Logistic Regression Model

1. Binary Logistic Regression Model
2. Nominal Logistic Regression Model
3. Ordinal Logistic Regression Model

Binary Logistic Regression Model

- Binary Logistic equation
- when the response has binary values

Binary Logistic Regression Model (mtcars dataset)

import statsmodels

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving CDAC_DataBook.xlsx to CDAC_DataBook (1).xlsx
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='mtcars')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

```
df = df[['hp', 'wt', 'am']]
# selecting specific columns from dataset
df.head()
# printing new DataFrame of only required columns
```

	hp	wt	am
0	110	2.620	1
1	110	2.875	1
2	93	2.320	1
3	110	3.215	0
4	175	3.440	0

df.drop('Pred_col',axis=1)

- drops response col

```
x_train = df.drop('am',axis=1)
# manually creating Predictor-Training dataFrame by dropping Response col
# without train_test_split()
# x_test is not created
x_train.head()
```

	hp	wt
0	110	2.620
1	110	2.875
2	93	2.320
3	110	3.215
4	175	3.440

df['Response']

- selects response col

```
y_train = df['am']
# manually creating Response-Training Series by selecting only Response col
# without train_test_split()
# y_test is not created
y_train.head()
```

```
0    1
1    1
2    1
3    0
4    0
Name: am, dtype: int64
```

sm.add_constant(x_train,prepend=False)

```
x_train=sm.add_constant(x_train,prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Binary Logistic Equation but without a constant
# if we add constant, model will follow Binary Logistic Equation with constant
x_train.head()
# checking columns in Predictor-training DataFrame after adding constant
```

	hp	wt	const
0	110	2.620	1.0
1	110	2.875	1.0
2	93	2.320	1.0
3	110	3.215	1.0
4	175	3.440	1.0

sm.Logit(y_train,x_train).fit()

```
mod1=sm.Logit(y_train,x_train).fit()
# creating model using sm.Logit(y, x).fit() method
```

```
Optimization terminated successfully.
         Current function value: 0.157174
         Iterations 9
```

```
print(mod1.summary())
# printing model summary / Logit Regression Results
```

```
                           Logit Regression Results                           
==============================================================================
Dep. Variable:                     am   No. Observations:                   32
Model:                          Logit   Df Residuals:                       29
Method:                           MLE   Df Model:                            2
Date:                Fri, 30 Jun 2023   Pseudo R-squ.:                  0.7673
Time:                        09:47:30   Log-Likelihood:                -5.0296
converged:                       True   LL-Null:                       -21.615
Covariance Type:            nonrobust   LLR p-value:                 6.267e-08
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
hp             0.0363      0.018      2.044      0.041       0.001       0.071
wt            -8.0835      3.069     -2.634      0.008     -14.098      -2.069
const         18.8663      7.444      2.535      0.011       4.277      33.455
==============================================================================

Possibly complete quasi-separation: A fraction 0.12 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

pd.DataFrame([list1, list2, list3], columns=index)

```
mydata = pd.DataFrame([[120, 2, 1], [120, 2, 5], [120, 3, 1]], columns=['hp', 'wt', 'const'])
# manually creating a Predictor-Test DataFrame using random data

# DataFrame created keeping hp constannt but varying 'wt' col to
# see influence of wt on 'am' col

mydata
# printing Predictor-Test DataFrame
```

	hp	wt	const
0	120	2	1
1	120	2	5
2	120	3	1

model.predict(Test_DataFrame)

```
mod1.predict(mydata)
# generating Prediction Series
```

```
0    0.999133
1    1.000000
2    0.262415
dtype: float64
```

np.log(Pred_a/(1-Pred_a)) - np.log(Pred_b/(1-Pred_b))

- Binary Logistic Regression Equation

```
np.log(0.999133/(1-0.999133)) - np.log(0.262415/(1-0.262415))
# np.log(Pred_a/(1-Pred_a)) - np.log(Pred_b/(1-Pred_b))
# follows Binary Logistic Regression Equation to generate coefficient

# Pred_a : take prediction for the lesser 'wt' of Test_DataFrame
# Pred_b : take prediction for the highest 'wt' of Test_DataFrame


# find difference of log of odds of two predictions
# odds = p(1-P)
```

```
8.083058320861667
```

- Binary Logistic Regression Equation/coefficient = difference of logs of odds of two predictors
- Binary Logistic Regression Equation/coefficient = `log(odds of A) - log(odds of B)`
- Binary Logistic Regression Equation/coefficient = `np.log(Pred_a/(1-Pred_a)) - np.log(Pred_b/(1-Pred_b))`

- `odds = p(1-P)`
- `odds = Proportion(1-Proportion)`

- Pred_a : take prediction for the lesser 'wt' of Test_DataFrame
- Pred_b : take prediction for the highest 'wt' of Test_DataFrame

Binary Logistic Regression Model (diabetes dataset)

import statsmodels & sklearn

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving CDAC_DataBook.xlsx to CDAC_DataBook.xlsx
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='diabetes')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Out
0	148	72	35	0	33.6	0.63	50	
1	85	66	29	0	26.6	0.35	31	
2	183	64	0	0	23.3	0.67	32	
3	89	66	23	94	28.1	0.17	21	
4	137	40	35	168	43.1	2.29	33	

df.columns

