# Factor Investing: Applying Regularized Regression Models

Princeton University

June 2021

Daniel Thomas
PhD student at Princeton University

EDHEC
Python and ML for Asset Management,

Week 2

# Outline

- Introduction
  - Motivation for Factor Models
  - Definition of Factor Models
- Datasets
  - Basic  Summary of the factors and returns we will model
  - Data Preprocessing
- Linear Factor Models
  - Basic OLS (Ordinary Least Squares)
  - Drawbacks of OLS
- Modern Machine Learning Techniques
  - LASSO Regression
  - Ridge Regression
  - Best Subset Regression
- Modeling Expected Returns
- Harvard 5 Factor Analysis for Multiple Assets
- References

# Introduction

# Motivation for Factor Models

1. Motivation 1: Reduce the complexity of modeling asset price movements
    1. One reduces full model of returns to a linear model of Factors chosen by the user
    2. A large percentage of a well diversified equity portfolio can be explained by a 3 factor model

2. Motivation 2: Explaining Hybrid Asset Returns
    1. Real Estate and Hedge Funds are best modeled as a combination of factors

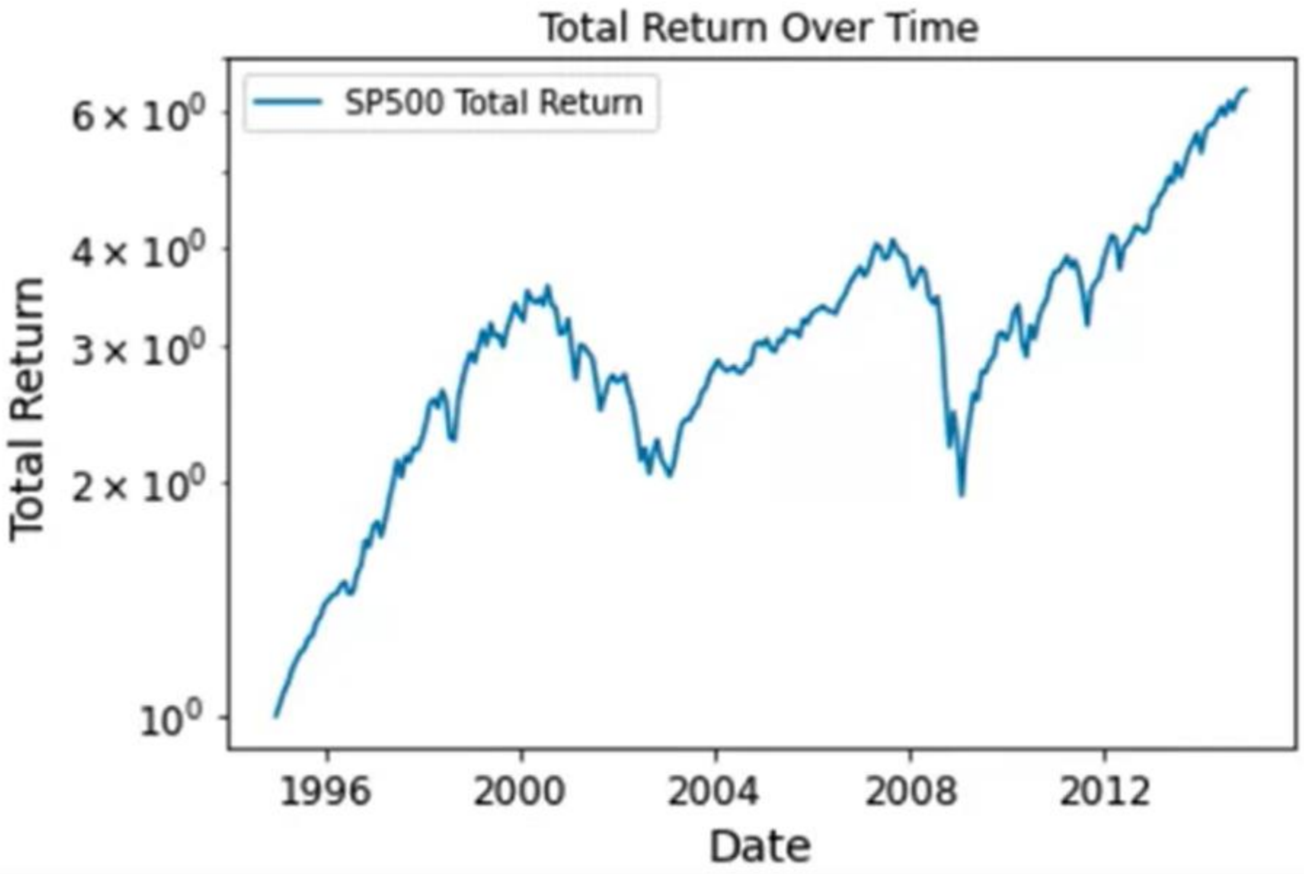
3. Motivation 3: Forecasting Future Expected Returns
    1. Given expected returns of the factors, we can map them onto expected returns of the asset

