Mushfiq Alam

CIS 4130 PTRA

Dr. Holowczak

Clash Royale S18 Ladder Dataset

Milestone 1/ Proposal

URL: Link

Description: The "Clash Royale S18 Ladder Dataset" reflects match data on the mobile game "Clash Royale". It includes the performance metrics for competing players, player levels/rarity of card decks, amount of damage taken, and average spending to summon troops.

Prediction intention: Determining the victor based on user standing.

Model Choice: Logistic Regression

Detail about Columns and Technical indicators used:

1) BattleTime - Duration of Battle
2) Arena.id - Specific rank at which player is competing
3) GameMode.id - Identifier of the game mode (competitive, league, event)
4) Average.startingTrophies - Average amount of trophies (cumulative wins) had at the start of the battle
5) Winner.tag - Unique ID of the winner player
6) Winner.startingTrophies - Numbers of trophies that winner player prior to the battle
7) Winner.trophyChange - Change in the number for the winner after the battle
8) Winner.crowns - Number of crowns (destroyed enemy towers) earned by winner
9) Winner.kingTowerHitPoints - Remaining health of the winning player's King Tower

10) Winner.princessTowersHitPoints - Remaining health of the winner's Princess Towers (both, if applicable)

11) Winner.clan.tag - Clan tag associated with the winning player

12) Winner.clan.badgeId - Badge ID representing the winner’s clan

13) Loser.tag - Unique ID of the player who lost the battle

14) Loser.startingTrophies - Number of trophies the losing player had before the battle

15) Loser.trophyChange - Change in the number of trophies for the loser after the battle

16) Loser.crowns - Number of crowns earned by the loser

17) Loser.kingTowerHitPoints - Remaining health of the losing player’s King Tower (can be zero)

18) Loser.clan.tag - Remaining health of the losing player's Princess Towers (if any survived)

19) Loser.clan.badgeId - Clan tag associated with the losing player

20) Loser.princessTowersHitPoints - Badge ID representing the loser’s clan

21) TournamentTag - Tag or ID of tournament in which the battle occurred, if applicable.

<u>Card and Deck Data for Winner and Loser:</u>

22) Winner/Loser.cardX.id - Unique ID for each of the cards in the player’s deck, where X refers to the card’s position in the 8-card deck (e.g., card1.id, card2.id)

23) Winner/Loser.cardX.level - Level of each card in the player’s deck (where X is the card's position)

24) Winner/Loser.cards.list: List or summary of all cards in the player’s deck.

25) Winner/Loser.totalcard.level: Average level of all cards in the player’s deck

<u>Card Count Data:</u>

26) Winner/Loser.troop.count - Number of troop cards in the player's deck

27) Winner/Loser.structure.count - Number of building (structure) cards in the player's deck

28) Winner/Loser.spell.count - Number of spell cards in the player's deck

29) Winner/Loser.common.count - The number of common-rarity cards in the player’s deck

30) Winner/Loser.rare.count - Number of rare-rarity cards in the player’s deck

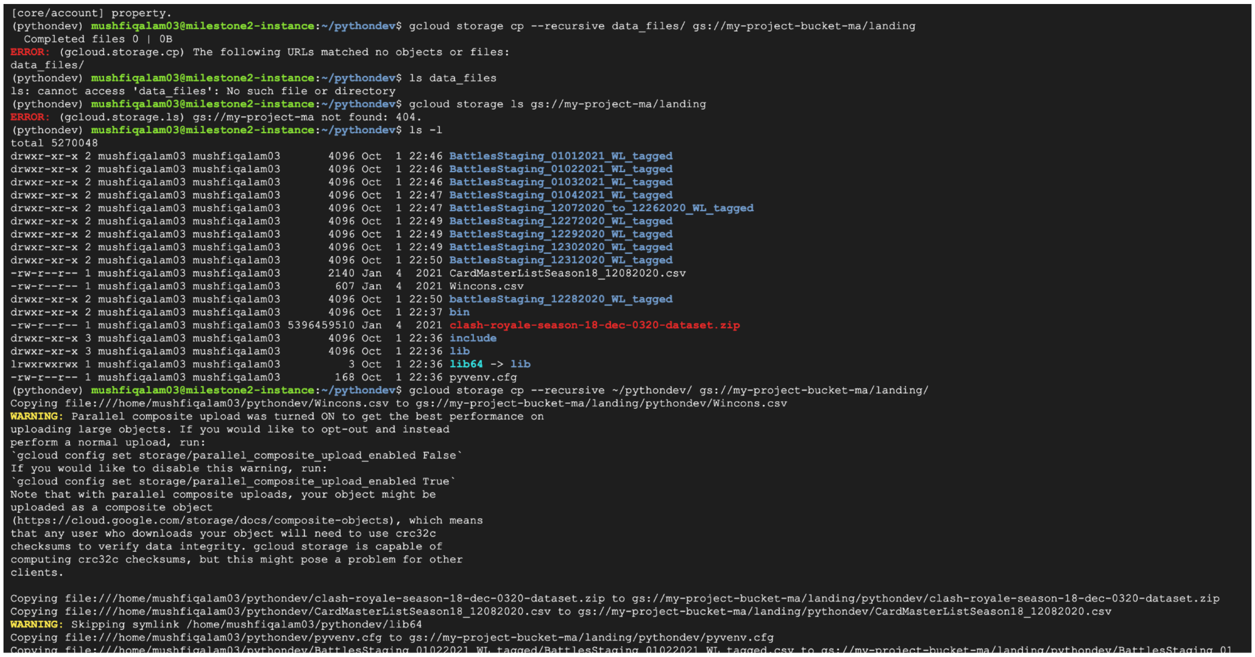
31) Winner/Loser.epic.count - Number of epic-rarity cards in the player’s deck

32) Winner/Loser.legendary.count - Number of legendary-rarity cards in the player’s deck

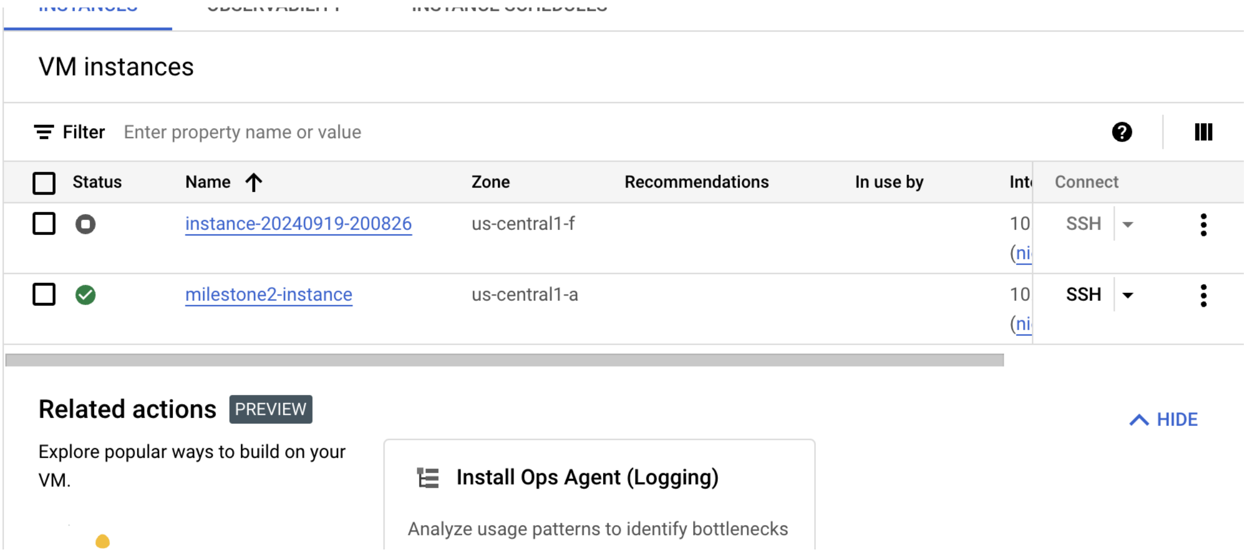
<u>Elixir Information:</u>

33) Winner/Loser.elixir.average: Average elixir cost of the player's entire 8-card deck

Milestone 2/Data Acquisition:



```
[core/account] property.
(pythondev) mushfiqalam03@milestone2-instance:~/pythondev$ gcloud storage cp --recursive data_files/ gs://my-project-bucket-ma/landing
  Completed files 0 | 0B
ERROR: (gcloud.storage.cp) The following URLs matched no objects or files:
data_files/
(pythondev) mushfiqalam03@milestone2-instance:~/pythondev$ ls data_files
ls: cannot access 'data_files': No such file or directory
(pythondev) mushfiqalam03@milestone2-instance:~/pythondev$ gcloud storage ls gs://my-project-ma/landing
ERROR: (gcloud.storage.ls) gs://my-project-ma not found: 404.
(pythondev) mushfiqalam03@milestone2-instance:~/pythondev$ ls -l
total 5270048
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:46 BattlesStaging_01012021_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:46 BattlesStaging_01022021_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:46 BattlesStaging_01032021_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:47 BattlesStaging_01042021_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:47 BattlesStaging_12072020_to_12262020_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:49 BattlesStaging_12272020_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:49 BattlesStaging_12292020_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:49 BattlesStaging_12302020_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:50 BattlesStaging_12312020_WL_tagged
-rw-r--r-- 1 mushfiqalam03 mushfiqalam03       2140 Jan  4  2021 CardMasterListSeason18_12082020.csv
-rw-r--r-- 1 mushfiqalam03 mushfiqalam03        607 Jan  4  2021 Wincons.csv
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:50 battlesStaging_12282020_WL_tagged
drwxr-xr-x 2 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:37 bin
-rw-r--r-- 1 mushfiqalam03 mushfiqalam03 5396459510 Jan  4  2021 clash-royale-season-18-dec-0320-dataset.zip
drwxr-xr-x 3 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:36 include
drwxr-xr-x 3 mushfiqalam03 mushfiqalam03       4096 Oct  1 22:36 lib
lrwxrwxrwx 1 mushfiqalam03 mushfiqalam03          3 Oct  1 22:36 lib64 -> lib
-rw-r--r-- 1 mushfiqalam03 mushfiqalam03        168 Oct  1 22:36 pyvenv.cfg
(pythondev) mushfiqalam03@milestone2-instance:~/pythondev$ gcloud storage cp --recursive ~/pythondev/ gs://my-project-bucket-ma/landing/
Copying file:///home/mushfiqalam03/pythondev/Wincons.csv to gs://my-project-bucket-ma/landing/pythondev/Wincons.csv
WARNING: Parallel composite upload was turned ON to get the best performance on
uploading large objects. If you would like to opt-out and instead
perform a normal upload, run:
`gcloud config set storage/parallel_composite_upload_enabled False`
If you would like to disable this warning, run:
`gcloud config set storage/parallel_composite_upload_enabled True`
Note that with parallel composite uploads, your object might be
uploaded as a composite object
(https://cloud.google.com/storage/docs/composite-objects), which means
that any user who downloads your object will need to use crc32c
checksums to verify data integrity. gcloud storage is capable of
computing crc32c checksums, but this might pose a problem for other
clients.

Copying file:///home/mushfiqalam03/pythondev/clash-royale-season-18-dec-0320-dataset.zip to gs://my-project-bucket-ma/landing/pythondev/clash-royale-season-18-dec-0320-dataset.zip
Copying file:///home/mushfiqalam03/pythondev/CardMasterListSeason18_12082020.csv to gs://my-project-bucket-ma/landing/pythondev/CardMasterListSeason18_12082020.csv
WARNING: Skipping symlink /home/mushfiqalam03/pythondev/lib64
Copying file:///home/mushfiqalam03/pythondev/pyvenv.cfg to gs://my-project-bucket-ma/landing/pythondev/pyvenv.cfg
Copying file:///home/mushfiqalam03/pythondev/BattlesStaging_01022021_WL_tagged/BattlesStaging_01022021_WL_tagged.csv to gs://my-project-bucket-ma/landing/pythondev/BattlesStaging_01
```



VM instances

Filter Enter property name or value

□ Status	Name ↑	Zone	Recommendations	In use by	Int	Connect
□ ●	instance-20240919-200826	us-central1-f			10 (ni	SSH ▾
□ ✓	milestone2-instance	us-central1-a			10 (ni	SSH ▾

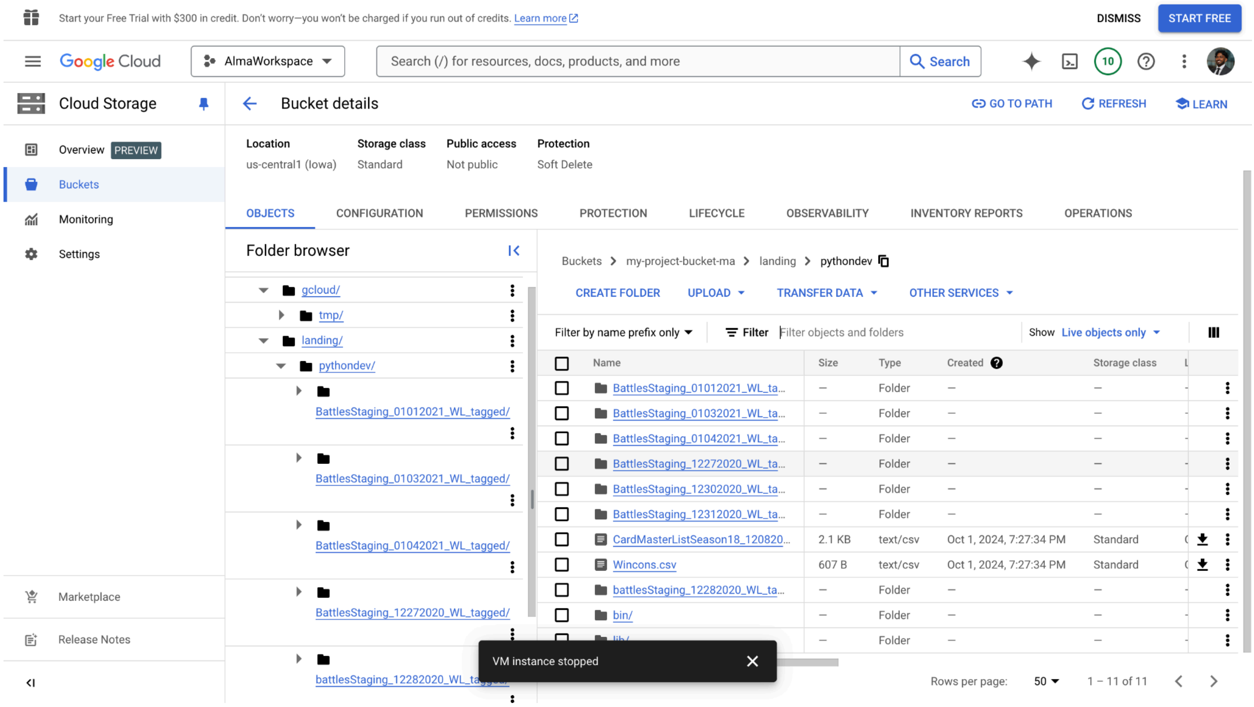
Related actions PREVIEW

HIDE

Explore popular ways to build on your VM.

Install Ops Agent (Logging)

Analyze usage patterns to identify bottlenecks



Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	Soft Delete

OBJECTS
CONFIGURATION
PERMISSIONS
PROTECTION
LIFECYCLE
OBSERVABILITY
INVENTORY REPORTS
OPERATIONS
Folder browser
gcloud/
tmp/
landing/
pythondev/
BattlesStaging_01012021_WL_tagged/
BattlesStaging_01032021_WL_tagged/
BattlesStaging_01042021_WL_tagged/
BattlesStaging_12272020_WL_tagged/
battlesStaging_12282020_WL_tagged/

Buckets > my-project-bucket-ma > landing > pythondev

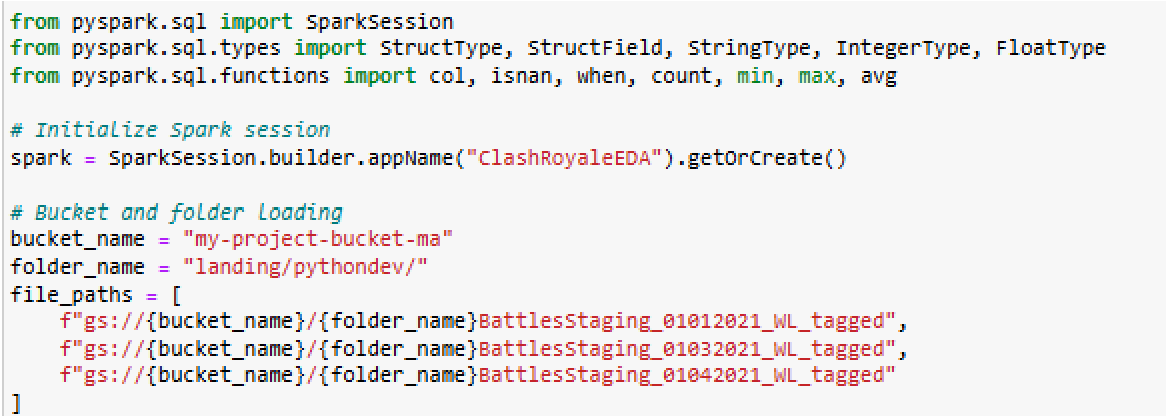
Filter by name prefix only ▾ Filter Filter objects and folders Show Live objects only ▾

□	Name	Size	Type	Created	Storage class
□	BattlesStaging_01012021_WL_ta...	–	Folder	–	–
□	BattlesStaging_01032021_WL_ta...	–	Folder	–	–
□	BattlesStaging_01042021_WL_ta...	–	Folder	–	–
□	BattlesStaging_12272020_WL_ta...	–	Folder	–	–
□	BattlesStaging_12302020_WL_ta...	–	Folder	–	–
□	BattlesStaging_12312020_WL_ta...	–	Folder	–	–
□	CardMasterListSeason18_120820...	2.1 KB	text/csv	Oct 1, 2024, 7:27:34 PM	Standard
□	Wincons.csv	607 B	text/csv	Oct 1, 2024, 7:27:34 PM	Standard
□	battlesStaging_12282020_WL_ta...	–	Folder	–	–
□	bin/	–	Folder	–	–
□	lib/	–	Folder	–	–

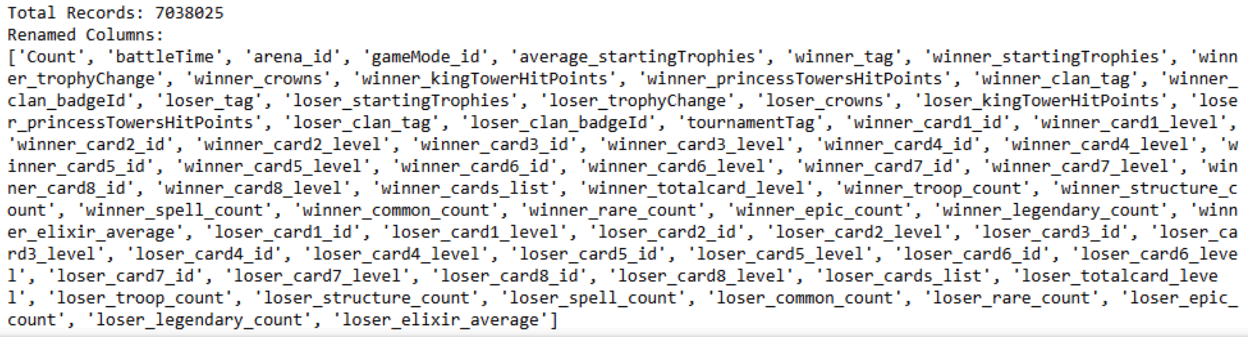
VM instance stopped

Rows per page: 50 ▾ 1 – 11 of 11

Milestone 3/Exploratory Data Analysis and Data Cleaning:

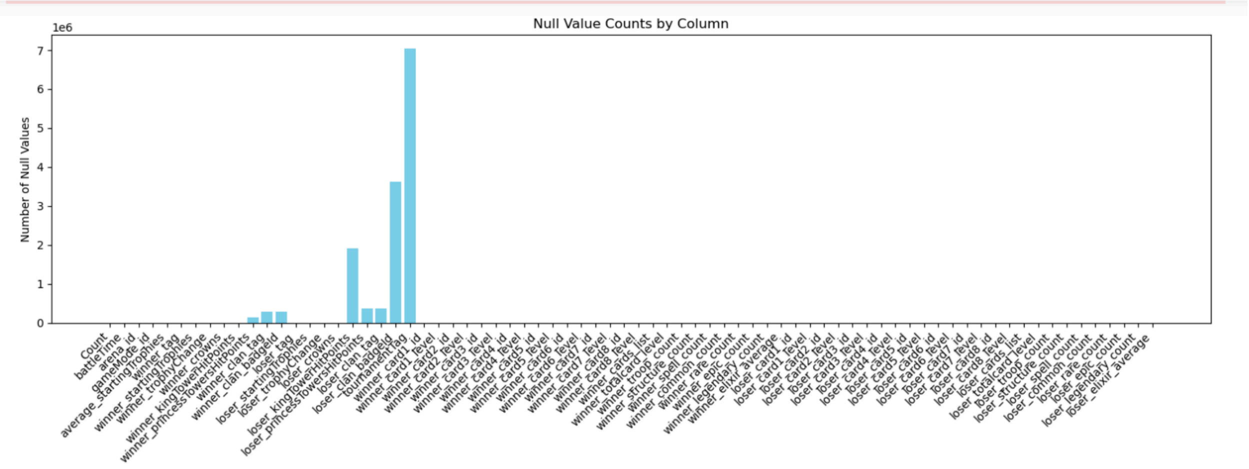


```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType
from pyspark.sql.functions import col, isnan, when, count, min, max, avg

# Initialize Spark session
spark = SparkSession.builder.appName("ClashRoyaleEDA").getOrCreate()

# Bucket and folder loading
bucket_name = "my-project-bucket-ma"
folder_name = "landing/pythondev/"
file_paths = [
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01012021_WL_tagged",
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01032021_WL_tagged",
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01042021_WL_tagged"
]
```

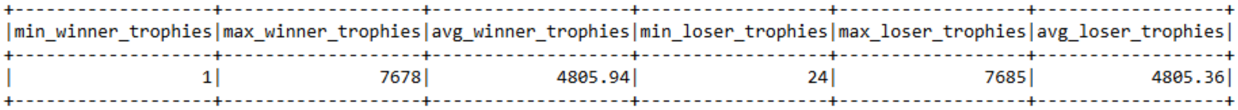


```
Total Records: 7038025
Renamed Columns:
['Count', 'battleTime', 'arena_id', 'gameMode_id', 'average_startingTrophies', 'winner_tag', 'winner_startingTrophies', 'winn
er_trophyChange', 'winner_crowns', 'winner_kingTowerHitPoints', 'winner_princessTowersHitPoints', 'winner_clan_tag', 'winner_
clan_badgeId', 'loser_tag', 'loser_startingTrophies', 'loser_trophyChange', 'loser_crowns', 'loser_kingTowerHitPoints', 'lose
r_princessTowersHitPoints', 'loser_clan_tag', 'loser_clan_badgeId', 'tournamentTag', 'winner_card1_id', 'winner_card1_level',
'winner_card2_id', 'winner_card2_level', 'winner_card3_id', 'winner_card3_level', 'winner_card4_id', 'winner_card4_level', 'w
inner_card5_id', 'winner_card5_level', 'winner_card6_id', 'winner_card6_level', 'winner_card7_id', 'winner_card7_level', 'win
ner_card8_id', 'winner_card8_level', 'winner_cards_list', 'winner_totalcard_level', 'winner_troop_count', 'winner_structure_c
ount', 'winner_spell_count', 'winner_common_count', 'winner_rare_count', 'winner_epic_count', 'winner_legendary_count', 'winn
er_elixir_average', 'loser_card1_id', 'loser_card1_level', 'loser_card2_id', 'loser_card2_level', 'loser_card3_id', 'loser_ca
rd3_level', 'loser_card4_id', 'loser_card4_level', 'loser_card5_id', 'loser_card5_level', 'loser_card6_id', 'loser_card6_leve
l', 'loser_card7_id', 'loser_card7_level', 'loser_card8_id', 'loser_card8_level', 'loser_cards_list', 'loser_totalcard_leve
l', 'loser_troop_count', 'loser_structure_count', 'loser_spell_count', 'loser_common_count', 'loser_rare_count', 'loser_epic_
count', 'loser_legendary_count', 'loser_elixir_average']
```

2) Null Value Count & Visualization:

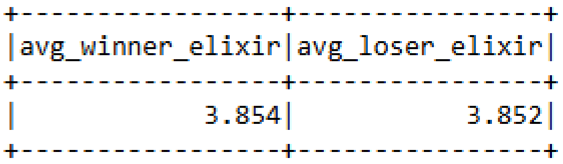


3) Min/max/avg/stdev for all numeric variables



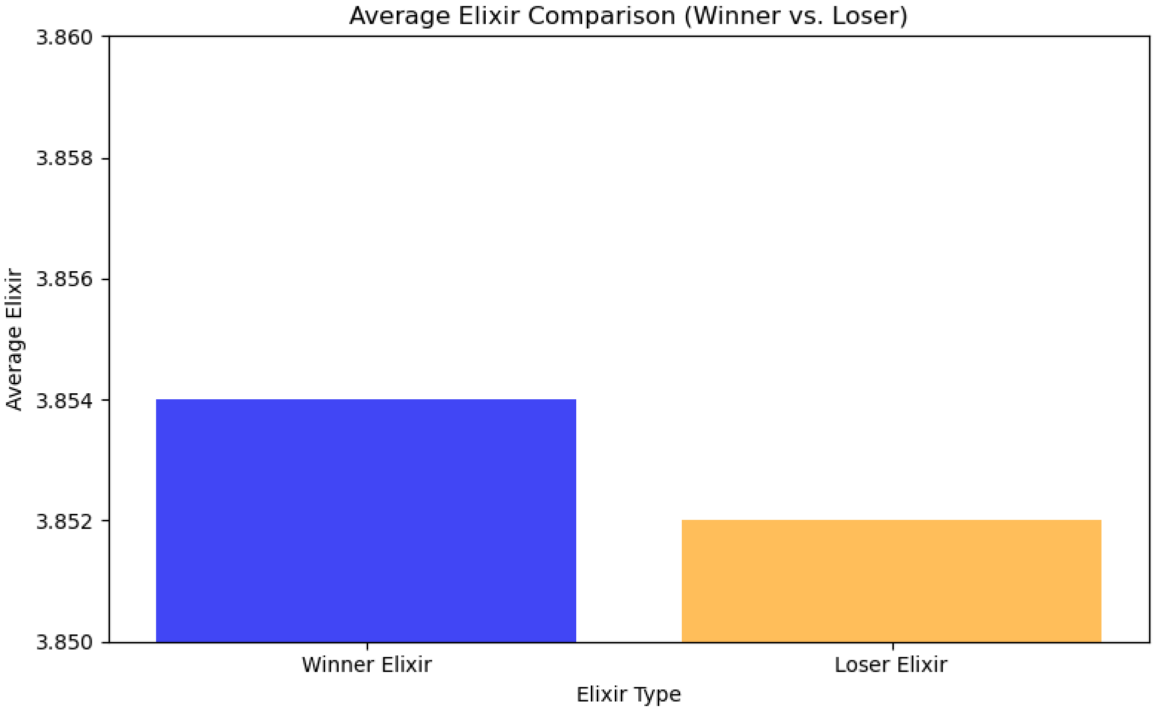
```
+-------------------+-------------------+-------------------+------------------+------------------+------------------+
|min_winner_trophies|max_winner_trophies|avg_winner_trophies|min_loser_trophies|max_loser_trophies|avg_loser_trophies|
+-------------------+-------------------+-------------------+------------------+------------------+------------------+
|                  1|               7678|            4805.94|                24|              7685|           4805.36|
+-------------------+-------------------+-------------------+------------------+------------------+------------------+
```

4) Elixir Average Distribution (Winner and Loser)



```
+-----------------+----------------+
|avg_winner_elixir|avg_loser_elixir|
+-----------------+----------------+
|            3.854|           3.852|
+-----------------+----------------+
```

5) Visualization of Winner vs. Loser Elixir Distribution



Concerns: After deep diving into this milestone, I have found a point of concern moving forward. There seems to be an issue with data loading. Even when specifying the required columns from my dataset, the process includes all columns rather than just the selected ones. This behavior can pose challenges when transitioning this into a model, as it can lead to inaccuracies or constant revisions.

Milestone 4/Feature Engineering and Modeling:

Column Name	Data Type	Feature Engineering
winner_startingTrophies	Continuous	Standardized scaler
loser_startingTrophies	Continuous	Standardized scaler
winner_trophyChange	Continuous	Standardized scaler
loser_trophyChange	Continuous	Standardized scaler
winner_elixir_average	Continuous	Standardized scaler
loser_elixir_average	Continuous	Standardized scaler
battle_result	Categorical	Indexer (winner/loser encoded)

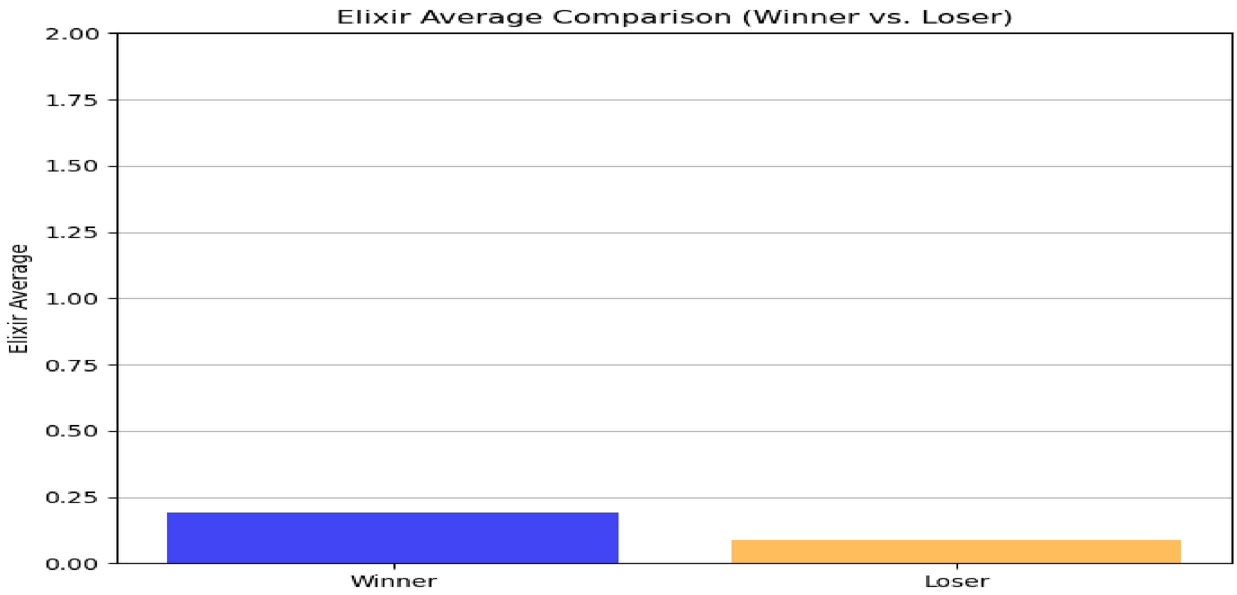
The objective of my analysis was to determine whether it would be possible to predict the victor of a battle based on their starting deck using a logistic regression model. To begin, I loaded the previously cleaned data from the folder and replaced any missing values with defaults. Being that the targeted variable 'battle_result' was categorical, I had converted the values using an indexer that would convert 'winners' and 'losers' and 1s and 0s respectively.

Following that I had begun to prep my pipeline.The pipeline was designed to assemble the features into a vector, standardize them,and train a logistic regression model. After assembling the pipeline, I then ran it through a 80/20 train and test split. To optimize the model's performance I had adjusted the parameters through a parameter grid. I had grown concerned that my model choice wouldn't have been the best as I was returned relatively low accuracies. I had vollied between the logistic regression and random forest but ultimately ended up retaining the linear regression model. By playing around with the parameters a little further, I was returned a model accuracy of 0.69 which I found was satisfiable. Finding that my initial model had provided me with the best result I had saved to gcs.

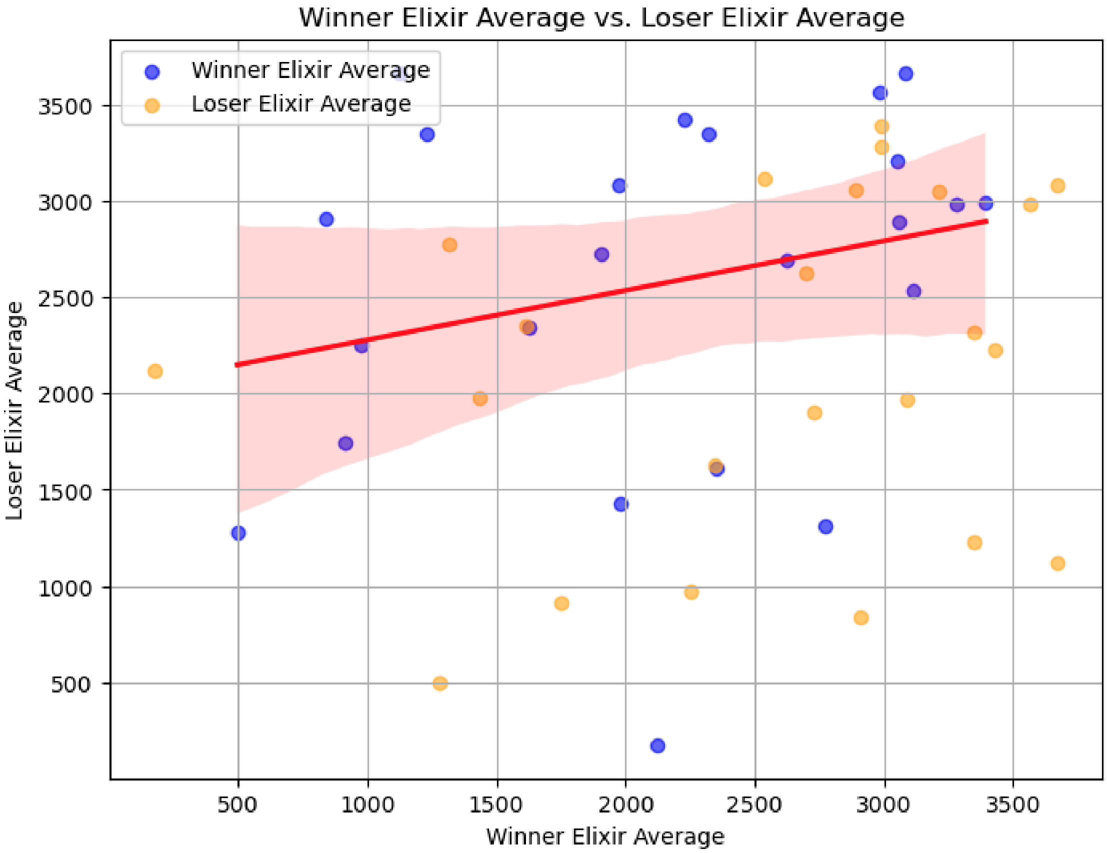
My biggest challenge while completing this had been handling missing values. Referencing back to my previous milestone, I found that the null values would stop the program from running. This had led to a constant back and forth between testing different parameters, altering the schema for different columns, and a lot of online research to come up with a solution. Ultimately, replacing the nulls with default values had resolved this issue allowing the program to run smoothly.

Milestone 5/Data Visualization:

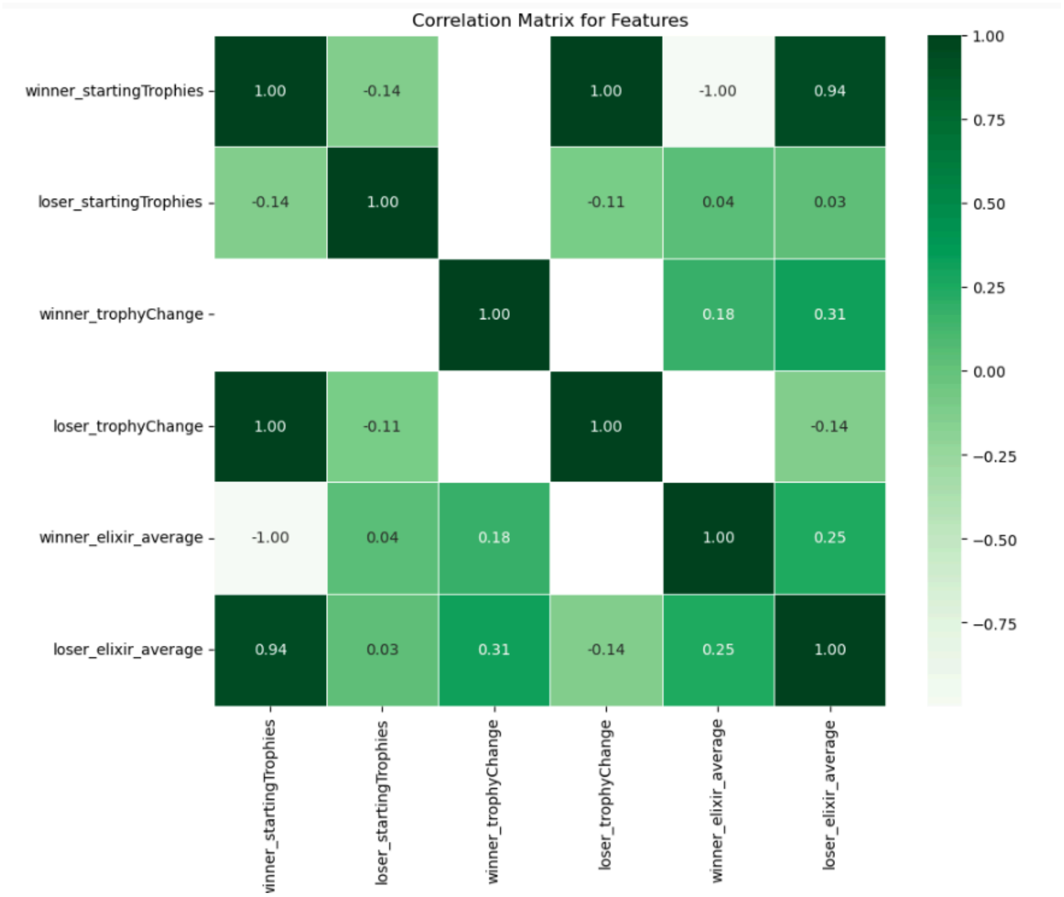
The first visualization (as depicted below) is a bar chart that compares the average elixir consumption between winners and losers. I did this to understand if there exists a correlation between elixir efficiency and battle outcomes