```
df.columns
# fetching column names
```

```
Index(['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
       'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
df = df[['Glucose', 'BloodPressure', 'Age', 'Outcome']]
# selecting specific columns from dataset
df.head()
# printing new DataFrame of only required columns
```

	Glucose	BloodPressure	Age	Outcome
0	148	72	50	1
1	85	66	31	0

train_test_split(predictor_cols, response_col, test_size=0.25)

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('Outcome', axis=1), df['Outcome'], test_size=.025)
# splitting dataset 4-ways
# rows are randomly selected for testing
```

sm.add_constant(x_train, prepend=False)

```
x_train = sm.add_constant(x_train, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Binary Logistic Equation but without a constant
# if we add constant, model will follow Binary Logistic Equation with constant
```

sm.OLS(y_train, x_train).fit()

```
mod1 = sm.OLS(y_train, x_train).fit()
# creating model using sm.OLS().fit()
```

```
print(mod1.summary())
# printing model summary / OLS Regression Results
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Outcome   R-squared:                       0.235
Model:                            OLS   Adj. R-squared:                  0.232
Method:                 Least Squares   F-statistic:                     76.17
Date:                Thu, 29 Jun 2023   Prob (F-statistic):           5.65e-43
Time:                        19:24:03   Log-Likelihood:                -408.23
No. Observations:                 748   AIC:                             824.5
Df Residuals:                     744   BIC:                             842.9
Df Model:                           3
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
Glucose           0.0066      0.000     13.221      0.000       0.006       0.008
BloodPressure    -0.0008      0.001     -0.942      0.346      -0.002       0.001
Age               0.0051      0.001      3.693      0.000       0.002       0.008
```

```
const           -0.5607      0.079     -7.118      0.000      -0.715      -0.406
==============================================================================
Omnibus:                       51.943   Durbin-Watson:                   1.904
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               42.118
Skew:                           0.495   Prob(JB):                     7.15e-10
Kurtosis:                       2.391   Cond. No.                         756.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Establishing Null Hypothesis `Hₒ`

- `Hₒ` : BloodPressure does not impact outcome
- BloodPresure `P-Value 0.346` > `0.05`, so we do not reject `Hₒ`, blood pressure does not impact outcome

sm.add_constant(x_test, prepend=False)

```
x_test = sm.add_constant(x_test, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Binary Logistic Equation but without a constant
# if we add constant, model will follow Binary Binary Logistic Binomial Equation with constant
x_test.head()
# checking columns in Predictor-testing DataFrame after adding constant
```

	Glucose	BloodPressure	Age	const
364	147	74	30	1.0
637	94	76	23	1.0
555	124	70	37	1.0
16	118	84	31	1.0
62	44	62	36	1.0

model.predict(x_test)

```
pred_y = mod1.predict(x_test)
# generate prediction using x_test sample
```

```
pred_y[:5]
# printing first 6 records from prediction Series
```

```
364    0.501328
637    0.115016
555    0.388427
16     0.307377
62    -0.137457
dtype: float64
```

```
y_test[:5]
# printing first 6 records from Response-Testing Series
```

```
364    0
637    0
555    0
16     1
62     0
Name: Outcome, dtype: int64
```

```
res = [] # creating empty list to store rounded off prediction values
for ctr in pred_y:
    if ctr < 0.5:
        res.append(0)
    else:
        res.append(1)
```

```
res[:5]
# printing first 6 records from prediction Series with rounding off
```

```
[0, 0, 0, 0, 1]
```

import confusion_matrix

```
from sklearn.metrics import confusion_matrix
#  importing confusion_matrix module
```

confusion_matrix(y_test, y_pred_rounded)

```
confusion_matrix(y_test, res)
# generating confusion matrix between Response-Testing Series & Prediction
```

```
array([[12,  0],
       [ 5,  3]])
```

Confusion Matrix

- `True / False` : Actual/rows
- `Positive / Negative` : Predicted/columns

Confusion Matrix Structure

```
              Predicted
               P    N
Test    T   [TP , TN]
        F   [FP , FN]

               P    N
Test    T   [12 , 0]
        F   [ 5 , 3]
```

- True Positive `TP` = 12 [predictions are correct for True]
- True Negative `TN` = 0 [Type 1 Error]
- False Positive `FP` = 5 [Type 2 Error]
- False Negative `FN` = 3 [Predictions are correct for False]
- diagonal elements are errorneous(Type1 or Type2)

Precision & Recall

- Precision = `TP / (TP + FP)`
- Recall = `TP / (TP + FN)`

import classification_report

```
from sklearn.metrics import classification_report
# importing classification_report
```

classification_report(y_test, y_pred_rounded)

```
print(classification_report(y_test, res))
# printing classification report
# shows precision, recall, f1-score & accuracy
```

```
              precision    recall  f1-score   support

           0       0.71      1.00      0.83        12
           1       1.00      0.38      0.55         8

    accuracy                           0.75        20
   macro avg       0.85      0.69      0.69        20
weighted avg       0.82      0.75      0.71        20
```

```
confusion_matrix(y_test, res)
# generating confusion matrix between Response-Testing Series & Prediction
```

```
array([[12,  0],
       [ 5,  3]])
```

```
12/(12+0)
# manually calculating  precision
# Precision = TP / (TP + FP)
```

```
1.0
```

```
12 / (12 + 5)
# manually calculating recall
# Recall = TP / (TP + FN)
```

```
0.7058823529411765
```

F1-Score

- F1-Score = `2 * Precision * Recall / (Precision + Recall)`
- F1-Score = `2 * P * R / (P + R)`

```
2 * 1.0 * 0.38 / (1.0 + 0.38)
# manually calculating  F1-Score
```

```
0.5507246376811594
```

```
(12 + 3)/(12 + 3 + 0 + 5)
# Ratio of corrrect predictions
```

```
0.75
```

```
(1.00 + 0.38)/2
# average recall value
```

```
(1.00 + 0.38) / (12 + 3 + 0 + 5)
#  weighted average for recall
```

```
(0.71 + 1.00)/2
# average precision value
```

```
(0.71 + 1.00) / (12 + 3 + 0 + 5)
#  weighted average for precision
```

Nominal Logistic Regression Model

- when response has multiple categories, and the response does not has a logical order, then we use nominal logistic regression

import statsmodels & sklearn

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name = 'nominal')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	ses	write	math	prog
0	1	35	41	1
1	2	33	41	2
2	3	39	44	3
3	1	37	42	1
4	2	31	40	2

```
df = df.drop('write', axis=1)
# selecting specific/required columns
# by fropping unwanted columns from dataset
df.head()
# checking dataset after selectng only required columns
```

	ses	math	prog
0	1	41	1
1	2	41	2
2	3	44	3
3	1	42	1
4	2	40	2

pd.get_dummies(categorical_col, drop_first=True)

```
ses_dummy = pd.get_dummies(df['ses'], drop_first=True)
# 'ses' is categorical data, so need to create dummies
# creating dummy column
```

```
df = df.drop('ses', axis=1)
# dropping actual categorical column
```

```
df = pd.concat([df, ses_dummy], axis=1)
# concatenating dummy column in place of actual categorical column
df.head()
# checking columns in DataFrame after concatenating categorical column
```

	math	prog	2	3
0	41	1	0	0
1	41	2	1	0
2	44	3	0	1
3	42	1	0	0
4	40	2	1	0

train_test_split(predictor_cols, response_col, test_size=0.25)

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('prog', axis=1), df['prog'], test_size=0.25)
# splitting dataset 4-ways
# rows are randomly selected for testing
```

sm.add_constant(x_train, prepend=False)

```
x_train = sm.add_constant(x_train, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Nominal Logistic Equation but without a constant
# if we add constant, model will follow Nominal Logistic Equation with constant
```

sm.MNLogit(y_train, x_train).fit()

- uses Multi-Nominal Logistic function

```
mod1 = sm.MNLogit(y_train, x_train).fit()
# creating model using sm.MNLogit().fit()
```

```
Optimization terminated successfully.
         Current function value: 0.706854
         Iterations 7
```

```
print(mod1.summary())
# printing model summary / MNLogit Regression Results
```

```
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                   prog   No. Observations:                  150
Model:                        MNLogit   Df Residuals:                      142
Method:                           MLE   Df Model:                            6
Date:                Wed, 21 Jun 2023   Pseudo R-squ.:                  0.3260
Time:                        06:10:07   Log-Likelihood:                -106.03
converged:                       True   LL-Null:                       -157.32
Covariance Type:            nonrobust   LLR p-value:                 7.239e-20
==============================================================================
    prog=2       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
math           0.0139      0.031      0.445      0.656      -0.047       0.075
2              2.7011      0.594      4.545      0.000       1.536       3.866
3              2.4412      1.138      2.145      0.032       0.211       4.671
const         -1.3353      1.583     -0.844      0.399      -4.437       1.767
------------------------------------------------------------------------------
    prog=3       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
math           0.1178      0.037      3.189      0.001       0.045       0.190
2              4.3356      0.947      4.577      0.000       2.479       6.192
3              6.1418      1.318      4.660      0.000       3.559       8.725
const         -8.8537      2.250     -3.935      0.000     -13.263      -4.444
==============================================================================
```

interpreting rows in MNLogit Regression Summary

- Predictor(X) is `ses`

  ```
  ses { 1: "Low", 2: "Middle", 3: "High" }
  ```

- Response(Y) is `prog`

  ```
  prog { 1: "Vocational", 2: "General", 3: "Academic" }
  ```

- row1, `prog-2`: when `ses` changes from refernce ses `1` to ses `2` then the prob of choosing `prog-2`
- row2, `prog-3`: when `ses` changes from reference ses `1` to ses `3` then the prob of choosing `prog-3`

interpreting coefficients

- if coefficient is `-ve`, probability is indicated by movement `towards reference value` or `away from target value`
- if coefficient is `+ve`, probability is indicated by movement `towards target value` or `away from reference value`

```
np.log(12)
# log(n) of n>1 is positive
```

```
2.4849066497880004
```

```
np.log(0.12)
# log(n) of n<1 is negative
```

```
-2.120263536200091
```

- in `prog-2` row, coeff of `2` = 2.7011 = `log( prob(prog2) / prob(prog1) )`
- since this value is positive, means `prob(prog2)/prob(prob1) > 1` because log(n>1) is positive
- This further implies `prob(prog2) > prob(prog1)`, because `Numerator/Denominator > 1` means `Numerator > Denominator`

Establishing Null Hypothesis `Hₒ`

- for math column
  - `Hₒ` : math will not impact changing/choosing the prog1 to prog2
- for ses column
  - Surya's `Hₒ` statement: changing `ses` from reference `ses-1` to target `ses-2` will not impact the changing the refernce prog `prog-1` to target `prog-2`
  - Sudeep's `Hₒ` statement: movement of ses from reference `ses-1` to target `ses-2` will not influence changing the reference `prog-1` to target `prog-2`

Prediction: `math` score impacts `prog`

Q. will score in maths impact the choice of course. if yes, then how?

sm.add_constant(x_test, prepend=False)

```
x_test = sm.add_constant(x_test, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Nominal Logistic Equation but without a constant
# if we add constant, model will follow Nominal Logistic Equation with constant
```

model.predict(x_test)

```
pred_y = mod1.predict(x_test)
# generate prediction using x_test sample
```

```
pred_y[:5]
# printing first five records from prediction for ses-1, 2, 3
```

```
            0         1         2
80   0.609625  0.340005  0.050370
119  0.081625  0.632541  0.285834
175  0.580719  0.342423  0.076858
49   0.085070  0.650130  0.264799
85   0.615967  0.338794  0.045239
```

```
x_train.head()
# printing first 5 records from Predictor-Training DataFrame
```

```
     math  2  3  const
148    55  1  0    1.0
88     42  1  0    1.0
139    58  0  1    1.0
135    56  0  1    1.0
89     46  1  0    1.0
```

```
mydata = pd.DataFrame([[60, 1, 0, 1], [70, 1, 0, 1], [80, 1, 0, 1]], columns=['math', '2', '3', 'const'])
# manually creating a Predictor-Test DataFrame using random data

# DataFrame created keeping 'ses' constant as 'ses-2' but varying 'math' col to
# see influence of "math" on 'prog' col

mydata
# printing Predictor-Test DataFrame
```

	math	2	3	const
0	60	1	0	1
1	70	1	0	1
2	80	1	0	1

model.predict(Test_DataFrame)

```
mod1.predict(mydata)
# generating Prediction Series
```

	0	1	2
0	0.043813	0.395686	0.560501
1	0.018896	0.196128	0.784976
2	0.006764	0.080695	0.912541

```
mydata = pd.DataFrame([[70, 0, 0, 1], [70, 1, 0, 1], [70, 0, 1, 1]], columns=['math', '2', '3', 'const'])
# keeping maths score 70 as constant, and varying ses value
# DataFrame created keeping 'math' constant as '70' but varying 'ses' col to
# see influence of 'ses' on 'prog' col
mydata
# printing Predictor-Test DataFrame
```

	math	2	3	const
0	70	0	0	1
1	70	1	0	1
2	70	0	1	1

```
mod1.predict(mydata)
# generating Prediction Series
```

	0	1	2
0	0.446277	0.310973	0.242749
1	0.018896	0.196128	0.784976
2	0.003819	0.030566	0.965615

Variance Inflation Factor (VIF)

- used to find correlation between two predictors
- VIF should be minimum, `VIF < 10` is good
- Sometimes, we may accept `VIF <= 20`, but not more than 20
- If some predictor has `VIF > 20` , then we drop that column to avoid any impact of the high correlation

import statsmodels

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving CDAC_DataBook.xlsx to CDAC_DataBook.xlsx
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='VIF')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	Education	Region	Gender	Exp	Union	Wage	Age	Race	Occupation	Sector	Married
0	9	0	1	42	0	4.95	57	3	6	1	1
1	12	0	0	1	0	6.67	19	3	6	1	0
2	12	0	0	4	0	4.00	22	3	6	0	0
3	12	0	0	17	0	7.50	35	3	6	0	1
4	13	0	0	9	1	13.07	28	3	6	0	0

df.columns

```
df.columns
# fetching column names
```

```
Index(['Education', 'Region', 'Gender', 'Exp', 'Union', 'Wage', 'Age', 'Race',
       'Occupation', 'Sector', 'Married'],
      dtype='object')
```

```
df = df[['Education', 'Gender', 'Exp', 'Age', 'Wage']]
# selecting specific columns from dataset
```

pd.get_dummies(categorical_col, drop_first=True)

```
gd_dummy = pd.get_dummies(df['Gender'], drop_first=True)
# 'Gender' is categorical data, so need to create dummies
# creating dummy column
gd_dummy.head()
```

	1
0	1
1	0
2	0
3	0
4	0

```
df = df.drop('Gender', axis=1)
# dropping actual categorical column
df.head()
```

	Education	Exp	Age	Wage
0	9	42	57	4.95
1	12	1	19	6.67
2	12	4	22	4.00
3	12	17	35	7.50
4	13	9	28	13.07

```
df = pd.concat([df,gd_dummy], axis=1)

df.head()
```

	Education	Exp	Age	Wage	1
0	9	42	57	4.95	1
1	12	1	19	6.67	0
2	12	4	22	4.00	0
3	12	17	35	7.50	0
4	13	9	28	13.07	0

```
x_train = df.drop('Wage', axis=1)
y_train = df['Wage']
```

```
mod1 = sm.OLS(y_train, x_train).fit()
```

```
print(mod1.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                   Wage   R-squared (uncentered):                   0.817
Model:                            OLS   Adj. R-squared (uncentered):              0.816
Method:                 Least Squares   F-statistic:                              591.8
Date:                Wed, 21 Jun 2023   Prob (F-statistic):                   1.10e-193
Time:                        11:26:07   Log-Likelihood:                         -1550.9
No. Observations:                 533   AIC:                                      3110.
Df Residuals:                     529   BIC:                                      3127.
Df Model:                           4                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Education      1.6271      0.270      6.027      0.000       1.097       2.157
Exp            0.8026      0.205      3.920      0.000       0.400       1.205
Age           -0.6891      0.195     -3.531      0.000      -1.072      -0.306
1             -2.3592      0.388     -6.073      0.000      -3.122      -1.596
==============================================================================
Omnibus:                      250.476   Durbin-Watson:                   1.870
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2546.360
Skew:                           1.795   Prob(JB):                         0.00
Kurtosis:                      13.088   Cond. No.                         94.9
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
variance_inflation_factor(df.values, 0)
```

```
368.9046844691842
```

```
variance_inflation_factor(df.values, 1)
```

```
544.6987999131815
```

```
variance_inflation_factor(df.values, 2)
```

```
1562.7050138396557
```

```
variance_inflation_factor(x_train, 2)
```

```
1526.713272763991
```

- If 0 <= VIF <= 10, it is okay
- if 10 < VID <=25 , it can be ignored
- for VIF > 25, we need to take action

```
x_train.head()
```

	Education	Exp	Age	1
0	9	42	57	1
1	12	1	19	0
2	12	4	22	0
3	12	17	35	0
4	13	9	28	0

```
x_train = x_train.drop('Age', axis=1)
```

```
mod1 = sm.OLS(y_train, x_train).fit()
```

```
print(mod1.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                   Wage   R-squared (uncentered):                   0.813
Model:                            OLS   Adj. R-squared (uncentered):              0.812
Method:                 Least Squares   F-statistic:                              768.2
Date:                Wed, 21 Jun 2023   Prob (F-statistic):                   1.73e-192
Time:                        11:26:27   Log-Likelihood:                         -1557.1
No. Observations:                 533   AIC:                                      3120.
Df Residuals:                     530   BIC:                                      3133.
Df Model:                           3                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
Education      0.6777      0.025     27.032      0.000       0.628       0.727
Exp            0.0813      0.014      5.735      0.000       0.053       0.109
1             -2.4805      0.391     -6.342      0.000      -3.249      -1.712
==============================================================================
Omnibus:                      256.586   Durbin-Watson:                   1.815
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2335.399
Skew:                           1.894   Prob(JB):                         0.00
Kurtosis:                      12.529   Cond. No.                         48.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
variance_inflation_factor(df.values, 0)
```

```
368.9046844691842
```

```
variance_inflation_factor(x_train, 0)
```

```
2.9134749789780923
```

```
variance_inflation_factor(x_train, 1)
```

```
2.4837216129295894
```

```

```

- if the response is binary, we don't need to create dummy columns

Ordinal Logistic Regression Model

- when response has multiple categories, and the response has a logical order, then we use ordinal logistic regression

import statsmodels

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
```

```
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving CDAC_DataBook.xlsx to CDAC_DataBook.xlsx
```

import sys

```
# import sys
# sys.path.append(r'D:/advanced-analytics-files/day10')
# to add a path to interpreter for current session
# to be used while running on local system
# optional : to avoid setting file path each time
```

install mord

```
pip install mord
```

```
Collecting mord
  Downloading mord-0.7.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: mord
```

```
  Building wheel for mord (setup.py) ... done
  Created wheel for mord: filename=mord-0.7-py3-none-any.whl size=9885 sha256=e12fe207f39d1f30c7ed935b0ef39a18a044b595a24023fd45c94db8ad199241
  Stored in directory: /root/.cache/pip/wheels/77/00/19/3cea86fbfc737ec4acb515cd94497dcc33f943fa157548b96c
Successfully built mord
Installing collected packages: mord
Successfully installed mord-0.7
```

import mord.LogisticAT

```
import mord
from mord import LogisticAT
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='ordinal')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	Survival	Region	ToxicLevel
0	1	1	62.0
1	1	2	46.0
2	2	1	48.5
3	3	2	32.0
4	2	1	63.5

pd.get_dummies(categorical_col, drop_first=True)

```
reg_dummy = pd.get_dummies(df['Region'], drop_first=True)
# 'Region' is categorical data, either 1 or 2, so need to create dummies
# creating dummy column
reg_dummy.head()

```

```
   2
0  0
1  1
```

```
df = df.drop('Region', axis=1)
# dropping actual categorical column 'Region'
df.head()
```

```
   Survival  ToxicLevel
0         1        62.0
1         1        46.0
2         2        48.5
3         3        32.0
4         2        63.5
```

```
df = pd.concat([df, reg_dummy], axis=1)
# concatenating dummy column in place of actual categorical column
df.head()
# checking columns in DataFrame after concatenating categorical column
```

```
   Survival  ToxicLevel  2
0         1        62.0  0
1         1        46.0  1
2         2        48.5  0
3         3        32.0  1
4         2        63.5  0
```

train_test_split(predictor_cols, response_col, test_size=0.25)

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('Survival', axis=1), df['Survival'], test_size=0.25)
# splitting dataset 4-ways
# rows are randomly selected for testing
```

sm.add_constant(x_train, prepend=False)

```
x_train = sm.add_constant(x_train, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Nominal Ordinal Equation but without a constant
# if we add constant, model will follow Nominal Ordinal Equation with constant
```

mord.LogisticAT().fit(x_train, y_train)

```
mod1 = LogisticAT().fit(x_train, y_train)
# creating model using mord.LogisticAT().fit(x_train, y_train)
```

```
# print(mod1.summary())
# cannot print summary for LogisticAT() model
# as it does not have summary() method
```

```
<bound method LogisticAT.score of LogisticAT()>
```

```
x_test = sm.add_constant(x_test, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Nominal Ordinal Equation but without a constant
# if we add constant, model will follow Nominal Ordinal Equation with constant
```

model.predict(x_test)

```
pred_y = mod1.predict(x_test)
# generate prediction using x_test sample
pred_y[:5]
```

```
array([2, 2, 2, 1, 2])
```

confusion_matrix(y_test, y_pred_rounded)

```
print(confusion_matrix(y_test, pred_y))
# generating confusion matrix between Response-Testing Series & Prediction
```

```
[[ 0  4  0]
 [ 2 10  0]
 [ 0  3  0]]
```

Double-click (or enter) to edit

Counts Regression Model

1. Poisson Regression Model
2. Negative Binomial Regression Model

- uses `.from_formula(Resp ~ P1 + P2 + P2)` to represent the relation between response & predictors

Poisson Regression

- when response is discrete data
- when the variation is expected to be low

import statsmodels

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='poisson')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	num_awards	prog	math
0	0	3	41
1	0	1	41
2	0	3	44
3	0	3	42
4	0	3	40

pd.get_dummies(categorical_col, drop_first=True)

```
prog_dummy = pd.get_dummies(df['prog'], drop_first=True)
# prog is categorical data, so need to create dummies
# creating dummy column
prog_dummy.head()
```

	2	3
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

```
df = df.drop('prog', axis=1)
# dropping actual categorical column
df.head()
# checking columns in DataFrame after dropping categorical column
```

	num_awards	math
0	0	41
1	0	41
2	0	44
3	0	42
4	0	40

```
df = pd.concat([df, prog_dummy], axis=1)
# concatenating dummy column in place of actual categorical column
df.head()
# checking columns in DataFrame after concatenating categorical column
```

	num_awards	math	2	3
0	0	41	0	1
1	0	41	0	0
2	0	44	0	1
3	0	42	0	1
4	0	40	0	1

train_test_split(predictor_cols, response_col, test_size=0.25)

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('num_awards', axis=1), df['num_awards'], test_size=0.25)
# splitting dataset 4-ways
# rows are randomly selected for testing
```

sm.add_constant(x_train, prepend=False)

```
x_train = sm.add_constant(x_train, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Poisson Equation but without a constant
# if we add constant, model will follow Poisson Equation with constant
x_train.head()
# checking columns in Predictor-training DataFrame after adding constant
```

import Poisson

```
from statsmodels.discrete.discrete_model import Poisson as psn
# import statsmodels.discrete.discrete_model.Poisson
```

pd.concat([x_train, y_train], axis=1)

```
df_train = pd.concat([x_train, y_train], axis=1)
# # concatenating Predictor(X) & Response(Y) into single DataFrame
# so that relation can be established in formula
df_train.head()
# checking columns in concatenated training DataFrame
```

	math	2	3	const	num_awards
148	55	1	0	1.0	0
199	73	1	0	1.0	3
27	46	0	0	1.0	1
93	50	1	0	1.0	0
118	54	1	0	1.0	0

df_train.columns

```
x = df_train.columns
# fetching column names
x
# printing column names
```

```
Index(['math', 2, 3, 'const', 'num_awards'], dtype='object')
```

df_train.rename(columns={2:'Col2_name', 3:'Col3_name'}, inplace=True)

```
df_train.rename(columns={2:'prog2', 3:'prog3'}, inplace=True)
# rename columns because column name is apprearing as column index
df_train.head()
# checking columns in renamed training DataFrame
```

	math	prog2	prog3	const	num_awards
148	55	1	0	1.0	0
199	73	1	0	1.0	3
27	46	0	0	1.0	1

Poisson.from_formula('Response ~ P1 + p2 + p3', data=DataSet).fit()

```python
mod1 = psn.from_formula('num_awards ~ math + prog2 + prog3', data=df_train).fit()
# establishing relation in formula
# creating model using Poisson.from_formula() method
```

```
Optimization terminated successfully.
         Current function value: 0.990968
         Iterations 6
```

```python
print(mod1.summary())
# printing model summary / Poisson Regression Results
```

```
                          Poisson Regression Results                          
==============================================================================
Dep. Variable:             num_awards   No. Observations:                  150
Model:                        Poisson   Df Residuals:                      146
Method:                           MLE   Df Model:                            3
Date:                Wed, 21 Jun 2023   Pseudo R-squ.:                  0.2003
Time:                        12:25:29   Log-Likelihood:                -148.65
converged:                       True   LL-Null:                       -185.87
Covariance Type:            nonrobust   LLR p-value:                 4.764e-16
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -4.9471      0.742     -6.667      0.000      -6.402      -3.493
math           0.0699      0.012      5.634      0.000       0.046       0.094
prog2          0.8920      0.364      2.452      0.014       0.179       1.605
prog3          0.1469      0.487      0.301      0.763      -0.808       1.102
==============================================================================
```

- `Hₒ` : if `prog` changes `prog-1` --> `prog-2` , then the number of awards is not impacted
    - since `P-Value 0.014` < `0.05` , so we reject this `Hₒ` , means if prog changes `prog-1` --> `prog-2` , then number of awards changes
- `Hₒ` : if `prog` changes `prog-1` --> `prog-3`, then the number of awards is not impacted
    - since `P-Value 0.763` > `0.05` , so we do not reject this `Hₒ` , means if prog changes `prog-1` --> `prog-3`, then number of awards does not change

```
```

Negative Binomial Regression Model

- when response is discrete data
- when the variation is expected to be high

import statsmodels

```
import numpy as np
import pandas as pd
# import numpy & pandas

import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
# import statsmodels

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# import sklearn
```

upload dataset

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving CDAC_DataBook.xlsx to CDAC_DataBook.xlsx
```

pd.read_excel('WorkBook_Name.xlsx', sheet_name='SheetName')

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='neg_bin')
# reading excelfile with specifying the sheet name to load dataset
df.head()
```

	math	prog	daysabs
0	63	Academic	4
1	27	Academic	4
2	20	Academic	2
3	16	Academic	3
4	2	Academic	3

import NegativeBinomial

```
from statsmodels.discrete.discrete_model import NegativeBinomial as ngb
# import statsmodels.discrete.discrete_model.NegativeBinomial
```

pd.get_dummies(categorical_col, drop_first=True)

- create dummies for categorical columns
- first value in alphabetical order is automatically taken as reference value

```
prog_dummy = pd.get_dummies(df['prog'], drop_first=True)
# creating dummy column
prog_dummy.head()
```

```
   General  Vocational
0        0           0
1        0           0
2        0           0
3        0           0
```

```
df = df.drop('prog', axis=1)
# dropping actual categorical column
df.head()
# checking columns in DataFrame after dropping categorical column
```

```
   math  daysabs
0    63        4
1    27        4
2    20        2
3    16        3
4     2        3
```

```
df = pd.concat([df, prog_dummy], axis=1)
# concatenating dummy column in place of actual categorical column
df.head()
# checking columns in DataFrame after concatenating categorical column
```

```
   math  daysabs  General  Vocational
0    63        4        0           0
1    27        4        0           0
2    20        2        0           0
3    16        3        0           0
4     2        3        0           0
```

train_test_split(predictor_cols, response_col, test_size=0.25)

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('daysabs', axis=1), df['daysabs'], test_size=0.25)
# splitting dataset 4-ways
# rows are randomly selected for testing
```

sm.add_constant(x_train, prepend=False)

```
x_train = sm.add_constant(x_train, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Negative Binomial Equation but without a constant
# if we add constant, model will follow Binary Negative Binomial Equation with constant
x_train.head()
# checking columns in Predictor-training DataFrame after adding constant
```

	math	General	Vocational	const
182	24	0	1	1.0
202	31	0	1	1.0
188	65	0	0	1.0
197	57	0	0	1.0
148	23	0	0	1.0

```
y_train.head()
# checking columns in Response-training Series
```

```
182     1
202     1
188     0
197     0
148    12
Name: daysabs, dtype: int64
```

pd.concat([x_train, y_train], axis=1)

```
df_train = pd.concat([x_train, y_train], axis=1)
# concatenating Predictor(X) & Response(Y) into single DataFrame
# so that relation can be established in formula
df_train.head()
# checking columns in concatenated training DataFrame
```

	math	General	Vocational	const	daysabs
182	24	0	1	1.0	1
202	31	0	1	1.0	1
188	65	0	0	1.0	0
197	57	0	0	1.0	0

NegativeBinomial.from_formula('Response ~ P1 + P2 + P3', data=DataSet).fit()

```
mod1 = ngb.from_formula('daysabs ~ math + General + Vocational', data=df_train).fit()
# establishing relation in formula
# creating model using NegativeBinomial.from_formula() method
```

```
Optimization terminated successfully.
         Current function value: 2.745232
         Iterations: 16
         Function evaluations: 23
         Gradient evaluations: 23
```

```
print(mod1.summary())
# printing model summary / NegativeBinomial Regression Results
```

```
                     NegativeBinomial Regression Results
==============================================================================
Dep. Variable:                daysabs   No. Observations:                  235
Model:               NegativeBinomial   Df Residuals:                      231
Method:                           MLE   Df Model:                            3
Date:                Thu, 29 Jun 2023   Pseudo R-squ.:                 0.04603
Time:                        10:10:07   Log-Likelihood:                -645.13
converged:                       True   LL-Null:                       -676.25
Covariance Type:            nonrobust   LLR p-value:                 1.942e-13
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.2312      0.151     14.812      0.000       1.936       2.526
math          -0.0057      0.003     -2.058      0.040      -0.011      -0.000
General        0.4778      0.209      2.282      0.023       0.067       0.888
Vocational    -1.0546      0.164     -6.418      0.000      -1.377      -0.733
alpha          0.9057      0.109      8.328      0.000       0.693       1.119
==============================================================================
```

Establishing Null Hypothesis `Hₒ`

- `Hₒ` : maths(predictor) does not influence the daysabs(response)
- `Hᴀ` : maths(predictor) influences the daysabs(response)

    - for maths- `P-Value 0.040` < `0.05`, so we reject the `H₀`, means maths score will impact the daysabs

interpreting coefficients

- sign of coefficient being -ve means, response will decrease if predictor increases.
    - for maths score, coefficient is -ve means, if maths score increases, daysabs decrease
- impact of `days of abscence in General` > `days of abscence in Academic`
- impact of `days of abscence in vocational` < `days of abscence in Academic`
- means highest absent days in General, lowest absent days in Vocational

sm.add_constant(x_test, prepend=False)

```python
x_test = sm.add_constant(x_test, prepend=False)
# adding constant to match equation
# if we do not add constant, model will follow Negative Binomial Equation but without a constant
# if we add constant, model will follow Binary Negative Binomial Equation with constant
x_test.head()
# checking columns in Predictor-testing DataFrame after adding constant
```