Step 1: First, let's import all the necessary libraries

```
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
```

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

	mid	date	venue	bat_team	bowl_team	batsman	bowler	runs	wickets	overs	runs_l
0	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	SC Ganguly	P Kumar	1	0	0.1	
1	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	1	0	0.2	
2	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	2	0	0.2	
3	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	2	0	0.3	
4	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar	2	0	0.4	

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

Step 3: Data Pre-processing 3.1 Dropping unimportant features

We have created a new dataframe by dropping several columns from the original DataFrame. The new DataFrame contains the remaining columns that we are going to train the predictive model.

```
#Dropping certain features
df = ipl.drop(['date', 'runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5','mid', 'striker'
```

3.2 Further Pre-Processing

We have split the data frame into independent variable (X) and dependent variables (y). Our dependent variables is the total score.

```
X = df.drop(['total'], axis =1)
y = df['total']
```

3.3 Label Encoding

We have applied label encoding to your categorical features in X.

We have created separate LabelEncoder objects for each categorical feature and encoded their values.

We have created mappings to convert the encoded labels back to their original values, which can be helpful for interpreting the results.

```
#Label Encoding

from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object for each categorical feature
venue_encoder = LabelEncoder()
batting_team_encoder = LabelEncoder()
bowling_team_encoder = LabelEncoder()
striker_encoder = LabelEncoder()
bowler_encoder = LabelEncoder()

# Fit and transform the categorical features with label encoding
X['venue'] = venue_encoder.fit_transform(X['venue'])
X['bat_team'] = batting_team_encoder.fit_transform(X['bat_team'])
X['bowl_team'] = bowling_team_encoder.fit_transform(X['bowl_team'])
X['batsman'] = striker_encoder.fit_transform(X['batsman'])
X['bowler'] = bowler_encoder.fit_transform(X['bowler'])
```

3.4 Train Test Split

We have split the data into training and testing sets. The training set contains 70 percent of the dataset and rest 30 percent is in test set. X_train contains the training data for your input features. X_test contains the testing data for your input features. y_train contains the training data for your target variable. y_test contains the testing data for your target variable.

```
# Train test Split
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)
```

3.5 Feature Scaling

We have performed Min-Max scaling on our input features to ensure all the features are on the same scale

Scaling is performed to ensure consistent

scale to improve model performance.

Scaling has transformed both training and testing data using the scaling parameters.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 4: Define the Neural Network We have defined a neural network using TensorFlow and Keras for regression.

After defining the model, we have compiled the model using the Huber Loss because of the robustness of the regression against outliers.

```
# Define the neural network model
model = keras.Sequential([
    keras.layers.Input( shape=(X_train_scaled.shape[1],)),  # Input layer
    keras.layers.Dense(512, activation='relu'),  # Hidden layer with 512 units and ReLU activation
    keras.layers.Dense(216, activation='relu'),  # Hidden layer with 216 units and ReLU activation
    keras.layers.Dense(1, activation='linear')  # Output layer with linear activation for regressio
])

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

Step 5: Model Training

We have trained the neural network model using the scaled training data.

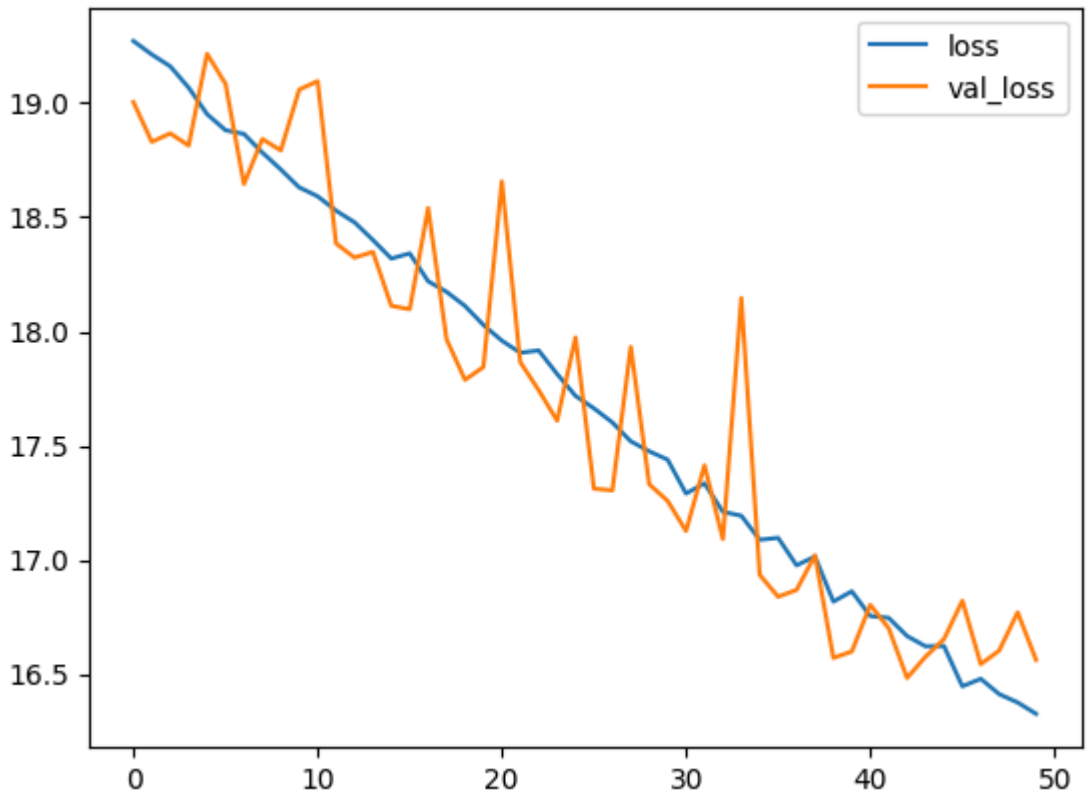
```
# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=64, validation_data=(X_test_scaled, y_test
```

```
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 17.3297 - val_loss: 17.4139
Epoch 33/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 17.1677 - val_loss: 17.0911
Epoch 34/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 17.0908 - val_loss: 18.1460
Epoch 35/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 17.1150 - val_loss: 16.9343
Epoch 36/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - loss: 17.1703 - val_loss: 16.8386
Epoch 37/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 17.0391 - val_loss: 16.8685
Epoch 38/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 17.0177 - val_loss: 17.0197
Epoch 39/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 16.8119 - val_loss: 16.5709
Epoch 40/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 16.6610 - val_loss: 16.5988
Epoch 41/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 16.7149 - val_loss: 16.8030
Epoch 42/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 16.7371 - val_loss: 16.7008
Epoch 43/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 16.5879 - val_loss: 16.4831
Epoch 44/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 16.5071 - val_loss: 16.5763
Epoch 45/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 16.6206 - val_loss: 16.6526
Epoch 46/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 16.4192 - val_loss: 16.8211
Epoch 47/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 16.4267 - val_loss: 16.5434
Epoch 48/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 16.4994 - val_loss: 16.6029
Epoch 49/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 16.3872 - val_loss: 16.7704
Epoch 50/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 16.3206 - val_loss: 16.5624
<keras.src.callbacks.history.History at 0x7d5df6f88a00>
```

After the training, we have stored the training and validation loss values to our neural network during the training process

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

<Axes: >



Step 6: Model Evaluation We have predicted using the trained neural network on the testing data.

The variable predictions contains the predicted total run scores for the test set based on the model's learned patterns

```
# Make predictions
predictions = model.predict(X_test_scaled)

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

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
17.053027915661858
```

Step 7: Let's create an Interactive Widget We have created an interactive widget using ipywidgets to predict the score based on user input for venue, batting team, bowling team, striker, and bowler.

We have created dropdown widgets to select values for venue, batting team, bowling team, striker, and bowler.

Then, we have added a "Predicted Score" button widget. Whenever, the button will be clicked, the predict_score function will be called and then perform the following steps: Decodes the user-selected values to their original categorical values.

Encodes and scales these values to match the format used in model training. Uses the trained model to make a prediction based on the user's input.

Displays the predicted score.

```
import ipywidgets as widgets
from IPython.display import display, clear_output

import warnings
warnings.filterwarnings("ignore")

venue = widgets.Dropdown(options=df['venue'].unique().tolist(),description='Select Venue:')
batting_team = widgets.Dropdown(options =df['bat_team'].unique().tolist(),  description='Select Bat
bowling_team = widgets.Dropdown(options=df['bowl_team'].unique().tolist(),  description='Select Bat
striker = widgets.Dropdown(options=df['batsman'].unique().tolist(), description='Select Striker:')
bowler = widgets.Dropdown(options=df['bowler'].unique().tolist(), description='Select Bowler:')

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

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


        # Decode the encoded values back to their original values
        decoded_venue = venue_encoder.transform([venue.value])
        decoded_batting_team = batting_team_encoder.transform([batting_team.value])
        decoded_bowling_team = bowling_team_encoder.transform([bowling_team.value])
        decoded_striker = striker_encoder.transform([striker.value])
        decoded_bowler = bowler_encoder.transform([bowler.value])


        input = np.array([decoded_venue,  decoded_batting_team, decoded_bowling_team,decoded_strike
        input = input.reshape(1,5)
        input = scaler.transform(input)
        #print(input)
        predicted_score = model.predict(input)
        predicted_score = int(predicted_score[0,0])

        print(predicted_score)
```

The widget-based interface allows you to interactively predict the score for specific match scenarios.

Now, we have set up the button to trigger the predict_score function when clicked and display the widgets for venue, batting team , bowling team, striker and bowler

```
predict_button.on_click(predict_score)
output = widgets.Output()
display(venue, batting_team, bowling_team, striker, bowler, predict_button, output)
```

Select Ven…	Eden Gardens
Select Batti…	Rajasthan Royals
Select Batti…	Chennai Super Kings
Select Strik…	R Dravid
Select Bow…	Pankaj Singh