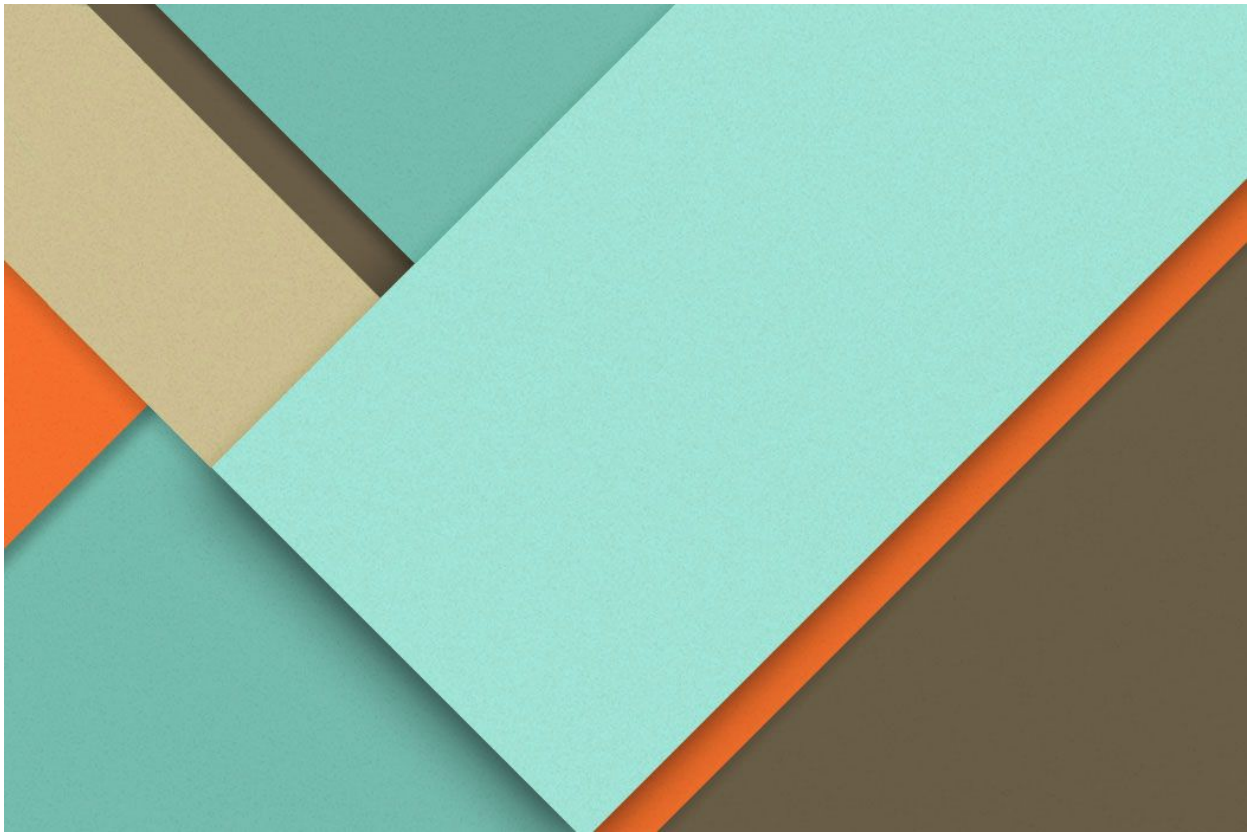


# IPL Match Prediction & Simulation

02.12.2018

Abhishek Sinha - 01FB16ECS014
Aditya Pandey - 01FB16ECS029
Aditya Venkatesh - 01FB16ECS032
Akash Bhat - 01FB16ECS038
Evaluator - Prof. Mamatha Shetty

# Overview

The aim of this project is to develop a reliable model to predict and simulate the result of an IPL match using appropriate Big Data technologies.

# Phase - I

Phase 1 of the project was to cluster players based on their stats which was implemented using K-means Clustering.

## Procedure:

Player stats was scraped from [www.espncricinfo.com](http://www.espncricinfo.com) using BeautifulSoup, which is a python library used for web scraping:

- The data scraped from the website included stats of all the players who have played t20 cricket.
- The stats included Player Name,Matches,Batting Innings,Not Outs,Runs Scored,Highest Score, Average Score, Balls Faced, Strike Rate, Hundreds, Fifties, Fours, Sixes, Bowling Innings, Balls Bowled, Runs Conceded, Wickets, Bowling Average, Economy, Bowling Strike Rate.
- Screenshot of the data collected:

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Player | Mat | Inns | NO | Runs | HS | Ave | BF | SR | 100 | 50 |
| 2 | VR Aaron | 10 | 3 | 3 | 35 | 17 | 0 | 34 | 102.94 | 0 | 0 |
| 3 | MA Agarwal | 8 | 8 | 0 | 115 | 31 | 14.37 | 91 | 126.37 | 0 | 0 |
| 4 | AN Ahmed | 4 | 1 | 1 | 4 | 4 | 0 | 1 | 400 | 0 | 0 |
| 5 | CJ Anderson | 12 | 11 | 2 | 265 | 95 | 29.44 | 181 | 146.4 | 0 | 1 |
| 6 | S Anirudha | 1 | 1 | 0 | 3 | 3 | 3 | 5 | 60 | 0 | 0 |

## K-Means clustering was done using pyspark which is the python library for spark.

K-Means Clustering: It is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

- The centroids of the K clusters, which can be used to label new data.
- Labels for the training data (each data point is assigned to a single cluster).

## The clustering criteria:

- For Batsman: Strike Rate, Runs, High Score, Average Runs Scored, Balls Faced.
- For Bowlers:  Economy, Bowling Average, Bowling Strike Rate, Wickets.

## Screenshot of Clustered data (in a csv):

- Batsman

| | Player | Features | Cluster |
|---|---|---|---|
| 0 | A Ashish Reddy | [3.207646131515503, 32.0, 23.915000915527344, ... | 0 |
| 1 | A Choudhary | [1.25, 15.0, 25.0, 20.0, 125.0] | 0 |
| 2 | A Dananjaya | [0.8202247023582458, 28.0, 7.300000190734863, ... | 0 |
| 3 | A Mishra | [2.4661171436309814, 13.0, 14.75, 85.0, 61.650... | 0 |
| 4 | A Nehra | [1.1428571939468384, 1.0, 0.25, 8.0, 28.569999... | 0 |

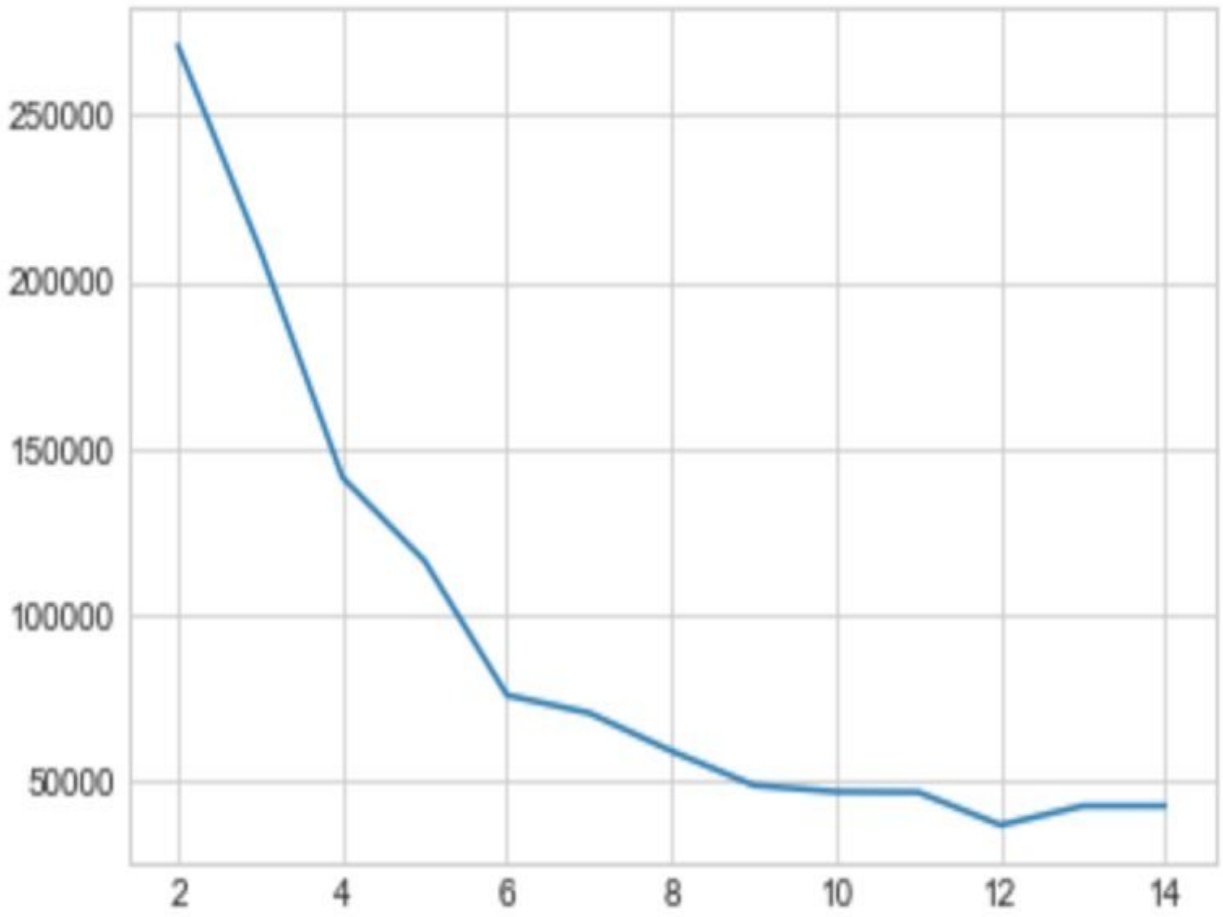
- Bowlers

| | Player | Features | Cluster |
|---|---|---|---|
| 0 | A Ashish Reddy | [17.5, 32.0, 1.25, 9.25, 6.0] | 4 |
| 1 | A Choudhary | [20.200000762939453, 28.799999237060547, 1.0, ... | 4 |
| 2 | A Dananjaya | [19.399999618530273, 25.72549057006836, 1.0980... | 4 |
| 3 | A Mishra | [25.825000762939453, 108.9047622680664, 3.0928... | 2 |
| 4 | A Nehra | [15.300000190734863, 121.27083587646484, 5.833... | 2 |

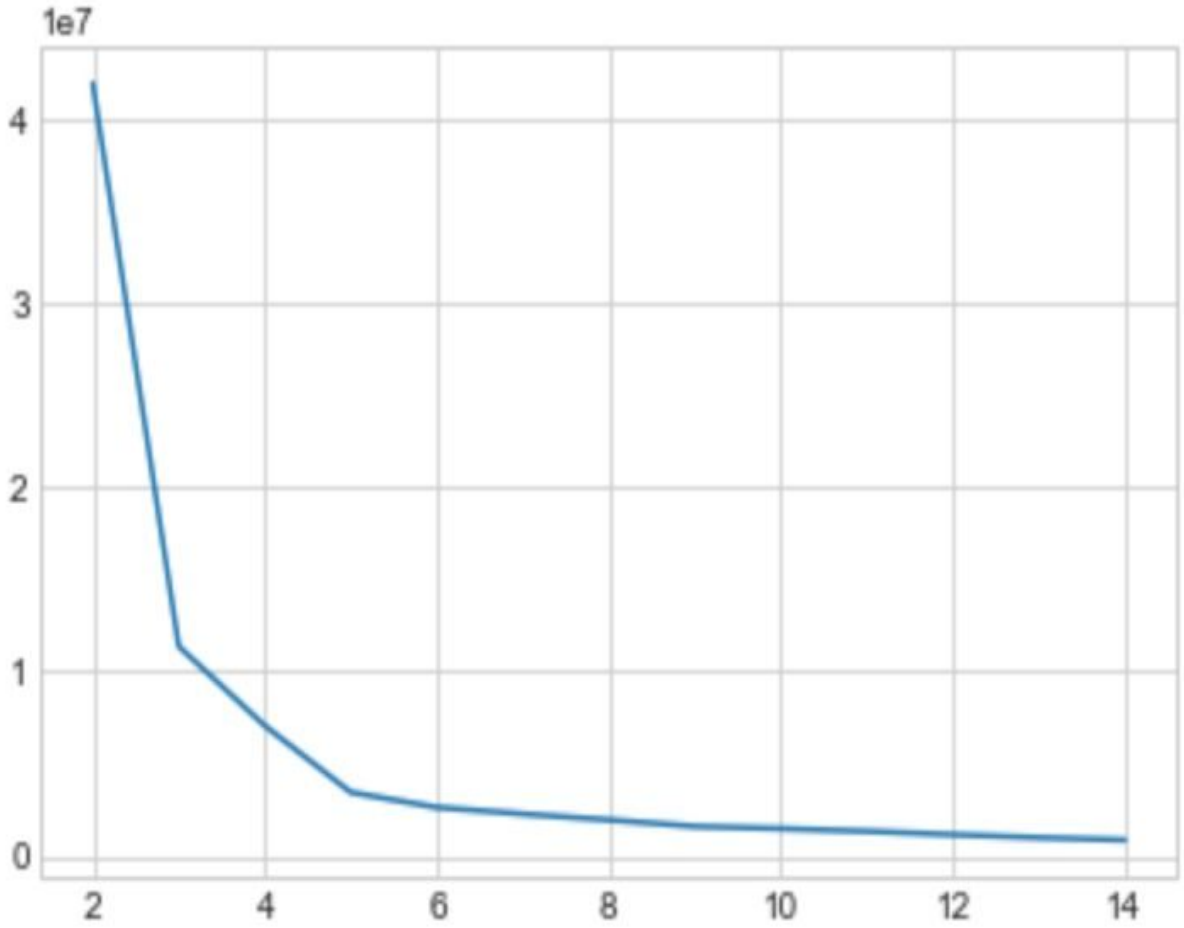
## Calculating optimal k-value

Optimal K value was calculated using the Elbow Algorithm. Below is the screenshot of the graph generated by the elbow algorithm:

- Batsman



- Bowler:



## Code Snippets:

Clustering:

```
vec = VectorAssembler(inputCols=features, outputCol="features")
df_clus = vec.transform(data).select('Player','features')
df_clus.show()


# In[42]:


vec_ball = VectorAssembler(inputCols=features_ball, outputCol="features_ball")
df_clus_ball = vec_ball.transform(dataBowl).select('Player','features_ball')
df_clus_ball.show()


# In[24]:


error = np.zeros(15)
for k in range(2,15):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
    model = kmeans.fit(df_clus.sample(False,0.25, seed=1))
    error[k] = model.computeCost(df_clus)


# In[25]:


errorBowl = np.zeros(15)
for k in range(2,15):
    kmeans_ball = KMeans().setK(k).setSeed(1).setFeaturesCol("features_ball")
    model_ball = kmeans_ball.fit(df_clus_ball.sample(False,0.25, seed=1))
    errorBowl[k] = model_ball.computeCost(df_clus_ball)
```

```
k = 5
kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
model = kmeans.fit(df_clus)
centers = model.clusterCenters()

print("Centers: ")
for i in range(len(centers)):
    print(i+1,": ",centers[i])
```

## Challenges Faced

- Installing and configuring pyspark.
- Understanding and implementing k-means clustering.

# PHASE 2

## Introduction

Implementation of phase 2 includes simulation of an entire IPL match using ball by ball data and clustered players from previous phase. Challenges to overcome were to map pairs whose ball to ball data were not available. Also to run an entire match with test cases such as, changing batsman on strike every time there's a single or a three, changing batsman every 6 balls and generating a new bowler every 6 balls and so on.

## Implementation

Wrote a function for player v/s player probability. If pair of players haven't played before then we generate probability using cluster probability.

```
def thresholdfilter(threshold):
    df_sub = new_df[new_df['balls']>threshold]
    df_sub['0'] = (df_sub['0']+1)/(df_sub['balls']+6)
    df_sub['1'] = (df_sub['1']+1)/(df_sub['balls']+6)
    df_sub['2'] = (df_sub['2']+1)/(df_sub['balls']+6)
    df_sub['3'] = (df_sub['3']+1)/(df_sub['balls']+6)
    df_sub['4'] = (df_sub['4']+1)/(df_sub['balls']+6)
    df_sub['6'] = (df_sub['6']+1)/(df_sub['balls']+6)
    df_sub['wickets'] = 1-((df_sub['wickets']+1)/(df_sub['balls']+1))
    l = [df_sub['0'], df_sub['1'], df_sub['2'], df_sub['3'], df_sub['4'], df_sub['6'], df_sub['wickets']]
    for i in range(len(l)-2):
        l[i+1] = l[i]+l[i+1]
        df_sub[df_sub.columns[i+3]]=l[i+1]
    return df_sub

filtered=thresholdfilter(9)
print(filtered)


def pvpprob(batsman,bowler,random):
    df_sub = filtered[(filtered['batsman']==batsman) & (filtered['bowler']==bowler)]
    if len(df_sub)>0:
        #print(df_sub)
        l = [df_sub['0'], df_sub['1'], df_sub['2'], df_sub['3'], df_sub['4'], df_sub['6']]
        res = [0,1,2,3,4,6]
        index=0
        for i in range(len(l)):
            if (l[i].iloc[0]<random):
                index=i+1
        return [int(res[index]),df_sub['wickets'].iloc[0]]
    else:
        return clusterprob(batsman,bowler,random)
```

Our function to calculate probability of two players who haven't played against each other before, includes calculating average probabilities of each cluster and then using that to calculate what that pair of players would generate.

```
def clusterprob(batsman,bowler,random):
    if((len(bat[bat['Player']==batsman])>0) and (len(ball[ball['Player']==bowler])>0)):

        batClus = bat[bat['Player']==batsman].iloc[0]['Cluster']
        ballClus = ball[ball['Player']==bowler].iloc[0]['Cluster']

        batsmenInClus = bat[bat['Cluster']==batClus].loc[:]["Player"]
        bowlersInClus = ball[ball['Cluster']==ballClus].loc[:]["Player"]

        df=[]
        wicketProbs=[]
        for aBatsman in batsmenInClus:
            df_sub = filtered[(filtered['batsman']==aBatsman) & (filtered['bowler']==bowler)]
            if len(df_sub)>0:
                l = [df_sub['0'].iloc[0], df_sub['1'].iloc[0], df_sub['2'].iloc[0], df_sub['3'].iloc[0], df_sub['4'].iloc[0], df_sub['6'].iloc[0]]
                df.append(l)
                wicketProbs.append(df_sub['wickets'].iloc[0])
        for aBowler in bowlersInClus:
            df_sub = filtered[(filtered['batsman']==batsman) & (filtered['bowler']==aBowler)]
            if len(df_sub)>0:
                l = [df_sub['0'].iloc[0], df_sub['1'].iloc[0], df_sub['2'].iloc[0], df_sub['3'].iloc[0], df_sub['4'].iloc[0], df_sub['6'].iloc[0]]
                df.append(l)
                wicketProbs.append(df_sub['wickets'].iloc[0])
        if(df==[]):
            #print("hello",batsman,bowler)
            df=df_fallprobs
            wicketProbs=df_wicketprob
        df=pd.DataFrame(df)
        runList=['0','1','2','3','4','6']
        df.columns=runList
        meanProb=df.mean()
        wicketProb=np.mean(wicketProbs)
        index=0
        for i in range(len(meanProb)):
            if (meanProb[i]<random):
                index=i+1
        return [int(runList[index%6]),wicketProb]
    else:
        df=df_fallprobs
        wicketProbs=df_wicketprob
        df=pd.DataFrame(df)
        runList=['0','1','2','3','4','6']
        df.columns=runList
        meanProb=df.mean()
        wicketProb=np.mean(wicketProbs)
        index=0
        for i in range(len(meanProb)):
            if (meanProb[i]<random):
                index=i+1
        return [int(runList[index]),wicketProb]
```

## Simulating the match using player v/s player probability and cluster probability

Each innings is simulated for 120 balls. Batsmen are interchanged and bowlers too every 6 balls. Bowler is selected at random every time.

Cumulative probability is found for all the indices and a random number “x” is generated and a prediction is made based on value of x and the cumulative probability values.

Based on the value, the runs or wickets are computed and final score is calculated.

```
def matchSimulation(t1bat,t1bowl,t2bat,t2bowl):
    innings=2
    overs=20
    ballsPerOver=6
    runs=[0,0]
    wickets=[0,0]
    batting=[t1bat,t2bat]
    bowling=[t2bowl,t1bowl]
    bbbrec=[]
    p=0
    batsmenPlaying={'strike':{'name':batting[0][0], 'wicketProb': 1}, 'nonStrike':{'name':batting[0][1],'wicketProb': 1}}
    print(batsmenPlaying)
    for i in range(innings):
        for j in range(overs):
            for k in range(ballsPerOver):
                #print(j, " : ",p)
                p+=1
                randomvar = random()
                #bias = 0.15
                #if j<6 or j>15:
                #    randomvar=(randomvar+bias)/(1+bias)
                pvpRes=pvpprob(batsmenPlaying['strike']['name'],bowling[i][j%len(bowling[i])],randomvar)
                batsmenPlaying['strike']['wicketProb']*=pvpRes[1]
                if(batsmenPlaying['strike']['wicketProb']>0.4):
                    runs[i]+=pvpRes[0]
                    bbbrec.append(pvpRes[0])
                    if(pvpRes[0]%2==1):
                        temp=batsmenPlaying['nonStrike']
                        batsmenPlaying['nonStrike']=batsmenPlaying['strike']
                        batsmenPlaying['strike']=temp
                else:
                    wickets[i]+=1
                    bbbrec.append(deepcopy(batsmenPlaying['strike']))
                    if(wickets[i]>=10):
                        break;
                    batsmenPlaying['strike']['name']=batting[i][wickets[i]+1]
                    batsmenPlaying['strike']['wicketProb']=1

            else:
                temp=batsmenPlaying['nonStrike']
                batsmenPlaying['nonStrike']=batsmenPlaying['strike']
                batsmenPlaying['strike']=temp
                #print(bbbrec)
                continue
            break
    print(bbbrec)
```

Each innings is simulated for 120 balls. Batsmen are interchanged and bowlers too every 6 balls. Bowler is selected at random every time.

Cumulative probability is found for all the indices and a random number “x” is generated and a prediction is made based on value of x and the cumulative probability values.

## Based on the value, the runs or wickets are computed and final score is calculated.

## Test case matches

```
[0, 0, 3, 2, 4, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 2, 'LMP Simmons', 1, 2, 0, 0, 0, 6, 1, 1, 0, 4, 'PA Patel', 4,
2, 6, 1, 1, 3, 4, 1, 0, 1, 4, 0, 0, 1, 1, 1, 4, 1, 2, 2, 1, 0, 1, 'AT Rayudu', 1, 0, 1, 3, 6, 0, 2, 4, 6, 1, 0, 0, 1,
'RG Sharma', 1, 'KH Pandya', 0, 1, 1, 0, 0, 0, 1, 3, 0, 0, 1, 1, 0, 1, 0, 0, 3, 2, 'KA Pollard', 0, 6, 0, 1, 1, 'HH P
andya', 6, 0, 2, 1, 2, 3, 1, 2, 0, 'KV Sharma', 0, 6, 0, 3, 0, 4, 1, 4, 2, 0, 4, 3, 3, 2, 1, 'JJ Bumrah', 1, 1, 6, 2,
'MG Johnson', 0, 1, 0, 1, 0, 1, 0, 4, 1, 1, 0, 1, 1, 0, 0, 1, 1, 3, 4, 0, 0, 3, 1, 6, 4, 6, 'MS Dhoni', 0, 2, 'SPD Sm
ith', 0, 0, 0, 1, 4, 1, 1, 3, 4, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 2, 1, 0, 0, 'MK Tiwary', 1, 'DT Christian', 1, 1, 1,
6, 1, 1, 0, 6, 4, 1, 3, 1, 6, 'LH Ferguson', 0, 1, 0, 1, 3, 0, 0, 3, 1, 'Washington Sundar', 0, 2, 0, 1, 0, 6, 3, 'A
Zampa', 2, 0, 0, 'SN Thakur']
Number of balls:  219
Team 1: 170/8   Team 2: 132/10

'Team 1'
```

## Accuracy

The matches which we simulated gave reasonable and realistic results to some extent. Matches with extreme scores(of scores above and around 200) were not too accurately predicted by our model. A match with an average score around ~140-180 put up for prediction stood well. Also, in most cases data doesn’t always tell you the whole story, there are many other factors present which influence the outcome.

## Learning outcomes

1. Clustering usage, cumulative probability
2. Handling large data efficiently
3. Handling many test cases, which need to be tweaked accordingly

# PHASE 3

## Introduction

Phase - III of the project was to simulate an IPL match using the Decision Tree Classifier. The  Decision Tree Classifier was used to train on the data(stats) collected in Phase-1 and predict the score. The results were compared to the results obtained in Phase-2 and inferences were made on their accuracies respectively.

## Accuracy

Decision tree model was fairly accurate for games which had a result which were close. Realistic RR and wickets could be seen in decision tree when compared to the probability model. Decision tree predictions were not random although sometimes generated very high or low scores.

## Compare with previous approach

The inferences we made were that we preferred the probability model since it generated ball by ball random data and not fixed data based on set parameters. There needs to be some amount of randomness during prediction. Even though both the models faired well, the probability model was the more flexible one, taking care of all outliers and cases. Decision tree model was tougher to implement logically when compared to the probability model which took more time.

## Snapshots

## Result:

```
['LMP Simmons', 'PA Patel', 'JD Unadkat', 4, 0]
['PA Patel', 'LMP Simmons', 'Washington Sundar', 10, 0]
['LMP Simmons', 'PA Patel', 'JD Unadkat', 9, 0]
['PA Patel', 'LMP Simmons', 'Washington Sundar', 8, 0]
['LMP Simmons', 'PA Patel', 'JD Unadkat', 8, 0]
['PA Patel', 'LMP Simmons', 'DT Christian', 8, 0]
['LMP Simmons', 'PA Patel', 'SN Thakur', 7, 0]
['PA Patel', 'LMP Simmons', 'DT Christian', 8, 0]
['LMP Simmons', 'PA Patel', 'A Zampa', 7, 0]
['PA Patel', 'LMP Simmons', 'LH Ferguson', 8, 0]
['LMP Simmons', 'PA Patel', 'A Zampa', 8, 0]
['PA Patel', 'LMP Simmons', 'LH Ferguson', 8, 0]
['LMP Simmons', 'PA Patel', 'A Zampa', 7, 0]
['PA Patel', 'LMP Simmons', 'Washington Sundar', 7, 1]
['LMP Simmons', 'AT Rayudu', 'A Zampa', 31, 1]
['AT Rayudu', 'RG Sharma', 'Washington Sundar', 5, 1]
['RG Sharma', 'KH Pandya', 'DT Christian', 14, 1]
['KH Pandya', 'KA Pollard', 'SN Thakur', 9, 0]
['KA Pollard', 'KH Pandya', 'JD Unadkat', 16, 0]
['KH Pandya', 'KA Pollard', 'DT Christian', 13, 1]


['AM Rahane', 'RA Tripathi', 'KH Pandya', 3, 0]
['RA Tripathi', 'AM Rahane', 'MG Johnson', 9, 0]
['AM Rahane', 'RA Tripathi', 'JJ Bumrah', 10, 0]
['RA Tripathi', 'AM Rahane', 'SL Malinga', 7, 0]
['AM Rahane', 'RA Tripathi', 'KV Sharma', 8, 0]
['RA Tripathi', 'AM Rahane', 'KH Pandya', 7, 0]
['AM Rahane', 'RA Tripathi', 'MG Johnson', 8, 0]
['RA Tripathi', 'AM Rahane', 'JJ Bumrah', 8, 0]
['AM Rahane', 'RA Tripathi', 'SL Malinga', 7, 0]
['RA Tripathi', 'AM Rahane', 'KV Sharma', 8, 0]
['AM Rahane', 'RA Tripathi', 'KH Pandya', 8, 0]
['RA Tripathi', 'AM Rahane', 'MG Johnson', 8, 0]
['AM Rahane', 'RA Tripathi', 'JJ Bumrah', 8, 0]
['RA Tripathi', 'AM Rahane', 'SL Malinga', 8, 0]
['AM Rahane', 'RA Tripathi', 'KV Sharma', 24, 1]
['RA Tripathi', 'SPD Smith', 'KH Pandya', 8, 0]
['SPD Smith', 'RA Tripathi', 'MG Johnson', 14, 1]
['RA Tripathi', 'MS Dhoni', 'JJ Bumrah', 9, 0]
['MS Dhoni', 'RA Tripathi', 'SL Malinga', 7, 0]
['RA Tripathi', 'MS Dhoni', 'KV Sharma', 13, 1]

1st Innings:  184 / 5

2nd Innings:  174 / 3
```

## Implementing Decision Tree Model

```
def train_create(vec):
    res = vec[1]

    feat = []
    feat.append(vec[0])
    feat = feat + list(vec[2:])
    feat = flatten(feat)
    return LabeledPoint(res,feat)

wicketData = Over_W.map(train_create)
runsData = Over_R.map(train_create)

print(wicketData)


# In[6]:


def getBatsmanStat(name):
    player = Bat_Clus[Bat_Clus['Player Name']==name]
    if(player.empty):
        l = np.array(Bat_Clus.mean())
    else:
        player.drop('Player Name', axis=1, inplace=True)
        l = np.array(player.values[0])
    return l

def getBowlerStat(name):
    player = Bowl_Clus[Bowl_Clus['Player Name']==name]
    if(player.empty):
        l = np.array(Bowl_Clus.mean())
    else:
        player.drop('Player Name', axis=1, inplace=True)
        l = np.array(player.values[0])
    return l

#getBowlerStat('DJ Bravo')
getBatsmanStat('DJ Bravo')


# In[7]:


wicketModel = DecisionTree.trainRegressor(wicketData, impurity='variance',categoricalFeaturesInfo={}, maxDepth=30, maxBins=40)
print(wicketModel.toDebugString())


# In[ ]:


runsModel = DecisionTree.trainRegressor(runsData, impurity='variance',categoricalFeaturesInfo={}, maxDepth=30, maxBins=40)
print(runsModel.toDebugString())
```

## Match Simulation

```
import random
random.seed(20)

def matchSimulation(inn1Ba,inn1Bo,inn2Ba,inn2Bo):
    batting=[inn1Ba,inn2Ba]
    bowling=[inn2Bo,inn1Bo]
    match=[[],[]]
    runPerInn=[0,0]
    wicketCount=[0,0]
    strike={}
    offStrike={}
    bowler={}
    for inn in range(2):
        strike['name']=batting[inn][0]
        offStrike['name']=batting[inn][1]

        for over in range(20):
            bowler['name']=bowling[inn][over%len(bowling[inn])]
            strike['data']= getBatsmanStat(strike['name'])
            offStrike['data']=getBatsmanStat(offStrike['name'])
            bowler['data']=getBowlerStat(bowler['name'])
            overData=[over]
            overData.extend(strike['data'])
            overData.extend(offStrike['data'])
            overData.extend(bowler['data'])

            runs=round(runsModel.predict(overData)*(random.uniform(1,1.2)))
            wickets=round(wicketModel.predict(overData)*(random.uniform(1,1.25)))
            match[inn].append([strike['name'],offStrike['name'],bowler['name'],runs,wickets])
            if(wickets):
                runs=round((runs/6)*(6-wickets))
                if((wicketCount[inn]+wickets)>=10):
                    wicketCount[inn]=10
                    runPerInn[inn]+=runs
                    match[inn].append([strike['name'],offStrike['name'],runs,'all out'])
                    break;
                wicketCount[inn]+=wickets
                strike['name']=batting[inn][wicketCount[inn]+1]

            runPerInn[inn]+=runs
            if(runPerInn[1]>runPerInn[0]):
                break
            strike,offStrike= offStrike,strike
    for i in match[0]:
        print(i)
    print('\n\n')
    for i in match[1]:
        print(i)
    print("\n1st Innings: ",runPerInn[0],"/",wicketCount[0])
    print("\n2nd Innings: ",runPerInn[1],"/",wicketCount[1])
```

## Learning Outcomes

1. Clustering of data, factors to be used, number of clusters to consider.
2. Inter conversion of spark data types and python data types.
3. How to build a k means clustering model
4. How to predict an outcome using it's cumulative probability.
5. Data pre-processing for large files.
6. Building an RDD from scratch using lists to pandas to RDD.
7. Understanding what a decision tree does and its implementation , using classification as well as regression!
8. Comparison between probability and decision tree models, two approaches for same problem.
9. Lastly installation of all the softwares which we used to implement this project.