In [1]:

```python
# Mounting the Google Drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing the dictories

In [2]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoCV
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as
from joblib import dump
```

Getting the data

In [3]:

```python
df=pd.read_csv('/content/drive/My Drive/ipl prediction/ipl_colab.csv')
print(f"Dataset successfully Imported of Shape : {df.shape}")
```

```
Dataset successfully Imported of Shape : (76014, 15)
```

In [4]:

```python
df.head()
```

Out[4]:

	mid	date	venue	batting_team	bowling_team	batsman	bowler
0	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	SC Ganguly	P Kumar
1	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar
2	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar
3	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar
4	1	2008-04-18	M Chinnaswamy Stadium	Kolkata Knight Riders	Royal Challengers Bangalore	BB McCullum	P Kumar

Exploratory Data Analysis

In [7]:

```
# Describing Numerical Values of the Dataset

df.describe()
```

Out[7]:

	mid	runs	wickets	overs	runs_last_5	w
count	76014.000000	76014.000000	76014.000000	76014.000000	76014.000000	7
mean	308.627740	74.889349	2.415844	9.783068	33.216434	
std	178.156878	48.823327	2.015207	5.772587	14.914174	
min	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	154.000000	34.000000	1.000000	4.600000	24.000000	
50%	308.000000	70.000000	2.000000	9.600000	34.000000	
75%	463.000000	111.000000	4.000000	14.600000	43.000000	
max	617.000000	263.000000	10.000000	19.600000	113.000000	

In [8]:

```
# Information (not-null count and data type) About Each Column

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76014 entries, 0 to 76013
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   mid             76014 non-null  int64  
 1   date            76014 non-null  object 
 2   venue           76014 non-null  object 
 3   batting_team    76014 non-null  object 
 4   bowling_team    76014 non-null  object 
 5   batsman         76014 non-null  object 
 6   bowler          76014 non-null  object 
 7   runs            76014 non-null  int64  
 8   wickets         76014 non-null  int64  
 9   overs           76014 non-null  float64
 10  runs_last_5     76014 non-null  int64  
 11  wickets_last_5  76014 non-null  int64  
 12  striker         76014 non-null  int64  
 13  non-striker     76014 non-null  int64  
 14  total           76014 non-null  int64  
dtypes: float64(1), int64(8), object(6)
memory usage: 8.7+ MB
```

In [9]:

```
# Number of Unique Values in each column

df.nunique()
```

Out[9]:

	0
mid	617
date	442
venue	35
batting_team	14
bowling_team	14
batsman	411
bowler	329
runs	252
wickets	11
overs	140
runs_last_5	102
wickets_last_5	8
striker	155
non-striker	88
total	138

dtype: int64

In [10]:

```
# Datatypes of all Columns

df.dtypes
```

Out[10]:

	0
mid	int64
date	object
venue	object
batting_team	object
bowling_team	object
batsman	object
bowler	object
runs	int64
wickets	int64
overs	float64
runs_last_5	int64
wickets_last_5	int64
striker	int64
non-striker	int64
total	int64

dtype: object

Data cleaning

Removing the irrevlant coloumn

In [12]:
```
# Names of all columns

df.columns
```

Out[12]:
```
Index(['mid', 'date', 'venue', 'batting_team', 'bowling_team', 'batsman',
       'bowler', 'runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5',
       'striker', 'non-striker', 'total'],
      dtype='object')
```

Here, we can see that columns *['mid', 'date', 'venue', 'batsman', 'bowler', 'striker', 'non-striker']* won't provide any relevant information for our model to train

In [13]:
```
irrelevant = ['mid', 'date', 'venue','batsman', 'bowler', 'striker', 'non-stri
print(f'Before Removing Irrelevant Columns : {df.shape}')
df = df.drop(irrelevant, axis=1) # Drop Irrelevant Columns
print(f'After Removing Irrelevant Columns : {df.shape}')
df.head()
```

```
Before Removing Irrelevant Columns : (76014, 15)
After Removing Irrelevant Columns : (76014, 8)
```

Out[13]:

	batting_team	bowling_team	runs	wickets	overs	runs_last_5	wickets_last_
0	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.1	1	(
1	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.2	1	(
2	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.2	2	(
3	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.3	2	(
4	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.4	2	(

Keeping only Consistent Teams

(teams that never change even in current season)

In [14]:

```
# Consistent Teams

const_teams = ['Kolkata Knight Riders', 'Chennai Super Kings', 'Rajasthan Roya
               'Mumbai Indians', 'Kings XI Punjab', 'Royal Challengers Bangalor
               'Delhi Daredevils', 'Sunrisers Hyderabad']
```

In [15]:

```
print(f'Before Removing Inconsistent Teams : {df.shape}')
df = df[(df['batting_team'].isin(const_teams)) & (df['bowling_team'].isin(cons
print(f'After Removing Irrelevant Columns : {df.shape}')
print(f"Consistent Teams : \n{df['batting_team'].unique()}")
df.head()
```

```
Before Removing Inconsistent Teams : (76014, 8)
After Removing Irrelevant Columns : (53811, 8)
Consistent Teams :
['Kolkata Knight Riders' 'Chennai Super Kings' 'Rajasthan Royals'
 'Mumbai Indians' 'Kings XI Punjab' 'Royal Challengers Bangalore'
 'Delhi Daredevils' 'Sunrisers Hyderabad']
```

Out[15]:

	batting_team	bowling_team	runs	wickets	overs	runs_last_5	wickets_last_
0	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.1	1	0
1	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.2	1	0
2	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.2	2	0
3	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.3	2	0
4	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.4	2	0

Remove First 5 Overs of every match

In [16]:

```python
print(f'Before Removing Overs : {df.shape}')
df = df[df['overs'] >= 5.0]
print(f'After Removing Overs : {df.shape}')
df.head()
```

```
Before Removing Overs : (53811, 8)
After Removing Overs : (40108, 8)
```

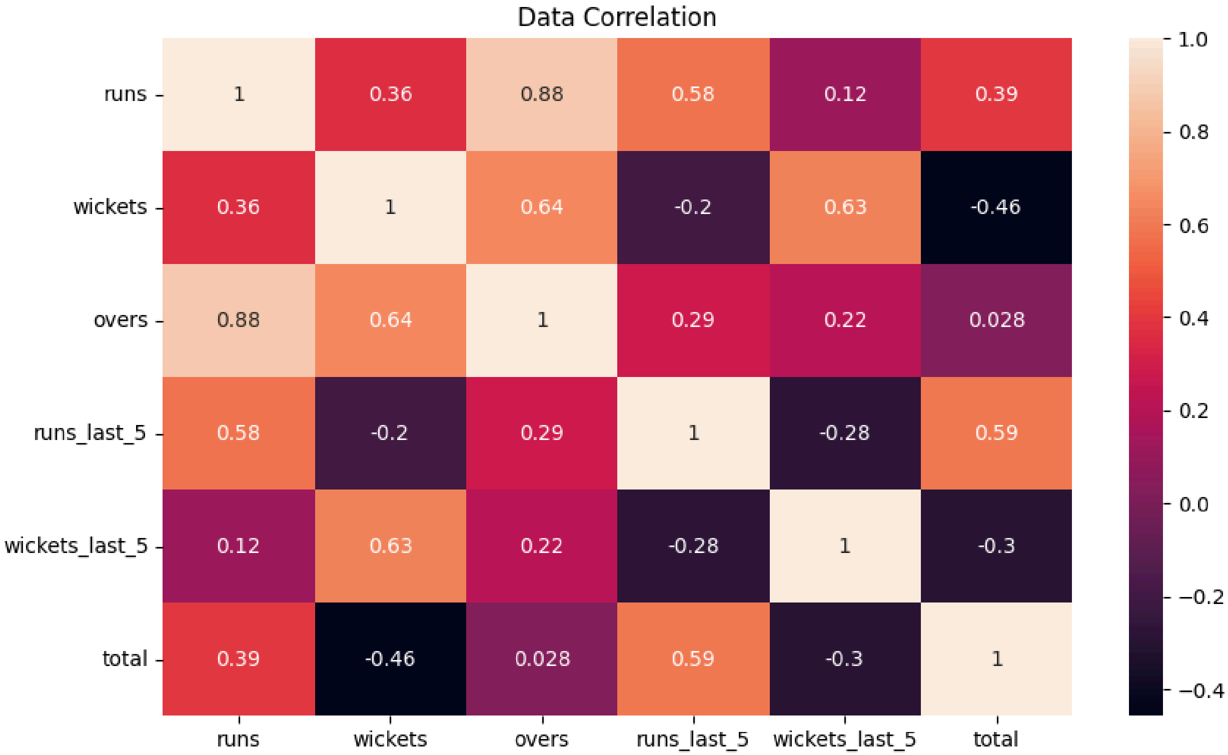
Out[16]:

	batting_team	bowling_team	runs	wickets	overs	runs_last_5	wickets_last
32	Kolkata Knight Riders	Royal Challengers Bangalore	61	0	5.1	59	
33	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.2	59	
34	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.3	59	
35	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.4	59	
36	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.5	58	

In [20]:

```python
plt.figure(figsize=(10, 6))
sns.heatmap(data=df.select_dtypes(include=np.number).corr(),annot=True)
```

```
plt.title('Data Correlation')
plt.show()
```



Data Preprocessing and Encoding

Performing Label Encoding

In [21]:
```
le = LabelEncoder()
for col in ['batting_team', 'bowling_team']:
  df[col] = le.fit_transform(df[col])
df.head()
```

Out[21]:

	batting_team	bowling_team	runs	wickets	overs	runs_last_5	wickets_last
32	3	6	61	0	5.1	59	
33	3	6	61	1	5.2	59	
34	3	6	61	1	5.3	59	
35	3	6	61	1	5.4	59	
36	3	6	61	1	5.5	58	

In [22]:
```
columnTransformer = ColumnTransformer([('encoder',
                                        OneHotEncoder(),
                                        [0, 1])],
                                      remainder='passthrough')
```

In [24]:

```
df = np.array(columnTransformer.fit_transform(df))
```

In [25]:

```
df
```

Out[25]:

```
array([[  0.,   0.,   0., ...,  59.,   0., 222.],
       [  0.,   0.,   0., ...,  59.,   1., 222.],
       [  0.,   0.,   0., ...,  59.,   1., 222.],
       ...,
       [  0.,   0.,   0., ...,  28.,   4., 107.],
       [  0.,   0.,   0., ...,  24.,   4., 107.],
       [  0.,   0.,   0., ...,  23.,   5., 107.]])
```

Saving the Numpy Array in a new DataFrame with transformed columns

In [26]:

```
cols = ['batting_team_Chennai Super Kings', 'batting_team_Delhi Daredevils', '
              'batting_team_Kolkata Knight Riders', 'batting_team_Mumbai India
              'batting_team_Royal Challengers Bangalore', 'batting_team_Sunris
              'bowling_team_Chennai Super Kings', 'bowling_team_Delhi Daredevi
              'bowling_team_Kolkata Knight Riders', 'bowling_team_Mumbai India
              'bowling_team_Royal Challengers Bangalore', 'bowling_team_Sunris
       'runs_last_5', 'wickets_last_5', 'total']
df_update = pd.DataFrame(df, columns=cols)
```

In [27]:

```
df_update.head()
```

Out[27]:

	batting_team_Chennai Super Kings	batting_team_Delhi Daredevils	batting_team_Kings XI Punjab	batting_team Knigh
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

5 rows × 22 columns

Model Building

Preparing Train and Test Splits

In [28]:

```
features=df_update.drop('total',axis=1)
labels=df_update['total']
```

In [29]:

```
# Perform 80 : 20 Train-Test split

train_features, test_features, train_labels, test_labels = train_test_split(fe
print(f"Training Set : {train_features.shape}\nTesting Set : {test_features.sh
```

```
Training Set : (32086, 21)
Testing Set : (8022, 21)
```

Model Algorithms

Training and Testing on different Machine Learning Algorithms for the best algorithm to choose from

In [30]:

```
# Keeping track of model perfomances
models = dict()
```

1. Decision Tree Regressor

In [31]:

```
tree = DecisionTreeRegressor()
# Train Model
tree.fit(train_features, train_labels)
```

Out[31]:

```
▾ DecisionTreeRegressor ⓘ ?

DecisionTreeRegressor()
```

In [32]:

```
# Evaluate Model
train_score_tree = str(tree.score(train_features, train_labels) * 100)
test_score_tree = str(tree.score(test_features, test_labels) * 100)
print(f'Train Score : {train_score_tree[:5]}%\nTest Score : {test_score_tree[:
models["tree"] = test_score_tree
```

```
Train Score : 99.98%
Test Score : 85.81%
```

In [35]:

```
print("---- Decision Tree Regressor - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, tree.predict(tes
print("Mean Squared Error (MSE): {}".format(mse(test_labels, tree.predict(test
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, tre
```

```
---- Decision Tree Regressor - Model Evaluation ----
Mean Absolute Error (MAE): 3.9846048366990776
Mean Squared Error (MSE): 124.57625903764647
Root Mean Squared Error (RMSE): 11.161373528273591
```

Linear Regression

In [36]:

```
linreg = LinearRegression()
# Train Model
linreg.fit(train_features, train_labels)
```

```
Out[36]:
▾ LinearRegression
LinearRegression()
```

```
In [37]:
# Evaluate Model
train_score_linreg = str(linreg.score(train_features, train_labels) * 100)
test_score_linreg = str(linreg.score(test_features, test_labels) * 100)
print(f'Train Score : {train_score_linreg[:5]}%\nTest Score : {test_score_linr
models["linreg"] = test_score_linreg
```

```
Train Score : 65.86%
Test Score : 66.14%
```

```
In [38]:
print("---- Linear Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, linreg.predict(t
print("Mean Squared Error (MSE): {}".format(mse(test_labels, linreg.predict(te
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, lir
```

```
---- Linear Regression - Model Evaluation ----
Mean Absolute Error (MAE): 12.887828256110065
Mean Squared Error (MSE): 297.3907656063563
Root Mean Squared Error (RMSE): 17.24502147306162
```

Random Forest Regression

```
In [39]:
forest = RandomForestRegressor()
# Train Model
forest.fit(train_features, train_labels)
```

```
Out[39]:
▾ RandomForestRegressor
RandomForestRegressor()
```

```
In [42]:
# Evaluate Model
train_score_forest = str(forest.score(train_features, train_labels)*100)
test_score_forest = str(forest.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_forest[:5]}%\nTest Score : {test_score_fore
models["forest"] = test_score_forest
```

```
Train Score : 99.08%
Test Score : 93.29%
```

```
In [43]:
print("---- Random Forest Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, forest.predict(t
print("Mean Squared Error (MSE): {}".format(mse(test_labels, forest.predict(te
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, for
```

```
---- Random Forest Regression - Model Evaluation ----
Mean Absolute Error (MAE): 4.488915440870939
Mean Squared Error (MSE): 58.90633267677083
Root Mean Squared Error (RMSE): 7.675046102582761
```

Lasso Regression

In [44]:
```
lasso = LassoCV()
# Train Model
lasso.fit(train_features, train_labels)
```

Out[44]:
```
▾ LassoCV i ?

LassoCV()
```

In [45]:
```
# Evaluate Model
train_score_lasso = str(lasso.score(train_features, train_labels)*100)
test_score_lasso = str(lasso.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_lasso[:5]}%\nTest Score : {test_score_lassc
models["lasso"] = test_score_lasso
```

```
Train Score : 64.85%
Test Score : 65.23%
```

In [46]:
```
print("---- Lasso Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, lasso.predict(te
print("Mean Squared Error (MSE): {}".format(mse(test_labels, lasso.predict(tes
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, las
```

```
---- Lasso Regression - Model Evaluation ----
Mean Absolute Error (MAE): 12.907475661661291
Mean Squared Error (MSE): 305.3495162925614
Root Mean Squared Error (RMSE): 17.474252953776347
```

Support Vector Machine

In [47]:
```
svm = SVR()
# Train Model
svm.fit(train_features, train_labels)
```

Out[47]:
```
▾ SVR i ?

SVR()
```

In [48]:
```
train_score_svm = str(svm.score(train_features, train_labels)*100)
test_score_svm = str(svm.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_svm[:5]}%\nTest Score : {test_score_svm[:5]
models["svm"] = test_score_svm
```

```
Train Score : 57.31%
Test Score : 57.77%
```

In [49]:
```
print("---- Support Vector Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, svm.predict(test
print("Mean Squared Error (MSE): {}".format(mse(test_labels, svm.predict(test_
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, svm
---- Support Vector Regression - Model Evaluation ----
Mean Absolute Error (MAE): 14.506947905764388
Mean Squared Error (MSE): 370.90307073456455
Root Mean Squared Error (RMSE): 19.258843961530104
```

Neural Networks

In [51]:
```
neural_net = MLPRegressor(activation='logistic', max_iter=500)
# Train Model
neural_net.fit(train_features, train_labels)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perc
eptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50
0) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[51]:
```
▾             MLPRegressor
MLPRegressor(activation='logistic', max_iter=500)
```

In [52]:
```
train_score_neural_net = str(neural_net.score(train_features, train_labels)*10
test_score_neural_net = str(neural_net.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_neural_net[:5]}%\nTest Score : {test_score_
models["neural_net"] = test_score_neural_net
```

```
Train Score : 86.17%
Test Score : 85.04%
```

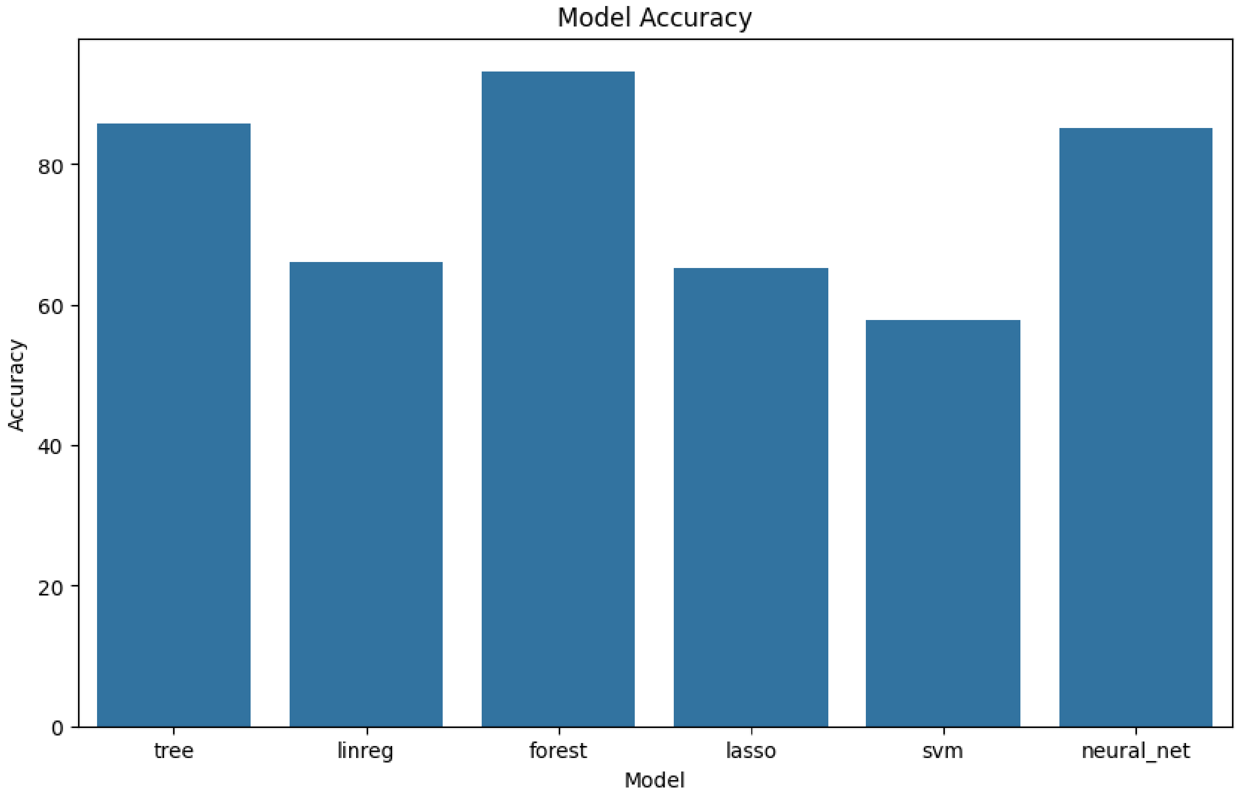
In [53]:
```
print("---- Neural Networks Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, neural_net.predi
print("Mean Squared Error (MSE): {}".format(mse(test_labels, neural_net.predic
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labels, neu
```

```
---- Neural Networks Regression - Model Evaluation ----
Mean Absolute Error (MAE): 8.139291198630994
Mean Squared Error (MSE): 131.379784745018
Root Mean Squared Error (RMSE): 11.462102108471115
```

Best Model Selection

In [56]:
```
model_names = list(models.keys())
accuracy = list(map(float, models.values()))

plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=accuracy)
plt.title('Model Accuracy')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.show()
```



From above, we can see that **Random Forest** performed the best, closely followed by **Decision Tree** and **Neural Networks**. So we will be choosing Random Forest for the final mode

Predictions

In [57]:

```
def predict_score(batting_team, bowling_team, runs, wickets, overs, runs_last_
  prediction_array = []

  # Batting Team
  if batting_team == 'Chennai Super Kings':
    prediction_array = prediction_array + [1,0,0,0,0,0,0,0]
  elif batting_team == 'Delhi Daredevils':
    prediction_array = prediction_array + [0,1,0,0,0,0,0,0]
  elif batting_team == 'Kings XI Punjab':
    prediction_array = prediction_array + [0,0,1,0,0,0,0,0]
  elif batting_team == 'Kolkata Knight Riders':
    prediction_array = prediction_array + [0,0,0,1,0,0,0,0]
  elif batting_team == 'Mumbai Indians':
    prediction_array = prediction_array + [0,0,0,0,1,0,0,0]
  elif batting_team == 'Rajasthan Royals':
    prediction_array = prediction_array + [0,0,0,0,0,1,0,0]
  elif batting_team == 'Royal Challengers Bangalore':
    prediction_array = prediction_array + [0,0,0,0,0,0,1,0]
  elif batting_team == 'Sunrisers Hyderabad':
    prediction_array = prediction_array + [0,0,0,0,0,0,0,1]
  # Bowling Team
  if bowling_team == 'Chennai Super Kings':
    prediction_array = prediction_array + [1,0,0,0,0,0,0,0]
  elif bowling_team == 'Delhi Daredevils':
    prediction_array = prediction_array + [0,1,0,0,0,0,0,0]
  elif bowling_team == 'Kings XI Punjab':
    prediction_array = prediction_array + [0,0,1,0,0,0,0,0]
  elif bowling_team == 'Kolkata Knight Riders':
    prediction_array = prediction_array + [0,0,0,1,0,0,0,0]
  elif bowling_team == 'Mumbai Indians':
    prediction_array = prediction_array + [0,0,0,0,1,0,0,0]
  elif bowling_team == 'Rajasthan Royals':
    prediction_array = prediction_array + [0,0,0,0,0,1,0,0]
  elif bowling_team == 'Royal Challengers Bangalore':
    prediction_array = prediction_array + [0,0,0,0,0,0,1,0]
  elif bowling_team == 'Sunrisers Hyderabad':
    prediction_array = prediction_array + [0,0,0,0,0,0,0,1]

  prediction_array = prediction_array + [runs, wickets, overs, runs_last_5, wi
  prediction_array = np.array([prediction_array])
  pred = model.predict(prediction_array)
  return int(round(pred[0]))
```

Test 1

- Batting Team : **Delhi Daredevils**
- Bowling Team : **Chennai Super Kings**
- Final Score : **147/9**

In [58]:
```
batting_team='Mumbai Indians'
bowling_team='Kings XI Punjab'
score = predict_score(batting_team, bowling_team, overs=12.3, runs=113, wicket
print(f'Predicted Score : {score} || Actual Score : 176')
```

```
Predicted Score : 188 || Actual Score : 176
```

Test 2 (2020 season)

- Batting Team : **Kings XI Punjab**
- Bowling Team : **Rajasthan Royals**
- Final Score : **185/4**

  These Test Was done before the match and final score were added later.

In [61]:
```
#  Test
batting_team="Kings XI Punjab"
bowling_team="Rajasthan Royals"
score = predict_score(batting_team, bowling_team, overs=14.0, runs=118, wicket
print(f'Predicted Score : {score} || Actual Score : 185')
```

```
Predicted Score : 185 || Actual Score : 185
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserW
arning: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 3 (2025 season)

- Batting Team : **Chennai Super Kings**
- Bowling Team : **Delhi Daredevils**
- Final Score : **183/6**

  These Test Was done before the match and final score were added later.

In [64]:
```
# Test
batting_team = 'Chennai Super Kings'
bowling_team = 'Delhi Daredevils'
score = predict_score(batting_team, bowling_team, overs=10.4, runs=90, wickets
print(f'Predicted Score : {score} || Actual Score : 183')
```

```
Predicted Score : 187 || Actual Score : 183
```

Test 4 (2025 season)

- Batting Team : **Royal Challengers Bangalore**
- Bowling Team : **Chennai Super Kings**
- Final Score : **213/5**

  These Test Was done before the match and final score were added later.

In [68]:
```
# Test
batting_team = 'Royal Challengers Bangalore'
bowling_team = 'Chennai Super Kings'
score = predict_score(batting_team, bowling_team, overs=11.5, runs=121, wicket
print(f'Predicted Score : {score} || Actual Score : 213')
```

```
Predicted Score : 198 || Actual Score : 213
```

Test 5 (2025 season)

- Batting Team : **Royal Challengers Bangalore**
- Bowling Team : **Kings XI Punjab**
- Final Score : **190/9**

  These Test Was done before the match and final score were added later.

In [70]:
```
# IPL 2025 Final – RCB vs Punjab Kings

batting_team = 'Royal Challengers Bangalore'
bowling_team = 'Kings XI Punjab'
score = predict_score(batting_team, bowling_team, overs=14.5, runs=131, wicket
print(f'Predicted Score : {score} || Actual Score : 190')
```

```
Predicted Score : 182 || Actual Score : 190
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserW
arning: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

In [71]:

```
dump(forest, "forest_model.pkl")
dump(tree, "tree_model.pkl")
dump(neural_net, "neural_nets_model.pkl")
```

Out[71]:

```
['neural_nets_model.pkl']
```

In [ ]: