# Week 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn import tree
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc, classification_report, accuracy_score
```

## Load the dataset into a pandas DataFrame

```
df = pd.read_csv('/content/odi.csv')
df = df.dropna()
# Display the first few rows of the dataframe
print(df.head())
```

```
   mid        date                                venue bat_team bowl_team  \
0    1  2006-06-13  Civil Service Cricket Club, Stormont  England   Ireland
1    1  2006-06-13  Civil Service Cricket Club, Stormont  England   Ireland
2    1  2006-06-13  Civil Service Cricket Club, Stormont  England   Ireland
3    1  2006-06-13  Civil Service Cricket Club, Stormont  England   Ireland
4    1  2006-06-13  Civil Service Cricket Club, Stormont  England   Ireland

          batsman       bowler  runs  wickets  overs  runs_last_5  \
0  ME Trescothick  DT Johnston     0        0    0.1            0
1  ME Trescothick  DT Johnston     0        0    0.2            0
2  ME Trescothick  DT Johnston     4        0    0.3            4
3  ME Trescothick  DT Johnston     6        0    0.4            6
4  ME Trescothick  DT Johnston     6        0    0.5            6

   wickets_last_5  striker  non-striker  total
0               0        0            0    301
1               0        0            0    301
2               0        0            0    301
3               0        0            0    301
4               0        0            0    301
```

# Week 2

## Calculate the probability of a team scoring more than 300 runs

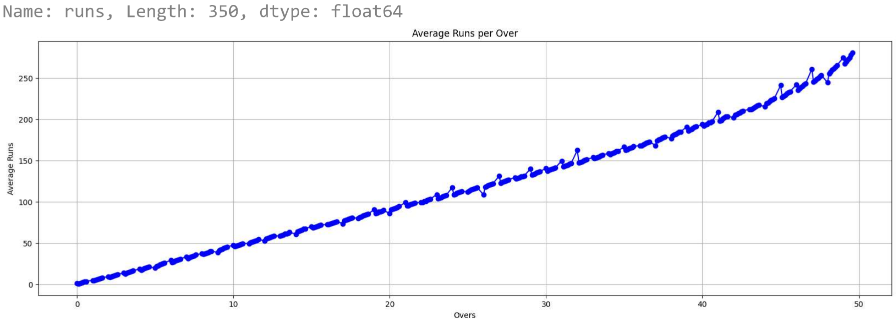
```
# Assuming 'total' is the target variable we want to analyze
total_runs = df['total']
probability = (total_runs > 300).mean()
print(f"The probability of scoring more than 300 runs is {probability:.2f}")

```

```
The probability of scoring more than 300 runs is 0.23
```

## Average runs per over

```
average_runs_per_over = df.groupby('overs')['runs'].mean()
print(average_runs_per_over)

plt.figure(figsize=(20, 6))
plt.plot(average_runs_per_over.index, average_runs_per_over.values, marker='o', linestyle='-', color='b')
plt.title('Average Runs per Over')
plt.xlabel('Overs')
plt.ylabel('Average Runs')
plt.grid(True)
plt.show()
```

```
overs
0.0       1.211864
0.1       0.635788
0.2       1.167340
0.3       1.793239
0.4       2.353323
            ...
49.2    270.046358
49.3    272.824661
49.4    275.153216
49.5    278.147129
49.6    281.134271
Name: runs, Length: 350, dtype: float64
```



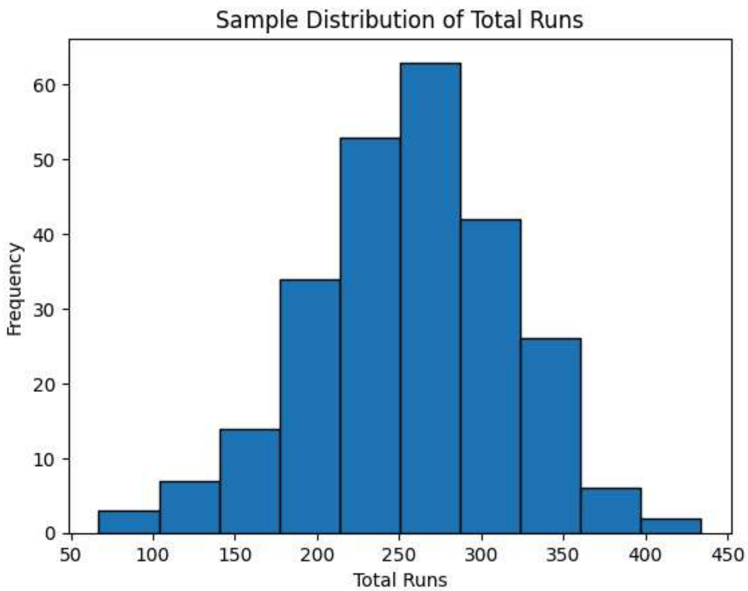

Double-click (or enter) to edit

## Week 3

## Sampling

```
# Define the sample size you want to draw
sample_size = 250

# Perform simple random sampling
sample_df = df.sample(n=sample_size, random_state=1)

# Calculate sample mean and standard deviation for 'total' runs
sample_mean = sample_df['total'].mean()
sample_std = sample_df['total'].std()

print(f"Sample Mean: {sample_mean}")
print(f"Sample Standard Deviation: {sample_std}")

plt.hist(sample_df['total'], bins=10, edgecolor='black')
plt.title('Sample Distribution of Total Runs')
plt.xlabel('Total Runs')
plt.ylabel('Frequency')
plt.show()
```

```
Sample Mean: 256.292
Sample Standard Deviation: 60.77160098223327
```




Double-click (or enter) to edit

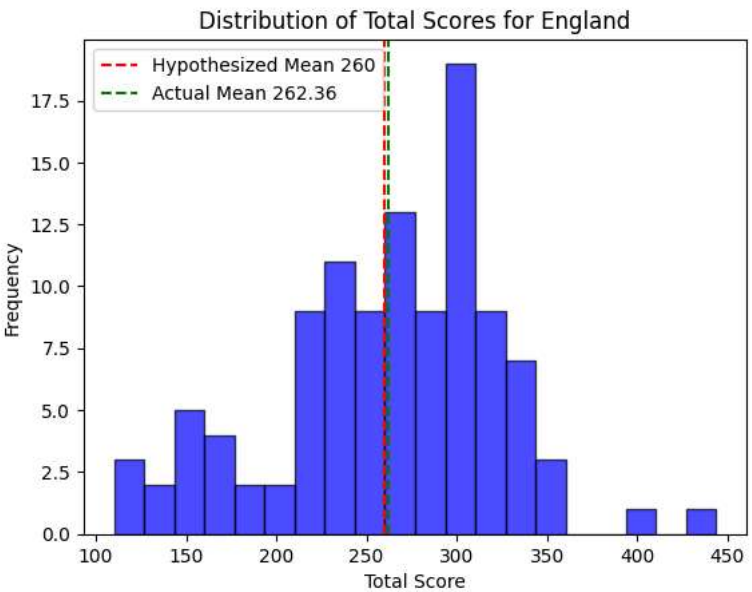
## Week 4

## Hypothesis Testing

```
# Let's say we want to test the hypothesis that the average total score for England is 260
# Null hypothesis (H0): mean total score for England is 260
# Alternative hypothesis (H1): mean total score for England is not 260

# Filter the dataset for England's batting team
england_df = df[df['bat_team'] == 'England']

# Group by match ID and take the maximum 'total' to get the total score per match for England
england_total_scores = england_df.groupby('mid')['total'].max().reset_index()

# Calculate the mean and standard deviation of total scores for England
mean_score = england_total_scores['total'].mean()
std_deviation = england_total_scores['total'].std()
print(f"Mean: {mean_score}")
print(f"Standard Deviation: {std_deviation}")

# Plot histogram of total scores for England
plt.hist(england_total_scores['total'], bins=20, alpha=0.7, color='blue', edgecolor='black')

```

```
hypothesized_mean = 260
# Add a vertical line for the hypothesized mean score
plt.axvline(x=hypothesized_mean, color='red', linestyle='--', label=f'Hypothesized Mean {hypothesized_mean}')
plt.axvline(x=mean_score, color='green', linestyle='--', label=f"Actual Mean {mean_score:.2f}")
# Add labels and title
plt.xlabel('Total Score')
plt.ylabel('Frequency')
plt.title('Distribution of Total Scores for England')

# Add legend
plt.legend()

# Show plot
plt.show()

# Perform a one-sample t-test
t_statistic, p_value = stats.ttest_1samp(england_total_scores['total'], hypothesized_mean)

# Set your significance level
alpha = 0.05

# Check if we reject or fail to reject the null hypothesis
if p_value < alpha:
    print(f"Reject the null hypothesis. p-value: {p_value}")
else:
    print(f"Fail to reject the null hypothesis. p-value: {p_value}")
```

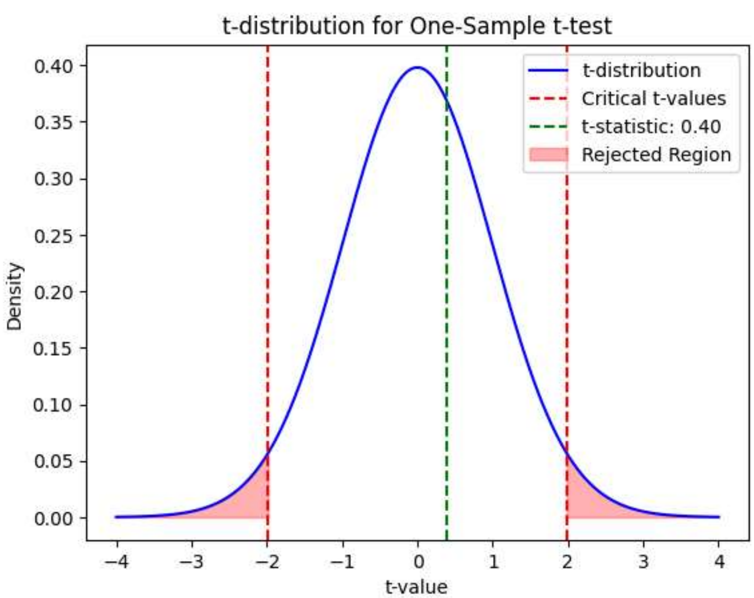
```
Mean: 262.35779816513764
Standard Deviation: 61.79798674291424
```




```
Fail to reject the null hypothesis. p-value: 0.6911716118103672
```

## T test

```
x = np.linspace(-4, 4, 1000)

# Calculate the t-distribution with degrees of freedom (df) equal to the sample size minus 1
df = len(england_total_scores) - 1
t_distribution = stats.t.pdf(x, df)

# Plot the t-distribution
plt.plot(x, t_distribution, 'b-', label='t-distribution')

# Plot the critical t-values for two-tailed test at alpha = 0.05
critical_t_values = [stats.t.ppf(0.025, df), stats.t.ppf(0.975, df)]
plt.axvline(critical_t_values[0], color='red', linestyle='--', label='Critical t-values')
plt.axvline(critical_t_values[1], color='red', linestyle='--')

# Plot the t-statistic
plt.axvline(t_statistic, color='green', linestyle='--', label=f't-statistic: {t_statistic:.2f}')

# Calculate the p-value
p_value = stats.t.sf(np.abs(t_statistic), df) * 2  # two-tailed test

# Shade the rejected region if p-value is less than alpha
alpha = 0.05
x_accepted_left = np.linspace(-4, critical_t_values[0], 100)
x_accepted_right = np.linspace(critical_t_values[1], 4, 100)
plt.fill_between(x_accepted_left, stats.t.pdf(x_accepted_left, df), color='red', alpha=0.3, label='Rejected Region')
plt.fill_between(x_accepted_right, stats.t.pdf(x_accepted_right, df), color='red', alpha=0.3)

# Add labels and title
plt.xlabel('t-value')
plt.ylabel('Density')
plt.title('t-distribution for One-Sample t-test')
plt.legend()