# Dataset: Basic Summary of Data

- 5 Factors:
  - World Equities Return
  - 10 Year US Treasury
  - High Yield
  - Inflation Protection
  - Currency Protection
- Dependent Variable: US S&P 500 Returns

## Dataset: Sanity Check

- As a sanity check, let's plot the Total Return of the S&P 500, and look at the worst month for US Equities



# Dataset: Sanity Check continued

```
In [7]: pd.options.display.float_format = "{:,.3f}".format #This rounds the display output to 3 decimals
        all_data.sort_values('SP500 Total Return').head(3)
```

Out[7]:

| | Date | World Equities | 10-year US Treasuries | High Yield | Inflation Protection | Currency Protection | U.S. Equity | SP500 Total Return | S&P 500 | International Equity | U.S. Treasury 20 years | Corporate Bond | Real Estate | Commodity | TIPS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **165** | 2008-10-01 | -0.194 | -0.029 | -0.084 | -0.058 | 0.087 | -0.173 | -0.166 | -0.168 | -0.205 | -0.036 | -0.070 | -0.312 | -0.295 | -0.087 |
| **166** | 2008-11-01 | -0.132 | 0.085 | 0.075 | -0.078 | 0.006 | -0.163 | -0.152 | -0.072 | -0.112 | 0.144 | 0.045 | -0.337 | -0.176 | 0.007 |
| **169** | 2009-02-01 | -0.131 | -0.002 | 0.033 | -0.018 | 0.028 | -0.148 | -0.148 | -0.106 | -0.127 | -0.004 | -0.019 | -0.269 | -0.097 | -0.020 |

# Ordinary Least Squares Factor Model

- A linear factor model can be expressed in the following equation

$$y_t = X_t^T \beta + \epsilon_t$$

- Where t is used to index each observation. $y_t$ is called the dependent variable, and $X_t$ is a vector of the factor returns.

$$\hat{\beta}^{\mathrm{OLS}} = argmin_{\beta} \left\{ \sum_{t=1}^{n} (y_t - X_t^T \beta)^2 \right\}.$$

# Ordinary Least Squares Factor Model Part 2

- OLS has a closed form solution.
  - Stack the factor returns on top of one another into a matrix $\boldsymbol{X_t}$
  - Stack the returns of the dependent variable into a vector $\boldsymbol{Y}$

$$\hat{\beta}^{\text{OLS}} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{Y}$$

- What do the factor loadings mean?
  - Answer: Effect on the dependent variable associated with the movement in the underlying factor

# Ordinary Least Squares Factor Model Part 3

- Let's run our first Factor Model on to explain the returns of the S&P500

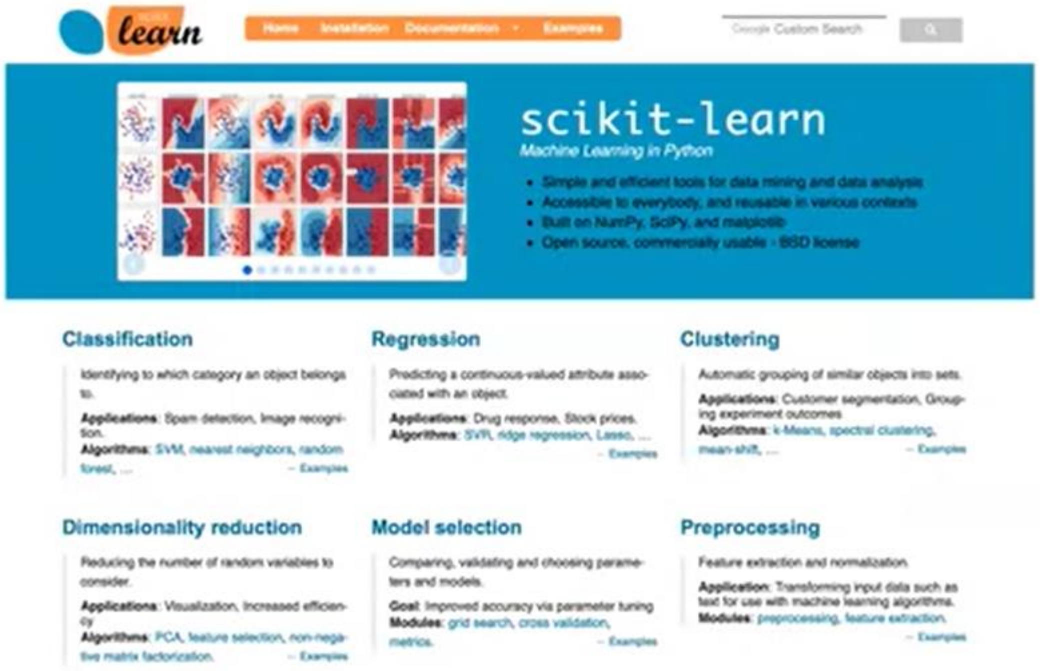
```
In [9]: #Linear Regression via Scikit-learn
        options = fm.create_options()
        options['name_of_reg'] = 'sikit-learn OLS'
        fm.linear_regression(all_data, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between January 1995 to December 2014 inclusive
                 Intercept  World Equities  10-year US Treasuries  High Yield  \
sikit-learn OLS      0.001           1.021                  0.128      -0.004

                 Inflation Protection  Currency Protection
sikit-learn OLS                -0.011                0.397
```

# How to build OLS: scikit-learn Library

- [scikit-learn]() is a machine learning library in Python
- Involves tools for data analysis and machine learning algorithms



# How to build OLS: The Code

- Below is a screenshot of the code underneath the fm.linear_regression command

```
#First function, linear factor model build
def linear_regression(data, dependentVar, factorNames, options):
    '''linear_regression takes in a dataset and returns the factor loadings using least squares regression
    INPUTS:
        data: pandas df, data matrix, should constain the date column and all of the factorNames columns
        dependentVar: string, name of dependent variable
        factorNames: list, elements should be strings, names of the independent variables
        options: dictionary, should constain at least two elements, timeperiod, and date
            timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
            date: name of datecol
            returnModel: boolean, if true, returns model
    Outputs:
        reg: regression object from sikitlearn
        also prints what was desired
    '''
    #first filter down to the time period
    if(options['timeperiod'] == 'all'):
        newData = data.copy()
    else:
        newData = data.copy()
        newData = newData.query(options['timeperiod'])

    #perform linear regression
    linReg = LinearRegression(fit_intercept=True)
    linReg.fit(newData[factorNames], newData[dependentVar])

    if (options['printLoadings'] == True):
        #Now print the results
        print_timeperiod(newData, dependentVar, options)
        # Now print the factor loadings
        display_factor_loadings(linReg.intercept_, linReg.coef_, factorNames, options)

    if(options['returnModel']):
        return linReg
```

# How to build OLS: The Code

- Below is a screenshot of the code underneath the fm.linear_regression command

```
#perform linear regression
linReg = LinearRegression(fit_intercept=True)
linReg.fit(newData[factorNames], newData[dependentVar])
```

# OLS Drawbacks

- OLS has two standard drawbacks, which we will cover here
  - 1: It has large uncertainty in the factor loadings when the factors are highly positively correlated (as is the case often in finance)
  - 2: It assumes factor loadings are constant over time

# Drawback 1: Highly Correlated Factors

- Here is a correlation matrix of the factors using the entire data set

```
In [10]: all_data[config.factorName].corr()
```

Out[10]:

| | World Equities | 10-year US Treasuries | High Yield | Inflation Protection | Currency Protection |
|---|---|---|---|---|---|
| **World Equities** | 1.000 | -0.176 | 0.308 | 0.332 | -0.528 |
| **10-year US Treasuries** | -0.176 | 1.000 | 0.131 | -0.616 | -0.140 |
| **High Yield** | 0.308 | 0.131 | 1.000 | 0.005 | -0.185 |
| **Inflation Protection** | 0.332 | -0.616 | 0.005 | 1.000 | -0.209 |
| **Currency Protection** | -0.528 | -0.140 | -0.185 | -0.209 | 1.000 |

# Drawback 2: Non-constant Loadings Over Time

In [12]:
```
options = fm.create_options()
normal_data = all_data[all_data['SP500 Total Return'] > 0].copy()
options['name_of_reg'] = 'OLS Normal'
options['return_model'] = False
fm.linear_regression(normal_data, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between January 1995 to December 2014 inclusive
            Intercept  World Equities  10-year US Treasuries  High Yield  \
OLS Normal      0.007           0.913                  0.201       0.025

            Inflation Protection  Currency Protection
OLS Normal                -0.003                0.388
```

Next we perform the analysis on crash months.

In [13]:
```
options = fm.create_options()
crash_data = all_data[all_data['SP500 Total Return'] <= 0].copy()
options['name_of_reg'] = 'OLS Crash'
options['return_model'] = False
fm.linear_regression(crash_data, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between July 1996 to September 2014 inclusive
           Intercept  World Equities  10-year US Treasuries  High Yield  \
OLS Crash     -0.009           0.822                  0.065      -0.004

           Inflation Protection  Currency Protection
OLS Crash                 0.159                0.229
```

# Baseline OLS Model

```
In [14]: train = all_data[(all_data['Date'] <= '2012-12-01') & (all_data['Date'] >= '1997-03-01')].copy()
         test = all_data[all_data['Date'] > '2012-12-01'].copy()
```

```
In [15]: options = fm.create_options()
         options['name_of_reg'] = 'OLS full data'
         options['return_model'] = True
         ols_model_train = fm.linear_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
               Intercept  World Equities  10-year US Treasuries  High Yield  \
OLS full data     -0.000           1.007                  0.132      -0.000

               Inflation Protection  Currency Protection
OLS full data                 0.058                0.374
```

# Modern Machine Learning Techniques

# Alternative Machine Learning Models

- Here we discuss newer methods to handle estimating factor loadings

- Versions of penalized regression
  - LASSO Regression
  - Ridge Regression
  - Elastic Net Regression

- Version of Constrained Regression
  - Best Subset Regression

## LASSO Regression

- We define LASSO regression as the following optimization problem
- As before, n is the number of data points, and m is the number factors.

$$\hat{\beta}^{\text{LASSO}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 + \lambda \sum_{j=1}^{m} |\beta_j| \right\}$$

- Lambda is called a hyperparameter which you need to choose. Scikit-learn, the most popular machine learning package (and the one used here) does not use lambda, but rather lambda hat, related via the following equation

$$\frac{\lambda}{2 * n} = \hat{\lambda}$$

# LASSO Regression Example

- Generally you would want to scale the factors
  - LASSO penalizes the size of the coefficient, which is related to the variance of the factor, you do not want to arbitrarily penalize factors more for having smaller variances (or vice versa).
  - We do not to aid in ease of interpretation
- What happens when we use a small alpha value? (alpha=.00001)

In [16]:
```
options = fm.create_options_lasso()
options['lambda_hat'] = .00001
options['print_loadings'] = True
options['name_of_reg'] = 'LASSO Regression with small Lambda'
fm.lasso_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
lambda_hat = 1e-05
                                    Intercept  World Equities  \
LASSO Regression with small Lambda      0.000           0.990

                                    10-year US Treasuries  High Yield  \
LASSO Regression with small Lambda                  0.063       0.000

                                    Inflation Protection  Currency Protection
LASSO Regression with small Lambda                 0.000                0.308
```

# LASSO Regression Example Continued

- Given too large a alpha value one can remove too many factors!

In [17]:
```
options = fm.create_options_lasso()
options['lambda_hat'] = .001 #The input alpha value
options['print_loadings'] = True
options['name_of_reg'] = 'LASSO Reg with large Lambda'
fm.lasso_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
lambda_hat = 0.001
                             Intercept  World Equities  10-year US Treasuries  \
LASSO Reg with large Lambda      0.003           0.551                 -0.000

                             High Yield  Inflation Protection  \
LASSO Reg with large Lambda       0.000                 0.000

                             Currency Protection
LASSO Reg with large Lambda               -0.000
```

# Cross Validation: Picking Lambda

- Heuristic Definition of Cross Validation:
  - Break the data set into k folds, and define a list of lambda values
  - For each fold, and for each lambda
    - Train the model on the k-1 other folds, and calculate the error on the test fold
  - At the end of the loops, you will have k out of sample errors for each value of lambda
  - Pick the lambda which minimizes the average error across your k sample tests

- Cross validation is the standard methodology of picking lambda for LASSO.

# Cross Validation: LASSO Example

- Below we use cross validation to pick lambda.

```
In [18]: options = fm.create_options_cv_lasso()
options['name_of_reg'] = 'CV Lasso'
options['max_lambda_hat'] = .001 #This specifies the maximum Alpha value tested by cross validation, minimum value is
options['return_model'] = True
options['n_folds'] = 5 #This states the number of folds
lasso_model_train = fm.cross_validated_lasso_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
Best lambda_hat = 1.6667957547420788e-05
          Intercept  World Equities  10-year US Treasuries  High Yield  \
CV Lasso      0.000           0.977                  0.036       0.000

          Inflation Protection  Currency Protection
CV Lasso                -0.000                0.272
```

# LASSO Code

- Below we give the code for building the the cross validated LASSO model

```python
def cross_validated_lasso_regression(data, dependentVar, factorNames, options):
    '''cross_validated_lasso_regression takes in a dataset and returns the factor loadings using lasso regression and cross validating the choice of lambda
    INPUTS:
        data: pandas df, data matrix, should constain the date column and all of the factorNames columns
        dependentVar: string, name of dependent variable
        factorNames: list, elements should be strings, names of the independent variables
        options: dictionary, should constain at least two elements, timeperiod, and date
            timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
            date: name of datecol
            returnModel: boolean, if true, returns model
            printLoadings: boolean, if true, prints the coeficients

            maxLambda: float, max lambda value passed
            nLambdas: int, number of lambda values to try
            randomState: integer, sets random state seed
            nFolds: number of folds
            NOTE: SKLearn calles Lambda Alpha.  Also, it uses a scaled version of LASSO argument, so here I scale when converting lambda to alpha
    Outputs:
        reg: regression object from sikitlearn
        also prints what was desired
    '''
    #Test timeperiod
    if(options['time_period'] == 'all'):
        newData = data.copy()
    else:
        newData = data.copy()
        newData = newData.query(options['time_period'])

    #Do CV Lasso
    alphas = np.logspace(-12, np.log(options['max_lambda_hat']), base=np.exp(1), num=options['n_lambda_hat'])
    #alphas = np.linspace(1e-12, alphaMax, options['nAlphas'])
    if(options['random_state'] == 'none'):
        lassoTest = Lasso(fit_intercept=True)
    else:
        lassoTest = Lasso(random_state = options['random_state'], fit_intercept=True)

    tuned_parameters = [{'alpha': alphas}]

    clf = GridSearchCV(lassoTest, tuned_parameters, cv=options['n_folds'], refit=True)
    clf.fit(newData[factorNames],newData[dependentVar])
    lassoBest = clf.best_estimator_
    alphaBest = clf.best_params_['alpha']

    if (options['print_loadings'] == True):
        #Now print the results
        print_timeperiod(newData, dependentVar, options)
        print('Best lambda_hat = ' + str(alphaBest))
        #Now print the factor loadings
        display_factor_loadings(lassoBest.intercept_, lassoBest.coef_, factorNames, options)
```

# LASSO Code

- Below we give the code for building the the cross validated LASSO model

```
alphas = np.logspace(-12, np.log(options['max_lambda_hat']), base=np.exp(1), num=options['n_lambda_hat'])
#alphas = np.linspace(1e-12, alphaMax, options['nAlphas'])
if(options['random_state'] == 'none'):
    lassoTest = Lasso(fit_intercept=True)
else:
    lassoTest = Lasso(random_state = options['random_state'], fit_intercept=True)

tuned_parameters = [{'alpha': alphas}]

clf = GridSearchCV(lassoTest, tuned_parameters, cv=options['n_folds'], refit=True)
clf.fit(newData[factorNames],newData[dependentVar])
lassoBest = clf.best_estimator_
```

# Elastic Net: Definition

- Here is the standard definition of Elastic Net

$$\hat{\beta}^{\text{LASSO}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 + \lambda_1 \sum_{j=1}^{m} |\beta_j| \right\}$$

$$\hat{\beta}^{\text{Ridge}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 + \lambda_2 ||\beta||_2^2 \right\}$$

$$\hat{\beta}^{\text{Elastic Net}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 + \lambda_1 \sum_{j=1}^{m} |\beta_j| + \lambda_2 ||\beta||_2^2 \right\}$$

- In scikit-learn, they write the Elastic Net slightly differently.

$$\hat{\beta}^{\text{Elastic Net}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 + \hat{\lambda} * \text{l1\_ratio} \sum_{j=1}^{m} |\beta_j| + .5 * \hat{\lambda} * (1 - \text{l1\_ratio}) ||\beta||_2^2 \right\}$$

# Elastic Net: Example

```
In [19]: options = fm.create_options_cv_elastic_net()
         options['nameOfReg'] = 'CV Elastic Net'
         options['maxAlpha'] = .01
         options['nFolds'] = 5
         options['returnModel'] = True
         el_model_train = fm.cross_validated_elastic_net_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
Best alpha = 1.759702749929848e-05
Best l1 ratio = 0.9378947894736842
                Intercept  World Equities  10-year US Treasuries  High Yield  \
CV Elastic Net      0.000           0.976                  0.036       0.000

                Inflation Protection  Currency Protection
CV Elastic Net                -0.000                0.271
```

# Elastic Net: Code

```python
def cross_validated_elastic_net_regression(data, dependentVar, factorNames, options):
    '''cross_validated_elastic_net_regression takes in a dataset and returns the factor loadings using elastic net, also chooses alpha and l1 ratio via cross validation
    INPUTS:
        data: pandas df, data matrix, should constain the date column and all of the factorNames columns
        dependentVar: string, name of dependent variable
        factorNames: list, elements should be strings, names of the independent variables
        options: dictionary, should constain at least two elements, timeperiod, and date
            timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
            date: name of datecol
            returnModel: boolean, if true, returns model
            printLoadings: boolean, if true, prints the coeficients

            maxLambda: float, max lambda value passed
            nLambdas: int, number of lambda values to try
            maxL1Ratio: float
            randomState: integer, sets random state seed
            nFolds: number of folds
            NOTE: SKLearn calles Lambda Alpha.  So I change Lambda -> Alpha in the following code
    Outputs:
        reg: regression object from sikitlearn
        also prints what was desired
    '''
    #Test timeperiod
    if(options['time_period'] == 'all'):
        newData = data.copy()
    else:
        newData = data.copy()
        newData = newData.query(options['time_period'])

    #Do CV Lasso
    alphaMax = options['max_lambda_hat']
    alphas = np.logspace(-12, np.log(alphaMax), num=options['n_lambda_hat'])
    l1RatioMax = options['max_l1_ratio']
    l1Ratios = np.linspace(1e-6, l1RatioMax, options['n_l1_ratio'])
    if(options['random_state'] == 'none'):
        elasticNetTest = ElasticNet(fit_intercept=True)
    else:
        elasticNetTest = ElasticNet(random_state = options['random_state'], fit_intercept=True)

    tuned_parameters = [{'alpha': alphas, 'l1_ratio': l1Ratios}]

    clf = GridSearchCV(elasticNetTest, tuned_parameters, cv=options['n_folds'], refit=True)
    clf.fit(newData[factorNames],newData[dependentVar])
    elasticNetBest = clf.best_estimator_
    alphaBest = clf.best_params_['alpha']
    l1RatioBest = clf.best_params_['l1_ratio']

    if (options['print_loadings'] == True):
        #Now print the results
        print_timeperiod(newData, dependentVar, options)
        print('Best lambda_hat = ' + str(alphaBest))
        print('Best l1 ratio = ' + str(l1RatioBest))
        #Now print the factor loadings
        display_factor_loadings(elasticNetBest.intercept_, elasticNetBest.coef_, factorNames, options)
```

# Elastic Net: Code

```
    alphaMax = options['max_lambda_hat']
    alphas = np.logspace(-12, np.log(alphaMax), num=options['n_lambda_hat'])
    l1RatioMax = options['max_l1_ratio']
    l1Ratios = np.linspace(1e-6, l1RatioMax, options['n_l1_ratio'])
    if(options['random_state'] == 'none'):
        elasticNetTest = ElasticNet(fit_intercept=True)
    else:
        elasticNetTest = ElasticNet(random_state = options['random_state'], fit_intercept=True)

    tuned_parameters = [{'alpha': alphas, 'l1_ratio': l1Ratios}]

    clf = GridSearchCV(elasticNetTest, tuned_parameters, cv=options['n_folds'], refit=True)
    clf.fit(newData[factorNames],newData[dependentVar])
    elasticNetBest = clf.best_estimator_
    alphaBest = clf.best_params_['alpha']
    l1RatioBest = clf.best_params_['l1_ratio']
```

## Constrained Regressions: Best Subset Regression

- We define a constrained regression as an OLS regression subject to constraints.
- Best subset regression is a simple constrained regression, loosely defined as "find the best linear model subject to the constraint only "x" factor loadings can be nonzero." In this case, "x" is an integer the user defines.
- Formally, let **z** be a vector of binary variables, let **M** be a very large number.
- For simplicity, let total_vars be the number of factors considered and max_vars be the number max number of factors allowed in the final model.

$$\hat{\beta}^{\text{Best Subset}} = argmin_{\beta}\left\{ \sum_{t=1}^{n}(y_t - X_t^T\beta)^2 \right\}.$$

$$\sum_{i=1}^{\text{max\_vars}} z_i \leq \text{max\_vars}, \;\; Mz + \beta \geq 0 \text{ and } \beta \leq Mz \text{ , } \mathbf{z} \text{ binary}$$

# Constrained Regressions: Best Subset Regression Examples

- We begin with max_vars set to 2.

```
In [20]: options = fm.create_options_best_subset()
         options['max_vars'] = 2
         options['name_of_reg'] = 'Best Subset with maxVars = 2'
         fm.best_subset_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
Max Number of Non-Zero Variables is 2
                              Intercept  World Equities  \
Best Subset with maxVars = 2      0.000           0.992

                              10-year US Treasuries  High Yield  \
Best Subset with maxVars = 2                  0.000       0.000

                              Inflation Protection  Currency Protection
Best Subset with maxVars = 2                 0.000                0.324
```

- Alternatively, we can set max_vars to 3.

```
In [21]: options = fm.create_options_best_subset()
         options['max_vars'] = 3
         options['return_model'] = True
         options['name_of_reg'] = 'Best Subset with maxVars = 3'
         best_subset_3 = fm.best_subset_regression(train, 'SP500 Total Return', config.factorName, options)
```

```
Dependent Variable is SP500 Total Return
Time period is between March 1997 to December 2012 inclusive
Max Number of Non-Zero Variables is 3
                              Intercept  World Equities  \
Best Subset with maxVars = 3     -0.000           1.009

                              10-year US Treasuries  High Yield  \
Best Subset with maxVars = 3                  0.104       0.000
```

# Constrained Regressions: Best Subset Regression Code

```
def best_subset(x,y,l_0):
    # Mixed Integer Programming in feature selection
    M = 1000
    n_factor = x.shape[1]
    z = cp.Variable(n_factor, boolean=True)
    beta = cp.Variable(n_factor)
    alpha = cp.Variable(1)

    def MIP_obj(x,y,b,a):
        return cp.norm(y-cp.matmul(x,b)-a,2)

    best_subset_prob = cp.Problem(cp.Minimize(MIP_obj(x, y, beta, alpha)),
                                  [cp.sum(z)<=l_0, beta+M*z>=0, M*z>=beta])
    best_subset_prob.solve(solver='ECOS_BB')
    return alpha.value, beta.value
```

# Forecasting Future Expected Returns

- Here are our Expected Return Assumptions for the 5 factors (quoted as annualized return)
  - World Equities: 8%
  - Treasury Bonds: 2%
  - High Yield Bonds: 5%
  - Inflation Protection: -.3%
  - Currency Protection: 0%

In [22]:

```
models = [ols_model_train, lasso_model_train, el_model_train, best_subset_3]

comparison = pd.DataFrame(np.zeros((4, 7)), columns=['Intercept'] + config.factorName +
                          ['Implied Expected Return S&P 500 (Annual)'])
expected_returns = pd.DataFrame(np.array([[.08/12, .02/12, .05/12, -.003/12, 0/12]]), columns=config.factorName)

for i in range(len(models)):
    model = models[i]
    comparison.loc[i,'Intercept'] = model.intercept_
    comparison.loc[i, config.factorName] = model.coef_
    comparison.loc[i, 'Implied Expected Return S&P 500'] = 12*model.predict(expected_returns)

comparison.index = ['OLS', 'LASSO', 'Elastic Net', 'Best Subset']
```

# Forecasting Future Expected Returns: Out of Sample Test

```
In [25]: predictions
```

Out[25]:

| | R^2 on Testing Set |
|---|---|
| **OLS** | 0.856 |
| **LASSO** | 0.858 |
| **Elastic Net** | 0.858 |
| **Best Subset** | 0.862 |

# Harvard 5 Factor Analysis for Multiple Assets

# Replicating the Harvard 5 Factor Study

- Asset Class Definitions
  - US Equities: Total Return of S&P 500
  - International Equities: Total Return of MSCI World EX US Index
  - Treasury Bond 20 Year: Return of 20 Year US Treasury Bond
  - Corporate Bonds: BofA Merrill Lynch US Corp Master Total Return Index
  - TIPS: Barclays US Treasury Inflation-Linked Bond Index (same index as when calculating inflation protection)

# Replicating the Harvard 5 Factor Study

In [25]:
```
factor_matrix
```

Out[25]:

| | Intercept | World Equities | 10-year US Treasuries | High Yield | Inflation Protection | Currency Protection | Implied Expected Return |
|---|---|---|---|---|---|---|---|
| **SP500 Total Return** | 0.000 | 0.977 | 0.036 | 0.000 | -0.000 | 0.272 | 0.082 |
| **International Equity** | 0.000 | 0.932 | -0.085 | 0.023 | -0.018 | -0.323 | 0.077 |
| **U.S. Treasury 20 years** | -0.000 | -0.000 | 1.340 | 0.026 | -0.000 | 0.098 | 0.025 |
| **Corporate Bond** | 0.002 | 0.081 | 0.566 | 0.074 | 0.234 | -0.004 | 0.039 |
| **Commodity** | 0.003 | 0.215 | 0.000 | -0.057 | 0.841 | -0.992 | 0.043 |
| **TIPS** | 0.000 | 0.002 | 0.958 | 0.000 | 0.939 | -0.000 | 0.019 |