Experiment/Practical 5 K-Nearest Neighbors (KNN)

Title: Implementation of Implementation of K-Nearest Neighbors (KNN) for Classification

Aim: To apply the K-Nearest Neighbors (KNN) algorithm for classification and to understand its mechanism and performance.

Objective: Students will learn

- Implementation of the KNN classification algorithm on the given dataset(s).
- To evaluate and understand the effect of different values of K (the number of neighbors) on model performance.
- To visualize and interpret the results effectively.

Problem statement

Use the given dataset(s) to demonstrate the application of K-Nearest Neighbors (KNN) classification. The task is to identify the class of the query instance using k-nearest neighbors.

Explanation/Stepwise Procedure/ Algorithm:

- Give a brief description of K-Nearest Neighbours

  K-Nearest Neighbours (KNN) is a simple, non-parametric classification algorithm used in machine learning. It classifies a data point based on the majority class among its 'k' closest neighbors in the feature space, typically measured via distance metrics like Euclidean distance. KNN can also be used for regression tasks by averaging the values of the nearest neighbors. It's intuitive and easy to implement but can be computationally expensive, especially with large datasets.

- Give mathematical formulation of K-Nearest Neighbours

  The fundamental principle behind k-NN assumes that similar data points exist in close proximity to each other. Here is a step-by-step breakdown of how the algorithm operates:

  Data Representation: Assume we have a dataset with features (attributes) and corresponding labels (for supervised learning tasks).

  Distance Calculation: For a given data point (or query point) that needs to be classified or predicted, the algorithm calculates the distance between this point and every other point in the dataset. The most common distance metrics used are Euclidean distance, Manhattan distance, minkowski distance.

$$Manhattan\ Distance = \sum_{i=1}^{d} |x_{1i} - x_{2i}|$$

$$Euclidean\ Distance = \left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2\right)^{\frac{1}{2}}$$

$$Minkowski\ Distance = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^p\right)^{\frac{1}{p}}$$

3. Finding Neighbors: Once distances are calculated, the algorithm identifies the k nearest neighbors to the query point.

4. Classification or Regression:

For classification, the algorithm assigns the query point to the class most common among its k nearest neighbors (using majority voting).

For regression, the algorithm predicts the average of the k nearest neighbors' target values.

5. Choosing the Value of k: The value of k is a crucial parameter (hyperparameter) in k-NN. A smaller value of k leads to more complex models (potentially overfitting), whereas a larger value of k makes the model simpler but may miss local patterns. Choosing the right value of k often involves experimentation and validation techniques like cross-validation.

- Write the importance of K-Nearest Neighbours

1. Simplicity: Easy to understand and implement; intuitive mechanism based on proximity.

2. Versatility: Applicable for both classification and regression tasks.

3. Non-parametric: Does not assume any underlying data distribution, useful for complex patterns.

4. No Training Phase: Instantly usable without a separate training step, storing the dataset for predictions.

5. Dynamic Adaptability: Can easily include new data points without needing retraining.

6. Good for Small Datasets: Performs well on smaller datasets, reducing the risk of overfitting.

7. Multi-class Support: Naturally handles multiclass classification problems without alterations.

8. Flexible Distance Metrics: Allows customization of distance calculations (e.g., Euclidean, Manhattan).

9. Baseline Model: Provides a straightforward benchmark for comparing more complex algorithms.

10. Computational Considerations: Sensitive to larger datasets in terms of computation and memory; must consider dimensionality issues (curse of dimensionality).

- Mention applications of K-Nearest Neighbours.

K-Nearest Neighbours (KNN) has a wide range of applications across various domains. Here are some notable applications:

1. Image Recognition: KNN can be used for classifying images based on their features, such as in facial recognition and object detection.

2. Recommendation Systems: KNN helps in recommending products or services by finding similar users or items based on preferences and behaviors.

3. Medical Diagnosis: In healthcare, KNN can aid in diagnosing diseases by comparing patient symptoms and medical history with those of existing cases.

4. Customer Segmentation: Businesses can group customers based on purchasing behavior for targeted marketing and personalized services.

5. Anomaly Detection: KNN can identify unusual data points by measuring their distance from typical cases, useful in fraud detection and network security.

6. Text Classification: In natural language processing, KNN can classify documents or categorize sentiments based on text features.

7. Financial Applications: KNN can be employed for credit scoring, evaluating the likelihood of loan defaults by comparing borrowers with similar profiles.

8. Handwriting Recognition: KNN is used to classify handwritten characters or digits based on feature extraction from images.

- Brief explanation of performance metrics (e.g., accuracy, precision, recall, f1-score, confusion matrix, cross-validation)

Performance Metrics in Machine Learning

1. Accuracy
Accuracy measures the proportion of correctly classified instances out of the total instances. It is useful when the dataset is balanced but can be misleading if the classes are imbalanced.

Formula:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

2. Precision
Precision evaluates how many of the predicted positive instances were actually positive. It is important in scenarios where false positives are costly.

Formula:

$$Precision = \frac{TP}{TP + FP}$$

3. Recall (Sensitivity or True Positive Rate)
Recall measures how many actual positive instances were correctly predicted. It is crucial when missing a positive instance is costly, such as in medical diagnoses.

Formula:

$$Recall = \frac{TP}{TP + FN}$$

4. F1-Score
   The F1-score is the harmonic mean of precision and recall, providing a balanced measure when there is an uneven class distribution.

   Formula:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

5. Confusion Matrix
   A confusion matrix is a table used to evaluate the performance of a classification model by showing the counts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

   Actual Positive - Predicted Positive (TP), Predicted Negative (FN)
   Actual Negative - Predicted Positive (FP), Predicted Negative (TN)

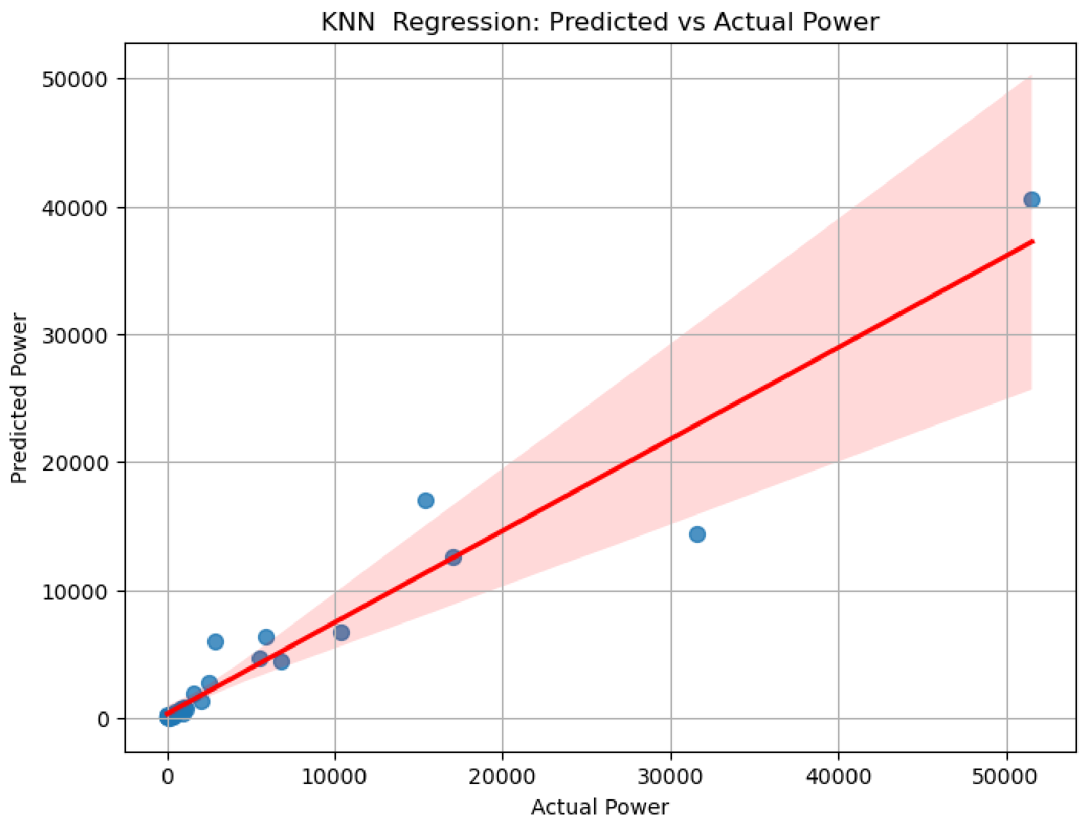
Confusion Matrix

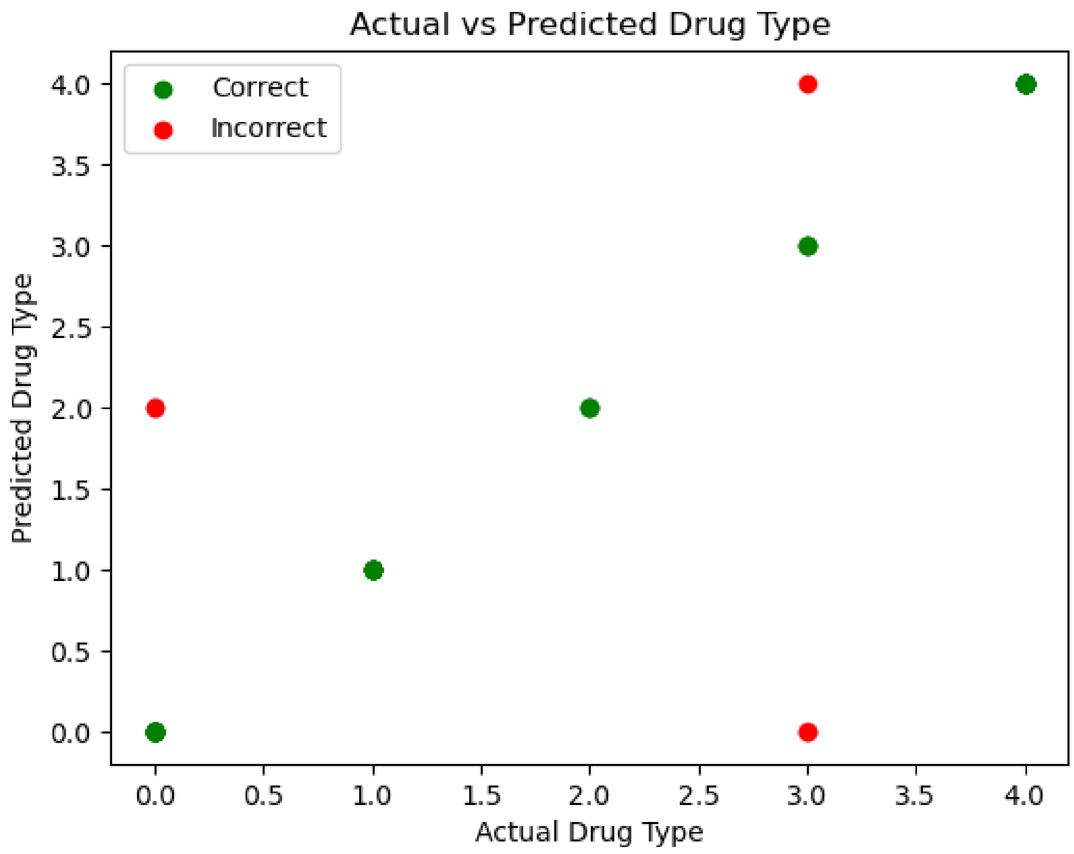
	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

6. Cross-Validation
   Cross-validation is a technique for assessing the performance of a model by splitting the dataset into multiple training and testing subsets. The most common method is k-fold cross-validation, where the dataset is divided into k subsets, and the model is trained k times, each time using a different subset as the test set.

   This technique helps in reducing overfitting and provides a more reliable measure of a model's generalization ability.

- ***Add necessary figure(s)/Diagram(s)***





Input & Output:

Input:

- Dataset: A labeled dataset with features and a target

  Practice dataset 1 KNN regression Hydropower_Consumption.csv

  Practice dataset 2 KNNClassifier drug_classification.csv

- User Input: The number of neighbors K can be selected for optimal performance.

Output:

- Predictions: The predicted class labels for test data.
- Model Evaluation Metrics: Accuracy, confusion matrix, and other metrics as needed.
- Visualizations: Plots showing decision boundaries, confusion matrix, and performance for different values of K.

Conclusion:

Impact of K:

Impact of K in KNN for Drug Classification

In the drug classification study, the choice of K significantly impacts the model's accuracy and generalization. A small K (e.g., K=1) results in a highly sensitive model prone to overfitting, as it closely follows the training data and is affected by noise. On the other hand, a large K (e.g., K=21, as selected through GridSearchCV) helps in smoothing decision boundaries and reducing variance while maintaining a good classification performance. The best model achieved an accuracy of 91.67% using the Manhattan distance metric and weighted voting.

Impact of K in KNN for Hydropower Consumption Prediction

For KNN regression in hydropower consumption prediction, a small K (e.g., K=3) captures local fluctuations well but is susceptible to noise, leading to higher variance. A larger K (e.g., K=12, as selected in the study) smooths predictions by considering more neighbors, reducing sensitivity to outliers while slightly increasing bias. The optimal model achieved an R^2 score of 0.87, balancing bias and variance effectively.

KNN_Hydropower_Cnsumption

February 28, 2025

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import seaborn as sns
```

[2]:
```python
df = pd.read_csv("Practice dataset 1 KNN regression Hydropower_Consumption.csv")
```

[3]:
```python
df
```

[3]:
```
         Country   2000   2001   2002   2003   2004   2005   2006   2007  \
0    Afghanistan    312    498    555     63    565     59    637    748
1         Africa  75246  80864  85181  82873  87405  89066  92241  95341
2        Albania   4548   3519   3477   5117   5411   5319   4951    276
3        Algeria     54     69     57    265    251    555    218    226
4         Angola    903   1007   1132   1229   1733   2197   2638   2472
..           …      …      …      …      …      …      …      …      …
148   Uzbekistan   5879   6017   6186   7155   6493   6876    585   6457
149    Venezuela  62886  60441  59534  60532  70075  77088  81413  83034
150      Vietnam  14551   1821  18198      0  17818  16535      0  22437
151       Zambia   7673   7814   8021   8174   8375   8794   9572   9535
152     Zimbabwe   3227   2968   3786   5305   5466   4866   5257   5329

      2008  …    2010    2011    2012    2013    2014    2015    2016  \
0      542  …     751     595      71     804     895     989    1025
1    97157  …  107427  110445  110952  117673  123727  115801  123816
2     3759  …    7673    4036    4725    6959    4726    5866    7136
3      283  …     173     378     389      99     193     145      72
4     3103  …    3666    3967    3734    4719    4991    5037    5757
..     …  …     …     …     …     …     …     …     …
148   4386  …    8192    5721    6355     627    6185     602    7327
149  86713  …    7666   83155   81736   83405   78747   73397   61699
150  25984  …   28524   41076   53305    5782   62165   57171   66048
151   9427  …   10331   11368   12227   13148   13902   12907   10915
152   5651  …    5741    5149    5336    4946    5377     494    2955
```

```
      2017    2018   2019
0      105     105    107
1   130388  132735      0
2      448     448   4018
3       56     117    152
4     7576    7576   8422
..     …       …      …
148   8427    5897     65
149  59296   56987  63267
150  88762   84485  65563
151  12076   12076  11799
152   3929    3929   3592

[153 rows x 21 columns]
```

[4]:
```
df.shape
```

[4]:
```
(153, 21)
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153 entries, 0 to 152
Data columns (total 21 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Country  153 non-null    object
 1   2000     153 non-null    int64
 2   2001     153 non-null    int64
 3   2002     153 non-null    int64
 4   2003     153 non-null    int64
 5   2004     153 non-null    int64
 6   2005     153 non-null    int64
 7   2006     153 non-null    int64
 8   2007     153 non-null    int64
 9   2008     153 non-null    int64
 10  2009     153 non-null    int64
 11  2010     153 non-null    int64
 12  2011     153 non-null    int64
 13  2012     153 non-null    int64
 14  2013     153 non-null    int64
 15  2014     153 non-null    int64
 16  2015     153 non-null    int64
 17  2016     153 non-null    int64
 18  2017     153 non-null    int64
 19  2018     153 non-null    int64
 20  2019     153 non-null    int64
```

```
dtypes: int64(20), object(1)
memory usage: 25.2+ KB
```

[6]:
```
df.isna().sum()
```

[6]:
```
Country    0
2000       0
2001       0
2002       0
2003       0
2004       0
2005       0
2006       0
2007       0
2008       0
2009       0
2010       0
2011       0
2012       0
2013       0
2014       0
2015       0
2016       0
2017       0
2018       0
2019       0
dtype: int64
```

[7]:
```
df.columns
```

[7]:
```
Index(['Country', '2000', '2001', '2002', '2003', '2004', '2005', '2006',
       '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015',
       '2016', '2017', '2018', '2019'],
      dtype='object')
```

[8]:
```
X = df.drop(columns = ['2019', 'Country'])
y = df['2019']
```

[9]:
```
X = pd.get_dummies(X, drop_first=True)
```

[10]:
```
sc = StandardScaler()
X = sc.fit_transform(X)
```

[11]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 42, shuffle = True)
```

[12]:
```
params = {
    'n_neighbors':[3,5,7,12],
```

```
    'weights' : ['uniform', 'distance'],
    'metric':['minkowski','manhattan','euclidean']
}
```

[13]:
```
dia_reg = GridSearchCV(KNeighborsRegressor(), params, cv = 10)
```

[14]:
```
dia_reg.fit(X_train, y_train)
```

```
c:\Users\Neil\anaconda3\Lib\site-
packages\joblib\externals\loky\backend\context.py:136: UserWarning: Could not
find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by
setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "c:\Users\Neil\anaconda3\Lib\site-
packages\joblib\externals\loky\backend\context.py", line 257, in
_count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^^

  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 548, in run
    with Popen(*popenargs, **kwargs) as process:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "c:\Users\Neil\anaconda3\Lib\subprocess.py", line 1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

[14]:
```
GridSearchCV(cv=10, estimator=KNeighborsRegressor(),
             param_grid={'metric': ['minkowski', 'manhattan', 'euclidean'],
                         'n_neighbors': [3, 5, 7, 12],
                         'weights': ['uniform', 'distance']})
```

[15]:
```
dia_reg.best_score_
```

[15]:
```
-0.7084955694221238
```

[16]:
```
dia_reg.best_params_
```

[16]:
```
{'metric': 'minkowski', 'n_neighbors': 12, 'weights': 'uniform'}
```

[17]:
```
regressor = KNeighborsRegressor(metric = 'manhattan', n_neighbors= 5,␣
 ↪weights='distance')
regressor.fit(X_train, y_train)
```

[17]:
```
KNeighborsRegressor(metric='manhattan', weights='distance')
```

[18]:

```
y_pred = regressor.predict(X_val)
```

[19]:

```
rmse = np.sqrt(mean_squared_error(y_val, y_pred))
rmse
```

[19]: 3884.94001975831

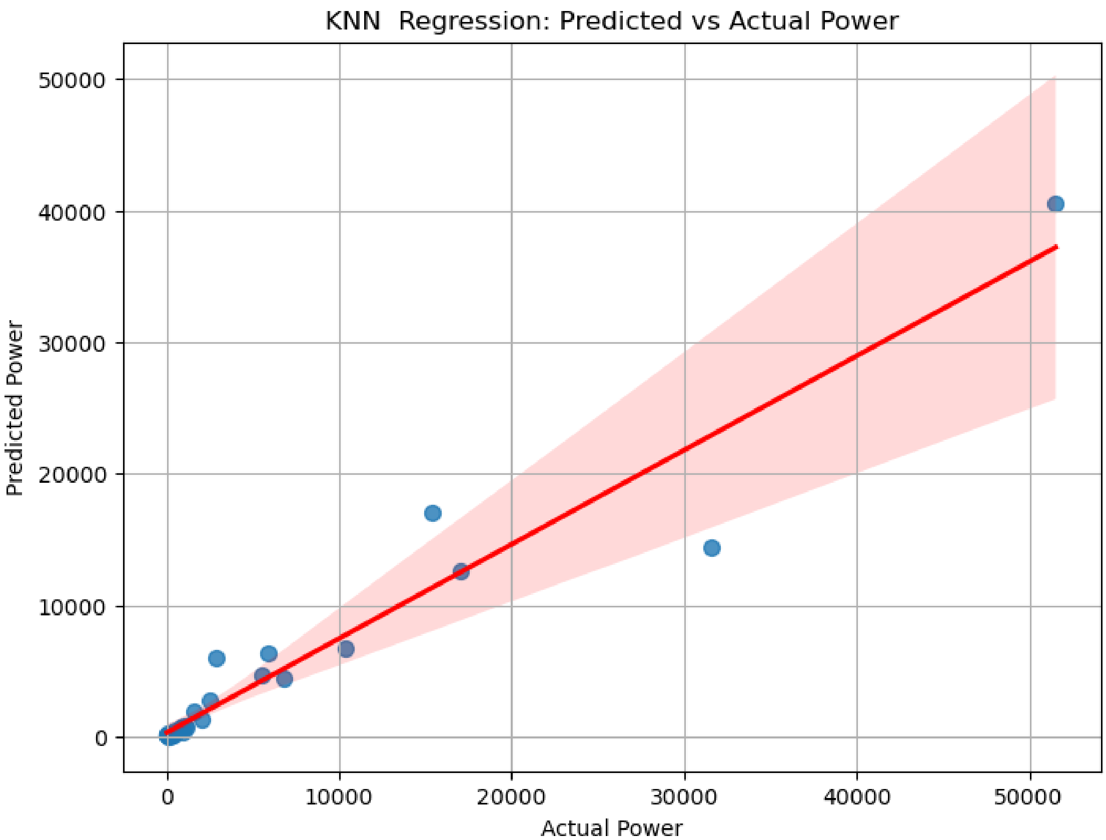
[20]:

```
from sklearn.metrics import mean_squared_error
# Calculate and display Mean Squared Error (MSE)
mse = mean_squared_error(y_val, y_pred)
print("MSE value : {:.4f}".format(mse))
from sklearn.metrics import mean_squared_error
# Calculate and display Root Mean Squared Error (RMSE)
mse = mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse)
print("RMSE value : {:.4f}".format(rmse))

from sklearn.metrics import r2_score
# Calculate and display R2 score (R2)
print("R2 score value : {:.4f}".format(r2_score(y_val, y_pred)))

from sklearn.metrics import mean_absolute_error
# Calculate and display Mean Absolute Error (MAE)
mae = mean_absolute_error(y_val, y_pred)
print("MAE value : {:.4f}".format(mae))
```

```
MSE value : 15092758.9571
RMSE value : 3884.9400
R2 score value : 0.8700
MAE value : 1572.8438
```

[21]:

```
plt.figure(figsize=(8,6))
sns.regplot(x=y_val, y=y_pred, scatter_kws={"s":50}, line_kws={"color": "red"})
plt.xlabel("Actual Power")
plt.ylabel("Predicted Power")
plt.title("KNN  Regression: Predicted vs Actual Power")
plt.grid(True)
plt.show()
```


KNN Regression: Predicted vs Actual Power
Predicted Power
Actual Power
0
10000
20000
30000
40000
50000

KNN_drug_classification

February 28, 2025

[ ]:

```

```

[289]:

```
import pandas as pd
import numpy  as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error,␣
 ↪mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

[290]:

```
url = r"D:\Supervised Machine Learning lab (SMLL)\6\Practice dataset 2␣
 ↪KNNClassifier drug_classification.csv"
df = pd.read_csv(url)
```

[291]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug_Type    200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

[292]:

```
df.head()
```

[292]:

```
   Age Sex     BP Cholesterol  Na_to_K Drug_Type
0   23   F   HIGH        HIGH   25.355     DrugY
```

1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

[293]:
```
df.describe(include='all')
```

[293]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug_Type
count	200.000000	200	200	200	200.000000	200
unique	NaN	2	3	2	NaN	5
top	NaN	M	HIGH	HIGH	NaN	DrugY
freq	NaN	104	77	103	NaN	91
mean	44.315000	NaN	NaN	NaN	16.084485	NaN
std	16.544315	NaN	NaN	NaN	7.223956	NaN
min	15.000000	NaN	NaN	NaN	6.269000	NaN
25%	31.000000	NaN	NaN	NaN	10.445500	NaN
50%	45.000000	NaN	NaN	NaN	13.936500	NaN
75%	58.000000	NaN	NaN	NaN	19.380000	NaN
max	74.000000	NaN	NaN	NaN	38.247000	NaN

[294]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[295]:
```
# Encode categorical columns
df['Sex'] = le.fit_transform(df['Sex'])
df['BP'] = le.fit_transform(df['BP'])
df['Cholesterol'] = le.fit_transform(df['Cholesterol'])
df['Drug_Type'] = le.fit_transform(df['Drug_Type'])

# Display the updated dataframe
df.head()
```

[295]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug_Type
0	23	0	0	0	25.355	0
1	47	1	1	0	13.093	3
2	47	1	1	0	10.114	3
3	28	0	2	0	7.798	4
4	61	0	1	0	18.043	0

[296]:
```
X = df.drop('Drug_Type', axis=1)
y = df['Drug_Type']
```

[297]:
```
x = pd.get_dummies(X, drop_first=True)
```

[298]:
```
sc = StandardScaler()
x = sc.fit_transform(x)
```

[299]:
```
x
```

```
[299]: array([[-1.29159102, -1.040833  , -1.11016894, -0.97043679,  1.28652212],
       [ 0.16269866,  0.96076892,  0.10979693, -0.97043679, -0.4151454 ],
       [ 0.16269866,  0.96076892,  0.10979693, -0.97043679, -0.82855818],
       [-0.988614  , -1.040833  ,  1.32976279, -0.97043679, -1.14996267],
       [ 1.0110343 , -1.040833  ,  0.10979693, -0.97043679,  0.27179427],
       [-1.35218642, -1.040833  ,  1.32976279, -0.97043679, -1.03769314],
       [ 0.28388946, -1.040833  ,  1.32976279, -0.97043679,  0.02643885],
       [-0.20087376,  0.96076892,  0.10979693, -0.97043679, -0.70046821],
       [ 0.9504389 ,  0.96076892,  1.32976279, -0.97043679, -0.12676951],
       [-0.07968296,  0.96076892,  0.10979693,  1.03046381,  0.45567206],
       [ 0.16269866, -1.040833  ,  0.10979693, -0.97043679, -0.59916196],
       [-0.62504158, -1.040833  , -1.11016894,  1.03046381,  0.43221897],
       [-0.07968296,  0.96076892,  0.10979693, -0.97043679, -0.09832049],
       [ 1.79877454, -1.040833  ,  0.10979693, -0.97043679,  0.674105  ],
       [ 0.34448487, -1.040833  ,  1.32976279, -0.97043679, -0.46926791],
       [-1.71575884, -1.040833  , -1.11016894,  1.03046381, -0.0788919 ],
       [ 1.49579753,  0.96076892,  0.10979693,  1.03046381, -0.64245998],
       [-0.07968296,  0.96076892, -1.11016894, -0.97043679, -0.29316156],
       [-1.29159102,  0.96076892,  0.10979693, -0.97043679, -1.21935052],
       [-0.74623239, -1.040833  , -1.11016894,  1.03046381,  1.37242427],
       [ 0.76865269,  0.96076892,  0.10979693,  1.03046381,  0.42236589],
       [ 1.13222511,  0.96076892,  1.32976279, -0.97043679,  1.36451406],
       [ 0.16269866,  0.96076892,  0.10979693,  1.03046381,  2.00995979],
       [ 0.22329406, -1.040833  ,  0.10979693, -0.97043679, -0.14550423],
       [-0.68563699, -1.040833  ,  0.10979693, -0.97043679,  2.41490725],
       [-0.988614  , -1.040833  , -1.11016894,  1.03046381,  0.37809645],
       [-0.80682779,  0.96076892, -1.11016894, -0.97043679,  1.9819271 ],
       [ 0.28388946, -1.040833  ,  1.32976279,  1.03046381, -0.93028076],
       [-0.32206457, -1.040833  ,  0.10979693,  1.03046381,  0.91765633],
       [ 0.04150785,  0.96076892,  0.10979693, -0.97043679,  0.25902691],
       [-1.59456803, -1.040833  ,  1.32976279,  1.03046381, -1.01784822],
       [ 1.79877454,  0.96076892, -1.11016894, -0.97043679, -0.90446848],
       [ 0.28388946,  0.96076892,  0.10979693,  1.03046381, -0.70366006],
       [ 1.25341591, -1.040833  , -1.11016894,  1.03046381,  2.19147839],
       [ 0.52627108,  0.96076892,  1.32976279, -0.97043679, -0.27081868],
       [ 0.10210325,  0.96076892,  1.32976279,  1.03046381, -1.2211546 ],
       [-0.74623239,  0.96076892, -1.11016894,  1.03046381, -0.92139911],
       [-0.32206457,  0.96076892,  0.10979693,  1.03046381, -0.29787994],
       [-0.32206457, -1.040833  ,  1.32976279,  1.03046381, -0.88476233],
       [-1.77635424,  0.96076892,  1.32976279, -0.97043679, -0.97149714],
       [ 1.73817914, -1.040833  ,  1.32976279, -0.97043679,  0.43527203],
       [ 0.82924809, -1.040833  , -1.11016894,  1.03046381, -0.25610845],
       [ 0.34448487,  0.96076892,  1.32976279,  1.03046381, -0.04086736],
       [-1.29159102,  0.96076892,  1.32976279, -0.97043679, -0.53074555],
       [ 0.34448487, -1.040833  ,  1.32976279,  1.03046381, -0.5258884 ],
       [ 1.31401132, -1.040833  ,  1.32976279,  1.03046381, -1.10708099],
       [-0.44325537, -1.040833  , -1.11016894, -0.97043679, -0.41542295],
```

```
[ 1.43520212,  0.96076892,  0.10979693, -0.97043679, -0.80399488],
[-1.29159102,  0.96076892,  1.32976279, -0.97043679,  2.16511101],
[-0.988614  , -1.040833  ,  0.10979693, -0.97043679,  0.51506806],
[ 0.82924809, -1.040833  , -1.11016894, -0.97043679,  0.46233329],
[ 1.37460672,  0.96076892,  1.32976279,  1.03046381, -0.71975804],
[ 1.0716297 ,  0.96076892,  0.10979693,  1.03046381,  1.54020408],
[-1.23099561, -1.040833  , -1.11016894,  1.03046381,  0.32924741],
[ 1.43520212, -1.040833  , -1.11016894,  1.03046381, -0.81815   ],
[-1.10980481, -1.040833  ,  0.10979693, -0.97043679, -0.26707173],
[ 1.25341591,  0.96076892, -1.11016894,  1.03046381, -0.65841918],
[-0.26146916,  0.96076892, -1.11016894, -0.97043679,  1.62943685],
[ 0.9504389 ,  0.96076892,  1.32976279,  1.03046381, -0.83175002],
[-0.62504158,  0.96076892, -1.11016894, -0.97043679,  0.36338623],
[-0.38265997, -1.040833  ,  0.10979693,  1.03046381,  1.91378824],
[-1.23099561,  0.96076892, -1.11016894,  1.03046381, -0.91723584],
[ 1.37460672,  0.96076892,  0.10979693,  1.03046381,  0.63954985],
[ 0.04150785,  0.96076892,  0.10979693,  1.03046381, -1.07058298],
[ 0.9504389 , -1.040833  , -1.11016894, -0.97043679, -0.3860025 ],
[ 1.43520212, -1.040833  ,  1.32976279,  1.03046381,  1.52174691],
[-0.9280186 ,  0.96076892, -1.11016894, -0.97043679, -0.44803523],
[-1.65516344,  0.96076892,  1.32976279,  1.03046381, -0.72891723],
[ 0.58686648,  0.96076892,  1.32976279, -0.97043679,  1.18979546],
[-1.59456803, -1.040833  , -1.11016894,  1.03046381,  1.13678315],
[ 1.55639293,  0.96076892, -1.11016894, -0.97043679, -0.29385544],
[-0.988614  , -1.040833  ,  1.32976279, -0.97043679,  0.4982762 ],
[-1.23099561, -1.040833  ,  1.32976279, -0.97043679, -0.76041931],
[-0.20087376, -1.040833  ,  1.32976279,  1.03046381,  0.94652168],
[-0.80682779,  0.96076892, -1.11016894,  1.03046381,  0.13662675],
[-1.10980481,  0.96076892,  0.10979693,  1.03046381,  0.6695254 ],
[-0.50385078, -1.040833  , -1.11016894, -0.97043679, -0.67812533],
[-1.10980481, -1.040833  , -1.11016894,  1.03046381,  0.42694549],
[-1.53397263, -1.040833  , -1.11016894, -0.97043679, -0.38461474],
[-0.74623239, -1.040833  ,  0.10979693,  1.03046381, -0.72780703],
[ 0.9504389 ,  0.96076892, -1.11016894, -0.97043679, -0.29843504],
[ 1.19282051,  0.96076892,  1.32976279, -0.97043679, -1.15509738],
[-0.74623239, -1.040833  ,  0.10979693, -0.97043679, -0.884346  ],
[-0.38265997, -1.040833  , -1.11016894,  1.03046381, -0.66036204],
[ 0.16269866, -1.040833  ,  0.10979693, -0.97043679, -0.83508063],
[ 0.88984349,  0.96076892, -1.11016894, -0.97043679, -0.29829626],
[ 0.40508027, -1.040833  ,  1.32976279, -0.97043679, -0.34520245],
[ 1.49579753,  0.96076892,  0.10979693, -0.97043679, -0.08416537],
[-0.44325537, -1.040833  , -1.11016894,  1.03046381,  0.97233395],
[ 0.34448487, -1.040833  ,  1.32976279,  1.03046381,  0.1563329 ],
[ 1.0716297 ,  0.96076892,  1.32976279, -0.97043679,  0.0707083 ],
[-0.20087376,  0.96076892, -1.11016894,  1.03046381, -0.12885115],
[-0.9280186 , -1.040833  , -1.11016894, -0.97043679,  1.85480857],
[-0.14027836, -1.040833  ,  0.10979693,  1.03046381,  1.82996772],
```

```
[ 0.70805729,  0.96076892,  0.10979693, -0.97043679, -0.14841852],
[-0.50385078,  0.96076892,  0.10979693,  1.03046381, -0.64676202],
[ 0.82924809, -1.040833  ,  0.10979693, -0.97043679,  3.07561832],
[ 0.70805729, -1.040833  , -1.11016894, -0.97043679,  1.29207314],
[-1.47337723,  0.96076892, -1.11016894,  1.03046381,  2.71369132],
[-1.77635424, -1.040833  , -1.11016894,  1.03046381,  0.08888791],
[-0.80682779,  0.96076892, -1.11016894,  1.03046381, -0.58472929],
[ 0.04150785, -1.040833  , -1.11016894, -0.97043679, -0.44831279],
[-0.988614  , -1.040833  ,  0.10979693, -0.97043679, -0.41042702],
[ 0.70805729,  0.96076892,  1.32976279, -0.97043679, -0.98787267],
[-1.35218642,  0.96076892, -1.11016894,  1.03046381,  1.69438387],
[-0.44325537,  0.96076892,  0.10979693,  1.03046381, -0.98759512],
[-1.35218642,  0.96076892,  1.32976279, -0.97043679, -0.57334968],
[-0.14027836,  0.96076892,  0.10979693, -0.97043679,  0.54518238],
[ 1.67758373,  0.96076892, -1.11016894,  1.03046381, -0.88920315],
[-1.29159102,  0.96076892,  1.32976279, -0.97043679,  0.10623487],
[ 0.34448487,  0.96076892, -1.11016894, -0.97043679, -1.19270559],
[ 0.16269866, -1.040833  ,  1.32976279,  1.03046381, -1.30469757],
[-0.56444618,  0.96076892,  0.10979693,  1.03046381, -0.95956243],
[ 1.25341591, -1.040833  ,  0.10979693,  1.03046381, -0.32133303],
[-1.47337723, -1.040833  ,  1.32976279,  1.03046381, -0.94415833],
[ 0.40508027,  0.96076892, -1.11016894, -0.97043679,  0.30676574],
[ 1.37460672,  0.96076892,  1.32976279,  1.03046381, -0.91182359],
[-0.26146916, -1.040833  ,  1.32976279, -0.97043679, -0.83008471],
[-0.74623239, -1.040833  , -1.11016894,  1.03046381, -0.8038561 ],
[ 1.0110343 , -1.040833  , -1.11016894, -0.97043679,  1.3031752 ],
[-0.988614  ,  0.96076892,  1.32976279, -0.97043679,  1.52368977],
[-1.77635424,  0.96076892, -1.11016894,  1.03046381,  0.15563902],
[-0.62504158,  0.96076892,  1.32976279, -0.97043679,  0.88421139],
[-0.50385078, -1.040833  ,  1.32976279, -0.97043679,  0.09277363],
[ 0.52627108, -1.040833  , -1.11016894,  1.03046381, -0.49813326],
[-1.53397263, -1.040833  , -1.11016894,  1.03046381,  1.37173039],
[ 1.31401132,  0.96076892, -1.11016894, -0.97043679,  0.0364307 ],
[-0.56444618,  0.96076892,  1.32976279,  1.03046381, -1.14344022],
[ 0.16269866,  0.96076892,  0.10979693,  1.03046381,  2.42267869],
[-0.74623239, -1.040833  ,  1.32976279, -0.97043679, -1.19450967],
[ 1.55639293, -1.040833  ,  1.32976279, -0.97043679,  0.61123961],
[ 0.46567567,  0.96076892,  0.10979693,  1.03046381,  2.33663776],
[ 0.28388946,  0.96076892,  0.10979693,  1.03046381, -0.34506367],
[-1.23099561,  0.96076892,  1.32976279, -0.97043679,  1.34633444],
[-0.14027836, -1.040833  , -1.11016894, -0.97043679,  0.68714991],
[ 1.79877454,  0.96076892,  0.10979693,  1.03046381, -0.57529254],
[ 0.64746188, -1.040833  , -1.11016894, -0.97043679, -0.70879476],
[-0.56444618, -1.040833  , -1.11016894, -0.97043679, -0.44276176],
[ 0.40508027,  0.96076892, -1.11016894,  1.03046381, -0.65800285],
[ 1.49579753, -1.040833  ,  1.32976279, -0.97043679, -0.83535819],
[ 0.28388946,  0.96076892, -1.11016894,  1.03046381, -1.36215071],
```

```
[ 1.19282051, -1.040833  ,  0.10979693,  1.03046381,  1.34008953],
[ 0.9504389 ,  0.96076892, -1.11016894,  1.03046381, -1.03575028],
[ 1.79877454,  0.96076892, -1.11016894,  1.03046381, -0.08999395],
[-0.32206457,  0.96076892, -1.11016894, -0.97043679, -0.89100724],
[ 1.0110343 ,  0.96076892,  1.32976279, -0.97043679, -0.92167666],
[-0.44325537, -1.040833  ,  0.10979693,  1.03046381, -0.56599457],
[-1.10980481, -1.040833  , -1.11016894,  1.03046381, -0.52422309],
[ 1.0110343 , -1.040833  ,  0.10979693,  1.03046381, -1.21352194],
[-1.35218642,  0.96076892,  0.10979693, -0.97043679, -1.10097486],
[ 0.28388946,  0.96076892, -1.11016894,  1.03046381, -1.024787  ],
[ 1.43520212,  0.96076892, -1.11016894, -0.97043679, -0.70435393],
[ 0.64746188,  0.96076892,  1.32976279,  1.03046381, -1.22448522],
[ 1.67758373, -1.040833  ,  0.10979693,  1.03046381, -0.20018185],
[-0.44325537,  0.96076892,  0.10979693,  1.03046381,  0.08874914],
[ 0.28388946,  0.96076892,  0.10979693, -0.97043679, -0.76985606],
[-0.80682779,  0.96076892, -1.11016894,  1.03046381, -0.67410083],
[ 0.52627108,  0.96076892,  0.10979693, -0.97043679,  0.95457067],
[ 0.88984349, -1.040833  ,  0.10979693, -0.97043679, -0.7827622 ],
[-0.62504158, -1.040833  ,  0.10979693,  1.03046381, -0.43873726],
[-0.8674232 , -1.040833  ,  1.32976279, -0.97043679, -0.78290097],
[ 0.76865269, -1.040833  , -1.11016894,  1.03046381, -0.85201127],
[-0.07968296,  0.96076892,  1.32976279,  1.03046381, -0.44761891],
[-1.41278182, -1.040833  , -1.11016894,  1.03046381,  1.74129005],
[-1.71575884,  0.96076892, -1.11016894,  1.03046381,  0.40557404],
[-0.38265997,  0.96076892,  0.10979693, -0.97043679,  0.30676574],
[ 0.82924809, -1.040833  ,  0.10979693, -0.97043679,  1.46554276],
[ 0.76865269, -1.040833  ,  1.32976279, -0.97043679, -0.2593003 ],
[ 0.40508027, -1.040833  ,  0.10979693,  1.03046381,  0.96012169],
[-1.47337723, -1.040833  , -1.11016894, -0.97043679, -0.66924368],
[-0.988614  , -1.040833  ,  1.32976279, -0.97043679, -0.44484339],
[ 0.04150785,  0.96076892,  0.10979693,  1.03046381, -0.84201942],
[-0.32206457, -1.040833  ,  1.32976279,  1.03046381,  0.15827576],
[-0.20087376, -1.040833  ,  0.10979693,  1.03046381,  0.36838215],
[-0.14027836,  0.96076892, -1.11016894,  1.03046381, -0.46052505],
[ 1.73817914, -1.040833  , -1.11016894, -0.97043679,  0.31412086],
[ 0.22329406,  0.96076892, -1.11016894,  1.03046381, -0.78248465],
[-1.17040021,  0.96076892,  1.32976279, -0.97043679,  0.40612914],
[-0.32206457,  0.96076892,  1.32976279, -0.97043679, -0.01602651],
[ 1.37460672, -1.040833  ,  1.32976279, -0.97043679, -0.02685101],
[-1.35218642, -1.040833  , -1.11016894,  1.03046381,  0.93444819],
[ 0.88984349, -1.040833  ,  1.32976279, -0.97043679, -0.30537382],
[-1.47337723, -1.040833  ,  0.10979693,  1.03046381, -0.61040279],
[-0.50385078, -1.040833  , -1.11016894,  1.03046381, -0.08250007],
[-1.59456803, -1.040833  , -1.11016894, -0.97043679,  2.92865486],
[ 0.76865269, -1.040833  ,  1.32976279,  1.03046381,  1.36118344],
[ 1.55639293,  0.96076892, -1.11016894, -0.97043679, -0.86533373],
[ 0.16269866,  0.96076892, -1.11016894, -0.97043679, -0.788452  ],
```

```
       [ 1.25341591,  0.96076892, -1.11016894,  1.03046381,  2.62459732],
       [ 1.19282051,  0.96076892, -1.11016894,  1.03046381,  0.67271724],
       [ 0.82924809,  0.96076892, -1.11016894, -0.97043679,  0.40335363],
       [-1.29159102,  0.96076892, -1.11016894, -0.97043679, -1.12040345],
       [ 1.67758373,  0.96076892,  0.10979693, -0.97043679,  0.031296  ],
       [ 1.67758373,  0.96076892,  0.10979693, -0.97043679, -1.29276286],
       [ 0.10210325, -1.040833  , -1.11016894, -0.97043679,  2.58143808],
       [ 0.70805729, -1.040833  ,  0.10979693, -0.97043679, -0.6269171 ],
       [-1.71575884,  0.96076892,  0.10979693, -0.97043679, -0.56599457],
       [ 0.46567567,  0.96076892,  1.32976279, -0.97043679, -0.85908883],
       [-1.29159102,  0.96076892,  1.32976279,  1.03046381, -0.28650033],
       [-0.26146916, -1.040833  ,  0.10979693,  1.03046381, -0.6571702 ]])
```

[300]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
 ↪random_state=42, shuffle=True)
```

[301]:
```
params = {
    'n_neighbors':np.array(range(1, 50)),
    'weights' : ['uniform', 'distance'],
    'metric':['minkowski','manhattan','euclidean']
}
```

[302]:
```
from sklearn.neighbors import KNeighborsClassifier
dia_reg = GridSearchCV(KNeighborsClassifier(), params, cv = 10)
```

[303]:
```
dia_reg.fit(X_train, y_train)
```

[303]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['minkowski', 'manhattan', 'euclidean'],
                         'n_neighbors': array([ 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, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                         'weights': ['uniform', 'distance']})
```

[304]:
```
dia_reg.best_score_
```

[304]:
```
0.8928571428571429
```

[305]:
```
dia_reg.best_params_
```

[305]:
```
{'metric': 'manhattan', 'n_neighbors': 21, 'weights': 'distance'}
```

[306]:
```
regressor = KNeighborsClassifier(metric = 'manhattan', n_neighbors= 21,
 ↪weights='distance')
regressor.fit(X_train, y_train)
```

[306]:
```
KNeighborsClassifier(metric='manhattan', n_neighbors=21, weights='distance')
```

[307]:
```python
y_pred = regressor.predict(X_test)
```

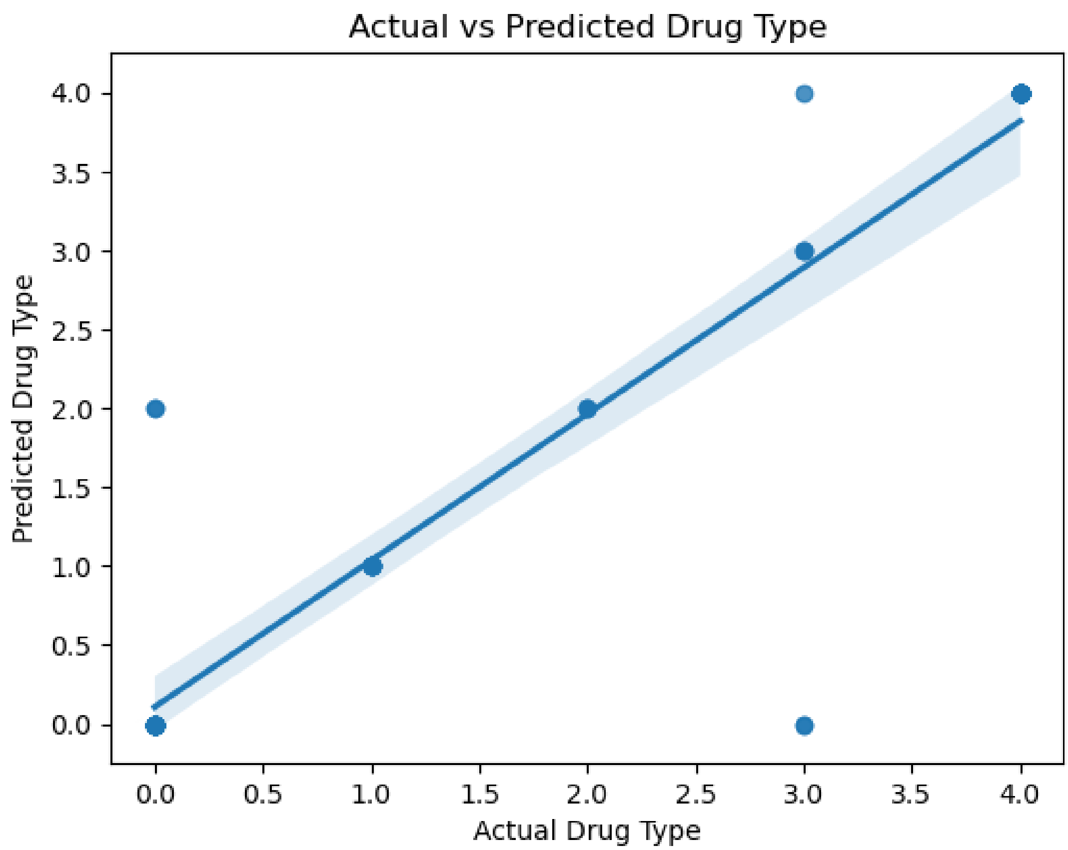
[308]:
```python
from sklearn.metrics import accuracy_score
# Calculate and display accuracy score
print("Accuracy score : {:.4f}".format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score : 0.9167
```

[309]:
```python
from sklearn.metrics import confusion_matrix, classification_report

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

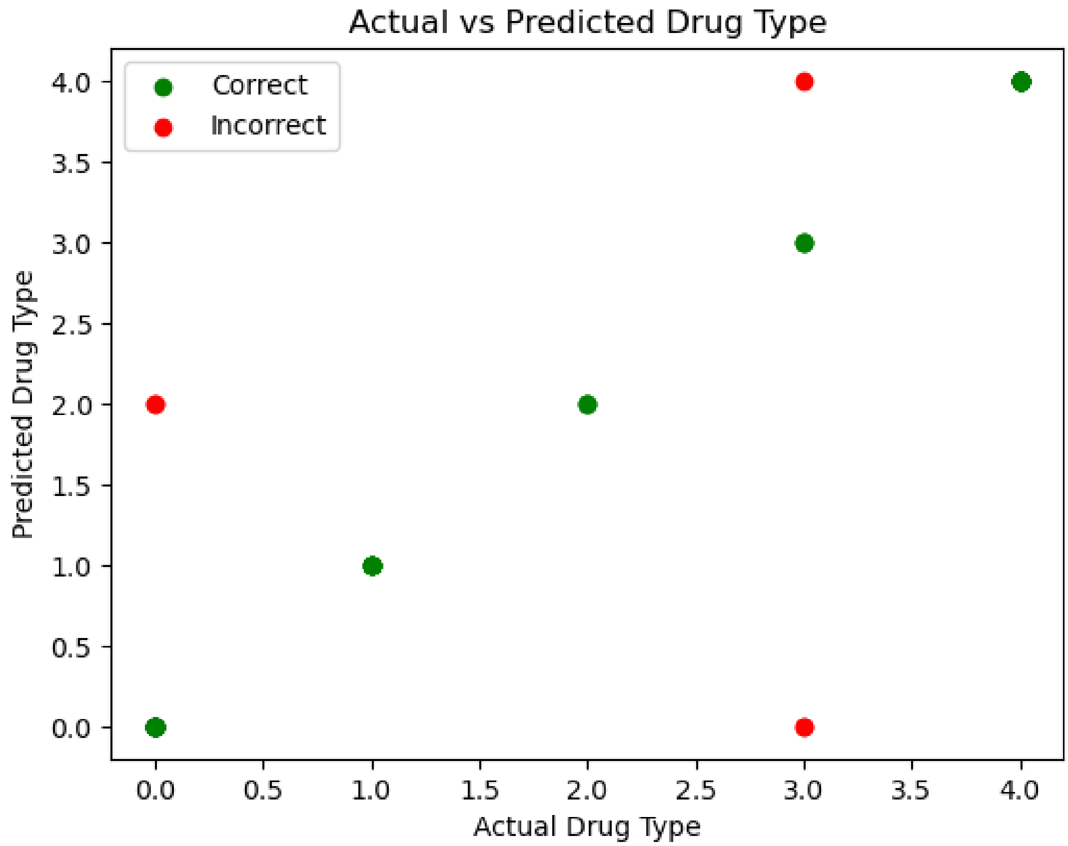
```
Confusion Matrix:
[[24  0  2  0  0]
 [ 0  7  0  0  0]
 [ 0  0  3  0  0]
 [ 2  0  0  3  1]
 [ 0  0  0  0 18]]

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        26
           1       1.00      1.00      1.00         7
           2       0.60      1.00      0.75         3
           3       1.00      0.50      0.67         6
           4       0.95      1.00      0.97        18

    accuracy                           0.92        60
   macro avg       0.89      0.88      0.86        60
weighted avg       0.93      0.92      0.91        60


Accuracy Score:
0.9166666666666666
```

[310]:
```python
# Plot the regplot for the predictions
sns.regplot(x=y_test, y=y_pred)
plt.xlabel('Actual Drug Type')
plt.ylabel('Predicted Drug Type')
plt.title('Actual vs Predicted Drug Type')
plt.show()
```



[311]:

```
# Create a boolean array indicating correct predictions
correct = y_test == y_pred

# Plot the correct predictions in green
plt.scatter(y_test[correct], y_pred[correct], color='green', label='Correct')

# Plot the incorrect predictions in red
plt.scatter(y_test[~correct], y_pred[~correct], color='red', label='Incorrect')

# Add labels and title
plt.xlabel('Actual Drug Type')
plt.ylabel('Predicted Drug Type')
plt.title('Actual vs Predicted Drug Type')

# Add legend
plt.legend()

# Show plot
plt.show()
```


Actual vs Predicted Drug Type
Correct
Incorrect
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Drug Type
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Actual Drug Type