

Battle Royale Game Analysis

BUDT758B_FINAL_PROJECT



Team name: GGWP
Tzu-Chieh Chen
Yu-Chen Chen
2021.5.10



Data storage strategy

AWS S3 : Upload the file in S3. In order to let group members download the files, we make the bucket and files publicly accessible.

Amazon S3 > 758bproject

758bproject

Publicly accessible

Objects Properties Permissions Metrics Management Access Points

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy URL Open Download Delete Actions Create folder Upload

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	agg_match_stats_4.csv	csv	April 29, 2021, 23:41:28 (UTC+08:00)	1.7 GB	Standard
<input type="checkbox"/>	kill_match_stats_final_4.csv	csv	April 30, 2021, 00:13:39 (UTC+08:00)	1.6 GB	Standard

Amazon S3 > 758bproject > agg_match_stats_4.csv

Copy S3 URI Object actions

Properties Permissions Versions

Access control list (ACL)

Grant basic read/write permissions to AWS accounts. [Learn more](#)

Grantee	Object	Object ACL
Object owner Canonical ID: 36517ed429ff415f32159b5abb23e8c2d79cd4e0062f9793dfc241aa05d97eaf	Read	Read, Write
Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers	Read	Read
Authenticated users group (anyone with an AWS account) Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers	Read	Read

Running queries on Databricks

First step - download the data from AWS S3

```
1 %ssh
2
3 #download files to the free version of Databricks
4
5 wget https://758bproject.s3.amazonaws.com/agg_match_stats_4.csv && wget https://758bproject.s3.amazonaws.com/kill_match_stats_final_4.csv
6
```

Create two schemas:

Cmd 3

```
1 %python
2 from pyspark.sql.types import *
3 agg_Schema = StructType([StructField("date", DateType(), True),
4                               StructField("game_size", IntegerType(), True),
5                               StructField("match_id", StringType(), True),
6                               StructField("match_mode", StringType(), True),
7                               StructField("party_size", IntegerType(), True),
8                               StructField("player_assists", IntegerType(), True),
9                               StructField("player_dbno", IntegerType(), True),
10                              StructField("player_dist_ride", DoubleType(), True),
11                              StructField("player_dist_walk", DoubleType(), True),
12                              StructField("player_dmg", IntegerType(), True),
13                              StructField("player_kills", IntegerType(), True),
14                              StructField("player_name", StringType(), True),
15                              StructField("player_survive_time", DoubleType(), True),
16                              StructField("team_id", IntegerType(), True),
17                              StructField("team_placement", IntegerType(), True)])
```

```
23 kill_Schema = StructType([StructField("killed_by", StringType(), True),
24                               StructField("killer_name", StringType(), True),
25                               StructField("killer_placement", DoubleType(), True),
26                               StructField("killer_position_x", DoubleType(), True),
27                               StructField("killer_position_y", DoubleType(), True),
28                               StructField("map", StringType(), True),
29                               StructField("match_id", StringType(), True),
30                               StructField("time", IntegerType(), True),
31                               StructField("victim_name", StringType(), True),
32                               StructField("victim_placement", DoubleType(), True),
33                               StructField("victim_position_x", DoubleType(), True),
34                               StructField("victim_position_y", DoubleType(), True)])
35
```

Descriptive Analysis

Explore the data by using SQL queries:

We want to find Top 10 Winner's weapons:


```
Cmd 4

1 from pyspark.sql.functions import col, column
2 kill_DF_withSchema.createOrReplaceTempView("kill_DF")
3 sqlDF = spark.sql("SELECT killed_by AS Weapon, COUNT(*) AS Number_of_usage \
4                   FROM (SELECT * FROM kill_DF WHERE killer_placement = 1.0) \
5                   GROUP BY killed_by \
6                   HAVING killed_by <> 'Down and Out' \
7                   ORDER BY 2 DESC \
8                   LIMIT 10")
9 sqlDF.show()
```

▶ (2) Spark Jobs

■ sqlDF: pyspark.sql.dataframe.DataFrame = [Weapon: string, Number_of_usage: long]

Weapon	Number_of_usage
M416	215679
SCAR-L	160044
M16A4	114191
AKM	100135
Kar98k	68766
Mini 14	53903
SKS	41609
UMP9	41063
Grenade	28756
S1897	17344



Fun Fact: **"Stay away from the grenade you just threw!"**

We want to find Top 10 player suicide weapons:


```
Cmd 5

1 sqlDF2 = spark.sql("SELECT killed_by AS Weapon, COUNT(*) AS count \
2                   FROM (SELECT * FROM kill_DF WHERE killer_name = victim_name) \
3                   GROUP BY killed_by \
4                   HAVING killed_by <> 'Down and Out' \
5                   ORDER BY 2 DESC \
6                   LIMIT 10")
7 sqlDF2.show()
```

▶ (2) Spark Jobs

■ sqlDF2: pyspark.sql.dataframe.DataFrame = [Weapon: string, count: long]

Weapon	count
Grenade	76936
Hit by Car	20030
Falling	8882
Motorbike (SideCar)	7519
Motorbike	6911
death.ProjMolotov...	5731
Uaz	5128
Buggy	2961
Bluezone	2795
Dacia	2214



GRENADE THROWN!



Death map

Partial Code

```
from scipy.ndimage.filters import gaussian_filter
import matplotlib.cm as cm
from matplotlib.colors import Normalize
plot_data_ev = evdf[['x','y']].values
plot_data_ek = ekdf[['x','y']].values
#match with the map
plot_data_ev = plot_data_ev * 4040 / 800000
plot_data_ek = plot_data_ek * 4040 / 800000
def heatmap(x, y, s, bins = 100):
    # x = plot_data_ev[:,0]
    # y = plot_data_ev[:,1]
    # s = 1.5
    # bins = 800
    heatmap, xedges, yedges = np.histogram2d(x, y, bins = bins)
    heatmap = gaussian_filter(heatmap, sigma = s)
    extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
    return heatmap.T, extent
hmap, extent = heatmap(plot_data_ev[:,0], plot_data_ev[:,1], 1.5, bins = 800)
alphas = np.clip(Normalize(0, hmap.max()/100, clip=True)(hmap)*1.5, 0.0, 1.)
colors = Normalize(hmap.max()/100, hmap.max()/20, clip=True)(hmap)
colors = cm.bwr(colors)
colors[:, :, -1] = alphas
#plot
fig, ax = plt.subplots(figsize = (24,24))
ax.set_xlim(0, 4096);ax.set_ylim(0, 4096)
ax.imshow(bg)
ax.imshow(colors, extent = extent, origin = 'lower', cmap = cm.bwr, alpha = 1)
#ax.imshow(colors2, extent = extent2, origin = 'lower', cmap = cm.RdBu, alpha = 0.5)
plt.gca().invert_yaxis()
plt.title('Erangel Death Map')
```

CATURNER'S PUBG MAPS

June 2017

<https://www.reddit.com/user/caturner/>

Predictive Analysis

We created a new column “top10” and assigned 1 to top 10 players in each match; Otherwise, 0.

Because winners may have similar performance in the game, we decided to use KNN to cluster them and predict the final top 10 winners in the match.

Features we picked are: “player assists”, “player downs”, “player ride distance”, “player walk distance”, “player damage output”, “player kills”, all of which measure player performance in a match.

Because of processing speed, we reduced the data to 300,000 instances for running model.

Feature scaling to normalize the different number scales in features.

Using the python ML package “scikit-learn” to create a KNN model and predict player final placement in a match.

Predict accuracy: 85.54%

Cmd 14

```
1 agg_df2= agg_df.dropna(axis = 0, how = 'any') #drop any rows with NA's
2 kill_df2 = kill_df.dropna(axis = 0, how = 'any') #drop any rows with NA's
3 conditions = [agg_df2['team_placement'] <= 10.0, agg_df2['team_placement'] > 10.0]
4 choices = [1, 0]
5 agg_df2['top10'] = np.select(conditions, choices)
```

Cmd 16

```
1 X = agg_df2.iloc[:300000, [5,6,7,8,9,10]].values
2 y = agg_df2.iloc[:300000, -1].values
3
```

Cmd 17

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

Command took 0.10 seconds -- by tchen114@umd.edu at 5/9/2021, 9:31:13 AM on Mycluster

Cmd 18

```
1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 X_train = sc.fit_transform(X_train)
4 X_test = sc.transform(X_test)
```

Command took 0.14 seconds -- by tchen114@umd.edu at 5/9/2021, 9:31:16 AM on Mycluster

Cmd 19

```
1 from sklearn.neighbors import KNeighborsClassifier
2 knn = KNeighborsClassifier(n_neighbors = 30, metric = 'minkowski', p = 2)
3 knn.fit(X_train, y_train)
4 y_pred = knn.predict(X_test)
```

Command took 1.47 minutes -- by tchen114@umd.edu at 5/9/2021, 9:34:45 AM on Mycluster

Cmd 20

```
1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_pred)
3 print(cm)
4 accuracy_score(y_test, y_pred)
```

```
[[60927 5112]
 [ 7899 16062]]
Out[32]: 0.8554333333333334
```

