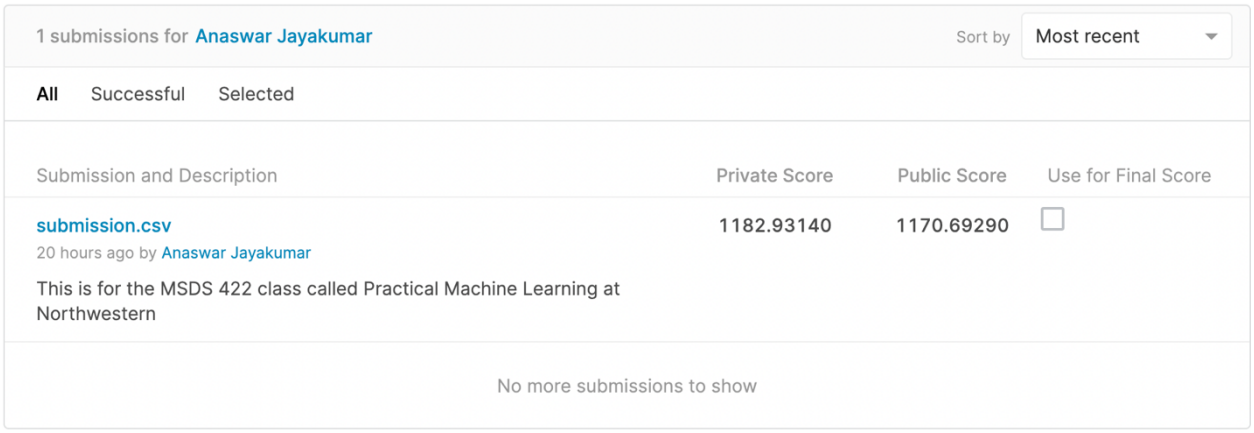
Kaggle Results Relative to Peers



1 submissions for Anaswar Jayakumar

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission.csv 20 hours ago by Anaswar Jayakumar This is for the MSDS 422 class called Practical Machine Learning at Northwestern	1182.93140	1170.69290	☐

No more submissions to show

MSDS 422 Assignment 2

April 11, 2021

1 MSDS 422 Assignment 2: Evaluating Regression Models

This week, you will be assigned one of two projects.

Compete in the Allstate Claims Severity Competition. You must establish an account with Kaggle.com (free). You must submit your forecasts to Kaggle.

Use many explanatory variables for your predictions. Employ at least two regression modeling methods selected from those discussed in Chapter 4 of the Géron (2017) textbook: linear regression, ridge regression, lasso regression, and elastic net. Evaluate these methods within a cross-validation design using the mean absolute error (MAE) as an index of prediction error. Submit your models to Kaggle.com for evaluation on the test set. Python should be your primary environment for conducting this research.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn 0.20.

2 Step 1 - Loading the required libraries and modules.

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os
```

```
# Import all required Python libraries such as pandas, numpy, matplotlib,␣
 ↪seaborn, etc

import pandas as pd
import matplotlib.pyplot as plt

from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[2]:
```
# Initialize poly_features as True
poly_features = True
```

3 Step 2 - Loading the data and performing basic data checks.

[3]:
```
# Import Allstate Insurance data which is a CSV file

train_dataset = '/Users/anaswarjayakumar/Downloads/train 2.csv'
train_df = pd.read_csv(train_dataset, sep = ",")
train_df.head()
```

[3]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	…	cont6	cont7	\
0	1	A	B	A	B	A	A	A	A	B	…	0.718367	0.335060	
1	2	A	B	A	A	A	A	A	A	B	…	0.438917	0.436585	
2	5	A	B	A	A	B	A	A	A	B	…	0.289648	0.315545	
3	10	B	B	A	B	A	A	A	A	B	…	0.440945	0.391128	
4	11	A	B	A	B	A	A	A	A	B	…	0.178193	0.247408	

	cont8	cont9	cont10	cont11	cont12	cont13	cont14	loss
0	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493	0.714843	2213.18
1	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431	0.304496	1283.60
2	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709	0.774425	3005.09
3	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077	0.602642	939.85
4	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011	0.432606	2763.85

```
[5 rows x 132 columns]
```

[4]:
```
train_df.dtypes
```

[4]:
```
id          int64
cat1       object
cat2       object
cat3       object
cat4       object
            …
cont11    float64
cont12    float64
cont13    float64
cont14    float64
loss      float64
Length: 132, dtype: object
```

[5]:
```
test_df = pd.read_csv('/Users/anaswarjayakumar/Downloads/test.csv')
test_df.head()
```

[5]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	…	cont5	cont6	\
0	4	A	B	A	A	A	A	A	A	B	…	0.281143	0.466591	
1	6	A	B	A	B	A	A	A	A	B	…	0.836443	0.482425	
2	9	A	B	A	B	B	A	B	A	B	…	0.718531	0.212308	
3	12	A	A	A	A	B	A	A	A	A	…	0.397069	0.369930	
4	15	B	A	A	A	A	B	A	A	A	…	0.302678	0.398862	

	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14
0	0.317681	0.61229	0.34365	0.38016	0.377724	0.369858	0.704052	0.392562
1	0.443760	0.71330	0.51890	0.60401	0.689039	0.675759	0.453468	0.208045
2	0.325779	0.29758	0.34365	0.30529	0.245410	0.241676	0.258586	0.297232
3	0.342355	0.40028	0.33237	0.31480	0.348867	0.341872	0.592264	0.555955
4	0.391833	0.23688	0.43731	0.50556	0.359572	0.352251	0.301535	0.825823

```
[5 rows x 131 columns]
```

[6]:
```
# Create dataframe for categorical variables
cat_df = train_df.select_dtypes(include = ["object"])

# Create dataframe for continuous variables and drop loss and id columns
cont_df = train_df.select_dtypes(include = ["float64", "int64"])
cont_df = cont_df.drop(columns = ['id'])
```

[7]:
```
cont_df = cont_df.drop(columns = ['loss'])
```

[8]:
```
print(cat_df.shape)
print(cont_df.shape)
```

```
(188318, 116)
(188318, 14)
```

[9]:
```
list(cat_df["cat1"].unique())
```

[9]:
```
['A', 'B']
```

[10]:
```
# Create dataframe for continuous variables and drop id columns in test data
test_cont_df = test_df.select_dtypes(include = ["float64", "int64"])
test_cont_df = test_cont_df.drop(columns = ['id'])
print(test_cont_df.shape)
```

```
(125546, 14)
```

3.1 Label Encoding

[11]:
```
# Import LabelEncoder from sklearn.preprocessing
from sklearn.preprocessing import LabelEncoder

# Iterate through each category column and convert to numeric using␣
 ↪LabelEncoder. Then transform the column
# and assign back to the original column
for key in cat_df.columns:
    le = LabelEncoder()
    labels = list(train_df[key].unique())
    labels += list(test_df[key].unique())

    # Create mapping from labels to integers
    le.fit(labels)
    # Transform the train and test consistently
    train_df[key] = le.transform(train_df[key])
    test_df[key] = le.transform(test_df[key])
```

3.1.1 Correlation between continuous columns and loss

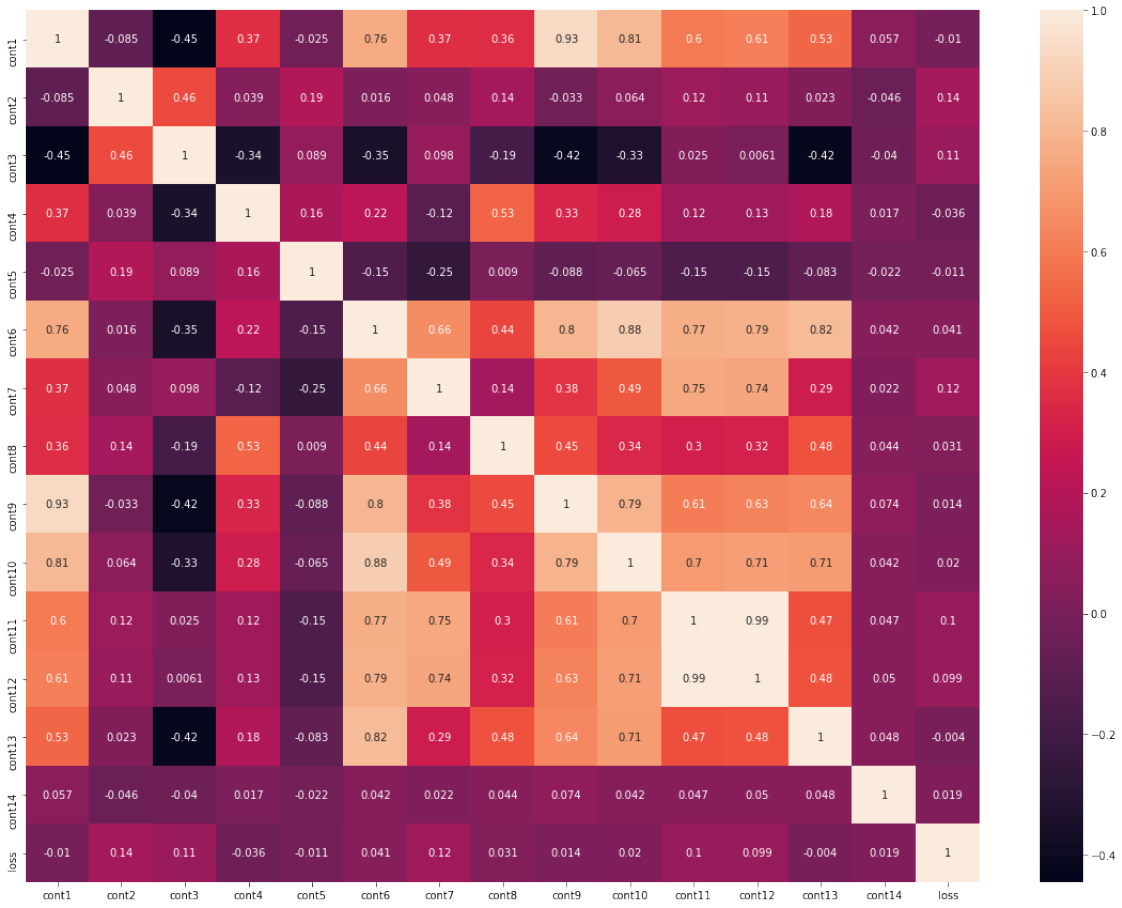
[12]:
```
corr_df = train_df.copy()
corr_df.drop(columns = cat_df.columns, inplace = True)
corr_df.drop(columns = ["id"], inplace = True)
corr_df
```

[12]:

	cont1	cont2	cont3	cont4	cont5	cont6	cont7	\
0	0.726300	0.245921	0.187583	0.789639	0.310061	0.718367	0.335060	
1	0.330514	0.737068	0.592681	0.614134	0.885834	0.438917	0.436585	
2	0.261841	0.358319	0.484196	0.236924	0.397069	0.289648	0.315545	
3	0.321594	0.555782	0.527991	0.373816	0.422268	0.440945	0.391128	

```
4       0.273204  0.159990  0.527991  0.473202  0.704268  0.178193  0.247408
…            …         …         …         …         …         …         …
188313  0.347403  0.785784  0.613660  0.473202  0.939556  0.242437  0.289949
188314  0.507661  0.555782  0.549770  0.802892  0.704268  0.334270  0.382000
188315  0.484469  0.785784  0.792378  0.189137  0.482436  0.345883  0.370534
188316  0.438385  0.422197  0.298977  0.383428  0.340543  0.704364  0.562866
188317  0.907272  0.620805  0.440642  0.821574  0.281143  0.844563  0.533048

          cont8    cont9   cont10    cont11    cont12    cont13    cont14  \
0       0.30260  0.67135  0.83510  0.569745  0.594646  0.822493  0.714843
1       0.60087  0.35127  0.43919  0.338312  0.366307  0.611431  0.304496
2       0.27320  0.26076  0.32446  0.381398  0.373424  0.195709  0.774425
3       0.31796  0.32128  0.44467  0.327915  0.321570  0.605077  0.602642
4       0.24564  0.22089  0.21230  0.204687  0.202213  0.246011  0.432606
…           …        …        …         …         …         …         …
188313  0.24564  0.30859  0.32935  0.223038  0.220003  0.333292  0.208216
188314  0.63475  0.40455  0.47779  0.307628  0.301921  0.318646  0.305872
188315  0.24564  0.45808  0.47779  0.445614  0.443374  0.339244  0.503888
188316  0.34987  0.44767  0.53881  0.863052  0.852865  0.654753  0.721707
188317  0.97123  0.93383  0.83814  0.932195  0.946432  0.810511  0.721460

          loss
0       2213.18
1       1283.60
2       3005.09
3        939.85
4       2763.85
…           …
188313  1198.62
188314  1108.34
188315  5762.64
188316  1562.87
188317  4751.72

[188318 rows x 15 columns]
```

