In [1]:
```python
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
np.seterr(divide = 'ignore')

from lifelines import CoxPHFitter
from scipy.stats import norm
```

In [ ]:
```python
from scipy.stats import chi2

# Set some options for printing all the columns
np.set_printoptions(precision = 10, threshold = sys.maxsize)
np.set_printoptions(linewidth = np.inf)
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)
pd.set_option('max_colwidth', None)
pd.options.display.float_format = '{:,.7f}'.format
```

In [2]:
```python
claim_history = pd.read_excel("claim_history.xlsx")
```

In [3]:
```python
claim_history.head()
```

Out[3]:

	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 [4]:
```python
yName = 'CLM_AMT'
eName = 'EXPOSURE'
intName = ['AGE', 'BLUEBOOK', 'CAR_AGE', 'HOME_VAL', 'HOMEKIDS', 'INCOME', 'YOJ', 'KIDSD
```

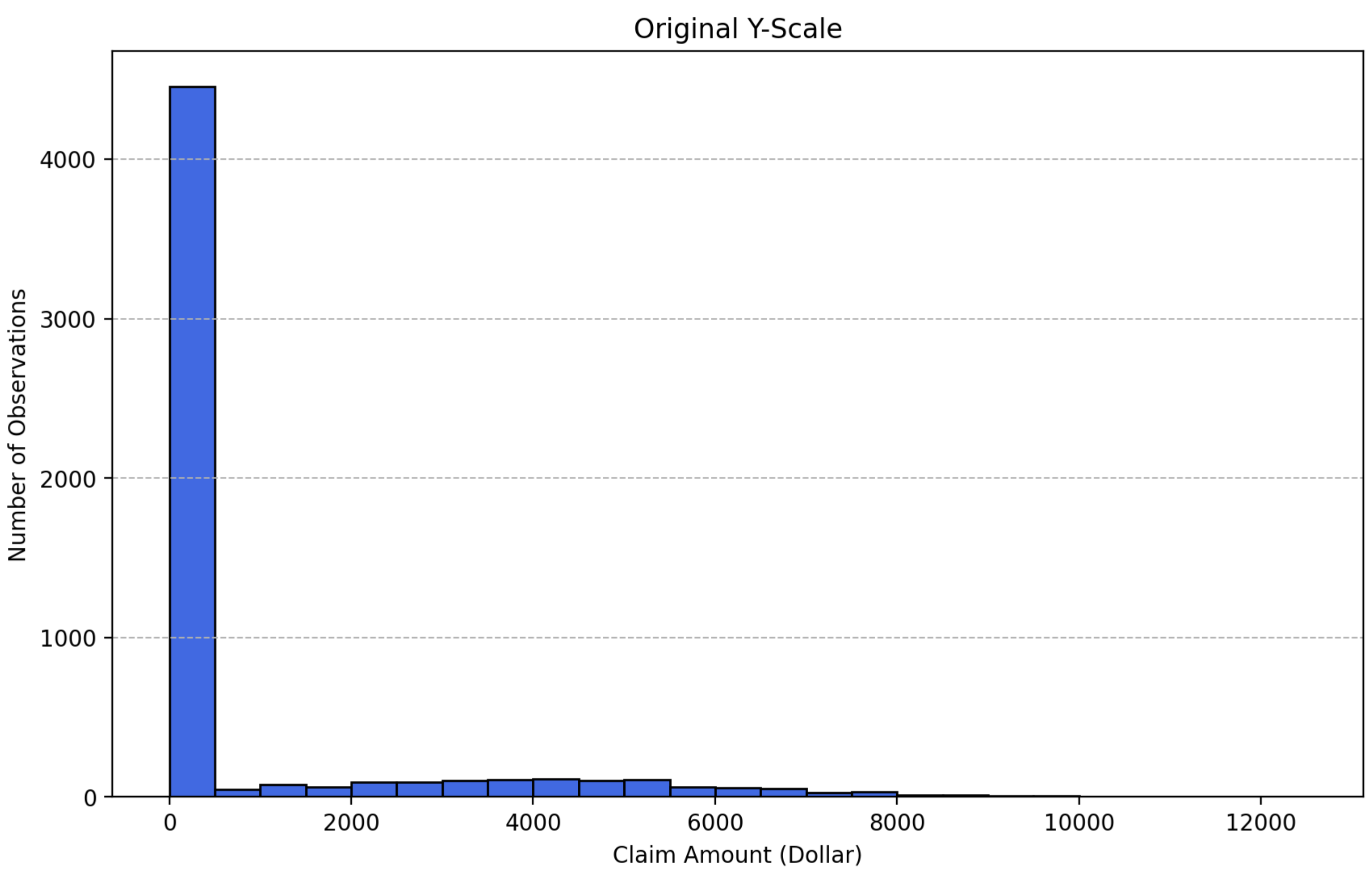
```
catName = ['CAR_TYPE', 'CAR_USE', 'EDUCATION', 'GENDER', 'MSTATUS', 'PARENT1', 'RED_CAR'

train_data = claim_history[[yName, eName] + catName + intName]
train_data = train_data[train_data[eName] > 0.0].dropna().reset_index(drop = True)
y_train = train_data[yName]
o_train = np.log(train_data[eName])
train_data.shape
```

Out[4]: (5715, 22)

Question 1

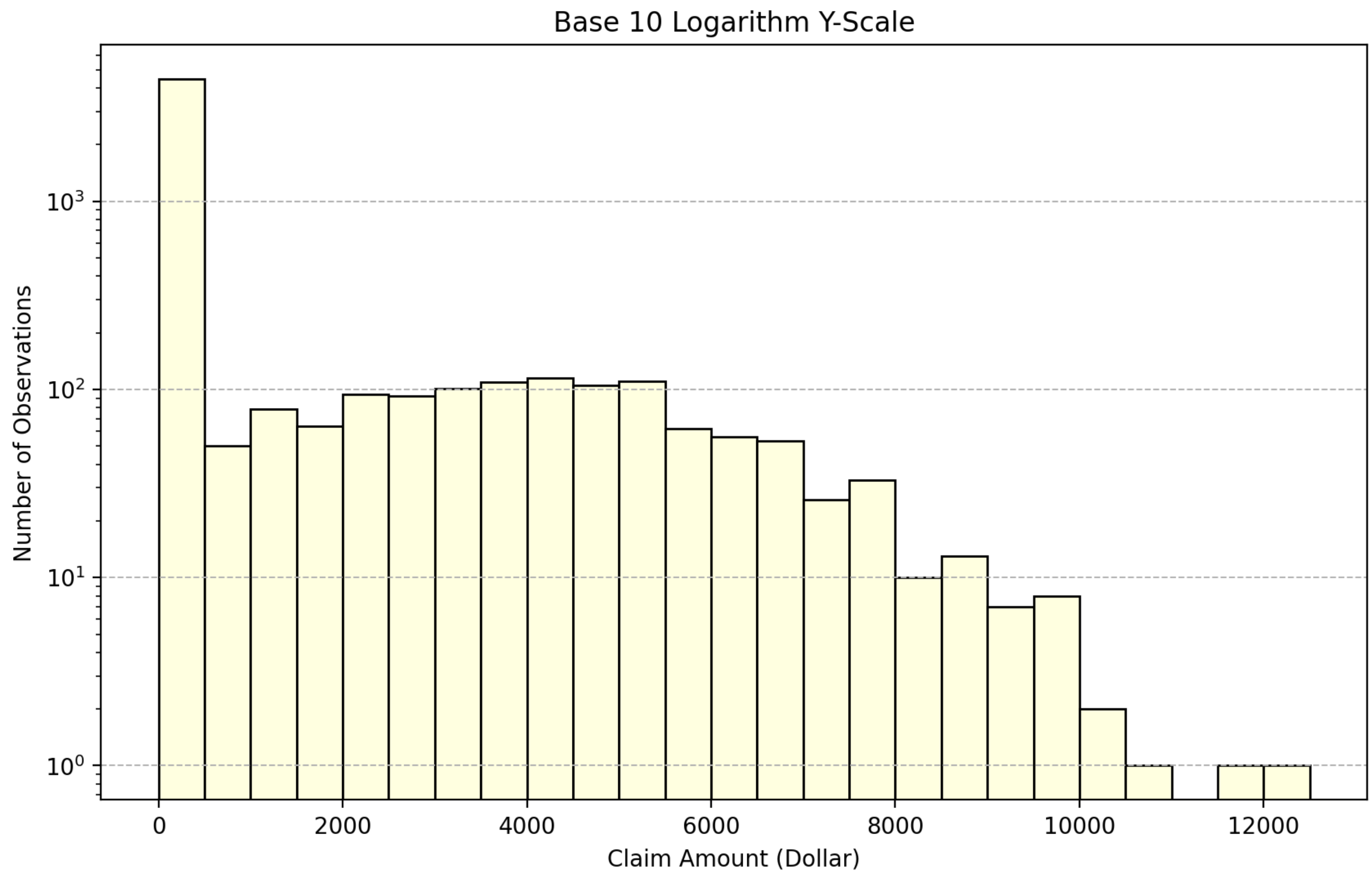
a) (10 points). We will first estimate the Tweedie distribution's Power parameter p and Scale parameter ϕ. To this end, we will calculate the sample means and the sample variances of the claim amount for each value combination of the categorical predictors. Then, we will train a linear regression model to help us estimate the two parameters. What are their values? Please provide us with your appropriate chart

In [5]:
```
# Histogram of Claim Amount
plt.figure(figsize = (10,6), dpi = 200)
plt.hist(y_train, bins = np.arange(0,13000,500), fill = True, color =
'royalblue', edgecolor = 'black')
plt.title('Original Y-Scale')
plt.xlabel('Claim Amount (Dollar)')
plt.ylabel('Number of Observations')
plt.xticks(np.arange(0,14000,2000))
plt.grid(axis = 'y', linewidth = 0.7, linestyle = 'dashed')
plt.show()
```

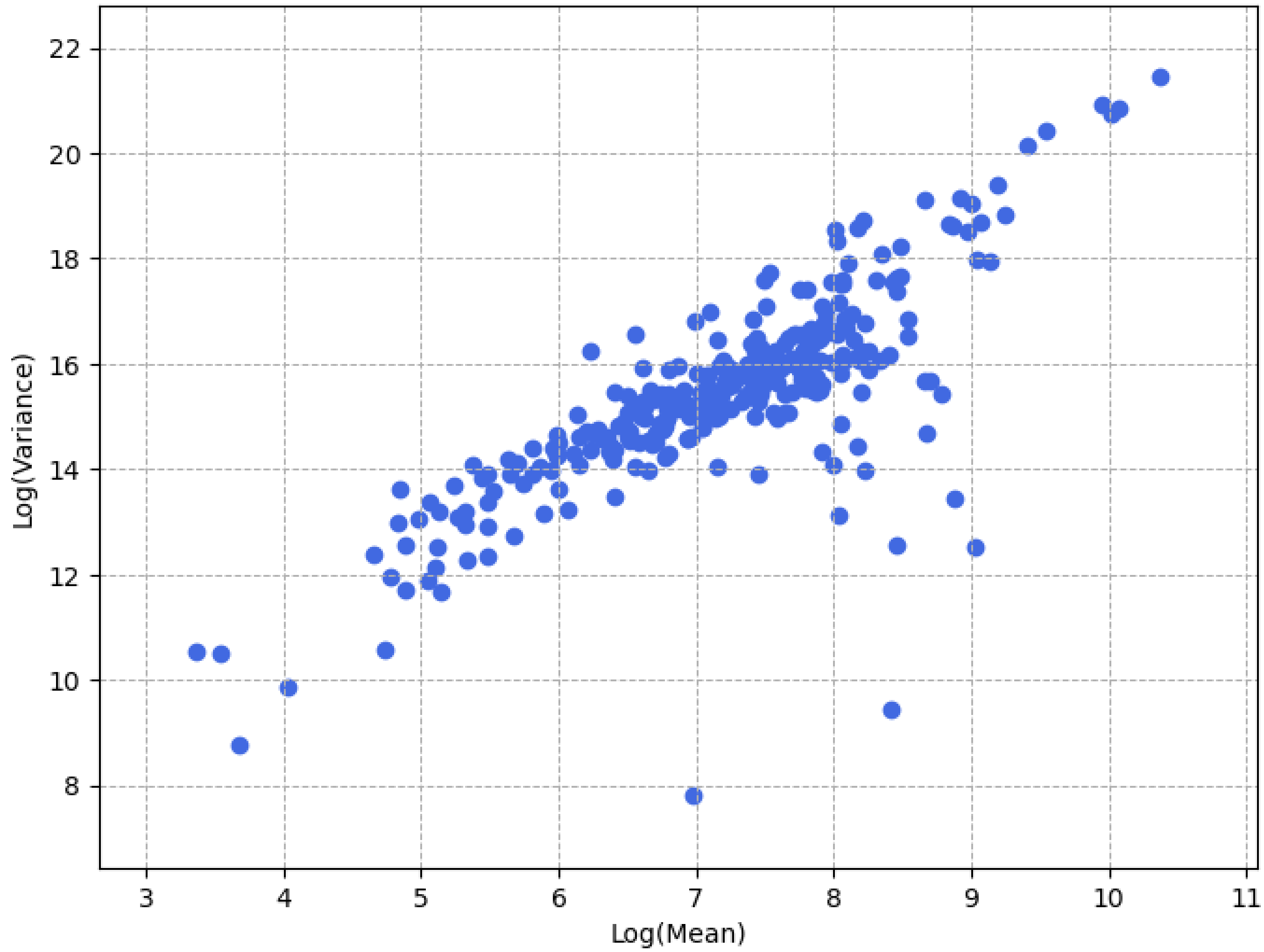


In [6]:
```
plt.figure(figsize = (10,6), dpi = 200)
plt.hist(y_train, bins = np.arange(0,13000,500), fill = True, color =
'lightyellow', edgecolor = 'black')
plt.title('Base 10 Logarithm Y-Scale')
```

```
plt.xlabel('Claim Amount (Dollar)')
plt.ylabel('Number of Observations')
plt.xticks(np.arange(0,14000,2000))
plt.yscale('log')
plt.grid(axis = 'y', linewidth = 0.7, linestyle = 'dashed')
plt.show()
```



In [7]:
```
# Estimate the Tweedie's P value
xtab = pd.pivot_table(train_data, values = yName, index = catName,
                      columns = None, aggfunc = ['count', 'mean', 'var'])
cell_stats = xtab[['mean','var']].reset_index().droplevel(1, axis = 1)
ln_Mean = np.where(cell_stats['mean'] > 1e-16, np.log(cell_stats['mean']),
np.NaN)
ln_Variance = np.where(cell_stats['var'] > 1e-16, np.log(cell_stats['var']),
np.NaN)
use_cell = np.logical_not(np.logical_or(np.isnan(ln_Mean),
np.isnan(ln_Variance)))
X_train = ln_Mean[use_cell]
y_train = ln_Variance[use_cell]
# Scatterplot of lnVariance vs lnMean
plt.figure(figsize = (8,6), dpi = 100)
plt.scatter(X_train, y_train, c = 'royalblue')
plt.xlabel('Log(Mean)')
plt.ylabel('Log(Variance)')
plt.margins(0.1)
plt.grid(axis = 'both', linewidth = 0.7, linestyle = 'dashed')
plt.show()
```



In [8]:
```
X_train = pd.DataFrame(X_train, columns = ['ln_Mean'])
X_train.insert(0, 'Intercept', 1.0)
y_train = pd.Series(y_train, name = 'ln_Variance')
result_list = Regression.LinearRegression(X_train, y_train)
```

In [9]:
```
tweediePower = result_list[0][1]
tweediePhi = np.exp(result_list[0][0])
```

In [40]:
```
print(f"Power parameter p : {tweediePower}")
print(f"Scale parameter phi : {tweediePhi}")
```

```
Power parameter p : 1.2840608802234919
Scale parameter phi : 495.39032035257253
```

In [12]:
```
# Begin Forward Selection
# The Deviance significance is the sixth element in each row of the test result
def takeDevSig(s):
    return s[7]
nPredictor = len(catName) + len(intName)
stepSummary = []
# Intercept only model
X0_train = train_data[[]]
X0_train.insert(0, 'Intercept', 1.0)
y_train = train_data[yName]
result_list = Regression.TweedieRegression (X0_train, y_train, o_train, tweedieP =
tweediePower)
qllk0 = result_list[3]
df0 = len(result_list[4])
phi0 = result_list[7]
stepSummary.append([0, 'Intercept', ' ', df0, qllk0, phi0, np.NaN, np.NaN,
np.NaN])
```

```
cName = catName.copy()
iName = intName.copy()
entryThreshold = 0.05
for step in range(nPredictor):
    enterName = ''
    stepDetail = []
    # Enter the next predictor
    for X_name in cName:
        X_train = pd.get_dummies(train_data[[X_name]].astype('category'))
        if (X0_train is not None):
            X_train = X0_train.join(X_train)
        result_list = Regression.TweedieRegression (X_train, y_train, o_train, tweedieP
        qllk1 = result_list[3]
        df1 = len(result_list[4])
        phi1 = result_list[7]
        devChiSq = 2.0 * (qllk1 - qllk0) / phi0
        devDF = df1 - df0
        devPValue = chi2.sf(devChiSq, devDF)
        stepDetail.append([X_name, 'categorical', df1, qllk1, phi1, devChiSq, devDF, dev
    for X_name in iName:
        X_train = train_data[[X_name]]
        if (X0_train is not None):
            X_train = X0_train.join(X_train)
        result_list = Regression.TweedieRegression (X_train, y_train, o_train, tweedieP
        qllk1 = result_list[3]
        df1 = len(result_list[4])
        phi1 = result_list[7]
        devChiSq = 2.0 * (qllk1 - qllk0) / phi0
        devDF = df1 - df0
        devPValue = chi2.sf(devChiSq, devDF)
        stepDetail.append([X_name, 'interval', df1, qllk1, phi1, devChiSq, devDF, devPVa
    # Find a predictor to add, if any
    stepDetail.sort(key = takeDevSig, reverse = False)
    minSig = takeDevSig(stepDetail[0])
    if (minSig <= entryThreshold):
        add_var = stepDetail[0][0]
        add_type = stepDetail[0][1]
        df0 = stepDetail[0][2]
        qllk0 = stepDetail[0][3]
        phi0 = stepDetail[0][4]
        stepSummary.append([step+1] + stepDetail[0])
        if (add_type == 'categorical'):
            X0_train = X0_train.join(pd.get_dummies(train_data[[add_var]].astype('catego
            cName.remove(add_var)
        else:
            X0_train = X0_train.join(train_data[[add_var]])
            iName.remove(add_var)
    else:
        break
# End of forward selection
stepSummary_df = pd.DataFrame(stepSummary, columns = ['Step','Predictor','Type','N Non-A
                                                      'Quasi Log-Likelihood', 'Phi',
                                                      'Deviance DF', 'Deviance Sig.'
# Retrain the final model
result_list = Regression.TweedieRegression (X0_train, y_train, o_train, tweedieP =
tweediePower)
y_pred_B = result_list[6]
```

b) (10 points). We will use the Forward Selection method to enter predictors into our model. Our entry threshold is 0.05. 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 Model Degree of Freedom (i.e., the number of non-aliased parameters), (4) the Quasi-Loglikelihood value, (5) the Deviance Chisquares statistic between the current and the previous models, (6) the corresponding Deviance Degree of Freedom, and (7) the corresponding Chi-square significance.

In [13]:
```
stepSummary_df[['Step','Predictor', 'N Non-Aliased Parameters',
                                          'Quasi Log-Likelihood', 'Devia
                                          'Deviance DF', 'Deviance Sig.'
```

Out[13]:

	Step	Predictor	N Non-Aliased Parameters	Quasi Log-Likelihood	Deviance ChiSquare	Deviance DF	Deviance Sig.
0	0	Intercept	1	-2.217255e+06	NaN	NaN	NaN
1	1	URBANICITY	2	-2.118974e+06	506.553405	1.0	3.565332e-112
2	2	EDUCATION	6	-2.057057e+06	333.869471	4.0	5.324835e-71
3	3	CAR_TYPE	11	-1.999000e+06	322.253774	5.0	1.639210e-67
4	4	PARENT1	12	-1.953492e+06	259.708324	1.0	1.986528e-58
5	5	MVR_PTS	13	-1.918088e+06	206.716848	1.0	7.147970e-47
6	6	TRAVTIME	14	-1.902663e+06	91.709156	1.0	1.003993e-21
7	7	CAR_USE	15	-1.888045e+06	87.600934	1.0	8.008682e-21
8	8	REVOKED	16	-1.873858e+06	85.661243	1.0	2.135578e-20
9	9	KIDSDRIV	17	-1.860341e+06	82.221123	1.0	1.216825e-19
10	10	TIF	18	-1.848416e+06	73.047518	1.0	1.265656e-17
11	11	INCOME	19	-1.836307e+06	74.641776	1.0	5.643625e-18
12	12	MSTATUS	20	-1.828104e+06	50.886436	1.0	9.786757e-13
13	13	CAR_AGE	21	-1.823990e+06	25.637908	1.0	4.118683e-07
14	14	YOJ	22	-1.820046e+06	24.622935	1.0	6.971704e-07
15	15	HOMEKIDS	23	-1.818925e+06	7.012163	1.0	8.095779e-03
16	16	GENDER	24	-1.818179e+06	4.670145	1.0	3.069134e-02
17	17	RED_CAR	25	-1.817282e+06	5.611384	1.0	1.784416e-02

c) (10 points). We will calculate the Root Mean Squared Error, the Relative Error, the Pearson correlation, and the Distance correlation between the observed and the predicted claim amounts of your final model. Please comment on their values.

In [14]:
```
# Simple Residual
y_simple_residual = y_train - y_pred_B
```

```python
# Mean Absolute Proportion Error
ape = np.abs(y_simple_residual) / y_train
mape = np.mean(ape)

# 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_train, ddof = 0)
print("Relative Error :", relerr)

# R-Squared
pearson_corr = Regression.PearsonCorrelation (y_train, y_pred_B)
spearman_corr = Regression.SpearmanCorrelation (y_train, y_pred_B)
kendall_tau = Regression.KendallTaub (y_train, y_pred_B)
distance_corr = Regression.DistanceCorrelation (y_train, y_pred_B)

print("Pearson Correlation :", pearson_corr)
print("Distance Correlation :", distance_corr)
```

```
RMSE : 4116.064009419275
Relative Error : 1.0078985075249884
Pearson Correlation : 0.18768098705727354
Distance Correlation : 0.27019055675583464
```

d) (10 points). 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. Please also provide us with the final estimate of the Tweedie distribution's scale parameter ϕ.

In [15]:
```python
result_list[0]
```

Out[15]:

	Estimate	Standard Error	Lower 95% CI	Upper 95% CI	Exponentiated
Intercept	8.004594	7.086575e-03	7.990705	8.018484	2994.685416
URBANICITY_Highly Rural/ Rural	-1.668163	2.775794e-03	-1.673603	-1.662722	0.188593
URBANICITY_Highly Urban/ Urban	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
EDUCATION_Bachelors	-0.140701	3.521999e-03	-0.147604	-0.133798	0.868749
EDUCATION_Below High Sc	0.201588	4.333039e-03	0.193096	0.210081	1.223344
EDUCATION_High School	0.096632	3.994529e-03	0.088803	0.104462	1.101455
EDUCATION_Masters	-0.138740	3.440745e-03	-0.145484	-0.131997	0.870454
EDUCATION_PhD	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
CAR_TYPE_Minivan	-0.750292	3.064477e-03	-0.756298	-0.744286	0.472229
CAR_TYPE_Panel Truck	0.017504	3.236289e-03	0.011161	0.023847	1.017659
CAR_TYPE_Pickup	-0.267047	2.918879e-03	-0.272768	-0.261326	0.765637
CAR_TYPE_SUV	0.052852	3.365622e-03	0.046255	0.059448	1.054273
CAR_TYPE_Sports Car	0.114225	3.757698e-03	0.106860	0.121590	1.121005
CAR_TYPE_Van	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
PARENT1_No	-0.487996	3.129989e-03	-0.494131	-0.481862	0.613855

PARENT1_Yes	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
MVR_PTS	0.096248	3.093628e-04	0.095641	0.096854	1.101032
TRAVTIME	0.011428	4.621827e-05	0.011337	0.011518	1.011493
CAR_USE_Commercial	0.504479	1.874834e-03	0.500804	0.508153	1.656122
CAR_USE_Private	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
REVOKED_No	-0.438032	1.964461e-03	-0.441882	-0.434182	0.645305
REVOKED_Yes	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
KIDSDRIV	0.268701	1.353898e-03	0.266048	0.271355	1.308264
TIF	-0.041796	1.885444e-04	-0.042165	-0.041426	0.959066
INCOME	-0.000006	2.375630e-08	-0.000006	-0.000006	0.999994
MSTATUS_No	0.451854	2.147406e-03	0.447645	0.456063	1.571222
MSTATUS_Yes	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
CAR_AGE	-0.023098	1.860426e-04	-0.023463	-0.022733	0.977167
YOJ	0.023394	2.143897e-04	0.022974	0.023814	1.023670
HOMEKIDS	0.057252	8.010420e-04	0.055682	0.058822	1.058922
GENDER_F	-0.194154	2.538526e-03	-0.199129	-0.189178	0.823531
GENDER_M	0.000000	-0.000000e+00	0.000000	0.000000	1.000000
RED_CAR_no	0.128131	2.137791e-03	0.123941	0.132321	1.136702
RED_CAR_yes	0.000000	-0.000000e+00	0.000000	0.000000	1.000000

In [16]:
```
stepSummary_df.tail(1)['Phi'].values[0]
```

Out[16]:
```
319.3817800403747
```

e) (10 points). Please generate a Two-way Lift chart for comparing your final model with the Intercept only model. Based on the chart, what will you conclude about your final model?

In [17]:
```
# Build a model with only the Intercept term
X_train = train_data[[yName]]
X_train.insert(0, 'Intercept', 1.0)
X_train = X_train.drop(columns = yName)

result = Regression.TweedieRegression (X_train, y_train, o_train, tweedieP = tweediePowe

y_pred_A = result[6]

y_pred_A.name = 'Model A'
y_pred_B.name = 'Model B'
```

In [18]:
```
train_data.shape
```

Out[18]:
```
(5715, 22)
```

In [19]:
```
# Normalize the model prediction
prediction = claim_history[[yName, eName]].join(y_pred_A).join(y_pred_B).dropna()
column_sums = np.sum(prediction[['EXPOSURE', 'CLM_AMT','Model A','Model B']], axis = 0)
adjP_CLM_AMT_A = prediction['Model A'] * (column_sums['CLM_AMT'] / column_sums['Model A'
adjP_CLM_AMT_B = prediction['Model B'] * (column_sums['CLM_AMT'] / column_sums['Model B'
print('Observed Sum = ', column_sums['CLM_AMT'])
```

```
print('Sum of Adjusted Model A = ', np.sum(adjP_CLM_AMT_A))
print('Sum of Adjusted Model B = ', np.sum(adjP_CLM_AMT_B))
```

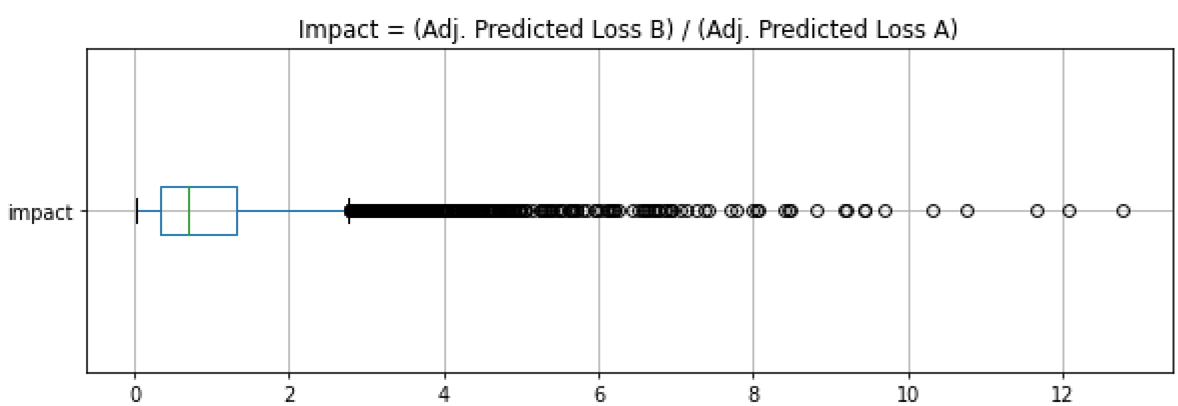
```
Observed Sum =  8669029.0
Sum of Adjusted Model A =  8669029.0
Sum of Adjusted Model B =  8669029.0
```

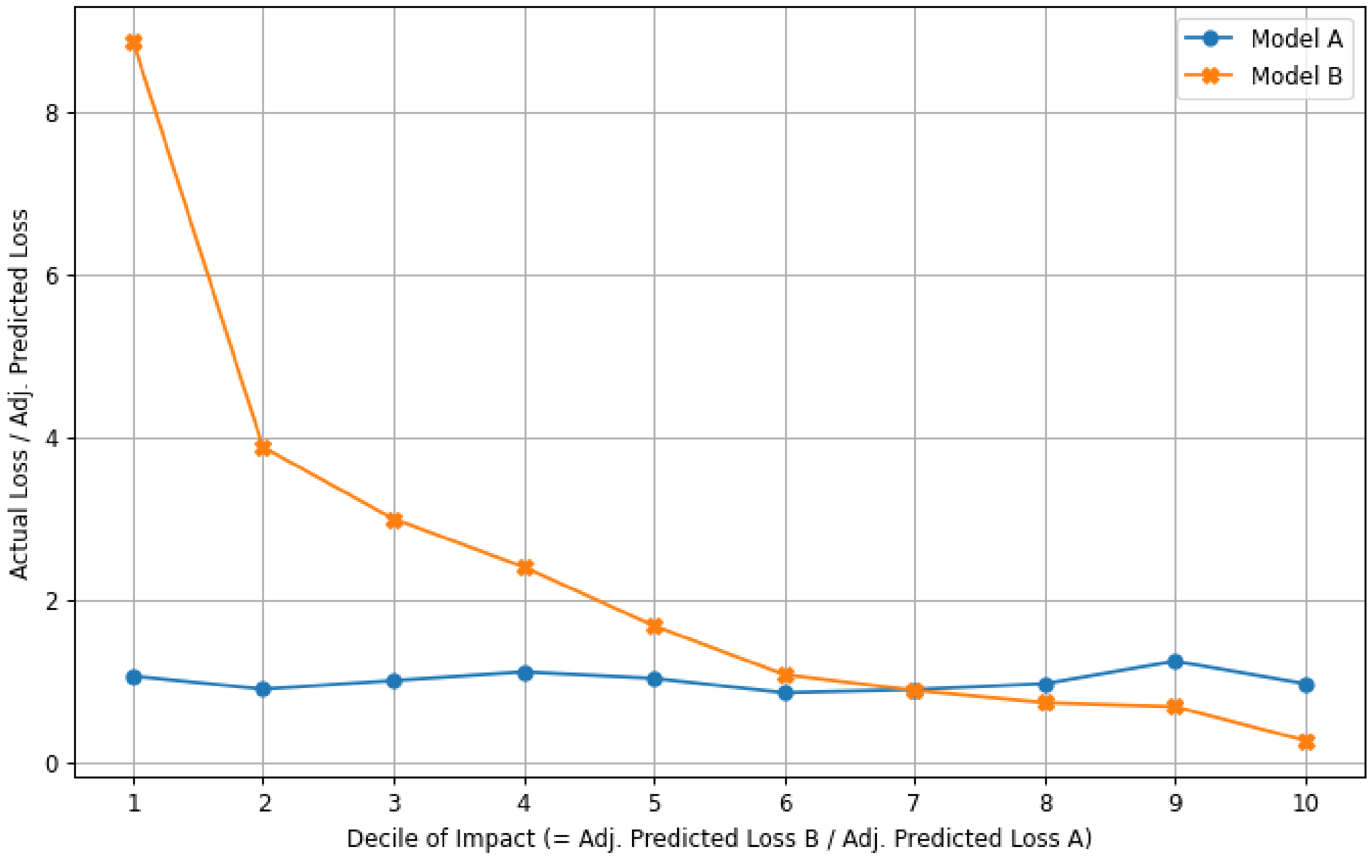
In [20]:

```
prediction = prediction.join(pd.DataFrame({'AdjModel A': adjP_CLM_AMT_A, 'AdjModel B': a
prediction['impact'] = adjP_CLM_AMT_B / adjP_CLM_AMT_A
```

In [42]:

```
plt.figure(figsize = (10,3), dpi = 70)
prediction.boxplot(column = ['impact'], vert = False)
plt.title('Impact = (Adj. Predicted Loss B) / (Adj. Predicted Loss A)')
plt.show()
prediction.sort_values(by = 'impact', axis = 0, ascending = True, inplace = True)
prediction['Cumulative Exposure'] = prediction['EXPOSURE'].cumsum()
cumulative_exposure_cutoff = np.arange(0.1, 1.1, 0.1) * column_sums['EXPOSURE']
decile = np.zeros_like(prediction['Cumulative Exposure'], dtype = int)
for i in range(10):
    decile = decile + np.where(prediction['Cumulative Exposure'] > cumulative_exposure_c

prediction['decile'] = decile + 1
xtab = pd.pivot_table(prediction, index = 'decile', columns = None,
                      values = ['EXPOSURE','CLM_AMT','AdjModel A', 'AdjModel B'],
                      aggfunc = ['sum'])
loss_ratio_A = xtab['sum','CLM_AMT'] / xtab['sum','AdjModel A']
loss_ratio_B = xtab['sum','CLM_AMT'] / xtab['sum','AdjModel B']
MAE_A = np.mean(np.abs((loss_ratio_A - 1.0)))
MAE_B = np.mean(np.abs((loss_ratio_B - 1.0)))

plt.figure(figsize = (10,6), dpi = 85)
plt.plot(xtab.index, loss_ratio_A, marker = 'o', label = 'Model A')
plt.plot(xtab.index, loss_ratio_B, marker = 'X', label = 'Model B')
plt.xlabel('Decile of Impact (= Adj. Predicted Loss B / Adj. Predicted Loss A)')
plt.ylabel('Actual Loss / Adj. Predicted Loss')
plt.xticks(range(1,11))
plt.grid()
plt.legend()
plt.show()
```





Question 2

In [24]:
```
myeloma = pd.read_csv("myeloma.csv")
myeloma.head()
```

Out[24]:

	Time	VStatus	LogBUN	HGB	Platelet	Age	LogWBC	Frac	LogPBM	Protein	SCalc
0	1.25	1	2.2175	9.4	1	67	3.6628	1	1.9542	12	10
1	1.25	1	1.9395	12.0	1	38	3.9868	1	1.9542	20	18
2	2.00	1	1.5185	9.8	1	81	3.8751	1	2.0000	2	15
3	2.00	1	1.7482	11.3	0	75	3.8062	1	1.2553	0	12
4	2.00	1	1.3010	5.1	0	57	3.7243	1	2.0000	3	9

In [25]:
```
myeloma.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Time      65 non-null     float64
 1   VStatus   65 non-null     int64  
 2   LogBUN    65 non-null     float64
 3   HGB       65 non-null     float64
 4   Platelet  65 non-null     int64  
 5   Age       65 non-null     int64  
 6   LogWBC    65 non-null     float64
 7   Frac      65 non-null     int64  
 8   LogPBM    65 non-null     float64
 9   Protein   65 non-null     int64  
 10  SCalc     65 non-null     int64  
```

```
dtypes: float64(5), int64(6)
memory usage: 5.7 KB
```

In [26]:
```
myeloma.VStatus.value_counts()
```

Out[26]:
```
1    48
0    17
Name: VStatus, dtype: int64
```

b) (10 points). We will use the Kaplan-Meier Product Limit Estimator to create the life table. Please provide us with the life table.

In [31]:
```
# Calculate the Kaplan-Meier Product Limit Estimator for the Survival Function
xtab = pd.crosstab(index = myeloma['Time'], columns = myeloma['VStatus'])
lifeTable = pd.DataFrame({'Survival Time': 0, 'Number Left': nUnit, 'Number of Events':
lifeTable = pd.concat([lifeTable, pd.DataFrame({'Survival Time':
xtab.index, 'Number of Events': xtab[1].to_numpy(),
                                                        'Number Censored':
xtab[0].to_numpy()})],
                          axis = 0, ignore_index = True)
lifeTable[['Number at Risk']] = nUnit
nTime = lifeTable.shape[0]
probSurvival = 1.0
hazardFunction = 0.0
seProbSurvival = 0.0
lifeTable.at[0,'Prob Survival'] = probSurvival
lifeTable.at[0,'Prob Failure'] = 1.0 - probSurvival
lifeTable.at[0,'Cumulative Hazard'] = hazardFunction
for i in np.arange(1,nTime):
    nDeath = lifeTable.at[i,'Number of Events']
    nAtRisk = lifeTable.at[i-1,'Number Left'] - lifeTable.at[i-1,'Number Censored']
    nLeft = nAtRisk - nDeath
    probSurvival = probSurvival * (nLeft / nAtRisk)
    seProbSurvival = seProbSurvival + nDeath / nAtRisk / nLeft
    hazardFunction = hazardFunction + (nDeath / nAtRisk)
    lifeTable.at[i, 'SE Prob Survival'] = seProbSurvival
    lifeTable.at[i,'Number Left'] = nLeft
    lifeTable.at[i,'Number at Risk'] = nAtRisk
    lifeTable.at[i,'Prob Survival'] = probSurvival
    lifeTable.at[i,'Prob Failure'] = 1.0 - probSurvival
    lifeTable.at[i,'Cumulative Hazard'] = hazardFunction
lifeTable['SE Prob Survival'] = lifeTable['Prob Survival'] * np.sqrt(lifeTable['SE Prob
CIHalfWidth = norm.ppf(0.975) * lifeTable['SE Prob Survival']
u = lifeTable['Prob Survival'] - CIHalfWidth
lifeTable['Lower CI Prob Survival'] = np.where(u < 0.0, 0.0, u)
u = lifeTable['Prob Survival'] + CIHalfWidth
lifeTable['Upper CI Prob Survival'] = np.where(u > 1.0, 1.0, u)
```

In [32]:
```
lifeTable
```

Out[32]:

	Survival Time	Number Left	Number of Events	Number Censored	Number at Risk	Prob Survival	Prob Failure	Cumulative Hazard	SE Prob Survival	Lower CI Prob Survival
0	0.00	65.0	0	0	65	1.000000	0.000000	0.000000	NaN	NaN
1	1.25	63.0	2	0	65	0.969231	0.030769	0.030769	0.021420	0.927249
2	2.00	60.0	3	0	63	0.923077	0.076923	0.078388	0.033051	0.858297
3	3.00	59.0	1	0	60	0.907692	0.092308	0.095055	0.035903	0.837324
4	4.00	59.0	0	2	59	0.907692	0.092308	0.095055	0.035903	0.837324
5	5.00	55.0	2	0	57	0.875843	0.124157	0.130143	0.041104	0.795281
6	6.00	51.0	4	0	55	0.812146	0.187854	0.202870	0.048921	0.716262

7	7.00	48.0	3	2	51	0.764372	0.235628	0.261693	0.053254	0.659996
8	8.00	46.0	0	1	46	0.764372	0.235628	0.261693	0.053254	0.659996
9	9.00	44.0	1	0	45	0.747386	0.252614	0.283916	0.054713	0.640151
10	11.00	39.0	5	1	44	0.662456	0.337544	0.397552	0.060254	0.544361
11	12.00	38.0	0	2	38	0.662456	0.337544	0.397552	0.060254	0.544361
12	13.00	35.0	1	1	36	0.644055	0.355945	0.425330	0.061326	0.523859
13	14.00	33.0	1	0	34	0.625112	0.374888	0.454742	0.062379	0.502851
14	15.00	32.0	1	0	33	0.606169	0.393831	0.485045	0.063300	0.482104
15	16.00	30.0	2	1	32	0.568283	0.431717	0.547545	0.064764	0.441347
16	17.00	27.0	2	0	29	0.529091	0.470909	0.616510	0.065961	0.399810
17	18.00	26.0	1	0	27	0.509496	0.490504	0.653547	0.066365	0.379422
18	19.00	24.0	2	2	26	0.470304	0.529696	0.730470	0.066796	0.339385
19	24.00	21.0	1	0	22	0.448926	0.551074	0.775925	0.067094	0.317425
20	25.00	20.0	1	0	21	0.427549	0.572451	0.823544	0.067218	0.295803
21	26.00	19.0	1	0	20	0.406171	0.593829	0.873544	0.067171	0.274519
22	28.00	19.0	0	1	19	0.406171	0.593829	0.873544	0.067171	0.274519
23	32.00	17.0	1	0	18	0.383606	0.616394	0.929099	0.067122	0.252049
24	35.00	16.0	1	0	17	0.361041	0.638959	0.987923	0.066859	0.229999
25	37.00	15.0	1	0	16	0.338476	0.661524	1.050423	0.066379	0.208375
26	41.00	13.0	2	1	15	0.293346	0.706654	1.183756	0.064747	0.166445
27	51.00	11.0	1	0	12	0.268900	0.731100	1.267090	0.063799	0.143856
28	52.00	10.0	1	0	11	0.244455	0.755545	1.357999	0.062507	0.121943
29	53.00	10.0	0	1	10	0.244455	0.755545	1.357999	0.062507	0.121943
30	54.00	8.0	1	0	9	0.217293	0.782707	1.469110	0.061180	0.097384
31	57.00	8.0	0	1	8	0.217293	0.782707	1.469110	0.061180	0.097384
32	58.00	6.0	1	0	7	0.186251	0.813749	1.611967	0.059798	0.069049
33	66.00	5.0	1	0	6	0.155209	0.844791	1.778634	0.057326	0.042853
34	67.00	4.0	1	0	5	0.124168	0.875832	1.978634	0.053610	0.019093
35	77.00	4.0	0	1	4	0.124168	0.875832	1.978634	0.053610	0.019093
36	88.00	2.0	1	0	3	0.082778	0.917222	2.311967	0.049187	0.000000
37	89.00	1.0	1	0	2	0.041389	0.958611	2.811967	0.038228	0.000000
38	92.00	0.0	1	0	1	0.000000	1.000000	3.811967	NaN	NaN

a) (10 points). How many risk sets are there?

In [37]:
```
lifeTable['Number at Risk'].nunique()
```

Out[37]:
```
38
```

c) (10 points). According to the life table, what is the Probability of Survival and the Cumulative Hazard at a survival time of 18 months? What do these two values mean to a layperson?

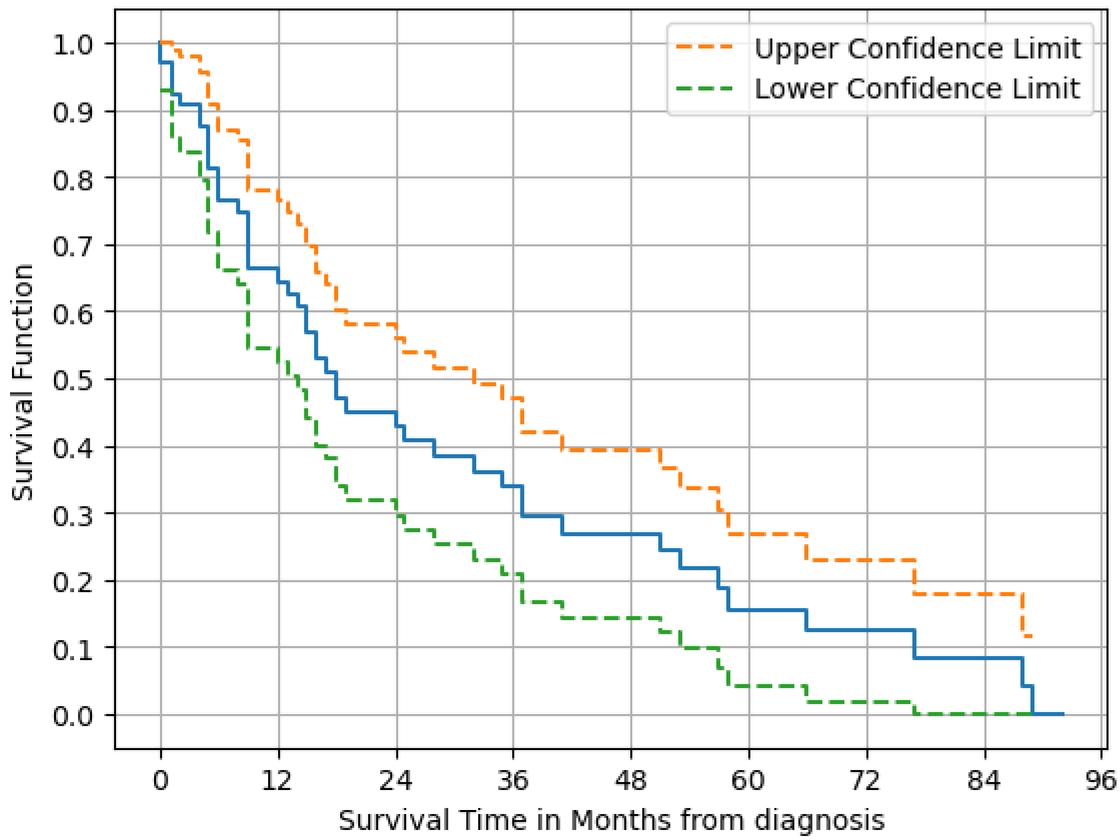
In [33]:
```
lifeTable.loc[lifeTable['Survival Time'] == 18.00][['Survival Time', 'Prob Survival', 'C
```

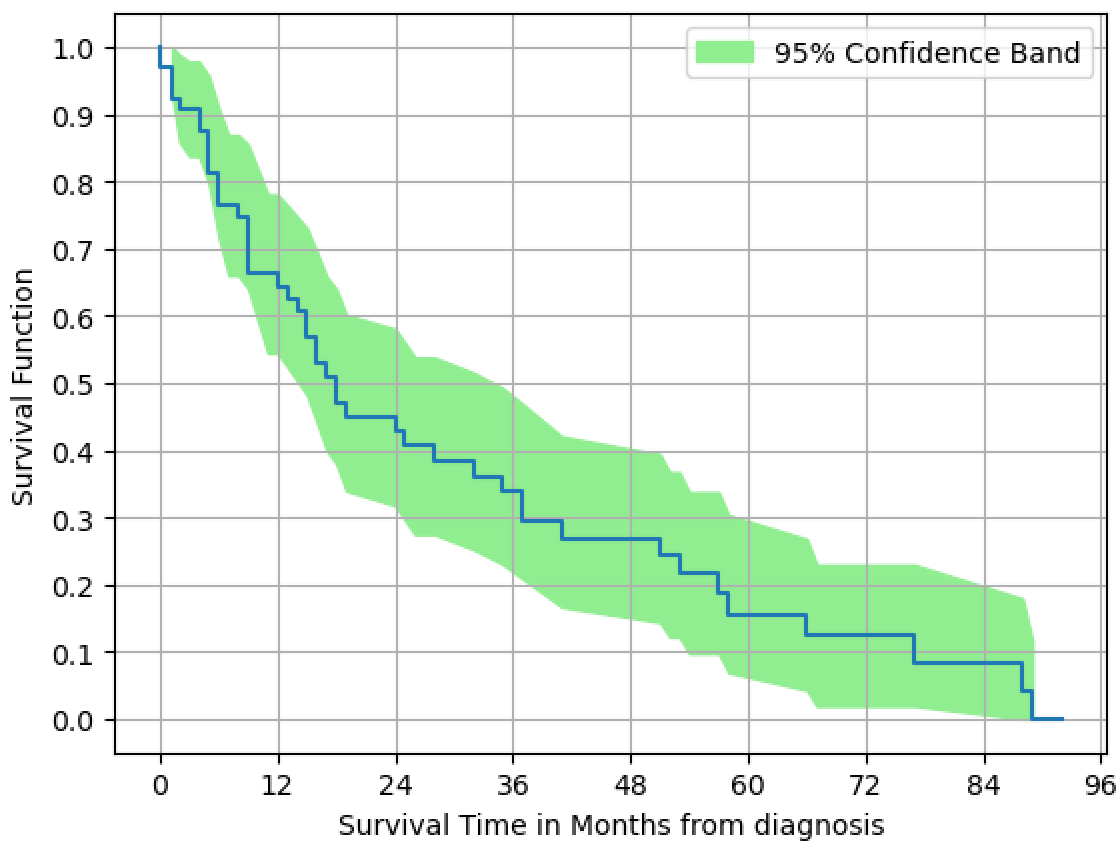
Out[33]:

	Survival Time	Prob Survival	Cumulative Hazard
17	18.0	0.509496	0.653547

d) (10 points). Please generate the Survival Function graph using the Kaplan-Meier Product Limit Estimator life table. Since we measure the Time variable in the number of months, we will specify the x-axis ticks from 0 with an increment of 12. Besides plotting the Survival Function versus Time, you must also add the 95% Confidence Band. You might use the matplotlib fill_between() function to generate the Confidence Band as a band around the Survival Function. To receive the full credits, you must label the chart elements properly

In [34]:
```
plt.figure(dpi = 100)
plt.plot(lifeTable['Survival Time'], lifeTable['Prob Survival'], drawstyle =
'steps')
plt.plot(lifeTable['Survival Time'], lifeTable['Upper CI Prob Survival'], drawstyle
= 'steps',
         linestyle = 'dashed', label = 'Upper Confidence Limit')
plt.plot(lifeTable['Survival Time'], lifeTable['Lower CI Prob Survival'], drawstyle
= 'steps',
         linestyle = 'dashed', label = 'Lower Confidence Limit')
plt.xlabel('Survival Time in Months from diagnosis')
plt.ylabel('Survival Function')
plt.xticks(np.arange(0,100,12))
plt.yticks(np.arange(0.0,1.1,0.1))
plt.grid(axis = 'both')
plt.legend()
plt.show()
# Plot the Survival Function with a Confidence Band
plt.plot(lifeTable['Survival Time'], lifeTable['Prob Survival'], drawstyle =
'steps')
plt.fill_between(lifeTable['Survival Time'], lifeTable['Lower CI Prob Survival'],
lifeTable['Upper CI Prob Survival'],
                 color = 'lightgreen', label = '95% Confidence Band')
plt.xlabel('Survival Time in Months from diagnosis')
plt.ylabel('Survival Function')
plt.xticks(np.arange(0,100,12))
plt.yticks(np.arange(0.0,1.1,0.1))
plt.grid(axis = 'both')
plt.legend()
plt.show()
```





e) (10 points). Use Linear Interpolation to determine the Median Survival Time (in number of months) from the Kaplan-Meier Product Limit Estimator life table. Please round your answer up to the tenths place.

$$t_M = t_s + \frac{\hat{S}(t_s|\mathbf{x}_i) - 0.5}{\hat{S}(t_s|\mathbf{x}_i) - \hat{S}(t_r|\mathbf{x}_i)}(t_r - t_s)$$

In [43]:

```
time_months = lifeTable['Survival Time'].values
survival_probs = lifeTable['Prob Survival'].values


ind_r = np.argmax(survival_probs < 0.5)
ind_s = ind_r-1

ts = time_months[ind_s]
tr = time_months[ind_r]

ps = survival_probs[ind_s]
pr = survival_probs[ind_r]

median_time = ts + ((ps - 0.5) / (ps - pr)) * (tr - ts)

# round median_time to the tenths place
median_time_rounded = round(median_time, 1)

print(f"Median Survival Time (in number of months) is {median_time_rounded} months.")
```

```
Median Survival Time (in number of months) is 18.2 months.
```

In [ ]: