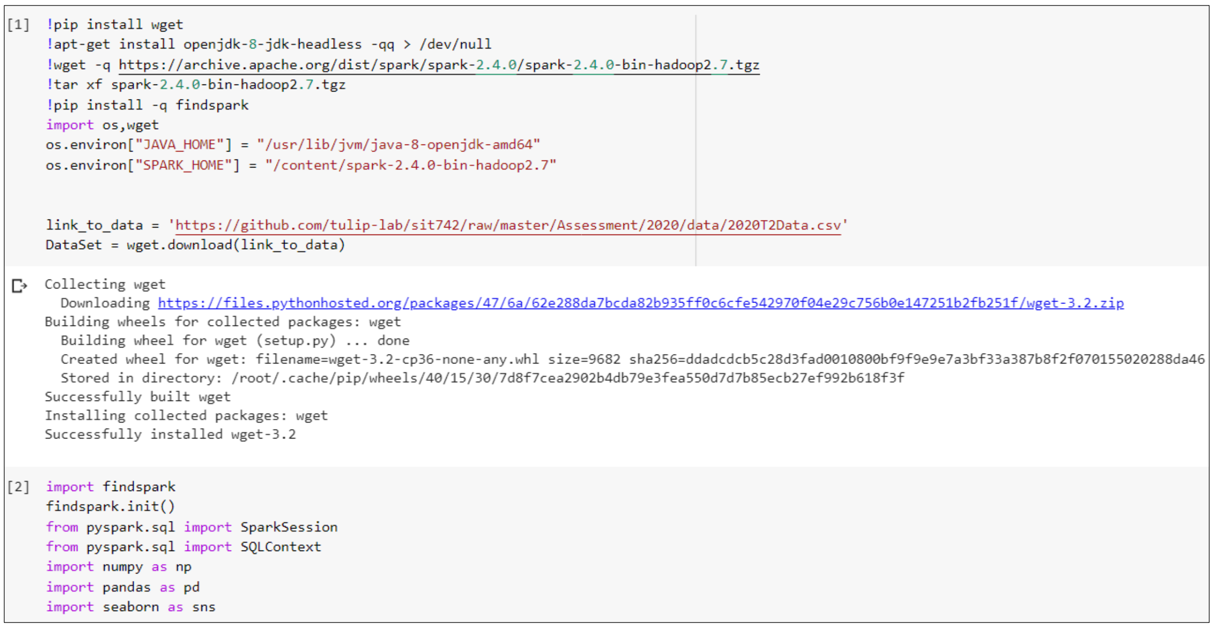
Table of Contents

Section 1: What we could know about FIFA 2019 Players?

1.0 Libraries and data files:

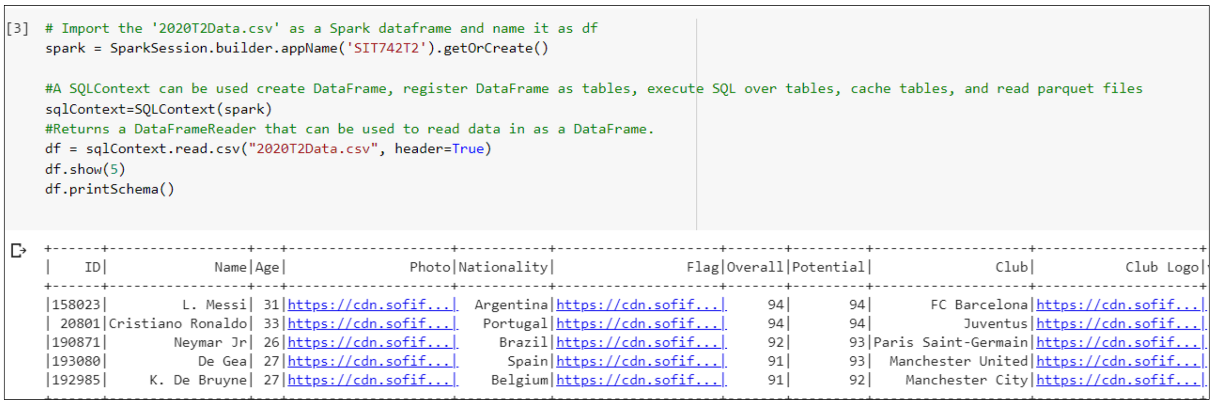


```
[1]  !pip install wget
     !apt-get install openjdk-8-jdk-headless -qq > /dev/null
     !wget -q https://archive.apache.org/dist/spark/spark-2.4.0/spark-2.4.0-bin-hadoop2.7.tgz
     !tar xf spark-2.4.0-bin-hadoop2.7.tgz
     !pip install -q findspark
     import os,wget
     os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
     os.environ["SPARK_HOME"] = "/content/spark-2.4.0-bin-hadoop2.7"


     link_to_data = 'https://github.com/tulip-lab/sit742/raw/master/Assessment/2020/data/2020T2Data.csv'
     DataSet = wget.download(link_to_data)
```

```
     Collecting wget
       Downloading https://files.pythonhosted.org/packages/47/6a/62e288da7bcda82b935ff0c6cfe542970f04e29c756b0e147251b2fb251f/wget-3.2.zip
     Building wheels for collected packages: wget
       Building wheel for wget (setup.py) ... done
       Created wheel for wget: filename=wget-3.2-cp36-none-any.whl size=9682 sha256=ddadcdcb5c28d3fad0010800bf9f9e9e7a3bf33a387b8f2f070155020288da46
       Stored in directory: /root/.cache/pip/wheels/40/15/30/7d8f7cea2902b4db79e3fea550d7d7b85ecb27ef992b618f3f
     Successfully built wget
     Installing collected packages: wget
     Successfully installed wget-3.2
```

```
[2]  import findspark
     findspark.init()
     from pyspark.sql import SparkSession
     from pyspark.sql import SQLContext
     import numpy as np
     import pandas as pd
     import seaborn as sns
```

1.1 Data Exploration:

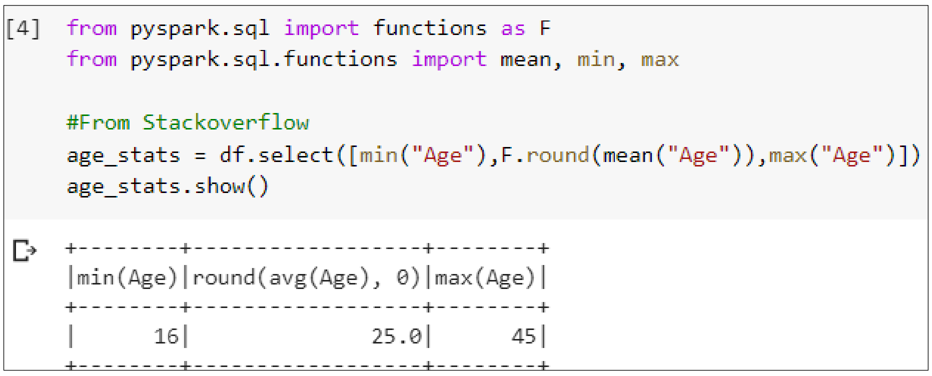


```
[3]  # Import the '2020T2Data.csv' as a Spark dataframe and name it as df
     spark = SparkSession.builder.appName('SIT742T2').getOrCreate()

     #A SQLContext can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files
     sqlContext=SQLContext(spark)
     #Returns a DataFrameReader that can be used to read data in as a DataFrame.
     df = sqlContext.read.csv("2020T2Data.csv", header=True)
     df.show(5)
     df.printSchema()
```

```
+------+-----------------+---+--------------------+-----------+--------------------+-------+---------+-------------------+--------------------+
|    ID|             Name|Age|               Photo|Nationality|                Flag|Overall|Potential|               Club|           Club Logo|
+------+-----------------+---+--------------------+-----------+--------------------+-------+---------+-------------------+--------------------+
|158023|         L. Messi| 31|https://cdn.sofif...|  Argentina|https://cdn.sofif...|     94|       94|       FC Barcelona|https://cdn.sofif...|
| 20801|Cristiano Ronaldo| 33|https://cdn.sofif...|   Portugal|https://cdn.sofif...|     94|       94|           Juventus|https://cdn.sofif...|
|190871|        Neymar Jr| 26|https://cdn.sofif...|     Brazil|https://cdn.sofif...|     92|       93|Paris Saint-Germain|https://cdn.sofif...|
|193080|           De Gea| 27|https://cdn.sofif...|      Spain|https://cdn.sofif...|     91|       93|  Manchester United|https://cdn.sofif...|
|192985|     K. De Bruyne| 27|https://cdn.sofif...|    Belgium|https://cdn.sofif...|     91|       92|    Manchester City|https://cdn.sofif...|
```

1.1.A Answer the following:

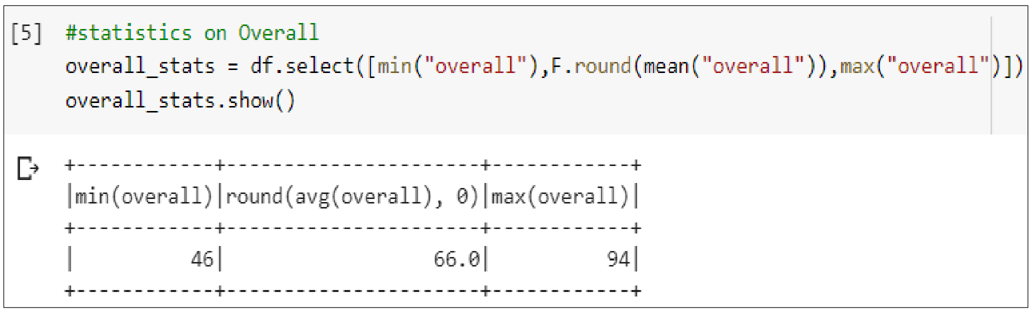
1. Min, max, and mean for Age



```
[4]  from pyspark.sql import functions as F
     from pyspark.sql.functions import mean, min, max

     #From Stackoverflow
     age_stats = df.select([min("Age"),F.round(mean("Age")),max("Age")])
     age_stats.show()
```

```
     +--------+-----------------+--------+
     |min(Age)|round(avg(Age), 0)|max(Age)|
     +--------+-----------------+--------+
     |      16|             25.0|      45|
     +--------+-----------------+--------+
```

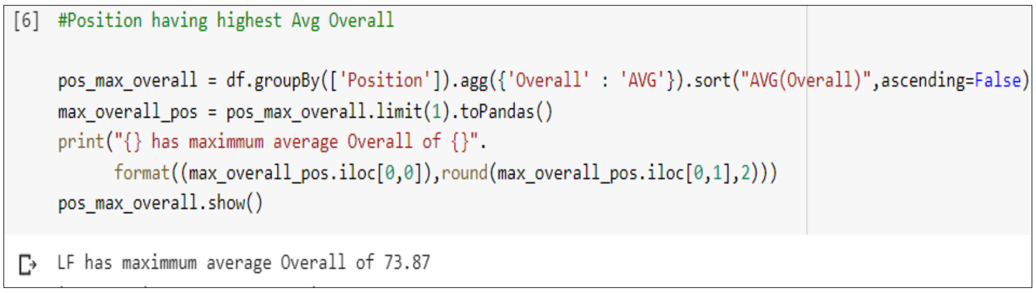
2. Min, max, and mean for Overall



```
[5]  #statistics on Overall
     overall_stats = df.select([min("overall"),F.round(mean("overall")),max("overall")])
     overall_stats.show()
```

```
+------------+---------------------+------------+
|min(overall)|round(avg(overall), 0)|max(overall)|
+------------+---------------------+------------+
|          46|                 66.0|          94|
+------------+---------------------+------------+
```

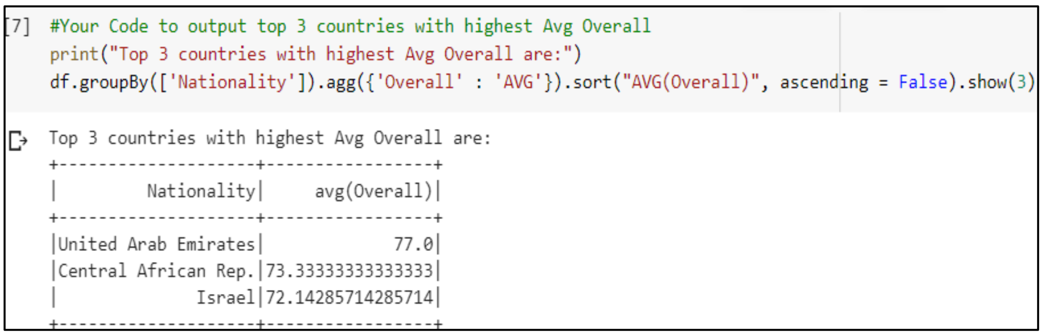
3. Position having highest Avg Overall:



```
[6]  #Position having highest Avg Overall

     pos_max_overall = df.groupBy(['Position']).agg({'Overall' : 'AVG'}).sort("AVG(Overall)",ascending=False)
     max_overall_pos = pos_max_overall.limit(1).toPandas()
     print("{} has maximmum average Overall of {}".
           format((max_overall_pos.iloc[0,0]),round(max_overall_pos.iloc[0,1],2)))
     pos_max_overall.show()
```

```
LF has maximmum average Overall of 73.87
```

4. Top 3 countries with highest Avg Overall:

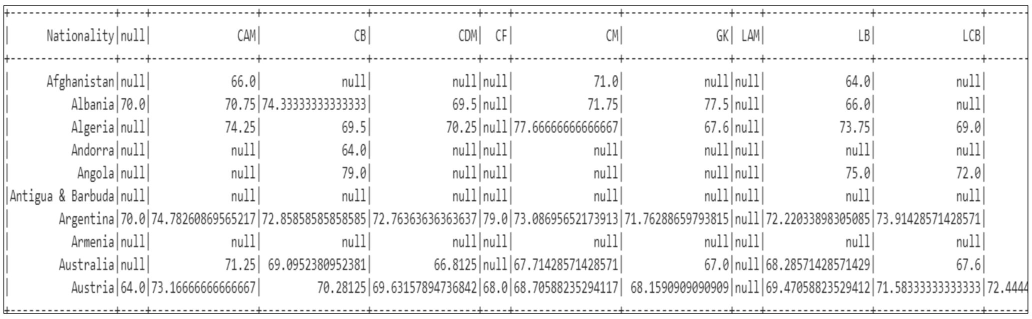


```
[7]  #Your Code to output top 3 countries with highest Avg Overall
     print("Top 3 countries with highest Avg Overall are:")
     df.groupBy(['Nationality']).agg({'Overall' : 'AVG'}).sort("AVG(Overall)", ascending = False).show(3)
```

```
Top 3 countries with highest Avg Overall are:
+--------------------+-----------------+
|         Nationality|     avg(Overall)|
+--------------------+-----------------+
|United Arab Emirates|             77.0|
|Central African Rep.|73.33333333333333|
|              Israel|72.14285714285714|
+--------------------+-----------------+
```

1.1.B Answer the following:

1. Avg Potentials on Country by Position sorted based on country alphabetically:



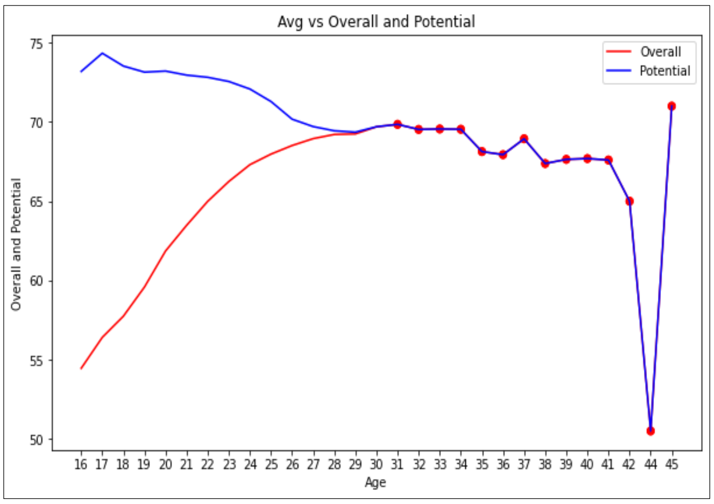
```
+---------------+----+----------------+----------------+----------------+----+----------------+----------------+----+----------------+----------------+------
|    Nationality|null|             CAM|              CB|             CDM|  CF|              CM|              GK| LAM|              LB|             LCB|
+---------------+----+----------------+----------------+----------------+----+----------------+----------------+----+----------------+----------------+------
|    Afghanistan|null|            66.0|            null|            null|null|            71.0|            null|null|            64.0|            null|
|        Albania|70.0|           70.75|74.3333333333333|            69.5|null|           71.75|            77.5|null|            66.0|            null|
|        Algeria|null|           74.25|            69.5|           70.25|null|77.6666666666667|            67.6|null|           73.75|            69.0|
|        Andorra|null|            null|            64.0|            null|null|            null|            null|null|            null|            null|
|         Angola|null|            null|            79.0|            null|null|            null|            null|null|            75.0|            72.0|
|Antigua & Barbuda|null|          null|            null|            null|null|            null|            null|null|            null|            null|
|      Argentina|70.0|74.7826086956521 7|72.8585858585858 5|72.7636363636363 7|79.0|73.0869565217391 3|71.7628865979381 5|null|72.2203389830508 5|73.9142857142857 1|
|        Armenia|null|            null|            null|            null|null|            null|            null|null|            null|            null|
|      Australia|null|           71.25| 69.0952380952381|         66.8125|null|67.7142857142857 1|            67.0|null|68.2857142857142 9|            67.6|
|        Austria|64.0|73.1666666666666 7|        70.28125|69.6315789473684 2|68.0|68.7058823529411 7| 68.1590909090909|null|69.4705882352941 2|71.5833333333333 3|72.4444
+---------------+----+----------------+----------------+----------------+----+----------------+----------------+----+----------------+----------------+------
```

2. Position having highest Avg Potential for Australia:

```
Position with highest Avg Potential for Australia is:
+--------+--------------+
|Position|avg(Potential)|
+--------+--------------+
|     RDM|          77.0|
+--------+--------------+
```

1.1.C Age when players fully released their potential:

1. Plot of Average age and average Overall w.r.t. Age of players



2. Overall and Potential score intersect at the age 30 years. Thus, on an average player fully released their potential by the age of 30 years.

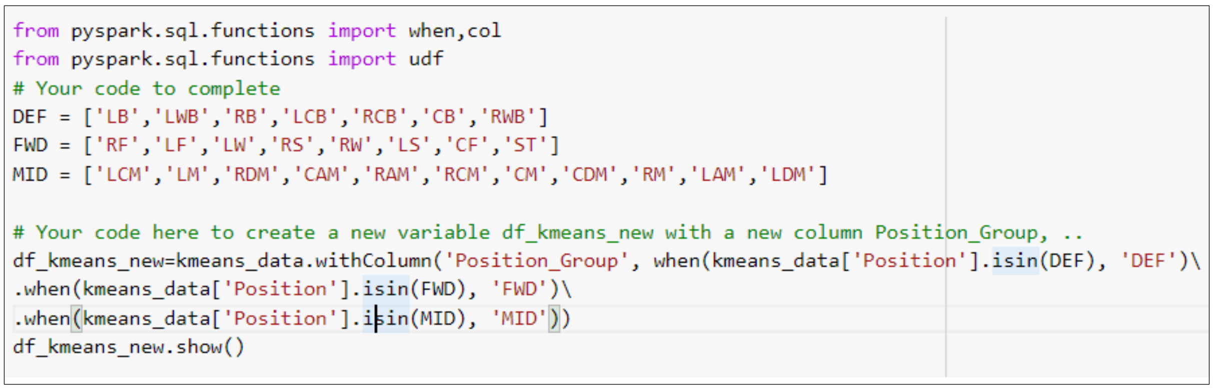
Section 2 - Unsupervised Learning: Kmeans

2.1 Data preparation:

Selecting specific features from the original dataframe.

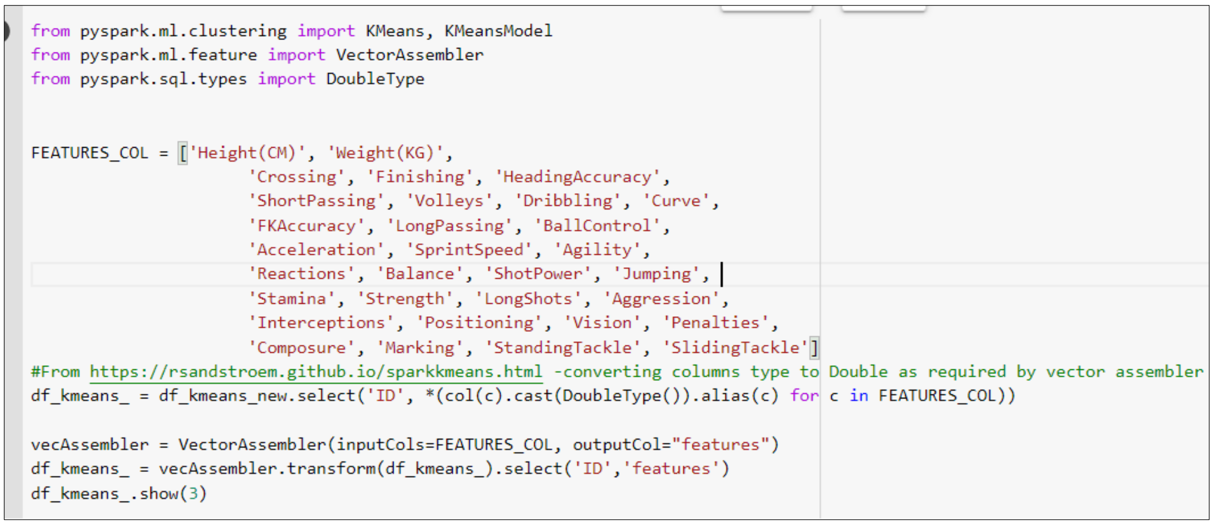
```
# Your code to select relevant features and filter the data by removing the GK
kmeans_data = df.filter(df['Position'] != 'GK')
kmeans_data = kmeans_data.select('ID','Position','Height(CM)','Weight(KG)','Crossing','Finishing','HeadingAccuracy',
                                 'ShortPassing','Volleys','Dribbling','Curve','FKAccuracy',
                                 'LongPassing','BallControl','Acceleration','SprintSpeed',
                                 'Agility','Reactions','Balance','ShotPower','Jumping','Stamina',
                                 'Strength','LongShots','Aggression','Interceptions',
                                 'Positioning','Vision','Penalties','Composure','Marking',
                                 'StandingTackle','SlidingTackle')


kmeans_data.show()
```

Creating a categorical variable based on Position categories



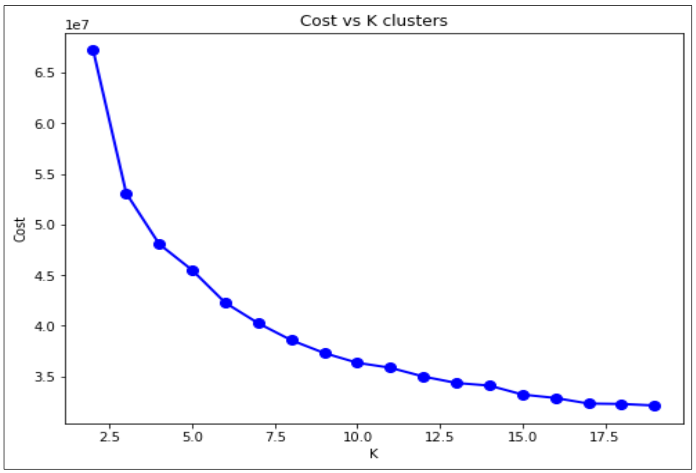
```
from pyspark.sql.functions import when,col
from pyspark.sql.functions import udf
# Your code to complete
DEF = ['LB','LWB','RB','LCB','RCB','CB','RWB']
FWD = ['RF','LF','LW','RS','RW','LS','CF','ST']
MID = ['LCM','LM','RDM','CAM','RAM','RCM','CM','CDM','RM','LAM','LDM']

# Your code here to create a new variable df_kmeans_new with a new column Position_Group, ..
df_kmeans_new=kmeans_data.withColumn('Position_Group', when(kmeans_data['Position'].isin(DEF), 'DEF')\
.when(kmeans_data['Position'].isin(FWD), 'FWD')\
.when(kmeans_data['Position'].isin(MID), 'MID'))
df_kmeans_new.show()
```

Creating ID column as a Key variable and independent variables concatenated as features



```
from pyspark.ml.clustering import KMeans, KMeansModel
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.types import DoubleType


FEATURES_COL = ['Height(CM)', 'Weight(KG)',
                      'Crossing', 'Finishing', 'HeadingAccuracy',
                      'ShortPassing', 'Volleys', 'Dribbling', 'Curve',
                      'FKAccuracy', 'LongPassing', 'BallControl',
                      'Acceleration', 'SprintSpeed', 'Agility',
                      'Reactions', 'Balance', 'ShotPower', 'Jumping',
                      'Stamina', 'Strength', 'LongShots', 'Aggression',
                      'Interceptions', 'Positioning', 'Vision', 'Penalties',
                      'Composure', 'Marking', 'StandingTackle', 'SlidingTackle']
#From https://rsandstroem.github.io/sparkkmeans.html -converting columns type to Double as required by vector assembler
df_kmeans_ = df_kmeans_new.select('ID', *(col(c).cast(DoubleType()).alias(c) for c in FEATURES_COL))

vecAssembler = VectorAssembler(inputCols=FEATURES_COL, outputCol="features")
df_kmeans_ = vecAssembler.transform(df_kmeans_).select('ID','features')
df_kmeans_.show(3)
```

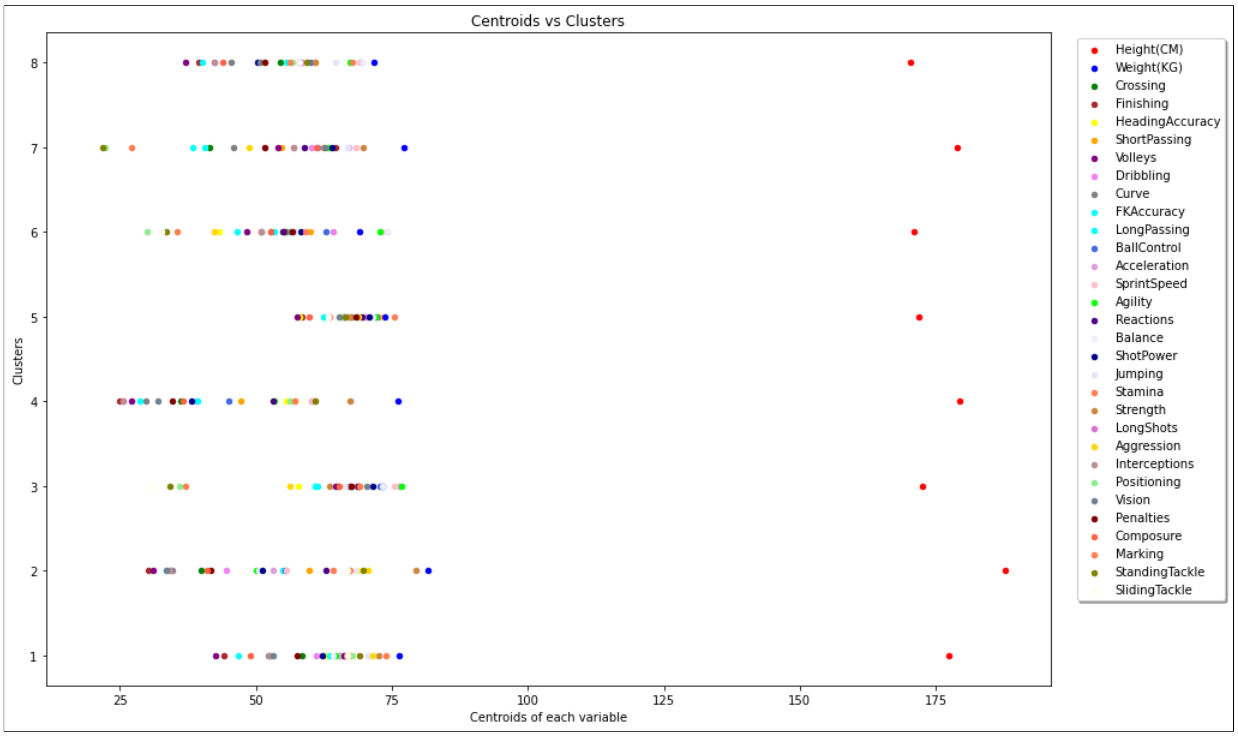
<u>Kmeans model elbow plot</u>: As K increases cost function reduces because as K increases model reduces variance and increases bias. Optimal number of K's using elbow method suggests K where it creates an elbow. Thus, I would use 7 clusters.



2.2 K-Means:

2.2.1 Cluster centroids:

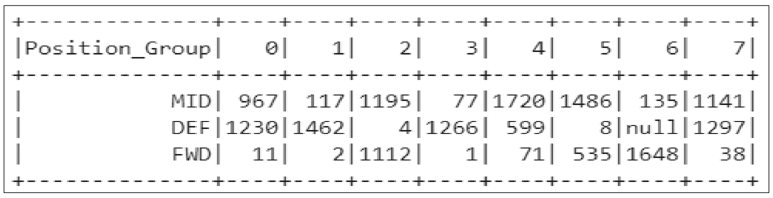
Centroids of each cluster are at distance. For eg, in below plot with clusters on Y axis, Height(CM) variable is similar in 5th and 6th cluster, but at distance for clusters.



2.2.2 Relationship between Position_Group and clusters:

We can see almost all the clusters are biased towards any two classes. (Refer fig 1 and 2). Position_group variable created is unbalanced and so do the clusters (Refer fig 3 and 4). We should have standardized features and then use feature extraction algorithms to identify important features. Also, data is sparse, so we should have replaced null values using appropriate method.

Fig 1



```
+--------------+----+----+----+----+----+----+----+----+
|Position_Group|   0|   1|   2|   3|   4|   5|   6|   7|
+--------------+----+----+----+----+----+----+----+----+
|           MID| 967| 117|1195|  77|1720|1486| 135|1141|
|           DEF|1230|1462|   4|1266| 599|   8|null|1297|
|           FWD|  11|   2|1112|   1|  71| 535|1648|  38|
+--------------+----+----+----+----+----+----+----+----+
```

Fig 2

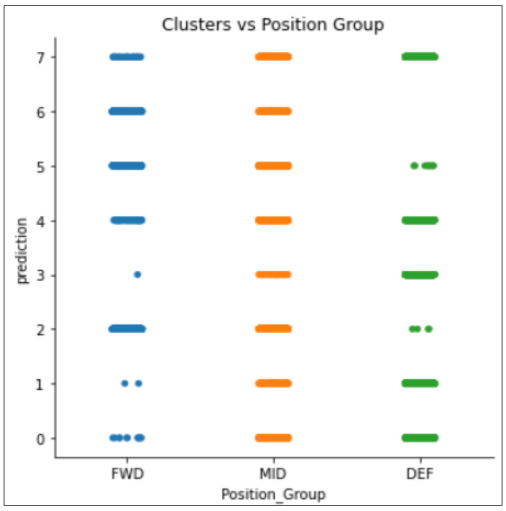


Fig 3

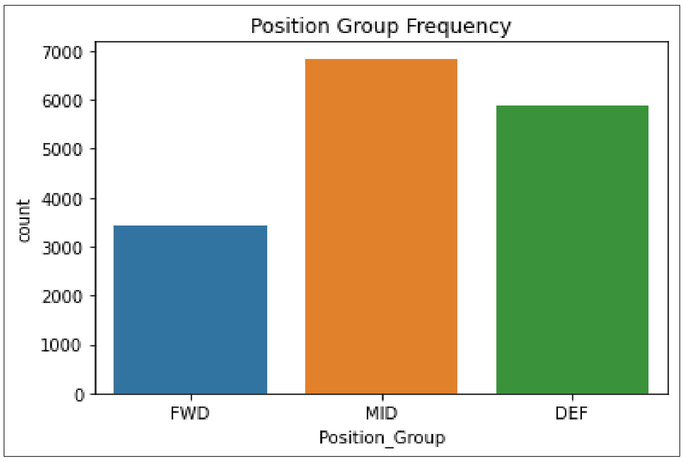
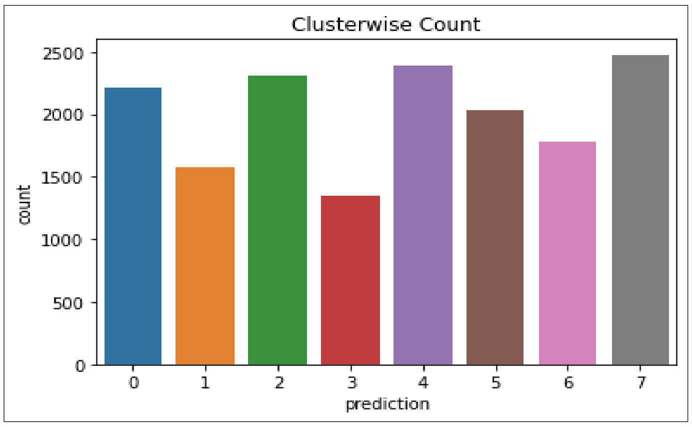


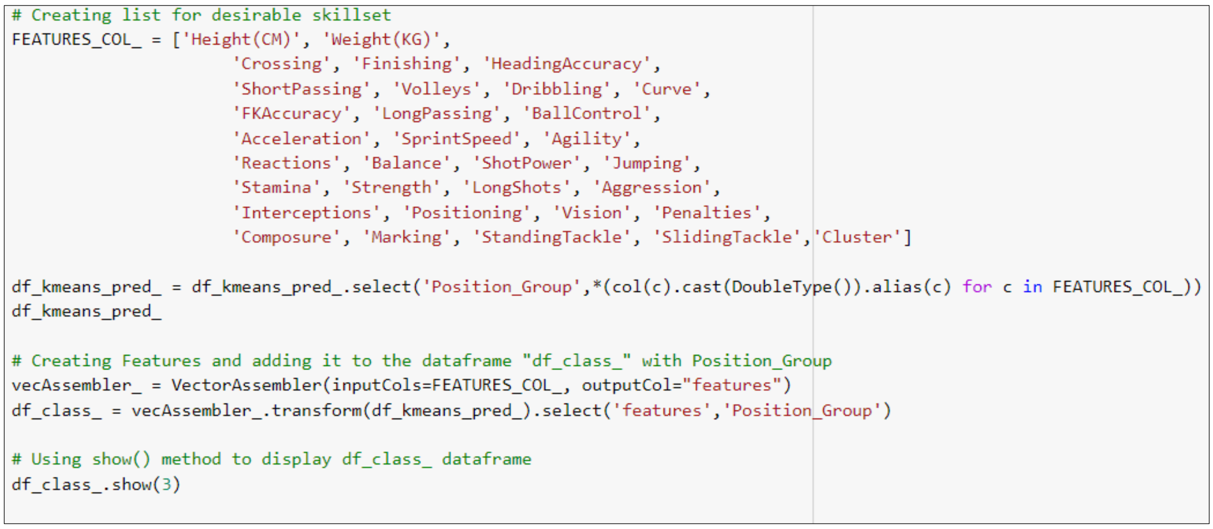
Fig 4



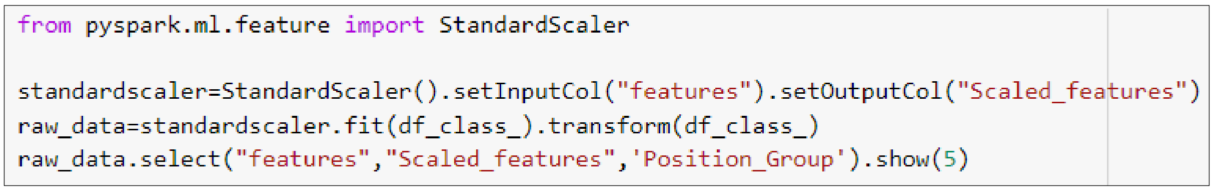
Section 3 - Supervised Learning: Classification on Position_Group

3.1 Data Preparation:

```
# Dropping position and features column
df_kmeans_pred_ = df_kmeans_new.drop('features').drop('position')

# Renaming prediction column to CLuster
df_kmeans_pred_ = df_kmeans_pred_.withColumnRenamed("prediction","Cluster")
```

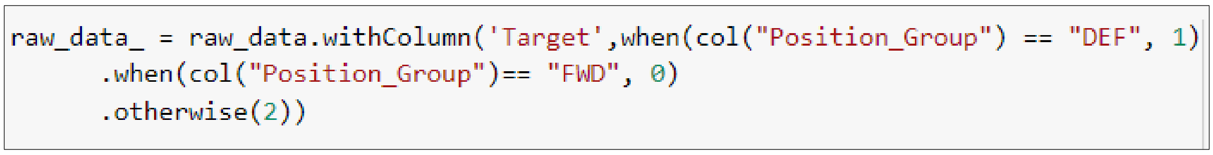


```
# Creating list for desirable skillset
FEATURES_COL_ = ['Height(CM)', 'Weight(KG)',
                      'Crossing', 'Finishing', 'HeadingAccuracy',
                      'ShortPassing', 'Volleys', 'Dribbling', 'Curve',
                      'FKAccuracy', 'LongPassing', 'BallControl',
                      'Acceleration', 'SprintSpeed', 'Agility',
                      'Reactions', 'Balance', 'ShotPower', 'Jumping',
                      'Stamina', 'Strength', 'LongShots', 'Aggression',
                      'Interceptions', 'Positioning', 'Vision', 'Penalties',
                      'Composure', 'Marking', 'StandingTackle', 'SlidingTackle','Cluster']

df_kmeans_pred_ = df_kmeans_pred_.select('Position_Group',*(col(c).cast(DoubleType()).alias(c) for c in FEATURES_COL_))
df_kmeans_pred_

# Creating Features and adding it to the dataframe "df_class_" with Position_Group
vecAssembler_ = VectorAssembler(inputCols=FEATURES_COL_, outputCol="features")
df_class_ = vecAssembler_.transform(df_kmeans_pred_).select('features','Position_Group')

# Using show() method to display df_class_ dataframe
df_class_.show(3)
```



```
from pyspark.ml.feature import StandardScaler

standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
raw_data=standardscaler.fit(df_class_).transform(df_class_)
raw_data.select("features","Scaled_features",'Position_Group').show(5)
```

Creating 'Position_Group' variable using 'Position' variable: Reducing categories to 3

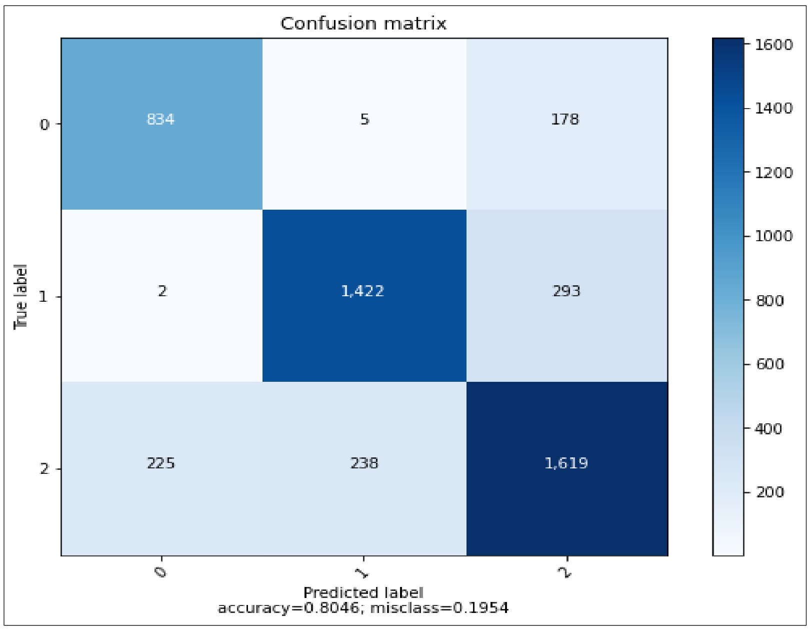


```
raw_data_ = raw_data.withColumn('Target',when(col("Position_Group") == "DEF", 1)
      .when(col("Position_Group")== "FWD", 0)
      .otherwise(2))
```

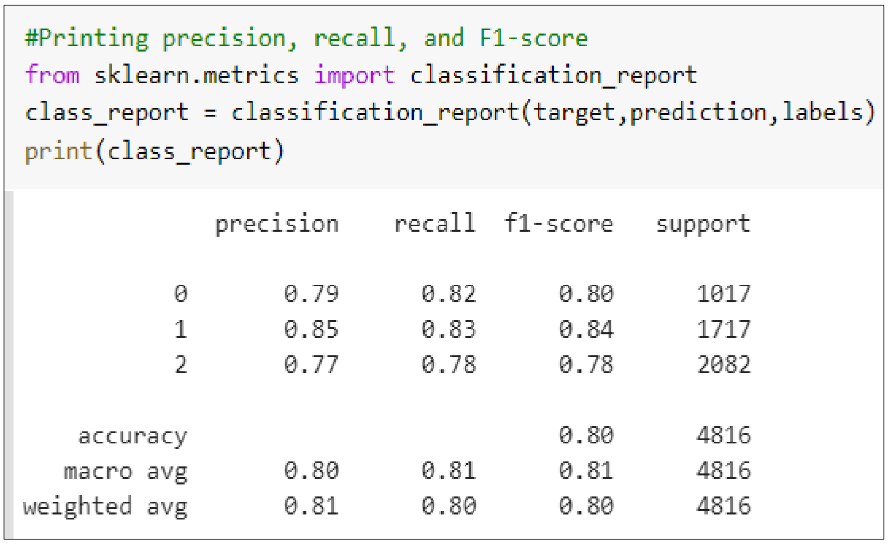
3.2 Training Test Evaluation: Logistic Regression

3.2.1 Confusion Matrix:

Confusion matrix is a crosstab with True labels or Y axis and Predicted labels on X axis. Diagonal (left to right) represents correct classifications by the model and rest of the cells represents misclassification. FWD was misclassified for 183 records, Position group for 295 records, and MID for 463 records. Thus, error is (941/4816= 0.1954) and accuracy is 0.8046.



3.2.2 Precision, Recall and F1 score:



```
#Printing precision, recall, and F1-score
from sklearn.metrics import classification_report
class_report = classification_report(target,prediction,labels)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.82      0.80      1017
           1       0.85      0.83      0.84      1717
           2       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
```

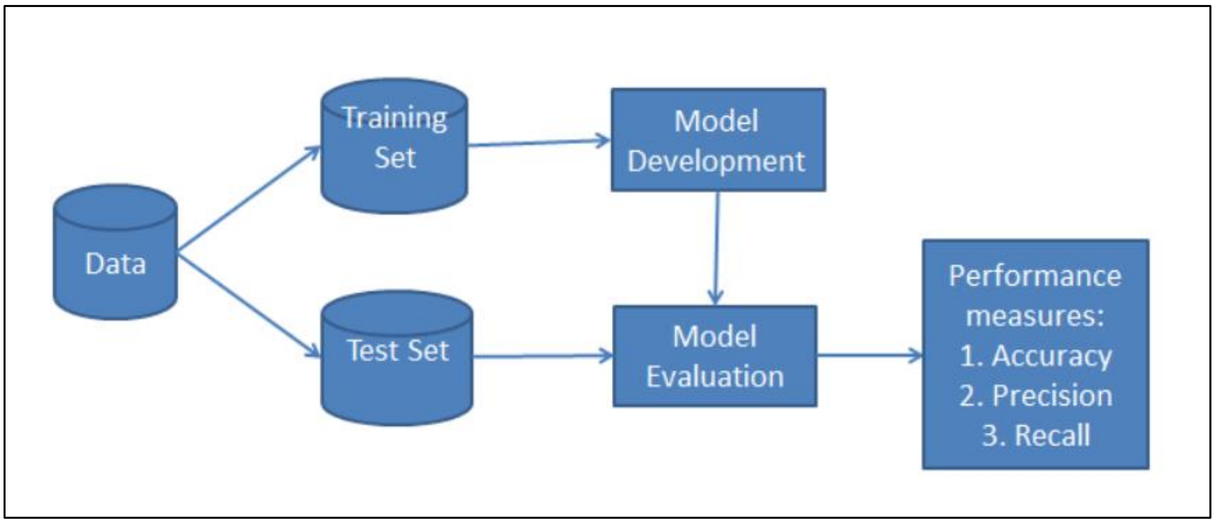
Precision:
Precision = Correct classification/Total predicated classifications for each class. It gives precise accuracy of models' performance, especially when cost of false positive is high. Eg. Prec_FWD = 834/(834+2+225) = 0.79. Precision for DEF i.e. model correctly classified records under DEF class comparatively better than other two classes. It seems, MID and FWD classes have some intersection or inconsistency in their subclasses. Thus, model got confused between MID and FWD.

Recall:
Recall = Correct classification/Total actual classification. It is useful metric when there is high cost of false negative classification. Eg. In medical test reports, false positive is acceptable instead of false negative. Eg. Rec_FWD = 834/(834+5+178) = 0.82. As expected, recall for DEF class is comparatively better than other two classes.

F1 Score:
F1 Score = (2*precision*recall)/(precision+recall). It is a function of Precision and Recall. It is used as metric when we need a balance between false positive and false negative scores (Precision and Recall respectively). Also, it is preferred metric when we have imbalance data where accuracy can be biased towards particular class/es. Eg.

F1 Score_FWD = (2*0.79*0.82)/(0.79+0.82) = 0.8047 ~0.80

3.3 K-fold Cross-Validation models:
Pipeline is used for transformers and estimators. Since, we had already transformed our images only estimators were included in the pipeline for all the models to make models comparable.



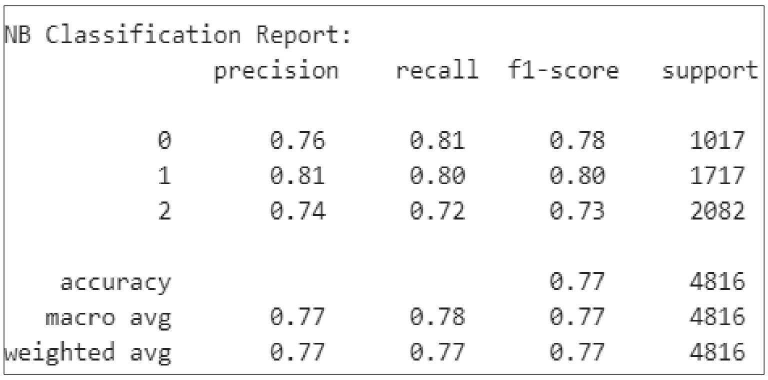
Source: Towards data science

3.3.1 Code design and running results:
Data:
Training and test data type were changed column name Target was renamed to label wherever required.

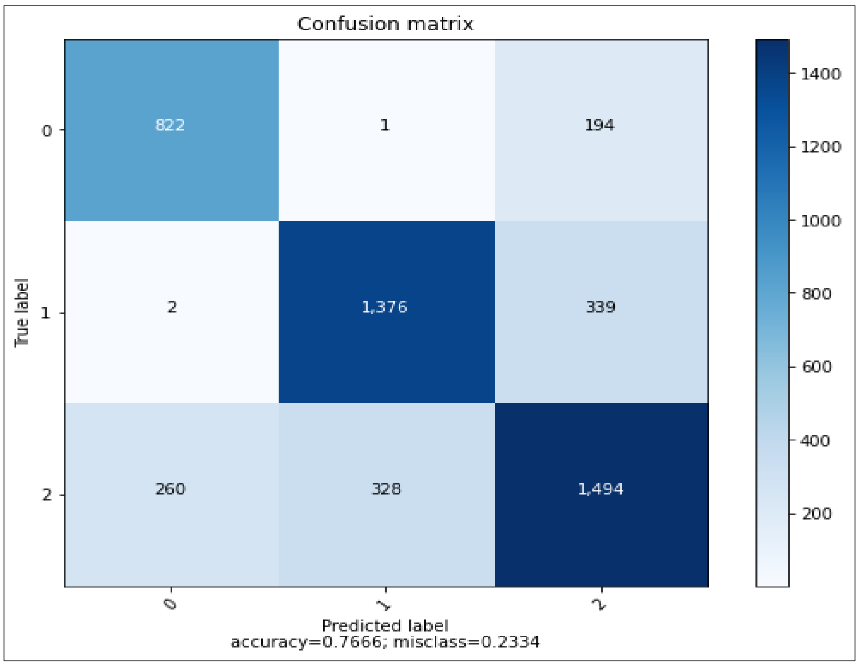
Naïve Bayes Classification (NB): (Yash)
Model development:
Estimator was initialized by giving features and labels, and with “modelType” as multinomial. Pipeline was used for estimator. Smoothing hyper-parameter used were (0,0.2,0.4,0.6,0.8,1). Multiclass classification evaluator was chosen as we have target which is not of binary class. Cross validator with estimator, parameters, evaluator and 10 folds was initialized. Then this was fitted on the training data.

Model Evaluation:
Training accuracy for all smoothing parameters was same. It did not add effect to the model.

Performance measures:
Accuracy on training dataset was 0.76 and on test dataset 0.77. Based on the accuracy, we can say that model did not overfit. Model performed well on DEF class comparatively based on F1 score. Below in the classification report and confusion matrix for test data.



```
NB Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.81      0.78      1017
           1       0.81      0.80      0.80      1717
           2       0.74      0.72      0.73      2082

    accuracy                           0.77      4816
   macro avg       0.77      0.78      0.77      4816
weighted avg       0.77      0.77      0.77      4816
```



Logistic Regression (CV): (Yash)

Model development:

Mode was initialized with features and labels and given to a pipeline. "Maximum iterations" was chosen as hyperparameter with values 50, 60, and 70 and given to parameter grid. Cross-validator was initialized with 20 folds and was trained on training data and fitted on test data.

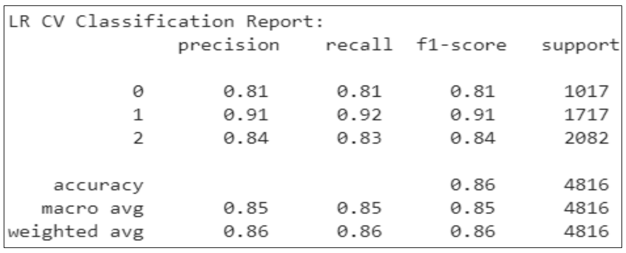
Model evaluation:

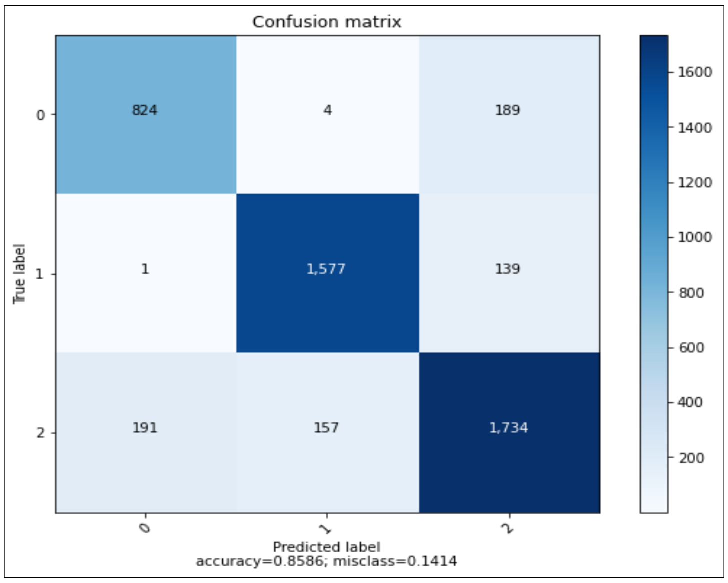
Model performance is similar across iterations. But accuracy dropped when iterations increased to 60 from 50 and again increased for 70 iterations. This seems to be some anomaly.

Max Iterations	50	60	70
Training accuracy	0.8638	0.8635	0.8645

Performance measures:

Cross-validation has increased the accuracy to 0.86 from 0.80 (7.5% improvement). Precision and recall score improved for all classes, especially for DEF class. Average training accuracy was 0.86 and Test accuracy also was 0.86. Thus, model did not overfit.


```
LR CV Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81      1017
           1       0.91      0.92      0.91      1717
           2       0.84      0.83      0.84      2082

    accuracy                           0.86      4816
   macro avg       0.85      0.85      0.85      4816
weighted avg       0.86      0.86      0.86      4816
```

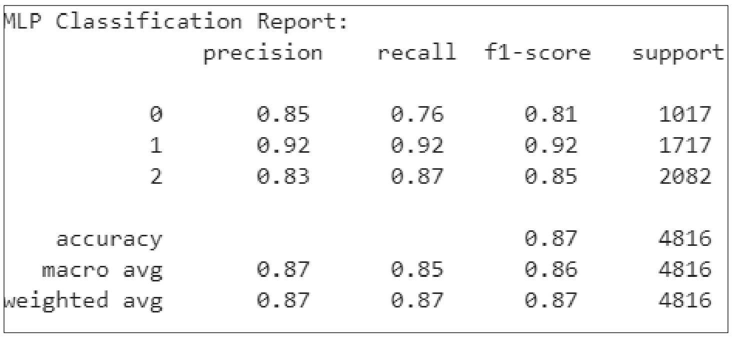


Multilayer Perceptron Classifier: (Vijay)
Model development:
Model was initialized using pipeline with parameters layers, solver, blocksize, tolerance level and cross-validation with 5 folds.

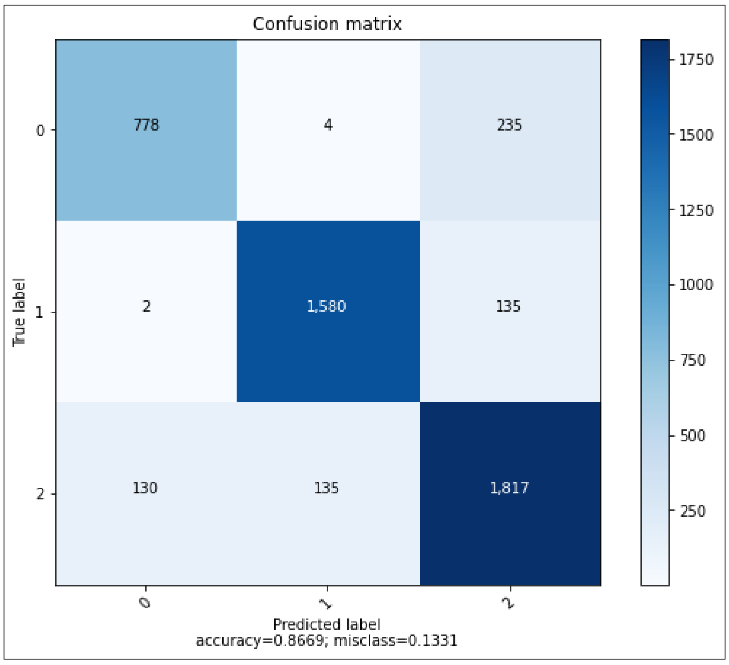
Model evaluation:
MLP outperformed other models with accuracy of 0.87 on training and test both. F1 score also increased with DEF class outperforming others. Depth of layers defines learning capacity of the model, but can perform poor if it is too big, Solver is the optimizer for cost function, blocksize is batch size and lower tolerance level gives better accuracy. Model with layers [32,32,3], blocksize 256 and Tolerance of 1.00E-.06 has outperformed other models.

MLP - Model Evaluation								
Accuracy	0.8647	0.8647	**0.8636**	0.8636	0.8694	0.8694	0.8696	**0.8696**
Layers	[32,32,3]	[32,32,3]	[32,32,3]	[32,31,30,3]	[32,31,30,3]	[32,31,30,3]	[32,31,30,3]	[32,32,3]
Block size	256	512	512	256	256	512	512	256
Tolerance	1.00E-07	1.00E-06	1.00E-07	1.00E-06	1.00E-07	1.00E-06	1.00E-07	1.00E-06

Performance measures:



```
MLP Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.76      0.81      1017
           1       0.92      0.92      0.92      1717
           2       0.83      0.87      0.85      2082

    accuracy                           0.87      4816
   macro avg       0.87      0.85      0.86      4816
weighted avg       0.87      0.87      0.87      4816
```



3.3.2 Findings on hyperparameters: (Yash)

As mentioned in model evaluations, smoothing hyperparameter did no affect model performance and there was anomaly result by Max iterations parameter in logistic regression model.

3.3.3 Difficulties: (Yash)

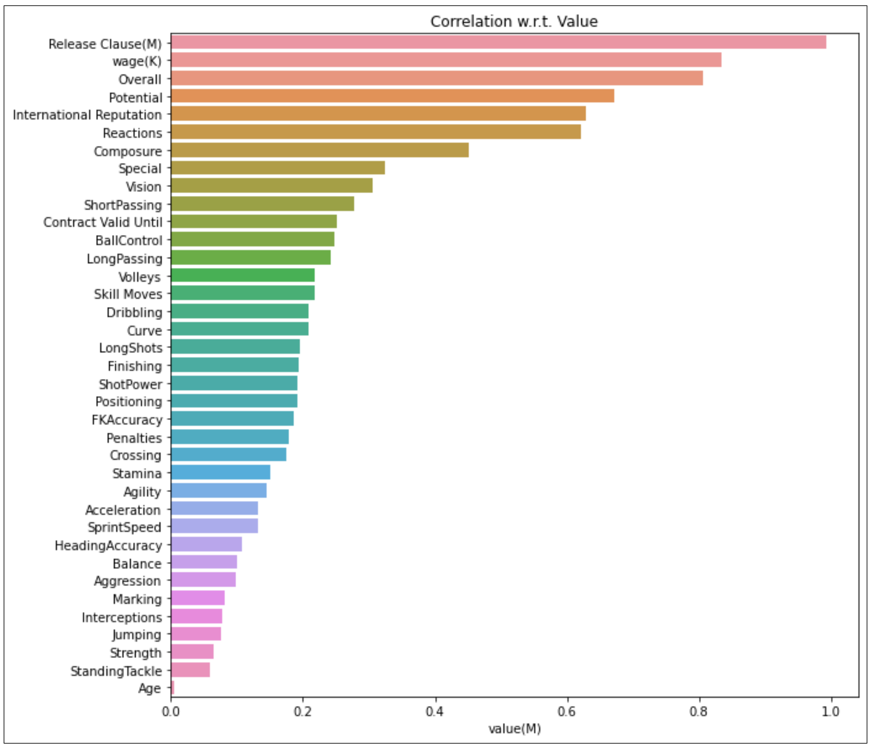
a) Documentation of pyspark is comparatively less comprehensive than other libraries.
b) Less models and parameters. KNN is not available in pyspark. Complement Naïve Bayes is better for unbalance data, but it is not available in pyspark.
c) Residual analysis of linear regression could not be performed as ANOVA and post-model diagnosis tools are not easily available.

3.3.4 Other tasks: (Yash)

We can use this data for valuation of players, predict winning team based on average potential team wise.

Section 4 - Any findings?

We have created a model considering General Manager as a user. Linear regression is chosen purposely to have parametric model. Objective was to check which factors affect player's value and how GM can predict profitable opportunity. Surprisingly, all skills has less correlation with market value of a player.



Data: Based on correlation coefficient, below eleven variables were chosen. All rows with null values were drop. Data split; 70% for training and 30% for test.

```
#Defining regression variables - Dropping Age and value from the list
reg_variables = ['Release Clause(M)','wage(K)','Overall','Potential',
'International Reputation','Reactions','Composure','Special',
'Vision','ShortPassing','Contract Valid Until']

#Vectorizing features
vecAssembler_reg = VectorAssembler(inputCols=reg_variables, outputCol="features")
df_reg = vecAssembler_reg.transform(df_val).select('value(M)','features')
df_reg.show(3)
```

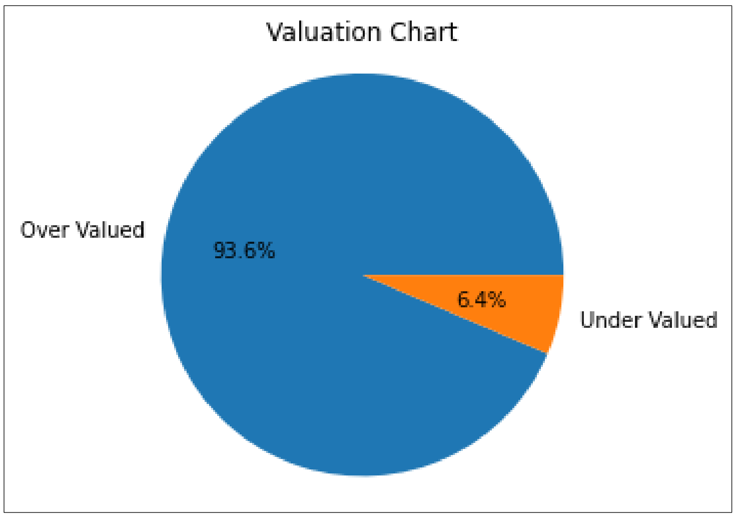
Model development:
Model initialized with features and labels, trained on training data and fitted on test data.

Model evaluation:
Regression evaluator was used. RMSE on test data was 0.91 and Adj R^2 was 0.987. Below is the equation of the model to estimate value of a player.

2.171 + 0.475*Release Clause(M) + 0.005*wage(K) + 0.076*Overall - 0.012*Potential + 0.273*International Reputation + 0.002*Reactions - 0.001*Composure + 0.0001*Special + 0.004*Vision - 0.002*ShortPassing - 0.004*Contract Valid Until

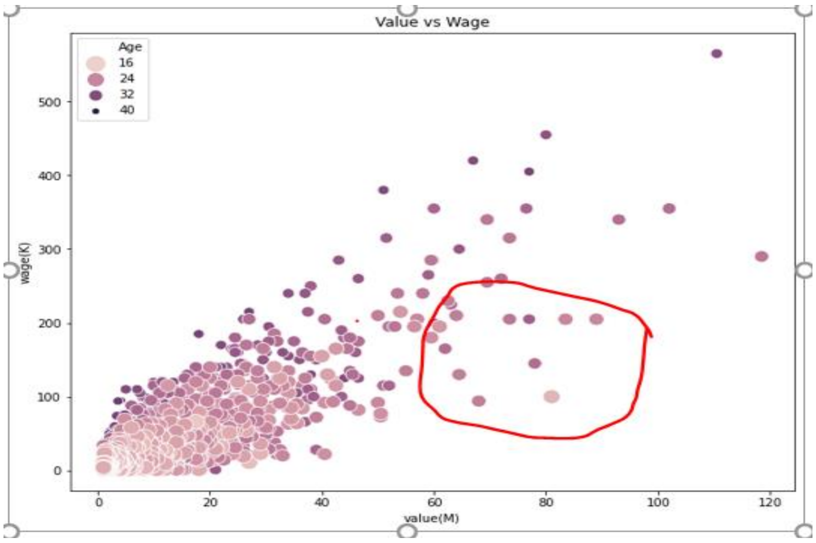
Due to unavailability of ANOVA table and residual analysis tool, insignificant variables were not dropped. But Special, contract valid until variables seems to be insignificant variables causing multicollinearity.

Considering model valuation is accurate than market value, we considered residuals as undervalued/ overvalued. Based on this, we can infer that 6.4% players are undervalued, and GM can buy undervalued players and transferring them to another club later for higher value later.

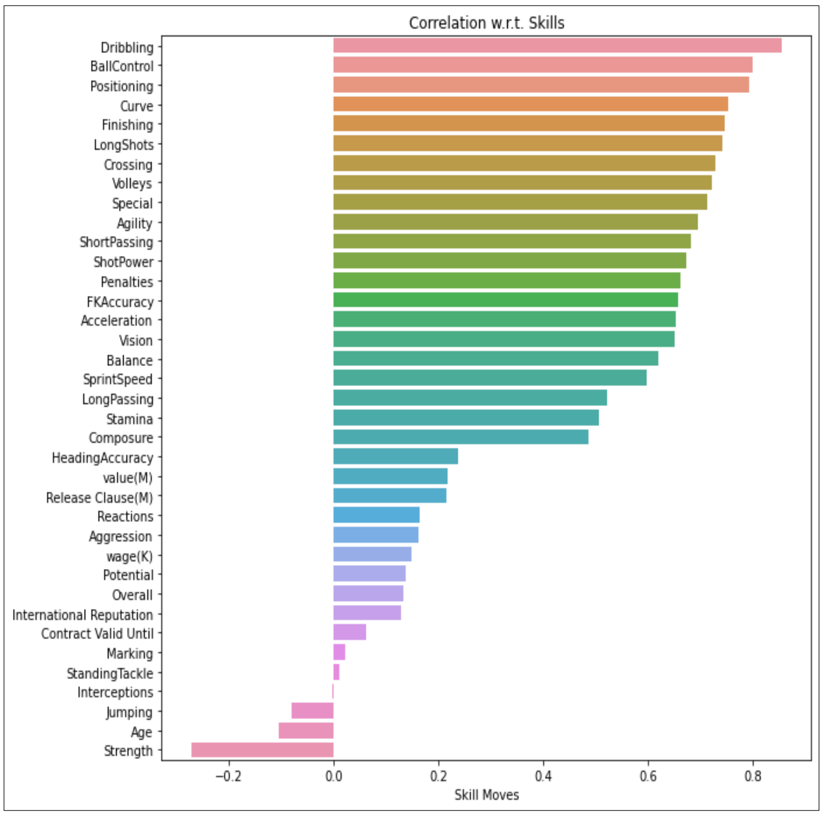


Scatter plot between wage and value:

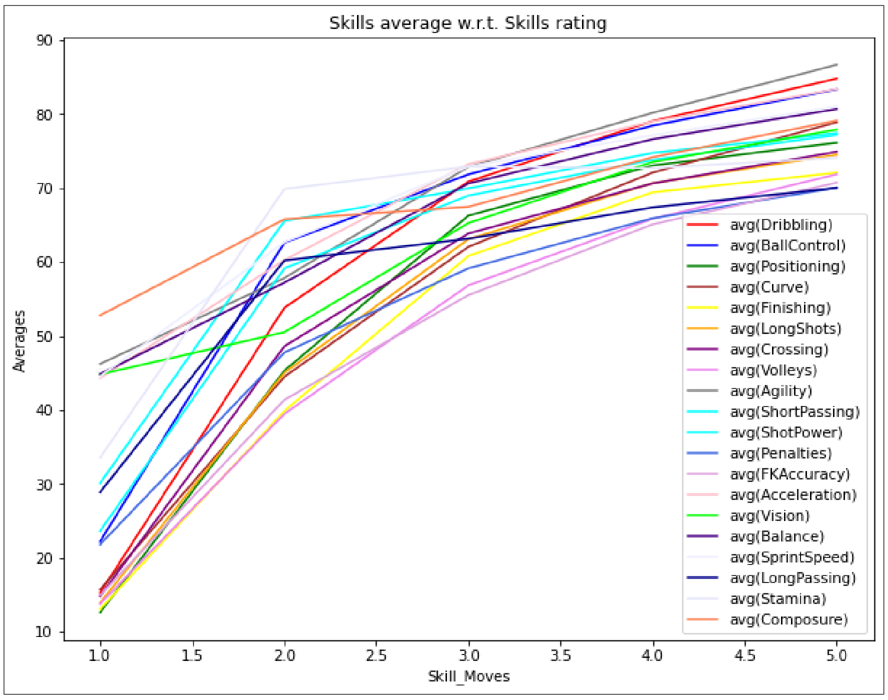
Below is the scatter plot with value as X and wage as Y with hue on Age. Wage and value are concentrated between wage=200K and value=40M. If GM, focuses on players falling in category circled in red in below plot, then it might be profitable deal. These are players with less age, so more margin for potential growth, also their wage is less but value is more.



From coaching perspective, all skillsets are measured in one variable i.e. Skills score. Thus, we can use this score to upskill and allocate position to a player. We can see positive correlation of skills and Skills score.



Correlation w.r.t. Skills
Dribbling
BallControl
Positioning
Curve
Finishing
LongShots
Crossing
Volleys
Special
Agility
ShortPassing
ShotPower
Penalties
FKAccuracy
Acceleration
Vision
Balance
SprintSpeed
LongPassing
Stamina
Composure
HeadingAccuracy
value(M)
Release Clause(M)
Reactions
Aggression
wage(K)
Potential
Overall
International Reputation
Contract Valid Until
Marking
StandingTackle
Interceptions
Jumping
Age
Strength
−0.2
0.0
0.2
0.4
0.6
0.8
Skill Moves



Skills average w.r.t. Skills rating
Averages
Skill_Moves
avg(Dribbling)
avg(BallControl)
avg(Positioning)
avg(Curve)
avg(Finishing)
avg(LongShots)
avg(Crossing)
avg(Volleys)
avg(Agility)
avg(ShortPassing)
avg(ShotPower)
avg(Penalties)
avg(FKAccuracy)
avg(Acceleration)
avg(Vision)
avg(Balance)
avg(SprintSpeed)
avg(LongPassing)
avg(Stamina)
avg(Composure)

Section 5 - Reflection

We could not collaborate well initially, and later we realized that tasks 3.3 and section 4 are the toughest one. So, Yash worked on task 1 and 3.3 NB and LR models with documentation of section 5. Vijay worked on tasks 2, 3.1,3.2,3.3 MLP model, section 4 along with the documentation. In this assignment, we learned how to use Apache Spark with Python i.e. pyspark. Also, we understood the importance of planning, communication, and collaboration as team. Apart from models, we also learned real-life and interesting tasks on FIFA dataset which would convert data into information.

References:

Stack Overflow. 2020. Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Available at: <https://stackoverflow.com/> [Accessed 2 June 2020].

Towards Data Science. 2020. *Towards Data Science*. [online] Available at: <https://towardsdatascience.com/> [Accessed 2 June 2020].

Spark.apache.org. 2020. *Pyspark.Sql Module — Pyspark 2.4.5 Documentation*. [online] Available at: <https://spark.apache.org/docs/latest/api/python/pyspark.sql.html> [Accessed 2 June 2020].

Runawayhorse001.github.io. 2020. *9. Regression — Learning Apache Spark With Python Documentation*. [online] Available at: <https://runawayhorse001.github.io/LearningApacheSpark/regression.html#id14> [Accessed 2 June 2020].