[13]:
```
import seaborn as sns
plt.subplots(figsize=(20,15))
sns.heatmap(corr_df.corr(), annot = True)
plt.show()
```



3.1.2 Correlation between significant categorical columns and loss

[14]:
```
df_list = []
col_to_remove_list = list(cat_df.columns)

for col in cat_df.columns:
    if len(cat_df[col].unique()) >= 5:
        df_list.append(col)
        col_to_remove_list.remove(col)

print(len(df_list))
print(len(col_to_remove_list))
```

```
28
88
```

[15]:
```
corr_df = train_df.copy()
corr_df.drop(columns = col_to_remove_list, inplace = True)
corr_df.drop(columns = cont_df.columns, inplace = True)
```

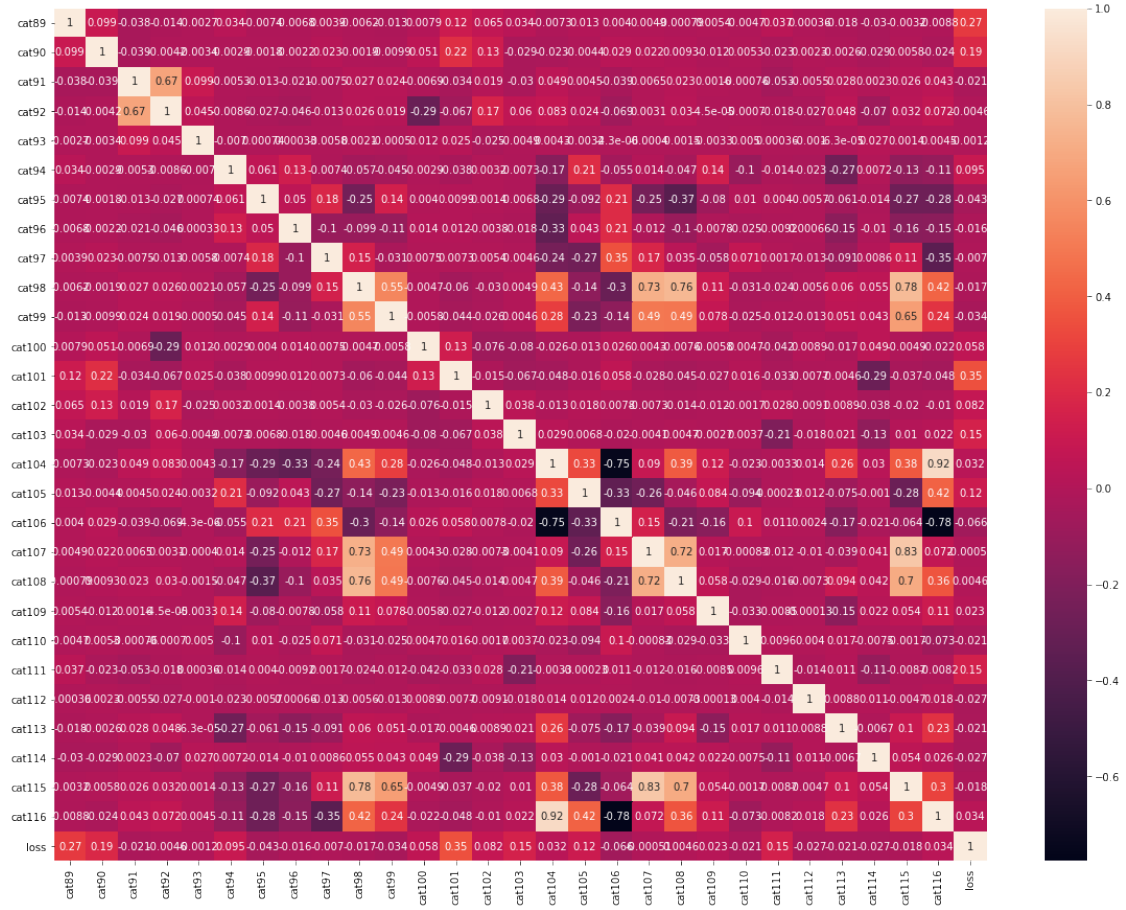
```
corr_df.drop(columns = ["id"], inplace = True)
corr_df
```

[15]:

```
        cat89  cat90  cat91  cat92  cat93  cat94  cat95  cat96  cat97  cat98  \
0           0      0      0      0      3      1      2      4      0      2
1           0      0      0      0      3      3      2      4      4      3
2           0      0      0      0      3      3      2      4      4      0
3           0      0      0      0      3      3      2      4      4      3
4           0      0      1      7      3      1      3      4      4      0
…         …    …    …    …    …    …    …    …    …    …
188313      0      0      0      0      3      3      3      4      4      0
188314      0      0      0      0      3      3      3      4      2      0
188315      0      0      1      7      3      3      3      4      2      0
188316      0      0      0      0      2      3      2      4      2      3
188317      0      0      4      1      3      3      2      4      0      2

        …  cat108  cat109  cat110  cat111  cat112  cat113  cat114  cat115  \
0       …       6      46      28       2      19      57       0      14
1       …      10      34      67       0      22      39       0      14
2       …       0       2      87       0      28       6       0       8
3       …      10      34      69       2      39       5       0      14
4       …       1      67      51       2      50      39       0      10
…     …    …    …    …    …    …    …    …    …
188313  …       1      34     108       0      32      39       0      11
188314  …       1      34      45       0      22      21       4       9
188315  …       1      34      89       0      48       6       0      10
188316  …      10      34       9       0      19      10       0      14
188317  …       6      34     108       8      19       4       0      14

        cat116     loss
0          288  2213.18
1           94  1283.60
2          167  3005.09
3           88   939.85
4           63  2763.85
…         …    …
188313      63  1198.62
188314      84  1108.34
188315      88  5762.64
188316     316  1562.87
188317     322  4751.72

[188318 rows x 29 columns]
```

