pca_eigen_portfolios_m2_ex3

October 25, 2018

0.1 Eigen-portfolio construction using Principal Component Analysis (PCA)

0.1.1 PCA via sklearn.decomposition using S&P 500 Index stock data

Welcome to your 2-nd assignment in Unsupervised Machine Learning in Finance.

In this assignment we look in-depth at model-free factor analysis using PCA. By model-free we mean that we do not rely on any factors such as value or momentum to decompose portfolio returns, but instead using Principal Component Analysis (PCA) to deduce structure of portfolio returns.

We work with S&P 500 index stock data.

0.2 About iPython Notebooks

iPython Notebooks are interactive coding environments embedded in a webpage. You will be using iPython notebooks in this class. You only need to write code between the ### START CODE HERE ### and ### END CODE HERE ### comments. After writing your code, you can run the cell by either pressing "SHIFT"+"ENTER" or by clicking on "Run Cell" (denoted by a play symbol) in the upper bar of the notebook.

We will often specify "(X lines of code)" in the comments to tell you about how much code you need to write. It is just a rough estimate, so don't feel bad if your code is longer or shorter.

```
In [114]: import os
          import os.path
          import numpy as np
          import datetime

          import sys
          sys.path.append("..")
          import grading

          try:
              import matplotlib.pyplot as plt
              %matplotlib inline
          except:
              pass

          try:
              import pandas as pd
```

```python
        print("  pandas: %s"% pd.__version__)
    except:
        print("Missing pandas package")
```

  pandas: 0.19.2

In [115]:
```python
### ONLY FOR GRADING. DO NOT EDIT ###
submissions=dict()
assignment_key="BBz-XobeEeegARIApDSa9g"
all_parts=["nvDA9", "ykDlW", "rpYVm","oWy6l","MWWt7","3VyJD"]
### ONLY FOR GRADING. DO NOT EDIT ###
```

In [140]:
```python
COURSERA_TOKEN = "HS7Xnr9EQucoKYNu"  # the key provided to the Student under his/her e
COURSERA_EMAIL =  "cilsya@yahoo.com"  # the email
```

In [117]:
```python
# load dataset
asset_prices = pd.read_csv(os.getcwd() + '/data/spx_holdings_and_spx_closeprice.csv',
                     date_parser=lambda dt: pd.to_datetime(dt, format='%Y-%m-%d'),
                     index_col = 0).dropna()
n_stocks_show = 12
print('Asset prices shape', asset_prices.shape)
asset_prices.iloc[:, :n_stocks_show].head()
```

Asset prices shape (3493, 419)

Out[117]:

	A	AA	AAPL	ABC	ABT	ADBE	ADI	\
2000-01-27	46.1112	78.9443	3.9286	4.5485	13.7898	15.6719	48.0313	
2000-01-28	45.8585	77.8245	3.6295	4.5485	14.2653	14.3906	47.7500	
2000-01-31	44.5952	78.0345	3.7054	4.3968	14.5730	13.7656	46.7500	
2000-02-01	47.8377	80.7640	3.5804	4.5333	14.7128	13.9688	49.0000	
2000-02-02	51.5434	83.4934	3.5290	4.5788	14.7968	15.3281	48.1250	

	ADM	ADP	ADSK	AEE	AEP
2000-01-27	10.8844	39.5477	8.1250	32.9375	33.5625
2000-01-28	10.7143	38.5627	7.7188	32.3125	33.0000
2000-01-31	10.6576	37.3807	7.6406	32.5625	33.5000
2000-02-01	10.8844	37.9717	7.9219	32.5625	33.6875
2000-02-02	10.6576	35.9032	7.9688	32.5625	33.6250

In [118]:
```python
print('Last column contains SPX index prices:')
asset_prices.iloc[:, -10:].head()
```

Last column contains SPX index prices:

Out[118]:

	STJ	SVU	SWY	TEG	TER	TGNA	THC	\
2000-01-27	5.5918	86.6178	26.3983	11.3873	65.8677	22.1921	60.9705	

```
2000-01-28  5.4520  82.4218  27.4137  11.2230  60.3487  21.7558  62.3032
2000-01-31  5.5499  86.3181  28.2444  11.0862  62.1484  22.0533  60.6373
2000-02-01  5.4240  83.0212  28.7982  11.1683  67.3674  22.2120  60.4708
2000-02-02  5.3541  81.5226  28.6136  11.1956  68.9271  22.6483  62.4698

                  X    MAR.1      SPX
2000-01-27  20.7086  12.2457  1398.56
2000-01-28  20.1183  12.0742  1360.16
2000-01-31  19.5772  12.1722  1394.46
2000-02-01  19.5772  12.5151  1409.28
2000-02-02  19.5281  12.3192  1409.12
```

Part 1 (Asset Returns Calculation) Instructions:

Calculate percent returns, also known as simple returns using asse_prices. assign the result to variable asset_returns. Keep only not-nan values in the resulting pandas.DataFrame

Calculate de-meaned returns and scale them by standard deviation σ. Assign result to normed_returns variable

We now compute stock returns and normalize stock returns data by subtracting the mean and dividing by standard diviation. This normalization is required by PCA.