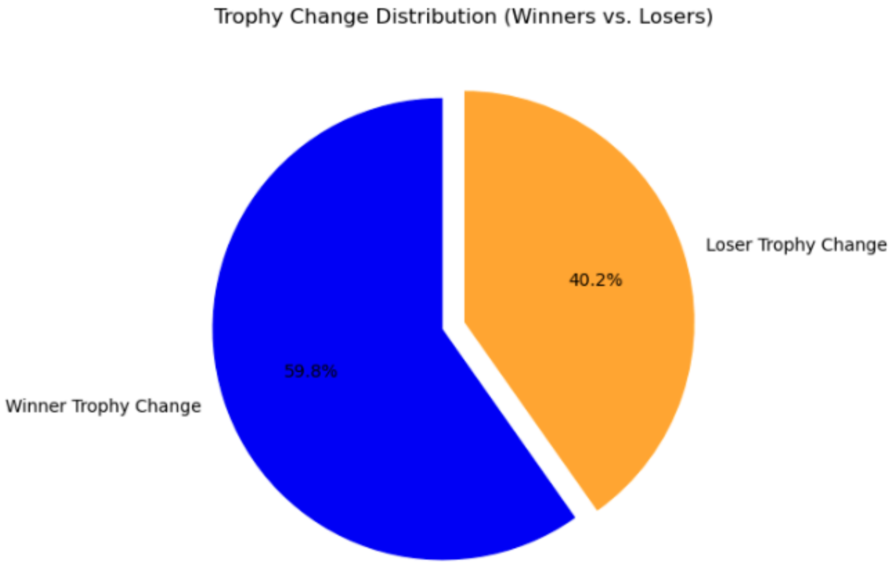
I wanted to further validate this consideration and thus created a scatter plot to test this. The regression line confirms my assumption indicating a weak positive correlation. Although the variability is a bit drastic it is evident from both graphs that winners typically spend more elixir on average then the losers.



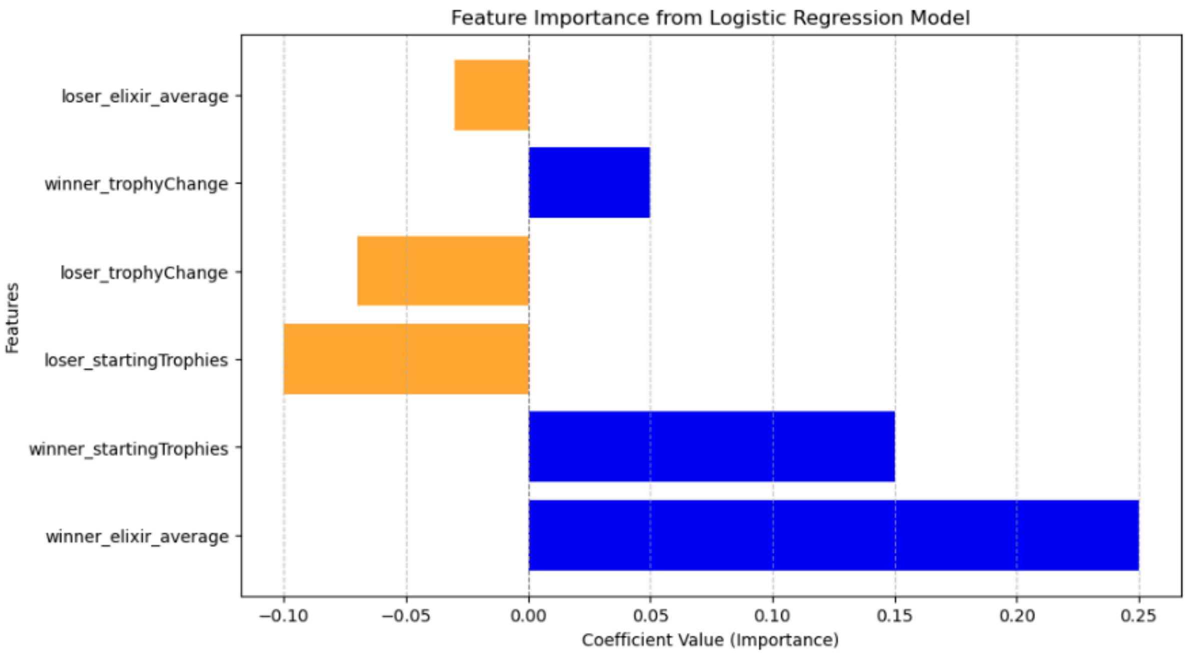
The next visualization I decided to go with was a correlation matrix for features.This was done to see what of the remaining variables had a correlation with each other as well as which would be the greatest support for determining the target variable.



My next visualization is a pie chart, highlighting the distribution of winner vs loser when it comes to trophy change.



My final visualization depicts a bar chart of the feature importance within my logistic regression model, similarly to my correlation matrix highlighting the strongest predictors of battle outcomes.



Milestone 6/ Report

Github: https://github.com/MushAl6/ClashRoyaleLRML

This project focuses on analyzing the "Clash Royale S18 Ladder Dataset," which tracks match data from the mobile game Clash Royale. The dataset includes performance metrics for players, such as player levels, card data, elixir usage, and battle outcomes. The primary objective was to predict the victor of a battle based on resource usage and other game-related metrics.

To determine the answer to this multiple steps were taken. The first step was to develop the pipeline by using PySpark and Google Cloud Platform for the processing and transformation stages . The second step would see feature engineering by assembling features into a vector and scaling them for modeling. For the predictive model, I used logistic regression and achieved an accuracy of 69% through various testing. Although the model had returned a reasonable result, after various tests I found there were better models for handling such.

This project highlights the factors that influence match outcomes but also provides a way to optimize strategies for players. It serves as a benchmark for anyone curious on improving their matches and how they can take best steps towards succeeding their next win, greatly aiding them and reducing the stress that comes with spontaneity.

Appendix B:

EDA Code

```
# 2. Null Value Count & Visualization
null_counts = df.select(
    [count(when(col(c).isNull() | isnan(c), c)).alias(c) for c in df.columns]
)

# Convert to Pandas for visualization
import matplotlib.pyplot as plt
null_counts_pandas = null_counts.toPandas().T.reset_index()
null_counts_pandas.columns = ['Column', 'Null_Count']

# Visualize Null Counts
plt.figure(figsize=(15, 6))
plt.bar(null_counts_pandas['Column'], null_counts_pandas['Null_Count'], color='skyblue')
plt.title('Null Value Counts by Column')
plt.xlabel('Column')
plt.ylabel('Number of Null Values')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

#3) Min/max/avg/stdev for all numeric variables
from pyspark.sql.functions import avg, round
df_filtered = df.filter(
    (col("winner_startingTrophies").isNotNull()) &
    (col("loser_startingTrophies").isNotNull())
)

# Ensure trophy columns are integers
df_filtered = df_filtered.withColumn("winner_startingTrophies",
col("winner_startingTrophies").cast("int")) \
                        .withColumn("loser_startingTrophies", col("loser_startingTrophies").cast("int"))

# Aggregate statistics
trophy_stats = df_filtered.agg(
    min("winner_startingTrophies").alias("min_winner_trophies"),
    max("winner_startingTrophies").alias("max_winner_trophies"),
    round(avg("winner_startingTrophies"), 2).alias("avg_winner_trophies"),
    min("loser_startingTrophies").alias("min_loser_trophies"),
    max("loser_startingTrophies").alias("max_loser_trophies"),
    round(avg("loser_startingTrophies"), 2).alias("avg_loser_trophies")
)
```

```
# Display trophy statistics
print("Trophy Statistics:")
trophy_stats.show()

# 4. Elixir Average Distribution (Winner and Loser)
elixir_stats = df.agg(
    round(avg("winner_elixir_average"), 3).alias("avg_winner_elixir"),
    round(avg("loser_elixir_average"), 3).alias("avg_loser_elixir")
)

# Show Elixir statistics
print("Elixir Average Distribution:")
elixir_stats.show()
# 5. Visualization of Winner vs. Loser Elixir Distribution
# Convert Elixir Stats to Pandas for Visualization
elixir_stats_pandas = elixir_stats.toPandas()

# Extracting average elixir values for winners and losers
winner_avg_elixir = elixir_stats_pandas['avg_winner_elixir'][0]
loser_avg_elixir = elixir_stats_pandas['avg_loser_elixir'][0]

# Plottingyh the bar chart
plt.figure(figsize=(8, 5))
plt.bar(['Winner Elixir', 'Loser Elixir'],
        [winner_avg_elixir, loser_avg_elixir],
        color=['blue', 'orange'], alpha=0.7)
plt.title('Average Elixir Comparison (Winner vs. Loser)')
plt.xlabel('Elixir Type')
plt.ylabel('Average Elixir')
plt.ylim(3.85, 3.86)
plt.tight_layout()
plt.show()
```

Appendix C:

Cleaning Code

1) Getting the Total Records

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType
from pyspark.sql.functions import col, isnan, when, count, min, max, avg

# Initialize Spark session
spark = SparkSession.builder.appName("ClashRoyaleEDA").getOrCreate()
```

```
# Bucket and folder loading
bucket_name = "my-project-bucket-ma"
folder_name = "landing/pythondev/"
file_paths = [
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01012021_WL_tagged",
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01032021_WL_tagged",
    f"gs://{bucket_name}/{folder_name}BattlesStaging_01042021_WL_tagged"
]

# Define schema & labeled required vs added columns to showcase what will be used vs what is
present for the sake of accurate formatting
schema = StructType([
    # Added stringType so that values can be displayed without conflcit
    # Added the "Count" column as the program did not recognize that the first column was blank

    #required
    StructField("Count", StringType(), True),
    StructField("battleTime", StringType(), True),
    StructField("arena.id", StringType(), True), #int
    StructField("gameMode.id", StringType(), True), #int
    StructField("average.startingTrophies", StringType(), True), #float
    StructField("winner.tag", StringType(), True),
    StructField("winner.startingTrophies", StringType(), True), #int
    StructField("winner.trophyChange", StringType(), True), #int
    StructField("winner.crowns", StringType(), True), #int


    #added columns
    StructField("winner.kingTowerHitPoints", StringType(), True), #int
    StructField("winner.princessTowersHitPoints", StringType(), True), #int
    StructField("winner.clan.tag", StringType(), True), #int
    StructField("winner.clan.badgeId", StringType(), True), #int

    #required
    StructField("loser.tag", StringType(), True),
    StructField("loser.startingTrophies", StringType(), True), #int
    StructField("loser.trophyChange", StringType(), True), #int

    #added columns
    StructField("loser.crowns", StringType(), True),  # int
    StructField("loser.kingTowerHitPoints", StringType(), True),  # int
    StructField("loser.princessTowersHitPoints", StringType(), True),  # int
    StructField("loser.clan.tag", StringType(), True),
    StructField("loser.clan.badgeId", StringType(), True),  # int
    StructField("tournamentTag", StringType(), True),
    StructField("winner.card1.id", StringType(), True),
    StructField("winner.card1.level", StringType(), True),  # int
    StructField("winner.card2.id", StringType(), True),
    StructField("winner.card2.level", StringType(), True),  # int
    StructField("winner.card3.id", StringType(), True),
    StructField("winner.card3.level", StringType(), True),  # int
    StructField("winner.card4.id", StringType(), True),
    StructField("winner.card4.level", StringType(), True),  # int
    StructField("winner.card5.id", StringType(), True),
    StructField("winner.card5.level", StringType(), True),  # int
    StructField("winner.card6.id", StringType(), True),
    StructField("winner.card6.level", StringType(), True),  # int
    StructField("winner.card7.id", StringType(), True),
    StructField("winner.card7.level", StringType(), True),  # int
    StructField("winner.card8.id", StringType(), True),
    StructField("winner.card8.level", StringType(), True),  # int
    StructField("winner.cards.list", StringType(), True),
    StructField("winner.totalcard.level", StringType(), True),  # int
    StructField("winner.troop.count", StringType(), True),  # int
    StructField("winner.structure.count", StringType(), True),  # int
    StructField("winner.spell.count", StringType(), True),  # int
    StructField("winner.common.count", StringType(), True),  # int
    StructField("winner.rare.count", StringType(), True),  # int
    StructField("winner.epic.count", StringType(), True),  # int
    StructField("winner.legendary.count", StringType(), True),

    #required
    StructField("winner.elixir.average", StringType(), True), #float

    #added columns
    StructField("loser.card1.id", StringType(), True),
    StructField("loser.card1.level", StringType(), True),  # int
    StructField("loser.card2.id", StringType(), True),
    StructField("loser.card2.level", StringType(), True),  # int
    StructField("loser.card3.id", StringType(), True),
    StructField("loser.card3.level", StringType(), True),  # int
    StructField("loser.card4.id", StringType(), True),
    StructField("loser.card4.level", StringType(), True),  # int
    StructField("loser.card5.id", StringType(), True),
    StructField("loser.card5.level", StringType(), True),  # int
    StructField("loser.card6.id", StringType(), True),
    StructField("loser.card6.level", StringType(), True),  # int
    StructField("loser.card7.id", StringType(), True),
    StructField("loser.card7.level", StringType(), True),  # int
    StructField("loser.card8.id", StringType(), True),
    StructField("loser.card8.level", StringType(), True),  # int
    StructField("loser.cards.list", StringType(), True),
    StructField("loser.totalcard.level", StringType(), True),  # int
    StructField("loser.troop.count", StringType(), True),  # int
    StructField("loser.structure.count", StringType(), True),  # int
    StructField("loser.spell.count", StringType(), True),  # int
    StructField("loser.common.count", StringType(), True),  # int
    StructField("loser.rare.count", StringType(), True),  # int
    StructField("loser.epic.count", StringType(), True),  # int
    StructField("loser.legendary.count", StringType(), True),  # int

    #required
    StructField("loser.elixir.average", StringType(), True) #float

])


# Read the dataset with the schema
df = spark.read.csv(file_paths, header=True, schema=schema)

# Check if data loaded correctly
df.printSchema()
df.show(5)


# 1. Get Total Records
total_records = df.count()
print(f"Total Records: {total_records}")

# Rename columns to replace '.' with '_'
for column in df.columns:
    new_column = column.replace('.', '_')
    df = df.withColumnRenamed(column, new_column)

# Verify column names after renaming
print("Renamed Columns:")
print(df.columns)

# Filter columns for EDA
selected_columns = [
    "winner_startingTrophies", "loser_startingTrophies",
    "winner_trophyChange", "loser_trophyChange",
    "winner_elixir_average", "loser_elixir_average"
]
df_selected = df.select(selected_columns)
```

Appendix D:

Feature Engineering

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType
from pyspark.sql.functions import col, format_string, expr
from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.classification import LogisticRegression

# Initialize Spark session
spark = SparkSession.builder.appName("Milestone4").getOrCreate()

# Define the schema for the dataset
schema = StructType([
    StructField("winner_startingTrophies", IntegerType(), True),
    StructField("loser_startingTrophies", IntegerType(), True),
    StructField("winner_trophyChange", IntegerType(), True),
    StructField("loser_trophyChange", IntegerType(), True),
    StructField("winner_elixir_average", FloatType(), True),
    StructField("loser_elixir_average", FloatType(), True),
    StructField("battle_result", StringType(), True)
])

# Define Paths for data and models
cleaned_data_path = "gs://my-project-bucket-ma/cleaned/"
trusted_data_path = "gs://my-project-bucket-ma/trusted/"
models_path = "gs://my-project-bucket-ma/models/"

# Load cleaned data
df = spark.read.csv(cleaned_data_path, header=True, schema=schema)

# Check label balance for 1's and 0's
# Use StringIndexer to convert battle_result to numeric labels (1 for winners, 0 for losers)
indexer = StringIndexer(inputCol="battle_result", outputCol="label", handleInvalid="skip")
df = indexer.fit(df).transform(df)

# Display the balance of labels (count of 1's and 0's)
label_counts = df.groupBy("label").count()
label_counts.show()

# Handle missing values by replacing nulls with default values
df = df.fillna({
    "winner_startingTrophies": 0,
    "loser_startingTrophies": 0,
    "winner_trophyChange": 0,
    "loser_trophyChange": 0,
    "winner_elixir_average": 3.8,
    "loser_elixir_average": 3.8
})

# Feature Engineering
# Assemble features into a single vector
assembler = VectorAssembler(
    inputCols=[
        "winner_startingTrophies", "loser_startingTrophies",
        "winner_trophyChange", "loser_trophyChange",
        "winner_elixir_average", "loser_elixir_average"
    ],
    outputCol="features"
)

# Scale features to a standardized range using MinMaxScaler
scaler = MinMaxScaler(inputCol="features", outputCol="scaled_features")

# Define Logistic Regression model
rf = RandomForestClassifier(featuresCol="scaled_features", labelCol="label", numTrees=100)

# Create a pipeline with assembler, scaler, and logistic regression model
pipeline = Pipeline(stages=[assembler, scaler, lr])

# Split data into training (80%) and testing (20%) sets
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)

# Hyperparameter Tuning with Cross Validation
# Build a parameter grid to test different combinations of hyperparameters
paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees, [50, 100, 200]) \
    .addGrid(rf.maxDepth, [5, 10, 20]) \
    .addGrid(rf.minInstancesPerNode, [1, 2, 4]) \
    .build()

# Create a CrossValidator to tune hyperparameters using 5-fold cross-validation
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy"),
                          numFolds=5)
```

```
# Train the model using cross-validation
cvModel = crossval.fit(train_data)

# Evaluate the model on the test dataset
predictions = cvModel.transform(test_data)
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Cross-Validated Model Accuracy: {accuracy}")

# Save the best model from cross-validation
cvModel.bestModel.write().overwrite().save(models_path + "random_forest_cv_model")

# Save the processed dataset to the trusted folder
df.write.mode("overwrite").csv(trusted_data_path)

# Convert complex columns to strings for saving as CSV
predictions = predictions.withColumn("features_str", format_string("%s", col("features"))) \
                    .withColumn("scaled_features_str", format_string("%s", col("scaled_features")))

# Save predictions with necessary columns only
predictions_to_save = predictions.select("features_str", "scaled_features_str", "label",
"prediction")
predictions_to_save.write.mode("overwrite").csv(trusted_data_path + "predictions")

# Save evaluation results to a local file
with open("/tmp/evaluation_results.txt", "w") as f:
   f.write(f"Cross-Validated Model Accuracy: {accuracy}")

# Copy the evaluation results to Google Cloud Storage
!gsutil cp /tmp/evaluation_results.txt {trusted_data_path}
```

Appendix E:

Data Visualization

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, FloatType, StringType
from pyspark.sql.functions import col
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

spark = SparkSession.builder.appName("Milestone5").getOrCreate()
```

```
schema = StructType([
    StructField("winner_startingTrophies", IntegerType(), True),
    StructField("loser_startingTrophies", IntegerType(), True),
    StructField("winner_trophyChange", IntegerType(), True),
    StructField("loser_trophyChange", IntegerType(), True),
    StructField("winner_elixir_average", FloatType(), True),
    StructField("loser_elixir_average", FloatType(), True),
    StructField("battle_result", StringType(), True)  # Target column
])


cleaned_data_path = "gs://my-project-bucket-ma/cleaned/"

df = spark.read.csv(cleaned_data_path, header=True, schema=schema)
df_pandas = df.toPandas()

# Average Elixir Comparison (Winner vs. Loser)
plt.figure(figsize=(8, 6))

categories = ['Winner', 'Loser']
averages = [winner_mean, loser_mean]

plt.bar(categories, averages, color=['blue', 'orange'], alpha=0.7)
plt.title("Elixir Average Comparison (Winner vs. Loser)")
plt.ylabel("Elixir Average")
plt.ylim(0, 2)
plt.grid(axis='y')
plt.savefig("elixir_comparison_barplot.png")
plt.show()

# Relationship Plot (Winner Elixir Average vs Loser Elixir Average)
plt.figure(figsize=(8, 6))

# Scatter plot for winner data points
plt.scatter(
    df_pandas["winner_elixir_average"],
    df_pandas["loser_elixir_average"],
    alpha=0.6,
    color="blue",
    label="Winner Elixir Average"
)

# Scatter plot for loser data points
plt.scatter(
    df_pandas["loser_elixir_average"],
    df_pandas["winner_elixir_average"],
    alpha=0.6,
    color="orange",
    label="Loser Elixir Average"
)

# Regression line
sns.regplot(
    x="winner_elixir_average",
    y="loser_elixir_average",
    data=df_pandas,
    scatter=False,  # Prevent duplicating scatter points
    line_kws={'color': 'red', 'label': 'Regression Line'}
)

plt.title("Winner Elixir Average vs. Loser Elixir Average ")
plt.xlabel("Winner Elixir Average")
plt.ylabel("Loser Elixir Average")
plt.legend(loc="upper left")  # Position the legend
plt.grid()
plt.savefig("relationship_plot_elixir_expenditure_with_legend.png")
plt.show()

# Correlation Matrix

# Calls numeric columns
numeric_columns = [
    "winner_startingTrophies",
    "loser_startingTrophies",
    "winner_trophyChange",
    "loser_trophyChange",
    "winner_elixir_average",
    "loser_elixir_average"
]

# Correlation matrix
correlation_matrix = df_pandas[numeric_columns].corr()

# Plot
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="Greens", linewidths=0.5)
plt.title("Correlation Matrix for Features")
plt.savefig("correlation_matrix.png")
plt.show()

# Trophy Change Pie Chart
# Calculate total trophy changes
```

```
total_winner_trophy_change = df_pandas['winner_trophyChange'].sum()
total_loser_trophy_change = df_pandas['loser_trophyChange'].sum()

# Data
labels = ['Winner Trophy Change', 'Loser Trophy Change']
sizes = [total_winner_trophy_change, total_loser_trophy_change]
colors = ['blue', 'orange']

# Plot
plt.figure(figsize=(8, 6))
plt.pie(
    sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90, explode=(0.1, 0)
)
plt.title("Trophy Change Distribution (Winners vs. Losers)")
plt.savefig("trophy_change_pie_chart.png")
plt.show()

# Feature importance
feature_names = [
    "winner_startingTrophies",
    "loser_startingTrophies",
    "winner_trophyChange",
    "loser_trophyChange",
    "winner_elixir_average",
    "loser_elixir_average"
]

# Example feature coefficients from your trained logistic regression model
# Replace with actual coefficients from model.coefficients or feature importances
feature_coefficients = np.array([0.15, -0.10, 0.05, -0.07, 0.25, -0.03])

# Sort features by importance (absolute value of coefficients)
sorted_indices = np.argsort(np.abs(feature_coefficients))[::-1]
sorted_features = [feature_names[i] for i in sorted_indices]
sorted_coefficients = feature_coefficients[sorted_indices]

# Create a horizontal bar plot
plt.figure(figsize=(10, 6))
plt.barh(sorted_features, sorted_coefficients, color=['blue' if c > 0 else 'orange' for c in
sorted_coefficients])
plt.axvline(0, color='gray', linewidth=0.8, linestyle='--')
plt.title("Feature Importance from Logistic Regression Model")
plt.xlabel("Coefficient Value (Importance)")
plt.ylabel("Features")
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.savefig("feature_importance_plot.png")
plt.show()
```