**Module 9**

**Model Selection and Regularization**

sequential feature selection and regularization

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3145279?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3145279/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3145280?wrap=1)

**Cross-Validation**

A family of methods to validate a model using only the existing data from the model; these techniques include K-fold cross-validation, leave-one-out cross-validation, and others

**Hyperparameter**

Any parameter used to control the learning that is set before training

**K-fold Cross-Validation**

A specific type of cross-validation where the number of ‘folds’ is selected, and one ‘fold’ becomes the test set, while all other ‘folds’ become the training set

**LASSO**

A regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of a model; this method performs L1 regularization

**Regularization**

Techniques used to optimize machine learning models by minimizing the adjusted loss function and preventing overfitting or underfitting

**Ridge Regression**

A method of estimating the coefficients of multiple-regression models in scenarios where independent variables are highly correlated; this method performs L2 regularization

**Sequential Feature Selection**

A family of algorithms that are used to reduce the number of features in a model

**Standardization**

A way to transform data to make features in the data approximately the same scale

**Mathilde’s session:**

1. Composition of different parameters
2. Combination of parameters, quality of prediction does up or drop drastically
3. Do not teach the noise to the model, overfit
4. Sequential Forward Selection (most features from the end) and Backward Selection (most features from the beginning) are 2 favorites
5. Ridge regression will never have 0 coefficient, still overfitting is a problem
6. Lasso regression may eliminate overfitting

**Savio’s session:**

<https://github.com/SavioSal/datasets/raw/master/Advertising.csv>

Hyperparameters which improve r-squared model score

cv=None in GridSearchCV() means 5 by default!

Dataframe of ttregressor

**Notes:**

Sequential feature selection adds or removes features based on the models' performance until a subset of features k of the desired size is reached.

Regularization involves techniques used to optimize machine learning models by minimizing the adjusted loss function and preventing overfitting or underfitting. Ridge regularization (L2) and LASSO regularization (L1) are the two main types of regularization.

Ridge regularization (L2) is a regularized version of linear regression. Having a regularization term when added to the cost function forces the learning algorithm to fit the data and keep the model weights as small as possible. Ridge regression uses the l₂ norm of the vector w, which is the vector of the feature weights.

LASSO regularization (L1) is another regularized version of linear regression that adds a regularization term to the cost function but uses the l₁ norm of the weight vector w.

In machine learning, the leave-one-out cross-validation (LOOCV) procedure is used to assess the performance of algorithms to make predictions without using the data they were trained on. This can be utilized when you have a small in-sample made up of a few examples.

K-fold cross-validation uses unseen data to estimate the performance of a model. Using this technique, hyperparameters (k-values) can be tuned to the optimal level to train the model. In addition, this approach has the advantage of using each example only once for training and validation (as part of a test fold).

Holdout cross-validation is the simplest form of cross-validation; therefore, it is sometimes termed a 'simple validation method' instead of a simplified or degenerate form of cross-validation. As part of this method, you randomly divide your data into two sets: training and test/validation (i.e., the holdout set). This technique has the advantage of performing well on unseen datasets.

**Plotting**

fig, ax = plt.subplots(1, 2, figsize = (15, 4))

ax[0].hist(insurance['charges'])

ax[0].grid()

ax[0].set\_title('Original charges column')

ax[1].hist(np.log1p(insurance['charges']))

ax[1].grid()

ax[1].set\_title('Logarithm of charges')

**Zip columns with values:**

list(zip(X\_train.columns, coef\_list[-1]))

# Uncomment to visualize solutions

plt.scatter(X, y, label = 'data')

plt.plot(X.squeeze(), ridge\_low\_preds, '--r', label = 'ridge predictions')

plt.plot(X.squeeze(), ols\_preds, '--g', label = 'OLS predictions')

plt.legend()

# Uncomment to visualize solutions

fig, ax = plt.subplots(1, 2, figsize = (15, 5))

ax[0].scatter(X, y, label = 'data')

ax[0].plot(X.squeeze(), ridge\_low\_preds, '--r', label = 'ridge low predictions')

ax[0].plot(X.squeeze(), ols\_preds, '--g', label = 'OLS predictions')

ax[0].plot(X.squeeze(), ridge\_high\_preds, '--', color = 'purple', label = 'ridge high predictions')

ax[0].set\_title('Comparing the shape of different models')

ax[0].legend()

ax[0].grid()

ax[1].plot(ols\_pipe.named\_steps['linreg'].coef\_, 'o', markersize = 10, label = 'OLS Coefs')

ax[1].plot(ridge\_low\_pipe.named\_steps['ridge\_low'].coef\_, 'v', markersize = 10, label = 'ridge low')

ax[1].plot(ridge\_high\_pipe.named\_steps['ridge\_high'].coef\_, '^', markersize = 10, label = 'ridge high')

ax[1].legend()

ax[1].axhline(color = 'black')

ax[1].grid()

ax[1].set\_title('Comparing the Coefficients')

—

Use the selector\_grid to extract both the feature names and their associated coefficients. This will involve:

* .best\_estimator\_: extract the best estimator/selector pair from your grid search
* .named\_steps['selector']: extract the selector from the pipeline
* .named\_steps['model']: extract the model from the pipeline
* .get\_support(): extract best features from selector. This returns booleans as to whether feature was selected, we can use this to slice our train data.

X\_train.columns[best\_selector.get\_support()]

* .coef\_: coefficients from best model

best\_estimator = selector\_grid.best\_estimator\_

best\_selector = best\_estimator.named\_steps['selector']

best\_model = best\_estimator.named\_steps['model']

feature\_names = X\_train.columns[best\_selector.get\_support()]

coefs = best\_model.coef\_

# Answer check

print(best\_estimator)

print(f'Features from best selector: {feature\_names}.')

print('Coefficient values: ')

print('===================')

pd.DataFrame([coefs.T], columns = feature\_names, index = ['model'])

**Module Issues:**

Codio 9.2 Problem 2: Correct output is something else, misleading samples listed in the cell as hint, ignore them:

[170 226 231 345 337]

[311 36 331 349 136]

Codio 9.3 Problem 4: ‘age^2’ in solution, not ‘children’

Codio 9.4 Problem 1,2: X.squeeze() in plt.plot(…) to make it plot!

Codio 9.6 Problem 2: variable name should be *grid*

Codio 9.7 Problem 2: variable *ridge\_param\_dict* = {'ridge\_\_alpha': np.logspace(0, 10, 50)} how to fill not described in the problem!

Codio 9.8 Problem 1: ignore *include\_bias = False* in PolynomialFeatures caused Problem 3 to fail as well!

In Video 9.5 thetas in the table shown should be actually phis!

y = 3

1/n sum of sq(theta x phi)

(3x1)^2 +

Codio Exercise:

9.1 Optional

backward\_pipe = Pipeline([

('transform', PolynomialFeatures(degree = 3, include\_bias = False)),

('column\_selector', SequentialFeatureSelector(LinearRegression(),

n\_features\_to\_select=3,

direction = 'backward')),

('linreg', TransformedTargetRegressor(LinearRegression()))

])

backward\_pipe.fit(X\_train[['age', 'bmi', 'children']], y\_train)

train\_preds = backward\_pipe.predict(X\_train[['age', 'bmi', 'children']])

test\_preds = backward\_pipe.predict(X\_test[['age', 'bmi', 'children']])

backward\_train\_mse = float(mean\_squared\_error(X\_train[['age', 'bmi', 'children']], train\_preds))

backward\_test\_mse = float(mean\_squared\_error(X\_test[['age', 'bmi', 'children']], test\_preds))

# Answer check

print(f'Train MSE: {backward\_train\_mse: .2f}')

print(f'Test MSE : {backward\_test\_mse: .2f}')

backward\_pipe

**Quizes:**

If a dataframe has two columns, “horsepower” and “weight”, then the output of a polynomial features object with degree two will have which features? : [“hp”, “weight”, “hp^2”, “hp weight”, “weight^2”]

*You are correct! The answer “*[“hp”, “weight”, “hp^2”, “hp weight”, “weight^2”]*” is correct because these are the five output features of the polynomial features object.*

A backslash in the code is used to indicate to Python that the code is finished here. : False

*You are correct! The answer “*False*” is correct because a backslash in the code is used to indicate to Python that the code is continued on the next line.*

A given dataframe has two columns: “horsepower” and “weight”. The output of a polynomial features object with degree three will have how many features? : 9

*You are correct! The answer “*9*” is correct because in this case, you have “[“hp”, “weight”, “hp^*2*”, “hp weight”, “weight^*2*”]” in addition to horsepower cubed, horsepower squared times weight, horsepower times weight squared, and lastly, weight cubed added, which sum up a total of 9 features.*

Sequential feature selection is a technique in which all features are tested one by one and the best features are selected. : False

*You are correct! The answer “False” is correct because sequential feature selection is a feature selection technique in which all features are tested one at a time to find the lowest MSE. The lowest MSE feature is then selected, and the remaining features are all tested again until a desired number of features is reached.*

Imagine that you are given the following five features: [“hp”, “weight”, “hp^2”, “hp weight”, “weight^2”].

The models built using only one of these features yield the following errors: [“12.36”, “13.87”, “19.32”, “21.35”, “18.57"]. Which of the following features will be selected in the first iteration of sequential feature selection? : hp

*You are correct! The answer “hp” is correct: since the error for "hp" is the lowest, it will be selected in the first iteration.*

In forward sequential feature selection, the selected features are [“hp”, “weight”]. In the third iteration, the model that contains horsepower, horsepower^2 has the lowest mean squared error. What would the final selected features be? : [“hp”, “weight”, “horsepower^2”]

*You are correct! The answer “*[“hp”, “weight”, “horsepower2*”]” is correct because the features “hp” and “weight” are already selected. In the third iteration, “horsepower2” has the lowest mean squared error, so it will also be a part of the final selected features.*

When using sequential feature selection in Python, what are the names of the two sets of indices that the data is divided into? : Training indices, Development indices

*You are correct! The answer “*Training indices*” is correct because for sequential feature selection in Python, this is one of the sets of indices that the data is divided into.*

*You are correct! The answer “*Development indices*” is correct because for sequential feature selection in Python, this is one of the sets of indices that the data is divided into.*

**feature\_select = SequentialFeatureSelector(estimator = LinearRegression(),** **scoring='neg\_mean\_squared\_error',** **cv=[[training\_indices, dev\_indices]],** **n\_feautres\_to\_select = 4)**

In the Python code above, the constructor scoring is used as “neg\_mean\_squared\_error” because the goal is to maximize the negative of the MSE. : True

*You are correct! The answer “*True*” is correct. You use the negative of the MSE because the sequential feature selector will pick the feature with the highest score. Since you want to minimize the MSE, this means you want to maximize the negative of the MSE.*

When using sequential feature selection on a dataset with 55 features, how many models would you need to fit if you want to select four features? : 214

*You are correct! The answer “*214*” is correct because to pick the first parameter, you have to fit 55 models. Then, for the second parameter, you fit 54 models. For the third parameter, you have to fit 53 models. And lastly, you fit 52 models. Therefore, the total number of models is 55 + 54 + 53 + 52 = 214.*

Sequential feature selection models built on the same dataset always return the same features in multiple runs. : False

*You are correct! The answer “*False*” is correct because any feature selection process depends on the samples that are selected. In multiple runs, there will be different samples each time and since there are different samples from the original dataset, there is a chance that it will end up selecting a different set of features.*

Reverse sequential feature selection starts with all of the features and takes out one at a time. : True

*You are correct! The answer “*True*” is correct because reverse sequential feature selection starts with all of the features and takes out one at a time until it has only as much as is required.*

Fundamentally, regularization is an approach for controlling the training error of a model. : False

*You are correct! The answer “*False*” is correct because regularization is an approach for controlling the complexity of a model.*

A regularized model creates an object of type linear regression. : False

*You are correct! The answer “*False*” is correct because a regularized model creates an object of type ridge regression.*

The difference between linear regression and the ridge object is the use of which constructor? : Alpha

*You are correct! The answer “*Alpha*” is correct because the ridge object uses a constructor alpha.*

If the alpha constructor in the ridge object is set to zero, then what is the regression similar to? : Linear regression

*You are correct! The answer “*Linear regression*” is correct because if the alpha constructor in the ridge object is set to zero, then the regression is similar to this.*

As the values of alpha increase, the sum of the squares of the features (blank). : Decreases

*You are correct! The answer “*Decreases*” is correct because as the values of alpha increase, the sum of the squares of the features decreases.*

Dataframe

|  |  |  |
| --- | --- | --- |
| **phi1** | **phi2** | **y** |
| 1 | −2 | −3 |
| 1 | 3 | 12 |

Consider the above dataframe. Suppose the predictions for “y” are “−3” and “7”. What will the mean squared error be? : 12.5

*You are correct! The answer “*12.5”*is correct because the*y*actual minus predicted for the first row equals –3 – (–3), which is 0, and for the second row it equals 12 – 7, which is 5. Therefore, (0^2 + 5^2) is 25. Then, you divide that total squared error by the number of samples, giving 25/2 = 12.5.*

In ridge regression, the model-fitting procedure minimizes the sum of the mean squared error plus alpha times the sum of the squares of the parameters. : True

*You are correct! The answer “*True*” is correct because in ridge regression, the model-fitting procedure minimizes the sum of the mean squared error plus alpha times the sum of the squares of the parameters.*

The term 'alpha' times the sum of θ1^2 + θ2^2 + ... + θd^2 is often called (blank). : Penalty term

*You are correct! The answer “*Penalty term*” is correct because the term 'alpha' times the sum of θ1^2 + θ2^2 + ... + θd^2 is called the penalty term.*

Given θ1 = 2, θ2 = 3 and alpha = 2, what would the penalty term be? : 26

*You are correct! The answer “*26*” is correct because θ1^2 + θ2^2 is equal to 13; then, you multiply this by the alpha value of 2, giving you 26.*

In ridge regression, when alpha becomes large, the penalty term becomes more sensitive to the magnitudes of the parameters, and the overall model size budget becomes unlimited. : False

*You are correct! The answer “*False*” is correct because when alpha becomes large, the penalty term becomes more sensitive to the magnitudes of the parameters, and the overall model size budget becomes constrained.*

The penalty term in regularization affects all the parameters of the regression equation differently. : False

*You are correct! The answer “*False*” is correct because the penalty term affects all the parameters equally, regardless of the scale of the parameters.*

The problem of the penalty term penalizing small features heavily is solved by introducing a technique called (blank). : Standardization

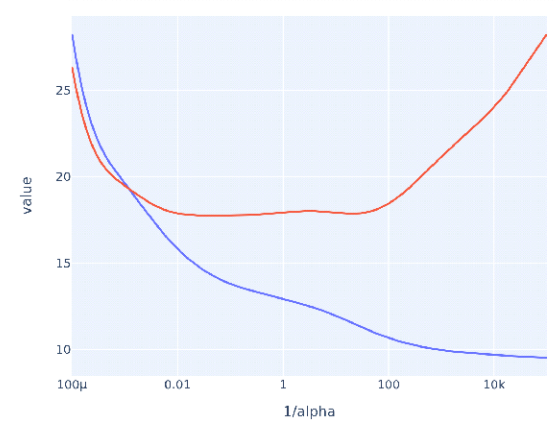
*You are correct! The answer “*Standardization*” is correct because it rescales the data.*

What is the formula for the Z-score? : z = x − mean/standard deviation

*You are correct! The answer “*z = x − mean/standard deviation*” is correct because this is the formula for calculating the Z-score.*

After standardization has been applied to a dataframe, what is the mean of each column? : Zero

*You are correct! The answer “*Zero*” is correct because after standardization has been applied to a dataframe, the mean of each column becomes 'zero'.*



In the given graph, what does the blue curve for alpha values on the x-axis and the MSE on the y-axis represent? : Training MSE

*You are correct! The answer “*Training MSE*” is correct because with the increase in alpha values, the MSE decreases for training.*

What can scikit-learn’s 'GridSearchCV' be used to find? : Optimal hyperparameters

*You are correct! The answer “*Optimal hyperparameters*” is correct because the library 'GridSearchCV' is used to find the most optimal hyperparameters.*

In the Python function GridSearchCV(), the constructor 'param\_grid' is used to give the scoring measure. : False

*You are correct! The answer “*False*” is correct because the constructor 'param\_grid' is used to give all the parameters to try.*

The GridSearchCV object is an estimator, and calling .predict() on it will give the best estimator results on some data. : False

*You are correct! The answer “*False*” is correct because the function*.predict()*will give the results for the most recent model built. Therefore, to get the best model, the function to use is*.best\_estimator\_*.*

LASSO regression is also known as L2 regularization. : False

*You are correct! The answer “*False*” is correct because LASSO regression is also known as L1 regularization.*

L1 regularization has a penalty term equal to the sum of the absolute values of the parameters times alpha. : True

*You are correct! The answer “*True*” is correct because the penalty term equal to the sum of the absolute values of the parameters times alpha is for L1 regularization.*

What is the regression that forces the parameters to be smaller and, in most circumstances, forces many of its parameters to zero? : Lasso regression

*You are correct! The answer “*Lasso regression*” is correct because this type of regression forces the parameters to be smaller and, in most cases, forces many of its parameters to zero.*

In simple cross-validation, the dataset is divided into how many datasets? : 2

*You are correct! The answer “*2*” is correct because simple cross-validation is divided into two sets: training sets and validation sets.*

What does the “K” represent in K-fold cross-validation? : The number of folds to split a dataset into

*You are correct! The answer “*The number of folds to split a dataset into*” is correct because K represents the number into which the data is divided.*

In K-fold cross-validation, if K equals 5, which fold would be the validation set in the third iteration? : 3rd fold

*You are correct! The answer “*3rd fold*'' is correct because in the first iteration, the first fold will be the validation set, and in the second iteration, the second fold will be the validation set. Similarly, in the third iteration, the third fold will be the validation set.*

K-fold cross-validation where K is equal to N (where N is the number of samples that you have) is also known as leave-one-out cross-validation. : True

*You are correct! The answer “*True*” is correct because when K is equal to N in K-fold cross-validation, it is also known as leave-one-out cross-validation.*

What should the constructor and its value be for 5-fold cross-validation in Python object 'GridSearchCV'? : cv=5

*You are correct! The answer “*cv=5*” is correct because K is equal to 5 and the constructor for K-fold cross-validation in the Python object 'GridSearchCV' is 'cv'.*

**Discussion 9.1: Which Cross-Validation Is Best?**

1. Evaluate the differences between leave-one-out cross-validation, k-fold cross-validation, and holdout cross-validation

* Codio Activity 9.1: Sequential Feature Selection
* Codio Activity 9.2: **Cross-Validation** with SequentialFeatureSelector
* Codio Activity 9.3: A First Look at the Ridge Regression Model
* Codio Activity 9.4: Comparing the Ridge to Ordinary Least Squares (OLS)
* Codio Activity 9.5: Using StandardScalar
* Codio Activity 9.6: Using **GridSearchCV**
* Codio Activity 9.7: **Ridge vs. Sequential Feature Selection**
* Codio Activity 9.8: **LASSO and Sequential Feature Selection**
* Self-Study Knowledge Check 9.1: Polynomial Features on Multidimensional Data
* Self-Study Knowledge Check 9.2: Sequential Feature Selection
* Self-Study Knowledge Check 9.3: Sequential Feature Selection in Scikit-Learn
* Self-Study Knowledge Check 9.4: A First Look at Regularization
* Self-Study Knowledge Check 9.5: How Regularization Works
* Self-Study Knowledge Check 9.6: Scaling
* Self-Study Knowledge Check 9.7: **GridSearchCV**
* Self-Study Knowledge Check 9.8: LASSO Regression
* Self-Study Knowledge Check 9.9: K-fold Cross-Validation

Video 9.3

9.2

**What is cross-validation?**

Cross-validation is a technique to train a model with a dataset and test out model performance with *unseen* dataset aim to address under-fitting and overfitting issues. For this purpose, the dataset is split into training and test datasets.

Please check out this document for Holdout, K-fold and Leave-one-out cross validation definitions: <https://www.cs.cmu.edu/~schneide/tut5/node42.html>

<https://student.emeritus.org/courses/4765/pages/overview-alternative-cross-validation-techniques?module_item_id=1114269>

**Dataset**

The dataset I found represents how much Americans trust online news outlets. It has 66 columns and 1021 rows.

Kaggle dataset: <https://www.kaggle.com/datasets/satoshidatamoto/how-access-to-data-affects-trust-in-newse?select=all_responses_coded.csv>

Each column in the dataset represents a question, the answer in the form of 0 or 1, therefore, it is imbalanced, none of the techniques above are suited for an imbalanced dataset.

**Data Cleanup**

I cleaned up data and created a target column from A1 through A5 columns as follows:

# ‘other’ news outlets from A21:

# set other news column as int64

news.loc[(news['A21'].str.lower() == 'none') | ( news['A21'] == '0'), 'other'] = 0

news.loc[~((news['A21'].str.lower() == 'none') | ( news['A21'] == '0')), 'other'] = 1

news['other'] = news['other'].astype('int64')

# target column by Very conservative, Conservative, Moderate, Liberal, Very liberal => 1,2,3,4,5

news.loc[news['A1'] == 1, 'target'] = 1

news.loc[news['A2'] == 1, 'target'] = 2

news.loc[news['A3'] == 1, 'target'] = 3

news.loc[news['A4'] == 1, 'target'] = 4

news.loc[news['A5'] == 1, 'target'] = 5

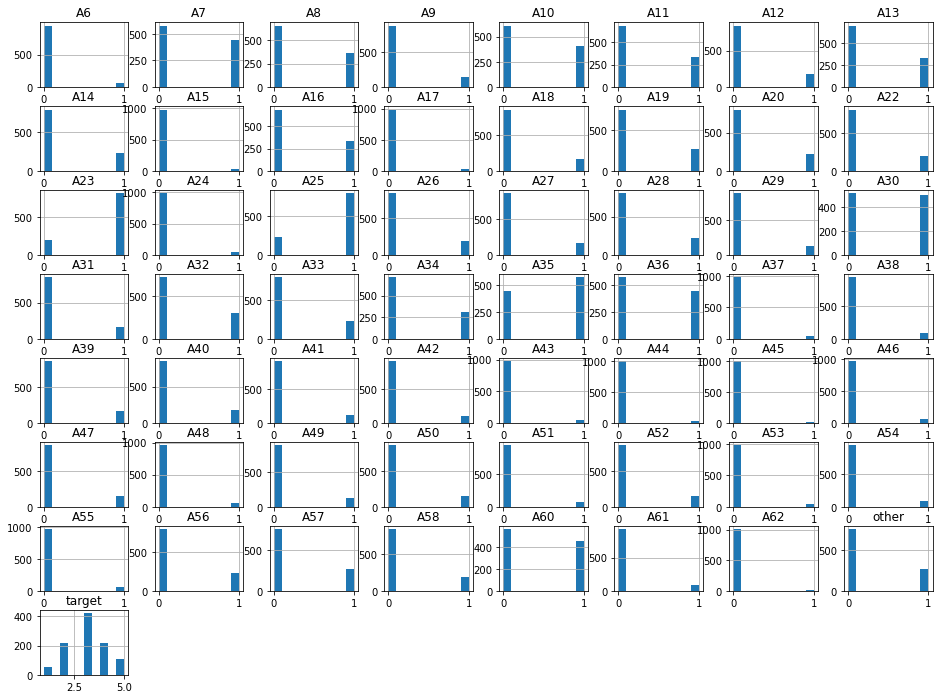
news['target'] = news['target'].astype('int64')

# drop unnecessary columns:

news\_outlets = news.drop(columns=['A1', 'A2', 'A3', 'A4', 'A4', 'A5', 'A21', 'A59', 'RespondentID', 'StartDate', 'EndDate', 'index'])

Finalized dataset:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A6** | **A7** | **A8** | **A9** | **A10** | **A11** | **A12** | **A13** | **A14** | **A15** | **A16** | **A17** | **A18** | **A19** | **A20** | **A22** | **A23** | **A24** | **A25** | **A26** | **A27** | **A28** | **A29** | **A30** | **A31** | **A32** | **A33** | **A34** | **A35** | **A36** | **A37** | **A38** | **A39** | **A40** | **A41** | **A42** | **A43** | **A44** | **A45** | **A46** | **A47** | **A48** | **A49** | **A50** | **A51** | **A52** | **A53** | **A54** | **A55** | **A56** | **A57** | **A58** | **A60** | **A61** | **A62** | **other** | **target** |
| **count** | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 | 1021.000000 |
| **mean** | 0.068560 | 0.433888 | 0.357493 | 0.140059 | 0.401567 | 0.330069 | 0.173359 | 0.318315 | 0.224290 | 0.037218 | 0.329089 | 0.034280 | 0.162586 | 0.265426 | 0.213516 | 0.200784 | 0.799216 | 0.040157 | 0.775710 | 0.184133 | 0.163565 | 0.213516 | 0.128306 | 0.494613 | 0.167483 | 0.294809 | 0.229187 | 0.306562 | 0.561214 | 0.436827 | 0.045054 | 0.091087 | 0.164545 | 0.172380 | 0.114594 | 0.102840 | 0.047992 | 0.030362 | 0.019589 | 0.055828 | 0.153771 | 0.063663 | 0.126347 | 0.155730 | 0.079334 | 0.149853 | 0.040157 | 0.083252 | 0.057786 | 0.227228 | 0.260529 | 0.180215 | 0.449559 | 0.092067 | 0.015671 | 0.262488 | 3.114594 |
| **std** | 0.252829 | 0.495853 | 0.479496 | 0.347218 | 0.490455 | 0.470468 | 0.378743 | 0.466051 | 0.417318 | 0.189389 | 0.470113 | 0.182037 | 0.369168 | 0.441776 | 0.409990 | 0.400783 | 0.400783 | 0.196423 | 0.417318 | 0.387783 | 0.370062 | 0.409990 | 0.334594 | 0.500216 | 0.373589 | 0.456180 | 0.420516 | 0.461292 | 0.496482 | 0.496236 | 0.207524 | 0.287874 | 0.370951 | 0.377896 | 0.318687 | 0.303899 | 0.213854 | 0.171666 | 0.138650 | 0.229701 | 0.360906 | 0.244271 | 0.332403 | 0.362777 | 0.270392 | 0.357102 | 0.196423 | 0.276398 | 0.233454 | 0.419247 | 0.439138 | 0.384555 | 0.497693 | 0.289262 | 0.124260 | 0.440202 | 1.031652 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 3.000000 |
| **75%** | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 4.000000 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 5.000000 |