In [119]:
```
asset_returns = pd.DataFrame(data=np.zeros(shape=(len(asset_prices.index), asset_price
                             columns=asset_prices.columns.values,
                             index=asset_prices.index)
normed_returns = asset_returns
### START CODE HERE ### ( 4 lines of code)
# normed_returns is pandas.DataFrame that should contain normalized returns

# Calculate percent returns, also known as simple returns using asse_prices.
# Assign the result to variable asset_returns.
asset_returns = asset_prices.pct_change()
#
# Keep only not-nan values in the resulting pandas.DataFrame
asset_returns = asset_returns.replace(np.nan, 0, regex=True)

# Calculate de-meaned returns and scale them by standard deviation.
# Assign result to normed_returns variable
# We now compute stock returns and normalize stock returns data by subtracting
# the mean and dividing by standard diviation. This normalization is required by PCA.
normed_returns = (asset_returns - asset_returns.mean(axis=0)) / asset_returns.std(axis
normed_returns = pd.DataFrame(normed_returns)

# Drop the first row
normed_returns = normed_returns.iloc[1:]

### END CODE HERE ###


normed_returns.iloc[-5:, -10:].head()
```

```
Out[119]:                  STJ       SVU       SWY       TEG       TER      TGNA  \
          2013-12-16  0.852856  0.965359 -1.169049  0.884888  0.095880  0.656736
          2013-12-17  0.275224  0.517383 -0.086115 -0.306246  0.589775 -0.118625
          2013-12-18  0.864621  0.509511  0.600804  1.210789 -0.190049  0.925596
          2013-12-19  0.210112  0.399634 -0.100170 -0.757517 -0.208051  0.304959
          2013-12-20  0.827436  0.748530  0.372500  1.048274  0.264086  0.436939

                           THC         X     MAR.1       SPX
          2013-12-16  0.180044 -0.238526  0.465122  0.468002
          2013-12-17 -0.549598  0.025277 -0.260042 -0.247953
          2013-12-18  0.757110  0.058442  0.952602  1.252886
          2013-12-19 -0.772312  1.544455 -0.167791 -0.056362
          2013-12-20  0.320691 -0.740955  0.373779  0.353914
```

```
In [120]: ### GRADED PART (DO NOT EDIT) ###
          part_1=list(normed_returns.iloc[0,: 100].as_matrix().squeeze())
          try:
              part1 = " ".join(map(repr, part_1))
          except TypeError:
              part1 = repr(part_1)
          submissions[all_parts[0]]=part1
          grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:1],all_parts,
          normed_returns.iloc[0,: 100].as_matrix().squeeze()
          ### GRADED PART (DO NOT EDIT) ###
```

```
Submission successful, please check on the coursera grader page for the status
```

```
Out[120]: array([-0.19007752, -0.51378354, -2.71508377, -0.04977229,  2.18325301,
                 -2.68450702, -0.21248702, -0.76709776, -1.54094625, -1.80419545,
                 -1.37318536, -0.99430738,  0.16138837,  0.72991552,  0.63495564,
                 -0.72142213, -0.0130274 , -0.8080893 ,  0.39929293, -0.75903493,
                 -1.43464957, -1.12799754, -1.29403324, -0.44808611, -2.14003095,
                  0.58959236, -0.87837898,  0.31433656, -1.0807495 , -0.31371438,
                  0.11821896, -1.86893679, -1.87300855, -0.22610904, -0.0418852 ,
                 -0.02135839, -0.60466587, -1.43107734, -1.16695823, -1.65616404,
                 -0.50499727, -1.51986323, -0.36364509, -0.58867176, -0.7329979 ,
                  0.87668379, -3.12454362, -1.33995546, -1.33884528, -0.53058613,
                 -1.28327282, -2.21743433,  1.75810464,  0.22819369, -0.48099914,
                 -0.21162737, -1.39183296, -1.89106681, -1.26540917, -0.90802631,
                  1.20025673, -1.13799044, -1.06749907, -1.49050331,  1.65215907,
                 -0.94853884,  3.36986014, -0.82355183,  1.76617483,  0.04145153,
                 -2.73724874, -0.93557347,  0.02500748, -0.5273333 , -0.34697322,
                 -3.31791296, -1.10547323, -0.79767652, -0.4545626 ,  1.58060226,
                 -1.05550235, -0.19734035, -0.85232869, -3.09491117, -2.41233778,
                 -0.93938364, -1.88393008, -2.73747984, -2.97119678, -0.52327684,
                 -0.71140214,  2.02611986, -1.26177708, -3.24599972, -1.04376025,
                 -0.21377559,  0.86667266, -0.53482316,  0.92667422, -0.51031526])
```

```
In [121]: train_end = datetime.datetime(2012, 3, 26)

          df_train = None
          df_test = None
          df_raw_train = None
          df_raw_test = None

          df_train = normed_returns[normed_returns.index <= train_end].copy()
          df_test = normed_returns[normed_returns.index > train_end].copy()

          df_raw_train = asset_returns[asset_returns.index <= train_end].copy()
          df_raw_test = asset_returns[asset_returns.index > train_end].copy()

          print('Train dataset:', df_train.shape)
          print('Test dataset:', df_test.shape)
```

```
Train dataset: (3055, 419)
Test dataset: (437, 419)
```

Now we compute PCA using all available data. Once we do have PCA computed we fix variance explained at some number and see what is the smallest number of components needed to explain this variance.

Part 2 (PCA fitting) **Instructions:** - Calculate covariance matrix using training data set, i.e. **df_train** for all assets. Assign results to **cov_matrix**. - Calculate covariance matrix using training data set, i.e. **df_raw_train** for all assets. Assign results to **cov_matrix_raw**. - Use scikit-learn PCA to fit PCA model to **cov_matrix**. Assign fitted model to **pca**

```
In [122]: import sklearn.decomposition
          import seaborn as sns

          stock_tickers = normed_returns.columns.values[:-1]
          assert 'SPX' not in stock_tickers, "By accident included SPX index"

          n_tickers = len(stock_tickers)
          pca = None
          cov_matrix = pd.DataFrame(data=np.ones(shape=(n_tickers, n_tickers)), columns=stock_ti
          cov_matrix_raw = cov_matrix

          if df_train is not None and df_raw_train is not None:
              stock_tickers = asset_returns.columns.values[:-1]
              assert 'SPX' not in stock_tickers, "By accident included SPX index"

              ### START CODE HERE ### ( 2-3 lines of code)

              # computing PCA on S&P 500 stocks
```

```
# Calculate covariance matrix using training data set,
# i.e. df_train for all assets. Assign results to cov_matrix.
#cov_matrix=df_train.cov()
cov_matrix = df_train[stock_tickers].cov()

# Calculate covariance matrix using training data set,
# i.e. df_raw_train for all assets. Assign results to cov_matrix_raw.
cov_matrix_raw=df_raw_train[stock_tickers].cov()

# Use scikit-learn PCA to fit PCA model to cov_matrix. Assign fitted model to pca
clf = sklearn.decomposition.PCA()
pca = clf.fit(cov_matrix)

# not normed covariance matrix

### END CODE HERE ###

cov_raw_df = pd.DataFrame({'Variance': np.diag(cov_matrix_raw)}, index=stock_ticke
# cumulative variance explained
var_threshold = 0.8
var_explained = np.cumsum(pca.explained_variance_ratio_)
num_comp = np.where(np.logical_not(var_explained < var_threshold))[0][0] + 1  # +1
print('%d components explain %.2f%% of variance' %(num_comp, 100* var_threshold))
```

```
4 components explain 80.00% of variance
```

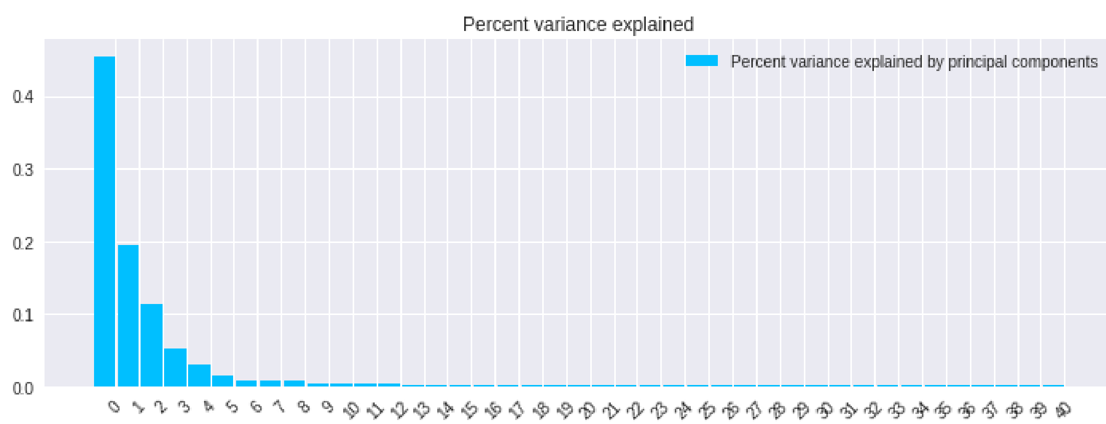
In [123]:
```
### GRADED PART (DO NOT EDIT) ###
part_2 = np.diag(cov_matrix[: 100])
try:
    part2 = " ".join(map(repr, part_2))
except TypeError:
    part2 = repr(part_2)
submissions[all_parts[1]]=part2
grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:2],all_parts,
### GRADED PART (DO NOT EDIT) ###
np.diag(cov_matrix[: 100])
```

```
Submission successful, please check on the coursera grader page for the status
```

Out[123]:
```
array([ 1.10478242,  1.09455432,  1.08221062,  1.10548586,  1.06972085,
        1.10629519,  1.11901325,  1.08425013,  1.09834527,  1.06621229,
        1.07829579,  1.10771015,  1.12450535,  1.10444381,  1.07751955,
        1.11984646,  1.11539247,  1.1071917 ,  1.04857046,  1.10832658,
        1.1051169 ,  1.04327362,  1.07497386,  1.12542467,  1.10863228,
        1.09149441,  1.08449312,  1.02697733,  1.09840268,  1.08537602,
        1.0805351 ,  1.08147741,  1.0962246 ,  0.99836261,  1.11100475,
        1.01462408,  1.10392495,  1.06629289,  1.11035646,  1.08830348,
```

```
                 1.08267576,  1.0942163 ,  1.08520176,  1.10535237,  0.99878741,
                 1.08378042,  1.10224712,  1.08963727,  1.08908082,  1.09592188,
                 1.10310166,  1.09182003,  1.07098094,  1.11227998,  1.07335399,
                 1.10657054,  1.10486349,  1.11563753,  1.06738213,  1.08956209,
                 1.07238537,  1.08182671,  1.11571354,  1.09594673,  1.0994683 ,
                 1.10130047,  1.09801796,  1.05345536,  1.08266256,  1.1045181 ,
                 1.10797538,  1.08555709,  1.02560735,  1.10627165,  1.10368709,
                 1.10945548,  1.08744706,  1.11857364,  1.1185181 ,  1.08153319,
                 1.11718141,  1.05624895,  1.09646017,  1.10243721,  1.06202598,
                 1.09049077,  1.09369544,  1.11218236,  1.04809318,  1.09233857,
                 1.09220974,  1.10276995,  1.09400944,  1.09430711,  1.09951659,
                 0.92383279,  1.0996432 ,  1.05928944,  1.08108677,  1.09932206])
```

```
In [124]: if pca is not None:
              bar_width = 0.9
              n_asset = int((1 / 10) * normed_returns.shape[1])
              x_indx = np.arange(n_asset)
              fig, ax = plt.subplots()
              fig.set_size_inches(12, 4)
              # Eigenvalues are measured as percentage of explained variance.
              rects = ax.bar(x_indx, pca.explained_variance_ratio_[:n_asset], bar_width, color='
              ax.set_xticks(x_indx + bar_width / 2)
              ax.set_xticklabels(list(range(n_asset)), rotation=45)
              ax.set_title('Percent variance explained')
              ax.legend((rects[0],), ('Percent variance explained by principal components',))
```




```
In [125]: if pca is not None:
              projected = pca.fit_transform(cov_matrix)
```

Part 3 (Eigen-portfolios construction) **Instructions:**

We now look a the first two eigen portfolios. We use definition of eigen portfolios as provided by Avellaneda http://math.nyu.edu/faculty/avellane/AvellanedaLeeStatArb20090616.pdf

Following Avellaneda we define eigen portfolio weights as:

$$Q_i^{(j)} = \frac{v_i^{(j)}}{\sigma_i}$$

where j is the index of eigen portfolio and v_i is the i-th element of j-th eigen vector.

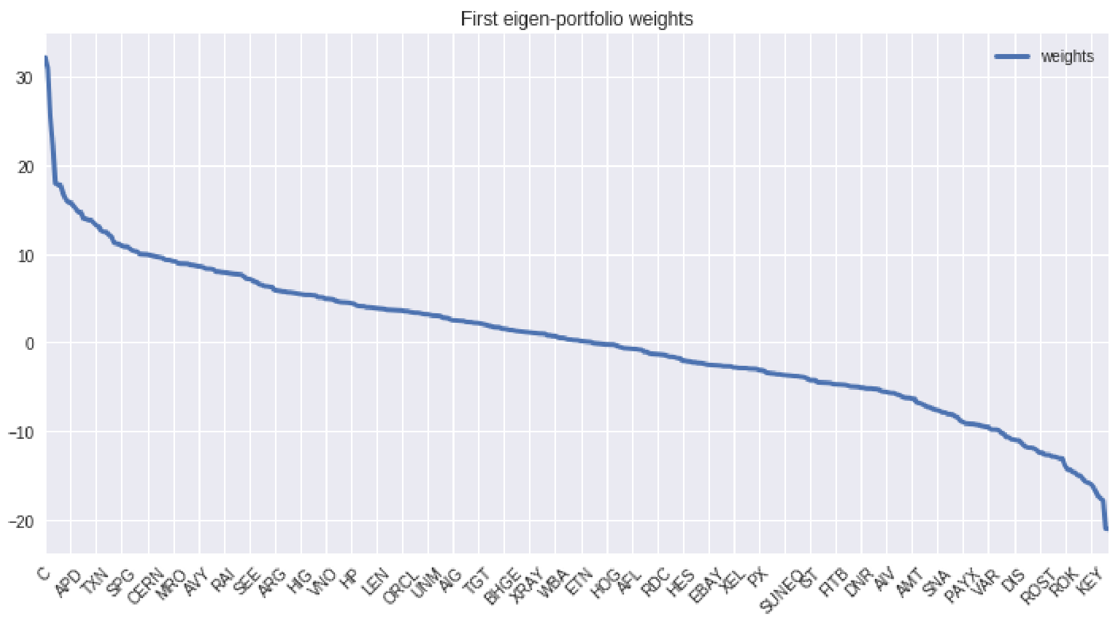
In the code the pca.components_ are the Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained_variance_.

Hint: do not forget to normalize portfolio wieghts such they sum up to 1.

Assign **pc_w** to be weights of the first eigen portfolio.

