Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
from sqlalchemy import create_engine

import matplotlib.pyplot as plt
import seaborn as sns
import sys
import copy
import math
from scipy.stats import chi2


import warnings
from pandas.core.common import SettingWithCopyWarning

import Regression

warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import dcor
from scipy.stats import pearsonr
```

Reading the data

In [2]:
```
claims = pd.read_excel('claim_history.xlsx')

claims.head()
```

Out[2]:

	ID	KIDSDRIV	BIRTH	AGE	HOMEKIDS	YOJ	INCOME	PARENT1	HOME_VAL	MSTATUS	...	T
0	63581743	0	1939-03-16	60.0	0	11.0	67000.0	No	NaN	No	...	
1	132761049	0	1956-01-21	43.0	0	11.0	91000.0	No	257000.0	No	...	
2	921317019	0	1951-11-18	48.0	0	11.0	53000.0	No	NaN	No	...	
3	727598473	0	1964-03-05	35.0	1	10.0	16000.0	No	124000.0	Yes	...	
4	450221861	0	1948-06-05	51.0	0	14.0	NaN	No	306000.0	Yes	...	

5 rows × 26 columns

In [3]:
```
claims.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10302 entries, 0 to 10301
Data columns (total 26 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          10302 non-null  int64
 1   KIDSDRIV    10302 non-null  int64
 2   BIRTH       10302 non-null  datetime64[ns]
 3   AGE         10295 non-null  float64
 4   HOMEKIDS    10302 non-null  int64
```

```
 5   YOJ         9754 non-null   float64
 6   INCOME      9732 non-null   float64
 7   PARENT1     10302 non-null  object
 8   HOME_VAL    6819 non-null   float64
 9   MSTATUS     10302 non-null  object
 10  GENDER      10302 non-null  object
 11  EDUCATION   10302 non-null  object
 12  OCCUPATION  10302 non-null  object
 13  TRAVTIME    10302 non-null  int64
 14  CAR_USE     10302 non-null  object
 15  BLUEBOOK    10302 non-null  int64
 16  TIF         10302 non-null  int64
 17  CAR_TYPE    10302 non-null  object
 18  RED_CAR     10302 non-null  object
 19  REVOKED     10302 non-null  object
 20  MVR_PTS     10302 non-null  int64
 21  CAR_AGE     9662 non-null   float64
 22  URBANICITY  10302 non-null  object
 23  CLM_AMT     10302 non-null  int64
 24  CLM_COUNT   10302 non-null  int64
 25  EXPOSURE    10302 non-null  float64
dtypes: datetime64[ns](1), float64(6), int64(9), object(10)
memory usage: 2.0+ MB
```

In [4]:
```
claims.isna().sum()
```

Out[4]:
```
ID               0
KIDSDRIV         0
BIRTH            0
AGE              7
HOMEKIDS         0
YOJ            548
INCOME         570
PARENT1          0
HOME_VAL      3483
MSTATUS          0
GENDER           0
EDUCATION        0
OCCUPATION       0
TRAVTIME         0
CAR_USE          0
BLUEBOOK         0
TIF              0
CAR_TYPE         0
RED_CAR          0
REVOKED          0
MVR_PTS          0
CAR_AGE        640
URBANICITY       0
CLM_AMT          0
CLM_COUNT        0
EXPOSURE         0
dtype: int64
```

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

Out[5]:

	ID	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOME
count	1.030200e+04	10302.000000	10295.000000	10302.000000	9754.000000	9732.000000	6819.0
mean	4.956631e+08	0.169288	44.837397	0.720443	10.474062	61566.892725	220421.7
std	2.864675e+08	0.506512	8.606445	1.116323	4.108943	47453.597835	96337.4
min	6.317500e+04	0.000000	16.000000	0.000000	0.000000	0.000000	50000.0
25%	2.442869e+08	0.000000	39.000000	0.000000	9.000000	28000.000000	153000.0

50%	4.970043e+08	0.000000	45.000000	0.000000	11.000000	54000.000000	206000.0(
75%	7.394551e+08	0.000000	51.000000	1.000000	13.000000	86000.000000	271000.0(
max	9.999264e+08	4.000000	81.000000	5.000000	23.000000	367000.000000	885000.0(

In [6]:
```
claims['RED_CAR'] = claims['RED_CAR'].replace(['yes'], 'Yes')
claims['RED_CAR'] = claims['RED_CAR'].replace(['no'], 'No')
```

Question 1

In [7]:
```
#Severity = CLM_AMT / CLM_COUNT if CLM_COUNT > 0
claims['SEVERITY'] = np.where(claims['CLM_COUNT']>0, claims['CLM_AMT']/claims['CLM_COUNT
```

In [8]:
```
target = 'SEVERITY'
int_pred = ['AGE', 'BLUEBOOK', 'CAR_AGE', 'HOME_VAL', 'HOMEKIDS', 'INCOME', 'YOJ', 'KIDS
cat_cols = ['CAR_TYPE', 'CAR_USE', 'EDUCATION', 'GENDER', 'MSTATUS', 'PARENT1', 'RED_CAR

claims[['BLUEBOOK', 'HOME_VAL', 'INCOME']] = claims[['BLUEBOOK', 'HOME_VAL', 'INCOME']]/
train_data = claims[claims['CLM_COUNT'] > 0.0] # Only positive claims
train_data = train_data[[target] + int_pred]    # Only necessary variables
train_data = train_data.dropna().reset_index(drop=True)              # Remove missing va
train_data.shape
```

Out[8]:
```
(1274, 21)
```

In [9]:
```
n_sample = train_data.shape[0]
y_train = train_data[target]

# Build a model with only the Intercept term
X_train = train_data[[target]]
X_train.insert(0, 'Intercept', 1.0)
X_train = X_train.drop(columns = target)

result = Regression.GammaRegression(X_train, y_train)

outCoefficient = result[0]
outCovb = result[1]
outCorb = result[2]
llk = result[3]
nonAliasParam = result[4]
outIterationTable = result[5]
y_pred_intercept_only = result[6]
```

a) Please generate a histogram and a horizontal boxplot to show the distribution of Severity. For the histogram, use a bin-width of 500 and put the number of policies on the vertical axis. Put the two graphs in the same chart where the histogram is above the boxplot.

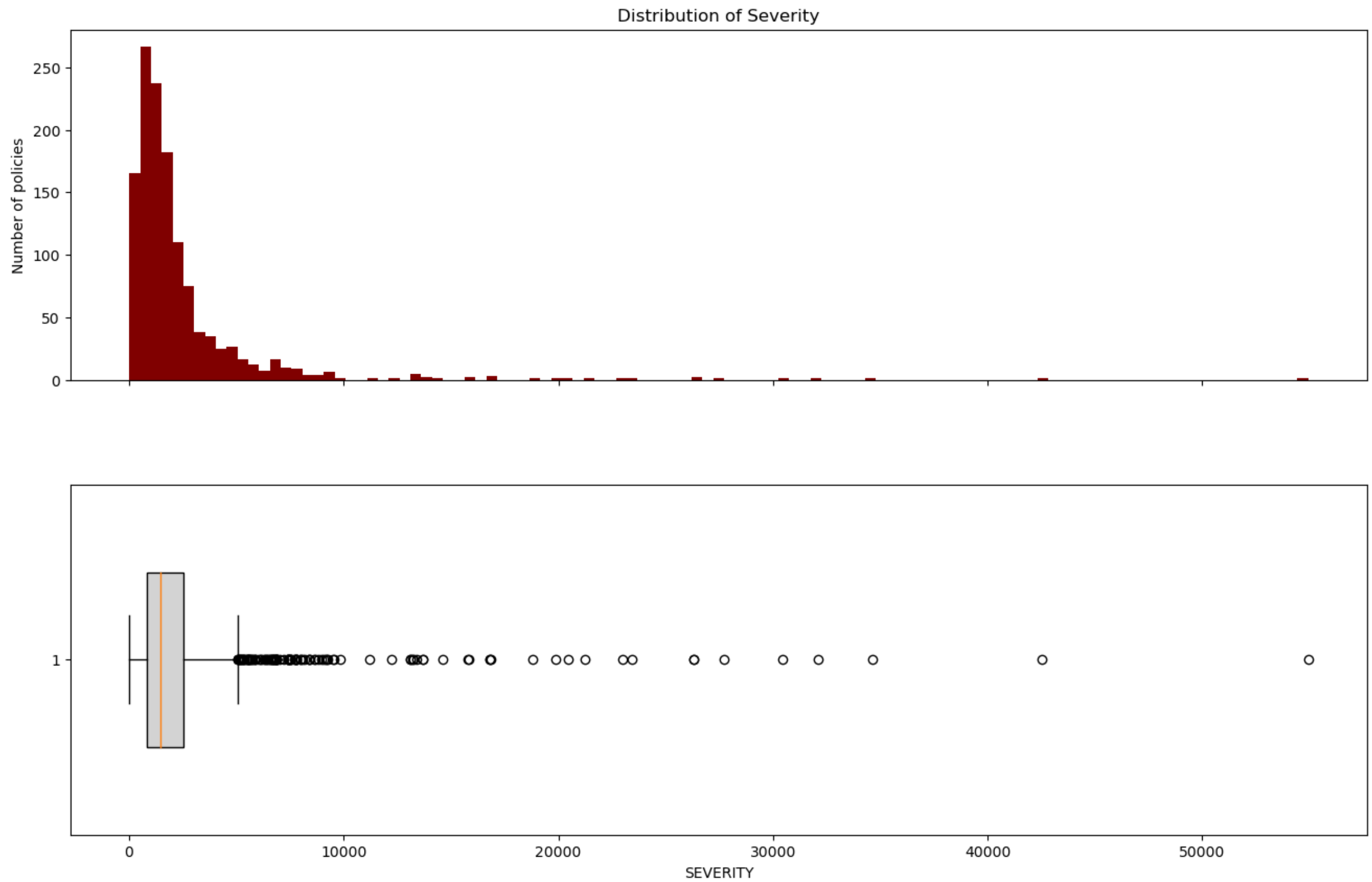
In [10]:
```
# Create a figure with two subplots, sharing the x-axis
fig, (ax1, ax2) = plt.subplots(nrows=2, sharex=True, figsize=(16, 10))

# Plot the histogram on the top subplot
binwidth = 500
ax1.hist(train_data['SEVERITY'], bins=int((max(train_data['SEVERITY']) - min(train_data[
ax1.set_ylabel('Number of policies')
ax1.set_title('Distribution of Severity')

# Plot the boxplot on the bottom subplot
ax2.boxplot(train_data['SEVERITY'], vert=False, widths=0.5, patch_artist=True, boxprops=
ax2.set_xlabel('SEVERITY')

# Adjust the layout and save the figure
```

```
plt.subplots_adjust(hspace=0.3)
plt.savefig('histogram_boxplot.png', dpi=300)
plt.show()
```



b) What is the log-likelihood value, the Akaike Information Criterion (AIC) value, and the Bayesian Information Criterion (BIC) value of the Intercept-only model?

In [11]:

```
print('Log-likelihood value : ', llk)
```

```
Log-likelihood value :  -11171.287135771177
```

In [12]:

```
def compute_aic_bic(llk, len_nonAliasParam, n_sample) :
    AIC = -2*llk + 2*len_nonAliasParam
    print('Akaike Information Criterion (AIC) value : ', AIC)
    BIC = -2*llk + len_nonAliasParam*math.log(n_sample)
    print('Bayesian Information Criterion (BIC) value : ', BIC)
```

In [13]:

```
compute_aic_bic(llk, len(nonAliasParam), n_sample)
```

```
Akaike Information Criterion (AIC) value :  22344.574271542355
Bayesian Information Criterion (BIC) value :  22349.724188378488
```

Question 2

Use the Forward Selection method to build our model. The Entry Threshold is 0.01.

a) Please provide a summary report of the Forward Selection in a table.

The report should include :

1. the step number,
2. the predictor entered,
3. the number of non-aliased parameters in the current model,
4. the log-likelihood value of the current model,

5. the Deviance Chi-squares statistic between the current and the previous models,
6. the corresponding Deviance Degree of Freedom, and
7. the corresponding Chi-square significance.

In [14]:
```python
def create_term_var(col) :
    if col in cat_cols :
        # Reorder the categories in ascending order of frequencies of the target field
        u = trainData[col].astype('category')
        u_freq = u.value_counts(ascending = True)
        pm = u.cat.reorder_categories(list(u_freq.index))
        term_var = pd.get_dummies(pm)
    else :
        term_var = trainData[[col]]
    return term_var


def update_step_summary(preds, train_model, llk_0, df_0):

    # Find the predictor
    step_detail = []
    for i in preds :
        X = train_model.join(create_term_var(i),rsuffix="_"+i)
        outList = Regression.GammaRegression(X, y_train)
        llk_1 = outList[3]
        df_1 = len(outList[4])

        deviance_chisq = 2 * (llk_1 - llk_0)
        deviance_df = df_1 - df_0
        deviance_sig = chi2.sf(deviance_chisq, deviance_df)
        step_detail.append([i, df_1, llk_1, deviance_chisq, deviance_df, deviance_sig, o
    step_detail_df = pd.DataFrame(step_detail, columns=columns+['output'])
    min_index = step_detail_df['Chi-Square Significance'].idxmin()
    min_row = step_detail_df.iloc[min_index].tolist()
    return min_row


def forward_selection() :
    preds = int_pred.copy()
    y_train = trainData[target]

    # Intercept only model
    X_train = trainData[[target]].copy()
    X_train.insert(0, 'Intercept', 1.0)
    X_train.drop(columns = [target], inplace = True)

    step_summary = []

    outList = Regression.GammaRegression(X_train, y_train)
    llk_0 = outList[3]
    df_0 = len(outList[4])
    step_summary.append(['INTERCEPT', df_0, llk_0, np.nan, np.nan, np.nan])

    chi_sig = 0
    threshold = 0.01
    while chi_sig < threshold :
        if len(preds) == 0 :
            break
        else :
            row = update_step_summary(preds, X_train, llk_0, df_0)
            llk_0 = row[2]
            df_0 = row[1]
            chi_sig = row[-2]
            if chi_sig < threshold :
                step_summary.append(row[:-1])
```

```
                X_train = X_train.join(create_term_var(row[0]),rsuffix="_"+row[0])
                out_latest_pr = row[-1]
            preds.remove(row[0])

    return step_summary, out_latest_pr
```

In [15]:

```
trainData = train_data.copy()
columns = ["Predictor", "Non-Aliased Parameters", "Log-Likelihood", "Deviance Chi-Square
                       "Degrees of Freedom", "Chi-Square Significance"]

report_data, out_pr = forward_selection()


report_df = pd.DataFrame(report_data, columns=columns).reset_index(drop=False)
report_df.rename(columns={'index': 'Step'}, inplace=True)
y_pred = out_pr[6]

report_df
```

Out[15]:

	Step	Predictor	Non-Aliased Parameters	Log-Likelihood	Deviance Chi-Squares	Degrees of Freedom	Chi-Square Significance
0	0	INTERCEPT	1	-11171.287136	NaN	NaN	NaN
1	1	BLUEBOOK	2	-11157.333240	27.907791	1.0	1.272364e-07
2	2	MSTATUS	3	-11145.906912	22.852656	1.0	1.749075e-06
3	3	RED_CAR	4	-11141.880881	8.052062	1.0	4.545189e-03
4	4	CAR_TYPE	9	-11133.503233	16.755296	5.0	4.988046e-03
5	5	YOJ	10	-11129.521708	7.963050	1.0	4.774191e-03
6	6	CAR_AGE	11	-11125.764730	7.513957	1.0	6.122271e-03

b) Our final model is the model when the Forward Selection ends. What are the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) of your final model?

In [16]:

```
last_row = report_df.iloc[-1].to_dict()
llk = last_row['Log-Likelihood']
len_nonAliasParam = last_row['Non-Aliased Parameters']
n_sample = trainData.shape[0]
```

In [17]:

```
compute_aic_bic(llk, len_nonAliasParam, n_sample)
```

```
Akaike Information Criterion (AIC) value :  22273.529459411366
Bayesian Information Criterion (BIC) value :  22330.178544608818
```

c) Please show a table of the complete set of parameters of your final model (including the aliased parameters). Besides the parameter estimates, please also include the standard errors, the 95% asymptotic confidence intervals, and the exponentiated parameter estimates. Conventionally, aliased parameters have zero standard errors and confidence intervals.

In [18]:

```
out_pr[0]
```

Out[18]:

	Estimate	Standard Error	Lower 95% CI	Upper 95% CI	Exponentiated
Intercept	7.309888	0.107671	7.098856	7.520919	1495.009028
BLUEBOOK	0.015852	0.004418	0.007193	0.024512	1.015979
No	0.319046	0.065540	0.190590	0.447502	1.375814
Yes	0.000000	0.000000	0.000000	0.000000	1.000000

Yes_RED_CAR	0.227949	0.074427	0.082074	0.373823	1.256021
No_RED_CAR	0.000000	0.000000	0.000000	0.000000	1.000000
Panel Truck	-0.033128	0.137021	-0.301685	0.235429	0.967414
Van	-0.062564	0.121346	-0.300398	0.175271	0.939353
Sports Car	0.049951	0.089328	-0.125128	0.225030	1.051219
Minivan	-0.337545	0.096101	-0.525900	-0.149191	0.713520
Pickup	-0.022679	0.087315	-0.193813	0.148455	0.977576
SUV	0.000000	0.000000	0.000000	0.000000	1.000000
YOJ	0.019040	0.007276	0.004778	0.033301	1.019222
CAR_AGE	-0.013062	0.005222	-0.023298	-0.002827	0.987023

Question 3

In [19]:

```
def PearsonCorrelation (x, y):
    '''Compute the Pearson correlation between two arrays x and y with the
       same number of values

       Argument:
       ---------
       x : a Pandas Series
       y : a Pandas Series

       Output:
       -------
       rho : Pearson correlation
       '''

    dev_x = x - np.mean(x)
    dev_y = y - np.mean(y)

    ss_xx = np.mean(dev_x * dev_x)
    ss_yy = np.mean(dev_y * dev_y)

    if (ss_xx > 0.0 and ss_yy > 0.0):
        ss_xy = np.mean(dev_x * dev_y)
        rho = (ss_xy / ss_xx) * (ss_xy / ss_yy)
        rho = np.sign(ss_xy) * np.sqrt(rho)
    else:
        rho = np.nan

    return (rho)

def RankOfValue (v):
    '''Compute the ranks of the values in an array v. For tied values, the
    average rank is computed.

    Argument:
    ---------
    v : a Pandas Series

    Output:
    -------
    rankv : Ranks of the values of v, minimum has a rank of zero
    '''

    uvalue, uinv, ucount = np.unique(v, return_inverse = True, return_counts = True)
    urank = []
    ur0 = 0
```

```
    for c in ucount:
        ur1 = ur0 + c - 1
        urank.append((ur0 + ur1)/2.0)
        ur0 = ur1 + 1

    rankv = []
    for j in uinv:
        rankv.append(urank[j])

    return (rankv)

def SpearmanCorrelation (x, y):
    '''Compute the Spearman rank-order correlation between two arrays x and y
    with the same number of values

    Argument:
    ---------
    x : a Pandas Series
    y : a Pandas Series

    Output:
    -------
    srho : Spearman rank-order correlation
    '''

    rank_x = RankOfValue(x)
    rank_y = RankOfValue(y)

    srho = PearsonCorrelation(rank_x, rank_y)
    return (srho)

def KendallTaub (x, y):
    '''Compute the Kendall's Tau-b correlation between two arrays x and y
    with the same number of values

    Argument:
    ---------
    x : a Pandas Series
    y : a Pandas Series

    Output:
    -------
    taub : Kendall's tau-b correlation
    '''

    nconcord = 0
    ndiscord = 0
    tie_x = 0
    tie_y = 0
    tie_xy = 0

    x_past = []
    y_past = []
    for xi, yi in zip(x, y):
        for xj, yj in zip(x_past, y_past):
            if (xi > xj):
                if (yi > yj):
                    nconcord = nconcord + 1
                elif (yi < yj):
                    ndiscord = ndiscord + 1
                else:
                    tie_y = tie_y + 1
            elif (xi < xj):
                if (yi < yj):
                    nconcord = nconcord + 1
                elif (yi > yj):
```

```
                    ndiscord = ndiscord + 1
                else:
                    tie_y = tie_y + 1
            else:
                if (yi == yj):
                    tie_xy = tie_xy + 1
                else:
                    tie_x = tie_x + 1

        x_past.append(xi)
        y_past.append(yi)

    denom = (nconcord + ndiscord + tie_x) * (nconcord + ndiscord + tie_y)
    if (denom > 0.0):
        taub = (nconcord - ndiscord) / np.sqrt(denom)
    else:
        taub = np.nan

    return (taub)

def AdjustedDistance (x):
    '''Compute the adjusted distances for an array x

    Argument:
    ---------
    x : a Pandas Series

    Output:
    -------
    adj_distance : Adjusted distances
    '''

    a_matrix = []
    row_mean = []

    for xi in x:
        a_row = np.abs(x - xi)
        row_mean.append(np.mean(a_row))
        a_matrix.append(a_row)
    total_mean = np.mean(row_mean)

    adj_m = []
    for row, rm in zip(a_matrix, row_mean):
        row = (row - row_mean) - (rm - total_mean)
        adj_m.append(row)

    return (np.array(adj_m))

def DistanceCorrelation (x, y):
    '''Compute the Distance correlation between two arrays x and y
    with the same number of values

    Argument:
    ---------
    x : a Pandas Series
    y : a Pandas Series

    Output:
    -------
    dcorr : Distance correlation
    '''

    adjD_x = AdjustedDistance (x)
    adjD_y = AdjustedDistance (y)

    v2sq_x = np.mean(np.square(adjD_x))
```

```
    v2sq_y = np.mean(np.square(adjD_y))
    v2sq_xy = np.mean(adjD_x * adjD_y)


    if (v2sq_x > 0.0 and v2sq_y > 0.0):
        dcorr = (v2sq_xy / v2sq_x) * (v2sq_xy / v2sq_y)
        dcorr = np.power(dcorr, 0.25)
    else :
        dcorr = None

    return (dcorr)
```

In [20]:
```
def compute_error_metrics(y_true, y_pred) :

    # Simple Residual
    y_simple_residual = y_true - y_pred

    # Root Mean Squared Error
    mse = np.mean(np.power(y_simple_residual, 2))
    rmse = np.sqrt(mse)
    print("RMSE :", rmse)

    # Relative Error
    relerr = mse / np.var(y_true, ddof = 0)
    print("Relative Error :", relerr)

    # Pearson Correlation
    pearson_corr = PearsonCorrelation (y_true, y_pred)
    print("Pearson Correlation:", pearson_corr)

    # Distance Correlation
    distance_corr = DistanceCorrelation (y_true, y_pred)
    print("Distance Correlation :", distance_corr)

    # Mean Absolute Proportion Error
    ape = np.abs(y_simple_residual) / y_train
    mape = np.mean(ape)
    print("Mean Absolute Proportion Error : ", mape)
```

a) Calculate the Root Mean Squared Error, the Relative Error, the Pearson correlation, the Distance correlation, and the Mean Absolute Proportion Error for the Intercept-only model.

In [21]:
```
compute_error_metrics(y_train, y_pred_intercept_only)
```

```
RMSE : 3667.071626635712
Relative Error : 0.9999999999999999
Pearson Correlation: -2.4563378930425157e-16
Distance Correlation : None
Mean Absolute Proportion Error :  1.8861830475373358
```

b) Calculate the Root Mean Squared Error, the Relative Error, the Pearson correlation, the Distance correlation, and the Mean Absolute Proportion Error for our final model in Question 2.

In [22]:
```
compute_error_metrics(y_train, y_pred)
```

```
RMSE : 3613.455853205644
Relative Error : 0.9709720320449472
Pearson Correlation: 0.1706247950343188
Distance Correlation : 0.15112268051220543
Mean Absolute Proportion Error :  1.8216669193905308
```

c) We will compare the goodness-of-fit of your model with that of the saturated model. We will calculate the Pearson Chi-Squares and the Deviance Chi-Squares statistics, their degrees of

freedom, and their significance values. Based on the results, do you think your model is statistically the same as the saturated Model?

In [23]:
```
# Pearson Residual
y_simple_residual = y_train - y_pred
y_pearson_residual = y_simple_residual / np.sqrt(y_pred)
# Deviance Residual
r_vec = y_train / y_pred
di_2 = 2 * (r_vec - np.log(r_vec) - 1)
y_deviance_residual = np.where(y_simple_residual > 0, 1.0, -1.0) * np.sqrt(di_2)

pearson_chisq = np.sum(np.power(y_pearson_residual, 2.0))
deviance_chisq = np.sum(np.power(y_deviance_residual, 2.0))

df_chisq = n_sample - len_nonAliasParam

pearson_sig = chi2.sf(pearson_chisq, df_chisq)
deviance_sig = chi2.sf(deviance_chisq, df_chisq)

pd.DataFrame(data = [['Pearson', pearson_chisq, df_chisq, pearson_sig],['Deviance', devi
            columns = ['Type', 'Statistic', 'Degrees of Freedom', 'Significance (p-value
```

Out[23]:

	Type	Statistic	Degrees of Freedom	Significance (p-value)
0	Pearson	5.887085e+06	1263	0.000000
1	Deviance	1.183202e+03	1263	0.946205

From the computed statistics we can clearly see that our model is not the statistically same as the saturated model.

Question 4

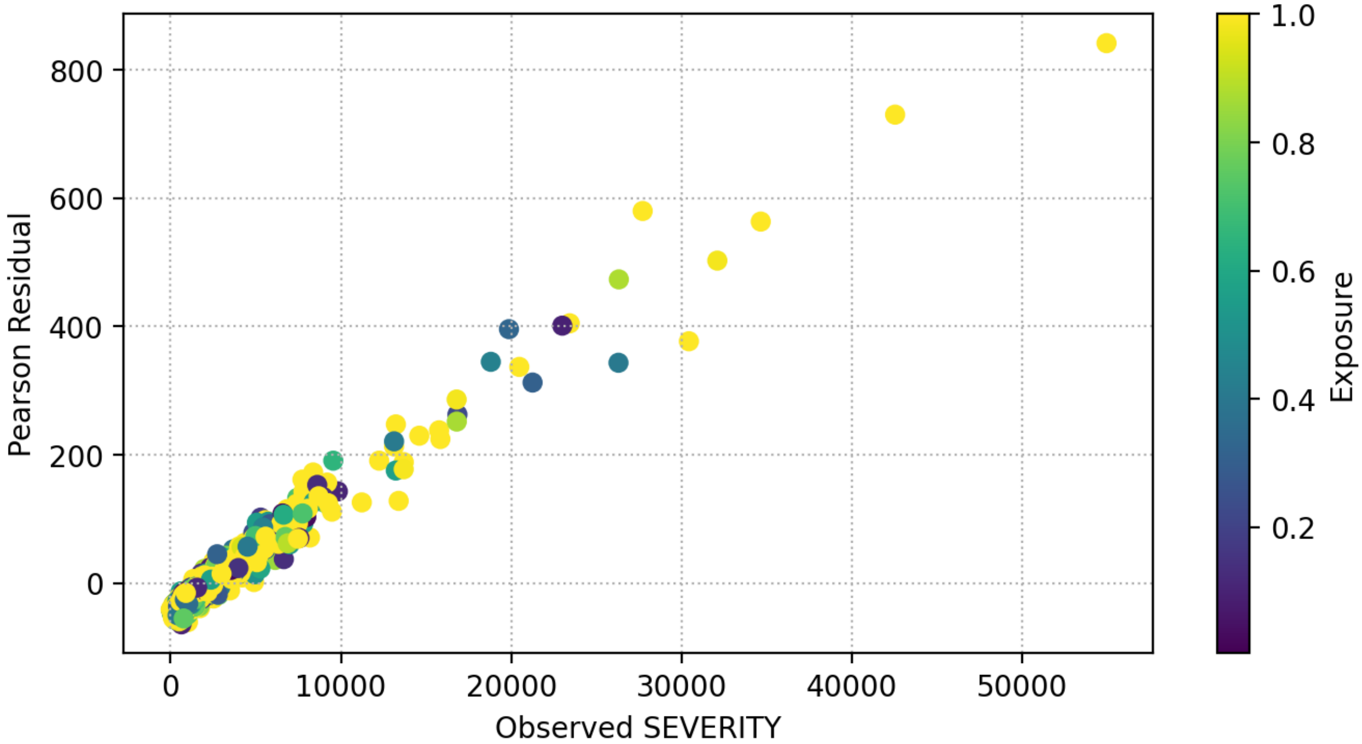
You will visually assess your final model in Question 2. Please color-code the markers according to the magnitude of the Exposure value. You must properly label the axes, add grid lines, and choose appropriate tick marks to receive full credit.

1. Plot the Pearson residuals versus the observed Severity.

In [24]:
```
exp_train_data = claims[claims['CLM_COUNT'] > 0.0] # Only positive claims
exp_train_data = exp_train_data[[target] + int_pred + ['EXPOSURE']]    # Only necessary v
exp_train_data = exp_train_data.dropna().reset_index(drop=True)              # Remove mi
```

In [25]:
```
# Plot Pearson residuals

y_resid = y_train - y_pred
pearsonResid = np.where(y_pred > 0.0, y_resid / np.sqrt(y_pred), np.NaN)
plt.figure(figsize = (8,4), dpi = 200)
sg = plt.scatter(y_train, pearsonResid, c = exp_train_data['EXPOSURE'], marker = 'o')
plt.xlabel('Observed SEVERITY')
plt.ylabel('Pearson Residual')
plt.grid(axis = 'both', linestyle = 'dotted')
plt.colorbar(sg, label = 'Exposure')
plt.show()
```

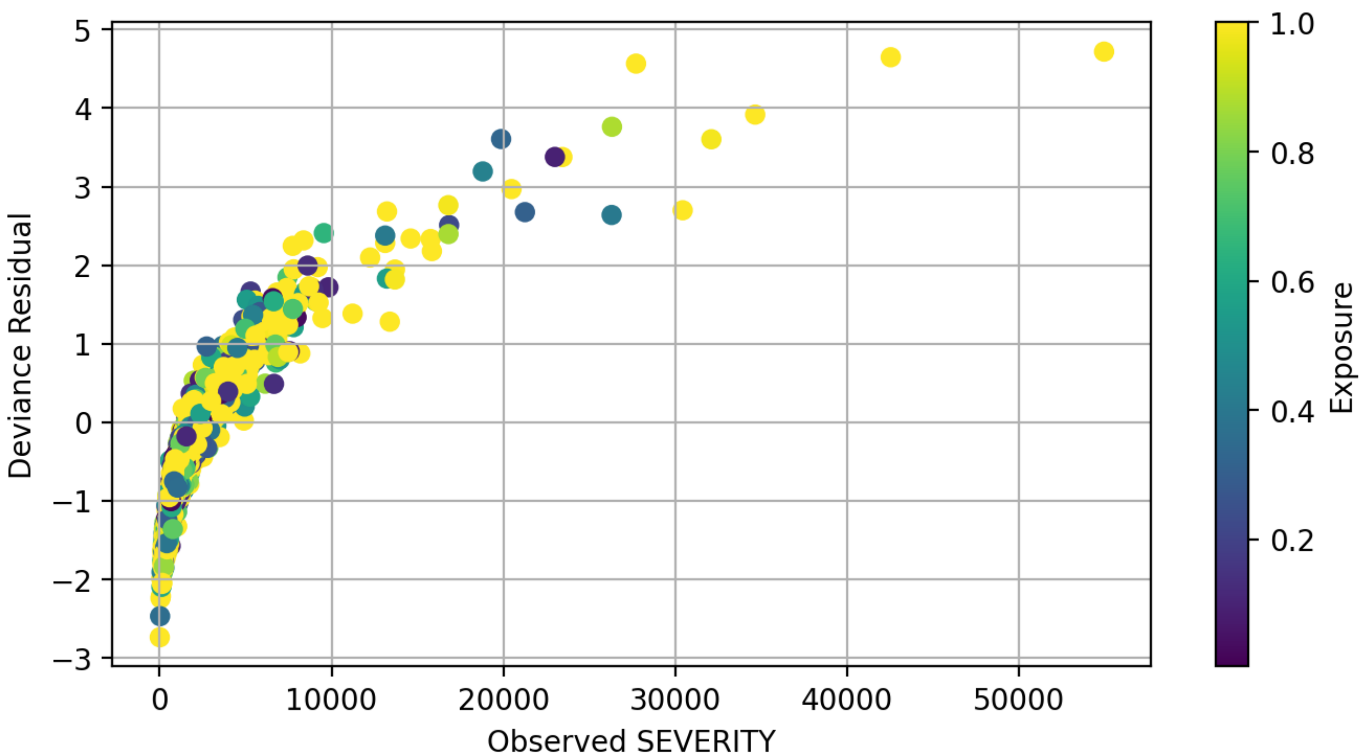


2. Plot the Deviance residuals versus the observed Severity.

In [26]:

```
# Plot Deviance residuals

plt.figure(figsize = (8,4), dpi = 200)
sg = plt.scatter(y_train, y_deviance_residual, c = exp_train_data['EXPOSURE'], marker =
plt.xlabel('Observed SEVERITY')
plt.ylabel('Deviance Residual')
plt.grid(axis = 'both')
plt.colorbar(sg, label = 'Exposure')
plt.show()
```



In [ ]: