In [47]:
```
import pandas as pd
import sqlite3
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
#Loading the database into dataframe
c = sqlite3.connect('database.sqlite')

# List all tables in the database
df = pd.read_sql_query("SELECT * FROM sqlite_master WHERE type='table'",c)

# Output dataframe
df
```

Out[2]:

	type	name	tbl_name	rootpage	sql
0	table	sqlite_sequence	sqlite_sequence	4	CREATE TABLE sqlite_sequence(name,seq)
1	table	Player_Attributes	Player_Attributes	11	CREATE TABLE "Player_Attributes" (\n\t\tIN...
2	table	Player	Player	14	CREATE TABLE *Player* (\n\t\tINTEGER PRIMA...
3	table	Match	Match	18	CREATE TABLE *Match* (\n\t\tINTEGER PRIMAR...
4	table	League	League	24	CREATE TABLE *League* (\n\t\tINTEGER PRIMA...
5	table	Country	Country	26	CREATE TABLE *Country* (\n\t\tINTEGER PRIM...
6	table	Team	Team	29	CREATE TABLE "Team" (\n\t\tINTEGER PRIMARY...
7	table	Team_Attributes	Team_Attributes	2	CREATE TABLE *Team_Attributes* (\n\t\tINTE...

In [4]:
```
player_attr_df = pd.read_sql("SELECT * FROM Player_Attributes",c)
player_attr_df.fillna(11, inplace=True)
```

In [5]:
```
player_attr_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183978 entries, 0 to 183977
Data columns (total 42 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   id                   183978 non-null  int64
 1   player_fifa_api_id   183978 non-null  int64
 2   player_api_id        183978 non-null  int64
 3   date                 183978 non-null  object
 4   overall_rating       183978 non-null  float64
 5   potential            183978 non-null  float64
 6   preferred_foot       183978 non-null  object
 7   attacking_work_rate  183978 non-null  object
 8   defensive_work_rate  183978 non-null  object
 9   crossing             183978 non-null  float64
 10  finishing            183978 non-null  float64
 11  heading_accuracy     183978 non-null  float64
 12  short_passing        183978 non-null  float64
 13  volleys              183978 non-null  float64
 14  dribbling            183978 non-null  float64
 15  curve                183978 non-null  float64
 16  free_kick_accuracy   183978 non-null  float64
 17  long_passing         183978 non-null  float64
 18  ball_control         183978 non-null  float64
 19  acceleration         183978 non-null  float64
 20  sprint_speed         183978 non-null  float64
 21  agility              183978 non-null  float64
 22  reactions            183978 non-null  float64
 23  balance              183978 non-null  float64
 24  shot_power           183978 non-null  float64
 25  jumping              183978 non-null  float64
 26  stamina              183978 non-null  float64
 27  strength             183978 non-null  float64
 28  long_shots           183978 non-null  float64
 29  aggression           183978 non-null  float64
 30  interceptions        183978 non-null  float64
 31  positioning          183978 non-null  float64
 32  vision               183978 non-null  float64
 33  penalties            183978 non-null  float64
 34  marking              183978 non-null  float64
 35  standing_tackle      183978 non-null  float64
 36  sliding_tackle       183978 non-null  float64
 37  gk_diving            183978 non-null  float64
 38  gk_handling          183978 non-null  float64
 39  gk_kicking           183978 non-null  float64
 40  gk_positioning       183978 non-null  float64
 41  gk_reflexes          183978 non-null  float64
dtypes: float64(35), int64(3), object(4)
memory usage: 59.0+ MB
```

Using 'gk_kicking' , 'gk_handling' and 'gk_reflexes'

In [35]:
```
player_attr_df['gk_reflexes'].head()
```

Out[35]:
```
0    8.0
1    8.0
2    8.0
3    7.0
4    7.0
Name: gk_reflexes, dtype: float64
```

In [13]:
```
player_attr_df['gk_reflexes'].min()
```

Out[13]:
```
1.0
```

In [14]:
```
player_attr_df['gk_reflexes'].max()
```

Out[14]:
```
96.0
```

Using Bins to create Categories

In [16]:
```
binInterval = [0.0, 24.0, 48.0, 72.0 , 96.0]
binLabels   = [ 1, 2, 3, 4]
player_attr_df['binned_gk_reflexes'] = pd.cut(player_attr_df['gk_reflexes'], bins = binInterval, labels=binLabels)
player_attr_df['binned_gk_reflexes'] = pd.Categorical(player_attr_df.binned_gk_reflexes)
```

In [38]:
```
df1 = player_attr_df[['binned_gk_reflexes','gk_kicking','gk_handling' ]]
```

In [39]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183978 entries, 0 to 183977
Data columns (total 3 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   binned_gk_reflexes  183978 non-null  category
 1   gk_kicking          183978 non-null  float64
 2   gk_handling         183978 non-null  float64
dtypes: category(1), float64(2)
memory usage: 3.0 MB
```

In [40]:
```
df1.isnull().sum()
```

Out[40]:
```
binned_gk_reflexes    0
gk_kicking            0
gk_handling           0
dtype: int64
```

In [42]:
```
x = df1[['gk_kicking','gk_handling']]
y = df1['binned_gk_reflexes']
```

Train Test Split

In [43]:
```
X_train, X_test, y_train, y_test= train_test_split(x, y, test_size = 0.3)  # Your Code Here
sc= StandardScaler()
sc.fit(X_train)
X_train_std= sc.fit_transform(X_train) # Your Code Here
X_test_std= sc.fit_transform(X_test) # Your Code Here
```

Logistic Regression

In [44]:
```
lr= LogisticRegression(C=1000.0, random_state=0,max_iter=1000)
lr.fit(X_train_std, y_train.ravel())
y_pred= lr.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9820089140123927
```

SVM

In [45]:
```
svm= SVC(kernel='linear', C=1.0, random_state=0, cache_size=7000)
svm.fit(X_train_std, y_train.ravel())
y_pred = svm.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9819002065441896
```

Decission Tree

In [50]:
```
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train_std, y_train.ravel())
y_pred= dt.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9783309780048556
```

KNN

In [51]:
```
knn = KNeighborsClassifier()
knn.fit(X_train_std, y_train.ravel())
y_pred= knn.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9776243794615357
```

Using only 'gk_kicking' and 'gk_handling'

Repeating all the above steps But including only 'gk_kicking' and 'gk_handling'

In [52]:
```
df1['gk_kicking'].min()
```

Out[52]:
```
1.0
```

In [53]:
```
df1['gk_kicking'].max()
```

Out[53]:
```
97.0
```

Using Bins to create Categories

In [104]:
```
binInterval = [0.0, 24.3, 48.5, 72.8 , 97.0]
binLabels   = [ 1, 2, 3, 4]
player_attr_df['binned_gk_kicking'] = pd.cut(player_attr_df['gk_kicking'], bins = binInterval, labels=binLabels)
player_attr_df['binned_gk_kicking'] = pd.Categorical(player_attr_df.binned_gk_kicking)
```

In [113]:
```
x = player_attr_df['gk_handling']
y = player_attr_df['binned_gk_kicking']
```

Train Test Split

In [106]:
```
X_train, X_test, y_train, y_test= train_test_split(x, y, test_size = 0.3)  # Your Code Here
sc= StandardScaler()
X_train = X_train.values.reshape(-1, 1)
X_test = X_test.values.reshape(-1, 1)
sc.fit(pd.DataFrame(X_train))
X_train_std= sc.fit_transform(X_train) # Your Code Here
X_test_std= sc.fit_transform(X_test) # Your Code Here
```

Logistic Regression

In [107]:
```
lr= LogisticRegression(C=1000.0, random_state=0,max_iter=1000)
lr.fit(X_train_std, y_train.ravel())
y_pred= lr.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9177809182157481
```

SVM

In [108]:
```
svm= SVC(kernel='linear', C=1.0, random_state=0, cache_size=7000)
svm.fit(X_train_std, y_train.ravel())
y_pred = svm.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9177809182157481
```

Decission Tree

In [109]:
```
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train_std, y_train.ravel())
y_pred= dt.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9189948182277349
```

KNN

In [111]:
```
knn = KNeighborsClassifier()
knn.fit(X_train_std, y_train.ravel())
y_pred= knn.predict(X_test_std)

print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.9139761568286408
```

Since this assignment (Classification) and the previous assignment (Regression) are with the same data, can you compare and conclude which technique is yielding best results?

It is very clear that when two independent variables are considered in this case 'gk_kicking','gk_handling' , dependent variable as the 'binned_gk_reflexes', Logistic regression; the most basic classification algorithm gave the best result of 98 % accuracy . When compared to the previous assignment Linear Regression algorithm.

But when we set 'gk_handling' as independent variable and 'binned_gk_kicking' as the target variable , as we did in the last assignmnet taking only these two variable with only diffrence the target variable was Continious in the last assignmnet , But here we have modified it to a Categorical variable . The Linear regression model performed better than the Logistic Regression Model, This says that the data was a Regression data and not Classification data .

In [114]:
```
!jupyter-nbconvert --to PDFviaHTML Assignment6_sharanbasav.ipynb
```

```
[NbConvertApp] Converting notebook Assignment6_sharanbasav.ipynb to PDFviaHTML
[NbConvertApp] Writing 293233 bytes to Assignment6_sharanbasav.pdf
```

In [ ]:
```

```