In [4]:
```
import pandas as pd


from sklearn.module import Model
model = Model()
model.fit(X,y)
predictions = model.predict(X_new)
print(predictions)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [4], in <cell line: 4>()
      1 import pandas as pd
----> 4 from sklearn.module import Model
      5 model = Model()
      6 model.fit(X,y)

ModuleNotFoundError: No module named 'sklearn.module'
```

In [5]:
```
import pandas as pd
import numpy as np

churn_df = pd.read_csv("churn_df.csv")

from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)
```

```
(3333, 2) (3333,)
```

In [6]:
```
knn = KNeighborsClassifier(n_neighbors = 15)
knn.fit(X,y)
```

Out[6]:
```
KNeighborsClassifier(n_neighbors=15)
```

In [7]:
```
X_new = np.array([[56.8,17.5],
                 [24.4,24.1],
                 [50.1, 10.9]])

print(X_new.shape)
```

```
(3, 2)
```

In [8]:
```
predictions = knn.predict(X_new)
print("Prediction: {}".format(predictions))
```

```
Prediction: [1 0 0]
```

In [9]:
```
import pandas as pd
import numpy as np

churn_df = pd.read_csv("churn_df.csv")

from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)

knn = KNeighborsClassifier(n_neighbors = 15)
knn.fit(X,y)

X_new = np.array([[30.0,17.5],
                  [107.0,24.1],
                  [213.0, 100.9]])

print(X_new.shape)

predictions = knn.predict(X_new)
print(f"Prediction: {predictions}")
```

```
(3333, 2) (3333,)
(3, 2)
Prediction: [0 1 1]
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
                                                    random_state = 21,
                                                    stratify=y)
knn = KNeighborsClassifier(n_neighbors = 6)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

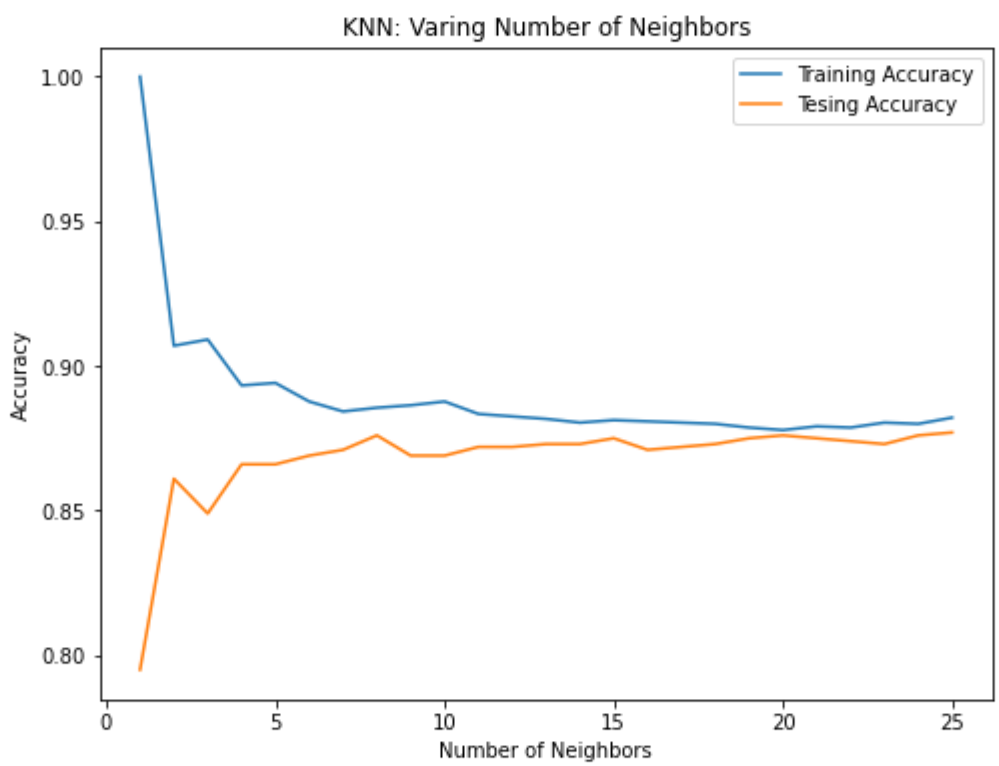
```
0.869
```

In [11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
                                                    random_state = 21,
                                                    stratify=y)
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.861
```

In [12]:
```
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1,26)
print(neighbors)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

#print(train_accuracies.values())
print(test_accuracies.values())
my_train = list(train_accuracies.values())
my_test = list(test_accuracies.values())

plt.figure(figsize=(8,6))
plt.title("KNN: Varing Number of Neighbors")
plt.plot(neighbors, my_train, label="Training Accuracy")
plt.plot(neighbors, my_test, label="Tesing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
dict_values([0.795, 0.861, 0.849, 0.866, 0.866, 0.869, 0.871, 0.876, 0.869,
0.869, 0.872, 0.872, 0.873, 0.873, 0.875, 0.871, 0.872, 0.873, 0.875, 0.876,
0.875, 0.874, 0.873, 0.876, 0.877])
```



In [13]:
```
#Exercise

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

X = df.drop('churn', axis=1).values
y = df['churn'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
accuracy = knn.score(X_test, y_test)
print("Accuracy of the KNN model on the test data:", accuracy)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [13], in <cell line: 7>()
      4 from sklearn.model_selection import train_test_split
      5 from sklearn.neighbors import KNeighborsClassifier
----> 7 X = df.drop('churn', axis=1).values
      8 y = df['churn'].values
     10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.2, random_state=42, stratify=y)

NameError: name 'df' is not defined
```

In [ ]:
```
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1,13)
print(neighbors)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

#print(train_accuracies.values())
print(test_accuracies.values())
my_train = list(train_accuracies.values())
my_test = list(test_accuracies.values())

plt.figure(figsize=(8,6))
plt.title("KNN: Varing Number of Neighbors")
plt.plot(neighbors, my_train, label="Training Accuracy")
plt.plot(neighbors, my_test, label="Tesing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

In [ ]:
```
import pandas as pd

diabetes_df = pd.read_csv("diabetes.csv")
print(diabetes_df.head())

import pandas as pd


print("Initial Shape:", diabetes_df.shape)

diabetes_df = diabetes_df[diabetes_df['bmi'] != 0]

print("Shape after Subset for BMI not equal to 0:", diabetes_df.shape)

diabetes_df = diabetes_df[diabetes_df['glucose'] != 0]
print("Shape after Subset for Glucose not equal to 0:", diabetes_df.shape)

diabetes_df.head()



X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values

print(type(X), type(y))

X_bmi = X[:, 3]
print(y.shape, X_bmi.shape)

X_bmi = X_bmi.reshape(-1, 1)
print(X_bmi.shape)



import matplotlib.pyplot as plt
plt.scatter(X_bmi,y, color="Blue")
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Blood Mass Index")
plt.show()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_bmi, y)
predictions = reg.predict(X_bmi)
plt.scatter(X_bmi, y, color="Red")
plt.plot(X_bmi, predictions)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```

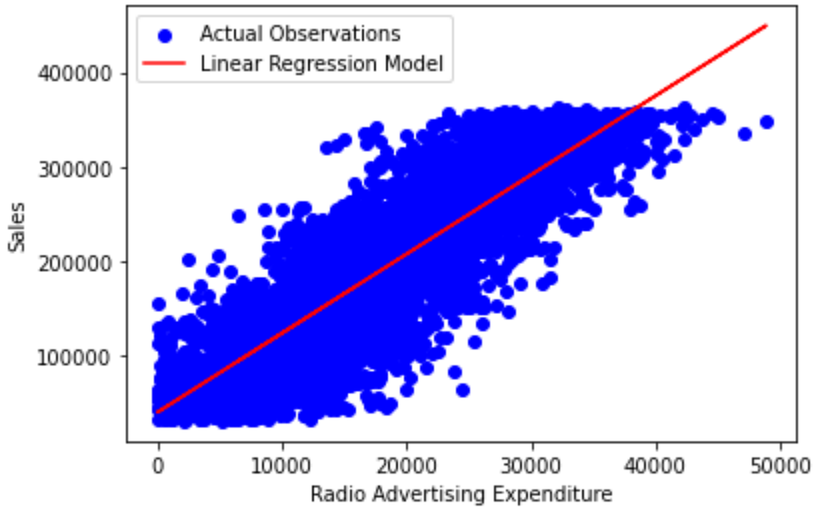
In [14]:
```
import pandas as pd
import numpy as np
sales_df = pd.read_csv('sales_df.csv')

X = sales_df['radio'].values
y = sales_df['sales'].values

X = X.reshape(-1, 1)
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (4546, 1)
Shape of y: (4546,)
```

In [15]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

predictions = model.predict(X)

print("Five prediction values:", predictions[:5])
```

```
Five prediction values: [ 95491.17119147 117829.51038393 173423.38071499 2916
03.11444202
 111137.28167129]
```

In [16]:
```
import matplotlib.pyplot as plt
plt.scatter(X, y, color='blue', label='Actual Observations')

plt.plot(X, predictions, color='red', label='Linear Regression Model')

plt.xlabel('Radio Advertising Expenditure')
plt.ylabel('Sales')
plt.legend(loc='best')
plt.show()
```




In [24]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,
                                                    random_state = 42)

reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)

reg_all.score(X_test, y_test)

from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared = False)
```

Out[24]:
```
2945.0531856107264
```

In [18]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

X = sales_df.drop('sales', axis=1).values
y = sales_df['sales'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("First two values of y_pred:", y_pred[:2])
print("First two values of y_test:", y_test[:2])

r_squared = r2_score(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("R-squared:", r_squared)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
First two values of y_pred: [53211.24654996 71094.18581089]
First two values of y_test: [55261.28 67574.9 ]
R-squared: 0.9990108723060241
Root Mean Squared Error (RMSE): 2884.261780626478
```

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import  LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random

reg_all = LinearRegression()
reg_all.fit(X_train, y_train)

y_pred = reg_all.predict(X_test)
```

In [20]:
```
reg_all.score(X_test, y_test)
```

Out[20]:
```
0.9990147957135925
```

In [21]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared = False)
```

Out[21]:
```
2945.0531856107264
```

In [29]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

sales_df = pd.read_csv('sales_df.csv')

X = sales_df.drop(columns=['sales'])
y = sales_df['sales']

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Predicted values:", y_pred[:2])
print("Actual target :", y_test[:2])

r_squared = r2_score(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print()

print("R-squared:", r_squared)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Predicted values: [53099.56399301 71056.14674591]
Actual target : [55261.28 67574.9 ]

R-squared: 0.9990147957135925
Root Mean Squared Error (RMSE): 2945.0531856107264
```

In [32]:
```
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_splits = 6, shuffle=True, random_state=42)
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y , cv = kf)

print(cv_results)
print(np.mean(cv_results), np.std(cv_results))
```

```
[0.99900932 0.99898731 0.99897213 0.99898864 0.99892796 0.99906015]
0.9989909165767226 3.972232714460531e-05
```

In [ ]:

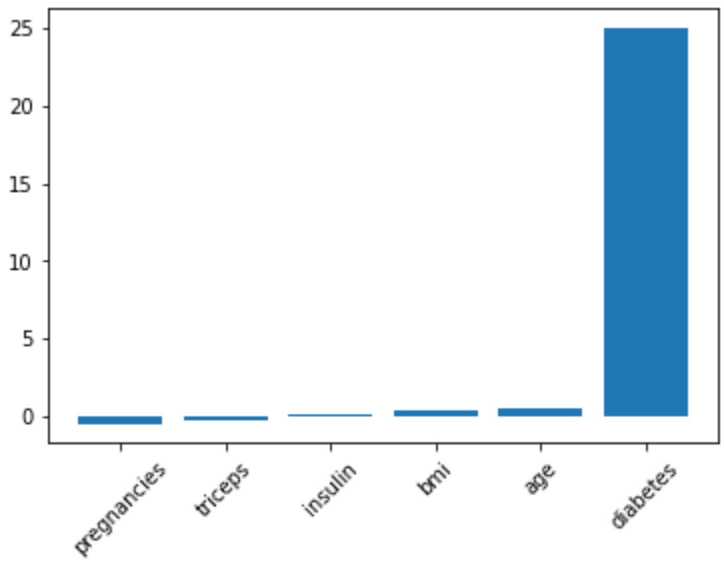
In [53]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression

sales_df = pd.read_csv('sales_df.csv')

X = sales_df[['radio', 'social_media']]
y = sales_df['sales']

kf = KFold(n_splits=6, shuffle=True, random_state=5)

reg = LinearRegression()
cv_scores = cross_val_score(reg, X, y, cv=kf)

print("CV scores: ", cv_scores)

mean_score = np.mean(cv_scores)
print("Mean:", mean_score)

std_score = np.std(cv_scores)
print("Std:", std_score)

confidence_interval = np.quantile(cv_scores, [0.025, 0.975])
print("95% Confidence Interval:", confidence_interval)
```

```
CV scores:  [0.74451678 0.77241887 0.76842114 0.7410406  0.75170022 0.7440648
4]
Mean: 0.7536937414361207
Std: 0.012305389070474664
95% Confidence Interval: [0.74141863 0.77191916]
```

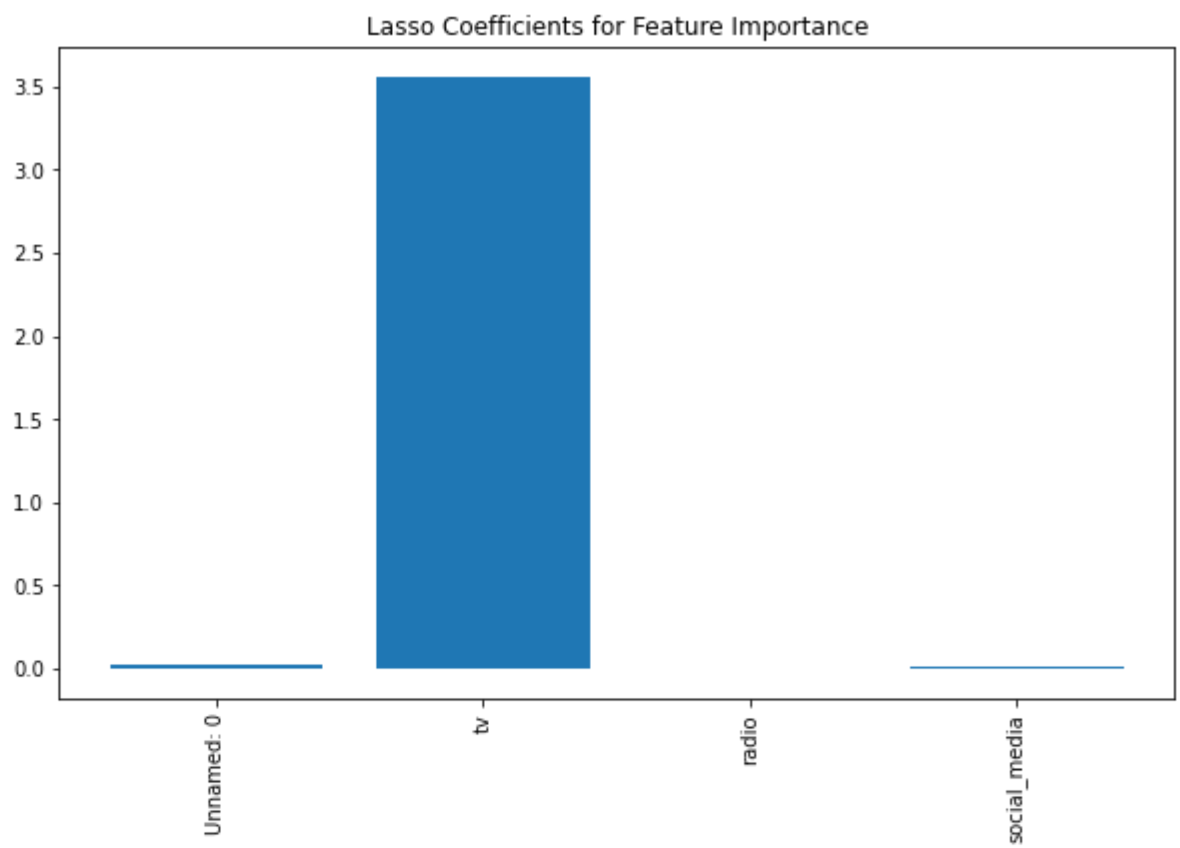
In [56]:
```
from sklearn.linear_model import Lasso
diabetes_df = pd.read_csv('diabetes.csv', index_col = 0)
diabetes_df = diabetes_df[diabetes_df['bmi'] != 0]
diabetes_df = diabetes_df[diabetes_df['glucose'] != 0]
X = diabetes_df.drop('glucose', axis=1).values
y = diabetes_df['glucose'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
scores = []
for alpha in [0.01,1.0,10.0, 20.0, 50.0]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    lasso_pred = lasso.predict(X_test)
    scores.append(lasso.score(X_test, y_test))
print(scores)
```

```
[0.3562250067582078, 0.34618285370900204, 0.201448239274153, 0.18595115472492
296, 0.14542319216659483]
```

In [57]:
```
diabetes_df = pd.read_csv('diabetes.csv', index_col = 0)
diabetes_df = diabetes_df[diabetes_df['bmi'] != 0]
diabetes_df = diabetes_df[diabetes_df['glucose'] != 0]
X = diabetes_df.drop('glucose', axis=1).values
y = diabetes_df['glucose'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
names = diabetes_df.drop('glucose', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X,y).coef_
plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
plt.show()
```



In [67]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso

sales_df = pd.read_csv('sales_df.csv')

X = sales_df.drop(columns=['sales'])
y = sales_df['sales']

sales_columns = X.columns

lasso = Lasso(alpha=0.3)

lasso.fit(X, y)

lasso_coef = lasso.coef_

plt.figure(figsize=(10, 6))
plt.bar(sales_columns, lasso_coef)
plt.title("Lasso Coefficients for Feature Importance")
plt.xticks(rotation=90)
plt.show()
```




In [69]:

In [73]:
```
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors = 7)
X_train, X_test, y_train, y_test =train_test_split(X,y, test_size=0.4, random_

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [72]:
```
music_df = pd.read_csv('music.csv', index_col = 0)
music_dummies = pd.get_dummies(music_df['genre'], drop_first=True)
#music_dummies.head()
music_dummies = pd.concat([music_df, music_dummies], axis = 1)
music_dummies = music_dummies.drop('genre', axis=1)
#music_dummies.head()
print(music_dummies.columns)
#from sklearn.model_selection import cross_val_score, KFold
#from sklearn.linear_model import LinearRegression
X = music_dummies.drop('popularity', axis=1).values
y = music_dummies['popularity'].values
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,
random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
linreg = LinearRegression()
linreg_cv = cross_val_score(linreg, X_train, y_train, cv=kf,
scoring='neg_mean_squared_error')
linreg_cv2 = cross_val_score(linreg, X_train, y_train, cv=kf)
print(np.sqrt(-linreg_cv))
print(linreg_cv2)
```

```
Index(['popularity', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'Anime', 'Blues', 'Classical', 'Country', 'Electronic',
       'Hip-Hop', 'Jazz', 'Rap', 'Rock'],
      dtype='object')
[8.15810501 8.63114581 7.52281687 8.62016985 7.91296943]
[0.66352267 0.68438203 0.68977273 0.64469603 0.63959551]
```

In [74]:
```
#code for SimpleImputer example

from sklearn.impute import SimpleImputer



music_df = pd.read_csv('music_unclean.csv', index_col = 0)
print(music_df.columns)
print(music_df.isna().sum().sort_values())
music_df = music_df.dropna(subset=['genre','popularity','loudness','liveness',
music_df['genre'] = np.where(music_df['genre'] == 'Rock', 1,0)
print(music_df.isna().sum().sort_values())



X_cat = music_df['genre'].values.reshape(-1,1)
X_num = music_df.drop(['genre','popularity'], axis=1).values
y = music_df['popularity'].values



X_train_cat, X_test_cat, y_train, y_test = train_test_split(X_cat, y, test_siz
random_state = 12)
X_train_num, X_test_num, y_train, y_test = train_test_split(X_num, y, test_siz
random_state = 12)



imp_cat = SimpleImputer(strategy='most_frequent')
X_train_cat = imp_cat.fit_transform(X_train_cat)
X_test_cat = imp_cat.transform(X_test_cat)



imp_num = SimpleImputer()
X_train_num = imp_num.fit_transform(X_train_num)
X_test_num = imp_num.transform(X_test_num)



X_train = np.append(X_train_num, X_train_cat, axis=1)
X_test = np.append(X_test_num, X_test_cat, axis=1)



columns = ['acousticness', 'danceability', 'duration_ms', 'energy',
'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
'valence', 'genre']
check = pd.DataFrame(X_train, columns = columns)
print(check.isna().sum().sort_values())



knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
print(knn.score(X_test, y_test))
```

```
Index(['popularity', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'genre'],
      dtype='object')
genre                 8
popularity           31
loudness             44
liveness             46
tempo                46
speechiness          59
duration_ms          91
instrumentalness     91
danceability        143
valence             143
acousticness        200
energy              200
dtype: int64
popularity            0
liveness              0
loudness              0
tempo                 0
genre                 0
duration_ms          29
instrumentalness     29
speechiness          53
danceability        127
valence             127
acousticness        178
energy              178
dtype: int64
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
genre               0
dtype: int64
0.011194029850746268
```

In [76]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
music_df = pd.read_csv('music_unclean.csv', index_col =
0)
music_df = music_df.dropna(subset=['genre',
'popularity','loudness','liveness','tempo'])
music_df['genre'] = np.where(music_df['genre'] ==
'Rock', 1, 0)
X = music_df.drop('genre', axis = 1).values
y = music_df['genre'].values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_s
steps = [('imputation', SimpleImputer()),
('Log_reg', LogisticRegression())]
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(pipeline.score(X_test,y_test))
```

```
[[ 79  56]
 [ 23 110]]
0.7052238805970149
```

In [78]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix

music_df = pd.read_csv('music_unclean.csv')

missing_values = music_df.isnull().sum().sort_values(ascending=True)
print("Missing Values per Column:")
print(missing_values)

music_df = music_df.dropna(thresh=len(music_df) - 50, axis=1)

music_df["genre"] = (music_df["genre"] == "Rock").astype(int)

X = music_df.drop(columns=["genre"])
y = music_df["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando

imputer = SimpleImputer(strategy="mean")

knn = KNeighborsClassifier(n_neighbors=3)

steps = [("imputer", imputer), ("knn", knn)]

pipeline = Pipeline(steps)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Missing Values per Column:
Unnamed: 0            0
genre                 8
popularity           31
loudness             44
liveness             46
tempo                46
speechiness          59
duration_ms          91
instrumentalness     91
danceability        143
valence             143
acousticness        200
energy              200
dtype: int64
Confusion Matrix:
[[ 89  50]
```

```
 [ 55 106]]
```

In [79]:
```
#Without using pipeline

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

music_df = pd.read_csv('music_unclean.csv')

missing_values = music_df.isnull().sum().sort_values(ascending=True)
print("Missing Values per Column:")
print(missing_values)
music_df = music_df.dropna(thresh=len(music_df) - 50, axis=1)

music_df["genre"] = (music_df["genre"] == "Rock").astype(int)

X = music_df.drop(columns=["genre"])
y = music_df["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando

imputer = SimpleImputer(strategy="mean")

X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train_imputed, y_train)

y_pred = knn.predict(X_test_imputed)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Missing Values per Column:
Unnamed: 0            0
genre                 8
popularity           31
loudness             44
liveness             46
tempo                46
speechiness          59
duration_ms          91
instrumentalness     91
danceability        143
valence             143
acousticness        200
energy              200
dtype: int64
Confusion Matrix:
[[ 89  50]
 [ 55 106]]
```

In [ ]: