

# 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



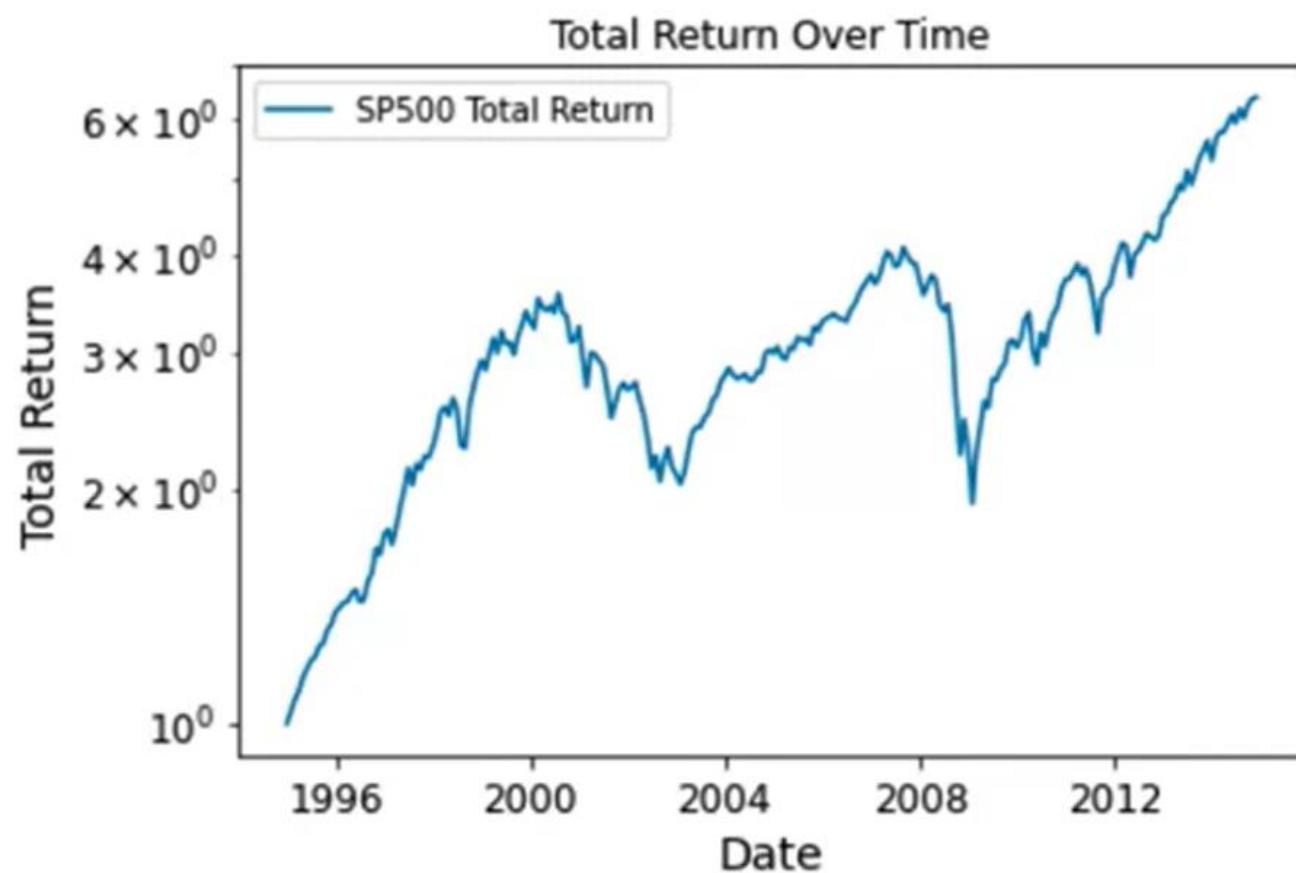
## Dataset: Basic Summary of Data

---

- 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 Treasures	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.

$$\hat{\beta}^{\text{OLS}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{t=1}^n (y_t - X_t^T \beta)^2 \right\}.$$



## 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  $\mathbf{X}_t$
  - Stack the returns of the dependent variable into a vector  $\mathbf{Y}$

