+

ANL252

Python for Data Analytics

End-of-Course Assessment – July 2021 Semester

1. (a)(i) Reading the csv file and declaring missing values

|  |
| --- |
| Import both pandas and numpy package  import pandas as pd  print('Imported Pandas Library.')  import numpy as np  print('Imported Numpy Library.')  Use pd.read\_csv to read the comma separated value file called ship.csv. As the default na\_values does not contain default strings ".", we have to declare it specifically to let the program know that the "." is a missing value in our dataset.  To ensure that the user is aware that we have already imported the ship.csv into python, we will indicate function to state that the data is loaded along with a portion of the Dataframe.  ship = pd.read\_csv('ship.csv', na\_values = '.')  print('Data Loaded - file \'ship.csv\' - Refer to first 5 observation of the Dataframe below.')  display(ship.head()) |

1. (a)(ii) Renaming Variable Names

|  |
| --- |
| Here, we take use the rename function to help us rename the headers in the table  ship = ship.rename(columns={"T": "types", "A": "c\_years", "P": "o\_periods", "MS": "s\_months","Y": "incidents"})  To ensure that the user is aware that the headers are renamed along with examples, print function is used along with displaying the first 5 observations of ship.csv in the DataFrame.  print('Ship Variables Renamed - Refer to first 5 observation of the Dataframe below.')  display(ship.head()) |

1. (a)(iii) Computing average service months and average number of incidents for cross product of every category in types and operation periods, rounded off.

|  |
| --- |
| Here, we take the column, "c\_years" out by using the drop function.  ship\_data = ship.drop(["c\_years"], axis=1)  Next, we reshuffle the DataFrame to have "types" and "o\_periods" as the indexes so that they are the cross-product of every category in types and operations periods. "s\_months" and "incidents" will be averaged and rouded to an integer.  shipgroup\_first = ship\_data.groupby(['types', 'o\_periods']).mean().round().astype(int)  Lastly, the indexes will be reset back to default as they are not needed to be paired together. All values will be in the same column.  shipgroup = shipgroup\_first.reset\_index()  Print function and display are then used to indicate to user the purpose of the DataFrame.  print('Cross-products of every category in types and operation periods (Avg and Rounded) - Shipgroup DataFrame is as follows:')  display(shipgroup) |

1. (a)(iv) Replacing missing values accordingly

|  |
| --- |
| For this, we use the for loop twice to make replace the mean of "s\_months" and incidents belonging to their respective cetegories for those values that are "NaN".  The for-loop is designed to look into different rows and detect if it is a "NaN" value.  If it is a NaN value, it will check on the "shipgroup" DataFrame and replace the values according to its "types" and "o\_periods".  To ensure that the user is aware of the replacement of values, we use the print function to show what was being done to the original DataFrame as well as allowing the user to check the missing values count are 0. |
| pd.options.mode.chained\_assignment = None  for i in range(10):  for j in range(40):  if (shipgroup['types'][i] == ship['types'][j]) & (shipgroup['o\_periods'][i] == ship['o\_periods'][j]):  if pd.isna(ship['s\_months'][j]):  ship['s\_months'][j] = shipgroup['s\_months'][i]  if pd.isna(ship['incidents'][j]):  ship['incidents'][j] = shipgroup['incidents'][i]    print("All missing values in the variable \"s\_months\" and \"incidents\" has been replaced to the means of similar ships \n")  print("Missing values count:")  display(ship.isnull().sum(axis=0)) |

1. (a)(v) Saving Target Variable “incidents” labelling as “Y”

|  |
| --- |
| Here, we drop all the unnecessary columns required and label it as Y  We inform the user that the DataFrame now has been saved using the print function and display the first 5 observations.  Y = ship.drop(columns=['types', 'c\_years', 'o\_periods', 's\_months'])  print("Saved column \"incidents\" as \"Y\" - Refer to first 5 observation of the Dataframe below. ")  display(Y.head()) |

1. (b)(i) Data Type Conversion to Categorical

|  |
| --- |
| To begin converting the data type to categorical variables, we first look at their types before the conversion.  print("The data type (before conversion) are as follows:")  display(ship.dtypes)  Here, we placed the columns that we want to change from int64 to category into a list. Afterwards, we use the astype function to convert it to category.  cat\_change = ['types', 'c\_years', 'o\_periods']  ship[cat\_change] = ship[cat\_change].astype('category')  We then look at the data type again to ensure they have been converted successfully into categorical variables.  print("The data type (after conversion) are as follows:")  display(ship.dtypes) |

1. (b)(ii) Converting all Categorical Variables to Dummy Variables and labelling as “X”

|  |
| --- |
| To convert into a dummy variable, we then use the get\_dummies function to convert all categorical variables into dummy variables.  We then show the user of the dummy variable of the first 5 observation that it was done successfully.  X = pd.get\_dummies(ship[cat\_change])  print("The dummy variable, \"X\" are as follows - Refer to first 5 observation of the Dataframe below.")  display(X.head()) |

1. (b)(iii) Creating “log\_s\_months” and attaching to DataFrames “X” and “ship”

|  |
| --- |
| Here, we use the log function from numpy library from s\_months. Then, we add it to ship and X.  To ensure that the user knows that both are added, we indicate the print function along with the first 5 rows.  ship["log\_s\_months"] = np.log(ship["s\_months"])  print("Added log-transform of log\_s\_months to \"ship\" DataFrame - Refer to first 5 observation of the Dataframe below.")  display(ship.head())  For X, we have to put them together by using the concatenate function as well as using the outer join to piece them together since we want both uncommon columns to be combined.  df\_log\_s\_months = ship["log\_s\_months"]  X = pd.concat([X, ship['log\_s\_months']], join = 'outer', axis = 1)  print("Added log-transform of log\_s\_months to \"X\" DataFrame - Refer to first 5 observation of the Dataframe below.")  display(X.head()) |

1. (c) Explanation for using Entire Dataset for Training Purpose

Generally, the larger we make our training set, the more confident we can be in the accuracy of the predictive model given that the machine learning model has enough data to to make predictions with it. For this case, we are given a small dataset which is similar to a sample size.

*#1 Predictive Model:- Garbage in, Garbage out*

Although we can still use the machine learning model to churn out algorithms to uncover patterns, make predictions and anticipate future results, working with small data has its pros and cons. For this case, small datasets are easier to deal with and cost effective (e.g. Extracting the info from Generalized Linear Models (New York: Chapman & Hall, 1983) is quick, simple and cost effective) extracting from but the predictive power of the model may be weak as a whole.

If we split the dataframe into training and testing datasets (E.g. 70-30), 70% of data, which in this case only 28 data are used while only 30% of the data are used to test, or 12 data, it will lead to very weak and unreliable output to predict data and ensure that the algorithm can sufficiently produce reliable results. (E.g. only 40% of the time it can predict the data well, which has a low predictive power.)

Given that the information provided to us is acting similar to a sample size, we should just use the entire training dataset to churn out more reliable results for the machine learning algorithms. That way, it will drastically improve the machine learning model to make more reliable predictions. (e.g. From 40%, it has increased.)

Additionally, if the testing data as a whole is small, the confirmation of the predictive power of the model would be weak too. This is because there are insufficient amount of data to provide strong evidence that that the predictive model is working well.

*#2 Opportunity Costs for small datasets*

Of course, by using entire dataset for training purpose, the drawback for this case is that we are unable to confidently ascertain the predictive power of the model. But for this case, the data scientist should favour testing the model completely to generate a much reliable predictive model rather than having a weak predictive model via test-train datasets.

1. (d) Saving csv text file and database as tables

|  |
| --- |
| Here, we shall save the DataFrame "ship" as a new csv file and inform user accordingly.  ship.to\_csv("ship\_prepared.csv", index = False)  print("\"ship\_prepared.csv\" generated.")  To create a database, we first begin by importing sqlite3 package.  import sqlite3  print('Imported sqlite3 Library.')  Since it is a new database we have to create, we use connect function to create the base for "ship.db". Next, we want to read the newly created csv to export the data into "ship.db". Finally, to put the csv data into "ship.db", we use the "to\_sql" function to state that we would like to put the already read csv into out "ship.db"  conn = sqlite3.connect("ship.db")  cur = conn.cursor()  shiptosql = pd.read\_csv("ship\_prepared.csv")  shiptosql  shiptosql.to\_sql("shiptosql", conn, if\_exists = "replace", index = False)  print("\"ship.db\" generated.")  Finally, for user be assured that "ship.db" has the appropriate data, he can refer to the information below. Here, we use the execute function to select table from the Database. Afterwards, we fetch the first and all other records for user to view. Thereafter, we show the description of the headers.  print("\n")  print("For user's info of SQL Database")  cur.execute("SELECT \* FROM shiptosql;")  print("\n")  print("First Observation")  print(cur.fetchone())  print("\n")  print("Second Observation Onwards")  print(cur.fetchall())  print("\n")  print("Headers")  print(cur.description) |

1. (a) Discuss the corresponding module, estimator, fit and predict functions, as well as their parameters

For this case, since Poission Regression was used to study the ship dataset, we shall use the Generalized Linear Model with a Poisson distribution called, "sklearn.linear\_model.PoissonRegressor". In this case, the module name used is linear\_model, given from the example in the website, from "sklearn import linear\_model" (or sklearn.linear\_model) was used. The estimator for this case would be Poission Regression which uses the 'log' link function which means that logE is used for Possion. (as opposed to linear which does not contain logE)

For the estimator, your whole purpose is to feed in data to get the equation of what it is estimated to be like. Estimator is indicated as clf = linear\_model.PoissonRegressor() under the examples in the Scikit Learn Poission Regression website.

For the fit function, is to get the best fit for the generalised linear model using the X which is the training data and y which is the target values. It will churn out the "best fit equation" of the GLM. Here, we can see the source code states it will return an instance of self which means return the response "PoissonRegressor()" once successful.

Lastly, for the predict function, we indicate the samples of X which has many values you want to predict and what it comes out are the list of predicted values. In this case, there are 40 variables that are extracted from the prediction because we are using clf.predict(X) because 40 variables of Y from 1(a)(v) were used.

1. (b) Fitting Poisson Regression and Generate DataFrame to Present Coefficients

|  |
| --- |
| Based on the official website, we are required to:-  1) import sklearn  2) use clf.fit  3) use clf.coef\_  4) use clf.predict  And generate a Dataframe to present the coefficients with the corresponding labels.  from sklearn import linear\_model  print("sklearn imported linear\_model")  print("\n")  Y = np.ravel(Y)  print("Use Poission Regressor to fit a Generalized Linear Model.")  clf = linear\_model.PoissonRegressor()  clf.fit(X, Y)  print(clf)  print("\n")  print("Estimated coefficients for the linear predictor (X @ coef\_ + intercept\_) in the GLM are as follows:")  print(clf.coef\_)  print("\n")  print("Predict using GLM with feature matrix X are as follows:")  print(clf.predict(X))  print("\n")  print("Intercept (a.k.a. bias) added to linear predictor are as follows:")  print(clf.intercept\_)  As stated, we are required to put the coefficients in a DataFrame.  df\_coefficients = pd.DataFrame(clf.coef\_, index = X.columns, columns=['Coefficients of Independent Variables (𝛽)'])  display(df\_coefficients) |

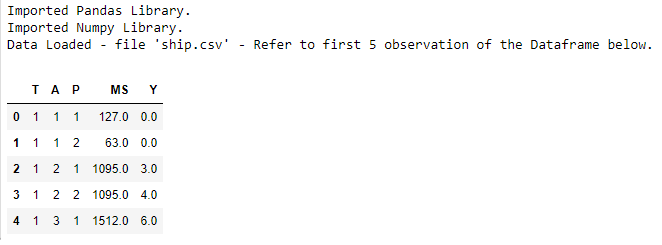
1. (c) Compute for D

|  |
| --- |
| First, we reset the index to allow us to use the DataFrame. We shall name it df\_coefficients\_new.  df\_coefficients\_new = df\_coefficients.reset\_index(drop=True) |
| We shall name it X1 for this case.  X1 = X.transpose().reset\_index(drop=True) |
| Next, we set out with the workings by concatenating both DataFrame together. They are joined on the outer since we want uncommon columns to be together.  working\_table = pd.concat([df\_coefficients\_new, X1], join = 'outer', axis = 1) |
| Here, we get the multiply each coefficient of independent variable (𝛽) with the working tables.  If there is a 1 on the column, it will register and mutiply to get the coefficient, otherwise it will be 0. This will be the Summation of E(Y), excluding the Y-intercept. We store the summation of these values into a list, "Sum\_EY".  Sum\_EY = []  for i in range(len(clf.predict(X))):  Sum\_EY.append(sum(np.array(working\_table['Coefficients of Independent Variables (𝛽)']) \* np.array(working\_table[i]))) |
| Then, we add in the y-intercept 𝛽0 of each summation of E(Y) columns.  EY = Sum\_EY + clf.intercept\_ |

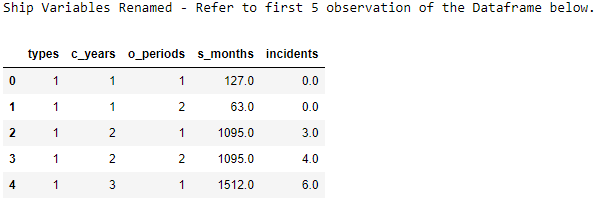
|  |
| --- |
| We have to turn the DataFrame, Y into a numpy array to do the computation. We shall convert it using np.array. We then do the computaton [Y - E(Y)]. We store the summation of these values into a list, "YA\_minus\_EY".  YA\_minus\_EY = []  for i in range(len(clf.predict(X))):  YA\_minus\_EY.append(Y[i] - EY[i]) |
| We shall import math and comppute the portion log[Y/E(Y)]. Here, we adhere to the question stating that "If Y = 0, the expression log[Y/(E(Y))] will be taken as zero." We then put the figures into a list math\_log\_part.  import math  math\_log\_part = []  for i in range(len(clf.predict(X))):  try:  math\_log\_part.append(math.log((Y[i])/(EY[i]),10))  except ValueError:  math\_log\_part.append(0) |
| We create a list, Ymath\_log\_part to store the figures altogether. Additionally, we use a loop to multiply the Y values.  math\_log\_part1 = np.array(math\_log\_part)  Ymath\_log\_part = []  for i in range(0,40): #go and amend the range  Ymath\_log\_part.append(math\_log\_part1[i] \* Y[i]) |
| We shall do the computation of Ylog[Y/E(Y)] - [Y - E(Y)]. We create a list, sum\_values\_indict to store the figures altogether. Additionally, we use a loop to deduct off the values.  sum\_values\_indict = []  for i in range(40): #go and amend the range  sum\_values\_indict.append(Ymath\_log\_part[i] - YA\_minus\_EY[i])  Full\_sum = np.sum(sum\_values\_indict) |
| Finally, we multiply the full sum amount by 2. For clarity, we round off the results to decimal places and use a formatted printing altogether.  Multiplier = 2  Final\_D = Multiplier \* Full\_sum  print(f"The D value is {Final\_D:.2f}") |

**Workings:**

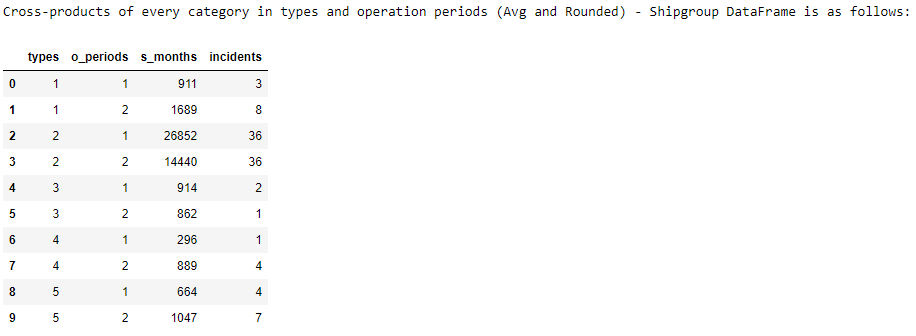
**Question (1)(a)(i) - Reading the csv file and declaring missing values**



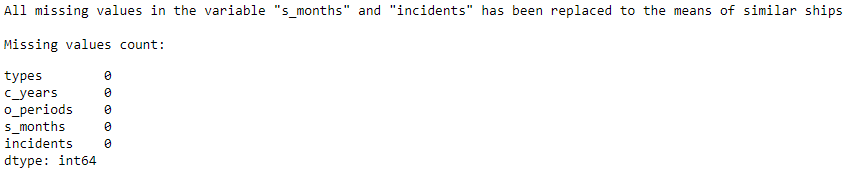
**Question (1)(a)(ii) - Renaming Variable Names**



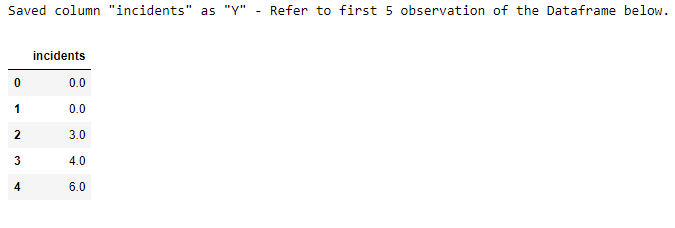
**Question (1)(a)(iii) - Computing average service months and average number of incidents for cross product of every category in types and operation periods, rounded off.**



**Question (1)(a)(iv) - Replacing missing values accordingly**

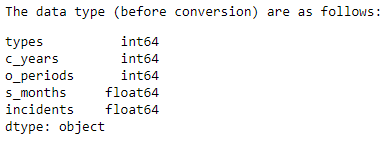


**Question (1)(a)(v) - Saving Target Variable “incidents” labelling as “Y”**

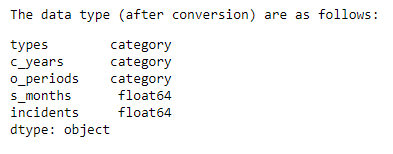


**Question (1)(b)(i) – Data Type Conversion to Categorical**

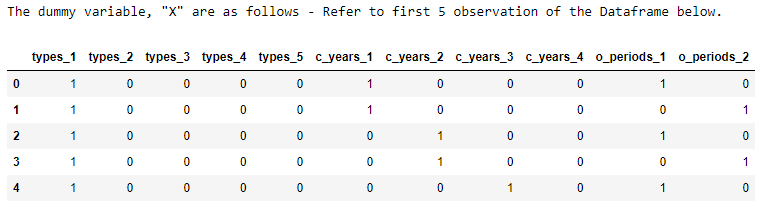
Before Conversion



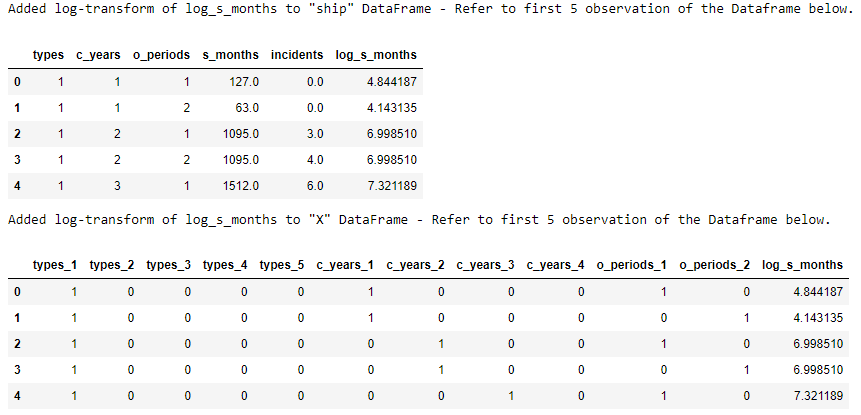
After Conversion



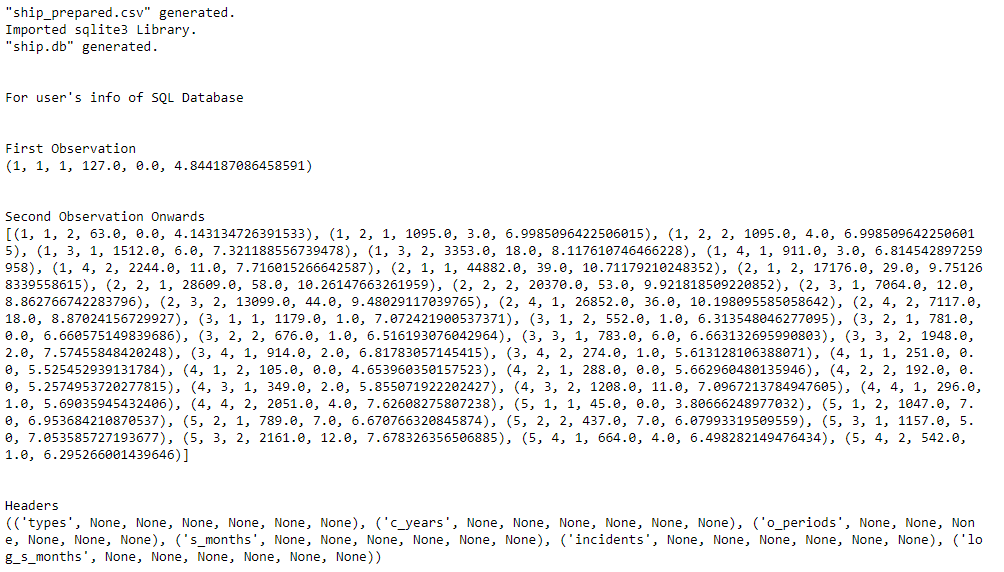
**Question (1)(b)(ii) – Converting all Categorical Variables to Dummy Variables and labelling as “X”**

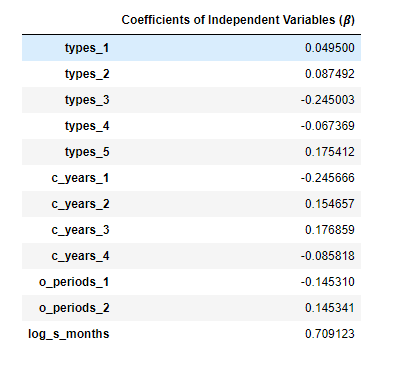


**Question (1)(b)(iii) – Explanation for using Entire Dataset for Training Purpose**



**Question (2)(b) – Fitting Poisson Regression and Generate DataFrame to Present Coefficients**





**Question (2)(c) – Compute for D**



**Jupyter Notebook Attachment:**

****