1. 📦 Installing Libraries

We start by importing all the necessary Python libraries required for data handling, visualization, preprocessing, and building machine learning and deep learning models.

The libraries used in this project include:

- **NumPy** – For numerical operations
- **Pandas** – For data manipulation and analysis
- **Matplotlib** & **Seaborn** – For data visualization
- **Scikit-learn** – For preprocessing and model building
- **Keras** – For building deep learning models (with TensorFlow backend)

Install them using pip:

```
pip install numpy pandas scikit-learn matplotlib seaborn tensorflow
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import keras
import tensorflow as tf
```

2. 📂 Loading the Dataset

The dataset used for this project contains IPL match data from **2008 to 2017**, including features like:

- Venue
- Date
- Batting and Bowling Team
- Batsman and Bowler Names
- Runs
- Wickets
- and more...

You can download the dataset from this source

We load the data into **Pandas DataFrames** using the following code:

```
ipl = pd.read_csv('ipl_data.csv')
ipl.head()
```

	mid	date	venue	bat_team	bowl_team	batsman	bowler	runs	wickets	overs	runs_last_5	wickets_last_5	striker	non-striker
0	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	SC Ganguly	P Kumar	1	0	0.1	1	0	0	0
1	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	1	0	0.2	1	0	0	0
2	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	2	0	0.2	2	0	0	0
3	1	2008-	M Chinnaswamy	Kolkata Knight	Royal Challengers	BB	P	2	0	0.3	2	0	0	0

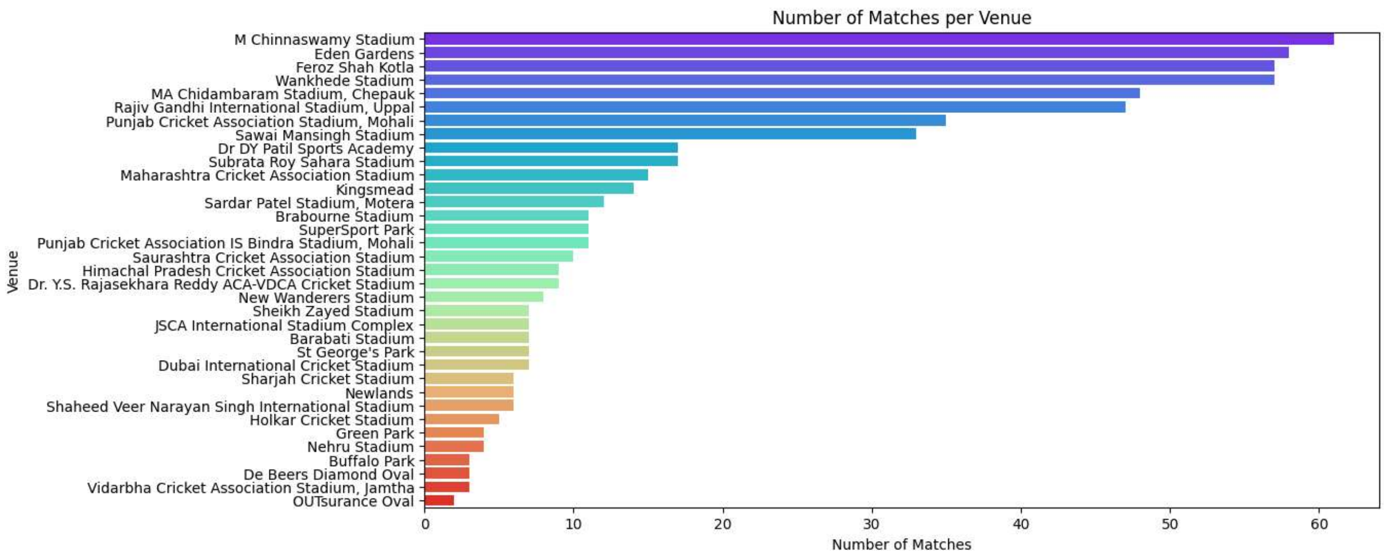
Next steps: Generate code with ipl | View recommended plots | New interactive sheet

3. 📊 Exploratory Data Analysis

We will do Exploratory Data Analysis (EDA) to analyze how many unique matches have been played at each venue by counting distinct match IDs for every venue. Then, we'll visualize this data using a horizontal bar chart to see which venues host the most matches.

```
data = ipl.copy()
matches_per_venue = data[['mid', 'venue']].drop_duplicates()
matches_count = matches_per_venue['venue'].value_counts()

plt.figure(figsize=(12,6))
sns.barplot(x=matches_count.values, y=matches_count.index,palette="rainbow")
plt.title('Number of Matches per Venue')
plt.xlabel('Number of Matches')
plt.ylabel('Venue')
plt.show()
```

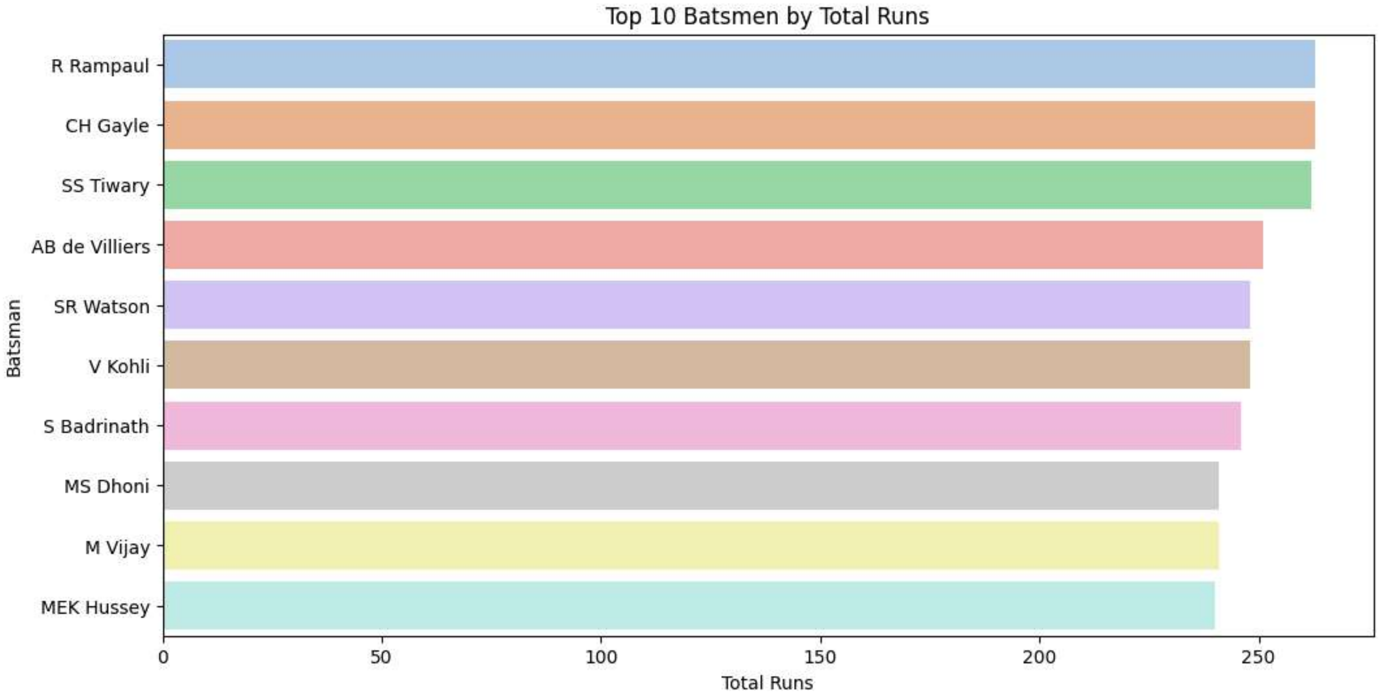
```
/tmp/ipython-input-3-65215632.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legenc

  sns.barplot(x=matches_count.values, y=matches_count.index,palette="rainbow")
```



Next we will calculate the maximum runs scored by each batsman by grouping the data by batsman their runs. Then we'll identify the top 10 batsmen with the highest runs and display this information using a horizontal bar chart.

```
runs_by_batsman = data.groupby('batsman')['runs'].max().sort_values(ascending=False).head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=runs_by_batsman.values, y=runs_by_batsman.index,palette="pastel")
plt.title('Top 10 Batsmen by Total Runs')
plt.xlabel('Total Runs')
plt.ylabel('Batsman')
plt.show()
```

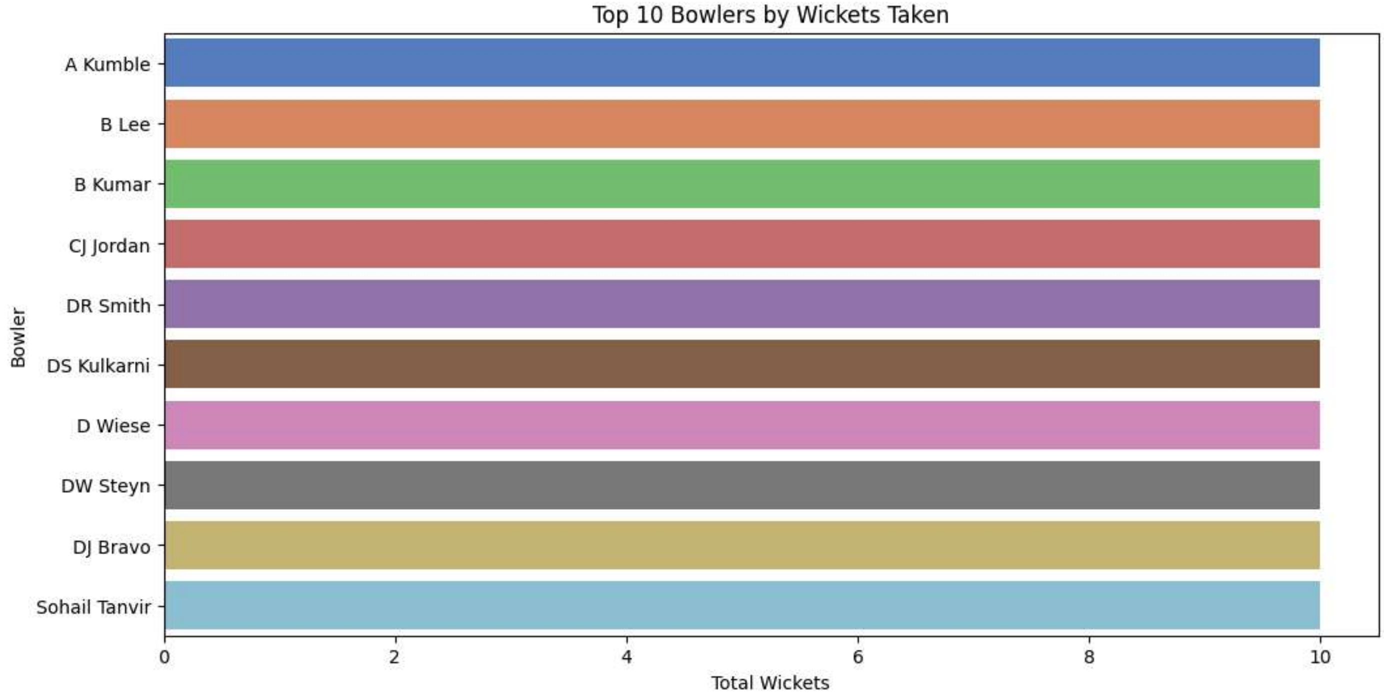
```
/tmp/ipython-input-4-1761167641.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legenc

  sns.barplot(x=runs_by_batsman.values, y=runs_by_batsman.index,palette="pastel")
```



After that we can do the same for the bowlers, in terms of total wicket

```
wickets_by_bowler = data.groupby('bowler')['wickets'].max().sort_values(ascending=False).head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=wickets_by_bowler.values, y=wickets_by_bowler.index, palette="muted")
plt.title('Top 10 Bowlers by Wickets Taken')
plt.xlabel('Total Wickets')
plt.ylabel('Bowler')
plt.show()
```

```
/tmp/ipython-input-5-2858948816.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legenc

  sns.barplot(x=wickets_by_bowler.values, y=wickets_by_bowler.index, palette="muted")
```



4. 🔤 Performing Label Encoding

We will convert categorical text data into numeric labels using **Label Encoding** because machine learning models work with numerical data.

- `LabelEncoder()` converts text labels into integers.
- `fit_transform()` learns the encoding and applies it.
- `copy()` creates a duplicate of the DataFrame to avoid modifying the original data.
- A dictionary assignment stores each encoder for future use, such as decoding or applying consistent transformations later.

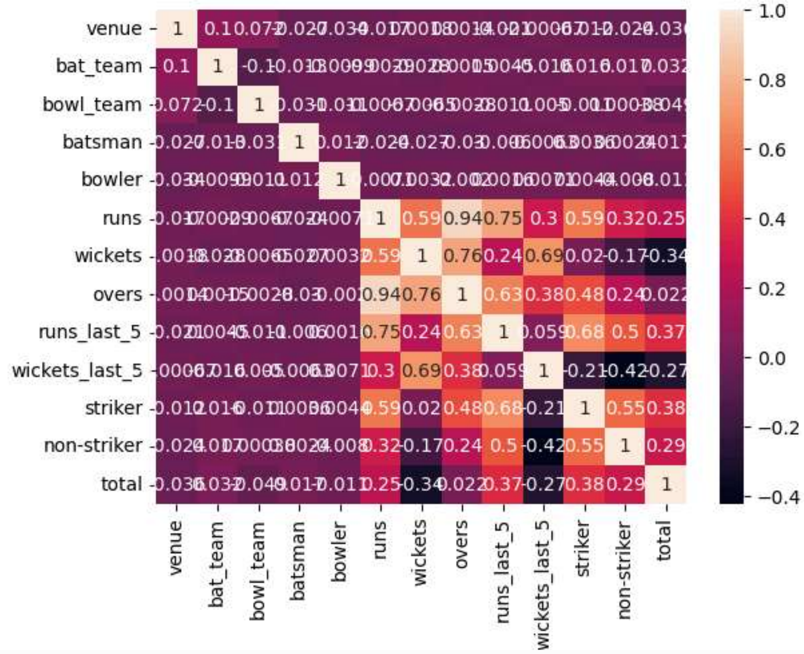
```
from sklearn.preprocessing import LabelEncoder

cat_cols = ['bat_team', 'bowl_team', 'venue', "batsman", "bowler"]

data_encoded = data.copy()

label_encoders = {}

for col in cat_cols:
    le = LabelEncoder()
    data_encoded[col] = le.fit_transform(data_encoded[col])
    label_encoders[col] = le
```

5. 📃 Performing Feature Selection

We drop `date` and `mid` columns because they are identifiers and don't provide meaningful information for correlation analysis. By removing these irrelevant columns, we focus on features that can reveal relationships useful for modeling or insights.

- `drop()` : removes specified columns from the DataFrame
- `corr()` : computes pairwise correlations between numerical features
- `sns.heatmap()` : creates a colored matrix to visualize correlations with values
- `plt.show()` : displays the plot on screen

```
data_corr=data_encoded.drop(columns=["date","mid"],axis=1)
sns.heatmap(data_corr.corr(),annot=True)
plt.show()
```



6. ✂️ Splitting the Dataset into Training and Testing

We will select relevant features and the target variable, then split the data into training and testing sets for model building and evaluation.

- `data_encoded[feature_cols]` : selects specified columns as features using DataFrame indexing
- `train_test_split()` : splits features and target into training and test subsets
- `test_size=0.3` : assigns 30% of data for testing
- `random_state=42` : ensures reproducible splits by fixing the random seed

```
feature_cols = ['bat_team', 'bowl_team', 'venue', 'runs', 'wickets', 'overs','striker','batsman','bowler']

X = data_encoded[feature_cols]
y = data_encoded['total']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

7. ⚖️ Performing Feature Scaling

We will perform **Min-Max scaling** on our input features to ensure all the features are on the same scale. This improves model performance by making learning more stable and consistent.

- `MinMaxScaler()` : scales features to a [0, 1] range
- `fit_transform()` : fits the scaler on training data and transforms it
- `transform()` : applies the same scaling to the test data using previously learned parameters

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

8. 🧠 Building the Neural Network

We will build a neural network using **TensorFlow** and **Keras** for regression tasks. The model is compiled using **Huber Loss** due to its robustness in handling outliers during regression.

- `keras.Sequential()` : creates a stack of layers
- `Dense` : defines fully connected layers
- `activation='relu'` : adds non-linearity to hidden layers
- Output layer uses `activation='linear'` since this is a regression problem
- **Huber Loss** : combines MSE and MAE benefits to better handle outliers
- **Adam Optimizer** : adjusts weights efficiently for faster convergence

```
model = keras.Sequential([
    keras.layers.Input( shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(216, activation='relu'),
    keras.layers.Dense(1, activation='linear')
])

huber_loss = tf.keras.losses.Huber(delta=1.0)  # You can adjust the 'delta' parameter as needed
model.compile(optimizer='adam', loss=huber_loss)
```

9. 🏋️ Training the Model

We train the model on the scaled training data for **10 epochs** with a **batch size of 64**, and validate it using the test set.

- `model.fit()` : trains the model
- `epochs=10` : the model sees the entire training data 10 times
- `batch_size=64` : updates weights after every 64 samples
- `validation_data` : evaluates model performance on the test set during training

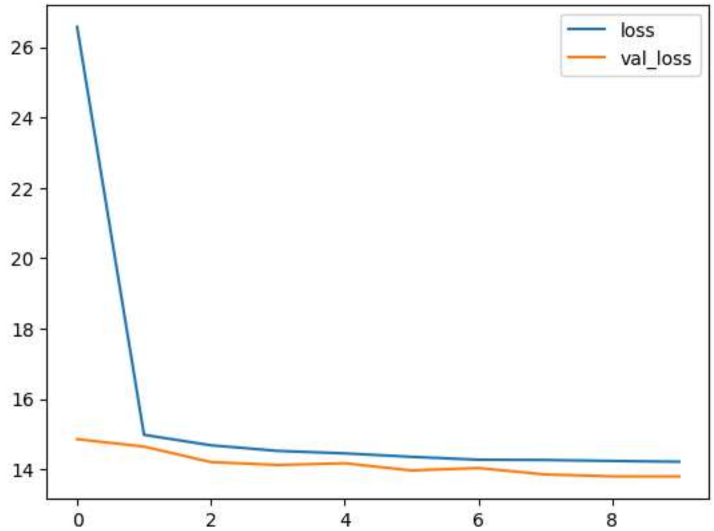
```
model.fit(X_train_scaled, y_train, epochs=10, batch_size=64, validation_data=(X_test_scaled, y_test))
```

```
Epoch 1/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 50.8363 - val_loss: 14.8594
Epoch 2/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 15.1739 - val_loss: 14.6484
Epoch 3/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 14.7838 - val_loss: 14.2078
Epoch 4/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 14.4648 - val_loss: 14.1260
Epoch 5/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 14.4911 - val_loss: 14.1745
Epoch 6/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 14.2183 - val_loss: 13.9706
Epoch 7/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 14.3523 - val_loss: 14.0393
Epoch 8/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 14.2533 - val_loss: 13.8563
Epoch 9/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 14.2440 - val_loss: 13.8011
Epoch 10/10
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 14.2972 - val_loss: 13.8011
<keras.src.callbacks.history.History at 0x7bd0aa98e150>
```

We can plot the loss and validation loss of the model.

```
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()
```

<Axes: >



10. 📈 Evaluating the Model

We predict scores on the test data and evaluate the model's performance using **Mean Absolute Error (MAE)**, which measures the average magnitude of prediction errors without considering their direction.

```
predictions = model.predict(X_test_scaled)

from sklearn.metrics import mean_absolute_error,mean_squared_error
mean_absolute_error(y_test,predictions)
```

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
14.291958808898926
```

11. 🖱️ Creating an Interactive Widget for Score Prediction

We build an interactive interface using **ipywidgets** so users can select match details and get a live predicted score.

- `widgets.Dropdown()` : creates dropdown menus for user input
- `widgets.Button()` : creates a clickable button to trigger prediction
- `predict_score()` : function that handles input, encoding, scaling, runs the model prediction, and displays the result
- `display()` : renders the widgets inside the notebook

```
import numpy as np
import ipywidgets as widgets
from IPython.display import display, clear_output
import warnings
warnings.filterwarnings("ignore")

venue = widgets.Dropdown(options=list(label_encoders['venue'].classes_), description='Select Venue:')
venue.style = {'description_width': 'initial'}

batting_team = widgets.Dropdown(options=list(label_encoders['bat_team'].classes_), description='Select Batting Team:')
batting_team.style = {'description_width': 'initial'}

bowling_team = widgets.Dropdown(options=list(label_encoders['bowl_team'].classes_), description='Select Bowling Team:')
bowling_team.style = {'description_width': 'initial'}

striker = widgets.Dropdown(options=list(label_encoders['batsman'].classes_), description='Select Striker:')
striker.style = {'description_width': 'initial'}

bowler = widgets.Dropdown(options=list(label_encoders['bowler'].classes_), description='Select Bowler:')
bowler.style = {'description_width': 'initial'}

runs = widgets.IntText(value=0, description='Runs:', style={'description_width': 'initial'})
wickets = widgets.IntText(value=0, description='Wickets:', style={'description_width': 'initial'})
overs = widgets.FloatText(value=0.0, description='Overs:', style={'description_width': 'initial'})
striker_ind = widgets.IntText(value=0, description='Striker:', style={'description_width': 'initial'})  # Assuming 0 or 1
```

```
predict_button = widgets.Button(description="Predict Score")

output = widgets.Output()

def predict_score(b):
    with output:
        clear_output()  # Clear previous output

        encoded_venue = label_encoders['venue'].transform([venue.value])[0]
        encoded_batting_team = label_encoders['bat_team'].transform([batting_team.value])[0]
        encoded_bowling_team = label_encoders['bowl_team'].transform([bowling_team.value])[0]
        encoded_striker = label_encoders['batsman'].transform([striker.value])[0]
        encoded_bowler = label_encoders['bowler'].transform([bowler.value])[0]

        input_features = [
```