SIT742 ASSESSMENT TASK 2

SIT742T2E GROUP 55

Rajeshkumar Mourya

Section 1: Data Manipulation

1.1.A: Age (min, mean and max)

```
+------------------+--------+--------+
|          avg(Age)|min(Age)|max(Age)|
+------------------+--------+--------+
|25.122205745043114|      16|      45|
+------------------+--------+--------+
```

1.1.A: Overall (min, mean and max)

```
+-----------------+------------+------------+
|     avg(Overall)|min(Overall)|max(Overall)|
+-----------------+------------+------------+
|66.23869940132916|          46|          94|
+-----------------+------------+------------+
```

1.1.A: Position of Talents

```
+--------+-----------------+
|Position|     avg(Overall)|
+--------+-----------------+
|      LF|73.86666666666666|
+--------+-----------------+
```

1.1.A: Top 3 Countries with Talents

```
+--------------------+-----------------+
|         Nationality|     avg(Overall)|
+--------------------+-----------------+
|United Arab Emirates|             77.0|
|Central African Rep.|73.33333333333333|
|              Israel|72.14285714285714|
+--------------------+-----------------+
```

1.1.B: Average Potentials on Country by Position with Ordering

Result was too large to be copied, please find below snapshot and full result can be viewed in ipynb file submitted separately, here is the snapshot

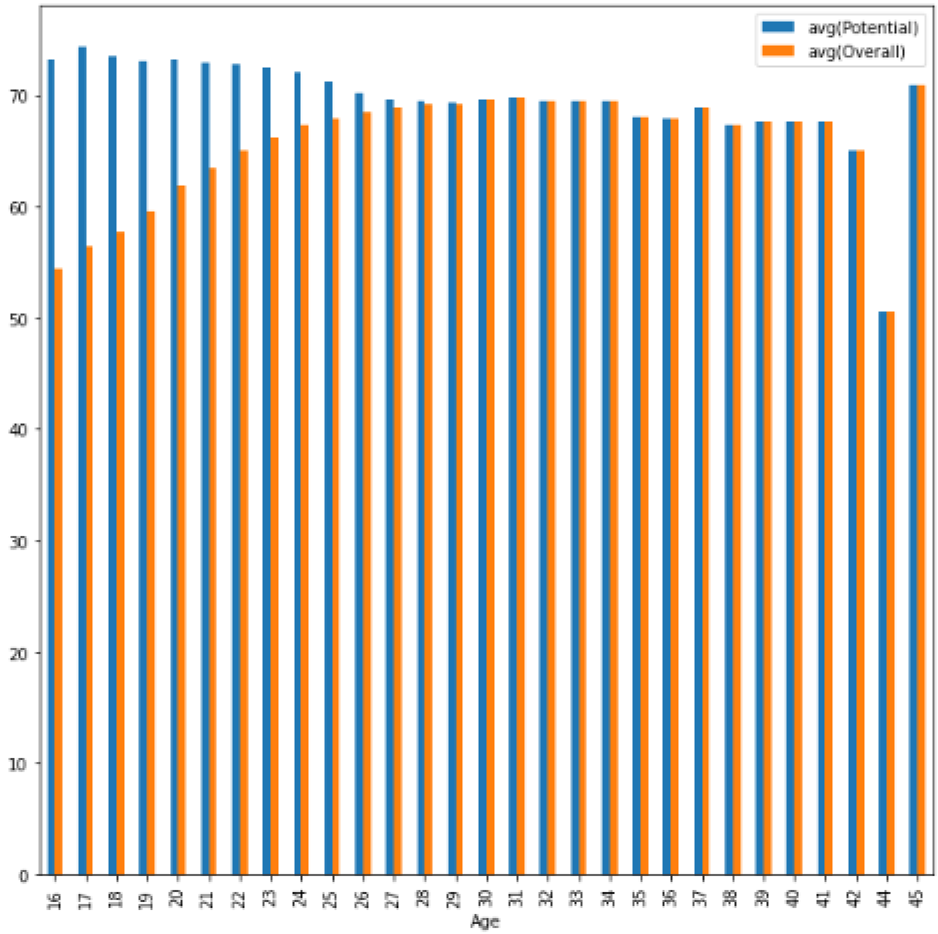
```
+-----------------+----+-----------------+-----------------+-----------------+----+-----------------+-----------------+
|      Nationality|null|              CAM|               CB|              CDM|  CF|               CM|               GK|
+-----------------+----+-----------------+-----------------+-----------------+----+-----------------+-----------------+
|      Afghanistan|null|             66.0|             null|             null|null|             71.0|             null|
|          Albania|70.0|            70.75|74.33333333333333|             69.5|null|            71.75|             77.5|
|          Algeria|null|            74.25|             69.5|            70.25|null|77.66666666666667|             67.6|
|          Andorra|null|             null|             64.0|             null|null|             null|             null|
|           Angola|null|             null|             79.0|             null|null|             null|             null|
|Antigua & Barbuda|null|             null|             null|             null|null|             null|             null|
|        Argentina|70.0|74.78260869565217|72.85858585858585|72.76363636363637|79.0|73.08695652173913|71.76288659793815|
|          Armenia|null|             null|             null|             null|null|             null|             null|
|        Australia|null|            71.25| 69.0952380952381|          66.8125|null|67.71428571428571|             67.0|
|          Austria|64.0|73.16666666666667|         70.28125|69.63157894736842|68.0|68.70588235294117| 68.1590909090909|
+-----------------+----+-----------------+-----------------+-----------------+----+-----------------+-----------------+
```

