In [1]:
```
##Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
# Loading the train data and drop 'Customer_ID' becos it won't be useful
train_dataset = pd.read_csv('train.csv')
train_dataset.head()
```

Out[2]:

	Customer_ID	months_as_customer	age	insured_sex	insured_education_level	insured_occu
0	Customer_541	239	41	FEMALE	JD	farming-
1	Customer_440	108	31	MALE	Masters	protecti
2	Customer_482	116	30	MALE	JD	handlers-c
3	Customer_422	8	21	MALE	High School	handlers-c
4	Customer_778	161	38	MALE	PhD	priv-hous

5 rows × 37 columns

In [3]:
```
test_dataset = pd.read_csv('test.csv')
test_dataset.head()
```

Out[3]:

	Customer_ID	months_as_customer	age	insured_sex	insured_education_level	insured_occu
0	Customer_521	5	26	FEMALE	PhD	farming-
1	Customer_737	160	33	FEMALE	High School	exec-mar
2	Customer_740	385	51	FEMALE	MD	craft
3	Customer_660	446	57	MALE	College	adm-
4	Customer_411	84	29	FEMALE	High School	machine-op

5 rows × 36 columns

In [4]:
```
## Concatenate train_test and test_data to data
data = pd.concat((train_dataset, test_dataset)).reset_index(drop=True)
data.head()
```

Out[4]:

	Customer_ID	months_as_customer	age	insured_sex	insured_education_level	insured_occu
0	Customer_541	239	41	FEMALE	JD	farming-
1	Customer_440	108	31	MALE	Masters	protecti
2	Customer_482	116	30	MALE	JD	handlers-c
3	Customer_422	8	21	MALE	High School	handlers-c
4	Customer_778	161	38	MALE	PhD	priv-hous

5 rows × 37 columns

In [5]:
```
## Deal with null values
```

```
data.isnull().sum()
```

Out[5]:
```
Customer_ID                    0
months_as_customer             0
age                            0
insured_sex                    0
insured_education_level        0
insured_occupation             0
insured_hobbies                0
insured_relationship           0
capital-gains                  0
capital-loss                   0
policy_number                  0
policy_bind_date               0
policy_state                   0
policy_csl                     0
policy_deductable              0
incident_location              0
incident_hour_of_the_day       0
number_of_vehicles_involved    0
property_damage                0
bodily_injuries                0
policy_annual_premium          0
umbrella_limit                 0
insured_zip                    0
incident_date                  0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          0
incident_state                 0
incident_city                  0
witnesses                      0
police_report_available        0
auto_make                      0
auto_model                     0
auto_year                      0
_c39                        1000
total_claim_amount           300
dtype: int64
```

In [7]:
```
data = data.drop('_c39', axis = 1)
#data['_c39'].isnull().sum()
```

In [8]:
```
##All the missing values is replaced
data.isnull().sum().sort_values(ascending=False)
```

Out[8]:
```
total_claim_amount             300
auto_year                        0
incident_location                0
policy_deductable                0
policy_csl                       0
policy_state                     0
policy_bind_date                 0
policy_number                    0
capital-loss                     0
capital-gains                    0
insured_relationship             0
insured_hobbies                  0
insured_occupation               0
insured_education_level          0
insured_sex                      0
age                              0
months_as_customer               0
incident_hour_of_the_day         0
number_of_vehicles_involved      0
property_damage                  0
authorities_contacted            0
```

```
auto_model                   0
auto_make                    0
police_report_available      0
witnesses                    0
incident_city                0
incident_state               0
incident_severity            0
bodily_injuries              0
collision_type               0
incident_type                0
incident_date                0
insured_zip                  0
umbrella_limit               0
policy_annual_premium        0
Customer_ID                  0
dtype: int64
```

In [9]:
```
## Categorical data
categorical = (data.dtypes == "object")
categorical_list = list(categorical[categorical].index)
print(categorical_list)
```

```
['Customer_ID', 'insured_sex', 'insured_education_level', 'insured_occupatio
n', 'insured_hobbies', 'insured_relationship', 'policy_bind_date', 'policy_sta
te', 'policy_csl', 'incident_location', 'property_damage', 'incident_date', 'i
ncident_type', 'collision_type', 'incident_severity', 'authorities_contacted',
'incident_state', 'incident_city', 'police_report_available', 'auto_make', 'au
to_model']
```

In [10]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['insured_sex'] = le.fit_transform(data['insured_sex'])
data['insured_education_level'] = le.fit_transform(data['insured_education_le
data['insured_occupation'] = le.fit_transform(data['insured_occupation'])
data['insured_hobbies'] = le.fit_transform(data['insured_hobbies'])
data['insured_relationship'] = le.fit_transform(data['insured_relationship'])
data['policy_bind_date'] = le.fit_transform(data['policy_bind_date'])
data['policy_state'] = le.fit_transform(data['policy_state'])
data['policy_csl'] = le.fit_transform(data['policy_csl'])
data['incident_location'] = le.fit_transform(data['incident_location'])
data['property_damage'] = le.fit_transform(data['property_damage'])
data['incident_date'] = le.fit_transform(data['incident_date'])
data['incident_type'] = le.fit_transform(data['incident_type'])
data['collision_type'] = le.fit_transform(data['collision_type'])
data['incident_severity'] = le.fit_transform(data['incident_severity'])
data['authorities_contacted'] = le.fit_transform(data['authorities_contacted'
data['incident_state'] = le.fit_transform(data['incident_state'])
data['incident_city'] = le.fit_transform(data['incident_city'])
data['police_report_available'] = le.fit_transform(data['police_report_availa
data['auto_make'] = le.fit_transform(data['auto_make'])
data['auto_model'] = le.fit_transform(data['auto_model'])
```

In [22]:
```
## Split the data into train and test data
train_data=data[:700]
test_data=data[700:]

print(train_data.shape)
print(test_data.shape)
```

```
(700, 36)
(300, 36)
```

In [23]:
```
train_data.drop(['Customer_ID'], axis ='columns', inplace= True)
```

In [24]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 700 entries, 0 to 699
Data columns (total 35 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           700 non-null    int64  
 1   age                          700 non-null    int64  
 2   insured_sex                  700 non-null    int64  
 3   insured_education_level      700 non-null    int64  
 4   insured_occupation           700 non-null    int64  
 5   insured_hobbies              700 non-null    int64  
 6   insured_relationship         700 non-null    int64  
 7   capital-gains                700 non-null    int64  
 8   capital-loss                 700 non-null    int64  
 9   policy_number                700 non-null    int64  
 10  policy_bind_date             700 non-null    int64  
 11  policy_state                 700 non-null    int64  
 12  policy_csl                   700 non-null    int64  
 13  policy_deductable            700 non-null    int64  
 14  incident_location            700 non-null    int64  
 15  incident_hour_of_the_day     700 non-null    int64  
 16  number_of_vehicles_involved  700 non-null    int64  
 17  property_damage              700 non-null    int64  
 18  bodily_injuries              700 non-null    int64  
 19  policy_annual_premium        700 non-null    float64
 20  umbrella_limit               700 non-null    int64  
 21  insured_zip                  700 non-null    int64  
 22  incident_date                700 non-null    int64  
 23  incident_type                700 non-null    int64  
 24  collision_type               700 non-null    int64  
 25  incident_severity            700 non-null    int64  
 26  authorities_contacted        700 non-null    int64  
 27  incident_state               700 non-null    int64  
 28  incident_city                700 non-null    int64  
 29  witnesses                    700 non-null    int64  
 30  police_report_available      700 non-null    int64  
 31  auto_make                    700 non-null    int64  
 32  auto_model                   700 non-null    int64  
 33  auto_year                    700 non-null    int64  
 34  total_claim_amount           700 non-null    float64
dtypes: float64(2), int64(33)
memory usage: 191.5 KB
```

In [25]:

```
test_data.drop(['Customer_ID'], axis ='columns', inplace= True)
test_data.drop(['total_claim_amount'], axis = 'columns', inplace= True)
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 700 to 999
Data columns (total 34 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           300 non-null    int64  
 1   age                          300 non-null    int64  
 2   insured_sex                  300 non-null    int64  
 3   insured_education_level      300 non-null    int64  
 4   insured_occupation           300 non-null    int64  
 5   insured_hobbies              300 non-null    int64  
 6   insured_relationship         300 non-null    int64  
 7   capital-gains                300 non-null    int64  
 8   capital-loss                 300 non-null    int64  
 9   policy_number                300 non-null    int64  
 10  policy_bind_date             300 non-null    int64  
 11  policy_state                 300 non-null    int64  
 12  policy_csl                   300 non-null    int64  
 13  policy_deductable            300 non-null    int64  
 14  incident_location            300 non-null    int64  
 15  incident_hour_of_the_day     300 non-null    int64  
 16  number_of_vehicles_involved  300 non-null    int64  
 17  property_damage              300 non-null    int64  
```

```
 18  bodily_injuries              300 non-null    int64
 19  policy_annual_premium        300 non-null    float64
 20  umbrella_limit               300 non-null    int64
 21  insured_zip                  300 non-null    int64
 22  incident_date                300 non-null    int64
 23  incident_type                300 non-null    int64
 24  collision_type               300 non-null    int64
 25  incident_severity            300 non-null    int64
 26  authorities_contacted        300 non-null    int64
 27  incident_state               300 non-null    int64
 28  incident_city                300 non-null    int64
 29  witnesses                    300 non-null    int64
 30  police_report_available      300 non-null    int64
 31  auto_make                    300 non-null    int64
 32  auto_model                   300 non-null    int64
 33  auto_year                    300 non-null    int64
dtypes: float64(1), int64(33)
memory usage: 79.8 KB
```

In [27]:
```
y = train_data['total_claim_amount']
X = train_data.drop(['total_claim_amount'], axis = 1)
#test_data=test_data.drop(['total_claim_amount'], axis = 1).iloc[:-1].values
index_col = test_dataset['Customer_ID']
```

In [28]:
```
print(y.shape, X.shape, test_data.shape)
```

```
(700,) (700, 34) (300, 34)
```

In [29]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra

## Scale the data
from sklearn.preprocessing import StandardScaler

# creating a standard scaler
sc = StandardScaler()

# feeding independents sets into the standard scaler
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [30]:
```
print(X_train.shape, X_test.shape, y_train.shape)
```

```
(560, 34) (140, 34) (560,)
```

In [32]:
```
import xgboost as xgb
from xgboost import XGBRegressor

xgbr = xgb.XGBRegressor(verbosity=0)

xgbr
```

Out[32]:
```
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=Non
e,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=0)
```

In [33]:
```
xgbr.fit(X_train,y_train)
```

Out[33]:
```
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=0.300000012, max_delta_step=0, max_depth=6,
min_child_weight=1, missing=nan, monotone_constraints='()',
n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
tree_method='exact', validate_parameters=1, verbosity=0)
```

In [34]:
```
ypred_xgbr = xgbr.predict(X_test)
print(ypred_xgbr)
```

```
[  8425.548   69267.805   82705.66     7671.695   84419.08    84247.79
  98116.914   78107.36    82471.19    57718.832   80842.266   92624.22
  67125.36    98787.62    91627.59    79903.94    88542.555   85931.16
  91887.48    93781.18    65102.523   71475.71    73228.01    72889.19
  91957.47    72549.36    98302.99    11999.929   98471.8     83646.13
  87044.33    84072.59    77842.5     73969.65    75601.76    77454.84
  90021.305   12181.744  118503.266   79297.34    57721.84    93808.7
  89464.83    90421.93    99026.96    90159.695   57824.97    80216.375
  87631.1     79008.26    70896.01    72558.      71968.27    81183.31
  74507.09   107025.73    79735.23    85632.91    63245.938   11548.167
 100372.06    93499.445  100733.37    15957.127   85933.01    66816.06
  61623.73    93846.445   95652.516   82187.63   102618.62    87167.94
  88987.766   79675.64    86026.94    92750.35     6477.5127  82882.945
  48026.41    82818.695   97730.78    83831.17    71023.07    99914.65
  82718.19    76233.13    81586.95    85204.51    11482.187   16417.74
  75738.836   95077.45     8558.778   92077.86    81289.56    71408.016
  85079.695   74963.03    96414.56    75598.414   69027.83     5221.8633
  13599.326   80574.82    82009.93    78788.43    87187.27    92581.695
  80405.32     1406.7446   7666.905   75680.984   80297.664   83188.445
 101217.34     3094.4949  77349.55     6669.7305  89347.914   76248.25
  90655.414   84750.336   82098.695   96895.695   13148.876    9785.976
  93550.945   76942.55    78707.734    9079.502    9412.716   95287.44
  91232.234    5252.0366  82001.12    87306.9     16253.604   87815.88
  82927.13    83556.22   ]
```

In [37]:
```
#from sklearn.metrics import mean_squared_error
from sklearn import metrics #import accuracy_score,mean_squared_error, r2_sco
from math import sqrt

MSE = metrics.mean_squared_error(y_test, ypred_xgbr)

RMSE = np.sqrt(MSE)
print('RMSE: ', RMSE)
```

```
RMSE:  19966.591919906707
```

In [43]:
```
from sklearn.model_selection import RandomizedSearchCV

xgbr = XGBRegressor()

parameters = {"objective":['reg:squarederror'],
              "learning_rate": [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3], #so call
              "min_child_weight": [1, 3, 5],
              "subsample": [0.5, 0.7],
              "colsample_bytree": [0.5, 0.7],
              "n_estimators": [100, 200, 400, 500, 600, 800, 1000],
              "max_depth": [3, None],
              "max_features": [1, 9],
              "min_samples_leaf": [1, 9],
              "criterion": ["gini", "entropy"]}

#random_search = RandomizedSearchCV(xgbr, param_distributions=parameters,
#                                   n_iter=10)
```

In [47]:
```
#def hyperParameterTuning(X_train, y_train):
#    param_tuning = {'learning_rate': [0.01, 0.1, 0.2, 0.3],
#                    'max_depth': [3, 5, 7],
```

```
#                     'min_child_weight': [1, 3, 5],
#                     'subsample': [0.5, 0.7],
#                     'colsample_bytree': [0.5, 0.7],
#                     'n_estimators' : [100, 500, 800, 1000],
#                     'objective': ['reg:squarederror']   }



xgbr_random_search = RandomizedSearchCV(xgbr, parameters,
                                        scoring= 'neg_mean_squared_error',  #
                                        cv = 10,
                                        n_jobs = -1,
                                        verbose = 0)

xgbr_random_search.fit(X_train,y_train)

print(xgbr_random_search.best_score_)
print(xgbr_random_search.best_params_)
```

```
-472341554.0386464
{'subsample': 0.5, 'objective': 'reg:squarederror', 'n_estimators': 1000, 'min
_samples_leaf': 9, 'min_child_weight': 1, 'max_features': 1, 'max_depth': Non
e, 'learning_rate': 0.1, 'criterion': 'gini', 'colsample_bytree': 0.7}
```

In [49]:

```
y_pred_X = xgbr_random_search.predict(X_test)

MSE_X = metrics.mean_squared_error(y_test, y_pred_X)

print("MSE: ", MSE_X)

print('RMSE: ', np.sqrt(MSE_X))
```

```
MSE:  337873788.5960296
RMSE:  18381.343492683813
```

In [50]:

```
y_pred_X = xgbr_random_search.predict(test_data.values)

submission = pd.DataFrame({'Customer_ID':index_col,'total_claim_amount':y_pre

# Save results
submission.to_csv("submission_xgbr_random.csv",index=False)
```

In [58]:

```
#from sklearn.model_selection import GridSearchCV

def hyperParameterTuning(X_train, y_train):
    param_tuning = {'learning_rate': [0.01, 0.1, 0.2],
                    'max_depth': [3, 5, 7],
                    'min_child_weight': [1, 3, 5],
                    'subsample': [0.5, 0.7],
                    'colsample_bytree': [0.5, 0.7],
                    'n_estimators' : [100, 500, 800, 1000],
                    'objective': ['reg:squarederror'] }


    xgbr = XGBRegressor()

    random_search = RandomizedSearchCV(estimator = xgbr,
                           param_grid = param_tuning,
                           scoring_MSE = 'neg_mean_squared_error',  #MSE
                           cv = 5,
                           n_jobs = -1,
                           verbose = 1)

    #random_search.fit(X_train,y_train)
```

In [63]:
```
random_search.fit(X_train,y_train)
```

Out[63]:
```
RandomizedSearchCV(estimator=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=0.300000012,
                                          max_delta_step=0, max_depth=6,
                                          min_child_weight=1, missing=nan,
                                          monotone_constraints='()',
                                          n_estimators=100, n_jobs=4,
                                          num_parallel...
                                          validate_parameters=1, verbosity=0),
                   param_distributions={'colsample_bytree': [0.5, 0.7],
                                        'criterion': ['gini', 'entropy'],
                                        'learning_rate': [0.0001, 0.001, 0.01,
                                                          0.1, 0.2, 0.3],
                                        'max_depth': [3, None],
                                        'max_features': [1, 9],
                                        'min_child_weight': [1, 3, 5],
                                        'min_samples_leaf': [1, 9],
                                        'n_estimators': [100, 200, 400, 500,
                                                         600, 800, 1000],
                                        'objective': ['reg:squarederror'],
                                        'subsample': [0.5, 0.7]})
```

In [67]:
```
print(random_search.best_score_)
```

```
0.6657636325600678
```

In [68]:
```
print(random_search.best_params_)
```

```
{'subsample': 0.7, 'objective': 'reg:squarederror', 'n_estimators': 500, 'min_
samples_leaf': 1, 'min_child_weight': 3, 'max_features': 1, 'max_depth': 3, 'l
earning_rate': 0.01, 'criterion': 'gini', 'colsample_bytree': 0.5}
```

In [69]:
```
y_pred = random_search.predict(X_test)
```

In [70]:
```
MSE = metrics.mean_squared_error(y_test, y_pred)

print("MSE: ", MSE)

print('RMSE: ', np.sqrt(MSE))
```

```
MSE:  332149887.81017196
RMSE:  18224.979775302138
```

In [71]:
```
y_pred = random_search.predict(test_data.values)

submission = pd.DataFrame({'Customer_ID':index_col,'total_claim_amount':y_pre

# Save results
submission.to_csv("submission_random_search.csv",index=False)
```

In [ ]:
```

```