

In [1]:

```
1 import findspark
2 findspark.init()
3 findspark.find()
4
5 import pyspark
6 from pyspark import SparkContext, SparkConf, SQLContext
7 from pyspark.sql import SparkSession
8
9 from pyspark.sql import SparkSession
10
11 spark = SparkSession.builder \
12     .master("local[*]") \
13     .appName("GenericAppName") \
14     .getOrCreate()
15 sc=spark.sparkContext
16 sqlContext = SQLContext(sc)
```

C:\temp\Hadoop\spark-3.3.0-bin-hadoop3\python\pyspark\sql\context.py:112: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.  
warnings.warn(

## Task I

In [2]:

```
1 # Ingest data 2015-2022
2 from pyspark import SparkFiles
3
4 players_15 = spark.read.csv('FIFA_DATA/players_15.csv',header=True, inferSchema = True)
5 players_16 = spark.read.csv('FIFA_DATA/players_16.csv',header=True, inferSchema = True)
6 players_17 = spark.read.csv('FIFA_DATA/players_17.csv',header=True, inferSchema = True)
7 players_18 = spark.read.csv('FIFA_DATA/players_18.csv',header=True, inferSchema = True)
8 players_19 = spark.read.csv('FIFA_DATA/players_19.csv',header=True, inferSchema = True)
9 players_20 = spark.read.csv('FIFA_DATA/players_20.csv',header=True, inferSchema = True)
10 players_21 = spark.read.csv('FIFA_DATA/players_21.csv',header=True, inferSchema = True)
11 players_22 = spark.read.csv('FIFA_DATA/players_22.csv',header=True, inferSchema = True)
12
```

```
In [3]: 1 print(players_15.columns[55])
```

movement\_agility

```
In [4]: 1 #Add new column for the year
2 from pyspark.sql.functions import lit
3
4 players_15 = players_15.withColumn('Year', lit(2015))
5 players_16 = players_16.withColumn('Year', lit(2016))
6 players_17 = players_17.withColumn('Year', lit(2017))
7 players_18 = players_18.withColumn('Year', lit(2018))
8 players_19 = players_19.withColumn('Year', lit(2019))
9 players_20 = players_20.withColumn('Year', lit(2020))
10 players_21 = players_21.withColumn('Year', lit(2021))
11 players_22 = players_22.withColumn('Year', lit(2022))
12
```

```
In [5]: 1 #Ensure every record can be uniquely identified
```

```
2 #combine fifa id and year as index
3 from pyspark.sql import functions as sf
4
5 players_15 = players_15.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
6 players_16 = players_16.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
7 players_17 = players_17.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
8 players_18 = players_18.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
9 players_19 = players_19.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
10 players_20 = players_20.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
11 players_21 = players_21.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
12 players_22 = players_22.withColumn('data_ID', sf.concat(sf.col('sofifa_id'),sf.lit('_'), sf.col('Year'))))
```

```
In [6]: 1 # union the data to one dataset
```

```
2 data = players_15.union(players_16)
3 data = data.union(players_17)
4 data = data.union(players_18)
5 data = data.union(players_19)
6 data = data.union(players_20)
7 data = data.union(players_21)
8 data = data.union(players_22)
```

In [7]:

```
1 db_properties={}
2 #update your db username
3 db_properties['username']="postgres"
4 #update your db password
5 db_properties['password']="psql"
6 #make sure you got the right port number here
7 db_properties['url']="jdbc:postgresql://localhost:5432/postgres"
8 #make sure you had the Postgres JAR file in the right Location
9 db_properties['driver']="org.postgresql.Driver"
10 db_properties['table']="fifa.data"
11
12
13
14 data.write.format("jdbc")\
15 .mode("overwrite")\
16 .option("url", db_properties['url'])\
17 .option("dbtable", db_properties['table'])\
18 .option("user", db_properties['username'])\
19 .option("password", db_properties['password'])\
20 .option("Driver", db_properties['driver'])\
21 .save()
22
```

In [8]: 1 data.printSchema()

```
-- league_name: string (nullable = true)
-- league_level: integer (nullable = true)
-- club_position: string (nullable = true)
-- club_jersey_number: integer (nullable = true)
-- club_loaned_from: string (nullable = true)
-- club_joined: timestamp (nullable = true)
-- club_contract_valid_until: integer (nullable = true)
-- nationality_id: integer (nullable = true)
-- nationality_name: string (nullable = true)
-- nation_team_id: double (nullable = true)
-- nation_position: string (nullable = true)
-- nation_jersey_number: integer (nullable = true)
-- preferred_foot: string (nullable = true)
-- weak_foot: integer (nullable = true)
-- skill_moves: integer (nullable = true)
-- international_reputation: integer (nullable = true)
-- work_rate: string (nullable = true)
-- body_type: string (nullable = true)
-- real_face: string (nullable = true)
-- release_clause_eur: string (nullable = true)
```

In [9]:

```
1 data_read = sqlContext.read.format("jdbc")\
2     .option("url", db_properties['url'])\
3     .option("dbtable", db_properties['table'])\
4     .option("user", db_properties['username'])\
5     .option("password", db_properties['password'])\
6     .option("Driver", db_properties['driver'])\
7     .load()
8
9 data_read.show(1, vertical=True)
```

	2014
ldm	26+1
cdm	26+1
rdm	26+1
rwb	26+1
lb	26+1
lcb	27+1
cb	27+1
rcb	27+1
rb	26+1
gk	68+1
player_face_url	<a href="https://cdn.sofif...">https://cdn.sofif...</a> ( <a href="https://cdn.sofif...">https://cdn.sofif...</a> )
club_logo_url	<a href="https://cdn.sofif...">https://cdn.sofif...</a> ( <a href="https://cdn.sofif...">https://cdn.sofif...</a> )
club_flag_url	<a href="https://cdn.sofif...">https://cdn.sofif...</a> ( <a href="https://cdn.sofif...">https://cdn.sofif...</a> )
nation_logo_url	null
nation_flag_url	<a href="https://cdn.sofif...">https://cdn.sofif...</a> ( <a href="https://cdn.sofif...">https://cdn.sofif...</a> )
Year	2017
data_ID	190745_2017

only showing top 1 row

## Task II

Find X

In [84]:

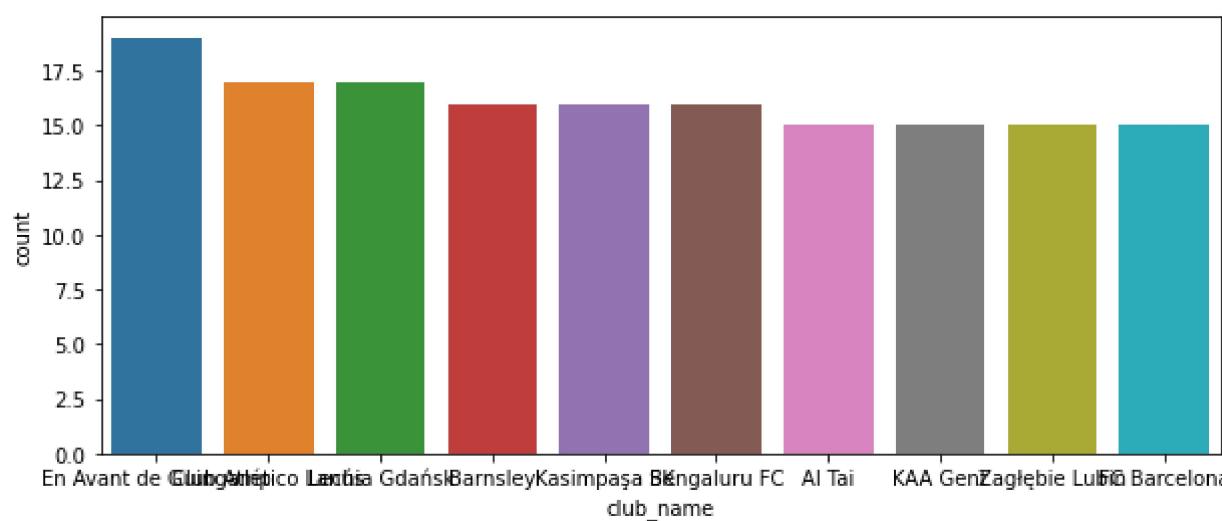
```
1 #In order to find the club with highest numbers of players
2 #use where to find the data in 2022
3 #use .agg to Compute aggregates when(data_read["club_contract_valid_until"]=="2023' and returns the result as a DataFrame
4 #group the data by club name and sort them by the count of data
5 def findX(x):
6     SORT_club = data_read.where(data_read['Year']=='2022').groupBy("club_name")
7     highest_number_club = SORT_club.agg(F.count(F.when(data_read["club_contract_valid_until"]=="2023", "club_name")).alias("cc"))
8     highest_number_club.show(x,vertical = True)
9 findX(5)
```



```
-RECORD 0-----
club_name | En Avant de Guingamp
count      | 19
-RECORD 1-----
club_name | Club Atlético Lanús
count      | 17
-RECORD 2-----
club_name | Lechia Gdańsk
count      | 17
-RECORD 3-----
club_name | Barnsley
count      | 16
-RECORD 4-----
club_name | Kasımpaşa SK
count      | 16
only showing top 5 rows
```

