**Berkeley 9 - 16 AI/ML Certificate Program Modules 9 - 16**

Alex Wang: ML posts on LI

Cassie Kozyrkov: Making Friends with Machine Learning

<https://thinkpsych.com/blog/cognitive-vs-emotional-intelligence/>

**Savio’s Colab:**

<https://colab.research.google.com/drive/1m640WoWTbTgw0ymZoIw9K0wyCfMXtbYH#scrollTo=YdnBR4JQ_MMA>

One 1:1 Meeting per participant across the period of Weeks 12-15

June 8 - July 5

One 1:1 Meeting per participant across the period of Modules 21-23

August 17 - September 6

<https://student.emeritus.org/courses/4765/pages/how-to-schedule-a-1-1-session?module_item_id=999645>

<https://student.emeritus.org/courses/4765/pages/capstone-project-overview?module_item_id=999644>

<https://student.emeritus.org>

**An outline of the BH-PCMLAI program calendar**

|  |  |  |  |
| --- | --- | --- | --- |
| **Module #** | **Module Title** | **Week #** | **Date** |
| **0** | Program Orientation | 0 | **Wednesday, March 02, 2022** |
| **1** | Introduction to Machine Learning | 1 | **Wednesday, March 09, 2022** |
| **2** | Fundamentals of Machine Learning | 2 | **Wednesday, March 16, 2022** |
| **3** | Introduction to Data Analysis | 3 | **Wednesday, March 23, 2022** |
| **4** | Fundamentals of Data Analysis | 4 | **Wednesday, March 30, 2022** |
| **5** | Practical Application 1 | 5 | **Wednesday, April 06, 2022** |
| Break Week |  |  | **Wednesday, April 13, 2022** |
| **6** | Clustering and Principal Component Analysis (PCA) | 6 | **Wednesday, April 20, 2022** |
| **7** | Linear and Multiple Regressions | 7 | **Wednesday, April 27, 2022** |
| **8** | Feature Engineering and Overfitting | 8 | **Wednesday, May 04, 2022** |
| **9** | Model Selection and Regularization | 9 | **Wednesday, May 11, 2022** |
| **10** | Time Series Analysis and Forecasting | 10 | **Wednesday, May 18, 2022** |
| **11** | Practical Application 2 | 11 | **Wednesday, May 25, 2022** |
| Break Week |  |  | **Wednesday, June 01, 2022** |
| **12** | Classification and k-Nearest Neighbors (KNN) | 12 | **Wednesday, June 08, 2022** |
| **13** | Logistic Regression | 13 | **Wednesday, June 15, 2022** |
| **14** | Decision Trees | 14 | **Wednesday, June 22, 2022** |
| **15** | Gradient Descent and Optimization | 15 | **Wednesday, June 29, 2022** |
| **16** | Support Vector Machines (SVMs) | 16 | **Wednesday, July 06, 2022** |
| **17** | Practical Application 3 | 17 | **Wednesday, July 13, 2022** |
| Break Week |  |  | **Wednesday, July 20, 2022** |
| **18** | Natural Language Procession (NLP) | 18 | **Wednesday, July 27, 2022** |
| **19** | Recommendation Systems | 19 | **Wednesday, August 03, 2022** |
| **20** | Capstone 1 | 20 | **Wednesday, August 10, 2022** |
| **21** | Ensemble Techniques (GBM, XGB, and Random Forest) | 21 | **Wednesday, August 17, 2022** |
| **22** | Deep Neural Networks 1 | 22 | **Wednesday, August 24, 2022** |
| **23** | Deep Neural Networks 2 | 23 | **Wednesday, August 31, 2022** |
| **24** | Capstone 2 | 24 | **Wednesday, September 07, 2022** |

 ————— o —————

**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

**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 Excersize:

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

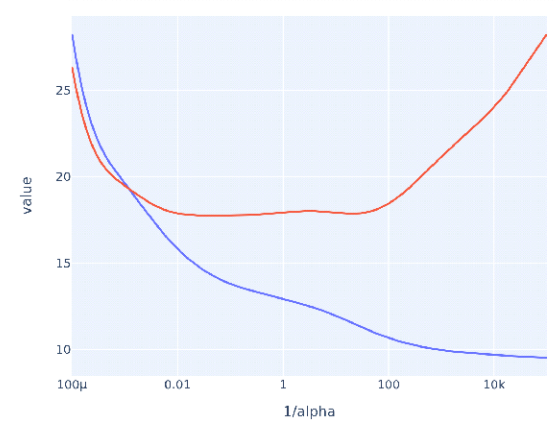
*You are correct! The answer “*Standardization*” is correct because it rescales the data.*

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

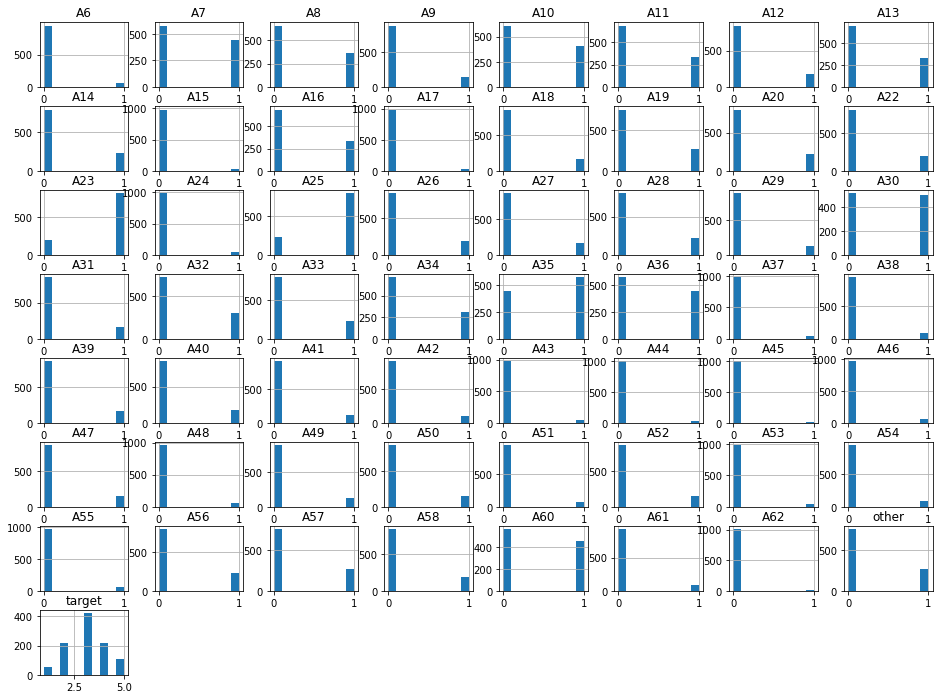
news['target'] = news['target'].astype('int64')

# 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')

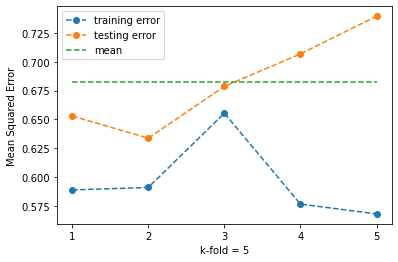
plt.plot(range(1, 6), [np.mean(test\_mses)]\*5, '--', label = 'mean')

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

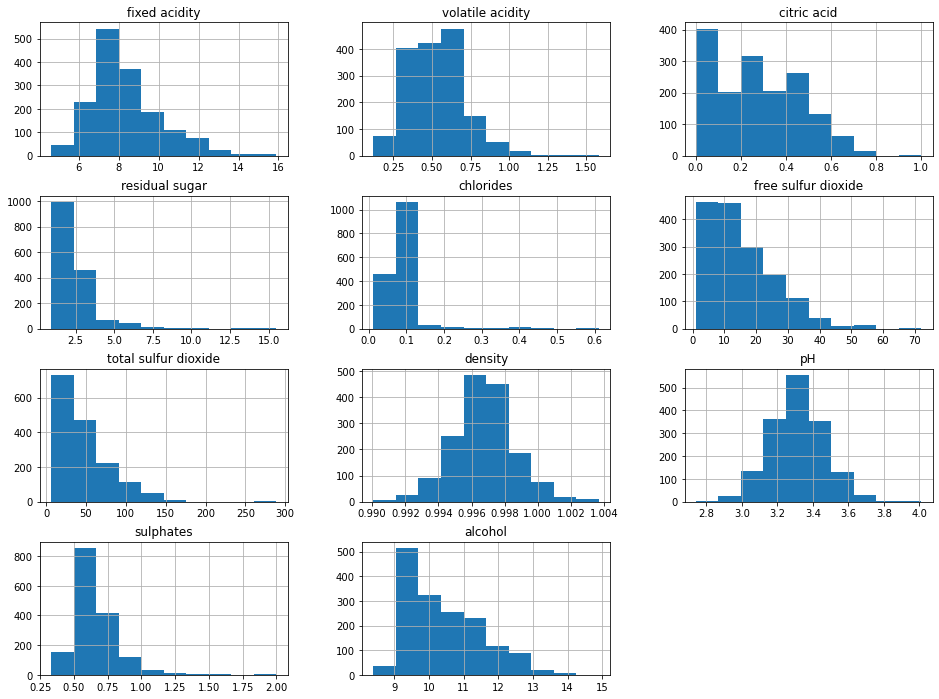
L2 is Ridge - sum of the square of the parameters

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))

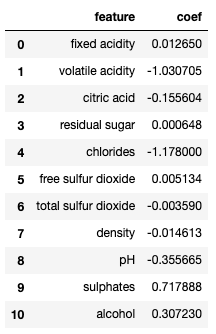
print('MSE train:', ridge\_train\_mse)

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)

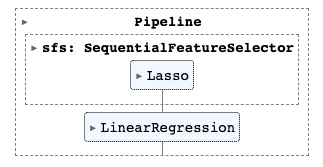
print('MSE test :', sequential\_test\_mse)

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)

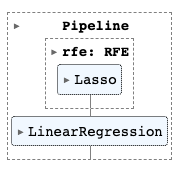
print('MSE test :', rfe\_test\_mse)

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')

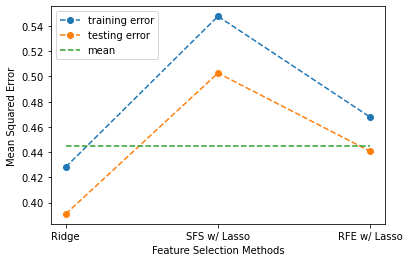
plt.plot(range(1, 4), [np.mean([ridge\_test\_mse, sequential\_test\_mse, rfe\_test\_mse])]\*3, '--', label = 'mean')

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

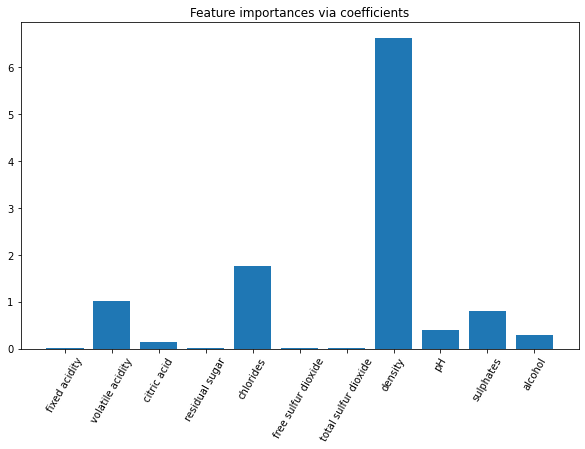
plt.xlabel('Feature Selection Methods')

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.

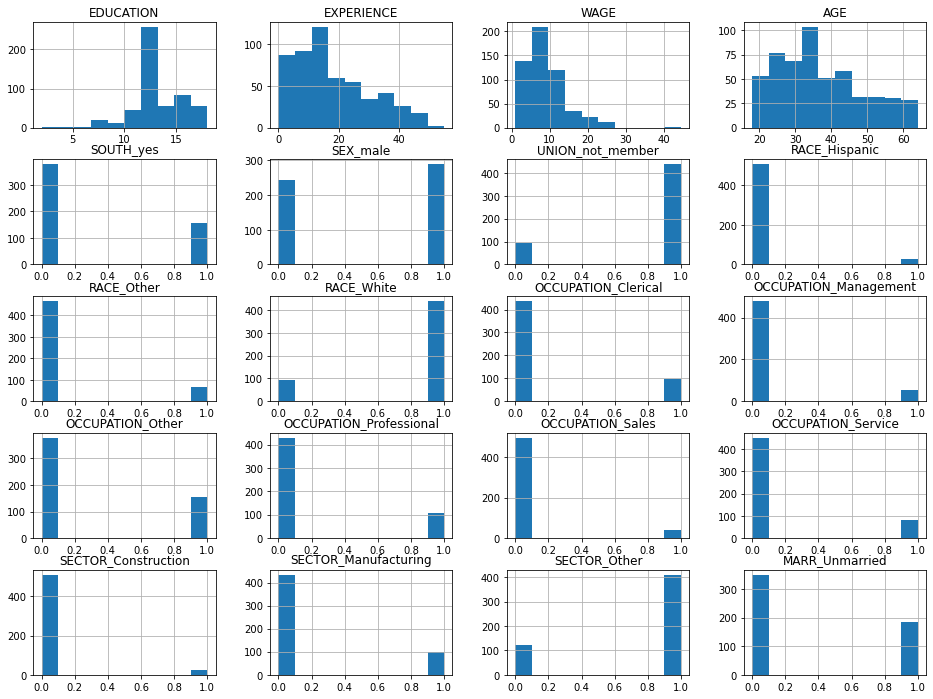
**Dataset Analysis**

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))

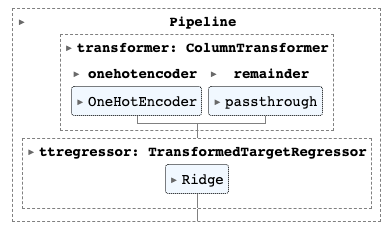
# Results

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')

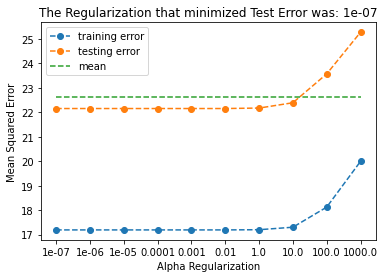
plt.xticks(range(1, complexity), [0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0])

plt.xlabel('Alpha Regularization')

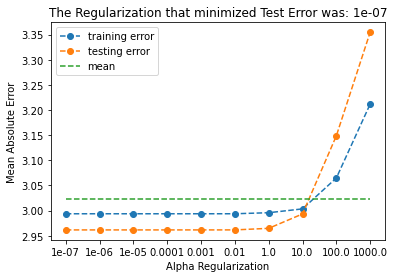
plt.ylabel('Mean Squared Error')

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)

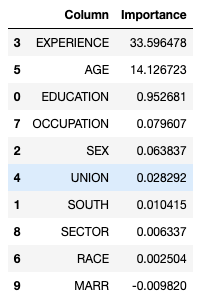
ranking

model r^2 : 0.22923826384702328

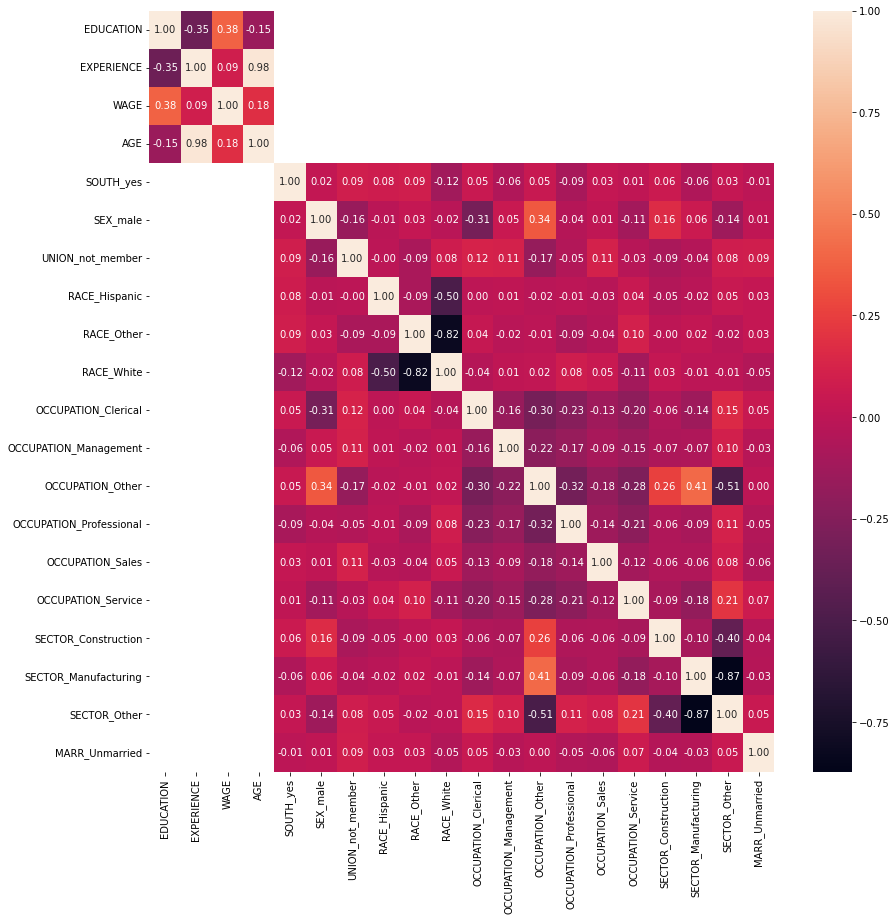
importance: [ 9.52680733e-01 1.04150893e-02 6.38366206e-02 3.35964783e+01

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

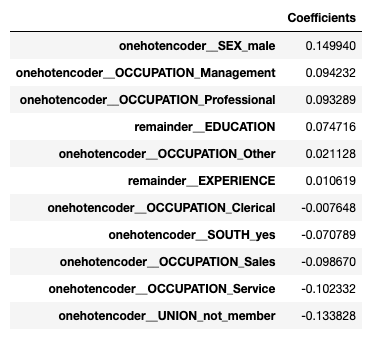
6.33716623e-03 -9.81998560e-03]



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.

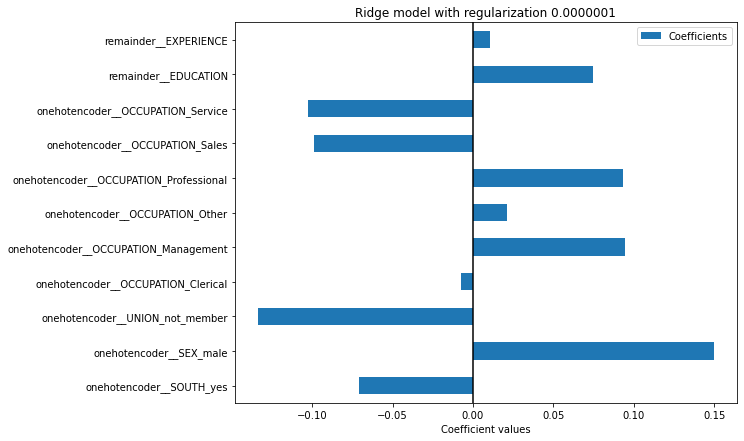
# plot coefficients

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

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

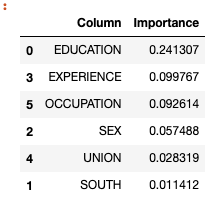
plt.axvline(x=0, color="black")

plt.xlabel("Coefficient values")



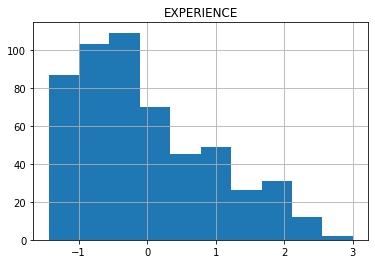
What led to **higher** wages are *male* whose occupation in *Management*, *Professional* with higher education. In contrast to that, not unionized whose occupation in Sales and Service who live in Southern region led **lower** wages.

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 —————

**Module 10**

**Time Series Analysis and Forecasting**

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3167413?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3167413/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3167412?wrap=1)
* [Solution Files for Codio Activity 10.1](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_1_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.2](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_2_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.3](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_3_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.4](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_4_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.5](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_5_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.6](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_6_solution.zip)(ZIP)
* [Lecture Datasets and Jupyter Notebooks](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/video_assets_starter.zip)(ZIP)

**Notes:**

**ARMA**

An acronym that stands for autoregressive integrated moving average; a forecasting model for a stationary time series

**Cycle**

Random, low-frequency variations in time series data

**Decomposition**

A technique to break time series data into trend, cycle, seasonality, and remainder

**Differencing Time Series**

A technique to convert a non-stationary time series into a stationary one

**Non-Stationary Time Series**

A series of data that shows seasonal effects, trends, and other structures related to the time index

**Remainder**

Any data that is not part of the trend, cycle, or seasonality in a time series dataset; also known as ‘residue’

**Seasonality**

Predictable, periodic variations in time series data known to the modeler; also known as periodicity

**Stationary Time Series**

A series of data that has no trend or seasonal effects; stationarity is assumed or required for many statistical models

**Trend**

Long-term behavior of time series data

Trends can cause a change in the mean over time, while seasonality can change the variance over time, both of which define a time series as *non-stationary*. Conversely, we make assumptions about times series datasets that do not have these same trends or seasonality, and they are referred to as *stationary*.

There are two main decomposition methods: Multiplicative and additive decomposition.

Additive decomposition states that time series data results from the sum of its components. Thus,

**Y = T + S + R**

where **Y** is the time-series data, **T** is the trend-cycle component, **S** is the seasonal component, and **R** is the remainder.

Multiplicative decomposition states that time-series data results from the product of its components. Thus,

**Y = T × S × R**

ARMA (sometimes notated as ARIMA) stands for autoregressive integrated moving average.

* **Autoregression**: A model based on observations that are correlated with lagged observations
* **Integrated**: A term that indicates that raw observations have been differentiated to make the time series stationary
* **Moving average**: A model based on the dependence between an observation and a residual error after applying a moving average model to lagged observations

**Augmented Dickey-Fuller test**

<https://machinelearningmastery.com/time-series-data-stationary-python/>

# check if stationary

# adf\_results = adfuller(tsla['Adj Close'].diff().dropna()) # check alternative if below is not stationary

adf\_results = adfuller(tsla['Adj Close'].dropna())

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[0] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -23.460407

p-value: 0.000000

Critical Values:

1%: -3.443 stationary

5%: -2.867 stationary

10%: -2.570 stationary

Means -23.460407 < -2.867 it is *stationary*!

adf\_results[1] is p-value!:

* **p-value > 0.05**: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
* **p-value <= 0.05**: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

fig, ax = plt.subplots(1, 4, figsize = (20, 5))

plot\_acf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'], ax = ax[0])

ax[0].set\_title('Original Series Autocorrelation')

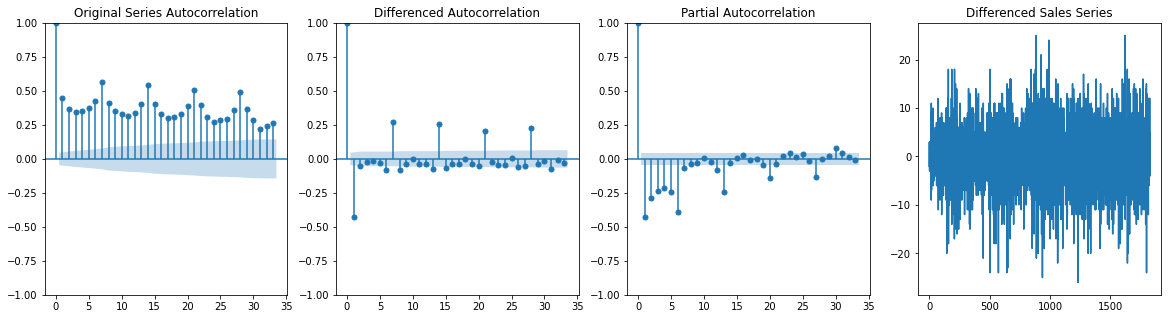
plot\_acf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna(), ax = ax[1])

ax[1].set\_title('Differenced Autocorrelation')

plot\_pacf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna(), ax = ax[2], method = 'ywm')

ax[3].plot(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna())

ax[3].set\_title('Differenced Sales Series')



# Log & diff if needed

y = np.log(gnp).diff().dropna()

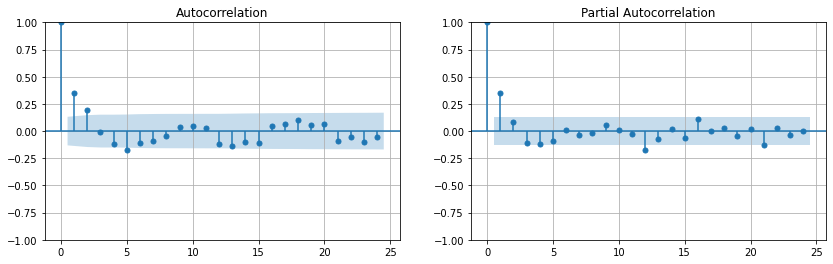
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(y, ax = ax[0])

ax[0].grid()

plot\_pacf(y, ax = ax[1])

ax[1].grid()



**Module Issues:**

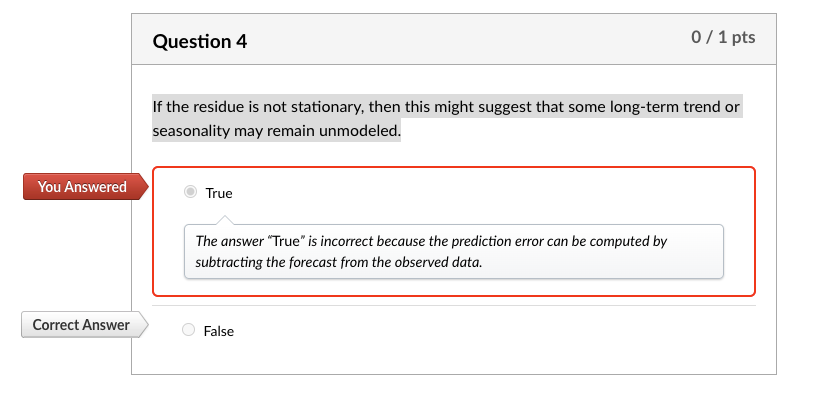
Codio 10.2 Problem 2: variable *sample* was not given in the cell upfront!

Codio 10.5 Problem 3: Fit an ARIMA model with p = 1 and q = **0**!

Codio 10.6 Problem 5: Create model with these options: “ARIMA(X\_train.diff(), order = (6, 0, 6), freq = 'D')”

Codio 10.6: Auto-grading failed to grade, upload *nbgrader\_config.py* per instructions!

In the module-end quiz the answer is wrong, pointing to something else:



**Quizes:**

Which of the following models does not represent time series data? : Autocorrelation

*You are correct! The answer “*Autocorrelation*” is correct because it is not the name of a model at all; it is an analytical tool.*

Given some historical data along a time series, predicting the future over some time period is referred to as a ‘forecasting problem.’ : True

*You are correct! The answer “*True*” is correct because a forecasting problem simply asks, “Given some historical data, what will happen in the future?”*

Prediction error is the sum of the predicted output and the actual data. : False

*You are correct! The answer “*False*” is correct because prediction error is the difference between the prediction and the actual data.*

Which of the following techniques is not commonly used to calculate the error of a model for time series data? : Mean squared error (MSE)

*You are correct! The answer “*Mean squared error (MSE)*” is correct because the most common techniques for calculating the error of a time series model are MAE and RMSE.*

What is a defining characteristic of stationary time series data? : The data has no trend or seasonal effects

*You are correct! “*The data has no trend or seasonal effects*” is correct because time series data is said to be stationary when there are no seasonal effects or clear trends in the data.*

A stochastic process is an unordered sequence of random variables. : False

*You are correct! The answer “*False*” is correct because a stochastic process is an ordered sequence of random variables.*

The stochastic process is represented as (Yt)1:T. What does the symbol “T” represent? : Length of stochastic process

*You are correct! The answer “*Length of stochastic process*” is correct because the symbol “T” represents the length of the stochastic process.*

A single sample from a stochastic process is known as (blank). : A time series

*You are correct! The answer “*A time series*” is correct because a single sample from a stochastic process is known as a time series.*

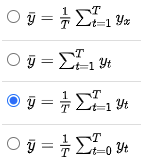
Independent and identically distributed (IID) processes are stationary. : False

*You are correct! The answer “*False*” is correct because IID processes are both stationary and independent.*

In the autocorrelation matrix, which value represents all values in the diagonal? : 1

*You are correct! The answer “*1*” is correct because in the diagonals, each value is the correlation of (Yt) with itself.*

What is the formula for the mean “y¯” of a time series? :



*You are correct! The answer "*y¯=1T∑t=1Tyt*" is correct because this is the formula for the mean of a time series.*

The Python library scikit-learn is used to build time series models. : False

*You are correct! The answer “*False*” is correct because the Python library statsmodels is used to build time series models.*

The ArmaProcess() object takes two constructors. What are these constructors? *(Check all that apply.)* : ar, ma

*You are correct! The answers “*ar*” and “*ma*” are correct because these are the two constructors used to create*ArmaProcess()*objects.*

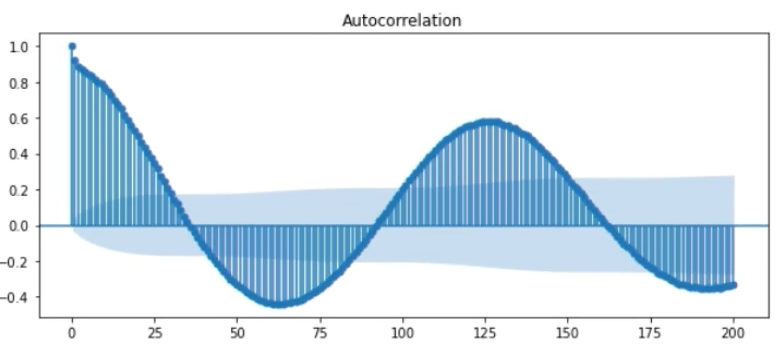
Which function is called on an ArmaProcess() object to give 20 autocorrelation values? : acf(lags=20)

*You are correct! The answer*“acf(lags=20)”*is correct because this is the function called on an*ArmaProcess()*object to get autocorrelation values.*

The function tsaplots.plot\_acf()is used to plot the autocorrelations of a sample of an ArmaProcess() object. : True

*You are correct! The answer “*True*” is correct because the function is used to plot the autocorrelations of a sample of an*ArmaProcess()*object.*

The plot below of the autocorrelation of a data sample shows that it is stationary. : False



*You are correct! The answer “*False*” is correct because for the data to be stationary, the autocorrelations should decay to zero.*

Any behavior in time series data that is not described by long-term behavior, random low-frequency variations, or known periodicity is considered a (blank). : Residual

*You are correct! The answer “*Residual*” is correct because any behavior in time series data that is not described by long-term behavior, random low-frequency variations, or known periodicity is considered a residual.*

To extract a trend from the time series data, a filter “f” is applied, which removes the components that are (blank). : Seasonal

*You are correct! The answer “*Seasonal*” is correct because the filter “f” is chosen as an array of positive numbers that sums up to 1. It will remove seasonal components for the period of the time series, leaving trends and cycles.*

If the residue is not stationary, then this might suggest that some long-term trend or seasonality may remain unmodeled. : True

*You are correct! The answer “*True*” is correct because if the residue is not stationary, it might suggest that trends or seasonality remain unmodeled.*

You can compute the time series prediction error by subtracting the trend from the observed data. : False

*You are correct! The answer “*False*” is correct because the prediction error can be computed by subtracting the forecast from the observed data.*

Which function is used in Python to extract the trend from time series data given the “filter” and the “data” as constructors? : convolution\_filter(data,filter)

*You are correct! The answer*“convolution\_filter(data,filter)”*is correct because this is the function used to extract trends from time series data given the “data” and “filter” as constructors.*

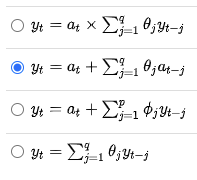
Which snippet of Python code allows you to get the mean absolute error of a data named “pred\_error”? : np.abs(pred\_error).mean()

*You are correct! The answer*“np.abs(pred\_error).mean()”*is correct because to get the mean absolute error, absolute values of the data are taken, and then their mean is computed.*

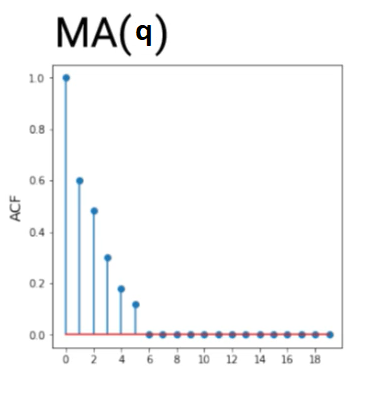
The ARMA family of models captures the time-variant structure exhibited in stationary time series. : False

*You are correct! The answer “*False*” is correct because the ARMA family of models captures the time-invariant structure exhibited in stationary time series.*

What is the mathematical formula for summarizing the moving average (MA) process with order “q”? : yt = at + sum of theta x at



*You are correct! The answer “*yt=at+∑j=1qθjat−j*” is correct because this is the formula for MA(q).*



Consider the above plot of ACF values for a moving average process with order “q”. What can you deduce as the value of q from the graph? : 5

*You are correct! The answer “*5*” is correct because the direct method for identifying the order of a moving average process is to look at the lag of the largest non-zero entry in the ACF.*

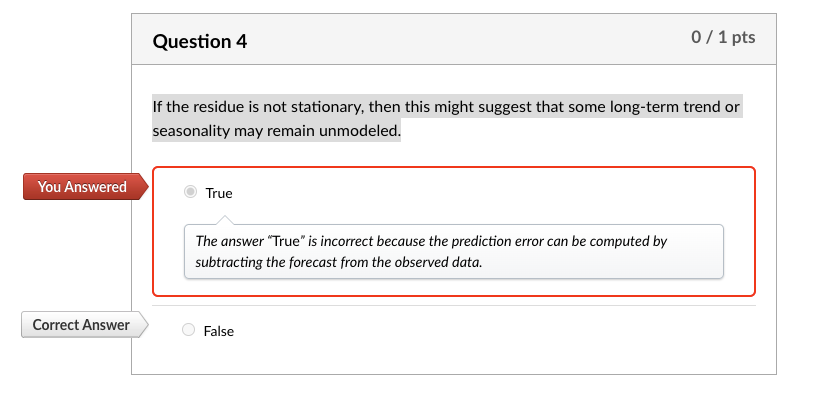
To get only the AR of an ARMA model, the parameter q is set to zero. : True

*You are correct! The answer “*True*” is correct because in the ARMA model, the parameter q is for MA. When q is set to zero, the model that remains is AR.*

What is the second step in building an ARMA model? : Use SACF and SPACF to select p and q

*You are correct! The answer “*Use SACF and SPACF to select p and q*” is correct because this is the second step in building an ARMA model.*

In the module-end quiz the answer is wrong, pointing to something else:



<https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/>

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

* **Level**: The average value in the series.
* **Trend**: The increasing or decreasing value in the series.
* **Seasonality**: The repeating short-term cycle in the series.
* **Noise**: The random variation in the series.

**Try it 10.1: Decomposing Time Series**

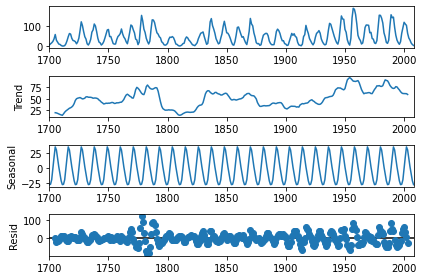
I studied sunspots and Air Passenger datasets for this exercise.

I directly set YEAR as index on sunspots, and transformed Month to timestamp and set it as index on air dataset, no cleanup or any other transformation. I set the period to 11 years (~128 month from the lecture video.):

ss\_results = seasonal\_decompose(ss, model = 'additive', period = 11)

ss\_results.plot()

plt.show



Reconstruct and overlay:

# check if aditive model match seasonal pattern when reconstructed

plt.plot(ss\_results.seasonal+ss\_results.trend, label = 'seasonal + trend')

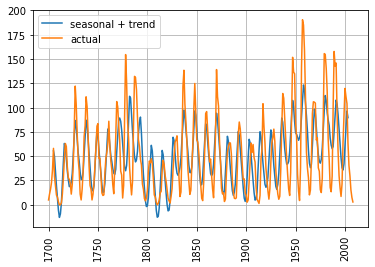
plt.plot(ss, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show



It looks overlap with the actual data.

I analyzed two given datasets by seasonal\_decompose, it required a parameter to specify whether the model is *additive* or *multiplicative*. **sunspots** dataset has 0 values in SUNACTIVITY column so it went by **additive** model by default.

I did an *additive* model first on Air Passenger dataset, however, when I plotted *'seasonal + trend'* it did not overlap with the actual data as below:

# check if aditive model match seasonal pattern when reconstructed

plt.plot(air\_results.seasonal + air\_results.trend, label = 'seasonal + trend')

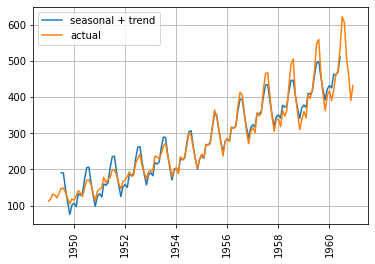
plt.plot(air, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show

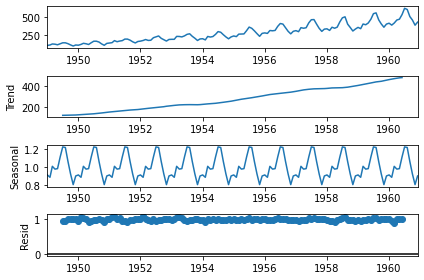


So, I went with a **multiplicative** model:

air\_results = seasonal\_decompose(air, model = 'multiplicative')

air\_results.plot()

plt.show



The **multiplicative** model plot overlapped well with the actual data comparing to the ‘additive’ model:

plt.plot(air\_results.seasonal \* air\_results.trend, label = 'seasonal \* trend')

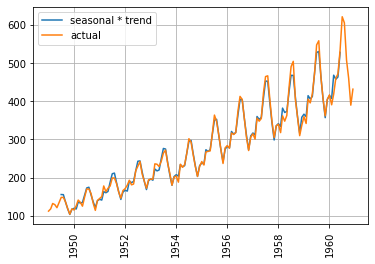
plt.plot(air, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

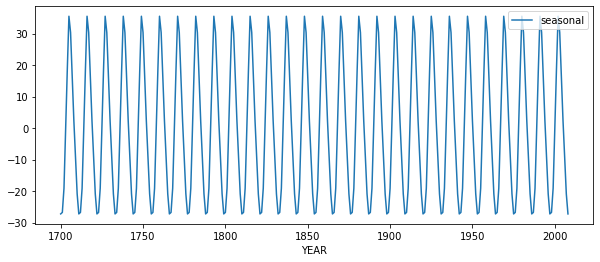
plt.show



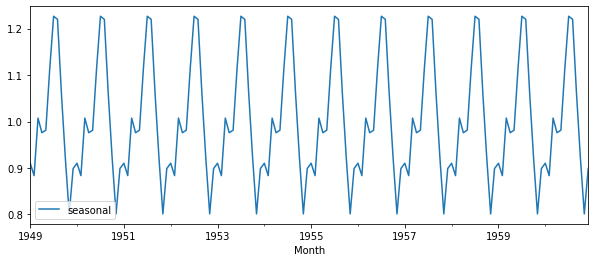
Just observe the height of each cycle for comparing these 2 plots, in the additive model they are the same heights of cycles whereas in the multiplicative model, the heights notably vary.

**Seasonal Components**

Sunspots seasonal component:



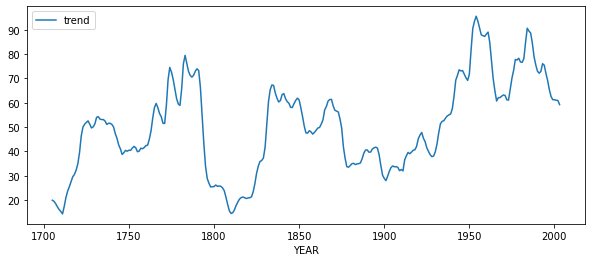
Air Passengers seasonal component:



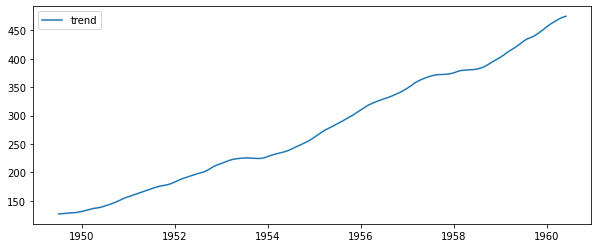
Both seasonal components on datasets showing similar behavior in periodical variability.

**Trend Components**

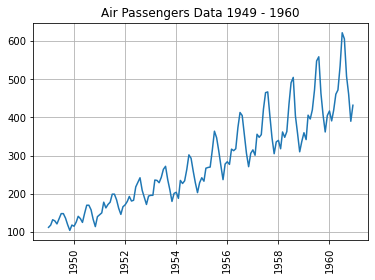
Sunspots trend component:



Air Passengers trend component:



*Air Passengers trend* shows gradual increase over time which differs from *Sunspots trend* shows erratic upward or downward movement.



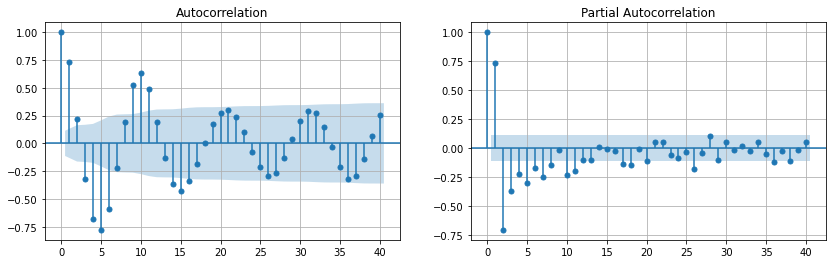
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(ss\_results.resid.dropna(), ax = ax[0], lags=40)

ax[0].grid()

plot\_pacf(ss\_results.resid.dropna(), ax = ax[1], lags=40)

ax[1].grid()



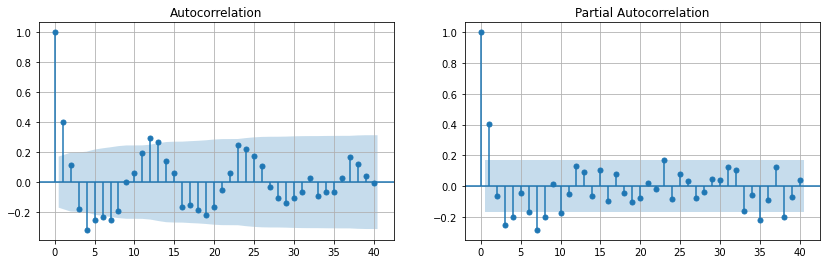
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(air\_results.resid.dropna(), ax = ax[0], lags=40)

ax[0].grid()

plot\_pacf(air\_results.resid.dropna(), ax = ax[1], lags=40)

ax[1].grid()



————— o —————

**Try-It Activity 10.2: Forecasting with Decomposition Models**

In this activity, your goal is to identify a new (to you) time series dataset and build a forecast using a seasonal and trend additive or multiplicative model using statsmodels.

Summarize your findings in an executive brief that explores the following:

* **Data description:** Provide a high-level overview of your data and its timeframe along with general information on your dataset
* **Forecast:** Give a description of the forecast. Describe the period that was projected and what the forecast implies about your data. Be sure to include presentation-ready plots with appropriate labels and titles.
* **Uncertainty:** Discuss the evaluation of your model on testing data, and explore the residuals. Discuss the consequence of this error for your model and forecasts. Is there still structure to uncover?

**ARIMA freq parameter**

freq : str {'B','D','W','M','A', 'Q'}

'B' - business day, ie., Mon. - Fri.

'D' - daily

'W' - weekly

'M' - monthly

'A' - annual

'Q' - quarterly

Additive / Multiplicative

Univariate Time Series

Detrended data: mean is at 0!

——

**Data Introduction and Cleanup**

I found a dataset contains 9357 records of air pollutants with hourly averaged responses from 5 sensors, the measurement was taken for one year at road level in an Italian town: <https://archive.ics.uci.edu/ml/datasets/Air+Quality>. I only used CO reading, I did some cleanup as follows:

1. Made CO column float after replacing decimal symbol to ‘.’ from ‘,’: airq['CO'] = airq['CO(GT)'].str.replace(',','.').astype('float')
2. Missing measurement marked as -200, replaced them with mean() value: airq[airq['CO'] == -200] = 2.15275
3. Transformed date and time columns to airq['datetime'] = pd.to\_datetime(aq['Date'] + ' ' + aq['Time'], format='%d/%m/%Y %H.%M.%S') and set index

**Build historical and future datasets**

Split last 4 days (96 hours) for future and rest in hist:

# split data into hist and future

y\_hist = airq[:-96]

y\_future = airq[-96:]

**Extracting the trend**

**10 Points**

I created an instance of the STL estimator and passed y\_hist, period = 24 for one day.

stl = STL(y\_hist, period=24)

results = stl.fit()

# Plot trend with original series

plt.plot(y\_hist)

plt.plot(results.trend)

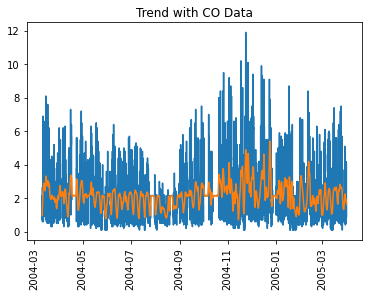
#plt.plot(results.seasonal)

plt.title('Trend with CO Data')

#plt.legend()

plt.xticks(rotation = 90)

plt.show



**Model Historical Series and Seasonal+Trend**

I checked the seasonal+trend plot if it is a good model, looked good:

# additive model fits well! no need for multiplicative model…

season\_and\_trend = results.seasonal + results.trend

# Plot

#plt.plot(season\_and\_trend, label = 'seasonal + trend')

plt.plot(y\_hist, label = 'actual')

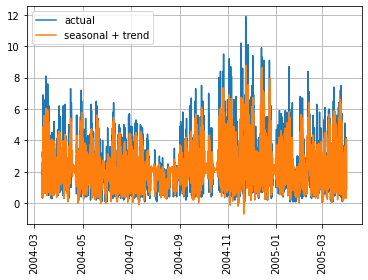
plt.plot(season\_and\_trend, label = 'seasonal + trend')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show



**Examining the residuals, Examining Error in Forecast**

Then, I checked both residual plot and Dickey Fuller test below:

# plot residual on actual series

plt.plot(y\_hist, label = 'actual')

plt.plot(results.resid, label = 'residue')

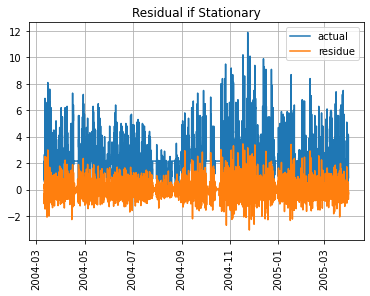
plt.grid()

plt.title('Residual if Stationary')

plt.legend()

plt.xticks(rotation = 90)

plt.show



# check stationarity by Dickey Fuller at different thresholds

adf\_results = adfuller(results.resid)

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[1] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -21.211974

p-value: 0.000000 stationary

Critical Values:

1%: -3.431 stationary

5%: -2.862 stationary

10%: -2.567 stationary

(-21.211974346936564,

0.0,

38,

9222,

{'1%': -3.431059295288066,

'5%': -2.8618534634125252,

'10%': -2.566936851507016},

9094.238191568482)

Passed! Stationary!

Forecast future values to compare, I played with p,d,q parameters, below produced better results in terms of MAE and RMSE values:

#instantiate

stlf = STLForecast(y\_hist, ARIMA, model\_kwargs={'order':(2, 0, 2), 'trend':"t", 'enforce\_stationarity': True})

#fit model using historical data

stlf\_results = stlf.fit()

#produce forecast for future data

forecast = stlf\_results.forecast(len(y\_future))

Plot entire series

plt.subplots(figsize=(16,6))

plt.plot(forecast, label = 'forecast')

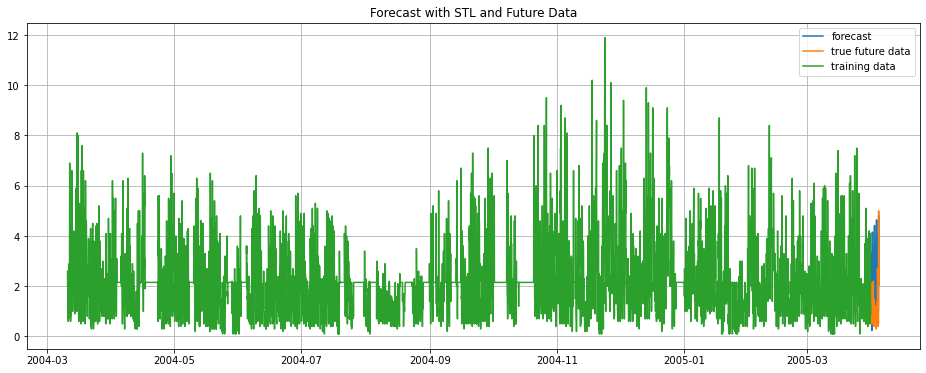
plt.plot(y\_future, label = 'true future data')

plt.plot(y\_hist, label = 'training data')

plt.legend()

plt.title('Forecast with STL and Future Data')

plt.grid()



I compared the forecast with future data if matched:

plt.subplots(figsize=(10,4))

plt.plot(forecast, label = 'Forecast')

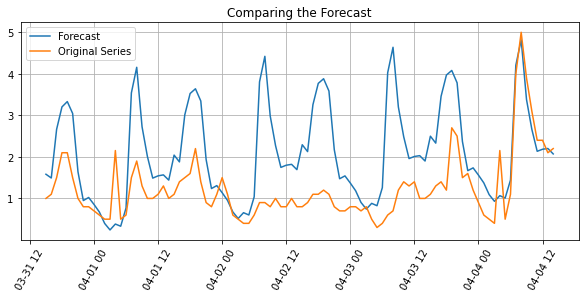
plt.plot(y\_future, label = 'Original Series')

plt.title('Comparing the Forecast')

plt.legend()

plt.xticks(rotation = 60)

plt.grid()



Prediction errors:

pred\_error = y\_future['CO'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

rmse2 = np.sqrt(mean\_squared\_error(y\_future['CO'], forecast))

# Results

print(f'MAE : {mae}')

print(f'RMSE : {rmse}')

print(f'RMSE2 : {rmse2}')

MAE : 0.9724595068578156

RMSE : 1.2966376019704622

RMSE2 : 1.2966376019704622

**Conclusion**

The dataset has only one year of data, may not be enough for showing seasonal pattern, trend, by looking at the forecast and actual plot above, there is some room for improvement perhaps with different modeling techniques.

——

Do not include:…..

I plotted Autocorrelation and Partial autocorrelation charts:

# plot ACF and PACF with plain series

fig, ax = plt.subplots(1, 2, figsize = (14, 4))

# ACF

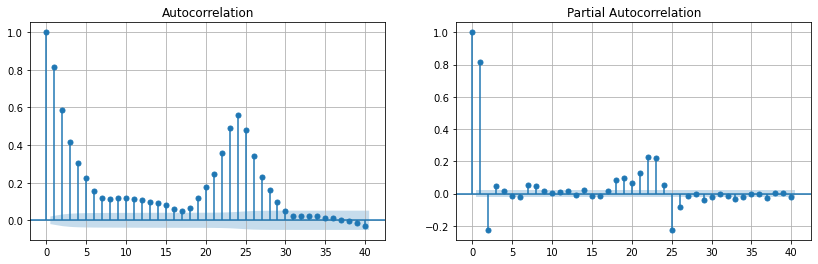
plot\_acf(airq, ax = ax[0])

ax[0].grid()

# PACF

plot\_pacf(airq, ax = ax[1])

ax[1].grid()



——

————— o —————

**Try-It Activity 10.3: Building and Evaluating ARMA Models**

First, you learned about the importance of transforming your data into a stationary series. You tested for stationarity using the adfuller function and interpreted the value of the hypothesis test. If the data was not stationary, you learned how to apply strategies, such as differencing and logarithmic transformations, to achieve stationarity.

Once the series was stationary, building an ARMA model involved using autocorrelation and partial autocorrelation plots to determine the appropriate *p* and *q* parameters of the model.

This activity asks you to identify a time series of interest to you and build an ARMA model to construct a basic forecast for the series and analyze the error. You might also consider building models with different *p* and *q* parameters because, while ACF and PACF plots are helpful, they provide rough ideas of the appropriate parameters, and it is usually good practice to perform a simple grid search on these.

* **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
* **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
* **MA**: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The parameters of the ARIMA model are defined as follows:

* **p**: The number of lag observations included in the model, also called the lag order, is the number of autoregressive terms.
* **d**: The number of times that the raw observations are differenced, also called the degree of differencing, is the number of nonseasonal differences needed for stationarity.
* **q**: The size of the moving average window, also called the order of moving average, is the number of lagged forecast errors in the prediction equation.

a suite of lag values (*p*) and just a few difference iterations (*d*) and residual error lag values (*q*).

*autocorrelations* (correlations with its own prior deviations from the mean) remain constant over time

[How to Grid Search ARIMA Model Hyperparameters with Python](https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/)

**Dataset Introduction**

I found a time series dataset for price of gold from 1970 to 2020 on Kaggle: <https://www.kaggle.com/datasets/arashnic/learn-time-series-forecasting-from-gold-price>, there are 10787 records, it is ready to use once Date is transferred to timestamp data type and set as index:

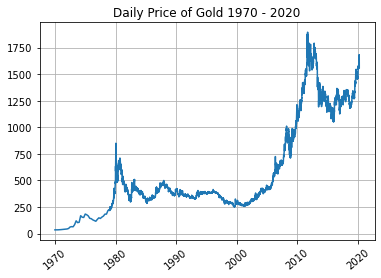
gold=gd.set\_index(pd.to\_datetime(gd['Date'])).drop('Date', axis=1)

plt.plot(gold['Value'])

plt.grid()

plt.xticks(rotation = 40)

plt.title('Daily Price of Gold 1970 - 2020')



I checked ACF and PACF also Dickey Fuller test on gold dataset:

# plot ACF and PACF with plain series

fig, ax = plt.subplots(1, 2, figsize = (14, 4))

# ACF

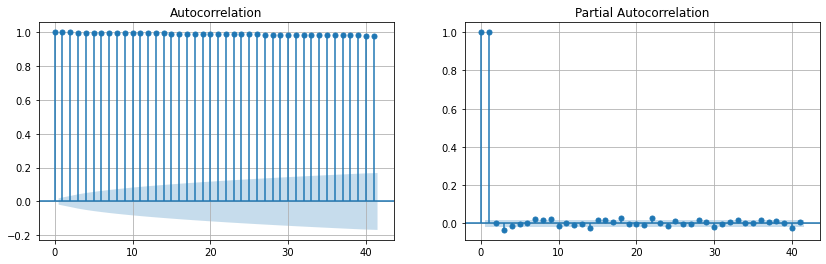
plot\_acf(gold, ax = ax[0])

ax[0].grid()

# PACF

plot\_pacf(gold, ax = ax[1])

ax[1].grid()



It is not stationary data, Dickey Fuller test confirms that too:

# check stationarity by Dickey Fuller at different thresholds

adf\_results = adfuller(gold)

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[1] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -0.071121

p-value: 0.952240 not stationary

Critical Values:

1%: -3.431 not stationary

5%: -2.862 not stationary

10%: -2.567 not stationary

Out[98]:

(-0.07112124453813447,

0.9522400769147394,

39,

10747,

{'1%': -3.4309586221840513,

'5%': -2.861808976860248,

'10%': -2.566913171245489},

77621.49868017703)

I created a differenced dataset from this series and plotted ACF, PACF again, Dickey Fuller test:

# do diff and dropna

gold\_df = gold.diff().dropna()

# plot charts

fig, ax = plt.subplots(1, 3, figsize = (20, 5))

plot\_acf(gold\_df, lags=20, ax = ax[0])

ax[0].set\_title('Gold Series Autocorrelation')

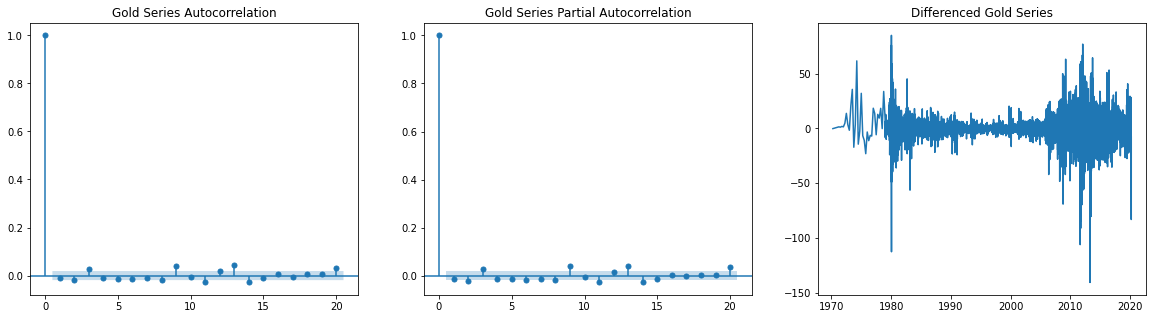
plot\_pacf(gold\_df, lags=20, ax = ax[1])

ax[1].set\_title('Gold Series Partial Autocorrelation')

ax[2].plot(gold\_df)

ax[2].set\_title('Differenced Gold Series')

plt.show



It is stationary!

I created a simple ARMA model and a grid search for it, I split dataset into historical and last 30 days for forecast in the function:

def arma\_grid\_search(data, p\_values, d\_values, q\_values):

# split data

y\_hist, y\_future = data[:-30], data[-30:]

best\_rmse, best\_order = float("inf"), None

for p in p\_values:

for d in d\_values:

for q in q\_values:

arima\_order = (p, d, q)

# build ARMA model & fit

arma = ARIMA(y\_hist, order = arima\_order).fit()

hist\_preds = arma.predict()

# forecast

forecast = arma.forecast(len(y\_future))

# set index same as y\_future!

forecast.index=y\_future.index

pred\_error = y\_future['Value'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

print('ARIMA Order=%s MSE=%.3f' % (arima\_order, rmse))

if rmse < best\_rmse:

best\_rmse, best\_config = rmse, arima\_order

# Results

print('Best ARIMA Order=%s MSE=%.3f' % (best\_config, best\_rmse))

# seed values from PACF plot!

p\_values = [0, 1, 2, 3, 9, 13]

d\_values = [0]

q\_values = range(0, 3)

arma\_grid\_search(gold\_df, p\_values, d\_values, q\_values)

The best order option came as (13,0,2), when I fed it into model and predict:

arma = ARIMA(y\_hist, order = (13, 0, 2)).fit()

hist\_preds = arma.predict()

# Results

print(hist\_preds.tail())

plt.figure(figsize = (12, 4))

plt.plot(y\_hist, label = 'Historical')

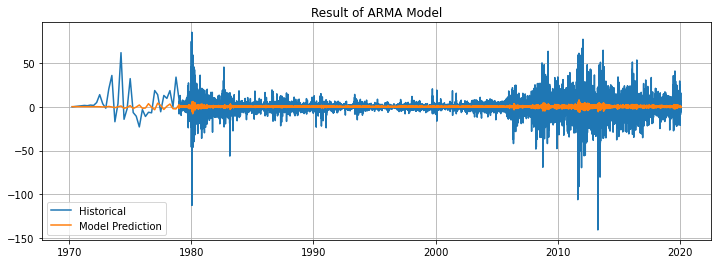
plt.plot(hist\_preds, label = 'Model Prediction')

plt.legend()

plt.grid()

plt.title('Result of ARMA Model')

plt.show



# forecast

forecast = arma.forecast(len(y\_future))

# set index same as y\_future!

forecast.index=y\_future.index

pred\_error = y\_future['Value'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

# Results

print(f'MAE : {mae}')

print(f'RMSE : {rmse}')

MAE : 15.945910619864206

RMSE : 22.13985346444464

plt.subplots(figsize=(10,4))

plt.plot(y\_future, label = 'Differenced Gold Series')

plt.plot(forecast, label = 'Forecast')

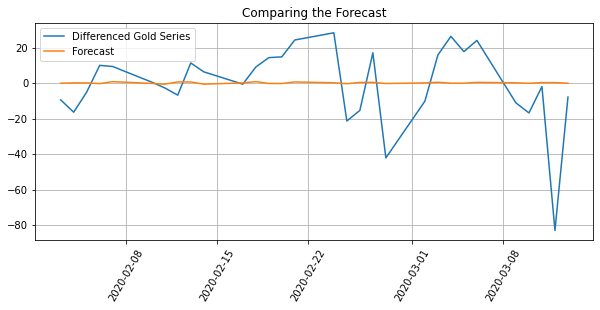
plt.title('Comparing the Forecast')

plt.legend()

plt.xticks(rotation = 60)

plt.grid()

plt.show



I could only come up a simple grid search by utilizing for loop, although, it vigorously iterating all given possibilities, the forecast fluctuates at zero line, however, it does not seem follow trend as its root mean squared error considerably remained high.

 ————— o —————

**Module 11**

**Practical Application 2**

In Module 11, you will define your problem statement and develop a prospectus of the project. The prospectus provides a general overview of the question you will be asking, what data you think you will need to answer the question, and the techniques you might use to answer the question. This submission is pass/fail because it is expected that your plan will change based on your facilitator’s feedback.

**Notes:**

[Health Care Hotspotting — A Randomized, Controlled Trial](https://www.nejm.org/doi/full/10.1056/NEJMsa1906848)

**Module Issues:**

**Quizes:**

**Practical Application Assignment 11.1: What Drives the Price of a Car?**

**sources:**

<https://notebooks.githubusercontent.com/view/ipynb?browser=chrome&color_mode=auto&commit=e9d972eb85ceb404bf3aecedb6bea6dd4de4b533&device=unknown&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f5473656c6d65672d432f556461636974792d446174615f536369656e746973745f4e616e6f6465677265655f50726f6a6563745f4f6e652d4361725f41756374696f6e5f446174617365745f416e616c797369732f653964393732656238356365623430346266336165636564623662656136646434646534623533332f4361725f41756374696f6e5f416e616c797369735f4769746875622e6970796e62&logged_in=false&nwo=Tselmeg-C%2FUdacity-Data_Scientist_Nanodegree_Project_One-Car_Auction_Dataset_Analysis&path=Car_Auction_Analysis_Github.ipynb&platform=android&repository_id=264167946&repository_type=Repository&version=99>

<https://www.kaggle.com/datasets/doaaalsenani/usa-cers-dataset>

Original set:

<https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data>

Use it for data cleaning:

<https://github.com/panambY/Used_Car_Price/blob/master/Used_Car_Price.ipynb>

<https://yxiegt.github.io/CS7641Project/>

Report:

<https://medium.com/swlh/exploring-and-analyzing-used-car-data-set-2e2bf1f24d52>

**CRISP-DM**

**Business Understanding**

**Determine Business Objectives**

* *Background*
* *Business Objectives*
* *Business Success Criteria*

**Assess Situation**

* *Inventory of Resources Requirements, Assumptions and Constraints*
* *Risks and Contingencies*
* *Terminology*
* *Cost and Benefits*

**Determine Data****Mining Goals**

* *Data Mining Goals*
* *Data Mining Success Criteria*

**Produce Project Plan**

* *Project plan*
* *Initial Assessment of Tools and Techniques*

We would like identify what factors make a car more or less expensive and suggest the outcome to a used car dealership.

Some factors weigh in as personal preferences such as color and style versus more tangible features like MPG or HP.

Data may be in unstructured form, so, it may have missing element, duplication across rows or columns.

Visualization and Outcome Findings: give much details

**Data understanding**

**Collect Initial Data**

* *Initial Data Collection Report*

**Describe Data**

* *Data Description Report*

**Explore Data**

* *Data Exploration Report*

**Verify Data Quality**

* *Data Quality Report*

**Data preparation**

**Select Data**

* *Rationale for Inclusion/Exclusion*

**Clean Data**

* *Data Cleaning Report*

**Construct Data**

* *Derived Attributes*
* *Generated Records*

**Integrate Data**

* *Merge Data*

**Format Data**

*Reformatted Data*

For regression: do .corr() and remove correlated 75% variables (do not list them more 2+ correlated ones)

Check if model overfitting!

Regions by states: <https://www.infoplease.com/us/states/regions-of-the-us>

**Modeling**

**Select Modeling Technique**

* *Modeling Techniques*
* *Modeling Assumptions*

**Generate Test Design**

* *Test Design*

**Build Model**

* *Parameter Setting*
* *Models*
* *Model Description*

**Assess Model**

* *Model Assessment*
* *Revised Parameter Setting*

**Evaluation**

**Evaluate Results**

* *Assessment of Data Mining Results w.r.t Business Success Criteria*
* *Approved Models*

**Review Process**

* *Review of Process*

**Determine Next Steps**

* *List of Possible Actions*
* *Decision*

Check if model overfitting!

**Deployment**

**Plan Deployment**

* *Deployment Plan*

**Plan Monitoring and Maintenance**

* *Monitoring and Maintenance Plan*

**Produce Final****Report**

* *Final Report*
* *Final Presentation*

**Review Project**

* *Experience Documentation*

Obtain Data

Understand Data

What columns missing, clean up,

**Module 11—Draft the Problem Statement**

In customer service, human interaction is an expensive operation where we try to avoid for cost saving. I would like to use our company data which is customer journey platform data first to predict customer interaction pain-points. Also, there are some areas which end up with negative customer experience, each customer is unique in the way how they interact with our product.

These are key operations which may impact negatively any customer:

* Payment transaction failure
* Failure of adding a bank account to wallet
* Failure of adding a credit card account to wallet
* Payment transaction put on hold

These models would be custom for each user

Dataset is time series when an event occurs so there is no fixed interval between records, they occur at arbitrary time interval, however, all in chronological order.

Data cleanup: mask data to generalize first

Data readiness: Identify above scenarios in dataset and build appropriate samples per segment for training and test data.

Link to book your 1:1 session during Weeks 12-15:

* Savio Saldanha (Section A): [https://calendly.com/consultsavio/bh-pcmlai-22-03-capstone-session-1 (Links to an external site.)](https://calendly.com/consultsavio/bh-pcmlai-22-03-capstone-session-1)
* Matilde D'Amelio (Section B): <https://calendly.com/matilde-damelio/bh-pcmlai-22-03-capstone-consultation-session-1>

*# Used-Car-Pricing*

Exploring a dataset from Kaggle that contains information on 3 million used cars for clear recommendations as to what consumers value in a used car.

The input file in data folder!

The assignment of Jupyter notebook for pricing model of a used car is [pricing.11.Aykan.ipynb](*pricing.11.Aykan.ipynb*).

**What drives the price of a car?**

In this assignment, a dataset from Kaggle is used which has 3 million used cars. The objective is to understand what factors influence price of a used car. As a result of the analysis, a recommendation about what consumers value in a used car was provided at the end below.

**Used Car Dealership - Business Understanding**

Used Car Dealership would like to understand what factors weigh in pricing a used car. There are highly influential features that directly contribute to the price of a car versus personal preferences. In this study, features will be identified as important and analyzed for multicollinearity. Finally, several models will be built and best model result will be used in the conclusion as outcome. The objective of this study to find out the important features which are making the price of a used car and create a model to estimate price of a used car and share the results with the dealership. A Kaggle dataset is shared for this purpose: Here, *price* is the target variable and the rest of dataset is features. Categorical features must be converted to numeric features so that they can be utilized in variance inflation factor and feature importance analysis also model creation.

**Data Understanding**

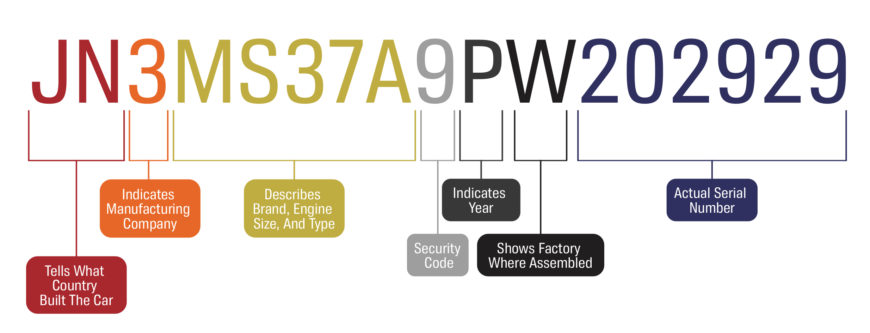
The dataset consists of these columns: id, region, price, year, manufacturer, model, condition, cylinders, fuel, odometer, title\_status, transmission, VIN, drive, size, type, paint\_color and state. Among those columns id is redundant; region, price and state have data consistency, there is no missing data; the rest of columns have all missing data. Records must be analyzed to remove duplicated information. Analyze all string data objects and come up with a method for each to convert some sort of numeric feature.

**Understanding Vehicle Identification number (VIN)**

For more information about what VIN is please refer to this wiki page: [<https://en.wikipedia.org/wiki/Vehicle_identification_number>](<https://en.wikipedia.org/wiki/Vehicle_identification_number>).

A VIN consists of Manufacturer, Brand/Model/type, Year, Factory and Serial Number as shown in below image:

![](images/VIN.jpg)



**World Manufacturer Identifier**

The first three digits of a vehicle’s VIN make up the World Manufacturer Identifier (WMI).

* The **first digit** defines the country of origin or the final point of assembly of your vehicle.
* The **second digit** indicates the manufacturer and the region where your vehicle was produced.
* The **third digit**, when coupled with the first two digits, indicates your vehicle’s type or manufacturing division.

**Vehicle Description Section**

Digits in position 4 through 9 make up the Vehicle Description Section (VDS).

* The **fourth through eighth digits** describe your vehicle’s model, body type, restraint system, transmission type, and engine code.
* The **ninth digit** is the check digit, which is used to detect fraudulent VINs. The number that appears in the ninth position varies and is based on a mathematical formula developed by the U.S. Department of Transportation.

**Vehicle Identifier Section**

Digits in position 10 through 17 make up the Vehicle Identifier Section (VIS).

* The **tenth digit** indicates the year, which only applies to vehicles built in or after 1981 since the VIN format was not standardized until that year.

**Data preparation**

Data transformation based per Data Understanding applied here. *model* column has some free form entries I can utilize to fill missing column values as well as VIN entries reveal some very useful information that I can fill some missing values.

**Data correction steps:**

* Drop duplicated records if any
* Remove duplicated VIN entries per price, odometer and VIN search
* Nullify VIN entries when they are invalid, containing ‘000000’
* Find out ’scion’ vehicles in model column and set them as ‘Toyota’ in manufacturer column
* Fill 4wd/2wd information from model in drive
* Fill automatic/manual from model in transmission
* Build a known lookup table per feature to fill manufacturer, type, paint color, fuel, size and condition from model
* Build a lookup table of VINs per manufacturer/model and optionally year to fill cylinders, fuel, transmission, drive, size, type, manufacturer, model and year
* Build a known lookup table per feature from model for cross reference of manufacturer/model and optionally year to fill cylinders, fuel, transmission, drive, size and type

**Data transformation steps:**

* VIN - make it binary 1/0 if exists and drop original VIN column
* Create a regions by states - 6 regions per [<https://www.infoplease.com/us/states/regions-of-the-us>](<https://www.infoplease.com/us/states/regions-of-the-us>)
* replace awd/rwd/fwd with 4wd/2wd - make it binary 1/0
* replace scion with toyota, ram with dodge in manufacturer
* replace manual/automatic with binary 0/1 in transmission
* fix missing condition from title
* Remove all unknown cylinder entries
* Fill paint color with unknown when null

**Outliers:**

* Remove all entries when price is out of this range ($1100, $249999)
* Remove all entries when odometer is out of this range (0, 9999999) except when condition is *new*.
* Remove manufacturer group has a single entry.

**Column Cleanup:**

* Remove id,city,state,VIN,size,title\_status and model columns.

**NaN Values:**

* Drop all remaining rows with NaN values.

**Convert categorical feature to numerical feature:**

* Cylinders as numbers only
* Fuel:
  + vehicles.loc[vehicles['fuel'] == 'gas', 'fuel'] = 1
  + vehicles.loc[vehicles['fuel'] == 'diesel', 'fuel'] = 2
  + vehicles.loc[vehicles['fuel'] == 'hybrid', 'fuel'] = 3
  + vehicles.loc[vehicles['fuel'] == 'electric', 'fuel'] = 4
  + vehicles.loc[vehicles['fuel'] == 'other', 'fuel'] = 5
* Condition:
  + vehicles.loc[vehicles['condition'] == 'new', 'condition'] = 1
  + vehicles.loc[vehicles['condition'] == 'like new', 'condition'] = 2
  + vehicles.loc[vehicles['condition'] == 'excellent', 'condition'] = 3
  + vehicles.loc[vehicles['condition'] == 'good', 'condition'] = 4
  + vehicles.loc[vehicles['condition'] == 'clean', 'condition'] = 5
  + vehicles.loc[vehicles['condition'] == 'fair', 'condition'] = 6
  + vehicles.loc[vehicles['condition'] == 'salvage', 'condition'] = 7
* Type:
  + vehicles.loc[vehicles['type'] == 'sedan', 'type'] = 1
  + vehicles.loc[vehicles['type'] == 'SUV', 'type'] = 2
  + vehicles.loc[vehicles['type'] == 'truck', 'type'] = 3
  + vehicles.loc[vehicles['type'] == 'pickup', 'type'] = 4
  + vehicles.loc[vehicles['type'] == 'coupe', 'type'] = 5
  + vehicles.loc[vehicles['type'] == 'hatchback', 'type'] = 6
  + vehicles.loc[vehicles['type'] == 'wagon', 'type'] = 7
  + vehicles.loc[vehicles['type'] == 'van', 'type'] = 8
  + vehicles.loc[vehicles['type'] == 'convertible', 'type'] = 9
  + vehicles.loc[vehicles['type'] == 'mini-van', 'type'] = 10
  + vehicles.loc[vehicles['type'] == 'offroad', 'type'] = 11
  + vehicles.loc[vehicles['type'] == 'bus', 'type'] = 12
  + vehicles.loc[vehicles['type'] == 'other', 'type'] = 13
* Region:
  + vehicles.loc[vehicles['region'] == 'west', 'region'] = 1
  + vehicles.loc[vehicles['region'] == 'south', 'region'] = 2
  + vehicles.loc[vehicles['region'] == 'midwest', 'region'] = 3
  + vehicles.loc[vehicles['region'] == 'southwest', 'region'] = 4
  + vehicles.loc[vehicles['region'] == 'midatlantic', 'region'] = 5
  + vehicles.loc[vehicles['region'] == 'newengland', 'region'] = 6
* Convert float to int for year, transmission, drive and vin.

**Correlation Analysis:**

There are very a few observations out of the correlation matrix:

* Priced higher when VIN number exists
* Four wheel drive vehicles priced higher
* Priced higher when fuel type is other than gas
* More the cylinders are priced higher
* Higher the odometer lower the price
* Newer the model higher the price
* Condition, manual/automatic transmission and type of vehicle do not seem play role
* Region has less influence

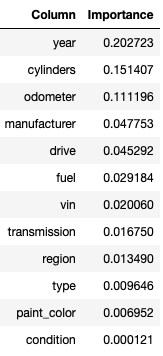
Also, some cross correlation among features per matrix:

* Newer vehicles tend to have less number of cylinders but more VIN listed in sales
* More number of cylinders if four wheel drive
* Newer models are slightly more four wheel drive
* When VIN is listed odometer has slightly less milage

**Used Car Modeling**

Before modeling I checked Variance Inflation Factor (VIF) and Permutation Importance. VIF showed no multicollinearity. Permutation Importance interestingly reported condition and region are not influencing models as below:

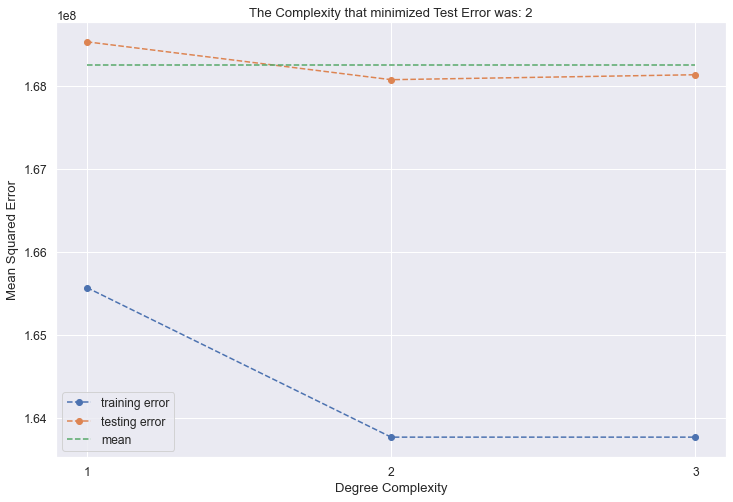
![](images/ColumnImportance.png)



I only picked up 5 features per permutation importance list: 'year', 'cylinders', 'odometer', 'manufacturer' and 'drive' to build models. Besides, I could include region and condition to categorize properly even though the attribute importance thinks differently per evaluation steps.

I ran a model complexity analysis to decide about degree complexity:

![](images/Complexity2.png)



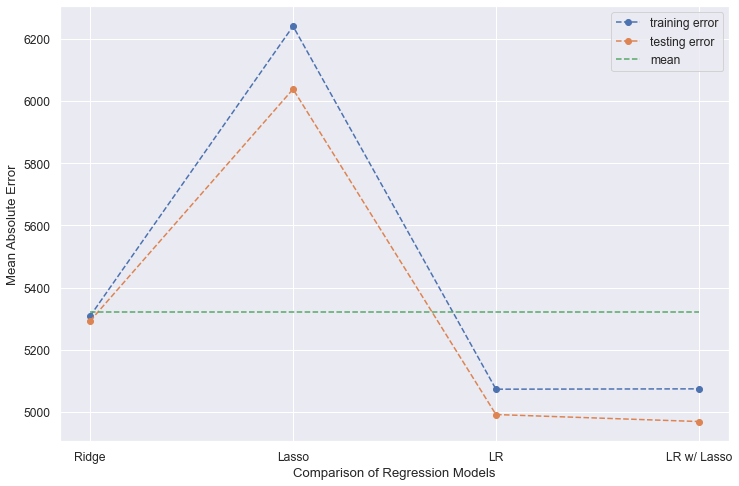
I kept it at degree=2 for all models. I built 4 models: Ridge, LASSO, Linear Regression and Linear Regression with LASSO feature selection after several evaluations.

**Evaluation**

I decided to include region and condition to categorize properly even though the attribute importance thinks differently per evaluation steps to influence models with more appropriate features. It helped slightly with models’ mean absolute errors.

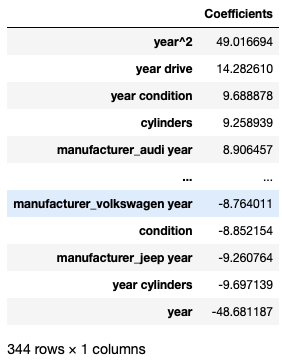
I build 4 regression models Linear Regression with LASSO feature selection outperformed others in mean absolute error:

![](images/trainingerror.png)



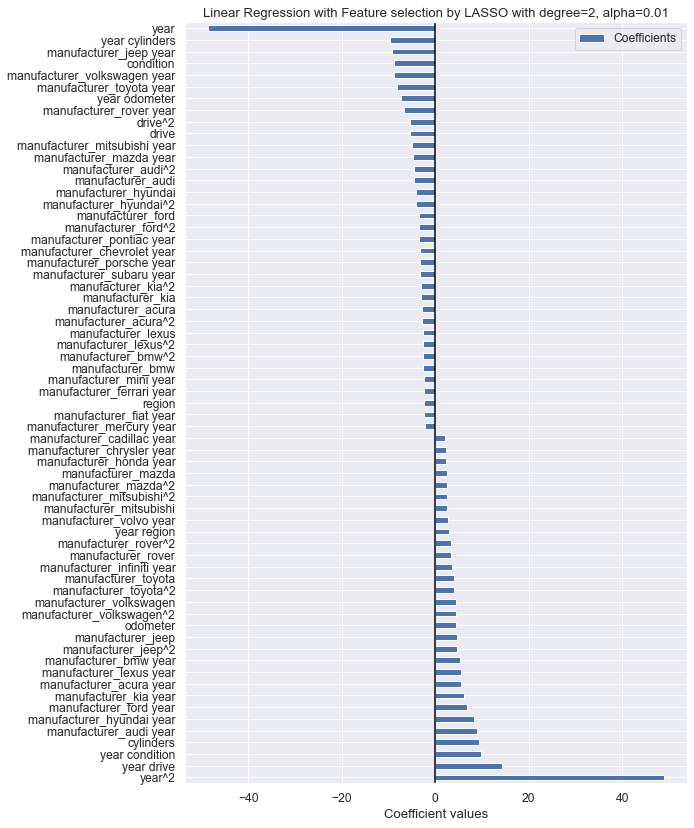
The model Linear Regression with LASSO feature selection outperformed all others, the model generated and kept only 344 feature degree=2 combinations.

![](images/Coefficients.png)



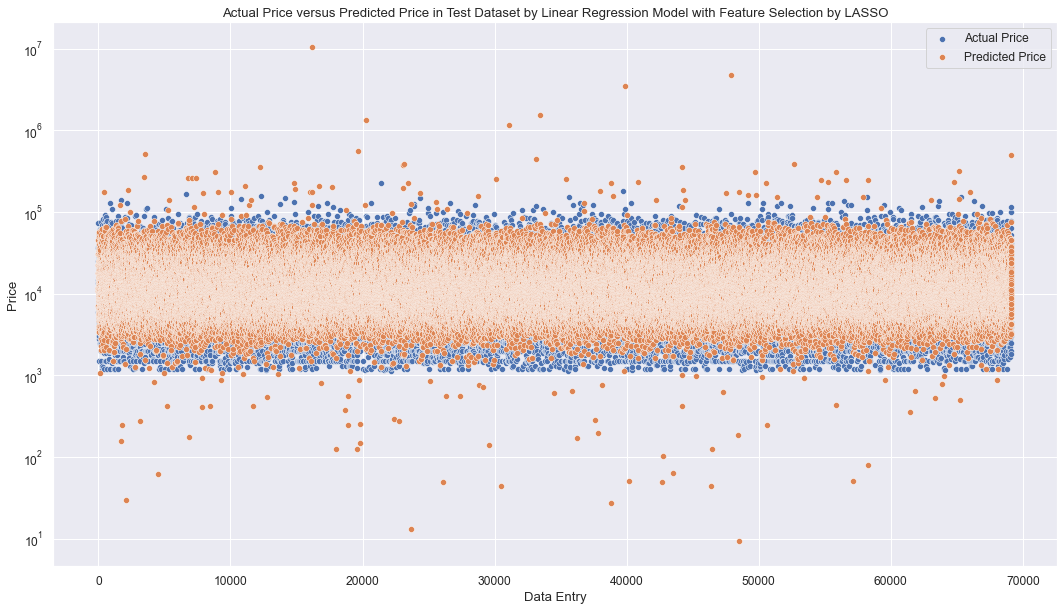
Condition is shown as a negative influencer in above and below charts:

![](images/CoefficientsValues.png)



The comparison of actual and predicted price shown in the below image for visualization:

![](images/ActualversusPredictedFeatureSelectionbyLASSO.png)



Region assignment may need to be reevaluated in later phases as a side note.

**Deployment**

Based on the research and model outcome, the influential features of a used vehicle can be narrowed down to these features:

* Year
* Cylinders
* Condition
* Drivetrain
* Region
* Odometer

Remarks about *positive* price points:

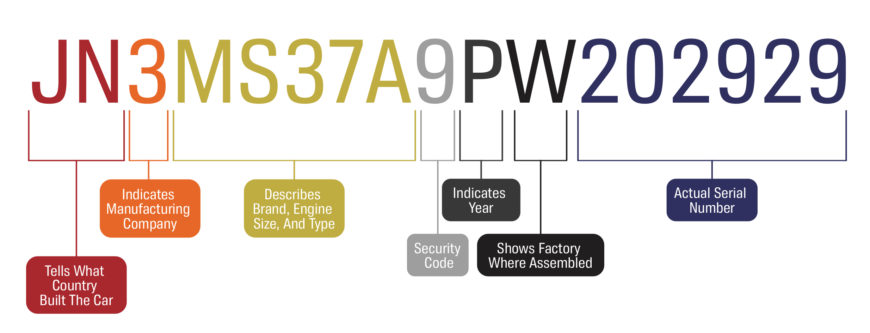
* New vehicles are high priced at the square power of year
* The product of year and drive influence higher price
* The product of year and condition influence higher price
* The product of year and region influence higher price
* Number of cylinders influence higher price
* The coefficient of year and along manufacturer influence higher price
* Manufacturers Volkswagen, Jeep, Rover, Mitsubishi, Mazda, Honda, Chrysler and Cadillac influence higher price.

Remarks about *negative* price points:

* Higher odometer reading on newer models penalize price, as condition worsens lower the price.
* Manufacturers BMW, Lexus, Acura, Kia, Ford, Hyundai and Audi tend to have lower prices.

VIN:

<https://en.wikipedia.org/wiki/Vehicle_identification_number>



**WORLD MANUFACTURER IDENTIFIER**

The first three digits of a vehicle’s VIN make up the World Manufacturer Identifier (WMI).

* The **first digit** defines the country of origin or the final point of assembly of your vehicle.
* The **second digit** indicates the manufacturer and the region where your vehicle was produced.
* The **third digit**, when coupled with the first two digits, indicates your vehicle’s type or manufacturing division.

**VEHICLE DESCRIPTION SECTION**

Digits in position 4 through 9 make up the Vehicle Description Section (VDS).

* The **fourth through eighth digits** describe your vehicle’s model, body type, restraint system, transmission type, and engine code.
* The **ninth digit** is the check digit, which is used to detect fraudulent VINs. The number that appears in the ninth position varies and is based on a mathematical formula developed by the U.S. Department of Transportation.

**VEHICLE IDENTIFIER SECTION**

Digits in position 10 through 17 make up the Vehicle Identifier Section (VIS).

* The **tenth digit** indicates the year, which only applies to vehicles built in or after 1981 since the VIN format was not standardized until that year.

 ————— o —————

**Module 12**

**Notes:**

**Module Issues:**

**Quizes:**

 ————— o —————

**Module 13**

**Notes:**

**Module Issues:**

**Quizes:**