Bharati Vidyapeeth (Deemed to be University)
Department of Engineering and Technology
Kharghar, Navi Mumbai

Department of Computer Science and Engineering (AIML)

PBL REPORT

Name: Smit More	**Roll No: 03**
Subject: Machine Learning(ML)	**Class/Batch: TYAIML/B1**
Date of Performance:	**Date of Submission:**

AIM

Dataset used:

ODI Cricket Matches(https://www.kaggle.com/datasets/jaykay12/odi-cricket-matches-19712017)

Dataset Description

The Cricket Match Dataset contains information about various cricket matches, including teams, venues, and outcomes. The target variable, Winner, indicates the team that won the match.

Features:

- **Scorecard**: Unique identifier for each match**.**
- **Team 1:** The first team that played in the match.
- **Team 2:** The second team that played in the match.
- **Margin:** Indicates how much one team won by (e.g., runs or wickets).
- **Ground:** The location where the match was played.
- **Match Date:** The date when the match occurred.
- **Winner:** Target variable indicating the winning team.
- **Venue_Team1:** The venue associated with Team 1.
- **Venue_Team2:** The venue associated with Team 2.
- **Innings_Team1:** Runs scored by Team 1 in their innings.
- **Innings_Team2:** Runs scored by Team 2 in their innings.
- **Host_Country:** The country hosting the match (may be dropped if not needed).

1. Descriptive Statistics

The summary statistics of the dataset offer valuable information regarding its central tendency, variability, and overall distribution by utilizing the .describe() function.

Data Types and Missing Values

To identify missing data within the dataset, the .isnull() function can be employed, highlighting any entries that are absent.

2. Preprocessing

Encoding Categorical Variables

Categorical features are transformed into numeric values utilizing **LabelEncoder**, allowing machine learning models to interpret these variables effectively.

Feature and Target Variable Separation

The dataset is divided into features (X) and the target variable (y), ensuring clear distinctions for model training.

Correlation Analysis

To understand the relationships between features, the **.corr()** method calculates the correlation coefficients.

3. Model Selection

The model selected for this analysis is the **Decision Tree Classifier**, which is particularly advantageous due to its ability to handle both numerical and categorical data without the need for feature scaling. The dataset is divided into training (70%) and testing (30%) sets to facilitate the evaluation of the model's performance on unseen data.

4. Model Evaluation

Predictions

Predictions are made on the test set following the training of the Decision Tree Classifier, providing insights into the model's performance.

Performance Metrics

Accuracy Score: This metric indicates the proportion of correct classifications among the total predictions made by the model.

R^2 Score: This statistic measures the variance in the target variable that can be explained by the features, giving an indication of the model's explanatory power.

Classification Report
The classification report outlines key metrics such as precision, recall, F1-score, and support for each class, offering a comprehensive view of the model's classification performance.

Confusion Matrix
The confusion matrix presents a summary of the correct and incorrect predictions made by the model, helping to visualize the performance across different classes and identify areas for improvement.

5. Visualization

Bar Plot
A bar plot was utilized to illustrate the number of matches played by each team. This visualization effectively conveys the frequency of matches for each team, helping to identify which teams are more actively participating in ODIs.

Heatmap
A heatmap was employed to display the correlation between various features in the dataset. By visualizing the relationships between features, this tool aids in understanding how different attributes interact with one another, assisting in feature selection for modeling purposes.

Decision Tree Plot
The decision tree plot visually represents the structure of the decision tree classifier. It showcases how decisions are made based on different feature values, enhancing the interpretability of the model and providing insights into the decision-making process.

Bharati Vidyapeeth (Deemed to be University)
Department of Engineering and Technology
Kharghar, Navi Mumbai

Department of Computer Science and Engineering (AIML)

Code

Step1: Load dataset

```
import pandas as pd
data = pd.read_csv('ContinousDataset.csv')
dfs = pd.DataFrame(data)
dfs.head()
```

ımed: 0	Scorecard	Team 1	Team 2	Margin	Ground	Match Date	Winner	Host_Country	Venue_Team1	Venue_Team2	Innings_Team1	Innings_Team2
0	ODI # 1	Australia	England	Winner2ndInning	Melbourne	Jan 5, 1971	Australia	Australia	Home	Away	Second	First
1	ODI # 2	England	Australia	Winner2ndInning	Manchester	Aug 24, 1972	England	England	Home	Away	Second	First
2	ODI # 3	England	Australia	Winner2ndInning	Lord's	Aug 26, 1972	Australia	England	Home	Away	First	Second
3	ODI # 4	England	Australia	Winner2ndInning	Birmingham	Aug 28, 1972	England	England	Home	Away	Second	First
4	ODI # 5	New Zealand	Pakistan	Winner1stInning	Christchurch	Feb 11, 1973	New Zealand	New Zealand	Home	Away	First	Second

Step2: Dataset Description

```
df1 = dfs.copy()
df1 = df1.drop(columns=['Unnamed: 0','Host_Country'])
df1.head()
```

	Scorecard	Team 1	Team 2	Margin	Ground	Match Date	Winner	Venue_Team1
0	ODI # 1	Australia	England	Winner2ndInning	Melbourne	Jan 5, 1971	Australia	Home
1	ODI # 2	England	Australia	Winner2ndInn			England	Home
2	ODI # 3	England	Australia	Winner2ndInn			Australia	Home
3	ODI # 4	England	Australia	Winner2ndInn			England	Home
4	ODI # 5	New Zealand	Pakistan	Winner1stInn			Zealand	Home

```
df1.isnull().sum()
```

```
Scorecard       0
Team 1          0
Team 2          0
Margin          0
Ground          0
Match Date      0
Winner          0
Venue_Team1     0
Venue_Team2     0
Innings_Team1   0
Innings_Team2   0
dtype: int64
```

Step3: Preprocessing

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df2 = ['Winner','Venue_Team1', 'Venue_Team2', 'Innings_Team1', 'Innings_Team2', 'Team 1', 'Team 2']
for col in df2:
    df1[col + '_encoded'] = label_encoder.fit_transform(df1[col])
df1.head()
```

	Winner_encoded	Venue_Team1_encoded	Venue_Team2_encoded	Innings_Team1_encoded	Innings_Team2_encoded	Team 1_encoded	Team 2_encoded
0	1	1	0	1	0	1	6
1	6	1	0	1	0	6	1
2	1	1	0	0	1	6	1
3	6	1	0	1	0	6	1
4	13	1	0	0	1	13	15

Step4: Split Dataset

```
from sklearn.model_selection import train_test_split, cross_val_score

x = df1
y = df1['Winner_encoded']
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=0.2,random_state=42)
y.head()
```

```
0     1
1     6
2     1
3     6
4    13
Name: Winner_encoded, dtype: int32
```

Step5: Training Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Support Vector Classifier": SVC(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}

for name, clf in classifiers.items():
    clf.fit(xtrain, ytrain)
    y_pred = clf.predict(xtest)
    accuracy = accuracy_score(ytest, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")
    print("-" * 60)
```

```
Logistic Regression Accuracy: 0.5864
------------------------------------------------------------
Support Vector Classifier Accuracy: 0.9166
------------------------------------------------------------
K-Nearest Neighbors Accuracy: 0.9473
------------------------------------------------------------
Decision Tree Accuracy: 0.9993
------------------------------------------------------------
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dt = DecisionTreeClassifier(random_state=42)
dt.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Step6: Model evaluation

```
y_pred = dt.predict(xtest)
accuracy = accuracy_score(ytest, y_pred)
print(f"Decision Tree Prediction Accuracy: {accuracy:.4f}")
```

```
Decision Tree Prediction Accuracy: 0.9993
```

```
from sklearn.metrics import classification_report, r2_score, confusion_matrix
print('R2 Score: ',r2_score(ytest, y_pred))
print('Classification Report:\n', classification_report(ytest, y_pred))
print('Confusion Matrix:\n', confusion_matrix(ytest, y_pred))
```

R2 Score: 0.9999856537627223

Classification Report:

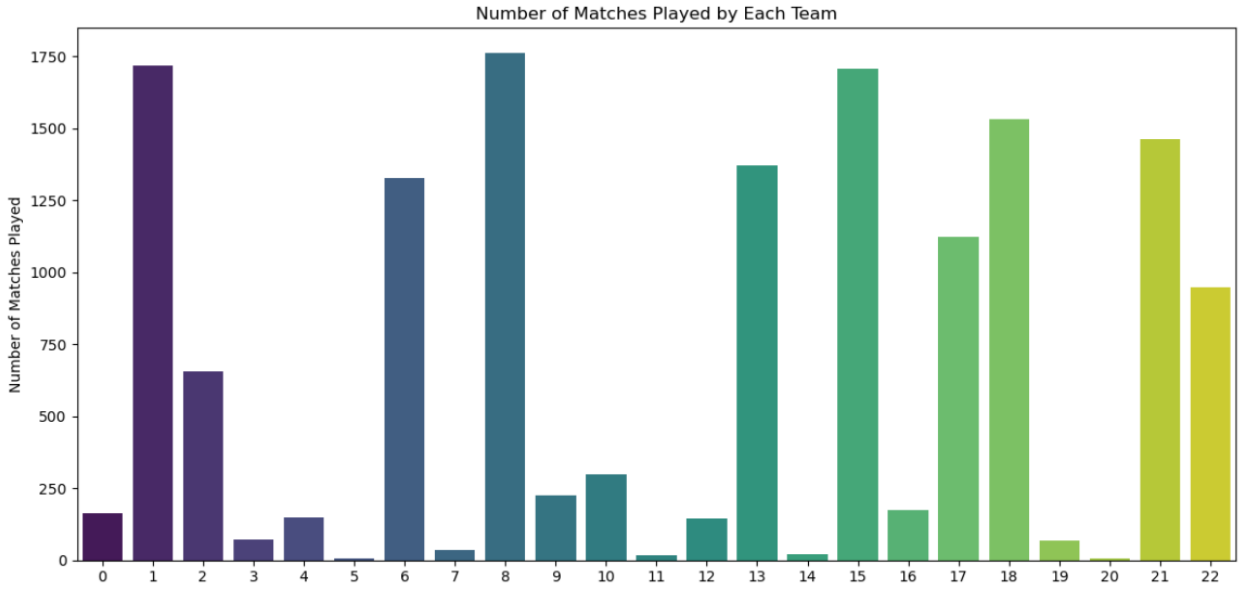
	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	1.00	1.00	1.00	218
2	1.00	1.00	1.00	34
3	1.00	1.00	1.00	2
4	1.00	1.00	1.00	8
6	1.00	1.00	1.00	137
7	1.00	1.00	1.00	3
8	1.00	1.00	1.00	204
9	1.00	1.00	1.00	16
10	1.00	1.00	1.00	17
12	1.00	1.00	1.00	9
13	1.00	1.00	1.00	139
14	1.00	1.00	1.00	1
15	1.00	1.00	1.00	193
16	1.00	1.00	1.00	10
17	1.00	1.00	1.00	136
18	1.00	1.00	1.00	145
19	0.83	1.00	0.91	5
20	0.00	0.00	0.00	1
21	1.00	1.00	1.00	151
22	1.00	1.00	1.00	53
accuracy			1.00	1499
macro avg	0.94	0.95	0.95	1499
weighted avg	1.00	1.00	1.00	1499

Confusion Matrix:

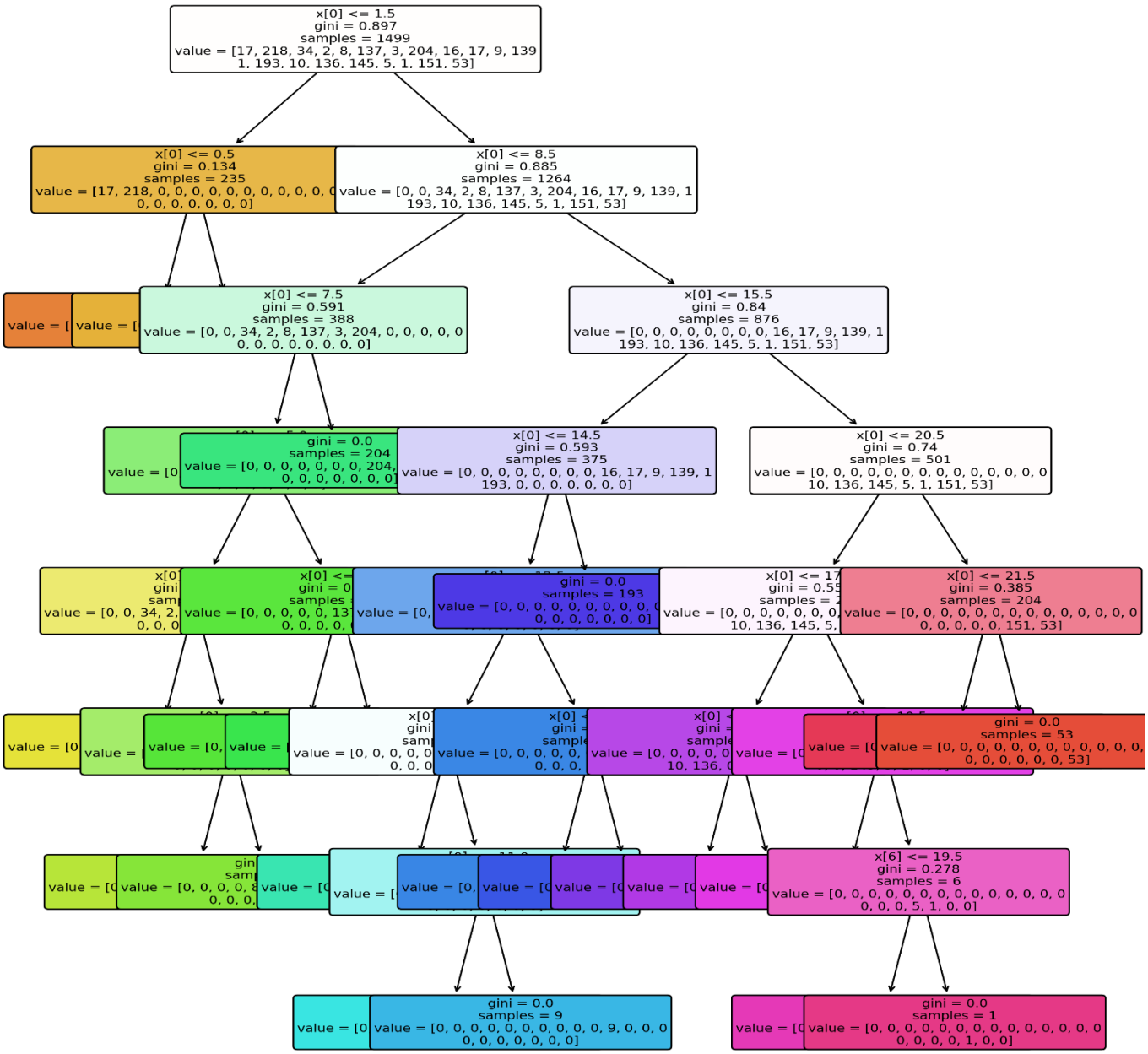
```
[[ 17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
 [  0 218   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
 [  0   0  34   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
 [  0   0   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
 [  0   0   0   0   8   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
 [  0   0   0   0   0 137   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
```

Step 7: Visualization

```
plt.figure(figsize=(12, 6))
sns.barplot(data=team_counts, x='Team', y='Matches Played', palette='viridis')
plt.title('Number of Matches Played by Each Team')
plt.xlabel('Team')
plt.ylabel('Number of Matches Played')
plt.tight_layout()
plt.show()
```




```
from sklearn import tree
plt.figure(figsize=(10,15))
dt.fit(xtest,ytest)
tree.plot_tree(dt, filled=True,  rounded=True, fontsize=8)
plt.savefig('decision_tree_plot.png', bbox_inches='tight', dpi=300)
plt.show()
```


x[0] <= 1.5
gini = 0.897
samples = 1499
value = [17, 218, 34, 2, 8, 137, 3, 204, 16, 17, 9, 139
1, 193, 10, 136, 145, 5, 1, 151, 53]
x[0] <= 0.5
gini = 0.134
samples = 235
value = [17, 218, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0]
x[0] <= 8.5
gini = 0.885
samples = 1264
value = [0, 0, 34, 2, 8, 137, 3, 204, 16, 17, 9, 139, 1
193, 10, 136, 145, 5, 1, 151, 53]
x[0] <= 7.5
gini = 0.591
samples = 388
value = [0, 0, 34, 2, 8, 137, 3, 204, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0]
x[0] <= 15.5
gini = 0.84
samples = 876
value = [0, 0, 0, 0, 0, 0, 0, 0, 16, 17, 9, 139, 1
193, 10, 136, 145, 5, 1, 151, 53]
gini = 0.0
samples = 204
value = [0, 0, 0, 0, 0, 0, 0, 204,
0, 0, 0, 0, 0, 0, 0]
x[0] <= 14.5
gini = 0.593
samples = 375
value = [0, 0, 0, 0, 0, 0, 0, 0, 16, 17, 9, 139, 1
193, 0, 0, 0, 0, 0, 0, 0]
x[0] <= 20.5
gini = 0.74
samples = 501
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
10, 136, 145, 5, 1, 151, 53]
gini = 0.0
samples = 193
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0]
x[0] <= 21.5
gini = 0.385
samples = 204
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 151, 53]
gini = 0.0
samples = 53
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 53]
x[6] <= 19.5
gini = 0.278
samples = 6
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 5, 1, 0, 0]
gini = 0.0
samples = 9
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0
0, 0, 0, 0, 0, 0, 0]
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 1, 0, 0]

```
predicted = print(f"Predicted: {y_pred}")
actual = print(f"Actual: {ytest.values}")

result = pd.DataFrame({"Actual" : ytest.values, "Predicted": y_pred})
```

```
Predicted: [18  8  1 ...  1  8  8]
Actual: [18  8  1 ...  1  8  8]
```

```
result.to_csv('Actual vs Predicted.csv', index=False)
```

Conclusion

The Decision Tree classifier was chosen for this ODI cricket analysis due to its distinct advantages:

1. **Handling Categorical Data:** Decision trees are well-suited for datasets with categorical features, like team names and match venues, as they don't require data scaling or encoding.

2. **No Assumptions about Feature Distribution**: Unlike linear models, decision trees don't require assumptions about the distribution of the data, which is beneficial when working with sports data, where relationships between variables are often non-linear.

3. **Interpretability:** The decision tree provides a clear visual representation of how decisions are made, making it easy to understand which features contribute the most to predicting match outcomes.