Project Name : Predicting Remaining Useful Life of Battery.

Dataset Link = https://www.kaggle.com/datasets/ignaciovinuales/battery-remaining-useful-life-rul

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv(r"C:\Users\shubh\Downloads\Battery_RUL.csv")
```

In [3]:
```
df
```

Out[3]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)
0	1.0	2595.30	1151.488500	3.670	3.211	5460.001
1	2.0	7408.64	1172.512500	4.246	3.220	5508.992
2	3.0	7393.76	1112.992000	4.249	3.224	5508.993
3	4.0	7385.50	1080.320667	4.250	3.225	5502.016
4	6.0	65022.75	29813.487000	4.290	3.398	5480.992
...	...	...	...	...	...	...
15059	1108.0	770.44	179.523810	3.773	3.742	922.775
15060	1109.0	771.12	179.523810	3.773	3.744	915.512
15061	1110.0	769.12	179.357143	3.773	3.742	915.513
15062	1111.0	773.88	162.374667	3.763	3.839	539.375
15063	1112.0	677537.27	142740.640000	4.206	3.305	49680.004

15064 rows × 9 columns

In [4]:
```
#Cycle_Index                : The cycle number of the battery
#Discharge Time (s)         : Time taken to discharge (in seconds)
#Decrement 3.6-3.4V (s)     : Time spent in voltage drop from 3.6V to 3.4V
#Max. Voltage Dischar. (V) : Maximum voltage during discharge
#Min. Voltage Charg. (V)    : Minimum voltage during charging
#Time at 4.15V (s)              : Duration battery stayed at 4.15V
#Time constant current (s) : Time for constant current charging
#Charging time (s)              : Total charging time in seconds
#RUL                        : Remaining Useful Life (Target Variable)
```

In [5]:
```
df.isnull()
```

Out[5]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	T const curr
0	False	False	False	False	False	False	F
1	False	False	False	False	False	False	F

2	False	False	False	False	False	False	F
3	False	False	False	False	False	False	F
4	False	False	False	False	False	False	F
...	...	...	...	...	...	...	
15059	False	False	False	False	False	False	F
15060	False	False	False	False	False	False	F
15061	False	False	False	False	False	False	F
15062	False	False	False	False	False	False	F
15063	False	False	False	False	False	False	F

15064 rows × 9 columns

In [6]:
```
df.notnull()
```

Out[6]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	T const curr
0	True	True	True	True	True	True	
1	True	True	True	True	True	True	
2	True	True	True	True	True	True	
3	True	True	True	True	True	True	
4	True	True	True	True	True	True	
...	...	...	...	...	...	...	
15059	True	True	True	True	True	True	
15060	True	True	True	True	True	True	
15061	True	True	True	True	True	True	
15062	True	True	True	True	True	True	
15063	True	True	True	True	True	True	

15064 rows × 9 columns

In [7]:
```
df.describe()
```

Out[7]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Vol Charg
count	15064.000000	15064.000000	15064.000000	15064.000000	15064.00
mean	556.155005	4581.273960	1239.784672	3.908176	3.57
std	322.378480	33144.012077	15039.589269	0.091003	0.12
min	1.000000	8.690000	-397645.908000	3.043000	3.02
25%	271.000000	1169.310000	319.600000	3.846000	3.48

50%	560.000000	1557.250000	439.239471	3.906000	3.57
75%	833.000000	1908.000000	600.000000	3.972000	3.66
max	1134.000000	958320.370000	406703.768000	4.363000	4.37

In [8]:

```
df.describe
```

Out[8]:

```
<bound method NDFrame.describe of        Cycle_Index  Discharge Time (s)  Decre
ment 3.6-3.4V (s)  \
0              1.0             2595.30              1151.488500
1              2.0             7408.64              1172.512500
2              3.0             7393.76              1112.992000
3              4.0             7385.50              1080.320667
4              6.0            65022.75             29813.487000
...            ...                 ...                      ...
15059       1108.0              770.44               179.523810
15060       1109.0              771.12               179.523810
15061       1110.0              769.12               179.357143
15062       1111.0              773.88               162.374667
15063       1112.0           677537.27            142740.640000

       Max. Voltage Dischar. (V)  Min. Voltage Charg. (V)  Time at 4.15V (s)  \
0                          3.670                    3.211           5460.001
1                          4.246                    3.220           5508.992
2                          4.249                    3.224           5508.993
3                          4.250                    3.225           5502.016
4                          4.290                    3.398           5480.992
...                          ...                      ...                ...
15059                      3.773                    3.742            922.775
15060                      3.773                    3.744            915.512
15061                      3.773                    3.742            915.513
15062                      3.763                    3.839            539.375
15063                      4.206                    3.305          49680.004

       Time constant current (s)  Charging time (s)   RUL
0                        6755.01           10777.82  1112
1                        6762.02           10500.35  1111
2                        6762.02           10420.38  1110
3                        6762.02           10322.81  1109
4                       53213.54           56699.65  1107
...                          ...                ...   ...
15059                    1412.38            6678.88     4
15060                    1412.31            6670.38     3
15061                    1412.31            6637.12     2
15062                    1148.00            7660.62     1
15063                  599830.14          599830.14     0

[15064 rows x 9 columns]>
```

In [9]:

```
df.isna()
```

Out[9]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	T const curr
0	False	False	False	False	False	False	F
1	False	False	False	False	False	False	F

2	False	False	False	False	False	False	F
3	False	False	False	False	False	False	F
4	False	False	False	False	False	False	F
...	...	...	...	...	...	...	
15059	False	False	False	False	False	False	F
15060	False	False	False	False	False	False	F
15061	False	False	False	False	False	False	F
15062	False	False	False	False	False	False	F
15063	False	False	False	False	False	False	F

15064 rows × 9 columns

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
Cycle_Index                  0
Discharge Time (s)           0
Decrement 3.6-3.4V (s)       0
Max. Voltage Dischar. (V)    0
Min. Voltage Charg. (V)      0
Time at 4.15V (s)            0
Time constant current (s)    0
Charging time (s)            0
RUL                          0
dtype: int64
```

In [11]:
```
df.notnull().sum()
```

Out[11]:
```
Cycle_Index                  15064
Discharge Time (s)           15064
Decrement 3.6-3.4V (s)       15064
Max. Voltage Dischar. (V)    15064
Min. Voltage Charg. (V)      15064
Time at 4.15V (s)            15064
Time constant current (s)    15064
Charging time (s)            15064
RUL                          15064
dtype: int64
```

In [12]:
```
df.isna().sum()
```

Out[12]:
```
Cycle_Index                  0
Discharge Time (s)           0
Decrement 3.6-3.4V (s)       0
Max. Voltage Dischar. (V)    0
Min. Voltage Charg. (V)      0
Time at 4.15V (s)            0
Time constant current (s)    0
Charging time (s)            0
RUL                          0
dtype: int64
```

