BIG DATA PROJECT – GROUP 7

ANALYZING PUBG GAME DATA

Our Team

Harsh Gupta
HXG171830

Siddharth Oza
SRO170030

Devarsh Patel
DXP173530

Adit Kansara
ASK172030

Player Unknown's Battle Grounds (PUBG)

100 players are dropped onto an island empty-handed and must explore, scavenge, and eliminate other players until only one is left standing, all while the play zone continues to shrink.

The Game

TRENDING

Beta Release: March 2017.

Worldwide Release: December 2017

MASS USERBASE

227 million monthly players, 87 million daily players, 400 million players till date

WORLD RECORD

World record for most simultaneous players at once

REVENUE

113 million monthly revenue, ~700k earning from daily user spending

MULTI PLATFORM

Available on Windows, Android, iOS, and Xbox

VIDEO

2.03 billion minutes of viewing on Twitch

Features of Our Data

4.5 MILLION ROWS

26 COLUMNS

47k+ MATCH DATA

1.88 MILLION GROUPS

3 STRING VARIABLES,
23 INTEGER VARIABLES

DATA SCOURCE: KAGGLE

METADATA

MATCHID
ID to identify match.

GROUPID
ID to identify a group within a match.

NUMGROUPS
Number of groups we have data for in the match.

MATCHTYPE
String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”

KILLS
Number of enemy players killed.

HEADSHOTKILLS
Number of enemy players killed with headshots.

VEHICLEDESTROYS
Number of vehicles destroyed.

WEAPONSACQUIRED
Number of weapons picked up.

HEALS
Number of healing items used.

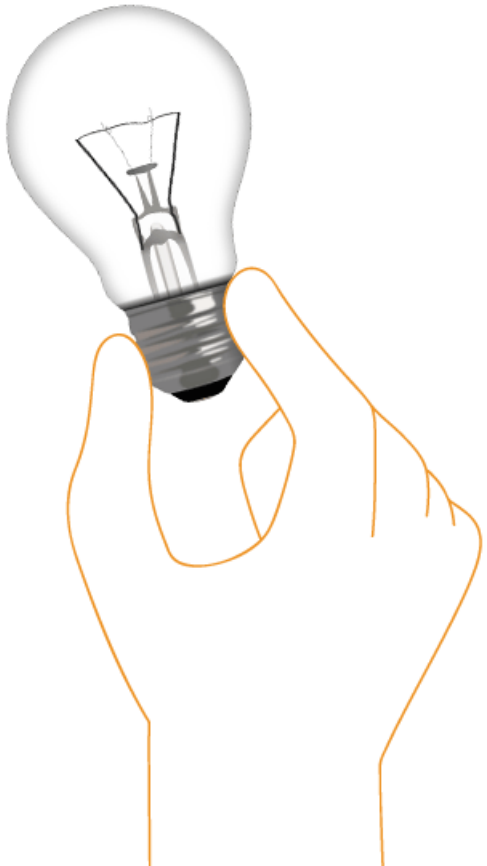
REVIVES
Number of times this player revived teammates.

SWIMDISTANCE
Total distance traveled by swimming measured in meters.

WINPLACEPERC
This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match.

Our Approach

Enhancing the game experience using **insights**



“Does killing more people increases the chance of winning the game?”

APPROACH

USING THE CORRELATION BETWEEN THE MATCH WINNING PERCENTAGE AND NUMBER OF KILLS TO DETERMINE THE RELATIONSHIP BETWEEN THE TWO.

Columns Used

WINPLACEPERC, KILLS

Data Pre-processing

None

Tool Used

Hive

DATA ANALYSIS

Data Preprocessing and Correlation

```
set hive.cli.print.header=true;
select avg(kills) as Average_kills, min(kills) as min_kills, max(kills) as Max_kills,
variance(kills) as variance, stddev_pop(kills) as Standard_Deviation,
corr(kills,winplaceperc) as Correlation from pubg_new ;
```

```
average_kills   min_kills       max_kills       variance        standard_deviation      corr
elation
0.9344957561225483      0       60      2.452957843208639       1.5661921476015128      0.41
534968073846773
```

```
set hive.cli.print.header=true;
select avg(winplaceperc) as Average_Winperc, min(winplaceperc) as min_WinPerc, max(winplaceperc) as Max_WinPerc,
variance(winplaceperc) as variance, stddev_pop(winplaceperc) as Standard_Deviation
from pubg_new ;
```

```
OK
average_wpp     min_wpp max_wpp variance        standard_deviation
0.47186630173457506     0.0     1.0     0.09481144563613568     0.3079146726548374
Time taken: 29.517 seconds, Fetched: 1 row(s)
```

	WINPLACEPERC	KILLS
Max	1	60
Min	0	0
Average	0.47	0.93
Standard Deviation	0.31	1.56
Variance	0.09	2.45
Missing Values	0	0
Correlation	**0.4153**	

THE CORRELATION SUGGESTS THAT THE RELATION BETWEEN MATCH WINNING PERCENTAGE AND NUMBER OF KILLS IS POSITIVE AND SUGGESTS THAT MORE KILLING DIRECTLY IMPACT THE WINNING PLACEMENT. HOWEVER, THE CORRELATION IS NOT STRONG AND IS NOT A SIGNIFICANT PREDICTOR OF MATCH WINNING.

"Can we predict the finishing position of a player in the game?"

APPROACH

Regression Problem: design and test a model using regression algorithms to predict the final position of the player at the end of the game.

Columns Used

WINPLACEPERC, All Columns

Data Pre-processing

Data standardization

Tool Used

Hive, Spark

DATA ANALYSIS

Data Preprocessing

```
set hive.cli.print.header=true;
select avg(heals) as Average_heals, min(heals) as min_heals, max(heals) as Max_heals,
variance(heals) as variance, stddev_pop(heals) as Standard_Deviation,
corr(heals,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(killPlace) as Average_KP, min(killplace) as min_kp, max(killplace) as Max_kp,
variance(killplace) as variance, stddev_pop(killplace) as Standard_Deviation,
corr(killplace,winplaceperc) as Correlation from pubg_new ;
```

```
set hive.cli.print.header=true;
select avg(revives) as Average_revives, min(revives) as min_revives, max(revives) as Max_revives,
variance(revives) as variance, stddev_pop(revives) as Standard_Deviation,
corr(revives,winplaceperc) as Correlation from pubg_new ;
```

```
OK
average_revives min_revives     max_revives     variance        standard_deviation      correlation
0.16493449208415417     0       41      0.2182761907508199      0.46720037537529857     0.25139898468036737
Time taken: 30.705 seconds, Fetched: 1 row(s)
```

```
OK
average_kp      min_kp  max_kp  variance        standard_deviation      correlation
47.03440198323012       1       100     746.8041872621832       27.32771829593871       -0.7083135059792309
Time taken: 30.327 seconds, Fetched: 1 row(s)
```

```
OK
average_heals   min_heals       max_heals       variance        standard_deviation      correlation
1.1871689491010105      0       59      5.599793283374966       2.3663882359779778      0.42798648152254226
Time taken: 30.251 seconds, Fetched: 1 row(s)
```

	HEALS	KILLPLACE	REVIVES
Max	59	100	41
Min	0	1	0
Average	1.18	47.03	0.16
Standard Deviation	2.36	27.32	0.47
Variance	5.59	746.80	0.22
Missing Values	0	0	0
Correlation with win percentage	**0.43**	**-0.71**	**0.25**

DATA ANALYSIS

Standardizing the data

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"DBNOs
",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"
maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"ve
hicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints",$"winorlose",$"winquartiles")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headsho
tKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","numGrou
ps","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDista
nce","weaponsacquired","winpoints","winorlose","winquartiles")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val scaler = new StandardScaler().setInputCol("features").setOutputCol("scaledFeatures").setWith
Std(true).setWithMean(false)
val scalerModel = scaler.fit(data1)
val scaledData = scalerModel.transform(data1)
scaledData.show() 20, False
scaledData.show(Int.MaxValue)
scaledData.show(20, false)

val lr = new LinearRegression()
val lrModel = lr.fit(scaledData)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

As the data in different columns differ in the range significantly. This can affect the regression models. To reduce the effect we will standardize the data.

DATA ANALYSIS

Standardized data output

```
--------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
|0.7407|(23,[0,1,2,4,5,6,10,11,12,18,19,20,22],[1.0,93.0,1.0,2.0,54.0,1404.0,28.0,28.0,1.0,3248.0,
5.0,1583.0,3.0])            |(23,[0,1,2,4,5,6,10,11,12,18,19,20,22],[0.6407619330288546,0.547483782
8212576,0.8392681003773699,0.8451695793802982,1.976015076940289,11.348944399256963,1.2346938649857
895,1.2688821723958992,2.1404088876149077,2.9095885125533094,2.0815032046434725,37.215779534417635
,2.5989929873332196])                                                         |
|0.1146|(23,[1,5,6,10,11,18,19,20,22],[95.0,86.0,1069.0,97.0,94.0,21.0,1.0,1489.0,1.0])
                            |(23,[1,5,6,10,11,18,19,20,22],[0.5592576276131126,3.1469869743863863,8
.641040999149354,4.277332317986485,4.2598187216148045,0.018811994693232605,0.4163006409286945,35.0
0587222157161,0.8663309957777399])

                                                                              |
|0.5217|(23,[0,4,5,6,10,11,18,19,20,22],[1.0,1.0,58.0,1034.0,47.0,41.0,640.0,4.0,1475.0,3.0])
                            |(23,[0,4,5,6,10,11,18,19,20,22],[0.6407619330288546,0.4225847896901491
,2.1223865641210513,8.358125718541096,2.072521844797575,1.858006038151138,0.5733179335080413,1.665
202563714778,34.67673708987114,2.5989929873332196])

                                                                              |
|0.9368|(23,[0,1,5,6,7,8,9,10,11,13,18,19,20,22],[2.0,128.0,25.0,1000.0,1.0,1.0,27.0,96.0,96.0,222
1.0,1016.0,4.0,1500.0,4.0])|(23,[0,1,5,6,7,8,9,10,11,13,18,19,20,22],[1.2815238660577093,0.7535260
666787201,0.9148217948797635,8.083293731664503,0.6384909631975312,1.3843793028549987,0.59012953727
25505,4.233236108522707,4.350453162500226,1.8161372934605524,0.9101422194440155,1.665202563714778,
35.26447839647913,3.4653239831109595])                                        |
|0.3721|(23,[1,5,6,7,8,9,10,11,13,18,19,20,22],[130.0,28.0,1037.0,1.0,1.0,5.0,44.0,40.0,721.0,280.
0,3.0,1495.0,2.0])          |(23,[1,5,6,7,8,9,10,11,13,18,19,20,22],[0.7652999114705751,1.024600410
265335,8.38237559973609,0.6384909631975312,1.3843793028549987,0.10928324764306489,1.94023321640624
06,1.8126888177084275,0.5895700083678785,0.2508265959097681,1.2489019227860836,35.146930135157525,
1.7326619915554797])                                                          |
|1.0   |[1.0,661.0,2.0,3.0,2.0,3.0,1148.0,5.0,2.0,36.0,46.0,46.0,0.0,0.0,0.0,0.0,0.0,0.0,2617.0,4.
0,1479.0,1.0,4.0]           |[0.6407619330288546,3.891255703708078,1.6785362007547397,4.91558944719
2029,0.8451695793802982,0.10977861538557161,9.27962120395085,3.192454815987656,2.768758605709973,
0.7868393830300673,2.028425635333797,2.0845921403646916,0.0,0.0,0.0,0.0,0.0,0.0,2.344332862485225,
1.665202563714778,34.77077569892842,5.834494257891479,3.4653239831109595]|
|0.7037|(23,[0,1,4,5,6,10,11,13,15,18,19,20,22],[3.0,94.0,5.0,50.0,1286.0,28.0,28.0,2963.0,28.0,31
39.0,5.0,1528.0,3.0])       |(23,[0,1,4,5,6,10,11,13,15,18,19,20,22],[1.922285799086564,0.553370705
2171851,2.112923948450746,1.829643589759527,10.395115738920552,1.2346938649857895,1.26888217239589
92,2.4228792438197284,1.017983432372457,2.811945302002721,2.0815032046434725,35.92274865988007,2.5
989929873332196])                                                             |
|0.0417|(23,[1,5,6,10,11,18,19,20,22],[137.0,81.0,1000.0,25.0,23.0,238.0,3.0,1500.0,1.0])
                            |(23,[1,5,6,10,11,18,19,20,22],[0.8065083682420676,2.9640226154104337,8
.083293731664503,1.1024052365944548,1.0422960701823458,0.21320260652330286,1.2489019227860836,35.2
6447839647913,0.8663309957777399])

                                                                              |
|0.4536|[1.0,166.0,0.0,1.0,1.0,35.0,1000.0,1.0,1.0,31.0,98.0,96.0,0.0,0.0,0.0,0.0,0.0,0.0,184.0,3.
0,1500.0,0.0,2.0]           |[0.6407619330288546,0.9772291177239651,0.0,1.6385298157306762,0.422584
7896901491,1.2807505128316687,8.083293731664503,0.6384909631975312,1.3843793028549987,0.6775561353
870023,4.321428527450263,4.350453162500226,0.0,0.0,0.0,0.0,0.0,0.0,0.16482890588356186,1.248901922
7860836,35.26447839647913,0.0,1.7326619915554797]                             |
|0.4828|(23,[1,5,6,10,11,18,19,20,22],[219.0,63.0,1180.0,30.0,29.0,1036.0,5.0,1533.0,2.0])
                            |(23,[1,5,6,10,11,18,19,20,22],[1.2892360047081228,2.305350923097004,9.
```

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 27 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 22 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_2c87590a98a4
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
scaler: org.apache.spark.ml.feature.StandardScaler = stdScal_7c809c1c6e86
scalerModel: org.apache.spark.ml.feature.StandardScalerModel = stdScal_7c809c1c6e86
scaledData: org.apache.spark.sql.DataFrame = [label: float, features: vector ... 1 more field]
<console>:1: error: ';' expected but integer literal found.
scaledData.show() 20, False
                  ^
org.apache.spark.sql.AnalysisException: The limit expression must be equal to or greater than 0, b
ut got -2147483648;;
GlobalLimit -2147483648
+- LocalLimit -2147483648
   +- Project [label#240, features#289, UDF(features#289) AS scaledFeatures#322]
      +- Project [label#240, features#289]
         +- Project [label#240, boosts#185, damageDealt#186, DBNOs#187, headshotKills#188, heals#1
89, killPlace#190, killPoints#191, kills#192, killStreaks#193, longestKill#194, maxPlace#195, numG
roups#196, revives#197, rideDistance#198, roadKills#199, swimDistance#200, teamKills#201, vehicleD
estroys#202, walkDistance#203, weaponsacquired#204, winpoints#205, winorlose#208, winquartiles#209
, UDF(named_struct(boosts_double_vecAssembler_2c87590a98a4, cast(boosts#185 as double), damageDeal
t_double_vecAssembler_2c87590a98a4, cast(damageDealt#186 as double), DBNOs_double_vecAssembler_2c8
7590a98a4, cast(DBNOs#187 as double), headshotKills_double_vecAssembler_2c87590a98a4, cast(headsho
tKills#188 as double), heals_double_vecAssembler_2c87590a98a4, cast(heals#189 as double), killPlac
e_double_vecAssembler_2c87590a98a4, cast(killPlace#190 as double), killPoints_double_vecAssembler_
2c87590a98a4, cast(killPoints#191 as double), kills_double_vecAssembler_2c87590a98a4, cast(kills#1
92 as double), killStreaks_double_vecAssembler_2c87590a98a4, cast(killStreaks#193 as double), long
estKill_double_vecAssembler_2c87590a98a4, cast(longestKill#194 as double), maxPlace_double_vecAsse
mbler_2c87590a98a4, cast(maxPlace#195 as double), numGroups_double_vecAssembler_2c87590a98a4, cast
(numGroups#196 as double), ... 22 more fields)) AS features#289]
            +- Project [winplaceperc#206 AS label#240, boosts#185, damageDealt#186, DBNOs#187, hea
dshotKills#188, heals#189, killPlace#190, killPoints#191, kills#192, killStreaks#193, longestKill#
194, maxPlace#195, numGroups#196, revives#197, rideDistance#198, roadKills#199, swimDistance#200,
teamKills#201, vehicleDestroys#202, walkDistance#203, weaponsacquired#204, winpoints#205, winorlos
e#208, winquartiles#209]
               +- Project [id#181, groupid#182, matchid#183, assists#184, boosts#185, damagedealt#
186, dbnos#187, headshotkills#188, heals#189, killplace#190, killpoints#191, kills#192, killstreak
s#193, longestkill#194, maxplace#195, numgroups#196, revives#197, ridedistance#198, roadkills#199,
 swimdistance#200, teamkills#201, vehicledestroys#202, walkdistance#203, weaponsacquired#204, ...
5 more fields]
                  +- SubqueryAlias pubg_new
                     +- HiveTableRelation `default`.`pubg_new`, org.apache.hadoop.hive.serde2.lazy
.LazySimpleSerDe, [id#181, groupid#182, matchid#183, assists#184, boosts#185, damagedealt#186, dbn
os#187, headshotkills#188, heals#189, killplace#190, killpoints#191, kills#192, killstreaks#193, l
ongestkill#194, maxplace#195, numgroups#196, revives#197, ridedistance#198, roadkills#199, swimdis
tance#200, teamkills#201, vehicledestroys#202, walkdistance#203, weaponsacquired#204, ... 5 more f
ields]
```

DATA ANALYSIS

Linear Regression Code

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"D
BNOs",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longest
Kill",$"maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"tea
mKills",$"vehicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints",$"winorlose",$"wi
nquartiles")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","hea
dshotKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace",
"numGroups","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys
","walkDistance","weaponsacquired","winpoints","winorlose","winquartiles")).setOutputCol("fe
atures")
val data1 = assembler.transform(Data).select($"label",$"features")
val scaler = new StandardScaler().setInputCol("features").setOutputCol("scaledFeatures").set
WithStd(true).setWithMean(false)
val scalerModel = scaler.fit(data1)
val scaledData = scalerModel.transform(data1)
scaledData.show() 20, False
scaledData.show(Int.MaxValue)
scaledData.show(20, false)

val lr = new LinearRegression()
val lrModel = lr.fit(scaledData)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

RUNNING LINEAR REGRESSION ON THE ENTIRE DATASET TO DEVELOP A MODEL TO PREDICT THE WINNING PERCENTAGE BASED ON VARIOUS PARAMETERS.

DATA ANALYSIS

Linear Regression Output

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 27 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 22 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_2bc7d92d85bf
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
lr: org.apache.spark.ml.regression.LinearRegression = linReg_f95c51c51d53
18/12/04 05:50:14 WARN org.apache.spark.ml.optim.WeightedLeastSquares: regParam is zero, which mig
ht cause numerical instability and overfitting.
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_f95c51c51d53
Coefficients: [0.004370530184686799,2.8340871808026174E-5,-4.7868355701602053E-4,-5.22342567368808
7E-4,-1.1072552274336239E-4,-0.001983745489922085,-8.982909541338697E-6,-0.005637546617050469,-0.0
3519339677059854,-1.168343534091021E-5,-0.001892786806077423,0.002336970754476911,0.00435914241402
1138,-9.662325941252138E-8,0.001310351777846154,3.513869073193153E-5,-0.005096918423303263,-0.0010
125946697538762,2.2509013343957433E-5,0.0031590702964546724,9.121853050740724E-5,0.115202566564300
46,0.2064543564870297] Intercept: -0.0970958211037593
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark
.ml.regression.LinearRegressionTrainingSummary@227b27d8
numIterations: 1
objectiveHistory: List(0.0)
+--------------------+
|           residuals|
+--------------------+
| 0.08176106933327754|
|0.007222844760086...|
|-0.04892566480554228|
| 0.08067235588625588|
|-0.00392568119158...|
|0.024089537571331476|
|0.037982250764471814|
|-0.05996774781729175|
| 0.05964943176107784|
| 0.10641501892209199|
|-0.09204073535375068|
|-0.09210428819908456|
|-0.07248369959894915|
|0.008579546399007265|
| 0.07430438829172833|
|  0.0634420997601155|
|-0.01214568439334...|
| 0.09606508782634779|
| 0.08134734375819552|
|-0.09948699460991456|
+--------------------+
only showing top 20 rows

RMSE: 0.06455368327891929
MSE: 0.004167178024875024
r2: 0.9560477229609481
```

The regression model provides with an R Square of 0.95

Based on the predictors we have included in the data we can say that the position of a player can be accurately predicted based on the various game parameters.

DATA ANALYSIS

Random Forest Feature Importance Code

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{ RandomForestRegressor, RandomForestRegressionModel }
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"D
BNOs",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longest
Kill",$"maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"tea
mKills",$"vehicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","hea
dshotKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace",
"numGroups","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys
","walkDistance","weaponsacquired","winpoints")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")



val rf = new RandomForestRegressor
val model: RandomForestRegressionModel = rf.fit(data1)
// GET FEATURE IMPORTANCE
val featImp = model.featureImportances
val featureMetadata = data1.schema("features").metadata
```

Using the Random Forest Regressor to determine the top important features affecting the winning percentage.

DATA ANALYSIS

Random Forest Feature Importance Output

```
Loading FI.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{RandomForestRegressor, RandomForestRegressionModel}
import org.apache.spark.ml.feature.StandardScaler
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 29 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 20 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_226b3aad571f
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
rf: org.apache.spark.ml.regression.RandomForestRegressor = rfr_1c6400a3f31e
model: org.apache.spark.ml.regression.RandomForestRegressionModel = RandomForestRegressionModel (u
id=rfr_1c6400a3f31e) with 20 trees
featImp: org.apache.spark.ml.linalg.Vector = (21,[0,1,2,4,5,6,7,8,9,10,11,13,15,18,19,20],[0.08977
283908881427,0.012744463106536524,1.946281262305920 8E-4,0.04897068194797123,0.23146347014760993,2.
7190917172233706E-4,0.004300331771402184,0.0036493766559726306,0.02226604772527401,0.0040755048499
96403,0.005133947226526865,0.0030861733807275755,9.423946469668577E-4,0.4649916973310746,0.1080778
5430338716,5.868051978687362E-5])
featureMetadata: org.apache.spark.sql.types.Metadata = {"ml_attr":{"attrs":{"numeric":[{"idx":0,"n
ame":"boosts"},{"idx":1,"name":"damageDealt"},{"idx":2,"name":"DBNOs"},{"idx":3,"name":"headshotKi
lls"},{"idx":4,"name":"heals"},{"idx":5,"name":"killPlace"},{"idx":6,"name":"killPoints"},{"idx":7
,"name":"kills"},{"idx":8,"name":"killStreaks"},{"idx":9,"name":"longestKill"},{"idx":10,"name":"m
axPlace"},{"idx":11,"name":"numGroups"},{"idx":12,"name":"revives"},{"idx":13,"name":"rideDistance
"},{"idx":14,"name":"roadKills"},{"idx":15,"name":"swimDistance"},{"idx":16,"name":"teamKills"},{"
idx":17,"name":"vehicleDestroys"},{"idx":18,"name":"walkDistance"},{"idx":19,"name":"weaponsacquir
ed"},{"idx":20,"name":"winpoints"}]},"num_attrs":21}}
```

18	0.464991697	walkDistance
5	0.23146347	killPlace
19	0.108077854	weaponsacquired
0	0.089772839	boosts
4	0.048970682	heals
9	0.022266048	longestKill
1	0.012744463	damageDealt
11	0.005133947	numGroups
7	0.004300332	kills
10	0.004075505	maxPlace
8	0.003649377	killStreaks
13	0.003086173	rideDistance
15	9.42E-04	swimDistance
6	2.72E-04	killPoints
2	1.95E-04	DBNOs
20	5.87E-05	winpoints

Based on the Random Forest Feature Importance Output we conclude that walkDistance, killPlace, weaponsacquired, boosts, and heals are the top features affecting the winning place percentage.

DATA ANALYSIS

Output of Linear Regression on Standardized data

```
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
only showing top 20 rows

lr: org.apache.spark.ml.regression.LinearRegression = linReg_567527e95eec
18/12/04 08:33:43 WARN org.apache.spark.ml.optim.WeightedLeastSquares: regParam is zero, which mig
ht cause numerical instability and overfitting.
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_567527e95eec
Coefficients: [0.004370530184686799,2.8340871808026174E-5,-4.7868355701602053E-4,-5.22342567368808
7E-4,-1.1072552274336239E-4,-0.001983745489922085,-8.982909541338697E-6,-0.005637546617050469,-0.0
3519339677059854,-1.168343534091021E-5,-0.001892786806077423,0.002336970754476911,0.00435914241402
1138,-9.662325941252138E-8,0.001310351777846154,3.513869073193153E-5,-0.005096918423303263,-0.0010
125946697538762,2.2509013343957433E-5,0.0031590702964546724,9.121853050740724E-5,0.115202566564300
46,0.2064543564870297] Intercept: -0.0970958211037593
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark
.ml.regression.LinearRegressionTrainingSummary@9704e52
numIterations: 1
objectiveHistory: List(0.0)
+--------------------+
|           residuals|
+--------------------+
| 0.08176106933327754|
|0.007222844760086...|
|-0.04892566480554228|
| 0.08067235588625588|
|-0.00392568119158...|
|0.024089537571331476|
|0.037982250764471814|
|-0.05996774781729175|
| 0.05964943176107784|
| 0.10641501892209199|
|-0.09204073535375068|
|-0.09210428819908456|
|-0.07248369959894915|
|0.008579546399007265|
| 0.07430438829172833|
|  0.0634420997601155|
|-0.01214568439334...|
| 0.09606508782634779|
| 0.08134734375819552|
|-0.09948699460991456|
+--------------------+
only showing top 20 rows

RMSE: 0.06455368327891929
MSE: 0.004167178024875024
r2: 0.9560477229609481
```

Running the regression model again on the standardized data to check the difference in output in standardized and non standardized data.

Comparing the R square we understand that standardization of the data does not make a difference on the model. Hence, the data in its original form can be used for further analysis.

“Can we predict the winner of the game?”

APPROACH

CLASSIFICATION PROBLEM: DIVIDE THE DATA INTO WINNERS AND LOSERS. DESIGN AND TEST A MODEL USING VARIOUS CLASSIFICATION ALGORITHMS TO PREDICT IF A PLAYER WILL WIN/LOSE.

Columns Used

WINORLOSE, WINPLACEPERC, All Columns

Data Pre-processing

Create new column "WINORLOSE" which will have value 1 for all the WINPLACEPERC=1 and 0 otherwise. Data Standardization.

Tool Used

Hive, Spark

DATA ANALYSIS

Creating new column "WINORLOSE" and validating it.

```
set hive.cli.print.header=true;
ALTER TABLE pubg_new ADD COLUMNS (WinOrLose Int);
INSERT OVERWRITE TABLE pubg_new
SELECT
        Id ,
        groupId ,
        matchId ,
        assists ,
        boosts ,
       `damageDealt` ,
       `DBNOs`,
       `headshotKills` ,
       `heals` ,
       `killPlace` ,
       `killPoints`,
       `kills` ,
       `killStreaks` ,
       `longestKill`,
       `maxPlace` ,
       `numGroups` ,
       `revives`,
       `rideDistance` ,
       `roadKills` ,
       `swimDistance` ,
       `teamKills` ,
       `vehicleDestroys` ,
       `walkDistance` ,
       `weaponsAcquired` ,
       `winPoints` ,
       `winPlacePerc`,
       'match_type',
 if(winplaceperc = 1, 1,0)
as WinOrLose from pubg_new;
```

```
OK
winperc winorlose
NULL    0
0.8571  0
0.04    0
0.7407  0
0.1146  0
0.5217  0
0.9368  0
0.3721  0
1.0     1
0.7037  0
Time taken: 2.226 seconds, Fetched: 10 row(s)
```

```
sidoza7802@cluster-3cf6-m:~$ hive -e "select WinOrLose,count(WinOrLose) from pubg_new group by WinOrLose;"
```

```
Stage-Stage-1: Map: 2  Reduce: 2   Cumulative CPU: 18.94 sec   HDFS Read: 425386101 HDFS Write: 217 SUCCESS
Total MapReduce CPU Time Spent: 18 seconds 940 msec
OK
0       4225337
1       132000
Time taken: 32.021 seconds, Fetched: 2 row(s)
```

DATA ANALYSIS

Data Preprocessing

```
set hive.cli.print.header=true;
select avg(boosts) as Average_boosts, min(boosts) as min_boosts, max(boosts) as Max_boosts,
variance(boosts) as variance, stddev_pop(boosts) as Standard_Deviation,
corr(boosts,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(damagedealt) as Average_DD, min(damagedealt) as min_DD, max(damagedealt) as Max_DD,
variance(damagedealt) as variance, stddev_pop(damagedealt) as Standard_Deviation,
corr(damagedealt,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(DBNOs) as Average_DBNOs, min(DBNOs) as min_DBNOs, max(DBNOs) as Max_DBNOs,
variance(DBNOs) as variance, stddev_pop(DBNOs) as Standard_Deviation,
corr(DBNOs,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(headshotkills) as Average_HSK, min(headshotkills) as min_HSK, max(headshotkills) as Max_HSK,
variance(headshotkills) as variance, stddev_pop(headshotkills) as Standard_Deviation,
corr(headshotkills,winplaceperc) as Correlation from pubg_new ;
```

```
OK
average_boosts  min_boosts      max_boosts      variance        standard_deviation      correlation
0.9636856097395289      0       18      2.4356051102717227      1.5606425312260725      0.6180749137981152
Time taken: 29.118 seconds, Fetched: 1 row(s)
```

```
OK
average_dd      min_dd  max_dd  variance        standard_deviation      correlation
132.60639597221788      0       6384    28855.125374886822      169.86796453388973      0.43830691001628214
Time taken: 30.297 seconds, Fetched: 1 row(s)
```

```
OK
average_dbnos   min_dbnos       max_dbnos       variance        standard_deviation      correlation
0.6901455384666227      0       63      1.4197057783254678      1.1915140697136009      0.2794746487402532
Time taken: 31.387 seconds, Fetched: 1 row(s)
```

```
OK
average_hsk     min_hsk max_hsk variance        standard_deviation      correlation
0.23858660429216383     0       26      0.3724699883150234      0.6103031937611202      0.2787052860462615
Time taken: 29.928 seconds, Fetched: 1 row(s)
```

	BOOSTS	DAMAGE DEALT	DBNO's	HEADSHOT KILLS
Max	18	6384	63	26
Min	0	0	0	0
Average	0.96	132.6	0.69	0.23
Standard Deviation	1.56	169.86	1.19	0.61
Variance	2.43	28855.12	1.41	0.37
Missing Values	0	0	0	0
Correlation with win percentage	**0.61**	**0.43**	**0.27**	**0.27**

DATA ANALYSIS

Logistic Regression Code

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary



val inputfile = sql("select * from pubg_new")
val Data=inputfile.select(inputfile("winorlose").as("label"),$"boosts",$"damageDealt",$"DBNOs",$
"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"max
Place",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"vehic
leDestroys",$"walkDistance",$"weaponsacquired",$"winpoints",$"winorlose",$"winquartiles")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headsho
tKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","numGrou
ps","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDista
nce","weaponsacquired","winpoints","winorlose","winquartiles")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val Array(training, test) = data1.randomSplit(Array(0.6, 0.4), seed = 11L)
training.cache()
val lg = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lgModel = lg.fit(data1)
println(s"Coefficients: ${lgModel.coefficients} Intercept: ${lgModel.intercept}")



val trainingSummary = lgModel.summary
val objectiveHistory = trainingSummary.objectiveHistory
println("objectiveHistory:")
objectiveHistory.foreach(loss => println(loss))
val binarySummary = trainingSummary.asInstanceOf[BinaryLogisticRegressionSummary]

val roc = binarySummary.roc
roc.show()
println(s"areaUnderROC: ${binarySummary.areaUnderROC}")
val fMeasure = binarySummary.fMeasureByThreshold
val maxFMeasure = fMeasure.select(max("F-Measure")).head().getDouble(0)
val bestThreshold = fMeasure.where($"F-Measure" === maxFMeasure).select("threshold").head().getD
ouble(0)
lgModel.setThreshold(bestThreshold)
```

RUNNING LOGISTIC REGRESSION USING THE NEWLY CREATED COLUMN WINORLOOSE AS THE DEPENDENT VARIABLE TO DETERMINE IF WE CAN PREDICT THE FINAL WINNER OF THE GAME ACCURATELY.

DATA ANALYSIS

Logistic Regression Output

```
scala> :load logistic.scala
Loading logistic.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 27 more fields]
Data: org.apache.spark.sql.DataFrame = [label: int, boosts: int ... 22 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_79b7c1d4cf2e
data1: org.apache.spark.sql.DataFrame = [label: int, features: vector]
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]
res16: training.type = [label: int, features: vector]
lg: org.apache.spark.ml.classification.LogisticRegression = logreg_3f531746da37
lgModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_3f531746da37
Coefficients: (23,[],[]) Intercept: -3.466051193999323
trainingSummary: org.apache.spark.ml.classification.LogisticRegressionTrainingSummary = org.apache
.spark.ml.classification.BinaryLogisticRegressionTrainingSummary@13f0558d
objectiveHistory: Array[Double] = Array(0.13576185285029335)
objectiveHistory:
0.13576185285029335
binarySummary: org.apache.spark.ml.classification.BinaryLogisticRegressionSummary = org.apache.spa
rk.ml.classification.BinaryLogisticRegressionTrainingSummary@13f0558d
roc: org.apache.spark.sql.DataFrame = [FPR: double, TPR: double]
+---+---+
|FPR|TPR|
+---+---+
|0.0|0.0|
|1.0|1.0|
|1.0|1.0|
+---+---+

areaUnderROC: 0.5
fMeasure: org.apache.spark.sql.DataFrame = [threshold: double, F-Measure: double]
maxFMeasure: Double = 0.05880607627148092
bestThreshold: Double = 0.030293766919757315
res22: lgModel.type = logreg_3f531746da37
```

Based on the output provided by the ROC curve which is 0.5 in this case we can conclude that the winner of the game can be predicted accurately 50% of the time.

"How different/similar are the strategies required to win the game when playing solo, duo, or in a group?"

APPROACH

DIVIDE THE DATA ON THE BASIS OF MATCH TYPE. RUN REGRESSION ANALYSIS ON THE THESE THREE TYPES INDEPENDENTLY, TO DETERMINE THE COEFFICIENTS AFFECTING EACH MATCH TYPE.

Columns Used

Major: NUMGROUPS, Derived: Match_Type, All other columns

Data Pre-processing

Create a new column from the no of groups column which will act as a filter. Data Standardization.

Tool Used

Hive, Spark

DATA ANALYSIS

Creating new column "MATCH_TYPE" and validating it.

```
set hive.cli.print.header=true;
ALTER TABLE pubg_new ADD COLUMNS (match_type string);
INSERT OVERWRITE TABLE pubg_new
SELECT
        Id ,
        groupId ,
        matchId ,
        assists ,
        boosts ,
       `damageDealt` ,
       `DBNOs`,
       `headshotKills` ,
       `heals` ,
       `killPlace` ,
       `killPoints`,
       `kills` ,
       `killStreaks` ,
       `longestKill`,
       `maxPlace` ,
       `numGroups` ,
       `revives`,
       `rideDistance` ,
       `roadKills` ,
       `swimDistance` ,
       `teamKills` ,
       `vehicleDestroys` ,
       `walkDistance` ,
       `weaponsAcquired` ,
       `winPoints` ,
       `winPlacePerc`, if(numgroups > 50, 'solo',if (numgroups > 25 AND numgroups <= 50,'Duo',
       'Squad'))
as match_type from pubg_new;
```

```
numgroups       match_type
NULL    Squad
28      Duo
23      Squad
28      Duo
94      solo
Time taken: 2.21 seconds, Fetched: 5 row(s)
```

```
sidoza7802@cluster-3cf6-m:~$ hive -e "select match_type,count(match_type) from pubg_new group by match_type;"
```

```
Total MapReduce CPU Time Spent: 18 seconds 100 msec
OK
Duo     3070150
Squad   723908
solo    563279
Time taken: 33.057 seconds, Fetched: 3 row(s)
```

DATA ANALYSIS

Linear Regression on SOLO match type

```
scala> :q
sidoza7802@cluster-3of6-m:~$ cat soloRegression.scala
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1 ='solo'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"DBNOs",
$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"maxP
lace",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"vehicleDestroys",$"wa
lkDistance",$"weaponsacquired",$"winpoints",$"winorlose")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headshotK
ills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","revives","r
ideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDistance","weaponsacqui
red","winpoints","winorlose")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val lr = new LinearRegression()
val lrModel = lr.fit(data1)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

USING THE NEWLY CREATED COLUMN MATCH_TYPE WE HAVE FILTERED OUT THE DATA FOR ONLY SOLO MATCH TYPE AND RAN THE LINEAR REGRESSION MODEL ON IT.

DATA ANALYSIS

Output of Linear Regression on SOLO match type

```
nt cause numerical instability and overfitting.
18/12/04 18:50:50 WARN org.apache.spark.ml.optim.WeightedLeastSquares: Cholesky solver failed due
to singular covariance matrix. Retrying with Quasi-Newton solver.
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_a45d3c764e1b
Coefficients: [0.008087111959259914,4.843614559275396E-5,0.0,0.003568151611454603,-0.0012596788649
563418,-0.003896642723613975,-2.541158257787473E-5,-0.012241267875645915,-0.08786426643984993,-2.5
964237605635255E-5,0.0034657707666070174,0.0,2.038249911647505E-6,0.015768799227271278,7.238962775
428959E-5,-0.010961987909785555,-0.003700129795273632,3.017885743939141E-5,0.0041737611005396935,1
.296159691531127E-4,0.16852225694213144] Intercept: -0.23685225628962409
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark
.ml.regression.LinearRegressionTrainingSummary@6820dd90
numIterations: 34
objectiveHistory: List(0.5, 0.12802457078347307, 0.09412803082606525, 0.035665581402225754, 0.0297
218319885304, 0.021288215758102913, 0.02043776091438998, 0.0201335423583342, 0.019989163567246226,
 0.019957285674438108, 0.0199380257615982, 0.019919698639523364, 0.01990651577134539, 0.0199004063
4329593, 0.019898910520508517, 0.01989880625411633, 0.019898745602862755, 0.019898692773195226, 0.
019898636676082715, 0.019898603170498852, 0.01989851476202631, 0.019898491267714036, 0.01989846771
3089563, 0.019898465755803008, 0.01989846553656971, 0.019898465524408326, 0.019898465517806496, 0.
019898465516165587, 0.019898465515390207, 0.01989846551514063, 0.019898465514787578, 0.01989846551
4639252, 0.01989846551446428, 0.01989846551432528)
+--------------------+
|           residuals|
+--------------------+
| 0.00661936475175412|
| 0.12180173647924364|
| 0.04553256860343152|
|-0.10321569668557862|
| -0.0971768247464746|
| 0.06576505754584505|
|0.017653451920060848|
|-0.05877889486189...|
|-0.07055160793225745|
|-0.04434504467605738|
| 0.07342625646799061|
| 0.06452770854245454|
| 0.04641707715805349|
|-0.03422301530371924|
|  0.0785393381502058|
|0.040913448182108736|
|  0.0922958653043136|
|  0.0893069762579049|
|0.035585333068642866|
|-0.02754494514356...|
+--------------------+
only showing top 20 rows

RMSE: 0.05900588831693975
MSE: 0.0034816948560711667
r2: 0.9602030689714525
```

THE REGRESSION MODEL PROVIDES WITH AN R SQUARE OF 0.96

DATA ANALYSIS

Random Forest Feature Importance Code on SOLO

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{ RandomForestRegressor, RandomForestRegressionModel }
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1='solo'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"D
BNOs",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longest
Kill",$"maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"tea
mKills",$"vehicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","hea
dshotKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace",
"numGroups","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys
","walkDistance","weaponsacquired","winpoints")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")



val rf = new RandomForestRegressor
val model: RandomForestRegressionModel = rf.fit(data1)
// GET FEATURE IMPORTANCE
val featImp = model.featureImportances
val featureMetadata = data1.schema("features").metadata
```

Using the Random Forest Regressor on SOLO Match Type to determine the top important features affecting the winning percentage.

DATA ANALYSIS

Random Forest Feature Importance Output for SOLO

```
scala> :load FIsolo.scala
Loading FIsolo.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{RandomForestRegressor, RandomForestRegressionModel}
import org.apache.spark.ml.feature.StandardScaler
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 29 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 20 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_8edccea70b63
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
rf: org.apache.spark.ml.regression.RandomForestRegressor = rfr_8c64c65f4c52
model: org.apache.spark.ml.regression.RandomForestRegressionModel = RandomForestRegressionModel (u
id=rfr_8c64c65f4c52) with 20 trees
featImp: org.apache.spark.ml.linalg.Vector = (21,[0,1,3,4,5,7,8,9,10,11,13,15,18,19,20],[0.0647224
7359647812,0.01680240131906145,3.663652791718646E-5,0.03806185135376311,0.23031685757925618,0.0078
67955523576772,0.0016645274580343507,0.022705289085934614,0.0013712026381888934,0.0030664871509737
592,0.0020325097041881957,7.210998674617875E-4,0.4615491139561409,0.149000002506286,8.159173273879
291E-5])
featureMetadata: org.apache.spark.sql.types.Metadata = {"ml_attr":{"attrs":{"numeric":[{"idx":0,"n
ame":"boosts"},{"idx":1,"name":"damageDealt"},{"idx":2,"name":"DBNOs"},{"idx":3,"name":"headshotKi
lls"},{"idx":4,"name":"heals"},{"idx":5,"name":"killPlace"},{"idx":6,"name":"killPoints"},{"idx":7
,"name":"kills"},{"idx":8,"name":"killStreaks"},{"idx":9,"name":"longestKill"},{"idx":10,"name":"m
axPlace"},{"idx":11,"name":"numGroups"},{"idx":12,"name":"revives"},{"idx":13,"name":"rideDistance
"},{"idx":14,"name":"roadKills"},{"idx":15,"name":"swimDistance"},{"idx":16,"name":"teamKills"},{"
idx":17,"name":"vehicleDestroys"},{"idx":18,"name":"walkDistance"},{"idx":19,"name":"weaponsacquir
ed"},{"idx":20,"name":"winpoints"}]},"num_attrs":21}}
```

Solo		
18	0.461549	walkDistance
5	0.230317	killPlace
19	0.149	weaponsacquired
0	0.064722	boosts
4	0.038062	heals
9	0.022705	longestKill
1	0.016802	damageDealt
7	0.007868	kills
11	0.003066	numGroups
13	0.002033	rideDistance
8	0.001665	killStreaks
10	0.001371	maxPlace
15	0.000721	swimDistance
20	8.16E-05	winpoints
3	3.66E-05	headshotKills

Based on the Random Forest Feature Importance Output we conclude that walkDistance, killPlace, weaponsacquired, boosts, and heals are the top features affecting the winning place percentage in a solo Match Type.

DATA ANALYSIS

Linear Regression on DUO match type

```
sidoza7802@cluster-3cf6-m:~$ cat duoRegression.scala
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1 ='Duo'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"DBNOs
$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"ma
lace",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"vehicleDestroys",$"
lkDistance",$"weaponsacquired",$"winpoints",$"winorlose")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headsho
ills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","revives",
ideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDistance","weaponsacq
red","winpoints","winorlose")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val lr = new LinearRegression()
val lrModel = lr.fit(data1)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

USING THE NEWLY CREATED COLUMN MATCH_TYPE WE HAVE FILTERED OUT THE DATA FOR ONLY DUO MATCH TYPE AND RAN THE LINEAR REGRESSION MODEL ON IT.

DATA ANALYSIS

Output of Linear Regression on DUO match type

```
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_d5839bb8ec61
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
lr: org.apache.spark.ml.regression.LinearRegression = linReg_71ba2395298b
18/12/04 18:58:51 WARN org.apache.spark.util.Utils: Truncated the string representation of a plan
since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' i
n SparkEnv.conf.
18/12/04 18:58:56 WARN org.apache.spark.ml.optim.WeightedLeastSquares: regParam is zero, which mig
ht cause numerical instability and overfitting.
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_71ba2395298b
Coefficients: [0.00627448411635141,4.844785826275093E-5,-0.005658390666458812,-9.76307683067864E-5
,-6.823690604045519E-4,-0.002261345703229498,-1.3750453619972789E-5,-5.759857574457855E-4,-0.04798
1404137549435,2.859735749420206E-5,4.2727335213095275E-4,0.003617229167834133,-3.0224132584910413E
-7,-2.3684346789234977E-4,3.3615983745048966E-5,-0.007079827104661359,-9.05535676475141E-4,2.56188
26372295775E-5,0.003125864661267802,1.273838372652109E-4,0.20088241797040418] Intercept: -0.115081
54595983673
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark
.ml.regression.LinearRegressionTrainingSummary@4fe3d15f
numIterations: 1
objectiveHistory: List(0.0)
+--------------------+
|           residuals|
+--------------------+
| -0.0629542936118439|
|-0.02046202631753...|
| 0.11900736069215112|
|0.024220277863547413|
|   0.093928899981433|
|-0.07190608566581129|
|-3.88422770377872E-4|
| 0.07031362046894117|
| 0.08786948571981447|
|  0.0907647362934424|
| 0.11210713821514062|
|-0.00403842419064...|
|-0.00695957346829118|
|  0.0869948649494684|
|-0.12831247482809782|
|-0.01722993551232016|
| 0.06362742803434018|
| 0.08396784449852152|
|-0.04111137043287...|
| -0.1310854172852639|
+--------------------+
only showing top 20 rows

RMSE: 0.06561430223758193
MSE: 0.004305236658124748
r2: 0.9542154689241803
```

The regression model provides with an R Square of 0.95

DATA ANALYSIS

Random Forest Feature Importance Code on DUO

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{ RandomForestRegressor, RandomForestRegressionModel }
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1='Duo'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"D
BNOs",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longest
Kill",$"maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"tea
mKills",$"vehicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","hea
dshotKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace",
"numGroups","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys
","walkDistance","weaponsacquired","winpoints")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")



val rf = new RandomForestRegressor
val model: RandomForestRegressionModel = rf.fit(data1)
// GET FEATURE IMPORTANCE
val featImp = model.featureImportances
val featureMetadata = data1.schema("features").metadata
```

Using the Random Forest Regressor on DUO Match Type to determine the top important features affecting the winning percentage.

DATA ANALYSIS

Random Forest Feature Importance Output for DUO

```
scala> :load FIduo.scala
Loading FIduo.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{RandomForestRegressor, RandomForestRegressionModel}
import org.apache.spark.ml.feature.StandardScaler
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 29 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 20 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_f71b1bead051
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
rf: org.apache.spark.ml.regression.RandomForestRegressor = rfr_b5f7fae431a1
model: org.apache.spark.ml.regression.RandomForestRegressionModel = RandomForestRegressionModel (u
id=rfr_b5f7fae431a1) with 20 trees
featImp: org.apache.spark.ml.linalg.Vector = (21,[0,1,2,3,4,5,7,8,9,10,11,12,13,15,18,19,20],[0.09
30416233965019,0.013304096231444532,1.1285281313132651E-4,8.650752203352032E-6,0.04917442300324782
,0.23748327095095423,0.009071932897012337,0.002798178516472281,0.022298534287990248,0.001455849936
1140569,0.0011168671138195988,2.6140532740498797E-5,0.0037646376196103044,9.823442880572638E-4,0.4
639740653816552,0.10134704186465804,3.949041438706183E-5])
featureMetadata: org.apache.spark.sql.types.Metadata = {"ml_attr":{"attrs":{"numeric":[{"idx":0,"n
ame":"boosts"},{"idx":1,"name":"damageDealt"},{"idx":2,"name":"DBNOs"},{"idx":3,"name":"headshotKi
lls"},{"idx":4,"name":"heals"},{"idx":5,"name":"killPlace"},{"idx":6,"name":"killPoints"},{"idx":7
,"name":"kills"},{"idx":8,"name":"killStreaks"},{"idx":9,"name":"longestKill"},{"idx":10,"name":"m
axPlace"},{"idx":11,"name":"numGroups"},{"idx":12,"name":"revives"},{"idx":13,"name":"rideDistance
"},{"idx":14,"name":"roadKills"},{"idx":15,"name":"swimDistance"},{"idx":16,"name":"teamKills"},{"
idx":17,"name":"vehicleDestroys"},{"idx":18,"name":"walkDistance"},{"idx":19,"name":"weaponsacquir
ed"},{"idx":20,"name":"winpoints"}]},"num_attrs":21}}
```

Duo		
18	0.463974	walkDistance
5	0.237483	killPlace
19	0.101347	weaponsacquired
0	0.093042	boosts
4	0.049174	heals
9	0.022299	longestKill
1	0.013304	damageDealt
7	0.009072	kills
13	0.003765	rideDistance
8	0.002798	killStreaks
10	0.001456	maxPlace
11	0.001117	numGroups
15	0.000982	swimDistance
2	0.000113	DBNOs
20	3.95E-05	winpoints
12	2.61E-05	revives
3	8.65E-06	headshotKills

Based on the Random Forest Feature Importance Output we conclude that walkDistance, killPlace, weaponsacquired, boosts, and heals are the top features affecting the winning place percentage in duo Match Type.

DATA ANALYSIS

Linear Regression on Squad match type

```
sidoza7802@cluster-3cf6-m:~$ cat squadRegression.scala
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1 ='Squad'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"DBNOs",
$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"maxP
lace",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"vehicleDestroys",$"wa
lkDistance",$"weaponsacquired",$"winpoints",$"winorlose")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headshotK
ills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","revives","r
ideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDistance","weaponsacqui
red","winpoints","winorlose")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val lr = new LinearRegression()
val lrModel = lr.fit(data1)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

USING THE NEWLY CREATED COLUMN MATCH_TYPE WE HAVE FILTERED OUT THE DATA FOR ONLY SQUAD MATCH TYPE AND RAN THE LINEAR REGRESSION MODEL ON IT.

DATA ANALYSIS

Output of Linear Regression on Squad match type

```
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
lr: org.apache.spark.ml.regression.LinearRegression = linReg_0d9abf2a6f0d
18/12/04 19:04:36 WARN org.apache.spark.util.Utils: Truncated the string representation of a plan
since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' i
n SparkEnv.conf.
18/12/04 19:04:41 WARN org.apache.spark.ml.optim.WeightedLeastSquares: regParam is zero, which mig
ht cause numerical instability and overfitting.
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_0d9abf2a6f0d
Coefficients: [0.004301205810074361,3.2160078313354334E-5,-0.003189545835881143,-0.002109208347099
147,-3.9246069601177173E-4,-0.0017601120100243152,-2.5006570295024655E-5,0.0032849595315685483,-0.
029592724702342154,-1.0689481972102625E-5,0.0028948674519242135,0.003305353203448822,-1.1068249286
08452E-6,-0.0036658843168841286,2.2098766344301068E-5,-0.004587672109748734,0.0011239675110309543,
1.801093380724154E-5,0.0016584052468683812,9.975894378131749E-5,0.2273415314871473] Intercept: -0.
21873440768896854
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark
.ml.regression.LinearRegressionTrainingSummary@658ec00c
numIterations: 1
objectiveHistory: List(0.0)
+--------------------+
|           residuals|
+--------------------+
|-0.03500932957090361|
| 0.04850816859643167|
|-0.10690827751483023|
|0.012264730344171165|
| 0.06914767975293154|
| -0.0943412265370045|
|-0.12907495013485448|
|0.009183580646961897|
| 0.06225895513535673|
| 0.06913047546892778|
| 0.06525540269166696|
|-0.08124749374996099|
| 0.11343078296976847|
|-0.01300958827673...|
|-0.07987237007400538|
|-0.01451261862171...|
|-0.02050029339856252|
|-0.05698193858443...|
|-0.02409108302336138|
|-0.05320576819369549|
+--------------------+
only showing top 20 rows

RMSE: 0.07058021196279532
MSE: 0.0049815663207131156
r2: 0.9516430350956964
```

THE REGRESSION MODEL PROVIDES WITH AN R SQUARE OF 0.95

DATA ANALYSIS

Random Forest Feature Importance Code on Squad

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{ RandomForestRegressor, RandomForestRegressionModel }
import org.apache.spark.ml.feature.StandardScaler
val inputfile = sql("select * from pubg_new where match_type1='Squad'")
val Data=inputfile.select(inputfile("winplaceperc").as("label"),$"boosts",$"damageDealt",$"D
BNOs",$"headshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longest
Kill",$"maxPlace",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"tea
mKills",$"vehicleDestroys",$"walkDistance",$"weaponsacquired",$"winpoints")
val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","hea
dshotKills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace",
"numGroups","revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys
","walkDistance","weaponsacquired","winpoints")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")



val rf = new RandomForestRegressor
val model: RandomForestRegressionModel = rf.fit(data1)
// GET FEATURE IMPORTANCE
val featImp = model.featureImportances
val featureMetadata = data1.schema("features").metadata
```

Using the Random Forest Regressor on SQUAD Match Type to determine the top important features affecting the winning percentage.

DATA ANALYSIS

Random Forest Feature Importance Output for Squad

```
scala> :load FIsquad.scala
Loading FIsquad.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature
import org.apache.spark.ml.regression.RandomForestRegressor
import org.apache.spark.ml.regression.{RandomForestRegressor, RandomForestRegressionModel}
import org.apache.spark.ml.feature.StandardScaler
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 29 more fields]
Data: org.apache.spark.sql.DataFrame = [label: float, boosts: int ... 20 more fields]
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_15c6c069d51e
data1: org.apache.spark.sql.DataFrame = [label: float, features: vector]
rf: org.apache.spark.ml.regression.RandomForestRegressor = rfr_2bc689712fcf
model: org.apache.spark.ml.regression.RandomForestRegressionModel = RandomForestRegressionModel (u
id=rfr_2bc689712fcf) with 20 trees
featImp: org.apache.spark.ml.linalg.Vector = (21,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,15,16,18,19,20],
[0.08772951219898695,0.013767458960470606,0.002883593440906411,3.058728503380063E-4,0.082406564411
08327,0.21048234980897304,6.109658719793594E-4,0.008816682311641096,0.0031640568809232043,0.022366
448285680282,0.004634633866578246,0.0027040522363098074,3.713717802393338E-5,0.003811941179330916,
1.427152732839956E-4,1.986168088382816E-6,0.4933537972116837,0.06249161799787737,2.886138678413781
E-4])
featureMetadata: org.apache.spark.sql.types.Metadata = {"ml_attr":{"attrs":{"numeric":[{"idx":0,"n
ame":"boosts"},{"idx":1,"name":"damageDealt"},{"idx":2,"name":"DBNOs"},{"idx":3,"name":"headshotKi
lls"},{"idx":4,"name":"heals"},{"idx":5,"name":"killPlace"},{"idx":6,"name":"killPoints"},{"idx":7
,"name":"kills"},{"idx":8,"name":"killStreaks"},{"idx":9,"name":"longestKill"},{"idx":10,"name":"m
axPlace"},{"idx":11,"name":"numGroups"},{"idx":12,"name":"revives"},{"idx":13,"name":"rideDistance
"},{"idx":14,"name":"roadKills"},{"idx":15,"name":"swimDistance"},{"idx":16,"name":"teamKills"},{"
idx":17,"name":"vehicleDestroys"},{"idx":18,"name":"walkDistance"},{"idx":19,"name":"weaponsacquir
ed"},{"idx":20,"name":"winpoints"}]},"num_attrs":21}}
```

Squad		
18	0.493354	walkDistance
5	0.210482	killPlace
0	0.08773	boosts
4	0.082407	heals
19	0.062492	weaponsacquired
9	0.022366	longestKill
1	0.013767	damageDealt
7	0.008817	kills
10	0.004635	maxPlace
13	0.003812	rideDistance
8	0.003164	killStreaks
2	0.002884	DBNOs
11	0.002704	numGroups
6	0.000611	killPoints
3	0.000306	headshotKills
20	0.000289	winpoints
15	0.000143	swimDistance
12	3.71E-05	revives
16	1.99E-06	teamKills

Based on the Random Forest Feature Importance Output we conclude that walkDistance, killPlace,, boosts, heals, and weaponsacquired are the top features affecting the winning place percentage in a squad Match Type.

DATA ANALYSIS

Comparison Analysis

Full			Solo			Duo			Squad		
18	0.464991697	walkDistance	18	0.461549	walkDistance	18	0.463974	walkDistance	18	0.493354	walkDistance
5	0.23146347	killPlace	5	0.230317	killPlace	5	0.237483	killPlace	5	0.210482	killPlace
19	0.108077854	weaponsacquired	19	0.149	weaponsacquired	19	0.101347	weaponsacquired	0	0.08773	boosts
0	0.089772839	boosts	0	0.064722	boosts	0	0.093042	boosts	4	0.082407	heals
4	0.048970682	heals	4	0.038062	heals	4	0.049174	heals	19	0.062492	weaponsacquired
9	0.022266048	longestKill	9	0.022705	longestKill	9	0.022299	longestKill	9	0.022366	longestKill
1	0.012744463	damageDealt	1	0.016802	damageDealt	1	0.013304	damageDealt	1	0.013767	damageDealt
11	0.005133947	numGroups	7	0.007868	kills	7	0.009072	kills	7	0.008817	kills
7	0.004300332	kills	11	0.003066	numGroups	13	0.003765	rideDistance	10	0.004635	maxPlace
10	0.004075505	maxPlace	13	0.002033	rideDistance	8	0.002798	killStreaks	13	0.003812	rideDistance
8	0.003649377	killStreaks	8	0.001665	killStreaks	10	0.001456	maxPlace	8	0.003164	killStreaks
13	0.003086173	rideDistance	10	0.001371	maxPlace	11	0.001117	numGroups	2	0.002884	DBNOs
15	9.42E-04	swimDistance	15	0.000721	swimDistance	15	0.000982	swimDistance	11	0.002704	numGroups
6	2.72E-04	killPoints	20	8.16E-05	winpoints	2	0.000113	DBNOs	6	0.000611	killPoints
2	1.95E-04	DBNOs	3	3.66E-05	headshotKills	20	3.95E-05	winpoints	3	0.000306	headshotKills
20	5.87E-05	winpoints				12	2.61E-05	revives	20	0.000289	winpoints
						3	8.65E-06	headshotKills	15	0.000143	swimDistance
									12	3.71E-05	revives
									16	1.99E-06	teamKills

COMPARING THE FEATURES OF WINNING IN VARIOUS GAME TYPES, WE UNDERSTAND THAT THE FEATURES MORE OR LESS REMAIN THE SAME IN VARIOUS GAME TYPES WITH SOME MINOR CHANGES. PLAYING IN SQUAD HAS SOME TOP FEATURES AND WEIGHTAGE. WEIGHTAGE OF FEATURES LIKE WALKDISTANCE AND HEALS IS MORE, AND KILLPLACE AND WEAPONS ACQUIRED IS LESS AS COMPARED TO OTHER GAME MODES. THE RANKING OF WEAPONS ACQUIRED IS ALSO LESS AND FEATURES AT 5TH PLACE AS COMPARED TO 2ND PLACE IN THE OTHER GAME TYPES. THIS CAN BE LARGELY ATTRIBUTED TO HIGH DEPENDENCY ON FRIENDS FOR AMMUNITION IN SQUAD GAME TYPE.

“How do we catch the cheaters in the game?”

APPROACH

USING VARIOUS LOGICAL CONDITIONS BASED ON GAME KNOWLEDGE TO DETERMINE CHEATERS IN THE GAME.

Columns Used

WINPLACEPERC, KILLS, RIDE DISTANCE, WALK DISTANCE

Data Pre-processing

None

Tool Used

Hive

DATA ANALYSIS

Data Preprocessing

```
set hive.cli.print.header=true;
select avg(ridedistance) as Average_RD, min(ridedistance) as min_RD, max(ridedistance) as Max_RD,
variance(ridedistance) as variance, stddev_pop(ridedistance) as Standard_Deviation,
corr(ridedistance,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(swimdistance) as Average_SD, min(swimdistance) as min_SD, max(swimdistance) as Max_swimdistance,
variance(swimdistance) as variance, stddev_pop(swimdistance) as Standard_Deviation,
corr(swimdistance,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(walkdistance) as Average_WD, min(walkdistance) as min_WD, max(walkdistance) as Max_WD,
variance(walkdistance) as variance, stddev_pop(walkdistance) as Standard_Deviation,
corr(walkdistance,winplaceperc) as Correlation from pubg_new ;
```

```
OK
average_wd      min_wd  max_wd  variance        standard_deviation      correlation
1054.8548704988552      0       17300   1246144.9360084352      1116.3086204130268      0.8118704234271266
Time taken: 30.535 seconds, Fetched: 1 row(s)
```

```
OK
average_sd      min_sd  max_swimdistance        variance        standard_deviation      correlation
4.105070850629835       0       5286    756.543933843444        27.50534373250849       0.15423533073988493
Time taken: 30.543 seconds, Fetched: 1 row(s)
```

```
OK
average_rd      min_rd  max_rd  variance        standard_deviation      correlation
423.8472562134295       0       48390   1495544.3741498112      1222.9245169469011      0.30120086364670007
Time taken: 29.473 seconds, Fetched: 1 row(s)
```

	RIDE DISTANCE	SWIM DISTANCE	WALK DISTANCE
Max	48390	5286	17300
Min	0	0	0
Average	423.84	4.11	1054.85
Standard Deviation	1222.92	27.50	1116.30
Variance	1495544	756.54	1246144
Missing Values	0	0	0
Correlation with win percentage	**0.30**	**0.15**	**0.81**

DATA ANALYSIS

Finding cheaters using logical conditions

```
sidoza7802@cluster-3cf6-m:~$ hive -e "set hive.cli.print.header=true;select id, kills, walkd
istance, ridedistance, winplaceperc from pubg_new where kills >0 AND walkdistance = 0 AND ri
dedistance =0 AND winplaceperc > 0.80 limit 10"

Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-2.1.1.jar!
/hive-log4j2.properties Async: true
OK
id      kills   walkdistance    ridedistance    winplaceperc
1777    12      0       0       1.0
3405    15      0       0       1.0
4609    4       0       0       1.0
7123    4       0       0       0.875
16765   6       0       0       1.0
19584   1       0       0       0.8571
19746   9       0       0       1.0
24601   4       0       0       0.875
24988   2       0       0       1.0
31658   4       0       0       0.9048
Time taken: 2.556 seconds, Fetched: 10 row(s)
```

THE GAME HAS BEEN DESIGNED IN SUCH A WAY THAT WALKING IS AN ESSENTIAL PART OF PLAYING THE GAME. IT IS IMPOSSIBLE THAT ANY PLAYER THAT IS KILLING SOMEONE IS NOT WALKING AT ALL. THE USERS THAT FALL IN THESE CONDITIONS ARE PROBABLY CHEATING OR IT IS A GLITCH IN THE DATA.

DATA ANALYSIS

Finding cheaters using logical conditions

```
sidoza7802@cluster-3cf6-m:~$ hive -e "set hive.cli.print.header=true; select id, kills, weaponsacq
uired,winplaceperc from pubg_new where weaponsacquired = 0 AND kills > 0 order by kills desc limit
 10"
```

```
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 18.27 sec   HDFS Read: 440563747 HDFS Write: 39
3 SUCCESS
Total MapReduce CPU Time Spent: 18 seconds 270 msec
OK
id      kills   weaponsacquired winplaceperc
455859  19      0       0.9643
2043714 14      0       0.6154
2722982 13      0       0.3333
2111015 11      0       0.6154
1706857 10      0       0.6538
1504462 8       0       1.0
3976520 8       0       0.24
5327065 7       0       0.9615
4439112 7       0       0.1667
823866  7       0       0.32
Time taken: 28.676 seconds, Fetched: 10 row(s)
```

THE GAME IS ESSENTIALLY WON BY KILLING MORE PEOPLE AND BEING THE LAST ONE STANDING. WEAPONS HELP THE PLAYERS DO THAT. IT IS NEARLY IMPOSSIBLE TO KILL SO MANY PEOPLE AND COME CLOSE TO WINNING WITHOUT ACQUIRING ANY WEAPONS. THESE PLAYERS APPEAR TO BE CHEATERS IN THE GAME.

DATA ANALYSIS

Finding cheaters using logical conditions

```
sidoza7802@cluster-3cf6-m:~$ hive -e "set hive.cli.print.header=true; select id, kills,winplaceper
c from pubg_new where  kills = 0 and winplaceperc=1 and match_type1='solo' order by winplaceperc =
 1 desc limit 10"
```

```
4 SUCCESS
Total MapReduce CPU Time Spent: 18 seconds 230 msec
OK
id      kills   winplaceperc
5269946 0       1.0
4961876 0       1.0
3402803 0       1.0
3273379 0       1.0
2978650 0       1.0
1531571 0       1.0
1464768 0       1.0
149774  0       1.0
Time taken: 29.095 seconds, Fetched: 8 row(s)
```

THE GAME ENDS BY KILLING THE LAST OPPONENT PLAYING THE GAME. THE LAST PERSON HAS TO KILL A PERSON TO END THE GAME. THUS, THE PLAYERS WHO ARE WINNING THE GAME WITHOUT KILLING ANYONE ARE LIKELY TO BE CHEATERS.

“How does the weapon acquisition strategy differ for players in different clusters?”

APPROACH

FORM CLUSTERS OF DATA USING CLUSTERING ALGORITHM/LOGICAL DIVISION. RUN ANOVA TO DETERMINE IF THE WEAPON ACQUISITION DIFFERS SIGNIFICANTLY IN DIFFERENT CLUSTERS OF THE DATA.

Columns Used

All Columns

Data Pre-processing

Create data clusters.

Tool Used

Hive, Spark

DATA ANALYSIS

Data Preprocessing

```
set hive.cli.print.header=true;
select avg(longestkill) as Average_LK, min(longestkill) as min_LK, max(longestkill) as Max_LK,
variance(longestkill) as variance, stddev_pop(longestkill) as Standard_Deviation,
corr(longestkill,winplaceperc) as Correlation from pubg_new ;
```

```
set hive.cli.print.header=true;
select avg(teamkills) as Average_TK, min(teamkills) as min_TK, max(teamkills) as Max_TK,
variance(teamkills) as variance, stddev_pop(teamkills) as Standard_Deviation,
corr(teamkills,winplaceperc) as Correlation from pubg_new ;

set hive.cli.print.header=true;
select avg(weaponsacquired) as Average_WA, min(weaponsacquired) as min_WA, max(weaponsacquired) as Max_WA,
variance(weaponsacquired) as variance, stddev_pop(weaponsacquired) as Standard_Deviation,
corr(weaponsacquired,winplaceperc) as Correlation from pubg_new ;
```

```
OK
average_lk      min_lk  max_lk  variance        standard_deviation      correlation
19.669181353010188      0       1323    2093.3046418477325      45.75264628245816       0.404875715899583
Time taken: 29.977 seconds, Fetched: 1 row(s)
```

```
OK
average_tk      min_tk  max_tk  variance        standard_deviation      correlation
0.013885548417657026    0       6       0.01766948171509859     0.13292660273661774     -0.006122422708281107
Time taken: 29.486 seconds, Fetched: 1 row(s)
```

```
OK
average_wa      min_wa  max_wa  variance        standard_deviation      correlation
3.457289270324804       0       76      5.770127279524312       2.402108923326399       0.5715205473647011
Time taken: 30.476 seconds, Fetched: 1 row(s)
```

	LONGEST KILL	TEAM KILLS	WEAPONS ACQUIRED
Max	1323	6	76
Min	0	0	0
Average	19.66	0.013	3.45
Standard Deviation	45.75	0.13	2.40
Variance	2093.30	0.017	5.77
Missing Values	0	0	0
Correlation with win percentage	**0.40**	**-0.006**	**0.57**

DATA ANALYSIS

Creating new column "WINQUARTILES" and validating it.

```
set hive.cli.print.header=true;
--ALTER TABLE pubg_new ADD COLUMNS (WinQuartiles Int);
INSERT OVERWRITE TABLE pubg_new
SELECT
        Id ,
        groupId ,
        matchId ,
        assists ,
        boosts ,
       `damageDealt` ,
       `DBNOs`,
       `headshotKills` ,
       `heals` ,
       `killPlace` ,
       `killPoints`,
       `kills` ,
       `killStreaks` ,
       `longestKill`,
       `maxPlace` ,
       `numGroups` ,
       `revives`,
       `rideDistance` ,
       `roadKills` ,
       `swimDistance` ,
       `teamKills` ,
       `vehicleDestroys` ,
       `walkDistance` ,
       `weaponsAcquired` ,
       `winPoints` ,
       `winPlacePerc`, match_type,WinORLose,
      if(winplaceperc >= 0.75, 4,if (winplaceperc >= 0.50 AND winplaceperc < 75 ,3,if(winplaceperc >=0.25 AND winplaceperc < 0.50, 2,1)))
as WinQuartiles from pubg_new;
```

```
Logging initialized using configuration in jar
OK
NULL    1
0.8571  4
0.04    1
0.7407  3
0.1146  1
Time taken: 2.074 seconds, Fetched: 5 row(s)
```

```
sidoza7802@cluster-3cf6-m:~$ hive -e "set hive.cli.print.header=true;select WinQuartiles, sum(weaponsacquired)as sum,count(WinQuartiles) as count from pubg_new group by WinQuartiles order by WinQu
artiles;"
```

```
OK
winquartiles    sum     count
1       2177340 1299535
2       3212748 1023044
3       4212211 960479
4       5462272 1074279
Time taken: 53.794 seconds, Fetched: 4 row(s)
```

DATA ANALYSIS

K-Means Clustering Algorithm Code

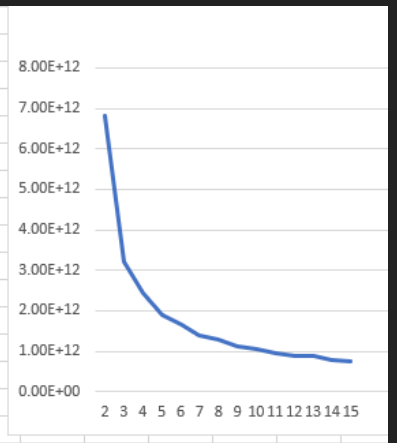
```
sidoza7802@cluster-3cf6-m:~$ cat cluster.scala
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator



val inputfile = sql("select * from pubg_new")
val Data=inputfile.select(inputfile("winorlose").as("label"),$"boosts",$"damageDealt",$"DBNOs",$"h
eadshotKills",$"heals",$"killPlace",$"killPoints",$"kills",$"killStreaks",$"longestKill",$"maxPlac
e",$"numGroups",$"revives",$"rideDistance",$"roadKills",$"swimDistance",$"teamKills",$"vehicleDest
roys",$"walkDistance",$"weaponsacquired",$"winpoints",$"winorlose",$"winquartiles")

val assembler = new VectorAssembler().setInputCols(Array("boosts","damageDealt","DBNOs","headshotK
ills","heals","killPlace","killPoints","kills","killStreaks","longestKill","maxPlace","numGroups",
"revives","rideDistance","roadKills","swimDistance","teamKills","vehicleDestroys","walkDistance","
weaponsacquired","winpoints","winorlose","winquartiles")).setOutputCol("features")
val data1 = assembler.transform(Data).select($"label",$"features")
val kmeans = new KMeans().setPredictionCol("cluster").setFeaturesCol("features").setK(5).setInitSt
eps(40).setMaxIter(99)
val kmodel = kmeans.fit(data1)
println(s"3,${kmodel.computeCost(data1)}")
println("Cluster centroids:")
kmodel.clusterCenters.foreach(println)
println(s"$3,${kmodel.computeCost(data1)}")
val predictions = kmodel.summary.predictions
predictions.orderBy("cluster").show()
predictions.count()
```

DATA ANALYSIS

K-Means Clustering Algorithm Output

```
scala> :load cluster.scala
Loading cluster.scala...
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary
import org.apache.spark.ml.clustering.KMeans
<console>:30: error: object ClusteringEvaluator is not a member of package org.apache.spark.ml.eva
luation
       import org.apache.spark.ml.evaluation.ClusteringEvaluator
              ^
ivysettings.xml file not found in HIVE_HOME or HIVE_CONF_DIR,/etc/hive/conf.dist/ivysettings.xml w
ill be used
inputfile: org.apache.spark.sql.DataFrame = [id: int, groupid: int ... 27 more fields]
Data: org.apache.spark.sql.DataFrame = [label: int, boosts: int ... 22 more fields]
2,6.815022790882889E12
3,3.2018018911103477E12
4,2.444465191619248E12
5,1.883135845773215E12
6,1.6731056800164707E12
7,1.403788224224326E12
8,1.2962698982431316E12
9,1.1188015136890576E12
10,1.0352321364222639E12
11,9.348031303733436E11
12,8.720674559302911E11
13,8.905427155835315E11
14,7.68770575427449E11
15,7.31907003906131E11
16,7.029192786920854E11
17,6.742171091512225E11
18,6.454963164592404E11
19,6.369627487796655E11
20,6.128988817100745E11
21,5.73722064830303E11
22,5.584435631352262E11
23,5.386933558049576E11
24,5.278286020247146E11
25,5.114777390409711E11
[Stage 6264:=====================>                    (4 + 2) / 10]Connected, host fi
```

2	6.82E+12
3	3.20E+12
4	2.44E+12
5	1.88E+12
6	1.67E+12
7	1.40E+12
8	1.30E+12
9	1.12E+12
10	1.04E+12
11	9.35E+11
12	8.72E+11
13	8.91E+11
14	7.69E+11
15	7.32E+11




```
scala> predictionResult.show()
+-----+--------------------+-------+
|label|            features|cluster|
+-----+--------------------+-------+
|    0|(23,[0,1,2,4,5,6,...|      0|
|    0|(23,[1,5,6,10,11,...|      2|
|    0|(23,[0,4,5,6,10,1...|      2|
|    0|(23,[0,1,5,6,7,8,...|      1|
|    0|(23,[1,5,6,7,8,9,...|      2|
|    1|[1.0,661.0,2.0,3....|      0|
|    0|(23,[0,1,4,5,6,10...|      1|
|    0|(23,[1,5,6,10,11,...|      2|
|    0|[1.0,166.0,0.0,1....|      2|
|    0|(23,[1,5,6,10,11,...|      3|
|    0|(23,[0,1,5,6,7,8,...|      2|
|    0|(23,[0,1,4,5,6,7,...|      0|
|    0|(23,[1,2,5,6,10,1...|      2|
|    0|(23,[0,1,4,5,6,10...|      3|
|    0|(23,[5,6,10,11,18...|      2|
|    0|[6.0,227.0,0.0,1....|      3|
|    0|(23,[1,4,5,6,10,1...|      3|
|    0|(23,[5,6,10,11,18...|      2|
|    0|[1.0,357.0,4.0,0....|      0|
|    0|(23,[1,2,5,6,7,8,...|      2|
+-----+--------------------+-------+
only showing top 20 rows
```

BASED ON THE ELBOW TEST FOR K MEANS CLUSTERING ALGORITHM, WE CAN CONCLUDE THAT THE OPTIMAL NUMBER OF CLUSTERS SHOULD BE 5.

DATA ANALYSIS

Correlation on the K means clusters

Correlation between weapon acquired and winning percentage

SOLO

```
8 SUCCESS
Total MapReduce CPU Time Spent: 16 seconds 940 msec
OK
c0
0.6776675780936803
Time taken: 27.802 seconds, Fetched: 1 row(s)
```

DUO

```
8 SUCCESS
Total MapReduce CPU Time Spent: 17 seconds 710 msec
OK
c0
0.6209369658853686
Time taken: 29.641 seconds, Fetched: 1 row(s)
```

SQUAD

```
8 SUCCESS
Total MapReduce CPU Time Spent: 17 seconds 150 msec
OK
c0
0.3940380667200964
Time taken: 28.579 seconds, Fetched: 1 row(s)
```

BASED ON THE CORRELATION OUTPUT WE CAN CONCLUDE THAT THE IMPACT OF WEAPON ACQUISITION STRATEGY ON WINNING PERCENTAGE GOES ON DECREASING AS THE NUMBER OF PLAYERS IN THE GAME GO ON INCREASING . IT INITIALLY DECREASES BY LITTLE FROM SOLO TO DUO BUT THE DECREASE IS SIGNIFICANT AS WE GO TOWARDS SQUAD.

Thank You.