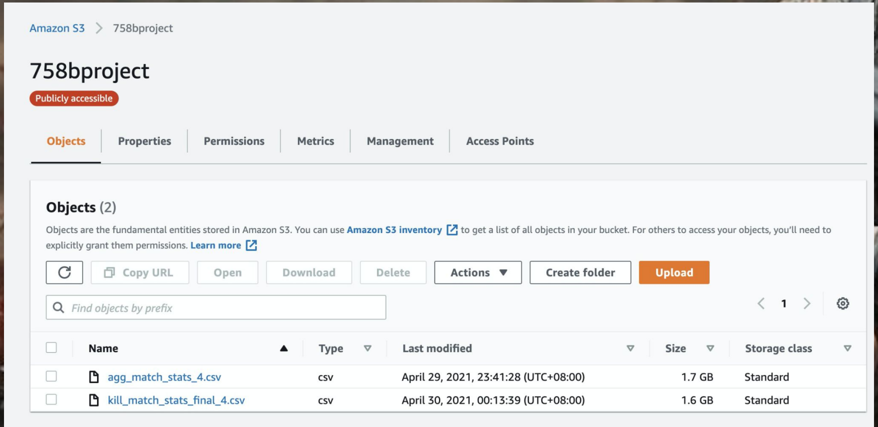
Battle Royale Game Analysis

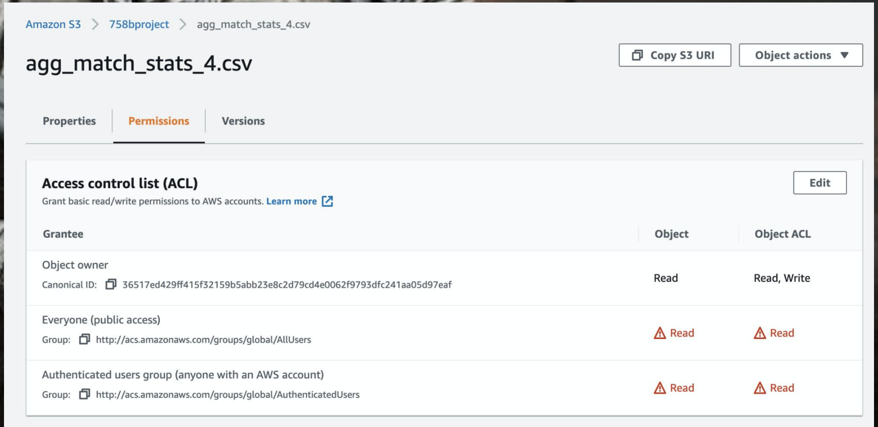
BUDT758B_FINAL_PROJECT

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.





Running queries on Databricks

First step - download the data from AWS S3

```
%sh

#download files to the free version of Databricks

wget https://758bproject.s3.amazonaws.com/agg_match_stats_4.csv && wget https://758bproject.s3.amazonaws.com/kill_match_stats_final_4.csv
```

Create two schemas:

```
Cmd 3

%python
from pyspark.sql.types import *
agg_Schema = StructType([StructField("date", DateType(), True),
                         StructField("game_size", IntegerType(), True),
                         StructField("match_id", StringType(), True),
                         StructField("match_mode", StringType(), True),
                         StructField("party_size", IntegerType(), True),
                         StructField("player_assists", IntegerType(), True),
                         StructField("player_dbno", IntegerType(), True),
                         StructField("player_dist_ride", DoubleType(), True),
                         StructField("player_dist_walk", DoubleType(), True),
                         StructField("player_dmg", IntegerType(), True),
                         StructField("player_kills", IntegerType(), True),
                         StructField("player_name", StringType(), True),
                         StructField("player_survive_time", DoubleType(), True),
                         StructField("team_id", IntegerType(), True),
                         StructField("team_placement", IntegerType(), True)])
```

```
kill_Schema = StructType([StructField("killed_by", StringType(), True),
                          StructField("killer_name", StringType(), True),
                          StructField("killer_placement", DoubleType(), True),
                          StructField("killer_position_x", DoubleType(), True),
                          StructField("killer_position_y", DoubleType(), True),
                          StructField("map", StringType(), True),
                          StructField("match_id", StringType(), True),
                          StructField("time", IntegerType(), True),
                          StructField("victim_name", StringType(), True),
                          StructField("victim_placement", DoubleType(), True),
                          StructField("victim_position_x", DoubleType(), True),
                          StructField("victim_position_y", DoubleType(), True)])
```

Descriptive Analysis

Explore the data by using SQL queries:

We want to find Top 10 Winner's weapons:

Cmd 4

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

```
sqlDF2 = spark.sql("SELECT killed_by AS Weapon, COUNT(*) AS count \
                   FROM (SELECT * FROM kill_DF WHERE killer_name = victim_name) \
                   GROUP BY killed_by \
                   HAVING killed_by <> 'Down and Out' \
                   ORDER BY 2 DESC \
                   LIMIT 10")
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|
+--------------------+-----+
```

Death map

Partial Code

```python
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')
```

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

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

Cmd 16

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

```

Cmd 17

```
from sklearn.model_selection import train_test_split
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

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
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

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 30, metric = 'minkowski', p = 2)
knn.fit(X_train, y_train)
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

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

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