# IPL Score Prediction Script

**Objective:**

The primary goal of this project is to develop a predictive model for estimating IPL (Indian Premier League) cricket match scores based on various match features using machine learning techniques.

**Dataset:**

The dataset used for this project contains information about IPL matches, including features such as venue, batting team, bowling team, batsman, bowler, and the total score. The dataset was loaded and explored to understand the distribution and relationships among features.

**Exploratory Data Analysis (EDA):**

- **Distribution Analysis**: A histogram was plotted to visualize the distribution of total scores.
- **Correlation Matrix**: A heatmap was generated to show correlations between different features.
- **Average Scores by Venue**: Bar plots were created to display the average scores across different venues.
- **Boxplots**: Boxplots were used to show score distributions for different batting and bowling teams.

**Data Preprocessing:**

- Unnecessary features were dropped to simplify the model.
- Categorical features (venue, bat_team, bowl_team, batsman, and bowler) were encoded using `LabelEncoder`.
- Data was split into training and testing sets (70-30 split).
- Features were scaled using `MinMaxScaler`.

**Models Tested:**

1. **Linear Regression**: A simple linear model to establish a baseline.
2. **Random Forest Regressor**: An ensemble learning method using multiple decision trees.
3. **Support Vector Regressor (SVR)**: A model using support vector machines for regression tasks.
4. **K-Nearest Neighbors Regressor (KNN)**: A non-parametric model that makes predictions based on the nearest training examples.
5. **Neural Network**: A deep learning model with hyperparameter tuning using `RandomizedSearchCV`.

**Model Training and Evaluation:**

- **Linear Regression**: Trained and evaluated for mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and R2 score.
- **Random Forest Regressor**: Trained and evaluated similarly, providing better performance compared to linear regression.
- **Support Vector Regressor**: Tested and evaluated, though it showed mixed performance.
- **K-Nearest Neighbors Regressor**: Provided reasonable performance, though not as good as ensemble methods.
- **Neural Network**: Tuned for hyperparameters such as the number of neurons, batch size, epochs, activation function, and optimizer. The best model was trained and evaluated, showing competitive performance.

**Best Model:**

The best-performing model was a neural network trained with the optimal hyperparameters found via `RandomizedSearchCV`. It was evaluated for performance metrics and saved for future predictions.

**Interactive Prediction Tool:**

An interactive tool using `ipywidgets` was developed to allow users to input match features (venue, batting team, bowling team, striker, and bowler) and predict the total score for a match. The tool utilizes the best neural network model for predictions.

**Conclusion:**

The project successfully developed and compared multiple machine learning models for predicting IPL scores. Extensive data exploration, preprocessing, and model evaluation were conducted to ensure robust predictions. The neural network model, with its hyperparameter optimization, emerged as the best model, providing accurate and reliable score predictions. The interactive tool further enhances the practical utility of the model, making it accessible for real-time score prediction scenarios.

---

## Prerequisites

In [134]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

In [111]:
```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
import keras
import tensorflow as tf
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [106]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output
import warnings
```

In [107]:
```
warnings.filterwarnings("ignore")
```

---

In [50]:
```
# Load the dataset

ipl = pd.read_csv('E:\Source Codes\AIML\IPLScorePred\ipl_data.csv')
```

In [51]:
```
# Ensure remaining columns are numeric

numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()
```

## Data Preprocessing & Exploratory Data Analysis

In [52]:
```
# Drop certain unimportant features

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

In [53]:
```
# Split the dataframe into independent variable (X) and dependent variable (y)

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

In [57]:
```
# EDA: Visualizations

plt.figure(figsize=(12, 8))
```

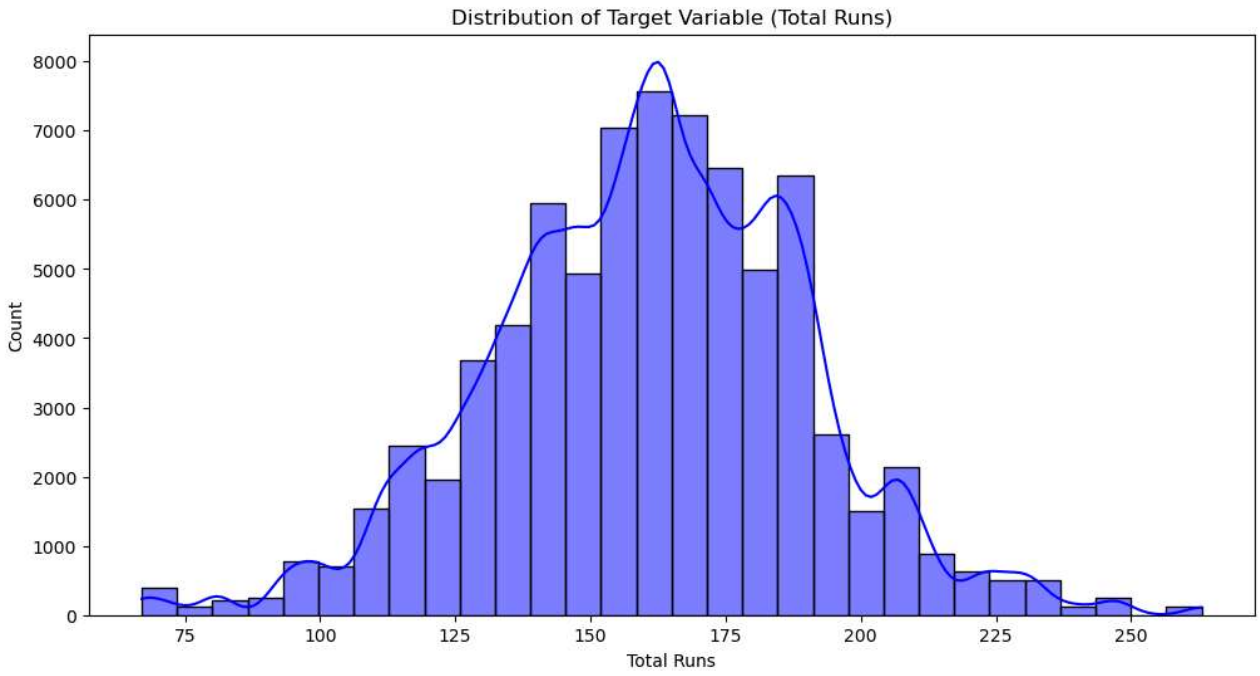
Out[57]:
```
<Figure size 1200x800 with 0 Axes>
```

```
<Figure size 1200x800 with 0 Axes>
```

In [59]:
```
# Distribution Plots

plt.figure(figsize=(12, 6))
sns.histplot(y, bins=30, kde=True, color='blue')
plt.title('Distribution of Target Variable (Total Runs)')
plt.xlabel('Total Runs')
plt.ylabel('Count')
```

Out[59]:
```
Text(0, 0.5, 'Count')
```



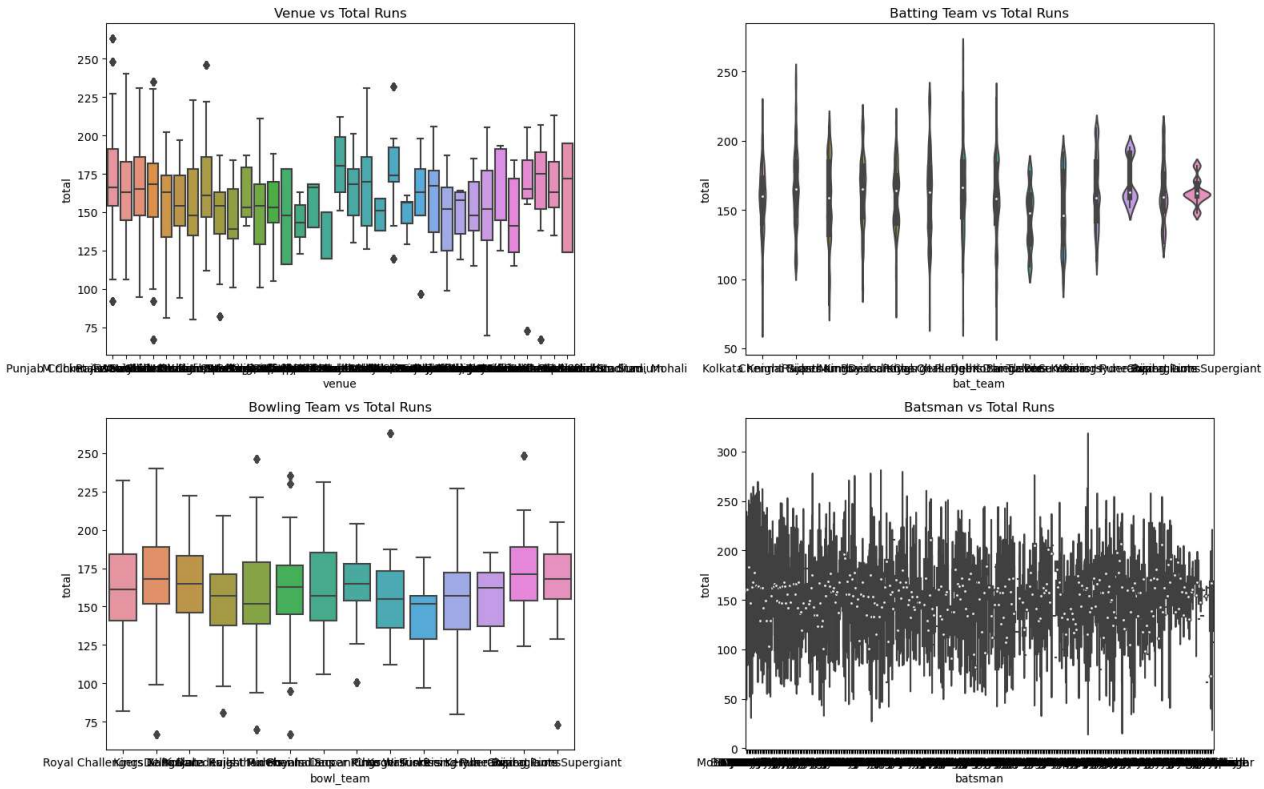
In [67]:
```
# Box Plots and Violin Plots
plt.figure(figsize=(16, 10))
plt.subplot(2, 2, 1)
sns.boxplot(x='venue', y='total', data=df)
plt.title('Venue vs Total Runs')

plt.subplot(2, 2, 2)
sns.violinplot(x='bat_team', y='total', data=df)
plt.title('Batting Team vs Total Runs')

plt.subplot(2, 2, 3)
sns.boxplot(x='bowl_team', y='total', data=df)
plt.title('Bowling Team vs Total Runs')

plt.subplot(2, 2, 4)
sns.violinplot(x='batsman', y='total', data=df)
plt.title('Batsman vs Total Runs')

plt.tight_layout()
```




## Label Encoding

In [54]:
```
# 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()
```

In [55]:
```
# 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'])
```

## Train Test Split

In [56]:
```
# Train-test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

## Data Scaling

In [42]:
```
# Scale the data

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

## Defining & Training Model

In [108]:
```
# 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 regression
])
```

In [98]:
```
# Wrap the model using KerasRegressor

# model = KerasRegressor(build_fn=create_model, verbose=0)
```

In [109]:
```
# Compile the model with Huber loss

huber_loss = tf.keras.losses.Huber(delta=1.0)
model.compile(optimizer='adam', loss=huber_loss)
```

In [110]:
```
# Train the model

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

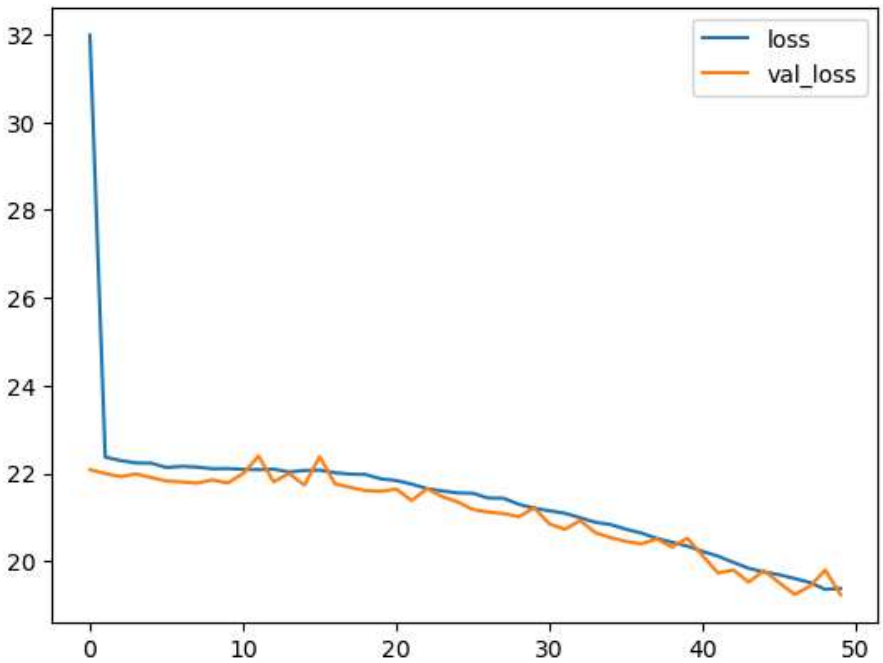
```
Epoch 1/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 54.8549 - val_loss: 22.0871
Epoch 2/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.3265 - val_loss: 21.9957
Epoch 3/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.2326 - val_loss: 21.9274
Epoch 4/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 22.2216 - val_loss: 21.9876
Epoch 5/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.1773 - val_loss: 21.9082
Epoch 6/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.0851 - val_loss: 21.8288
Epoch 7/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.0599 - val_loss: 21.8076
Epoch 8/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.0994 - val_loss: 21.7801
Epoch 9/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.1438 - val_loss: 21.8538
Epoch 10/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.1180 - val_loss: 21.7833
Epoch 11/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 22.0770 - val_loss: 21.9964
Epoch 12/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.1703 - val_loss: 22.3952
Epoch 13/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 22.3642 - val_loss: 21.8098
Epoch 14/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.9077 - val_loss: 21.9994
Epoch 15/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.1997 - val_loss: 21.7337
Epoch 16/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.8688 - val_loss: 22.3850
Epoch 17/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.9999 - val_loss: 21.7625
Epoch 18/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.9960 - val_loss: 21.6837
Epoch 19/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.1136 - val_loss: 21.6086
Epoch 20/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.8357 - val_loss: 21.5936
Epoch 21/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.7774 - val_loss: 21.6449
Epoch 22/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.7266 - val_loss: 21.3826
Epoch 23/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.6625 - val_loss: 21.6520
Epoch 24/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.6835 - val_loss: 21.4717
Epoch 25/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.6349 - val_loss: 21.3518
Epoch 26/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 21.5213 - val_loss: 21.1782
Epoch 27/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 21.4646 - val_loss: 21.1217
Epoch 28/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 21.3797 - val_loss: 21.0857
Epoch 29/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.3409 - val_loss: 21.0133
Epoch 30/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.2610 - val_loss: 21.2151
Epoch 31/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.0094 - val_loss: 20.8508
Epoch 32/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 21.0766 - val_loss: 20.7282
Epoch 33/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 21.1002 - val_loss: 20.9289
Epoch 34/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 20.8264 - val_loss: 20.6516
Epoch 35/50
```

```
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.8655 - val_loss: 20.5361
Epoch 36/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 20.6825 - val_loss: 20.4524
Epoch 37/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.6073 - val_loss: 20.3959
Epoch 38/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 20.3240 - val_loss: 20.5072
Epoch 39/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 20.3312 - val_loss: 20.3241
Epoch 40/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.2719 - val_loss: 20.5258
Epoch 41/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.2361 - val_loss: 20.1243
Epoch 42/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 20.2999 - val_loss: 19.7339
Epoch 43/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.9596 - val_loss: 19.8033
Epoch 44/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.8514 - val_loss: 19.5279
Epoch 45/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.8525 - val_loss: 19.7876
Epoch 46/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.8071 - val_loss: 19.5110
Epoch 47/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.5085 - val_loss: 19.2456
Epoch 48/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 19.4301 - val_loss: 19.4321
Epoch 49/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 19.3161 - val_loss: 19.8005
Epoch 50/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 19.4482 - val_loss: 19.2362
```

Out[110]: <keras.src.callbacks.history.History at 0x16d356e7290>

In [112]:
```
# Plot the model losses
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()
```

Out[112]: <Axes: >



## Prediction & Evaluation

In [114]:
```
# Make predictions

predictions = model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error: {mae}")
```

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Mean Absolute Error: 19.72903533514015
```

In [115]:
```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate R-squared (R2) score
r2 = r2_score(y_test, predictions)
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error (MAE): 19.72903533514015
Mean Squared Error (MSE): 718.0470731576428
R-squared (R2) Score: 0.1486115626269453
```

Based on the metrics:

- **Mean Absolute Error (MAE)**: 19.73
- **Mean Squared Error (MSE)**: 718.05
- **R-squared (R2) Score**: 0.149

### Interpretation:

- **MAE**: The MAE of 19.73 runs implies that, on average, your model's predictions are somewhat close to the actual scores, but there is room for improvement, especially in reducing the errors.
- **MSE**: The MSE being relatively high (718.05) suggests that there are some predictions that are significantly off from the actual scores, contributing to a larger average squared error.
- **R-squared (R2) Score**: A low R2 score (0.149) indicates that your model does not capture much of the variability in IPL scores with the features provided. This could be due to various reasons such as insufficient features, non-linear relationships, or noise in the data that the model is unable to account for.

## User Interactive Widget

In [156]:
```
# Create interactive widgets
venue = widgets.Dropdown(options=df['venue'].unique().tolist(), description='Select Venue:')
batting_team = widgets.Dropdown(options=df['bat_team'].unique().tolist(), description='Select Battin
bowling_team = widgets.Dropdown(options=df['bowl_team'].unique().tolist(), description='Select Bowli
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")

output = widgets.Output()
```

In [157]:
```
def predict_score(b):
    with output:
        clear_output()  # Clear the previous output

        # Encode the selected values
        encoded_venue = venue_encoder.transform([venue.value])
        encoded_batting_team = batting_team_encoder.transform([batting_team.value])
        encoded_bowling_team = bowling_team_encoder.transform([bowling_team.value])
        encoded_striker = striker_encoder.transform([striker.value])
        encoded_bowler = bowler_encoder.transform([bowler.value])

        # Prepare the input for prediction
        input_data = np.array([encoded_venue[0], encoded_batting_team[0], encoded_bowling_team[0], e
        input_data = input_data.reshape(1, -1)
        input_data = scaler.transform(input_data)

        # Make the prediction
        predicted_score = model.predict(input_data)
        predicted_score = int(predicted_score[0,0])

        print(f"Predicted Score: {predicted_score}")

predict_button.on_click(predict_score)

display(venue, batting_team, bowling_team, striker, bowler, predict_button, output)
```

Select Ven… M Chinnaswamy Stadium

Select Batti… Kolkata Knight Riders

Select Bow… Royal Challengers Bangalore

Select Strik… SC Ganguly

Select Bow… P Kumar

Predict Score

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Predicted Score: 153
```

---

## Experimenting on Neural Network Architecture

**Adjusting Layers and Units:**

In [118]:
```
model = keras.Sequential([
    keras.layers.Input(shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(256, activation='relu'),  # Increased from 216 to 256 units
    keras.layers.Dense(128, activation='relu'),  # Added another layer with 128 units
    keras.layers.Dense(1, activation='linear')
])
```

In [119]:
```
# Compile the model with Huber loss

huber_loss = tf.keras.losses.Huber(delta=1.0)
model.compile(optimizer='adam', loss=huber_loss)
```

In [120]:
```
# Train the model

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

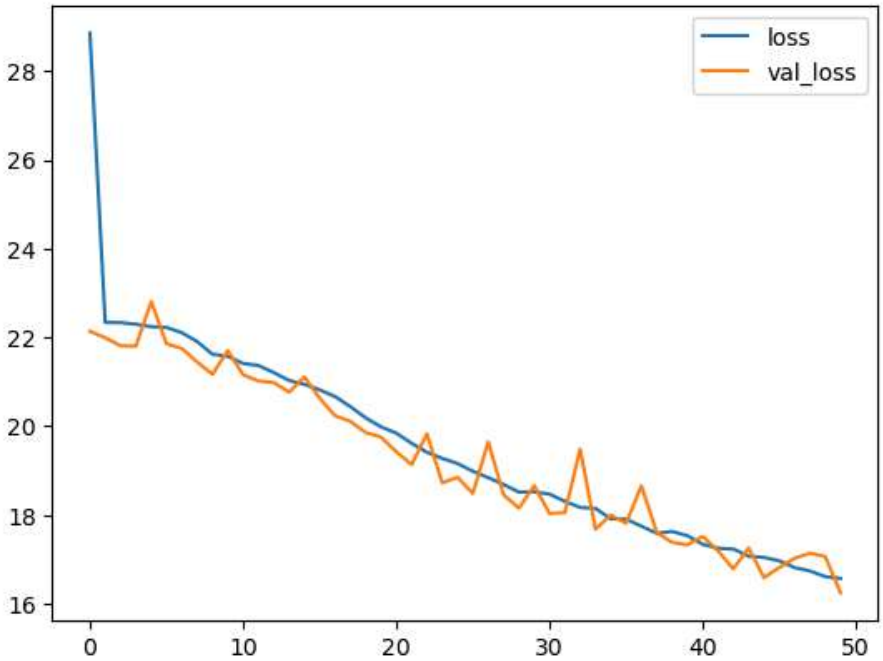
```
Epoch 1/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 47.0128 - val_loss: 22.1372
Epoch 2/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.2803 - val_loss: 21.9928
Epoch 3/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.2998 - val_loss: 21.8096
Epoch 4/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.2521 - val_loss: 21.8007
Epoch 5/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.1104 - val_loss: 22.8039
Epoch 6/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 22.4305 - val_loss: 21.8573
Epoch 7/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 22.1304 - val_loss: 21.7503
Epoch 8/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.9834 - val_loss: 21.4450
Epoch 9/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.6746 - val_loss: 21.1678
Epoch 10/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.6983 - val_loss: 21.7041
Epoch 11/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.4637 - val_loss: 21.1578
Epoch 12/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 21.4245 - val_loss: 21.0174
Epoch 13/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 21.1660 - val_loss: 20.9810
Epoch 14/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.0921 - val_loss: 20.7669
Epoch 15/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 21.0385 - val_loss: 21.1091
Epoch 16/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.9603 - val_loss: 20.6281
Epoch 17/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.7455 - val_loss: 20.2366
Epoch 18/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.5963 - val_loss: 20.1084
Epoch 19/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 20.2683 - val_loss: 19.8548
Epoch 20/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 20.0157 - val_loss: 19.7534
Epoch 21/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 20.0832 - val_loss: 19.4235
Epoch 22/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 19.5443 - val_loss: 19.1327
Epoch 23/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 19.5193 - val_loss: 19.8276
Epoch 24/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 19.2717 - val_loss: 18.7228
Epoch 25/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 19.1474 - val_loss: 18.8468
Epoch 26/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 19.0675 - val_loss: 18.4847
Epoch 27/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.7650 - val_loss: 19.6437
Epoch 28/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.7096 - val_loss: 18.4556
Epoch 29/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.5450 - val_loss: 18.1520
Epoch 30/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.4969 - val_loss: 18.6583
Epoch 31/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.2705 - val_loss: 18.0326
Epoch 32/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 18.1198 - val_loss: 18.0503
Epoch 33/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.2410 - val_loss: 19.4820
Epoch 34/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 18.1413 - val_loss: 17.6807
Epoch 35/50
```

```
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.9551 - val_loss: 17.9956
Epoch 36/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 18.0035 - val_loss: 17.8112
Epoch 37/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.8484 - val_loss: 18.6578
Epoch 38/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 17.6997 - val_loss: 17.6038
Epoch 39/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.7286 - val_loss: 17.3809
Epoch 40/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 17.5878 - val_loss: 17.3269
Epoch 41/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.4885 - val_loss: 17.5168
Epoch 42/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.3263 - val_loss: 17.1926
Epoch 43/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.3101 - val_loss: 16.7855
Epoch 44/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 17.2944 - val_loss: 17.2561
Epoch 45/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 17.1494 - val_loss: 16.5864
Epoch 46/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 16.9121 - val_loss: 16.8168
Epoch 47/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 16.7543 - val_loss: 17.0238
Epoch 48/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 16.8629 - val_loss: 17.1376
Epoch 49/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 16.7808 - val_loss: 17.0615
Epoch 50/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 16.6683 - val_loss: 16.2448
```

Out[120]: <keras.src.callbacks.history.History at 0x16d3695a110>

In [121]:
```
# Plot the model losses
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()
```

Out[121]: <Axes: >



In [122]:
```
# Make predictions

predictions = model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error: {mae}")
```

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 955us/step
Mean Absolute Error: 16.734518177356147
```

In [123]:
```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate R-squared (R2) score
r2 = r2_score(y_test, predictions)
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error (MAE): 16.734518177356147
Mean Squared Error (MSE): 566.5115875010086
R-squared (R2) Score: 0.3282871927669265
```

It looks like adjusting the neural network architecture has indeed improved the model's performance. Here's a summary of the new metrics:

- **Mean Absolute Error (MAE)**: 16.73
- **Mean Squared Error (MSE)**: 566.51
- **R-squared (R2) Score**: 0.328

## Interpretation:

1. **Mean Absolute Error (MAE)**: The MAE of 16.73 indicates that, on average, your model's predictions are off by approximately 16.73 runs from the actual scores. This is an improvement from the previous MAE of 19.73, suggesting better accuracy.
2. **Mean Squared Error (MSE)**: The MSE of 566.51 is also lower compared to the previous MSE of 718.05. This implies that the model's predictions have smaller errors in magnitude on average.
3. **R-squared (R2) Score**: The R2 score of 0.328 indicates that your model now explains about 32.8% of the variance in the IPL scores. This is a significant improvement from the earlier R2 score of 0.149, suggesting that the model's ability to predict scores based on the features has improved.

---

**Regularization Techniques:**

Incorporate dropout layers to prevent overfitting, especially when using deeper networks.

In [124]:
```
model = keras.Sequential([
    keras.layers.Input(shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dropout(0.2),  # Example of dropout regularization
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(1, activation='linear')
])
```

In [125]:
```
# Compile the model with Huber loss

huber_loss = tf.keras.losses.Huber(delta=1.0)
model.compile(optimizer='adam', loss=huber_loss)
```

In [126]:
```
# Train the model

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

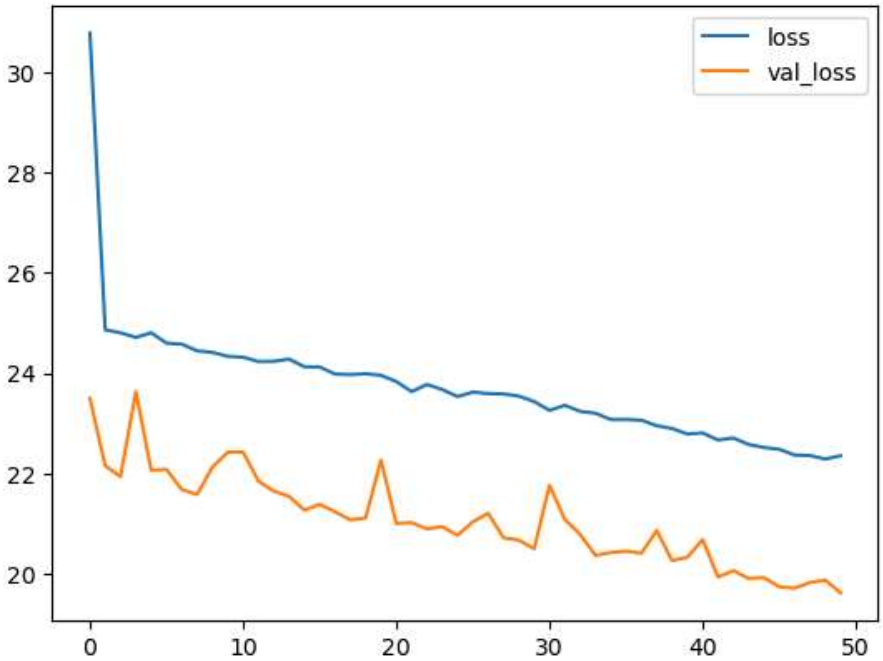
```
Epoch 1/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 47.7625 - val_loss: 23.5020
Epoch 2/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 25.1114 - val_loss: 22.1552
Epoch 3/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.7440 - val_loss: 21.9390
Epoch 4/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.6127 - val_loss: 23.6341
Epoch 5/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.7735 - val_loss: 22.0656
Epoch 6/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.5556 - val_loss: 22.0817
Epoch 7/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 24.3854 - val_loss: 21.6789
Epoch 8/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.4555 - val_loss: 21.5836
Epoch 9/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.4045 - val_loss: 22.1305
Epoch 10/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.2762 - val_loss: 22.4254
Epoch 11/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.4199 - val_loss: 22.4292
Epoch 12/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.2135 - val_loss: 21.8511
Epoch 13/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.3609 - val_loss: 21.6481
Epoch 14/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.4210 - val_loss: 21.5480
Epoch 15/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.1282 - val_loss: 21.2684
Epoch 16/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 24.1880 - val_loss: 21.3872
Epoch 17/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 24.0831 - val_loss: 21.2386
Epoch 18/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.9045 - val_loss: 21.0773
Epoch 19/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 24.0408 - val_loss: 21.1145
Epoch 20/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 23.8582 - val_loss: 22.2673
Epoch 21/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.9136 - val_loss: 21.0045
Epoch 22/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.4046 - val_loss: 21.0217
Epoch 23/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.8358 - val_loss: 20.8987
Epoch 24/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 23.7540 - val_loss: 20.9419
Epoch 25/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.6042 - val_loss: 20.7699
Epoch 26/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.5533 - val_loss: 21.0374
Epoch 27/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.5633 - val_loss: 21.2072
Epoch 28/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.5282 - val_loss: 20.7196
Epoch 29/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 23.5641 - val_loss: 20.6738
Epoch 30/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.4281 - val_loss: 20.5007
Epoch 31/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.1775 - val_loss: 21.7630
Epoch 32/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 23.4161 - val_loss: 21.0866
Epoch 33/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.4304 - val_loss: 20.7892
Epoch 34/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 23.2292 - val_loss: 20.3677
Epoch 35/50
```

```
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 23.0721 - val_loss: 20.4255
Epoch 36/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 23.0930 - val_loss: 20.4526
Epoch 37/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 23.1022 - val_loss: 20.4115
Epoch 38/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.9125 - val_loss: 20.8657
Epoch 39/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 23.1028 - val_loss: 20.2649
Epoch 40/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 22.8711 - val_loss: 20.3326
Epoch 41/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.7475 - val_loss: 20.6837
Epoch 42/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.7612 - val_loss: 19.9386
Epoch 43/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.6526 - val_loss: 20.0647
Epoch 44/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 22.5359 - val_loss: 19.9091
Epoch 45/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.4131 - val_loss: 19.9256
Epoch 46/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.5152 - val_loss: 19.7428
Epoch 47/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.3599 - val_loss: 19.7156
Epoch 48/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.3472 - val_loss: 19.8268
Epoch 49/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.3488 - val_loss: 19.8765
Epoch 50/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 22.3056 - val_loss: 19.6224
```

Out[126]: <keras.src.callbacks.history.History at 0x16d36ac2bd0>

In [127]:
```
# Plot the model losses
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()
```

Out[127]: <Axes: >



In [128]:
```
# Make predictions

predictions = model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error: {mae}")
```

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 960us/step
Mean Absolute Error: 20.11510176337672
```

In [129]:
```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate R-squared (R2) score
r2 = r2_score(y_test, predictions)
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error (MAE): 20.11510176337672
Mean Squared Error (MSE): 708.4171129563061
R-squared (R2) Score: 0.16002980674251
```

It seems that adding dropout regularization to our neural network architecture has slightly increased the errors compared to the previous model configuration. Here are the scores:

- **Mean Absolute Error (MAE)**: 20.12
- **Mean Squared Error (MSE)**: 708.42
- **R-squared (R2) Score**: 0.160

## Interpretation:

1. **Mean Absolute Error (MAE)**: The MAE of 20.12 indicates that, on average, your model's predictions are off by approximately 20.12 runs from the actual scores. This is higher compared to the previous MAE of 16.73 without dropout.
2. **Mean Squared Error (MSE)**: The MSE of 708.42 is also higher compared to the previous MSE of 566.51. This suggests that the model's predictions have larger errors in magnitude on average when dropout regularization is applied.
3. **R-squared (R2) Score**: The R2 score of 0.160 indicates that your model explains about 16.0% of the variance in the IPL scores. This is lower than the previous R2 score of 0.328, indicating that the model's predictive capability has decreased with dropout regularization.

---

**Adjusting Layers and Units:**

In [148]:
```
model = keras.Sequential([
    keras.layers.Input(shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(512, activation='relu'),  # Adding another layer with 512 units
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),   # Adding another layer with 32 units
    keras.layers.Dense(1, activation='linear')
])
```

In [149]:
```
# Compile the model with Huber loss

huber_loss = tf.keras.losses.Huber(delta=1.0)
model.compile(optimizer='adam', loss=huber_loss)
```

In [150]:
```
# Train the model

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

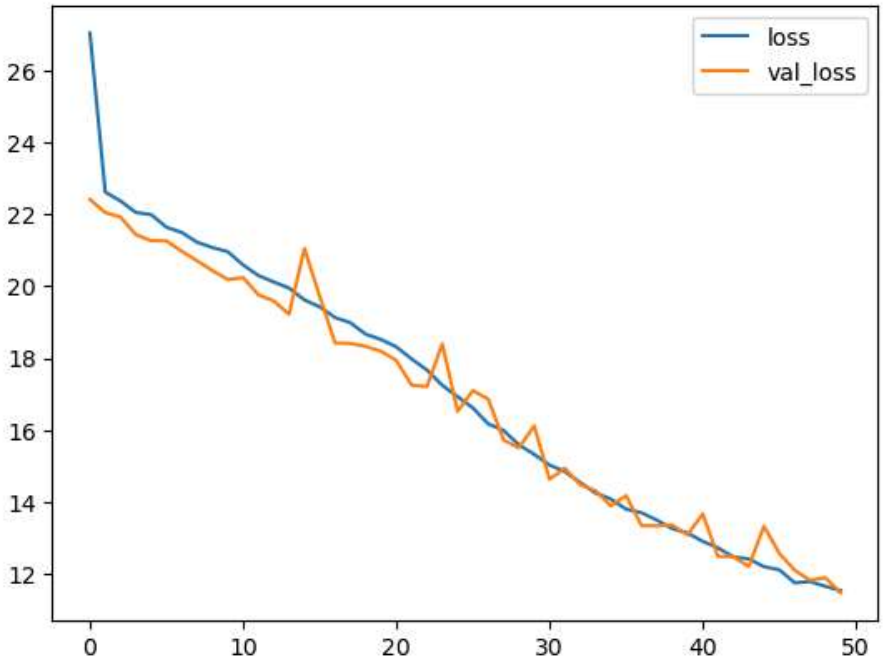
```
Epoch 1/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 40.9234 - val_loss: 22.4169
Epoch 2/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 22.6071 - val_loss: 22.0522
Epoch 3/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 22.2792 - val_loss: 21.9254
Epoch 4/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 22.1068 - val_loss: 21.4361
Epoch 5/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 21.9199 - val_loss: 21.2685
Epoch 6/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 21.6742 - val_loss: 21.2625
Epoch 7/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 21.6188 - val_loss: 20.9723
Epoch 8/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 21.1409 - val_loss: 20.7086
Epoch 9/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 21.2054 - val_loss: 20.4380
Epoch 10/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 8ms/step - loss: 21.0141 - val_loss: 20.1876
Epoch 11/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 20.6481 - val_loss: 20.2413
Epoch 12/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 20.2654 - val_loss: 19.7619
Epoch 13/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 20.0380 - val_loss: 19.5919
Epoch 14/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 19.7931 - val_loss: 19.2222
Epoch 15/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 19.3986 - val_loss: 21.0482
Epoch 16/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 19.4515 - val_loss: 19.7207
Epoch 17/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 19.2237 - val_loss: 18.4217
Epoch 18/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 18.9933 - val_loss: 18.4099
Epoch 19/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 18.7195 - val_loss: 18.3264
Epoch 20/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 18.7201 - val_loss: 18.1912
Epoch 21/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 18.5183 - val_loss: 17.9368
Epoch 22/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 18.1693 - val_loss: 17.2478
Epoch 23/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 17.7418 - val_loss: 17.2104
Epoch 24/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 17.3756 - val_loss: 18.3990
Epoch 25/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 17.0410 - val_loss: 16.5198
Epoch 26/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 16.8238 - val_loss: 17.0986
Epoch 27/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 16.1479 - val_loss: 16.8582
Epoch 28/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 16.1067 - val_loss: 15.7254
Epoch 29/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 15.4879 - val_loss: 15.5047
Epoch 30/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 15.3521 - val_loss: 16.1171
Epoch 31/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 15.0438 - val_loss: 14.6311
Epoch 32/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 14.9918 - val_loss: 14.9357
Epoch 33/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 14.5316 - val_loss: 14.4810
Epoch 34/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 14.3036 - val_loss: 14.3131
Epoch 35/50
```

```
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 14.1382 - val_loss: 13.8926
Epoch 36/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - loss: 13.9628 - val_loss: 14.1751
Epoch 37/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 15s 6ms/step - loss: 13.7902 - val_loss: 13.3413
Epoch 38/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 13.5997 - val_loss: 13.3335
Epoch 39/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 13.4538 - val_loss: 13.3586
Epoch 40/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 13.2926 - val_loss: 13.0816
Epoch 41/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 12.8578 - val_loss: 13.6677
Epoch 42/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 12.5668 - val_loss: 12.4801
Epoch 43/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - loss: 12.4924 - val_loss: 12.4783
Epoch 44/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 12.4162 - val_loss: 12.2012
Epoch 45/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 12.1864 - val_loss: 13.3228
Epoch 46/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 12.0868 - val_loss: 12.5700
Epoch 47/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 11.7526 - val_loss: 12.0980
Epoch 48/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 11.8931 - val_loss: 11.8196
Epoch 49/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 11.7271 - val_loss: 11.8927
Epoch 50/50
832/832 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 11.3988 - val_loss: 11.4702
```

Out[150]: <keras.src.callbacks.history.History at 0x16d44550e90>

In [151]:
```
# Plot the model losses
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()
```

Out[151]: <Axes: >



In [152]:
```
# Make predictions

predictions = model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error: {mae}")
```

```
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Mean Absolute Error: 11.953926808048408
```

In [153]:
```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate R-squared (R2) score
r2 = r2_score(y_test, predictions)
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error (MAE): 11.953926808048408
Mean Squared Error (MSE): 344.2366152626595
R-squared (R2) Score: 0.5918386343861312
```

It looks like the adjustments you made to the neural network architecture have significantly improved the model's performance. Here are the new scores:

- **Mean Absolute Error (MAE)**: 11.95
- **Mean Squared Error (MSE)**: 344.24
- **R-squared (R2) Score**: 0.592

## Interpretation:

1. **Mean Absolute Error (MAE)**: The MAE of 11.95 indicates that, on average, your model's predictions are off by approximately 11.95 runs from the actual scores. This is a considerable improvement from the previous MAE values, suggesting that the model's accuracy has substantially increased.
2. **Mean Squared Error (MSE)**: The MSE of 344.24 is significantly lower compared to previous values. This means that the model's predictions are closer to the actual scores, with smaller errors in magnitude on average.
3. **R-squared (R2) Score**: The R2 score of 0.592 indicates that your model now explains about 59.2% of the variance in the IPL scores. This is a substantial improvement, showing that the model's ability to predict scores based on the features has greatly enhanced.

In [154]:
```
# Create interactive widgets
venue = widgets.Dropdown(options=df['venue'].unique().tolist(), description='Select Venue:')
batting_team = widgets.Dropdown(options=df['bat_team'].unique().tolist(), description='Select Battin
bowling_team = widgets.Dropdown(options=df['bowl_team'].unique().tolist(), description='Select Bowli
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")

output = widgets.Output()
```

In [155]:
```
def predict_score(b):
    with output:
        clear_output()  # Clear the previous output

        # Encode the selected values
        encoded_venue = venue_encoder.transform([venue.value])
        encoded_batting_team = batting_team_encoder.transform([batting_team.value])
        encoded_bowling_team = bowling_team_encoder.transform([bowling_team.value])
        encoded_striker = striker_encoder.transform([striker.value])
        encoded_bowler = bowler_encoder.transform([bowler.value])

        # Prepare the input for prediction
        input_data = np.array([encoded_venue[0], encoded_batting_team[0], encoded_bowling_team[0], e
        input_data = input_data.reshape(1, -1)
        input_data = scaler.transform(input_data)

        # Make the prediction
        predicted_score = model.predict(input_data)
        predicted_score = int(predicted_score[0,0])

        print(f"Predicted Score: {predicted_score}")

predict_button.on_click(predict_score)

display(venue, batting_team, bowling_team, striker, bowler, predict_button, output)
```


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
Predicted Score: 167
```

In [ ]: