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)
```

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.isna().sum()
```

Out[3]:

```
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 [4]:

```
claims.describe()
```

Out[4]:

	ID	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOMI
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(

Question 1

In [5]:

```
target = 'CLM_COUNT'
exposure = 'EXPOSURE'
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[exposure] > 0.0] # Only positive exposure
train_data = train_data[[target] + [exposure] + int_pred]    # Only necessary variables
train_data = train_data.dropna().reset_index(drop=True)            # Remove missing va
train_data.shape
```

Out[5]:

```
(5715, 22)
```

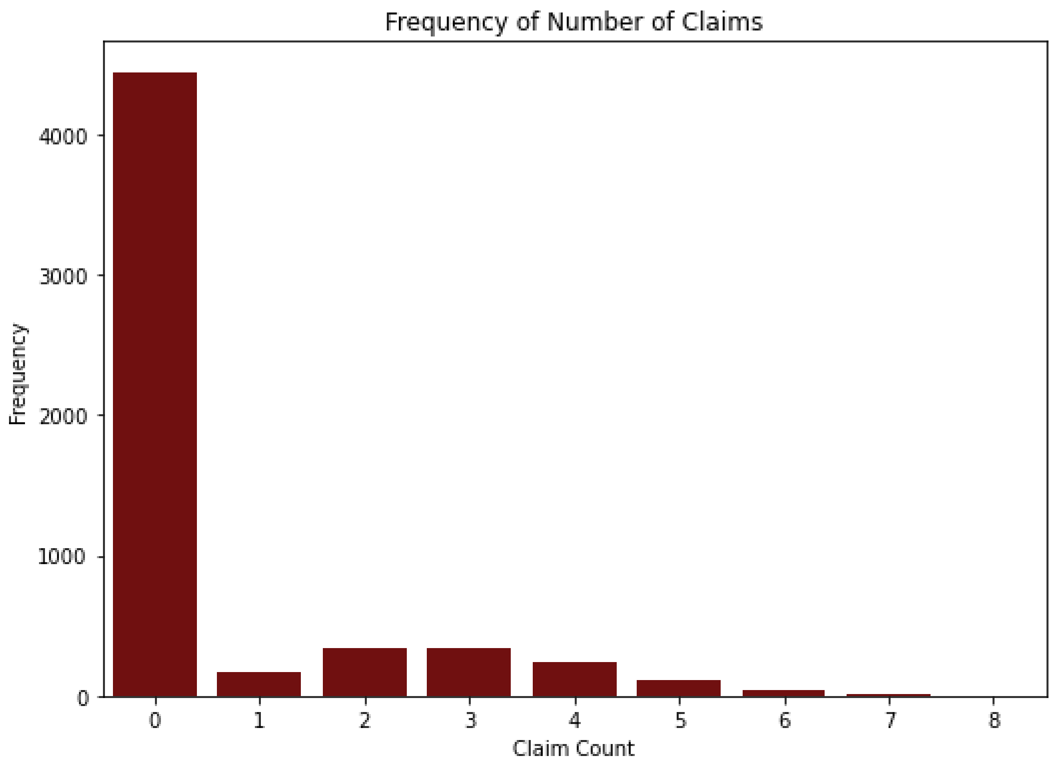
In [6]:

```
n_sample = train_data.shape[0]
y_train = train_data[target]
o_train = np.log(train_data[exposure])

# 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.PoissonRegression (X_train, y_train, o_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 vertical bar chart to show the frequency of the number of claims.

In [7]:
```
plt.rcParams["figure.figsize"] = [7.50, 5.50]
plt.rcParams["figure.autolayout"] = True

sns.countplot(x='CLM_COUNT', data=train_data, color = 'maroon')
plt.xlabel('Claim Count')
plt.ylabel('Frequency')
plt.title('Frequency of Number of Claims')
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? Here are the formulas for AIC and BIC. Suppose l is the log-likelihood of a model, p is the number of non-aliased parameters in the model, and n is the number of observations used for training the model. Then $AIC = -2l + 2p$ and $BIC = -2l + p \log e\ n$.

In [8]:
```
result
```

Out[8]:
```
[            Estimate  Standard Error  Lower 95% CI  Upper 95% CI  Exponentiated
 Intercept -0.263669        0.016178     -0.295376     -0.231962       0.768228,
            Intercept
 Intercept   0.000262,
            Intercept
 Intercept        1.0,
 -9202.190712554877,
 [0],
    Iteration  Log-Likelihood  N Step-Halving  Intercept
 0          0    -9231.843623               0  -0.141620
 1          1    -9202.288726               0  -0.256515
 2          2    -9202.190714               0  -0.263643
 3          3    -9202.190713               0  -0.263669
 4          4    -9202.190713               0  -0.263669
 5          5    -9202.190713               0  -0.263669,
 0        0.768228
```

```
1       0.636093
2       0.768228
3       0.768228
4       0.768228
          ...
5710    0.768228
5711    0.768228
5712    0.768228
5713    0.768228
5714    0.768228
Name: EXPOSURE, Length: 5715, dtype: float64]
```

In [9]:
```
print('Log-likelihood value : ', llk)
```

```
Log-likelihood value :  -9202.190712554877
```

In [10]:
```
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 [11]:
```
compute_aic_bic(llk, len(nonAliasParam), n_sample)
```

```
Akaike Information Criterion (AIC) value :  18406.381425109754
Bayesian Information Criterion (BIC) value :  18413.032274685982
```

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 [12]:
```
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.PoissonRegression(X, y_train, o_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]
    o_train = np.log(trainData[exposure])

    # 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.PoissonRegression(X_train, y_train, o_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 [13]:

```
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[13]:

Step	Predictor	Non-Aliased Parameters	Log-Likelihood	Deviance Chi-Squares	Degrees of Freedom	Chi-Square Significance

0	0	INTERCEPT	1	-9202.190713	NaN	NaN	NaN
1	1	URBANICITY	2	-8796.722613	810.936199	1.0	2.261281e-178
2	2	EDUCATION	6	-8488.805338	615.834550	4.0	5.795205e-132
3	3	MVR_PTS	7	-8349.865822	277.879032	1.0	2.176697e-62
4	4	CAR_TYPE	12	-8234.985620	229.760403	5.0	1.203794e-47
5	5	TRAVTIME	13	-8163.342263	143.286714	1.0	5.087948e-33
6	6	CAR_USE	14	-8097.875925	130.932677	1.0	2.561479e-30
7	7	KIDSDRIV	15	-8049.321029	97.109791	1.0	6.558776e-23
8	8	INCOME	16	-8000.010438	98.621184	1.0	3.057312e-23
9	9	REVOKED	17	-7958.349167	83.322541	1.0	6.970030e-20
10	10	TIF	18	-7921.746150	73.206035	1.0	1.167981e-17
11	11	PARENT1	19	-7887.529214	68.433872	1.0	1.312044e-16
12	12	BLUEBOOK	20	-7868.500958	38.056513	1.0	6.872492e-10
13	13	MSTATUS	21	-7859.308943	18.384030	1.0	1.805650e-05
14	14	HOMEKIDS	22	-7854.368864	9.880158	1.0	1.670706e-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 [14]:
```
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 [15]:
```
compute_aic_bic(llk, len_nonAliasParam, n_sample)
```

```
Akaike Information Criterion (AIC) value :  15752.737727294354
Bayesian Information Criterion (BIC) value :  15899.05641797139
```

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 [16]:
```
out_pr[0]
```

Out[16]:

	Estimate	Standard Error	Lower 95% CI	Upper 95% CI	Exponentiated
Intercept	-0.401503	0.076407	-0.551257	-0.251749	0.669313
Highly Rural/ Rural	-1.943926	0.081602	-2.103863	-1.783989	0.143141
Highly Urban/ Urban	0.000000	0.000000	0.000000	0.000000	1.000000
PhD	0.139734	0.076870	-0.010929	0.290397	1.149968
Below High Sc	0.477629	0.055029	0.369773	0.585485	1.612247
Masters	-0.068989	0.055441	-0.177651	0.039673	0.933337
High School	0.403391	0.044824	0.315538	0.491244	1.496892
Bachelors	0.000000	0.000000	0.000000	0.000000	1.000000
MVR_PTS	0.080916	0.006532	0.068114	0.093718	1.084280
Panel Truck	-0.038398	0.080251	-0.195687	0.118891	0.962330

Van	-0.038103	0.066997	-0.169415	0.093210	0.962614
Sports Car	0.073540	0.052107	-0.028588	0.175668	1.076311
Pickup	-0.234377	0.050293	-0.332949	-0.135805	0.791063
Minivan	-0.549465	0.050585	-0.648610	-0.450320	0.577258
SUV	0.000000	0.000000	0.000000	0.000000	1.000000
TRAVTIME	0.011964	0.001009	0.009986	0.013942	1.012036
Commercial	0.529038	0.040710	0.449247	0.608829	1.697299
Private	0.000000	0.000000	0.000000	0.000000	1.000000
KIDSDRIV	0.217802	0.028881	0.161196	0.274407	1.243340
INCOME	-0.004413	0.000528	-0.005449	-0.003378	0.995596
Yes	0.368634	0.042196	0.285931	0.451336	1.445758
No	0.000000	0.000000	0.000000	0.000000	1.000000
TIF	-0.037425	0.004211	-0.045678	-0.029173	0.963266
Yes_PARENT1	0.203617	0.072481	0.061557	0.345677	1.225828
No_PARENT1	0.000000	0.000000	0.000000	0.000000	1.000000
BLUEBOOK	-0.016369	0.002708	-0.021675	-0.011062	0.983765
No_MSTATUS	0.240219	0.048193	0.145763	0.334675	1.271528
Yes_MSTATUS	0.000000	0.000000	0.000000	0.000000	1.000000
HOMEKIDS	0.053730	0.016894	0.020619	0.086841	1.055200

Question 3

We will use accuracy metrics to assess the Intercept-only model and our final model in Question 2. These metrics inform us from various perspectives how well the predicted number of claims agrees with the observed number of claims.

In [17]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import dcor
from scipy.stats import pearsonr

def compute_error_metrics(y_true, y_pred) :

    # Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print("RMSE:", rmse)

    # Relative Error
    obs_mean = np.mean(y_true)
    rel_error = (np.sum(np.power(y_train - y_pred, 2)) / n_sample) / (np.var(y_train, dd
    print("Relative Error:", rel_error)

    # Pearson Correlation
    pearson, _ = pearsonr(y_true, y_pred)
    print("Pearson Correlation:", pearson)

    # Distance Correlation
    distance_correlation = dcor.distance_correlation(y_true.astype(float), y_pred.astype
    print("Distance Correlation:", distance_correlation)
    list(np.float_(y_true))
```

```
    # R-Squared
    # r2 = r2_score(y_true, y_pred)
    num = np.sum(((y_pred - np.mean(y_pred))*(y_true - np.mean(y_true)))**2)
    den = np.sum((y_pred - np.mean(y_pred))**2) * np.sum((y_true - np.mean(y_true))**2)
    r2 = num/den
    print("R-Squared:", r2)
```

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

In [18]:

```
compute_error_metrics(y_train, y_pred_intercept_only)
```

```
RMSE: 1.4635157608954519
Relative Error: 1.0759932377154993
Pearson Correlation: -0.19138309783283575
Distance Correlation: 0.22968068175486067
R-Squared: 0.00023764977519819723
```

b) Calculate the Root Mean Squared Error, the Relative Error, the Pearson correlation, the Distance correlation, and the R-squared metrics for our final model in Question 2.

In [19]:

```
compute_error_metrics(y_train, y_pred)
```

```
RMSE: 1.3946224733422352
Relative Error: 0.9770753409645692
Pearson Correlation: 0.2614873869285943
Distance Correlation: 0.2806093073756397
R-Squared: 0.0002784862880145033
```

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 [20]:

```
# Calculate Pearson residual
y_resid = y_train - y_pred
pearsonResid = np.where(y_pred > 0.0, y_resid / np.sqrt(y_pred), np.NaN)

# Calculate Deviance residual
yPos = -1*((y_train * np.log(y_pred/y_train))+(y_train-y_pred))
dR2 = np.where(y_train > 0.0, yPos, 0)
devResid = np.where(y_train > y_pred, 1.0, -1.0) * np.where(dR2 > 0.0, np.sqrt(2.0 * dR2


pearson_chisq = np.sum(np.power(pearsonResid, 2.0))
deviance_chisq = np.sum(np.power(devResid, 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[20]:

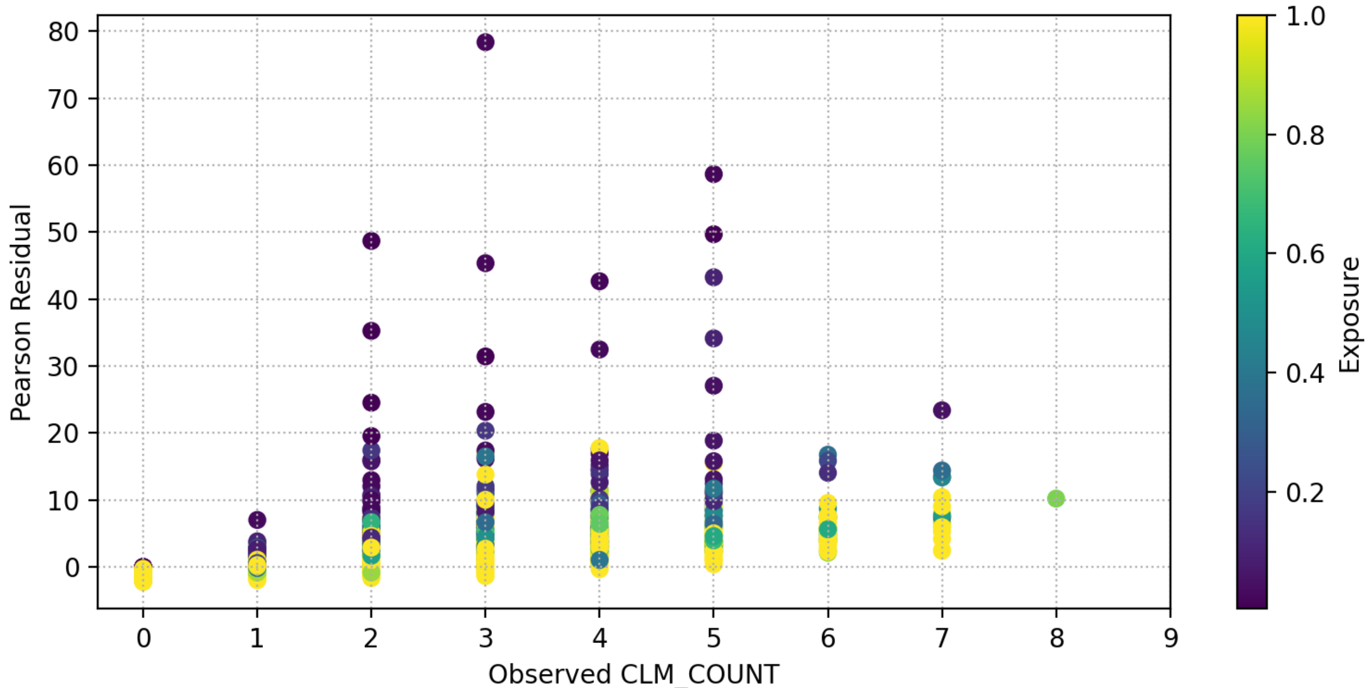
	Type	Statistic	Degrees of Freedom	Significance (p-value)
0	Pearson	54445.804414	5693	0.000000e+00
1	Deviance	7047.060487	5693	1.736820e-32

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.

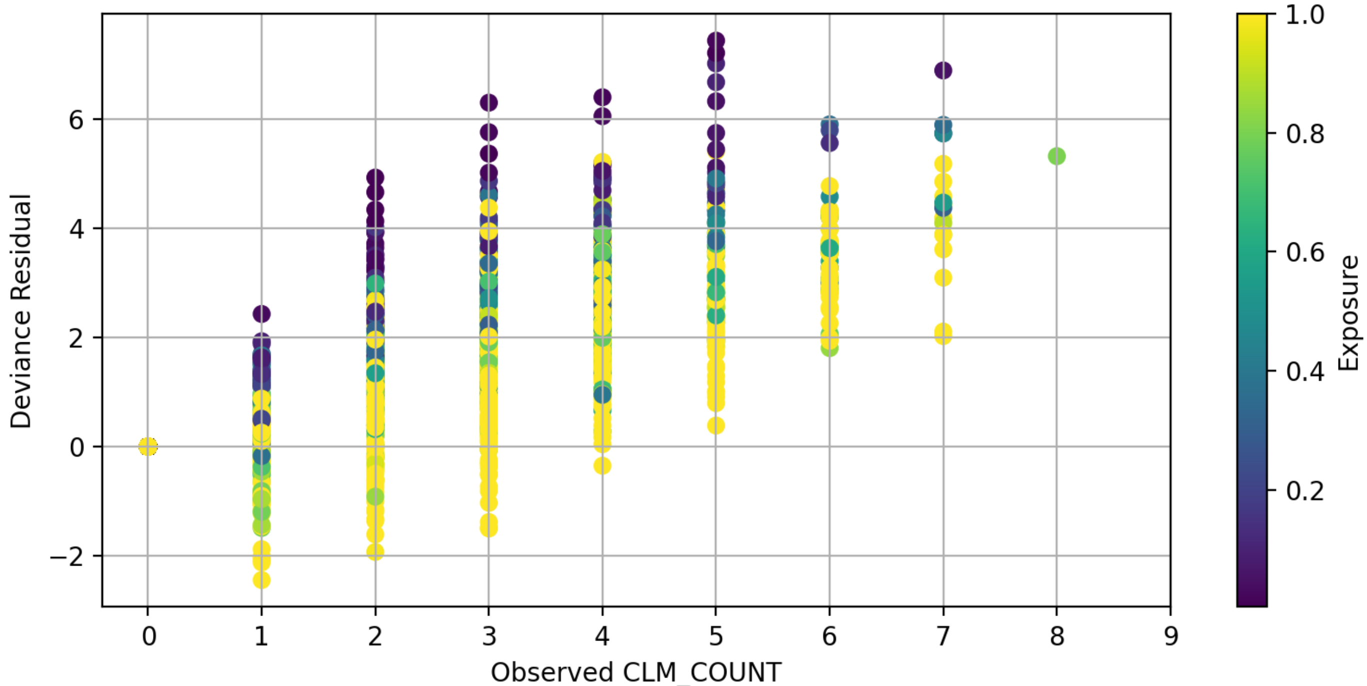
1. Plot the Pearson residuals versus the observed number of claims.

In [21]:
```
# 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 = train_data['EXPOSURE'], marker = 'o')
plt.xlabel('Observed CLM_COUNT')
plt.ylabel('Pearson Residual')
plt.xticks(range(10))
plt.grid(axis = 'both', linestyle = 'dotted')
plt.colorbar(sg, label = 'Exposure')
plt.show()
```



2. Plot the Deviance residuals versus the observed number of claims.

In [22]:
```
# Plot Deviance residuals

plt.figure(figsize = (8,4), dpi = 200)
sg = plt.scatter(y_train, devResid, c = train_data['EXPOSURE'], marker = 'o')
plt.xlabel('Observed CLM_COUNT')
plt.ylabel('Deviance Residual')
plt.xticks(range(10))
plt.grid(axis = 'both')
plt.colorbar(sg, label = 'Exposure')
plt.show()
```



Deviance Residual
Observed CLM_COUNT
Exposure
−2
0
2
4
6
0
1
2
3
4
5
6
7
8
9
0.2
0.4
0.6
0.8
1.0

In [ ]: