IPL Score Prediction using Machine Learning

This Machine Learning model adapts a Regression Approach to predict the score of the First Inning of an IPL Match.

Team Members: Tejas Gawde, Raghav Tripathi

Import Necessary Libraries

In [1]:
```
# Importing Necessary Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

```

Load the dataset

In [2]:
```
#Importing dataset
ipl_df = pd.read_csv(r"C:\Machine Learning\IPL_Data.csv")
print(f"Dataset successfully Imported of Shape : {ipl_df.shape}")
```

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

Exploratory Data Analysis

In [3]:
```
# First 5 Columns Data
ipl_df.head()
```

Out[3]:

	mid	date	venue	bat_team	bowl_team	batsman	bowler	runs	wickets	overs
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

In [4]:
```
# Describing the ipl_dfset
ipl_df.describe()
```

Out[4]:

	mid	runs	wickets	overs	runs_last_5	wickets_last_5
count	76014.000000	76014.000000	76014.000000	76014.000000	76014.000000	76014.000000
mean	308.627740	74.889349	2.415844	9.783068	33.216434	1.120307
std	178.156878	48.823327	2.015207	5.772587	14.914174	1.053343
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	154.000000	34.000000	1.000000	4.600000	24.000000	0.000000
50%	308.000000	70.000000	2.000000	9.600000	34.000000	1.000000
75%	463.000000	111.000000	4.000000	14.600000	43.000000	2.000000
max	617.000000	263.000000	10.000000	19.600000	113.000000	7.000000

In [5]:
```
# Information about Each Column
ipl_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   bat_team        76014 non-null  object
 4   bowl_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 [6]:
```
# Number of Unique Values in each column
ipl_df.nunique()
```

Out[6]:
```
mid              617
date             442
venue             35
bat_team          14
bowl_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
```

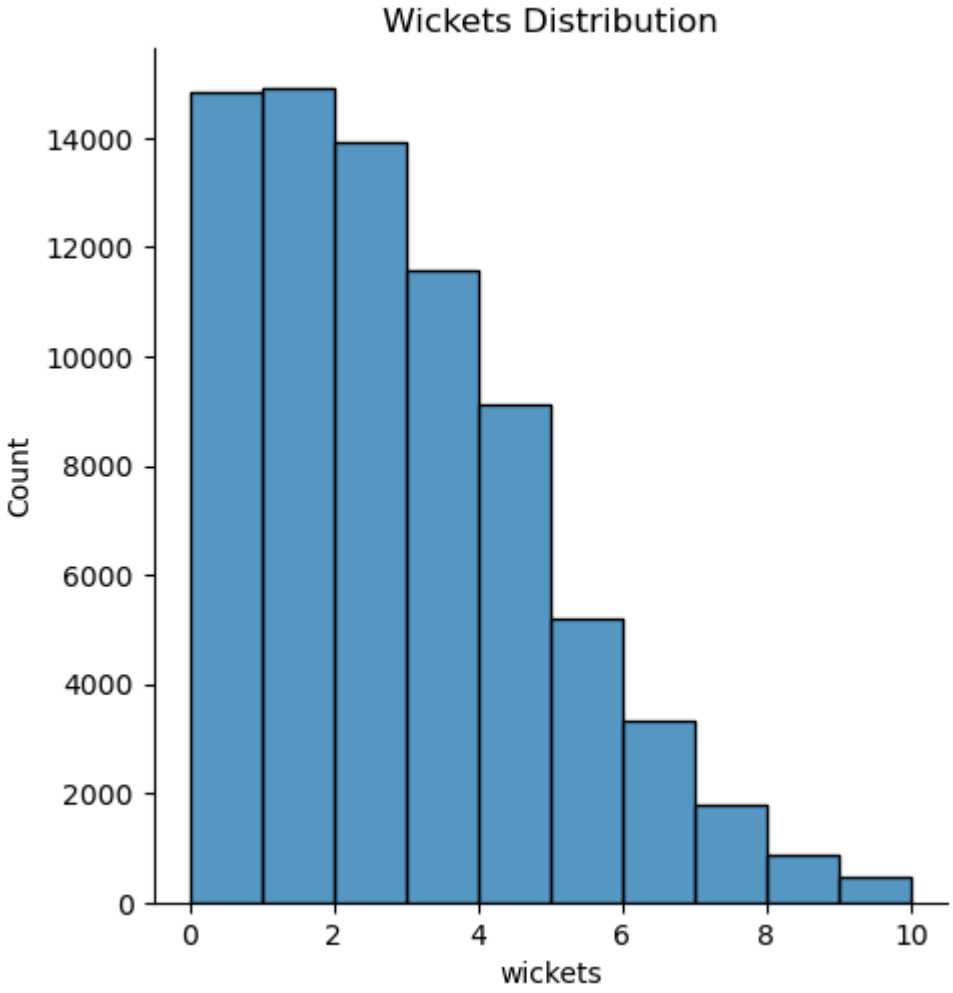
In [7]:
```
# ipl_df types of all Columns
ipl_df.dtypes
```

Out[7]:
```
mid                int64
date              object
venue             object
bat_team          object
bowl_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
```

In [8]:
```
#Wickets Distribution
sns.displot(ipl_df['wickets'],kde=False,bins=10)
plt.title("Wickets Distribution")

plt.show()
```

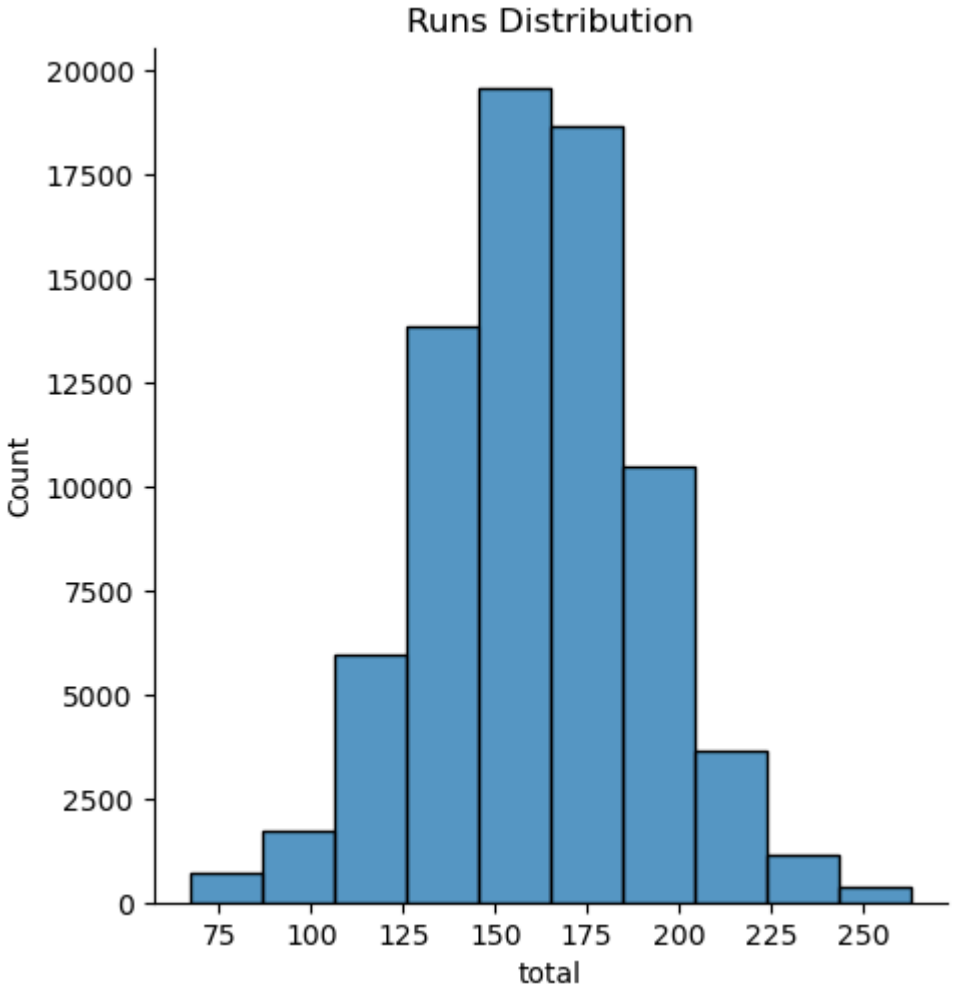
```
C:\Users\tejas\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [9]:
```
#Runs Distribution
sns.displot(ipl_df['total'],kde=False,bins=10)
plt.title("Runs Distribution")

plt.show()
```

```
C:\Users\tejas\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Data Cleaning

Removing Irrelevant Data colunms

In [10]:
```
# Names of all columns
ipl_df.columns
```

Out[10]:
```
Index(['mid', 'date', 'venue', 'bat_team', 'bowl_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 [11]:
```
no_use = ['mid', 'date', 'venue','batsman', 'bowler', 'striker', 'non-s
print(f'Before Removing Irrelevant Columns : {ipl_df.shape}')
ipl_df = ipl_df.drop(no_use, axis=1) # Drop Irrelevant Columns
print(f'After Removing Irrelevant Columns : {ipl_df.shape}')
ipl_df.head()
```

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

Out[11]:

	bat_team	bowl_team	runs	wickets	overs	runs_last_5	wickets_last_5	total
0	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.1	1	0	222
1	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.2	1	0	222
2	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.2	2	0	222
3	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.3	2	0	222
4	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.4	2	0	222

Keeping only Consistent Teams

In [12]:
```
# Define Consistent Teams
const_teams = ['Kolkata Knight Riders', 'Chennai Super Kings', 'Rajasth
               'Mumbai Indians', 'Kings XI Punjab', 'Royal Challengers B
               'Delhi Daredevils', 'Sunrisers Hyderabad']
```

In [13]:
```
print(f'Before Removing Inconsistent Teams : {ipl_df.shape}')
ipl_df = ipl_df[(ipl_df['bat_team'].isin(const_teams)) & (ipl_df['bowl_
print(f'After Removing Irrelevant Columns : {ipl_df.shape}')
print(f"Consistent Teams : \n{ipl_df['bat_team'].unique()}")
ipl_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[13]:

	bat_team	bowl_team	runs	wickets	overs	runs_last_5	wickets_last_5	total
0	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.1	1	0	222
1	Kolkata Knight Riders	Royal Challengers Bangalore	1	0	0.2	1	0	222
2	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.2	2	0	222
3	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.3	2	0	222
4	Kolkata Knight Riders	Royal Challengers Bangalore	2	0	0.4	2	0	222

Remove First 5 Overs of every match

In [14]:
```
print(f'Before Removing Overs : {ipl_df.shape}')
ipl_df = ipl_df[ipl_df['overs'] >= 5.0]
print(f'After Removing Overs : {ipl_df.shape}')
ipl_df.head()
```

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

Out[14]:

	bat_team	bowl_team	runs	wickets	overs	runs_last_5	wickets_last_5	total
32	Kolkata Knight Riders	Royal Challengers Bangalore	61	0	5.1	59	0	222
33	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.2	59	1	222
34	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.3	59	1	222
35	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.4	59	1	222
36	Kolkata Knight Riders	Royal Challengers Bangalore	61	1	5.5	58	1	222

Plotting a Correlation Matrix of current data

Data Preprocessing and Encoding

Performing Label Encoding

In [16]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
for col in ['bat_team', 'bowl_team']:
  ipl_df[col] = le.fit_transform(ipl_df[col])
ipl_df.head()
```

Out[16]:

	bat_team	bowl_team	runs	wickets	overs	runs_last_5	wickets_last_5	total
32	3	6	61	0	5.1	59	0	222
33	3	6	61	1	5.2	59	1	222
34	3	6	61	1	5.3	59	1	222
35	3	6	61	1	5.4	59	1	222
36	3	6	61	1	5.5	58	1	222

Performing One Hot Encoding and Column Transformation

In [17]:
```
from sklearn.compose import ColumnTransformer
columnTransformer = ColumnTransformer([('encoder',
                                        OneHotEncoder(),
                                        [0, 1])],
                                      remainder='passthrough')
```

In [18]:
```
ipl_df = np.array(columnTransformer.fit_transform(ipl_df))
```

Save the Numpy Array in a new DataFrame with transformed columns

In [19]:
```
cols = ['batting_team_Chennai Super Kings', 'batting_team_Delhi Daredev
              'batting_team_Kolkata Knight Riders', 'batting_team_Mumba
              'batting_team_Royal Challengers Bangalore', 'batting_team
              'bowling_team_Chennai Super Kings', 'bowling_team_Delhi D
              'bowling_team_Kolkata Knight Riders', 'bowling_team_Mumba
              'bowling_team_Royal Challengers Bangalore', 'bowling_team
       'runs_last_5', 'wickets_last_5', 'total']
df = pd.DataFrame(ipl_df, columns=cols)
```

In [20]:
```
# Encoded Data
df.head()
```

Out[20]:

	batting_team_Chennai Super Kings	batting_team_Delhi Daredevils	batting_team_Kings XI Punjab	batting_team_Kolkata Knight Riders	batti
0	0.0	0.0	0.0	1.0	
1	0.0	0.0	0.0	1.0	
2	0.0	0.0	0.0	1.0	
3	0.0	0.0	0.0	1.0	
4	0.0	0.0	0.0	1.0	

5 rows × 22 columns

Model Building

Prepare Train and Test Data

In [21]:
```
features = df.drop(['total'], axis=1)
labels = df['total']
```

In [22]:
```
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_s
print(f"Training Set : {train_features.shape}\nTesting Set : {test_feat
```

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

ML Algorithms

In [23]:
```
models = dict()
```

1. Decision Tree Regressor

In [24]:
```
from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor()
# Train Model
tree.fit(train_features, train_labels)
```

Out[24]:

▾ DecisionTreeRegressor

DecisionTreeRegressor()

In [25]:
```
# 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
models["tree"] = test_score_tree
```

```
Train Score : 99.98%
Test Score : 85.63%
```

In [26]:
```
from sklearn.metrics import mean_absolute_error as mae, mean_squared_er
print("---- Decision Tree Regressor - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, tree.pred
print("Mean Squared Error (MSE): {}".format(mse(test_labels, tree.predi
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- Decision Tree Regressor - Model Evaluation ----
Mean Absolute Error (MAE): 4.082398404387933
Mean Squared Error (MSE): 128.27511842433307
Root Mean Squared Error (RMSE): 11.325860604136583
```

Linear Regression

In [27]:
```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
# Train Model
linreg.fit(train_features, train_labels)
```

Out[27]:
```
LinearRegression()
```

In [28]:
```
# Evaluate Model
train_score_linreg = str(linreg.score(train_features, train_labels) * 1
test_score_linreg = str(linreg.score(test_features, test_labels) * 100)
print(f'Train Score : {train_score_linreg[:5]}%\nTest Score : {test_sco
models["linreg"] = test_score_linreg
```

```
Train Score : 65.87%
Test Score : 66.09%
```

In [29]:
```
print("---- Linear Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, linreg.pr
print("Mean Squared Error (MSE): {}".format(mse(test_labels, linreg.pre
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- Linear Regression - Model Evaluation ----
Mean Absolute Error (MAE): 12.944916773498107
Mean Squared Error (MSE): 302.78211420081846
Root Mean Squared Error (RMSE): 17.400635453937262
```

Random Forest Regression

In [30]:
```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor()
# Train Model
forest.fit(train_features, train_labels)
```

Out[30]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [31]:
```
# 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_scc
models["forest"] = test_score_forest
```

```
Train Score : 99.04%
Test Score : 93.39%
```

In [32]:
```
print("---- Random Forest Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, forest.pr
print("Mean Squared Error (MSE): {}".format(mse(test_labels, forest.pre
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- Random Forest Regression - Model Evaluation ----
Mean Absolute Error (MAE): 4.485131817561229
Mean Squared Error (MSE): 59.010869770473896
Root Mean Squared Error (RMSE): 7.681853277072785
```

Support Vector Machine

In [33]:
```
from sklearn.svm import SVR
svm = SVR()
# Train Model
svm.fit(train_features, train_labels)
```

Out[33]:
```
SVR
SVR()
```

In [34]:
```
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_
models["svm"] = test_score_svm
```

```
Train Score : 57.41%
Test Score : 56.97%
```

In [35]:
```
print("---- Support Vector Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, svm.predi
print("Mean Squared Error (MSE): {}".format(mse(test_labels, svm.predic
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- Support Vector Regression - Model Evaluation ----
Mean Absolute Error (MAE): 14.744224539744033
Mean Squared Error (MSE): 384.22338991308527
Root Mean Squared Error (RMSE): 19.601617022916383
```

XGBoost

In [36]:
```
from xgboost import XGBRegressor
xgb = XGBRegressor()
# Train Model
xgb.fit(train_features, train_labels)
```

Out[36]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds
=None,
             enable_categorical=False, eval_metric=None, feature_types
=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin
=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
```

In [37]:
```
train_score_xgb = str(xgb.score(train_features, train_labels)*100)
test_score_xgb = str(xgb.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_xgb[:5]}%\nTest Score : {test_score_
models["xgb"] = test_score_xgb
```

```
Train Score : 88.66%
Test Score : 85.29%
```

In [38]:
```
print("---- XGB Regression - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, xgb.predi
print("Mean Squared Error (MSE): {}".format(mse(test_labels, xgb.predic
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- XGB Regression - Model Evaluation ----
Mean Absolute Error (MAE): 8.263026630512885
Mean Squared Error (MSE): 131.301222251419
Root Mean Squared Error (RMSE): 11.458674541648305
```

KNN

In [39]:
```
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor()
# Train Model
knr.fit(train_features, train_labels)
```

Out[39]:
```
KNeighborsRegressor

KNeighborsRegressor()
```

In [40]:
```
train_score_knr = str(knr.score(train_features, train_labels)*100)
test_score_knr = str(knr.score(test_features, test_labels)*100)
print(f'Train Score : {train_score_knr[:5]}%\nTest Score : {test_score_
models["knr"] = test_score_knr
```

```
Train Score : 86.84%
Test Score : 77.73%
```

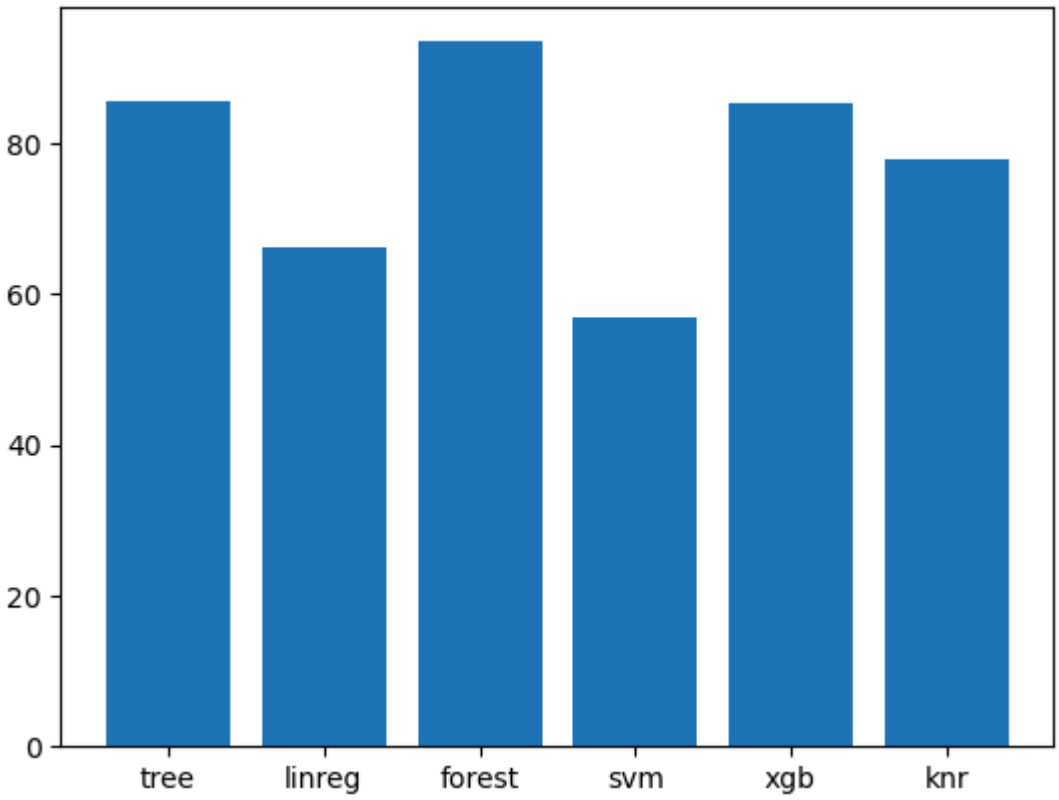
In [41]:
```
print("---- KNR - Model Evaluation ----")
print("Mean Absolute Error (MAE): {}".format(mae(test_labels, knr.predi
print("Mean Squared Error (MSE): {}".format(mse(test_labels, knr.predic
print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test_labe
```

```
---- KNR - Model Evaluation ----
Mean Absolute Error (MAE): 9.705011219147345
Mean Squared Error (MSE): 198.84178509099976
Root Mean Squared Error (RMSE): 14.10112708583962
```

Best Model

In [42]:
```
model_names = list(models.keys())
accuracy = list(map(float, models.values()))
# creating the bar plot
plt.bar(model_names, accuracy)
```

Out[42]: <BarContainer object of 6 artists>



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

Predictions

In [43]:

```
def score_predict(batting_team, bowling_team, runs, wickets, overs, run
  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_las
  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 [44]:
```
batting_team='Delhi Daredevils'
bowling_team='Chennai Super Kings'
score = score_predict(batting_team, bowling_team, overs=10.2, runs=68,
print(f'Predicted Score : {score} || Actual Score : 147')
```

```
Predicted Score : 146 || Actual Score : 147

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 2

- Batting Team : **Mumbai Indians**
- Bowling Team : **Kings XI Punjab**
- Final Score : **176/7**

In [45]:
```
batting_team='Mumbai Indians'
bowling_team='Kings XI Punjab'
score = score_predict(batting_team, bowling_team, overs=12.3, runs=113,
print(f'Predicted Score : {score} || Actual Score : 176')
```

```
Predicted Score : 183 || Actual Score : 176

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 3

- 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 [46]:
```
batting_team="Kings XI Punjab"
bowling_team="Rajasthan Royals"
score =score_predict(batting_team, bowling_team, overs=14.0, runs=118,
print(f'Predicted Score : {score} || Actual Score : 185')
```

```
Predicted Score : 187 || Actual Score : 185

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 4

- Batting Team : **Kolkata Knight Riders**
- Bowling Team : **Chennai Super Kings**
- Final Score : **172/5**

In [47]:
```
batting_team="Kolkata Knight Riders"
bowling_team="Chennai Super Kings"
score = score_predict(batting_team, bowling_team, overs=18.0, runs=150,
print(f'Predicted Score : {score} || Actual Score : 172')
```

```
Predicted Score : 172 || Actual Score : 172

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 5

- Batting Team : **Delhi Daredevils**
- Bowling Team : **Mumbai Indians**
- Final Score : **110/7**

In [48]:
```
batting_team='Delhi Daredevils'
bowling_team='Mumbai Indians'
score = score_predict(batting_team, bowling_team, overs=18.0, runs=96,
print(f'Predicted Score : {score} || Actual Score : 110')
```

```
Predicted Score : 108 || Actual Score : 110

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 6

- Batting Team : **Kings XI Punjab**
- Bowling Team : **Chennai Super Kings**
- Final Score : **153/9**

In [49]:
```
batting_team='Kings XI Punjab'
bowling_team='Chennai Super Kings'
score = score_predict(batting_team, bowling_team, overs=18.0, runs=129,
print(f'Predicted Score : {score} || Actual Score : 153')
```

```
Predicted Score : 147 || Actual Score : 153

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Test 7

- Batting Team : **Sunrisers Hyderabad**
- Bowling Team : **Royal Challengers Banglore**
- Final Score : **146/10**

In [50]:
```
batting_team='Sunrisers Hyderabad'
bowling_team='Royal Challengers Bangalore'
score = score_predict(batting_team, bowling_team, overs=10.5, runs=67,
print(f'Predicted Score : {score} || Actual Score : 146')
```

```
Predicted Score : 155 || Actual Score : 146

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

RCB VS CSK

In [51]:
```
batting_team='Royal Challengers Bangalore'
bowling_team='Chennai Super Kings'
score = score_predict(batting_team, bowling_team, overs=19.0, runs=164,
print(f'Predicted Score : {score} || Actual Score : 173 ')
```

```
Predicted Score : 180 || Actual Score : 173

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

MI VS RR

In [52]:
```
batting_team='Mumbai Indians'
bowling_team='Rajasthan Royals'
score = score_predict(batting_team, bowling_team, overs=14.0, runs=97,
print(f'Predicted Score : {score} || Actual Score : 125 ')
```

```
Predicted Score : 133 || Actual Score : 125

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```

Type *Markdown* and LaTeX: α^2

Finale SRH VS KKR

In [53]:
```
batting_team='Sunrisers Hyderabad'
bowling_team='Kolkata Knight Riders'
score = score_predict(batting_team, bowling_team, overs=12.0, runs=72,
print(f'Predicted Score : {score} || Actual Score : 114 ')
```

```
Predicted Score : 115 || Actual Score : 114

C:\Users\tejas\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
  warnings.warn(
```