Kevin Guan

12/12/24

CIS 4130

Big Data Technologies

Milestone 1

For milestone 1, we had to research and find a data set larger than 10 GB. Trying to find a dataset I like that I would use for my Big Data Project. The dataset I've chosen and looked into is called Clash Royale Battles. The link for this is from Kaggle: https://www.kaggle.com/datasets/s1m0n38/clash-royale-games/data. This dataset covers from September 2022 to December 2023. The dataset structure covers each season's total games they've played for that month and how many days they were in the season, but we're looking more towards the players' data with what type of cards they're using and trophies. I've chosen to do this dataset because this is a game I sometimes play, and I think it would be interesting to do for my semester project. The data set attributes I will be using in the dataset are the trophies, crowns, and the list of 8 cards each of the players is using. Through these dataset attributes, I will be predicting who will win the game based on the starting setup they have, which we'll see in their starting setup by the eight cards and trophies they have. The results will be shown by the crown's column, showing who the winner is and who has the most crowns left. From using the data attributes of trophies and the eight cards they hold, we'll be predicting who would win against one another through using classification.

Milestone 2

For milestone 2, we had to collect and download the data into a bucket on Google Cloud Storage. Make a bucket called my-bigdata-project-KG, which will have 5 folders, landing, cleaned, trusted, code, and models. In order to do this, I had first to download the Kaggle Data Sets using the Linux Command Line. Then, move the kaggle.json file to the Kaggle directory and secure the file. I would also add software packages and set up a Python development environment, and once I have that in place, I would download the Datasets from Kaggle and unzip the archive files to get the individual files. Once everything is downloaded, I will create my buckets, permit it, and copy all the files into the landing folder.

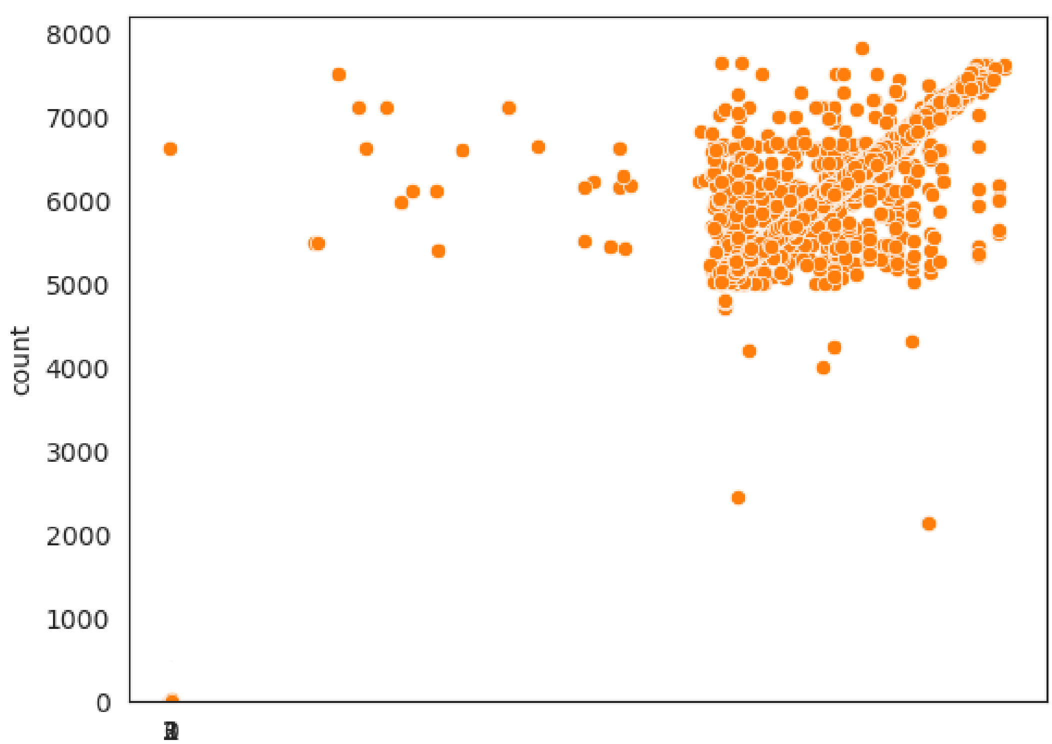
Milestone 3

For milestone 3, I wrote a Python code that loaded the GCS data set and produced descriptive statistics about my data. I used a computing engine to produce these statistics.

One highlight that I noticed in Appendix B is that my Clash Royale units are shown through ID instead of their full name, as they would normally be called, like “Baby Dragon.” When I also used the code “df.info,” it showed me the column names and data type, and something I noticed from it was that none of my columns were complete words and mostly were IDs that were integers. I’ve concluded that loading the dataset to see the descriptive statistics in the data showed me what is inside my files and what I’ll need to focus on for my next milestone.

I also made a histogram plot, but I am unsure if it is correct, as I don't know if this is the best graph to pick for my model.

From what I can see from the data, I can see the two-player ID, total trophies, the timestamp, the 8 card IDs the players use for their deck, and the number of crowns left for each side that will tell you who won. In my cleaning, I've cleaned up the null values and missing values to ensure everything flows well and to see everything clearly. It is just hard to see my columns as they're not words and only ID, which is a hassle to notice what column is what. A challenge I see in feature engineering is probably my columns, as they're all ID and not words, which may be hard for me to notice what I should put in as I don't know what some of these columns are. Finding the winner may also be challenging, as you need to see the leftover crowns on each side to see who the winners are.



Milestone 4

In milestone 4, I wrote PySpark code that would read and process my data using a Dataproc cluster. In it, I would do the feature engineering and modeling for milestone #4. To do my feature engineering code, I would get the libraries that I need, load the data, take a small data sample that I would be using to test, create a target variable (which is one if player one wins, 0 if player two wins), define the columns that I need for my feature engineering, input my feature engineering pipeline, then fit the pipeline and transform the data, once I have transformed the data I would print it to see the results and write those results into my trusted files. For my modeling code, I would first get all my libraries, load my trusted file's data, split the data into training and testing sets, define the random forest classifier, create a parameter grid for hyperparameter tuning, create an evaluator and a cross-validator, once I have created these two I would train the model using Cross Validator and make it make predictions on the test data and evaluate the model using Area Under ROC. Using this information, I would then manually calculate Precision, Recall, F1, and Accuracy using predictions DataFrame, print the metrics results, and save them into my model file for my bucket.

The program I've made is used to predict the winner of the Clash Royale match using the player card information, crowns, and trophies. I'm predicting it by having the cards indexed, encoded, and assembled into feature vectors, then scaling the features into a random forest classifier to test and train the data to predict. After the scaling, it would go through cross-validation and evaluate the AUC, which is the Area under the ROC Curve. Finally, my data is stored in trusted, and my model is stored in my model file. One challenge I encountered for my feature engineering and modeling the data was how I couldn't use my own bucket, as I had to use a different bucket for mine because my cleaned file was corrupted. Another challenge was trying to find the best way to test my data, as I soon found out that the classifier was the best one, and I was having trouble knowing what to put for my index, encode, and vectors.

Feature Engineering:

```
root
 |-- datetime: string (nullable = true)
 |-- gamemode: long (nullable = true)
 |-- player1_tag: string (nullable = true)
 |-- player1_trophies: integer (nullable = true)
 |-- player1_crowns: integer (nullable = true)
 |-- player1_card1: long (nullable = true)
 |-- player1_card2: long (nullable = true)
 |-- player1_card3: long (nullable = true)
 |-- player1_card4: long (nullable = true)
 |-- player1_card5: long (nullable = true)
 |-- player1_card6: long (nullable = true)
 |-- player1_card7: long (nullable = true)
 |-- player1_card8: long (nullable = true)
 |-- player2_tag: string (nullable = true)
 |-- player2_trophies: integer (nullable = true)
 |-- player2_crowns: integer (nullable = true)
 |-- player2_card1: long (nullable = true)
 |-- player2_card2: long (nullable = true)
 |-- player2_card3: long (nullable = true)
 |-- player2_card4: long (nullable = true)
 |-- player2_card5: long (nullable = true)
 |-- player2_card6: long (nullable = true)
 |-- player2_card7: long (nullable = true)
 |-- player2_card8: long (nullable = true)
 |-- gametime: timestamp_ntz (nullable = true)
```

```
+-------------+-------------+-----+--------------------------------------------------------------------------------------------
--------------------------------------------------+
|player1_card1|player1_card2|label|features
|
+-------------+-------------+-----+--------------------------------------------------------------------------------------------
--------------------------------------------------+
|26000001     |26000011     |0.0  |(1760,[7,110,236,346,527,557,661,770,889,990,1105,1216,1392,1435,1571,1679],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])  |
|26000014     |26000018     |1.0  |(1760,[19,113,220,334,470,555,661,771,883,996,1116,1226,1345,1501,1541,1650],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]) |
|26000047     |26000051     |0.0  |(1760,[15,181,244,357,480,554,661,771,882,1004,1111,1227,1346,1466,1561,1654],[1.0,1.0,1.
0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
|26000010     |26000011     |1.0  |(1760,[1,110,256,356,444,566,670,773,891,1029,1135,1280,1401,1432,1545,1650],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]) |
|26000006     |26000018     |1.0  |(1760,[6,113,237,361,446,550,667,773,892,991,1104,1246,1320,1437,1545,1656],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])  |
|26000010     |26000014     |0.0  |(1760,[1,112,220,330,452,551,660,770,881,990,1102,1221,1329,1434,1542,1653],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])  |
```

Modeling Code Results:

Area Under ROC (AUC) on test data = 0.5561

Precision: 0.5951

Recall: 0.0738

F1 Score: 0.1313

Accuracy: 0.5223


```
Area Under ROC (AUC) on test data = 0.5561

Precision: 0.5951
Recall: 0.0738
F1 Score: 0.1313
Accuracy: 0.5223
```


player1_card1	player1_card2	label	prediction	probability
26000004	26000011	0.0	0.0	[0.5235160866076481,0.4764839133923518]
26000023	26000069	0.0	0.0	[0.5041793090124443,0.49582069098755566]
26000011	26000018	1.0	0.0	[0.5230938890600617,0.4769061109399383]
26000010	26000014	0.0	1.0	[0.49190962939351507,0.508090370606485]
26000012	26000024	1.0	0.0	[0.5040158976752062,0.49598410232479384]
26000007	26000012	1.0	0.0	[0.5160107510778386,0.48398924892216144]
26000043	26000063	1.0	0.0	[0.5033014949744126,0.49669850502558743]
26000000	26000006	0.0	1.0	[0.49190962939351507,0.508090370606485]
26000012	26000063	1.0	0.0	[0.5093821462284868,0.49061785377151323]
26000009	26000010	0.0	0.0	[0.5174858677117989,0.48251413228820106]
26000010	26000013	0.0	0.0	[0.5063480814713438,0.493651918528656２]
26000010	26000024	0.0	0.0	[0.518315595633066１,0.481684404366933９]
26000020	26000041	1.0	1.0	[0.4894523145096742,0.5105476854903258]
26000004	26000006	1.0	0.0	[0.5043099175262322,0.4956900824737677]
26000010	26000030	1.0	0.0	[0.5093821462284868,0.49061785377151323]
26000000	26000010	0.0	0.0	[0.5197824976031415,0.48021750239685845]
26000010	26000020	0.0	0.0	[0.5112819929388291,0.48871800706117086]
26000010	26000030	0.0	0.0	[0.5179017865730888,0.4820982134269111]
26000048	26000063	1.0	0.0	[0.5112934697341683,0.48870653026583166]
26000002	26000011	1.0	0.0	[0.5073754285840227,0.49262457141597726]

only showing top 20 rows

Milestone 5

From my visualizations, it has a flat-out line for most of my visualizations. The visualizations that I have chosen were the Life Curve, precision-recall curve, Confusion Matrix, and ROC.