In [126]:
```
# the first two eigen-portfolio weights# the fi
# first component
# get the Principal components
pc_w = np.zeros(len(stock_tickers))
eigen_prtf1 = pd.DataFrame(data ={'weights': pc_w.squeeze()*100}, index = stock_ticker
if pca is not None:
    pcs = pca.components_

    ### START CODE HERE ### ( 1-2 lines of code)
    # normalized to 1

    # NOTE: You use 0 because it is the first portfolio
    pc_w = pcs[:,0] / np.sum(pcs[:,0])


    ### END CODE HERE ###

    eigen_prtf1 = pd.DataFrame(data ={'weights': pc_w.squeeze()*100}, index = stock_ti
    eigen_prtf1.sort_values(by=['weights'], ascending=False, inplace=True)
    print('Sum of weights of first eigen-portfolio: %.2f' % np.sum(eigen_prtf1))
    eigen_prtf1.plot(title='First eigen-portfolio weights',
                     figsize=(12,6),
                     xticks=range(0, len(stock_tickers),10),
                     rot=45,
                     linewidth=3)
```

```
Sum of weights of first eigen-portfolio: 100.00
```




```
In [127]: ### GRADED PART (DO NOT EDIT) ###
          part_3 = list(eigen_prtf1.squeeze().values)
          try:
              part3 = " ".join(map(repr, part_3))
          except TypeError:
              part3 = repr(part_3)
          submissions[all_parts[2]]=part3
          grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:3],all_parts,
          eigen_prtf1.squeeze().values
          ### GRADED PART (DO NOT EDIT) ###
```

```
Submission successful, please check on the coursera grader page for the status
```

```
Out[127]: array([ 32.15547082,  30.93331571,  25.50690946,  22.16692719,
                  18.0084551 ,  17.79274279,  17.76739043,  16.81887906,
                  16.18044798,  15.83751903,  15.80622057,  15.43514505,
                  15.08183151,  14.72599396,  14.69233946,  14.01724021,
                  13.94498039,  13.82736864,  13.79893355,  13.50764428,
                  13.22700267,  13.07689841,  12.60979677,  12.49351237,
                  12.4903706 ,  12.13617091,  11.9322199 ,  11.25764885,
                  11.18816081,  11.09465525,  10.92698727,  10.82310861,
                  10.81853368,  10.64480777,  10.42272303,  10.32425886,
                  10.25665844,   9.99412232,   9.97134908,   9.96661664,
                   9.93448604,   9.86862787,   9.80218276,   9.71240557,
                   9.65882792,   9.62251218,   9.55778555,   9.33968423,
                   9.31616556,   9.29653033,   9.17835941,   9.14977202,
```

```
 8.96821037,   8.9177161 ,   8.91322329,   8.89646924,
 8.8799625 ,   8.75396254,   8.73041028,   8.66509701,
 8.59805673,   8.56422707,   8.49296757,   8.34200401,
 8.31680707,   8.3057816 ,   8.24337229,   8.01697141,
 8.01096448,   7.95181494,   7.92632961,   7.89394253,
 7.82586628,   7.77922591,   7.73977603,   7.7102848 ,
 7.67616568,   7.65502354,   7.46049124,   7.16904627,
 7.16499153,   7.07692559,   6.88752121,   6.79697204,
 6.57166439,   6.51115034,   6.35907629,   6.34806269,
 6.2959234 ,   6.26471226,   5.91766594,   5.87986826,
 5.83148134,   5.75485922,   5.75154895,   5.67796318,
 5.66518754,   5.63156086,   5.58097947,   5.52810798,
 5.48664341,   5.42941712,   5.39066262,   5.38534655,
 5.38016212,   5.33456593,   5.32420457,   5.12045453,
 5.11200473,   5.08350238,   4.94674916,   4.94186195,
 4.89834571,   4.87174841,   4.6964071 ,   4.61284181,
 4.55988048,   4.54195027,   4.52345153,   4.50194742,
 4.44229078,   4.39519593,   4.18692015,   4.11507943,
 4.10315254,   4.07944548,   3.98368565,   3.98232014,
 3.95128997,   3.91389828,   3.88748504,   3.84520555,
 3.80683415,   3.78278605,   3.70337093,   3.68669643,
 3.66231812,   3.64800547,   3.61877551,   3.6156949 ,
 3.60230469,   3.53728682,   3.53400538,   3.45089396,
 3.41445369,   3.36509603,   3.36246114,   3.31898242,
 3.24149733,   3.19563602,   3.18543828,   3.12807285,
 3.0586226 ,   3.04506868,   3.0292527 ,   2.98808882,
 2.819787  ,   2.77915188,   2.74135916,   2.60589448,
 2.521452  ,   2.50983641,   2.47472813,   2.4596093 ,
 2.44461919,   2.3367548 ,   2.32204847,   2.2997673 ,
 2.22884839,   2.21381473,   2.19521506,   2.13298126,
 2.0549353 ,   1.99192831,   1.9118519 ,   1.8125287 ,
 1.74306812,   1.71493054,   1.70515543,   1.57693067,
 1.54512793,   1.51493463,   1.42282797,   1.42173825,
 1.35874791,   1.3038053 ,   1.26817757,   1.22984722,
 1.18228405,   1.17452971,   1.11102132,   1.09734363,
 1.05287933,   1.0367257 ,   1.0088328 ,   1.00585178,
 0.97290835,   0.81703713,   0.81121118,   0.73682225,
 0.7274888 ,   0.59832347,   0.52122702,   0.51450865,
 0.46007593,   0.36156475,   0.34605725,   0.29243735,
 0.26727476,   0.26189998,   0.1773287 ,   0.15123653,
 0.10719077,   0.08639683,   0.05970625,  -0.05573653,
-0.08692352,  -0.11208059,  -0.15119666,  -0.18794833,
-0.21559107,  -0.21614727,  -0.23280606,  -0.24093846,
-0.32796309,  -0.46555867,  -0.53203284,  -0.62553826,
-0.64875671,  -0.67843623,  -0.68739994,  -0.72023649,
-0.74362167,  -0.80231265,  -0.81925744,  -1.04238522,
-1.05617989,  -1.2189019 ,  -1.27562113,  -1.29537084,
-1.2980971 ,  -1.34253224,  -1.38819411,  -1.41692674,
```

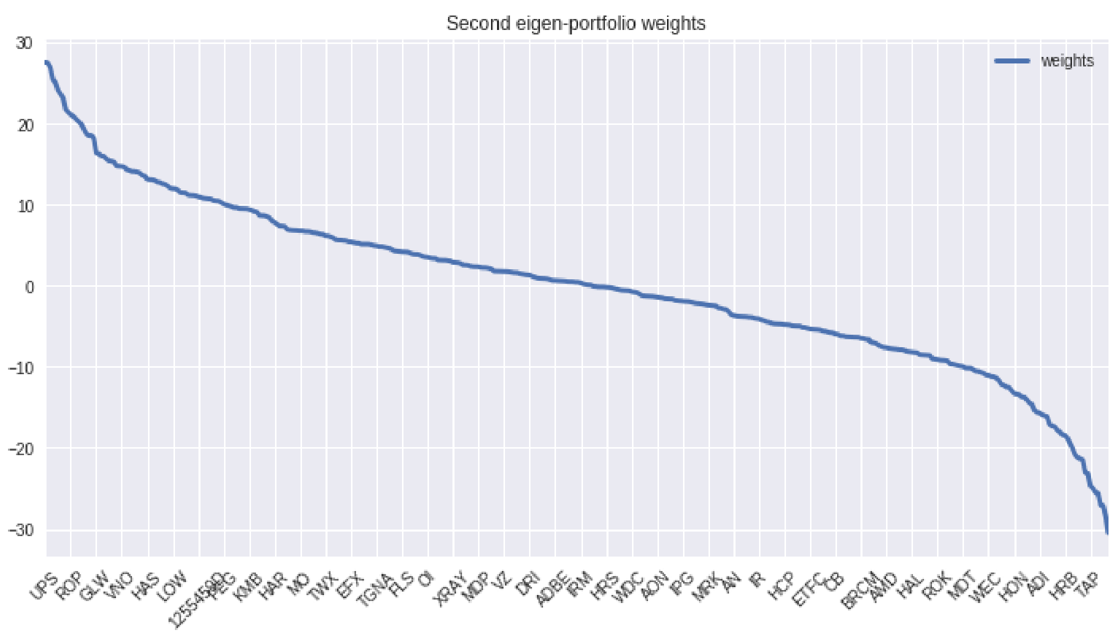
```
 -1.51100274,  -1.59591389,  -1.60325386,  -1.65657878,
 -1.7540267 ,  -1.75912442,  -2.04598586,  -2.04650266,
 -2.10900163,  -2.14468095,  -2.22780657,  -2.22937006,
 -2.29880875,  -2.31455323,  -2.35359176,  -2.4283302 ,
 -2.49393071,  -2.51540113,  -2.55182815,  -2.57177059,
 -2.57885033,  -2.59468997,  -2.65383346,  -2.65590554,
 -2.67790177,  -2.67807124,  -2.79735343,  -2.8159846 ,
 -2.86844249,  -2.87283021,  -2.87836277,  -2.88932058,
 -2.94093753,  -2.95431688,  -2.95621033,  -2.97890885,
 -3.09008828,  -3.12567731,  -3.20524044,  -3.40977088,
 -3.43664574,  -3.48764722,  -3.50659075,  -3.57235712,
 -3.5739352 ,  -3.6365622 ,  -3.6620914 ,  -3.69772069,
 -3.71058234,  -3.75390851,  -3.78702451,  -3.78878482,
 -3.85135599,  -3.87640647,  -3.92661548,  -4.08953478,
 -4.24189207,  -4.24681924,  -4.25226134,  -4.49474819,
 -4.50381198,  -4.53206282,  -4.53700259,  -4.55951819,
 -4.57476545,  -4.67124556,  -4.7117251 ,  -4.71289825,
 -4.7334616 ,  -4.75164045,  -4.78592493,  -4.87531123,
 -4.94821379,  -4.96532081,  -4.99110688,  -4.99895873,
 -5.0770602 ,  -5.09564626,  -5.17411777,  -5.18415817,
 -5.18746594,  -5.22827789,  -5.23839093,  -5.30124274,
 -5.48543984,  -5.53514696,  -5.55614567,  -5.64941817,
 -5.6906019 ,  -5.69375103,  -5.83697955,  -5.92611836,
 -6.07688579,  -6.18943918,  -6.24404286,  -6.24885578,
 -6.33084848,  -6.35275967,  -6.7871732 ,  -6.83516854,
 -6.93653265,  -7.09531114,  -7.25793205,  -7.30919606,
 -7.4577652 ,  -7.5833315 ,  -7.6359054 ,  -7.71905909,
 -7.8885551 ,  -7.89465967,  -8.06153001,  -8.06764322,
 -8.14697999,  -8.3439122 ,  -8.44648366,  -8.82363324,
 -8.97023948,  -9.12272994,  -9.13866584,  -9.17715331,
 -9.1826925 ,  -9.24600763,  -9.28651925,  -9.39579099,
 -9.46280065,  -9.51774542,  -9.54194637,  -9.81251943,
 -9.81724376,  -9.81930154,  -9.90087072, -10.17343146,
-10.29734916, -10.66160996, -10.68232544, -10.92937531,
-10.94272105, -11.04029044, -11.0423272 , -11.3280597 ,
-11.61501431, -11.81124124, -11.86465393, -11.86537868,
-11.98459368, -12.16712784, -12.41847219, -12.44025296,
-12.63369804, -12.6803311 , -12.68576713, -12.8600044 ,
-12.87611734, -12.96894765, -13.06258295, -13.07790021,
-13.87866761, -14.31076478, -14.34915142, -14.61401339,
-14.65577859, -14.99299063, -15.0215979 , -15.37692364,
-15.67342681, -15.7902799 , -15.93650179, -16.22205681,
-16.77748843, -17.31302391, -17.60036091, -17.79534609,
-21.00083593, -21.04153965])
```

We sort the first two eigen portfolio weights and plot the results.

```
In [128]: pc_w = np.zeros(len(stock_tickers))
          eigen_prtf2 = pd.DataFrame(data ={'weights': pc_w.squeeze()*100}, index = stock_ticker
```

```
if pca is not None:
    pcs = pca.components_

    ### START CODE HERE ### ( 1-2 lines of code)
    # normalized to 1

    # NOTE: You use 1 because it is the second portfolio
    pc_w = pcs[:,1] / np.sum(pcs[:,1])


    ### END CODE HERE ###

    eigen_prtf2 = pd.DataFrame(data ={'weights': pc_w.squeeze()*100}, index = stock_ti
    eigen_prtf2.sort_values(by=['weights'], ascending=False, inplace=True)
    print('Sum of weights of second eigen-portfolio: %.2f' % np.sum(eigen_prtf2))
    eigen_prtf2.plot(title='Second eigen-portfolio weights',
                     figsize=(12,6),
                     xticks=range(0, len(stock_tickers),10),
                     rot=45,
                     linewidth=3)
```

```
Sum of weights of second eigen-portfolio: 100.00
```




```
In [129]: ### GRADED PART (DO NOT EDIT) ###
          part_4 = list(eigen_prtf2.as_matrix().squeeze())
```

```python
    try:
        part4 = " ".join(map(repr, part_4))
    except TypeError:
        part4 = repr(part_4)
    submissions[all_parts[3]]=part4
    grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:4],all_parts,
    eigen_prtf2.as_matrix().squeeze()
    ### GRADED PART (DO NOT EDIT) ###
```

```
Submission successful, please check on the coursera grader page for the status
```

```
Out[129]: array([ 27.52926321,  27.44201015,  26.91912098,  25.43007608,
                  25.02939292,  24.120357  ,  23.62815802,  23.13556471,
                  21.73435704,  21.38917326,  21.10303677,  20.86893243,
                  20.56895902,  20.26671208,  19.97031181,  19.41576762,
                  18.77654354,  18.51692484,  18.49032438,  18.25195368,
                  16.3910445 ,  16.25365337,  15.97909736,  15.96398355,
                  15.63040988,  15.40128862,  15.34351604,  15.2496275 ,
                  14.77600677,  14.73057796,  14.70733294,  14.63299748,
                  14.31693031,  14.2116605 ,  14.08525619,  14.08329266,
                  14.0149747 ,  13.91668447,  13.62954479,  13.53923653,
                  13.13017796,  13.07568068,  13.07272994,  12.9622257 ,
                  12.76949114,  12.73596898,  12.5662838 ,  12.47754512,
                  12.29017578,  12.04493395,  11.96548822,  11.93953123,
                  11.79244045,  11.50787112,  11.45576397,  11.4510577 ,
                  11.20105258,  11.14540149,  11.13534306,  11.0941578 ,
                  10.99691232,  10.91003565,  10.78051307,  10.74277639,
                  10.72036228,  10.677604  ,  10.50431215,  10.4611124 ,
                  10.43117792,  10.26058325,  10.0835625 ,   9.96058247,
                   9.86144938,   9.76362435,   9.63616744,   9.62966767,
                   9.50427713,   9.49827203,   9.49531784,   9.47010792,
                   9.39384041,   9.31133865,   9.1432615 ,   9.08144116,
                   8.68210878,   8.65381694,   8.62012833,   8.51964647,
                   8.37800858,   7.99095624,   7.79924779,   7.57230966,
                   7.35259256,   7.33912619,   7.30036767,   6.93346945,
                   6.86392906,   6.83223421,   6.82861925,   6.80344896,
                   6.76155202,   6.75937969,   6.69135526,   6.6862127 ,
                   6.64523378,   6.55137736,   6.53315186,   6.43945212,
                   6.34829503,   6.32582866,   6.15097015,   6.1287244 ,
                   6.01587807,   5.90894473,   5.66887862,   5.62312012,
                   5.62011783,   5.58206871,   5.54561524,   5.42462665,
                   5.41192572,   5.34353349,   5.25517779,   5.24634084,
                   5.12256836,   5.12228464,   5.11217363,   5.11128155,
                   5.01536194,   4.94193491,   4.91370978,   4.81914589,
                   4.81731161,   4.74579387,   4.67054008,   4.61979546,
                   4.45586522,   4.29749281,   4.27227268,   4.22492054,
                   4.18200613,   4.17999509,   4.1592227 ,   4.039383  ,
```

```
 3.92147927,   3.87512349,   3.87121129,   3.77659622,
 3.62909013,   3.55469932,   3.51447332,   3.4263402 ,
 3.37936863,   3.36656712,   3.17825594,   3.15446008,
 3.14745757,   3.14099194,   3.10078622,   3.04217461,
 2.89064905,   2.86680452,   2.83197991,   2.74044616,
 2.55089784,   2.54722793,   2.5091511 ,   2.38641479,
 2.34648275,   2.33820598,   2.33152566,   2.23579226,
 2.22443649,   2.21378853,   2.15393363,   2.06351413,
 1.79627931,   1.78086927,   1.77533257,   1.76639383,
 1.74878934,   1.74225802,   1.708953  ,   1.66057453,
 1.60796373,   1.60700772,   1.4993662 ,   1.44491936,
 1.3791175 ,   1.34266129,   1.31519353,   1.17964624,
 1.0689554 ,   0.94963963,   0.91400558,   0.90343093,
 0.87723101,   0.84195711,   0.76108907,   0.66350373,
 0.64488039,   0.60576184,   0.59827384,   0.56973822,
 0.56060465,   0.51314868,   0.50727247,   0.49058884,
 0.4473627 ,   0.44574174,   0.35987965,   0.22441011,
 0.15272366,   0.10523943,   0.08438262,  -0.06919778,
-0.08262796,  -0.13600845,  -0.13846557,  -0.14337321,
-0.17760186,  -0.20933811,  -0.24102944,  -0.32808924,
-0.39524808,  -0.48438356,  -0.54929592,  -0.57905245,
-0.59199663,  -0.60821956,  -0.71407626,  -0.77242903,
-0.83666635,  -0.99914715,  -1.19680969,  -1.25537772,
-1.27325858,  -1.28616543,  -1.30759622,  -1.34993192,
-1.38764156,  -1.44988295,  -1.49608529,  -1.59856076,
-1.6151462 ,  -1.64594459,  -1.65679399,  -1.79213569,
-1.82924952,  -1.87295074,  -1.9098851 ,  -1.93193133,
-1.93399655,  -1.99499376,  -2.05804923,  -2.14036448,
-2.17146171,  -2.18578111,  -2.25088304,  -2.31288936,
-2.38476143,  -2.41317099,  -2.43373236,  -2.47881086,
-2.72889581,  -2.74951669,  -2.86593054,  -2.92164252,
-3.1717942 ,  -3.54009819,  -3.65360946,  -3.74026189,
-3.77874458,  -3.79049154,  -3.82165783,  -3.83063327,
-3.88076016,  -3.88999548,  -3.98908973,  -4.04224315,
-4.06859184,  -4.22484159,  -4.31088886,  -4.44456227,
-4.4848678 ,  -4.62328989,  -4.68698339,  -4.69031898,
-4.71479618,  -4.73870457,  -4.78646229,  -4.79490684,
-4.84060919,  -4.94607849,  -4.96846506,  -4.97861509,
-4.98447109,  -5.13182022,  -5.1382051 ,  -5.22459046,
-5.32975098,  -5.35078195,  -5.39142971,  -5.40521469,
-5.44395953,  -5.56256634,  -5.60875339,  -5.71129815,
-5.77320996,  -5.79956872,  -5.96079911,  -5.99640283,
-6.17184532,  -6.17688501,  -6.27297325,  -6.27937755,
-6.3053169 ,  -6.33326026,  -6.33505007,  -6.34987006,
-6.45988537,  -6.49884017,  -6.59605818,  -6.60830844,
-6.9909809 ,  -7.02131025,  -7.112174  ,  -7.33294522,
-7.48874914,  -7.58619698,  -7.60886783,  -7.70452093,
-7.7490464 ,  -7.78623098,  -7.80920056,  -7.87460157,
```

```
 -7.87469804,  -7.95225813,  -8.09483169,  -8.13040031,
 -8.17983029,  -8.21796319,  -8.25092311,  -8.49161445,
 -8.51718107,  -8.52135544,  -8.5827165 ,  -8.58722795,
 -8.99724134,  -9.0396622 ,  -9.09507774,  -9.15713535,
 -9.17385307,  -9.21039316,  -9.26548117,  -9.63600933,
 -9.66661559,  -9.75493865,  -9.82205863,  -9.87865416,
 -9.95516894, -10.13000507, -10.15177999, -10.15886124,
-10.33906892, -10.5144022 , -10.53528029, -10.66655129,
-10.75809131, -10.96610692, -11.02478956, -11.16876458,
-11.19110666, -11.34532709, -11.62577146, -12.14716067,
-12.23493346, -12.46772486, -12.5101109 , -12.87649127,
-13.2325826 , -13.35474751, -13.39988886, -13.69954533,
-13.71732993, -14.02147389, -14.39926939, -14.58764325,
-15.32813128, -15.59512407, -15.64964847, -15.87490214,
-16.02546614, -16.10158561, -17.08401191, -17.26615798,
-17.36422146, -17.78048478, -18.01513329, -18.38220207,
-18.40594703, -18.76652469, -19.43088749, -20.05849876,
-20.83220959, -21.14454234, -21.31983712, -21.44504692,
-23.02174099, -23.11863215, -24.63771375, -24.9307291 ,
-25.46025358, -25.56138179, -27.05898262, -27.0774758 ,
-28.39394627, -30.44830912])
```

Part 4 (Compute performance of several eigen portfolios) **Instructions:** - Implement sharpe_ratio() function. The function takes ts_returns argument of type pd.Series and returns a tuple of annualized return, annualized vol, and annualized sharpe ratio, where sharpe ratio is defined as annualized return divided by annualized volatility - find portfolio (an index into sharpe_metric) that has the highest sharpe ratio

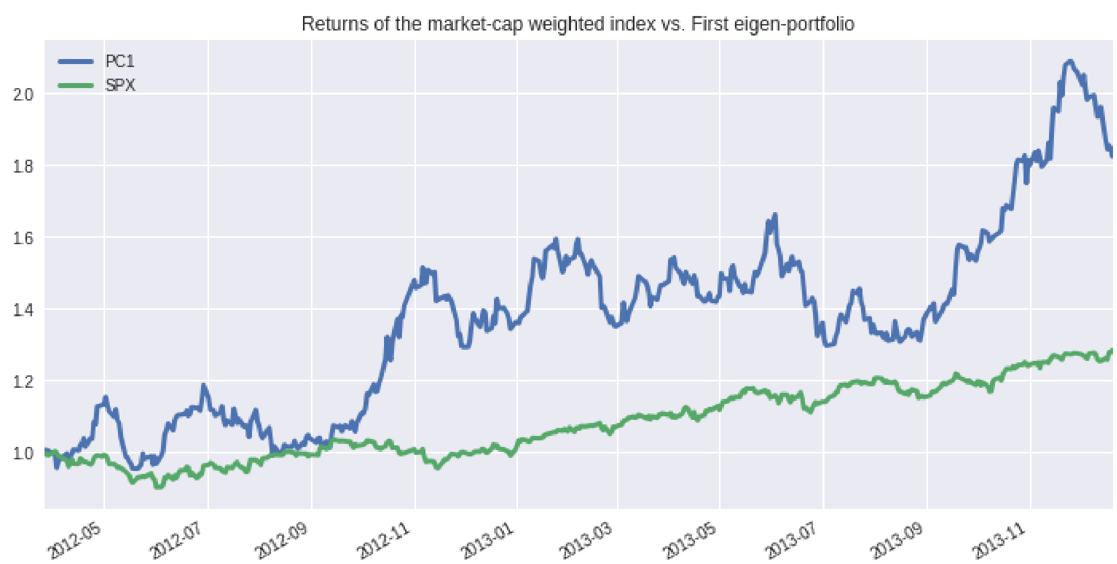
```
In [130]: def sharpe_ratio(ts_returns, periods_per_year=252):
              """
              sharpe_ratio - Calculates annualized return, annualized vol, and annualized sharpe
                                where sharpe ratio is defined as annualized return divided by annu

              Arguments:
              ts_returns - pd.Series of returns of a single eigen portfolio

              Return:
              a tuple of three doubles: annualized return, volatility, and sharpe ratio
              """

              annualized_return = 0.
              annualized_vol = 0.
              annualized_sharpe = 0.

              ### START CODE HERE ### ( 4-5 lines of code)
              ### ...
              n_years = ts_returns.shape[0]/periods_per_year
              annualized_return = np.power(np.prod(1+ts_returns),(1/n_years))-1
```

```
    annualized_vol = ts_returns.std() * np.sqrt(periods_per_year)
    annualized_sharpe = annualized_return / annualized_vol

    ### END CODE HERE ###

    return annualized_return, annualized_vol, annualized_sharpe
```

We compute the annualized return, volatility, and Sharpe ratio of the first two eigen portfolios.

```
In [131]: if df_raw_test is not None:
              eigen_prtf1_returns = np.dot(df_raw_test.loc[:, eigen_prtf1.index], eigen_prtf1 /
              eigen_prtf1_returns = pd.Series(eigen_prtf1_returns.squeeze(), index=df_test.index
              er, vol, sharpe = sharpe_ratio(eigen_prtf1_returns)
              print('First eigen-portfolio:\nReturn = %.2f%%\nVolatility = %.2f%%\nSharpe = %.2f
              year_frac = (eigen_prtf1_returns.index[-1] - eigen_prtf1_returns.index[0]).days /

              df_plot = pd.DataFrame({'PC1': eigen_prtf1_returns, 'SPX': df_raw_test.loc[:, 'SPX
              np.cumprod(df_plot + 1).plot(title='Returns of the market-cap weighted index vs. F
                                           figsize=(12,6), linewidth=3)
```

```
First eigen-portfolio:
Return = 41.39%
Volatility = 31.50%
Sharpe = 1.31
```




```
In [132]: if df_raw_test is not None:
              eigen_prtf2_returns = np.dot(df_raw_test.loc[:, eigen_prtf2.index], eigen_prtf2 /
              eigen_prtf2_returns = pd.Series(eigen_prtf2_returns.squeeze(), index=df_test.index
              er, vol, sharpe = sharpe_ratio(eigen_prtf2_returns)
              print('Second eigen-portfolio:\nReturn = %.2f%%\nVolatility = %.2f%%\nSharpe = %.2
```

```
Second eigen-portfolio:
Return = 15.76%
Volatility = 42.84%
Sharpe = 0.37
```

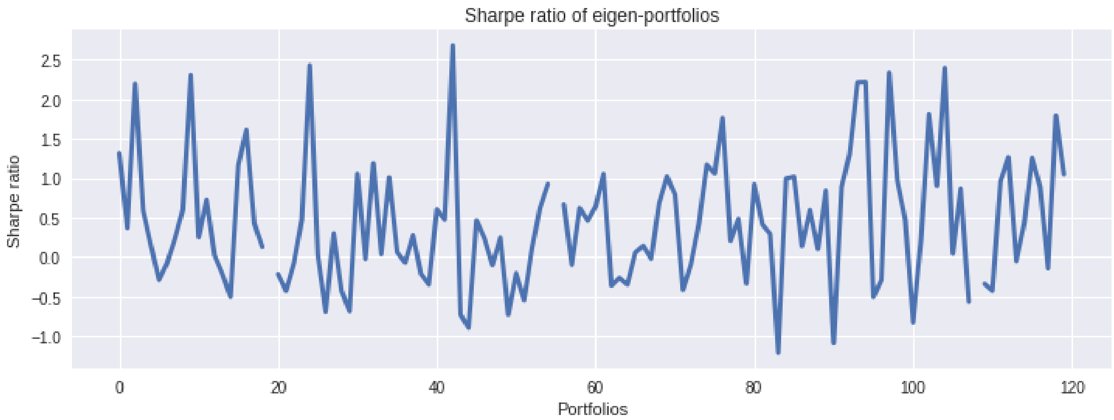
We repeat the exercise of computing Sharpe ratio for the first N portfolios and select portfolio with the highest postive Sharpe ratio.

```
In [133]: n_portfolios = 120
          annualized_ret = np.array([0.] * n_portfolios)
          sharpe_metric = np.array([0.] * n_portfolios)
          annualized_vol = np.array([0.] * n_portfolios)
          idx_highest_sharpe = 0 # index into sharpe_metric which identifies a portfolio with rh

          if pca is not None:
              for ix in range(n_portfolios):

                  ### START CODE HERE ### ( 4-5 lines of code)
                  pc_w = pcs[:,ix] / np.sum(pcs[:,ix])

                  eigen_prtf = pd.DataFrame(data ={'weights': pc_w}, index = stock_tickers)
                  eigen_returns = np.dot(df_raw_test.loc[:, eigen_prtf.index], eigen_prtf  )
                  eigen_returns = pd.Series(eigen_returns.squeeze(), index=df_test.index)
                  annualized_ret[ix], annualized_vol[ix], sharpe_metric[ix] = sharpe_ratio( eige

                  ### END CODE HERE ###


              # find portfolio with the highest Sharpe ratio
              ### START CODE HERE ### ( 2-3 lines of code)
              ### ...
              results = pd.DataFrame(data={'Return': annualized_ret, 'Vol': annualized_vol, 'Sha
              results.sort_values(by=['Sharpe'], ascending=False, inplace=True)
              idx_highest_sharpe = results.index[0]

              ### END CODE HERE ###

              print('Eigen portfolio #%d with the highest Sharpe. Return %.2f%%, vol = %.2f%%, S
                    (idx_highest_sharpe,
                     annualized_ret[idx_highest_sharpe]*100,
                     annualized_vol[idx_highest_sharpe]*100,
                     sharpe_metric[idx_highest_sharpe]))

              fig, ax = plt.subplots()
              fig.set_size_inches(12, 4)
              ax.plot(sharpe_metric, linewidth=3)
              ax.set_title('Sharpe ratio of eigen-portfolios')
```

```
            ax.set_ylabel('Sharpe ratio')
            ax.set_xlabel('Portfolios')
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel/__main__.py:20: RuntimeWarning: invalid value e
```

```
Eigen portfolio #42 with the highest Sharpe. Return 61.14%, vol = 22.80%, Sharpe = 2.68
```



In [135]:
```
results = pd.DataFrame(data={'Return': annualized_ret, 'Vol': annualized_vol, 'Sharpe'
results.sort_values(by=['Sharpe'], ascending=False, inplace=True)
results.head(10)
```

Out[135]:

	Return	Sharpe	Vol
42	0.611434	2.681348	0.228032
24	1.032188	2.431337	0.424535
104	0.512464	2.398724	0.213640
97	1.425566	2.337932	0.609755
9	0.753551	2.306594	0.326694
94	0.502024	2.221589	0.225975
93	0.601081	2.216771	0.271152
2	0.453435	2.198514	0.206246
102	0.274141	1.813004	0.151208
118	0.874385	1.793424	0.487551

In [138]:
```
# https://www.coursera.org/learn/fundamentals-machine-learning-in-finance/discussions/
# 1) Use the COLUMNS in the pcs matrix to get the weights for the eigenportfolios, alt
# 2) Calculate annualized returns in the function sharpe_ratio() using the GEOMETRIC M
# 3) You will get NaN's in the results array (most likely for 3 eigenportfolios). Ther
# If you want to read the detailed discussion that led up to these insights:
# https://www.coursera.org/learn/fundamentals-machine-learning-in-finance/discussions/
results.dropna(inplace=True)
```

In [141]:
```
### GRADED PART (DO NOT EDIT) ###
part_5 = list(results.iloc[:, 1].values.squeeze())
try:
    part5 = " ".join(map(repr, part_5))
except TypeError:
    part5 = repr(part_5)
submissions[all_parts[4]]=part5
grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:5],all_parts,
results.iloc[:, 1].values.squeeze()
### GRADED PART (DO NOT EDIT) ###
```

```
Submission successful, please check on the coursera grader page for the status
```

Out[141]:
```
array([ 2.6813484 ,  2.43133729,  2.39872441,  2.33793196,  2.30659408,
        2.2215888 ,  2.21677072,  2.19851392,  1.81300448,  1.79342355,
        1.76312102,  1.61425529,  1.31396078,  1.30787333,  1.26660644,
        1.25886217,  1.18686551,  1.17035236,  1.16468931,  1.06187955,
        1.05578097,  1.05505619,  1.04882183,  1.02165311,  1.01830017,
        1.00828466,  0.9950413 ,  0.97529078,  0.95814138,  0.92726174,
        0.92483908,  0.9041755 ,  0.892538  ,  0.88642444,  0.86855357,
        0.84287489,  0.79154656,  0.72805392,  0.68482588,  0.66836196,
        0.6409441 ,  0.62474111,  0.62004933,  0.60334582,  0.60023437,
        0.59475285,  0.59099198,  0.48426658,  0.47799353,  0.4744462 ,
        0.47279402,  0.4633251 ,  0.46119895,  0.42591032,  0.42364012,
        0.4154465 ,  0.41216641,  0.36782358,  0.30061062,  0.29467349,
        0.27701825,  0.25218297,  0.24688822,  0.23702365,  0.23110684,
        0.22909072,  0.20739045,  0.14233782,  0.14092441,  0.14041398,
        0.12992884,  0.11645889,  0.10386647,  0.06261485,  0.05660946,
        0.04381496,  0.03628373,  0.02335065,  0.0196972 , -0.02080937,
       -0.02324732, -0.05237539, -0.07090431, -0.07540448, -0.08175437,
       -0.09488243, -0.09824429, -0.10093641, -0.14388952, -0.20466851,
       -0.21334344, -0.21533153, -0.21828648, -0.26267369, -0.2892743 ,
       -0.29044642, -0.33643461, -0.33996271, -0.34752596, -0.34790642,
       -0.36088246, -0.41386509, -0.42601372, -0.4260432 , -0.42930941,
       -0.50531452, -0.50667305, -0.54441457, -0.56360898, -0.68829319,
       -0.6969187 , -0.72947431, -0.73094077, -0.82976763, -0.89658443,
       -1.0902614 , -1.21303801])
```

In [142]:
```
### GRADED PART (DO NOT EDIT) ###
part6 = str(idx_highest_sharpe)
submissions[all_parts[5]]=part6
grading.submit(COURSERA_EMAIL, COURSERA_TOKEN, assignment_key,all_parts[:6],all_parts,
idx_highest_sharpe
### GRADED PART (DO NOT EDIT) ###
```

```
Submission successful, please check on the coursera grader page for the status
```

Out[142]:
```
42
```