BIG DATA PLATFORMS

Report Coursework

By Christopher Dillon

Matrix No: S1514278

03/05/2019

Table of Contents

Table of Figures

1.0 Overall Concept

1.1 Game Background

League of Legends (LoL) is a multiplayer online battle arena video game (Moba) by Riot Games which has gained a massive following over the years, becoming the most played game in 2019 (Newzoo, 2019). The game involves a similar setup to mainstream sports such as football whereby a set number of players (five in this instance) are matched against a similar team with the same number of players, each with the same end goal; destroying the enemies' nexus. Each team is assigned a side (e.g. blue side or red side) every match with slight differences to each of the sides being played on. The roles a player can play include top lane, middle lane, jungle and bottom lane (attack damage carry and support as two separate roles). The objective of the game is to get to the enemies 'Nexus' and destroy it. In order to do this, the players must work as a team taking objectives: towers, dragon, rift herald and baron while killing the enemy champions which resets them back to their base situated before their nexus. Gold is awarded for killing other players' champions and objectives, which can be used to buy items making your champion and team stronger. A map of the battleground played on can be seen in Figure 1

The League of Legends Competitive matches is data collected from professional games within the period of 2015-2018. Professional matches are played similar to real sports where there are huge prize pools on the line and each organisation has a roster along with substitute players etc. The game has hundreds of small individual variables that cannot be accurately measured by hand. While the original download contains eight total excel documents, the main focus will be centred around the 'LeagueofLegends' csv file. The other csv files considered are 'gold' and 'kills' which are extensions to the original file. By analysing these games using a data pipeline, teams and players' performances can be highlighted and can be shown their potential weaknesses/strengths to improve their play. Furthermore, this pipeline could be fed data live through the games API which can help predict a game's outcome in real-time.

Figure 1: The map outlining the 'battleground' of the MOBA League of Legends

1.2 Illustration of Data Scheme

Feature	Description
League	The region the game is played on (e.g. North America)
Year	The region the game is played on (e.g. North America)
Season	The literal season the game was played (e.g. Spring, Summer)
Type	The literal season the game was played (e.g. Spring, Summer)
blueTeamTag	The esports team (organisation) playing on this side (e.g. Cloud 9, Team Solo Mid)
bResult	Whether the blue team wins the game or not
rResult	Whether the red team wins the game or not
redTeamTag	The esports team (organisation) playing on this side
gamelength	The length of the match
golddiff	Gold difference between teams (always calculated as blue minus red) every minute
goldblue	The team playing on blue sides total gold value every minute
bKills	List of the blue team's kills showing kill information for each champion slain
bTowers	List of the blue team's tower kills showing information on each tower destroyed
bInhibs	List of the blue team's inhibitor kills showing the timestamp of each inhibitor destroyed
bDragons	List of the blue team's dragon kills and the timestamp of when they were killed.
bHeralds	List of blue team's rift herald kills and the timestamp of when they were killed
rKills	List of the blue team's kills showing kill information for each champion slain
rTowers	List of the blue team's tower kills showing information on each tower destroyed
rInhibs	List of the blue team's inhibitor kills showing the timestamp of each inhibitor destroyed
rDragons	List of the blue team's dragon kills and the timestamp of when they were killed.
rHeralds	List of red team's rift herald kills and the timestamp of when they were killed
blueTop	The player who is playing top lane for blue side (e.g. their gamer tag)
blueTopChampion	The champion the blue side top laner is playing
goldBlueTop	The overall gold the blue side top lane player has per minute
blueJungle	The player who is playing the jungle role (e.g. their gamer tag)
goldBlueJungle	The overall gold the blue side jungle has per minute
blueMiddle	The player who is playing the middle lane for blue side (e.g. their gamer tag)
blueMiddleChamp	The champion the blue side mid laner is playing
goldBlueMiddle	The champion the blue side mid laner is playing
blueADC	The player who is playing the 'attack damage carry' bottom lane for blue side (e.g. their gamer tag)
blueADCChampion	The champion the blue side 'attack damage carry' laner is playing
goldBlueADC	The overall gold the blue side the 'attack damage carry' bottom lane player has per minute
blueSupport	The player who is playing the 'support' bottom lane for blue side (e.g. their gamer tag)
blueSupportChamp	The champion the blue side support player is playing
goldBlueSupport	The champion the blue side support player is playing
blueSupport	The player who is playing the 'support' bottom lane for blue side (e.g. their gamer tag)
blueSupportChamp	The champion the blue side support player is playing
goldBlueSupport	The overall gold the blue side the 'support' bottom lane player has per minute
blueBans	The champion bans the blue team decided before the match
redBans	The champion bans the red team decided before the match
redTop	The player who is playing top lane for red side (e.g. their gamer tag)
redTopChampion	The champion the red side top laner is playing
goldRedTop	The overall gold the red side top lane player has per minute
redJungle	The player who is playing the jungle role (e.g. their gamer tag)
redJungleChampion	The champion the red side jungle is playing
goldRedJungle	The overall gold the red side jungle player has per minute
redMiddle	The player who is playing the middle lane for red side (e.g. their gamer tag)
redMiddleChampion	The player who is playing the middle lane for red side (e.g. their gamer tag)
goldRedMiddle	The overall gold the red side jungle player has per minute
redADC	The player who is playing the 'attack damage carry' bottom lane for red side (e.g. their gamer tag)
redADCChampion	The champion the red side 'attack damage carry' laner is playing
goldRedADC	The overall gold the red side the 'attack damage carry' bottom lane player has per minute
redSupport	The player who is playing the 'support' bottom lane for red side (e.g. their gamer tag)
redSupportChamp	The champion the red side support player is playing
goldRedSupport	The overall gold the red side the 'support' bottom lane player has per minute

Table 1: Data illustration showing the features of the 'LeagueofLegends' CSV

1.3 Visualisation Dashboard

Finding useful insight on w*hy* a game was won is the sole purpose of this experiment. From the initial interpretation of the data scheme, the following areas will be mainly focused on:

- Gold difference between the teams. How does a gold lead impact game success? Do certain teams win consistently with their gold lead? Is the opposite for other teams true? Do they usually throw the game?
- Objectives slain (dragon, towers, heralds, barons, inhibitors). Objectives give perks such as empowered abilities, additional gold etc. Having these obviously increases a team's chance at success, but is there any specific objective that when taken always results in a win? Do other objectives have little impact on a team's probability of winning?
- Game length. Do certain teams play better when games stagnate? Do some teams consistently win games shorter than a given time length? Traditionally some champions scale better as the game goes on and others 'fall off' and become irrelevant. This could give insight to the 'type' of champions a team usually selects.
- Kill positioning. By creating a visualisation of kills throughout games we can gain an understanding of where most deaths occur during a game. This could be useful for teams as the result may show that a player is consistently dying at the same spot and prompt them to change their positioning in future games.
- Game side. It is frequently debated that playing on the blue side has a statistical advantage over playing on red side, win blue side winning approximately 4 in every 7 games (Kee, 2018). With the data available, this should be able to be confirmed even at the highest level of play.

The aforementioned variables are considered constants and have not changed over the games patch history. Frequently, the game is 'patched' whereby features such as champions are changed (becoming less or more powerful). Champions and champion bans were not considered as they would only be considered important features to impact games *within* the given patch. As patches normally occur every 2 weeks, these features are not considered due to the scale of the data.

1.4 Potential Ad-hoc Queries

From initial analysis of the raw data, the following ad-hoc queries will be potentially carried out:

- Drop all irrelevant tables relating to champions, bans etc.
- Show missing values to determine how noisy the data is
- Filter out rows with missing values
- Count objectives such as towers and inhibitors destroyed
- Determine the variable type of each column and assign appropriate datatypes
- Scalable operations - Lambda, MapReduce, Map, ReduceByKey

2.0 Platforms

2.1 Data Store

MongoDB was selected due its schema-less architecture as final structure of the data is not currently known and this feature allows the data to be moulded and joined with limitless possibilities. Its high scalability makes it ideal for real-time streaming of data (Jayaram, 2016). With MongoDB also being loosely coupled, this can be beneficial for an ever changing game. New features are added all the time to League of Legends meaning variables will be changed and added over time.

2.2 File System

Apache Kafka is an ideal file system due to being fault-tolerant, fast, scalable and ideal for handling live stream data. It accomplishes these features using its own distributed publish-subscribe messaging system Being distributed, it is run on multiple machines which work together in a cluster, appearing as one single node. By being optimised for horizontal scaling, no downtime is ever required and theoretically the number of additional nodes that could be added is limitless. This is optimal for streaming live game data due to the fault tolerant nature of the architecture and being ideal with its scalability as the game is continually achieving exponential growth. As for the prototype variant, the file system will remain as a local virtual machine.

2.3 Analytic Engine

Apache Spark is deemed most suitable for this prototype experiment due to its ability to process real time data and whereas Hadoop is more suitable for batch type processing (Bhandari, n.d.). This, along with its in memory processing feature (Ibm.com, n.d.), makes it ideal for building Extract, Transform, Load data pipelines. Spark works as a cluster, using SparkContext, and its cluster management system, managing each cluster using both ‘driver’ and ‘worker’ type of nodes (Velida, n.d.). With the introduction of SparkSQL, working with structured data is significantly easier as the data is organised into a DataFrame, with columns, similar to the traditional relational database format. Regular Hadoop does not support this, but rather a resilient distribute dataset, making Spark a more suitable option due to the nature of the data. Spark also supports receiving data from both local sources such as CSV files and a file system such as Kafka.

2.4 Programming Platform

Jupyter Notebook using Python was selected for the prototype version of this experiment. Python allows for impressive analytics using its Pandas library making it ideal for the visualisation component of this experiment. By also having libraries such as PyMongo and PySpark which allow access to the data store MongoDB and the analytic engine Spark, these technologies can be easily integrated into a singular Jupyter Notebook. By creating a prototype experiment of this type, it gives us the ability to see how these platforms would handle this kind of scenario.

2.5 Riot Games API

As games are played and streamed live, there is an API available from Riot Games that can grab live game data. Theoretically, this could be streamed live as the players are playing, giving insights of the performance of the match in real time. All the variables included in original player match history can be gathered live as the match being played. Variables such as gold per minute can give immediate insights on which players are doing well in the match and this could be referenced with the insights given on the past data.

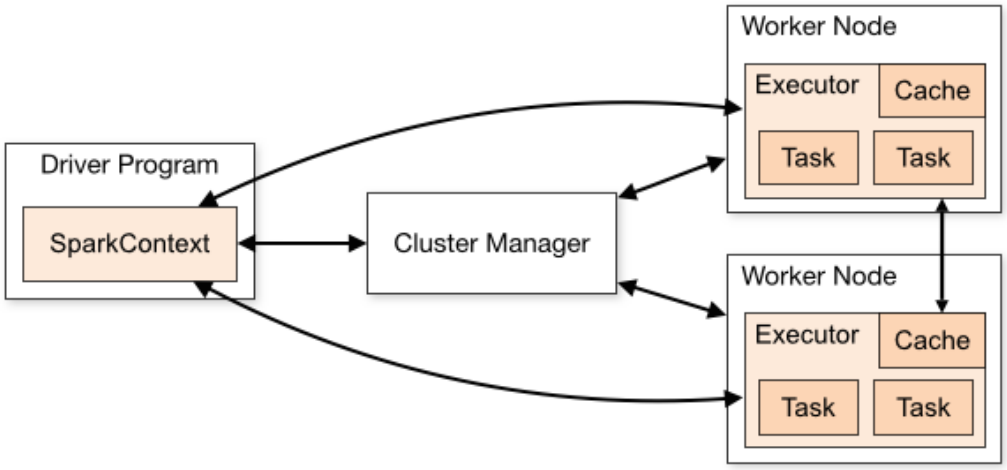


Figure 2: Diagram highlighting the cluster management underlying system (source: spark.apache.org)

3.0 Integration & Deployment

3.1 Software Requirements

The software components required to generate a prototype version of this model are as follows:

- MongoDB
- Apache Spark
- Apache Kafka
- Python Version 2.7 or higher
- Python Libraries: PyMongo, PySpark, Kafka, Pandas, NumPy, MatPlotLib, Seaborn

3.2 Ingestion

For the prototype, ingestion will be carried out locally using the given CSV files. This will be carried out by configuring the virtual machine to run PySpark. This can be integrated with Jupyter Notebook, making it ideal for prototyping purchases.

Live match data can be gathered by Riot Games 'MATCH-V4' API section (shown in Figure 2). The data grabbed from the API is very similar to the data originally gathered in the CSV file, making it ideal for integrating with this system. This data can ingested via Kafka and passed onto the processing engine, Apache Spark in this instance.

Figure 3: Shows the 'MATCH-V4' section of Riot Games' API used to access game data

3.3 Extract

Extraction will begin by importing PySpark with the purpose of partitioning the given dataset. The object 'SparkContext' represents a computing cluster connection and 'SQLContext' is used to create a data frame and as an entry point to working with the structured data (Navarro, 2016). This will load the data into a cluster with a given number of partitions. For the prototype, the data will be read locally from the CSV files. In a real world scenario, the data would be loaded from the Kafka after the ingestion phase.

3.4 Transformation

After the data has been read into Spark, the data gathered from the 'LeagueofLegends' and 'Kills' file will be merged. The 'gold' CSV file will be kept separate in a different DataFrame due to its huge volume of columns and it being able to be processed independently from the two other data sources. As the dataframes are Spark SQL based, this means SQL statements can be used to filter the data. Spark SQL also offers functions such as filter, withColumn and select. As this technology is built with scalability in mind, Lambda functions in conjunction with Map, MapReduce, ReduceByKey can also be utilised to filtering and transforming data. Custom functions can be used to carry out more complicated queries and more custom filtering.

3.5 Load

Once the data has been transformed, it can then be loaded into a MongoDB database by utilising MongoDB's API available on the Python language using the library PyMongo. Mongo accepts Spark SQLs format and allows for it to be easily appended into a given database. By having the data in a MongoDB format, it allows for extremely fast querying and filtering along with easy interaction with the Pandas library, allowing for easy visualisation of the data.

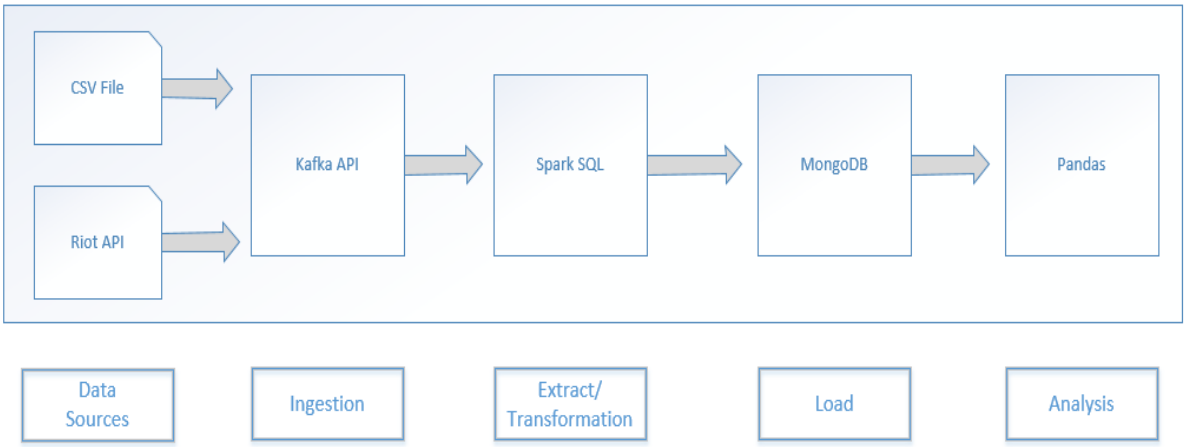


Figure 4: Block Diagram showing theoretical data pipeline

4.0 Conclusion

In the prototype demonstration, it was shown that game data can be filtered and transformed in a scalable fashion. The analysis showed how game data could theoretically help predict the outcome of a game in a real-time fashion. The insights received from this data could be used to assist commentators in real-time, giving them additional stats and information to go on while commentating a live game. The data could also be used by teams to assess progress amongst players and compare their current performance to their prior through analysis.

5.0 References

1. Newzoo. (2019). *Most Popular Core PC Games | Global | Newzoo*. [online] Available at: https://newzoo.com/insights/rankings/top-20-core-pc-games/ [Accessed 23 Apr. 2019].

2. Bhandari, T. (n.d.). *Hadoop Vs Spark: Which Is The Best Data Analytics Engine? | Articles | Big Data*. [online] Channels.theinnovationenterprise.com. Available at: https://channels.theinnovationenterprise.com/articles/hadoop-vs-spark-which-is-the-best-data-analytics-engine [Accessed 25 Apr. 2019].

3. Ibm.com. (n.d.). *PySpark High-performance data processing without learning Scala*. [online] Available at: https://www.ibm.com/downloads/cas/DVRQZYOE [Accessed 29 Apr. 2019].

4. Navarro, Á. (2016). *Understanding the Data Partitioning Technique*. [online] Datio. Available at: https://www.datio.com/iaas/understanding-the-data-partitioning-technique/ [Accessed 30 Apr. 2019].

5. Kee, J. (2018). *Why Does Blue Side Win More Games In League of Legends? - 5v5 Esports Malaysia*. [online] 5v5 Esports Malaysia. Available at: https://www.5v5esports.com/lol/why-does-blue-side-win-more-games-in-league-of-legends/ [Accessed 3 May 2019].

6. Velida, W. (n.d.). *How does Apache Spark run on a cluster?*. [online] Towards Data Science. Available at: https://towardsdatascience.com/how-does-apache-spark-run-on-a-cluster-974ec2731f20 [Accessed 3 May 2019].

In [109]:

```
#EXTRACTION PHASE
```

In [110]:

```
sc
```

Out[110]:

```
<pyspark.context.SparkContext at 0x7f82d9d74d10>
```

In [111]:

```
#Spark Imports
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf

#Create an sqlContext with a default of 20 partitions
spark=SparkSession.builder.appName('Sample').getOrCreate()
sqlContext.sql("set spark.sql.shuffle.partitions=20")
```

Out[111]:

```
DataFrame[key: string, value: string]
```

In [112]:

```
#Read datasources into Spark Dataframes
lol = spark.read.csv('LeagueofLegends.csv', header='true', inferSchema='true')
kills = spark.read.csv('kills.csv', header='true', inferSchema='true')
gold = spark.read.csv('gold.csv', header='true', inferSchema='true')
```

In [113]:

```
#TRANSFORMATION PHASE
```

In [114]:

```
#Join kills & main dataframe together
df=lol.join(kills,'Address', how='inner')
df.show(1)
```

```
+--------------------+------+----+------+--------+----------+-------+-------+---------+---------+------------------+------------------+------------------+------------
-------+-------+--------+-------+-------+------------------+------------------+-----------------+------------------+------------------+------------------+--------+-
------+-----------+------------------+---------+-------------+------------------+---------+-------------+------------------+--------+-----------+-----------------
--+----------+---------------+-----------------+------------------+------+----------+-----------------+--------+-------------+------------------+--------+------
-------+------------------+------+----------+------------------+---------+-------------+------------------+------------------+-----+-----+----------+-------+---
-----+--------+--------+--------+-----+-----+
|             Address|League|Year|Season|    Type|blueTeamTag|bResult|rResult|redTeamTag|gamelength|          golddiff|          goldblue|              bKi
lls|          bTowers|bInhibs|bDragons|bBarons|bHeralds|           goldred|           rKills|           rTowers|           rInhibs|           rDragons|              rB
arons|rHeralds|blueTop|blueTopChamp|       goldblueTop|blueJungle|blueJungleChamp|    goldblueJungle|blueMiddle|blueMiddleChamp|    goldbl
ueMiddle|  blueADC|blueADCChamp|       goldblueADC|blueSupport|blueSupportChamp|    goldblueSupport|          blueBans|redTop|redTopCham
p|         goldredTop|redJungle|redJungleChamp|      goldredJungle|redMiddle|redMiddleChamp|      goldredMiddle| redADC|redADCChamp|         g
oldredADC|redSupport|redSupportChamp|      goldredSupport|          redBans|  Team| Time|    Victim|  Killer|Assist_1|Assist_2|Assist_3|Assist_4|x
_pos|y_pos|
+--------------------+------+----+------+--------+----------+-------+-------+---------+---------+------------------+------------------+------------------+------------
-------+-------+--------+-------+-------+------------------+------------------+-----------------+------------------+------------------+------------------+--------+-
------+-----------+------------------+---------+-------------+------------------+---------+-------------+------------------+--------+-----------+-----------------
--+----------+---------------+-----------------+------------------+------+----------+-----------------+--------+-------------+------------------+--------+------
-------+------------------+------+----------+------------------+---------+-------------+------------------+------------------+-----+-----+----------+-------+---
-----+--------+--------+--------+-----+-----+
|http://matchhisto...| CBLoL|2017|Spring|Playoffs|    PNG|  0|  1|   RED|      34|[0, 0, -48, 11, 5...|[2500, 2500, 2729...|[[3.543, 'RED YoD...|[
[16.035, 'TOP_LA...|    []|    []|    []|    []|[2500, 2500, 2777...|[[8.003, 'PNG Sir...|[[28.25, 'BOT_LAN...|[[28.534, 'MID_LA...|[[16.034, 'EARTH_...|[[25
.446], [32.58]]|    []|  Mylon|    Maokai|[500, 500, 500, 8...|    SirT|    LeeSin|[500, 500, 612, 1...|   Kami|    Orianna|[500, 500, 577, 8...|Mats
ukaze|    Jhin|[500, 500, 520, 8...|   Loop|    TahmKench|[500, 500, 520, 7...|['Graves', 'Cassi...|  Robo|    Shen|[500, 500, 500, 8...|  Nappon|
Khazix|[500, 500, 612, 9...|  YoDa SL|     Ahri|[500, 500, 557, 9...|brTT RX|    Ashe|[500, 500, 560, 8...|  Dioud|    Malzahar|[500, 500, 548, 7..
.|['Rumble', 'Syndr...|bKills|3.543|RED YoDa SL|PNG SirT|PNG Kami|   null|   null|   null| 8680| 8457|
+--------------------+------+----+------+--------+----------+-------+-------+---------+---------+------------------+------------------+------------------+------------
-------+-------+--------+-------+-------+------------------+------------------+-----------------+------------------+------------------+------------------+--------+-
------+-----------+------------------+---------+-------------+------------------+---------+-------------+------------------+--------+-----------+-----------------
--+----------+---------------+-----------------+------------------+------+----------+-----------------+--------+-------------+------------------+--------+------
-------+------------------+------+----------+------------------+---------+-------------+------------------+------------------+-----+-----+----------+-------+---
-----+--------+--------+--------+-----+-----+
```

only showing top 1 row

In [115]:

```
#Drop all irrelevant columns
drop_list = ['League', 'Season', 'Type', 'Year', 'golddiff', 'goldblue', 'goldred',
             'blueTop', 'blueTopChamp', 'goldblueTop', 'blueJungle', 'blueJungleChamp', 'goldblueJungle', 'blueMiddle',
             'blueMiddleChamp', 'goldblueMiddle', 'blueADC', 'blueADCChamp', 'goldblueADC', 'blueSupport', 'blueSupportChamp',
             'goldblueSupport', 'blueBans',
             'redTop', 'redTopChamp', 'goldredTop', 'redJungle', 'redJungleChamp', 'goldredJungle', 'redMiddle',
             'redMiddleChamp', 'goldredMiddle', 'redADC', 'redADCChamp', 'goldredADC', 'redSupport', 'redSupportChamp',
             'goldredSupport', 'redBans',
             'rKills', 'bKills', 'Team', 'Time', 'Victim', 'Killer', 'Assist_1', 'Assist_2', 'Assist_3', 'Assist_4']

df = df.select([column for column in df.columns if column not in drop_list])
```

In [116]:

```
#Shows missing values
from pyspark.sql.functions import *
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show(1)
```

```
+-------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
|Address|blueTeamTag|bResult|rResult|redTeamTag|gamelength|bTowers|bInhibs|bDragons|bBarons|bHeralds|rTowers|rInhibs|rDragons|rBarons|rHeralds|x_pos|y_pos|
+-------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
|      0|        896|      0|      0|       848|         0|      0|      0|       0|      0|       0|      0|      0|       0|      0|       0|  113|  113|
+-------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
```

In [117]:

```
#Remove rows with null values
df = df.filter(df.blueTeamTag.isNotNull()).filter(df.redTeamTag.isNotNull()).filter(df.x_pos. isNotNull())
```

In [118]:

```
from pyspark.sql.functions import col, size, split

#Found consistencies in the data relating to the naming of objectives
#Counts the given value and replaces the current array data with that count
df = df.withColumn('rTowers', size(split(col("rTowers"), "TOWER")))
df = df.withColumn('bTowers', size(split(col("bTowers"), "TOWER")))
df = df.withColumn('bInhibs', size(split(col("bInhibs"), "LANE")))
df = df.withColumn('rInhibs', size(split(col("rInhibs"), "LANE")))
df = df.withColumn('bDragons', size(split(col("bDragons"), "DRAGON")))
df = df.withColumn('rDragons', size(split(col("rDragons"), "DRAGON")))

df.show(1)
```

```
+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
|             Address|blueTeamTag|bResult|rResult|redTeamTag|gamelength|bTowers|bInhibs|bDragons|bBarons|bHeralds|rTowers|rInhibs|rDragons|rBarons|rHeralds|x_pos|y_pos|
+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
|http://matchhisto...|        PNG|      0|      1|       RED|        34|      1|      1|       1|     []|      []|      1|      4|       5|[[25.446], [32.58]]|      []| 8680| 8457|
+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+-------+--------+-------+-------+--------+-------+--------+-----+-----+
only showing top 1 row
```

