Background on Data

For each insurance claim provided in the Allstate Claims Severity Competition, there are a numb er of categorical variables as well as continuous variables that contribute to the severity of a filed claim. Training data for claims consisted of 188,318 claims (unique claim numbers) with 116 cat egorical variables and 14 continuous variables (Figure 4).

Data Preparation

As it pertains to data preparation for the training set, I used a one hot encoder to change the alphabetical nature of the categorical variables to represent binary values (Figure 6). Initially, I considered conducting a PCA analysis on the training set to determine how many features truly explain the spread of the data. However, after closely observing the distribution of the 'loss column', I decided that this step would exclude potential outliers that would prove to be crucial for the regression methods to be employed when teaching the model (Figure 5). I then created variable X, which would represent the training data to be used and dropped certain features (i.e. 'ID' and 'loss') that would be insignificant in the predictions to be made. The y variable represented the 'loss' feature, exclusively which is our variable to be predicted. I then created a function that would first test regression methods such as Elastic Net, Lasso and Ridge regression, then it would perform cross validation to produce the Mean Absolute Error (MAE) associated with each regression method (Figure 7).

Results/Evaluation

As a result of the 'perform_reg' function created, I was able to test the aforementioned regression models and determine their MAEs, respectively by using the provided test set. I also used the one hot encoder to change categorical features to binary variables (Figure 8). Elastic Net performed the worst, producing an MAE of 1,687.72, while the resulting MAE for the Ridge model came to be 1,335.49. Lasso regression showed the best performance with a MAE of 1,328.87, hence I decided to choose the Lasso regression model to submit to Kaggle (Figure 1).

Model Performance and Recommendation

After sharing my predicted loss values using Lasso regression, I received a public score of 1319.25256. It was challenging working with data with unlabeled features. If the features were to have been labeled, I believe it would have been more helpful to determine which features contributed to the spread of the data, thus making it easier to determine which features effected the prediction of 'loss' more than others. It would be possible to improve the accuracy of the model by fine tuning the hyperparameters by using grid search find alpha values that work best for the training dataset. However, I do believe that Lasso regression is the appropriate model to use for this particular dataset.

Appendix

Submission and Description	Private Score	Public Score	Use for Final Score
Lasso_preds.csv 6 hours ago by Michael Venit I determined that Lasso regression was the best method to use as it generated a higher MAE on the training data than Ridge regression and Elastic Net. I then used Lasso regression to generate predictions on the test set for loss.	1326.07984	1319.25256	☐

Figure 1

```
In [1]: import os
        import pandas as pd
        import seaborn as sns
        import numpy as np
        import plotly.graph_objs as go
        from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
        init_notebook_mode(connected=True)
        from sklearn import tree
        from sklearn.metrics import accuracy_score
        from sklearn import preprocessing
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.preprocessing import minmax_scale
        from sklearn.preprocessing import MaxAbsScaler
        from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import RobustScaler
        from sklearn.preprocessing import Normalizer
        from sklearn.preprocessing import QuantileTransformer
        from sklearn.preprocessing import PowerTransformer
        %matplotlib inline
        import matplotlib as mpl
        from matplotlib import cm
        import matplotlib.pyplot as plt
        from IPython.core.interactiveshell import InteractiveShell
        from sklearn.linear_model import RidgeCV, ElasticNetCV, LassoCV
        from sklearn.decomposition import PCA #Principal Component Analysis
        from sklearn.preprocessing import LabelEncoder #transforms categorical into numbers
        from sklearn.model_selection import KFold
        from sklearn.model_selection import train_test_split, cross_val_score
```

```
In [2]: claims_train= pd.read_csv('train.csv')
        claims_test= pd.read_csv('test.csv')
```

Figure 2

```
In [3]: claims_train.head()
```

Out[3]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8	cont9	cont10	cont11	cc
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.59
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.36
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.37
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.32
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.20

5 rows × 132 columns

Figure 3

```
In [5]: claims_train.describe()
```

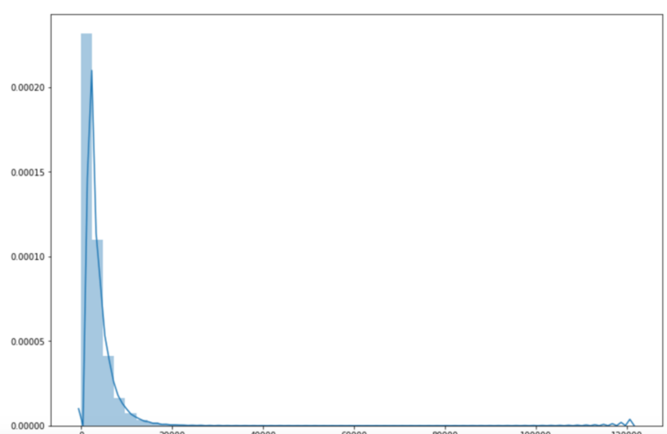
Out[5]:

	id	cont1	cont2	cont3	cont4	cont5	cont6	c
count	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.000000	188318.00
mean	294135.982561	0.493861	0.507188	0.498918	0.491812	0.487428	0.490945	0.48
std	169336.084867	0.187640	0.207202	0.202105	0.211292	0.209027	0.205273	0.17
min	1.000000	0.000016	0.001149	0.002634	0.176921	0.281143	0.012683	0.06
25%	147748.250000	0.346090	0.358319	0.336963	0.327354	0.281143	0.336105	0.35
50%	294539.500000	0.475784	0.555782	0.527991	0.452887	0.422268	0.440945	0.43
75%	440680.500000	0.623912	0.681761	0.634224	0.652072	0.643315	0.655021	0.59
max	587633.000000	0.984975	0.862654	0.944251	0.954297	0.983674	0.997162	1.00

Figure 4

```
In [8]: plt.figure(figsize=(13,9))
        sns.distplot(claims_train["loss"])
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c08dd1b50>




```
In [7]: plt.figure(figsize=(13,9))
        sns.boxplot(claims_train["loss"])
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c3b9616d0>

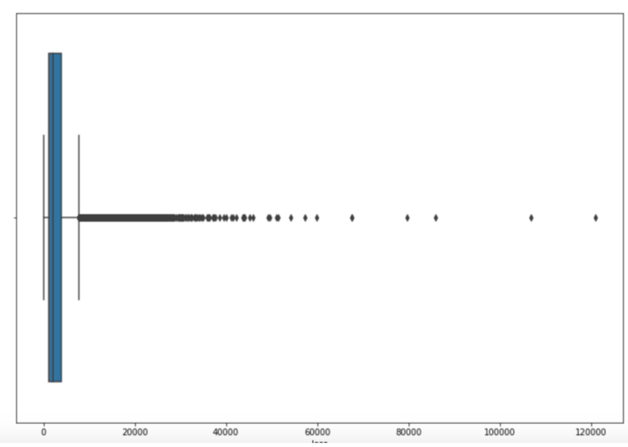


Figure 5

```
In [12]: from sklearn.preprocessing import LabelEncoder
         enc = LabelEncoder()
         for i in claims_train:
             if 'cat' in i:
                 claims_train[i] = enc.fit_transform(claims_train[i])
```

```
In [13]: claims_train.head()
```

Out[13]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	cont7	cont8	cont9	cont10	cont11	cc
0	1	0	1	0	1	0	0	0	0	1	...	0.718367	0.335060	0.30260	0.67135	0.83510	0.569745	0.59
1	2	0	1	0	0	0	0	0	0	1	...	0.438917	0.436585	0.60087	0.35127	0.43919	0.338312	0.36
2	5	0	1	0	0	1	0	0	0	1	...	0.289648	0.315545	0.27320	0.26076	0.32446	0.381398	0.37
3	10	1	1	0	1	0	0	0	0	1	...	0.440945	0.391128	0.31796	0.32128	0.44467	0.327915	0.32
4	11	0	1	0	1	0	0	0	0	1	...	0.178193	0.247408	0.24564	0.22089	0.21230	0.204687	0.20

5 rows × 132 columns

Figure 6

```
In [17]: def perform_reg(clf, X, y):
             X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2)
             clf.fit(X_train, y_train)
             y_preds = clf.predict(X_test)
             MAE = np.mean(np.abs(y_preds-y_test))
             return MAE, y_preds
```

```
In [22]: for clf in [RidgeCV(), ElasticNetCV(), LassoCV()]:
             print(clf.__class__.__name__)
             MAE, y_preds = perform_reg(clf, X.values, y.values)
             print(MAE)
```

```
RidgeCV
1335.492866695999
ElasticNetCV
1687.715484594647
LassoCV
1328.8682883696931
```

Figure 7

In [20]:
```
enc = LabelEncoder()
for i in claims_test:
    if 'cat' in i:
        claims_test[i] = enc.fit_transform(claims_test[i])
```

In [51]:
```
y_preds = clf.predict(claims_test.drop(['id'], axis=1))
preds = claims_test['id'].copy()
loss = pd.DataFrame(y_preds)
loss.columns= ['loss']
file= pd.concat([preds, loss], axis= 1)
file
```

Out[51]:

	id	loss
0	4	1274.940446
1	6	2228.458810
2	9	10656.537583
3	12	5684.028717
4	15	141.278251
...	...	...
125541	587617	2909.449206
125542	587621	2786.094915
125543	587627	2772.890976
125544	587629	989.380931
125545	587634	4774.309015

125546 rows × 2 columns

Figure 8