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

- 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

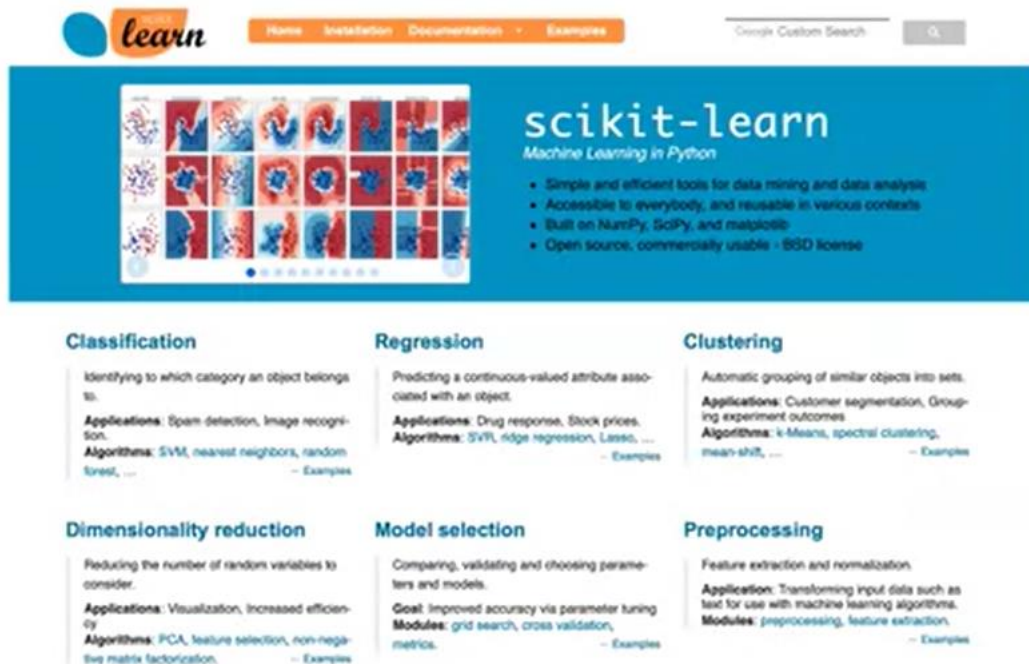
Time period is between January 1995 to December 2014 inclusive

	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



The screenshot shows the scikit-learn website homepage. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. Below this is a large blue banner featuring a grid of colorful, abstract images on the left and the text "scikit-learn Machine Learning in Python" on the right. The banner also lists key features: simple and efficient tools for data mining and data analysis, accessibility to everybody, built on NumPy, SciPy, and matplotlib, and being open source and commercially usable under the BSD license. Below the banner, the website is organized into six categories: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing. Each category has a brief description, applications, algorithms, and a link to examples.

**scikit-learn**  
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

**Classification**  
Identifying to which category an object belongs to.  
**Applications:** Spam detection, image recognition.  
**Algorithms:** SVM, nearest neighbors, random forest, ...

**Regression**  
Predicting a continuous-valued attribute associated with an object.  
**Applications:** Drug response, stock prices.  
**Algorithms:** SVM, ridge regression, Lasso, ...

**Clustering**  
Automatic grouping of similar objects into sets.  
**Applications:** Customer segmentation, grouping experiment outcomes.  
**Algorithms:** k-Means, spectral clustering, mean-shift, ...

**Dimensionality reduction**  
Reducing the number of random variables to consider.  
**Applications:** Visualization, increased efficiency.  
**Algorithms:** PCA, feature selection, non-negative matrix factorization, ...

**Model selection**  
Comparing, validating and choosing parameters and models.  
**Goal:** Improved accuracy via parameter tuning.  
**Modules:** grid search, cross validation, metrics, ...

**Preprocessing**  
Feature extraction and normalization.  
**Application:** Transforming input data such as text for use with machine learning algorithms.  
**Modules:** preprocessing, feature extraction, ...

## How to build OLS: The Code

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

```
218 #First function, linear factor model build
219 def linear_regression(data, dependentVar, factorNames, options):
220     '''linear_regression takes in a dataset and returns the factor loadings using least squares regression
221     INPUTS:
222         data: pandas df, data matrix, should contain the date column and all of the factorNames columns
223         dependentVar: string, name of dependent variable
224         factorNames: list, elements should be strings, names of the independent variables
225         options: dictionary, should contain at least two elements, timeperiod, and date
226             timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
227             date: name of datecol
228             returnModel: boolean, if true, returns model
229     Outputs:
230         reg: regression object from sklearn
231         also prints what was desired
232     ...
233     #first filter down to the time period
234     if(options['timeperiod'] == 'all'):
235         newData = data.copy()
236     else:
237         newData = data.copy()
238         newData = newData.query(options['timeperiod'])
239
240     #perform linear regression
241     linReg = LinearRegression(fit_intercept=True)
242     linReg.fit(newData[factorNames], newData[dependentVar])
243
244     if (options['printLoadings'] == True):
245         #Now print the results
246         print_timeperiod(newData, dependentVar, options)
247         # Now print the factor loadings
248         display_factor_loadings(linReg.intercept_, linReg.coef_, factorNames, options)
249
250     if(options['returnModel']):
251         return linReg
```

## How to build OLS: The Code

---

- Below is a screenshot of the code underneath the fm.linear\_regression command

```
240 | #perform linear regression
241 | linReg = LinearRegression(fit_intercept=True)
242 | 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.

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

- 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 2
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

```
386 def cross_validated_lasso_regression(data, dependentVar, factorNames, options):
387     '''cross_validated_lasso_regression takes in a dataset and returns the factor loadings using lasso regression and cross
    validating the choice of lambda
    INPUTS:
    388         data: pandas df, data matrix, should contain the date column and all of the factorNames columns
    389         dependentVar: string, name of dependent variable
    390         factorNames: list, elements should be strings, names of the independent variables
    391         options: dictionary, should contain at least two elements, timeperiod, and date
    392             timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
    393             date: name of datecol
    394             returnModel: boolean, if true, returns model
    395             printLoadings: boolean, if true, prints the coefficients
    396             maxLambda: float, max lambda value passed
    397             nLambdas: int, number of lambda values to try
    398             randomState: integer, sets random state seed
    399             nFolds: number of folds
    400             NOTE: sklearn calls Lambda Alpha. Also, it uses a scaled version of LASSO argument, so here I scale when converting
    401             lambda to alpha
    402     OUTPUTS:
    403         reg: regression object from sklearn
    404         also prints what was desired
    405     '''
    406     #Test timeperiod
    407     if(options['time_period'] == 'all'):
    408         newData = data.copy()
    409     else:
    410         newData = data.copy()
    411         newData = newData.query(options['time_period'])
    412     #Do CV Lasso
    413     alphas = np.logspace(-12, np.log(options['max_lambda_hat']), base=np.exp(1), num=options['n_lambda_hat'])
    414     #alphas = np.linspace(1e-12, alphaMax, options['nAlphas'])
    415     if(options['random_state'] == 'none'):
    416         lassoTest = Lasso(fit_intercept=True)
    417     else:
    418         lassoTest = Lasso(random_state=options['random_state'], fit_intercept=True)
    419     tuned_parameters = [{'alpha': alphas}]
    420     clf = GridSearchCV(lassoTest, tuned_parameters, cv=options['n_folds'], refit=True)
    421     clf.fit(newData[factorNames],newData[dependentVar])
    422     lassoBest = clf.best_estimator_
    423     alphaBest = clf.best_params_['alpha']
    424     if (options['print_loadings'] == True):
    425         #Now print the results
    426         print_timeperiod(newData, dependentVar, options)
    427         print('Best lambda_hat = ' + str(alphaBest))
    428         #Now print the factor loadings
    429         display_factor_loadings(lassoBest.intercept_, lassoBest.coef_, factorNames, options)
```

## LASSO Code

---

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

```
415 alphas = np.logspace(-12, np.log(options['max_lambda_hat']), base=np.exp(1), num=options['n_lambda_hat'])
416 #alphas = np.linspace(1e-12, alphaMax, options['nAlphas'])
417 if(options['random_state'] == 'none'):
418     lassoTest = Lasso(fit_intercept=True)
419 else:
420     lassoTest = Lasso(random_state = options['random_state'], fit_intercept=True)
421
422 tuned_parameters = [{'alpha': alphas}]
423
424 clf = GridSearchCV(lassoTest, tuned_parameters, cv=options['n_folds'], refit=True)
425 clf.fit(newData[factorNames],newData[dependentVar])
426 lassoBest = clf.best_estimator_
```



## Elastic Net: Definition

- Here is the standard definition of Elastic Net

$$\hat{\beta}^{\text{LASSO}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^n (y_i - X_i^T \beta)^2 + \lambda_1 \sum_{j=1}^m |\beta_j| \right\}$$

$$\hat{\beta}^{\text{Ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^n (y_i - X_i^T \beta)^2 + \lambda_2 \|\beta\|_2^2 \right\}$$

$$\hat{\beta}^{\text{Elastic Net}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^n (y_i - X_i^T \beta)^2 + \lambda_1 \sum_{j=1}^m |\beta_j| + \lambda_2 \|\beta\|_2^2 \right\}$$

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

$$\hat{\beta}^{\text{Elastic Net}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^n (y_i - X_i^T \beta)^2 + \hat{\lambda} * \text{ll\_ratio} \sum_{j=1}^m |\beta_j| + .5 * \hat{\lambda} * (1 - \text{ll\_ratio}) \|\beta\|_2^2 \right\}$$



## 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

```
492 def cross_validated_elastic_net_regression(data, dependentVar, factorNames, options):
493     '''cross_validated_elastic_net_regression takes in a dataset and returns the factor loadings using elastic net, also chooses
494     alpha and ll ratio via cross validation
495     INPUTS:
496         data: pandas df, data matrix, should contain the date column and all of the factorNames columns
497         dependentVar: string, name of dependent variable
498         factorNames: list, elements should be strings, names of the independent variables
499         options: dictionary, should contain at least two elements, timeperiod, and date
500         timeperiod: string, if == all, means use entire dataframe, otherwise filter the df on this value
501         date: name of datecol
502         returnModel: boolean, if true, returns model
503         printLoadings: boolean, if true, prints the coefficients
504
505         maxLambda: float, max lambda value passed
506         nLambdas: int, number of lambda values to try
507         maxLlRatio: float
508         randomState: integer, sets random state seed
509         nFolds: number of folds
510     NOTE: SKLearn calls Lambda Alpha. So I change Lambda -> Alpha in the following code
511
512     Outputs:
513         reg: regression object from sklearn
514         also prints what was desired
515         ...
516
517     #Test timeperiod
518     if(options['time_period'] == 'all'):
519         newData = data.copy()
520     else:
521         newData = data.copy()
522         newData = newData.query(options['time_period'])
523
524     #DO CV Lasso
525     alphaMax = options['max_lambda_hat']
526     alphas = np.logspace(-12, np.log(alphaMax), num=options['n_lambda_hat'])
527     llRatioMax = options['max_ll_ratio']
528     llRatios = np.linspace(1e-6, llRatioMax, options['n_ll_ratio'])
529     if(options['random_state'] == 'none'):
530         elasticNetTest = ElasticNet(fit_intercept=True)
531     else:
532         elasticNetTest = ElasticNet(random_state=options['random_state'], fit_intercept=True)
533
534     tuned_parameters = [{'alpha': alphas, 'll_ratio': llRatios}]
535
536     clf = GridSearchCV(elasticNetTest, tuned_parameters, cv=options['n_folds'], refit=True)
537     clf.fit(newData[factorNames],newData[dependentVar])
538     elasticNetBest = clf.best_estimator_
539     alphaBest = clf.best_params_['alpha']
540     llRatioBest = clf.best_params_['ll_ratio']
541
542     if (options['print_loadings'] == True):
543         #Now print the results
544         print_timeperiod(newData, dependentVar, options)
545         print('Best lambda_hat = ' + str(alphaBest))
546         print('Best ll ratio = ' + str(llRatioBest))
547         #Now print the factor loadings
548         display_factor_loadings(elasticNetBest.intercept_, elasticNetBest.coef_, factorNames, options)
```

## Elastic Net: Code

```
522 alphaMax = options['max_lambda_hat']
523 alphas = np.logspace(-12, np.log(alphaMax), num=options['n_lambda_hat'])
524 llRatioMax = options['max_ll_ratio']
525 llRatios = np.linspace(1e-6, llRatioMax, options['n_ll_ratio'])
526 if(options['random_state'] == 'none'):
527     elasticNetTest = ElasticNet(fit_intercept=True)
528 else:
529     elasticNetTest = ElasticNet(random_state = options['random_state'], fit_intercept=True)
530
531 tuned_parameters = [{'alpha': alphas, 'll_ratio': llRatios}]
532
533 clf = GridSearchCV(elasticNetTest, tuned_parameters, cv=options['n_folds'], refit=True)
534 clf.fit(newData[factorNames],newData[dependentVar])
535 elasticNetBest = clf.best_estimator_
536 alphaBest = clf.best_params_['alpha']
537 llRatioBest = clf.best_params_['ll_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  $\mathbf{z}$  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.

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

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



## 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

---

```
201 def best_subset(x,y,l_0):
202     # Mixed Integer Programming in feature selection
203     M = 1000
204     n_factor = x.shape[1]
205     z = cp.Variable(n_factor, boolean=True)
206     beta = cp.Variable(n_factor)
207     alpha = cp.Variable(1)
208
209     def MIP_obj(x,y,b,a):
210         return cp.norm(y-cp.matmul(x,b)-a,2)
211
212     best_subset_prob = cp.Problem(cp.Minimize(MIP_obj(x, y, beta, alpha)),
213                                   [cp.sum(z)<=l_0, beta+M*z>=0, M*z>=beta])
214     best_subset_prob.solve(solver='ECOS_BB')
215     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 <sup>2</sup> 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