ANL252

ECA

Z1970078

Muhammad Syairazi Bin Mahdhar

13 Sep 2021

**Question 1(a)(i)**

# Importing packages.

# scikit-learn is not imported yet, as per instructions: "using functions and methods of NumPy and Pandas, respectively."

# sqlite3 is imported for use in Q1(d).

import pandas as pd

import numpy as np

import sqlite3

# Reading "ship.csv", and assign to DataFrame "ship". Missing values "." are declared as "NaN".

# Note: The rounding of decimals is reserved until the end at Q2(c), to preserve accuracy.

ship = pd.read\_csv("ship.csv", na\_values = ".")

print('The DataFrame "ship" is:')

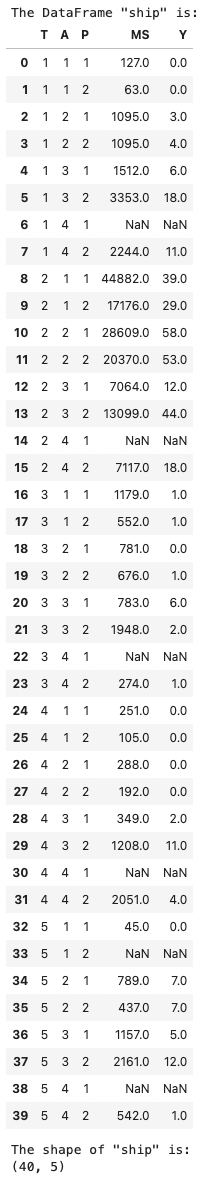
display(ship)

print('The shape of "ship" is:')

display(ship.shape)

***Output on next page.***

**Output:**



**Question 1(a)(ii)**

# Renaming 5 variables in the columns. Set up a Dictionary with old and new variables, to use in .rename method.

rename\_dict = {"T": "types", "A": "c\_years", "P": "o\_periods", "MS": "s\_months", "Y": "incidents"}

ship.rename(columns = rename\_dict, inplace = True)

# Display both .head() and .shape of "ship".

print('The renamed columns of DataFrame "ship" are as follows:')

display(ship.head())

print('The shape of "ship" is:')

display(ship.shape)

**Output:**

**Table

Description automatically generated**

**Question 1(a)(iii)**

# Group "types" and "o\_periods" into Row Labels such that the cross-product between the two are clear:

# i.e. "type" = 1 and "o\_periods" = 1, "type" = 1 and "o\_periods" = 2, ... , "type" = 5 and "o\_periods" = 2.

# Take the mean of the columns "s\_months" and "incidents" respectively, and round it off to the nearest integers.

# Assign to object "shipgroup".

shipgroup = ship[["s\_months", "incidents", "types", "o\_periods"]].groupby(by = ["types", "o\_periods"]).mean().round()

# Converting the values in "s\_months" and "incidents" into "int" type.

shipgroup = shipgroup.astype({"s\_months": "int", "incidents": "int"})

# Displaying the new "shipgroup" DataFrame, and its info, including data types.

print('The DataFrame "shipgroup" is:')

display(shipgroup)

print('The info for DataFrame "shipgroup" is:')

shipgroup.info()

**Output:**

Table

Description automatically generated

**Question 1(a)(iv)**

# In this question, I chose not to replace the "NaN" values manually.

# Such a technique is not ideal, especially if the DataFrame was larger, with thousands of "NaN" values.

# Instead, I employed a For Loop to run through each row and replace the "NaN" values automatically.

# Sort "ship" by labels "types" and "o\_periods".

# Re-index "ship" so that "types" and "o\_periods" become Row Indexes.

# Assign to DataFrame "ship\_newindex".

ship\_newindex = ship.sort\_values(by = ["types", "o\_periods"], ascending = [True, True]).set\_index(["types", "o\_periods"])

# First create an empty DataFrame and assign to "ship2".

# Then employ a nested For Loop as follows:

# Variable "i" represents the "types" label. Variable "j" represents the "o\_periods" label.

# DataFrame "ship\_newindex" is accessed via .loc[(i, j)].

# The "NaN" values in labels "s\_months" and "incidents" of "ship\_newindex" are filled by the mean values in labels "s\_months" and "incidents" of "shipgroup".

# The changes are appended and assigned back to DataFrame "ship2", iterated through the ranges of "i" and "j".

ship2 = pd.DataFrame()

for i in range(1, 6):

for j in range(1, 3,):

ship2 = ship2.append(ship\_newindex.loc[[(i, j)]].fillna({"s\_months": shipgroup.loc[(i, j), "s\_months"], "incidents": shipgroup.loc[(i, j), "incidents"]}))

# Converting values in "s\_months" and "incidents" in "ship2" to "int" type.

# Sort the column labels of "ship" back according to "types" and "c\_years", like the original "ship".

# Reset index of DataFrame "ship" back to the default row integers.

# Assign all changes back to "ship".

ship = ship2.astype({"s\_months": "int", "incidents": "int"}).sort\_values(by = ["types", "c\_years"], ascending = [True, True]).reset\_index()

# Display "ship" to check the concatenated DataFrame with all NaN values replaced, and also the data types.

print('DataFrame "ship" has been updated to:')

display(ship)

print('The data types for the columns in "ship" are:')

display(ship.dtypes)

**Output:**

**A picture containing text, window

Description automatically generated**

**A picture containing text

Description automatically generated**

**Question 1(a)(v)**

# Saving variable "incidents" from "ship" to a DataFrame named "Y".

Y = ship[["incidents"]]

print('The head of DataFrame "Y" is:')

display(Y.head())

print('The shape of "Y" is:')

display(Y.shape)

**Output:**

**Table

Description automatically generated**

**Question 1(b)(i)**

# Using .astype method to convert "types", "c\_years" and "o\_periods" to "category" type, and assign back to "ship".

ship = ship.astype({"types": "category", "c\_years": "category", "o\_periods": "category"})

# Showing the types of each variable after the data type conversion.

print('The data types of the columns in "ship" are updated to:')

display(ship.dtypes)

**Output:**

**Text

Description automatically generated**

**Question 1(b)(ii)**

# Converting the categorical variables "types", "o\_periods" and "c\_years" into Dummy Variables.

ship\_bii1 = pd.get\_dummies(ship[["types", "o\_periods", "c\_years"]])

# Dropping the labels "types", "o\_periods" and "c\_years".

ship\_bii2 = ship.drop(["types", "o\_periods", "c\_years"], axis = 1)

# Merging "ship\_bii1" and "ship\_bii2" and assign to "X".

X = pd.concat([ship\_bii1, ship\_bii2], axis = 1)

# NOTE on X: As per instructions, DataFrame "X" is shown. The Dependent Variable "incidents" is still included among the other Independent Variables.

# In Q2 later, "incidents" shall be dropped from Dataframe "X" when performing the Poisson Regression.

print('The DataFrame "X" is:')

X

***Output on next page.***

**Output:**

**Table

Description automatically generated**

**Question 1(b)(iii)**

# log transformation is done using NumPy methods (not scikit-learn), as instructed for Q1.

# Applying log transformation to "s\_months" variable from "ship". Assign to variable "log\_s\_m".

# Note: "log\_s\_m" decimal places are not rounded off, to increase accuracy for Q2 later.

log\_s\_m = np.log(ship["s\_months"])

# A new column with label "log\_s\_months" in "ship" is created, and assigned values from "log\_s\_m".

ship["log\_s\_months"] = log\_s\_m

print('DataFrame "ship" is updated to:')

display(ship.head())

# A new variable "log\_s\_months" in "X" is created, also assigned values from "log\_s\_m".

X["log\_s\_months"] = log\_s\_m

# Drop column label "s\_months", as "log\_s\_months" is its replacement for this question.

X = X.drop("s\_months", axis = 1)

print('DataFrame "X" is updated to:')

display(X.head())

**Output:**

**Table

Description automatically generated**

**Question 1(c )**

Our dataset consists of 40 rows, or ‘data points’. If we split the dataset into Training Sets and Testing Sets in the split ratio of 70%–30%, we will only have 28 data points to train the model on. We will also have to test our model’s predictive ability with just 12 data points. The model will use both Training and Test Sets synergistically – the Training Set is for the model to learn, and the Test Set is for the model to validate its predictions. [Geron, 2017]

However, with a low amount of Training data points, the model is at risk of performing an overfit or underfit [Volpi, 2019]. Coupled with an even smaller Test set, the model will not have a sufficient amount of data points against which it can validate its predictions reliably. This presents a problem in creating a robust model on which to base our predictions. [Brownlee, 2020]. It will then lead to prediction errors that can cause many problems in the future.

**Question 1(d)**

# Part 1: Saving the DataFrame "ship" to the .csv file "ship\_prepared.csv"

# index is set to False, so as not to create an additional column in the .csv file.

ship.to\_csv("ship\_prepared.csv", index=False)

# Checking the "ship\_prepared.csv" file by reading it into variable "ship\_prep". Then display.

ship\_prep = pd.read\_csv("ship\_prepared.csv")

print('For checking purposes, the head of DataFrame "ship\_prepared" is:')

display(ship\_prep.head())

display(ship.shape)

**Output 1:**

**Table

Description automatically generated**

# Part 2: Saving the DataFrame "ship" to the database "ship.db", which is created upon definition.

# Let "conn" be the Variable for the Connection Object.

# Let "cur" be the Variable for the Cursor Object.

# Using .to\_sql to send DataFrame "ship" to the database "ship.db". "index" parameter is False so an additional column isn't created.

conn = sqlite3.connect("ship.db")

cur = conn.cursor()

# Exporting the DataFrame "ship" to database "ship.db" as a table called "ship\_tbl".

ship.to\_sql("ship\_tbl", conn, if\_exists = "replace", index=False)

# Checking "ship.db" by using .fetchone(), and comparing against the first row of "ship".

cur.execute("SELECT \* FROM ship\_tbl;")

print(cur.fetchone())

**Output 2:**

****

**Question 2(a)**

The **module** chosen is linear\_model. In essence, linear models assume that the data is mostly linear, and that the distance between the data points and the straight line can be ignored [Geron, 2017]. More specifically, we narrow our selection to a generalized type of linear model, where the independent variables *X*1, *X*2, …, *X*n are linear combinations of each other. This linear combination expresses a function of E(*Y*) – the predicted values of *y* – and the inverse of the function exists.

In our generalized linear model, since the *X* variables adopt a Poisson distribution, hence

the **estimator** chosen is PoissonRegressor.

The fit function makes the PoissonRegressor estimator train on the data (the independent *X* variables). This function also calculates all the coefficients for the *X* variables within the linear combination equation. This training is akin to a “learning” process for the estimator. The parameters of the fit function are *X* and *y*. *X* consists of an array of shape (*n*, *n*) that contains the independent variables. *y* consists of an 1-dimensional array of shape (*n*,) that contain the dependent variable*.*

The predict function makes the PoissonRegressor estimator calculate the E(*Y*) values, also known as the “expected” values of *y*. These are essentially the values of *y* that the estimator is predicting, after having trained on the set of independent variables *X* earlier. The parameters of predict is just *X*, the array of independent variables of shape (*n*, *n*).

**Question 2(b)**

# Numpy and Pandas have already been imported for Question 1.

# Importing the other required packages.

import math

import matplotlib.pyplot as plt

from sklearn.linear\_model import PoissonRegressor

# Drop label "incidents" from Dataframe "X", as it is a Dependent Variable used in "Y".

X = X.drop("incidents", axis = 1)

print('The DataFrame "X" is updated for Regression as follows:')

display(X.head())

display(X.shape)

**Output 1:**

**Table

Description automatically generated with medium confidence**

# Extracting "incidents" values from DataFrame "Y", and assign to "y".

y = np.array(Y["incidents"])

# Show "y" and its shape.

print('The Array "y" is:')

display(y)

print('The shape of "y" is:')

display(y.shape)

**Output 2:**

**A picture containing chart

Description automatically generated**

# The Poisson Regression Model, with reported parameters as shown.

# Assign model to "pr".

pr = PoissonRegressor(alpha = 1, fit\_intercept = True, max\_iter = 100, tol = 1e-4, warm\_start = False, verbose = 0)

# Train the Model and print the output.

print(pr.fit(X, y))

**Output 3:**

****

# Finding intercept and coefficient values, then append them to an Array. Assign to "new\_arr".

coeff\_arr = np.append(pr.intercept\_, pr.coef\_).reshape((1, 13))

# Create column labels for all 13 coefficients. Assign to "col".

col = ['Beta\_0', 'Beta\_1', 'Beta\_2', 'Beta\_3', 'Beta\_4', 'Beta\_5', 'Beta\_6', 'Beta\_7', 'Beta\_8', 'Beta\_9', 'Beta\_10', 'Beta\_11', 'Beta\_12']

# Create a DataFrame using the Array "new\_arr" with "col" as column labels. Assign to "coeff".

coeff = pd.DataFrame(coeff\_arr, columns = col)

print('The DataFrame "coeff" contains the coefficients. It is:')

display(coeff)

print('The shape of "coeff" is:')

display(coeff.shape)

**Output 4:**

**Table

Description automatically generated with low confidence**

**Question 2(c )**

# Create Array for values of E(Y). Assign to "E\_Y"

E\_Y = pr.predict(X.iloc[:])

print('The Array "E\_Y" is:')

display(E\_Y)

print('The shape for "E\_Y" is:')

display(E\_Y.shape)

**Output 1:**

**Table

Description automatically generated with medium confidence**

# Let "D" be the Variable to assign the calculation formulas. Let its initial value be 0.

# Employ a For Loop, with range(40) to fit indexes 0-39 for Arrays "E\_Y" and "y".

# Specify an if-else statement to account for how the formula changes when y = 0.

# For each calculation, we add the values of "D" to itself using += to satisfy the Summation.

# Lastly, compute 2 \* D and assign back to "D", for the Deviance.

D = 0

for i in range(40):

if y[i] == 0:

D += E\_Y[i]

else:

D += y[i]\*(math.log(y[i]/E\_Y[i])) - (y[i] - E\_Y[i])

D = round(2\*D, 6)

print(f'The value of the Deviance is {D}.')

**Output 2:**

****

**References**

Geron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and Tensorflow*

(1st ed, pp. 29). O’Reilly Media, Inc.

Volpi, G.F. (2019). *6 Amateur Mistakes I’ve Made Working With Train-Test Splits.*

Retrieved from: <https://towardsdatascience.com/6-amateur-mistakes-ive-made-working-with-train-test-splits-916fabb421bb>

Brownlee, J. (2020). *Train-Test Split for Evaluating Machine Learning Algorithms*.

Retrieved from: <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>