In [ ]:

```
!pip install ipython-autotime
%load_ext autotime
```

Creating Pipeline

In [ ]:

```
## System info
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Out[ ]:

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 4854769104786567707, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 15469771520
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 4388253141469529268
 physical_device_desc: "device: 0, name: Tesla V100-SXM2-16GB, pci bus id: 0000:00:04.0,
compute capability: 7.0"]
```

```
time: 4.96 s
```

In [2]:

```
# loading library
import joblib
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import mean_absolute_error
```

```
time: 837 ms
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/My Drive/Applied AI Course/Assignments/23. Self Case Study 1
```

```
Mounted at /content/drive
/content/drive/My Drive/Applied AI Course/Assignments/23. Self Case Study 1
time: 22 s
```

In [4]:

```
# Loading Train and Test Dataset.
Train_Data = pd.read_csv('train.csv')
Test_Data = pd.read_csv('test.csv')
```

```
time: 6.74 s
```

In [ ]:

```
def preprocessing(Train_Test):
  '''
    This funtion takes pandas dataframe as input, perform preprocessing(replaceing the na
n value with most counted class) etc.
    and then return pandas dataframe as output

    data : pandas dataframe
    return : pandas dataframe
    '''

  Columns = [feature for feature in Train_Test.columns if 'c' in feature]
  if Train_Test[Columns].isnull().values.any() == True:
    Train_Test[Columns].apply(lambda col:fillna(np.nan))
  else:
    return Train_Test
  return Train_Test
```

```
time: 2.54 ms
```

In [ ]:

```
def featurization(Train_Data,Test_Data,Train_Test):
  '''
    This function takes pandas dataframe as input, create features and then return pandas
dataframe as output

    input : pandas dataframe
    return : pandas dataframe
    '''
  if __name__ == '__main__':
    cat_feature = [n for n in Train_Data.columns if n.startswith('cat')]
    for column in cat_feature:
      if Train_Data[column].nunique() != Test_Data[column].nunique():
        Unique_classes_Train = set(Train_Data[column].unique())
        Unique_classes_Test = set(Test_Data[column].unique())
        missing_train = Unique_classes_Train.difference(Unique_classes_Test)           #
set_A.difference(set_B) for (A - B)
        missing_test =  Unique_classes_Test.difference(Unique_classes_Train)

        All_misisng = missing_train.union(missing_test)
        # Replace all misisng categories with a common category instead of removing.
        def missing_common(x):
          if x in All_misisng:
            return np.nan
          return x

        Train_Test[column] = Train_Test[column].apply(lambda x: missing_common(x), 1)
# Axis 1 :: columns

      Train_Test[column] = pd.factorize(Train_Test[column].values, sort=True)[0]
  return Train_Test
```

```
time: 11.1 ms
```

In [ ]:

```
def final_Data(Train_Data,Test_Data):
  '''
    This function creates a final dataframe after all of the preprocessing, featurization
, prparation and normalization.

    input : pandas dataframe
    return : pandas dataframe
    '''

  Train_Data.drop(['id'], axis=1, inplace=True)
  Test_Data.drop(['id'], axis=1, inplace=True)
  Test_Data['loss'] = np.nan
  Train_Test = pd.concat((Train_Data, Test_Data)).reset_index(drop=True)

  # preprocessing
  Train_Test_final = preprocessing(Train_Test)
  # Featurization
  Train_Test_ = featurization(Train_Data,Test_Data,Train_Test_final)

  Train_Data_final = Train_Test_[Train_Test_['loss'].notnull()]
  Test_Data_final = Train_Test_[Train_Test_['loss'].isnull()]
  return Train_Data_final,Test_Data_final
```

```
time: 5.5 ms
```

In [ ]:

```
Train_Data_final,Test_Data_final = final_Data(Train_Data,Test_Data)
```

```
time: 18.4 s
```

In [ ]:

```
# saving csv to disk
Train_Data_final.to_csv('Train_Data_final.csv', index=False)
Test_Data_final.to_csv('Test_Data_final.csv', index=False)
```

```
time: 10.1 s
```

In [6]:

```
def predict(data):
  '''
    This function is used to take single or multiple observations, and predict probabilities for them

    input : single or multiple observations from a pandas dataframe
    return : predicted cliam amount for the observations
    '''
  data = data.drop(['loss'], axis=1, inplace=False)
  data = xgb.DMatrix(data)
  clf = joblib.load('allstateserevity.pkl')
  pred = clf.predict(data)
  return pred
```

```
time: 2.36 ms
```

In [5]:

```
def mae(data, labels):
  '''
    This function is used to take single or multiple observations and class labels, and predict MAE of each observation.

    input : single or multiple observations from a pandas dataframe
    labels : Data frame of ground truth values
    return : MAE of each observation
    '''

  data = data.drop(['loss'], axis=1, inplace=False)
  data = xgb.DMatrix(data)
  clf = joblib.load('allstateserevity.pkl')
  pred = clf.predict(data)
  return mean_absolute_error(labels, pred)
```

```
time: 2.59 ms
```

Single Observation Predicted

In [ ]:

```
sampled_train = Train_Data_final.sample(1)
sampled_train
```

Out[ ]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	cat13	cat14	cat15	cat16	cat17	cat18
158572	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

1 rows × 131 columns

```
time: 94.1 ms
```

In [ ]:

```
predict(sampled_train)
```

Out[ ]:

```
array([8.000027], dtype=float32)
```

```
time: 655 ms
```

In [ ]:

```
mae(sampled_train,sampled_train['loss'])
```

Out[ ]:

```
812.2299732971192
```

```
time: 245 ms
```

Multiple Observation Predicted

In [9]:

```
Train_Data_final = pd.read_csv("Train_Data_final.csv")
```

```
time: 1.95 s
```

In [10]:

```
sampled_train = Train_Data_final.sample(15)
sampled_train
```

Out[10]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	cat13	cat14	cat15	cat16	cat17	cat18
68489	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1346	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
88965	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
132559	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
128986	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
168928	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45545	0	1	0	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0
140333	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
62415	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
20913	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0
180526	0	1	0	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0
45325	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
157461	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
141045	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
106290	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

15 rows × 131 columns

```
time: 132 ms
```

In [11]:

```
predict(sampled_train)
```

Out[11]:

```
array([7.8227715, 7.284207 , 8.320862 , 8.542786 , 7.7198963, 7.3475814,
       8.268406 , 7.9695573, 7.571107 , 8.540485 , 8.654859 , 7.2890344,
       8.0878525, 7.9571137, 7.900096 ], dtype=float32)
```

```
time: 1.56 s
```

In [13]:

```
mae(sampled_train,sampled_train['loss'])
```

Out[13]:

```
2854.0242257207233
```

```
time: 283 ms
```