

**ANL252**

**Python for Data Analytics**

# **End-of-Course Assessment**

**July 2021 Semester**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Tan Mei Ping** | **Q1872131** |

**Submission Date: 13/09/2021**

# Question 1(a)

# Part (i)

# import NumPy and pandas

import pandas as pd

import numpy as np

# read csv file as DataFrame

ship = pd.read\_csv('ship.csv')

# declare '.' as missing values

ship.replace(".", np.nan)

­

A picture containing text, electronics, calculator

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidence

# Part (ii)

# Rename variable names

# Ship type (T) > "types"

# Construction years (A) > "c\_years"

# Operation periods (P) > "o\_periods"

# Aggregrated months of service (MS) > "s\_months"

# Number of incidents (Y) > "incidents"

ship.rename(columns = {"T":"types","A":"c\_years","P":"o\_periods","MS":"s\_months","Y":"incidents"}, inplace=True)

ship.head()

Table

Description automatically generated

# Part (iii)

#Converting data frame objects into numeric

ship['s\_months'] = pd.to\_numeric(ship['s\_months'], errors ='ignore')

ship['incidents'] = pd.to\_numeric(ship['incidents'], errors ='ignore')

ship.dtypes

Text

Description automatically generated with medium confidence

# compute the average service months and the average number of incidents for the cross-products of every category in types and

operation periods.

shipgroup\_smonths = ship.groupby(['o\_periods', 's\_months']).sum().reset\_index().groupby('o\_periods').mean()

shipgroup\_smonths

Table

Description automatically generated

shipgroup\_types = ship.groupby(['types', 's\_months']).sum().reset\_index().groupby('types').mean()

Table

Description automatically generated

# Part (iv)

# Replace the missing values in the variable "s\_months" and "incidents" by the respective means of the other ships that share the same type AND the same operation period

ship["s\_months"] = ship.groupby("types").transform(lambda x: x.fillna(x.mean()))

ship["incidents"] = ship.groupby("types").transform(lambda x: x.fillna(x.mean()))

# In this program, the column of s\_month from the data frame is selected and their missing value is transformed into their respective means by the aid of the function lambda

Table

Description automatically generated

# Part (v)

# Construct a Python program to save the target variable "incidents" in a pandas DataFrame named "Y".

Y = ship['incidents']

# Question 1(b)

# Part (i)

# Perform an appropriate data type conversion for these variables so that they can be recognised as categorical variables

# astype function used to convert variables into categories

ship["types"] = ship["types"].astype('category')

ship["c\_years"] = ship["c\_years"].astype('category')

ship['o\_periods'] = ship["o\_periods"].astype('category')

# Part (ii)

# Construct Python code to convert all categorical variables to dummy variables and save the result as a pandas DataFrame named "X"

ship['types'] = pd.get\_dummies(ship['types'])

ship['c\_years'] = pd.get\_dummies(ship['c\_years'])

ship['o\_periods'] = pd.get\_dummies(ship['o\_periods'])

x = ship

print (x)

A picture containing window blind

Description automatically generated

Table

Description automatically generated

# Part (iii)

# Perform a log-transformation of this variable in the DataFrame and name the transformed variable "log\_s\_months"

# The transformed variable should be attached to both DataFrames "X" and "ship"

x['log\_s\_month'] = np.log(x['s\_months'])

ship['log\_s\_month'] = np.log(ship['s\_months'])

print (x)

print (ship)

Table

Description automatically generated

Table

Description automatically generated

# Part (c)

In this, we are not doing inferential statistics on the data we are just describing and understanding the nature of the variables. Hence, when we will be evaluating a data model, we will be dividing the data into training and test dataset

# Part (d)

# save the prepared DataFrame "ship" as a new csv text file called "ship\_prepared.csv

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

Table

Description automatically generated

Table, calendar

Description automatically generated with medium confidence

# create a database called "ship.db" and export the DataFrame to the database as tables.

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

cur = conn.cursor()

cur.execute('CREATE TABLE IF NOT EXISTS EXPORTS ({})' .format(' ,'.join(ship.columns)))

for rows in ship.iterrows():

sql = ('INSERT INTO EXPORTS ({}) VALUES ({})'.format(' ,'.join(ship.columns), ','.join(['?']\*len(ship.columns))))

cur.execute(sql, tuple(rows[1]))

conn.commit()

Question 2

a)

# Find the corresponding scikit-learn module in the official website of scikit-learn

# discuss the corresponding module, estimator, fit and predict functions, as well as their parameters

# Estimator function

def get\_params(self, deep=True):

The above function is to get parameters for the estimator.

Parameters: deep : bool, default=True

If True, will return the parameters for this estimator and

contained subobjects that are estimators.

Returns: params : dict

Parameter names mapped to their values.

out = dict()

for key in self.\_get\_param\_names():

value = getattr(self, key)

if deep and hasattr(value, 'get\_params'):

deep\_items = value.get\_params().items()

out.update((key + '\_\_' + k, val) for k, val in deep\_items)

out[key] = value

return out

# Fit function

def fit(self, X, y, sample\_weight=None):

The above function is to fit a Generalized Linear Model.

Parameters:

X : array-like of shape of shape (n\_samples, n\_features)

y : array-like of shape (n\_samples,)

sample\_weight : array-like of shape (n\_samples,), default=None

# Predict function

def predict(self, X):

Predict using Generalised Linear Model with feature matrix X.

Parameters:

X : {array-like, sparse matrix} of shape (n\_samples, n\_features)

Returns:

y\_pred : array of shape (n\_samples)

# Returns predicted values.

# check\_array is done in \_linear\_predictor

eta = self.\_linear\_predictor(X)

y\_pred = self.\_link\_instance.inverse(eta)

return

b)

# Analyse the data by fitting a Poisson regression based on the DataFrames X and Y generated in Question 1

# report the parameters of the estimated model

# Create a Python program to fit a Poisson regression and generate a table or a DataFrame to present the coefficients with the corresponding labels.

import pandas as pd

from patsy import dmatrices

import numpy as np

import statsmodels.api as sm

import matplotlib.pyplot as plt

X = pd.read\_csv('ship\_prepared.csv')

X.head()

Table

Description automatically generated

Y = X['incidents']

#Creating training and test data set for:

data = np.random.rand(len(X)) < 0.8

df\_train = X[data]

df\_test = X[~data]

print('Training data set length = '+str(len(df\_train)))

print('Testing data set length = '+str(len(df\_test)))



expr = """incidents~s\_months + o\_periods +types """

y\_train, X\_train = dmatrices(expr, df\_train, return\_type='dataframe')

y\_test, X\_test = dmatrices(expr, df\_test, return\_type='dataframe')

poisson\_training\_results = sm.GLM(y\_train, X\_train, family=sm.families.Poisson()).fit()

print(poisson\_training\_results.summary())

Table

Description automatically generated

c)

#Checking the poisson distribution and finding deviation

poisson\_predictions = poisson\_training\_results.get\_prediction(X\_test)

#summary\_frame() returns a pandas DataFrame

predictions\_summary\_frame = poisson\_predictions.summary\_frame()

print(predictions\_summary\_frame)

Text

Description automatically generated

predicted\_counts=predictions\_summary\_frame['mean']

actual\_counts = y\_test['incidents']

fig = plt.figure()

fig.suptitle('Predicted versus actual')

predicted, = plt.plot(X\_test.index, predicted\_counts, 'go-', label='Predicted counts')

actual, = plt.plot(X\_test.index, actual\_counts, 'ro-', label='Actual counts')

plt.legend(handles=[predicted, actual])

plt.show()

Chart, line chart

Description automatically generated

#Actual vs predicted

plt.clf()

fig = plt.figure()

fig.suptitle('Scatter plot of Actual versus Predicted counts')

plt.scatter(x=predicted\_counts, y=actual\_counts, marker='.')

plt.xlabel('Predicted counts')

plt.ylabel('Actual counts')

Chart, scatter chart

Description automatically generated