So, k-fold cross-validation evaluates entire dataset, I looked further and I came across StratifiedKFold which keeps same data ratio of target column in samples. I decided to go with it which is suited for dealing with imbalanced classifications versus plain k-fold.

I also did standardization for the dataset in the pipeline:

#Evaluate StratifiedKFold with 5 splits

train\_mses = []

test\_mses = []

# standardization before LR:

pipe = Pipeline([('scaler', StandardScaler()),

('quad\_model', LinearRegression())])

skf = StratifiedKFold(n\_splits=5, random\_state=93, shuffle=True)

for train\_index, test\_index in skf.split(X, y):

#print("TRAIN:", train\_index, "TEST:", test\_index)

X\_train, X\_test = X.loc[train\_index], X.loc[test\_index]

y\_train, y\_test = y.loc[train\_index], y.loc[test\_index]

#fit pipeline on training data

pipe.fit(X\_train, y\_train)

#mse of training data

train\_mse = mean\_squared\_error(y\_train, pipe.predict(X\_train))

train\_mses.append(train\_mse)

#mse of testing data

test\_mse = mean\_squared\_error(y\_test, pipe.predict(X\_test))

test\_mses.append(test\_mse)

# check k-fold trend!

plt.plot(range(1, 6), train\_mses, '--o', label = 'training error')

plt.plot(range(1, 6), test\_mses, '--o', label = 'testing error')

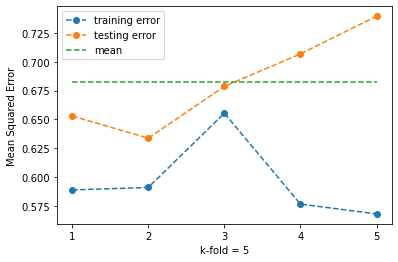
plt.plot(range(1, 6), [np.mean(test\_mses)]\*5, '--', label = 'mean')

plt.xticks(range(1, 6), range(1, 6))

plt.xlabel('k-fold = 5')

plt.ylabel('Mean Squared Error')

plt.legend()



High variance vs. low variance

**Try-It Activity 9.1: LASSO vs. SFS**

Standardize dataset?

Penalty Term

L1 is LASSO - sum of the absolute values of the parameters

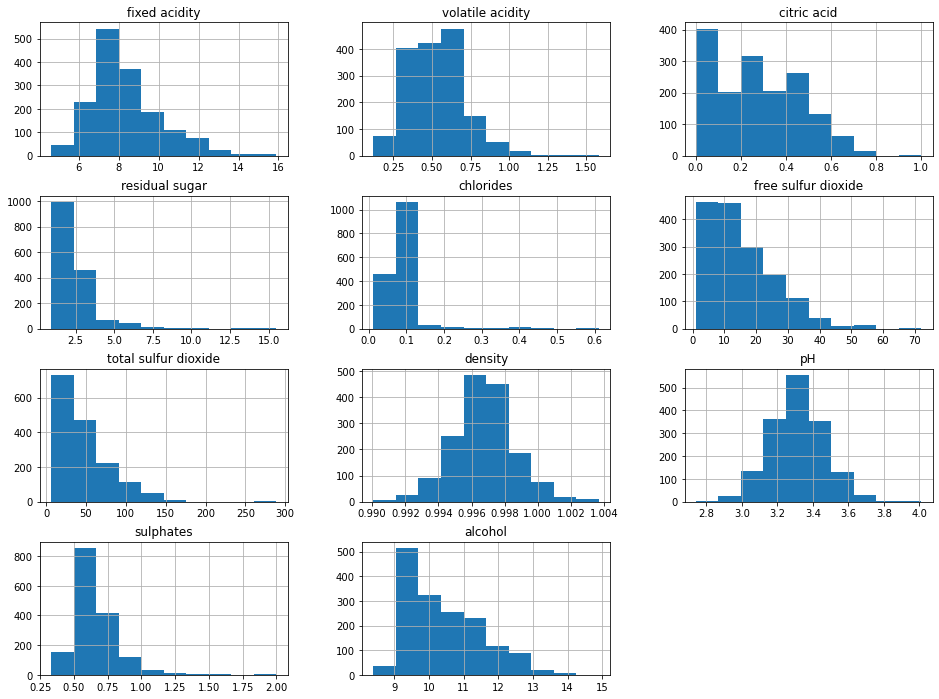
L2 is Ridge - sum of the square of the parameters

resulted in feature selection

I built regression models per:

* Ridge regressor to extract coefficients
* SequentialFeatureSelection using the Lasso to select 4 features.
* RFE using Lasso to select 4 features.

Since it is a pretty small dataset, I checked the features, almost all are skewed and its heatmap.



Target variable is correlated with alcohol, volatile acidity, sulphates, citric acid and total sulfur dioxide at first glance before I moved on to exercise.

**Ridge regression model as base to capture coefficients and MSEs**

# Ridge model as base

ridge\_model = Ridge().fit(X\_train, y\_train)

ridge\_coefs = ridge\_model.coef\_

print(f'Ridge Coefs: {np.round(ridge\_coefs, 2)}')

ridge\_train\_mse = mean\_squared\_error(y\_train, ridge\_model.predict(X\_train))

ridge\_test\_mse = mean\_squared\_error(y\_test, ridge\_model.predict(X\_test))

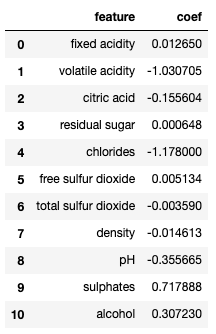
print('MSE train:', ridge\_train\_mse)

print('MSE test :', ridge\_test\_mse)

Ridge Coefs: [ 0.01 -1.03 -0.16 0. -1.18 0.01 -0. -0.01 -0.36 0.72 0.31]

MSE train: 0.4282141930152601

MSE test : 0.39099217285755516



**SequentialFeatureSelection using the Lasso to select 4 features**

# SequentialFeatureSelection using the Lasso to select 4 features.

# forward

sequential\_pipe = Pipeline([

# ('scaler', StandardScaler()),

('sfs', SequentialFeatureSelector(n\_features\_to\_select = 4, estimator=Lasso())),

('linreg', LinearRegression())

])

sequential\_pipe.fit(X\_train, y\_train)

sequential\_train\_mse = mean\_squared\_error(y\_train, sequential\_pipe.predict(X\_train))

sequential\_test\_mse = mean\_squared\_error(y\_test, sequential\_pipe.predict(X\_test))

print('SFS with LASSO:')

print('MSE train:', sequential\_train\_mse)

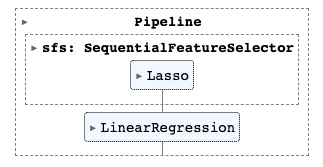
print('MSE test :', sequential\_test\_mse)

sequential\_pipe

SFS with LASSO:

MSE train: 0.5477973118099638

MSE test : 0.5027384933740707



The results of this model came out worse.

**RFE using Lasso to select 4 features**

# RFE using Lasso to select 4 features.

rfe\_pipe = Pipeline([

# ('scaler', StandardScaler()),

('rfe', RFE(n\_features\_to\_select = 4, estimator=Lasso())),

('linreg', LinearRegression())

])

rfe\_pipe.fit(X\_train, y\_train)

rfe\_train\_mse = mean\_squared\_error(y\_train, rfe\_pipe.predict(X\_train))

rfe\_test\_mse = mean\_squared\_error(y\_test, rfe\_pipe.predict(X\_test))

print('RFE with LASSO:')

print('MSE train:', rfe\_train\_mse)

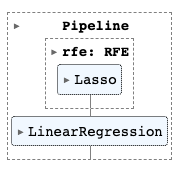
print('MSE test :', rfe\_test\_mse)

rfe\_pipe

RFE with LASSO:

MSE train: 0.4677700441355884

MSE test : 0.4406456053947725



This model slightly better than SFS but still worse than Ridge.

**Conclusion**

With default settings on Lasso made them worse, a close look at Lasso revealed that it only picked a single feature:

# Lasso by itself

lasso = Lasso() # default alpha = 1

lasso.fit(X\_train, y\_train)

print(list(zip(X\_train.columns, lasso.coef\_)))

lasso\_df = pd.DataFrame({'feature': X\_train.columns, 'coef': lasso.coef\_})

lasso\_df.loc[lasso\_df['coef'] != 0]

[('fixed acidity', 0.0), ('volatile acidity', -0.0), ('citric acid', 0.0), ('residual sugar', 0.0), ('chlorides', -0.0), ('free sulfur dioxide', 0.0), ('total sulfur dioxide', -0.004238947409537814), ('density', -0.0), ('pH', -0.0), ('sulphates', 0.0), ('alcohol', 0.0)]



Ridge model has the lowest score as shown below.

# Compare all 3 results!

print(f'The Complexity that minimized Test Error was: {test\_mses.index(min(test\_mses)) + 1}')

plt.plot(range(1, 4), [ridge\_train\_mse, sequential\_train\_mse, rfe\_train\_mse], '--o', label = 'training error')

plt.plot(range(1, 4), [ridge\_test\_mse, sequential\_test\_mse, rfe\_test\_mse], '--o', label = 'testing error')

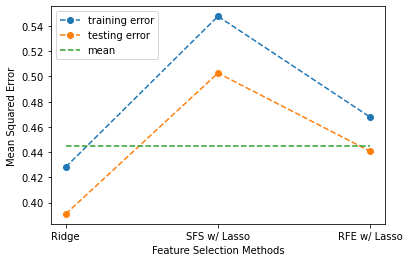
plt.plot(range(1, 4), [np.mean([ridge\_test\_mse, sequential\_test\_mse, rfe\_test\_mse])]\*3, '--', label = 'mean')

plt.xticks(range(1, 4), ['Ridge', 'SFS w/ Lasso', 'RFE w/ Lasso'])

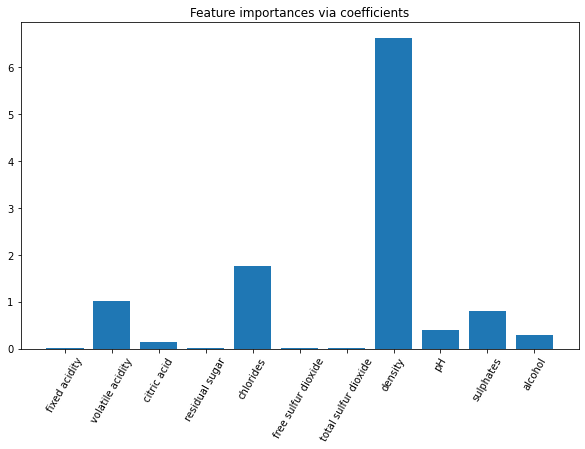
plt.xlabel('Feature Selection Methods')

plt.ylabel('Mean Squared Error')

plt.legend()



Final note, the default alpha setting is 1 on Lasso high penalty for this use case, lowering alpha=0.01 started to improve magnitude of the results of SFS with Lasso and RFE with Lasso regularization regression models.



**Try-It Activity 9.2: Conclusion Exercise**

In this exercise Ridge regression model is explored more. “Wage dataset” described here in detail: <https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html>

I built Ridge models with various regularizations.

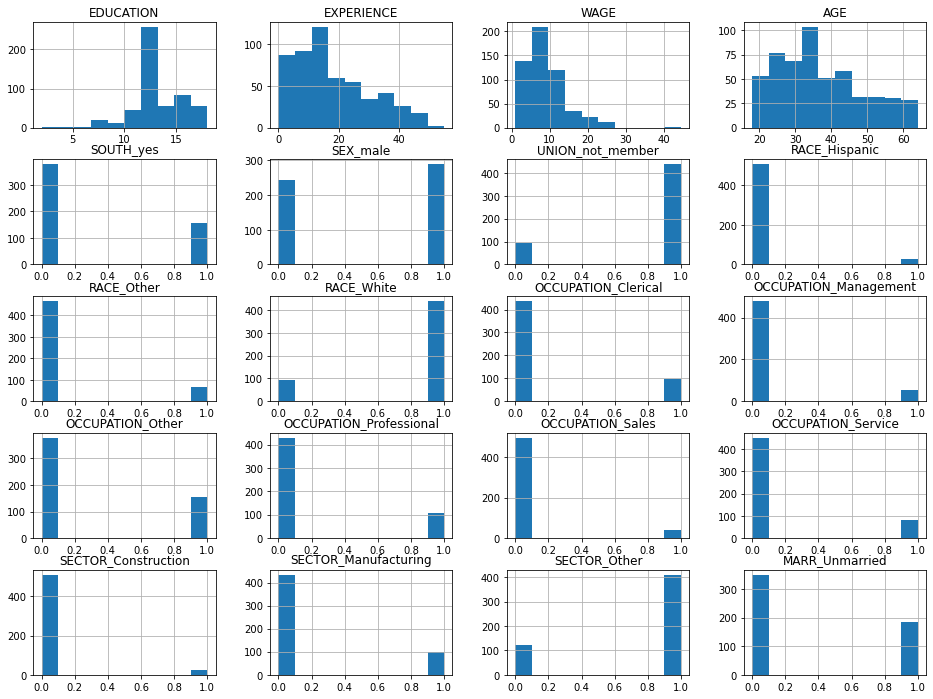
**Dataset Analysis**

There are 7 categorical variable, 4 of them are binary 'SOUTH', 'SEX', 'UNION', 'MARR’, and 3 columns 'RACE', 'OCCUPATION', 'SECTOR' have multiple values. I used OneHotEncoder in the pipeline. WAGE is the target column, splitting the dataset:

X, y = w.drop(columns='WAGE'), w['WAGE']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 93, test\_size = 0.3)

Histogram after transforming features:



The data is imbalanced, also, WAGE is skewed, therefore, requires logarithmic transformation.

**Ridge Model with variety of Regularization**

Smaller regularization yielded better results, minimal MSE and higher score.

alphas = [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0]

coef\_list = []

train\_mses = []

test\_mses = []

train\_maes = []

test\_maes = []

train\_scores = []

test\_scores = []

for a in alphas:

features = [c for c in ['SOUTH', 'SEX', 'UNION', 'RACE', 'OCCUPATION', 'SECTOR', 'MARR'] if c in X\_train.columns]

ohe\_step = make\_column\_transformer((OneHotEncoder(drop = 'if\_binary'), features), remainder="passthrough" )

pipe = Pipeline([('transformer', ohe\_step),

# ('scaler', StandardScaler()),

('ttregressor', TransformedTargetRegressor(func=np.log1p,

inverse\_func=np.expm1,

regressor=Ridge(alpha = a))) ])

#fit on train

pipe.fit(X\_train, y\_train)

coef\_list.append(list(pipe.named\_steps['ttregressor'].regressor\_.coef\_))

train\_mses.append(mean\_squared\_error(y\_train, pipe.predict(X\_train)))

test\_mses.append(mean\_squared\_error(y\_test, pipe.predict(X\_test)))

train\_maes.append(mean\_absolute\_error(y\_train, pipe.predict(X\_train)))

test\_maes.append(mean\_absolute\_error(y\_test, pipe.predict(X\_test)))

train\_scores.append(pipe.score(X\_train, y\_train))

test\_scores.append(pipe.score(X\_test, y\_test))

print('Score:', pipe.score(X\_test, y\_test))

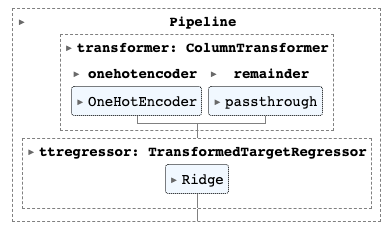
# Results

print(len(coef\_list))

print('Best alpha:', alphas[test\_mses.index(min(test\_mses))], 'coefficients:' )

print(list(zip(pipe.named\_steps['transformer'].get\_feature\_names\_out(), coef\_list[test\_mses.index(min(test\_mses))])))

pipe



# check effectiveness of regularization!

complexity = 11

plt.plot(range(1, complexity), train\_mses, '--o', label = 'training error')

plt.plot(range(1, complexity), test\_mses, '--o', label = 'testing error')

plt.plot(range(1, complexity), [np.mean(test\_mses)]\*(complexity-1), '--', label = 'mean')

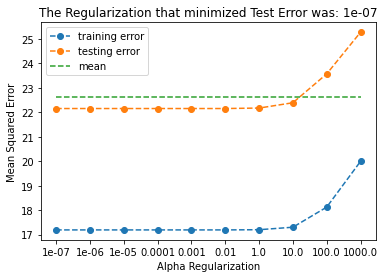
plt.xticks(range(1, complexity), [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0])

plt.xlabel('Alpha Regularization')

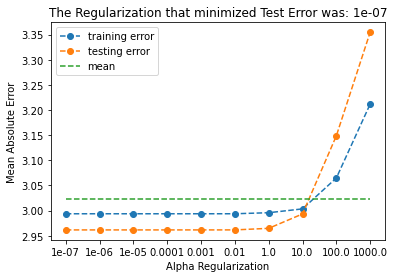
plt.ylabel('Mean Squared Error')

plt.title(f'The Regularization that minimized Test Error was: {alphas[test\_mses.index(min(test\_mses))]}')

plt.legend()



Similarly, MAE:



And, model scores per iteration:



I created a function to check permutation importance:

# permutation importance

#function to execute permutation importance!

def column\_importance(X, y):

features = [c for c in ['SOUTH', 'SEX', 'UNION', 'RACE', 'OCCUPATION', 'SECTOR', 'MARR'] if c in X.columns]

ohe\_step = make\_column\_transformer((OneHotEncoder(drop = 'if\_binary'), features), remainder="passthrough" )

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 93, train\_size=0.7, test\_size=0.3)

# fit model with training set

pipe = Pipeline([('transformer', ohe\_step),

# ('scaler', StandardScaler()),

('ttr\_ridge', TransformedTargetRegressor(func=np.log1p,

inverse\_func=np.expm1,

regressor=Ridge(alpha = 0.0000001))) ])

model = pipe.fit(X\_train, y\_train)

# score with test set

print('model r^2 :', model.score(X\_test, y\_test))

# permutation importance

r = permutation\_importance(model, X\_test, y\_test, n\_repeats = 50, random\_state = 93)

print('importance:', r.importances\_mean)

return model, X\_test, y\_test, pd.DataFrame({"Column":X.columns, "Importance":r.importances\_mean}).sort\_values(

by = "Importance", ascending = False)

model, Xp\_test, yp\_test, ranking = column\_importance(X, y)

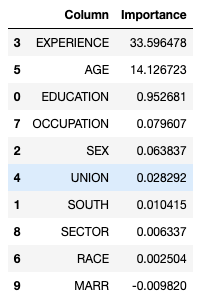
ranking

model r^2 : 0.22923826384702328

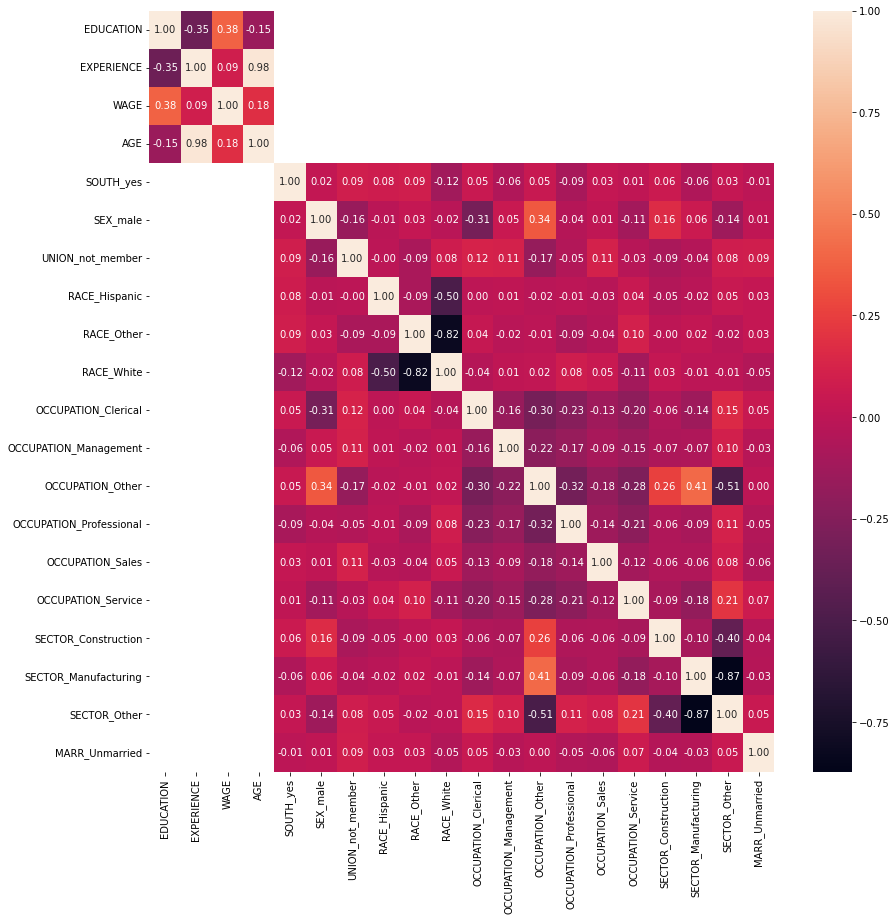
importance: [ 9.52680733e-01 1.04150893e-02 6.38366206e-02 3.35964783e+01

2.82918634e-02 1.41267230e+01 2.50403214e-03 7.96072003e-02

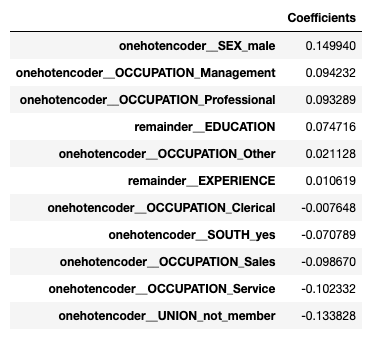
6.33716623e-03 -9.81998560e-03]



I dropped less important 'MARR', 'RACE', 'SECTOR' columns, there was high correlation on ’AGE' with EXPERIENCE, dropped it as well.



The remaining coefficients after removing 'MARR', 'RACE', 'SECTOR' and ‘AGE’ which are impacting the model are:



I tried the modified dataset again on models, the best outcome came from the smallest regularization.

**Conclusion**

The model score is not high even with minimized mean squared error. The most impacting features are SEX, UNION and OCCUPATION. UNION is skewed and OCCUPATION has multiple values in where **Sales** and **Service** are again skewed as shown in the histogram which effecting model performance, however, they are nominal variables.

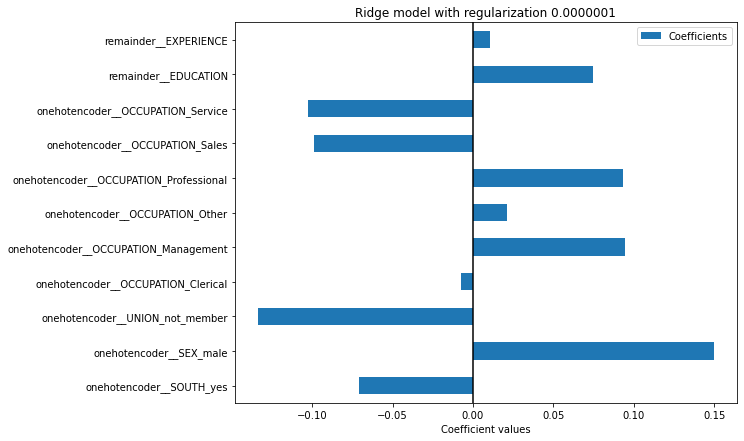
# plot coefficients

coefs.plot.barh(figsize=(9, 7))

plt.title("Ridge model with regularization 0.0000001")

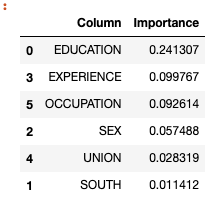
plt.axvline(x=0, color="black")

plt.xlabel("Coefficient values")



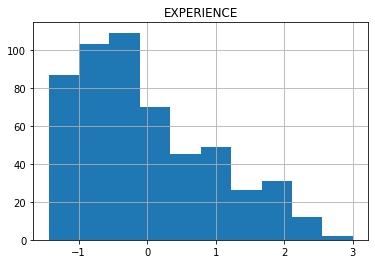
What led to **higher** wages are *male* whose occupation in *Management*, *Professional* with higher education. In contrast to that, not unionized whose occupation in Sales and Service who live in Southern region led **lower** wages.

Dataset detailed analysis: <https://www.openml.org/search?type=data&sort=runs&id=534&status=active>



#scale [['EXPERIENCE']]

((X[['EXPERIENCE']] - X[['EXPERIENCE']].mean())/X[['EXPERIENCE']].std()).hist()



Scoring = ‘neg\_mean\_squared\_error’

 ————— o —————