In [119]:

```
#Problematic assigned data types. Most arrays have automatically been assigned as strings as well as numerical data
df.printSchema()
```

```
root
 |-- Address: string (nullable = true)
 |-- blueTeamTag: string (nullable = true)
 |-- bResult: integer (nullable = true)
 |-- rResult: integer (nullable = true)
 |-- redTeamTag: string (nullable = true)
 |-- gamelength: integer (nullable = true)
```

```
 |-- bTowers: integer (nullable = false)
 |-- bInhibs: integer (nullable = false)
 |-- bDragons: integer (nullable = false)
 |-- bBarons: string (nullable = true)
 |-- bHeralds: string (nullable = true)
 |-- rTowers: integer (nullable = false)
 |-- rInhibs: integer (nullable = false)
 |-- rDragons: integer (nullable = false)
 |-- rBarons: string (nullable = true)
 |-- rHeralds: string (nullable = true)
 |-- x_pos: string (nullable = true)
 |-- y_pos: string (nullable = true)
```

In [120]:

```
#Function used to cast columns as the given type and remove any brackets using regex
def convertColumnType(df, names, newType):
    for name in names:
        if newType == "array<int>":
            df = df.withColumn(name, split(regexp_replace(name, r"(^\[\[)|(\]\]$)", ""), ", ").cast("array<int>"))
        else:
            df = df.withColumn(name, df[name].cast(newType))
    return df
```

In [121]:

```
#Arrays holding values to be used with the convertColumnType function
toInt = ['rTowers', 'bTowers', 'bInhibs', 'rInhibs', 'bDragons', 'rDragons', 'x_pos', 'y_pos']

#Casting these columns to array allows for array manipulation in later transformations
toArray = ['rBarons', 'bBarons', 'rHeralds', 'bHeralds']

df = convertColumnType(df, toArray, "array<int>")
df = convertColumnType(df, toInt, "int")

df.printSchema()
df.show(1)
```

```
root
 |-- Address: string (nullable = true)
 |-- blueTeamTag: string (nullable = true)
 |-- bResult: integer (nullable = true)
 |-- rResult: integer (nullable = true)
 |-- redTeamTag: string (nullable = true)
 |-- gamelength: integer (nullable = true)
 |-- bTowers: integer (nullable = false)
 |-- bInhibs: integer (nullable = false)
 |-- bDragons: integer (nullable = false)
 |-- bBarons: array (nullable = true)
 |    |-- element: integer (containsNull = true)
 |-- bHeralds: array (nullable = true)
 |    |-- element: integer (containsNull = true)
 |-- rTowers: integer (nullable = false)
 |-- rInhibs: integer (nullable = false)
 |-- rDragons: integer (nullable = false)
 |-- rBarons: array (nullable = true)
 |    |-- element: integer (containsNull = true)
 |-- rHeralds: array (nullable = true)
 |    |-- element: integer (containsNull = true)
 |-- x_pos: integer (nullable = true)
 |-- y_pos: integer (nullable = true)

+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+--------+--------+--------+--------+--------+------------+--------+-----+-----+
|             Address|blueTeamTag|bResult|rResult|redTeamTag|gamelength|bTowers|bInhibs|bDragons|bBarons|bHeralds|rTowers|rInhibs|rDragons|
rBarons|rHeralds|x_pos|y_pos|
+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+--------+--------+--------+--------+--------+------------+--------+-----+-----+
|http://matchhisto...|        PNG|      0|      1|       RED|        34|      1|      1|       1| [null]|  [null]|      1|      4|       5|[null, null]|  [null]| 8680| 8457|
+--------------------+-----------+-------+-------+----------+----------+-------+-------+--------+--------+--------+--------+--------+--------+------------+--------+-----+-----+
only showing top 1 row
```

In [122]:

```
#Function designed for counting array elements and casting the result an int
def countArrayElements(df, names):
    for name in names:
        df = df.withColumn(name, size(name).cast('int'))
    return df
```

In [123]:

```
#Count the array entries for these variables as they have no unique string occurence like in the dragon columns for instance
arrayObjects = ['rBarons', 'bBarons', 'rHeralds', 'bHeralds']
df = countArrayElements(df, arrayObjects)
```

In [126]:

```
#Much cleaner dataset
df.show(1)
```

```
+--------------------+----------+------+------+---------+---------+------+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-----+-----+
|             Address|blueTeamTag|bResult|rResult|redTeamTag|gamelength|bTowers|bInhibs|bDragons|bBarons|bHeralds|rTowers|rInhibs|rDragons|rB
arons|rHeralds|x_pos|y_pos|
+--------------------+----------+------+------+---------+---------+------+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-----+-----+
|http://matchhisto...|       PNG|     0|     1|      RED|       34|     1|     1|      1|      1|      1|      1|      4|      5|      2|      1| 8680| 8457|
+--------------------+----------+------+------+---------+---------+------+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-----+-----+
only showing top 1 row
```

In [128]:

```
#All types are correctly assigned
df.printSchema()
```

```
root
 |-- Address: string (nullable = true)
 |-- blueTeamTag: string (nullable = true)
 |-- bResult: integer (nullable = true)
 |-- rResult: integer (nullable = true)
 |-- redTeamTag: string (nullable = true)
 |-- gamelength: integer (nullable = true)
 |-- bTowers: integer (nullable = false)
 |-- bInhibs: integer (nullable = false)
 |-- bDragons: integer (nullable = false)
 |-- bBarons: integer (nullable = false)
 |-- bHeralds: integer (nullable = false)
 |-- rTowers: integer (nullable = false)
 |-- rInhibs: integer (nullable = false)
 |-- rDragons: integer (nullable = false)
 |-- rBarons: integer (nullable = false)
 |-- rHeralds: integer (nullable = false)
 |-- x_pos: integer (nullable = true)
 |-- y_pos: integer (nullable = true)
```

In [ ]:

```
#LOAD PHASE
```

In [129]:

```
#Write to mongodb
df.write.format("com.mongodb.spark.sql.DefaultSource") \
.mode("append") \
.option("database","test") \
.option("collection", "league").save()

gold.write.format("com.mongodb.spark.sql.DefaultSource") \
.mode("append") \
.option("database","test") \
.option("collection", "gold").save()
```

In [130]:

```
#Import MongoClient to interact with saved MongoDB
import pymongo
from pymongo import MongoClient
client = MongoClient()
```

In [131]:

```
import pandas as pd
```

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

In [ ]:

```
#VISUALISATION PHASE
```

In [132]:

```
#Test mongodb
result = client.test.league.find()
lol= pd.DataFrame(list(result))
lol.head(3)
```

Out[132]:

	Address	_id	bBarons	bDragons	bHeralds	bInhibs	bResult	bTowers	bI
0	http://matchhistory.br.leagueoflegends.com/pt/...	5ccc76eee88b100c0a83d08f	1	1	1	1	0	1	P
1	http://matchhistory.br.leagueoflegends.com/pt/...	5ccc76eee88b100c0a83d090	1	1	1	1	0	1	P
2	http://matchhistory.br.leagueoflegends.com/pt/...	5ccc76eee88b100c0a83d091	1	1	1	1	0	1	P

In [133]:

```
#Create seperate DF for wins analysis
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = lol.copy(deep=True)
df['win_team'] = np.where(df['bResult']==1, 'blue', 'red')
df[['win_team', 'bResult', 'rResult']].head(2)
```

Out[133]:

	win_team	bResult	rResult
0	red	0	1
1	red	0	1

In [134]:

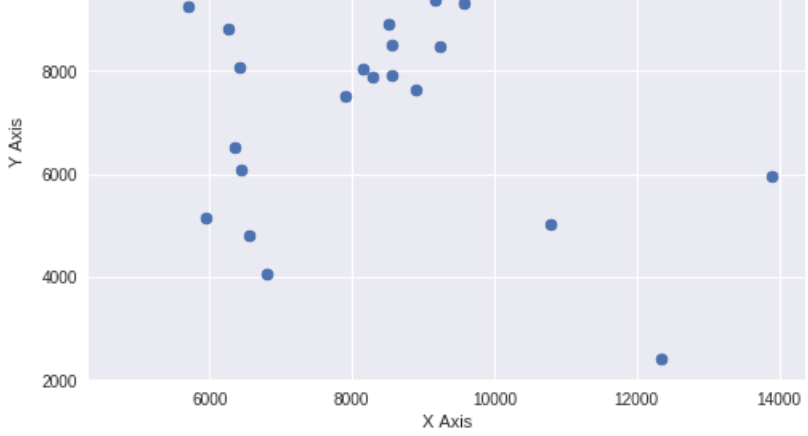
```
#Generate a sample game
sampleGame = lol.loc[lol['Address'] == "http://matchhistory.na.leagueoflegends.com/en/#match-details/TRLH1/30030?gameHash=fbb300951ad8327c"]
```

In [135]:

```
#Grab a sample team 'TSM', both their play history on both sides of the map
tsmBlueSide = lol.loc[lol['blueTeamTag'] == "TSM"]
tsmRedSide = lol.loc[lol['redTeamTag'] == "TSM"]
```

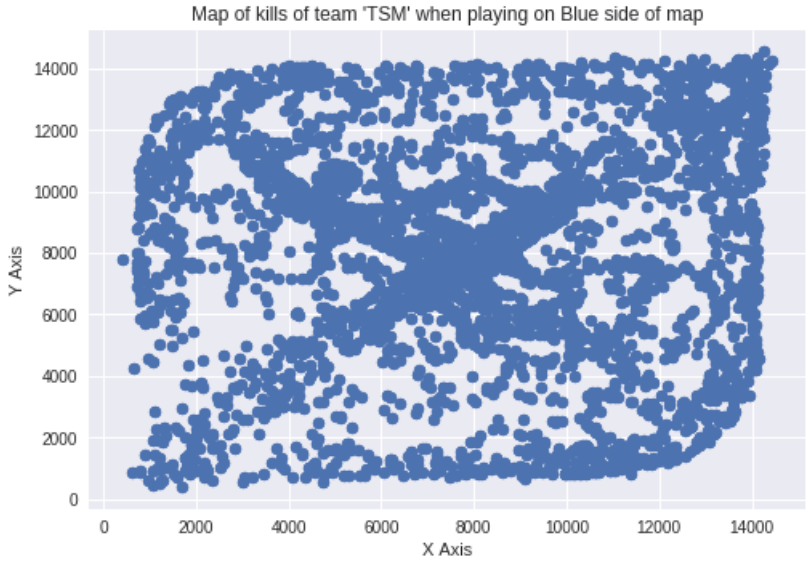
In [136]:

```
#Map a sample games kills
#As we can see most kills appeared on the top side of the map
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.scatter(sampleGame.x_pos, sampleGame.y_pos)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
plt.title("Map of kills in a singular profession match")
plt.show()
```



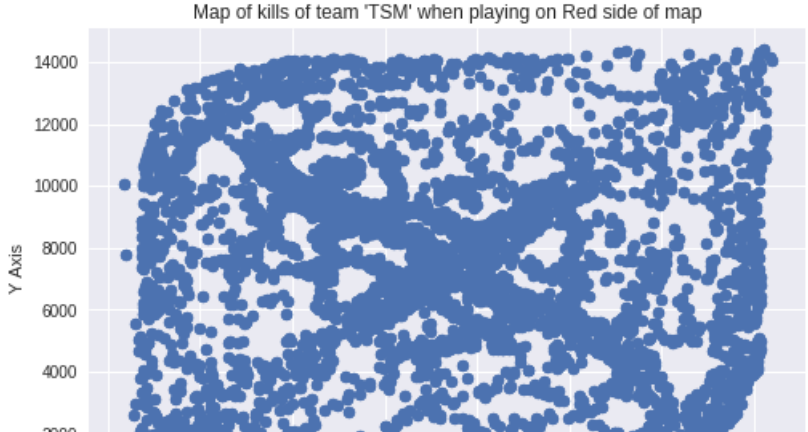
In [137]:

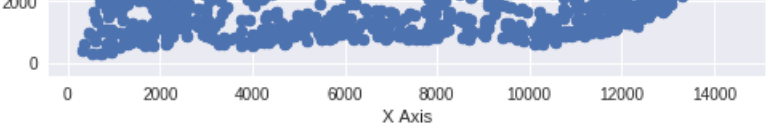
```
#Sample kill map of the team 'TSM' when playing blue side
#The map looks almost identical to the actual map the game is played on
fig, ax = plt.subplots()
ax.scatter(tsmBlueSide.x_pos, tsmBlueSide.y_pos)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
plt.title("Map of kills of team 'TSM' when playing on Blue side of map")
plt.show()
```



In [138]:

```
#Sample kill map of the team 'TSM' when playing red side
#More kills appear to be present in the opposite teams base in both instances,
#suggesting they are playing successfully over their entire play history

fig, ax = plt.subplots()
ax.scatter(tsmRedSide.x_pos, tsmRedSide.y_pos)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
plt.title("Map of kills of team 'TSM' when playing on Red side of map")
plt.show()
```

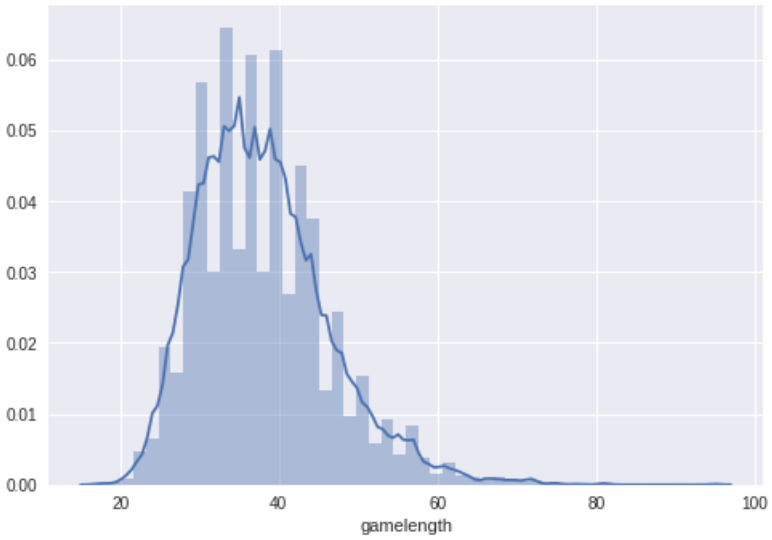




In [139]:

```
#Generate overall game length data
print('Max game length:', df.gamelength.max())
print('Min game length:', df.gamelength.min())
print('Mean game length:', df.gamelength.mean())
sns.distplot(df.gamelength)
plt.show()
```

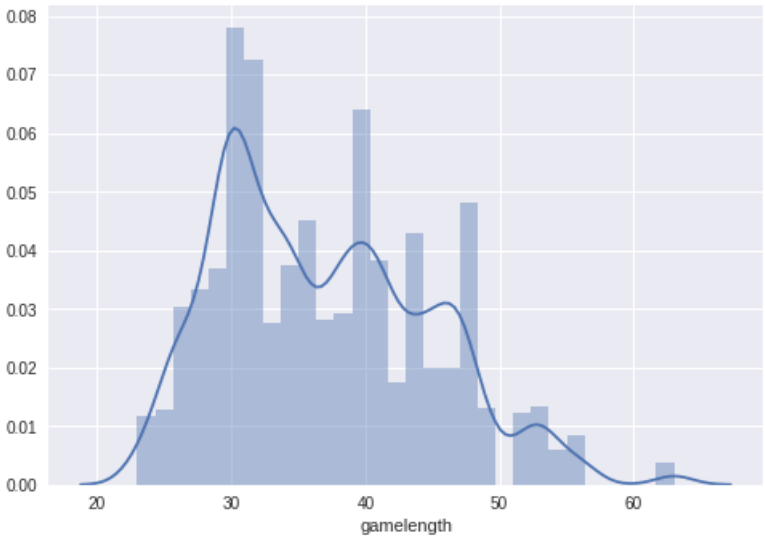
```
('Max game length:', 95)
('Min game length:', 17)
('Mean game length:', 37.99868462590761)
```



In [140]:

```
#Generate average game length of team 'TSM' while playing on blueside (regardless of win/loss)
print('Max game length:', tsmBlueSide.gamelength.max())
print('Min game length:', tsmBlueSide.gamelength.min())
print('Mean game length:', tsmBlueSide.gamelength.mean())
sns.distplot(tsmBlueSide.gamelength)
plt.show()
```

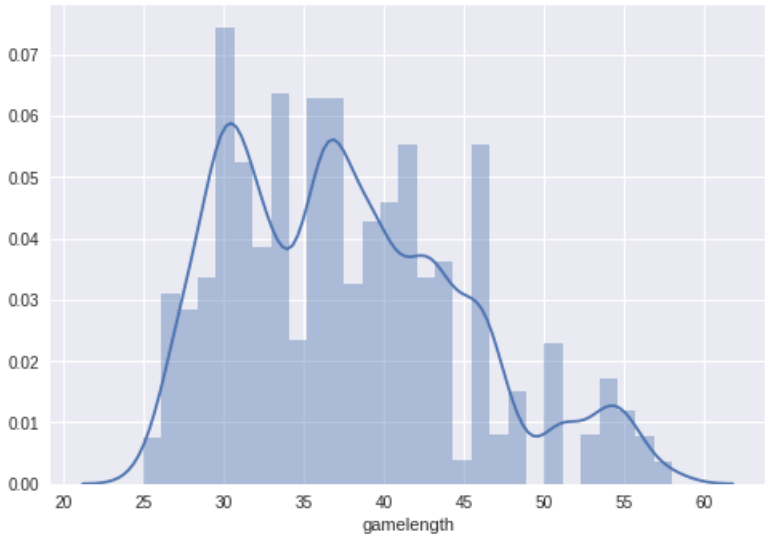
```
('Max game length:', 63)
('Min game length:', 23)
('Mean game length:', 36.93126022913257)
```



In [141]:

```
#Generate average game length of team 'TSM' while playing on redside (regardless of win/loss)
#It seems on average the team play longer games while playing on red side
print('Max game length:', tsmRedSide.gamelength.max())
print('Min game length:', tsmRedSide.gamelength.min())
print('Mean game length:', tsmRedSide.gamelength.mean())
sns.distplot(tsmRedSide.gamelength)
plt.show()
```

```
('Max game length:', 58)
('Min game length:', 25)
('Mean game length:', 37.724137931034484)
```



In [142]:

```
#See if blue side really does win more games
print("Number of Blue Wins", df.bResult.sum())
print("Number of Red Wins", df.rResult.sum())
```

```
('Number of Blue Wins', 203140L)
('Number of Red Wins', 176980L)
```

In [143]:

```
#Create a seperate dataframe for win comparison
team_comparison = pd.DataFrame(data={'asRed': df.redTeamTag.value_counts(),
                                     'asRedWins': df.groupby('redTeamTag').rResult.sum(),
                                     'asBlue': df.blueTeamTag.value_counts(),
                                     'asBlueWins': df.groupby('blueTeamTag').bResult.sum()})
```

In [144]:

```
#Calculate total games and win percentage for each side
team_comparison['totalGames'] = team_comparison.asBlue + team_comparison.asRed
team_comparison['blueWinPct'] = team_comparison.asBlueWins / team_comparison.asBlue
team_comparison['redWinPct'] = team_comparison.asRedWins / team_comparison.asRed
team_comparison.head()
```

Out[144]:

	asBlue	asBlueWins	asRed	asRedWins	totalGames	blueWinPct	redWinPct
100	102.0	102.0	46.0	0.0	148.0	1.000000	0.000000
17A	208.0	68.0	208.0	70.0	416.0	0.326923	0.336538
7h	1588.0	910.0	1778.0	670.0	3366.0	0.573048	0.376828
A	1030.0	382.0	1036.0	442.0	2066.0	0.370874	0.426641
AE	2170.0	902.0	2004.0	594.0	4174.0	0.415668	0.296407

In [145]:

```
#It seems blue side does indeed win more frequently on average
blueWin = team_comparison[team_comparison.redWinPct < team_comparison.blueWinPct]
redWin = team_comparison[team_comparison.redWinPct > team_comparison.blueWinPct]
```

```
plt.plot(blueWin.redWinPct, blueWin.blueWinPct, 'b.')
plt.plot(redWin.redWinPct, redWin.blueWinPct, 'r.')
plt.xlabel('Red Win Percentage')
plt.ylabel('Blue Win Percentage')
plt.show()
```

In [146]:

```
#Grab gold dataset from MongoDB
result = client.test.gold.find()
gold = pd.DataFrame(list(result))
gold.head(3)
```

Out[146]:

	Address	Type	_id	min_1	min_10	min_11	min_12	min_13	mi
0	http://matchhistory.na.leagueoflegends.com/en/...	goldblueSupport	5ccc76f7e88b100c0a86b6fb	500	2014	2193	2334	2493	27
1	http://matchhistory.na.leagueoflegends.com/en/...	goldblueSupport	5ccc76f7e88b100c0a86b6fc	500	1817	1955	2222	2388	26
2	http://matchhistory.na.leagueoflegends.com/en/...	goldblueSupport	5ccc76f7e88b100c0a86b6fd	500	1783	2037	2186	2331	25

3 rows × 98 columns

In [147]:

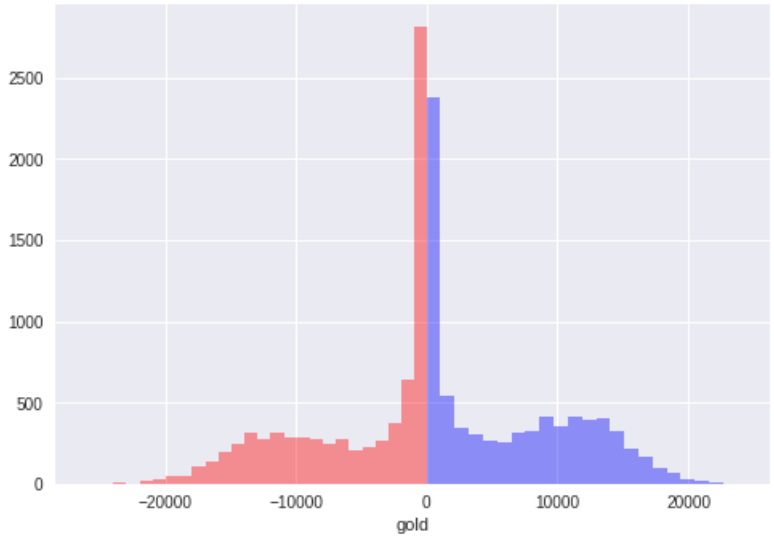
```
#Function to melt gold per minute values
def melt_gold(data):
    # Create the minute columns in array form.
    minutes = ['min_' + str(x) for x in range(1, 82)]
    # Melt the columns.
    melted = pd.melt(data, id_vars=['Address'], value_vars=minutes,
                     var_name='minute', value_name='gold')
    # Convert the minutes to an integer.
    melted.minute = melted.minute.str.strip('min_').astype(int)
    # Remove rows where gold is  NA.
    melted = melted[melted.gold.notnull()]
    return melted

gold_diff = gold[gold.Type == 'golddiff']
gold_diff = melt_gold(gold_diff)
gold_diff.head()
```

Out[147]:

	Address	minute	gold
0	http://matchhistory.na.leagueoflegends.com/en/...	1	0.0
1	http://matchhistory.na.leagueoflegends.com/en/...	1	0.0
2	http://matchhistory.na.leagueoflegends.com/en/...	1	0.0
3	http://matchhistory.na.leagueoflegends.com/en/...	1	0.0
4	http://matchhistory.na.leagueoflegends.com/en/...	1	40.0

In [148]:

```
#Show gold difference as games progress for each side on average
sns.distplot(gold_diff.groupby('Address').gold.max(), color='blue', kde=False)
sns.distplot(gold_diff.groupby('Address').gold.min(), color='red', kde=False)
plt.show()
```

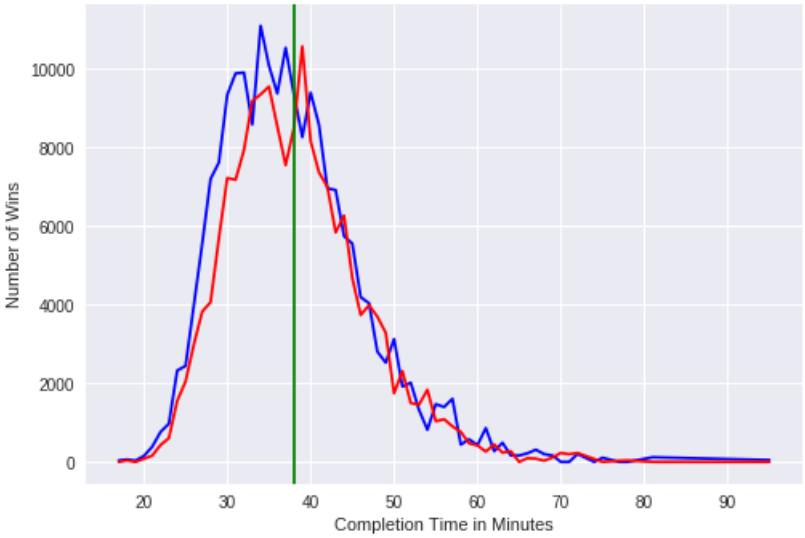


In [149]:

```
#Create a seperate dataframe for visualising game completion on average with each side
color_minute_comparison = pd.DataFrame(data={'blueWins': lol.groupby('gamelength').bResult.sum(),
                                             'redWins': lol.groupby('gamelength').rResult.sum()})
```

In [150]:

```
#Line graph showing average game completition on each side
plt.plot(color_minute_comparison.index, color_minute_comparison.blueWins, 'b-')
plt.plot(color_minute_comparison.index, color_minute_comparison.redWins, 'r-')
plt.axvline(x=lol.gamelength.mean(), color='green')
plt.xlabel('Completion Time in Minutes')
plt.ylabel('Number of Wins')
plt.show()
```



In [151]:

```
#Reassign to a specific team 'TSM'
color_minute_comparison = pd.DataFrame(data={'blueWins': tsmBlueSide.groupby('gamelength').bResult.sum(),
                                             'redWins': tsmRedSide.groupby('gamelength').rResult.sum()})
```

In [152]:

```
#Highlights the team 'TSM' winning games
#Shows they seem to win games on blue side much quicker on average than red side
```

```
plt.plot(color_minute_comparison.index, color_minute_comparison.blueWins, 'b-')
plt.plot(color_minute_comparison.index, color_minute_comparison.redWins, 'r-')
plt.axvline(x=lol.gamelength.mean(), color='green')
plt.xlabel('Completion Time in Minutes')
plt.ylabel('Number of Wins')
plt.show()
```