For my Lift Curve visualization, my line goes from 2.0 to 1.0, meaning that the model is highly effective at the beginning as it can identify positive cases. However, as recall increases, the model precision decreases, and when it decreases to 1.0, it shows the predictive power going down to where your model is performing better than random guessing.

For my precision-recall curve visualization, it was a straight line at 1.00, meaning the model was performing well and concisely throughout the whole recall level. This shows that the model is good with its precision, as the model is able to classify all positive instances with no false positives and false negatives.

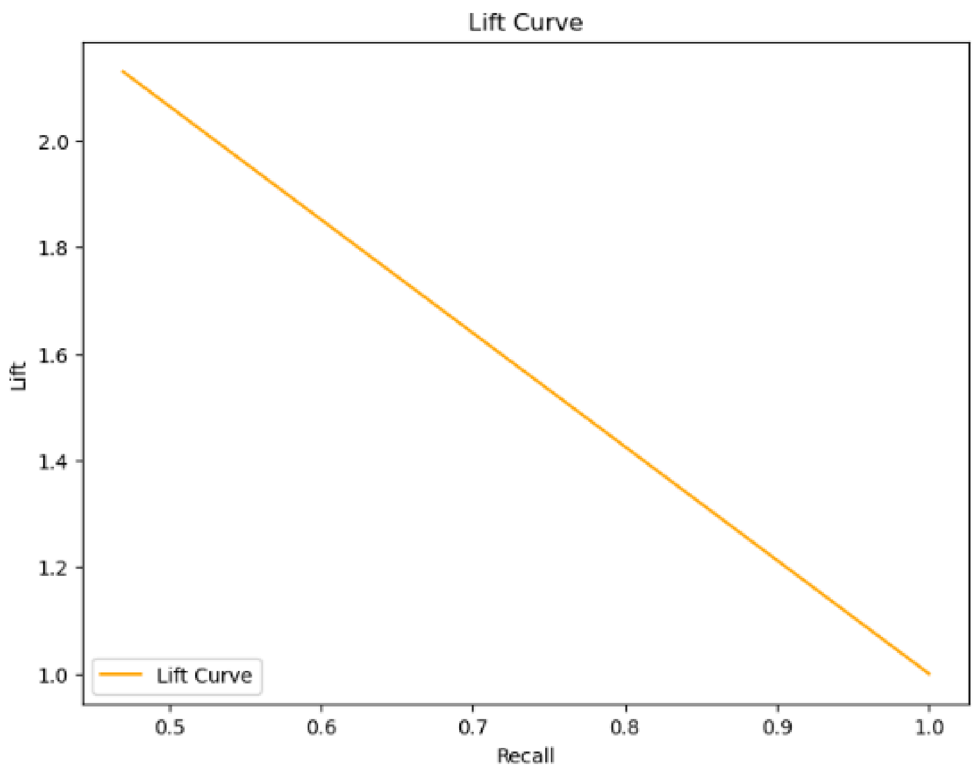
For my Confusion Matrix visualization, most of my data in the visualizations are on the negative side, as in "positive-negative" and "negative-negative." I think it is mostly on the negative side because there is always one true winner at the end, as the number of crowns they have left shows who is the winner. There are a few on the positive-positive and positive-negative, but it doesn't reach the numbers on the negative side.

My ROC visualization shows an upward, curvy ROC curve that shows a Higher True Positive Rate than a Lower False Positive Rate. It shows my model correctly identifies my positive cases to ensure that it doesn't give me a false positive. My model also shows that it has a strong classification performance, as it is close to 1.0.

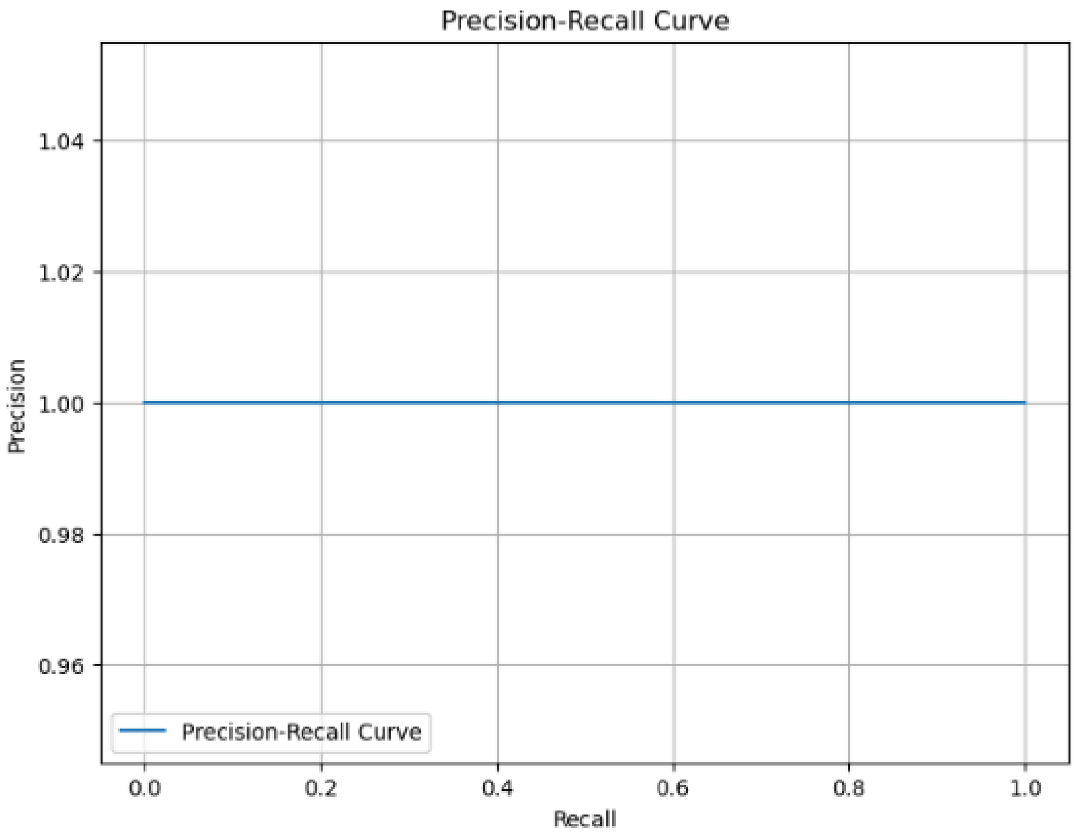
The most important features in our model are seeing their trophies and the player crown that is left over to see who will win.

Milestone 5 Visualizations

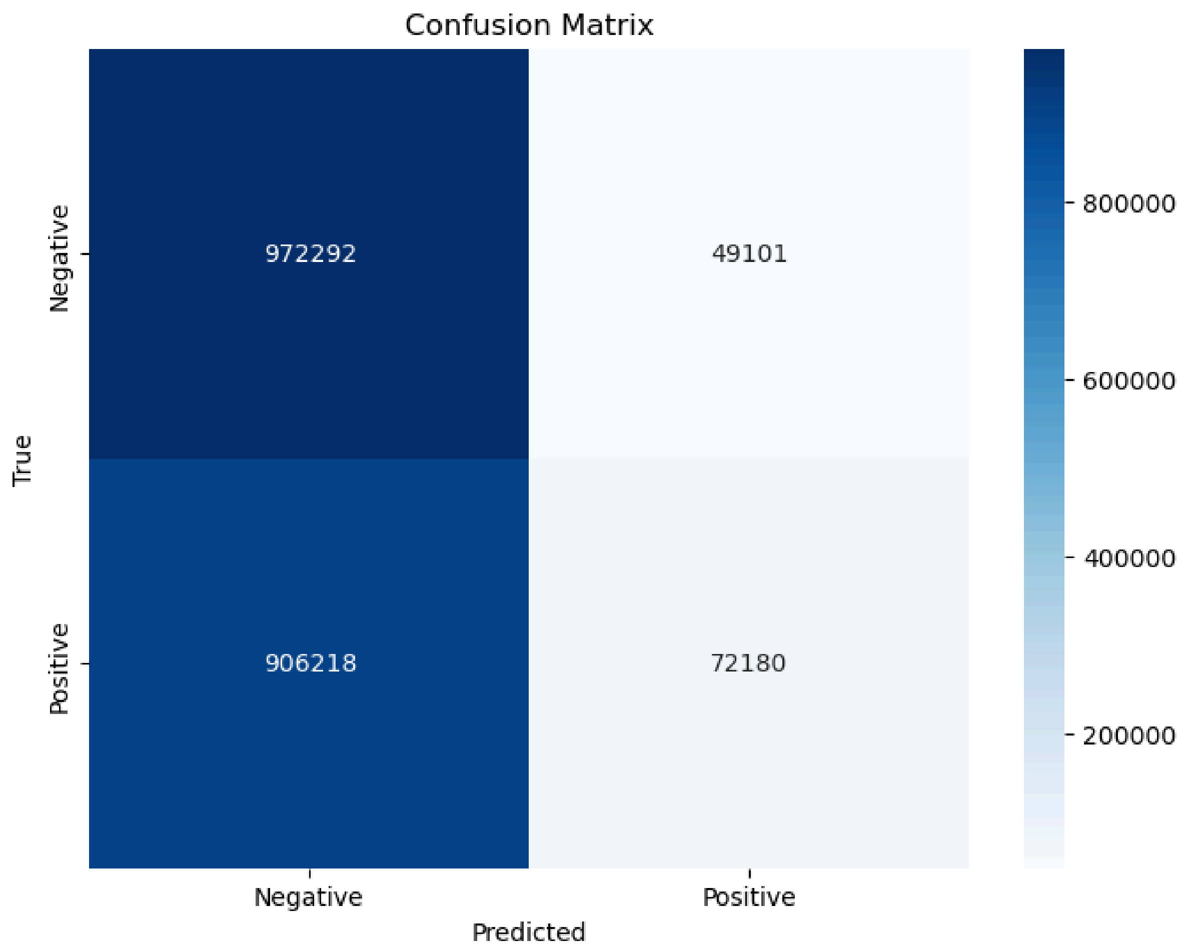
Life Curve



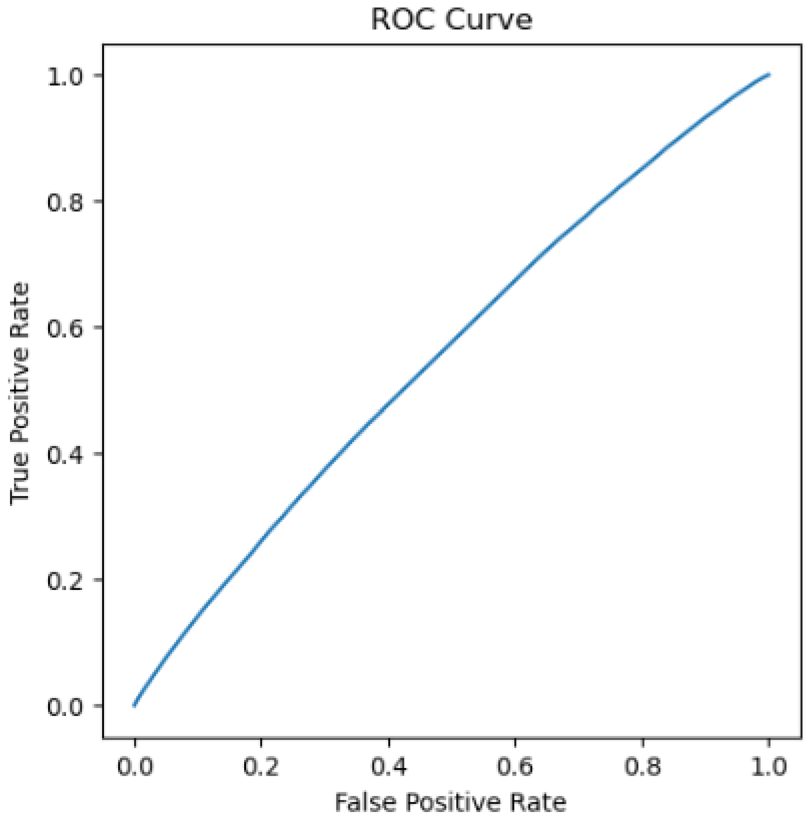
Precision-Recall Curve



Confusion Matrix



ROC Curve



Milestone 6

Overall, I built a complete machine learning pipeline in this semester-long project that incorporated big data technologies using a cloud infrastructure. I've dealt with struggles and successes in making my machine learning pipeline, as I had to go through a proposal, data acquisition, exploratory data analysis and data cleaning, feature engineering and modeling, and data visualization. Going through this whole process has been amazing, and a new way to incorporate coding to help get information from tons of data that is over 10GB. Even though my prediction was pretty straightforward, which was finding out if player one or player two would win based on the starting setup they have from the eight cards and the trophies they have. I enjoy every moment of learning something new that I could use for my daily life if I ever try to look up or want to predict a certain thing from a large dataset. To conclude, even though my predictions from the data processing pipeline weren't that solid and perfect, they still did a good job predicting which player 1 or 2 would win. But, it was still able to extract all the information they had gotten and give a result on who would win.

Appendix A

Downloading Kaggle Data Sets using the Linux Command Line:

- mkdir .kaggle
- Uploading my kaggle.json file
- Ls -la

Move the kaggle.json file to the .kaggle directory using the command:

- mv kaggle.json .kaggle/

Securing the file:

- chmod 600 .kaggle/kaggle.json

Software packages and set up a Python development environment:

- sudo apt -y install zip
- sudo apt -y install python3-pip python3.11-venv
- python3 -m venv pythondev
- cd pythondev
- source bin/activate
- pip3 install kaggle
- kaggle datasets list

Downloading Datasets from Kaggle

- kaggle datasets download -d s1m0n38/clash-royale-games

Unzipping the Archive file to get the individual data files

- unzip clash-royale-games.zip

Creating a bucket

- gcloud storage buckets create gs://my-bigdata-project-kg --project=kevin2003 --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-accessz

Granting permission

- gcloud auth login

Once I created the bucket, I copied files from the local file system into the new bucket:

- gsutil cp -r * gs://my-bigdata-project-kg/landing/

Appendix B

Milestone 3 Part # 1 Coding (Appendix B):

```
# Importing Libraries
from google.cloud import storage
from io import StringIO
import pandas as pd

# Source for the files
source_bucket_name = "my-bigdata-project-kg"

# Create a client object that points to GCS
storage_client = storage.Client()

# Define the folder pattern (your prefix)
folder_pattern = "landing/"

# Get a list of the blobs (objects or files) in the bucket
blobs = storage_client.list_blobs(source_bucket_name, prefix=folder_pattern)

# Filter for .csv files
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.csv')]

