

```
!jupyter nbconvert --to HTML "Assignment #3.ipynb"
```

```
[NbConvertApp] WARNING | pattern u'Assignment #3.ipynb' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.
```

```
WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.
```

Options

```
-----
```

Arguments that take values are actually convenience aliases to full Configurables, whose aliases are listed on the help line. For more information on full configurables, see '--help-all'.

```
--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and :
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of e;
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
```

✓ 1s completed at 1:50 AM



```

Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
Default: u''
The URL prefix for reveal.js (version 3.x). This defaults to the reveal CI
but can be any url pointing to a copy of reveal.js.

```

Question: 1

Part A

Is linear with respect to ? Is linear with respect to ?

Answer:

The value varies with the increase in the measurement.

Part B

Answer

If X is a normal random variable with mean μ and variance σ^2 , i.e,

$X \sim N(\mu, \sigma^2)$,

then $f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$

$F_X(x) = P(X \leq x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$

italicized text $P(a < X \leq b) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)$

Part C

Answer

we define the likelihood function by reversing the roles of the data vector x and the (distribution) parameter vector θ in $f(x|\theta)$, i.e.,

$L(\theta; x) = f(x|\theta)$

In MLE, we can assume that we have a likelihood function $L(\theta; x)$, where θ is the distribution parameter vector and x is the set of observations. We are interested in finding the value of θ that maximizes the likelihood with given observations (values of x).

The mathematical problem at hand becomes simpler if we assume that the observations (x_i) are independent and identically distributed random variables drawn from a Probability Distribution, f_0 (where f_0 = Normal Distribution) This reduces the Likelihood function to:

$$L(\theta; x) = f_0(x_1, x_2, x_3, \dots, x_n | \theta) \\ = f_0(x_1 | \theta) \cdot f_0(x_2 | \theta) \cdot f_0(x_3 | \theta) \cdot \dots \cdot f_0(x_n | \theta)$$

Part D

Answer

The least-squares estimator (LSE) is a special case of a maximum-likelihood estimator (MLE). The special case is that the probability distribution used for the likelihood is the normal distribution.

Question: 2

Part: A

Answer

$$p^*(X) = e^{-6+0.05X_1+X_2} (1 + e^{-6+0.05X_1+X_2}) = 0.3775.$$

Part: B

Answer

$$e^{-6+0.05X_1+3.5} (1 + e^{-6+0.05X_1+3.5}) = 0.5,$$

which comes in equivalence

$$e^{-6+0.05X_1+3.5} = 1.$$

Part: C

Answer:

$$X_1 = 2.50.05 = 50.$$

Question: 3

```
# Importing necessary libraries for training and data wrangling
import pandas as pd
import numpy as np
import sklearn as sk
import matplotlib.pyplot as plt
```

```
# Uploading the data to google colab storage
dataset = pd.read_csv('/content/Weekly.csv')
# dropping unnecessary columns
dataset.drop(columns='Unnamed: 0', inplace=True)
dataset
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	Down
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	Down
2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	Up
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	Up
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	Up
...
1084	2010	-0.861	0.043	-2.173	3.599	0.015	3.205160	2.969	Up
1085	2010	2.969	-0.861	0.043	-2.173	3.599	4.242568	1.281	Up
1086	2010	1.281	2.969	-0.861	0.043	-2.173	4.835082	0.283	Up
1087	2010	0.283	1.281	2.969	-0.861	0.043	4.454044	1.034	Up
1088	2010	1.034	0.283	1.281	2.969	-0.861	2.707105	0.069	Up

1089 rows × 9 columns

Task: 01 Splitting Training and Testing Dataset

```
# splitting the data into training and testing
X_train = dataset.head(985)
X_test = dataset.tail(1089-985)
print(X_train, X_test)
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	Down
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	Down

2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	Up
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	Up
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	Up
...
980	2008	12.026	-8.389	-6.198	-3.898	10.491	5.841565	-2.251	Down
981	2008	-2.251	12.026	-8.389	-6.198	-3.898	6.093950	0.418	Up
982	2008	0.418	-2.251	12.026	-8.389	-6.198	5.932454	0.926	Up
983	2008	0.926	0.418	-2.251	12.026	-8.389	5.855972	-1.698	Down
984	2008	-1.698	0.926	0.418	-2.251	12.026	3.087105	6.760	Up
[985 rows x 9 columns]									
	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volum		
985	2009	6.760	-1.698	0.926	0.418	-2.251	3.793110	-4.448	Down
986	2009	-4.448	6.760	-1.698	0.926	0.418	5.043904	-4.518	Down
987	2009	-4.518	-4.448	6.760	-1.698	0.926	5.948758	-2.137	Down
988	2009	-2.137	-4.518	-4.448	6.760	-1.698	6.129763	-0.730	Down
989	2009	-0.730	-2.137	-4.518	-4.448	6.760	5.602004	5.173	Up
...
1084	2010	-0.861	0.043	-2.173	3.599	0.015	3.205160	2.969	Up
1085	2010	2.969	-0.861	0.043	-2.173	3.599	4.242568	1.281	Up
1086	2010	1.281	2.969	-0.861	0.043	-2.173	4.835082	0.283	Up
1087	2010	0.283	1.281	2.969	-0.861	0.043	4.454044	1.034	Up
1088	2010	1.034	0.283	1.281	2.969	-0.861	2.707105	0.069	Up
[104 rows x 9 columns]									

```
# Dropping the columns other than features
```

```
xtrain = X_train.drop(columns=['Year', 'Today', 'Direction'])
```

```
ytrain = X_train['Direction']
```

```
# Dropping the columns other than features
```

```
xtest = X_test.drop(columns=['Year', 'Today', 'Direction'])
```

```
ytest = X_test['Direction']
```

Description of data.

```
# Description of the Dataset features
```

```
print("Training Features\n\n", xtrain.describe())
```

```
print("Training Target\n\n", ytrain.describe())
```

```
print("Testing Features\n\n", xtest.describe())
```

```
print("Testing Target\n\n", ytest.describe())
```

Training Features

	Lag1	Lag2	Lag3	Lag4	Lag5	Volum
count	985.000000	985.000000	985.000000	985.000000	985.000000	985.000000
mean	0.124501	0.127820	0.122884	0.122227	0.120976	1.205973
std	2.269346	2.269069	2.272617	2.272625	2.274273	1.258204
min	-18.195000	-18.195000	-18.195000	-18.195000	-18.195000	0.087465
25%	-1.154000	-1.147000	-1.154000	-1.154000	-1.154000	0.307337
50%	0.231000	0.234000	0.231000	0.230000	0.230000	0.804848

```

50%      0.231000      0.237000      0.231000      0.230000      0.230000      0.004040
75%      1.334000      1.337000      1.337000      1.337000      1.337000      1.515846
max      12.026000      12.026000      12.026000      12.026000      12.026000      9.328214
Training Target

count      985
unique      2
top         Up
freq        544
Name: Direction, dtype: object
Testing Features

```

```

              Lag1      Lag2      Lag3      Lag4      Lag5      Volume
count  104.000000  104.000000  104.000000  104.000000  104.000000  104.000000
mean    0.397635    0.371365    0.377548    0.369250    0.319058    5.066105
std     3.068538    3.074725    3.075192    3.073895    3.073650    1.147652
min    -7.035000   -7.035000   -7.035000   -7.035000   -7.035000    2.390427
25%    -1.060750   -1.214250   -1.214250   -1.214250   -1.323250    4.234228
50%     0.522000    0.461500    0.522000    0.461500    0.437000    4.850717
75%     2.239250    2.239250    2.239250    2.239250    2.204250    5.793837
max    10.707000   10.707000   10.707000   10.707000   10.707000    7.963276
Testing Target

```

```

count      104
unique      2
top         Up
freq        61
Name: Direction, dtype: object

```

Task: 02 SGD Procedure Usage

Same prebuilt Stochastic Gradient Descent will be used for logistic regression fitting as well. As the model can be controlled for overfitting with SGD.

Task: 03 Model learning with and without SGD

Model learning with SGD

```

# Importing libraries for data visualizations and metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics

# Importing SDG Classifier
from sklearn.linear_model import SGDClassifier

```

```

# Setting up the model
with_SGD_Model = SGDClassifier(loss="log", penalty="l2", max_iter=5)
# Fitting the SGD Classifier model
with_SGD_Model.fit(xtrain, ytrain)

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_stochastic_gradient_descent.py:100: ConvergenceWarning:
SGDClassifier(loss='log', max_iter=5)

# Predicting the SGD Model
with_SGD_Model_predictions = with_SGD_Model.predict(xtest)
with_SGD_Model_predictions

array(['Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
      'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down'],
      dtype='<U4')

# Using score function to get accuracy of model
with_SGD_score = with_SGD_Model.score(xtest, ytest)
print(with_SGD_score)

```

```
0.41346153846153844
```

Learning without SGD

```

# Importing the Logistic Regression Model
from sklearn.linear_model import LogisticRegression
without_SGD_Model = LogisticRegression()

# Training the model
without_SGD_Model.fit(xtrain, ytrain)

LogisticRegression()

# Use score method to get accuracy of model
without_SGD_score = without_SGD_Model.score(xtest, ytest)
print(without_SGD_score)

```

```
print(without_SGD_score,
```

```
0.46153846153846156
```

```
# Predicting the model for test dataset
```

```
without_SGD_Model_predictions = without_SGD_Model.predict(xtest)
```

```
without_SGD_Model_predictions
```

```
array(['Down', 'Up', 'Down', 'Down', 'Down', 'Down', 'Up', 'Down', 'Down',
       'Down', 'Down', 'Up', 'Down', 'Down', 'Down', 'Down', 'Down',
       'Down', 'Down', 'Up', 'Down', 'Down', 'Down', 'Up', 'Down', 'Down',
       'Up', 'Down', 'Down', 'Up', 'Up', 'Down', 'Down', 'Down', 'Down',
       'Down', 'Down', 'Down', 'Up', 'Down', 'Down', 'Up', 'Up', 'Down',
       'Down', 'Down', 'Up', 'Up', 'Down', 'Down', 'Down', 'Down', 'Up',
       'Down', 'Up', 'Down', 'Down', 'Down', 'Up', 'Up', 'Up', 'Down',
       'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
       'Down', 'Down', 'Up', 'Down', 'Up', 'Down', 'Up', 'Up', 'Down',
       'Down', 'Up', 'Down', 'Up', 'Down', 'Up', 'Down', 'Down', 'Down',
       'Up', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down', 'Down',
       'Down', 'Up', 'Down', 'Up', 'Down', 'Up', 'Down', 'Down'],
      dtype=object)
```

Task: 04 Cofusion Matrix Function

```
# Defining the confusion matrix function
```

```
def conf_matrix(predicted_values, actual_values):
```

```
    cm = metrics.confusion_matrix(actual_values, predicted_values)
```

```
    print(cm)
```

```
    plt.figure(figsize=(10,10))
```

```
    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blue
```

```
    plt.ylabel('Actual Values')
```

```
    plt.xlabel('Predicted Values')
```

```
    plt.title("Confusion Matrix for Predicted and Actual Values", size = 15)
```

Task: 05

A: Model without regularization

```
# plot confusion matrix for model without Gradient Descent
```

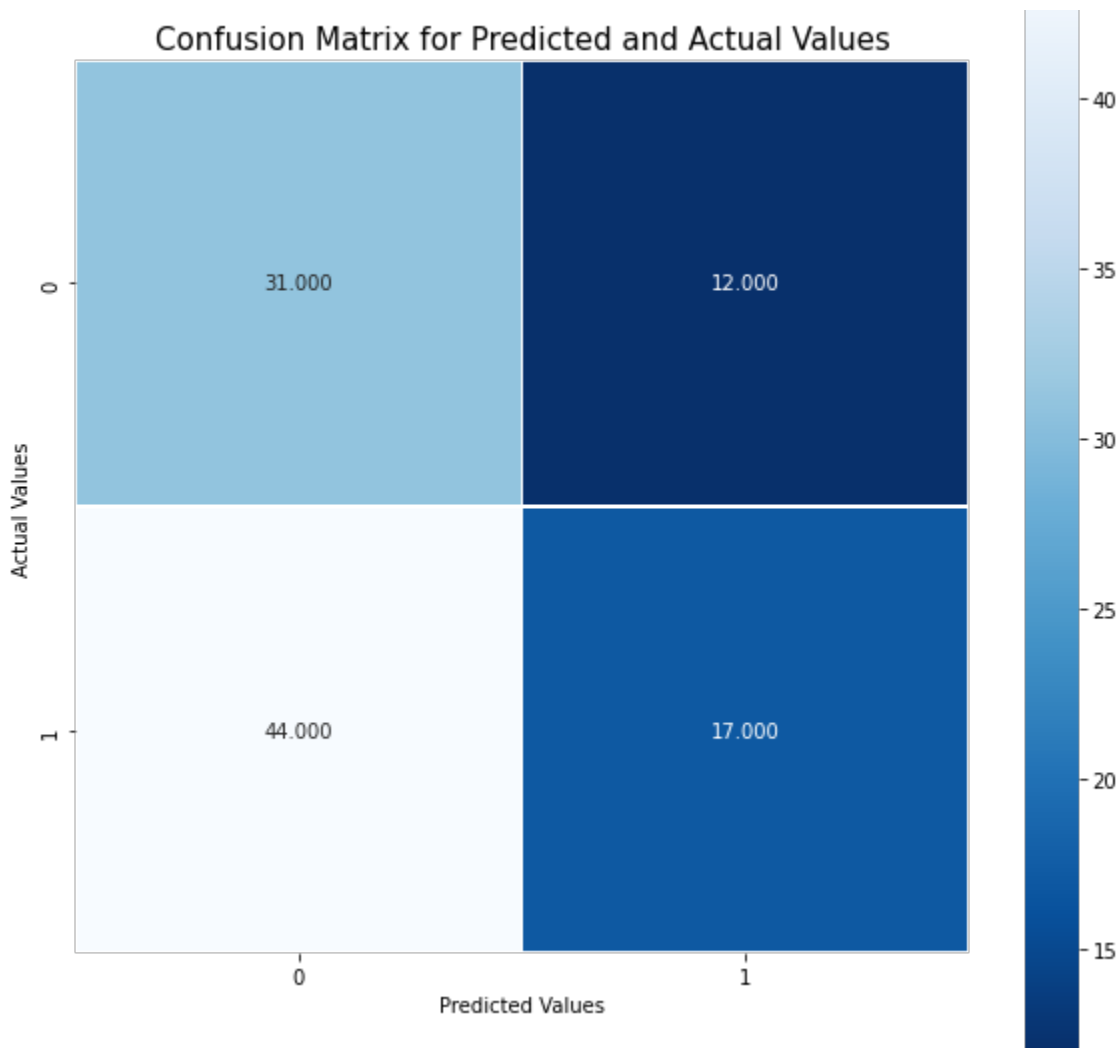
```
conf_matrix(without_SGD_Model_predictions, ytest)
```

```
print("Accuracy:", without_SGD_score)
```

```
[[31 12]
```

```
 [44 17]]
```

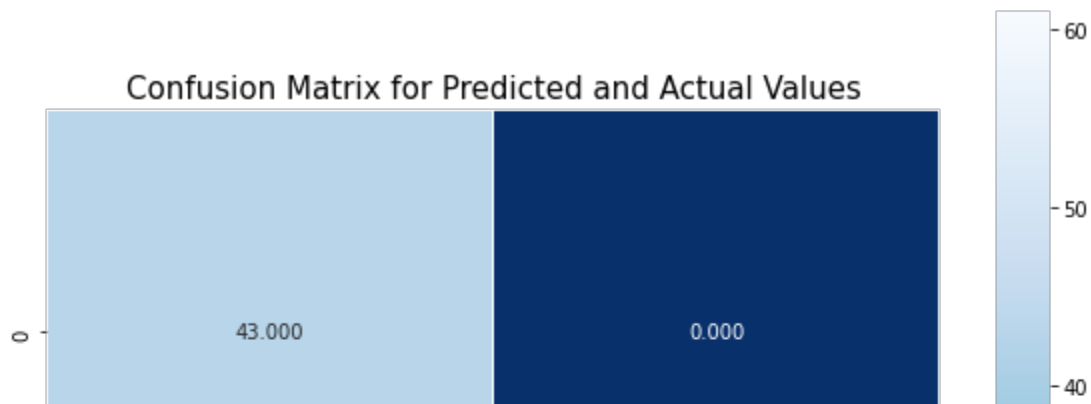
```
Accuracy: 0.46153846153846156
```

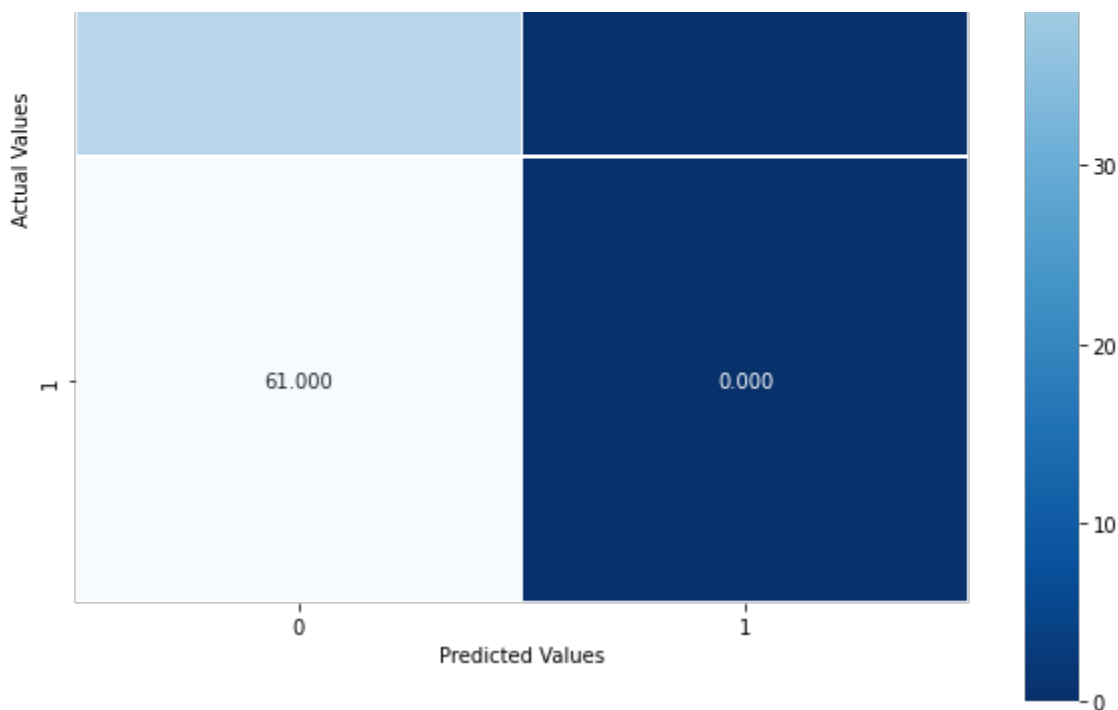



B: Model with Gradient Descent

```
# plot confusion matrix for model without Gradient Descent
conf_matrix(with_SGD_Model_predictions, ytest)
print("Accuracy:", with_SGD_score)
```

```
[[43  0]
 [61  0]]
Accuracy: 0.41346153846153844
```





Task: 06 Plot Losses for two models

Log Loss

```
# Importing the Logistic Regression method
from sklearn.metrics import log_loss
```

Predicting probability values of Model without SGD

Training

```
# Without SGD Model
without_SGD_training_prob = without_SGD_Model.predict_proba(xtrain)
without_SGD_training_prob

array([[0.36889318, 0.63110682],
       [0.38253764, 0.61746236],
       [0.36513753, 0.63486247],
       ...,
       [0.50409034, 0.49590966],
       [0.56291625, 0.43708375],
       [0.54653507, 0.45346493]])
```

Testing

```

# Without SGD Model
without_SGD_testing_prob = without_SGD_Model.predict_proba(xtest)
without_SGD_testing_prob

array([[0.60969555, 0.39030445],
       [0.39135576, 0.60864424],
       [0.55077637, 0.44922363],
       [0.58955137, 0.41044863],
       [0.56605545, 0.43394455],
       [0.55909251, 0.44090749],
       [0.36052772, 0.63947228],
       [0.50105897, 0.49894103],
       [0.60498057, 0.39501943],
       [0.51286899, 0.48713101],
       [0.70077369, 0.29922631],
       [0.3756385 , 0.6243615 ],
       [0.59346907, 0.40653093],
       [0.5619091 , 0.4380909 ],
       [0.67504543, 0.32495457],
       [0.64592206, 0.35407794],
       [0.63840998, 0.36159002],
       [0.6235713 , 0.3764287 ],
       [0.68820335, 0.31179665],
       [0.43200828, 0.56799172],
       [0.64644471, 0.35355529],
       [0.63304205, 0.36695795],
       [0.55691138, 0.44308862],
       [0.48103917, 0.51896083],
       [0.5242452 , 0.4757548 ],
       [0.60951101, 0.39048899],
       [0.49134885, 0.50865115],
       [0.50340038, 0.49659962],
       [0.61548942, 0.38451058],
       [0.48651852, 0.51348148],
       [0.49107589, 0.50892411],
       [0.63589694, 0.36410306],
       [0.60083843, 0.39916157],
       [0.61482473, 0.38517527],
       [0.54851993, 0.45148007],
       [0.53857208, 0.46142792],
       [0.59664974, 0.40335026],
       [0.57880824, 0.42119176],
       [0.47147993, 0.52852007],
       [0.54782649, 0.45217351],
       [0.63864513, 0.36135487],
       [0.49467411, 0.50532589],
       [0.48501999, 0.51498001],
       [0.52221831, 0.47778169],
       [0.67370715, 0.32629285],
       [0.50373138, 0.49626862],
       [0.45511002, 0.54488998],
       [0.48715036, 0.51284964],
       [0.58499378, 0.41500622]

```

```
[0.50155578, 0.41300022],
[0.51538029, 0.48461971],
[0.54110895, 0.45889105],
[0.53269266, 0.46730734],
[0.44205984, 0.55794016],
[0.56989433, 0.43010567],
[0.48207229, 0.51792771],
[0.51449091, 0.48550909],
[0.55656634, 0.44343366],
[0.54180788, 0.45819212],
```

Predicting probability values of Model with SGD

Training

```
# With SGD Model
with_SGD_training_prob = with_SGD_Model.predict_proba(xtrain)
with_SGD_training_prob

array([[8.88178420e-16, 1.00000000e+00],
       [1.22225623e-02, 9.87777438e-01],
       [4.95159469e-14, 1.00000000e+00],
       ...,
       [1.00000000e+00, 3.42150326e-42],
       [1.00000000e+00, 8.49039198e-11],
       [1.00000000e+00, 1.34650909e-33]])
```

Testing

```
# With SGD Model
with_SGD_testing_prob = with_SGD_Model.predict_proba(xtest)
with_SGD_testing_prob

array([[1.00000000e+00, 3.68306938e-33],
       [9.52008260e-01, 4.79917397e-02],
       [1.00000000e+00, 2.16917043e-42],
       [1.00000000e+00, 1.16467875e-27],
       [1.00000000e+00, 7.61546340e-41],
       [1.00000000e+00, 4.42705846e-35],
       [9.91381019e-01, 8.61898067e-03],
       [1.00000000e+00, 3.73839929e-33],
       [1.00000000e+00, 1.23848151e-37],
       [1.00000000e+00, 6.35814407e-39],
       [1.00000000e+00, 3.08497881e-63],
       [9.98183752e-01, 1.81624832e-03],
       [1.00000000e+00, 1.48142195e-52],
       [1.00000000e+00, 2.14418690e-13],
       [1.00000000e+00, 6.01477038e-57],
       [1.00000000e+00, 9.57530275e-42],
       [1.00000000e+00, 5.74375291e-47],
```

```
[1.00000000e+00, 3.85492839e-43],  
[1.00000000e+00, 2.68135354e-54],  
[1.00000000e+00, 3.53183086e-16],  
[1.00000000e+00, 2.08650089e-51],  
[1.00000000e+00, 6.94020048e-32],  
[1.00000000e+00, 9.87578289e-40],  
[1.00000000e+00, 5.23567673e-17],  
[1.00000000e+00, 4.14187054e-23],  
[1.00000000e+00, 5.54129057e-39],  
[1.00000000e+00, 4.35841365e-18],  
[1.00000000e+00, 3.37168450e-27],  
[1.00000000e+00, 5.34524433e-35],  
[1.00000000e+00, 1.22265284e-17],  
[1.00000000e+00, 2.33307491e-25],  
[1.00000000e+00, 4.69639573e-37],  
[1.00000000e+00, 2.42814998e-35],  
[1.00000000e+00, 2.10961811e-40],  
[1.00000000e+00, 1.10714259e-26],  
[1.00000000e+00, 4.42043230e-32],  
[1.00000000e+00, 2.37759768e-33],  
[1.00000000e+00, 3.35972243e-34],  
[1.00000000e+00, 4.12697617e-21],  
[1.00000000e+00, 1.36017586e-28],  
[1.00000000e+00, 1.57265147e-37],  
[1.00000000e+00, 3.41503199e-21],  
[1.00000000e+00, 4.65918305e-24],  
[1.00000000e+00, 2.49116778e-23],  
[1.00000000e+00, 8.09650015e-50],  
[1.00000000e+00, 1.11036871e-17],  
[1.00000000e+00, 1.29930793e-19],  
[1.00000000e+00, 4.22741918e-11],  
[1.00000000e+00, 2.00016141e-31],  
[1.00000000e+00, 5.31164417e-23],  
[1.00000000e+00, 1.50161675e-30],  
[1.00000000e+00, 9.30332151e-20],  
[9.9999294e-01, 7.05634191e-07],  
[1.00000000e+00, 4.47761401e-32],  
[1.00000000e+00, 4.72652226e-13],  
[1.00000000e+00, 1.81453078e-31],  
[1.00000000e+00, 4.11643419e-28],  
[1.00000000e+00, 3.84331706e-29],
```

Model without SGD

Training Loss

```
# Calculating logistic loss value  
without_SGD_training_loss = log_loss(ytrain, without_SGD_training_prob)  
without_SGD_training_loss
```

0.6813875468279803

Testing Loss

```
# Calculating logistic loss value
without_SGD_testing_loss = log_loss(ytest, without_SGD_testing_prob)
without_SGD_testing_loss

0.7163296252177602
```

Model with SGD

Training Loss

```
# Calculating logistic loss value
with_SGD_training_loss = log_loss(ytrain, with_SGD_training_prob)
with_SGD_training_loss

7.731104088792359
```

Testing Loss

```
# Calculating logistic loss value
with_SGD_testing_loss = log_loss(ytest, with_SGD_testing_prob)
with_SGD_testing_loss

18.335893026528097
```

Task: 06 Part: A

The testing loss for the model trained without the SGD, is 0.7163. Meanwhile, the training loss for the same model is 0.6814. This indicates that the model is not showing overfitting while training or testing.

The testing loss for the model trained with the SGD, is 11.5837. Furthermore, the testing loss for the same model is 12.0169. This indicates that the model that was trained using SGD, is not showing overfitting.

Task: 06 Part: B

While the model is being trained with and without the usage of regularization, not so significant occurrence has been considered. Thus, this concludes in the negation of overfitting of the model.

```
# Training Loss
print(without_SGD_training_loss)
# Testing Loss
print(without_SGD_testing_loss)

0.6813875468279803
0.7163296252177602
```

Task: 06 Part: C

Due to the increase in the regularization, the model displayed a comparatively large measure of loss error.

```
# Training Loss
print(with_SGD_training_loss)
# Testing Loss
print(with_SGD_testing_loss)

7.731104088792359
18.335893026528097
```

Task: 07

The comparison between the logistic regression models trained with and without the regularization and SGD, shows the variance in the accuracy and loss values. Due to the underfitting of the model infused with the regularization and SGD, the value of logistic loss increases.

Task: 08

The increase in the learning rate might cause to skip some instances while training the model. Thus, a moderate value of learning rate parameter has to be set to achieve required accuracy. Whereas, the regularization helps in controlling the overfitting of the model while training, it does

have a critical impact as well. Due to the increase in the regularization co-efficient, the model could go into underfitting phenomenon and cause not to meet the required amount of accuracy.

```
#Printing the accuracy values for both models
print("Accuracy of model without SGD: ", without_SGD_score)
print("Accuracy of model with SGD: ", with_SGD_score)
```

```
Accuracy of model without SGD:  0.46153846153846156
Accuracy of model with SGD:    0.41346153846153844
```

Task: 09

```
# Importing the f1_score method
from sklearn.metrics import f1_score
```

```
# Training probability values
with_SGD_training_prob
```

```
array([[8.88178420e-16, 1.00000000e+00],
       [1.22225623e-02, 9.87777438e-01],
       [4.95159469e-14, 1.00000000e+00],
       ...,
       [1.00000000e+00, 3.42150326e-42],
       [1.00000000e+00, 8.49039198e-11],
       [1.00000000e+00, 1.34650909e-33]])
```

Task: 10

```
# Separating the dataset for training
x_train_lag_data = xtrain[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5']]
x_train_lag_data
```

	Lag1	Lag2	Lag3	Lag4	Lag5
0	0.816	1.572	-3.936	-0.229	-3.484
1	-0.270	0.816	1.572	-3.936	-0.229
2	-2.576	-0.270	0.816	1.572	-3.936
3	3.514	-2.576	-0.270	0.816	1.572
4	0.712	3.514	-2.576	-0.270	0.816
...


```

980 12.026 -8.389 -6.198 -3.898 10.491
981 -2.251 12.026 -8.389 -6.198 -3.898
982 0.418 -2.251 12.026 -8.389 -6.198
983 0.926 0.418 -2.251 12.026 -8.389
984 -1.698 0.926 0.418 -2.251 12.026

```

985 rows × 5 columns

```

# Separating the dataset for testing
x_test_lag_data = xtest[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5']]
x_test_lag_data

```

	Lag1	Lag2	Lag3	Lag4	Lag5
985	6.760	-1.698	0.926	0.418	-2.251
986	-4.448	6.760	-1.698	0.926	0.418
987	-4.518	-4.448	6.760	-1.698	0.926
988	-2.137	-4.518	-4.448	6.760	-1.698
989	-0.730	-2.137	-4.518	-4.448	6.760
...
1084	-0.861	0.043	-2.173	3.599	0.015
1085	2.969	-0.861	0.043	-2.173	3.599
1086	1.281	2.969	-0.861	0.043	-2.173
1087	0.283	1.281	2.969	-0.861	0.043
1088	1.034	0.283	1.281	2.969	-0.861

104 rows × 5 columns

```

# Separating the dataset for for target values of training dataset
y_train_lag_data = ytrain
y_train_lag_data

```

```

0      Down
1      Down
2       Up
3       Up
4       Up
...
980    Down
981     Up
982     Up

```

```

983     Down
984     Up
Name: Direction, Length: 985, dtype: object

```

```
# Separating the dataset for target values of testing dataset
```

```
y_test_lag_data = ytest
```

```
y_test_lag_data
```

```

985     Down
986     Down
987     Down
988     Down
989     Up
...
1084    Up
1085    Up
1086    Up
1087    Up
1088    Up
Name: Direction, Length: 104, dtype: object

```

Fitting

Lag 1 Fitting

```

# Importing the Logistic Regression Model
from sklearn.linear_model import LogisticRegression
# Importing the Logistic Regression Model
lag1_Model = LogisticRegression()
lag1_Model.fit(x_train_lag_data[['Lag1']], y_train_lag_data)

# Use score method to get accuracy of model
lag1_Model_score = lag1_Model.score(x_test_lag_data[['Lag1']], y_test_lag_data)
print("lag1 Model Accuracy: ", lag1_Model_score)
# Predicting the values
lag1_Model_predictions = lag1_Model.predict(x_train_lag_data[['Lag1']])
print("lag1 Model Predictions: \n", lag1_Model_predictions)

```

```
lag1 Model Accuracy: 0.5673076923076923
```

```
lag1 Model Predictions:
```

```

['Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Down'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Down' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Down' 'Up' 'Up' 'Down' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Down' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'

```

Lag 2 Fitting

```
# Training the model
lag2_Model = LogisticRegression()
lag2_Model.fit(x_train_lag_data[['Lag2']], y_train_lag_data)
```

[illegible]

Lag 3 Fitting

[illegible]

3/1/22, 1:54 AM

[illegible]


```

'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Down' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Down' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'
'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up' 'Up'

```

Confusion Matrices

```

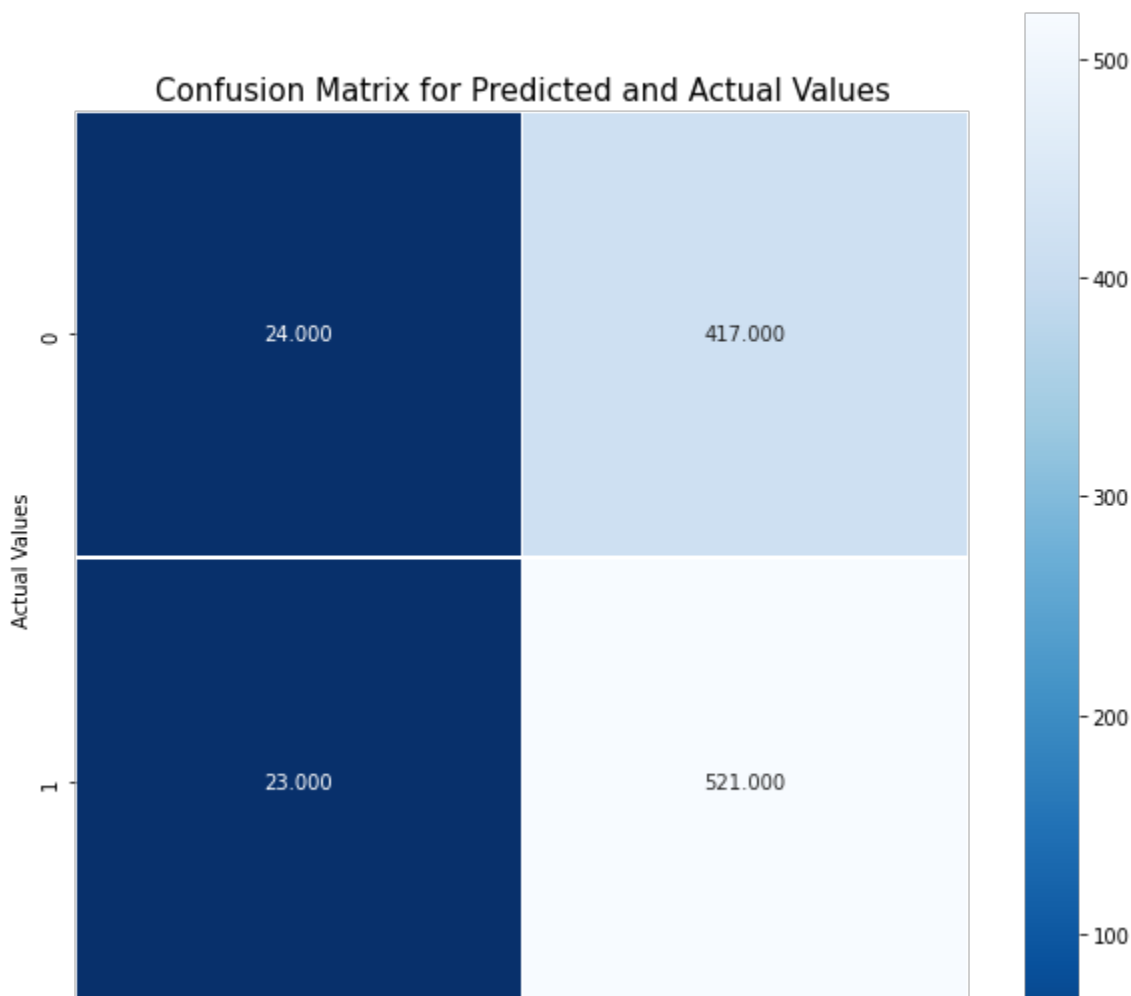
# plot confusion matrix for model
conf_matrix(lag1_Model_predictions, ytrain)
print("Accuracy:", lag1_Model_score)

```

```

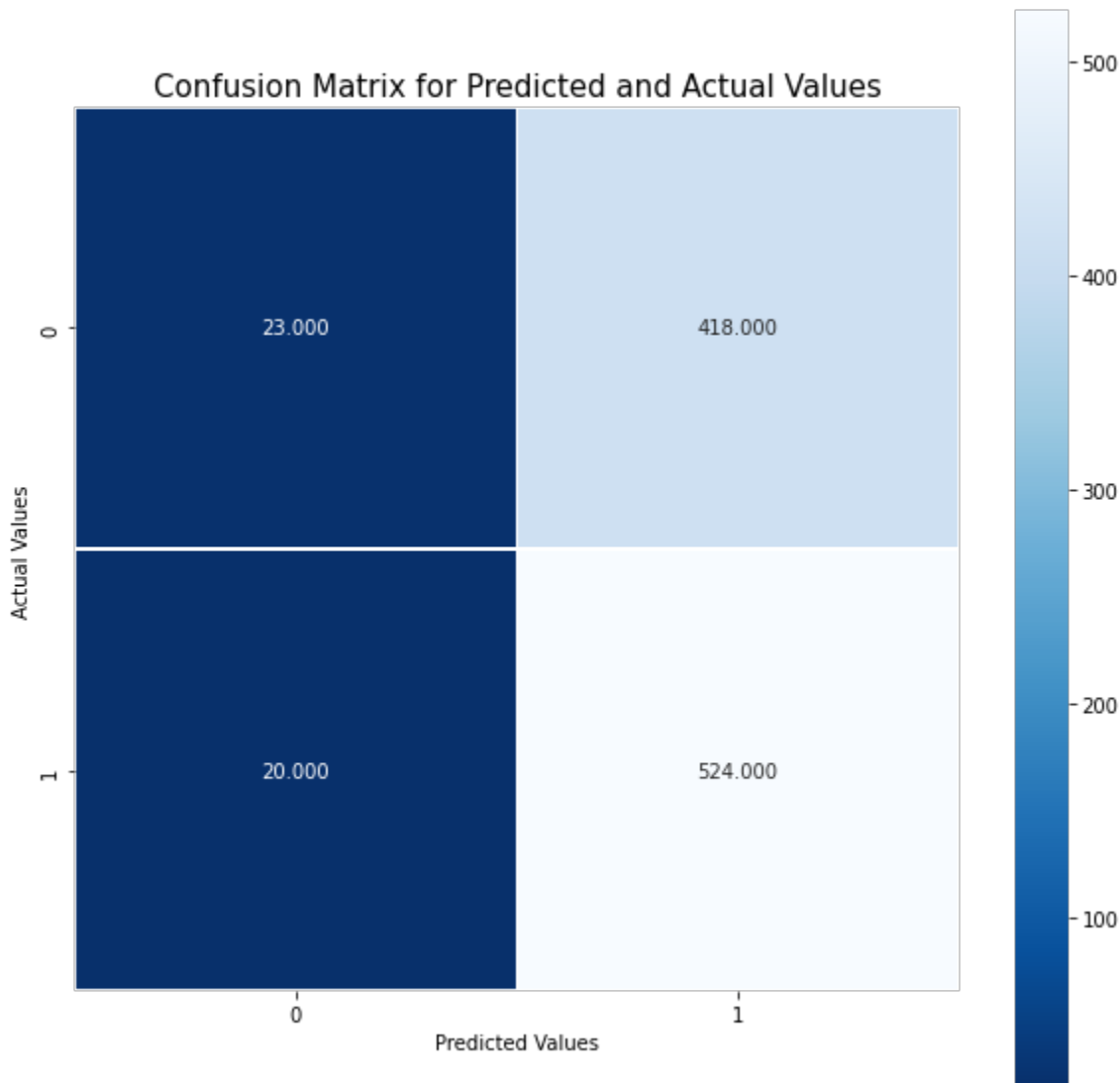
[[ 24 417]
 [ 23 521]]
Accuracy: 0.5673076923076923

```



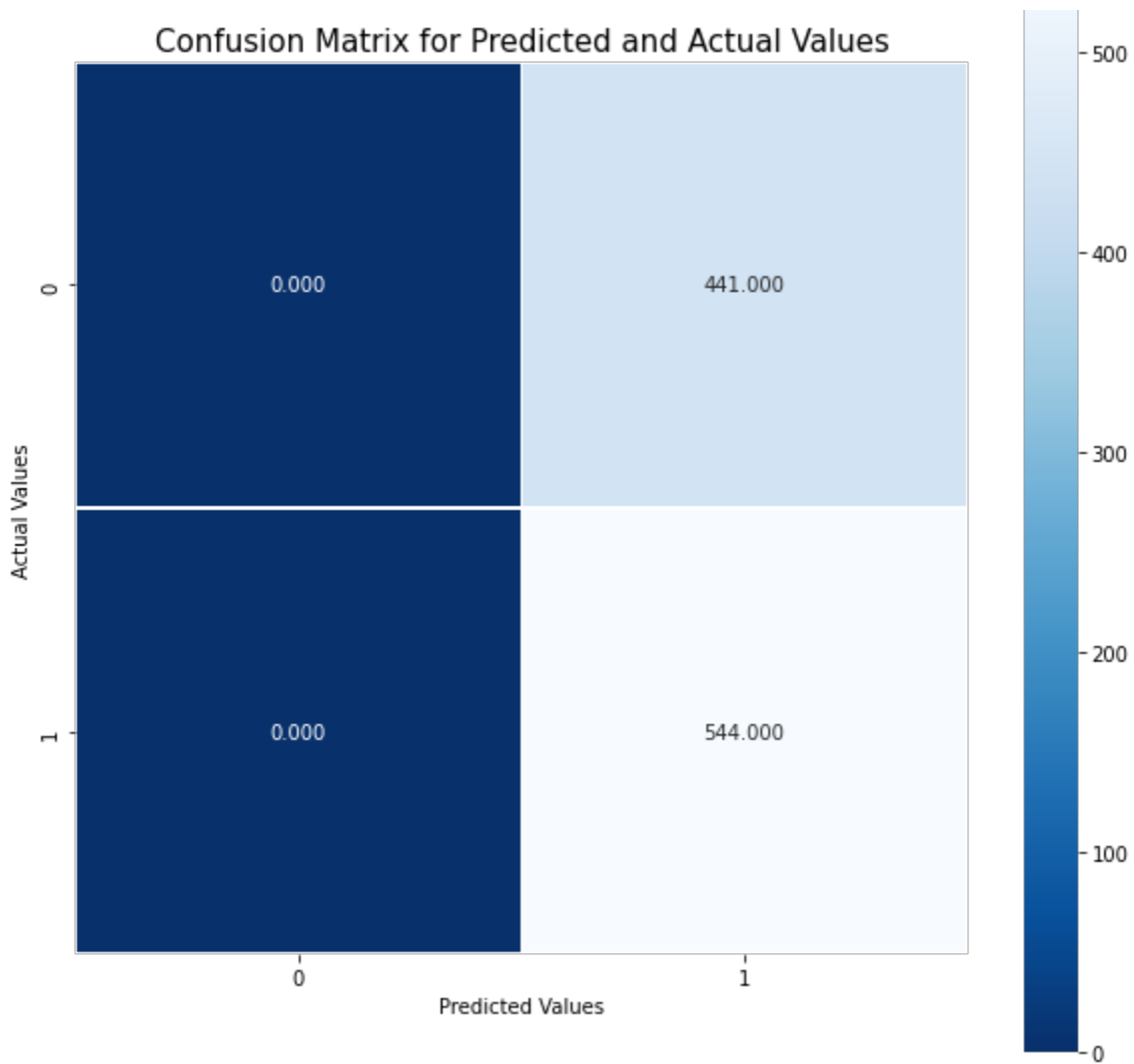
```
# plot confusion matrix for model
conf_matrix(lag2_Model_predictions, ytrain)
print("Accuracy:", lag2_Model_score)
```

```
[[ 23 418]
 [ 20 524]]
Accuracy: 0.5480769230769231
```



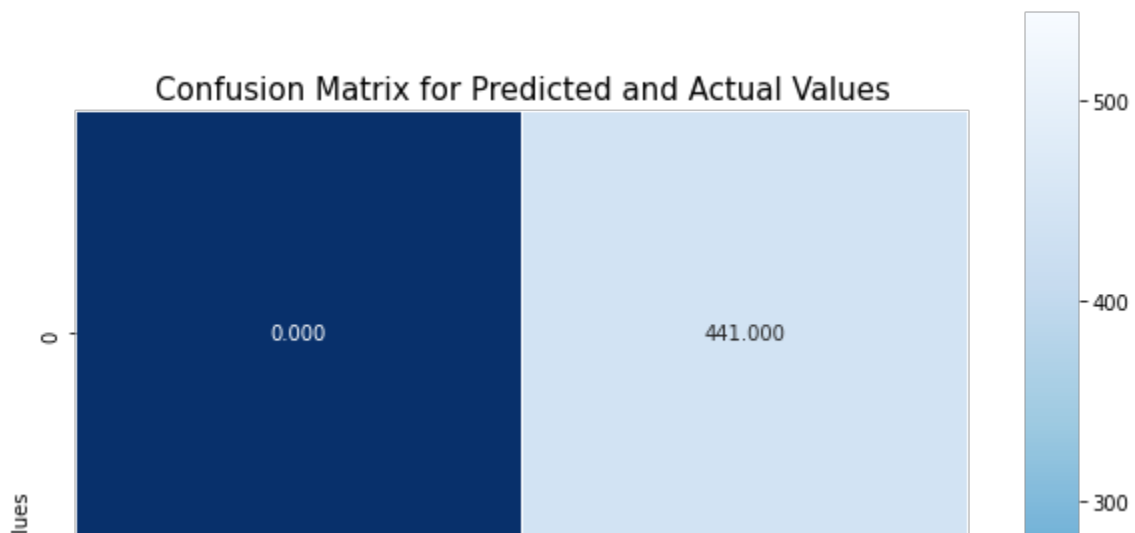
```
# plot confusion matrix for model
conf_matrix(lag3_Model_predictions, ytrain)
print("Accuracy:", lag3_Model_score)
```

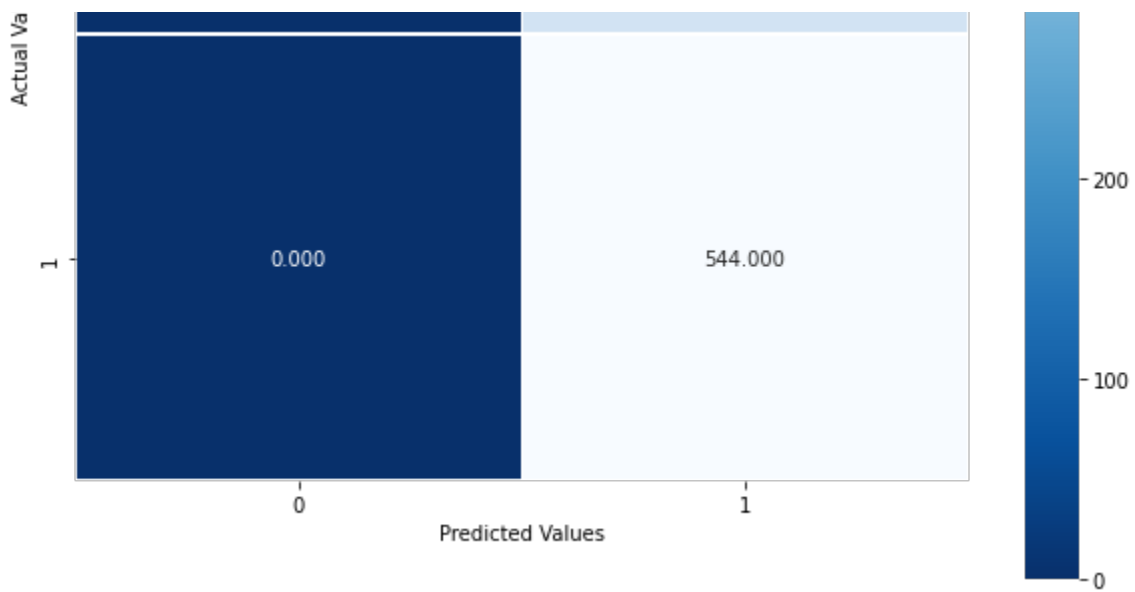
```
[[ 0 441]
 [ 0 544]]
Accuracy: 0.5865384615384616
```



```
# plot confusion matrix for model
conf_matrix(lag4_Model_predictions, ytrain)
print("Accuracy:", lag4_Model_score)
```

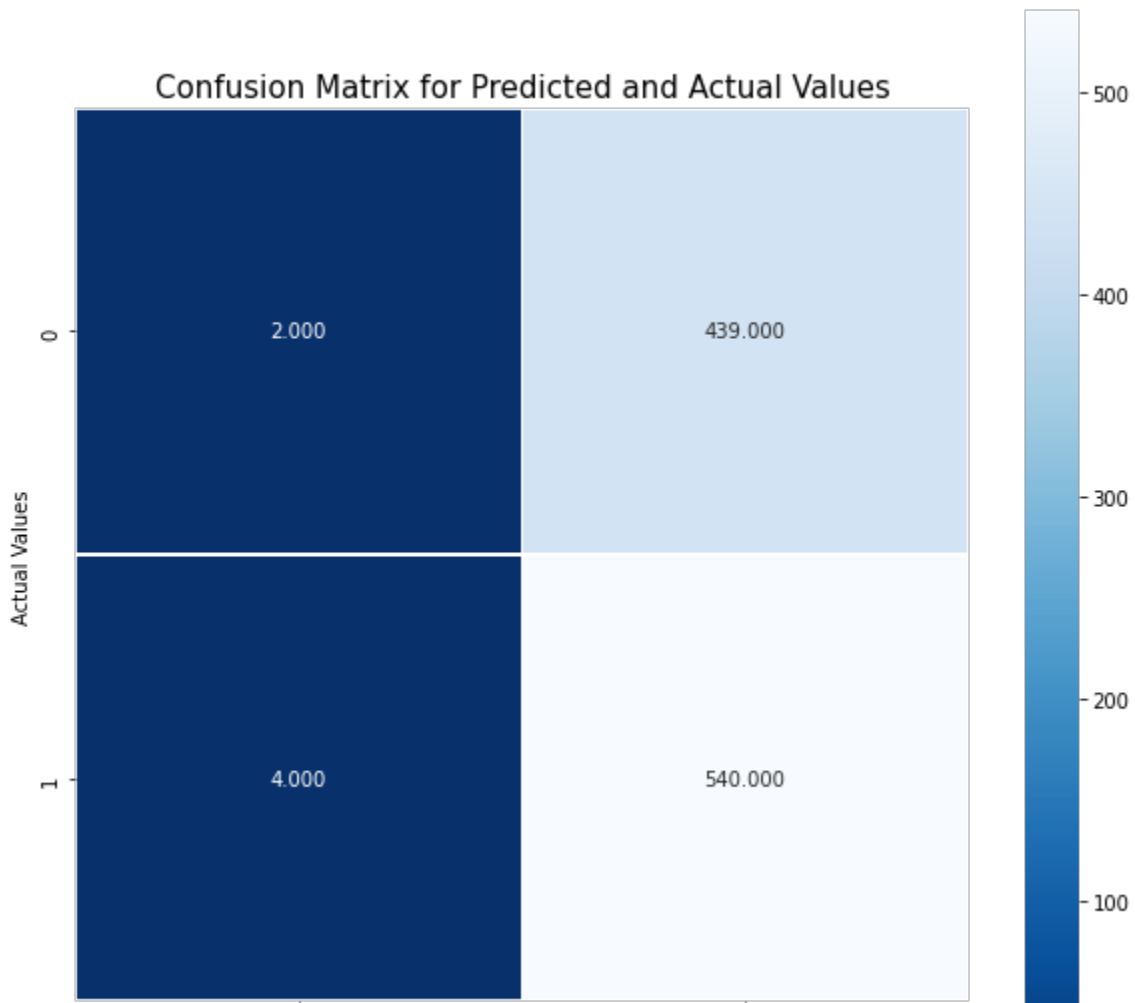
```
[[ 0 441]
 [ 0 544]]
Accuracy: 0.5865384615384616
```





```
# plot confusion matrix for model
conf_matrix(lag5_Model_predictions, ytrain)
print("Accuracy:", lag5_Model_score)
```

```
[[ 2 439]
 [ 4 540]]
Accuracy: 0.5096153846153846
```



0

Predicted Values

1



Task: 10 A

The model "lag4_Model" has the highest value of accuracy and indicates the best model trained so far for the "Weekly" Dataset.

Task: 10 B

Yes, the model "lag4_Model" has the larger value of accuracy and f1_score in terms of training dataset as well.

Question: 4

Part A

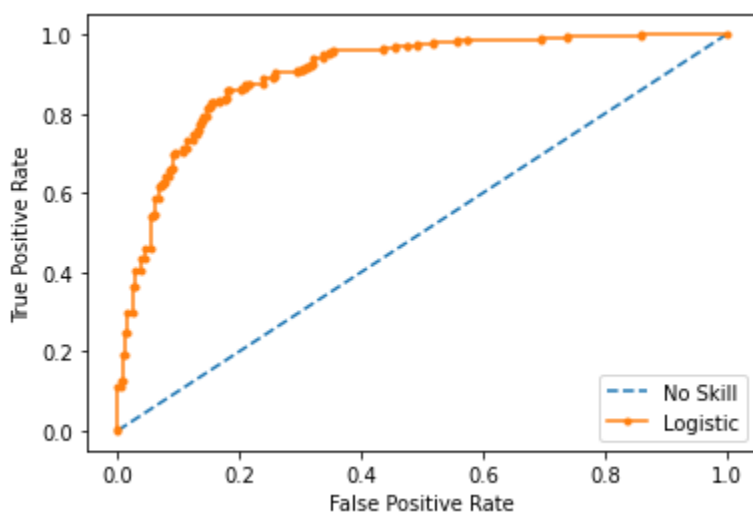
```
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
```

```

.. -----
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

```

No Skill: ROC AUC=0.500
 Logistic: ROC AUC=0.903



Part B

```

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)

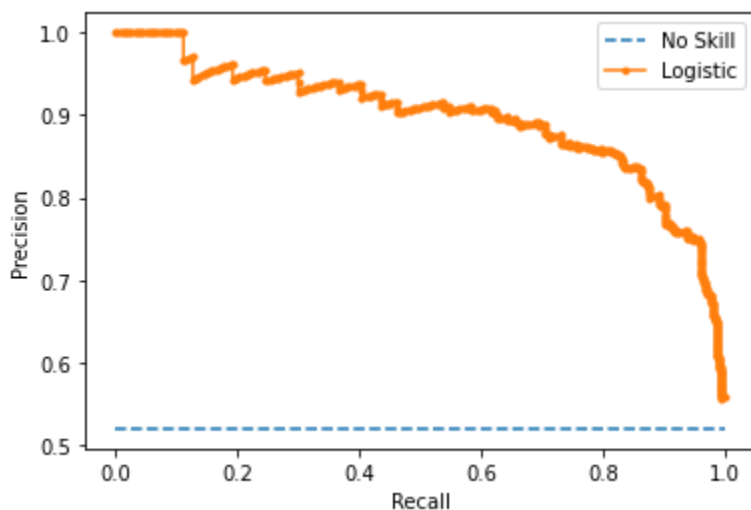
```

```

# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# predict class values
yhat = model.predict(testX)
lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)
lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
# plot the precision-recall curves
no_skill = len(testy[testy==1]) / len(testy)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

```

Logistic: f1=0.841 auc=0.898



Part C

```

# Importing the KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(xtrain, ytrain)

```

```
KNeighborsClassifier()
```

```

# Predicting values
v pred = classifier.predict(xtest)

```

```
# Reporting the results
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(ytest, y_pred))
print(classification_report(ytest, y_pred))
```

```
[[19 24]
 [28 33]]
```

	precision	recall	f1-score	support
Down	0.40	0.44	0.42	43
Up	0.58	0.54	0.56	61
accuracy			0.50	104
macro avg	0.49	0.49	0.49	104
weighted avg	0.51	0.50	0.50	104

```
error = []
```

```
# Calculating error for K values between 1 and 40
```

```
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn_acc=knn.fit(xtrain, ytrain)
    pred_i = knn.predict(xtest)
    error.append(np.mean(pred_i != ytest))
```

```
# Importing the LabelEncoder
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
label = le.fit_transform(ytest)
ytest=label
print(ytest)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
label = le.fit_transform(pred_i)
pred_i=label
print(pred_i)
```

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

```
# Plotting the KNN Graph
```

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
```

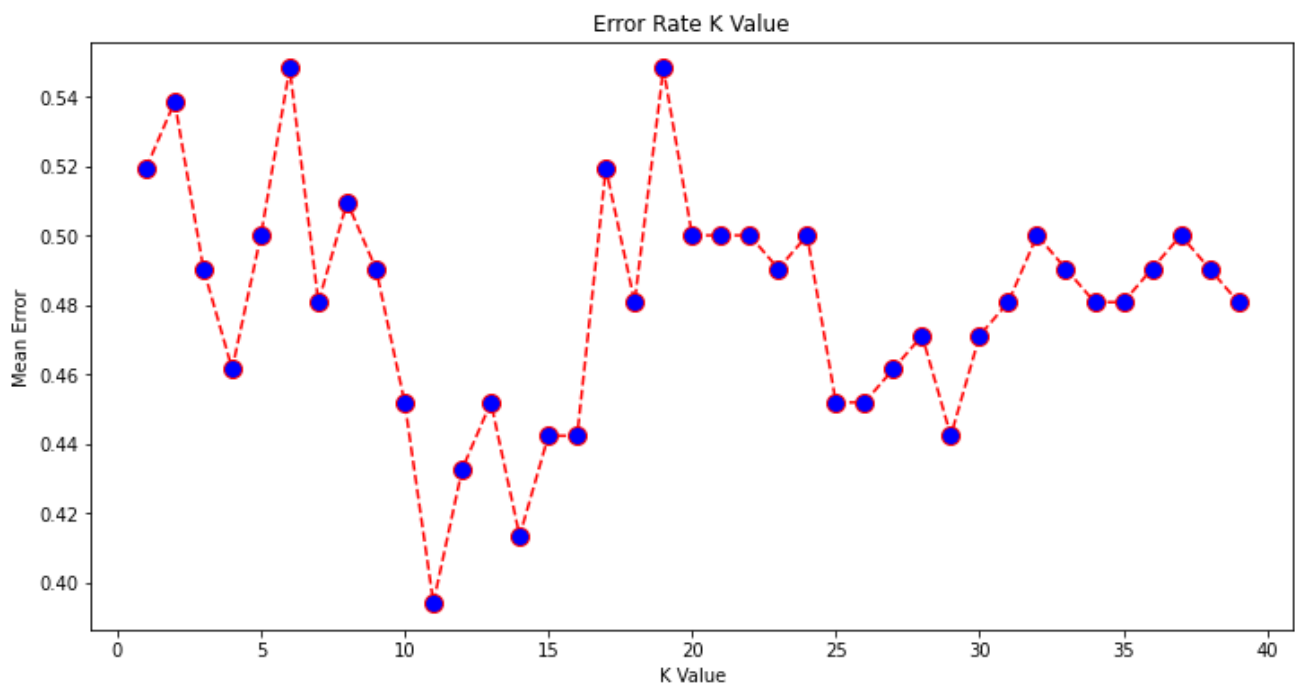


```

        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

```

```
Text(0, 0.5, 'Mean Error')
```



Accuracy Of Decision Tree Model

Part D

```

# Setting up the decision tree model
from sklearn import tree
decision_tree = tree.DecisionTreeClassifier()
decision_tree = decision_tree.fit(xtrain, ytrain)

# Predicting the values
res_pred = decision_tree.predict(xtest)
Decesion_tree_acc = decision_tree.score(xtest, res_pred)
print(Decesion_tree_acc)

```

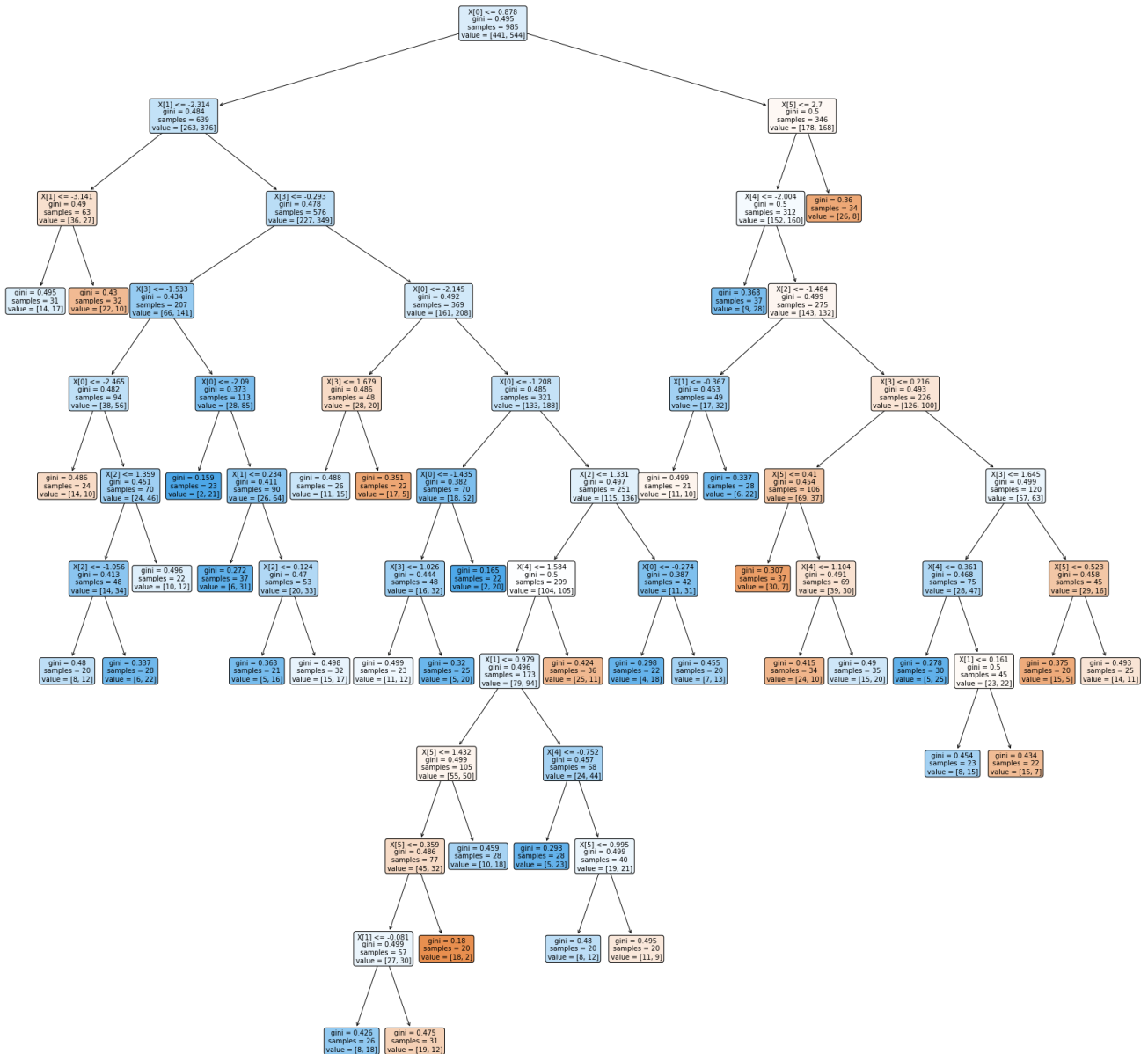
```
1.0
```

Plot The Graph of Decision Tree Model

```
# Plotting the decision tree graph
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=20, random_state=10)
clf = clf.fit(xtrain, ytrain)
```

```
# Importing the decision tree model
from sklearn import tree
plt.figure(figsize=(30, 30))
tree.plot_tree(clf, fontsize = 10, rounded = True, filled = True);
```



```
!jupyter nbconvert --to html Assignment3.ipynb
```

```
[NbConvertApp] WARNING | pattern u'Assignment3.ipynb' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.
```

```
WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.
```

Options

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

```
--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and :
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    ..  ..  ..  ..  ..
```

```
Choices: [1, 2, 3, 4]
The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
Default: ''
Directory to write output(s) to. Defaults to output to the directory of the
notebook. To recover previous default behaviour (outputting to the current
working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
Default: 'FilesWriter'
Writer class used to write the results of the conversion
--log-level=<Enum> (Application.log_level)
Default: 30
Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL')
Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
Default: u''
The URL prefix for reveal.js (version 3.x). This defaults to the reveal Cdn
but can be any url pointing to a copy of reveal.js.
```