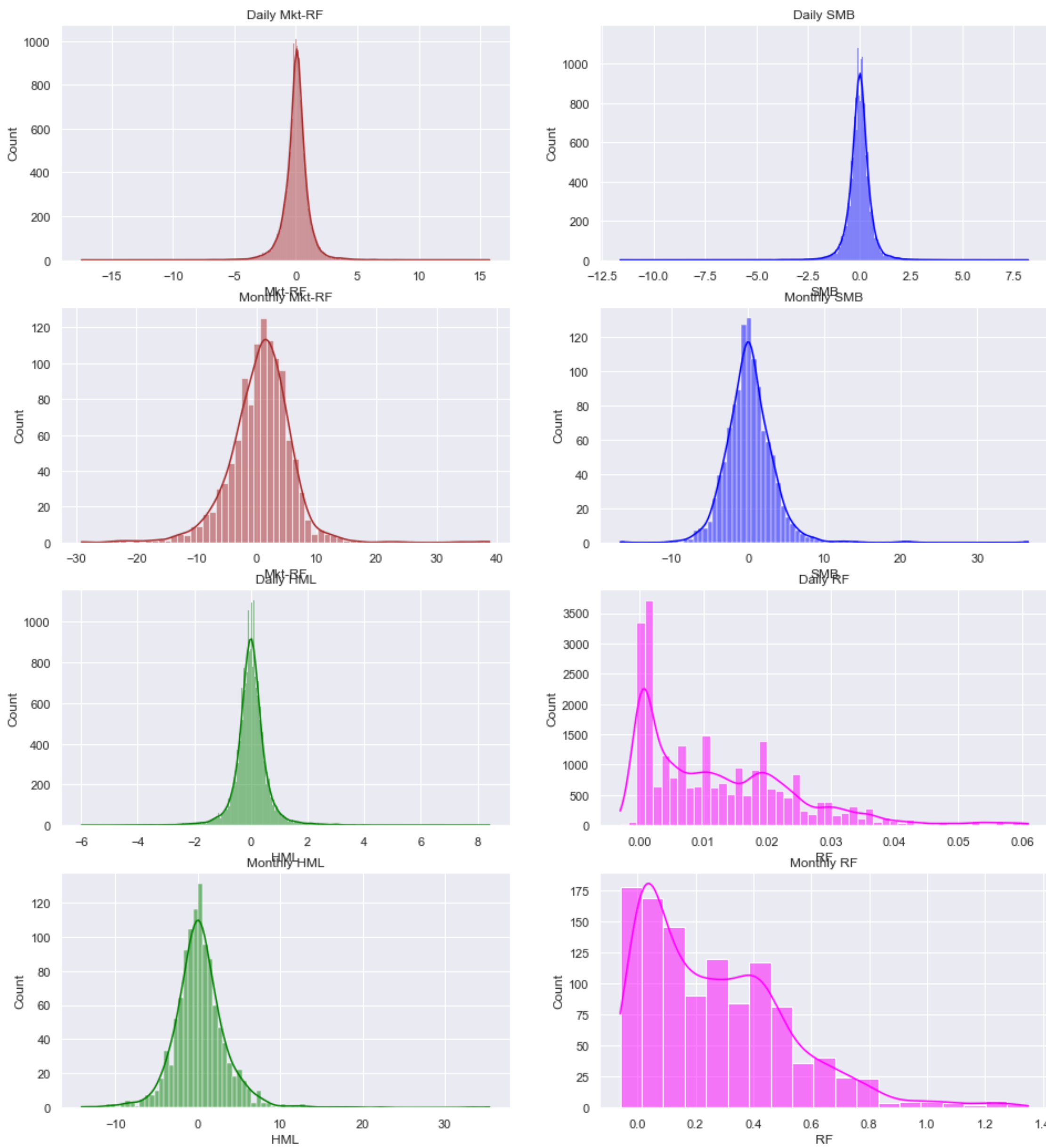


Doubts: Is this Right?

B.2.b Daily and monthly histogram plot

[15]:
```
fig, axes = plt.subplots(4,2,figsize=(16, 18))
fig.suptitle('Histograms', fontsize=20)

sns.histplot(ax=axes[0,0], data=daily_dataframe["Mkt-RF"], color="Brown",␣
 ↪kde=True)
axes[0,0].set_title("Daily Mkt-RF")
sns.histplot(ax=axes[1,0], data=monthly_dataframe["Mkt-RF"], color="Brown",␣
 ↪kde=True)
axes[1,0].set_title("Monthly Mkt-RF")

sns.histplot(ax=axes[0,1], data=daily_dataframe["SMB"], color="Blue", kde=True)
axes[0,1].set_title("Daily SMB")
sns.histplot(ax=axes[1,1], data=monthly_dataframe["SMB"], color="Blue", kde=True)
axes[1,1].set_title("Monthly SMB")

sns.histplot(ax=axes[2,0], data=daily_dataframe["HML"], color="Green", kde=True)
axes[2,0].set_title("Daily HML")
sns.histplot(ax=axes[3,0], data=monthly_dataframe["HML"], color="Green",␣
 ↪kde=True)
axes[3,0].set_title("Monthly HML")

sns.histplot(ax=axes[2,1], data=daily_dataframe["RF"], color="Magenta", kde=True)
axes[2,1].set_title("Daily RF")
sns.histplot(ax=axes[3,1], data=monthly_dataframe["RF"], color="Magenta",␣
 ↪kde=True)
axes[3,1].set_title("Monthly RF")

plt.show()
```

Histograms



1.1.3 C. Statistical analysis of the daily data

[16]:
```
def calculate_monthly_monthly_stats(daily_factor=None):

    monthly_min = list()
    monthly_max = list()
```

```
monthly_median = list()
monthly_mean = list()
monthly_var = list()
monthly_3mom = list()
monthly_4mom = list()
monthly_5mom = list()

monthly_skew = list()
monthly_kurt = list()

for month in months_from_daily:
    monthly_min.append(daily_factor[month].min())
    monthly_max.append(daily_factor[month].max())

    monthly_median.append(abs(daily_factor[month].median()))
    monthly_mean.append(abs(daily_factor[month].mean()))
    monthly_var.append(daily_factor[month].var())
    monthly_3mom.append(abs(moment(daily_factor[month], moment=3)))
    monthly_4mom.append(moment(daily_factor[month], moment=4))
    monthly_5mom.append(abs(moment(daily_factor[month], moment=5)))

    monthly_skew.append(abs(daily_factor[month].skew()))
    monthly_kurt.append(abs(daily_factor[month].kurt()))

tmp_monthly_m_v_m = {"Months":months_from_monthly,\
                     "curr_min":monthly_min,\
                     "curr_max":monthly_max,\
                     "curr_med":monthly_median,\
                     "curr_mean":monthly_mean,\
                     "curr_var":monthly_var,\
                     "curr_std":np.sqrt(monthly_var),\
                     "curr_3mom":monthly_3mom,\
                     "curr_3dev":np.power(monthly_3mom,1/3),\
                     "curr_4mom":monthly_4mom,\
                     "curr_4dev":np.power(monthly_4mom,1/4),\
                     "curr_5mom":monthly_5mom,\
                     "curr_5dev":np.power(monthly_5mom, 1/5),\
                     "curr_skew":monthly_skew,\
                     "curr_kurt":monthly_kurt}

monthly_m_v_m = pd.DataFrame(data=tmp_monthly_m_v_m)
monthly_m_v_m = monthly_m_v_m.set_index("Months")

return monthly_m_v_m
```

C.1. Calculate min, max, median, mean, std, variance, 3-moment, cube-root of 3-moment, 4-moment, quad-root of 4-moment, 5-moment, pent-root of 5-moment, skewness and kurtosis for each month

[17]:
```
monthly_mkt_rf = calculate_monthly_monthly_stats(daily_factor=daily_dataframe.
 ↪loc[:,'Mkt-RF'])
monthly_smb = calculate_monthly_monthly_stats(daily_factor=daily_dataframe.loc[:
 ↪,'SMB'])
monthly_hml = calculate_monthly_monthly_stats(daily_factor=daily_dataframe.loc[:
 ↪,'HML'])
monthly_rf = calculate_monthly_monthly_stats(daily_factor=daily_dataframe.loc[:
 ↪,'RF'])
```

[18]:
```
def plot_1(monthly_factor, name="NoName",kind="violin"):
   ␣
 ↪colors=["green","red","mediumseagreen","limegreen","darkviolet","violet","teal",\
           ␣
 ↪"cadetblue","gold","khaki","brown","lightcoral","steelblue","skyblue"]
    fig,axes = plt.subplots(4,4,figsize=(17.5, 20))
    fig.suptitle('Monthly %s\'s behaviour' %name)
    for i, column in enumerate(monthly_factor.columns):
        k = i%4
        j = int((i-k)/4)
        if kind=="violin":
            sns.violinplot(ax=axes[j,k], data=monthly_factor[column],␣
 ↪orient="h",\
                           color=colors[i])
        else:
            sns.boxplot(ax=axes[j,k], data=monthly_factor[column], orient="h",\
                           color=colors[i])
        axes[j,k].set_title(column)
```

C.2.a. Visualization of Mkt-RF statistics `plot_1(monthly_factor=monthly_mkt_rf, name="Mkt-RF")`

C.2.b. Visualization of SMB statistics `plot_1(monthly_factor=monthly_smb, name="SMB")`

C.2.c. Visualization of HML statistics `plot_1(monthly_factor=monthly_hml, name="HML")`

C.2.d. Visualization of RF statistics `plot_1(monthly_factor=monthly_rf, name="RF")`

[19]:
```
monthly_fators = [monthly_mkt_rf, monthly_smb, monthly_hml, monthly_rf]
def plot_2(monthly_factors=monthly_fators, what="curr_var", name="Variance"):
    title= "Months Vs. {}".format(name)
    monthly_factors[0].loc[:, what].plot(linewidth=0.85, color="Brown",␣
 ↪title=title)
    monthly_factors[1].loc[:, what].plot(linewidth=0.85, color="Blue")
```

```python
    monthly_factors[2].loc[:, what].plot(linewidth=0.85, color="Green")
    ax = monthly_factors[3].loc[:, what].plot(linewidth=0.85, color="Magenta")
    ax.set_ylabel(name)
    ax.legend(["Mkt-RF", "SMB", "HML", "RF"])
    plt.show()
```

C.3.a. Visualization of monthly minima of individual factors `plot_2(what="curr_min", name="Minimum")`

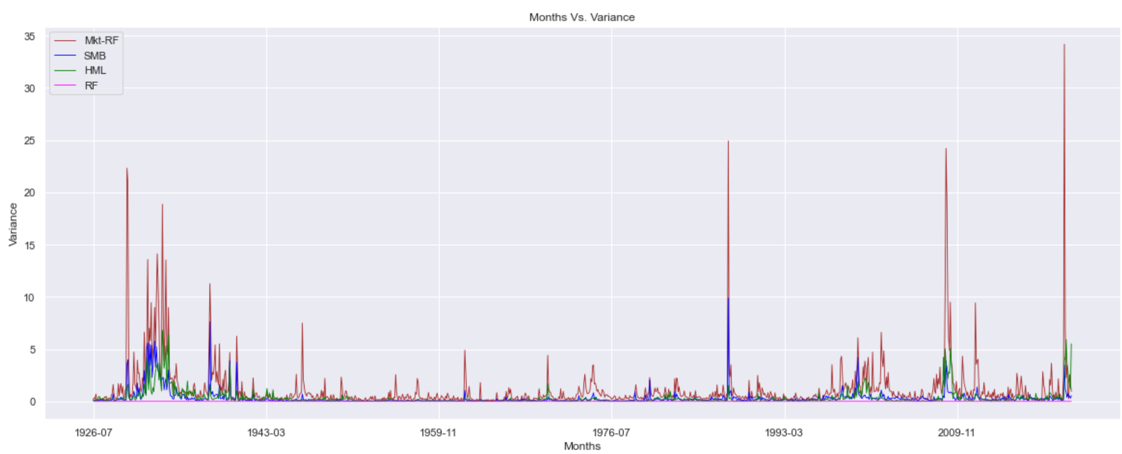
C.3.b. Visualization of monthly maxima of individual factors `plot_2(what="curr_max", name="Maximum")`

C.3.c. Visualization of monthly medians of individual factors `plot_2(what="curr_med", name="Median")`

C.3.d. Visualization of monthly means of individual factors `plot_2(what="curr_mean", name="Mean")`

C.3.e. Visualization of monthly variances of individual factors

[20]:
```python
plot_2(what="curr_var", name="Variance")
```



C.3.f. Visualization of monthly standard deviations of individual factors `plot_2(what="curr_std", name="STD")`

C.3.g. Visualization of monthly 3rd-moments of individual factors `plot_2(what="curr_3mom", name="3rd moment")`

C.3.h. Visualization of monthly cube-roots-of-3rd-moments of individual factors `plot_2(what="curr_3dev", name="3rd root of 3rd moment")`

C.3.i. Visualization of monthly 4th-moments of individual factors `plot_2(what="curr_4mom", name="4th moment")`

C.3.j. Visualization of monthly quad-roots-of-4th-moments of individual factors `plot_2(what="curr_4dev", name="4th root of 4th moment")`

C.3.k. Visualization of monthly 5th-moments of individual factors `plot_2(what="curr_5mom", name="5th moment")`

C.3.l. Visualization of monthly pent-roots-of-5th-moments of individual factors `plot_2(what="curr_5dev", name="5th root of 5th moment")`

C.3.m. Visualization of monthly skewness of individual factors `plot_2(what="curr_skew", name="skewness")`

C.3.n. Visualization of monthly kurtosis of individual factors `plot_2(what="curr_kurt", name="kurtosis")`

1.1.4 D. Expanatory strength analysis of each monthly statistics (lagged) towards predicting next month's (current) statistics and return

D.1.a. Adding the current return from the monthly dataframe to the individual factors statistics dataframe

[21]:
```
monthly_mkt_rf["curr_ret"] = monthly_dataframe["Mkt-RF"].abs()
monthly_smb["curr_ret"] = monthly_dataframe["SMB"].abs()
monthly_hml["curr_ret"] = monthly_dataframe["HML"].abs()
monthly_rf["curr_ret"] = monthly_dataframe["RF"].abs()
```

Doubt: Is it right? I used absolute here.

D.1.b. Adding the recession factor to them

[22]:
```
monthly_mkt_rf["rec_fact"] = monthly_dataframe["USREC"]
monthly_smb["rec_fact"] = monthly_dataframe["USREC"]
monthly_hml["rec_fact"] = monthly_dataframe["USREC"]
monthly_rf["rec_fact"] = monthly_dataframe["USREC"]
```

D.1.c Adding the lagged values of the statistics and return

[23]:
```
def include_lagged_values(monthly_factor):
    monthly_factor["lagd_ret"] = monthly_factor.curr_ret.shift(1)
    monthly_factor["lagd_med"] = monthly_factor.curr_med.shift(1)
    monthly_factor["lagd_mean"] = monthly_factor.curr_mean.shift(1)
    monthly_factor["lagd_var"] = monthly_factor.curr_var.shift(1)
    monthly_factor["lagd_std"] = monthly_factor.curr_std.shift(1)
    monthly_factor["lagd_3mom"] = monthly_factor.curr_3mom.shift(1)
    monthly_factor["lagd_3dev"] = monthly_factor.curr_3dev.shift(1)
```

```python
    monthly_factor["lagd_4mom"] = monthly_factor.curr_4mom.shift(1)
    monthly_factor["lagd_4dev"] = monthly_factor.curr_4dev.shift(1)
    monthly_factor["lagd_5mom"] = monthly_factor.curr_5mom.shift(1)
    monthly_factor["lagd_5dev"] = monthly_factor.curr_5dev.shift(1)
    monthly_factor["lagd_skew"] = monthly_factor.curr_skew.shift(1)
    monthly_factor["lagd_kurt"] = monthly_factor.curr_kurt.shift(1)

    return monthly_factor
```

[24]:
```python
monthly_mkt_rf = include_lagged_values(monthly_factor=monthly_mkt_rf)
monthly_smb = include_lagged_values(monthly_factor=monthly_smb)
monthly_hml = include_lagged_values(monthly_factor=monthly_hml)
monthly_rf = include_lagged_values(monthly_factor=monthly_rf)
```

[25]:
```python
# The analysis can be extended to all the below bags,
# This was done in version 1,
# All option is not very informative
# For the sake of explanational clarity
# the bags are restricted to the kinds:
# Median, Mean, STD, Variance, Skewness, Kurtosis
bag_kinds_all = ["med_bag","mean_bag","std_bag","var_bag",\
                 "3mom_bag","3dev_bag","4mom_bag","4dev_bag",\
                 "5mom_bag","5dev_bag","skew_bag","kurt_bag"]
bag_kinds = bag_kinds_all[:4] + bag_kinds_all[-2:]

# The dictionary and lists below are just for the sake of plotting
dict_for_plot_kinds = {"med_bag":"Median","mean_bag":"Mean",\
                       "std_bag":"STD","var_bag":"Variance",\
                       "3mom_bag":"3rd Momentum","3dev_bag":"3rd STD",\
                       "4mom_bag":"4th Momentum","4dev_bag":"4th STD",\
                       "5mom_bag":"5th Momentum","5dev_bag":"5th STD",\
                       "skew_bag":"Skewness","kurt_bag":"Kurtosis"}
# The dictionary and lists below are just for the sake of plotting
dict_for_plot_STD_Moments = {"std_bag":"Variance",\
                             "3dev_bag":"3rd Momentum",\
                             "4dev_bag":"4th Momentum",\
                             "5dev_bag":"5th Momentum"}
#
baggable_columns_all = ["lagd_med","lagd_mean","lagd_std","lagd_var",\
                        "lagd_3mom","lagd_3dev","lagd_4mom","lagd_4dev",\
                        "lagd_5mom","lagd_5dev","lagd_skew","lagd_kurt"]
baggable_columns = baggable_columns_all[:4] + baggable_columns_all[-2:]
#
corres_curr_all = ["curr_med","curr_mean","curr_std","curr_var",\
                   "curr_3mom","curr_3dev","curr_4mom","curr_4dev",\
                   "curr_5mom","curr_5dev","curr_skew","curr_kurt"]
corres_curr = corres_curr_all[:4] + corres_curr_all[-2:]
```

```
#
corres_scald_ret_all = ["curr_med_scald_ret","curr_mean_scald_ret",\
                        "curr_std_scald_ret","curr_var_scald_ret",\
                        "curr_3mom_scald_ret","curr_3dev_scald_ret",\
                        "curr_4mom_scald_ret","curr_4dev_scald_ret",\
                        "curr_5mom_scald_ret","curr_5dev_scald_ret",\
                        "curr_skew_scald_ret","curr_kurt_scald_ret"]
corres_scald_ret = corres_scald_ret_all[:4] + corres_scald_ret_all[-2:]
# This is done for plotting
# momentum scaled monthly return whenever
# correspongíng STD is plotted
# This gives closer watch if
# momentum or STD scaling is better while
# we split into bags
other_scald_ret_all = [None, None,\
                       "curr_var_scald_ret", None,\
                       None,"curr_3mom_scald_ret",\
                       None,"curr_4mom_scald_ret",\
                       None,"curr_5mom_scald_ret",\
                       None,None]
other_scald_ret = other_scald_ret_all[:4] + other_scald_ret_all[-2:]
```

[26]:
```
# decided not to include monthly_rf in the version 2
# not so informative
# refer version 0 for a look
# monthly_fators = [monthly_mkt_rf, monthly_smb, monthly_hml, monthly_rf]
# title_part = "(Mkt-RF, SMB, HML, RF)"
monthly_fators = [monthly_mkt_rf, monthly_smb, monthly_hml]
title_part = "(Mkt-RF, SMB, HML)"
list_title_part = list(title_part.replace("(","").replace(")","")\
                                 .replace(",","").split())
```

[27]:
```
def plot_3(y, name, kind="scatter"):

    colors=["mediumseagreen","limegreen","violet",\
            "cadetblue","khaki","lightcoral","steelblue","skyblue"]
    columns_all = ["lagd_med","lagd_mean","lagd_std","lagd_3dev",\
                   "lagd_4dev","lagd_5dev","lagd_skew","lagd_kurt"]
    columns = columns_all[:3] + columns_all[-2:]
    corres_kind_all = ["med_bag","mean_bag","std_bag",\
                       "3dev_bag","4dev_bag","5dev_bag",\
                       "skew_bag","kurt_bag"]
    corres_kind = corres_kind_all[:3] + corres_kind_all[-2:]
    fig,axes = plt.subplots(len(columns),\
                            len(monthly_fators),\
                            figsize=(len(monthly_fators)*5,\
                                     len(columns)*4))
```

```
    fig.suptitle('Monthly %s for %s' %(name, title_part), fontsize=15)



    for i, monthly_factor in enumerate(monthly_fators):
        for j, column in enumerate(columns):
            bag_kind = dict_for_plot_kinds[corres_kind[j]]
            if kind=="scatter":
                sns.scatterplot(ax=axes[j,i], data=monthly_factor,\
                                x=column, y=y, color=colors[j])
                axes[j,i].set_xlabel(("Lagged "+bag_kind))
                axes[j,i].set(ylabel=None)
            else:
                pass

    for k, ax in enumerate(axes[0]):
        ax.set_title(list_title_part[k])

    for ax in axes[:,0]:
        ax.set_ylabel(name)
```

D.2.a Visualization current return vs other lagged statistics `plot_3(y="curr_ret", name="Current return", kind="scatter")`

D.2.a Visualization current recession prediction vs other lagged statistics `plot_3(y="rec_fact", name="Recession prediction", kind="scatter")`

1.1.5 E. Calculate the scaled monthly return

[28]:
```
def include_scaled_returns(monthly_factor):
    monthly_factor["curr_med_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_med
    monthly_factor["curr_mean_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_mean
    monthly_factor["curr_std_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_std
    monthly_factor["curr_var_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_var
    monthly_factor["curr_3mom_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_3mom
    monthly_factor["curr_3dev_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_3dev
    monthly_factor["curr_4mom_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_4mom
    monthly_factor["curr_4dev_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_4dev
```

```
    monthly_factor["curr_5mom_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_5mom
    monthly_factor["curr_5dev_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_5dev
    monthly_factor["curr_skew_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_skew
    monthly_factor["curr_kurt_scald_ret"] = monthly_factor.curr_ret/
↪monthly_factor.curr_kurt

    return monthly_factor
```

[29]:
```
monthly_mkt_rf = include_scaled_returns(monthly_factor=monthly_mkt_rf)
monthly_smb = include_scaled_returns(monthly_factor=monthly_smb)
monthly_hml = include_scaled_returns(monthly_factor=monthly_hml)
monthly_rf = include_scaled_returns(monthly_factor=monthly_rf)
```

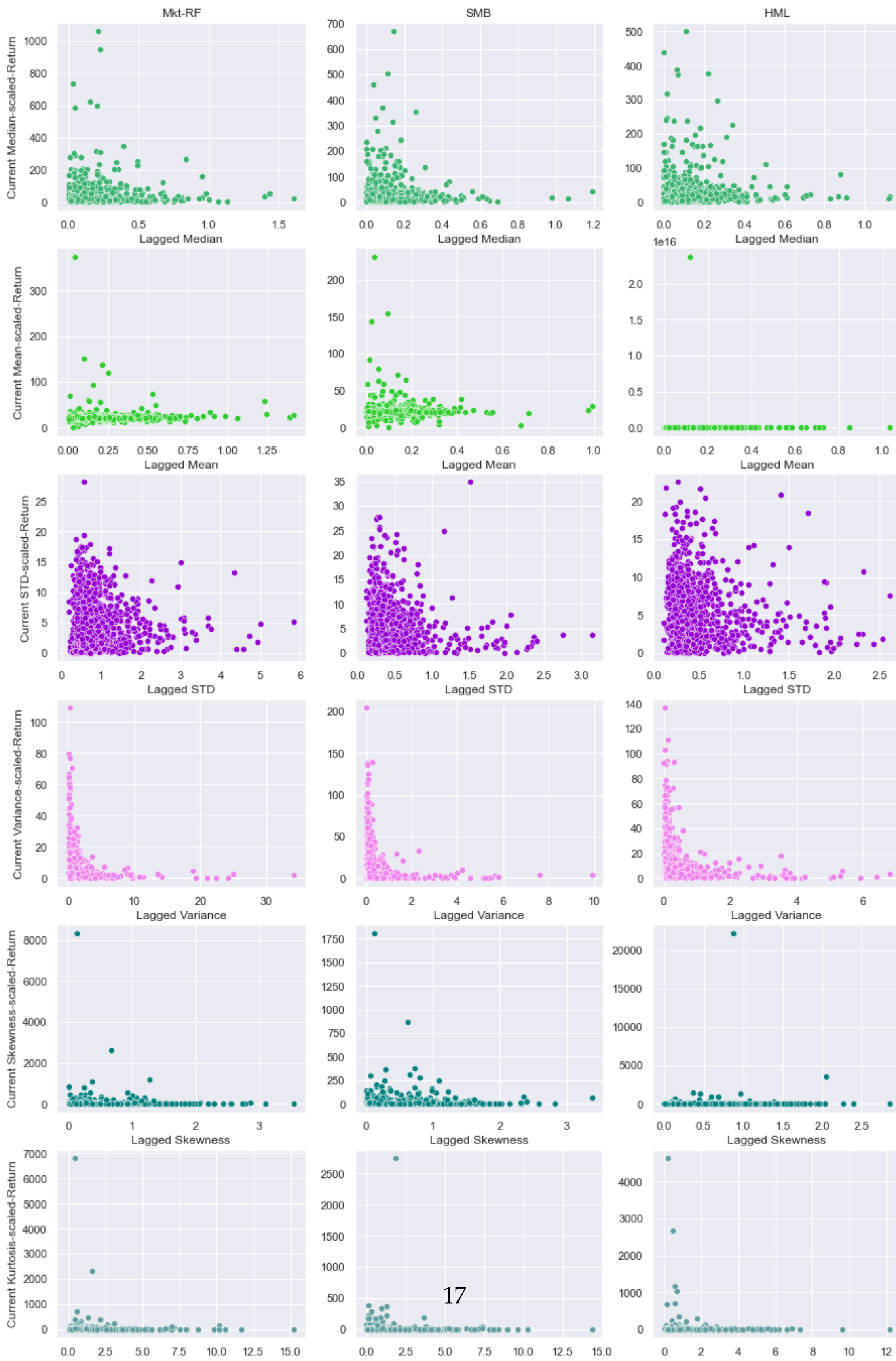
[30]:
```
def plot_4(kind="scatter"):
    colors=["mediumseagreen","limegreen","darkviolet","violet","teal",\
            ␣
↪"cadetblue","gold","khaki","brown","lightcoral","steelblue","skyblue"]

    fig,axes = plt.subplots(len(bag_kinds),\
                            len(monthly_fators),\
                            figsize=(len(monthly_fators)*5,\
                                     len(bag_kinds)*4))
    fig.suptitle("Lagged vs Corresponding scaled return " + title_part,␣
↪fontsize=15)

    for i, monthly_factor in enumerate(monthly_fators):
        for j, column in enumerate(baggable_columns):
            bag_kind = dict_for_plot_kinds[bag_kinds[j]]
            if kind=="scatter":
                sns.scatterplot(ax=axes[j,i], data=monthly_factor,\
                                x=column, y=corres_scald_ret[j],\
                                color=colors[j])
                axes[j,i].set_xlabel("Lagged "+bag_kind)
                #axes[j,i].set_ylabel("Current {}-scaled-Return".
↪format(bag_kind))
                axes[j,i].set(ylabel=None)
            else:
                pass

    for k, ax in enumerate(axes[0]):
        ax.set_title(list_title_part[k])

    for l, ax in enumerate(axes[:,0]):
```

```
        bag_kind = dict_for_plot_kinds[bag_kinds[1]]
        ylabel = "Current {}-scaled-Return".format(bag_kind)
        ax.set_ylabel(ylabel)
```

E.1. Lagged volatilty vs. Scaled-return

[31]:
```
plot_4()
```

Lagged vs Corresponding scaled return (Mkt-RF, SMB, HML)



1.1.6 F. Bagging

[32]:
```python
# The analysis can be extended to all the below bags,
# This was done in version 1,
# All option is not very informative
# For the sake of explanational clarity
# the bags are restricted to the kinds:
# Median, Mean, STD, Variance, Skewness, Kurtosis
bag_kinds_all = ["med_bag","mean_bag","std_bag","var_bag",\
             "3mom_bag","3dev_bag","4mom_bag","4dev_bag",\
             "5mom_bag","5dev_bag","skew_bag","kurt_bag"]
bag_kinds = bag_kinds_all[:4] + bag_kinds_all[-2:]

# The dictionary and lists below are just for the sake of plotting
dict_for_plot_kinds = {"med_bag":"Median","mean_bag":"Mean",\
                       "std_bag":"STD","var_bag":"Variance",\
                       "3mom_bag":"3rd Momentum","3dev_bag":"3rd STD",\
                       "4mom_bag":"4th Momentum","4dev_bag":"4th STD",\
                       "5mom_bag":"5th Momentum","5dev_bag":"5th STD",\
                       "skew_bag":"Skewness","kurt_bag":"Kurtosis"}
# The dictionary and lists below are just for the sake of plotting
dict_for_plot_STD_Moments = {"std_bag":"Variance",\
                             "3dev_bag":"3rd Momentum",\
                             "4dev_bag":"4th Momentum",\
                             "5dev_bag":"5th Momentum"}
#
baggable_columns_all = ["lagd_med","lagd_mean","lagd_std","lagd_var",\
                    "lagd_3mom","lagd_3dev","lagd_4mom","lagd_4dev",\
                    "lagd_5mom","lagd_5dev","lagd_skew","lagd_kurt"]
baggable_columns = baggable_columns_all[:4] + baggable_columns_all[-2:]
#
corres_curr_all = ["curr_med","curr_mean","curr_std","curr_var",\
               "curr_3mom","curr_3dev","curr_4mom","curr_4dev",\
               "curr_5mom","curr_5dev","curr_skew","curr_kurt"]
corres_curr = corres_curr_all[:4] + corres_curr_all[-2:]
#
corres_scald_ret_all = ["curr_med_scald_ret","curr_mean_scald_ret",\
                    "curr_std_scald_ret","curr_var_scald_ret",\
                    "curr_3mom_scald_ret","curr_3dev_scald_ret",\
                    "curr_4mom_scald_ret","curr_4dev_scald_ret",\
                    "curr_5mom_scald_ret","curr_5dev_scald_ret",\
                    "curr_skew_scald_ret","curr_kurt_scald_ret"]
corres_scald_ret = corres_scald_ret_all[:4] + corres_scald_ret_all[-2:]
# This is done for plotting
```

```
# momentum scaled monthly return whenever
# corresponging STD is plotted
# This gives closer watch if
# momentum or STD scaling is better while
# we split into bags
other_scald_ret_all = [None, None,\
                   "curr_var_scald_ret", None,\
                   None,"curr_3mom_scald_ret",\
                   None,"curr_4mom_scald_ret",\
                   None,"curr_5mom_scald_ret",\
                   None,None]
other_scald_ret = other_scald_ret_all[:4] + other_scald_ret_all[-2:]
```

[33]:
```
def give_bagged_factor(monthly_factor, column, scald_ret, kind, no_bags,␣
 ↪other_ret=None):

    _kind = kind
    kind = kind + "_in{}".format(no_bags)
    bags = [i+1 for i in range(no_bags)]

    min_monthly_factor = monthly_factor[column].min()
    max_monthly_factor = monthly_factor[column].max()
    dev_monthly_factor = max_monthly_factor-min_monthly_factor
    monthly_factor[kind] = np.uintc(1 + np.floor((no_bags-0.0001)\
                                    *(monthly_factor[column]-\
                                      min_monthly_factor)/
 ↪dev_monthly_factor))


    factor_bag_avg_ret = list()
    factor_bag_avg_column = list()
    factor_bag_avg_rec_prob = list()
    factor_bag_avg_column_scald_ret = list()

    if other_ret:
        factor_bag_avg_other_scald_ret = list()

    for bag in bags:
        tmp_mean_avg_ret = np.
 ↪mean(monthly_factor[monthly_factor[kind]==bag]["curr_ret"])
        tmp_mean_avg_column = np.
 ↪mean(monthly_factor[monthly_factor[kind]==bag][column])
        tmp_mean_recession_chance = np.mean(monthly_factor[monthly_factor[kind\
                                                                        ␣
 ↪]==bag]["rec_fact"])
        factor_bag_avg_ret.append(tmp_mean_avg_ret)
        factor_bag_avg_column.append(tmp_mean_avg_column)
```

```
        factor_bag_avg_rec_prob.append(tmp_mean_recession_chance)
        factor_bag_avg_column_scald_ret.append(np.
↪mean(monthly_factor[monthly_factor[kind]\
                                                                    ␣
↪==bag][scald_ret]))

        if other_ret:
            factor_bag_avg_other_scald_ret.append(np.
↪mean(monthly_factor[monthly_factor[kind]\
                                                                       ␣
↪==bag][other_ret]))

    avg_column_name = "avg" + column[4:]
    avg_column_scald_ret_name = "avg" + scald_ret[4:]

    tmp_bagged_factor = { kind : bags,\
                          "avg_ret" : factor_bag_avg_ret,\
                          avg_column_name : factor_bag_avg_column,\
                          avg_column_scald_ret_name :␣
↪factor_bag_avg_column_scald_ret,\
                          "avg_rec_fact" : factor_bag_avg_rec_prob}

    if other_ret:
        avg_other_scald_ret_name = "avg" + other_ret[4:]
        tmp_bagged_factor[avg_other_scald_ret_name] =␣
↪factor_bag_avg_other_scald_ret

    bagged_factor = pd.DataFrame(data=tmp_bagged_factor)

    return monthly_factor, bagged_factor
```

F.1. Bagging each monthly factor for each statistic: median, mean, standard deviation, variance, 3rd moment, 3rd deviation, 4th moment, 4th deviation, 5th moment, 5th deviation, skewness, kurtosis

[34]:
```
# bag adjuster
# values_of_some_bags = range(5,16)
# values_of_some_bags = np.unique(np.random.randint(5,15, 6))
# values_of_some_bags = np.array([5,7,10,12, 15])
values_of_some_bags = np.array([5,10,15])
def big_bagger(monthly_factor):
    _monthly = monthly_factor
    tmp = dict()
    for i, kind in enumerate(bag_kinds):
        tmp[kind] = dict()
        for no_bags in values_of_some_bags:
            _monthly, _bagged = give_bagged_factor(monthly_factor=_monthly,\
```

```
                                                    column=baggable_columns[i],\
                                                    kind=bag_kinds[i],\
                                                   ␣
↪scald_ret=corres_scald_ret[i],\
                                                    other_ret=other_scald_ret[i],\
                                                    no_bags=no_bags)
            tmp[kind][no_bags] = _bagged

    return _monthly, tmp
```

[35]:
```
monthly_mkt_rf, big_bag_mkt_rf = big_bagger(monthly_factor=monthly_mkt_rf)
monthly_smb, big_bag_smb = big_bagger(monthly_factor=monthly_smb)
monthly_hml, big_bag_hml = big_bagger(monthly_factor=monthly_hml)
monthly_rf, big_bag_rf = big_bagger(monthly_factor=monthly_rf)
```

Note: The bag looks like this..

[36]:
```
# from IPython.display import display, HTML
# display(HTML(big_bag_mkt_rf["med_bag"][values_of_some_bags[0]].to_html()))
big_bag_mkt_rf["med_bag"][values_of_some_bags[0]]
```

[36]:

	med_bag_in5	avg_ret	avg_med	avg_med_scald_ret	avg_rec_fact
0	1	3.612604	0.130365	inf	0.154002
1	2	4.648348	0.432087	inf	0.321739
2	3	7.847391	0.770870	inf	0.608696
3	4	6.557500	1.046250	18.875282	0.500000
4	5	19.583333	1.481667	38.174802	1.000000

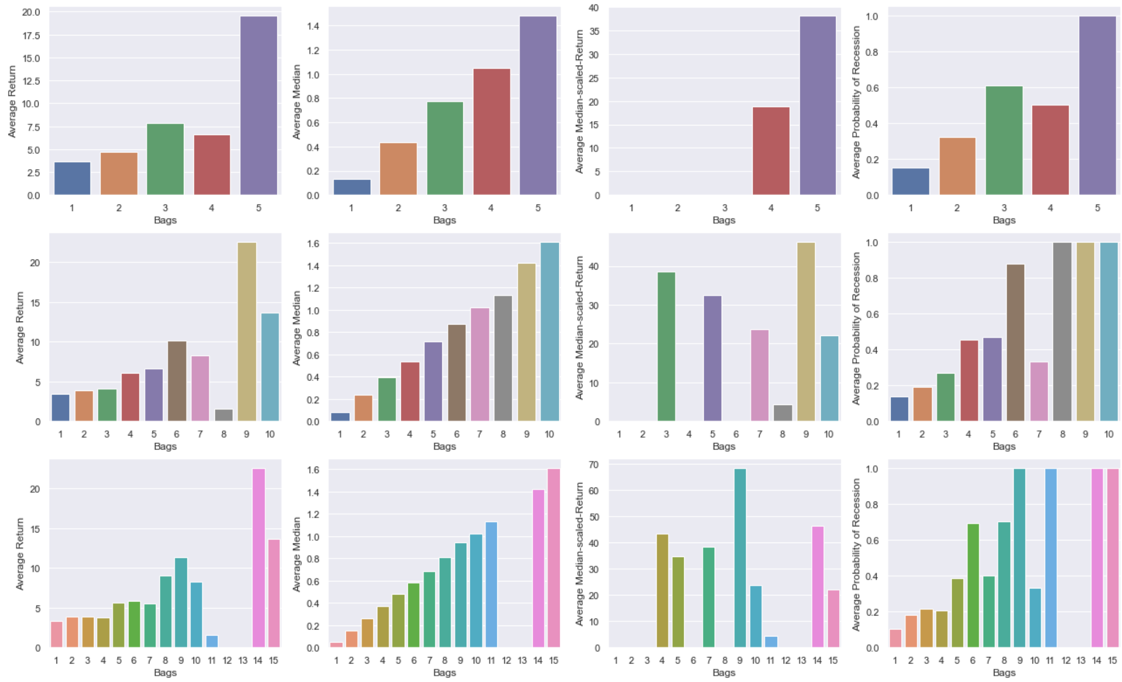
[37]:
```
def plot_big_bag(big_bag):
    for i, kind in enumerate(bag_kinds):
        bagged_factor = big_bag[kind]

        nos_cols = 4
        if other_scald_ret[i]:
            nos_cols = 5

        nos_rows = len(values_of_some_bags)

        fig, axes = plt.subplots(nos_rows, nos_cols, figsize=(22, nos_rows*4.5))
        plt.grid(True)
        bag_name = dict_for_plot_kinds[kind]
        title_part = ""
        for i, bag_division in enumerate(values_of_some_bags):
            title_part += str(bag_division)
            if len(values_of_some_bags)-(i+1):
                title_part += ", "
```

```
        name = "Bags made by dividing " + bag_name +\
               " into " + title_part +\
               " equal segments"
        fig.suptitle(name, fontsize=20)

        column_names = ["Return", bag_name,\
                        bag_name+"-scaled-Return",\
                        "Probability of Recession"]
        if nos_cols == 5:
            corres_mom = dict_for_plot_STD_Moments[kind]
            column_names = ["Return", bag_name,\
                            bag_name+"-scaled-Return",\
                            "Probability of Recession",\
                            corres_mom+"-scaled-Return"]

        for _no, nos_bags in enumerate(values_of_some_bags):
            _bagged_factor = bagged_factor[nos_bags]
            _kind = kind + "_in{}".format(nos_bags)
            _columns = _bagged_factor.columns
            _x = _columns[0]

            j = _no
            for k  in range(nos_cols):
                _y = _columns[k+1]
                sns.barplot(ax=axes[j,k], data=_bagged_factor, x=_x, y=_y,␣
↪orient="v")
                axes[j,k].set_xlabel("Bags")
                axes[j,k].set_ylabel("Average "+column_names[k])

        plt.show()
```

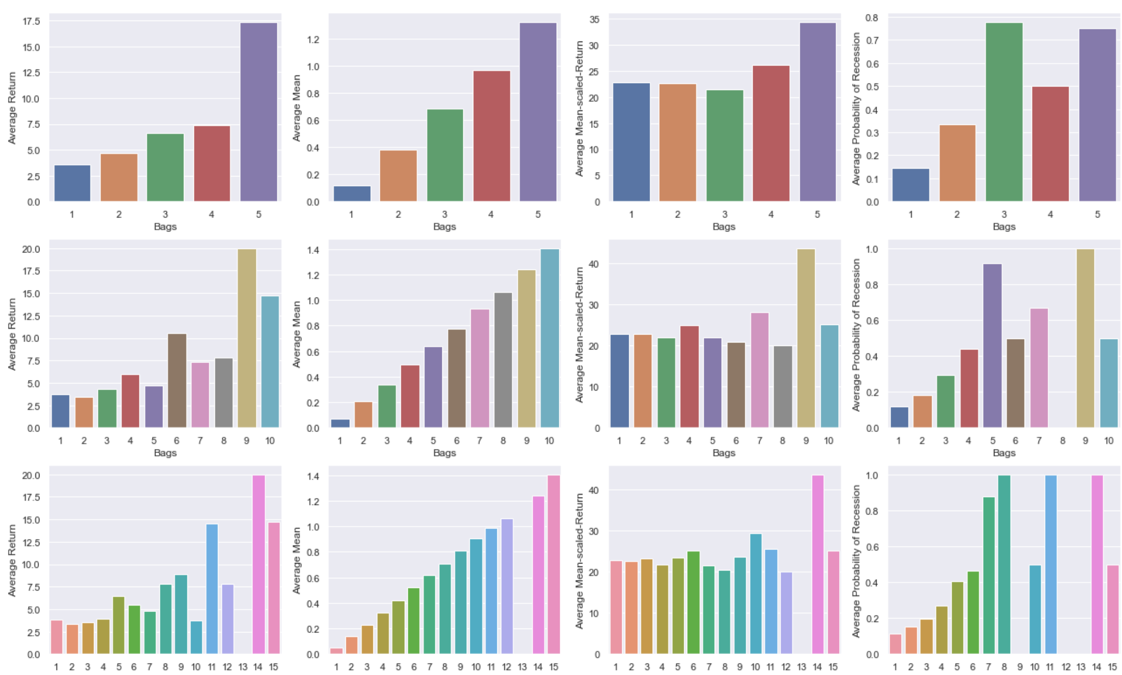
F.2.a. Visualization of Mkt-RF bags

[38]:
```
plot_big_bag(big_bag=big_bag_mkt_rf)
```

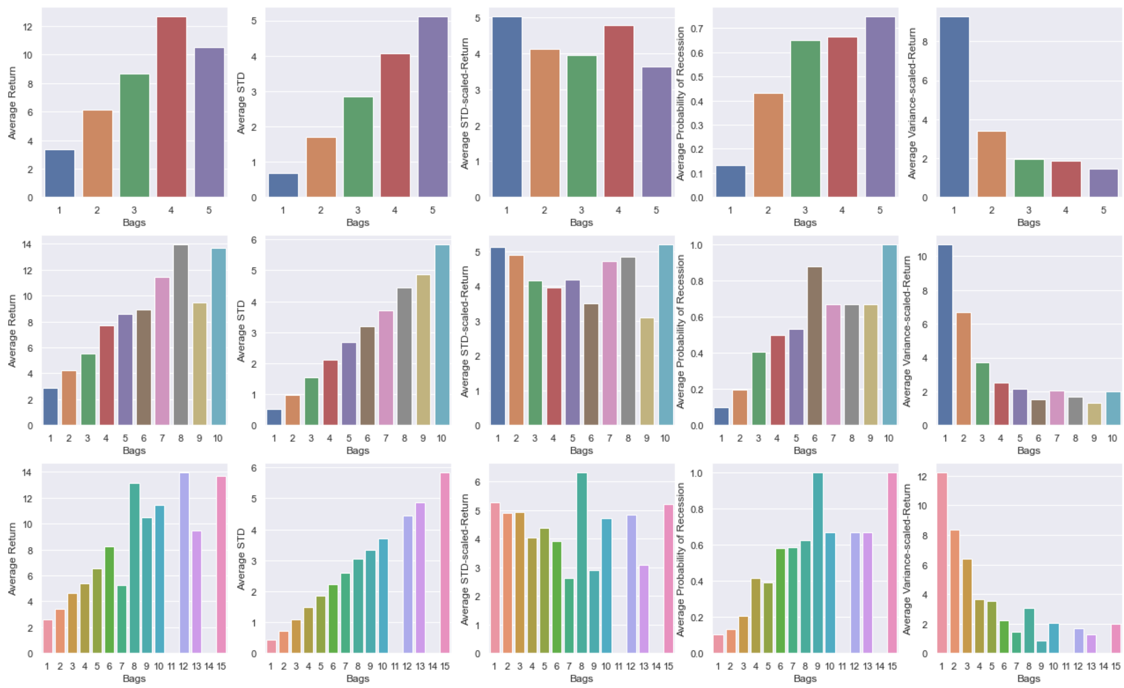
Bags made by dividing Median into 5, 10, 15 equal segments



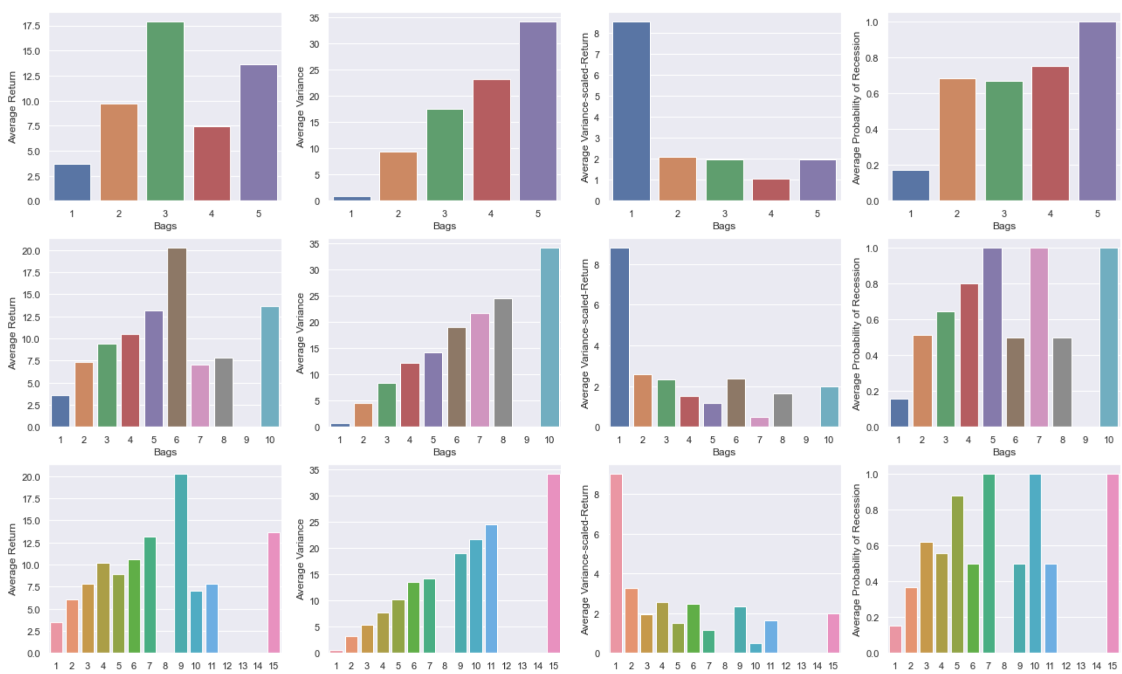
Bags made by dividing Mean into 5, 10, 15 equal segments



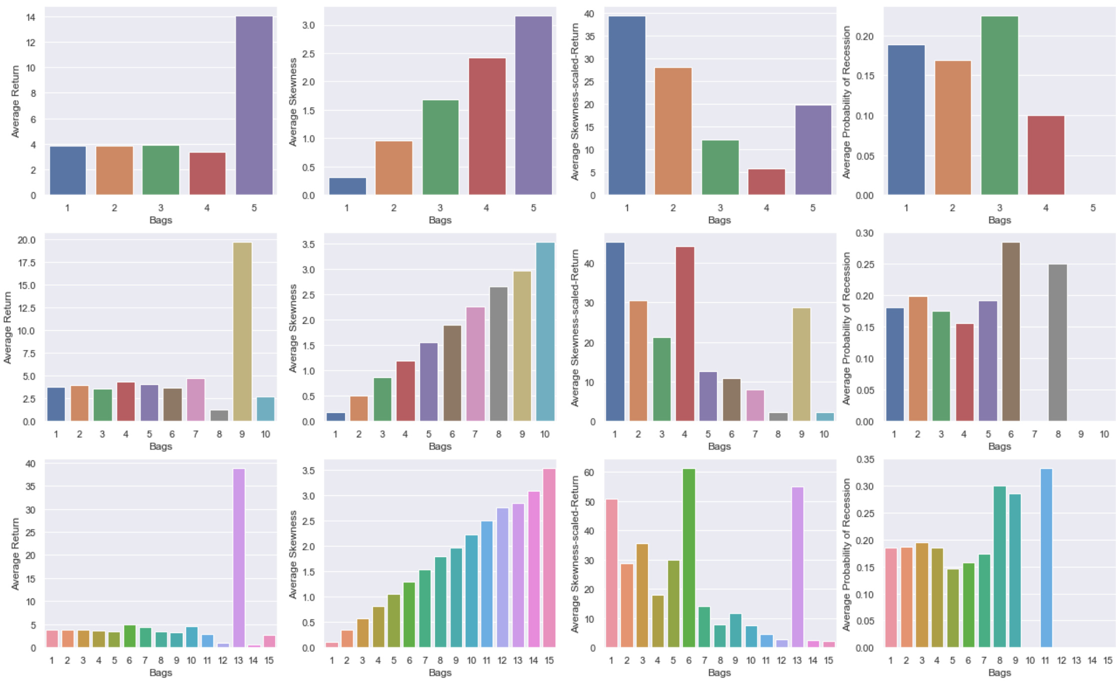
Bags made by dividing STD into 5, 10, 15 equal segments

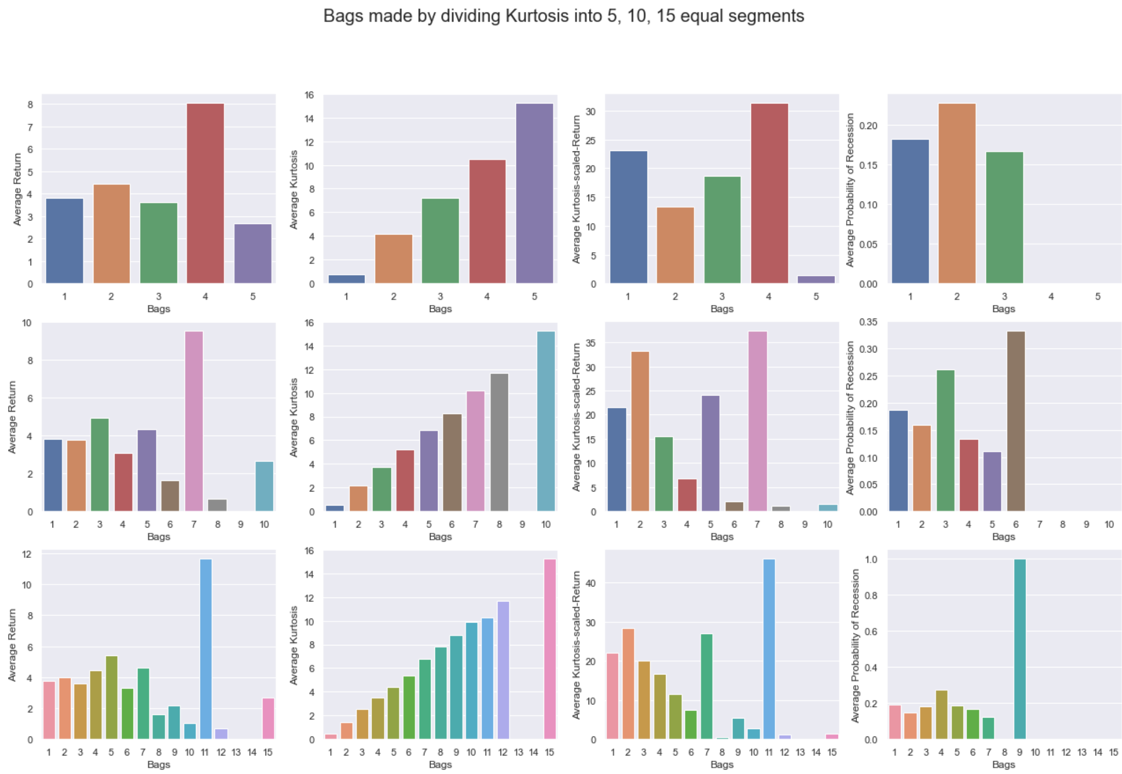


Bags made by dividing Variance into 5, 10, 15 equal segments



Bags made by dividing Skewness into 5, 10, 15 equal segments





F.2.b. Visualization of SMB bags `plot_big_bag(big_bag=big_bag_smb)`

F.2.c. Visualization of HML bags `plot_big_bag(big_bag=big_bag_hml)`

1.1.7 G. saving the monthly preprocessed individual factors

[ ]:
```
monthly_mkt_rf.to_csv("../../data/processed/mkt_rf/0_monthly_mkt_rf.csv")
monthly_smb.to_csv("../../data/processed/smb/0_monthly_smb.csv")
monthly_hml.to_csv("../../data/processed/hml/0_monthly_hml.csv")
```