In [90]:

```
1 # we drew the barplot of the count to give a direct interpretation
2 import seaborn as sns
3 from IPython import display
4 import matplotlib.pyplot as plt
5
6 SORT_club = data_read.where(data_read['Year']=='2022').groupBy("club_name")
7 highest_number_club = SORT_club.agg(F.count(F.when(data_read["club_contract_valid_until"]==2023, "club_name")).alias("count"))
8 plt.figure( figsize = ( 10, 4 ) )
9 sns.barplot( x="club_name", y="count", data=highest_number_club[:10])
10 plt.show()
```





In [12]:

```
1 # the number of players older than 27 years old for each club
2 # we used where to find the players who are older than 27 and group the data by club_name, then we sort the data
3 # to find the club who has the highest number
4 # we sum the age for each club
5
6 import numpy as np
7 highest_number_over27_count = data_read.where(data_read['age']>27).groupBy("club_name").count().sort("count", ascending=False).
8 highest_number_over27_sum = data_read.where(data_read['age']>27).groupBy("club_name").sum('age').toPandas()
9 print(highest_number_over27_count)
10 print(highest_number_over27_sum)
11 average_over27 = highest_number_over27_sum['sum(age)'].values/highest_number_over27_count['count'].values
12 print(type(average_over27),average_over27)
```

	club_name	count
0	None	874
1	İstanbul Başakşehir FK	133
2	Jeonbuk Hyundai Motors	118
3	FC Lokomotiv Moscow	108
4	Crystal Palace	106
...	...	...
1007	Caracas Fútbol Club	3
1008	SC Freiburg II	2
1009	FC Helsingør	2
1010	FC Dordrecht	1
1011	Borussia Dortmund II	1

[1012 rows x 2 columns]

	club_name	sum(age)
0	Palermo	1105
1	CD Everton de Viña del Mar	1014
2	Shonan Bellmare	1676
3	Yeovil Town	847
4	Göztepe SK	1553
...	...	...
1007	Bury	1731
1008	Rotherham United	1945
1009	Bohemian FC	1269
1010	Middlesbrough	2064
1011	Como	219

[1012 rows x 2 columns]

```
<class 'numpy.ndarray'> [1.26430206e+00 7.62406015e+00 1.42033898e+01 ... 6.34500000e+02
2.06400000e+03 2.19000000e+02]
```

In [92]:

```
1 # we calculated the average and found the highest average
2 def findY(y):
3     highest_number_over27_avg = data_read.where(data_read['age']>27).groupBy("club_name").avg('age').sort("avg(age)",ascending=True)
4     highest_number_over27_avg.show(y,vertical =True)
5 findY(5)
```

-RECORD 0-----  
club\_name | Yokohama FC  
avg(age) | 34.7037037037037  
-RECORD 1-----  
club\_name | Wexford Youths  
avg(age) | 34.0  
-RECORD 2-----  
club\_name | Zamora Fútbol Club  
avg(age) | 33.857142857142854  
-RECORD 3-----  
club\_name | Centro Atlético F...  
avg(age) | 33.6  
-RECORD 4-----  
club\_name | CF Fuenlabrada  
avg(age) | 33.54545454545455  
only showing top 5 rows

In [93]:

```
1 top1_highest_number_over27_avg = data_read.where(data_read['age']>27).groupBy("club_name").avg('age').sort("avg(age)",ascending=True)
2 print("club with highest numbers over 27:",top1_highest_number_over27_avg['club_name'].values)
3 findY(1)
```

club with highest numbers over 27: ['Yokohama FC']  
-RECORD 0-----  
club\_name | Yokohama FC  
avg(age) | 34.7037037037037  
only showing top 1 row

```
In [15]: 1 # most frequent nation_positions for each year
2 # we also used where to filter the year, and group them by nation_position, count the number of each position and sort
3 # them to search the highest one.
4 most_frequent_nation_2015 = data_read.where(data_read['Year']==2015).groupBy("nation_position").count().sort("count",ascending)
5 print("Most frequent nation position 2015:",most_frequent_nation_2015['nation_position'].values)
6 most_frequent_nation_2016 = data_read.where(data_read['Year']==2016).groupBy("nation_position").count().sort("count",ascending)
7 print("Most frequent nation position 2016:",most_frequent_nation_2016['nation_position'].values)
8 most_frequent_nation_2017 = data_read.where(data_read['Year']==2017).groupBy("nation_position").count().sort("count",ascending)
9 print("Most frequent nation position 2017:",most_frequent_nation_2017['nation_position'].values)
10 most_frequent_nation_2018 = data_read.where(data_read['Year']==2018).groupBy("nation_position").count().sort("count",ascending)
11 print("Most frequent nation position 2018:",most_frequent_nation_2018['nation_position'].values)
12 most_frequent_nation_2019 = data_read.where(data_read['Year']==2019).groupBy("nation_position").count().sort("count",ascending)
13 print("Most frequent nation position 2019:",most_frequent_nation_2019['nation_position'].values)
14 most_frequent_nation_2020 = data_read.where(data_read['Year']==2020).groupBy("nation_position").count().sort("count",ascending)
15 print("Most frequent nation position 2020:",most_frequent_nation_2020['nation_position'].values)
16 most_frequent_nation_2021 = data_read.where(data_read['Year']==2021).groupBy("nation_position").count().sort("count",ascending)
17 print("Most frequent nation position 2021:",most_frequent_nation_2021['nation_position'].values)
18 most_frequent_nation_2022 = data_read.where(data_read['Year']==2022).groupBy("nation_position").count().sort("count",ascending)
19 print("Most frequent nation position 2022:",most_frequent_nation_2022['nation_position'].values)
```

```
Most frequent nation position 2015: ['SUB']
Most frequent nation position 2016: ['SUB']
Most frequent nation position 2017: ['SUB']
Most frequent nation position 2018: ['SUB']
Most frequent nation position 2019: ['SUB']
Most frequent nation position 2020: ['SUB']
Most frequent nation position 2021: ['SUB']
Most frequent nation position 2022: ['SUB']
```

```
In [16]: 1 # most_frequent_nation_2015 = data_read.where(data_read['Year']==2015).groupBy("nation_position").count().sort("count",ascending)
2 # print(most_frequent_nation_2015)
```

## Task III

### Data Engineering and cleaning

```
In [17]: 1 # Since we only use the skillsets of the players, we dropped all the unnecessary columns  
2 drop_list = ['player_position', 'nationality_name', 'work_rate', 'body_type', 'real_face', 'data_ID', 'short_name', 'long_name', 'dob'  
3             , 'club_loaned_from', 'club_joined', 'club_contract_valid_until',  
4             , 'nationality_id', 'nation_team_id', 'nation_position', 'nation_jersey_number',  
5             , 'release_clause_eur', 'player_tags', 'player_traits', 'player_face_url',  
6             , 'club_logo_url', 'club_flag_url', 'nation_logo_url', 'nation_flag_url',  
7             , 'player_url', 'mentality_composure', 'goalkeeping_speed', 'club_position', 'league_level', 'league_name']
```



```
In [18]: 1 print(len(drop_list))
```

33

```
In [19]: 1 df_read = data_read.drop('player_positions', 'nationality_name', 'work_rate', 'body_type', 'real_face', 'data_ID', 'short_name', 'lon  
2             , 'club_loaned_from', 'club_joined', 'club_contract_valid_until',  
3             , 'nationality_id', 'nation_team_id', 'nation_position', 'nation_jersey_number',  
4             , 'release_clause_eur', 'player_tags', 'player_traits', 'player_face_url',  
5             , 'club_logo_url', 'club_flag_url', 'nation_logo_url', 'nation_flag_url',  
6             , 'player_url', 'mentality_composure', 'goalkeeping_speed', 'club_position', 'league_level', 'league_name']
```



```
In [20]: 1 print(len(data_read.columns))  
2 print(len(df_read.columns))
```

112

79

```
In [21]: 1 df_read.printSchema()
```

```
root
|-- sofifa_id: integer (nullable = true)
|-- overall: integer (nullable = true)
|-- potential: integer (nullable = true)
|-- value_eur: double (nullable = true)
|-- wage_eur: double (nullable = true)
|-- age: integer (nullable = true)
|-- height_cm: integer (nullable = true)
|-- weight_kg: integer (nullable = true)
|-- preferred_foot: string (nullable = true)
|-- weak_foot: integer (nullable = true)
|-- skill_moves: integer (nullable = true)
|-- international_reputation: integer (nullable = true)
|-- pace: integer (nullable = true)
|-- shooting: integer (nullable = true)
|-- passing: integer (nullable = true)
|-- dribbling: integer (nullable = true)
|-- defending: integer (nullable = true)
|-- physic: integer (nullable = true)
```

```
In [22]: 1
2 # from pyspark.sql.functions import *
3
4 # # Notice I dropped isPenalty and isSTPlay
5 # null_counts_plays_df = df_read.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) \
6 #                                         for c in df_read.columns])
7
8 # null_counts_plays_df.show(truncate=False, vertical=True)
```

**impute 'pace','shooting','passing','dribbling','defending','physic'**

```
In [23]: 1 # we checked the missing values of the whole dataset and we found out that the columns with
2 # larger amount of missing value: 'pace', 'shooting', 'passing', 'dribbling', 'defending', 'physic'
3 # Because there were many missing value in the data, we want to impute the missing value with the median value
```

In [24]:

```

1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['pace']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na = df_read.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed = imputer.fit(df_with_filled_na).transform(df_with_filled_na)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced = df_imputed.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed = df_imputed_enhanced.withColumnRenamed("pace_imputed", "pace")

```

In [25]:

```

1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['shooting']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na2 = df_fully_imputed.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed2 = imputer.fit(df_with_filled_na2).transform(df_with_filled_na2)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced2 = df_imputed2.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed2 = df_imputed_enhanced2.withColumnRenamed("shooting_imputed", "shooting")

```

In [26]:

```
1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['passing']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na3 = df_fully_imputed2.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed3 = imputer.fit(df_with_filled_na3).transform(df_with_filled_na3)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced3 = df_imputed3.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed3 = df_imputed_enhanced3.withColumnRenamed("passing_imputed","passing")
```

In [27]:

```
1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['dribbling']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na4 = df_fully_imputed3.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed4 = imputer.fit(df_with_filled_na4).transform(df_with_filled_na4)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced4 = df_imputed4.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed4 = df_imputed_enhanced4.withColumnRenamed("dribbling_imputed","dribbling")
```

In [28]:

```

1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['defending']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na5 = df_fully_imputed4.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed5 = imputer.fit(df_with_filled_na5).transform(df_with_filled_na5)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced5 = df_imputed5.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed5 = df_imputed_enhanced5.withColumnRenamed("defending_imputed","defending")

```

In [29]:

```

1 from pyspark.ml.feature import Imputer
2 columns_to_be_imputed = ['physic']
3 value_not_in_dataset = -200
4
5 # Replace None/Missing Value with a value that can't be present in the dataset.
6 df_with_filled_na6 = df_fully_imputed5.fillna(-200, columns_to_be_imputed)
7
8 #Create new columns with imputed values. New columns will be suffixed with "_imputed"
9 imputer = Imputer (
10     inputCols=columns_to_be_imputed,
11     outputCols=["{}_imputed".format(c) for c in columns_to_be_imputed])\
12     .setStrategy("median").setMissingValue(value_not_in_dataset)
13
14 df_imputed6 = imputer.fit(df_with_filled_na6).transform(df_with_filled_na6)
15 # we will drop the old column without imputation. We have only one column to be imputed
16 df_imputed_enhanced6 = df_imputed6.drop(columns_to_be_imputed[0])
17
18 # We will rename our newly imputed column with the correct name
19 df_fully_imputed6 = df_imputed_enhanced6.withColumnRenamed("physic_imputed","physic")

```

```
In [30]: 1 from pyspark.sql.functions import *
2
3 # # Notice I dropped isPenalty and isSTPlay
4 # null_counts_plays_df = df_fully_imputed6.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) \
5 #                                     for c in df_read.columns])
6
7 # null_counts_plays_df.show(truncate=False, vertical=True)
```

## drop null

```
In [31]: 1 # we checked the missing value again and found out the rows with missing value
2 # Then we drop the rows
3 # Reason: The number of row was small compared to the entire dataset, so we just drop them
```

```
In [32]: 1 df_read = df_fully_imputed6.dropna()
```

```
In [33]: 1 null_counts_plays_df = df_read.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) \
2                                         for c in df_read.columns])
3
4 null_counts_plays_df.show(truncate=False, vertical=True)
```

lwb	0
ldm	0
cdm	0
rdm	0
rwb	0
lb	0
lcb	0
cb	0
rcb	0
rb	0
gk	0
Year	0
pace	0
shooting	0
passing	0
dribbling	0
defending	0
physic	0

## Data preprocessing

### label to binary

```
In [34]: 1 # Because there are only two values in the 'preferred_foot' column, we convert the value to 0 and 1
2 label_to_binary = udf(lambda name: 0.0 if name == 'Left' else 1.0)
3 df_read_bi = df_read.withColumn('preferred_foot', label_to_binary(col('preferred_foot')))
```

```
In [35]: 1 # df_read_bi.show(vertical = True)
```

### string to int

```
In [36]: 1 # We found the format "XX+X" in some columns and we calculated them and converted them to int
2 def addall(string):
3     if len(string)== 4:
4         string = int(string[0:2])+int(string[-1])
5     elif len(string) ==2:
6         string = int(string)
7     elif len(string)== 6:
8         string = int(string[-2:])
9     return string
```

```
In [37]: 1 from pyspark.sql.functions import udf
2 from pyspark.sql.types import IntegerType
```

```
In [38]: 1 addallUDF = udf(lambda i:addall(i), IntegerType())
```

```
In [39]: 1 str_to_int_list = ['ls','st','rs','lw','lf','cf','rf','rw','lam','cam','ram','lm','lcm','cm','rcm','rm','lwb',
2                      'ldm','cdm','rdm','rwb','lb','lcb','cb','rcb','rb','gk']
```

In [40]:

```
1 df_read_int = df_read_int.withColumn('ls', addallUDF(col('ls')))
2 df_read_int = df_read_int.withColumn('st', addallUDF(col('st')))
3 df_read_int = df_read_int.withColumn('rs', addallUDF(col('rs')))
4 df_read_int = df_read_int.withColumn('lw', addallUDF(col('lw')))
5 df_read_int = df_read_int.withColumn('lf', addallUDF(col('lf')))
6 df_read_int = df_read_int.withColumn('cf', addallUDF(col('cf')))
7 df_read_int = df_read_int.withColumn('rf', addallUDF(col('rf')))
8 df_read_int = df_read_int.withColumn('rw', addallUDF(col('rw')))
9 df_read_int = df_read_int.withColumn('lam', addallUDF(col('lam')))
10 df_read_int = df_read_int.withColumn('cam', addallUDF(col('cam')))
11 df_read_int = df_read_int.withColumn('ram', addallUDF(col('ram')))
12 df_read_int = df_read_int.withColumn('lm', addallUDF(col('lm')))
13 df_read_int = df_read_int.withColumn('lcm', addallUDF(col('lcm')))
14 df_read_int = df_read_int.withColumn('cm', addallUDF(col('cm')))
15 df_read_int = df_read_int.withColumn('rcm', addallUDF(col('rcm')))
16 df_read_int = df_read_int.withColumn('rm', addallUDF(col('rm')))
17 df_read_int = df_read_int.withColumn('lwb', addallUDF(col('lwb')))
18 df_read_int = df_read_int.withColumn('ldm', addallUDF(col('ldm')))
19 df_read_int = df_read_int.withColumn('cdm', addallUDF(col('cdm')))
20 df_read_int = df_read_int.withColumn('rdm', addallUDF(col('rdm')))
21 df_read_int = df_read_int.withColumn('rwb', addallUDF(col('rwb')))
22 df_read_int = df_read_int.withColumn('lb', addallUDF(col('lb')))
23 df_read_int = df_read_int.withColumn('lcb', addallUDF(col('lcb')))
24 df_read_int = df_read_int.withColumn('cb', addallUDF(col('cb')))
25 df_read_int = df_read_int.withColumn('rcb', addallUDF(col('rcb')))
26 df_read_int = df_read_int.withColumn('rb', addallUDF(col('rb')))
27 df_read_int = df_read_int.withColumn('gk', addallUDF(col('gk')))
```

cast data type

```
In [41]: 1 print(df_read_int.columns)
```

```
['sofifa_id', 'overall', 'potential', 'value_eur', 'wage_eur', 'age', 'height_cm', 'weight_kg', 'preferred_foot', 'weak_foot', 'skill_moves', 'international_reputation', 'attacking_crossing', 'attacking_finishing', 'attacking_heading_accuracy', 'attacking_short_passing', 'attacking_volleys', 'skill_dribbling', 'skill_curve', 'skill_fk_accuracy', 'skill_long_passing', 'skill_ball_control', 'movement_acceleration', 'movement_sprint_speed', 'movement_agility', 'movement_reactions', 'movement_balance', 'power_shot_power', 'power_jumping', 'power_stamina', 'power_strength', 'power_long_shots', 'mentality_aggression', 'mentality_interception', 'mentality_positioning', 'mentality_vision', 'mentality_penalties', 'defending_marking_awareness', 'defending_standing_tackle', 'defending_sliding_tackle', 'goalkeeping_diving', 'goalkeeping_handling', 'goalkeeping_kicking', 'goalkeeping_positioning', 'goalkeeping_reflexes', 'ls', 'st', 'rs', 'lw', 'lf', 'cf', 'rf', 'rw', 'lam', 'cam', 'ram', 'lm', 'lcm', 'cm', 'rcm', 'rm', 'lw_b', 'ldm', 'cdm', 'rdm', 'rwb', 'lb', 'lcb', 'cb', 'rcb', 'rb', 'gk', 'Year', 'pace', 'shooting', 'passing', 'dribbling', 'defending', 'physic']
```

```
In [42]: 1 # we changed the type of data to DoubleType
```

```
In [43]: 1 from pyspark.sql.types import DoubleType
2 for i in df_read_int.columns:
3     df_read_int = df_read_int.withColumn(i, df_read_int[i].cast(DoubleType()))
```

```
In [44]: 1 df_read_int.printSchema()
```

```
root
|-- sofifa_id: double (nullable = true)
|-- overall: double (nullable = true)
|-- potential: double (nullable = true)
|-- value_eur: double (nullable = true)
|-- wage_eur: double (nullable = true)
|-- age: double (nullable = true)
|-- height_cm: double (nullable = true)
|-- weight_kg: double (nullable = true)
|-- preferred_foot: double (nullable = true)
|-- weak_foot: double (nullable = true)
|-- skill_moves: double (nullable = true)
|-- international_reputation: double (nullable = true)
|-- attacking_crossing: double (nullable = true)
|-- attacking_finishing: double (nullable = true)
|-- attacking_heading_accuracy: double (nullable = true)
|-- attacking_short_passing: double (nullable = true)
|-- attacking_volleys: double (nullable = true)
|-- skill_dribbling: double (nullable = true)
```

```
In [45]: 1 df_read_features = df_read_int.withColumn('overall_new', col('overall')).drop('overall')
```

```
In [46]: 1 df_read_int.show(5,vertical = True)
```

```
-RECORD 0-----  
sofifa_id          | 190745.0  
overall            | 69.0  
potential           | 74.0  
value_eur           | 1100000.0  
wage_eur            | 10000.0  
age                 | 25.0  
height_cm           | 191.0  
weight_kg           | 80.0  
preferred_foot      | 1.0  
weak_foot           | 2.0  
skill_moves          | 1.0  
international_reputation | 1.0  
attacking_crossing   | 12.0  
attacking_finishing    | 20.0  
attacking_heading_accuracy | 12.0  
attacking_short_passing  | 22.0  
attacking_volleys     | 20.0  
skill_dribbling       | 18.0  
...                | ...
```

```
In [47]: 1 feature_cols = df_read_int.columns  
2 del feature_cols[2]
```

```
In [48]: 1 from pyspark.sql import Row  
2 from pyspark.ml.linalg import Vectors
```

```
In [49]: 1 len(df_read_features.columns)
```

```
Out[49]: 79
```

In [50]:

```
1 null_counts_plays_df_2 = df_read_features.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) \
2                                                 for c in df_read_features.columns])
3
4
5 null_counts_plays_df_2.show(truncate=False, vertical=True)
```

lwb	0
ldm	0
cdm	0
rdm	0
rwb	0
lb	0
lcb	1
cb	1
rcb	1
rb	0
gk	451
Year	0
pace	0
shooting	0
passing	0
dribbling	0
defending	0
physic	0
overall_new	0

In [51]:

```
1 df_read_features = df_read_features.dropna()
```

In [52]:

```
1 # we checked the data again to ensure there is no missing value in the dataset
2 null_counts_plays_df_2 = df_read_features.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) \
3                                                 for c in df_read_features.columns])
4
5 null_counts_plays_df_2.show(truncate=False, vertical=True)
```

Column	Count
ldm	0
cdm	0
rdm	0
rwb	0
lb	0
lcb	0
cb	0
rcb	0
rb	0
gk	0
Year	0
pace	0
shooting	0
passing	0
dribbling	0
defending	0
physic	0
overall_new	0

In [53]:

```
1 # we assembled the features in the data to a vector so that it can be recognized by pyspark models
```

In [54]:

```
1 def transData(data):
2     return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
3         toDF(['output','features'])
4
5 data= transData(df_read_features)
6 data.show()
```

```
+-----+
|output|      features|
+-----+
| 69.0|[190745.0,74.0,11...|
| 69.0|[190780.0,73.0,10...|
| 69.0|[190783.0,69.0,65...|
| 69.0|[190792.0,72.0,10...|
| 69.0|[190883.0,69.0,77...|
| 69.0|[191079.0,73.0,11...|
| 69.0|[191096.0,70.0,95...|
| 69.0|[191135.0,72.0,12...|
| 69.0|[191222.0,71.0,10...|
| 69.0|[191252.0,69.0,10...|
| 69.0|[191443.0,69.0,32...|
| 69.0|[191548.0,69.0,65...|
| 69.0|[191679.0,69.0,72...|
| 69.0|[191834.0,69.0,82...|
| 69.0|[192009.0,76.0,12...|
| 69.0|[192179.0,69.0,87...|
| 69.0|[192297.0,69.0,87...|
| 69.0|[192449.0,72.0,12...|
| 69.0|[192457.0,70.0,10...|
| 69.0|[192473.0,69.0,11...|
+-----+
only showing top 20 rows
```

In [55]:

```
1 # we normalized the features by using StandardScaler
```

In [56]:

```
1 from pyspark.ml.feature import StandardScaler
2 Scalerizer=StandardScaler().setInputCol("features").setOutputCol("norm_features")
3 data_norm = Scalerizer.fit(data).transform(data)
```

```
In [57]: 1 data_norm = data_norm.drop('features')
2 data_norm = data_norm.withColumn('features', col('norm_features')).drop('norm_features')
```

```
In [58]: 1 # data_norm.select('features').show(vertical = False)
```

```
In [59]: 1 # (trainingData, testData) = data_norm.randomSplit([0.8, 0.2])
2
3 # then we splited the data in to training and test dataset
4
5 (trainingData, testData) = data_norm.randomSplit([0.8, 0.2])
```

```
In [60]: 1 trainingData.describe().show()
```

summary	output
count	111721
mean	65.68281701739154
stddev	7.07615617760127
min	40.0
max	94.0

```
In [61]: 1 print("Training Dataset Count: " + str(trainingData.count()))
2 print("Test Dataset Count: " + str(testData.count()))
```

Training Dataset Count: 111721  
Test Dataset Count: 28009

## Pyspark

### Random Forest

In [63]:

```
1 from pyspark.ml.regression import RandomForestRegressor
2 from pyspark.ml.feature import VectorIndexer
3 from pyspark.ml.evaluation import RegressionEvaluator
4 from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
5
6 rf = RandomForestRegressor(featuresCol = 'features', labelCol = 'output')
7
8 # Create ParamGrid for Cross Validation
9 rf_paramGrid = (ParamGridBuilder()
10             .addGrid(rf.numTrees, [50,100,150,200,300,])
11             .addGrid(rf.maxDepth, [3,4,5])
12             .build())
13
14 evaluator = RegressionEvaluator(
15     labelCol="output", predictionCol="prediction", metricName="rmse")
16
17 rf_cv = CrossValidator(estimator=rf, estimatorParamMaps=rf_paramGrid,
18                         evaluator=evaluator, numFolds=5)
19
20 # Train model. This also runs the indexer.
21 model_rf = rf_cv.fit(trainingData)
22
23 # Make predictions.
24 predictions_rf = model_rf.transform(testData)
25 predictions_rf.show(5)
```

output	features	prediction
41.0	[6.40057623302199...]	53.669842725778224
42.0	[6.12293559300367...]	53.71640172811433
44.0	[6.45133304662317...]	53.65709730806286
44.0	[6.51719225878871...]	53.86573062028856
44.0	[6.53197579673080...]	53.65709730806286

only showing top 5 rows

In [65]:

```

1 evaluator = RegressionEvaluator(
2     labelCol="output", predictionCol="prediction", metricName="rmse")
3 rmse = evaluator.evaluate(predictions_rf)
4 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

```

Root Mean Squared Error (RMSE) on test data = 1.58698

## Tune the parameters one by one and see the impact of each parameter

In [60]:

```

1 from pyspark.ml.regression import RandomForestRegressor
2 from pyspark.ml.feature import VectorIndexer
3 from pyspark.ml.evaluation import RegressionEvaluator
4
5 rf3 = RandomForestRegressor(featuresCol = 'features', labelCol = 'output', numTrees=100, maxDepth=5)
6
7 # Train model. This also runs the indexer.
8 model3 = rf3.fit(trainingData)
9
10 # Make predictions.
11 predictions3 = model3.transform(testData)
12
13 predictions3.show(5)
14
15 evaluator3 = RegressionEvaluator(
16     labelCol="output", predictionCol="prediction", metricName="rmse")
17 rmse3 = evaluator3.evaluate(predictions3)
18 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse3)

```

output	features	prediction
41.0	[6.53690364271165...]	53.512072880945354
44.0	[6.41599749220878...]	53.512072880945354
44.0	[6.58719665904405...]	53.512072880945354
44.0	[6.60650222035666...]	53.76495696951452
45.0	[6.17380835592218...]	53.60100459946284

only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 1.60538

```
In [61]: 1 trainingData.show(5)
```

```
+-----+  
|output|      features|  
+-----+  
| 40.0|[6.30738196932836...|  
| 40.0|[6.46263810504961...|  
| 41.0|[6.40057623302213...|  
| 42.0|[6.12293559300381...|  
| 42.0|[6.22430428355967...|  
+-----+  
only showing top 5 rows
```

In [62]:

```
1 from pyspark.ml.regression import RandomForestRegressor
2 from pyspark.ml.feature import VectorIndexer
3 from pyspark.ml.evaluation import RegressionEvaluator
4
5 rf_1 = RandomForestRegressor(featuresCol = 'features', labelCol = 'output', numTrees=100, maxDepth=6)
6
7 # Train model. This also runs the indexer.
8 model_1 = rf_1.fit(trainingData)
9
10 # Make predictions.
11 predictions_1 = model_1.transform(testData)
12
13 predictions_1.show(5)
14
15 evaluator_1 = RegressionEvaluator(
16     labelCol="output", predictionCol="prediction", metricName="rmse")
17 rmse_1 = evaluator_1.evaluate(predictions_1)
18 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_1)
```

```
+-----+-----+
|output|      features|      prediction|
+-----+-----+
| 41.0|[6.53690364271165...|52.39016251553484|
| 44.0|[6.41599749220878...|52.39016251553484|
| 44.0|[6.58719665904405...|52.39016251553484|
| 44.0|[6.60650222035666...|52.77358279353466|
| 45.0|[6.17380835592218...|52.47513371165757|
+-----+-----+
only showing top 5 rows
```

Root Mean Squared Error (RMSE) on test data = 1.33804

## Linear Regression

In [66]:

```
1 from pyspark.ml.classification import LogisticRegression
2 from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
3 from pyspark.ml.evaluation import BinaryClassificationEvaluator
4
5 from sklearn.metrics import roc_curve
6 import pyspark.sql.functions as F
7 import pyspark.sql.types as T
8 import numpy
9 from matplotlib import pyplot as plt
10
11 from pyspark.ml.regression import RandomForestRegressor
12 from pyspark.ml.feature import VectorIndexer
13 from pyspark.ml.evaluation import RegressionEvaluator
14
15 from pyspark.sql.types import Row
16 from pyspark.ml.linalg import Vectors
17 from pyspark.ml.classification import MultilayerPerceptronClassifier
```

```
In [67]: 1 from pyspark.ml.regression import LinearRegression
2
3 lrg = LinearRegression(featuresCol = 'features', labelCol = 'output')
4
5 # Create ParamGrid for Cross Validation
6 lrg_paramGrid = ParamGridBuilder() \
7     .addGrid(lrg.maxIter, [10, 30, 200, 10000]) \
8     .addGrid(lrg.regParam, [0, 0.1, 2]) \
9     .build()
10
11 evaluator = RegressionEvaluator(
12     labelCol="output", predictionCol="prediction", metricName="rmse")
13
14 lrg_cv = CrossValidator(estimator=lr, estimatorParamMaps=lr_paramGrid,
15                         evaluator=evaluator, numFolds=5)
16
17 model_lrg = lrg_cv.fit(trainingData)
18
19 predictions_lrg = model_lrg.transform(testData)
20 predictions_lrg.show(5)
```

```
+-----+-----+
|output|      features|      prediction|
+-----+-----+
| 41.0|[6.40057623302199...|46.439497176678344|
| 42.0|[6.12293559300367...| 49.2242871853893|
| 44.0|[6.45133304662317...|46.83448039704086|
| 44.0|[6.51719225878871...|43.92618287833449|
| 44.0|[6.53197579673080...|41.81840415146371|
+-----+-----+
only showing top 5 rows
```

```
In [69]: 1 evaluator = RegressionEvaluator(
2     labelCol="output", predictionCol="prediction", metricName="rmse")
3 rmse5 = evaluator.evaluate(predictions_lrg)
4 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse5)
```

```
Root Mean Squared Error (RMSE) on test data = 1.82176
```

Tune one by one to find the impact for each hyperparameter

In [66]:

```
1 from pyspark.ml.regression import LinearRegression
2
3 # Define LinearRegression algorithm
4 lrg_1 = LinearRegression(featuresCol = 'features', labelCol = 'output', regParam=2.0)
5 # maxIter: int = 100
6 model_1 = lrg_1.fit(trainingData)
7 predictions_1 = model_1.transform(testData)
8
9 predictions_1.show(5)
10
11 evaluator_1 = RegressionEvaluator(
12     labelCol="output", predictionCol="prediction", metricName="rmse")
13 rmse_1 = evaluator_1.evaluate(predictions_1)
14 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_1)
```

output	features	prediction
41.0	[6.53690364271165...]	44.227559545991866
44.0	[6.41599749220878...]	47.81024214005528
44.0	[6.58719665904405...]	50.707025368422734
44.0	[6.60650222035666...]	48.697781658265555
45.0	[6.17380835592218...]	48.883788636232325

only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 2.14361

In [67]:

```
1 from pyspark.ml.regression import LinearRegression
2
3 # Define LinearRegression algorithm
4 lrg_2 = LinearRegression(featuresCol = 'features', labelCol = 'output', maxIter=10)
5 model_2 = lrg_2.fit(trainingData)
6 predictions_2 = model_2.transform(testData)
7
8 predictions_2.show(5)
9
10 evaluator_2 = RegressionEvaluator(
11     labelCol="output", predictionCol="prediction", metricName="rmse")
12 rmse_2 = evaluator_2.evaluate(predictions_2)
13 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_2)
```

output	features	prediction
41.0	[6.53690364271165...]	41.53811522459333
44.0	[6.41599749220878...]	45.92256921728185
44.0	[6.58719665904405...]	48.79175549801562
44.0	[6.60650222035666...]	44.533694117718795
45.0	[6.17380835592218...]	45.96323644678648

only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 1.86664

In [68]:

```
1 from pyspark.ml.regression import LinearRegression
2
3 # Define LinearRegression algorithm
4 lrg_3 = LinearRegression(featuresCol = 'features', labelCol = 'output', maxIter=30)
5 model_3 = lrg_3.fit(trainingData)
6 predictions_3 = model_3.transform(testData)
7
8 predictions_3.show(5)
9
10 evaluator_3 = RegressionEvaluator(
11     labelCol="output", predictionCol="prediction", metricName="rmse")
12 rmse_3 = evaluator_3.evaluate(predictions_3)
13 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_3)
```

```
+-----+-----+-----+
|output|    features|    prediction|
+-----+-----+-----+
| 41.0|[6.53690364271165...| 40.53397568304155|
| 44.0|[6.41599749220878...| 45.15551922724518|
| 44.0|[6.58719665904405...|48.524183250218016|
| 44.0|[6.60650222035666...| 44.02133666015817|
| 45.0|[6.17380835592218...| 45.43579327618232|
+-----+-----+-----+
only showing top 5 rows
```

Root Mean Squared Error (RMSE) on test data = 1.83285

In [69]:

```
1 from pyspark.ml.regression import LinearRegression
2
3 # Define LinearRegression algorithm
4 lrg_4 = LinearRegression(featuresCol = 'features', labelCol = 'output', maxIter=200)
5 model_4 = lrg_4.fit(trainingData)
6 predictions_4 = model_4.transform(testData)
7
8 predictions_4.show(5)
9
10 evaluator_4 = RegressionEvaluator(
11     labelCol="output", predictionCol="prediction", metricName="rmse")
12 rmse_4 = evaluator_4.evaluate(predictions_4)
13 print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_4)
```

```
+-----+-----+
|output|      features|      prediction|
+-----+-----+
| 41.0|[6.53690364271165...|40.73535289532248|
| 44.0|[6.41599749220878...|45.15357021804499|
| 44.0|[6.58719665904405...|48.66151100088803|
| 44.0|[6.60650222035666...| 44.129994473203|
| 45.0|[6.17380835592218...|45.64501921754493|
+-----+-----+
only showing top 5 rows
```

Root Mean Squared Error (RMSE) on test data = 1.82613

## Tensorflow

In [70]:

```
1 from pyspark.sql.types import FloatType
```

```
In [71]: 1 to_array = udf(lambda v: v.toArray().tolist(), ArrayType(FloatType()))
```

```
2
```

```
3 df_test = testData
```

```
4 df_validate,df_train = trainingData.randomSplit([0.5,0.5])
```

```
5
```

```
6 df_train_pandas = df_train.withColumn('features', to_array('features')).toPandas()
```

```
7 df_validate_pandas = df_validate.withColumn('features', to_array('features')).toPandas()
```

```
8 df_test_pandas = df_test.withColumn('features', to_array('features')).toPandas()
```

```
In [72]: 1 import tensorflow as tf
```

```
2 from tensorflow import keras
```

```
3
```

```
4 x_train = tf.constant(np.array(df_train_pandas['features'].values.tolist()))
```

```
5 y_train = tf.constant(np.array(df_train_pandas['output'].values.tolist()))
```

```
6
```

```
7 x_validate = tf.constant(np.array(df_validate_pandas['features'].values.tolist()))
```

```
8 y_validate = tf.constant(np.array(df_validate_pandas['output'].values.tolist()))
```

```
9
```

```
10
```

```
11 x_test = tf.constant(np.array(df_test_pandas['features'].values.tolist()))
```

```
12 y_test = tf.constant(np.array(df_test_pandas['output'].values.tolist()))
```

```
In [73]: 1 print(x_train)
```

```
2 print(y_train)
```

```
tf.Tensor([[6.30738211e+00 7.97132349e+00 2.59242672e-03 ... 3.16481352e+00
```

```
3.29432011e+00 7.40527344e+00]
```

```
[6.22430420e+00 8.29017639e+00 3.45656904e-03 ... 6.43171787e+00
```

```
3.48437715e+00 7.18747139e+00]
```

```
[6.40443134e+00 8.60902977e+00 3.45656904e-03 ... 6.43171787e+00
```

```
3.48437715e+00 7.18747139e+00]
```

```
...
```

```
[6.02965415e-01 1.49860888e+01 1.50360746e+01 ... 9.29025936e+00
```

```
2.09062624e+00 8.71208668e+00]
```

```
[6.02965415e-01 1.49860888e+01 1.65051174e+01 ... 9.18816853e+00
```

```
2.09062624e+00 8.71208668e+00]
```

```
[4.58066463e+00 1.51455145e+01 1.91839581e+01 ... 9.69862270e+00
```

```
1.52045548e+00 6.75186682e+00]], shape=(56176, 78), dtype=float64)
```

```
tf.Tensor([40. 42. 42. ... 94. 94. 94.], shape=(56176,), dtype=float64)
```

# Neural networks

```
In [74]: 1 model_nn = keras.Sequential( [keras.layers.Dense(78,activation='relu'),
2                                keras.layers.Dense(10,activation='relu'),
3                                keras.layers.Dense(10,activation='relu'),
4                                keras.layers.Dense(10,activation='relu') ,
5                                keras.layers.Dense(1)] )  
6
```

```
In [75]: 1 y_pred = model_nn(x_train)
2 model_nn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(56176, 78)	6162
dense_1 (Dense)	(56176, 10)	790
dense_2 (Dense)	(56176, 10)	110
dense_3 (Dense)	(56176, 10)	110
dense_4 (Dense)	(56176, 1)	11
=====		

Total params: 7,183

Trainable params: 7,183

Non-trainable params: 0

---

```
In [76]: 1 print(y_pred)
2 print(y_train)
```

```
tf.Tensor(
[[74.732086]
[74.12477 ]
[74.16112 ]
...
[73.94921 ]
[74.17415 ]
[74.34247 ]], shape=(56176, 1), dtype=float32)
tf.Tensor([40. 42. 42. ... 94. 94. 94.], shape=(56176,), dtype=float64)
```

```
In [77]: 1 mse = keras.losses.MeanSquaredError()
2
3 model_nn.compile(optimizer = 'adam',
4     loss=mse,
5     metrics=[mse])
6 model_nn.fit(x_train,y_train, epochs = 20,validation_data=(x_validate,y_validate),verbose = 2)
7
8 loss = mse(y_train, y_pred).numpy()
9 print(loss)
```

```
Epoch 15/20
1756/1756 - 4s - loss: 2.2276 - mean_squared_error: 2.2274 - val_loss: 2.0466 - val_mean_squared_error: 2.0521 - 4s/epoch -
3ms/step
Epoch 16/20
1756/1756 - 5s - loss: 2.1436 - mean_squared_error: 2.1435 - val_loss: 1.8553 - val_mean_squared_error: 1.8600 - 5s/epoch -
3ms/step
Epoch 17/20
1756/1756 - 5s - loss: 1.9760 - mean_squared_error: 1.9758 - val_loss: 1.9688 - val_mean_squared_error: 1.9746 - 5s/epoch -
3ms/step
Epoch 18/20
1756/1756 - 5s - loss: 1.9285 - mean_squared_error: 1.9283 - val_loss: 1.8612 - val_mean_squared_error: 1.8662 - 5s/epoch -
3ms/step
Epoch 19/20
1756/1756 - 4s - loss: 1.8439 - mean_squared_error: 1.8442 - val_loss: 1.6813 - val_mean_squared_error: 1.6880 - 4s/epoch -
3ms/step
Epoch 20/20
1756/1756 - 5s - loss: 1.8171 - mean_squared_error: 1.8171 - val_loss: 1.6386 - val_mean_squared_error: 1.6437 - 5s/epoch -
3ms/step
132.08322
```

```
In [78]: 1 model_nn.evaluate(x_test,y_test, verbose = 2)
```

```
869/869 - 1s - loss: 1.6491 - mean_squared_error: 1.6550 - 985ms/epoch - 1ms/step
```

```
Out[78]: [1.6490752696990967, 1.6549885272979736]
```

```
In [79]: 1 def cross_valiation(hyper,k,s_r,x,y,logdir):
2
3     def data_split():
4         for i in range(k):
5             idx=tf.range(df_train_pandas.shape[0])
6             splt_idx = int(s_r*df_train_pandas.shape[0])
7             idx = tf.random.shuffle(idx)
8             x_train, y_train = tf.gather(x, idx[:splt_idx]), tf.gather(y, idx[:splt_idx])
9             x_valid, y_valid = tf.gather(x, idx[splt_idx:]), tf.gather(y, idx[splt_idx:])
10            return x_train,y_train,x_valid,y_valid
11
12    model = keras.Sequential()
13    for _ in range(hparams[HP_DEPTH]):
14        model.add(keras.layers.Dense(hparams[HP_WIDTH],activation='relu'))
15        model.add(keras.layers.Dense(1))
16        model.compile(optimizer = 'adam',
17                      loss=keras.losses.MeanSquaredError(),
18                      metrics=[keras.losses.MeanSquaredError(name = 'MSE')])
19        x_train,y_train,x_valid,y_valid = data_split()
20        history = model.fit(x_train, y_train, epochs=5, verbose = 2,validation_data = (x_valid, y_valid),
21                            callbacks=[tf.keras.callbacks.TensorBoard(log_dir=logdir, histogram_freq=1)])
22        accuracy = np.mean(history.history[ "MSE"])
23        model.summary()
24    return accuracy
```

In [80]:

```
1 from tensorboard.plugins.hparams import api as hp
2
3 HP_WIDTH = hp.HParam('NN_width', hp.Discrete([10,20,30]))
4 HP_DEPTH = hp.HParam('NN_depth', hp.Discrete([3,4,5]))
5
6
7 with tf.summary.create_file_writer('logs14813/hparam_tuning').as_default():
8     hp.hparams_config(
9         hparams=[HP_WIDTH, HP_DEPTH],
10        metrics=[hp.Metric('MSE')])
11
12
```

In [81]:

```

1 import datetime
2 for hp_width in HP_WIDTH.domain.values:
3     for hp_depth in (HP_DEPTH.domain.values):
4         hparams = {
5             HP_WIDTH: hp_width,
6             HP_DEPTH: hp_depth,
7         }
8         run_name = f"run-WIDTH{int(hparams[HP_WIDTH])}-DEPTH{hparams[HP_DEPTH]}"
9         print('--- Starting trial: %s' % run_name)
10        print({h.name: hparams[h] for h in hparams})
11
12        run_dir = 'logs14813/hparam_tuning/' + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") + run_name
13        accuracy = cross_valiation(hparams, 10, 0.7, x_train, y_train, run_dir)
14
15        with tf.summary.create_file_writer(run_dir).as_default():
16            hp.hparams(hparams) # record the values used in this trial
17            tf.summary.scalar("MSE", accuracy, step=1)

```

Epoch 3/5

1229/1229 - 3s - loss: 4.6565 - MSE: 4.6563 - val\_loss: 5.5949 - val\_MSE: 5.5945 - 3s/epoch - 2ms/step

Epoch 4/5

1229/1229 - 2s - loss: 4.5016 - MSE: 4.5014 - val\_loss: 3.9336 - val\_MSE: 3.9329 - 2s/epoch - 2ms/step

Epoch 5/5

1229/1229 - 3s - loss: 4.5062 - MSE: 4.5065 - val\_loss: 4.5744 - val\_MSE: 4.5736 - 3s/epoch - 2ms/step

Model: "sequential\_9"

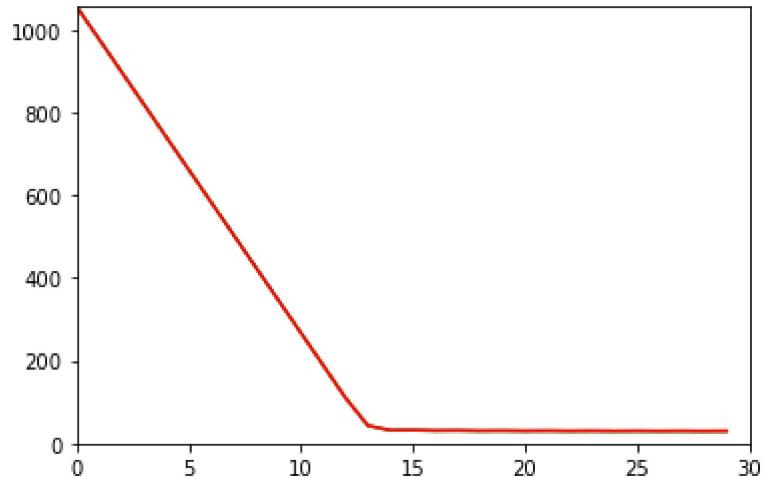
Layer (type)	Output Shape	Param #
<hr/>		
dense_67 (Dense)	(None, 30)	2370
dense_68 (Dense)	(None, 1)	31
dense_69 (Dense)	(None, 30)	60
dense_70 (Dense)	(None, 1)	31
dense_71 (Dense)	(None, 30)	60

## Linear Regression

In [82]:

```
1 tf.compat.v1.disable_eager_execution()
2 # tf.compat.v1.enable_eager_execution()
3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 import numpy as np
6
7 from sklearn.datasets import load_boston
8
9 def append_bias_reshape(x_train,y_train):
10    n_training_samples = x_train.shape[0]
11    n_dim = x_train.shape[1]
12    f = np.reshape(np.c_[np.ones(n_training_samples),x_train],[n_training_samples,n_dim+1])
13    l = np.reshape(y_train,[n_training_samples,1])
14
15    return f,l
16
17 if __name__ == '__main__':
18
19
20 #     x,y = read_boston_data()
21 #     norm_features = feature_normalize(x)
22 f,l = append_bias_reshape(x_train,y_train)
23 n_dim = f.shape[1]
24
25 rnd_indices = np.random.rand(len(f)) < 0.80
26
27 x_train = f[rnd_indices]
28 y_train = l[rnd_indices]
29 x_test = f[~rnd_indices]
30 y_test = l[~rnd_indices]
31
32
33
34 learning_rate = 0.0001
35 training_epochs = 30
36 cost_history = []
37 test_history = []
38
39 X = tf.compat.v1.placeholder(tf.float32,[None,n_dim])
40 Y = tf.compat.v1.placeholder(tf.float32,[None,1])
41 W = tf.Variable(tf.ones([n_dim,1]))
42
43 init = tf.compat.v1.initialize_all_variables()
44
45 y_ = tf.matmul(X,W)
```

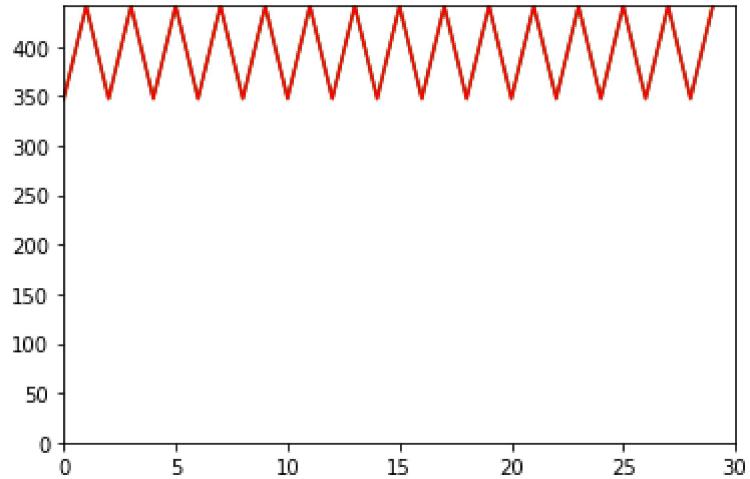
```
46     cost = tf.reduce_mean(tf.abs(y_-Y))
47     training_step = tf.compat.v1.train.GradientDescentOptimizer(learning_rate).minimize(cost)
48
49     sess = tf.compat.v1.Session()
50     sess.run(init)
51
52     for epoch in range(training_epochs):
53         sess.run(training_step,feed_dict={X:x_train,Y:y_train})
54         c = sess.run(cost,feed_dict={X:x_train,Y:y_train})
55         print(c)
56         t = sess.run(cost,feed_dict={X:x_test,Y:y_test})
57         print(t)
58         cost_history.append(c)
59         test_history.append(t)
60
61     plt.plot(range(len(test_history)),test_history,color = 'green')
62     plt.plot(range(len(cost_history)),cost_history,color = 'red')
63
64     plt.axis([0,training_epochs,0,np.max(cost_history)])
65     plt.show()
```



In [86]:

```
1 # tf.compat.v1.disable_eager_execution()
2 # tf.compat.v1.enable_eager_execution()
3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 import numpy as np
6
7 from sklearn.datasets import load_boston
8
9 def append_bias_reshape(x_train,y_train):
10    n_training_samples = x_train.shape[0]
11    n_dim = x_train.shape[1]
12    f = np.reshape(np.c_[np.ones(n_training_samples),x_train],[n_training_samples,n_dim+1])
13    l = np.reshape(y_train,[n_training_samples,1])
14
15    return f,l
16
17 if __name__ == '__main__':
18
19
20 #     x,y = read_boston_data()
21 #     norm_features = feature_normalize(x)
22 f,l = append_bias_reshape(x_train,y_train)
23 n_dim = f.shape[1]
24
25 rnd_indices = np.random.rand(len(f)) < 0.80
26
27 x_train = f[rnd_indices]
28 y_train = l[rnd_indices]
29 x_test = f[~rnd_indices]
30 y_test = l[~rnd_indices]
31
32
33
34 learning_rate = 0.001
35 training_epochs = 30
36 cost_history = []
37 test_history = []
38
39 X = tf.compat.v1.placeholder(tf.float32,[None,n_dim])
40 Y = tf.compat.v1.placeholder(tf.float32,[None,1])
41 W = tf.Variable(tf.ones([n_dim,1]))
42
43 init = tf.compat.v1.initialize_all_variables()
44
45 y_ = tf.matmul(X,W)
```

```
46     cost = tf.reduce_mean(tf.abs(y_-Y))
47     training_step = tf.compat.v1.train.GradientDescentOptimizer(learning_rate).minimize(cost)
48
49     sess = tf.compat.v1.Session()
50     sess.run(init)
51
52     for epoch in range(training_epochs):
53         sess.run(training_step,feed_dict={X:x_train,Y:y_train})
54         c = sess.run(cost,feed_dict={X:x_train,Y:y_train})
55         print(c)
56         t = sess.run(cost,feed_dict={X:x_test,Y:y_test})
57         print(t)
58         cost_history.append(c)
59         test_history.append(t)
60
61     plt.plot(range(len(test_history)),test_history,color = 'green')
62     plt.plot(range(len(cost_history)),cost_history,color = 'red')
63
64     plt.axis([0,training_epochs,0,np.max(cost_history)])
65     plt.show()
```



In [87]:

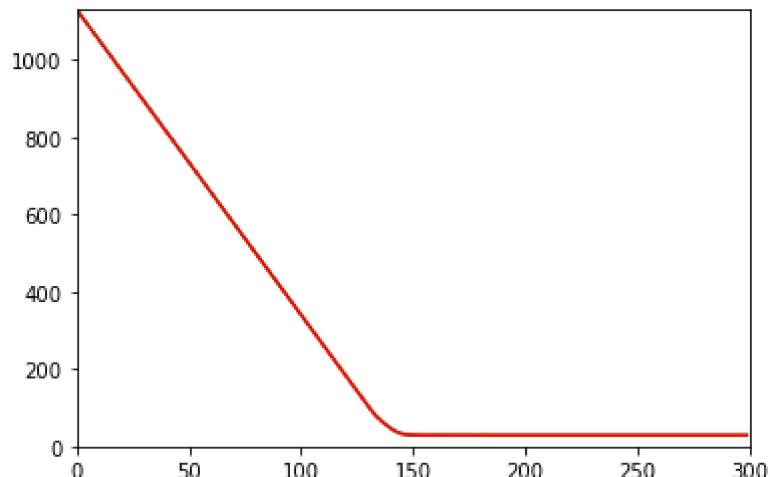
```
1 # tf.compat.v1.disable_eager_execution()
2 # tf.compat.v1.enable_eager_execution()
3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 import numpy as np
6
7 from sklearn.datasets import load_boston
8
9 def append_bias_reshape(x_train,y_train):
10    n_training_samples = x_train.shape[0]
11    n_dim = x_train.shape[1]
12    f = np.reshape(np.c_[np.ones(n_training_samples),x_train],[n_training_samples,n_dim+1])
13    l = np.reshape(y_train,[n_training_samples,1])
14
15    return f,l
16
17 if __name__ == '__main__':
18
19
20 #     x,y = read_boston_data()
21 #     norm_features = feature_normalize(x)
22 f,l = append_bias_reshape(x_train,y_train)
23 n_dim = f.shape[1]
24
25 rnd_indices = np.random.rand(len(f)) < 0.80
26
27 x_train = f[rnd_indices]
28 y_train = l[rnd_indices]
29 x_test = f[~rnd_indices]
30 y_test = l[~rnd_indices]
31
32
33
34 learning_rate = 0.00001
35 training_epochs = 300
36 cost_history = []
37 test_history = []
38
39 X = tf.compat.v1.placeholder(tf.float32,[None,n_dim])
40 Y = tf.compat.v1.placeholder(tf.float32,[None,1])
41 W = tf.Variable(tf.ones([n_dim,1]))
42
43 init = tf.compat.v1.initialize_all_variables()
44
45 y_ = tf.matmul(X,W)
```

```

46     cost = tf.reduce_mean(tf.abs(y_-Y))
47     training_step = tf.compat.v1.train.GradientDescentOptimizer(learning_rate).minimize(cost)
48
49     sess = tf.compat.v1.Session()
50     sess.run(init)
51
52     for epoch in range(training_epochs):
53         sess.run(training_step,feed_dict={X:x_train,Y:y_train})
54         c = sess.run(cost,feed_dict={X:x_train,Y:y_train})
55         print(c)
56         t = sess.run(cost,feed_dict={X:x_test,Y:y_test})
57         print(t)
58         cost_history.append(c)
59         test_history.append(t)
60
61     plt.plot(range(len(test_history)),test_history,color = 'green')
62     plt.plot(range(len(cost_history)),cost_history,color = 'red')
63
64     plt.axis([0,training_epochs,0,np.max(cost_history)])
65     plt.show()

```

28.475595  
 28.635077  
 28.475382  
 28.634863  
 28.475172

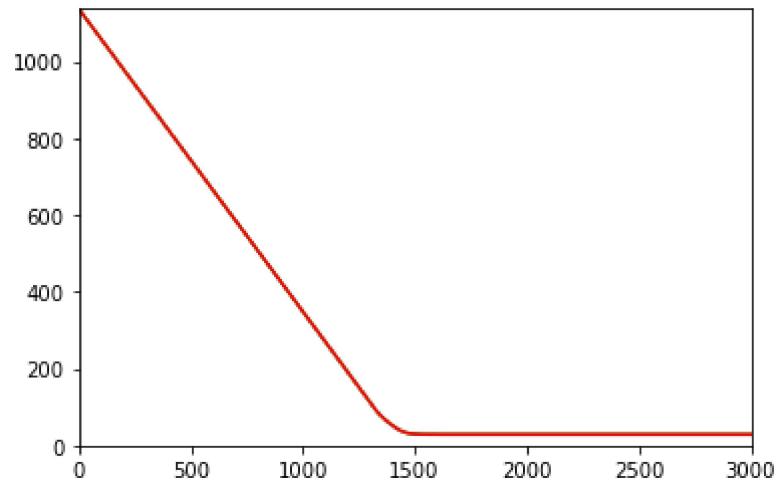


In [88]:

```
1 # tf.compat.v1.disable_eager_execution()
2 # tf.compat.v1.enable_eager_execution()
3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 import numpy as np
6
7 from sklearn.datasets import load_boston
8
9 def append_bias_reshape(x_train,y_train):
10    n_training_samples = x_train.shape[0]
11    n_dim = x_train.shape[1]
12    f = np.reshape(np.c_[np.ones(n_training_samples),x_train],[n_training_samples,n_dim+1])
13    l = np.reshape(y_train,[n_training_samples,1])
14
15    return f,l
16
17 if __name__ == '__main__':
18
19
20 #     x,y = read_boston_data()
21 #     norm_features = feature_normalize(x)
22 f,l = append_bias_reshape(x_train,y_train)
23 n_dim = f.shape[1]
24
25 rnd_indices = np.random.rand(len(f)) < 0.80
26
27 x_train = f[rnd_indices]
28 y_train = l[rnd_indices]
29 x_test = f[~rnd_indices]
30 y_test = l[~rnd_indices]
31
32
33
34 learning_rate = 0.000001
35 training_epochs = 3000
36 cost_history = []
37 test_history = []
38
39 X = tf.compat.v1.placeholder(tf.float32,[None,n_dim])
40 Y = tf.compat.v1.placeholder(tf.float32,[None,1])
41 W = tf.Variable(tf.ones([n_dim,1]))
42
43 init = tf.compat.v1.initialize_all_variables()
44
45 y_ = tf.matmul(X,W)
```

```
46     cost = tf.reduce_mean(tf.abs(y_-Y))
47     training_step = tf.compat.v1.train.GradientDescentOptimizer(learning_rate).minimize(cost)
48
49     sess = tf.compat.v1.Session()
50     sess.run(init)
51
52     for epoch in range(training_epochs):
53         sess.run(training_step,feed_dict={X:x_train,Y:y_train})
54         c = sess.run(cost,feed_dict={X:x_train,Y:y_train})
55         print(c)
56         t = sess.run(cost,feed_dict={X:x_test,Y:y_test})
57         print(t)
58         cost_history.append(c)
59         test_history.append(t)
60
61     plt.plot(range(len(test_history)),test_history,color = 'green')
62     plt.plot(range(len(cost_history)),cost_history,color = 'red')
63
64     plt.axis([0,training_epochs,0,np.max(cost_history)])
65     plt.show()
```

```
28.593613
28.796995
28.593592
28.796976
```



In [ ]:

1