In [13]:
```
df.dropna()
```

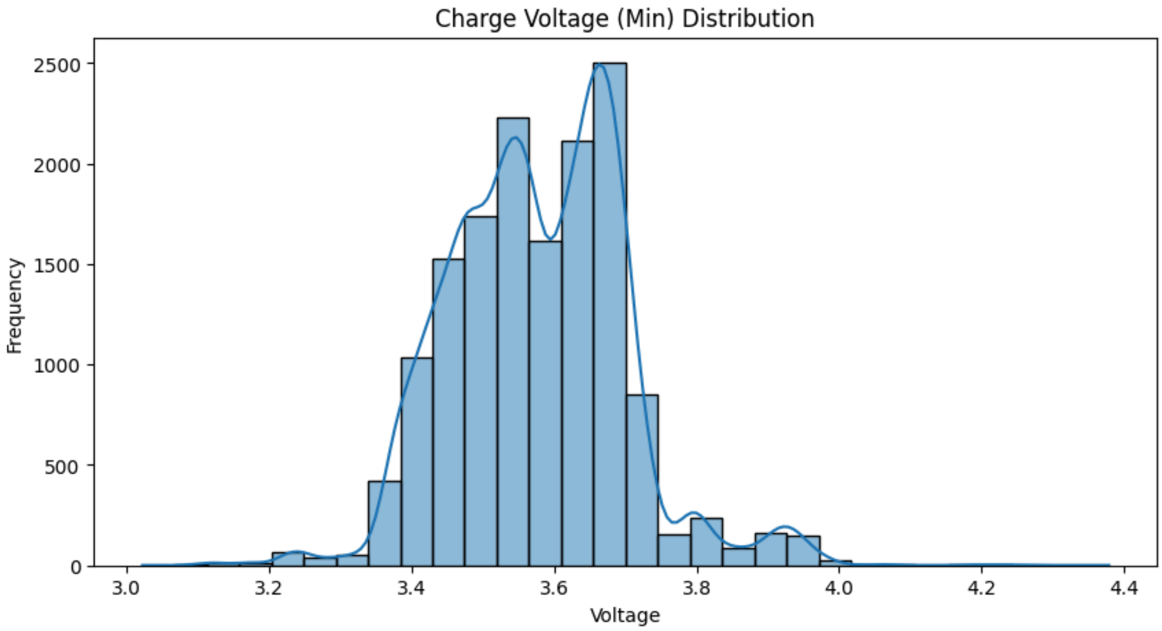
Out[13]:

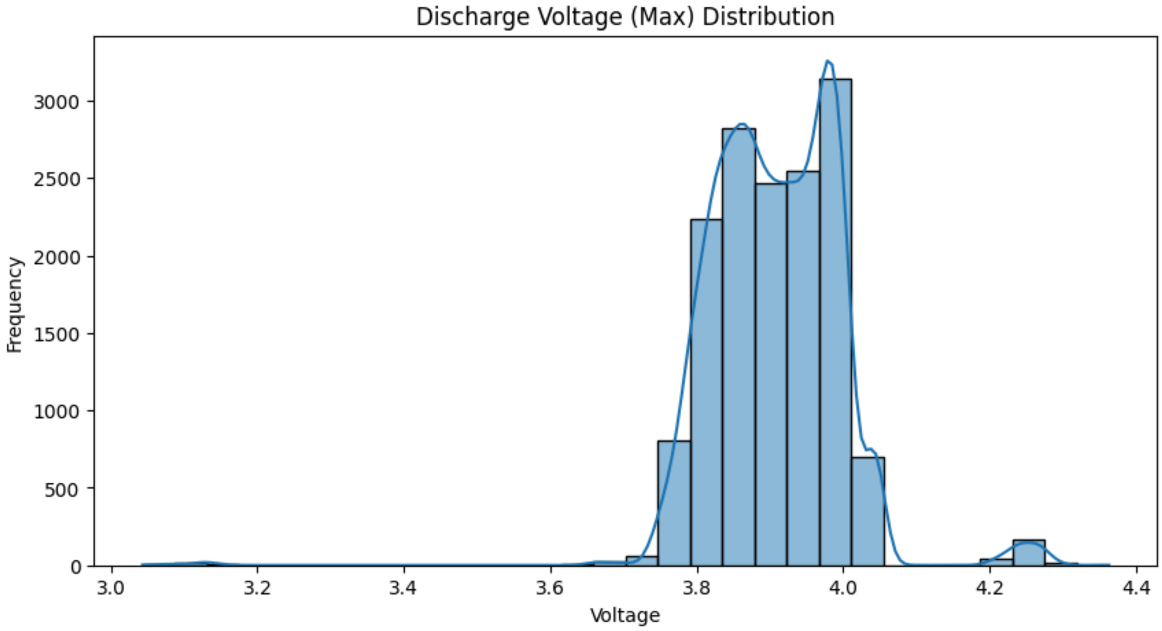
	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)
0	1.0	2595.30	1151.488500	3.670	3.211	5460.001
1	2.0	7408.64	1172.512500	4.246	3.220	5508.992
2	3.0	7393.76	1112.992000	4.249	3.224	5508.993
3	4.0	7385.50	1080.320667	4.250	3.225	5502.016
4	6.0	65022.75	29813.487000	4.290	3.398	5480.992
...	...	...	...	...	...	...
15059	1108.0	770.44	179.523810	3.773	3.742	922.775
15060	1109.0	771.12	179.523810	3.773	3.744	915.512
15061	1110.0	769.12	179.357143	3.773	3.742	915.513
15062	1111.0	773.88	162.374667	3.763	3.839	539.375
15063	1112.0	677537.27	142740.640000	4.206	3.305	49680.004

15064 rows × 9 columns

In [14]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.histplot(df["Min. Voltage Charg. (V)"], bins = 30, kde = True)
plt.title("Charge Voltage (Min) Distribution")
plt.xlabel("Voltage")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(10, 5))
sns.histplot(df["Max. Voltage Dischar. (V)"], bins = 30, kde = True)
plt.title("Discharge Voltage (Max) Distribution")
plt.xlabel("Voltage")
plt.ylabel("Frequency")
plt.show()
```





In [15]:
```
df["RUL_Class"] = pd.cut(df["RUL"], bins=[-1, 400, 800, float("inf")], labels=[0

# 0 : Critical
# 1 : Moderate
# 2 : Better

X = df.drop(columns=["RUL", "RUL_Class"])
Y = df["RUL_Class"].astype(int)
```

In [16]:
```
X
```

Out[16]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)
0	1.0	2595.30	1151.488500	3.670	3.211	5460.001
1	2.0	7408.64	1172.512500	4.246	3.220	5508.992
2	3.0	7393.76	1112.992000	4.249	3.224	5508.993

3	4.0	7385.50	1080.320667	4.250	3.225	5502.016
4	6.0	65022.75	29813.487000	4.290	3.398	5480.992
...	...	...	...	...	...	...
15059	1108.0	770.44	179.523810	3.773	3.742	922.775
15060	1109.0	771.12	179.523810	3.773	3.744	915.512
15061	1110.0	769.12	179.357143	3.773	3.742	915.513
15062	1111.0	773.88	162.374667	3.763	3.839	539.375
15063	1112.0	677537.27	142740.640000	4.206	3.305	49680.004

15064 rows × 8 columns

In [17]:
```
Y
```

Out[17]:
```
0        2
1        2
2        2
3        2
4        2
        ..
15059    0
15060    0
15061    0
15062    0
15063    0
Name: RUL_Class, Length: 15064, dtype: int64
```

In [18]:
```
import sklearn as sk
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8)
```

In [19]:
```
x_train
```

Out[19]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	co c
11849	1094.0	981.94	259.885714	3.793	3.679	1371.812	1
8942	316.0	1839.61	569.600000	3.971	3.500	3896.352	4
6776	310.0	1888.64	597.600000	3.975	3.495	4047.551	4
738	762.0	1255.19	346.500000	3.886	3.632	2054.312	2
6637	171.0	2035.00	688.000000	3.997	3.446	4529.944	5
...	...	...	...	...	...	...	
6757	291.0	1864.09	580.000000	3.973	3.492	3956.375	4
4984	691.0	1392.00	380.700000	3.877	3.611	2522.320	3
10728	1050.0	936.00	256.285714	3.813	3.698	1312.344	1
5383	1128.0	864.00	208.428571	3.760	3.717	1096.313	1
7988	459.0	1704.00	512.000000	3.955	3.524	3476.476	4

12051 rows × 8 columns

In [20]:
```
x_test
```

Out[20]:

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)
9933	227.0	1932.00	633.600000	3.987	3.472	4220.359000
12168	309.0	1899.14	538.000000	3.926	3.518	3982.720000
14758	770.0	1097.38	307.324675	3.842	3.670	1643.616571
14341	334.0	1770.00	532.000000	3.957	3.527	3626.344000
6932	484.0	1541.98	431.142857	3.928	3.561	2912.351000
...	...	...	...	...	...	...
1685	632.0	1513.62	404.400000	3.856	3.587	2751.544000
1417	343.0	1785.06	482.000000	3.924	3.536	3629.944000
12466	628.0	1462.06	376.114286	3.856	3.612	2570.375000
4063	854.0	1239.12	339.333333	3.847	3.638	2013.964286
4423	112.0	2124.00	746.002000	3.993	3.414	4790.351000

3013 rows × 8 columns

In [21]:
```
y_train
```

Out[21]:
```
11849    0
8942     1
6776     1
738      0
6637     2
        ..
6757     2
4984     1
10728    0
5383     0
7988     1
Name: RUL_Class, Length: 12051, dtype: int64
```

In [22]:
```
y_test
```

Out[22]:
```
9933     2
12168    1
14758    0
14341    1
6932     1
        ..
1685     1
1417     1
12466    1
4063     0
4423     2
Name: RUL_Class, Length: 3013, dtype: int64
```

MODEL 1 :

Logistic Regression

In [23]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [24]:
```
model
```

Out[24]:
```
▾ LogisticRegression
LogisticRegression()
```

In [25]:
```
model.fit(x_train, y_train)
```

```
C:\Users\shubh\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn
\linear_model\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta
tus=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[25]:
```
▾ LogisticRegression
LogisticRegression()
```

In [26]:
```
y_pred = model.predict(x_test)
```

In [27]:
```
y_pred
```

Out[27]:
```
array([2, 2, 0, ..., 1, 0, 2], shape=(3013,))
```

In [28]:
```
model.score(x_test, y_pred)
```

Out[28]:
```
1.0
```

In [29]:
```python
import sklearn as sk
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [30]:
```python
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

In [31]:
```python
mae
```

Out[31]:
```
0.026883504812479257
```

In [32]:
```python
mse
```

Out[32]:
```
0.027547295054762694
```

In [33]:
```python
import numpy as np
```

In [34]:
```python
np.sqrt(mse)
```

Out[34]:
```
np.float64(0.16597377821439957)
```

In [35]:
```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of Logistic Model : ", accuracy, "%")
```

```
Accuracy of Logistic Model :  97.34483903086625 %
```

In [36]:
```python
from sklearn.metrics import confusion_matrix

CM = confusion_matrix(y_test, y_pred)
CM
```

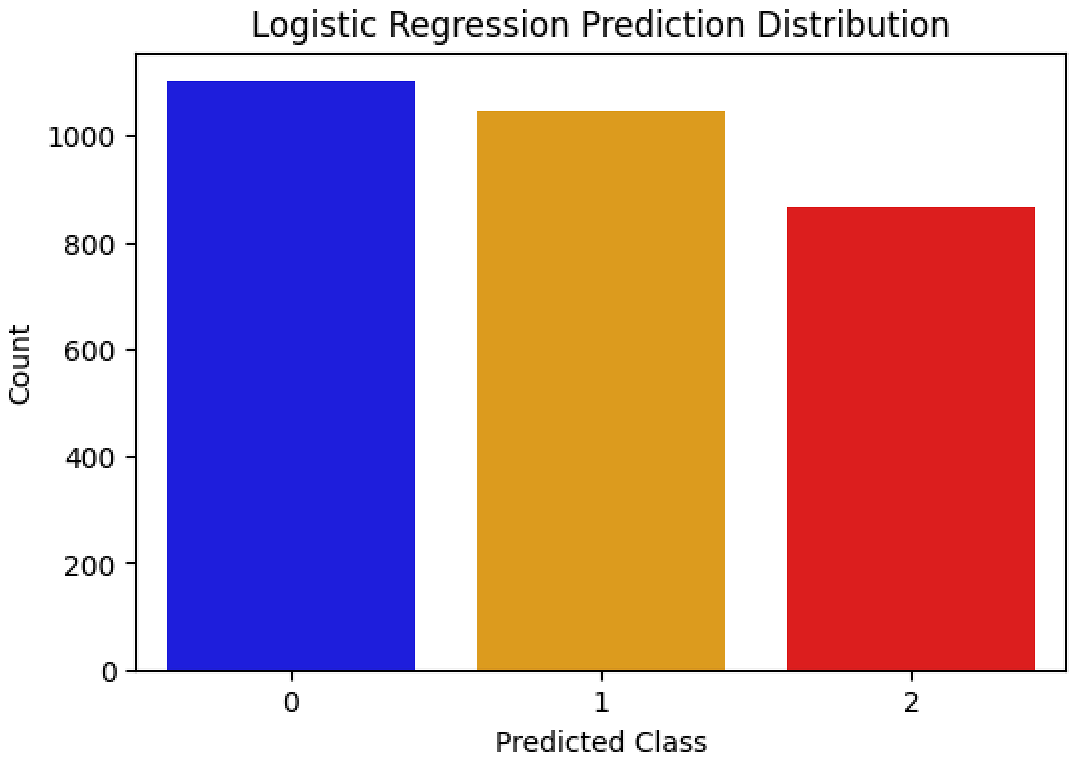
Out[36]:
```
array([[1073,   21,    0],
       [  27, 1019,   24],
       [   1,    7,  841]])
```

In [37]:
```python
CM.ravel()
```

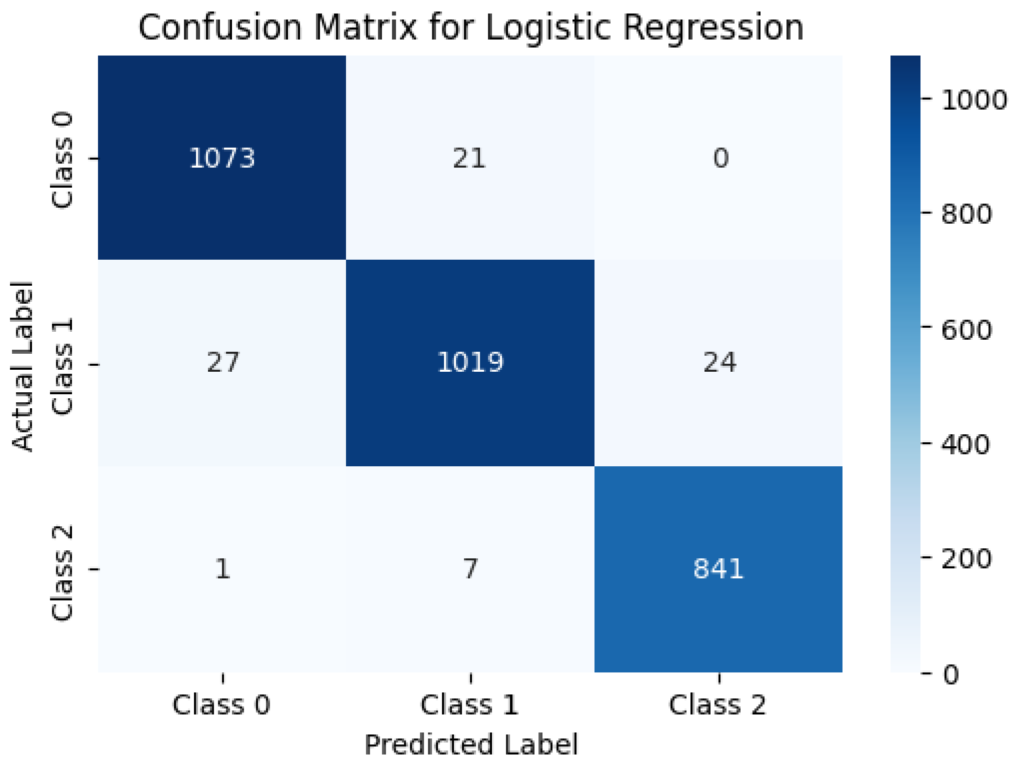
Out[37]:
```
array([1073,   21,    0,   27, 1019,   24,    1,    7,  841])
```

In [38]:
```python
classes, counts = np.unique(y_pred, return_counts=True)

plt.figure(figsize=(6, 4))
sns.barplot(x=classes, y=counts, palette=['blue', 'orange', 'red'])  # Corrected
plt.xlabel("Predicted Class")
plt.ylabel("Count")
plt.title("Logistic Regression Prediction Distribution")
plt.show()
```

```
C:\Users\shubh\AppData\Local\Temp\ipykernel_18224\450399267.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=classes, y=counts, palette=['blue', 'orange', 'red'])  # Correcte
d color usage
```



In [39]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(CM, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Clas
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.title("Confusion Matrix for Logistic Regression")
plt.show()
```



In [40]:
```
from sklearn.metrics import precision_score
PS = precision_score(y_test, y_pred, average = 'macro')
PS
```

Out[40]: 0.9733599446100177

In [41]:
```
from sklearn.metrics import recall_score
RS = recall_score(y_test, y_pred, average = 'macro')
RS
```

Out[41]: 0.9745726619181457

In [42]:
```
from sklearn.metrics import f1_score
F1 = f1_score(y_test, y_pred, average = 'macro')
F1
```

Out[42]: 0.9738966004432207

In [43]:
```
from sklearn.metrics import classification_report
CR = classification_report(y_test, y_pred)
print(CR)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      1094
           1       0.97      0.95      0.96      1070
           2       0.97      0.99      0.98       849

    accuracy                           0.97      3013
   macro avg       0.97      0.97      0.97      3013
weighted avg       0.97      0.97      0.97      3013
```

MODEL 2 :

Gaussian Naive-Bayes

In [44]:
```
from sklearn.naive_bayes import GaussianNB
model2 = GaussianNB()
```

In [45]:
```
model2
```

Out[45]:

▾ GaussianNB

GaussianNB()

In [46]:
```
model2.fit(x_train, y_train)
```

Out[46]:

▾ GaussianNB

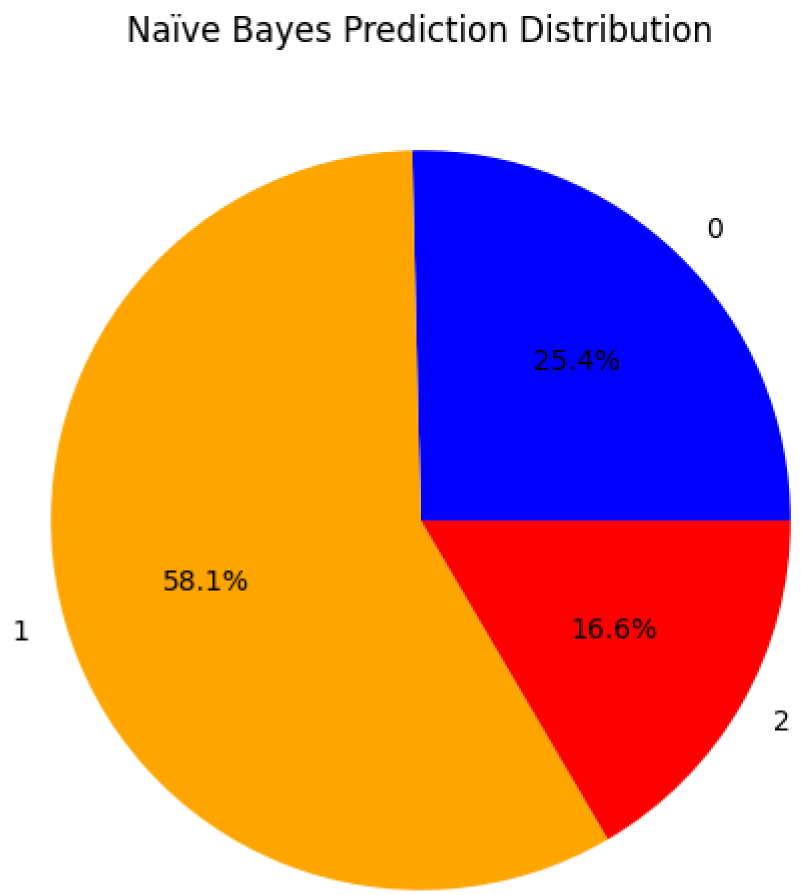
GaussianNB()

In [47]:
```
y_pred = model2.predict(x_test)
```

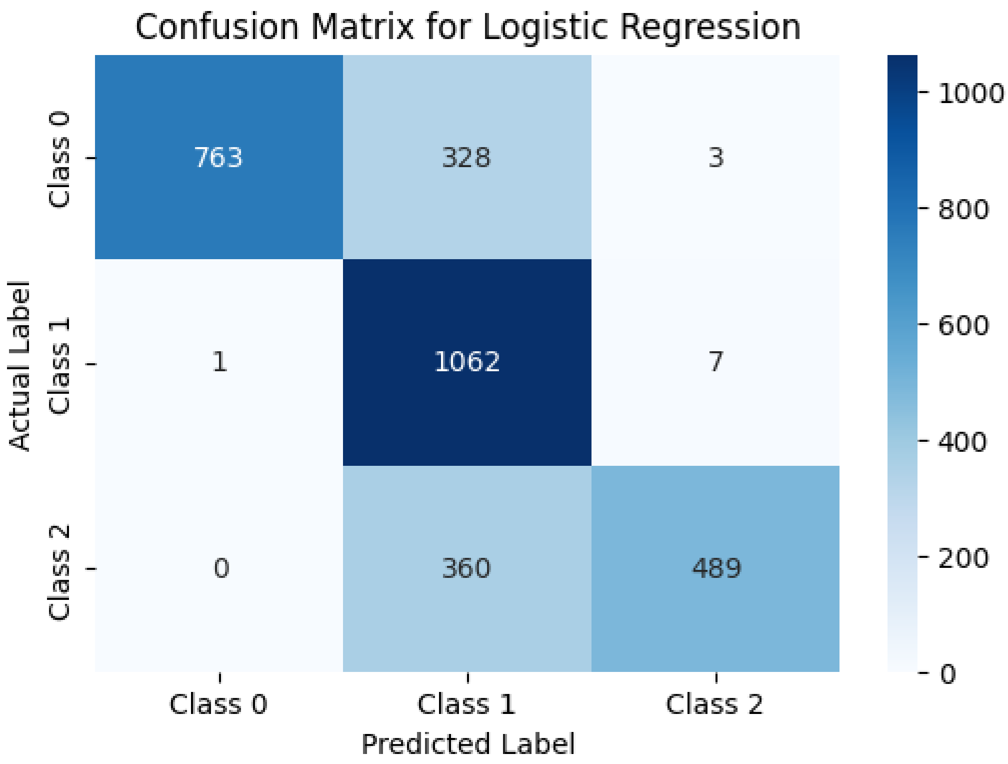
In [48]:
```
y_pred
```

Out[48]: array([1, 1, 1, ..., 1, 0, 2], shape=(3013,))

In [49]:
```
model2.score(x_test, y_pred)
```
Out[49]:
```
1.0
```
In [50]:
```
import sklearn as sk
from sklearn.metrics import mean_absolute_error, mean_squared_error
```
In [51]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```
In [52]:
```
mae
```
Out[52]:
```
0.2329903750414869
```
In [53]:
```
mse
```
Out[53]:
```
0.23498174576833722
```
In [54]:
```
import numpy as np
```
In [55]:
```
np.sqrt(mse)
```
Out[55]:
```
np.float64(0.484749157573623)
```
In [56]:
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of Gaussian Naive-Bayes Model : ", accuracy, "%")
```
```
Accuracy of Gaussian Naive-Bayes Model :  76.80053103219383 %
```
In [57]:
```
from sklearn.metrics import confusion_matrix

CM = confusion_matrix(y_test, y_pred)
CM
```
Out[57]:
```
array([[ 763,  328,    3],
       [   1, 1062,    7],
       [   0,  360,  489]])
```
In [58]:
```
classes, counts = np.unique(y_pred, return_counts=True)

# Create Pie Chart
plt.figure(figsize=(6, 6))
plt.pie(counts, labels=classes, autopct="%1.1f%%", colors=['blue', 'orange', 're
plt.title("Naïve Bayes Prediction Distribution")
plt.show()
```



In [59]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(CM, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Clas
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.title("Confusion Matrix for Logistic Regression")
plt.show()
```



In [60]:
```
from sklearn.metrics import precision_score
PS = precision_score(y_test, y_pred, average = 'macro')
PS
```

Out[60]: 0.8618360540577541

In [61]:
```
from sklearn.metrics import recall_score
RS = recall_score(y_test, y_pred, average = 'macro')
RS
```

Out[61]: 0.7553118936479618

In [62]:
```
from sklearn.metrics import f1_score
F1 = f1_score(y_test, y_pred, average = 'macro')
F1
```

Out[62]: 0.7666746724128624

In [63]:
```
from sklearn.metrics import classification_report
CR = classification_report(y_test, y_pred)
print(CR)
```

```
              precision    recall  f1-score   support

           0       1.00      0.70      0.82      1094
           1       0.61      0.99      0.75      1070
           2       0.98      0.58      0.73       849

    accuracy                           0.77      3013
   macro avg       0.86      0.76      0.77      3013
weighted avg       0.85      0.77      0.77      3013
```

MODEL 3 :

Decision Tree Classifier

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier()
```

In [65]:
```
model3
```

Out[65]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [66]:
```
model3.fit(x_train, y_train)
```

Out[66]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [67]:
```
y_pred = model3.predict(x_test)
```

In [68]:
```
y_pred
```

Out[68]:
```
array([2, 1, 0, ..., 1, 0, 2], shape=(3013,))
```

In [69]:
```
model3.score(x_test, y_pred)
```

Out[69]:
```
1.0
```

In [70]:
```
import sklearn as sk
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [71]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

In [72]:
```
mae
```

Out[72]:
```
0.003982741453700631
```

In [73]:
```
mse
```

Out[73]:
```
0.003982741453700631
```

In [74]:
```
np.sqrt(mse)
```

Out[74]:
```
np.float64(0.06310896492338176)
```

In [75]:
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of Decision Tree Classifier Model : ", accuracy, "%")
```

```
Accuracy of Decision Tree Classifier Model :  99.60172585462995 %
```

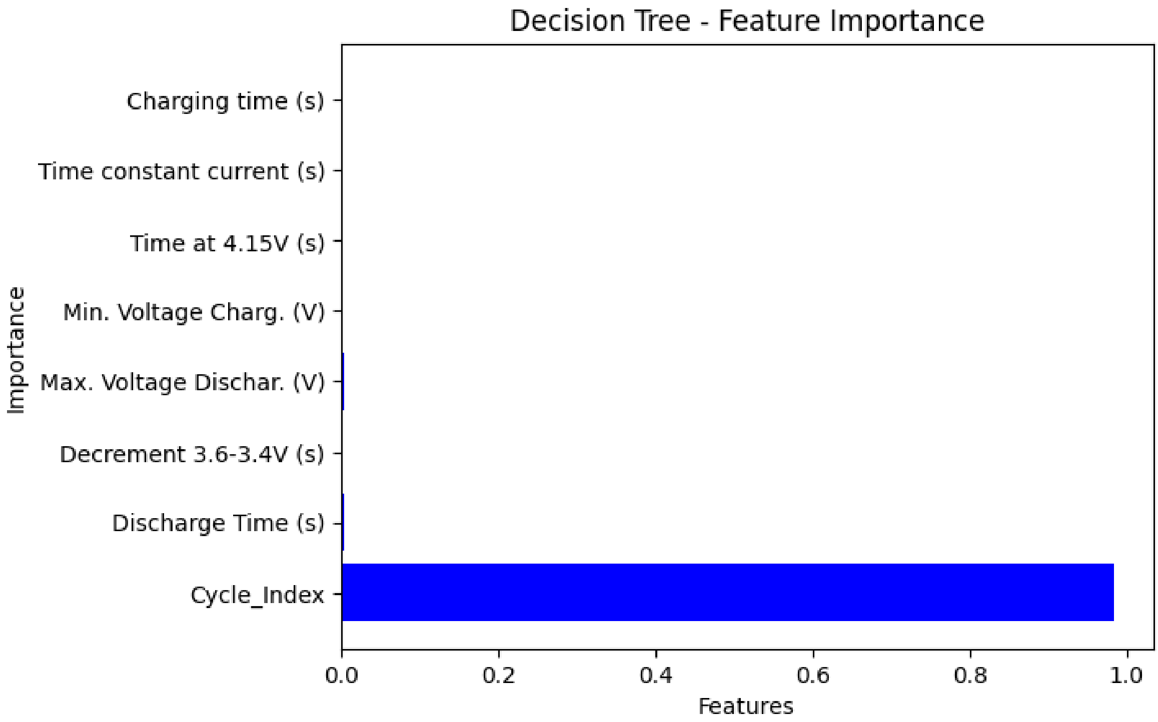
In [76]:
```
from sklearn.metrics import confusion_matrix
```

```
CM = confusion_matrix(y_test, y_pred)
CM
```

Out[76]:
```
array([[1090,    4,    0],
       [   4, 1063,    3],
       [   0,    1,  848]])
```

In [77]:
```
importance = model3.feature_importances_
print(importance)
plt.barh(x_train.columns, importance, color="blue")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Decision Tree - Feature Importance")
plt.show()
```

```
[9.84451373e-01 5.05941720e-03 2.06477227e-03 4.32464315e-03
 5.03150559e-04 1.85583867e-03 3.46383265e-04 1.39442212e-03]
```



In [78]:
```
from sklearn.metrics import precision_score
PS = precision_score(y_test, y_pred, average = 'macro')
PS
```

Out[78]: 0.9960455935117656

In [79]:
```
from sklearn.metrics import recall_score
RS = recall_score(y_test, y_pred, average = 'macro')
RS
```

Out[79]: 0.9962079268313012

In [80]:
```
from sklearn.metrics import f1_score
F1 = f1_score(y_test, y_pred, average = 'macro')
F1
```

Out[80]: 0.9961260098411593

In [81]:
```
from sklearn.metrics import classification_report
CR = classification_report(y_test, y_pred)
print(CR)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1094
           1       1.00      0.99      0.99      1070
           2       1.00      1.00      1.00       849

    accuracy                           1.00      3013
   macro avg       1.00      1.00      1.00      3013
weighted avg       1.00      1.00      1.00      3013
```

In [ ]:

MODEL 4 :

Linear-SVM

In [82]:
```
from sklearn.svm import SVC
model4 = SVC(kernel='rbf')
```

In [83]:
```
model4
```

Out[83]:
```
▾ SVC
SVC()
```

In [84]:
```
model4.fit(x_train, y_train)
```

Out[84]:
```
▾ SVC
SVC()
```

In [85]:
```
y_pred = model4.predict(x_test)
```

In [86]:
```
y_pred
```

Out[86]:
```
array([2, 2, 0, ..., 1, 0, 2], shape=(3013,))
```

In [87]:
```
model4.score(x_test, y_pred)
```

Out[87]:
```
1.0
```

In [88]:
```
import sklearn as sk
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [89]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

In [90]:
```
mae
```

Out[90]: 0.06837039495519416

In [91]:
```
mse
```

Out[91]: 0.07036176568204447

In [92]:
```
np.sqrt(mse)
```

Out[92]: np.float64(0.2652579229392488)

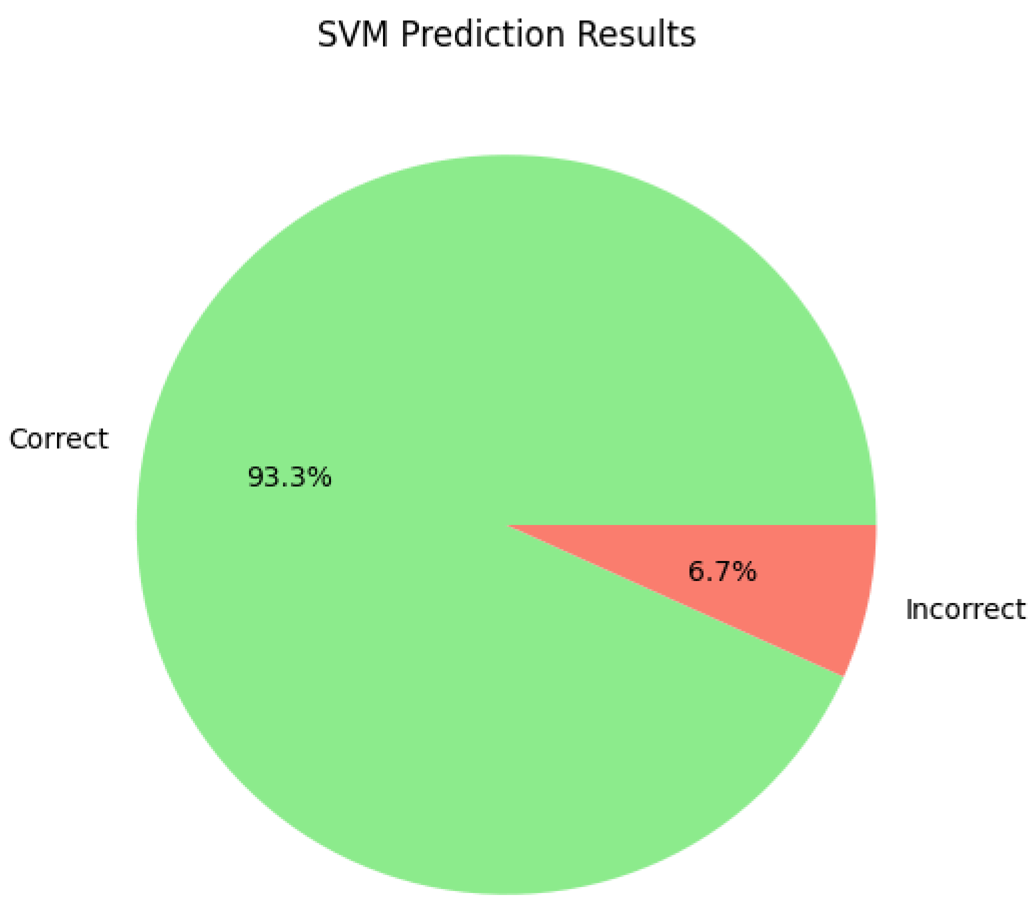
In [93]:
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of Linear-SVM Model : ", accuracy, "%")
```

```
Accuracy of Linear-SVM Model :  93.2625290408231 %
```

In [94]:
```
from sklearn.metrics import confusion_matrix

CM = confusion_matrix(y_test, y_pred)
CM
```

Out[94]:
```
array([[1055,   38,    1],
       [  74,  924,   72],
       [   2,   16,  831]])
```

In [95]:
```
correct = sum(y_test == y_pred)  # Number of correct predictions
incorrect = len(y_test) - correct  # Number of incorrect predictions

labels = ['Correct', 'Incorrect']
sizes = [correct, incorrect]
colors = ['lightgreen', 'salmon']

plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('SVM Prediction Results')
plt.show()
```



In [96]:
```
from sklearn.metrics import precision_score
PS = precision_score(y_test, y_pred, average = 'macro')
PS
```

Out[96]:
```
0.9322786310129308
```

In [97]:
```
from sklearn.metrics import recall_score
RS = recall_score(y_test, y_pred, average = 'macro')
RS
```

Out[97]:
```
0.9355669979753526
```

In [98]:
```
from sklearn.metrics import f1_score
F1 = f1_score(y_test, y_pred, average = 'macro')
F1
```

Out[98]:
```
0.9329157823479707
```

In [99]:
```
from sklearn.metrics import classification_report
CR = classification_report(y_test, y_pred)
print(CR)
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      1094
           1       0.94      0.86      0.90      1070
           2       0.92      0.98      0.95       849

    accuracy                           0.93      3013
```

macro avg	0.93	0.94	0.93	3013
weighted avg	0.93	0.93	0.93	3013

In [ ]:

MODEL 5 :

Non Linear-SVM

In [100…
```
from sklearn.svm import SVC
model5 = SVC(kernel='poly')
```

In [101…
```
model5
```

Out[101…

SVC

SVC(kernel='poly')

In [102…
```
model5.fit(x_train, y_train)
```

Out[102…

SVC

SVC(kernel='poly')

In [103…
```
y_pred = model5.predict(x_test)
```

In [104…
```
y_pred
```

Out[104…
```
array([0, 0, 0, ..., 0, 0, 0], shape=(3013,))
```

In [105…
```
model5.score(x_test, y_pred)
```

Out[105…
```
1.0
```

In [106…
```
import sklearn as sk
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [107…
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

In [108…
```
mae
```

Out[108…
```
0.9070693660803186
```

In [109…
```
mse
```

Out[109…
```
1.4600066379024228
```

In [110…
```
np.sqrt(mse)
```

Out[110…
```
np.float64(1.2083073441399017)
```

In [111…

```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of Non Linear-SVM Model : ", accuracy, "%")
```

```
Accuracy of Non Linear-SVM Model :  36.93992698307335 %
```

In [112…

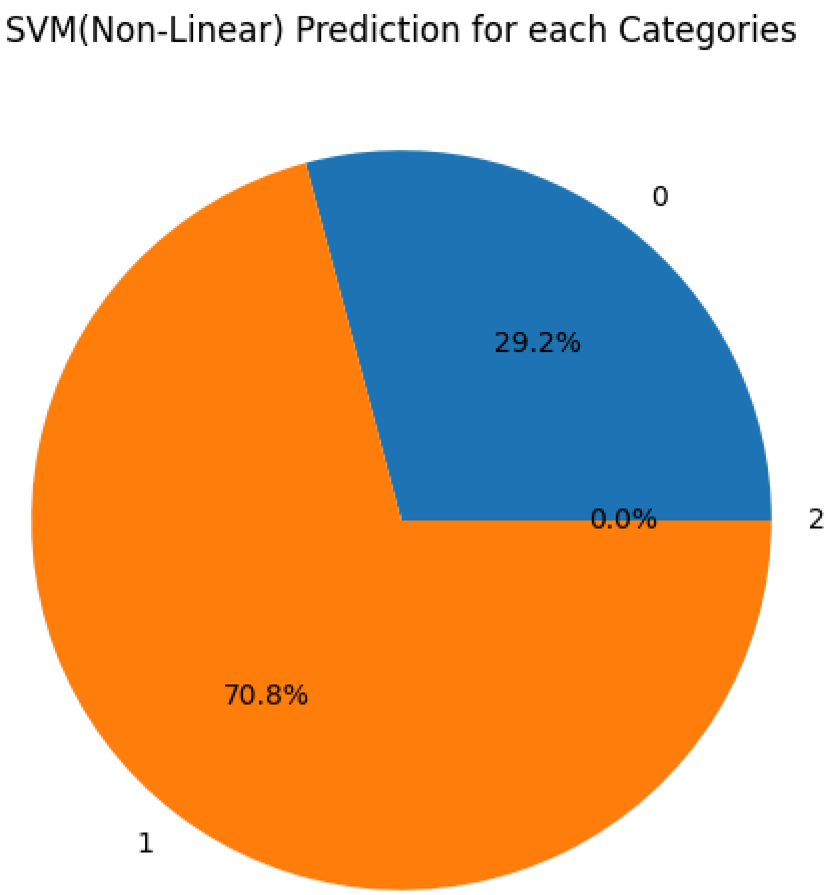
```python
from sklearn.metrics import confusion_matrix

CM = confusion_matrix(y_test, y_pred)
CM
```

Out[112…

```
array([[1094,    0,    0],
       [1062,    5,    3],
       [ 833,    2,   14]])
```

In [113…

```python
counts = [sum(y_pred == i) for i in [1, 2, 3]]
plt.figure(figsize=(6, 6))
plt.pie(counts, labels=['0', '1', '2'], autopct='%1.1f%%')
plt.title('SVM(Non-Linear) Prediction for each Categories')
plt.show()
```



In [114…

```python
from sklearn.metrics import precision_score
PS = precision_score(y_test, y_pred, average = 'macro')
PS
```

Out[114…

```
0.6346079415372706
```

In [115…

```
from sklearn.metrics import recall_score
RS = recall_score(y_test, y_pred, average = 'macro')
RS
```

Out[115…

```
0.3403876284725662
```

In [116…

```
from sklearn.metrics import f1_score
F1 = f1_score(y_test, y_pred, average = 'macro')
F1
```

Out[116…

```
0.19249936487245345
```

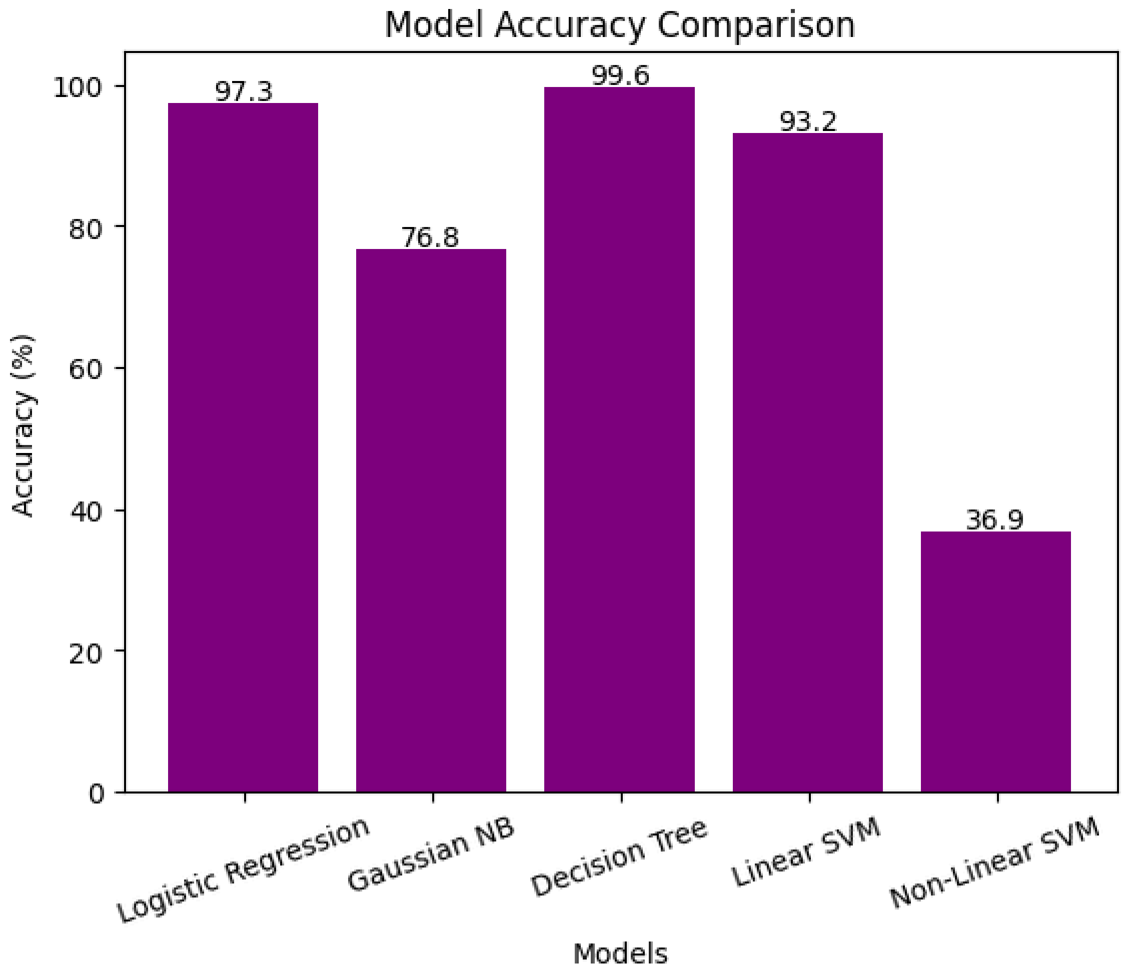
In [117…

```
from sklearn.metrics import classification_report
CR = classification_report(y_test, y_pred)
print(CR)
```

```
              precision    recall  f1-score   support

           0       0.37      1.00      0.54      1094
           1       0.71      0.00      0.01      1070
           2       0.82      0.02      0.03       849

    accuracy                           0.37      3013
   macro avg       0.63      0.34      0.19      3013
weighted avg       0.62      0.37      0.21      3013
```

In [123…

```
models = ["Logistic Regression", "Gaussian NB",
          "Decision Tree", "Linear SVM", "Non-Linear SVM"]
accuracies = [97.3, 76.8, 99.6, 93.2, 36.9]

for i, v in enumerate(accuracies):
    plt.text(i, v + 0.5, str(v), ha='center', fontsize=10)

plt.bar(models, accuracies, color="purple")
plt.xticks(rotation=20)
plt.xlabel("Models")
plt.ylabel("Accuracy (%)")
plt.title("Model Accuracy Comparison")

plt.show()
```


Model Accuracy Comparison
97.3
76.8
99.6
93.2
36.9
100
80
60
40
20
0
Accuracy (%)
Logistic Regression
Gaussian NB
Decision Tree
Linear SVM
Non-Linear SVM
Models

```
In [ ]:
```