1.1.B: Position with Talents (Australia)

```
+--------+-----------+--------------+
|Position|Nationality|avg(Potential)|
```

```
+--------+----------+-------------+
|     RDM| Australia |          77.0|
+--------+----------+-------------+
```

1.1.C: Plot as required

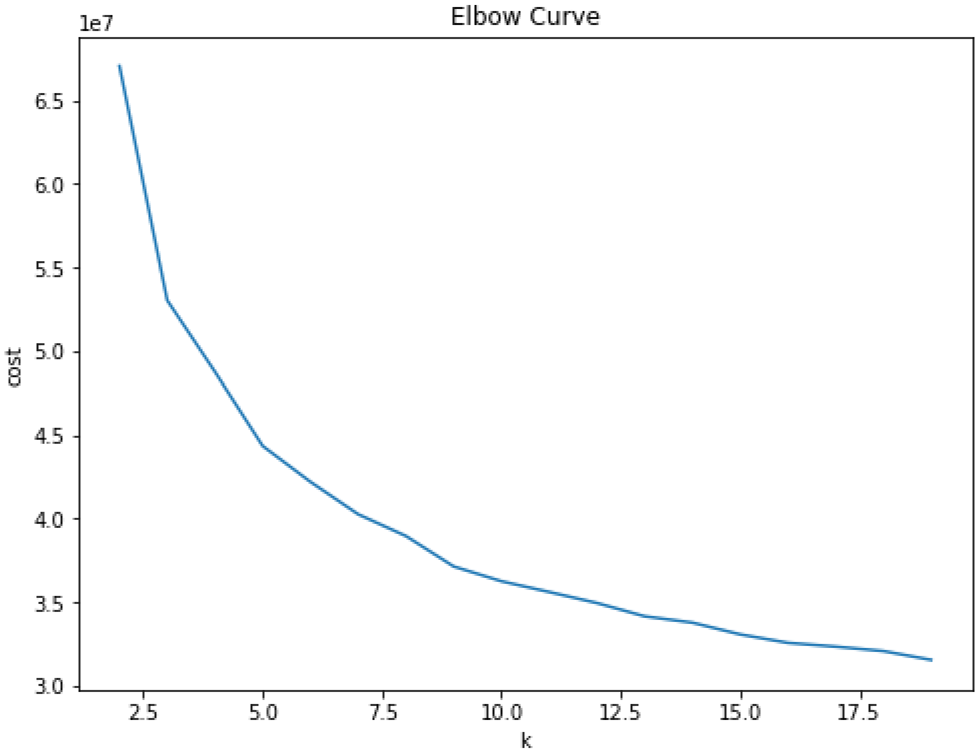


1.1.C: Identify the elbow point for Age

```
+---+-----------------+
|Age|  avg (Potential)|
+---+-----------------+
| 17|74.33910034602076|
+---+-----------------+
```

Section 2: Data Analytics (Part 2)

2.1: Plot with K [2-20]



2.2: Clustering findings

```
Cluster Centers:
[177.3317464   76.31794486  58.41639945  44.02061383  62.79111315
  67.3220339   42.5387082   61.02611086  52.17911131  46.79294549
  63.5510765   65.5758131   65.25240495  66.06596427  64.62757673
  66.20568026  64.10581768  62.28263857  70.52038479  73.82638571
  72.67384333  52.59688502  71.43426477  67.65048099  53.27072836
  57.47457627  48.97938617  64.57627119  66.67155291  68.96472744
  66.88914338]
[187.68026262  81.70680941  39.8684377   30.21252372  68.72675522
  59.71600253  31.19354839  44.54332701  34.72485769  33.42631246
  54.99114485  55.46110057  53.19481341  55.6059456   50.15876028
  62.97975965  50.6116382   51.14231499  68.34977862  64.29095509
  79.52941176  34.24351676  70.56799494  67.06388362  33.600253
  41.73561037  41.05566097  61.12903226  67.29411765  69.6116382
  66.76533839]
[172.46617578  73.21508518  65.18944637  68.70069204  57.79541522
  69.00043253  64.56055363  73.22923875  67.01859862  61.3399654
  60.86245675  72.86548443  76.23096886  75.48442907  76.85856401
  68.82093426  73.3416955   71.57871972  66.4217128   69.08953287
  63.61937716  67.48788927  56.35856401  35.85250865  70.45458478
  67.55363322  65.27032872  68.70804498  36.98140138  34.27465398
  30.68425606]
[179.48039466  76.0082903   36.06083086  24.84124629  55.70029674
```

```
  47.25816024  27.14169139  39.07121662  29.75296736  28.67655786
  39.29896142  45.00667656  60.3189911   60.53189911  53.30267062
  53.134273    59.04154303  38.19065282  67.60089021  60.8805638
  67.21216617  25.61053412  57.08234421  56.22922849  31.92062315
  34.70697329  36.51632047  47.67507418  57.19139466  60.84718101
  58.89836795]
[171.83121978  73.63665232  67.45570663  58.56901525  58.09394314
  72.50432633  57.66131026  70.06098063  66.77997528  62.4223321
  69.03461063  72.1223733   69.90688092  69.05644829  71.96744953
  69.5434693   71.48413679  70.77832715  67.38648537  75.37206428
  67.24639473  66.29377833  68.8170581   66.05274001  65.21137206
  68.49938195  59.71693449  69.01771735  63.46147507  66.48413679
  63.407911  ]
[170.91016533  69.059326    55.29234568  55.09580247  43.15901235
  60.04740741  48.40049383  64.16888889  52.72395062  46.47061728
  53.46666667  62.81580247  74.15802469  73.06765432  72.88296296
  54.81975309  74.18222222  58.15407407  57.08493827  59.05530864
  50.88        51.04098765  42.44888889  30.00395062  56.25580247
  56.67012346  52.63111111  55.06469136  35.40345679  33.6108642
  32.67061728]
[179.04405592  77.27376948  41.5253664   64.62852311  62.52367531
  54.67080045  54.12401353  60.1572717   45.93179256  38.34892897
  40.62514092  61.22886133  67.10372041  68.4475761   63.3934611
  58.98816234  61.21364149  63.87373168  66.86583991  62.50281849
  69.78015784  56.90078918  48.6854566   22.34441939  62.61555806
  51.56313416  61.20744081  56.67080045  27.09244645  21.86414882
  19.75479143]
[170.42364272  71.65120522  54.40978964  39.52872168  51.07605178
  60.99029126  37.06432039  58.46197411  45.36407767  40.24959547
  55.51173139  59.90978964  69.55987055  69.1565534   67.27548544
  58.08859223  69.60679612  50.26456311  64.68608414  67.79652104
  60.88754045  42.32686084  59.25444984  56.58576052  50.67435275
  51.5262945   43.85234628  53.67677994  56.21480583  59.43891586
  57.93082524]
```

2.2: Position Group for Clusters

```
Cluster 0 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID|  967|
|           DEF| 1230|
|           FWD|   11|
+--------------+-----+

Cluster 1 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID|  117|
|           DEF| 1462|
|           FWD|    2|
+--------------+-----+
```

```
Cluster 2 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID| 1195|
|           DEF|    4|
|           FWD| 1112|
+--------------+-----+

Cluster 3 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID|   77|
|           DEF| 1266|
|           FWD|    1|
+--------------+-----+

Cluster 4 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID| 1720|
|           DEF|  599|
|           FWD|   71|
+--------------+-----+

Cluster 5 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID| 1486|
|           DEF|    8|
|           FWD|  535|
+--------------+-----+

Cluster 6 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID|  135|
|           FWD| 1648|
+--------------+-----+

Cluster 7 has:
+--------------+-----+
|Position_Group|count|
+--------------+-----+
|           MID| 1141|
|           DEF| 1297|
|           FWD|   38|
+--------------+-----+
```

Section 3: Data Analytics

3.2: Confusion Matrix

```
array([[ 834,    5,  178],
       [   2, 1422,  293],
       [ 225,  238, 1619]])
```

3.2: P/R/F1 Score

```
[ ] from sklearn.metrics import classification_report

    # Your Code
    target_names = ['FWD', 'DEF', 'MID']
    print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

         FWD       0.87      0.74      0.80      1017
         DEF       0.86      0.90      0.88      1717
         MID       0.79      0.82      0.81      2082

    accuracy                           0.83      4816
   macro avg       0.84      0.82      0.83      4816
weighted avg       0.83      0.83      0.83      4816
```

3.3: KfCV Results

3.3.1 Random forest model

A. Code

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Random forest classification model

#Train a random forest model
rf = RandomForestClassifier(labelCol='label', featuresCol='Scaled_features
', numTrees=10)

#Chain forest in Pipeline
pipeline = Pipeline(stages=[rf])

#Set parameters for cross validation
```

```
paramGrid = ParamGridBuilder(). addGrid(rf.numTrees,[10,20,30,40,50]) .bui
ld()

#Cross Validator
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(),
                          numFolds=5)

#Fit training data to the best hyper parameters identified by validator
cvModel = crossval.fit(train)

print("**Best model is:\n",cvModel.bestModel.stages[0]) #Best case for num
trees in rf

prediction = cvModel.bestModel.transform(test)
selected = prediction.select("label", "prediction")

#Evaluate accuracy of model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
RFM_accurary = evaluator.evaluate(prediction)
print("\nRandom forest Model accuracy:", RFM_accurary)

print("\n**Classification report for Random forest model**\n")

y_true = [int(row.label) for row in prediction.collect()]
y_pred = [int(row.prediction) for row in prediction.collect()]

print(classification_report(y_true, y_pred, target_names=target_names))
```

B. Result

```
**Best model is:
 RandomForestClassificationModel (uid=RandomForestClassifier_559b62593514) with 20 trees

Random forest Model accuracy: 0.8306779495418989

**Classification report for Random forest model**

              precision    recall  f1-score   support

         FWD       0.87      0.74      0.80      1017
         DEF       0.86      0.90      0.88      1717
         MID       0.79      0.82      0.81      2082

    accuracy                           0.83      4816
   macro avg       0.84      0.82      0.83      4816
weighted avg       0.83      0.83      0.83      4816
```

3.3.2 Decision tree model

A. Code

```
#Decision Tree classification Model

from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(labelCol="label", featuresCol="Scaled_features
")
pipeline = Pipeline(stages=[dt])

paramGrid = ParamGridBuilder(). addGrid(dt.maxDepth, [2,5]) .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(),
                          numFolds=5)

cvModel = crossval.fit(train)

print("**Best model is:\n", cvModel.bestModel.stages[0]) #Best case for de
pth in dt

prediction = cvModel.bestModel.transform(test)
selected = prediction.select("label", "prediction")

evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
DT_accurary = evaluator.evaluate(prediction)
print("\nDecision tree Model accurary:", DT_accurary)
```

```
print("\n**Classification report for Decision tree model**\n")
y_true = [int(row.label) for row in prediction.collect()]
y_pred = [int(row.prediction) for row in prediction.collect()]

print(classification_report(y_true, y_pred, target_names=target_names))
```

B. Result

```
[ ]
      **Best model is:
      DecisionTreeClassificationModel (uid=DecisionTreeClassifier_6d1adc3ea104) of depth 5 with 47 nodes

      Decision tree Model accurary: 0.811376127079013

      **Classification report for Decision tree model**

                    precision    recall  f1-score   support

               FWD       0.81      0.73      0.77      1017
               DEF       0.85      0.88      0.87      1717
               MID       0.78      0.79      0.79      2082

          accuracy                           0.81      4816
         macro avg       0.81      0.80      0.81      4816
      weighted avg       0.81      0.81      0.81      4816
```

3.3.3 Logistic regression model

A. Code

```
#Logistic regression model
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator

lr = LogisticRegression(labelCol ='label', featuresCol = 'Scaled_features'
,maxIter=10)
pipeline = Pipeline(stages=[lr])

paramGrid = ParamGridBuilder().build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=5)
model = crossval.fit(train)
print("**Best model is:\n",model.bestModel.stages[0]) #Best model
```

```
prediction = model.bestModel.transform(test)
selected = prediction.select("label", "prediction")

evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
LR_accurary = evaluator.evaluate(prediction)
print("\nLogistic Regression Model accurary is:", LR_accurary)

print("\n**Classification report for Logistic regression model**\n")
y_true = [int(row.label) for row in prediction.collect()]
y_pred = [int(row.prediction) for row in prediction.collect()]

print(classification_report(y_true, y_pred, target_names=target_names))
```

B. Result

```
**Best model is:
 LogisticRegressionModel: uid = LogisticRegression_229f166b22ed, numClasses = 3, numFeatures = 32

Logistic Regression Model accurary is: 0.804838079692376

**Classification report for Logistic regression model**

              precision    recall  f1-score   support

         FWD       0.79      0.82      0.80      1017
         DEF       0.85      0.83      0.84      1717
         MID       0.77      0.78      0.78      2082

    accuracy                           0.80      4816
   macro avg       0.80      0.81      0.81      4816
weighted avg       0.81      0.80      0.80      4816
```

3.3: KfCV Findings on Hyperparameters

As we can see in the results, Random forest model gives an accuracy of 83% and an accuracy of 81.1% and 80.4% for Decision tree model and logistics regression model respectively. Also, the classification report suggest Random forest model works best out of the three, for our data with better precision in predicting features for defenders, midfielders and forwards. We can effectively conclude that, we can use Random forest model to perform predictions on 'FIFA – 19' data provided.

3.3: Difficulties and Other Possible Tasks

Difficulties : We received memory dumps when we tried to fit training data to cross validation model, but it was fixed after resetting the environment parameters. We also tried implementing GBT model as well, although it resulted in creating an system error when we tried to fit 'train' data to cross validated model, checked stack overflow for errors although didn't find any working solutions, we assumed the process is creating RDD issues and since all parameters were in place, we proceeded with Logistic regression model. Remaining tasks but not very challenging, time consuming but less challenging than 3.3 and the references were very useful to work on the data.

Possible tasks: The data can used to predict potential that a player can reach over time and it can be very useful to scout young and existing talent for a position in team.

Section 4: Findings and reflections

The data analysis we have performed has high value in terms of the football transfer market. Football clubs can use to data to find player combinations to improve the existing squad and achieve better results in championship/premier leagues. Ex. If Manchester United are looking to fill the position of a Right forward, options they have is to find a young prospect or to find a player with high potential in existing market. Either ways, we can use the data analysis and predictions to find if player can provide desired quality and has the potential to fit in the squad and deliver for the team. They can send their scouts accordingly to different countries to evaluate the performance of shortlisted players and optimize their search.

Section 5: Group Task distribution

Student Name (id)	Sections worked on	Percentage of tasks
Rajeshkumar Mourya (218615876)	Section 2 and 3, Report	33.33%
Priyanka Naidu (219306186)	Section 1 and 2, Report	33.33%
Saikumar Chimakurthi (219380896)	Section 2 and 3, Report	33.33%

Section 6: References

Apache spark 2020, *Pyspark.ml package,* Retrieved 21 May 2020 <https://spark.apache.org/docs/2.0.0/api/python/pyspark.ml.html>

Apache spark 2020, *Classification and regression,* Retrieved 21 May 2020, <https://spark.apache.org/docs/latest/ml-classification-regression.html>

Apache spark 2020, *Naive Bayer,* Retrieved 22 May 2020, <https://spark.apache.org/docs/latest/ml-classification-regression.html#naive-bayes>

Apache spark 2020, *Random forest classifier,* Retrieved 22 May 2020, <https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier>

Apache spark 2020, *ML tuning,* retrieved 26 May 2020, <https://spark.apache.org/docs/latest/ml-tuning.html>

Databricks 2020, *GBT regression,* Retrieved 25 May 2020, <https://docs.azuredatabricks.net/_static/notebooks/gbt-regression.html>

Databricks 2020, *decision trees example,* Retrieved 22 May 2020, <https://docs.databricks.com/applications/machine-learning/mllib/decision-trees.html#decision-trees-examples>

Krishnan, Muthu 2020, *Understanding classification report in sklearn,* retrieved 25 May 2020 <https://muthu.co/understanding-the-classification-report-in-sklearn/>

Ali 2019, *how to find appropriate number of clusters,* retrieved 22 May 2020 <https://stackoverflow.com/questions/56371503/pyspark-how-to-find-appropriate-number-of-clusters>