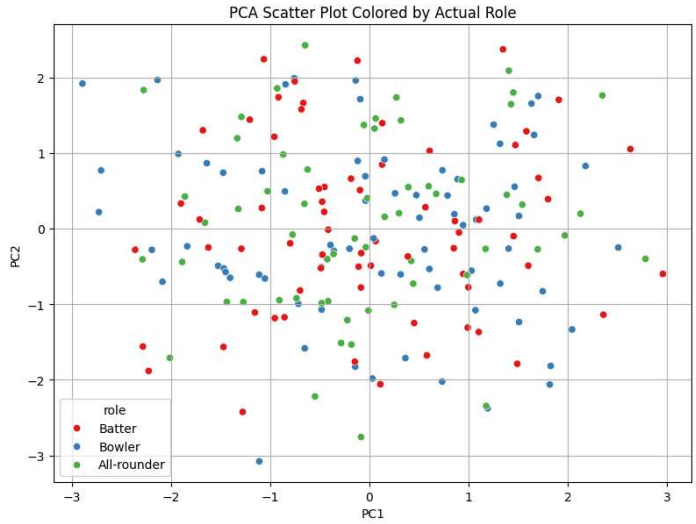
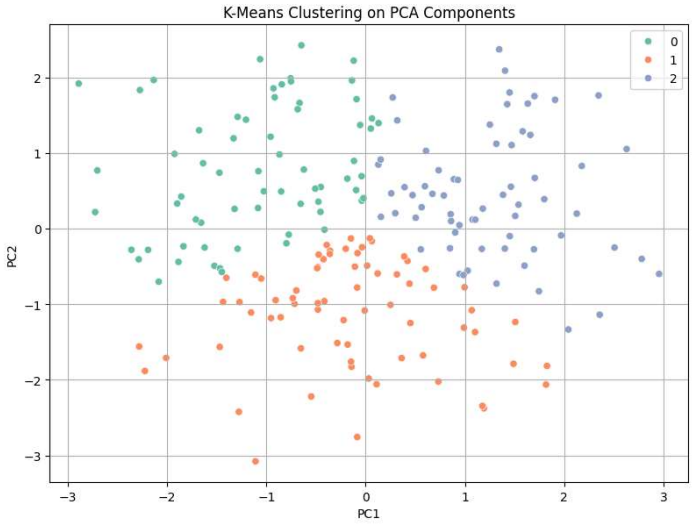
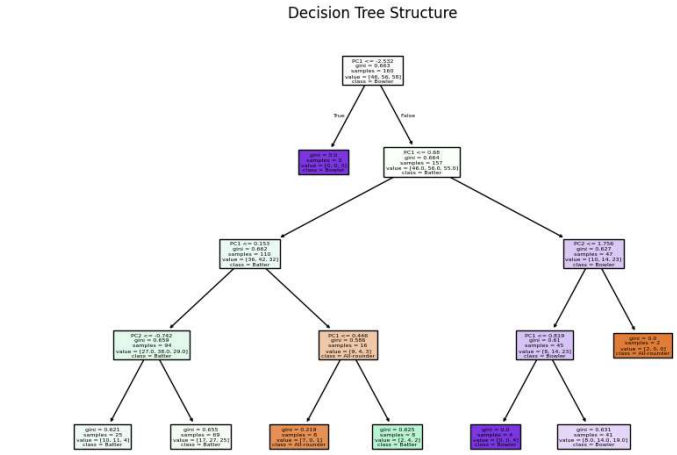
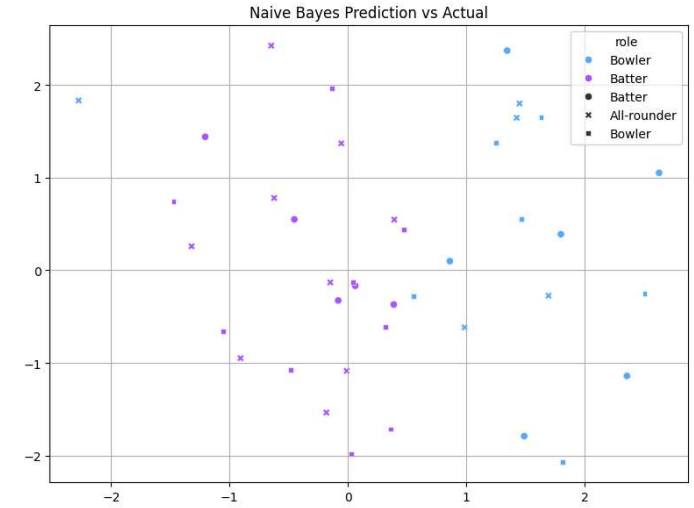
```
import pandas as pd

df = pd.read_csv('ODI Cricket Data new.csv')


from sklearn.model_selection import train_test_split

X = df.drop('target_variable', axis=1)  # Replace 'target_variable' with the actual target column
y = df['target_variable']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Simulate loading of dataset (use your actual dataset in real use)
np.random.seed(42)
sample_data = {
    'total_runs': np.random.randint(0, 10000, 200),
    'strike_rate': np.random.uniform(50, 150, 200),
    'total_balls_faced': np.random.randint(100, 10000, 200),
    'total_wickets_taken': np.random.randint(0, 300, 200),
    'total_runs_conceded': np.random.randint(0, 10000, 200),
    'total_overs_bowled': np.random.randint(0, 2000, 200),
    'total_matches_played': np.random.randint(1, 400, 200),
    'matches_played_as_batter': np.random.randint(0, 300, 200),
    'matches_played_as_bowler': np.random.randint(0, 300, 200),
    'matches_won': np.random.randint(0, 300, 200),
    'matches_lost': np.random.randint(0, 300, 200),
    'player_of_match_awards': np.random.randint(0, 100, 200),
    'role': np.random.choice(['Batter', 'Bowler', 'All-rounder'], 200)
}
df = pd.DataFrame(sample_data)

# Preprocessing
X = df.drop(columns=['role'])
y = df['role']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])

# Train-test split for classifiers
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# 1. Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_preds = nb_model.predict(X_test)

# 2. Decision Tree
dt_model = DecisionTreeClassifier(max_depth=4, random_state=42)
dt_model.fit(X_train, y_train)
dt_preds = dt_model.predict(X_test)

# 3. KNN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_preds = knn_model.predict(X_test)

# 4. K-Means Clustering (Unsupervised)
kmeans = KMeans(n_clusters=3, random_state=42)
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_pca)

# Visualizations
plt.figure(figsize=(16, 12))

# PCA Components
plt.subplot(2, 2, 1)
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'], hue=y, palette='Set1')
plt.title('PCA Scatter Plot Colored by Actual Role')
plt.grid(True)

# K-Means Clustering
plt.subplot(2, 2, 2)
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'], hue=kmeans_labels, palette='Set2')
plt.title('K-Means Clustering on PCA Components')
plt.grid(True)

# Decision Tree Plot
plt.subplot(2, 2, 3)
plot_tree(dt_model, filled=True, feature_names=['PC1', 'PC2'], class_names=dt_model.classes_)
plt.title('Decision Tree Structure')

# Naive Bayes Predicted vs Actual
plt.subplot(2, 2, 4)
sns.scatterplot(x=X_test[:, 0], y=X_test[:, 1], hue=nb_preds, palette='cool', style=y_test)
plt.title('Naive Bayes Prediction vs Actual')
plt.grid(True)

plt.tight_layout()
plt.show()

# Print metrics
nb_acc = accuracy_score(y_test, nb_preds)
dt_acc = accuracy_score(y_test, dt_preds)
knn_acc = accuracy_score(y_test, knn_preds)

(nb_acc, dt_acc, knn_acc)
```









(0.275, 0.275, 0.275)

```
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assume df is already preprocessed
x = df.drop(columns=['role'])
y = df['role']

# Encode target labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Train models
models = {
    "Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5)
}
```

```
for model in models.values():
    model.fit(x_train_scaled, y_train)



from sklearn.metrics import roc_curve, auc, confusion_matrix, ConfusionMatrixDisplay, classification_report, precision_recall_fscore_support
from sklearn.preprocessing import label_binarize
from sklearn.metrics import RocCurveDisplay

# Binarize the target labels for ROC Curve (multiclass case)
y_test_binarized = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_binarized.shape[1]

# ROC Curve for each classifier (Naive Bayes, Decision Tree, KNN)
# Predict probabilities
nb_probs = models['Naive Bayes'].predict_proba(x_test)
dt_probs = models['Decision Tree'].predict_proba(x_test)
knn_probs = models['KNN'].predict_proba(x_test)

# Initialize ROC Curve plots
plt.figure(figsize=(12, 6))
for probs, label in zip([nb_probs, dt_probs, knn_probs], ['Naive Bayes', 'Decision Tree', 'KNN']):
    fpr, tpr, _ = roc_curve(y_test_binarized.ravel(), probs.ravel())
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{label} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# Confusion Matrix Heatmap for each model
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, (name, model) in zip(axes, models.items()):
    y_pred = model.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(ax=ax, cmap='Blues', colorbar=False)
    ax.set_title(f'{name} Confusion Matrix')
plt.tight_layout()
plt.show()

# Scatter Plot of predicted values (PCA)
pca = PCA(n_components=2)
pca_components = pca.fit_transform(x_scaled)
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, (name, model) in zip(axes, models.items()):
    preds = model.predict(x_scaled)
    sns.scatterplot(x=pca_components[:, 0], y=pca_components[:, 1], hue=preds, palette='Set2', ax=ax, alpha=0.6)
    ax.set_title(f'{name} Prediction Scatter Plot')
    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
plt.tight_layout()
plt.show()

# Bar Graph of Accuracy Scores
accuracies = [accuracy_score(y_test, model.predict(x_test)) for model in models.values()]
plt.figure(figsize=(8, 5))
sns.barplot(x=list(models.keys()), y=accuracies, palette='viridis')
plt.ylabel('Accuracy Score')
plt.title('Model Accuracy Comparison')
plt.ylim(0, 1)
plt.grid(axis='y')
plt.show()

# Pie Chart for Precision, Recall, F1-Score for each model
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, (name, model) in zip(axes, models.items()):
    y_pred = model.predict(x_test)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, average='weighted')
    metrics = [precision, recall, f1]
    labels = ['Precision', 'Recall', 'F1-Score']
    ax.pie(metrics, labels=labels, autopct='%1.1f%%', startangle=90, colors=sns.color_palette('pastel'))
    ax.set_title(f'{name} Metrics')
```

```
plt.tight_layout()
plt.show()
```