[16]:

```
plt.subplots(figsize=(20,15))
sns.heatmap(corr_df.corr(), annot = True)
plt.show()
```



3.1.3 Normalize categorical columns

[17]:
```
# Import MinMaxScaler from sklearn.preprocessing and ColumnTransformer from
 ↪sklearn.compose
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer

# Normalize the values in the columns of the categorical dataframe
minmax_transformer = Pipeline(steps=[('minmax', MinMaxScaler())])
preprocessor = ColumnTransformer(
        remainder='passthrough', # passthough features not listed
        transformers=[('mm', minmax_transformer, cat_df.columns)])

preprocessor.fit(train_df.drop(columns = ['id', 'loss']))

# Create an array containing the normalized values for both the train and the
 ↪test
norm_train = preprocessor.transform(train_df.drop(columns = ['id', 'loss']))
```

```
norm_test = preprocessor.transform(test_df.drop(columns = ['id']))
```

[18]:
```
print(norm_train.shape)
```

```
(188318, 130)
```

[19]:
```
norm_train = np.c_[norm_train, train_df['loss'].values]
print(norm_train.shape)
```

```
(188318, 131)
```

[20]:
```
print(norm_test.shape)
```

```
(125546, 130)
```

[21]:
```
train_df['loss'].values.shape
```

[21]:
```
(188318,)
```

[22]:
```
train_df.columns.drop(['id'])
```

[22]:
```
Index(['cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7', 'cat8', 'cat9',
       'cat10',
       …
       'cont6', 'cont7', 'cont8', 'cont9', 'cont10', 'cont11', 'cont12',
       'cont13', 'cont14', 'loss'],
      dtype='object', length=131)
```

[23]:
```
# Convert the array containing the normalized values to a dataframe
train_numeric_df = pd.DataFrame(data = norm_train, index = train_df.index,
                                columns = train_df.columns.drop(['id']))
print(train_numeric_df)

test_numeric_df = pd.DataFrame(data = norm_test, index = test_df.index,
                               columns = test_df.columns.drop(['id']))
print(test_numeric_df)
```

```
        cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10  …  \
0        0.0   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0    0.0  …
1        0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0    1.0  …
2        0.0   1.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0    1.0  …
3        1.0   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0    0.0  …
4        0.0   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0    1.0  …
…       …   …   …   …   …   …   …   …   …   …  …
188313   0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0    0.0  …
188314   0.0   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0    0.0  …
188315   0.0   1.0   0.0   0.0   0.0   0.0   0.0   1.0   1.0    0.0  …
188316   0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0    1.0  …
188317   1.0   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0    0.0  …
```

```
           cont6     cont7    cont8    cont9   cont10    cont11    cont12  \
0       0.718367  0.335060  0.30260  0.67135  0.83510  0.569745  0.594646
1       0.438917  0.436585  0.60087  0.35127  0.43919  0.338312  0.366307
2       0.289648  0.315545  0.27320  0.26076  0.32446  0.381398  0.373424
3       0.440945  0.391128  0.31796  0.32128  0.44467  0.327915  0.321570
4       0.178193  0.247408  0.24564  0.22089  0.21230  0.204687  0.202213
…            …         …       …        …        …         …
188313  0.242437  0.289949  0.24564  0.30859  0.32935  0.223038  0.220003
188314  0.334270  0.382000  0.63475  0.40455  0.47779  0.307628  0.301921
188315  0.345883  0.370534  0.24564  0.45808  0.47779  0.445614  0.443374
188316  0.704364  0.562866  0.34987  0.44767  0.53881  0.863052  0.852865
188317  0.844563  0.533048  0.97123  0.93383  0.83814  0.932195  0.946432

          cont13    cont14     loss
0       0.822493  0.714843  2213.18
1       0.611431  0.304496  1283.60
2       0.195709  0.774425  3005.09
3       0.605077  0.602642   939.85
4       0.246011  0.432606  2763.85
…            …         …       …
188313  0.333292  0.208216  1198.62
188314  0.318646  0.305872  1108.34
188315  0.339244  0.503888  5762.64
188316  0.654753  0.721707  1562.87
188317  0.810511  0.721460  4751.72

[188318 rows x 131 columns]
        cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10  …  \
0        0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0    0.0  …
1        0.0   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0    0.0  …
2        0.0   1.0   0.0   1.0   1.0   0.0   1.0   0.0   1.0    1.0  …
3        0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0   0.0    0.0  …
4        1.0   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0    0.0  …
…      …   …     …     …     …     …     …   …     …     …    …  …
125541   0.0   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0    0.0  …
125542   0.0   0.0   0.0   0.0   1.0   1.0   0.0   1.0   0.0    0.0  …
125543   1.0   1.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0    1.0  …
125544   0.0   0.0   0.0   0.0   0.0   1.0   0.0   1.0   0.0    0.0  …
125545   0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0    0.0  …

           cont5     cont6     cont7    cont8    cont9   cont10    cont11  \
0       0.281143  0.466591  0.317681  0.61229  0.34365  0.38016  0.377724
1       0.836443  0.482425  0.443760  0.71330  0.51890  0.60401  0.689039
2       0.718531  0.212308  0.325779  0.29758  0.34365  0.30529  0.245410
3       0.397069  0.369930  0.342355  0.40028  0.33237  0.31480  0.348867
4       0.302678  0.398862  0.391833  0.23688  0.43731  0.50556  0.359572
…            …         …         …        …        …        …
125541  0.281143  0.438917  0.815941  0.39455  0.48740  0.40666  0.550529
```

```
125542  0.674529  0.346948  0.424968  0.47669  0.25753  0.26894  0.324486
125543  0.794794  0.808958  0.511502  0.72299  0.94438  0.83510  0.933174
125544  0.302678  0.372125  0.388545  0.31796  0.32128  0.36974  0.307628
125545  0.413817  0.221699  0.242044  0.25461  0.31399  0.25183  0.245410

          cont12    cont13    cont14
0       0.369858  0.704052  0.392562
1       0.675759  0.453468  0.208045
2       0.241676  0.258586  0.297232
3       0.341872  0.592264  0.555955
4       0.352251  0.301535  0.825823
…            …         …         …
125541  0.538473  0.298734  0.345946
125542  0.352251  0.490001  0.290576
125543  0.926619  0.848129  0.808125
125544  0.301921  0.608259  0.361542
125545  0.241676  0.287682  0.220323

[125546 rows x 130 columns]
```

3.2 Add bias

[24]:
```
train_numeric_df.insert(1, "bias", 1)
```

[25]:
```
train_numeric_df
```

[25]:
```
         cat1  bias  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  …  \
0         0.0     1   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0  …
1         0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0  …
2         0.0     1   1.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0  …
3         1.0     1   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0  …
4         0.0     1   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0  …
…       …   …   …   …   …   …   …   …   …   …  …
188313    0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0  …
188314    0.0     1   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0  …
188315    0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   1.0   1.0  …
188316    0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0  …
188317    1.0     1   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0  …

            cont6     cont7    cont8    cont9   cont10    cont11    cont12  \
0        0.718367  0.335060  0.30260  0.67135  0.83510  0.569745  0.594646
1        0.438917  0.436585  0.60087  0.35127  0.43919  0.338312  0.366307
2        0.289648  0.315545  0.27320  0.26076  0.32446  0.381398  0.373424
3        0.440945  0.391128  0.31796  0.32128  0.44467  0.327915  0.321570
4        0.178193  0.247408  0.24564  0.22089  0.21230  0.204687  0.202213
…             …         …        …        …        …         …         …
188313   0.242437  0.289949  0.24564  0.30859  0.32935  0.223038  0.220003
188314   0.334270  0.382000  0.63475  0.40455  0.47779  0.307628  0.301921
```

```
188315  0.345883  0.370534  0.24564  0.45808  0.47779  0.445614  0.443374
188316  0.704364  0.562866  0.34987  0.44767  0.53881  0.863052  0.852865
188317  0.844563  0.533048  0.97123  0.93383  0.83814  0.932195  0.946432

          cont13    cont14     loss
0       0.822493  0.714843  2213.18
1       0.611431  0.304496  1283.60
2       0.195709  0.774425  3005.09
3       0.605077  0.602642   939.85
4       0.246011  0.432606  2763.85
…            …         …       …
188313  0.333292  0.208216  1198.62
188314  0.318646  0.305872  1108.34
188315  0.339244  0.503888  5762.64
188316  0.654753  0.721707  1562.87
188317  0.810511  0.721460  4751.72

[188318 rows x 132 columns]
```

[26]:
```
train_numeric_df.dtypes
```

[26]:
```
cat1      float64
bias        int64
cat2      float64
cat3      float64
cat4      float64
           …
cont11    float64
cont12    float64
cont13    float64
cont14    float64
loss      float64
Length: 132, dtype: object
```

[27]:
```
test_numeric_df.insert(1, "bias", 1)
```

[28]:
```
test_numeric_df
```

[28]:
```
        cat1  bias  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  …  \
0        0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0  …
1        0.0     1   1.0   0.0   1.0   0.0   0.0   0.0   0.0   1.0  …
2        0.0     1   1.0   0.0   1.0   1.0   0.0   1.0   0.0   1.0  …
3        0.0     1   0.0   0.0   0.0   1.0   0.0   0.0   0.0   0.0  …
4        1.0     1   0.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0  …
…      …   …   …   …   …   …   …   …   …   …   …
125541   0.0     1   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0  …
125542   0.0     1   0.0   0.0   0.0   1.0   1.0   0.0   1.0   0.0  …
```

```
125543   1.0     1   1.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0  …
125544   0.0     1   0.0   0.0   0.0   0.0   1.0   0.0   1.0   0.0  …
125545   0.0     1   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0  …

           cont5     cont6     cont7    cont8    cont9   cont10    cont11  \
0       0.281143  0.466591  0.317681  0.61229  0.34365  0.38016  0.377724
1       0.836443  0.482425  0.443760  0.71330  0.51890  0.60401  0.689039
2       0.718531  0.212308  0.325779  0.29758  0.34365  0.30529  0.245410
3       0.397069  0.369930  0.342355  0.40028  0.33237  0.31480  0.348867
4       0.302678  0.398862  0.391833  0.23688  0.43731  0.50556  0.359572
…            …         …         …        …        …        …         …
125541  0.281143  0.438917  0.815941  0.39455  0.48740  0.40666  0.550529
125542  0.674529  0.346948  0.424968  0.47669  0.25753  0.26894  0.324486
125543  0.794794  0.808958  0.511502  0.72299  0.94438  0.83510  0.933174
125544  0.302678  0.372125  0.388545  0.31796  0.32128  0.36974  0.307628
125545  0.413817  0.221699  0.242044  0.25461  0.31399  0.25183  0.245410

           cont12    cont13    cont14
0       0.369858  0.704052  0.392562
1       0.675759  0.453468  0.208045
2       0.241676  0.258586  0.297232
3       0.341872  0.592264  0.555955
4       0.352251  0.301535  0.825823
…            …         …         …
125541  0.538473  0.298734  0.345946
125542  0.352251  0.490001  0.290576
125543  0.926619  0.848129  0.808125
125544  0.301921  0.608259  0.361542
125545  0.241676  0.287682  0.220323

[125546 rows x 131 columns]
```

[29]:
```
test_numeric_df.dtypes
```

[29]:
```
cat1      float64
bias        int64
cat2      float64
cat3      float64
cat4      float64
           …
cont10    float64
cont11    float64
cont12    float64
cont13    float64
cont14    float64
Length: 131, dtype: object
```

4 Step 3 - Creating arrays for the features and the response variable.

[30]:
```
# Take the loss column and set it as the target column since the loss variable␣
 ↪is what is being predicted
target_column = ['loss']
# Create the list of predictors variables
predictors = list(set(list(train_numeric_df.columns))-set(target_column))
```

5 Step 4 - Creating the Training and Test Datasets

[31]:
```
train_numeric_df.describe()
```

[31]:

	cat1	bias	cat2	cat3	cat4	\
count	188318.000000	188318.0	188318.000000	188318.000000	188318.000000	
mean	0.248346	1.0	0.433294	0.054827	0.318201	
std	0.432055	0.0	0.495532	0.227644	0.465779	
min	0.000000	1.0	0.000000	0.000000	0.000000	
25%	0.000000	1.0	0.000000	0.000000	0.000000	
50%	0.000000	1.0	0.000000	0.000000	0.000000	
75%	0.000000	1.0	1.000000	0.000000	1.000000	
max	1.000000	1.0	1.000000	1.000000	1.000000	

	cat5	cat6	cat7	cat8	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	0.342936	0.300688	0.024289	0.058645	
std	0.474692	0.458559	0.153944	0.234961	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	1.000000	1.000000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

	cat9	…	cont6	cont7	cont8	\
count	188318.000000	…	188318.000000	188318.000000	188318.000000	
mean	0.399303	…	0.490945	0.484970	0.486437	
std	0.489757	…	0.205273	0.178450	0.199370	
min	0.000000	…	0.012683	0.069503	0.236880	
25%	0.000000	…	0.336105	0.350175	0.312800	
50%	0.000000	…	0.440945	0.438285	0.441060	
75%	1.000000	…	0.655021	0.591045	0.623580	
max	1.000000	…	0.997162	1.000000	0.980200	

	cont9	cont10	cont11	cont12	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	0.485506	0.498066	0.493511	0.493150	

```
std         0.181660       0.185877       0.209737       0.209427
min         0.000080       0.000000       0.035321       0.036232
25%         0.358970       0.364580       0.310961       0.311661
50%         0.441450       0.461190       0.457203       0.462286
75%         0.566820       0.614590       0.678924       0.675759
max         0.995400       0.994980       0.998742       0.998484

              cont13         cont14           loss
count  188318.000000  188318.000000  188318.000000
mean        0.493138       0.495717    3037.337686
std         0.212777       0.222488    2904.086186
min         0.000228       0.179722       0.670000
25%         0.315758       0.294610    1204.460000
50%         0.363547       0.407403    2115.570000
75%         0.689974       0.724623    3864.045000
max         0.988494       0.844848  121012.250000

[8 rows x 132 columns]
```

5.1 Polynomial Features for Training Set

[32]:
```
from sklearn.preprocessing import PolynomialFeatures

# If poly_features is True, create the polynomial features for the continuous␣
 ↪variables in the training set
if poly_features:
    poly_features_cont = PolynomialFeatures(degree = 2, include_bias = False)
    # Fit the polynomial features
    poly_features_cont.fit(cont_df.values)

    X_poly_train_cont = poly_features_cont.transform(cont_df.values)
```

[33]:
```
if poly_features:
    print(X_poly_train_cont.shape)
```

```
(188318, 119)
```

[34]:
```
"id" in predictors
```

[34]:
```
False
```

[35]:
```
len(predictors)
```

[35]:
```
131
```

[36]:
```
if poly_features:
    X = np.c_[train_numeric_df[predictors].values, X_poly_train_cont]
else:
```

```
    X = train_numeric_df[predictors].values

y = train_numeric_df[target_column].values

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.20)
print(X_train.shape)
print(X_val.shape)
```

```
(150654, 250)
(37664, 250)
```

5.2 Polynomial Features for Test Set

[37]:
```
if poly_features:
    X_poly_test_cont = poly_features_cont.transform(test_cont_df.values)
```

[38]:
```
if poly_features:
    print(X_poly_test_cont.shape)
```

```
(125546, 119)
```

[39]:
```
if poly_features:
    X_submission = np.c_[test_numeric_df[predictors].values, X_poly_test_cont]
else:
    X_submission = test_numeric_df[predictors].values
```

[40]:
```
X_submission.shape
```

[40]:
```
(125546, 250)
```

6 Step 5 - Build, Predict and Evaluate the Regression Models

7 Linear regression

[41]:
```
# Function to evaluate different models for different values of alpha.
def run_model(model_class, alphas, **model_kargs):
    for alpha in alphas:
        model = model_class(alpha = alpha, **model_kargs) if alpha > 0 else␣
 ↪LinearRegression()
        model.fit(X_train, y_train)
        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

        print(f'alpha = {alpha}')
        print("Evaluation metrics, MAE and R-squared, for the training data:")
        print(mean_absolute_error(y_train, y_train_pred))
        print(r2_score(y_train, y_train_pred))
```

```
        print("Evaluation metrics, MAE and R-squared, for the validation set:")
        print(mean_absolute_error(y_val, y_val_pred))
        print(r2_score(y_val, y_val_pred))

        print("\n")
```

[42]:
```
# instantiate the LinearRegression() algorithm
lin_reg = LinearRegression()

# fit the model on the training set.
lin_reg.fit(X_train, y_train)
```

[42]:
```
LinearRegression()
```

[43]:
```
from sklearn.metrics import mean_absolute_error
```

[44]:
```
# predict on the training set.
pred_train_lin_reg = lin_reg.predict(X_train)

print("Evaluation metrics, MAE and R-squared, for the training set:")
print(mean_absolute_error(y_train, pred_train_lin_reg))
print(r2_score(y_train, pred_train_lin_reg))
```

```
Evaluation metrics, MAE and R-squared, for the training set:
1321.1505849728258
0.48912726415158403
```

[45]:
```
# predict on the validation set.
pred_test_lin_reg = lin_reg.predict(X_val)

print("Evaluation metrics, MAE and R-squared, for the validation set:")
print(mean_absolute_error(y_val, pred_test_lin_reg))
print(r2_score(y_val, pred_test_lin_reg))
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1326.2096104440227
0.4939452016753376
```

8 Ridge Regression

[49]:
```
%time run_model(Ridge, alphas=(0.01, 0.1, 1), solver="cholesky",␣
↪random_state=42)
```

```
alpha = 0.01
Evaluation metrics, MAE and R-squared, for the training data:
1321.0737916353219
0.4891225595307598
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1326.1519933055004
0.4939777891609708


alpha = 0.1
Evaluation metrics, MAE and R-squared, for the training data:
1320.907927896929
0.48906355403066715
Evaluation metrics, MAE and R-squared, for the validation set:
1326.0214944517809
0.49403307844380306


alpha = 1
Evaluation metrics, MAE and R-squared, for the training data:
1320.5158970713874
0.4889372657785953
Evaluation metrics, MAE and R-squared, for the validation set:
1325.5308492986546
0.4941016028486831


CPU times: user 4.85 s, sys: 785 ms, total: 5.64 s
Wall time: 970 ms
```

[46]:
```
# instantiate the Ridge Regression model with an alpha value of 0.01
model_ridge = Ridge(alpha = 1, solver="cholesky", random_state=42)

# fit the model to the training data.
model_ridge.fit(X_train, y_train)
```

[46]:
```
Ridge(alpha=1, random_state=42, solver='cholesky')
```

[47]:
```
# Predict on the training data
pred_train_ridge = model_ridge.predict(X_train)

print("Evaluation metrics, MAE and R-squared, for the training set:")
print(mean_absolute_error(y_train, pred_train_ridge))
print(r2_score(y_train, pred_train_ridge))
```

```
Evaluation metrics, MAE and R-squared, for the training set:
1320.5158970713874
0.4889372657785953
```

[48]:
```
# Predict on the validation data
pred_test_ridge = model_ridge.predict(X_val)
```

```
print("Evaluation metrics, MAE and R-squared, for the validation set:")
print(mean_absolute_error(y_val, pred_test_ridge))
print(r2_score(y_val, pred_test_ridge))
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1325.5308492986546
0.4941016028486831
```

9 Lasso Regression

[69]:
```
%time run_model(Lasso, alphas=(0, 0.01, 0.1, 0.5, 1, 1.5, 2, 5, 10),␣
↪random_state=42)
```

```
alpha = 0
Evaluation metrics, MAE and R-squared, for the training data:
1321.1505849728258
0.48912726415158403
Evaluation metrics, MAE and R-squared, for the validation set:
1326.2096104440227
0.4939452016753376
```

```
/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 265148281668.37112, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(
```

```
alpha = 0.01
Evaluation metrics, MAE and R-squared, for the training data:
1320.7719408292025
0.4887417585912073
Evaluation metrics, MAE and R-squared, for the validation set:
1325.443559414626
0.4941641202505742
```

```
/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 56229619022.9035, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(
```

```
alpha = 0.1
Evaluation metrics, MAE and R-squared, for the training data:
1319.7086643295384
0.4882350389432545
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1324.1222103267837
0.4942859251535121



/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1037649963.144043, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(

alpha = 0.5
Evaluation metrics, MAE and R-squared, for the training data:
1320.112167119926
0.48580297516950166
Evaluation metrics, MAE and R-squared, for the validation set:
1323.8085232771136
0.4932782926487893



alpha = 1
Evaluation metrics, MAE and R-squared, for the training data:
1321.3951017768484
0.4835882693321928
Evaluation metrics, MAE and R-squared, for the validation set:
1324.8512646473368
0.49179833163167075



alpha = 1.5
Evaluation metrics, MAE and R-squared, for the training data:
1323.088686005294
0.4817165419657812
Evaluation metrics, MAE and R-squared, for the validation set:
1326.4434888788105
0.49027325630674057



alpha = 2
Evaluation metrics, MAE and R-squared, for the training data:
1325.1298392840663
0.4798207389465914
Evaluation metrics, MAE and R-squared, for the validation set:
1328.3955520770803
0.48857535316857137



alpha = 5
```

```
Evaluation metrics, MAE and R-squared, for the training data:
1328.6997308268421
0.4742545134063959
Evaluation metrics, MAE and R-squared, for the validation set:
1332.2260300362022
0.4829724232609113


alpha = 10
Evaluation metrics, MAE and R-squared, for the training data:
1333.5820762491555
0.46777287746356677
Evaluation metrics, MAE and R-squared, for the validation set:
1337.5575005297549
0.47575939109667453


CPU times: user 6min 37s, sys: 22.1 s, total: 6min 59s
Wall time: 58.9 s
```

[70]:
```
model_lasso = Lasso(alpha=0.1, max_iter = 5000)
%time model_lasso.fit(X_train, y_train)
```

```
CPU times: user 9min 48s, sys: 30.1 s, total: 10min 18s
Wall time: 1min 24s

/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1909968027.9354248, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(
```

[70]:
```
Lasso(alpha=0.1, max_iter=5000)
```

[71]:
```
# Predict on the training data
pred_train_lasso = model_lasso.predict(X_train)

print("Evaluation metrics, MAE and R-squared, for the training set:")
print(mean_absolute_error(y_train, pred_train_lasso))
print(r2_score(y_train, pred_train_lasso))
```

```
Evaluation metrics, MAE and R-squared, for the training set:
1319.7271537305896
0.4881887650963205
```

[72]:
```
# Predict on the validation data
pred_test_lasso= model_lasso.predict(X_val)

print("Evaluation metrics, MAE and R-squared, for the validation set:")
```

```
print(mean_absolute_error(y_val, pred_test_lasso))
print(r2_score(y_val, pred_test_lasso))
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1324.0785597432318
0.4942579367809906
```

10 ElasticNet Regression

[57]:
```
%time run_model(ElasticNet, alphas=(0.005, 0.05, 0.1), random_state=42)
```

```
/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 322084687092.5278, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(

alpha = 0.005
Evaluation metrics, MAE and R-squared, for the training data:
1322.6663361910864
0.48258975328311016
Evaluation metrics, MAE and R-squared, for the validation set:
1326.0199614617757
0.49054076348354536


alpha = 0.05
Evaluation metrics, MAE and R-squared, for the training data:
1340.6205339234416
0.4633733687314753
Evaluation metrics, MAE and R-squared, for the validation set:
1343.9685120666609
0.4715438155070234


alpha = 0.1
Evaluation metrics, MAE and R-squared, for the training data:
1361.8323504601044
0.4450488892605431
Evaluation metrics, MAE and R-squared, for the validation set:
1365.4387918385653
0.4528480390269626


CPU times: user 5min 25s, sys: 20.9 s, total: 5min 46s
Wall time: 48.9 s
```

[73]:
```
model_elastic_net = ElasticNet(alpha = 0.005, max_iter = 2500)
model_elastic_net.fit(X_train, y_train)
```

```
/Users/anaswarjayakumar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 290288990056.2658, tolerance: 127080421.81718636
  model = cd_fast.enet_coordinate_descent(
```

[73]:
```
ElasticNet(alpha=0.005, max_iter=2500)
```

[74]:
```
# Predict on the training data
pred_train_elastic_net = model_elastic_net.predict(X_train)

print("Evaluation metrics, MAE and R-squared, for the training set:")
print(mean_absolute_error(y_train, pred_train_elastic_net))
print(r2_score(y_train, pred_train_elastic_net))
```

```
Evaluation metrics, MAE and R-squared, for the training set:
1322.6578928754896
0.48260409459772935
```

[75]:
```
# Predict on the validation data
pred_test_elastic_net = model_elastic_net.predict(X_val)

print("Evaluation metrics, MAE and R-squared, for the validation set:")
print(mean_absolute_error(y_val, pred_test_elastic_net))
print(r2_score(y_val, pred_test_elastic_net))
```

```
Evaluation metrics, MAE and R-squared, for the validation set:
1325.9956201481602
0.49054805601806506
```

11 XGBoost Regressor

The linear regression, the ridge, lasso and ElasticNet didnt provide great results even with the polynomial features which is why I tried the XGBRegressor using the same features. The XGBRegressor seems to provide better results using the same features but it was slower to train

[58]:
```
import xgboost
from xgboost import XGBRegressor

xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05, subsample=0.5)
%time xgb_reg.fit(X_train, y_train.ravel())
```

```
CPU times: user 32min 18s, sys: 4.5 s, total: 32min 22s
Wall time: 36min 32s
```

[58]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=1, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.5,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[66]:
```
%time pred_train_xgb = xgb_reg.predict(X_train)

print("Evaluation metrics, MAE and R-squared, for the training set:")
print(mean_absolute_error(y_train, pred_train_xgb))
print(r2_score(y_train, pred_train_xgb))
```

```
CPU times: user 2.59 s, sys: 8.02 ms, total: 2.6 s
Wall time: 2.6 s
Evaluation metrics, MAE and R-squared, for the training set:
1041.5109971093416
0.7326851364557081
```

[67]:
```
%time pred_test_xgb = xgb_reg.predict(X_val)

print("Evaluation metrics, MAE and R-squared, for the test set:")
print(mean_absolute_error(y_val, pred_test_xgb))
print(r2_score(y_val, pred_test_xgb))
```

```
CPU times: user 676 ms, sys: 3.03 ms, total: 679 ms
Wall time: 677 ms
Evaluation metrics, MAE and R-squared, for the test set:
1190.2092494964113
0.555915672990569
```

11.1 Submission

[172]:
```
test_numeric_df[predictors].values.shape
```

[172]:
```
(125546, 131)
```

[174]:
```
%time submission_xgb = xgb_reg.predict(X_submission)
```

```
CPU times: user 2.19 s, sys: 9.65 ms, total: 2.2 s
Wall time: 2.2 s
```

[175]:
```
submission_xgb.shape
```

[175]:
```
(125546,)
```

[176]:
```
test_df["id"].values.shape
```

[176]: (125546,)

[177]:
```
submission_data = np.c_[test_df["id"].values, submission_xgb]
```

[178]:
```
submission_df = pd.DataFrame(data = submission_data, columns = ["id", "loss"])
```

[179]:
```
submission_df['id'] = submission_df['id'].astype('int64')
```

[180]:
```
submission_df
```

[180]:

	id	loss
0	4	1741.572632
1	6	1839.869019
2	9	9740.742188
3	12	6144.730957
4	15	920.303589
…	…	…
125541	587617	2820.655762
125542	587621	2768.545898
125543	587627	2758.085205
125544	587629	1176.777100
125545	587634	3616.588379

[125546 rows x 2 columns]

[181]:
```
submission_df.to_csv("/Users/anaswarjayakumar/Downloads/submission.csv", index␣
↪= False)
```

12 Conclusion

12.1 Data preparation, exploration, visualization

I tried both One-Hot encoding and Label encoding for the categorical columns. Label Encoding simply converts each value in a column to a number and One-Hot Encoding converts each value in a column to a number and every unique value in the category will be added as a feature. Min-Max normalization was performed on the categorical columns but not on the continuous columns as they were already scaled to fit the range [0,1]. In addition, the id and loss columns were removed and the polynomial features were obtained for the continuous columns but not for the categorical columns because of memory issues.

12.2 Research Design/Review results, evaluate models

In this assignment, five methods were used, Linear Regression, Ridge Regression, Lasso Regression, ElasticNet Regression, and XGBoost for Regression (XGBRegressor). Compared to the Linear Regression, Ridge Regression, and Lasso Regression, ElasticNet Regression seemed to be the best, however the XGBREgressor generated the best MAE score and the best score when submitted to Kaggle. In addition, the run_model function was used to find the best alpha via cross validation

and that alpha was used for the Linear, Ridge, Lasso, and ElasticNet Regressions. I couldnt do the cross validation for the XGBRegressor because it was taking much longer to train

I tried both One-Hot encoding and Label encoding for the categorical columns. One-Hot encoding performed better for the linear algorithms such as the Lasso, Ridge and ElasticNet with an MAE of around 1280 for the training data. I also tried generating interaction features for the categorical data. However, the number of features was too much and I was running into memory issues. So I only considered polynomial features for the continuous data. I decided to stick with label encoding since that provided the optimal model for the XGBRegressor. I also briefly tried the SGDRegressor prior to switching to the XGBRegressor since it was preferred by many Kagglers. The ElasticNet model gave me the smallest MAE for the training set. The MAE of the XGBRegressor for the same features was significantly lower at 1041. The MAE of the validation data for the XGBRegressor was significantly higher at 1190, indicating some degree of overfitting to the training data. Lastly, varying the value of alpha had a small impact on the MAE and R-Squared. Once I got a suitable value for alpha, I then tried running the algorithms with more iterations to see if it would converge.

Some Kagglers were using the LGBMRegressor instead of the XGBRegressor. Initial research seems to indicate that the LGBMRegressor is much faster than the XGBRegressor. I did not get a chance to test this. Generating all of the interaction features for the categorical variables creates a "curse of dimensionality" and it is best for models to be simpler rather than complex. With more time, I could have included specific interaction features based on the correlation of the categorical and continuous variables

12.3 Exposition and Management Recommendations

The results in Kaggle were very good which means there is still more work to be done in terms of feature generation. I should consider combinations of different features to help improve the results. However, this is an exponential problem. One way to reduce the scale of this problem is to be strategic when creating these combinations. One potential approach is to consider combinations of features that are somewhat positively correlated to each other.

[ ]: