Zurich Auto insurance_model_ML-Final Version - LAAVANYA GANESH

May 8, 2018

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
```

Reading in Training Data

In [2]:
```
df = pd.read_csv('auto_policies_2017.csv')
```

Data Pre-processing to fill in missing values and create new variables

In [3]:
```
df['date'] = pd.to_datetime(df['date_of_birth'])
```

In [4]:
```
df['year'] = df['date'].dt.year
```

In [5]:
```
df1 = df.groupby('agecat')['year'].min()
df2 = df.groupby('agecat')['year'].max()
```

In [6]:
```
df1
```

Out[6]:
```
agecat
1.0    1990
2.0    1980
3.0    1970
4.0    1960
5.0    1950
6.0    1923
Name: year, dtype: int64
```

In [7]:
```
df2
```

```
Out[7]: agecat
        1.0    1999
        2.0    1989
        3.0    1979
        4.0    1969
        5.0    1959
        6.0    1949
        Name: year, dtype: int64
```

```
In [8]: conditions = [
            (df['year'] >= 1990) & (df['year'] <= 1999),
            (df['year'] >= 1980) & (df['year'] <= 1989),
            (df['year'] >= 1970) & (df['year'] <= 1979),
            (df['year'] >= 1960) & (df['year'] <= 1969),
            (df['year'] >= 1950) & (df['year'] <= 1959),]
        choices = [1,2,3,4,5]
        df['agecat_imputed'] = np.select(conditions, choices,default=6)
```

```
In [9]: df3 = df.groupby('agecat_imputed')['credit_score'].mean()
```

```
In [10]: df3
```

```
Out[10]: agecat_imputed
         1    577.405131
         2    594.119094
         3    659.271850
         4    677.017937
         5    737.101795
         6    691.745888
         Name: credit_score, dtype: float64
```

```
In [11]: df4 = df.groupby('area')['traffic_index'].mean()
```

```
In [12]: df4
```

```
Out[12]: area
         A     80.201316
         B    120.211648
         C    128.474092
         D     99.400110
         E     45.018979
         F    115.208244
         Name: traffic_index, dtype: float64
```

```
In [13]: conditions1 = [
             (df['agecat_imputed'] ==1),
             (df['agecat_imputed'] ==2),
             (df['agecat_imputed'] ==3),
             (df['agecat_imputed'] ==4),
```

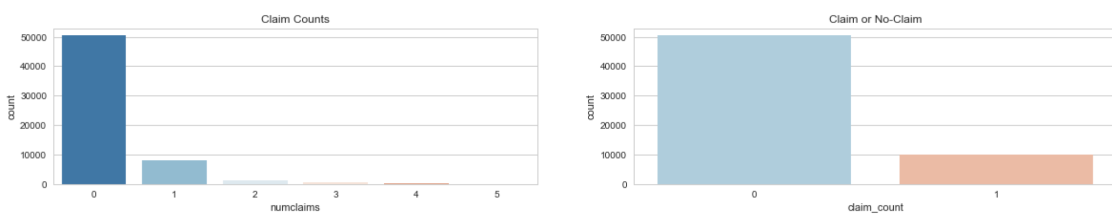
```
    (df['agecat_imputed'] ==5)]
choices1 = [577.405131,594.119094,659.271850,677.017937,737.101795]
df['credit_score_imputed'] = np.select(conditions1, choices1,default=691.745888)
```

In [14]:
```
df['credit_score_imp']=(df.credit_score_imputed/df.credit_score_imputed.max())*100
```

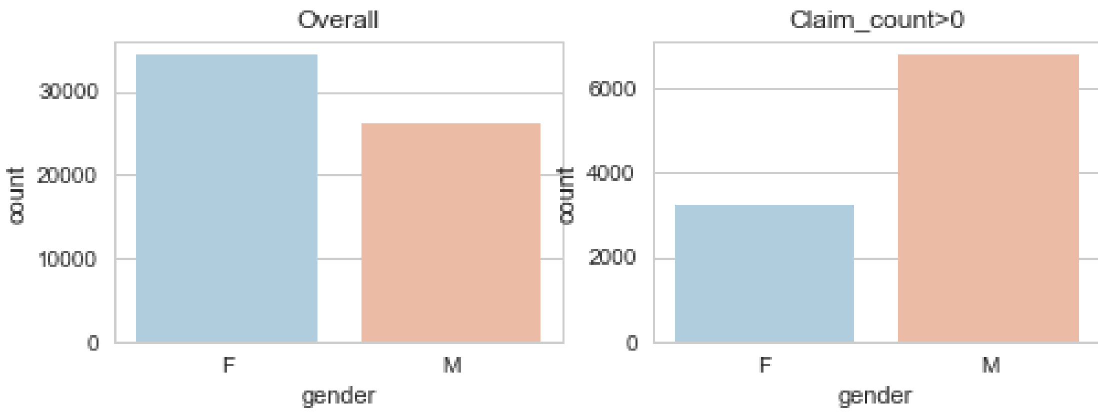
In [15]:
```
conditions2 = [
    (df['area'] =='A'),
    (df['area'] =='B'),
    (df['area'] =='C'),
    (df['area'] =='D'),
    (df['area'] =='E')]
choices2 = [80.201316,120.211648,128.474092,99.400110,45.018979]
df['traffic_index_imputed'] = np.select(conditions2, choices2,default=115.208244)
```

0.0.1 Exploratory Data Analysis

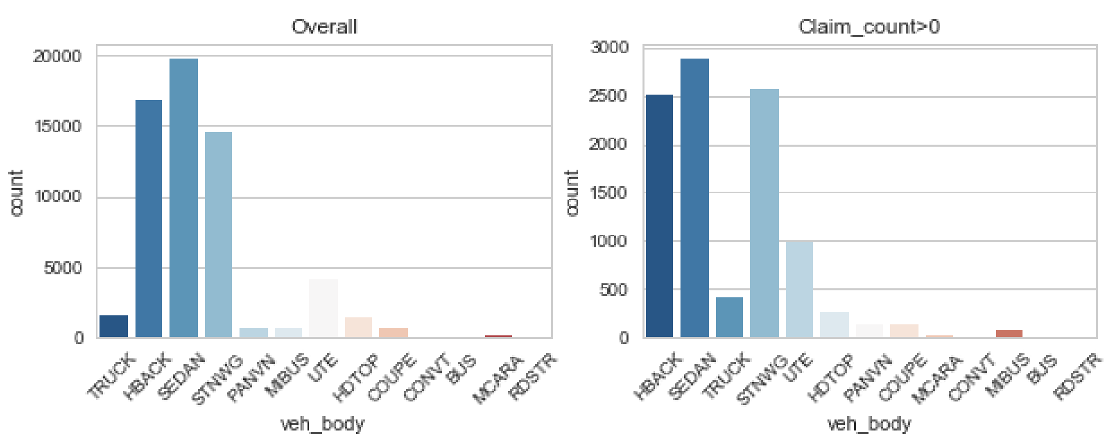
In [16]:
```
# Plot the Claim counts
fig, (ax1, ax2) = plt.subplots(ncols=2)
fig.set_size_inches(20,3)
g = sns.countplot(x='numclaims',data=df, palette='RdBu_r', ax=ax1)
title = g.set_title('Claim Counts')

# Since there are very few incidences with claim count>1, we club them together with
df['claim_count'] = df['numclaims'].apply(lambda x:1 if x>0 else 0)
g = sns.countplot(x='claim_count',data=df, palette='RdBu_r', ax=ax2)
title = g.set_title('Claim or No-Claim')
```



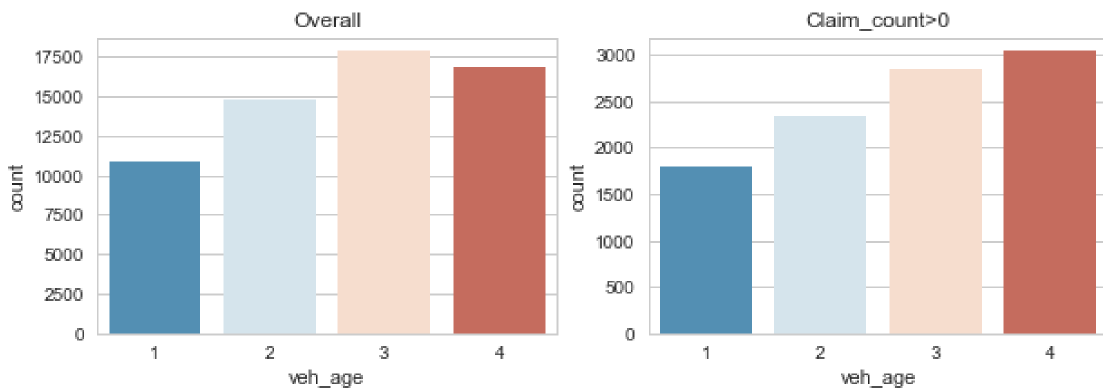
In [17]:
```
#Sort by sex
fig, (ax1, ax2) = plt.subplots(ncols=2)
fig.set_size_inches(8,2.5)
g = sns.countplot(x='gender',data=df.sort_values('gender'), palette='RdBu_r', ax=ax1)
title = g.set_title('Overall')
g = sns.countplot(x='gender',data=df[df['claim_count']>0].sort_values('gender'), pale
title = g.set_title('Claim_count>0')
```



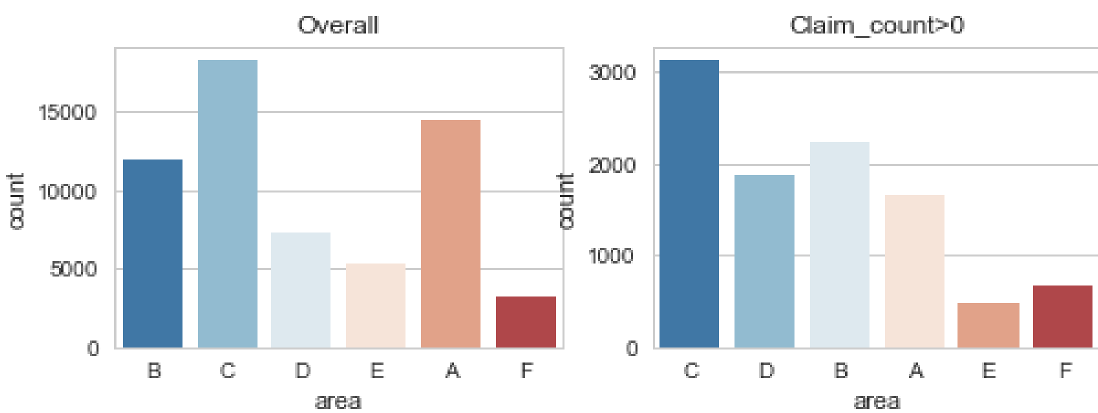

```
In [18]: #Visualize Vehicle Body Type
         fig, (ax1, ax2) = plt.subplots(ncols=2)
         fig.set_size_inches(10,3)
         g = sns.countplot(x='veh_body',data=df, palette='RdBu_r', ax=ax1)
         label = g.set_xticklabels(g.get_xticklabels(), rotation=45)
         title = g.set_title('Overall')
         g = sns.countplot(x='veh_body',data=df[df['claim_count']>0], palette='RdBu_r', ax=ax2
         label = g.set_xticklabels(g.get_xticklabels(), rotation=45)
         title = g.set_title('Claim_count>0')
```



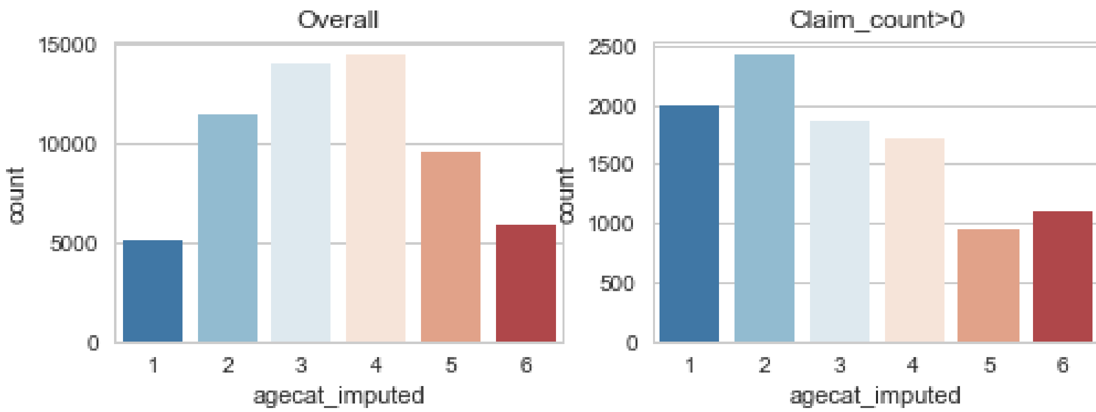

```
In [19]: #Visualize Vehicle Age
         fig, (ax1, ax2) = plt.subplots(ncols=2)
         fig.set_size_inches(10,3)
         g = sns.countplot(x='veh_age',data=df, palette='RdBu_r', ax=ax1)
         title = g.set_title('Overall')
         g = sns.countplot(x='veh_age',data=df[df['claim_count']>0], palette='RdBu_r', ax=ax2)
         title = g.set_title('Claim_count>0')
```



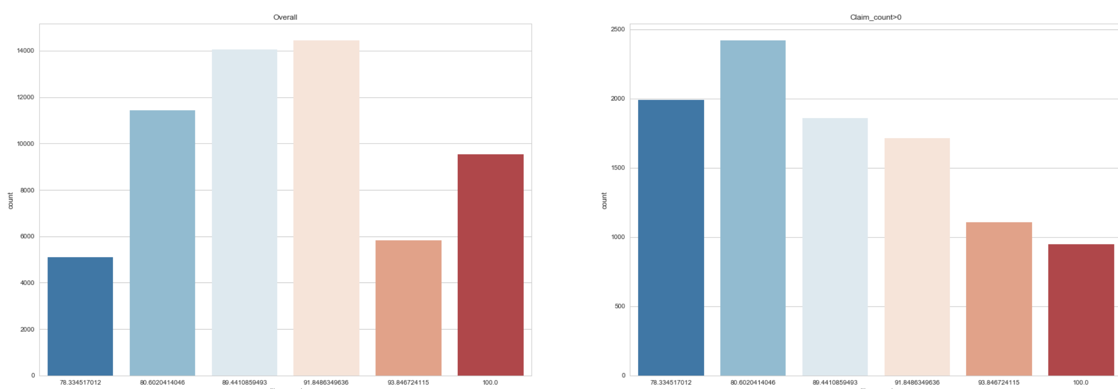

```
In [20]: #Visualize Area
         fig, (ax1, ax2) = plt.subplots(ncols=2)
         fig.set_size_inches(8,2.5)
         g = sns.countplot(x='area',data=df, palette='RdBu_r', ax=ax1)
         title = g.set_title('Overall')
         g = sns.countplot(x='area',data=df[df['claim_count']>0], palette='RdBu_r', ax=ax2)
         title = g.set_title('Claim_count>0')
```



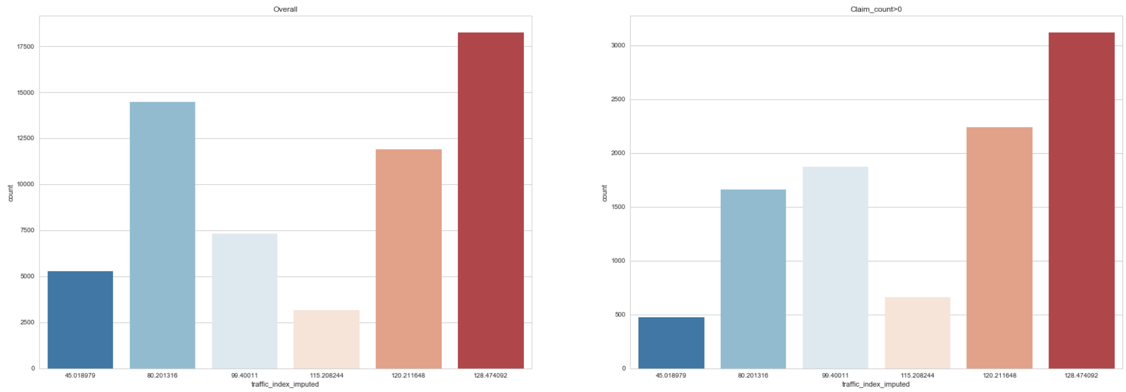

```
In [21]: #Visualize Driver Age
         fig, (ax1, ax2) = plt.subplots(ncols=2)
         fig.set_size_inches(8,2.5)
         g = sns.countplot(x='agecat_imputed',data=df, palette='RdBu_r', ax=ax1)
         title = g.set_title('Overall')
         g = sns.countplot(x='agecat_imputed',data=df[df['claim_count']>0], palette='RdBu_r', a
         title = g.set_title('Claim_count>0')
```



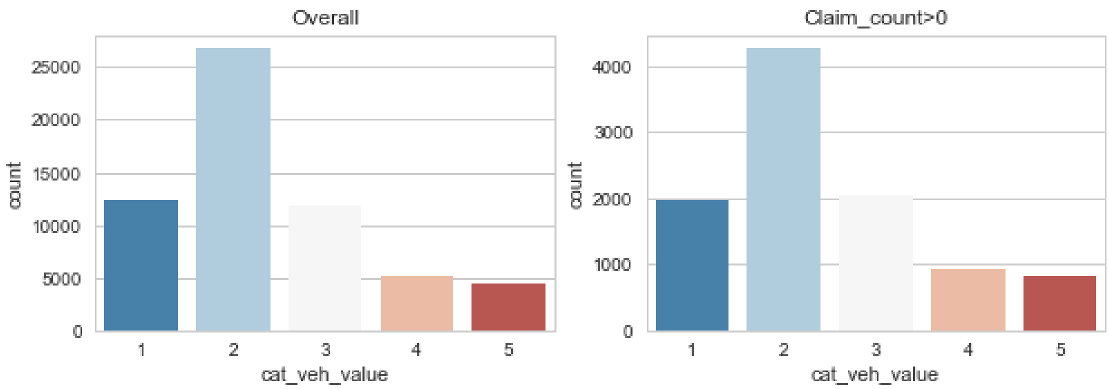
In [22]:
```
#Visualize Credit_score
fig, (ax1, ax2) = plt.subplots(ncols=2)
fig.set_size_inches(30,10)
g = sns.countplot(x='credit_score_imp',data=df, palette='RdBu_r', ax=ax1)
title = g.set_title('Overall')
g = sns.countplot(x='credit_score_imp',data=df[df['claim_count']>0], palette='RdBu_r'
title = g.set_title('Claim_count>0')
```



In [23]:
```
#Visualize Traffic_index
fig, (ax1, ax2) = plt.subplots(ncols=2)
fig.set_size_inches(30,10)
g = sns.countplot(x='traffic_index_imputed',data=df, palette='RdBu_r', ax=ax1)
title = g.set_title('Overall')
g = sns.countplot(x='traffic_index_imputed',data=df[df['claim_count']>0], palette='Rdl
title = g.set_title('Claim_count>0')
```




```
In [24]: #Make Vehicle Value categorical
         df['cat_veh_value'] = df['veh_value'].apply(lambda x:1 if x<=1 else 2 if x<=2 else 3 i
         #Visualize Vehicle Value
         fig, (ax1, ax2) = plt.subplots(ncols=2)
         fig.set_size_inches(10,3)
         g = sns.countplot(x='cat_veh_value',data=df, palette='RdBu_r', ax=ax1)
         title = g.set_title('Overall')
         g = sns.countplot(x='cat_veh_value',data=df[df['claim_count']>0], palette='RdBu_r', ax
         title = g.set_title('Claim_count>0')
```



0.1 Creating the Model

0.1.1 One Hot Encoding, creating dummy variables of each attribute

```
In [25]: #Function to Prepare the data

         def prep_data(df):
             #Prepare the data
             df_dummy = df[['pol_number','credit_score_imp','traffic_index_imputed','claim_cou
```

```
    #Make dummy variables using Pandas
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['veh_age'],prefix="veh_age")],ax
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['agecat_imputed'],prefix="agecat_
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['veh_body'],prefix="veh_body")],a
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['gender'],prefix="gender")],axis=
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['area'],prefix="area")],axis=1)
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['cat_veh_value'],prefix="cat_veh_


    #We are trying to predict whether there is claim or no claim
    df_dummy['claim_count'] = df_dummy['claim_count'].apply(lambda x: 1 if x>0 else 0)
    return(df_dummy)
```

```
In [26]: df_train = df.copy()
```

Splitting Data into train and validation

```
In [27]: from sklearn.model_selection import train_test_split
         train, test = train_test_split(df_train, test_size=0.2, random_state=12345)
```

```
In [28]: #We create a separate X_train and X_test dataset because we would require the categor
         #X_train and X_test dataframes have only dummy variables
         X_train = prep_data(train)
         X_test = prep_data(test)

         #Create two additional columns for the individual probabilities of claim (prob1) and
         X_train['prob0'] = np.zeros(len(X_train))
         X_train['prob1'] = np.zeros(len(X_train))

         X_test['prob0'] = np.zeros(len(X_test))
         X_test['prob1'] = np.zeros(len(X_test))
```

0.1.2 Logistic Regression to get the probabilities of atleast one claim or no claim

```
In [29]: def logfunc(X_train, X_test):

             #In the first part, we use Logistic Regression to get the probabilities of claim
             from sklearn.linear_model import LogisticRegression
             logmodel = LogisticRegression(C=1000, class_weight='balanced')

             n=10

             for i in range (0,n):

                 #Choose 3000 Negative Classes
                 X_train_temp = X_train.loc[np.random.choice(X_train[X_train['claim_count']==0]
                 #Choose 3000 Positive Classes
                 X_train_pos = X_train.loc[np.random.choice(X_train[X_train['claim_count']==1]
```

```
            #Append the positive classes
            X_train_temp = X_train_temp.append(X_train_pos, ignore_index=True)

            #Separate into X and Y to train the model
            y_train_temp = X_train_temp['claim_count']
            X_train_temp.drop(['pol_number','claim_count','claimcst0','prob0','prob1'], ax


            #Fit the Logistic Regression Model
            logmodel.fit(X_train_temp,y_train_temp)
            X_train[['prob0','prob1']] = X_train[['prob0','prob1']] + logmodel.predict_pro
            X_test[['prob0','prob1']] = X_test[['prob0','prob1']] + logmodel.predict_proba

        #Divide the log_proba and log_probb values by 10 to get the average log probabili
        X_train['prob0']=X_train['prob0']/n
        X_train['prob1']=X_train['prob1']/n

        X_test['prob0']=X_test['prob0']/n
        X_test['prob1']=X_test['prob1']/n

        #Check the metrics on training and test data
        logProb = 1
        temp = logProb > (X_test['prob0']/X_test['prob1'])
        X_test['predicted_claim_count']= [1 if (p==True) else 0 for p in temp]

        #Metrics on Test set
        from sklearn.metrics import classification_report
        print(classification_report(X_test['claim_count'],X_test['predicted_claim_count']]

        #Calculate Claim Frequency for Test Set
        X_test['predicted_freq']= X_test['prob1'].divide(X_test['prob0'])
        X_test['predicted_freq']= X_test['predicted_freq'].apply(lambda x: x**3.65)
        return logmodel, X_test, X_train
```

In [30]:
```
logmodel, X_test , X_train = logfunc(X_train,X_test)
```

```
             precision    recall  f1-score   support

          0       0.91      0.67      0.77     10052
          1       0.29      0.69      0.41      2027

avg / total       0.81      0.67      0.71     12079
```

In [31]:
```
X_test1 = X_test.copy() # Gradient Boosted
```

In [32]:
```
X_test2 = X_test.copy() # GLM
```

In [33]:
```
X_test3 = X_test.copy() # Decision Trees
```

In [34]:
```
X_test4 = X_test.copy()  # Extra Trees
```

In [35]:
```
X_test5 = X_test.copy() # Random Forest
```

0.1.3 Different Functions for Regression Analysis to Predict Claim Cost

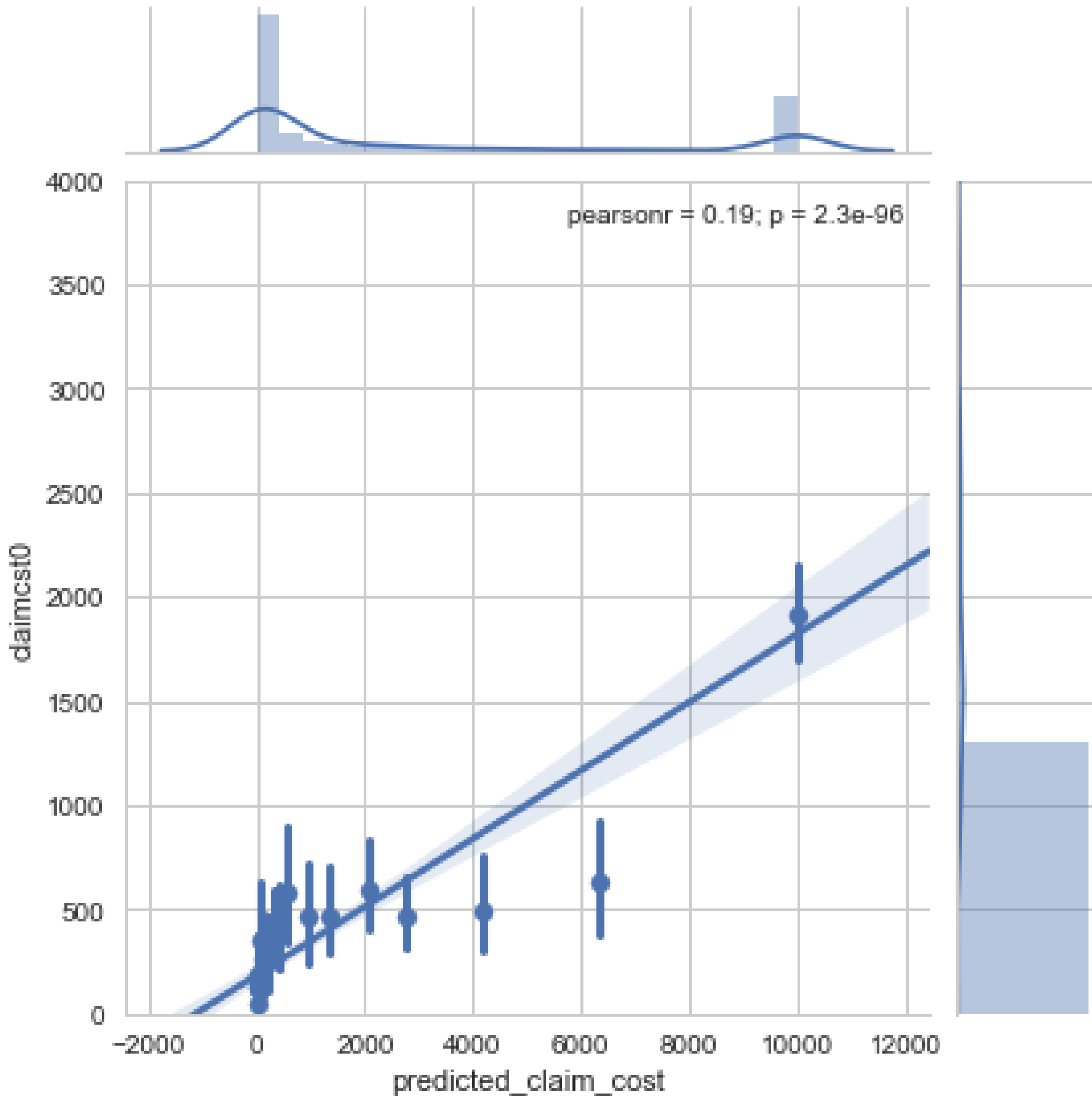
Gradient Boosted Function

In [36]:
```
def xgboostfunc(X_train, X_test1):
    #Part 2: Use Regression for predicting Claim Severity

    X_test1.drop('predicted_claim_count',axis=1,inplace=True, errors='ignore')

    #Train only on the subset of positive claim counts
    X_train_regress = X_train[X_train['claim_count']>0].copy()
    y_train_regress = np.log(X_train[X_train['claim_count']>0]['claimcst0'])

    #Implement XGBoost for regression
    import xgboost as xgb
    num_round = 1

    T_train_xgb = xgb.DMatrix(X_train_regress.drop(['pol_number','claim_count','claim
    params = {"objective": "reg:linear"}
    gbm = xgb.train(dtrain=T_train_xgb,params=params)
    predictions = gbm.predict(xgb.DMatrix(X_test1.drop(['pol_number','claim_count','cl
    X_test1['predicted_claim_cost']=[np.exp(p) for p in predictions]
    X_test1['predicted_claim_cost']=X_test1['predicted_claim_cost'].multiply(X_test1[
    return gbm, X_test1.copy()
```

In [37]:
```
gbm, X_test1 = xgboostfunc(X_train,X_test1)
```

In [38]:
```
sns.jointplot(X_test1.predicted_claim_cost.clip(upper=10000), X_test1.claimcst0, kind=
```

Out[38]:
```
<seaborn.axisgrid.JointGrid at 0x10e0d3be0>
```




```
In [39]: from sklearn.metrics import mean_squared_error
         mean_squared_error(X_test1.claimcst0, X_test1.predicted_claim_cost.clip(upper=10000))
```

```
Out[39]: 5212.8275872903332
```

Generalised Linear Model Function

```
In [40]: def glmfunc(X_train, X_test2):

             import statsmodels.api as sm

             X_test2.drop(['predicted_claim_count','predicted_claim_cost'],axis=1,inplace=True
```

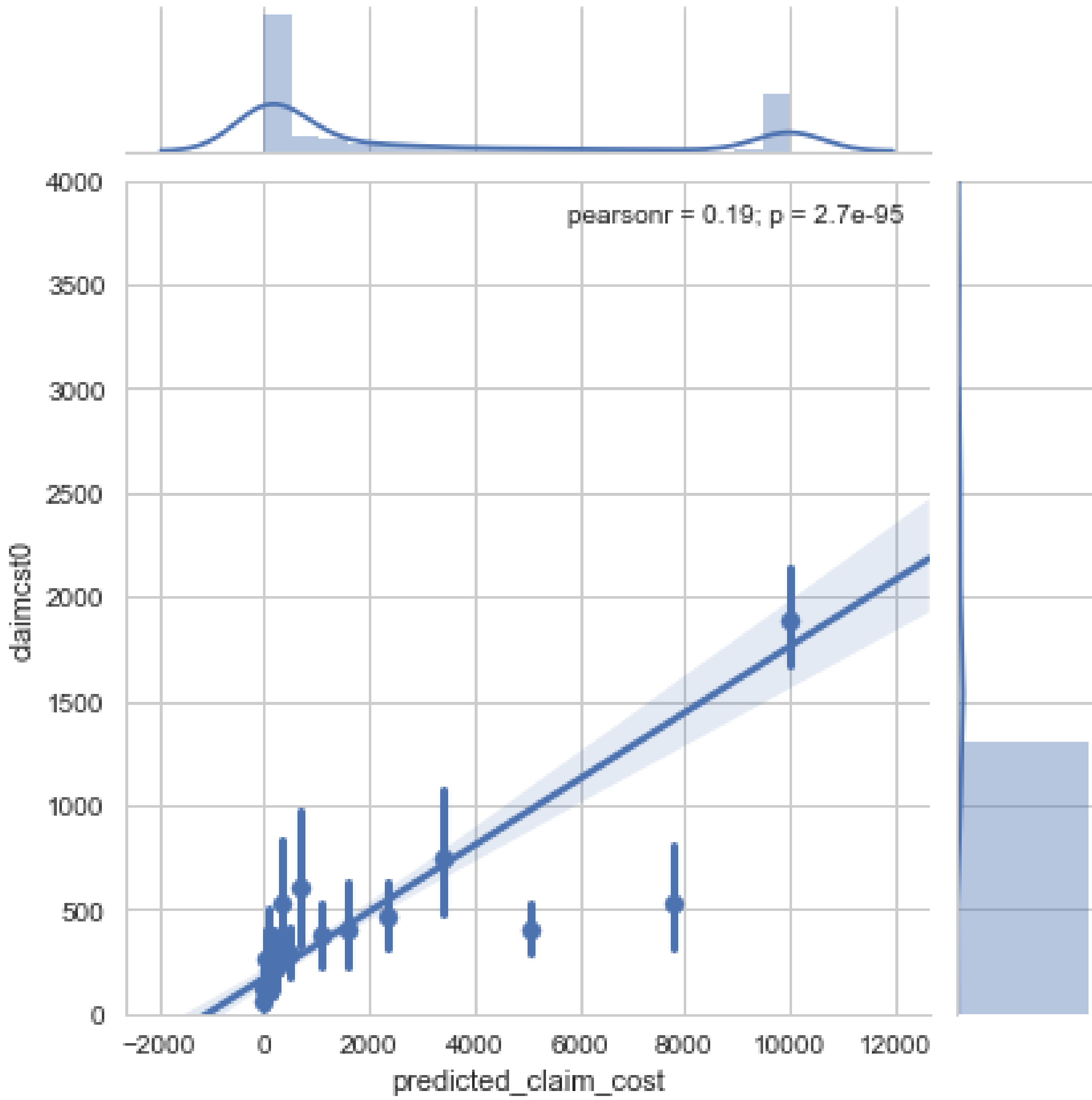
```
        #Train only on the subset of positive claim counts
        X_train_regress = X_train[X_train['claim_count']>0].copy()
        y_train_regress = np.log(X_train[X_train['claim_count']>0]['claimcst0'])

        fam = sm.families.Poisson()
        glm = sm.GLM(y_train_regress,X_train_regress.drop(['pol_number','claim_count','cla
        res = glm.fit()
        predict = res.predict(X_test2.drop(['pol_number','claim_count','claimcst0','prob0
        X_test2['predicted_claim_cost']=[np.exp(p) for p in predict]
        X_test2['predicted_claim_cost']=X_test2['predicted_claim_cost'].multiply(X_test2[
        return res, X_test2.copy()
```

In [41]:
```
glm, X_test2 = glmfunc(X_train,X_test2)
```

```
/Applications/anaconda/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarn
  from pandas.core import datetools
```

In [42]:
```
sns.jointplot(X_test2.predicted_claim_cost.clip(upper=10000), X_test2.claimcst0, kind=
```

Out[42]:
```
<seaborn.axisgrid.JointGrid at 0x112677128>
```




```
In [43]: from sklearn.metrics import mean_squared_error
         mean_squared_error(X_test2.claimcst0, X_test2.predicted_claim_cost.clip(upper=10000))
```

```
Out[43]: 5343.7993274896289
```

Decision Trees Regressor Function

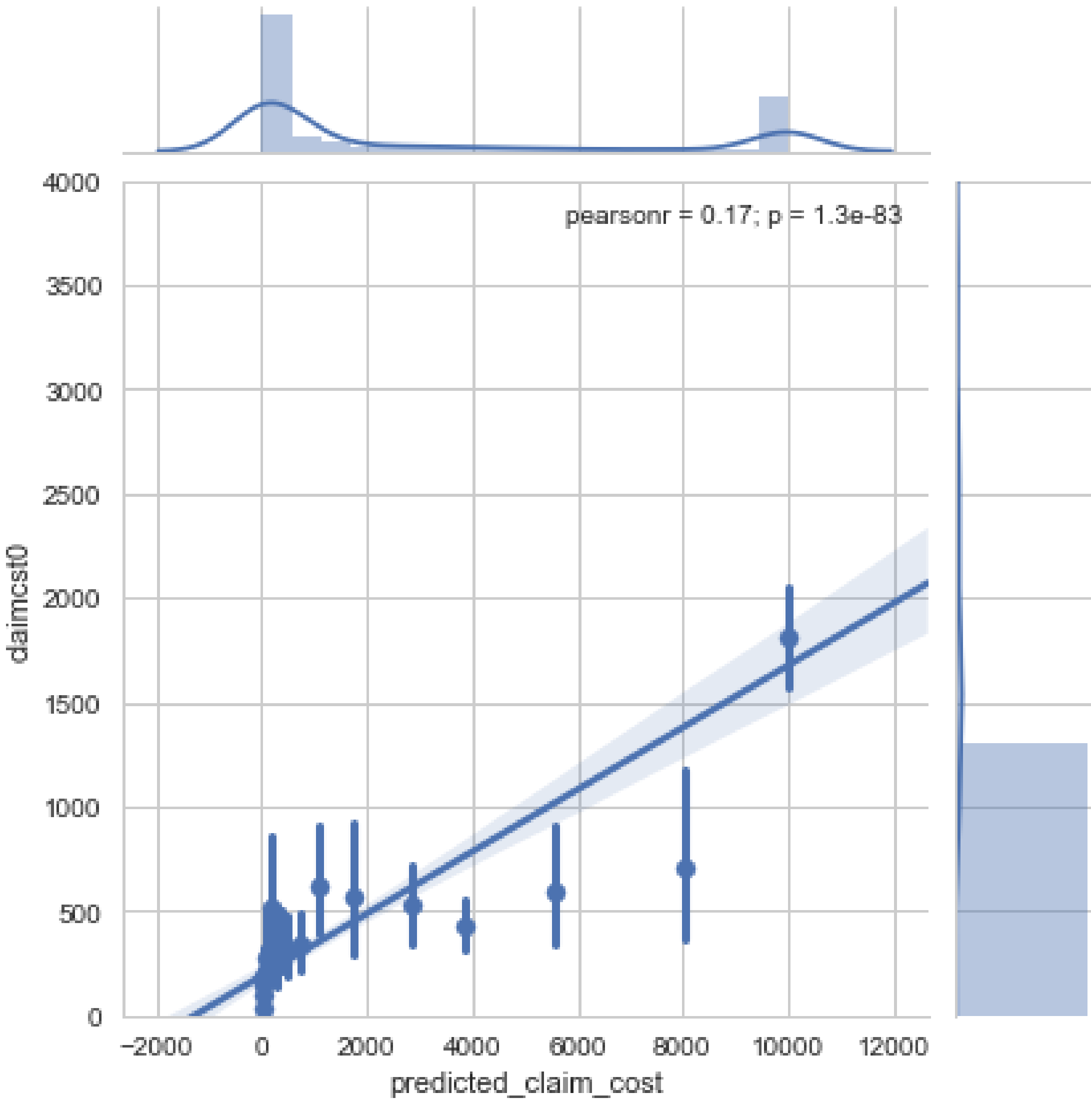
```
In [44]: def DTfunc(X_train, X_test3):
             #Part 2: Use Regression for predicting Claim Severity

             from sklearn.tree import DecisionTreeRegressor

             X_test3.drop(['predicted_claim_count','predicted_claim_cost'],axis=1,inplace=True
```

```
    #Train only on the subset of positive claim counts
    X_train_regress = X_train[X_train['claim_count']>0].copy()
    y_train_regress = np.log(X_train[X_train['claim_count']>0]['claimcst0'])

    #Implement XGBoost for regression
    from sklearn.tree import DecisionTreeRegressor
    num_round = 1
    clf_entropy = DecisionTreeRegressor(criterion = "mse", random_state = 12345, max_d

    # Performing training
    dT = clf_entropy.fit(X_train_regress.drop(['pol_number','claim_count','claimcst0'

    predictions = dT.predict(X_test3.drop(['pol_number','claim_count','claimcst0','pre
    X_test3['predicted_claim_cost']=[np.exp(p) for p in predictions]
    X_test3['predicted_claim_cost']=X_test3['predicted_claim_cost'].multiply(X_test3[
    return dT, X_test3.copy()
```

In [45]:
```
dT, X_test3 = DTfunc(X_train,X_test3)
```

In [46]:
```
sns.jointplot(X_test3.predicted_claim_cost.clip(upper=10000), X_test3.claimcst0, kind=
```

Out[46]:
```
<seaborn.axisgrid.JointGrid at 0x1129ce780>
```




```
In [47]: from sklearn.metrics import mean_squared_error
         mean_squared_error(X_test3.claimcst0, X_test3.predicted_claim_cost.clip(upper=10000))
```

```
Out[47]: 5419.0163294149907
```

Extra Trees Regressor Function

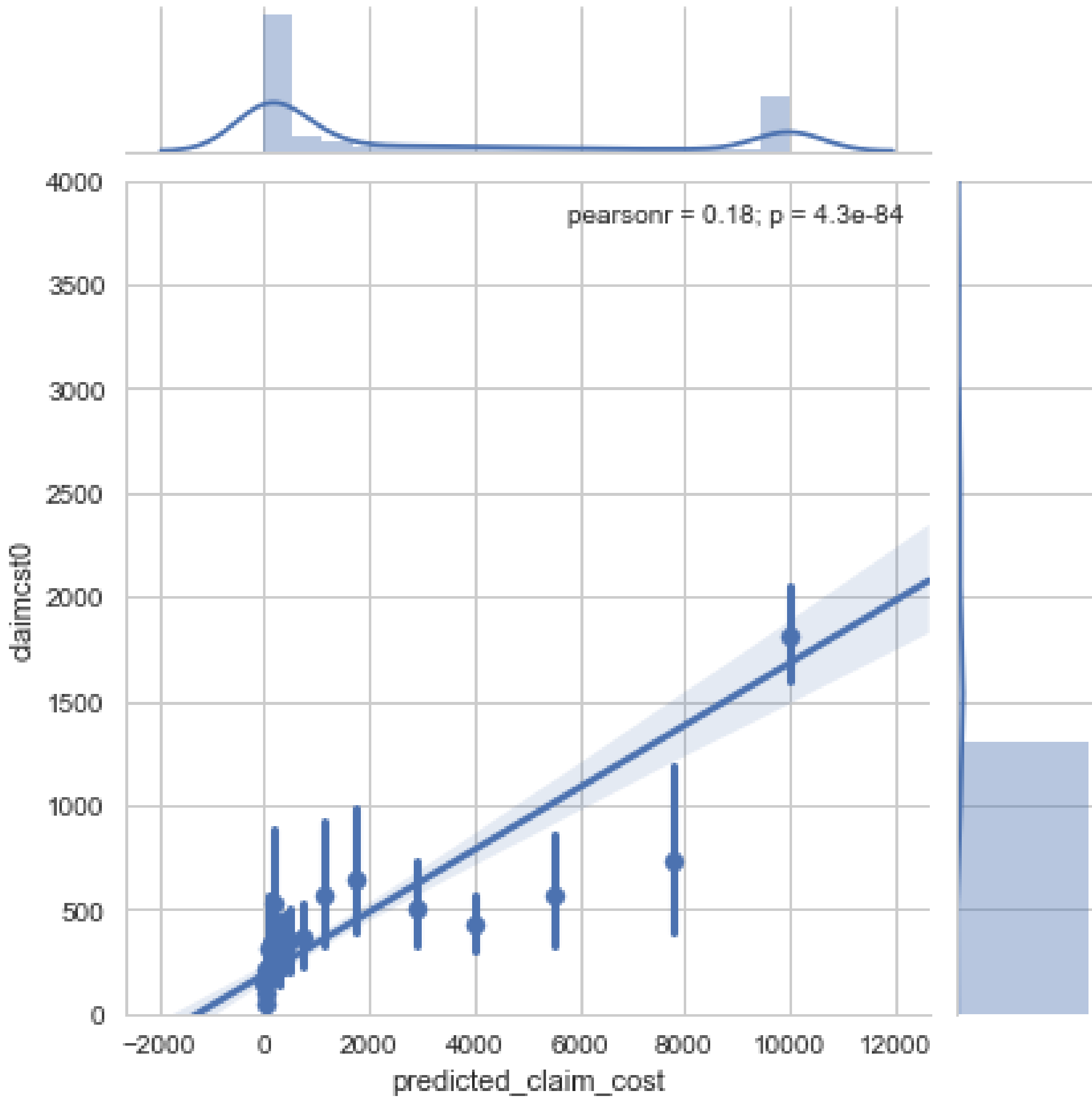
```
In [48]: def ETfunc(X_train, X_test4):
             #Part 2: Use Regression for predicting Claim Severity

             from sklearn.ensemble import ExtraTreesRegressor

             X_test4.drop(['predicted_claim_count','predicted_claim_cost'],axis=1,inplace=True
```

```
        #Train only on the subset of positive claim counts
        X_train_regress = X_train[X_train['claim_count']>0].copy()
        y_train_regress = np.log(X_train[X_train['claim_count']>0]['claimcst0'])

        #Implement XGBoost for regression
        from sklearn.ensemble import ExtraTreesRegressor
        num_round = 1

        clf = ExtraTreesRegressor(n_estimators=10, criterion = "mse", random_state = 1234

        clf2= clf.fit(X_train_regress.drop(['pol_number','claim_count','claimcst0','prob0
        predictions = clf2.predict(X_test4.drop(['pol_number','claim_count','claimcst0','

        X_test4['predicted_claim_cost']=[np.exp(p) for p in predictions]
        X_test4['predicted_claim_cost']=X_test4['predicted_claim_cost'].multiply(X_test4[
        return clf2, X_test4.copy()
```

```
In [49]: et, X_test4 = ETfunc(X_train,X_test4)
```

```
In [50]: sns.jointplot(X_test4.predicted_claim_cost.clip(upper=10000), X_test4.claimcst0, kind
```

```
Out[50]: <seaborn.axisgrid.JointGrid at 0x11689db00>
```




```
In [51]: from sklearn.metrics import mean_squared_error
         mean_squared_error(X_test4.claimcst0, X_test4.predicted_claim_cost.clip(upper=10000))
```

```
Out[51]: 5410.2703286864962
```

Random Forest Regressor Function

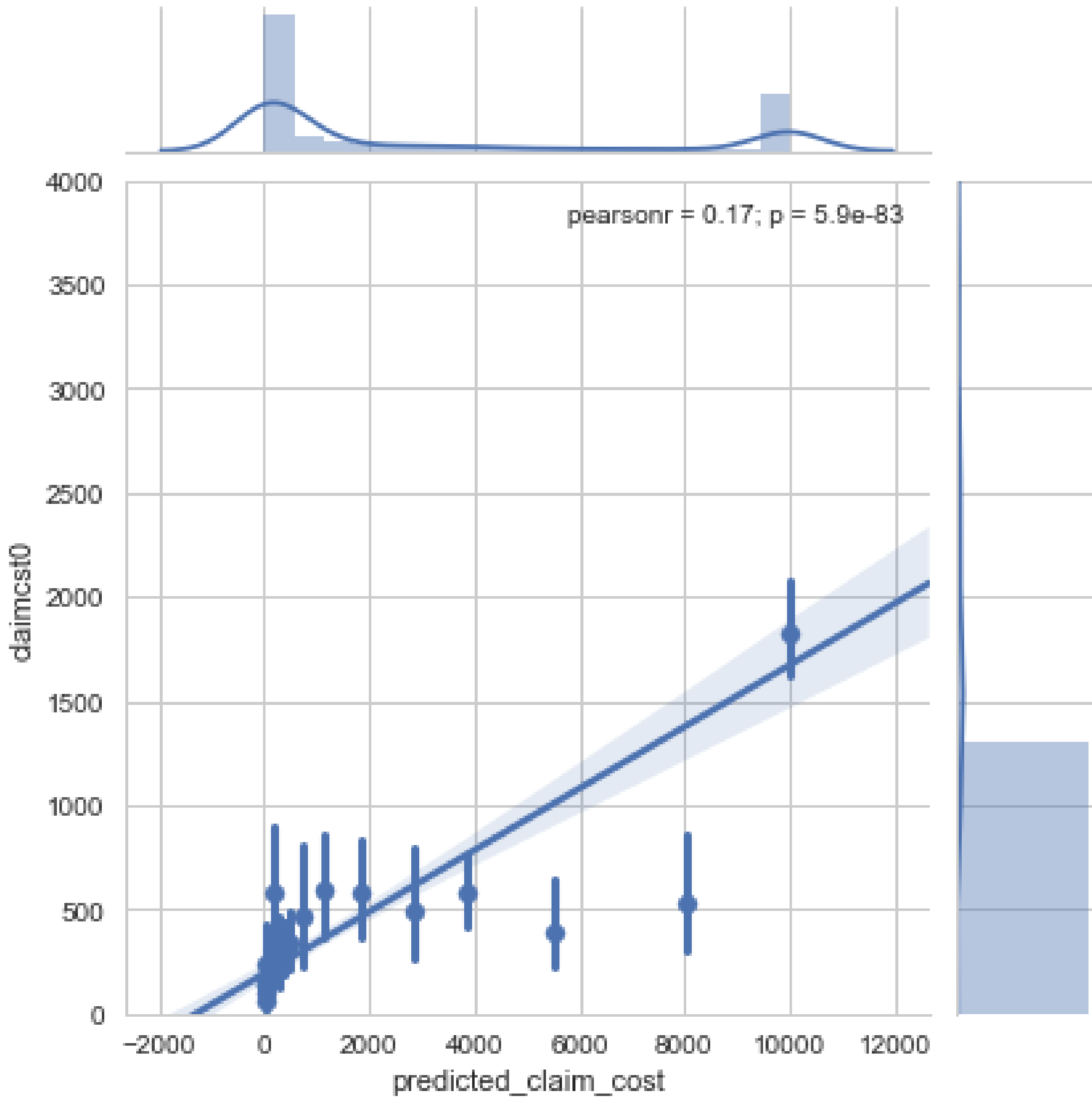
```
In [52]: def RFfunc(X_train, X_test5):
             #Part 2: Use Regression for predicting Claim Severity

             from sklearn.ensemble import RandomForestRegressor

             X_test5.drop(['predicted_claim_count','predicted_claim_cost'],axis=1,inplace=True
```

```
    #Train only on the subset of positive claim counts
    X_train_regress = X_train[X_train['claim_count']>0].copy()
    y_train_regress = np.log(X_train[X_train['claim_count']>0]['claimcst0'])

    num_round = 1

    rf = RandomForestRegressor(criterion = "mse", random_state = 12345, max_depth = 3
    rf2= rf.fit(X_train_regress.drop(['pol_number','claim_count','claimcst0','prob0',
    predictions = rf2.predict(X_test5.drop(['pol_number','claim_count','claimcst0','p

    X_test5['predicted_claim_cost']=[np.exp(p) for p in predictions]
    X_test5['predicted_claim_cost']=X_test5['predicted_claim_cost'].multiply(X_test5[
    return rf2, X_test5.copy()
```

```
In [53]: rf, X_test5 = RFfunc(X_train,X_test5)
```

```
In [54]: sns.jointplot(X_test5.predicted_claim_cost.clip(upper=10000), X_test5.claimcst0, kind=
```

```
Out[54]: <seaborn.axisgrid.JointGrid at 0x113e0cb70>
```




```
In [55]: from sklearn.metrics import mean_squared_error
         mean_squared_error(X_test5.claimcst0, X_test5.predicted_claim_cost.clip(upper=10000))
```

```
Out[55]: 5420.5194603386844
```

Function to calculate normalized gini coeeficients to evaluate model performance

```
In [56]: def gini(actual, pred):
             assert (len(actual) == len(pred))
             all = np.asarray(np.c_[actual, pred, np.arange(len(actual))], dtype=np.float)
             all = all[np.lexsort((all[:, 2], -1 * all[:, 1]))]
             totalLosses = all[:, 0].sum()
             giniSum = all[:, 0].cumsum().sum() / totalLosses
```

```
        giniSum -= (len(actual) + 1) / 2.
        return giniSum / len(actual)


    def gini_normalized(actual, pred):
        return gini(actual, pred) / gini(actual, actual)
```

In [57]:
```
from sklearn.model_selection import KFold
```

Model Selection by Running KFold cross validation

In [58]:
```
ginilist_xgb = []
ginilist_glm = []
ginilist_dt = []
ginilist_et = []
ginilist_rf = []
ginilist_avg = []

df_log= pd.DataFrame(data=[])
df_xgb= pd.DataFrame(data=[])
df_glm= pd.DataFrame(data=[])
df_dT= pd.DataFrame(data=[])
df_eT= pd.DataFrame(data=[])
df_RF= pd.DataFrame(data=[])

fold=1
kf = KFold(n_splits=5, random_state=12345)
for train_index, test_index in kf.split(df_train):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = df_train.iloc[train_index].copy(), df_train.iloc[test_index].co

    #We create a separate X_train and X_test dataset because we would require the cat
    #X_train and X_test dataframes have only dummy variables
    X_train = prep_data(X_train)
    X_test = prep_data(X_test)

    #Create two additional columns for the individual probabilities of claim (prob1)
    X_train['prob0'] = np.zeros(len(X_train))
    X_train['prob1'] = np.zeros(len(X_train))

    X_test['prob0'] = np.zeros(len(X_test))
    X_test['prob1'] = np.zeros(len(X_test))

    logmodel, X_testlog , X_train = logfunc(X_train,X_test)

    trainDF_log = X_train.copy()
    testDF_log = X_testlog.copy()
```

```
gbm, X_test1 = xgboostfunc(X_train,X_testlog)
testDF_xgb = X_test1.copy()

glm, X_test2 = glmfunc(X_train,X_testlog)
testDF_glm = X_test2.copy()
testDF_glm_factors = X_test2.copy()

dT, X_test3 = DTfunc(X_train,X_testlog)
testDF_dt = X_test3.copy()

eT, X_test4 = ETfunc(X_train,X_testlog)
testDF_et = X_test4.copy()

rF, X_test5 = RFfunc(X_train,X_testlog)
testDF_rf = X_test5.copy()


testDF_log['iteration'] = '%s%d' % ('V', fold)
df_log = pd.concat([df_log, testDF_log])
testDF_xgb['iteration'] = '%s%d' % ('V', fold)
df_xgb = pd.concat([df_xgb, testDF_xgb])
testDF_glm['iteration'] = '%s%d' % ('V', fold)
df_glm = pd.concat([df_glm, testDF_glm])
testDF_dt['iteration'] = '%s%d' % ('V', fold)
df_dT = pd.concat([df_dT, testDF_dt])
testDF_et['iteration'] = '%s%d' % ('V', fold)
df_eT = pd.concat([df_eT, testDF_et])
testDF_rf['iteration'] = '%s%d' % ('V', fold)
df_RF = pd.concat([df_RF, testDF_rf])


prediction_xgb = X_test1['predicted_claim_cost']
gininorm_xgb = gini_normalized(X_test1.claimcst0.values,X_test1.predicted_claim_co
prediction_glm = X_test2['predicted_claim_cost']
gininorm_glm = gini_normalized(X_test2.claimcst0.values,X_test2.predicted_claim_co
prediction_dt = X_test3['predicted_claim_cost']
gininorm_dt = gini_normalized(X_test3.claimcst0.values,X_test3.predicted_claim_cos
prediction_et = X_test4['predicted_claim_cost']
gininorm_et = gini_normalized(X_test4.claimcst0.values,X_test4.predicted_claim_cos
prediction_rf = X_test5['predicted_claim_cost']
gininorm_rf = gini_normalized(X_test5.claimcst0.values,X_test5.predicted_claim_cos


prediction_avg = (prediction_xgb+prediction_glm+prediction_dt+prediction_et+predic
gininorm_avg = gini_normalized(X_test1.claimcst0.values,prediction_avg.values)

ginilist_xgb.append(gininorm_xgb)
```

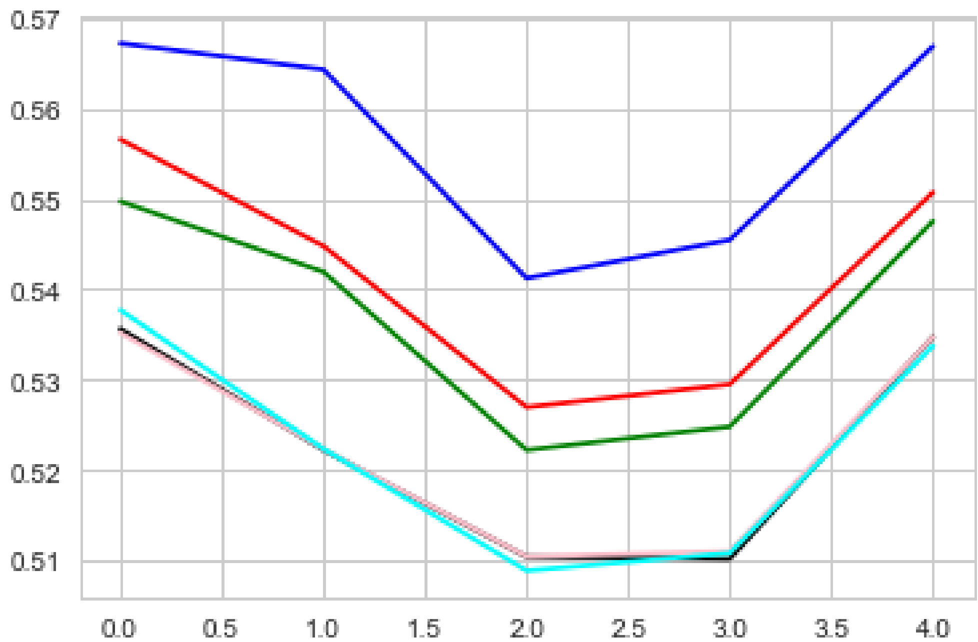
```
        ginilist_glm.append(gininorm_glm)
        ginilist_dt.append(gininorm_dt)
        ginilist_et.append(gininorm_et)
        ginilist_rf.append(gininorm_rf)
        ginilist_avg.append(gininorm_avg)
```

```
TRAIN: [12079 12080 12081 ..., 60389 60390 60391] TEST: [    0     1     2 ..., 12076 12077 120
             precision    recall  f1-score   support

          0       0.91      0.67      0.77     10043
          1       0.29      0.67      0.40      2036

avg / total       0.80      0.67      0.71     12079

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [12079 12080 12081 ..., 24155 24156 241
             precision    recall  f1-score   support

          0       0.91      0.67      0.77     10065
          1       0.29      0.69      0.41      2014

avg / total       0.81      0.67      0.71     12079

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [24158 24159 24160 ..., 36233 36234 362
             precision    recall  f1-score   support

          0       0.91      0.68      0.78     10092
          1       0.29      0.66      0.40      1986

avg / total       0.81      0.68      0.72     12078

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [36236 36237 36238 ..., 48311 48312 483
             precision    recall  f1-score   support

          0       0.91      0.68      0.78     10102
          1       0.29      0.67      0.40      1976

avg / total       0.81      0.67      0.72     12078

TRAIN: [    0     1     2 ..., 48311 48312 48313] TEST: [48314 48315 48316 ..., 60389 60390 603
             precision    recall  f1-score   support

          0       0.92      0.67      0.78     10060
          1       0.30      0.70      0.42      2018

avg / total       0.82      0.68      0.72     12078
```

0.1.4 GLM gave highest gini

In [59]:
```
plt.plot(ginilist_xgb, color='red')
plt.plot(ginilist_glm, color='blue')
plt.plot(ginilist_dt, color='black')
plt.plot(ginilist_et, color='pink')
plt.plot(ginilist_rf, color='cyan')
plt.plot(ginilist_avg, color='green')
```

Out[59]:
```
[<matplotlib.lines.Line2D at 0x119c47cc0>]
```



0.1.5 Feature Importance for GLM

In [60]:
```
print(glm.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              claimcst0   No. Observations:                 8012
Model:                            GLM   Df Residuals:                     7981
Model Family:                 Poisson   Df Model:                           30
Link Function:                    log   Scale:                             1.0
Method:                          IRLS   Log-Likelihood:                -15952.
Date:                Tue, 08 May 2018   Deviance:                       1271.1
Time:                        05:48:58   Pearson chi2:                 1.26e+03
No. Iterations:                     4
==========================================================================================
```

```
                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
credit_score_imp        0.0224      0.000     64.509      0.000       0.022       0.023
traffic_index_imputed  5.951e-05    0.000      0.288      0.773      -0.000       0.000
veh_age_1               0.1506      0.010     15.814      0.000       0.132       0.169
veh_age_2               0.0212      0.008      2.590      0.010       0.005       0.037
veh_age_3              -0.0099      0.007     -1.376      0.169      -0.024       0.004
veh_age_4              -0.0741      0.011     -6.989      0.000      -0.095      -0.053
agecat_imputed_1        0.2940      0.009     32.400      0.000       0.276       0.312
agecat_imputed_2        0.1879      0.009     21.515      0.000       0.171       0.205
agecat_imputed_3       -0.0374      0.009     -4.046      0.000      -0.056      -0.019
agecat_imputed_4       -0.1032      0.010    -10.844      0.000      -0.122      -0.085
agecat_imputed_5       -0.2503      0.012    -21.672      0.000      -0.273      -0.228
agecat_imputed_6       -0.0032      0.011     -0.293      0.769      -0.025       0.018
veh_body_BUS            0.0666      0.110      0.605      0.545      -0.149       0.282
veh_body_CONVT          0.1234      0.117      1.056      0.291      -0.106       0.353
veh_body_COUPE          0.0368      0.042      0.885      0.376      -0.045       0.118
veh_body_HBACK         -0.1079      0.027     -4.002      0.000      -0.161      -0.055
veh_body_HDTOP          0.0596      0.033      1.787      0.074      -0.006       0.125
veh_body_MCARA          0.1186      0.077      1.542      0.123      -0.032       0.269
veh_body_MIBUS          0.0700      0.049      1.415      0.157      -0.027       0.167
veh_body_PANVN         -0.0133      0.039     -0.342      0.732      -0.090       0.063
veh_body_RDSTR          0.0139      0.224      0.062      0.950      -0.425       0.453
veh_body_SEDAN         -0.0770      0.026     -2.959      0.003      -0.128      -0.026
veh_body_STNWG         -0.1157      0.026     -4.535      0.000      -0.166      -0.066
veh_body_TRUCK          0.0012      0.031      0.038      0.969      -0.059       0.061
veh_body_UTE           -0.0884      0.027     -3.255      0.001      -0.142      -0.035
gender_F                0.0572      0.005     12.036      0.000       0.048       0.067
gender_M                0.0306      0.005      6.356      0.000       0.021       0.040
area_A                  0.0078      0.010      0.747      0.455      -0.013       0.028
area_B                  0.0104      0.008      1.249      0.211      -0.006       0.027
area_C                 -0.0434      0.008     -5.438      0.000      -0.059      -0.028
area_D                  0.0387      0.009      4.163      0.000       0.021       0.057
area_E                  0.1241      0.007     16.835      0.000       0.110       0.139
area_F                 -0.0499      0.014     -3.630      0.000      -0.077      -0.023
cat_veh_value_1        -0.0435      0.015     -2.919      0.004      -0.073      -0.014
cat_veh_value_2        -0.0086      0.008     -1.037      0.300      -0.025       0.008
cat_veh_value_3         0.0172      0.009      1.978      0.048       0.000       0.034
cat_veh_value_4         0.0486      0.013      3.807      0.000       0.024       0.074
cat_veh_value_5         0.0741      0.015      5.033      0.000       0.045       0.103
=======================================================================================
```

```
In [61]: colsV1 = ['pol_number','credit_score_imp','veh_age_1','veh_age_2','veh_age_4','agecat_
```

Running Kfold and evaluating model performance after inputing important features

```
In [62]: ginilist_xgb = []
         ginilist_glm = []
```

```
ginilist_dt = []
ginilist_et = []
ginilist_rf = []
ginilist_avg = []

df_log= pd.DataFrame(data=[])
df_xgb= pd.DataFrame(data=[])
df_glm= pd.DataFrame(data=[])
df_dT= pd.DataFrame(data=[])
df_eT= pd.DataFrame(data=[])
df_RF= pd.DataFrame(data=[])

fold=1
kf = KFold(n_splits=5, random_state=12345)
for train_index, test_index in kf.split(df_train):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = df_train.iloc[train_index].copy(), df_train.iloc[test_index].cop

    #We create a separate X_train and X_test dataset because we would require the cat
    #X_train and X_test dataframes have only dummy variables
    X_train = prep_data(X_train)
    X_test = prep_data(X_test)

    X_train = X_train[colsV1]
    X_test = X_test[colsV1]

    #Create two additional columns for the individual probabilities of claim (prob1) 
    X_train['prob0'] = np.zeros(len(X_train))
    X_train['prob1'] = np.zeros(len(X_train))

    X_test['prob0'] = np.zeros(len(X_test))
    X_test['prob1'] = np.zeros(len(X_test))

    logmodel, X_testlog , X_train = logfunc(X_train,X_test)

    trainDF_log = X_train.copy()
    testDF_log = X_testlog.copy()

    gbm, X_test1 = xgboostfunc(X_train,X_testlog)
    testDF_xgb = X_test1.copy()

    glm, X_test2 = glmfunc(X_train,X_testlog)
    testDF_glm = X_test2.copy()
    testDF_glm_factors = X_test2.copy()

    dT, X_test3 = DTfunc(X_train,X_testlog)
    testDF_dt = X_test3.copy()
```

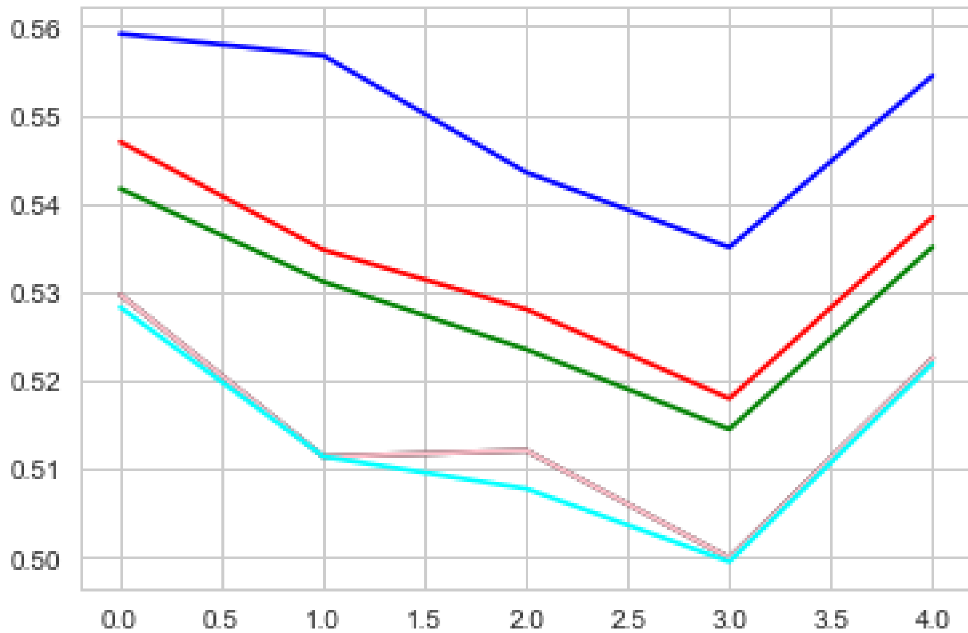
```
eT, X_test4 = ETfunc(X_train,X_testlog)
testDF_et = X_test4.copy()

rF, X_test5 = RFfunc(X_train,X_testlog)
testDF_rf = X_test5.copy()


testDF_log['iteration'] = '%s%d' % ('V', fold)
df_log = pd.concat([df_log, testDF_log])
testDF_xgb['iteration'] = '%s%d' % ('V', fold)
df_xgb = pd.concat([df_xgb, testDF_xgb])
testDF_glm['iteration'] = '%s%d' % ('V', fold)
df_glm = pd.concat([df_glm, testDF_glm])
testDF_dt['iteration'] = '%s%d' % ('V', fold)
df_dT = pd.concat([df_dT, testDF_dt])
testDF_et['iteration'] = '%s%d' % ('V', fold)
df_eT = pd.concat([df_eT, testDF_et])
testDF_rf['iteration'] = '%s%d' % ('V', fold)
df_RF = pd.concat([df_RF, testDF_rf])


prediction_xgb = X_test1['predicted_claim_cost']
gininorm_xgb = gini_normalized(X_test1.claimcst0.values,X_test1.predicted_claim_c
prediction_glm = X_test2['predicted_claim_cost']
gininorm_glm = gini_normalized(X_test2.claimcst0.values,X_test2.predicted_claim_c
prediction_dt = X_test3['predicted_claim_cost']
gininorm_dt = gini_normalized(X_test3.claimcst0.values,X_test3.predicted_claim_cos
prediction_et = X_test4['predicted_claim_cost']
gininorm_et = gini_normalized(X_test4.claimcst0.values,X_test4.predicted_claim_cos
prediction_rf = X_test5['predicted_claim_cost']
gininorm_rf = gini_normalized(X_test5.claimcst0.values,X_test5.predicted_claim_cos


prediction_avg = (prediction_xgb+prediction_glm+prediction_dt+prediction_et+predi
gininorm_avg = gini_normalized(X_test1.claimcst0.values,prediction_avg.values)

ginilist_xgb.append(gininorm_xgb)
ginilist_glm.append(gininorm_glm)
ginilist_dt.append(gininorm_dt)
ginilist_et.append(gininorm_et)
ginilist_rf.append(gininorm_rf)
ginilist_avg.append(gininorm_avg)
```

```
TRAIN: [12079 12080 12081 ..., 60389 60390 60391] TEST: [    0     1     2 ..., 12076 12077 120
             precision    recall  f1-score   support

          0       0.91      0.64      0.75     10043
```

```
           1       0.28      0.69      0.40      2036

avg / total       0.81      0.65      0.70     12079

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [12079 12080 12081 ..., 24155 24156 24
             precision    recall  f1-score   support

          0       0.92      0.64      0.75     10065
          1       0.28      0.71      0.40      2014

avg / total       0.81      0.65      0.69     12079

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [24158 24159 24160 ..., 36233 36234 36
             precision    recall  f1-score   support

          0       0.91      0.65      0.76     10092
          1       0.28      0.68      0.39      1986

avg / total       0.81      0.66      0.70     12078

TRAIN: [    0     1     2 ..., 60389 60390 60391] TEST: [36236 36237 36238 ..., 48311 48312 48
             precision    recall  f1-score   support

          0       0.91      0.64      0.75     10102
          1       0.27      0.69      0.39      1976

avg / total       0.81      0.65      0.69     12078

TRAIN: [    0     1     2 ..., 48311 48312 48313] TEST: [48314 48315 48316 ..., 60389 60390 60
             precision    recall  f1-score   support

          0       0.92      0.66      0.76     10060
          1       0.29      0.70      0.41      2018

avg / total       0.81      0.66      0.71     12078
```

In [63]:
```
plt.plot(ginilist_xgb, color='red')
plt.plot(ginilist_glm, color='blue')
plt.plot(ginilist_dt, color='black')
plt.plot(ginilist_et, color='pink')
plt.plot(ginilist_rf, color='cyan')
plt.plot(ginilist_avg, color='green')
```

Out[63]: [<matplotlib.lines.Line2D at 0x11b154da0>]




```
In [64]: print(glm.summary())

                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              claimcst0   No. Observations:                 8012
Model:                            GLM   Df Residuals:                     7990
Model Family:                 Poisson   Df Model:                           21
Link Function:                    log   Scale:                             1.0
Method:                          IRLS   Log-Likelihood:                -15956.
Date:                Tue, 08 May 2018   Deviance:                       1278.8
Time:                        05:49:46   Pearson chi2:                 1.27e+03
No. Iterations:                     4
==================================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
credit_score_imp    0.0226      0.000     97.447      0.000       0.022       0.023
veh_age_1           0.1570      0.014     11.329      0.000       0.130       0.184
veh_age_2           0.0286      0.013      2.287      0.022       0.004       0.053
veh_age_4          -0.0604      0.013     -4.569      0.000      -0.086      -0.035
agecat_imputed_1    0.2984      0.014     21.914      0.000       0.272       0.325
agecat_imputed_2    0.1937      0.014     14.314      0.000       0.167       0.220
agecat_imputed_3   -0.0332      0.015     -2.178      0.029      -0.063      -0.003
agecat_imputed_4   -0.0998      0.016     -6.342      0.000      -0.131      -0.069
agecat_imputed_5   -0.2482      0.019    -13.134      0.000      -0.285      -0.211
veh_body_HBACK     -0.1323      0.017     -7.582      0.000      -0.166      -0.098
veh_body_SEDAN     -0.1026      0.016     -6.459      0.000      -0.134      -0.071
```

```
veh_body_UTE        -0.1148      0.018     -6.349      0.000      -0.150      -0.079
veh_body_STNWG      -0.1436      0.015     -9.451      0.000      -0.173      -0.114
gender_F             0.0590      0.005     11.961      0.000       0.049       0.069
gender_M             0.0305      0.005      6.029      0.000       0.021       0.040
area_C              -0.0512      0.010     -5.080      0.000      -0.071      -0.031
area_D               0.0286      0.012      2.428      0.015       0.006       0.052
area_E               0.1082      0.019      5.584      0.000       0.070       0.146
area_F              -0.0592      0.018     -3.233      0.001      -0.095      -0.023
cat_veh_value_1     -0.0398      0.014     -2.788      0.005      -0.068      -0.012
cat_veh_value_3      0.0276      0.012      2.255      0.024       0.004       0.052
cat_veh_value_4      0.0602      0.018      3.379      0.001       0.025       0.095
cat_veh_value_5      0.0876      0.020      4.366      0.000       0.048       0.127
=====================================================================================
```

0.2 GLM Model Uncertainity Removal

```
In [66]: # from sklearn.metrics import mutual_info_score
         # score = mutual_info_score(X_testlog.claim_count,X_testlog.predicted_claim_count)
         # score
```

```
In [ ]: # p_true = pre_holdout_data['Churning'].mean()
        # p_false = 1-p_true
        # from scipy.stats import entropy
        # ent = entropy([p_true,p_false])
```

```
In [67]: # uncertainity_explained = score/ent
         # uncertainity_explained
```

Running for Test Data

```
In [68]: df_test_valid = pd.read_csv('auto_potential_customers_2018.csv')
```

Data pre processing on Test Data

```
In [69]: df_test_valid['date'] = pd.to_datetime(df_test_valid['date_of_birth'])
```

```
In [70]: df_test_valid['year'] = df_test_valid['date'].dt.year
```

```
In [71]: df1_test_valid = df_test_valid.groupby('agecat')['year'].min()
         df2_test_valid= df_test_valid.groupby('agecat')['year'].max()
```

```
In [72]: df1_test_valid
```

```
Out[72]: agecat
         1.0    1990
         2.0    1980
         3.0    1970
         4.0    1960
         5.0    1950
         6.0    1924
         Name: year, dtype: int64
```

In [73]: df2_test_valid

Out[73]:
```
agecat
1.0    1999
2.0    1989
3.0    1979
4.0    1969
5.0    1959
6.0    1949
Name: year, dtype: int64
```

In [74]:
```
conditions = [
    (df_test_valid['year'] >= 1990) & (df_test_valid['year'] <= 1999),
    (df_test_valid['year'] >= 1980) & (df_test_valid['year'] <= 1989),
    (df_test_valid['year'] >= 1970) & (df_test_valid['year'] <= 1979),
    (df_test_valid['year'] >= 1960) & (df_test_valid['year'] <= 1969),
    (df_test_valid['year'] >= 1950) & (df_test_valid['year'] <= 1959),]
choices = [1,2,3,4,5]
df_test_valid['agecat_imputed'] = np.select(conditions, choices,default=6)
```

In [75]:
```
df3_test_valid = df_test_valid.groupby('agecat_imputed')['credit_score'].mean()
```

In [76]: df3_test_valid

Out[76]:
```
agecat_imputed
1    582.104396
2    599.992200
3    655.919780
4    670.562428
5    735.518641
6    696.704641
Name: credit_score, dtype: float64
```

In [77]:
```
df4_test_valid = df_test_valid.groupby('area')['traffic_index'].mean()
```

In [78]: df4_test_valid

Out[78]:
```
area
A     80.464980
B    120.267909
C    129.115227
D    100.589112
E     44.573322
F    114.634286
Name: traffic_index, dtype: float64
```

In [79]:
```
conditions1 = [
    (df_test_valid['agecat_imputed'] ==1),
    (df_test_valid['agecat_imputed'] ==2),
```

```
         (df_test_valid['agecat_imputed'] ==3),
         (df_test_valid['agecat_imputed'] ==4),
         (df_test_valid['agecat_imputed'] ==5)]
choices1 = [582.104396,599.992200,655.919780,670.562428,735.518641]
df_test_valid['credit_score_imputed'] = np.select(conditions1, choices1,default=696.7
```

In [80]:
```
df_test_valid['credit_score_imp']=(df_test_valid.credit_score_imputed/df_test_valid.c
```

In [81]:
```
conditions2 = [
         (df_test_valid['area'] =='A'),
         (df_test_valid['area'] =='B'),
         (df_test_valid['area'] =='C'),
         (df_test_valid['area'] =='D'),
         (df_test_valid['area'] =='E')]
choices2 = [80.464980,120.267909,129.115227,100.589112,44.573322]
df_test_valid['traffic_index_imputed'] = np.select(conditions2, choices2,default=114.
```

In [82]:
```
df_test_valid.rename(columns={'quote_number': 'pol_number'}, inplace=True)
```

In [83]:
```
df_test_valid_V1 = df_test_valid.copy()
```

One Hot Encoding on Test Data

In [84]:
```
#Function to Prepare the data

def prep_data_test(df):
    #Prepare the data
    #Make Vehicle Value categorical
    df['cat_veh_value'] = df['veh_value'].apply(lambda x:1 if x<=1 else 2 if x<=2 else
    df_dummy = df[['pol_number','credit_score_imp','traffic_index_imputed']].copy()
    #Make dummy variables using Pandas
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['veh_age'],prefix="veh_age")],ax
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['agecat_imputed'],prefix="agecat_
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['veh_body'],prefix="veh_body")],
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['gender'],prefix="gender")],axis
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['area'],prefix="area")],axis=1)
    df_dummy = pd.concat([df_dummy,pd.get_dummies(df['cat_veh_value'],prefix="cat_veh_

    return(df_dummy)
```

In [85]:
```
X_test_final = prep_data_test(df_test_valid)
```

In [86]:
```
X_test_final_V1 = X_test_final.copy()
```

In [87]:
```
X_test_final.columns
```

Out[87]:
```
Index(['pol_number', 'credit_score_imp', 'traffic_index_imputed', 'veh_age_1',
       'veh_age_2', 'veh_age_3', 'veh_age_4', 'agecat_imputed_1',
       'agecat_imputed_2', 'agecat_imputed_3', 'agecat_imputed_4',
```

```
'agecat_imputed_5', 'agecat_imputed_6', 'veh_body_BUS',
 'veh_body_CONVT', 'veh_body_COUPE', 'veh_body_HBACK', 'veh_body_HDTOP',
 'veh_body_MCARA', 'veh_body_MIBUS', 'veh_body_PANVN', 'veh_body_RDSTR',
 'veh_body_SEDAN', 'veh_body_STNWG', 'veh_body_TRUCK', 'veh_body_UTE',
 'gender_F', 'gender_M', 'area_A', 'area_B', 'area_C', 'area_D',
 'area_E', 'area_F', 'cat_veh_value_1', 'cat_veh_value_2',
 'cat_veh_value_3', 'cat_veh_value_4', 'cat_veh_value_5'],
dtype='object')
```

Inputting only improtant features before prediction

In [88]:
```
colsV2 = ['pol_number','credit_score_imp','veh_age_1','veh_age_2','veh_age_4','agecat_
X_test_final = X_test_final[colsV2]
```

0.3 Customers with a low probability of having any claims in 2018

0.3.1 Predicted_claim_count is 0

In [89]:
```
#Create two additional columns for the individual probabilities of claim (prob1) and 
X_test_final['prob0'] = np.zeros(len(X_test_final))
X_test_final['prob1'] = np.zeros(len(X_test_final))

n=10

for i in range (0,n):

    X_test_final[['prob0','prob1']] = X_test_final[['prob0','prob1']] + logmodel.predi


#Divide the log_proba and log_probb values by 10 to get the average log probabilities
X_test_final['prob0']=X_test_final['prob0']/n
X_test_final['prob1']=X_test_final['prob1']/n


#Check the metrics on training and test data
logProb = 1
temp = logProb > (X_test_final['prob0']/X_test_final['prob1'])
X_test_final['predicted_claim_count']= [1 if (p==True) else 0 for p in temp]


#Calculate Claim Frequency for Test Set
X_test_final['predicted_freq']= X_test_final['prob1'].divide(X_test_final['prob0'])
X_test_final['predicted_freq']= X_test_final['predicted_freq'].apply(lambda x: x**3.65
```

In [90]:
```
X_test_final_V2 = X_test_final.copy()

#Train only on the subset of positive claim counts
X_test_final_regress = X_test_final[X_test_final['predicted_claim_count']>0].copy()
```

0.4 Customers with the lowest cost per claim in 2018, given that a claim occurs

In [91]:
```
predict = glm.predict(X_test_final_regress.drop(['pol_number','predicted_claim_count'
X_test_final_regress['predicted_claim_cost']=[np.exp(p) for p in predict]
#X_test_final_regress['predicted_claim_cost']=X_test_final_regress['predicted_claim_c
```

0.5 Risk Profile Groups

In [92]:
```
X_test_final_regress.describe()
```

Out[92]:

	pol_number	credit_score_imp	veh_age_1	veh_age_2	veh_age_4	\
count	2.947000e+03	2947.000000	2947.000000	2947.000000	2947.000000	
mean	5.496991e+07	86.961674	0.188327	0.220224	0.317611	
std	2.596977e+07	6.401339	0.391040	0.414468	0.465626	
min	1.009346e+07	79.142032	0.000000	0.000000	0.000000	
25%	3.211660e+07	81.574030	0.000000	0.000000	0.000000	
50%	5.535833e+07	89.177859	0.000000	0.000000	0.000000	
75%	7.767625e+07	91.168652	0.000000	0.000000	1.000000	
max	9.997116e+07	100.000000	1.000000	1.000000	1.000000	

	agecat_imputed_1	agecat_imputed_2	agecat_imputed_3	agecat_imputed_4	\
count	2947.000000	2947.000000	2947.000000	2947.000000	
mean	0.213437	0.228029	0.198846	0.178826	
std	0.409803	0.419632	0.399200	0.383272	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

	agecat_imputed_5	...	area_F	cat_veh_value_1	\
count	2947.000000	...	2947.000000	2947.000000	
mean	0.057007	...	0.068205	0.205972	
std	0.231896	...	0.252140	0.404479	
min	0.000000	...	0.000000	0.000000	
25%	0.000000	...	0.000000	0.000000	
50%	0.000000	...	0.000000	0.000000	
75%	0.000000	...	0.000000	0.000000	
max	1.000000	...	1.000000	1.000000	

	cat_veh_value_3	cat_veh_value_4	cat_veh_value_5	prob0	\
count	2947.000000	2947.000000	2947.000000	2947.000000	
mean	0.216491	0.085171	0.093994	0.354981	
std	0.411923	0.279184	0.291870	0.098953	
min	0.000000	0.000000	0.000000	0.066360	
25%	0.000000	0.000000	0.000000	0.294530	
50%	0.000000	0.000000	0.000000	0.374126	
75%	0.000000	0.000000	0.000000	0.432697	
max	1.000000	1.000000	1.000000	0.499845	

```
          prob1  predicted_claim_count  predicted_freq  \
count  2947.000000                 2947.0     2947.000000
mean      0.645019                    1.0      136.645614
std       0.098953                    0.0      768.821967
min       0.500155                    1.0        1.002266
25%       0.567303                    1.0        2.687515
50%       0.625874                    1.0        6.541221
75%       0.705470                    1.0       24.246056
max       0.933640                    1.0    15530.741371

       predicted_claim_cost
count           2947.000000
mean            2536.248075
std             4790.210382
min              234.888395
25%              675.804195
50%             1276.769349
75%             2576.491287
max           142797.715332

[8 rows x 29 columns]
```

In [93]:
```python
df_final_stats = X_test_final_regress.describe()
```

In [94]:
```python
conditions_final = [
    (X_test_final_regress['predicted_claim_cost'] <= 1000),
    (X_test_final_regress['predicted_claim_cost'] > 1000) & (X_test_final_regress['pre
    (X_test_final_regress['predicted_claim_cost'] >= 10000)]
choices_final = [1,2,3]
X_test_final_regress['risk_profile_groups'] = np.select(conditions_final,choices_final
```

In [95]:
```python
X_test_final_regress_riskgrp1 = X_test_final_regress[X_test_final_regress.risk_profile
```

In [96]:
```python
X_test_final_regress_riskgrp2 = X_test_final_regress[X_test_final_regress.risk_profile
```

In [97]:
```python
X_test_final_regress_riskgrp3 = X_test_final_regress[X_test_final_regress.risk_profile
```

In [98]:
```python
X_test_final_regress_riskgrp1.describe()
```

Out[98]:
```
         pol_number  credit_score_imp  veh_age_1    veh_age_2    veh_age_4  \
count  1.173000e+03       1173.000000     1173.0  1173.000000  1173.000000
mean   5.427818e+07         88.316455        0.0     0.119352     0.582268
std    2.572378e+07          5.380759        0.0     0.324341     0.493396
min    1.031372e+07         79.142032        0.0     0.000000     0.000000
25%    3.176463e+07         81.574030        0.0     0.000000     0.000000
50%    5.345764e+07         89.177859        0.0     0.000000     1.000000
75%    7.650403e+07         91.168652        0.0     0.000000     1.000000
max    9.984322e+07        100.000000        0.0     1.000000     1.000000
```

```
       agecat_imputed_1  agecat_imputed_2  agecat_imputed_3  agecat_imputed_4  \
count       1173.000000       1173.000000       1173.000000       1173.000000
mean           0.082694          0.196931          0.329923          0.299233
std            0.275537          0.397849          0.470386          0.458117
min            0.000000          0.000000          0.000000          0.000000
25%            0.000000          0.000000          0.000000          0.000000
50%            0.000000          0.000000          0.000000          0.000000
75%            0.000000          0.000000          1.000000          1.000000
max            1.000000          1.000000          1.000000          1.000000

       agecat_imputed_5         ...            cat_veh_value_1  \
count       1173.000000         ...                1173.000000
mean           0.069054         ...                   0.387894
std            0.253654         ...                   0.487478
min            0.000000         ...                   0.000000
25%            0.000000         ...                   0.000000
50%            0.000000         ...                   0.000000
75%            0.000000         ...                   1.000000
max            1.000000         ...                   1.000000

       cat_veh_value_3  cat_veh_value_4  cat_veh_value_5        prob0  \
count      1173.000000      1173.000000      1173.000000  1173.000000
mean          0.154305         0.035806         0.006820     0.374424
std           0.361395         0.185884         0.082337     0.090664
min           0.000000         0.000000         0.000000     0.103011
25%           0.000000         0.000000         0.000000     0.309238
50%           0.000000         0.000000         0.000000     0.395119
75%           0.000000         0.000000         0.000000     0.442936
max           1.000000         1.000000         1.000000     0.499845

             prob1  predicted_claim_count  predicted_freq  \
count  1173.000000                 1173.0     1173.000000
mean      0.625576                    1.0       62.845956
std       0.090664                    0.0      279.782916
min       0.500155                    1.0        1.002266
25%       0.557064                    1.0        2.308915
50%       0.604881                    1.0        4.731947
75%       0.690762                    1.0       18.792154
max       0.896989                    1.0     2695.565231

       predicted_claim_cost  risk_profile_groups
count           1173.000000               1173.0
mean             599.495267                  1.0
std              203.249216                  0.0
min              234.888395                  1.0
25%              422.905623                  1.0
50%              573.702127                  1.0
```

```
75%              767.676133                  1.0
max              991.336493                  1.0

[8 rows x 30 columns]
```

In [99]: riskprof1_stats=X_test_final_regress_riskgrp1.describe()

In [100]: X_test_final_regress_riskgrp2.describe()

Out[100]:

```
         pol_number  credit_score_imp    veh_age_1    veh_age_2    veh_age_4  \
count  1.653000e+03       1653.000000  1653.000000  1653.000000  1653.000000
mean   5.543672e+07         86.059983     0.270417     0.303085     0.153055
std    2.601978e+07          6.804156     0.444310     0.459731     0.360150
min    1.009346e+07         79.142032     0.000000     0.000000     0.000000
25%    3.238024e+07         79.142032     0.000000     0.000000     0.000000
50%    5.652248e+07         81.574030     0.000000     0.000000     0.000000
75%    7.847139e+07         91.168652     1.000000     1.000000     0.000000
max    9.997116e+07        100.000000     1.000000     1.000000     1.000000

       agecat_imputed_1  agecat_imputed_2  agecat_imputed_3  agecat_imputed_4  \
count       1653.000000       1653.000000       1653.000000       1653.000000
mean           0.289776          0.257713          0.119177          0.104053
std            0.453796          0.437508          0.324095          0.305422
min            0.000000          0.000000          0.000000          0.000000
25%            0.000000          0.000000          0.000000          0.000000
50%            0.000000          0.000000          0.000000          0.000000
75%            1.000000          1.000000          0.000000          0.000000
max            1.000000          1.000000          1.000000          1.000000

       agecat_imputed_5         ...           cat_veh_value_1  \
count       1653.000000         ...               1653.000000
mean           0.051422         ...                  0.091954
std            0.220923         ...                  0.289048
min            0.000000         ...                  0.000000
25%            0.000000         ...                  0.000000
50%            0.000000         ...                  0.000000
75%            0.000000         ...                  0.000000
max            1.000000         ...                  1.000000

       cat_veh_value_3  cat_veh_value_4  cat_veh_value_5        prob0  \
count      1653.000000      1653.000000      1653.000000  1653.000000
mean          0.256503         0.103448         0.136116     0.343986
std           0.436835         0.304636         0.343016     0.100265
min           0.000000         0.000000         0.000000     0.066423
25%           0.000000         0.000000         0.000000     0.286418
50%           0.000000         0.000000         0.000000     0.361798
75%           1.000000         0.000000         0.000000     0.424032
max           1.000000         1.000000         1.000000     0.498143
```

	prob1	predicted_claim_count	predicted_freq
count	1653.000000	1653.0	1653.000000
mean	0.656014	1.0	158.162906
std	0.100265	0.0	830.527957
min	0.501857	1.0	1.027479
25%	0.575968	1.0	3.058059
50%	0.638202	1.0	7.937693
75%	0.713582	1.0	27.991281
max	0.933577	1.0	15472.991576

	predicted_claim_cost	risk_profile_groups
count	1653.000000	1653.0
mean	2732.386661	2.0
std	1865.096725	0.0
min	1002.727364	2.0
25%	1408.264283	2.0
50%	2065.122857	2.0
75%	3336.750093	2.0
max	9918.436214	2.0

[8 rows x 30 columns]

In [101]:
```
riskprof2_stats=X_test_final_regress_riskgrp2.describe()
```

In [102]:
```
X_test_final_regress_riskgrp3.describe()
```

Out[102]:

	pol_number	credit_score_imp	veh_age_1	veh_age_2	veh_age_4
count	1.210000e+02	121.000000	121.000000	121.000000	121.0
mean	5.529849e+07	86.146275	0.892562	0.066116	0.0
std	2.768185e+07	7.487210	0.310957	0.249517	0.0
min	1.085354e+07	79.142032	0.000000	0.000000	0.0
25%	3.281682e+07	79.142032	1.000000	0.000000	0.0
50%	5.649260e+07	81.574030	1.000000	0.000000	0.0
75%	8.020378e+07	94.722907	1.000000	0.000000	0.0
max	9.918589e+07	100.000000	1.000000	1.000000	0.0

	agecat_imputed_1	agecat_imputed_2	agecat_imputed_3	agecat_imputed_4
count	121.000000	121.000000	121.000000	121.000000
mean	0.438017	0.123967	0.016529	0.033058
std	0.498206	0.330914	0.128028	0.179531
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	agecat_imputed_5	...	cat_veh_value_1

```
count       121.000000        ...                  121.0
mean          0.016529        ...                    0.0
std           0.128028        ...                    0.0
min           0.000000        ...                    0.0
25%           0.000000        ...                    0.0
50%           0.000000        ...                    0.0
75%           0.000000        ...                    0.0
max           1.000000        ...                    0.0

       cat_veh_value_3  cat_veh_value_4  cat_veh_value_5       prob0  \
count       121.000000       121.000000       121.000000  121.000000
mean          0.272727         0.314050         0.363636    0.316697
std           0.447214         0.466066         0.483046    0.121937
min           0.000000         0.000000         0.000000    0.066360
25%           0.000000         0.000000         0.000000    0.217672
50%           0.000000         0.000000         0.000000    0.339050
75%           1.000000         1.000000         1.000000    0.406846
max           1.000000         1.000000         1.000000    0.499676

            prob1  predicted_claim_count  predicted_freq  \
count  121.000000                  121.0      121.000000
mean     0.683303                    1.0      558.124253
std      0.121937                    0.0     2002.998959
min      0.500324                    1.0        1.004738
25%      0.593154                    1.0        3.959500
50%      0.660950                    1.0       11.432828
75%      0.782328                    1.0      106.631618
max      0.933640                    1.0    15530.741371

       predicted_claim_cost  risk_profile_groups
count            121.000000                121.0
mean           18632.065941                  3.0
std            14717.498474                  0.0
min            10017.995749                  3.0
25%            11248.790942                  3.0
50%            13756.092738                  3.0
75%            21934.239909                  3.0
max           142797.715332                  3.0

[8 rows x 30 columns]
```

In [103]:
```
riskprof3_stats=X_test_final_regress_riskgrp3.describe()
```

0.5.1 CSV finals for TABLEAU VISUALIZATIONS

In [104]:
```
X_test_final_V2.to_csv("low_probable_customers_2018.csv", header=True)
```

In [105]:
```
X_test_final_regress.to_csv("low_claim_cost_2018.csv", header=True)
```

```
In [106]: X_test_final_regress_riskgrp3.to_csv("risk_profile_3.csv",header=True)

In [107]: X_test_final_regress_riskgrp2.to_csv("risk_profile_2.csv",header=True)

In [108]: X_test_final_regress_riskgrp1.to_csv("risk_profile_1.csv",header=True)

In [109]: df_final_stats.to_csv("final_stats.csv",header=True)

In [110]: riskprof1_stats.to_csv("riskprof1_stats.csv",header=True)

In [111]: riskprof2_stats.to_csv("riskprof2_stats.csv",header=True)

In [112]: riskprof3_stats.to_csv("riskprof3_stats.csv",header=True)

In [ ]:
```