Insurance Ultimate Claim - Head on Collision Dataset

This notebook provide full data exploration and model selection process for insurance claim project using head-on-collision data of FNOL. All stages of ML project phases are explained with visulation and findings suppoeted with data. Goal of this project is to predict ultimate claim amount usining Incurred amount given the claim features. Serval ML techniques of Regression has been discussed and implemented in this notebook.

Import necessary packages

In [75]:
```
import numpy as np
import pandas as pd
import os
import glob
import seaborn as sns
import matplotlib.pyplot as plt
from pathlib import Path
from functools import reduce
from datetime import datetime
from termcolor import colored

from sklearn.preprocessing import Binarizer, OneHotEncoder, PowerTransformer, MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, ShuffleSplit, cross_val_score, StratifiedShuffleSplit
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn.pipeline import Pipeline
from sklearn import set_config


from xgboost import XGBRegressor

from yellowbrick.regressor import residuals_plot
from yellowbrick.regressor import prediction_error

set_config(display="diagram")
pd.set_option('display.max_columns', None)
# pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

In [76]:
```
# data_path = r"C:\Users\kumar\OneDrive\Desktop\Akshat\insurance"
# file_path = Path(data_path + r'\Data')

# ls_files = glob.glob(str(file_path) + r'/*.xlsx')
# latest_file = [os.path.basename(filename) for filename in ls_files]
```

Data Exploration

In this section we will load the train and test dataset and analyze the features, shape and statistics on each feature to get some insight about data we are dealing with.

In [77]:
```
raw_excel_dff = pd.read_excel(r"C:\Users\kumar\OneDrive\Desktop\Akshat\insurance\Data\Data 08-10-2021.xlsx", engine='openpyxl')
excel_dff = raw_excel_dff.copy()
```

In [78]:
```
print(f"Total row: {excel_dff.shape[0]} and columns: {excel_dff.shape[1]} in raw dataset", end='\n\n')
print(excel_dff.info())
```

```
Total row: 7691 and columns: 46 in raw dataset

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7691 entries, 0 to 7690
Data columns (total 46 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Claim Number                7691 non-null   int64
 1   date_of_loss                7691 non-null   object
 2   Notifier                    7691 non-null   object
 3   Loss_code                   7691 non-null   object
 4   Loss_description            7691 non-null   object
 5   Notification_period         7691 non-null   int64
 6   Inception_to_loss           7691 non-null   int64
 7   Location_of_incident        7691 non-null   object
 8   Weather_conditions          7346 non-null   object
 9   Vehicle_mobile              7691 non-null   object
 10  Time_hour                   7691 non-null   int64
 11  Main_driver                 7691 non-null   object
 12  PH_considered_TP_at_fault   7691 non-null   object
 13  Vechile_registration_present 7691 non-null  int64
 14  Incident_details_present    7691 non-null   int64
 15  Injury_details_present      7691 non-null   int64
 16  TP_type_insd_pass_back      7691 non-null   int64
 17  TP_type_insd_pass_front     7691 non-null   int64
 18  TP_type_driver              7691 non-null   int64
 19  TP_type_pass_back           7691 non-null   int64
 20  TP_type_pass_front          7691 non-null   int64
 21  TP_type_bike                7691 non-null   int64
 22  TP_type_cyclist             7691 non-null   int64
 23  TP_type_pass_multi          7691 non-null   int64
 24  TP_type_pedestrian          7691 non-null   int64
 25  TP_type_other               7691 non-null   int64
 26  TP_type_nk                  7691 non-null   int64
 27  TP_injury_whiplash          7691 non-null   int64
 28  TP_injury_traumatic         7691 non-null   int64
 29  TP_injury_fatality          7691 non-null   int64
 30  TP_injury_unclear           7691 non-null   int64
 31  TP_injury_nk                7691 non-null   int64
 32  TP_region_eastang           7691 non-null   int64
 33  TP_region_eastmid           7691 non-null   int64
 34  TP_region_london            7691 non-null   int64
 35  TP_region_north             7691 non-null   int64
 36  TP_region_northw            7691 non-null   int64
 37  TP_region_outerldn          7691 non-null   int64
 38  TP_region_scotland          7691 non-null   int64
 39  TP_region_southe            7691 non-null   int64
 40  TP_region_southw            7691 non-null   int64
 41  TP_region_wales             7691 non-null   int64
 42  TP_region_westmid           7691 non-null   int64
 43  TP_region_yorkshire         7691 non-null   int64
 44  Incurred                    7691 non-null   float64
 45  Capped Incurred             7691 non-null   float64
dtypes: float64(2), int64(35), object(9)
memory usage: 2.7+ MB
None
```

In [79]:
```
excel_dff.head(20)
```

Out[79]:

	Claim Number	date_of_loss	Notifier	Loss_code	Loss_description	Notification_period	Inception_to_loss	Location_of_incident	Weather_conditions
0	1	2003-04-15	PH	LD003	Head on collision	22	13	Main Road	NORMAL
1	2	2003-04-20	CNF	LD003	Head on collision	1	9	Main Road	WET
2	3	2003-04-24	CNF	LD003	Head on collision	5	17	Main Road	WET
3	4	2003-05-13	CNF	LD003	Head on collision	1	23	Main Road	N/K
4	5	2003-06-11	CNF	LD003	Head on collision	1	48	Other	N/K
5	6	2003-06-24	PH	LD003	Head on collision	16	23	Other	N/K
6	7	2003-07-16	Other	LD003	Head on collision	5	4	Main Road	NORMAL
7	8	2003-07-17	Other	LD003	Head on collision	0	40	Main Road	WET
8	9	2003-07-20	PH	LD003	Head on collision	4	26	Minor Road	WET
9	10	2003-07-29	CNF	LD003	Head on collision	2	85	Main Road	N/K
10	11	2003-07-30	Other	LD003	Head on collision	0	109	Minor Road	WET
11	12	2003-08-12	Other	LD003	Head on collision	1	22	Main Road	NORMAL
12	13	2003-08-13	Other	LD003	Head on collision	0	46	Main Road	NORMAL
13	14	2003-08-19	CNF	LD003	Head on collision	2	7	Other	NORMAL
14	15	2003-08-27	PH	LD003	Head on collision	0	57	Minor Road	NORMAL
15	16	2003-08-28	Other	LD003	Head on collision	6	104	Minor Road	WET
16	17	2003-08-29	TP	LD003	Head on collision	61	55	Not Applicable	NaN
17	18	2003-08-30	PH	LD003	Head on collision	2	27	Main Road	NORMAL
18	19	2003-09-08	PH	LD003	Head on collision	0	128	Minor Road	WET
19	20	2003-09-10	CNF	LD003	Head on collision	0	18	Other	NORMAL

Analyze Statistics on Features

Analyzing statistics on all numeric feature is done to determine what range of values are depicted by each of these features. As we can observe from below stats, values for most of numeric features lies between 0 to 7. Mean is closer to 0.3 and standard deviation is not more than 0.6. So, most of the data is somewhat evenly distributed however, there are few features with extreame mean and standard deviation. This include our target column **Incurred** as well, which has 9k and 49k as mean and standard deviation. Also, data doesn't have any NULL values as empyt in TP column signifies 0 count (assumption as per data information).

Also, since target column has empty, negative and 0 values so assumption is made that either these are fraudulent claims or eiither insurer is not liable to pay anything. At this stage we don't want to make any harsh hypothesis and replace these values with mean or mode

In [80]:
```
def descriptive_fun(dff):

    # percentage of zeros
    count_zero = pd.DataFrame((dff == 0).sum(axis=0)/len(dff) * 100, columns = ['zeros%'])

    # skewness and kurtosis
    skew_df = pd.DataFrame(dff.agg(['skew', 'kurtosis']).transpose())

    # unique values
    unique_df = pd.DataFrame(dff.nunique(), columns = ['unique'])

    #describe
    describe_df = pd.DataFrame(dff.describe())

    descriptive_data = [unique_df, count_zero, skew_df]

    descriptive_df = reduce(lambda left, right: pd.merge(left, right, left_index=True, right_index=True), descriptive_data).transpose()
    descriptive_df = pd.concat([describe_df, descriptive_df])

    return descriptive_df
```

Analyze Skewness and Kurtosis on Numeric Features

Pandas skew() and kurtosis() methods are used to check skewness kurtosis of all numeric features. From this analysis we can determine whether normalization and standardization is required to make data gaussian/normal.

In general skewness is defined in three stages:

- If the skewness is between -0.5 and 0.5, the data are fairly symmetrical
- If the skewness is between -1 and -0.5 or between 0.5 and 1, the data are moderately skewed
- If the skewness is less than -1 or greater than 1, the data are highly skewed

Kurtosis, refers to the degree of presence of outliers in the distribution. Again, in general kurtosis is measured in three ranges:

- Leptokurtic or heavy-tailed distribution (kurtosis more than normal distribution) - kurtosis > 3
- Mesokurtic (kurtosis same as the normal distribution) - kurtosis = 3
- Platykurtic or short-tailed distribution (kurtosis less than normal distribution) - kurtosis < 3

From **descriptive_fun** functions observation we can see that all features are either moderately or highly skewed and non has Mesokurtic distribution. Again, we can see target column **Incurred** is highly skewed and has exterem outliers.

Another important observation from descriptive analysis is that there are few columns which has just 1 unique value, hence it will not make any difference to model predictive power.

In [81]:
```
descriptive_fun(excel_dff)
```

Out[81]:

	Claim Number	Notification_period	Inception_to_loss	Time_hour	Vechile_registration_present	Incident_details_present	Injury_details_present
count	7691.000	7691.000	7691.000	7691.000	7691.000	7691.000	7691.000
mean	3846.000	7.163	166.855	12.730	0.999	0.808	0.232
std	2220.345	39.138	104.453	5.107	0.028	0.394	0.422
min	1.000	-18.000	0.000	0.000	0.000	0.000	0.000
25%	1923.500	0.000	75.000	9.000	1.000	1.000	0.000
50%	3846.000	1.000	161.000	13.000	1.000	1.000	0.000
75%	5768.500	2.000	253.000	17.000	1.000	1.000	0.000
max	7691.000	1042.000	365.000	23.000	1.000	1.000	1.000
unique	7691.000	189.000	366.000	24.000	2.000	2.000	2.000
zeros%	0.000	44.637	0.338	4.642	0.078	19.165	76.791
skew	0.000	13.597	0.171	-0.498	-35.768	-1.567	1.269
kurtosis	-1.200	251.053	-1.156	-0.052	1277.665	0.456	-0.389

In [82]:
```
# Removing columns with just 1 unique value
columns_drop = ['Loss_code','Loss_description','TP_type_insd_pass_front', 'TP_type_pass_multi']
excel_dff.drop(columns_drop, axis = 1, inplace=True)
```

Exploratory Visualization

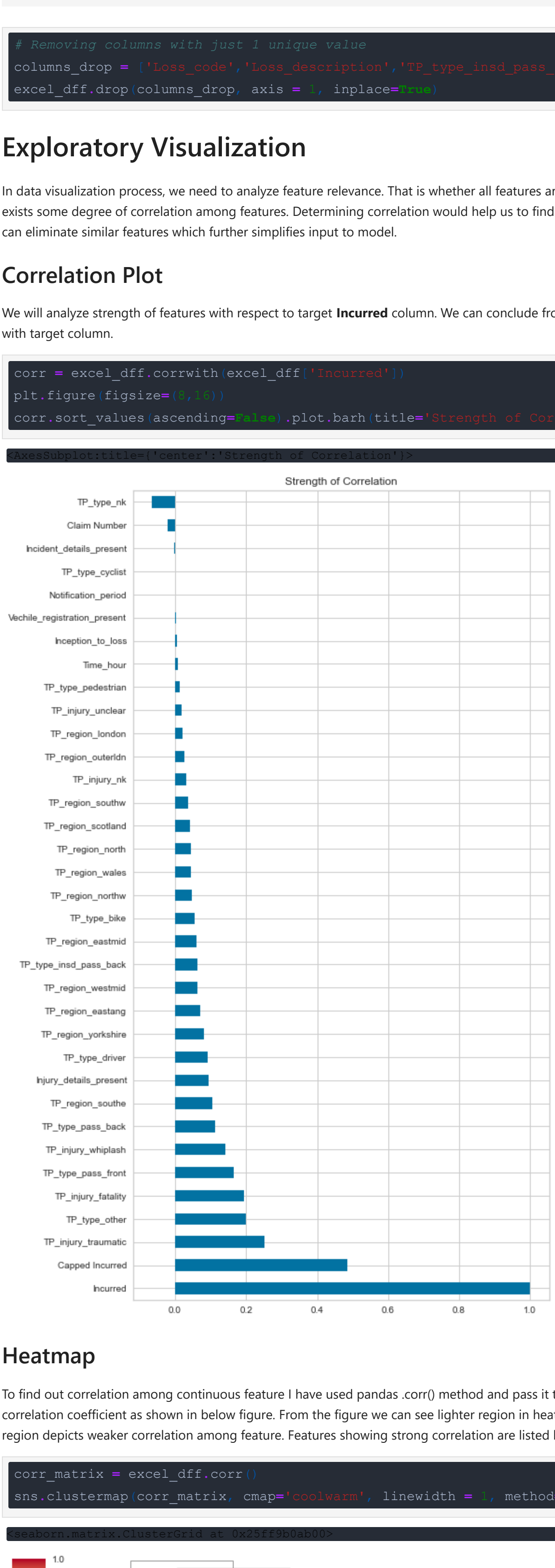
In data visualization process, we need to analyze feature relevance. That is whether all features are independent of each other or there exists some degree of correlation among features. Determining correlation would help us to find out redundancy among features. Then we can eliminate similar features which further simplifies input to model.

Correlation Plot

We will analyze strength of features with respect to target **Incurred** column. We can conclude from plot that no feature is highly correlated with target column.

In [83]:
```
corr = excel_dff.corrwith(excel_dff['Incurred'])
plt.figure(figsize=(8,16))
corr.sort_values(ascending=False).plot.barh(title='Strength of Correlation')
```

Out[83]:
```
<AxesSubplot:title={'center':'Strength of Correlation'}>
```

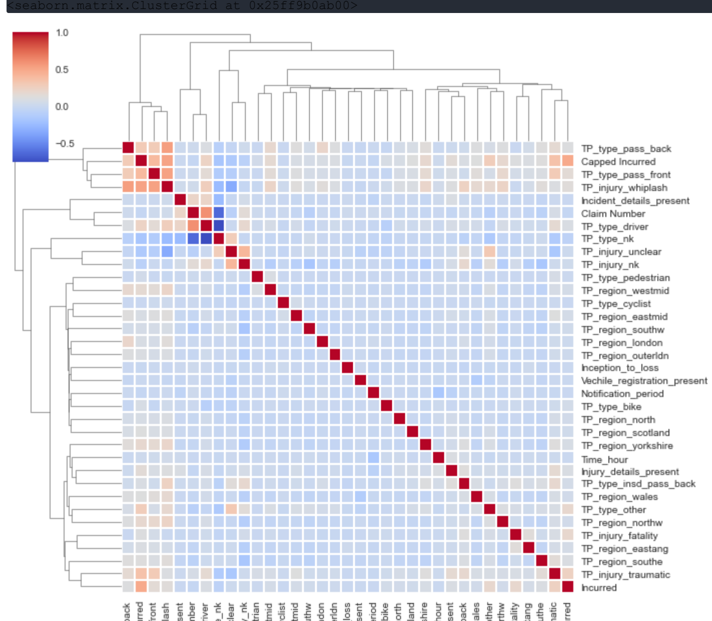


Heatmap

To find out correlation among continuous feature I have used pandas .corr() method and pass it to seaborn heatmap to visualize Pearson correlation coefficient as shown in below figure. From the figure we can see lighter region in heatmap shows strong correlation and darker region depicts weaker correlation among feature. Features showing strong correlation are listed below.

In [84]:
```
corr_matrix = excel_dff.corr()
sns.clustermap(corr_matrix, cmap='coolwarm', linewidth = 1, method='ward')
```

Out[84]:
```
<seaborn.matrix.ClusterGrid at 0x25ff9b0ab00>
```



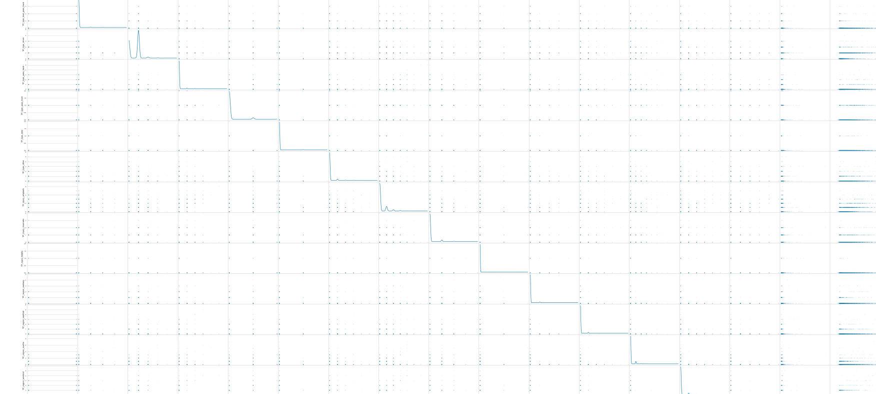
Scatter Matrix

Scatter Matrix is plotted between features where correlation with respect to target column is greater that 0.5, this was done to avoid clutter in visual. It is use to review distribution of features and to understand correlation and skewness. Scatter plot visualization depicts similar correlation betwen features compared to what we discovered in heatmap visuals.

In [85]:
```
# only columns where correlation with incurred is more than 0.04
pd.plotting.scatter_matrix(excel_dff[corr[corr.sort_values(ascending=False) > 0.05].index],
alpha=0.1, figsize = (80,50), diagonal='kde')
```

Out[85]:

```
array([[<AxesSubplot:xlabel='Injury_details_present', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='Incurred', ylabel='Injury_details_present'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='Injury_details_present'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='Incurred', ylabel='TP_type_insd_pass_back'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='TP_type_insd_pass_back'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='TP_type_driver'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='TP_type_driver'>,
        ...
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='Incurred', ylabel='TP_region_southe'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='TP_region_southe'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='Incurred', ylabel='TP_region_westmid'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='TP_region_westmid'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='Incurred', ylabel='TP_region_yorkshire'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='TP_region_yorkshire'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='Incurred'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='Incurred'>,
        <AxesSubplot:xlabel='Incurred', ylabel='Incurred'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='Incurred'>],
       [<AxesSubplot:xlabel='Injury_details_present', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_insd_pass_back', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_driver', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_pass_back', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_pass_front', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_bike', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_type_other', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_injury_whiplash', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_injury_traumatic', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_injury_fatality', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_region_eastang', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_region_eastmid', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_region_southe', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_region_westmid', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='TP_region_yorkshire', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='Incurred', ylabel='Capped Incurred'>,
        <AxesSubplot:xlabel='Capped Incurred', ylabel='Capped Incurred'>]],
      dtype=object)
```

Outlier Detection

It is another major task in ML pipeline, since outliers can push predicted regression line towards it even if only small amout of outliers are present in data. As we discovered in descriptive analysis part, no categorical features have missing however some of them has n/k and not applicable which can be mearged together to create missing category.

For numeric data we will run **get_outlier** function which will output columns with count of outliers. Outliers are defined data points below 5th and above 95th percentile values. Only two columns are indentified, amoung them is our target column.

In [86]:
```
def get_outliers(dff, n_sigma = 4):
    # n_sigma is number of sigma, which define the boundary along mean

    outliers = pd.DataFrame()
    stats = dff.describe()

    # select only numeric columns
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    outlier_df = dff.select_dtypes(include=numerics)

    # select columns having more than 500 unique values
    outlier_dff = outlier_df[[col for col in outlier_df if outlier_df[col].nunique() > 500]]

    for col in outlier_dff.columns:
        mu = stats.loc['mean', col]
        sigma = stats.loc['std', col]
        condition = (outlier_dff[col] > mu + sigma * n_sigma) | (outlier_dff[col] < mu - sigma * n_sigma)
        outliers[f'{col}_outliers'] = outlier_dff[col][condition]

    return outliers
```

In [87]:
```
outliers = get_outliers(excel_dff, n_sigma = 1)
outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3250 entries, 0 to 7690
Data columns (total 3 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Claim Number_outliers     3250 non-null   int64
 1   Incurred_outliers         48 non-null     float64
 2   Capped Incurred_outliers  336 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 101.6 KB
```

Winsorization

Winsorization is the process of replacing a specified number of extreme values with a smaller data value.A typical strategy is to set all outliers to a specified percentile of the data; for example, a 95% winsorization would see all data below the 5th percentile set to the 5th percentile, and data above the 95th percentile set to the 95th percentile. Note here, the shape of the dataframe remains the same.

Normal Distribution

Considering if data is normally distributed, the values within one standard deviation of the mean account for about 68% of the set; while within two standard deviations account for about 95%; and within three standard deviations account for about 99.7%

In [88]:
```
outlier_columns = ["Incurred", "Capped Incurred"]
out_new_dff = excel_dff[excel_dff.columns.intersection(outlier_columns)].reset_index(drop=True)
```

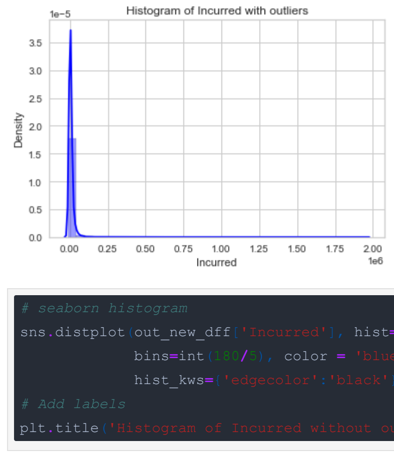
In [89]:
```
outlier_cutoff = 0.01

out_new_dff.pipe(lambda x: x.clip(lower=x.quantile(outlier_cutoff), upper=x.quantile(1-outlier_cutoff), axis=1, inplace=True))
```

In [90]:
```
# seaborn histogram
sns.distplot(excel_dff['Incurred'], hist=True, kde=True,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
# Add labels
plt.title('Histogram of Incurred with outliers')
```

```
C:\Users\kumar\.conda\envs\ins\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[90]:
```
Text(0.5, 1.0, 'Histogram of Incurred with outliers')
```

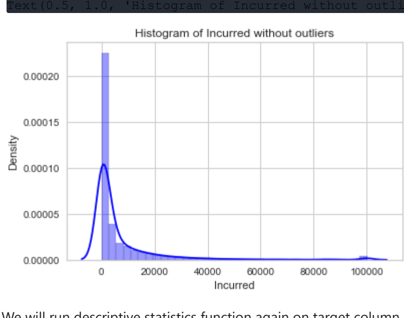


In [91]:
```
# seaborn histogram
sns.distplot(out_new_dff['Incurred'], hist=True, kde=True,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
# Add labels
plt.title('Histogram of Incurred without outliers')
```

```
C:\Users\kumar\.conda\envs\ins\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[91]:

```
Text(0.5, 1.0, 'Histogram of Incurred without outliers')
```



We will run descriptive statistics function again on target column, to check if outlier removal have any effect on skewness and kurtosis at all. Interestingly, both skewness and kurtosis decreased by upto 7 times as compared with outliers

In [92]:

```python
descriptive_fun(out_new_dff)
```

Out[92]:

	Incurred	Capped Incurred
count	7691.000	7691.000
mean	6877.285	6111.512
std	14875.706	11053.888
min	0.000	0.000
25%	24.909	24.909
50%	1237.714	1237.714
75%	6258.404	6258.404
max	100110.251	50000.000
unique	5595.000	5474.000
zeros%	23.430	23.430
skew	3.995	2.559
kurtosis	18.806	6.302

In [93]:

```python
# Adding back columns to main dataframe
excel_dff.drop(outlier_columns, axis=1, inplace=True)
excel_dff = pd.merge(excel_dff, out_new_dff, left_index=True, right_index=True)
```

Feature Engineering

Since no given column has strong correlation with target column therefore we will create some new features by combining already present features

- Adding Flag Columns : All TP columns are count therefore adding flag for all TP columns where count is greater than 0
- Date Column - In algorithm, dates doesn't have any effect on its own therefore converting date_of_loss colummn to "Month" and "Weekend" flag
- Range reduction - Converted Inception_to_loss to Months Since Loss (MSL), this kind of techniques is used to reduce spread of data
- Injury Flag - If count is greater than 0 for "Wiplash", "Traumatic", "Fatality" or "Unclear"
- Insurred Passanger Injury - If count is greater than to 0 for insurred passenger at back and Injury Flag is 1
- Combined Categories - Some of the categories are combined to create missing category (like n/k and n/a)

In [94]:

```python
def flag_int(dff, col_list):

    binarizer = Binarizer(threshold=0, copy = True)

    for columns in col_list:
        dff[f'{columns}_flag'] =  binarizer.fit_transform(dff[columns].values.reshape(-1, 1))

    return dff
```

In [95]:

```python
tp_cols = [col for col in excel_dff if col.startswith('TP_') and excel_dff[col].nunique() > 2]

new_dff = flag_int(excel_dff, tp_cols)
```

In [ ]:

```python
new_dff['Month'] = pd.DatetimeIndex(new_dff['date_of_loss']).month
new_dff['Weekend'] = np.where((pd.to_datetime(new_dff['date_of_loss']).dt.dayofweek) > 4, 1, 0)
new_dff['MSL'] = round(new_dff['Inception_to_loss']/30, 0)

new_dff['TP_injury_flag'] = np.where((new_dff['TP_injury_whiplash'] > 0) |
(new_dff['TP_injury_traumatic'] > 0) | (new_dff['TP_injury_fatality'] > 0) |
(new_dff['TP_injury_unclear'] > 0), 1, 0)
new_dff['TP_insd_pass_injury'] = np.where((new_dff['TP_type_insd_pass_back'] > 0) &
(new_dff['TP_injury_flag'] > 0), 1, 0)

new_dff['Vehicle_mobile'] = np.where(new_dff['Vehicle_mobile'].str.lower().isin(['n/k']) ==
True, 'missing', new_dff['Vehicle_mobile'].str.lower())
new_dff['PH_considered_TP_at_fault'] =
np.where(new_dff['PH_considered_TP_at_fault'].str.lower().isin(['n/k', '#']) == True, 'missing',
new_dff['PH_considered_TP_at_fault'].str.lower())
new_dff['Location_of_incident'] =
np.where(new_dff['Location_of_incident'].str.lower().isin(['n/k', 'not applicable']) == True,
'missing', new_dff['Location_of_incident'].str.lower())
new_dff['Weather_conditions'] = np.where(new_dff['Weather_conditions'].str.lower() ==
'snow,ice,fog', 'chilly', new_dff['Weather_conditions'].str.lower())
new_dff['Weather_conditions'] = np.where(new_dff['Weather_conditions'].isin(['n/a', 'n/k']) |
new_dff['Weather_conditions'].isnull() == True, 'missing', new_dff['Weather_conditions'])
```

In [97]:

```python
new_dff.drop(["date_of_loss",'Loss_code','Loss_description','TP_type_insd_pass_front',
'TP_type_pass_multi'], axis = 1, inplace=True, errors="ignore")
```

In [99]:

```python
print(f"Number of columns in dataset before feature engineering: {raw_excel_dff.shape[1]}")
print(f"Number of columns in dataset after feature engineering: {new_dff.shape[1]}")
```

```
Number of columns in dataset before feature engineering: 46
Number of columns in dataset after feature engineering: 70
```

In [100…

```python
new_dff.head(20)
```

Out[100…

	Claim Number	Notifier	Notification_period	Inception_to_loss	Location_of_incident	Weather_conditions	Vehicle_mobile	Time_hour	Main_driver
0	1	PH	22	13	main road	normal	y	10	Other
1	2	CNF	1	9	main road	wet	y	18	Other
2	3	CNF	5	17	main road	wet	y	16	Y
3	4	CNF	1	23	main road	missing	y	14	Other
4	5	CNF	1	48	other	missing	n	9	Other
5	6	PH	16	23	other	missing	n	0	Other
6	7	Other	5	4	main road	normal	n	18	Other
7	8	Other	0	40	main road	wet	y	7	Other
8	9	PH	4	26	minor road	wet	y	22	Other
9	10	CNF	2	85	main road	missing	n	22	Other
10	11	Other	0	109	minor road	wet	y	7	Other
11	12	Other	1	22	main road	normal	n	23	Other
12	13	Other	0	46	main road	normal	y	11	Other
13	14	CNF	2	7	other	normal	y	16	Y
14	15	PH	0	57	minor road	normal	n	18	Other
15	16	Other	6	104	minor road	wet	y	10	Other
16	17	TP	61	55	missing	missing	missing	0	Other
17	18	PH	2	27	main road	normal	n	13	Other
18	19	PH	0	128	minor road	wet	n	8	Other
19	20	CNF	0	18	other	normal	n	11	Y

Partition dataset for Train and Test split

StratifiedShuffleSplit from sklearn is used to get similar distribution of train and test set

In [101…

```python
min_value = descriptive_fun(pd.DataFrame(new_dff["Incurred"])).loc['min', :][0]
max_value = descriptive_fun(pd.DataFrame(new_dff["Incurred"])).loc['max', :][0]
print(min_value)
print(max_value)

bins = np.linspace(min_value, max_value, 5)
print(bins)
```

```
0.0
100110.25054704066
[     0.          25027.56263676  50055.12527352  75082.68791028
 100110.25054704]
```

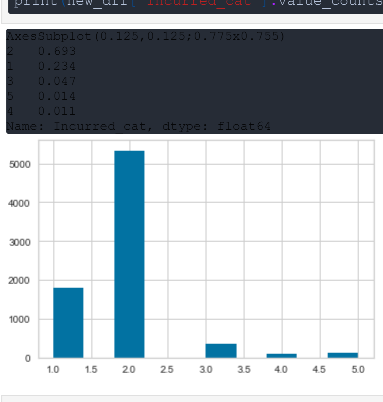
In [102…

```python
new_dff["Incurred_cat"] = pd.cut(new_dff["Incurred"],
                               bins = [-np.inf, 0., 25027., 50055., 75082., np.inf],
                               labels = range(1, 6))

print(new_dff["Incurred_cat"].hist())

print(new_dff["Incurred_cat"].value_counts()/len(new_dff))
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
2    0.693
1    0.234
3    0.047
5    0.014
4    0.011
Name: Incurred_cat, dtype: float64
```



In [103…

```python
# Split data using "Incurred_cat" column defined above
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=1234)
for train_index, test_index in split.split(new_dff, new_dff["Incurred_cat"]):
    strat_train_set = new_dff.loc[train_index].reset_index(drop=True)
    strat_test_set = new_dff.loc[test_index].reset_index(drop=True)

print(strat_test_set["Incurred_cat"].value_counts()/len(strat_test_set))

print(f"Total number of rows in train set: {strat_train_set.shape[0]} and columns:
{strat_train_set.shape[1]}")
print(f"Total number of rows in test set: {strat_test_set.shape[0]} and columns:
{strat_test_set.shape[1]}")
```

```
2    0.693
1    0.235
3    0.047
5    0.014
4    0.011
Name: Incurred_cat, dtype: float64
Total number of rows in train set: 6152 and columns: 71
Total number of rows in test set: 1539 and columns: 71
```

In [104…

```python
# =====================================================
# Seprating featues, target and ID column for train set
# =====================================================
train_features = strat_train_set.drop(['Incurred', 'Claim Number', 'Incurred_cat'], axis = 1)
train_target = strat_train_set['Incurred']

# Drop ID Column and save it for future use
train_id_col = strat_train_set['Claim Number']
```

In [105…

```python
# =====================================================
# Seprating featues, target and ID column for test set
# =====================================================
test_features = strat_test_set.drop(['Incurred', 'Claim Number', 'Incurred_cat'], axis = 1)
test_target = strat_test_set['Incurred']

# Drop ID Column and save it for future use
test_id_col = strat_test_set['Claim Number']
```

Seprate Numeric and Categorical features

In [106…

```python
# Integer continuous columns
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
int_list = train_features.select_dtypes(include=numerics).columns
int_cols = [col for col in train_features[int_list] if train_features[col].nunique() > 2]
int_cols =  [*int_cols, 'TP_type_cyclist', 'TP_type_pedestrian']

# Integer Flag columns
int_flag = [col for col in train_features[int_list] if train_features[col].nunique() <= 2]
int_flag_remove = ['TP_type_cyclist', 'TP_type_pedestrian']
int_flag = list(set(int_flag) - set(int_flag_remove))

# Categorical columns
cat_list = train_features.select_dtypes(exclude=numerics).columns
cat_cols = [col for col in train_features[cat_list]]
```

In [108…

```python
descriptive_fun(train_features)
```

Out[108…

	Notification_period	Inception_to_loss	Time_hour	Vechile_registration_present	Incident_details_present	Injury_details_present	TP_type_i
count	6152.000	6152.000	6152.000	6152.000	6152.000	6152.000	
mean	7.170	166.799	12.708	0.999	0.809	0.233	
std	37.255	104.246	5.122	0.025	0.393	0.423	
min	-18.000	0.000	0.000	0.000	0.000	0.000	
25%	0.000	75.000	9.000	1.000	1.000	0.000	
50%	1.000	161.000	13.000	1.000	1.000	0.000	
75%	2.000	252.250	17.000	1.000	1.000	0.000	
max	925.000	365.000	23.000	1.000	1.000	1.000	
unique	174.000	366.000	24.000	2.000	2.000	2.000	
zeros%	44.262	0.341	4.698	0.065	19.083	76.658	
skew	12.405	0.175	-0.493	-39.189	-1.574	1.261	
kurtosis	207.285	-1.150	-0.056	1534.248	0.477	-0.411	

Target column scaling validation

In [113…

```python
power = PowerTransformer(method='yeo-johnson', standardize=True)
pipeline_iso = Pipeline(steps=[('p', power)])

data_trans = pipeline_iso.fit_transform(train_target.values.reshape(-1, 1))
```

In [114…

```python
descriptive_fun(pd.DataFrame(data_trans))
```

Out[114…

	0
count	6152.000
mean	0.000
std	1.000
min	-1.458
25%	-0.848
50%	0.170
75%	0.759
max	2.043
unique	4489.000
zeros%	0.000
skew	-0.170
kurtosis	-1.071

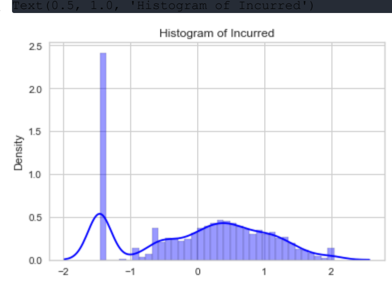
In [115…

```python
# seaborn histogram
sns.distplot(pd.DataFrame(data_trans, columns = ["Incurred"]), hist=True, kde=True,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
# Add labels
plt.title('Histogram of Incurred')
```

```
C:\Users\kumar\.conda\envs\ins\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[115…

```
Text(0.5, 1.0, 'Histogram of Incurred')
```



Data Transformation

From the prior analysis we found that Incurred target feature is skewed and do not show uniform distribution. To deal with this, we can transform data to Power Scale using **PowerTransformer** method provided by sklearn using **yeo-johnson** method which can scale negative and zero values. As we observed in above plot, skewness and kurtosis reduce drastically after transformation and now data looks more like normal distribution (**Note:** peak at begining may suggest another distribution however with over data analysis we know that this is due to high number of 0 values so we can safely ignore it in first iteration).

For features we again use **PowerTransformer** method but before that we scaled data with **StandardScaler** method. Categorical Features are converted into binary One Hot Encoded features. Since we don't have too many unique categories

In [188…

```python
X = train_features.copy()
y = train_target.copy()

# Hot encode categorical features
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_encoder.fit(X[cat_cols])
cat_features = pd.DataFrame(OH_encoder.transform(X[cat_cols]))
cat_features.columns = OH_encoder.get_feature_names(cat_cols)

# Numerical continuous features
num_features = X[int_cols]
scaler = StandardScaler()
power = PowerTransformer(method='yeo-johnson', standardize = True)
num_pipeline = Pipeline(steps=[('s', scaler), ('p', power)])
num_pipeline.fit(num_features)

num_trans_features = pd.DataFrame(num_pipeline.transform(num_features))
num_trans_features.columns = num_features.columns

# Numerical flag features
flag_features = X[int_flag]

data_df = pd.concat([num_trans_features, cat_features, flag_features], axis = 1,
ignore_index=False)

X_new = data_df.copy()

# scaler = MinMaxScaler()
power = PowerTransformer(method='yeo-johnson', standardize= True)
target_pipeline = Pipeline([('p', power)])
target_pipeline.fit(y.values.reshape(-1, 1))

y_new = pd.DataFrame(target_pipeline.transform(y.values.reshape(-1, 1)))

print(f"Feature space increased from {X.shape[1]} to {X_new.shape[1]} after data transformation
step")
```

```
Feature space increased from 68 to 87 after data transformation step
```

In [189…

```python
# Now all features seems to follow normal distribution with mean 0 and standard deviation close
to 1
descriptive_fun(X_new)
```

Out[189…

	Notification_period	Inception_to_loss	Time_hour	TP_type_insd_pass_back	TP_type_driver	TP_type_pass_back	TP_type_pass_front	TP_ty
count	6152.000	6152.000	6152.000	6152.000	6152.000	6152.000	6152.000	
mean	0.000	-0.000	0.000	0.000	0.000	-0.000	0.000	
std	1.000	1.000	1.000	1.000	1.000	1.000	1.000	
min	-26.447	-1.701	-2.133	-0.146	-1.184	-0.177	-0.246	
25%	-0.644	-0.876	-0.782	-0.146	-1.184	-0.177	-0.246	
50%	-0.222	0.005	-0.049	-0.146	0.643	-0.177	-0.246	
75%	0.148	0.840	0.838	-0.146	0.643	-0.177	-0.246	
max	2.759	1.783	2.394	6.860	7.858	5.664	4.072	
unique	174.000	366.000	24.000	2.000	6.000	3.000	3.000	
zeros%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
skew	-1.541	0.037	-0.032	6.716	0.309	5.488	3.827	
kurtosis	80.564	-1.191	-0.454	43.120	1.307	28.130	12.649	

Model Benchmarking

Benchmark has been established on LinearRegressor model from sklearn. Training data is feed through LinearRegressor (with feature input and target) without much data preprocessing. Some of the basic preprocessing steps done on data are listed below.

- Removed feature "Claim Number" from train and validation data as it doesn't provide any value for Incurred prediction. Its like index attribute to uniquely identify each claim row.
- Categorical variables are one hot encoded so that numerical values are given as input to model.

After fitting through LinearRegression(considering all default parameters), prediction is done on test data to compute MAE.

In [125…

```python
# Benchmark model. Running Linear Regression on features data

# Split train and test data in ratio of 9
X_train, X_dev, y_train, y_dev = train_test_split(X_new, y_new, test_size=0.1, random_state=0)

import math
# Run Linear Regression on input train and validation data. Compute Mean absolute error.
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# Make predictions using the test data
y_pred = regr.predict(X_dev)
# y_pred = np.nan_to_num(y_pred)

# Remove high outlier predictions
for i, num in enumerate(y_pred):
  if num > descriptive_fun(y_dev).loc['max', :][0]:
    y_pred[i] = y_pred[i-1]

#Compute MAE from prediction and truth
mae = mean_absolute_error(target_pipeline.inverse_transform(y_dev),
(target_pipeline.inverse_transform(y_pred)))
print("Mean Absolute error: %.2f"% (mae))
```

```
Mean Absolute error: 3650.20
```

Principal Component Analysis

From the data visualization on features we concluded no evidence of strong correlation. Therefore we are using unsupervised dimentionality reduction technique to remove not so useful features. So we will set the variance explainability component to 99% and let PCA select components on its own.

After running PCA fit on Numeric and Categorical data, using explained variance ratio we got 69 components. Original dimention of data is 87

In [184…

```python
# ==============================
# PCA on Numeric features
# ============================
list_ = []
# Instantiate PCA with number of component = 0.99
pca = PCA(n_components=0.99)

# Perform fit on 11 features
pca.fit(num_trans_features)

# Perform transform
reduced_cont_feature = pca.transform(num_trans_features)

list_.append(pca.explained_variance_ratio_)

print(f"Total variance explained by PCA on Numeric Features {sum(list_[0])}")
print(f"Number of components selected by PCA on Numeric Features
{reduced_cont_feature.shape[1]}")
```

```
Total variance explained by PCA on Numeric Features 0.9957146446688513
Number of components selected by PCA on Numeric Features 29
```

In [185…

```python
# ==============================
# PCA on Categorical features
# ============================
cat_dff= pd.concat([cat_features, flag_features], axis = 1,
ignore_index=False).reset_index(drop=True)
cat_dff.shape

list_ = []

pca = PCA(n_components=0.99)

#perform fit on categorical features
pca.fit(cat_dff)

#perform transform
reduced_cat_feature = pca.transform(cat_dff)

list_.append(pca.explained_variance_ratio_)

print(f"Total variance explained by PCA on Categorical Features {sum(list_[0])}")
print(f"Number of components selected by PCA on Categorical Features
{reduced_cat_feature.shape[1]}")
```

```
Total variance explained by PCA on Categorical Features 0.9902581480597915
Number of components selected by PCA on Categorical Features 40
```

Combine Datasets

After PCA and achieving dimensionality reduction on Cat and Num feartures, we will combine both dataset into one and review final shape

In [187…

```python
# After PCA combine Cat and Cont features into single dataset.
reduced_feature = np.hstack((reduced_cat_feature,reduced_cont_feature))
print(f"Number of features present in data {data_df.shape[1]}")
print(f"Number of components selected by PCA {reduced_feature.shape[1]}")
```

```
Number of features present in data 87
Number of components selected by PCA 69
```

Metrics

Mean Absolute error (MAE) is the metric we have chosen to evaluate performance of the model. MAE is measure of absolute difference between actual Incurred and predicted Incurred averaged over all samples.

This metric is considered for the following reasons:

- In this problem statement, prediction of Incurred (claim) value is performed which is numerical quantity. Predicting numerical value from input feature is the case of linear regression. Closer the predicted value to actual, lesser the error term and more accurate regression function can be approximated.
- Another metric for regression problem is Mean Squared Error (MSE). In MSE, square of the difference is taken when evaluating error function. If the error term lies between 0 and 1 then MSE is preferred over MAE as square of number (error term) further reduces. But in given problem statement predicted Incurred value can error by significant margin in hundreds or even thousands by positive or negative margin, so considering MAE is more viable as error term and is not further amplified by taking square.

In [151…

```python
# Mean absolute error performance metric.

def performance_metric(y_true, y_predict):
    """ Calculates and returns the performance score between
        true and predicted values based on the metric chosen. """

    # TODO: Calculate the performance score between 'y_true' and 'y_predict'
    mae = mean_absolute_error(y_true, y_predict)

    # Return the score
    return mae
```

DecisionTreeRegressor

DecisionTreeRegressor model training is done on train data with default parameters to get initial MAE estimate. As we can see resulted MAE of 3377.15 is comparatively less than benchmarked we established with LinearRegressor.

In [152…

```python
#Running DecisionTreeRegressor with default parameters

# Split original data into train and test in ratio of 9:1

X_train, X_dev, y_train, y_dev = train_test_split(reduced_feature,
                                                  y_new,
                                                  test_size = 0.1,
                                                  random_state = 0)

#Instantiate class
regressor = DecisionTreeRegressor(max_depth=8)

# Fit train feature and target
regressor.fit(X_train,y_train)

# Predict on Test features
y_pred = regressor.predict(X_dev)

# #Print MAE from predicted and actual Loss
print("Mean Absolute error: %.2f"
      % performance_metric(target_pipeline.inverse_transform(y_dev),
target_pipeline.inverse_transform(y_pred.reshape(-1, 1))))
```

```
(6152, 1)
(616, 1)
Mean Absolute error: 3377.15
```

Model Complexity Analysis

In this process we have to find the most optimal parameter setting which gives best MAE score for DecisionTreeRegressor. I have chosen max_depth parameter with range options from 5 to 15.

max_depth = [5,6,7,8,9,10,11,12,13,14,15].

Validation curve of training and testing scores are plotted with max_depth on x-axis and scores on y-axis. Scoring function 'neg_mean_absolute_square' is used for evaluation.

In [153…

```python
# Model complexity graph for max_depth parameter
from sklearn.model_selection import validation_curve

def ModelComplexity(X, y):

    # Vary the max_depth parameter from 1 to 10
    max_depth = np.arange(5,15)

    # Calculate the training and testing scores
    train_scores, test_scores = validation_curve(DecisionTreeRegressor(), X, y, \
        param_name = "max_depth", param_range = max_depth, cv = None, scoring =
'neg_mean_absolute_error')

    # Find the mean and standard deviation for smoothing
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    print(train_mean,test_mean)

    # Plot the validation curve
    plt.figure(figsize=(7, 5))
    plt.title('Decision Tree Regressor Complexity Performance')
    plt.plot(max_depth, train_mean, 'o-', color = 'r', label = 'Training Score')
    plt.plot(max_depth, test_mean, 'o-', color = 'g', label = 'Validation Score')
    plt.fill_between(max_depth, train_mean - train_std, \
        train_mean + train_std, alpha = 0.15, color = 'r')
    plt.fill_between(max_depth, test_mean - test_std, \
        test_mean + test_std, alpha = 0.15, color = 'g')

    # Visual aesthetics
    plt.legend(loc = 'lower right')
    plt.xlabel('Max depth')
    plt.ylabel('Score')
    # plt.ylim([-0.40,-0.60])
    plt.show()
```

Observation

As we can observe from model complexity plot below:

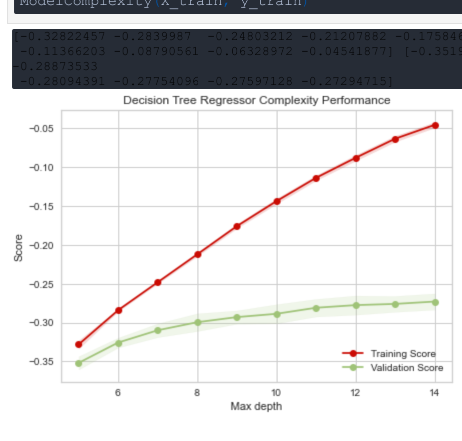
- For max_depth = [5, 6] both training and testing scores are low showing model is underfitting.
- For max_depth = [7 : 15] training score keep increasing as max_depth increased on the other hand, testing score increased upto max_depth 9 and then remain constant comparatevily. This shows sign of high variance as model is overfitting and falling to generalize well.

Hence conclusion can be made from this plot is that we can generalize model more prafectly if we can tune all hyperparameters together to get optimal MAE.

In [154…

```python
ModelComplexity(X_train, y_train)
```

```
[-0.32822457 -0.2839987  -0.24803212 -0.21207882 -0.17584648 -0.14372237
 -0.11366203 -0.08790561 -0.06328972 -0.04541877] [-0.35194781 -0.32574234 -0.30992049 -0.29953626 -0.29297691
 -0.28873533
 -0.28094391 -0.27754096 -0.27597128 -0.27294715]
```



Model Selection

This is a final step where two functions are constructed namely, **split_train_holdout** and **runModel**.

- The first function takes data and split it into train and evaluation. Train data is used to train model and evaluation is used as a test set to check model metrics like MAE.
- Second function is used to train different models (new models can be easily added and tested) and test using evaluation set

In [155…
```
seed = 1234
cv = ShuffleSplit(n_splits = 5, test_size = 0.2, random_state = seed)

def split_train_holdout(features, target, verbose = True):

    train_features = features
    new_target = target

    # Target normalizer
    # scaler = MinMaxScaler()
    power = PowerTransformer(method='yeo-johnson', standardize = True)
    # target_pipeline = Pipeline(steps=[('s', scaler), ('p', power)])
    target_pipeline = Pipeline(steps=[('p', power)])
    target_pipeline.fit(new_target.values.reshape(-1, 1))

    target_y = pd.DataFrame(target_pipeline.transform(new_target.values.reshape(-1, 1)))

    # Take holdout from train
    train_cv, train_holdout, train_cv_y, train_holdout_y = train_test_split(train_features,
target_y, test_size = 0.1,
                                                                            random_state =
seed)

    if(verbose):
        print("\nTrain dataset (Full)")
        print(train_features.shape)

        print("\nTrain CV dataset (subset)")
        print(train_cv.shape)
        print("Train Holdout dataset (subset)")
        print(train_holdout.shape)

    return target_pipeline, train_cv, train_holdout, train_cv_y, train_holdout_y
```

Train Model

Below function perform gri search using **RandomizedSearchCV** from sklearn which is much faster than **GridSearchCV** howvwer it is a trade off between computation power and accuracy.

The 5 fold random shuffled cross validation is performed with grid search (Computationally expensive step). However GPU is used throughout model training process

In [162…
```
def ruModel(pipeline, train_vector, trian_target, holdout_vector, holdout_target, type_, name):

    global cv
    global seed

    ## Regressor type
    if (type_ == "xgb"):
        regressor_type = XGBRegressor(tree_method = "gpu_hist", verbose = 1)
        grid = {
              #'nthread':[4], #when use hyperthread, xgboost may become slower
              #'objective':['reg:squaredlogerror', 'reg:squarederror'],
              'learning_rate': [0.01, 0.03, 0.04, 0.05], #so called `eta` value
              'max_depth': [10, 20],
              'min_child_weight': [8, 10, 20],
              'subsample': [0.5],
              'colsample_bytree': [0.5, 0.6, 0.8, 0.9],
              'colsample_bylevel': [0.5, 0.6, 0.8, 0.9],
              'n_estimators': [100, 500, 1000]
            #   'alpha' : [1]
        }

    if (type_ == "rf"):
        regressor_type = RandomForestRegressor(random_state = seed, n_jobs = -1)
        grid = {
            'bootstrap': [True],
            'max_depth': [10, 20],
            'min_samples_leaf': [20, 40, 100],
            'min_samples_split': [5, 10, 20, 40, 60, 80],
            'n_estimators': [100, 500, 1000]
        }
        trian_target = trian_target.values.reshape(-1,)

    if (type_ == "dt"):
        regressor_type = DecisionTreeRegressor(random_state = seed, criterion = "mae")
        grid = {
            'max_depth': [4, 6, 8, 10, 20],
            'min_samples_leaf': [20, 40, 100],
            'min_samples_split': [5, 10, 20, 40, 60, 80],
            "max_leaf_nodes": [20, 100, 200],
            # 'max_features': ['auto', 'log2', 'sqrt']
            # "criterion": ["mse", "mae"]
        }

    if (type_ == "linear"):
        regressor_type = linear_model.LinearRegression(n_jobs = -1)
        grid = {

        }

    # Model
    print(colored(name, 'red'))

    model = RandomizedSearchCV(estimator = regressor_type, cv = cv, param_distributions = grid,
n_jobs=-1)
    print(colored(model.fit(train_vector, trian_target), "yellow"))

    # Score
    print(colored("\nCV-scores", "blue"))
    mean_score = model.cv_results_['mean_test_score']
    std_score = model.cv_results_["std_test_score"]

    for mean, std, params in sorted(zip(mean_score, std_score, model.cv_results_['params']), key
= lambda x: -x[0]):
        print("Mean Test Score: %0.3f (+/-%0.03f) for params: %r" % (mean, std * 2, params))
    print("\n")

    print(colored("\nBest Estimator Params", "blue"))
    print(colored(model.best_estimator_, "yellow"))

    # Predictions
    print(colored("\nPredictions:", "blue"))
    y_pred = model.predict(holdout_vector).reshape(-1,1)

    # for i, num in enumerate(y_pred):
    #     if num > descriptive_fun(trian_target).loc['max', :][0]:
    #         print("Prediction has outliers")
    #         y_pred[i] = y_pred[i-1]

    model_train_pred = pipeline.inverse_transform(y_pred)
    print(model_train_pred[:10])

    # Compute MAE from prediction and actual values
    MAE = performance_metric(pipeline.inverse_transform(holdout_target), model_train_pred)
    print("\nMean Absolute Error: %2.f" % MAE)

    # Compute RMSE
    MSE = mean_squared_error(pipeline.inverse_transform(holdout_target), model_train_pred)
    RMSE = np.round(np.sqrt(MSE), 2)
    print(f"\nRoot Mean Squared Error: {RMSE}")

    # Compute R-squared
    R_squared = round(r2_score(pipeline.inverse_transform(holdout_target), model_train_pred),2)
    print(f"\nR-squared: {R_squared}")

    # Adjusted R-squared
    Adj_r2 = round(1 - (1-R_squared) * (len(trian_target)-1)/(len(trian_target)-
train_vector.shape[1]-1),2)
    print(f"\nAdjusted R-squared: {Adj_r2}")

    return [name, model, MAE, MSE, RMSE, R_squared, Adj_r2]
```

In [158…
```
target_pipeline, train_cv, train_holdout, train_cv_y, train_holdout_y =
split_train_holdout(features = reduced_feature,

target = strat_train_set['Incurred'])
```

```
Train dataset (Full)
(6152, 69)

Train CV dataset (subset)
(5536, 69)
Train Holdout dataset (subset)
(616, 69)
```

In [167…
```
models = pd.DataFrame(columns = ["model_name", "model_object", "MAE", "MSE", "RMSE", "R_Square",
"Adj_r2"])
```

In [160…
```
linear_regressor = ruModel(pipeline=target_pipeline,
train_vector=train_cv,
trian_target=train_cv_y,
holdout_vector=train_holdout,
holdout_target=train_holdout_y,
type_ = "linear",
name = "linear_regression")
```

```
linear_regression
C:\Users\kumar\.conda\envs\ins\lib\site-packages\sklearn\model_selection\_search.py:289: UserWarning: The total
space of parameters 1 is smaller than n_iter=10. Running 1 iterations. For exhaustive searches, use GridSearchC
V.
  % (grid_size, self.n_iter, grid_size), UserWarning)
RandomizedSearchCV(cv=ShuffleSplit(n_splits=5, random_state=1234, test_size=0.2, train_size=None),
                   estimator=LinearRegression(n_jobs=-1), n_jobs=-1,
                   param_distributions={})

CV-scores
Mean Test Score: -34317801695392.594 (+/-85599029611759.125) for params: {}



Best Estimator Params
LinearRegression(n_jobs=-1)

Predictions:
[[  16.404637  ]
 [1595.24983994]
 [ 471.06247805]
 [ 206.87705922]
 [  22.00660347]
 [  17.12870515]
 [8621.86559605]
 [8029.44595945]
 [  60.72949028]
 [  34.53798793]]

Mean Absolute Error: 3367

Root Mean Squared Error: 8258.67

R-squared: 0.65

Adjusted R-squared: 0.65
```

In [159…
```
dt_regressor = ruModel(pipeline=target_pipeline,
train_vector=train_cv,
trian_target=train_cv_y,
holdout_vector=train_holdout,
holdout_target=train_holdout_y,
type_ = "dt",
name = "decision_tree")
```

```
decision_tree
RandomizedSearchCV(cv=ShuffleSplit(n_splits=5, random_state=1234, test_size=0.2, train_size=None),
                   estimator=DecisionTreeRegressor(criterion='mae',
                                                   random_state=1234),
                   n_jobs=-1,
                   param_distributions={'max_depth': [4, 6, 8, 10, 12, 20],
                                        'max_leaf_nodes': [20, 100, 200],
                                        'min_samples_leaf': [20, 40, 100],
                                        'min_samples_split': [5, 10, 20, 40, 60,
                                                              80]})

CV-scores
Mean Test Score: 0.767 (+/-0.025) for params: {'min_samples_split': 10, 'min_samples_leaf': 20, 'max_leaf_node
s': 200, 'max_depth': 12}
Mean Test Score: 0.741 (+/-0.044) for params: {'min_samples_split': 5, 'min_samples_leaf': 20, 'max_leaf_node
s': 200, 'max_depth': 6}
Mean Test Score: 0.740 (+/-0.045) for params: {'min_samples_split': 40, 'min_samples_leaf': 100, 'max_leaf_node
s': 200, 'max_depth': 6}
Mean Test Score: 0.738 (+/-0.043) for params: {'min_samples_split': 60, 'min_samples_leaf': 100, 'max_leaf_node
s': 200, 'max_depth': 12}
Mean Test Score: 0.738 (+/-0.043) for params: {'min_samples_split': 10, 'min_samples_leaf': 100, 'max_leaf_node
s': 100, 'max_depth': 12}
Mean Test Score: 0.738 (+/-0.043) for params: {'min_samples_split': 5, 'min_samples_leaf': 100, 'max_leaf_node
s': 100, 'max_depth': 8}
Mean Test Score: 0.738 (+/-0.043) for params: {'min_samples_split': 40, 'min_samples_leaf': 100, 'max_leaf_node
s': 100, 'max_depth': 10}
Mean Test Score: 0.729 (+/-0.052) for params: {'min_samples_split': 60, 'min_samples_leaf': 20, 'max_leaf_node
s': 20, 'max_depth': 12}
Mean Test Score: 0.727 (+/-0.045) for params: {'min_samples_split': 20, 'min_samples_leaf': 100, 'max_leaf_node
s': 20, 'max_depth': 10}
Mean Test Score: 0.664 (+/-0.052) for params: {'min_samples_split': 80, 'min_samples_leaf': 20, 'max_leaf_node
s': 20, 'max_depth': 4}



Best Estimator Params
DecisionTreeRegressor(criterion='mae', max_depth=12, max_leaf_nodes=200,
                      min_samples_leaf=20, min_samples_split=10,
                      random_state=1234)

Predictions:
[[   0.        ]
 [3340.03348132]
 [2519.879744  ]
 [1630.3672001 ]
 [  85.6299472 ]
 [   0.        ]
 [6245.07623512]
 [2688.42333691]
 [   0.        ]
 [   0.        ]]

Mean Absolute Error: 3000

Root Mean Squared Error: 9789.9

R-squared: 0.51

Adjusted R-squared: 0.5
```

In [163…
```
rf_regressor = ruModel(pipeline=target_pipeline,
train_vector=train_cv,
trian_target=train_cv_y,
holdout_vector=train_holdout,
holdout_target=train_holdout_y,
type_ = "rf",
name = "random_forest")
```

```
random_forest
RandomizedSearchCV(cv=ShuffleSplit(n_splits=5, random_state=1234, test_size=0.2, train_size=None),
                   estimator=RandomForestRegressor(n_jobs=-1,
                                                   random_state=1234),
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [10, 20],
                                        'min_samples_leaf': [20, 40, 100],
                                        'min_samples_split': [5, 10, 20, 40, 60,
                                                              80],
                                        'n_estimators': [100, 500, 1000]})

CV-scores
Mean Test Score: 0.851 (+/-0.032) for params: {'n_estimators': 100, 'min_samples_split': 80, 'min_samples_lea
f': 20, 'max_depth': 20, 'bootstrap': True}
Mean Test Score: 0.844 (+/-0.034) for params: {'n_estimators': 1000, 'min_samples_split': 80, 'min_samples_lea
f': 40, 'max_depth': 10, 'bootstrap': True}
Mean Test Score: 0.844 (+/-0.035) for params: {'n_estimators': 100, 'min_samples_split': 40, 'min_samples_lea
f': 40, 'max_depth': 10, 'bootstrap': True}
Mean Test Score: 0.844 (+/-0.033) for params: {'n_estimators': 500, 'min_samples_split': 60, 'min_samples_lea
f': 40, 'max_depth': 10, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 100, 'min_samples_split': 60, 'min_samples_lea
f': 100, 'max_depth': 10, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 100, 'min_samples_split': 60, 'min_samples_lea
f': 100, 'max_depth': 20, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 100, 'min_samples_split': 40, 'min_samples_lea
f': 100, 'max_depth': 20, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 1000, 'min_samples_split': 80, 'min_samples_lea
f': 100, 'max_depth': 20, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 1000, 'min_samples_split': 20, 'min_samples_lea
f': 100, 'max_depth': 10, 'bootstrap': True}
Mean Test Score: 0.803 (+/-0.035) for params: {'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf':
 100, 'max_depth': 20, 'bootstrap': True}



Best Estimator Params
RandomForestRegressor(max_depth=20, min_samples_leaf=20, min_samples_split=80,
                      n_jobs=-1, random_state=1234)

Predictions:
[[2.28707138e+00]
 [3.47006895e+03]
 [2.53919391e+03]
 [6.00015758e+02]
 [1.61953490e+01]
 [6.57131992e-01]
 [5.85071883e+03]
 [6.92706043e+03]
 [1.17850769e+01]
 [2.68521495e+00]]

Mean Absolute Error: 2581

Root Mean Squared Error: 9687.68

R-squared: 0.52

Adjusted R-squared: 0.51
```

In [161…
```
xgb_regressor = ruModel(pipeline=target_pipeline,
train_vector=train_cv,
trian_target=train_cv_y,
holdout_vector=train_holdout,
holdout_target=train_holdout_y,
type_ = "xgb",
name = "xgb_regressor")
```

```
xgb_regressor
[16:20:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "verbose" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


RandomizedSearchCV(cv=ShuffleSplit(n_splits=5, random_state=1234, test_size=0.2, train_size=None),
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_chi...
                                          scale_pos_weight=None, subsample=None,
                                          tree_method='gpu_hist',
                                          validate_parameters=None, verbose=1,
                                          verbosity=None),
                   n_jobs=-1,
                   param_distributions={'colsample_bylevel': [0.5, 0.6, 0.8,
                                                              0.9],
                                        'colsample_bytree': [0.5, 0.6, 0.8,
                                                             0.9],
                                        'learning_rate': [0.01, 0.03, 0.04,
                                                          0.05],
                                        'max_depth': [10, 20],
                                        'min_child_weight': [8, 10, 20],
                                        'n_estimators': [100, 500, 1000],
                                        'subsample': [0.5]})

CV-scores
Mean Test Score: 0.920 (+/-0.011) for params: {'subsample': 0.5, 'n_estimators': 1000, 'min_child_weight': 10,
 'max_depth': 10, 'learning_rate': 0.03, 'colsample_bytree': 0.8, 'colsample_bylevel': 0.6}
Mean Test Score: 0.920 (+/-0.014) for params: {'subsample': 0.5, 'n_estimators': 1000, 'min_child_weight': 10,
 'max_depth': 20, 'learning_rate': 0.03, 'colsample_bytree': 0.8, 'colsample_bylevel': 0.8}
Mean Test Score: 0.918 (+/-0.015) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 8, 'm
ax_depth': 20, 'learning_rate': 0.05, 'colsample_bytree': 0.9, 'colsample_bylevel': 0.6}
Mean Test Score: 0.915 (+/-0.014) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 10,
 'max_depth': 10, 'learning_rate': 0.01, 'colsample_bytree': 0.9, 'colsample_bylevel': 0.9}
Mean Test Score: 0.914 (+/-0.015) for params: {'subsample': 0.5, 'n_estimators': 1000, 'min_child_weight': 8,
 'max_depth': 20, 'learning_rate': 0.04, 'colsample_bytree': 0.6, 'colsample_bylevel': 0.9}
Mean Test Score: 0.914 (+/-0.014) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 10,
 'max_depth': 10, 'learning_rate': 0.05, 'colsample_bytree': 0.6, 'colsample_bylevel': 0.8}
Mean Test Score: 0.913 (+/-0.018) for params: {'subsample': 0.5, 'n_estimators': 1000, 'min_child_weight': 10,
 'max_depth': 20, 'learning_rate': 0.03, 'colsample_bytree': 0.5, 'colsample_bylevel': 0.8}
Mean Test Score: 0.910 (+/-0.015) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 8, 'm
ax_depth': 20, 'learning_rate': 0.04, 'colsample_bytree': 0.6, 'colsample_bylevel': 0.5}
Mean Test Score: 0.908 (+/-0.020) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 10,
 'max_depth': 20, 'learning_rate': 0.04, 'colsample_bytree': 0.5, 'colsample_bylevel': 0.5}
Mean Test Score: 0.901 (+/-0.016) for params: {'subsample': 0.5, 'n_estimators': 500, 'min_child_weight': 20,
 'max_depth': 10, 'learning_rate': 0.01, 'colsample_bytree': 0.6, 'colsample_bylevel': 0.5}



Best Estimator Params
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.6,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=0,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.03, max_delta_step=0, max_depth=10,
             min_child_weight=10, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.5,
             tree_method='gpu_hist', validate_parameters=1, verbose=1,
             verbosity=None)

Predictions:
[[2.8976963e+00]
 [3.3987195e+03]
 [1.8884628e+03]
 [5.3858191e+02]
 [1.7376553e+01]
 [1.3893664e+00]
 [7.3092920e+03]
 [1.3052926e+04]
 [3.9899754e+01]
 [4.4515753e-01]]

Mean Absolute Error: 2192

Root Mean Squared Error: 8884.36

R-squared: 0.6

Adjusted R-squared: 0.59
```

In [168…
```
models.loc[len(models)] = linear_regressor
models.loc[len(models)] = dt_regressor
models.loc[len(models)] = rf_regressor
models.loc[len(models)] = xgb_regressor
```

And the best model is.....

XGBRegressor performed best among all selected models with least Mean Absolute error of ~2k.

XGBRegressor is parallelizable onto GPU's and across networks of computers making it feasible to train on very large datasets as in our case. It is an implementation of gradient boosted decision trees designed for speed and performance. The two reasons to use XGBRegressor

Bagging and Boosting features Model Performance I have used XGBRegressor API from XGBoost class. It has lots of trainable parameters. The ones best perfoming parameters are available in model_object column.

Boosting and Bagging process can be computationally very extensive and time consuming. With large dataset it becomes worse unless appropriate params are tuned to optimal values. After instantiating XGBRegressor with optimal parameters, model is fitted with train data features and Incurred target variable. Prediction is done on trained model with test data and finally MAE is computed and returned based on ground truth and predicted values. I have used these parameters to train algorithm on entire training data (Code is available in **model_pipeline_full.ipynb**).

In [171…
```
models.head(4).sort_values("MAE")
```

Out[171…

	model_name	model_object	MAE	MSE	RMSE	R_Square	Adj_r2
3	xgb_regressor	RandomizedSearchCV(cv=ShuffleSplit(n_splits=5,...	2191.557	78931857.050	8884.360	0.600	0.590
2	random_forest	RandomizedSearchCV(cv=ShuffleSplit(n_splits=5,...	2581.439	93851047.603	9687.680	0.520	0.510
1	decision_tree	RandomizedSearchCV(cv=ShuffleSplit(n_splits=5,...	3000.281	95842084.436	9789.900	0.510	0.500
0	linear_regression	RandomizedSearchCV(cv=ShuffleSplit(n_splits=5,...	3366.532	68205596.370	8258.670	0.650	0.650

In [174…
```
# Comparision is saved in Report Folder
output_path = r"C:\Users\kumar\OneDrive\Desktop\Akshat\insurance\Output\Reports"
now = datetime.now()
models.to_excel(str(output_path) + r"/Regression Model Metrics " + now.strftime("%d%B%y") +
".xlsx", index = False, header = True)
```

In [ ]: