

**ANL252**

**Python for Data Analytics**

# **ECA**

**12 September 2021 Submission**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Elijah Song Ming Fu** | **K2070438** |

Q1a/b/d:

|  |
| --- |
| # Libraries and Utilities  import pandas as pd  import numpy as np  import sqlite3  from sqlite3 import Error  # Reading ship.csv as pandas dataframe ship  ship = pd.read\_csv('ship.csv')  # applying to\_numeric function which identifies character "."and converts to missing value NaN  ship['MS'] = pd.to\_numeric(ship['MS'], errors='coerce')  ship['Y'] = pd.to\_numeric(ship['Y'], errors='coerce')  # Renaming Columns  ship = ship.rename(columns={'T': 'types', 'A': 'c\_years', 'P': 'o\_periods', 'MS': 's\_months', 'Y': 'incidents'})  # caluclating averages of columns service months and incidents  avg\_s\_months = round(ship['s\_months'].mean())  avg\_incidents = round(ship['incidents'].mean())  # filling missing values  ship['s\_months'] = ship['s\_months'].fillna(float(avg\_s\_months))  ship['incidents'] = ship['incidents'].fillna(float(avg\_incidents))  # save target variable incidents in Y  X = ship.iloc[:, :-1]  Y = ship.iloc[:, -1]  # 'types', c\_years, o\_periods as categorical variables  ship['types'] = ship['types'].astype('category')  ship['c\_years'] = ship['c\_years'].astype('category')  ship['o\_periods'] = ship['o\_periods'].astype('category')  # categorical varables to dummy values  ship['types'] = ship['types'].cat.codes  ship['c\_years'] = ship['c\_years'].cat.codes  ship['o\_periods'] = ship['o\_periods'].cat.codes  # log transformation of s\_months to log\_s\_months  ship['log\_s\_months'] = np.log(ship['s\_months'])  X = ship  X = X.drop(columns=['incidents'])  y = ship['incidents']  ship.to\_csv('ship\_prepared.csv')  ship.head()  # Making a database and connecting to it  DB\_FILE\_PATH = 'ship.db'  def connect\_to\_db(db\_file):  sqlite3\_conn = None  try:  sqlite3\_conn = sqlite3.connect(db\_file)  return sqlite3\_conn  except Error as err:  print(err)  if sqlite3\_conn is not None:  sqlite3\_conn.close()  # Making database table and entering data from dataframe to db  def insert\_values\_to\_table(table\_name):  conn = connect\_to\_db(DB\_FILE\_PATH)  if conn is not None:  c = conn.cursor()  # Create table if it is not exist  c.execute('CREATE TABLE IF NOT EXISTS ' + table\_name +  '( INTEGER,'  'types INTEGER,'  'c\_years INTEGER,'  'o\_periods INTEGER,'  's\_months DECIMAL,'  'incidents DECIMAL,'  'log\_s\_months DECIMAL)')  ship.to\_sql(name=table\_name, con=conn, if\_exists='append', index=False)  conn.close()  print('SQL insert process finished')  else:  print('Connection to database failed')  # Calling Function  insert\_values\_to\_table('imdb\_temp') |

Q1c:

Because the dataset is so small, splitting the Data Frame is not a good idea. Splitting the dataset into training and testing will result in the model being unable to be trained due to a lack of data features. The more data your deployed model has seen, the better it should generalize, theoretically. As a result, if the model is trained on the whole set of data available, it should generalize better than a model that is just trained on a subset of the data.

Second, there is a significant quantity of information that is lacking. There are 6 records with no target variables, accounting for 15% of the total dataset. If the data is further divided into training and test, the model will become less comprehensive and accurate. As a result, in this case, utilizing the whole dataset for training purposes may be more sensible.

Q2a:

Scikit-learn is a simple and efficient open source and commercially accessible module for predictive data analysis. The module is free source and helpful in a variety of situations because it is built on NumPy, SciPy, and Matplotlib. An estimator is anything that learns from data; it might be a regression, classification, or clustering method, or a transformer that extracts/filters important data. The fit function will attempt to fit the model to the input variables' training instances, whereas the predict function will make predictions on the input and target variables' testing instances based on the parameters learned during fit. Finally, the parameters are found in the model's inputs for the data characteristics.

Q2b/c:

|  |
| --- |
| # DataFrames X and Y generated in Question 1.  # converting them to numpy array  import numpy as np  X=X.drop(columns=['types','c\_years','o\_periods','s\_months'])  X\_=X.to\_numpy()  Y\_=Y.to\_numpy()  from sklearn import linear\_model  model = linear\_model.PoissonRegressor()  model.fit(X\_, Y\_)  from collections import defaultdict  table = defaultdict(float)  table['β0'] = model.intercept\_  i = 1  for val in model.coef\_:  table['β' + str(i)] = val  i = i + 1  print(table)  Y\_pred = model.predict(X)  D = []  # applying given formula for deviance  for y in Y\_pred:  if (y == 0):  D.append(-(y - np.exp(np.mean(Y\_))))  else:  D.append(y \* np.log(y / np.exp(np.mean(Y\_))) - (y - np.exp(np.mean(Y\_))))  print("Deviance is ", 2 \* np.mean(D)) |