Senior Project Estimation Scratch Sheet

RESULTS

With no solver parameter specified, the unprocessed data set looks to perform better than the processed dataset (BAD)

Different FOld Accuracies

FOld Numbers tested: pd.Series([2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 18, 20, 24, 28, 32, 40, 50, 60, 80])

**Solver not Specified in LogisticRegression Object**

Unprocessed DataSet Accuracy Results:

0     0.584034

1     0.573532

2     0.535452

3     0.565132

4     0.563309

5     0.558298

6     0.561712

7     0.573792

8     0.568000

9     0.579832

10    0.579569

11    0.575175

12    0.578002

13    0.583804

14    0.581145

15    0.591649

16    0.592390

17    0.593698

18    0.593924

19    0.602770

Processed DataSet Accuracy Results:

0     0.516019

1     0.587452

2     0.484244

3     0.479773

4     0.544117

5     0.352153

6     0.522847

7     0.529409

8     0.562473

9     0.580357

10    0.581408

11    0.569304

12    0.577178

13    0.564288

14    0.571954

15    0.576418

16    0.577930

17    0.579258

18    0.581911

19    0.588281

**Solver=’liblinear’ (Takes WAY too long)**

Unprocessed DataSet Accuracy Results:

0     0.580357

1     0.572483

2     0.533876

3     0.555675

4     0.552542

5     0.550683

6     0.573529

7     0.562507

8     0.589047

9     0.574317

10    0.580357

11    0.582284

12    0.581189

13    0.592530

14    0.591387

15    0.597426

16    0.593473

17    0.597673

18    0.597698

19    0.604466

Processed DataSet Accuracy Results:

0     0.518382

1     0.586138

2     0.480567

3     0.481874

4     0.541228

5     0.351891

6     0.521008

7     0.526785

8     0.562476

9     0.579044

10    0.581408

11    0.567991

12    0.576660

13    0.564552

14    0.571691

15    0.575893

16    0.576615

17    0.579778

18    0.581651

19    0.587217

**Solver=’sag’**

Unprocessed DataSet

0     0.549632

1     0.504712

2     0.529674

3     0.523100

4     0.519956

5     0.509191

6     0.543067

7     0.519985

8     0.546785

9     0.528887

10    0.536502

11    0.535494

12    0.533659

13    0.539154

14    0.541492

15    0.542542

16    0.542569

17    0.541729

18    0.540298

19    0.540896

Processed Dataset

0     0.515756

1     0.587452

2     0.483193

3     0.479773

4     0.543854

5     0.352153

6     0.522321

7     0.528621

8     0.562474

9     0.579569

10    0.581670

11    0.569042

12    0.576655

13    0.564552

14    0.571954

15    0.576155

16    0.577407

17    0.578995

18    0.581647

19    0.588010

**Solver=’saga’ Saga taken way too long as well**

Unprocessed DataSet

0     0.542017

1     0.514957

2     0.537027

3     0.525983

4     0.532813

5     0.529149

6     0.544118

7     0.534420

8     0.559114

9     0.537027

10    0.548845

11    0.541516

12    0.545200

13    0.545445

14    0.550158

15    0.548582

16    0.547552

17    0.549087

18    0.546582

19    0.549535

Processed Dataset

0     0.517595

1     0.586664

2     0.480567

3     0.481349

4     0.542541

5     0.352153

6     0.521008

7     0.527834

8     0.561950

9     0.579832

10    0.580882

11    0.568779

12    0.576133

13    0.564817

14    0.571691

15    0.575893

16    0.576886

17    0.579262

18    0.581395

19    0.587749

**Using saying 2nd Normalization Function**

We only test with the ‘sag’ solver as it takes shortest time to run

Unprocessed DataSet

0     0.549632

1     0.504712

2     0.529937

3     0.523100

4     0.521269

5     0.509716

6     0.543067

7     0.519196

8     0.545471

9     0.528624

10    0.536765

11    0.534703

12    0.533662

13    0.539154

14    0.541492

15    0.542279

16    0.542045

17    0.541722

18    0.540034

19    0.541944

Processed Dataset  
0     0.514443

1     0.587714

2     0.484244

3     0.479773

4     0.543854

5     0.351891

6     0.522584

7     0.529146

8     0.562736

9     0.580357

10    0.581408

11    0.569042

12    0.577440

13    0.564288

14    0.571954

15    0.576155

16    0.577930

17    0.579258

18    0.582172

19    0.588281

**Using Standardization Function**

We only test with the ‘sag’ solver as it takes shortest time to run

Unprocessed DataSet

0     0.549632

1     0.504712

2     0.529937

3     0.523100

4     0.521269

5     0.509716

6     0.543067

7     0.519196

8     0.545471

9     0.528624

10    0.536765

11    0.534703

12    0.533662

13    0.539154

14    0.541492

15    0.542279

16    0.542045

17    0.541722

18    0.540034

19    0.541944

Processed Dataset  
0     0.514443

1     0.587714

2     0.484244

3     0.479773

4     0.543854

5     0.351891

6     0.522584

7     0.529146

8     0.562736

9     0.580357

10    0.581408

11    0.569042

12    0.577440

13    0.564288

14    0.571954

15    0.576155

16    0.577930

17    0.579258

18    0.582172

19    0.588281

REDO TESTS!!!!

FAILED TO RECOGNIZE DEEP VS SHALLOW COPY (Solver=’sag’, normalize 1st formula)

Unprocessed DataSet

0     0.549632

1     0.505237

2     0.530200

3     0.523100

4     0.520481

5     0.509716

6     0.542542

7     0.518934

8     0.547310

9     0.531775

10    0.538866

11    0.536544

12    0.532348

13    0.539154

14    0.541492

15    0.542279

16    0.542045

17    0.540677

18    0.540034

19    0.541151

Processed DataSet

0     0.515756

1     0.587452

2     0.483456

3     0.479773

4     0.543854

5     0.352153

6     0.522321

7     0.528359

8     0.562474

9     0.579569

10    0.581670

11    0.569042

12    0.576655

13    0.564552

14    0.571954

15    0.576155

16    0.577407

17    0.579258

18    0.581647

19    0.588010

Normalize 2nd Formula

Unprocessed DataSet

0     0.549632

1     0.504975

2     0.530200

3     0.522312

4     0.521269

5     0.509979

6     0.543067

7     0.518672

8     0.547835

9     0.528099

10    0.536765

11    0.536806

12    0.532348

13    0.538628

14    0.541492

15    0.542805

16    0.542045

17    0.542252

18    0.539769

19    0.541938

Processed DataSet

0     0.514706

1     0.587714

2     0.484244

3     0.479773

4     0.543854

5     0.351891

6     0.522584

7     0.529146

8     0.562736

9     0.580357

10    0.581408

11    0.569042

12    0.577440

13    0.564288

14    0.571954

15    0.576155

16    0.577930

17    0.579258

18    0.582172

19    0.588281

**Standardize: One of the attributes has standard deviation of 0 (operatingProfitMargin), so dividing over 0 causes problems. Every value is one except for missing values. Delete column? If it isnt giving me any info, then go ahead.**

After Feature Selection using Multicollinearity and SelectKFeatures(50):

**Standardization**

After Preprocessing:

0     0.542805

1     0.559082

2     0.525210

3     0.504998

4     0.597153

5     0.446429

6     0.557248

7     0.581649

8     0.616081

9     0.605305

10    0.611345

11    0.611413

12    0.610602

13    0.614171

14    0.616334

15    0.617910

16    0.619529

17    0.623985

18    0.620581

19    0.622285

After Preprocessing (Solver=Saga):

0     0.543330

1     0.557770

2     0.521271

3     0.499223

4     0.590854

5     0.442752

6     0.552784

7     0.582961

8     0.615556

9     0.604254

10    0.610294

11    0.610362

12    0.611126

13    0.613645

14    0.616334

15    0.617910

16    0.619526

17    0.623195

18    0.620317

19    0.622545

After Preprocessing (Solver=Liblinear)

0     0.542805

1     0.557769

2     0.521271

3     0.500536

4     0.591904

5     0.443540

6     0.553309

7     0.582437

8     0.617129

9     0.605830

10    0.611082

11    0.612199

12    0.612966

13    0.614957

14    0.617384

15    0.618172

16    0.620047

17    0.624508

18    0.621883

19    0.623327

**We will stick to the SAGA scorer for now henceforth**

**Next Step:** Do SelectKBest before Multicollinearity test

0     0.546744

1     0.559610

2     0.520221

3     0.498695

4     0.599516

5     0.447742

6     0.558298

7     0.587687

8     0.618449

9     0.614496

10    0.616334

11    0.617461

12    0.619273

13    0.617319

14    0.619223

15    0.619223

16    0.623999

17    0.627163

18    0.625343

19    0.624928

No significant improvement

However, Correlation Matrix looks wildly different:

SelectKBestAfter:

A picture containing diagram

Description automatically generated

SelectKBestBefore:

Text

Description automatically generated

We did some hyperparameter testing on selectKBest Features using 50-Fold Cross Validation and here are the plotted results:

Chart, line chart

Description automatically generated

50-60 features score the highest but the variance between scores is quite insignificant

**More advanced Preprocessing**

We will take the following steps before running hyperparameter testing:

1. Removing Variables with high percent missing values
2. Removing variables with very low variation/ zero variation
3. Pairwise correlation

* If two variables are highly correlated, remove one, keep the one with a **higher correlation coefficient** with the target variable

1. If variable has low correlation with target variable, drop
2. Forward Selection, Backward Elimination

* In preprocess method, before imputation of missing values, we remove variables with more than 30 percent missing values (1)
* Still in preprocess method, after imputation of missing values, we remove variables with variation between -0.1 and 0.1 (2)
* We also remove variables with correlations with the target variable between -0.01 and 0.01

***Combination of above 3 steps in preprocess method offer us no improvement whatsoever***

=> We select a value for selectKBest (60) using 50-Fold Cross Validation, and experiment different combinations of the above preprocess steps

* Using only (1) and (2), we get 0 difference in final result (0.6190464798359535)
* Using only (1) and (3), we get a little improvement in score (0.628732057416268), which is actually more than highest score doing hyperparameter testing of select K best
* Using only (2) and (3), we get no improvement on final result (0.6174641148325358)
* Using only (1), we get a slight improvement (0.628732057416268) [Same score as the second???]
* Using only (2), we get 0.6174641148325358
* Using only (3), we get some improvement, but not as much ans using only (1) (0.6271496924128503)

**We decide to use both 1 and 3, as this gives us the most optimal score AND efficiently reduces dimensionality**

\*\***WE USE DIFFERENT ENCODING TECHNIQUES (HASH AND BINARY)**

\*\***WE DONT USE SELECTKBEST AND NARROW DOWN FEATURES WITH ADVANCED PREPROCESSING STEPS**

**\*\*WE ALSO INTEND TO DO PCA (If there is enough time)**

**Feature Engineering Techniques we have already employed:**

One hot encoding, Regularization (Standardization), Missing Value Imputation

**Feature Engineering Techniques we hope to employ:**

*Log transformation, Creating Polynomial Features*

**Log transformation**

You can use log transformation to center data if it is skewed. When interpreting the results, you have to remember to take the exponent.

You can transform the data using [NumPy’s](https://numpy.org/doc/stable/reference/generated/numpy.log1p.html) `log1p` method. This implementation adds one to each number before taking the logarithm and prevents taking the log of zero. Taking the log of zero results in error.

* We use log1p for variables with positive skewness, and we use exponential logarithm for variables with negative skewness
* Log Transformations
* Asghar, Muhammad. (2015). Re: How can I log transform a series with both positive and negative values?. Retrieved from: https://www.researchgate.net/post/How-can-I-log-transform-a-series-with-both-positive-and-negative-values/54ed5e50f15bc72b228b45a1/citation/download.
* Add the Maximum negative value in the whole data series (Value + Maximum Negative value + 1)
* In this way the minimum value of the data would be 1
* **HOLY \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***
* **We get a score of 0.9726862611073137(!!!!!!!!!!!!!!!!!!!!!!!!!!)**
* Problem: InputCols does not seem to have any columns when printed, must fix and see results
* Problem fixed, RESULTS REMAIN THE SAME
* Now we compare with CHeckpoint 1 using HyperParameter testing

HyperParameter Testing Before Advanced Preprocessing

Chart, line chart

Description automatically generated

HyperParameter Testing After Advanced Preprocessing

Chart, line chart

Description automatically generated

Feature Engineering Techniques include log transformation and hash encodiong

* We show results of hyperparameter testing after replacing one hot encoding with hash
* Higher result when I used Hash Encoding for select 60 Best, 50 CV: 0.9752973342447026

**BEGIN K NEAREST NEIGHBOUR MODEL**

* Coding is not hard but it takes LOADS of time
* Selected 60 Best dataset, 1NN, accuracy:  0.6223274094326725
* Selected 60 Best dataset, 5NN, accuracy:  0.6635645933014357
* Selected 60 Best dataset, 10NN, accuracy:  0.665170881749829
* Selected 60 Best dataset, 20NN, accuracy:  0.6683663704716335
* Selected 60 Best dataset, 30NN, accuracy:  0.6685782638414216
* Selected 60 Best dataset, 50NN, accuracy:  0.6678161312371838
* Selected 60 Best dataset, 70NN, accuracy:  0.6680416951469581
* Selected 60 Best dataset, 100NN, accuracy:  0.6643677375256322
* Selected 60 Best dataset, 150NN, accuracy:  0.6630895420369104

Gradient Boosting Machines

* Gradient boosting is provided via the [GradientBoostingRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html) and [GradientBoostingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html) classes.
* We will evaluate the model using repeated stratified k-fold cross-validation, with three repeats and 10 folds. We will report the mean and standard deviation of the accuracy of the model across all repeats and folds.
* Without log transformations and using one hot encoding with standardization, we obtain the following results selecting best 10 features:
* For 5.000 estimators, Mean Accuracy: 0.598 (0.030)
* For 10.000 estimators, Mean Accuracy: 0.616 (0.036)
* For 20.000 estimators, Mean Accuracy: 0.619 (0.040)
* For 30.000 estimators, Mean Accuracy: 0.626 (0.043)
* For 40.000 estimators, Mean Accuracy: 0.623 (0.044)
* For 50.000 estimators, Mean Accuracy: 0.620 (0.046)
* For 70.000 estimators, Mean Accuracy: 0.616 (0.051)
* For 100.000 estimators, Mean Accuracy: 0.617 (0.050)
* For 200.000 estimators, Mean Accuracy: 0.616 (0.052)
* For 500.000 estimators, Mean Accuracy: 0.610 (0.053)
* For 1000.000 estimators, Mean Accuracy: 0.600 (0.051)
* For 5000.000 estimators, Mean Accuracy: 0.581 (0.051)
* Same, selecting 5 best features:
* For 5.000 estimators, Mean Accuracy: 0.615 (0.036)
* For 10.000 estimators, Mean Accuracy: 0.615 (0.036)
* For 20.000 estimators, Mean Accuracy: 0.615 (0.038)
* For 30.000 estimators, Mean Accuracy: 0.609 (0.039)
* For 40.000 estimators, Mean Accuracy: 0.611 (0.043)
* For 50.000 estimators, Mean Accuracy: 0.610 (0.043)
* For 70.000 estimators, Mean Accuracy: 0.607 (0.046)
* For 100.000 estimators, Mean Accuracy: 0.615 (0.051)
* For 200.000 estimators, Mean Accuracy: 0.611 (0.052)
* For 500.000 estimators, Mean Accuracy: 0.595 (0.054)
* For 1000.000 estimators, Mean Accuracy: 0.582 (0.058)
* For 5000.000 estimators, Mean Accuracy: 0.574 (0.057)
* Same, selecting 20 best features:
* For 5.000 estimators, Mean Accuracy: 0.622 (0.035)
* For 10.000 estimators, Mean Accuracy: 0.625 (0.037)
* For 20.000 estimators, Mean Accuracy: 0.634 (0.042)
* For 30.000 estimators, Mean Accuracy: 0.640 (0.046)
* For 40.000 estimators, Mean Accuracy: 0.642 (0.046)
* For 50.000 estimators, Mean Accuracy: 0.640 (0.046)
* For 70.000 estimators, Mean Accuracy: 0.642 (0.048)
* For 100.000 estimators, Mean Accuracy: 0.640 (0.049)
* For 200.000 estimators, Mean Accuracy: 0.626 (0.045)
* For 500.000 estimators, Mean Accuracy: 0.613 (0.050)
* For 1000.000 estimators, Mean Accuracy: 0.608 (0.050)
* Same, selecting 35 best features:
* For 5.000 estimators, Mean Accuracy: 0.620 (0.031)
* For 10.000 estimators, Mean Accuracy: 0.624 (0.035)
* For 20.000 estimators, Mean Accuracy: 0.633 (0.043)
* For 30.000 estimators, Mean Accuracy: 0.641 (0.042)
* For 40.000 estimators, Mean Accuracy: 0.639 (0.044)
* For 50.000 estimators, Mean Accuracy: 0.642 (0.045)
* For 70.000 estimators, Mean Accuracy: 0.641 (0.048)
* For 100.000 estimators, Mean Accuracy: 0.642 (0.049)
* For 200.000 estimators, Mean Accuracy: 0.639 (0.051)
* For 500.000 estimators, Mean Accuracy: 0.628 (0.056)

**AFTER REALIZATION OF MASSIVE DATA LEAKAGE, WE RETEST:**

After Log Transformation and Hash Encoding:

Using 50 CV Fold testing:

Logistic Regression, selectKBest Hyperparameter testing:

Chart, line chart

Description automatically generated

KNN Algorithm

Selected 60 Best dataset, 1NN, accuracy:  0.5595146958304853

Selected 60 Best dataset, 5NN, accuracy:  0.5892276144907727

Selected 60 Best dataset, 10NN, accuracy:  0.5990259740259741

Selected 60 Best dataset, 20NN, accuracy:  0.5961380724538617

Selected 60 Best dataset, 50NN, accuracy:  0.5913807245386191

Selected 60 Best dataset, 30NN, accuracy:  0.5990293916609706

Selected 60 Best dataset, 70NN, accuracy:  0.5947915242652084

Selected 60 Best dataset, 100NN, accuracy:  0.589542036910458

Selected 60 Best dataset, 150NN, accuracy:  0.5929596719070402

Gradient Boosting:

For 5.000 estimators, Mean Accuracy: 0.620 (0.035)

For 10.000 estimators, Mean Accuracy: 0.630 (0.036)

For 20.000 estimators, Mean Accuracy: 0.634 (0.036)

For 30.000 estimators, Mean Accuracy: 0.639 (0.038)

For 40.000 estimators, Mean Accuracy: 0.640 (0.040)

For 50.000 estimators, Mean Accuracy: 0.642 (0.043)

For 70.000 estimators, Mean Accuracy: 0.641 (0.044)

For 100.000 estimators, Mean Accuracy: 0.640 (0.046)

For 200.000 estimators, Mean Accuracy: 0.640 (0.051)

For 500.000 estimators, Mean Accuracy: 0.632 (0.052)

For 1000.000 estimators, Mean Accuracy: 0.626 (0.053)

For 5000.000 estimators, Mean Accuracy: 0.631 (0.050)

**With Hash Encoding but no log transformation**

Using 50 CV Fold testing:

Logistic Regression, selectKBest Hyperparameter testing:

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

KNN HyperParameter Testing

Selected 60 Best dataset, 1NN, accuracy:  0.5514832535885167

Selected 60 Best dataset, 5NN, accuracy:  0.5680177717019822

Selected 60 Best dataset, 10NN, accuracy:  0.5905980861244019

Selected 60 Best dataset, 20NN, accuracy:  0.5884996582365004

Selected 60 Best dataset, 50NN, accuracy:  0.5895317840054681

Selected 60 Best dataset, 30NN, accuracy:  0.5869480519480518

Selected 60 Best dataset, 70NN, accuracy:  0.5926930963773068

Selected 60 Best dataset, 100NN, accuracy:  0.5961175666438824

Selected 60 Best dataset, 150NN, accuracy:  0.585857826384142

Using 120 best features

Gradient Boosting:

For 5.000 estimators, Mean Accuracy: 0.618 (0.037)

For 10.000 estimators, Mean Accuracy: 0.620 (0.041)

For 20.000 estimators, Mean Accuracy: 0.630 (0.044)

For 30.000 estimators, Mean Accuracy: 0.635 (0.046)

For 40.000 estimators, Mean Accuracy: 0.642 (0.046)

For 50.000 estimators, Mean Accuracy: 0.642 (0.047)

For 70.000 estimators, Mean Accuracy: 0.642 (0.047)

For 100.000 estimators, Mean Accuracy: 0.644 (0.046)

For 200.000 estimators, Mean Accuracy: 0.638 (0.050)

For 500.000 estimators, Mean Accuracy: 0.627 (0.051)

For 1000.000 estimators, Mean Accuracy: 0.624 (0.056)

Trying out effects of advanced preprocessing 1,2 and 3, with Hash encoding, with log transformation (All previous testing has just been 1 and 3)

With 1,2, and 3:

Logistic regression hyperparameter testing:

For selectK = 40, 50, 60, 80, 100, 120, 130, 140, 149

[0.604295967190704, 0.5979801777170198, 0.6006220095693781, 0.5961346548188653, 0.6076657552973342, 0.5966165413533835, 0.5968762816131238, 0.5963499658236501, 0.5895625427204375]

For selectK = 3,4,5,6,7,8,10,12,14,16,18,20,24,28,32

[0.6024846206425154, 0.6024846206425154, 0.6001127819548873, 0.6077136021872864, 0.6140498974709501, 0.6069583048530416, 0.6066883116883116, 0.60823991797676, 0.6087696514012303, 0.6108817498291182, 0.6050580997949419, 0.601907040328093, 0.6140157211209842, 0.6124367737525632, 0.612717019822283]

Chart, line chart

Description automatically generated

(***Forgot to copy knn and gradient boosting results-Redo; Also, change selectKBest for kNN hyperparameter testing to select60Best)***

With 2, and 3:

Logistic Regression hyperparameter testing

For selectK:3,4,5,6,7,8,10,12,14,16,18,20,24,28,32,40, 50, 60, 80, 100, 120, 130, 140, 150

[0.6024846206425154, 0.6024846206425154, 0.6001127819548873, 0.6077136021872864, 0.6140498974709501, 0.6069583048530416, 0.6066883116883116, 0.60823991797676, 0.6087696514012303, 0.6108817498291182, 0.6050580997949419, 0.601907040328093, 0.6140157211209842, 0.6124367737525632, 0.612717019822283, 0.604295967190704, 0.5979801777170198, 0.6011414900888584, 0.6045522898154477, 0.6105502392344497, 0.6008168147641831, 0.597922077922078, 0.6010765550239234, 0.5992344497607656]

(***Plot later)***

Selected 60 Best dataset, 1NN, accuracy:  0.5440704032809296

Selected 60 Best dataset, 5NN, accuracy:  0.5725051264524947

Selected 60 Best dataset, 10NN, accuracy:  0.5862269309637731

Selected 60 Best dataset, 20NN, accuracy:  0.5848598769651402

Selected 60 Best dataset, 50NN, accuracy:  0.595881749829118

Selected 60 Best dataset, 30NN, accuracy:  0.5856596035543404

Selected 60 Best dataset, 70NN, accuracy:  0.5887764866712233

Selected 60 Best dataset, 100NN, accuracy:  0.585905673274094

Selected 60 Best dataset, 150NN, accuracy:  0.5875153793574847

For 5.000 estimators, Mean Accuracy: 0.617 (0.034)

For 10.000 estimators, Mean Accuracy: 0.628 (0.036)

For 20.000 estimators, Mean Accuracy: 0.631 (0.036)

For 30.000 estimators, Mean Accuracy: 0.635 (0.043)

For 40.000 estimators, Mean Accuracy: 0.639 (0.043)

For 50.000 estimators, Mean Accuracy: 0.641 (0.043)

For 70.000 estimators, Mean Accuracy: 0.642 (0.047)

For 100.000 estimators, Mean Accuracy: 0.638 (0.048)

For 200.000 estimators, Mean Accuracy: 0.636 (0.050)

For 500.000 estimators, Mean Accuracy: 0.629 (0.050)

For 1000.000 estimators, Mean Accuracy: 0.633 (0.050)

**From hyperparameter testing, we observe that 50 estimators is the most optimal hyperparameter value for number of estimators. We will do grid Search to find optima hyperparameters for gradient boosters**

**Using Normalize, removeMissing, removeLowCorrWTV, log transformations, Hash encoding (8 components),**

**Grid Search Results:**

**length of inputCols:  156**

**Best Score: 0.6430690362269309**

**Best Hyperparameters: {'learning\_rate': 0.05, 'loss': 'exponential', 'max\_depth': 2, 'n\_estimators': 50, 'subsample': 0.5}**

**Using Grid Search Results in selectKBest features (without multicollinearity filter):**

**0.605828197230931, 0.6063522118867862, 0.608715566270379, 0.6205311885291606, 0.6210577324352197, 0.6193948653921544, 0.6217574150143189, 0.624734802367838, 0.6257849010660792, 0.6267470508367794, 0.6249124247116942, 0.6256083134154881, 0.6270979268195943, 0.6352357893502472, 0.6319109749914636, 0.6368157659661794, 0.6382159358858068, 0.6337520392079768, 0.6418065514478233, 0.64346861372946, 0.6438193747463564, 0.6414572849878654, 0.6416322631155995, 0.6433813545974296**

**Chart, line chart

Description automatically generated**

**With multicollinearity filter:**

**0.6049531916263425, 0.6072277923209664, 0.6117800977900103, 0.6189566153113909, 0.6204465736132524, 0.6225445866572854, 0.6235975595034852, 0.6242971271166662, 0.6282371241045592, 0.6271856458152981, 0.6261371566399141, 0.6282360894112939, 0.6276233210664697, 0.637252406524086, 0.6341010757360981, 0.6361117146821823, 0.6362017329962533, 0.6354162858420621, 0.6422425022102198, 0.6439072087079787, 0.642507613617943, 0.644870623103781, 0.6390057517448953, 0.6411948177962643**

**Chart, line chart

Description automatically generated**

**Performance seems to stagnate at around 0.64**

**Chart, line chart

Description automatically generated**

**BEFORE GRID SEARCH FOR RANDOM FORESTS**

**Normalize+remove missing (210 features) gives us**

**length of inputCols:  210**

**mean is 0.653**

**Normalize2: 0.652**

**CV Testing**

**length of inputCols:  210**

**For cv= 5.000, mean is 0.652**

**For cv= 10.000, mean is 0.655**

**For cv= 20.000, mean is 0.657**

**For cv= 30.000, mean is 0.653**

**For cv= 40.000, mean is 0.652**

**For cv= 50.000, mean is 0.652**

**For cv= 70.000, mean is 0.653**

**For cv= 100.000, mean is 0.654**

**For cv= 200.000, mean is 0.654**

**Will use CV=20 from now**

**Testing Advancing Preprocessing steps**

**Just removeLowVar()(2):**

**length of inputCols:  200**

**mean is 0.649**

**Just (3)(0.001):**

**length of inputCols:  208**

**mean is 0.652**

**Just (3)(0.01):**

**length of inputCols:  156**

**mean is 0.652**

**Just (4)(Log transformations):**

**length of inputCols:  210**

**mean is 0.653**

**(2) and (3)(0.001):**

**length of inputCols:  199**

**mean is 0.654**

**(2) and (3)(0.01):**

**length of inputCols:  149**

**mean is 0.649**

**(2) and (4):**

**length of inputCols:  200**

**mean is 0.654**

**(3)(0.01) and (4):**

**length of inputCols:  156**

**mean is 0.649**

**(3)(0.001) and (4):**

**length of inputCols:  208**

**mean is 0.655**

**Will use this for selectKBest and gridSearch**

**SelectKBestTesting (without multicollinearity filter):**

**0.543325525856526, 0.5710843207495179, 0.5740608064664278, 0.581243685129053, 0.5862418480756866, 0.5870088178561589, 0.6037485073941399, 0.6092445118030678, 0.6132823551024156, 0.6206374575181408, 0.6233503260769725, 0.6277266464590795, 0.6362193441719483, 0.6394557729402038, 0.6381418205198862, 0.6392762009736382, 0.64970147882796, 0.6481266648296132, 0.6516221181225315, 0.6499650959860385, 0.6495168549646366, 0.6545136401212456, 0.6468898686506841, 0.6550344447506201, 0.6529360705428492, 0.6518852760172682, 0.6534582529622486, 0.6510126756682283**

**Chart, line chart

Description automatically generated**

**With multicollinearity**

**0.5424543032975109, 0.5731808579039221, 0.5739749242215487, 0.5744190318728759, 0.5886773215761918, 0.591830623679618, 0.6027725727932395, 0.6030334343712686, 0.6102925507485991, 0.6172177826765867, 0.6191319922843758, 0.6206278129879672, 0.6342812528703959, 0.6409442454303297, 0.635857903922109, 0.6463630935978689, 0.6501304307890143, 0.6499572885092313, 0.6497896573895472, 0.6503954257371177, 0.6516193625424819, 0.6570492330302196, 0.6536245062919076, 0.6544254615596584, 0.6512648112427666, 0.6548585468907872, 0.6572269679434188, 0.6517061633140441**

**Chart, line chart

Description automatically generated**

**Best scores for Random Forest Grid Search**

**0.5544516395701297, 0.5823647469458989, 0.5880738495453294, 0.5886851290529989, 0.6026017268301642, 0.6039138421971157, 0.6116120143290161, 0.6136157802884175, 0.6140566730963534, 0.6230830348121613, 0.627456140350877, 0.6333204739597685, 0.6369977955359603, 0.6440920363736566, 0.6441774593551943, 0.6463718196013595, 0.6545949297327086, 0.6548608432074953, 0.6514439239459905, 0.6537209515936436, 0.6556507761550473, 0.6538118857352806, 0.6593221273077982, 0.6563419674841552, 0.6496955084045192, 0.6576540828511066, 0.6599366216588594, 0.6548626802608614**

**Chart, line chart

Description automatically generated**

Subsample gradient booster hyperparameter testing **Using 50 estimators**

For 50 estimators, and 0.100 percent of subsample, Mean Accuracy: 0.623 (0.049)

For 50 estimators, and 0.200 percent of subsample, Mean Accuracy: 0.632 (0.050)

For 50 estimators, and 0.300 percent of subsample, Mean Accuracy: 0.639 (0.048)

For 50 estimators, and 0.400 percent of subsample, Mean Accuracy: 0.635 (0.048)

For 50 estimators, and 0.500 percent of subsample, Mean Accuracy: 0.644 (0.042)

For 50 estimators, and 0.600 percent of subsample, Mean Accuracy: 0.641 (0.044)

For 50 estimators, and 0.700 percent of subsample, Mean Accuracy: 0.642 (0.047)

For 50 estimators, and 0.800 percent of subsample, Mean Accuracy: 0.643 (0.045)

For 50 estimators, and 0.900 percent of subsample, Mean Accuracy: 0.641 (0.044)

For 50 estimators, and 1.000 percent of subsample, Mean Accuracy: 0.642 (0.043)

Hyperparameter testing for learning rate

50 estimators, 0.5 of subsample, and 0.0001 learning rate, Mean Accuracy: 0.571 (0.006)

50 estimators, 0.5 of subsample, and 0.001 learning rate, Mean Accuracy: 0.571 (0.006)

50 estimators, 0.5 of subsample, and 0.010 learning rate, Mean Accuracy: 0.618 (0.028)

50 estimators, 0.5 of subsample, and 0.100 learning rate, Mean Accuracy: 0.640 (0.045)

50 estimators, 0.5 of subsample, and 1.000 learning rate, Mean Accuracy: 0.585 (0.055)

50 estimators, 0.5 of subsample, and 0.050 learning rate, Mean Accuracy: 0.641 (0.041)

50 estimators, 0.5 of subsample, and 0.100 learning rate, Mean Accuracy: 0.639 (0.043)

50 estimators, 0.5 of subsample, and 0.200 learning rate, Mean Accuracy: 0.630 (0.047)

50 estimators, 0.5 of subsample, and 0.300 learning rate, Mean Accuracy: 0.622 (0.050)

50 estimators, 0.5 of subsample, and 0.400 learning rate, Mean Accuracy: 0.611 (0.053)

50 estimators, 0.5 of subsample, and 0.500 learning rate, Mean Accuracy: 0.603 (0.056)

50 estimators, 0.5 of subsample, and 1.000 learning rate, Mean Accuracy: 0.578 (0.056)

Hyperparameter testing for tree depth (with log transformation and hash encoding):

50 estimators, 0.5 of subsample, 0.05 learning rate, and 1.000 max tree depth, Mean Accuracy: 0.623 (0.034)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 2.000 max tree depth, Mean Accuracy: 0.633 (0.038)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 3.000 max tree depth, Mean Accuracy: 0.636 (0.040)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 4.000 max tree depth, Mean Accuracy: 0.642 (0.039)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 5.000 max tree depth, Mean Accuracy: 0.642 (0.046)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 6.000 max tree depth, Mean Accuracy: 0.641 (0.049)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 7.000 max tree depth, Mean Accuracy: 0.642 (0.047)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 8.000 max tree depth, Mean Accuracy: 0.635 (0.048)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 9.000 max tree depth, Mean Accuracy: 0.637 (0.051)

50 estimators, 0.5 of subsample, 0.05 learning rate, and 10.000 max tree depth, Mean Accuracy: 0.640 (0.050)

**XGBoost Classifier Initial Results(With Hash encoding, standardization, and Log transformation. Also 1 and 3 advanced preprocessing) :**

Mean Accuracy: 0.626 (0.053)

Without log transformation:

Mean Accuracy: 0.625 (0.061)

Without standardization but with log transformation

Mean Accuracy: 0.630 (0.049)

Makes you question, what if we don't use standardization for precious experiments???

Gradient Boosting HyperParameter testing of number of estimators **without Standardization but with log transformation:**

For 5.000 estimators, Mean Accuracy: 0.620 (0.035)

For 10.000 estimators, Mean Accuracy: 0.630 (0.036)

For 20.000 estimators, Mean Accuracy: 0.634 (0.036)

For 30.000 estimators, Mean Accuracy: 0.640 (0.038)

For 40.000 estimators, Mean Accuracy: 0.640 (0.040)

For 50.000 estimators, Mean Accuracy: 0.641 (0.043)

For 70.000 estimators, Mean Accuracy: 0.640 (0.043)

For 100.000 estimators, Mean Accuracy: 0.642 (0.046)

For 200.000 estimators, Mean Accuracy: 0.637 (0.048)

For 500.000 estimators, Mean Accuracy: 0.631 (0.050)

For 1000.000 estimators, Mean Accuracy: 0.630 (0.051)

Not Much Improvement from previous experiments

**XGBoost with normalization**

Mean Accuracy: 0.632 (0.056)

**lightGBM with normalization:**

Mean Accuracy: 0.634 (0.048)

***Try decision trees***

***Random Forests with log transformation and normalization (All with Isolation Forest):***

***Mean Accuracy: 0.645 (0.044)***

***Maybe do hyper parameter testing on this***

*Testing different number of estimators:*

*5,10,20,30,40,50,70,100,200,500,1000*

Mean Accuracy: 0.602 (0.054)

Mean Accuracy: 0.620 (0.048)

Mean Accuracy: 0.625 (0.053)

Mean Accuracy: 0.635 (0.047)

Mean Accuracy: 0.638 (0.053)

Mean Accuracy: 0.638 (0.053)

Mean Accuracy: 0.640 (0.051)

Mean Accuracy: 0.639 (0.050)

Mean Accuracy: 0.647 (0.054)

Mean Accuracy: 0.643 (0.052)

Testing with only log transformation and no normalization

Mean Accuracy: 0.594 (0.052)

Mean Accuracy: 0.624 (0.045)

Mean Accuracy: 0.632 (0.046)

Mean Accuracy: 0.639 (0.049)

Mean Accuracy: 0.643 (0.049)

Mean Accuracy: 0.643 (0.049)

Mean Accuracy: 0.650 (0.050)

Mean Accuracy: 0.648 (0.047)

Mean Accuracy: 0.650 (0.046)

Mean Accuracy: 0.649 (0.045)

Mean Accuracy: 0.651 (0.044)

Testing with standardization and log transformation:

For 5.000 estimators, Mean Accuracy: 0.594 (0.059)

For 10.000 estimators, Mean Accuracy: 0.622 (0.052)

For 20.000 estimators, Mean Accuracy: 0.629 (0.050)

For 30.000 estimators, Mean Accuracy: 0.636 (0.051)

For 40.000 estimators, Mean Accuracy: 0.643 (0.048)

For 50.000 estimators, Mean Accuracy: 0.642 (0.049)

For 70.000 estimators, Mean Accuracy: 0.643 (0.051)

For 100.000 estimators, Mean Accuracy: 0.645 (0.049)

For 200.000 estimators, Mean Accuracy: 0.646 (0.050)

For 500.000 estimators, Mean Accuracy: 0.647 (0.046)

Testing with only standardization/normalization without log transformation:

Mean Accuracy: 0.596 (0.056)

Mean Accuracy: 0.625 (0.049)

Mean Accuracy: 0.630 (0.057)

Mean Accuracy: 0.640 (0.051)

Mean Accuracy: 0.642 (0.052)

Mean Accuracy: 0.643 (0.049)

Mean Accuracy: 0.641 (0.058)

Mean Accuracy: 0.642 (0.054)

Mean Accuracy: 0.644 (0.049)

Mean Accuracy: 0.648 (0.052)

Mean Accuracy: 0.651 (0.053)

**Implementing Grid Search for Different Classification Models**

1. Gradient Boosting

New Models do not seem to be given us much improvement, so we research more advanced preprocessing techniques to implement

1. Deep Feature Synthesis

Deep feature synthesis stacks multiple transformation and aggregation operations (which are called [feature primitives](https://docs.featuretools.com/automated_feature_engineering/primitives.html) in the vocab of featuretools) to create features from data spread across many tables.

Entities and Entity Sets: We use the stock name as the index in the entityset.

Creating entities: might need to examine excel file to group columns into appropriate related tables

1. Outlier Detection Algorithm: Isolation Forest

IsolationForests were built based on the fact that anomalies are the data points that are “few and different”.

In an Isolation Forest, randomly sub-sampled data is processed in a tree structure based on randomly selected features. The samples that travel deeper into the tree are less likely to be anomalies as they require more cuts to isolate them. Similarly, the samples which end up in shorter branches indicate anomalies as it was easier for the tree to separate them from other observations.

We will test out the isolation forest and plot the scores before and after log transformation. We will also exclude log transformation and do simple imputation and then test and plot it after that.

After identifying anomalies, should we delete them???

1. First step we try is to delete observations(rows)

* If a data point does not meet a certain threshold, we simply delete it to prevent it from skewing our model
* **After deleting data points we get no change in Gradient Boosting model accuracy:**

Mean Accuracy: 0.631 (0.049)

* Here are the accuracies for different hypwerParameter values for the feature selection method selectKBest:

[0.5683751902587518, 0.6075570776255708, 0.6045433789954336, 0.6075532724505327, 0.6075532724505327, 0.6072716894977169, 0.6116628614916286, 0.6124847792998478, 0.6124771689497718, 0.6157800608828006, 0.6166019786910197, 0.6179756468797565, 0.6144140030441401, 0.6163356164383561, 0.6138698630136986, 0.6141438356164384, 0.6166095890410959, 0.6138660578386606, 0.6141400304414003, 0.6081088280060882, 0.6103006088280061, 0.609482496194825, 0.6056430745814307, 0.602355403348554]

* Knearest neighbours was also unimpressive

**Regression Analysis**

After missing value imputation, one hot encoding, and Standardization

-1.214313098522406e+26 (invalid score)

With normalization2:

-8.566391659346102e+25 (also invalid)

With normalize:

-7.379683635270182e+25 (invalid AGAIN)

NO Regularization:

-8785.275819187631 (at least it's understandable, but why the hell is it negative??)

**False scores; We need to use r^2 to evaluate accuracy of models**

**However, r^2 is not good for evaluation of small test sets: when it's used to evaluate a sufficiently-small test set, the score can be far into the negatives despite good predictions.**

**(\*) represents scores with RepeatedKFold**

With standardization (5 folds):

-9148202927458077099687936.000 (AAAAAhhhh)

***\*-197333639.314***

With normalization:

-135028071561955927578902528.000

***\*-6757411260381922.000***

Normalization2:

-3525418653723857938022400.000

***\*-154174928730431424.000***

With no regularization:

-941681.625

***\*-192480516.592***

**IT'S MUCH BETTER WITH NO REGULARIZATION**

**Not necessarily(\*)**

Next steps:

Do accuracy tests for different cv values

For cv= 5.000, mean is -245016.454

For cv= 10.000, mean is -941681.625

For cv= 20.000, mean is -34827076.971

For cv= 30.000, mean is -65663239.081

For cv= 40.000, mean is -24472387515.341

For cv= 50.000, mean is -79259700.126

For cv= 70.000, mean is -30513835063.833

For cv= 100.000, mean is -95300990.768

For cv= 200.000, mean is -20599609670.506

For cv= 500.000, mean is -10139881506.048

For cv= 1000.000, mean is -5410570267.600

***(\*) accuracy tests for different cv values***

***For standardization:***

***For cv= 5.000, mean is -197333639.314 or -197333639.314***

***For cv= 10.000, mean is -27940888004.272 or -27940888004.272***

***For cv= 20.000, mean is -60541098265.762 or -60541098265.762***

***For cv= 30.000, mean is -44781413817.242 or -44781413817.242***

***For cv= 40.000, mean is -45183349718.966 or -45183349718.966***

***For cv= 50.000, mean is -45057755511.544 or -45057755511.544***

***For cv= 70.000, mean is -49758238986.596 or -49758238986.596***

***For cv= 100.000, mean is -33103679358.853 or -33103679358.853***

***For cv= 200.000, mean is -33680354749.354 or -33680354749.354***

***For cv= 500.000, mean is -37025704414.088 or -37025704414.088***

***For cv= 1000.000, mean is -50701271957.226 or -50701271957.226***

***For Normalization:***

***For cv= 5.000, mean is -6757411260381922.000 or -6757411260381922.000***

***For cv= 10.000, mean is -27050358245.186 or -27050358245.186***

***For cv= 20.000, mean is -58429688981.596 or -58429688981.596***

***For cv= 30.000, mean is -43896687246.644 or -43896687246.644***

***For cv= 40.000, mean is -42565058618.131 or -42565058618.131***

***For cv= 50.000, mean is -46276524432.120 or -46276524432.120***

***For cv= 70.000, mean is -52189122493.930 or -52189122493.930***

***For cv= 100.000, mean is -31522841124.625 or -31522841124.625***

***For cv= 200.000, mean is -33025857331.655 or -33025857331.655***

***For cv= 500.000, mean is -37852325418.485 or -37852325418.485***

***For Normalize2:***

***For cv= 5.000, mean is -154174928730431424.000 or -154174928730431424.000***

***For cv= 10.000, mean is -28943432437.599 or -28943432437.599***

***For cv= 20.000, mean is -55511694727.029 or -55511694727.029***

***For cv= 30.000, mean is -44273180812.128 or -44273180812.128***

***For cv= 40.000, mean is -45361470906.123 or -45361470906.123***

***For cv= 50.000, mean is -48697643977.382 or -48697643977.382***

***For cv= 70.000, mean is -58491487962.853 or -58491487962.853***

***For cv= 100.000, mean is -34662688990.448 or -34662688990.448***

***For cv= 200.000, mean is -36586744246.411 or -36586744246.411***

***For cv= 500.000, mean is -37313007206.755 or -37313007206.755***

***No Regularization:***

***For cv= 5.000, mean is -192480516.592 or -192480516.592***

***For cv= 10.000, mean is -25035066608.244 or -25035066608.244***

***For cv= 20.000, mean is -54147159946.297 or -54147159946.297***

***For cv= 30.000, mean is -28666577129.083 or -28666577129.083***

***For cv= 40.000, mean is -38450797215.902 or -38450797215.902***

***For cv= 50.000, mean is -21208697684.553 or -21208697684.553***

***For cv= 70.000, mean is -11132006438.947 or -11132006438.947***

***For cv= 100.000, mean is -24039503040.749 or -24039503040.749***

***For cv= 200.000, mean is -17614185338.130 or -17614185338.130***

***For cv= 500.000, mean is -7464612004.326 or -7464612004.326***

**Trying advanced preprocessing steps**

* Try shuffling the data set before doing cross\_val\_score

No difference with no regularization:

For cv= 5.000, mean is -21941215.972

For cv= 10.000, mean is -7044196.717

For cv= 20.000, mean is -14241061.519

For cv= 30.000, mean is -64123386.687

For cv= 40.000, mean is -4356702.576

For cv= 50.000, mean is -112855103.296

For cv= 70.000, mean is -8126899331.084

For cv= 100.000, mean is -7385508959.280

For cv= 200.000, mean is -6040063266.769

For cv= 500.000, mean is -5197820067.854

For cv= 1000.000, mean is -6623781023.353

Big difference with normalized scores:

For cv= 5.000, mean is -17373824.957

For cv= 10.000, mean is -26052010.503

For cv= 20.000, mean is -6562635.861

For cv= 30.000, mean is -73112863.332

For cv= 40.000, mean is -4855876.965

For cv= 50.000, mean is -180867334.494

For cv= 70.000, mean is -39732493770.616

For cv= 100.000, mean is -31902280691.704

For cv= 200.000, mean is -44882895828.829

For cv= 500.000, mean is -33719108656.390

For cv= 1000.000, mean is -39317320244.309

Normalize2:

For cv= 5.000, mean is -36852386.405

For cv= 10.000, mean is -12808140.789

For cv= 20.000, mean is -7811612.878

For cv= 30.000, mean is -70810120.662

For cv= 40.000, mean is -5460518.949

For cv= 50.000, mean is -172128051.341

For cv= 70.000, mean is -45664161395.827

For cv= 100.000, mean is -45744975629.719

For cv= 200.000, mean is -50444124539.297

For cv= 500.000, mean is -34984755807.588

For cv= 1000.000, mean is -38534910580.124

Standardization:

For cv= 5.000, mean is -93027693526.271

For cv= 10.000, mean is -967389134171.000

For cv= 20.000, mean is -8621203.321

For cv= 30.000, mean is -72241898.378

For cv= 40.000, mean is -4938020.147

For cv= 50.000, mean is -173666628.859

For cv= 70.000, mean is -44268624331.033

For cv= 100.000, mean is -50979687144.541

For cv= 200.000, mean is -58213995574.985

For cv= 500.000, mean is -25308357909.781

For cv= 1000.000, mean is -29863742875.677

* Advanced preprocessing (1): Removing features with high percentage of missing values

Results with standardization:

For cv= 5.000, mean is -8354866667759.201

For cv= 10.000, mean is -5016200398355.493

For cv= 20.000, mean is -7632225.664

For cv= 30.000, mean is -73721698.437

For cv= 40.000, mean is -4520355.395

For cv= 50.000, mean is -178880345.385

For cv= 70.000, mean is -42918889255.320

For cv= 100.000, mean is -49095462460.448

For cv= 200.000, mean is -45338287363.163

For cv= 500.000, mean is -31317821620.288

For cv= 1000.000, mean is -37825815768.509

***(\*)***

***For cv= 5.000, mean is -197003483.195 or -197003483.195***

***For cv= 10.000, mean is -29742529486.362 or -29742529486.362***

***For cv= 20.000, mean is -58399131188.601 or -58399131188.601***

***For cv= 30.000, mean is -48615569381.587 or -48615569381.587***

***For cv= 40.000, mean is -43672697128.496 or -43672697128.496***

***For cv= 50.000, mean is -49493527003.172 or -49493527003.172***

***For cv= 70.000, mean is -47684988277.160 or -47684988277.160***

***For cv= 100.000, mean is -35384599908.945 or -35384599908.945***

***For cv= 200.000, mean is -36357392361.531 or -36357392361.531***

***For cv= 500.000, mean is -39353153682.707 or -39353153682.707***

***-197003483.195***

***-29742529486.362***

***-58399131188.601***

***-48615569381.587***

***-43672697128.496***

***-49493527003.172***

***-47684988277.160***

***-35384599908.945***

***-36357392361.531***

***-39353153682.707***

Normalization

For cv= 5.000, mean is -139880189893.650

For cv= 10.000, mean is -12290191.051

For cv= 20.000, mean is -6946459.787

For cv= 30.000, mean is -77759615.726

For cv= 40.000, mean is -4652231.766

For cv= 50.000, mean is -162871508.042

For cv= 70.000, mean is -42596926931.286

For cv= 100.000, mean is -50459823603.712

For cv= 200.000, mean is -48793309009.062

For cv= 500.000, mean is -36674103618.669

For cv= 1000.000, mean is -41854958027.757

***(\*)***

***For cv= 5.000, mean is -49632178256085256.000 or -49632178256085256.000***

***For cv= 10.000, mean is -28899753655.028 or -28899753655.028***

***For cv= 20.000, mean is -60543592805.966 or -60543592805.966***

***For cv= 30.000, mean is -42966244869.301 or -42966244869.301***

***For cv= 40.000, mean is -43653680193.903 or -43653680193.903***

***For cv= 50.000, mean is -48480081951.137 or -48480081951.137***

***For cv= 70.000, mean is -46963846554.051 or -46963846554.051***

***For cv= 100.000, mean is -36946672269.917 or -36946672269.917***

***For cv= 200.000, mean is -37100004640.295 or -37100004640.295***

***For cv= 500.000, mean is -41633880272.191 or -41633880272.191***

Normalize 2:

For cv= 5.000, mean is -393868894.957

For cv= 10.000, mean is -174079735.489

For cv= 20.000, mean is -7722180.852

For cv= 30.000, mean is -70122791.529

For cv= 40.000, mean is -5570502.054

For cv= 50.000, mean is -154556635.785

For cv= 70.000, mean is -49649738395.024

For cv= 100.000, mean is -29368801100.296

For cv= 200.000, mean is -50198843860.786

For cv= 500.000, mean is -39672331412.139

For cv= 1000.000, mean is -42006357782.990

***(\*)***

***For cv= 5.000, mean is -138125868939412.594 or -138125868939412.594***

***For cv= 10.000, mean is -28882667042.010 or -28882667042.010***

***For cv= 20.000, mean is -57315823932.040 or -57315823932.040***

***For cv= 30.000, mean is -50902116096.741 or -50902116096.741***

***For cv= 40.000, mean is -44030678324.555 or -44030678324.555***

***For cv= 50.000, mean is -48817828550.535 or -48817828550.535***

***For cv= 70.000, mean is -49337861549.778 or -49337861549.778***

***For cv= 100.000, mean is -35261922402.517 or -35261922402.517***

***For cv= 200.000, mean is -36212286910.926 or -36212286910.926***

***For cv= 500.000, mean is -34601603068.172 or -34601603068.172***

Advanced Preprocessing (3):Remove variables with low correlation with target variable (-0.1<x<0.1) (normalize2)-> removes 199 features and leaves only 5:

For cv= 5.000, mean is -43.915

For cv= 10.000, mean is -107.554

For cv= 20.000, mean is -231.334

For cv= 30.000, mean is -269.567

For cv= 40.000, mean is -274.114

For cv= 50.000, mean is -353.028

For cv= 70.000, mean is -525.113

For cv= 100.000, mean is -874.019

For cv= 200.000, mean is -836.209

For cv= 500.000, mean is -998.089

For cv= 1000.000, mean is -2602.504

***(\*)(1) and (3), sama as just (3), Normalize2***

***For cv= 5.000, mean is -4.216 or -4.216***

***For cv= 10.000, mean is -44.956 or -44.956***

***For cv= 20.000, mean is -334.973 or -334.973***

***For cv= 30.000, mean is -390.994 or -390.994***

***For cv= 40.000, mean is -445.075 or -445.075***

***For cv= 50.000, mean is -454.989 or -454.989***

***For cv= 70.000, mean is -567.466 or -567.466***

***For cv= 100.000, mean is -499.914 or -499.914***

***For cv= 200.000, mean is -723.039 or -723.039***

***For cv= 500.000, mean is -971.223 or -971.223***

***-4.216***

***-44.956***

***-334.973***

***-390.994***

***-445.075***

***-454.989***

***-567.466***

***-499.914***

***-723.039***

***-971.223***

***Normalize***

***For cv= 5.000, mean is -4.216 or -4.216***

***For cv= 10.000, mean is -46.593 or -46.593***

***For cv= 20.000, mean is -334.621 or -334.621***

***For cv= 30.000, mean is -393.520 or -393.520***

***For cv= 40.000, mean is -447.987 or -447.987***

***For cv= 50.000, mean is -452.765 or -452.765***

***For cv= 70.000, mean is -565.645 or -565.645***

***For cv= 100.000, mean is -500.767 or -500.767***

***For cv= 200.000, mean is -719.729 or -719.729***

***For cv= 500.000, mean is -974.262 or -974.262***

***Standardization***

***For cv= 5.000, mean is -4.203 or -4.203***

***For cv= 10.000, mean is -45.779 or -45.779***

***For cv= 20.000, mean is -334.815 or -334.815***

***For cv= 30.000, mean is -395.409 or -395.409***

***For cv= 40.000, mean is -448.742 or -448.742***

***For cv= 50.000, mean is -461.368 or -461.368***

***For cv= 70.000, mean is -569.559 or -569.559***

***For cv= 100.000, mean is -503.103 or -503.103***

***For cv= 200.000, mean is -727.779 or -727.779***

***For cv= 500.000, mean is -986.938 or -986.938***

***We will stick with normalize2 from now on***

Much better but we want to be even better and we can't do that with just 5 features

Therefore,

Remove variables with low correlation with target variable (-0.001<x<0.001) (normalize2)->Left me with 133 features

For cv= 5.000, mean is -151754.444

For cv= 10.000, mean is -6488542.465

For cv= 20.000, mean is -3443722.524

For cv= 30.000, mean is -58710320.526

For cv= 40.000, mean is -1783455.719

For cv= 50.000, mean is -80739633.471

For cv= 70.000, mean is -1260624.130

For cv= 100.000, mean is -74265714.626

For cv= 200.000, mean is -101613330.889

For cv= 500.000, mean is -175543191.241

For cv= 1000.000, mean is -473531615.438

Remove variables with low correlation with target variable (-0.005<x<0.005) (normalize2)->Left me with 21 features

For cv= 5.000, mean is -121123.329

For cv= 10.000, mean is -5150727.611

For cv= 20.000, mean is -2715752.526

For cv= 30.000, mean is -46752198.573

For cv= 40.000, mean is -1404895.731

For cv= 50.000, mean is -64138571.002

For cv= 70.000, mean is -825140.387

For cv= 100.000, mean is -58547605.306

For cv= 200.000, mean is -80289950.631

For cv= 500.000, mean is -138916540.091

For cv= 1000.000, mean is -372133286.578

***(\*)***

***For cv= 5.000, mean is -40585.353 or -40585.353***

***For cv= 10.000, mean is -1332566.791 or -1332566.791***

***For cv= 20.000, mean is -13385887.352 or -13385887.352***

***For cv= 30.000, mean is -31075761.501 or -31075761.501***

***For cv= 40.000, mean is -30068606.344 or -30068606.344***

***For cv= 50.000, mean is -29183701.905 or -29183701.905***

***For cv= 70.000, mean is -18638956.961 or -18638956.961***

***For cv= 100.000, mean is -53095413.898 or -53095413.898***

***For cv= 200.000, mean is -62321588.827 or -62321588.827***

***For cv= 500.000, mean is -73755976.860 or -73755976.860***

Remove variables with low correlation with target variable (-0.007<x<0.007) (normalize2)-> left me with 12 features

For cv= 5.000, mean is -82429.389

For cv= 10.000, mean is -3630932.844

For cv= 20.000, mean is -1958878.941

For cv= 30.000, mean is -32991068.511

For cv= 40.000, mean is -1012781.104

For cv= 50.000, mean is -45228307.063

For cv= 70.000, mean is -587779.383

For cv= 100.000, mean is -42474118.043

For cv= 200.000, mean is -58050921.582

For cv= 500.000, mean is -100459408.355

For cv= 1000.000, mean is -268970069.091

Advanced Preprocessing (3): remove variables with low variation(left me with 193 features)x<>(0.1)

For cv= 5.000, mean is -177779514.954

For cv= 10.000, mean is -42037172.336

For cv= 20.000, mean is -7911828.939

For cv= 30.000, mean is -72100597.813

For cv= 40.000, mean is -6919917.713

For cv= 50.000, mean is -191633414.241

For cv= 70.000, mean is -50821251986.405

For cv= 100.000, mean is -42308247428.442

For cv= 200.000, mean is -57659026437.293

For cv= 500.000, mean is -47353769004.004

For cv= 1000.000, mean is -44449625631.493

* Log transformations

Normalize2, Remove features with low correlation with target variable 0.01, left with 8 features:

For cv= 5.000, mean is -4161.714

For cv= 10.000, mean is -196377.559

For cv= 20.000, mean is -106770.179

For cv= 30.000, mean is -1802615.027

For cv= 40.000, mean is -55268.067

For cv= 50.000, mean is -2479523.542

For cv= 70.000, mean is -32464.609

For cv= 100.000, mean is -2323601.641

For cv= 200.000, mean is -3174479.584

For cv= 500.000, mean is -5490338.654

For cv= 1000.000, mean is -14724480.009

***(\*)***

***For cv= 5.000, mean is -1020.350 or -1020.350***

***For cv= 10.000, mean is -48266.537 or -48266.537***

***For cv= 20.000, mean is -495687.449 or -495687.449***

***For cv= 30.000, mean is -1150708.682 or -1150708.682***

***For cv= 40.000, mean is -1113706.393 or -1113706.393***

***For cv= 50.000, mean is -1081246.649 or -1081246.649***

***For cv= 70.000, mean is -691913.321 or -691913.321***

***For cv= 100.000, mean is -1969030.285 or -1969030.285***

***For cv= 200.000, mean is -2307852.881 or -2307852.881***

***For cv= 500.000, mean is -2731822.966 or -2731822.966***

Normalize2, Remove features with low correlation with target variable 0.1, left with 5 features

For cv= 5.000, mean is -43.915

For cv= 10.000, mean is -107.554

For cv= 20.000, mean is -231.334

For cv= 30.000, mean is -269.567

For cv= 40.000, mean is -274.114

For cv= 50.000, mean is -353.028

For cv= 70.000, mean is -525.113

For cv= 100.000, mean is -874.019

For cv= 200.000, mean is -836.209

For cv= 500.000, mean is -998.089

For cv= 1000.000, mean is -2602.504

(Same as without log transformation)

***(\*)***

***For cv= 5.000, mean is -4.216 or -4.216***

***For cv= 10.000, mean is -44.956 or -44.956***

***For cv= 20.000, mean is -334.973 or -334.973***

***For cv= 30.000, mean is -390.994 or -390.994***

***For cv= 40.000, mean is -445.075 or -445.075***

***For cv= 50.000, mean is -454.989 or -454.989***

***For cv= 70.000, mean is -567.466 or -567.466***

***For cv= 100.000, mean is -499.914 or -499.914***

***For cv= 200.000, mean is -723.039 or -723.039***

***For cv= 500.000, mean is -971.223 or -971.223***

***=>No difference when doing log transformation after removing low correlation values***

Doing log transformation before removing low correlation variables leaves me with 7 features:

For cv= 5.000, mean is -45.795

For cv= 10.000, mean is -110.462

For cv= 20.000, mean is -240.100

For cv= 30.000, mean is -278.621

For cv= 40.000, mean is -287.122

For cv= 50.000, mean is -370.555

For cv= 70.000, mean is -539.240

For cv= 100.000, mean is -894.229

For cv= 200.000, mean is -862.678

For cv= 500.000, mean is -1034.037

For cv= 1000.000, mean is -2729.489

***(\*)***

***For cv= 5.000, mean is -4.221 or -4.221***

***For cv= 10.000, mean is -46.178 or -46.178***

***For cv= 20.000, mean is -338.659 or -338.659***

***For cv= 30.000, mean is -401.993 or -401.993***

***For cv= 40.000, mean is -453.254 or -453.254***

***For cv= 50.000, mean is -463.887 or -463.887***

***For cv= 70.000, mean is -580.530 or -580.530***

***For cv= 100.000, mean is -518.332 or -518.332***

***For cv= 200.000, mean is -747.932 or -747.932***

***For cv= 500.000, mean is -1007.868 or -1007.868***

***=>Very Slightly worse doing log transformation before removing low correlation values, but not much difference***

* **SelectKBest**

We decide to try out selectKBest without removing variables from advanced preprocessing and using normalize2, and also 5 CV folds, and also removing high percentage of missing values, firstly with log transformations,

Shape

Description automatically generated with low confidence

[-68.16598509424314, -69.76008017854558, -71.9285176717113, -74.3081834631125, -76.0137779249533, -76.61109702687908, -90.82745144232153, -158.42331738722498, -158.43979455500576, -169.83840598926093, -133.4428243955195, -149.05447241087614, -191.01129318961517, -147.79206465931185, -154.01135344048149, -11482.338356646931, -11973.31478466012, -12835.639942261345, -449.78730401016185, -388.92618492099007, -853.107323791501, -1116.6781600125948, -1066.8292807837454, -612.589820924315

***(\*)***

***Chart, line chart

Description automatically generated***

***Here are the accuracies for different hypwerParameter values for the feature selection method selectKBest:***

***[-0.5510104267321101, -0.5690084582323335, -0.5782817374788073, -0.6015329023382818, -0.6156079451670707, -0.6172877526062723, -0.6345085534431991, -0.73985454690197, -0.7409099818184244, -0.7450067050793995, -0.7438626999506054, -0.7439557374788219, -0.787262998096854, -194.10353788252726, -194.64211467802426, -197.96738000735002, -437.1132242388515, -442.0217283922878, -134.96779029429828, -137.42633136483272, -131.85000351476452, -133.53320543418937, -132.01299235179403, -130.73056421980732]***

Now without log transformation

**Chart, shape

Description automatically generated**

**[-68.16598509424314, -69.76008017854558, -71.11738509385187, -73.99110269607621, -75.67640350769867, -76.17920722904196, -114.0273265266811, -171.60226559634822, -34338304.901241824, -34338304.901241824, -34617531.31336985, -34617531.31336985, -34617531.31336985, -34660262.58424123, -34717300.48874055, -35093796.18092222, -34126960.003605664, -33748400.71050501, -33292148.121020436, -39836024.195407145, -4491336.472681736, -4485247.725295702, -4056875.442545511, -4150294.2114651157]**

***(\*)***

***Shape, rectangle

Description automatically generated***

***Here are the accuracies for different hypwerParameter values for the feature selection method selectKBest:***

***[-0.5510104267321101, -0.5690084582323335, -0.5901858873810083, -0.6100146770742901, -0.6242441810552619, -0.6261120392726475, -0.6326823810681181, -14.710221950737878, -589805.4941563642, -589805.4941563642, -591279.013456803, -591279.013456803, -591279.013456803, -592474.7957224797, -592408.7552120269, -594905.0746166585, -582217.7596921085, -582709.4639457955, -563527.1857590649, -648676.1532135302, -22186.13863900609, -19084.352509696648, -18815.947190296552, -20486.531483557326]***

***Scores are worse without log transformation***

* **Multicollinearity**

After multicollinearity test with selectK=18(\*), we have

Shape

Description automatically generated

[-68.16598509424314, -69.76008017854558, -71.11738509385187, -73.99110269607621, -75.67640350769867, -114.39812846729714, -170.40292164301528, -34338304.901241824, -34617531.31336985, -34626903.98035667, -34682879.67827262, -34666010.46171544, -35055149.25119893, -35204779.024328806, -34670677.61000777, -33748400.71050501]

***[-0.5510104267321101, -0.5690084582323335, -0.5782817374788073, -0.6015329023382818, -0.6156079451670707, -0.634381464181201, -0.7389397372839139, -0.7409099818184244, -0.7438626999506054]***

**Hyperparameter Testing**

A better approach is to objectively search different values for model hyperparameters and choose a subset that results in a model that achieves the best performance on a given dataset. This is called hyperparameter optimization or hyperparameter tuning and is available in the scikit-learn Python machine learning library. The result of a hyperparameter optimization is a single set of well-performing hyperparameters that you can use to configure your model.

Both classes require two arguments. The first is the model that you are optimizing. This is an instance of the model with values of hyperparameters set that you want to optimize. The second is the search space. This is defined as a [dictionary](https://docs.python.org/3/tutorial/datastructures.html#dictionaries) where the names are the hyperparameter arguments to the model and the values are discrete values or a distribution of values to sample in the case of a random search.

Finally, the search can be made parallel, e.g. use all of the CPU cores by *specifying the “n\_jobs” argument as an integer with the number of cores in your system, e.g. 8. Or you can set it to be -1 to automatically use all of the cores in your system.*

**We do GridSearchCV for our model with the best score (normalize2 and selectKBest=3)**

**=> Grid Search reveals Best hyperparameter values for our linear regression model:**

**Best Score: -0.2295753926521467**

**Best Hyperparameters: {'fit\_intercept': False, 'normalize': True}**

**Bayesian Optimization**

[**Bayesian optimization**](https://sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf) **is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric**

CHECKPOINT 4

Testing with other regression models:

1. **Ridge Regression**

* Without any model hyperparameters for the Ridge Model, after removeMissing(-18), filling missing values, one hot encoding (+10) (which leaves us with 214 features) and normalize2, using repeatedKFolds(5 splits/cv)  for CrossValidation, we get an r2 score of -22.33, a neg\_mean\_absolute\_error of -2326.167, and a neg\_mean\_squared\_error of -1701927897.477
* *Trying different regularization techniques:* No regularization gives us -176960144.778. Normalize gives us the same r2 score, standardize gives us -13822.298, so chances are we stick to normalize or normalize2.
* *Tring different cv values for each regularization technique:*

No Regularization->

For cv= 5.000, mean is -176960144.778

For cv= 6.000, mean is -79255355.427

For cv= 7.000, mean is -469367.623

For cv= 8.000, mean is -136571099.344

For cv= 9.000, mean is -3673204393.903

For cv= 10.000, mean is -28345630940.245

For cv= 20.000, mean is -51845922339.610

For cv= 30.000, mean is -46717121940.558

For cv= 40.000, mean is -41713398434.180

For cv= 50.000, mean is -47041218039.878

(Side note, really long warnings starting popping right off the bat for no regularization, but for normalization, they started appearing after cv=1000)

Normalize->

For cv= 5.000, mean is -22.330

For cv= 10.000, mean is -1751.843

For cv= 20.000, mean is -5704.265

For cv= 30.000, mean is -8154.520

For cv= 40.000, mean is -12826.014

For cv= 50.000, mean is -12084.413

For cv= 70.000, mean is -13489.894

For cv= 100.000, mean is -12952.285

For cv= 200.000, mean is -17713.254

For cv= 500.000, mean is -26781.107

For cv= 1000.000, mean is -137729.182

Normalize2 [Same as Normalize again]->

For cv= 5.000, mean is -22.330

For cv= 10.000, mean is -1751.843

For cv= 20.000, mean is -5704.265

For cv= 30.000, mean is -8154.520

For cv= 40.000, mean is -12826.014

For cv= 50.000, mean is -12084.413

For cv= 70.000, mean is -13489.894

For cv= 100.000, mean is -12952.285

For cv= 200.000, mean is -17713.254

For cv= 500.000, mean is -26781.107

For cv= 1000.000, mean is -137729.182

-22.330

-1751.843

-5704.265

-8154.520

-12826.014

-12084.413

-13489.894

-12952.285

-17713.254

-26781.107

-137729.182

Standardization->

For cv= 5.000, mean is -13822.298

For cv= 10.000, mean is -569519.431

For cv= 20.000, mean is -5772563.986

For cv= 30.000, mean is -13339480.277

For cv= 40.000, mean is -12908128.119

For cv= 50.000, mean is -12534853.359

For cv= 70.000, mean is -8000856.379

For cv= 100.000, mean is -22777613.752

For cv= 200.000, mean is -26715920.710

For cv= 500.000, mean is -31644579.946

For cv= 1000.000, mean is -576490852.119

**Looks like we stick to normalize2 using 5cv**

* *Advanced Preprocessing Techniques:* We try out our advanced preprocessing techniques

1. Removing features with high percentage of missing values (Done)
2. Removing features with low variation
3. Removing Variables with low correlation with target variable
4. Log Transformations

We try each one solitarily and then test how well they do working in tandem

* For just (2) with normalization2 [199 features]->

For cv= 5.000, mean is -21.867

For cv= 10.000, mean is -1749.892

For cv= 20.000, mean is -5640.813

For cv= 30.000, mean is -8088.032

For cv= 40.000, mean is -12753.527

For cv= 50.000, mean is -11986.159

For cv= 70.000, mean is -13362.897

For cv= 100.000, mean is -12774.301

For cv= 200.000, mean is -17508.988

For cv= 500.000, mean is -26396.727

For cv= 1000.000, mean is -138369.867

(Minor Improvement, could increase variation threshold)

* For just (3) with normalization2 [15 features]->

For cv= 5.000, mean is -16.513

For cv= 10.000, mean is -1473.152

For cv= 20.000, mean is -4412.332

For cv= 30.000, mean is -6036.120

For cv= 40.000, mean is -10293.428

For cv= 50.000, mean is -9441.365

For cv= 70.000, mean is -11040.061

For cv= 100.000, mean is -9516.378

For cv= 200.000, mean is -13370.876

For cv= 500.000, mean is -20616.313

For cv= 1000.000, mean is -93141.279

* For (2) and (3) with normalization2 [14  features] we have same scores as just (3)
* For just (4) with normalization2 [still with 214 features]->

For cv= 5.000, mean is -46.498

For cv= 10.000, mean is -2822.493

For cv= 20.000, mean is -10687.773

For cv= 30.000, mean is -15777.151

For cv= 40.000, mean is -20470.123

For cv= 50.000, mean is -21695.691

For cv= 70.000, mean is -25233.112

For cv= 100.000, mean is -26181.283

For cv= 200.000, mean is -34220.693

For cv= 500.000, mean is -53617.812

For cv= 1000.000, mean is -403638.055

* For (4) and (3) with normalization2 [15 features]->

For cv= 5.000, mean is -16.513

For cv= 10.000, mean is -1473.152

For cv= 20.000, mean is -4412.332

For cv= 30.000, mean is -6036.120

For cv= 40.000, mean is -10293.428

For cv= 50.000, mean is -9441.365

For cv= 70.000, mean is -11040.061

For cv= 100.000, mean is -9516.378

For cv= 200.000, mean is -13370.876

For cv= 500.000, mean is -20616.313

For cv= 1000.000, mean is -93141.279

* For (4) and (2) with normalization2 [199 features]->

For cv= 5.000, mean is -45.760

For cv= 10.000, mean is -2801.796

For cv= 20.000, mean is -10562.576

For cv= 30.000, mean is -15602.247

For cv= 40.000, mean is -20274.429

For cv= 50.000, mean is -21473.587

For cv= 70.000, mean is -24973.958

For cv= 100.000, mean is -25866.319

For cv= 200.000, mean is -33851.405

For cv= 500.000, mean is -53036.668

For cv= 1000.000, mean is -398386.945

* (4) and Increasing threshold of variation for (3) from -0.5<x<0.5 to -0.99<x<0.99 []-> No bloody difference
* Using **selectKBest (multicollinearity test included)**, starting with no advanced preprocessing, 5CV folds, and normalize2 (r2 scoring)
* -0.5510104267321108,

-0.5690084582323331,

-0.5901858873810081,

-0.6100146770742899,

-0.6242441810552616,

-0.6261120392726474,

-0.6326823810681184,

-14.710221950737807,

-589805.4941514756,

-589805.4941514756,

-591279.013480284,

-591279.013480284,

-591279.013480284,

-592474.795718825,

-592408.7552138658,

-594905.0746274034,

-582217.7597357762,

-582709.464009105,

-563527.1857759741,

-648676.1532190351,

-22186.138640633875,

-19084.352509970522,

-18815.94719058203,

-20486.53148361204

Shape, rectangle

Description automatically generated

Negative mean absolute squared

[-1950.8483214700368, -1951.1153634502382, -1968.0459604457635, -1994.8163501912513, -1999.644760218321, -2002.587184208725, -2015.1532165640922, -2107.987396291242, -315996.6843408529, -315996.6843408529, -316836.8526101113, -316836.8526101113, -316836.8526101113, -317152.7527155531, -317145.6733254567, -318038.8472222656, -330670.1970387309, -331426.78908828716, -326839.0552361482, -349416.04364759725, -84681.9799644872, -80405.25405671899, -80027.1216357199, -79718.3610779732]

* Using selectKBest, with advanced preprocessing (2), 5CV folds, and normalize2 (r2 scoring)

[-0.5510104267321108, -0.5690084582323331, -0.5839939448157152, -0.5834742736792586, -0.5834740731313554, -27.917358507632287, -596699.0717879713, -596699.0717879713, -597507.9201573126, -597953.632037526, -597953.632037526, -597953.632037526, -598178.0468019118, -599051.956785138, -599462.912171733, -602559.4424934705, -594303.8738513265, -593889.3759759406, -574349.6023416087, -574892.42670434, -24014.13735538303, -21130.269965204905, -20391.140046191023, -20214.925787451728]

**The same scores**

* Using selectKBest, with advanced preprocessing (3) but reducing threshold to increase features (|x|<0.001 gave us 143 features instead of 15), 5CV folds, and normalize2 (r2 scoring)

[-0.5510104267321108, -0.5690084582323331, -0.5901858873810081, -0.6100146770742899, -0.6242441810552616, -0.6261120392726474, -0.6326823810681184, -0.6729588126059013, -0.6824646275125732, -0.6832081052627623, -1.9840142861004797, -2.0062205781310536, -1.9985025833288652, -1.996067804712343, -2.00048295244866, -21.38306230325374, -162.61294901707026, -134.5953447672881, -134.58066352430586, -135.7769438823245, -137.19029683763026, -137.89062677192737, -138.40640295292627]

**We dont see hugely negative numbers again but still not a good performing model**

**Shape

Description automatically generated**

* Using selectKBest, with advanced preprocessing (3) but reducing threshold to increase features (|x|<0.001 gave us 143 features instead of 15), **adding log transformations (4),** 5CV folds, and normalize2 (r2 scoring)

-0.5510104267321108,

-0.5690084582323331,

-0.5782817374788073,

-0.6015329023382813,

-0.6156079451670706,

-0.6172877526062726,

-0.6345085534431987,

-0.6378549286816425,

-124.07894136123264,

-199.323139073713,

-199.83177850033866,

-199.8946154920029,

-201.78352032662178,

-316.21581003695707,

-443.2200221931206,

-134.43464914781777,

-134.17057205704901,

-136.25612165064882, -136.96719682522138, -129.73974998307202, -133.1732006436854, -133.54572064479362, -133.6948650808177]

Chart, line chart

Description automatically generated

* Using selectKBest, with advanced preprocessing (3) but reducing threshold to increase features (|x|<0.001 gave us 143 features instead of 15), **adding removelowVariation (2),** 5CV folds, and normalize2 (r2 scoring)
* [-0.5510104267321108, -0.5690084582323331, -0.5901858873810081, -0.6100146770742899, -0.6242441810552616, -0.6261120392726474, -0.6326823810681184, -0.6729588126059013, -0.6824646275125732, -0.6832081052627623, -1.9840142861004797, -2.0062205781310536, -1.9985025833288652, -1.996067804712343, -2.00048295244866, -21.38306230325374, -162.61294901707026, -134.5953447672881, -134.58066352430586, -135.7769438823245, -137.19029683763026, -137.89062677192737, -138.40640295292627]

Shape

Description automatically generated

* Same as above, **except we remove multicollinearity filter in selectKBest**

[-0.5510104267321108, -0.5690084582323331, -0.5901858873810081, -0.6100146770742899, -0.6242441810552616, -0.6261120392726474, -0.6326905695573385, -0.6729673337193214, -0.6824791259977321, -0.68322093645946, -2.002685193212547, -1.9607298642323734, -1.9459487078079654, -1.9742424466382154, -6.990650094707901e+21, -44.827951771049996, -2.774704809713585e+21, -134.6916279862627, -134.6335515601357, -135.83892850519618, -1705.793371555541, -1706.056393655498, -1716.3685320283453]

**Chart, line chart

Description automatically generated**

**->**models with high numbers become worse

* Adding (4) to above and putting multicollinearity filter back

[-0.5510104267321108, -0.5690084582323331, -0.5782817374788073, -0.6015329023382813, -0.6156079451670706, -0.6172877526062726, -0.6345085534431987, -88.83282794616395, -89.39427368457972, -89.4155523987959, -90.90512027576162, -90.99018305312131, -234.1171879483607, -134.2480041882872, -134.35820503545744, -134.7630190707661, -134.4511563630358, -132.80873381760506, -130.38669575647444, -131.64961871978835, -134.7729314296219, -79.01923982372332]

Chart, line chart

Description automatically generated

**Trying Grid Search with the following model setup that gave us highest scores and lowest range (before going into hyperparameterTesting):**

**For (4) and (3) with normalization2 [15 features]->**

**For cv= 5.000, mean is -16.513**

length of inputCols:  131 (**Shouldn’t it be 15??**) double checked to confirm it is 131! Because I increased the threshold for (3) from -0.01<x<0.01 to -0.001<x<0.001

Best Score: -6.346477215232976 **WRONG!! (3) applied twice**

**Correct results:**

length of inputCols:  143

Best Score: -6.348952962829813

Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}

***Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}***

Resetting threshold for (3) to -0.01<x<0.01 gives us :

length of inputCols:  17

Best Score: -3.2950904979728524

Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}

\*\*\*

(4) and (3) with normalization2 except (2) is done after (4):

length of inputCols:  149

Best Score: -6.82967280869537

Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}

Just (3)(0.001) with normalization 2 gives us:

length of inputCols:  143

Best Score: -3.321623712656341

Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}

(3)(0.001) and (2) with normalization2 gives us:

length of inputCols:  131

Best Score: -2.543698407009177

Best Hyperparameters: {'alpha': 100, 'fit\_intercept': False, 'normalize': True}

\*\*\*

**Setting hyperparameter values for selectKBest testing:**

* Using selectKBest, with advanced preprocessing (3) but reducing threshold to increase features (|x|<0.001 gave us 143 features instead of 15), **adding removelowVariation (2),** 5CV folds, and normalize2 (r2 scoring)

***-0.006449834410345515, -0.08662956618229559, -0.10043884843905818, -0.10305508475277862, -0.1030518791888775, -0.10305445218298405, -0.10305442334549217, -0.1030477680868948, -0.10303258944526164, -0.10296779599969016, -0.10300079915159707, -0.10299715998533873, -0.10307095755522583, -0.10310686912179116, -0.10316215095128035, -0.10320723360475043, -0.10319966687323508, -0.10354279336792889, -0.10373258855402406, -0.10293668372075696, -0.10268606025913418, -0.11440663873540523, -2.509321884941376, -2.5413456548347275***

**=> Huge Improvement!**

**Shape

Description automatically generated**

**Unfortunately, still no positive scores**

* Adding log transformation (4)

-0.006449834410345515, -0.08662956618229559, -0.10043884843905818, -0.10305508475277862, -0.1030518791888775, -0.10305445218298405, -0.10305442334549217, -0.10305301071626156, -0.10305351138515483, -0.103053712041695, -0.10298449103455151, -0.10297501891920083, -0.10298430100593757, -0.10298413069789782, -0.1031429592070465, -0.10314392242958158, -0.10315348946030772, -0.10211413635156877, -0.1536633980765678, -0.19800494805159316, -0.9468815712660262, -2.0458641865919676, -4.618130740797998, -4.656903982549677

**Not as good**

**Shape

Description automatically generated**

**Trying different Ridge solvers: practically no difference**

**Using negative mean absolute error for scoroing:**

-1022.2767093860615, -1202.2793330866511, -1240.5956625558533, -1251.2349324176337, -1251.2261265541101, -1251.2312558723056, -1251.2311941752241, -1251.2324891239452, -1251.2391165461731, -1251.2183957937573, -1251.2823534655722, -1251.282493632387, -1251.6714735855176, -1251.6922542386164, -1251.865640218696, -1251.9696213038144, -1251.9534851813323, -1253.3652079199771, -1253.9452775650798, -1250.0291640121384, -1249.6735407121598, -1255.7670273658293, -1901.2527235482569, -1904.318182590837

Shape, rectangle

Description automatically generated

**2) Lasso Regression**

* Without any model hyperparameters for the Lasso Model, after removeMissing(-18), filling missing values, one hot encoding (+10) (which leaves us with 214 features) and normalize2, using repeatedKFolds(5 splits/cv)  for CrossValidation, we get an r2 score of -11413.171, a neg\_mean\_absolute\_error of -5956.415, and a neg\_mean\_squared\_error of -1701927897.477
* *Trying different regularization techniques(r2):* No regularization gives us -214660873.909. Normalize gives us the same r2 score, standardize gives us -48292.114, so chances are we stick to normalize or normalize2.
* *Tring different cv values for each regularization technique:*

No Regularization->Extremely Wild Scores

Normalize2->

For cv= 5.000, mean is -11413.171

For cv= 10.000, mean is -299075.871

For cv= 20.000, mean is -2573157.264

For cv= 30.000, mean is -5629319.456

For cv= 40.000, mean is -5291721.393

For cv= 50.000, mean is -5050529.846

For cv= 70.000, mean is -3152747.440

For cv= 100.000, mean is -8854663.886

For cv= 200.000, mean is -10201575.509

For cv= 500.000, mean is -11938309.935

For cv= 1000.000, mean is -217231875.763

Normalize->

For cv= 5.000, mean is -11413.171

For cv= 10.000, mean is -299075.871

For cv= 20.000, mean is -2573157.264

For cv= 30.000, mean is -5629319.456

For cv= 40.000, mean is -5291721.393

For cv= 50.000, mean is -5050529.846

For cv= 70.000, mean is -3152747.440

For cv= 100.000, mean is -8854663.886

For cv= 200.000, mean is -10201575.509

For cv= 500.000, mean is -11938309.935

Standardize->Not worthy of reporting

For cv= 5.000, mean is -48292.114

For cv= 10.000, mean is -1673346.942

For cv= 20.000, mean is -16820754.929

For cv= 40.000, mean is -37575333.794

* *Advanced Preprocessing Techniques:* We try out our advanced preprocessing techniques

1. Removing features with high percentage of missing values (Done)
2. Removing features with low variation
3. Removing Variables with low correlation with target variable
4. Log Transformations

We try each one solitarily and then test how well they do working in tandem

* With just (2)(199 features) and using normalize2

For cv= 5.000, mean is -11390.099

For cv= 10.000, mean is -298200.892

For cv= 20.000, mean is -2565295.579

For cv= 30.000, mean is -5611946.867

For cv= 40.000, mean is -5275280.731

For cv= 50.000, mean is -5034977.211

For cv= 70.000, mean is -3142694.577

For cv= 100.000, mean is -8827601.169

For cv= 200.000, mean is -10170901.115

* With just (3)(|x|<0.001)(143 Features) using normalize2->

For cv= 5.000, mean is -11313.218

For cv= 10.000, mean is -296540.651

For cv= 20.000, mean is -2552873.158

For cv= 30.000, mean is -5585519.915

For cv= 40.000, mean is -5250828.360

For cv= 50.000, mean is -5011616.725

For cv= 70.000, mean is -3128547.207

For cv= 100.000, mean is -8787068.115

For cv= 200.000, mean is -10124287.539

For cv= 500.000, mean is -11847972.487

* With just (3)(|x|<0.01)(17 Features) using normalize2

For cv= 5.000, mean is -96.443

For cv= 10.000, mean is -2435.741

For cv= 20.000, mean is -16167.908

For cv= 30.000, mean is -31389.686

For cv= 40.000, mean is -27779.948

For cv= 50.000, mean is -25520.361

For cv= 70.000, mean is -15506.709

For cv= 100.000, mean is -40988.016

For cv= 200.000, mean is -45327.502

For cv= 500.000, mean is -51692.344

* With just (4) using normalize2

For cv= 5.000, mean is -9675.440

For cv= 10.000, mean is -245467.425

For cv= 20.000, mean is -2067500.681

For cv= 30.000, mean is -4506125.744

For cv= 40.000, mean is -4225470.170

For cv= 50.000, mean is -4023729.656

For cv= 70.000, mean is -2508947.209

For cv= 100.000, mean is -7030281.392

For cv= 200.000, mean is -8086585.475

For cv= 500.000, mean is -9446863.276

* With (2) and (3)(|x|<0.001)(131 features)

For cv= 5.000, mean is -11289.389

For cv= 10.000, mean is -295588.102

For cv= 20.000, mean is -2544371.771

For cv= 30.000, mean is -5566869.086

For cv= 40.000, mean is -5233214.612

For cv= 50.000, mean is -4994908.348

For cv= 70.000, mean is -3117867.908

For cv= 100.000, mean is -8757794.825

For cv= 200.000, mean is -10091039.780

For cv= 500.000, mean is -11808874.472

* With (2) and (3)(|x|<0.01)(17 features)

For cv= 5.000, mean is -96.443

For cv= 10.000, mean is -2435.741

For cv= 20.000, mean is -16167.908

For cv= 30.000, mean is -31389.686

For cv= 40.000, mean is -27779.948

For cv= 50.000, mean is -25520.361

For cv= 70.000, mean is -15506.709

For cv= 100.000, mean is -40988.016

For cv= 200.000, mean is -45327.502

For cv= 500.000, mean is -51692.344

* With (2) and (4) (199 features)->

For cv= 5.000, mean is -9631.047

For cv= 10.000, mean is -243502.194

For cv= 20.000, mean is -2049696.576

For cv= 30.000, mean is -4463999.837

For cv= 40.000, mean is -4184930.631

For cv= 50.000, mean is -3985076.785

For cv= 70.000, mean is -2483517.813

For cv= 100.000, mean is -6962747.767

For cv= 200.000, mean is -8008781.617

For cv= 500.000, mean is -9356201.861

* With (3)(|x|<0.001) and (4) (143 features)

For cv= 5.000, mean is -10081.793

For cv= 10.000, mean is -254400.932

For cv= 20.000, mean is -2159947.698

For cv= 30.000, mean is -4702470.197

For cv= 40.000, mean is -4411728.032

For cv= 50.000, mean is -4202757.338

For cv= 70.000, mean is -2618879.908

For cv= 100.000, mean is -7344932.911

For cv= 200.000, mean is -8449752.707

For cv= 500.000, mean is -9874447.094

* With (3)(|x|<0.01) and (4) (18 features)

For cv= 5.000, mean is -96.443

For cv= 10.000, mean is -2435.741

For cv= 20.000, mean is -16167.909

For cv= 30.000, mean is -31389.688

For cv= 40.000, mean is -27779.951

For cv= 50.000, mean is -25520.363

For cv= 70.000, mean is -15506.712

For cv= 100.000, mean is -40988.018

For cv= 200.000, mean is -45327.506

For cv= 500.000, mean is -51692.349

* With (2)(3)(|x|<0.001) and (4) (131 features)

For cv= 5.000, mean is -10036.880

For cv= 10.000, mean is -252839.675

For cv= 20.000, mean is -2146261.969

For cv= 30.000, mean is -4672559.668

For cv= 40.000, mean is -4383849.085

For cv= 50.000, mean is -4176609.807

For cv= 70.000, mean is -2602025.643

For cv= 100.000, mean is -7299334.419

For cv= 200.000, mean is -8397742.582

For cv= 500.000, mean is -9814340.728

**Using selectKBest features (with multicollinearity filter)**

* Starting with just (3)(|x|<0.001):

***-0.9000395577598871, -1.0352389094852592, -1.0862375825073511, -1.1170686880318272, -1.1321050404660433, -1.1948973248311212, -1.1990322725687075, -1.1991213095989226, -1.2056702668573118, -1.2052109770295152, -1.2073537610908818, -1.2098609513249237, -1.652280677593443, -1.653134925285474, -1.669737526092844, -1.7094120830719604, -1.7139636323812013, -6.165736602306411, -7286.166073855119, -21.2344209600112, -21.238626140658532, -21.32112524733719, -21.610914048122808, -21.856391023230362, -21.892152933294028, -5503.665585351872***

A picture containing shape

Description automatically generated

\*Note: selectKBest also has multicollinearity filter, which may explain difference in scores

**Without multicollinearity:**

-0.9000395577598871, -1.0352389094852592, -1.0862375825073511, -1.1170686880318272, -1.1321050404660433, -1.1948973248311212, -1.1990322725687075, -1.1991213095989226, -1.2056707571989937, -1.2052114611907168, -1.2073544148909607, -1.2098616170489076, -1.6522812718230184, -1.6539981088821607, -1.6705250816071355, -1.7102395219998088, -1.7146948634648622, -9.48604146996411, -7286.703298951013, -21.23468222611118, -21.238786895517016, -21.32930874348024, -508.91080544214884, -506.722181487801, -508.2095278524048, -11313.217557543625

Shape

Description automatically generated with medium confidence

**3) Random Forests**

* Without any model hyperparameters for the Random Forests Model, after removeMissing(-18), filling missing values, one hot encoding (+10) (which leaves us with 214 features) and normalize2, using repeatedKFolds(5 splits/cv)  for CrossValidation, we get an r2 score of -165.123, a neg\_mean\_absolute\_error of -1578.266, and a neg\_mean\_squared\_error of -1582559222.384
* *Trying different regularization techniques(r2):* No regularization gives us -186.408. Normalize gives us -161.270, standardize gives us -161.529, so doesn't matter, we stick to normalize or normalize2.
* Testing CV Values:

Takes way too long, 5CV: -162.689, 10CV: -7489.155, we stick with 5 cv to test advanced preprocessing

* Just (2):

length of inputCols:  199

mean is -193.377

* Just (3)(0.001):

length of inputCols:  143

mean is -208.580

* Just (3)(0.01):

length of inputCols:  18

mean is -193.916

* Just (4):

length of inputCols:  214

mean is -234.536

* (2) and (3)(0.001):

length of inputCols:  131

mean is -170.861

* (2) and (3)(0.01):

length of inputCols:  17

mean is -169.001

* (2) and (4):

length of inputCols:  199

mean is -214.938

(log transformation made model perform worse)

* **Will Use** (2) and (3)(0.001): for selectKBest features testing (with multicollinearity):

-0.8899652588015599, -0.9781335866149354, -1.1351123719099703, -1.1962351258539028, -1.6144709669073407, -1.9222759088804637, -2.147604463335402, -16.230258461369644, -19.006643462080966, -35.24236110547499, -124.09392354097601, -49.97682867176213, -111.31823071297187, -160.0927693199363, -163.09615765080167, -136.8577747193565, -210.4173686280716, -67.03476159970802, -47.229679862397724, -88.5040531369154, -79.20527418798308, -129.16789006061583, -112.37735475683782, -112.5291336655974

Chart, line chart

Description automatically generated

Grid Search with Random Forests

***-0.8973348248491214, -0.9817551651533332, -1.2608781308378192, -1.2050021082526159, -1.6020722085620782, -2.2258636038597075, -2.0046898117808483, -10.928052417359293, -5.732392952673267, -8.49894455242966, -31.07849947271361, -25.23679208712984, -21.668004428879257, -17.567512699890504, -17.989191432604585, -27.392862008209196, -25.67357140156099, -13.315608959885337, -10.83673670310097, -7.17106769828815, -13.227682227288467, -7.2855000059167185, -6.7824836847302725, -7.930591950805773***