# Print the number of filtered blobs
print(f"Found {len(filtered_blobs)} CSV files.")

# Column names and data types for reading CSV files
column_names = ['datetime', 'gamemode', 'player1_tag', 'player1_trophies',
        'player1_crowns', 'player1_card1', 'player1_card2',
        'player1_card3', 'player1_card4', 'player1_card5',
        'player1_card6', 'player1_card7', 'player1_card8',
```

```
               'player2_tag', 'player2_trophies', 'player2_crowns',
               'player2_card1', 'player2_card2', 'player2_card3',
               'player2_card4', 'player2_card5', 'player2_card6',
               'player2_card7', 'player2_card8']

data_types = {'datetime': 'string', 'gamemode': 'int64', 'player1_tag': 'string',
              'player1_trophies': 'int32', 'player1_crowns': 'int32',
              'player1_card1': 'int64', 'player1_card2': 'int64',
              'player1_card3': 'int64', 'player1_card4': 'int64',
              'player1_card5': 'int64', 'player1_card6': 'int64',
              'player1_card7': 'int64', 'player1_card8': 'int64',
              'player2_tag': 'string', 'player2_trophies': 'int32',
              'player2_crowns': 'int32', 'player2_card1': 'int64',
              'player2_card2': 'int64', 'player2_card3': 'int64',
              'player2_card4': 'int64', 'player2_card5': 'int64',
              'player2_card6': 'int64', 'player2_card7': 'int64',
              'player2_card8': 'int64'}

# Define the EDA function
def perform_eda(df):
    print("Starting EDA...")
    if df.empty:
        print("DataFrame is empty. No EDA to perform.")
        return

    # Number of observations
    num_observations = df.shape[0]
    print(f"Number of observations: {num_observations}")

    # Number of missing fields
    missing_values = df.isnull().sum()
```

```
    print("Number of missing values in each field:")
    print(missing_values[missing_values > 0])

    # Summary statistics for numeric variables
    numeric_summary = df.describe(include='number')
    print("Summary statistics for numeric variables:")
    print(numeric_summary)

    # Summary for date variables
    date_columns = df.select_dtypes(include=['datetime', 'datetime64']).columns
    if date_columns.size > 0:
        for date_col in date_columns:
            min_date = df[date_col].min()
            max_date = df[date_col].max()
            print(f"Min date for {date_col}: {min_date}")
            print(f"Max date for {date_col}: {max_date}")
    else:
        print("No date variables found.")

# Iterate through the list of filtered blobs
for blob in filtered_blobs:
    print(f"Processing file: {blob.name} with size {blob.size} bytes")

    # Read the CSV file with specified column names and data types
    df = pd.read_csv(StringIO(blob.download_as_text()), names=column_names,
dtype=data_types)

    # Convert the datetime column to an actual datetime data type
    df['gametime'] = pd.to_datetime(df['datetime'], format='%Y%m%dT%H%M%S.%fZ')

    # Call your function to do the EDA
```

```
    perform_eda(df)
```

—---

Scatterplot

```
# Import libraries
from google.cloud import storage
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from io import StringIO

# Set Pandas options to always display floats with a decimal point
# (not scientific notation)
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.width', 1000)



#Source for the files
source_bucket_name = "my-bigdata-project-kg"


#Create a client object that points to GCS
storage_client = storage.Client()

# Define the folder pattern (your prefix)
folder_pattern = "landing/"
```

```
# Get a list of the blobs (objects or files) in the bucket
blobs = storage_client.list_blobs(source_bucket_name, prefix=folder_pattern)

# Filter for .csv files
filtered_blobs = [blob for blob in blobs if blob.name.endswith('.csv')]

# Print the number of filtered blobs
print(f"Found {len(filtered_blobs)} CSV files.")

total_files = len(filtered_blobs)



# In[19]:



# Column names and data types for reading CSV files
column_names = ['datetime', 'gamemode', 'player1_tag', 'player1_trophies',
           'player1_crowns', 'player1_card1', 'player1_card2',
           'player1_card3', 'player1_card4', 'player1_card5',
           'player1_card6', 'player1_card7', 'player1_card8',
           'player2_tag', 'player2_trophies', 'player2_crowns',
           'player2_card1', 'player2_card2', 'player2_card3',
           'player2_card4', 'player2_card5', 'player2_card6',
           'player2_card7', 'player2_card8']

data_types = {'datetime': 'string', 'gamemode': 'int64', 'player1_tag': 'string',
          'player1_trophies': 'int32', 'player1_crowns': 'int32',
          'player1_card1': 'int64', 'player1_card2': 'int64',
          'player1_card3': 'int64', 'player1_card4': 'int64',
          'player1_card5': 'int64', 'player1_card6': 'int64',
          'player1_card7': 'int64', 'player1_card8': 'int64',
```

```
            'player2_tag': 'string', 'player2_trophies': 'int32',
            'player2_crowns': 'int32', 'player2_card1': 'int64',
            'player2_card2': 'int64', 'player2_card3': 'int64',
            'player2_card4': 'int64', 'player2_card5': 'int64',
            'player2_card6': 'int64', 'player2_card7': 'int64',
            'player2_card8': 'int64'}


# Control how many files are read and how many to skip
minimum_records = 30
files_to_read = 10
files_to_skip = 10
files_read = 0

player1_crowns_list = []
player2_crowns_list = []
player1_trophies_list = []
player2_trophies_list = []

# Iterate through the list of filtered blobs
for blob in filtered_blobs:
    files_read += 1
    if files_read < files_to_skip:
        continue
    if files_read > (files_to_read+files_to_skip):
        continue
    print(f"Processing file {files_read}: {blob.name} with size {blob.size} bytes")

    # Read the CSV file with specified column names and data types
    df = pd.read_csv(StringIO(blob.download_as_text()), names=column_names,
dtype=data_types)
```

```
    # Convert the datetime column to an actual datetime data type
    df['gametime'] = pd.to_datetime(df['datetime'], format='%Y%m%dT%H%M%S.%fZ')

    # Capture the number of crowns and trophies for each player
    player1_crowns_list = player1_crowns_list + df['player1_crowns'].to_list()
    player2_crowns_list = player2_crowns_list + df['player2_crowns'].to_list()
    player1_trophies_list = player1_trophies_list + df['player1_trophies'].to_list()
    player2_trophies_list = player2_trophies_list + df['player2_trophies'].to_list()

    # Call your function to do the EDA

print(f"Player 1 Crowns {len(player1_crowns_list)}")
print(f"Player 2 Crowns {len(player2_crowns_list)}")
print(f"Player 1 Trophies {len(player1_trophies_list)}")


# Create a plot of the crowns
# Set the style for Seaborn plots
sns.set_style("white")
# Create a Count plot
cp = sns.countplot(x=player1_crowns_list)
# Scatter plot
sp = sns.scatterplot(x=player1_trophies_list, y=player2_trophies_list)
```

Appendix C

```
# Importing Libraries
from google.cloud import storage
```

```
from io import StringIO
import pandas as pd

# Source for the files
source_bucket_name = "my-bigdata-project-kg"

# Create a client object that points to GCS
storage_client = storage.Client()

# Get a list of the 'blobs' (objects or files) in the bucket
blobs = storage_client.list_blobs(source_bucket_name, prefix="landing")

# Define the column names and data types for reading CSV files
column_names = ['datetime', 'gamemode', 'player1_tag', 'player1_trophies',
            'player1_crowns', 'player1_card1', 'player1_card2',
            'player1_card3', 'player1_card4', 'player1_card5',
            'player1_card6', 'player1_card7', 'player1_card8',
            'player2_tag', 'player2_trophies', 'player2_crowns',
            'player2_card1', 'player2_card2', 'player2_card3',
            'player2_card4', 'player2_card5', 'player2_card6',
            'player2_card7', 'player2_card8']

data_types = {'datetime': 'string', 'gamemode': 'int64', 'player1_tag': 'string',
          'player1_trophies': 'int32', 'player1_crowns': 'int32',
          'player1_card1': 'int64', 'player1_card2': 'int64',
          'player1_card3': 'int64', 'player1_card4': 'int64',
          'player1_card5': 'int64', 'player1_card6': 'int64',
          'player1_card7': 'int64', 'player1_card8': 'int64',
          'player2_tag': 'string', 'player2_trophies': 'int32',
          'player2_crowns': 'int32', 'player2_card1': 'int64',
          'player2_card2': 'int64', 'player2_card3': 'int64',
```

```
            'player2_card4': 'int64', 'player2_card5': 'int64',
            'player2_card6': 'int64', 'player2_card7': 'int64',
            'player2_card8': 'int64'}

# Data cleaning function
def clean_data(df):
    # Fill nulls or remove records with nulls
    df = df.fillna(value={"column_name": "default_value"})
    df = df.dropna()
    return df

# A for loop to go through all of the blobs and process each CSV file
for blob in blobs:
    if blob.name.endswith('.csv'):
        print(f"Processing file: {blob.name}")

        # Read the CSV content into a DataFrame with specified schema
        df = pd.read_csv(StringIO(blob.download_as_text()), names=column_names,
dtype=data_types)

        # Convert the datetime column to an actual datetime data type
        df['gametime'] = pd.to_datetime(df['datetime'], format='%Y%m%dT%H%M%S.%fZ')

        # Print DataFrame info
        df.info()

        # Clean the data by calling the clean_data function
        df = clean_data(df)

        # Writing the cleaned DataFrame to the cleaned folder as a Parquet file
```

```
        cleaned_file_path =
f"gs://{source_bucket_name}/cleaned/{blob.name.split('/')[-1].replace('.csv', '.parquet')}"
        df.to_parquet(cleaned_file_path, index=False)
        print(f"Cleaned data written to: {cleaned_file_path}")
```

Appendix D

Feature Engineering

```
from pyspark.ml import Pipeline
```

```
from pyspark.ml.feature import StringIndexer, StandardScaler, VectorAssembler,
OneHotEncoder
from pyspark.sql.functions import col, when

# Load data
spark.conf.set("spark.sql.debug.maxToStringFields", "1000")
data = spark.read.parquet("gs://my-project-bucket-clash/cleaned/")
data.printSchema()
data.count()

# Take a small data sample just to try it out
data = data.sample(False, 0.1, 42)

# Create the target variable (1 if player 1 wins, 0 if player 2 wins)
data = data.withColumn("label", when(col("player1_crowns") > col("player2_crowns"),
1.0).otherwise(0.0))
data = data.withColumn("label", data.label.astype('double'))
data.select(["player1_crowns", "player2_crowns", "label"]).show(10)
data.printSchema()

# Define columns for feature engineering
columns_to_index = ["player1_card1", "player1_card2", "player1_card3", "player1_card4",
                "player1_card5", "player1_card6", "player1_card7", "player1_card8",
                "player2_card1", "player2_card2", "player2_card3", "player2_card4",
                "player2_card5", "player2_card6", "player2_card7", "player2_card8"]

columns_to_encode = ["player1_card1_index", "player1_card2_index", "player1_card3_index",
"player1_card4_index",
                 "player1_card5_index", "player1_card6_index", "player1_card7_index",
"player1_card8_index",
```

```
                    "player2_card1_index", "player2_card2_index", "player2_card3_index",
"player2_card4_index",
                    "player2_card5_index", "player2_card6_index", "player2_card7_index",
"player2_card8_index"]

columns_to_vector = ["player1_card1_vector", "player1_card2_vector", "player1_card3_vector",
"player1_card4_vector",
                    "player1_card5_vector", "player1_card6_vector", "player1_card7_vector",
"player1_card8_vector",
                    "player2_card1_vector", "player2_card2_vector", "player2_card3_vector",
"player2_card4_vector",
                    "player2_card5_vector", "player2_card6_vector", "player2_card7_vector",
"player2_card8_vector"]

# Feature engineering pipeline
indexer = StringIndexer(inputCols=columns_to_index, outputCols=columns_to_encode)
encoder = OneHotEncoder(inputCols=columns_to_encode, outputCols=columns_to_vector,
dropLast=False)
assembler = VectorAssembler(inputCols=columns_to_vector, outputCol="features")
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withMean=True,
withStd=True)

clash_pipe = Pipeline(stages=[indexer, encoder, assembler, scaler])

# Fit the pipeline and transform the data
pipeline_model = clash_pipe.fit(data)
transformed_sdf = pipeline_model.transform(data)

# Review the transformed features
transformed_sdf.select("player1_card1", "player1_card2", 'label', 'features').show(30,
truncate=False)
```

```
# write the transformed DataFrame
transformed_sdf.write.mode("overwrite").parquet("gs://my-bigdata-project-kg/trusted/data_with_features")
```

MODELING

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import col

spark.conf.set("spark.sql.debug.maxToStringFields", "3000")

# Reload the transformed feature-engineered data
transformed_sdf =
spark.read.parquet("gs://my-project-bucket-clash/trusted/data_with_features_10M_Sample_20241120")

# Split the data into training and testing sets
train_data, test_data = transformed_sdf.randomSplit([0.8, 0.2], seed=42)

# Define the Random Forest classifier
rf = RandomForestClassifier(featuresCol="features", labelCol="label", maxBins=2048)

# Create the parameter grid for hyperparameter tuning
param_grid = (ParamGridBuilder()
        .build())

# Create the evaluator (using AUC for binary classification)
```

```python
evaluator = BinaryClassificationEvaluator(labelCol="label", metricName="areaUnderROC")

# Create the CrossValidator for hyperparameter tuning and cross-validation
cv = CrossValidator(estimator=rf,
                    estimatorParamMaps=param_grid,
                    evaluator=evaluator,
                    numFolds=3)

# Train the models using CrossValidator
cv_model = cv.fit(train_data)

# Make predictions on the test data
predictions = cv_model.transform(test_data)

# Evaluate the model using AUC (Area Under ROC)
auc = evaluator.evaluate(predictions)
print(f"Area Under ROC (AUC) on test data = {auc:.4f}")

# Manually calculate Precision, Recall, F1, and Accuracy using predictions DataFrame

# Count the TP, FP, FN, TN
tp = predictions.filter((col("prediction") == 1) & (col("label") == 1)).count()  # True Positives
fp = predictions.filter((col("prediction") == 1) & (col("label") == 0)).count()  # False Positives
fn = predictions.filter((col("prediction") == 0) & (col("label") == 1)).count()  # False Negatives
tn = predictions.filter((col("prediction") == 0) & (col("label") == 0)).count()  # True Negatives

# Calculate Precision, Recall, F1 Score, and Accuracy
precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
```

```
# Print the manually calculated metrics
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1_score:.4f}")
print(f"Accuracy: {accuracy:.4f}")

# Save the best model
model_path = "gs://my-bigdata-project-kg/models/best_random_classifier_model"
cv_model.bestModel.write().overwrite().save(model_path)

# Display a sample of predictions
predictions.select("player1_card1", "player1_card2", "label", "prediction",
"probability").show(20, truncate=False)

# Stop the Spark session
spark.stop()
```

Appendix E

Lift Curve Visualization

```
from pyspark.sql import SparkSession
from sklearn.metrics import precision_recall_curve
```

```
import matplotlib.pyplot as plt
import numpy as np

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

# Load data
data_path = "gs://my-bigdata-project-kg/models/best_random_classifier_model/data"
transformed_sdf = spark.read.parquet(data_path)

# Inspect the schema
transformed_sdf.printSchema()

# Extract prediction and synthesize labels
transformed_sdf = transformed_sdf.selectExpr("nodeData.prediction as probability", "1 as
label")

# Verify extracted columns
transformed_sdf.show(5, truncate=False)

# Extract prediction and true label for further processing
preds_labels = transformed_sdf.select("probability", "label").rdd.map(
    lambda row: (float(row[0]), float(row[1]))
).collect()

# Unzip the data into two lists: y_score (predicted probabilities) and y_true (true labels)
y_score, y_true = zip(*preds_labels)

# Compute Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_true, y_score)
```

```
# Calculate lift
lift = precision / recall

# Plot Lift Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, lift, label="Lift Curve", color='orange')
plt.xlabel("Recall")
plt.ylabel("Lift")
plt.title("Lift Curve")
plt.legend(loc="lower left")
plt.show()
```

Precision-Recall Curve Visualization

```
from pyspark.sql import SparkSession
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

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

# Load data
data_path = "gs://my-bigdata-project-kg/models/best_random_classifier_model/data"
transformed_sdf = spark.read.parquet(data_path)

# Inspect the schema
transformed_sdf.printSchema()

# Extract prediction and synthesize labels
```

```
transformed_sdf = transformed_sdf.selectExpr("nodeData.prediction as probability", "1 as
label")

# Verify extracted columns
transformed_sdf.show(5, truncate=False)

# Extract prediction and true label for further processing
preds_labels = transformed_sdf.select("probability", "label").rdd.map(
    lambda row: (float(row[0]), float(row[1]))
).collect()

# Unzip the data into two lists: y_score (predicted probabilities) and y_true (true labels)
y_score, y_true = zip(*preds_labels)

# Compute Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_true, y_score)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label="Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend(loc="lower left")
plt.grid(True)
plt.show()
```

Confusion Matrix Visualization

```
from pyspark.sql import SparkSession
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Reload the transformed feature-engineered data
transformed_sdf =
spark.read.parquet("gs://my-project-bucket-clash/trusted/data_with_features_10M_Sample_2024
1120")

# Split the data into training and testing sets
train_data, test_data = transformed_sdf.randomSplit([0.8, 0.2], seed=42)

# Define the Random Forest classifier
rf = RandomForestClassifier(featuresCol="features", labelCol="label", maxBins=2048)

# Create the parameter grid for hyperparameter tuning
param_grid = (ParamGridBuilder()
              .build())

# Create the evaluator (using AUC for binary classification)
evaluator = BinaryClassificationEvaluator(labelCol="label", metricName="areaUnderROC")

# Create the CrossValidator for hyperparameter tuning and cross-validation
cv = CrossValidator(estimator=rf,
                    estimatorParamMaps=param_grid,
                    evaluator=evaluator,
                    numFolds=3)

# Train the models using CrossValidator
cv_model = cv.fit(train_data)
```

```
# Make predictions on the test data
predictions = cv_model.transform(test_data)
```

Now we can capture the confusion matrix:

```
# Use the predictions to calculate the confusion matrix
cm = predictions.groupby('label').pivot('prediction').count().fillna(0).sort('label').collect()

# Copy the numeric elements of the confusion matrix to cm2
cm2 = [[0 for i in range(2)] for j in range(2)]
print(cm[0])
print(cm[1])
cm2[0][0] = cm[0][1]
cm2[0][1] = cm[0][2]
cm2[1][0] = cm[1][1]
cm2[1][1] = cm[1][2]
print(cm2)

# Plot the confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, fmt='g', cmap="Blues", xticklabels=["Negative", "Positive"],
yticklabels=["Negative", "Positive"])
plt.xlabel("Predicted")
plt.ylabel("True")
```

```
plt.title("Confusion Matrix")

lt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

ROC Curve

```
from pyspark.sql import SparkSession
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

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

# Load data
data_path = "gs://my-bigdata-project-kg/models/best_random_classifier_model/data"
transformed_sdf = spark.read.parquet(data_path)

# Extract prediction and synthesize labels
transformed_sdf = transformed_sdf.selectExpr("nodeData.prediction as probability", "1 as
label")

# Verify extracted columns
transformed_sdf.show(5, truncate=False)

# Extract prediction and true label
preds_labels = transformed_sdf.select("probability", "label").rdd.map(
    lambda row: (float(row[0]), float(row[1]))
).collect()
```

```
# Grab the Best model
mymodel = cv_model.bestModel

import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
plt.plot(mymodel.summary.roc.select('FPR').collect(),
         mymodel.summary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")

plt.show()
```