# Show plot
plt.show()

# Print p-value
print(f"P-value: {p_value:.4f}")
```

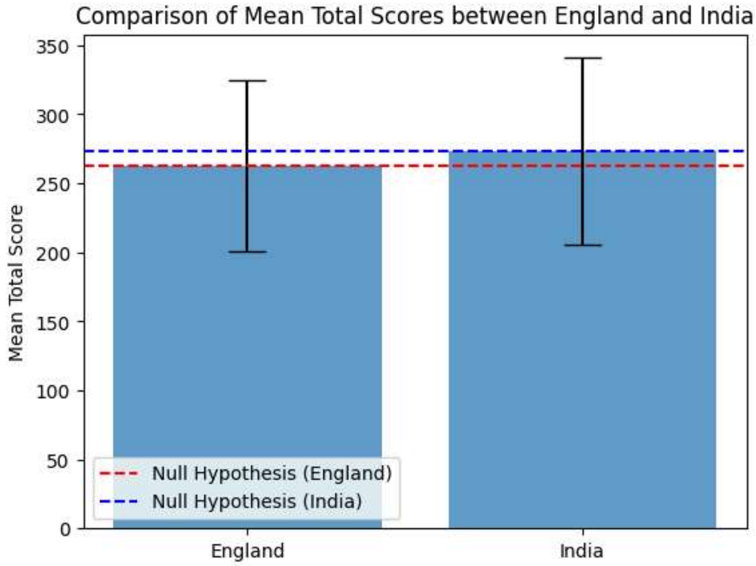



```
P-value: 0.6912
```

## Week 5

## Two Sample Testing and ANOVA

```
# say we want to compare the average total scores of two teams, England and India
# Null hypothesis (H0): There is no difference in the average total scores of England and India
# Alternative hypothesis (H1): There is a difference in the average total scores of England and India

# Filter the dataset for each team
england_df = df[df['bat_team'] == 'England']
england_total_scores = england_df.groupby('mid')['total'].max().reset_index()
india_df = df[df['bat_team'] == 'India']
india_total_scores = india_df.groupby('mid')['total'].max().reset_index()

# Calculate means and standard deviations for each team
england_mean = england_total_scores['total'].mean()
india_mean = india_total_scores['total'].mean()
england_std = england_total_scores['total'].std()
india_std = india_total_scores['total'].std()

# Set up the plot
teams = ['England', 'India']
x_pos = np.arange(len(teams))
means = [england_mean, india_mean]
std_devs = [england_std, india_std]

# Plot the bar chart
plt.bar(x_pos, means, yerr=std_devs, align='center', alpha=0.7, ecolor='black', capsize=10)
plt.xticks(x_pos, teams)
plt.ylabel('Mean Total Score')
plt.title('Comparison of Mean Total Scores between England and India')

# Add a horizontal line representing the null hypothesis
plt.axhline(y=england_mean, color='red', linestyle='--', label='Null Hypothesis (England)')
plt.axhline(y=india_mean, color='blue', linestyle='--', label='Null Hypothesis (India)')

# Add legend
plt.legend()

# Show plot
plt.show()

# Perform a two-sample t-test
t_statistic, p_value = stats.ttest_ind(england_total_scores['total'], india_total_scores['total'])

# Print p-value
print(f"P-value: {p_value:.4f}")

# Set your significance level
alpha = 0.05

# Check if we reject or fail to reject the null hypothesis
if p_value < alpha:
    print(f'Reject the null hypothesis. p-value: {p_value:.4f}')
else:
    print(f'Fail to reject the null hypothesis. p-value: {p_value:.4f}')

```




```
P-value: 0.2059
Fail to reject the null hypothesis. p-value: 0.2059
```

## Week 6

### Linear regression

```
def custom_accuracy(y_test,y_pred,thresold):
    right = 0

    l = len(y_pred)
    for i in range(0,l):
        if(abs(y_pred[i]-y_test[i]) <= thresold):
            right += 1
    return ((right/l)*100)


X = df.iloc[:,[7,8,9,12,13]].values
y = df.iloc[:, 14].values


# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the dataset
lin = LinearRegression()
lin.fit(X_train,y_train)

# Testing the dataset on trained model
y_pred = lin.predict(X_test)
score = lin.score(X_test,y_test)*100
print("R square value:" , score)
print("Custom accuracy:" , custom_accuracy(y_test,y_pred,20))

# Testing with a custom input
new_prediction = lin.predict(sc.transform(np.array([[100,0,13,50,50]])))
print("Prediction score:" , new_prediction)
```

```
R square value: 52.737657811129445
Custom accuracy: 43.354801937874036
Prediction score: [322.42983935]
```

# Week 7

## Linear Regression and Multiple Regression

```
# predict the 'total' score using multiple predictors.
# For this example, we'll use 'runs', 'wickets', 'overs', 'runs_last_5', and 'wickets_last_5'.

# Define the predictors and the response variable
predictors = ['runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5']
X = df[predictors]
y = df['total']

# Add a constant to the model (intercept)
X = sm.add_constant(X)

# Fit the multiple linear regression model
model = sm.OLS(y, X).fit()

# Get the summary of the regression
summary = model.summary()
print(summary)

# You can also use the model to make predictions
# Here's an example of predicting the 'total' for a new observation
new_observation = {'runs': 50, 'wickets': 2, 'overs': 10, 'runs_last_5': 25, 'wickets_last_5': 1}
new_observation['total'] = 0
new_X = pd.DataFrame([new_observation])
new_X = sm.add_constant(new_X)

predicted_total = model.predict(new_X)
print(f"Predicted Total: {predicted_total[0]}")
```

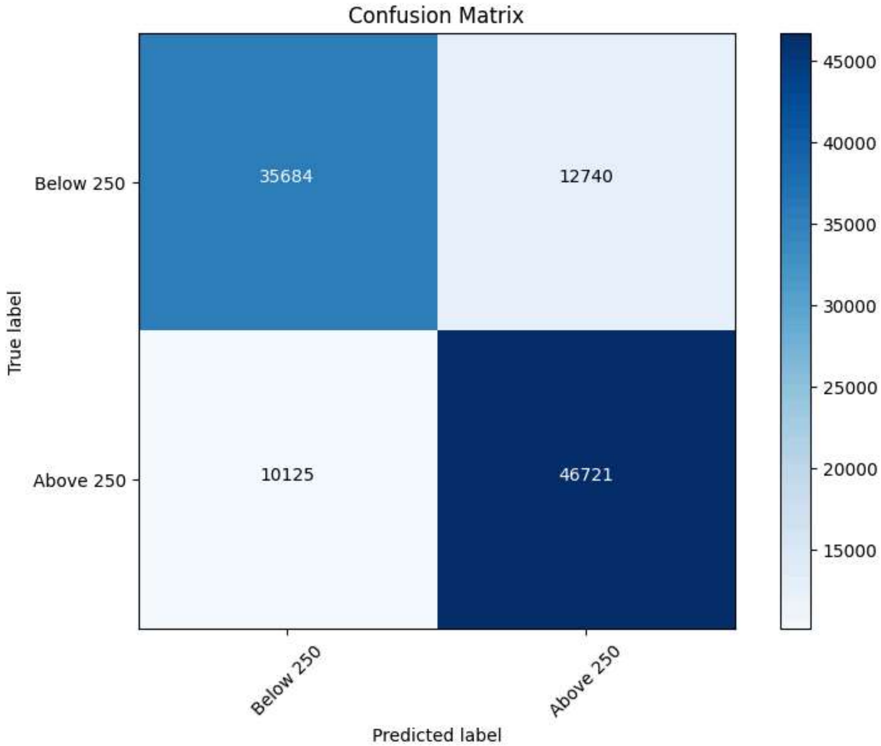
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  total   R-squared:                       0.527
Model:                            OLS   Adj. R-squared:                  0.527
Method:                 Least Squares   F-statistic:                 7.834e+04
Date:                Mon, 08 Apr 2024   Prob (F-statistic):               0.00
Time:                        09:29:32   Log-Likelihood:            -1.8166e+06
No. Observations:              350899   AIC:                         3.633e+06
Df Residuals:                  350893   BIC:                         3.633e+06
Df Model:                           5
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const            252.8874      0.214   1182.966      0.000     252.468     253.306
runs               1.0811      0.003    313.979      0.000       1.074       1.088
wickets          -12.9809      0.063   -205.223      0.000     -13.105     -12.857
overs             -3.5606      0.021   -172.843      0.000      -3.601      -3.520
runs_last_5        0.2146      0.009     22.615      0.000       0.196       0.233
wickets_last_5    -3.6459      0.106    -34.291      0.000      -3.854      -3.438
==============================================================================
Omnibus:                    12201.805   Durbin-Watson:                   0.016
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            30831.871
Skew:                          -0.158   Prob(JB):                         0.00
Kurtosis:                       4.417   Cond. No.                         429.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Predicted Total: 12427.924298030672
```

# Week 8

## Concepts of MLE (Maximum Likelihood Estimation) and Logistic Regression

```
df['high_score'] = df['total'].apply(lambda x: 1 if x > 250 else 0)

# Define the predictors and the binary target variable
predictors = ['runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5']
X = df[predictors]
y = df['high_score']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model to the training data
log_reg.fit(X_train, y_train)

# Predict on the testing data
y_pred = log_reg.predict(X_test)

# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Visualize Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ['Below 250', 'Above 250']
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

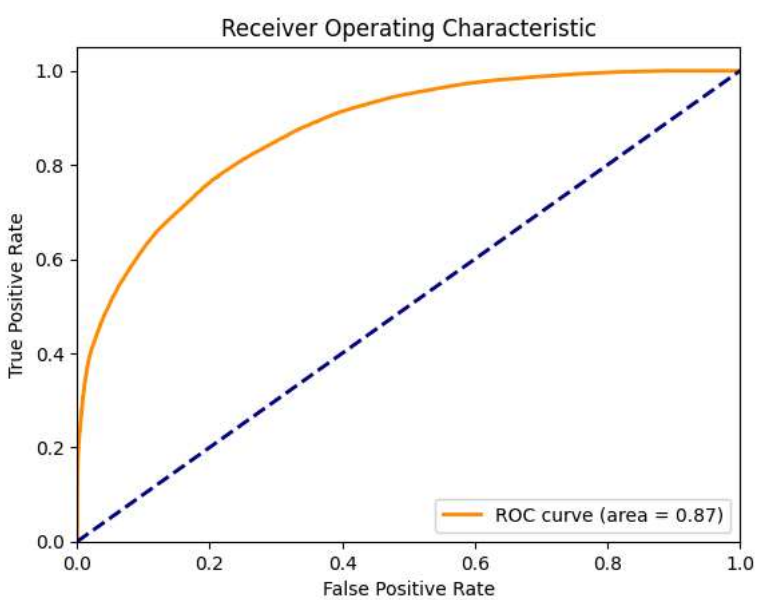
```
[[35684 12740]
 [10125 46721]]
              precision    recall  f1-score   support

           0       0.78      0.74      0.76     48424
           1       0.79      0.82      0.80     56846

    accuracy                           0.78    105270
   macro avg       0.78      0.78      0.78    105270
weighted avg       0.78      0.78      0.78    105270
```




## Week 9

## ROC (Receiver Operating Characteristic) and Regression Analysis Model Building

```
# For logistic regression, we need a binary target variable. Let's create one for illustration.
# For example, we'll predict if the total score will be above or below 250.
df['high_score'] = df['total'].apply(lambda x: 1 if x > 250 else 0)

# Define the predictors and the binary target variable
predictors = ['runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5']
X = df[predictors]
y = df['high_score']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model to the training data
log_reg.fit(X_train, y_train)

# Predict probabilities for the test data
y_probs = log_reg.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

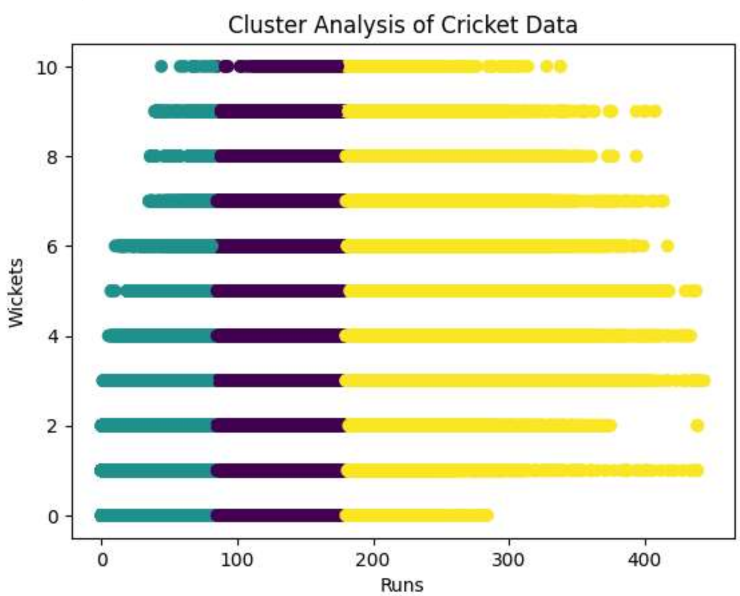



## Week 10

## Chi-square Test and Introduction to Cluster Analysis

```python
# Chi-square Test
# to test if there's a significant association between 'bat_team' and 'bowl_team' winning.

# Create a contingency table
contingency_table = pd.crosstab(df['bat_team'], df['bowl_team'])

# Perform the Chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Set your significance level
alpha = 0.05

# Check if we reject or fail to reject the null hypothesis
print(f'Chi-square Statistic: {chi2}, p-value: {p}')
if p < alpha:
    print('Reject the null hypothesis, indicating a significant association between teams.')
else:
    print('Fail to reject the null hypothesis, no significant association was found.')

# Introduction to Cluster Analysis
# For cluster analysis, let's use 'runs' and 'wickets' for clustering the data points.

# Define the data for clustering
cluster_data = df[['runs', 'wickets']]

# Initialize KMeans
kmeans = KMeans(n_clusters=3, random_state=0)

# Fit the model
kmeans.fit(cluster_data)

# Predict the clusters
labels = kmeans.predict(cluster_data)

# Plot the clusters
plt.scatter(cluster_data['runs'], cluster_data['wickets'], c=labels)
plt.xlabel('Runs')
plt.ylabel('Wickets')
plt.title('Cluster Analysis of Cricket Data')
plt.show()
```

```
Chi-square Statistic: 920349.3894003384, p-value: 0.0
Reject the null hypothesis, indicating a significant association between teams.
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
  warnings.warn(
```



```
# Separate bat_team and bowl_team by country
# Extract the country from the bat_team column
df['bat_country'] = df['bat_team'].apply(lambda x: x.split()[0])
# Extract the country from the bowl_team column
df['bowl_country'] = df['bowl_team'].apply(lambda x: x.split()[0])

# Chi-square Test

# Create a contingency table
contingency_table = pd.crosstab(df['bat_country'], df['bowl_country'])
# Perform the Chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Set the significance level
alpha = 0.05

print(f'Chi-square Statistic: {chi2}, p-value: {p}')
if p < alpha:
    print('Reject the null hypothesis, indicating a significant association between teams.')
else:
    print('Fail to reject the null hypothesis, no significant association was found.')

# Cluster Analysis

# Select relevant columns for clustering
cluster_data = df[['runs', 'wickets', 'bat_country', 'bowl_country']]
# Select matches where India is the batting team and England is the bowling team
india_data = cluster_data[(cluster_data['bat_country'] == 'India') & (cluster_data['bowl_country'] == 'England')]

# Fit KMeans model

# Initialize KMeans model with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=0)
# Fit the model to the selected data
kmeans.fit(india_data[['runs', 'wickets']])

# Predict clusters

# Predict cluster labels for the data points
labels = kmeans.predict(india_data[['runs', 'wickets']])

# Plot clusters

# Scatter plot of runs vs. wickets with clusters colored according to labels
plt.scatter(india_data['runs'], india_data['wickets'], c=labels)
plt.xlabel('Runs')
plt.ylabel('Wickets')
plt.title('Cluster Analysis of Cricket Data (India)')
plt.show()

```

## Week 11

Cluster Analysis of Cricket Data (India)

## Clustering Analysis

```python
# Selecting features for clustering (for example, 'runs' and 'wickets')
features = df[['runs', 'wickets']]

# Standardizing the features for better clustering performance
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Using the elbow method to find the optimal number of clusters
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(features_scaled)
    wcss.append(kmeans.inertia_)

# Plotting the results onto a line graph to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') # Within cluster sum of squares
plt.show()

# Applying K-Means to the dataset with the optimal number of clusters
optimal_clusters = 3 # This is an example, you should choose the number based on the elbow method
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(features_scaled)

# Visualizing the clusters
plt.scatter(features_scaled[y_kmeans == 0, 0], features_scaled[y_kmeans == 0, 1], s=100, c='red', label ='Cluster 1')
plt.scatter(features_scaled[y_kmeans == 1, 0], features_scaled[y_kmeans == 1, 1], s=100, c='blue', label ='Cluster 2')
plt.scatter(features_scaled[y_kmeans == 2, 0], features_scaled[y_kmeans == 2, 1], s=100, c='green', label ='Cluster 3')
plt.title('Clusters of players')
plt.xlabel('Runs (standardized)')
plt.ylabel('Wickets (standardized)')
plt.legend()
plt.show()
```