BEHAVIORAL PATTERN RECOGNITION OF MULTIPLAYER ONLINE ROLE-PLAYING GAME PLAYERS USING BIG DATA ANALYTICS AND MACHINE LEARNING

INTERIM REPORT

TEAM MEMBERS:

- SULEKHA ALOORRAVI
- DEEPA VENUGOPAL
- NIHARIKA THANAVARAPU
- LAKSHMIPRIYA M

MENTOR:

DR. NARAYANA D

CONTENTS

I. DOMAIN AND CONTEXT

1. Domain

A massively multiplayer online game (more commonly, MMO) is an online game which is capable of supporting large numbers of players, typically from hundreds to thousands, simultaneously from around the world.

These games can be found for most network-capable platforms, including the personal computer, video game console, or smartphones and other mobile devices. MMOs can enable players to cooperate and compete with each other on a large scale, and sometimes to interact meaningfully with people around the world.

2. Industry worth

The UK MMO-market is worth £195 million in 2009 compared to the £165 million and £145 million spent by German and French online gamers. The US gamers spend more, however, spending about $3.8 billion overall on MMO games. $1.8 billion of that money is spent on monthly subscription fees. The money spent averages out to $15.10 between both subscription and free-to-play MMO gamers. The study published by "*Today's Gamers MMO Focus Report*" also found that 46% of 46 million players in the US pay real money to play MMO games.

3. Context of this Project

It is challenging to develop the database engines that are needed to run a successful MMOG with millions of players. Understanding the behavior of players using their activity data is more important for these game developers to come up with better strategies in game development.

The variety, volume, velocity, value and veracity (Big Data 5Vs) of data that is involved in these Gaming environments exceed the limits of analysis and manipulation of conventional tools, therefore, Big Data platforms are required to handle and interpret this data.

Great volumes of data are generated all the time in these environments. Each interaction made by a player creates data that are transferred and stored, and if properly analyzed, can contain valuable information. This information can be vital for the continuity and improvement of a game. Patterns can be detected from these data and even predictive analysis can be made to foresee the actions and intentions of the players inside the game.

4. Objective

Objective of this Project is to perform analytics on one such Big Data Gaming Environment and the results would help game developers in:

- Optimizing user experience
- Improving revenue
- Raise the level of control over the environment

II. PROBLEM STATEMENT

1. Perform exploratory analysis

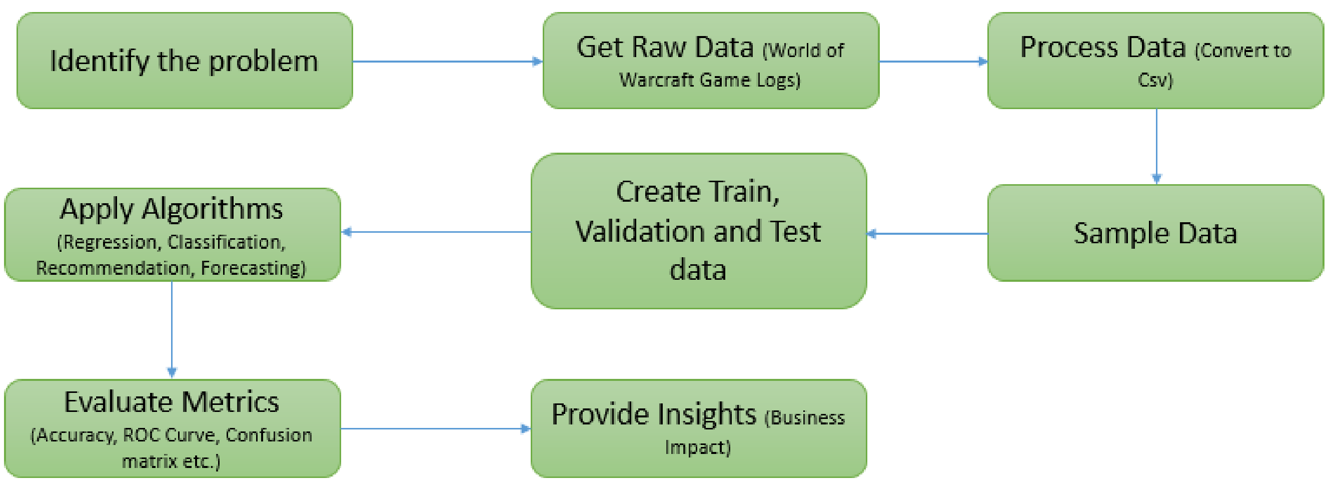
1.1 To cluster players into different groups based on features in dataset
1.2 To analyze and visualize timeline patterns of players by different groups and parameters
1.3 To create heat map based on the gaming zones
1.4 To visualize patterns based on Guilds they belong to

2. Perform Predictive Analytics by applying Machine Learning Models

2.1 Forecast the number of players expected in future time point
2.2 Predict player churning
2.3 Recommend guilds to players for effective gaming

III. PROPOSED SOLUTION

Following workflow will be followed to solve the identified problems in this Project:



IV. EVALUATION METRICS

Problem Statement	Evaluation Metrics	Definition	Formula
2.1 Forecast the number of players expected in future time point	Root Mean Square Error (RMSE)	The standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.	$RMSE = \sqrt{(f - o)^2}$ f = forecasts (expected values or unknown results), o = observed values (known results).
2.2 Predict player churning	Receiver Operating Characteristic (ROC) Curve and Area under	Plot of the true positive rate against the false positive rate	sensitivity vs (1 – specificity)
2.3 Recommend guilds to players for effective gaming	Mean average precision (MAP)	Mean average precision is an extension of average precision where we take average of all AP's to get the MAP.	AP@k is: sum k=1:x of (precision at k * change in recall at k)

V. EXPLORATORY DATA ANALYSIS

We have chosen an online game named "World of Warcraft" which is most suitable for this Project.

A large and scalable dataset with 3 years of player logs are released by Blizzard Entertainment for research purposes. We are using this dataset of our Project.

Data set Summary	
Attribute	**Value**
Data duration (in days)	1107
Sampling Rate per day	124
No. of Samples	138084
No. of Records (rows)	36,513,647
No. of Values (Data points)	438,163,764
Size of data (in GB)	3.4
Dataset Type	Logs
Format	Text Files
No. of Folders	1095

Field Description		
Field	**Description**	**Data Type**
Query Time	Date and time when logs were generated	integer
Query Seq. #	Sequence of queries	integer
Avatar ID	Unique id for each user	integer
Guild	Group id of the player	integer
Level	Game level of the player	integer
Race	Blood Elf, Orc, Tauren, Troll, Undead	String
Class	Death Knight, Druid, Hunter, Mage, Paladin, Priest, Rogue, Shaman, Warlock, Warrior	String
Zone	One of the 229 Zones in World of WarCraft game	String

1. Parse Wow Logs

```
class wow_parser:
    def parse_logs(self,root_dir,output_file):
        import numpy as np
        import os
        import re
        strings = []
        for root, subdirs, files in os.walk(root_dir):
            for filename in files:
                if filename.endswith(".txt"):
                    file_path = os.path.join(root, filename)
                    with open(file_path) as f:
                        for line in f:
                            if "/" in line:
                                strings.extend(re.findall(r'"(.*?)"', line, re.DOTALL))
        thefile = open(output_file, 'w')
        for item in strings:
            thefile.write("%s\n" % item)
```

```
parse = wow_parser()
```

```
dirpath = "H:\WoWAH"
outputpath = "H:\Output\wowlogs.csv"
parse.parse_logs(root_dir = dirpath, output_file = outputpath)
```

	QueryTime	QuerySeq	AvatarID	Guild	Level	Race	Class	Zone
0	12/31/05 23:59:46	1	0		5	Orc	Warrior	Durotar
1	12/31/05 23:59:46	1	1		9	Orc	Shaman	Durotar
2	12/31/05 23:59:52	2	2		13	Orc	Shaman	Durotar
3	12/31/05 23:59:52	2	3	0	14	Orc	Warrior	Durotar
4	12/31/05 23:59:52	2	4		14	Orc	Shaman	Durotar

2. Clean up incorrect records

Following values are incorrect Warcraft races:
'373族', '547人', '3033','27410', '74622妖'
Let us look at the records which have these incorrect races.

```
df_incorrect_race = df[df['Race'].isin(['373族', '547人', '3033','27410', '74622妖'])]
```

```
df_incorrect_race.AvatarID.unique()
```

```
array([  373,   547,  3033, 27410, 74622], dtype=int64)
```

```
df_incorrect_race.count()
```

```
QueryTime    50085
QuerySeq     50085
AvatarID     50085
Guild        50085
Level        50085
Race         50085
Class        50085
Zone         50085
dtype: int64
```

Following values are incorrect Warcraft classes:
'482', '2400', '3485伊'
Let us look at the records which have these incorrect classes.

```
df_incorrect_class = df[df['Class'].isin(['482', '2400', '3485伊'])]
```

```
df_incorrect_class.AvatarID.unique()
```

```
array([ 482, 2400, 3485], dtype=int64)
```

```
df_incorrect_class.count()
```

```
QueryTime    376
QuerySeq     376
AvatarID     376
Guild        376
Level        376
Race         376
Class        376
Zone         376
dtype: int64
```

```
df_incorrect_zone.Zone.unique()
```

```
array([' 未知', ' 監獄', ' 時光洞穴', ' 達納蘇斯', '8585', '1608峽谷', '2029', '15641',
       '1007城', ' 北方海岸', ' 毒牙沼澤', ' 麥克那爾', '61477', ' 龍骨荒野', '1231崔茲',
       ' Dalaran競技場'], dtype=object)
```

```
df_incorrect_zone.count()
```

```
QueryTime    370298
QuerySeq     370298
AvatarID     370298
Guild        370298
Level        370298
Race         370298
Class        370298
Zone         370298
dtype: int64
```

```
df_incorrect_zone.AvatarID.nunique()
```

```
5441
```

```
(removed_records/total_records)*100
```

```
QueryTime    1.024008
QuerySeq     1.024008
AvatarID     1.024008
Guild        1.024008
Level        1.024008
Race         1.024008
Class        1.024008
Zone         1.024008
dtype: float64
```

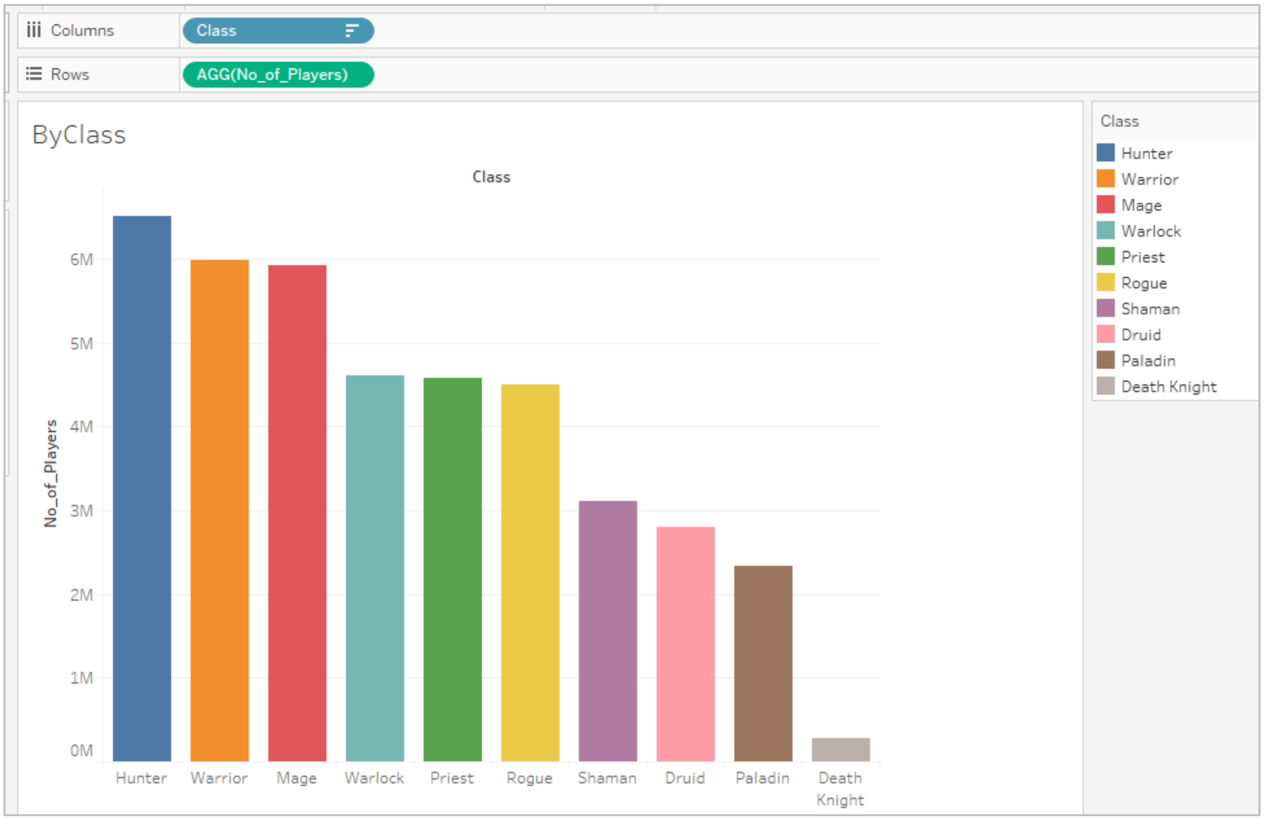
We will have to remove 1% of the records (420491 out of 41063255) to avoid incorrect analysis and inferences from warcraft logs. This data is relatively less compared to the total size of warcraft logs we have gathered.

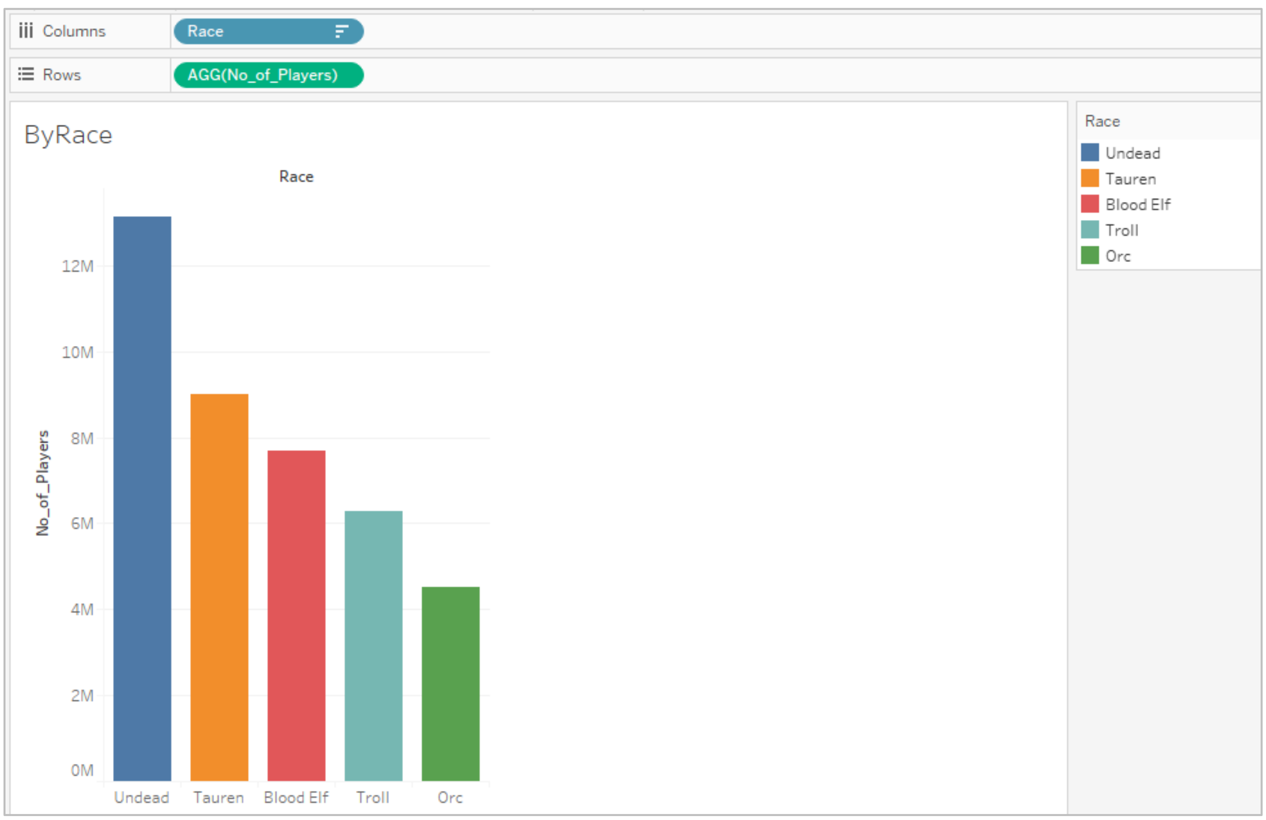
Save the final set of records into a new csv file to be used in further steps.

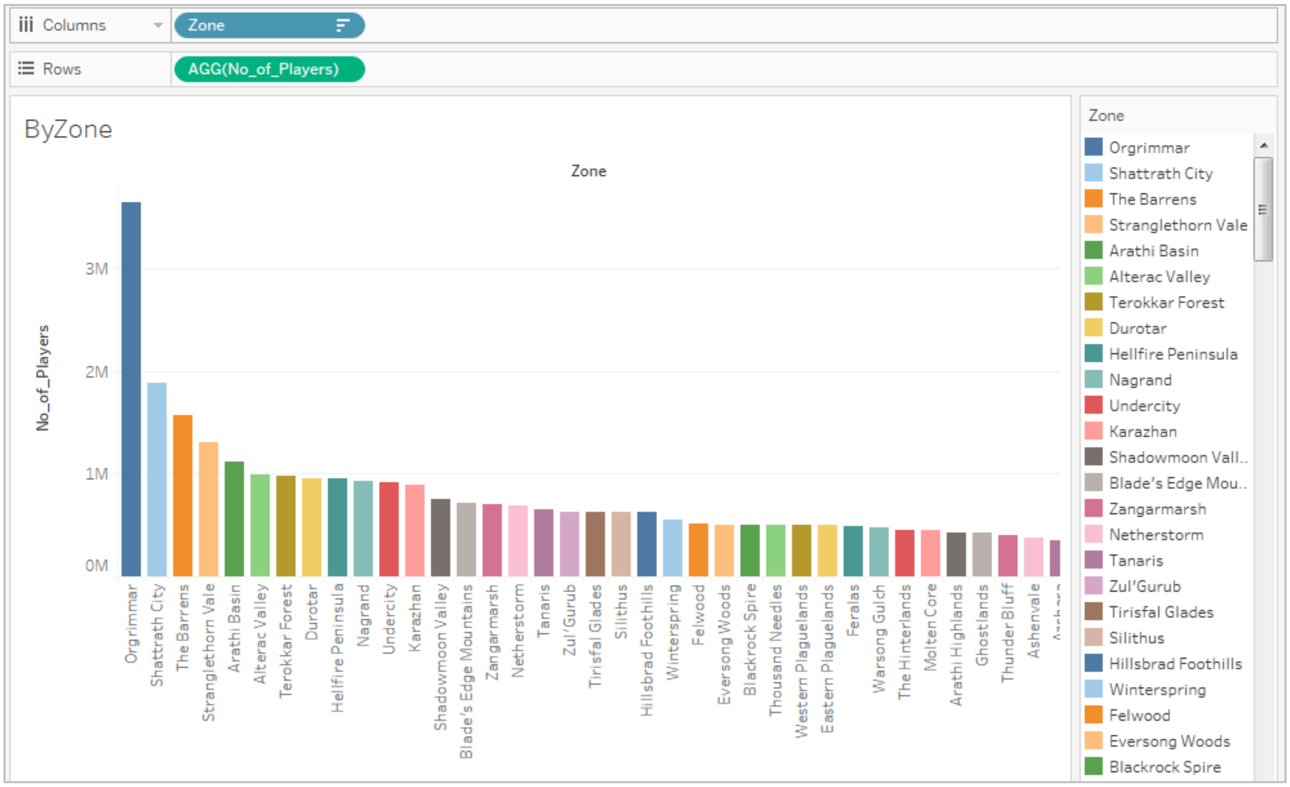
	QueryTime	QuerySeq	AvatarID	Guild	Level	Race	Class	Zone
40692952	01/10/09 05:08:48	56	36893	104	80	Blood Elf	Mage	Dalaran
40692953	01/10/09 05:08:48	56	39532	204	80	Blood Elf	Mage	The Storm Peaks
40692954	01/10/09 05:08:59	58	90033	502	80	Blood Elf	Death Knight	Sholazar Basin
40692955	01/10/09 05:08:59	58	87974	251	80	Blood Elf	Death Knight	Blade's Edge Mountains
40692956	01/10/09 05:08:59	58	86679	459	80	Blood Elf	Death Knight	Shadowmoon Valley

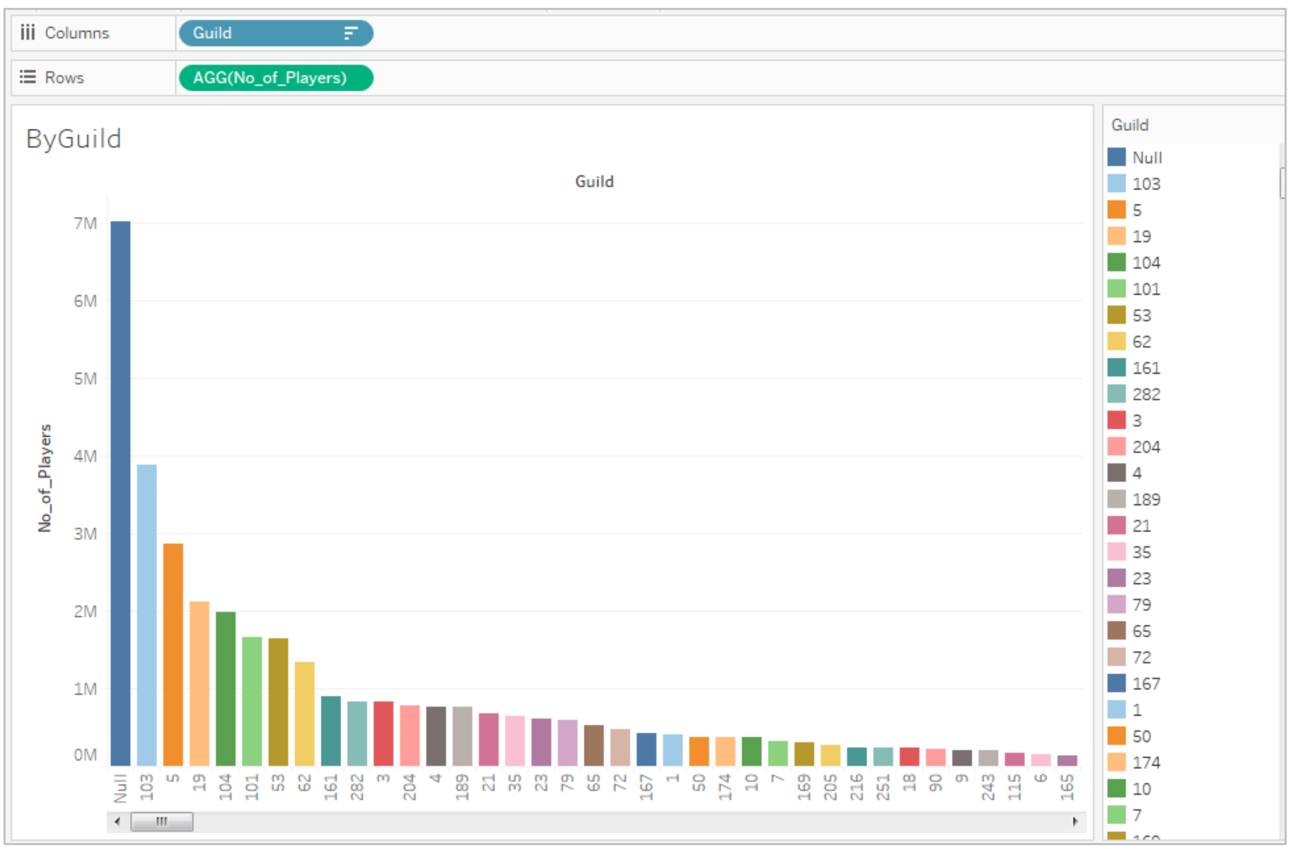
VI. EXPLORATORY VISUALIZATION

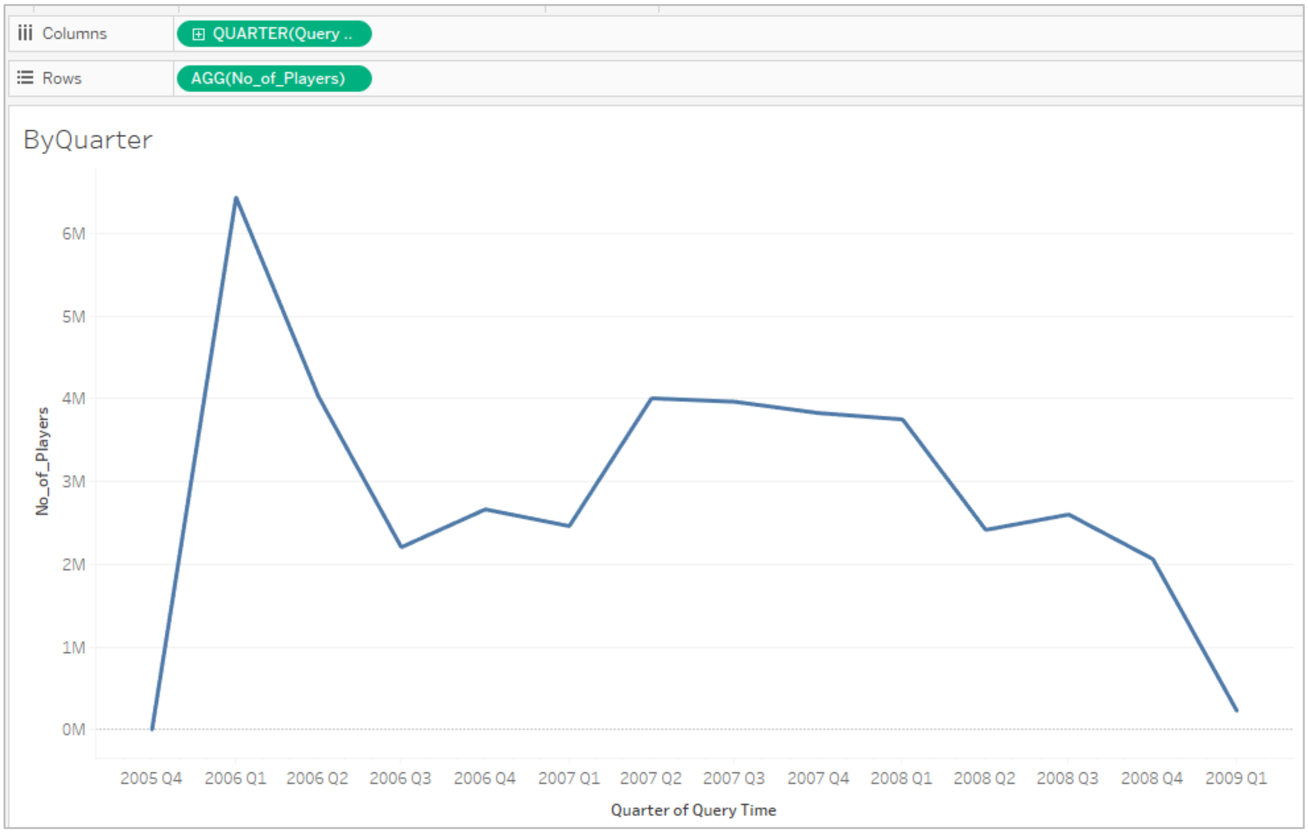
1. Tableau Visualizations



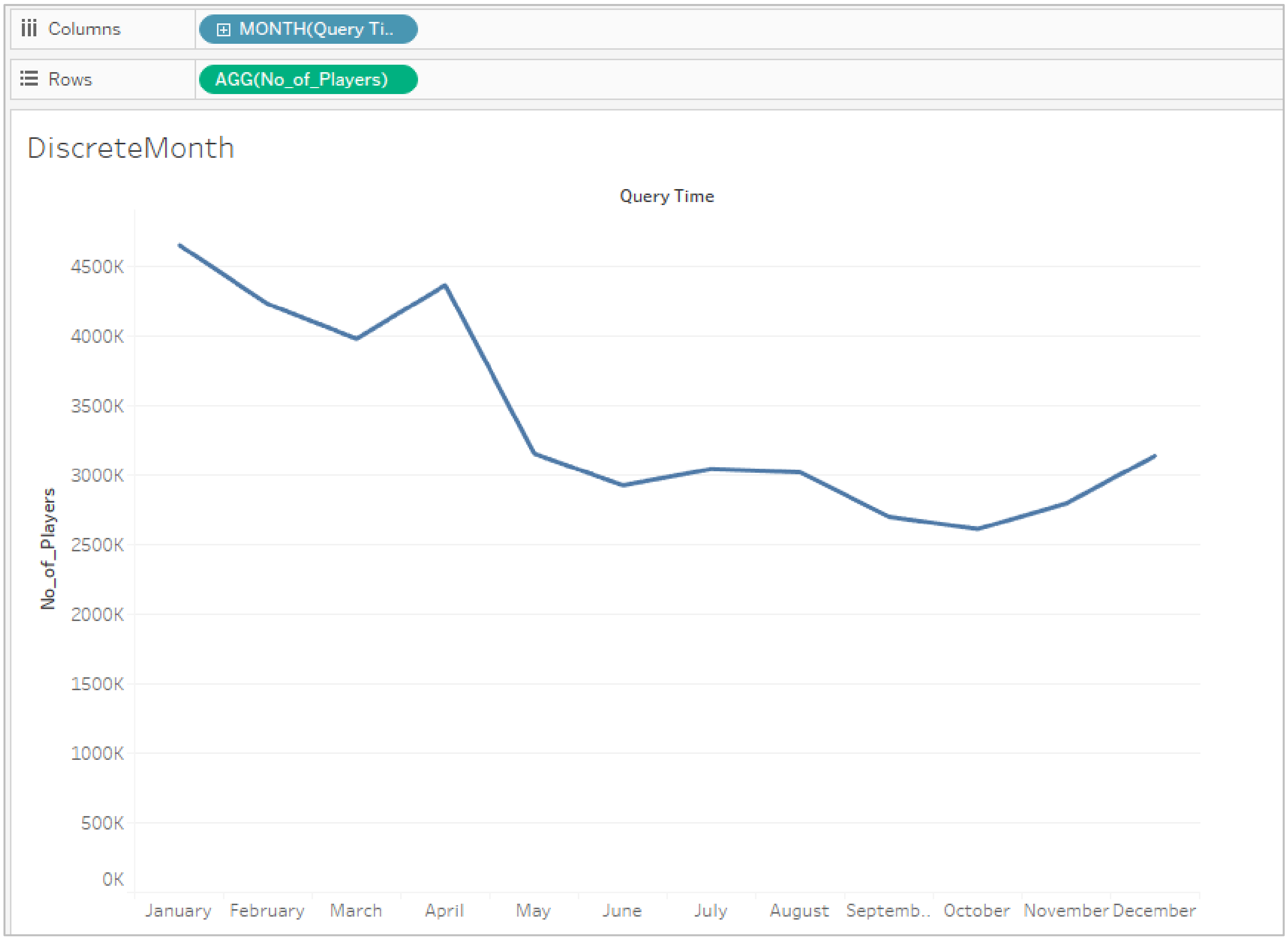


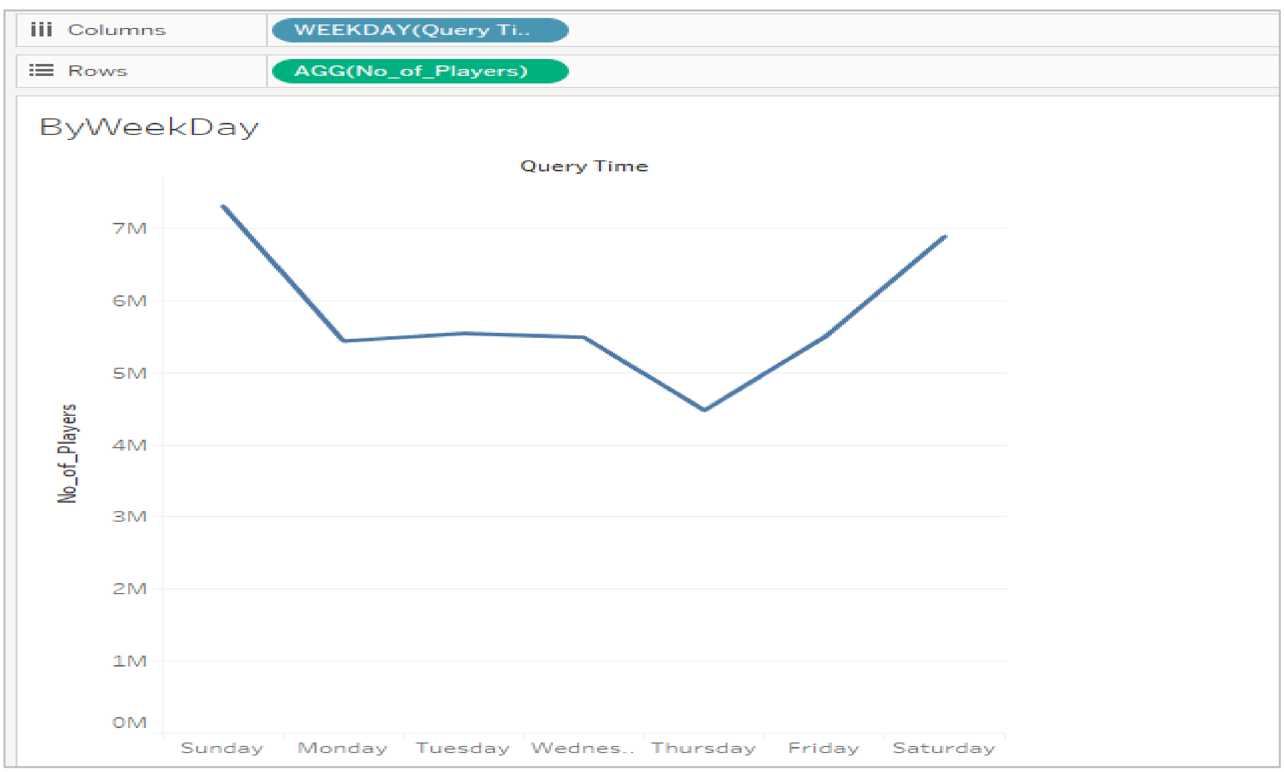

Columns
Zone
Rows
AGG(No_of_Players)
ByZone
Zone
No_of_Players
0M
1M
2M
3M
Orgrimmar
Shattrath City
The Barrens
Stranglethorn Vale
Arathi Basin
Alterac Valley
Terokkar Forest
Durotar
Hellfire Peninsula
Nagrand
Undercity
Karazhan
Shadowmoon Valley
Blade's Edge Mountains
Zangarmarsh
Netherstorm
Tanaris
Zul'Gurub
Tirisfal Glades
Silithus
Hillsbrad Foothills
Winterspring
Felwood
Eversong Woods
Blackrock Spire
Thousand Needles
Western Plaguelands
Eastern Plaguelands
Feralas
Warsong Gulch
The Hinterlands
Molten Core
Arathi Highlands
Ghostlands
Thunder Bluff
Ashenvale
Zone
Orgrimmar
Shattrath City
The Barrens
Stranglethorn Vale
Arathi Basin
Alterac Valley
Terokkar Forest
Durotar
Hellfire Peninsula
Nagrand
Undercity
Karazhan
Shadowmoon Vall..
Blade's Edge Mou..
Zangarmarsh
Netherstorm
Tanaris
Zul'Gurub
Tirisfal Glades
Silithus
Hillsbrad Foothills
Winterspring
Felwood
Eversong Woods
Blackrock Spire

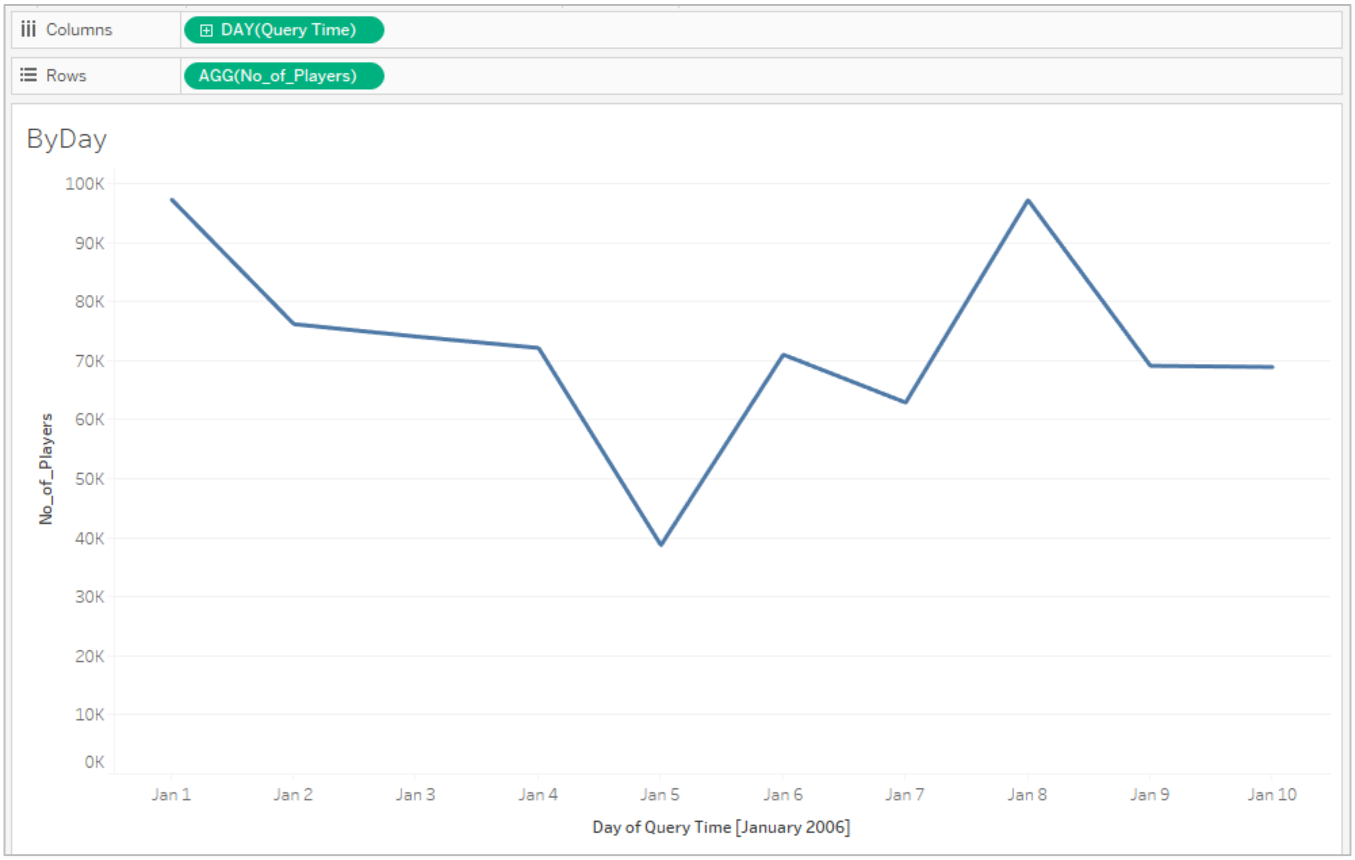

Columns
Guild
Rows
AGG(No_of_Players)
ByGuild
Guild
No_of_Players
0M
1M
2M
3M
4M
5M
6M
7M
Null
103
5
19
104
101
53
62
161
282
3
204
4
189
21
35
23
79
65
72
167
1
50
174
10
7
169
205
216
251
18
90
9
243
115
6
165
Guild
Null
103
5
19
104
101
53
62
161
282
3
204
4
189
21
35
23
79
65
72
167
1
50
174
10
7

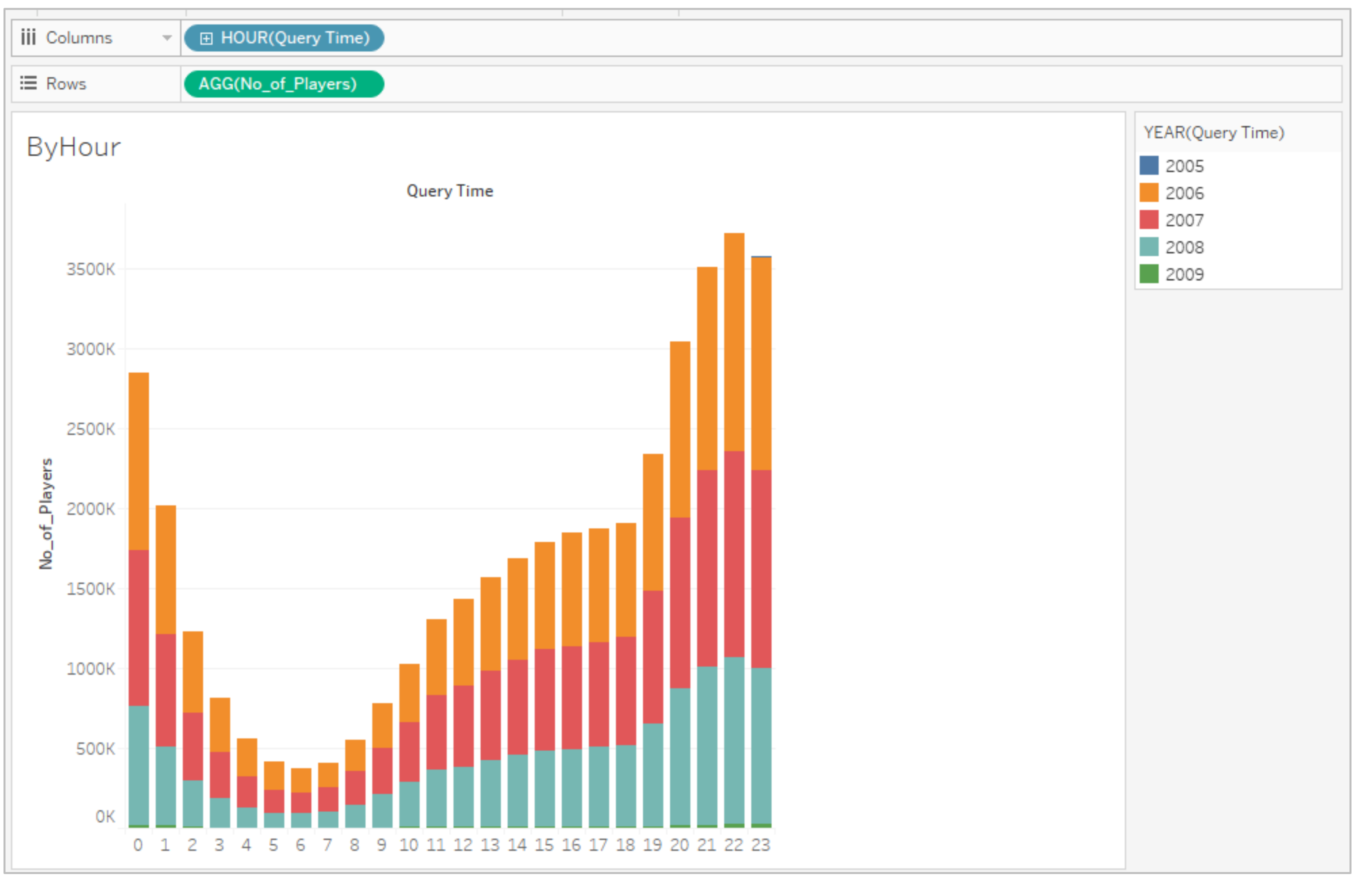

Columns
QUARTER(Query ..
Rows
AGG(No_of_Players)
ByQuarter
No_of_Players
6M
5M
4M
3M
2M
1M
0M
2005 Q4
2006 Q1
2006 Q2
2006 Q3
2006 Q4
2007 Q1
2007 Q2
2007 Q3
2007 Q4
2008 Q1
2008 Q2
2008 Q3
2008 Q4
2009 Q1
Quarter of Query Time


Columns
MONTH(Query Ti..
Rows
AGG(No_of_Players)
ByMonth
No_of_Players
2000K
1500K
1000K
500K
0K
November 2005
May 2006
November 2006
May 2007
November 2007
May 2008
November 2008
Month of Query Time


Columns
MONTH(Query Ti..
Rows
AGG(No_of_Players)
DiscreteMonth
Query Time
No_of_Players
4500K
4000K
3500K
3000K
2500K
2000K
1500K
1000K
500K
0K
January
February
March
April
May
June
July
August
Septemb..
October
November
December


Columns
WEEKDAY(Query Ti..
Rows
AGG(No_of_Players)
ByWeekDay
Query Time
No_of_Players
7M
6M
5M
4M
3M
2M
1M
0M
Sunday
Monday
Tuesday
Wednes..
Thursday
Friday
Saturday


Columns
DAY(Query Time)
Rows
AGG(No_of_Players)
ByDay
No_of_Players
100K
90K
80K
70K
60K
50K
40K
30K
20K
10K
0K
Jan 1
Jan 2
Jan 3
Jan 4
Jan 5
Jan 6
Jan 7
Jan 8
Jan 9
Jan 10
Day of Query Time [January 2006]


Columns
HOUR(Query Time)
Rows
AGG(No_of_Players)
ByHour
Query Time
No_of_Players
3500K
3000K
2500K
2000K
1500K
1000K
500K
0K
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
YEAR(Query Time)
2005
2006
2007
2008
2009

2. Insights from Exploratory Analysis

1. Hunter is the class chosen by most of the players
2. Death Knight is chosen by least number of players
3. Undead is the Race chosen by most of the players
4. Orc is chosen by least number of players
5. Orgrimmar is the Zone chosen by most of the players
6. 7,014,160 players are playing without joining any guilds
7. Within the Data collection period, maximum number of players played world of Warcraft during Q1 2006
8. 2005 and 2009 data cannot be considered for exploratory insights since it covers data of less than a month
9. January is the month with most players and October is with least players every year
10. Sunday and Saturday (Weekends) are the days with more players and Thursday is the day with least players every week
11. 10:00 PM is the most played hour in a day and 6:00 AM is the least played

VII. SUMMARY OF INITIAL FINDINGS

1. Models attempted

1. NEURAL NETWORKS FOR PATTERN RECOGNITION - EXPLORATORY

In the process of solving problem statements, we first came up with identifying the game play patterns of players from the dataset.

For this process, we have converted the dataset into numerical data that can be provided as input to Pattern recognizing neural networks.

This model would still be part of exploratory data analysis.

Step 1: Read data into a dataframe

```
df2 = pd.read_csv("newlogs.csv", usecols = ['Guild','Level','Race','Class'])
```

Step 2: Group records by Guild, Level, Race and Class and count the number of records following each unique combination

```
df2 = df2.groupby(['Guild','Level','Race','Class']).size().reset_index()
```

```
df2.rename(columns = {0: 'PatternCount'},  inplace=True)
```

```
df2.head(5)
```

	Guild	Level	Race	Class	PatternCount
0		1	Blood Elf	Hunter	3834
1		1	Blood Elf	Mage	6621
2		1	Blood Elf	Paladin	10121
3		1	Blood Elf	Priest	4293
4		1	Blood Elf	Rogue	5787

Step 3: Convert string values on the above columns into their encoded numeric value

```
def addguildflag(x):
    if x== ' ':
        return 0
    else:
        return 1
```

```
df2['GuildFlag'] = df2.apply(lambda col: addguildflag(col['Guild']), axis = 1)
```

```
def addlevelflag(x):
    if x>=1 and x<=23:
        return 2
    elif x>=24 and x<=47:
        return 3
    else:
        return 4
```

```
df2['LevelFlag'] = df2.apply(lambda col: addlevelflag(col['Level']), axis = 1)
```

```
def addraceflag(x):
    if x == ' Blood Elf':
        return 5
    elif x == ' Orc':
        return 6
    elif x == ' Tauren':
        return 7
    elif x == ' Troll':
        return 8
    elif x == ' Undead':
        return 9
```

```
df2['RaceFlag'] = df2.apply(lambda col: addraceflag(col['Race']), axis = 1)
```

```
def addclassflag(x):
    if x in [' Warrior', ' Hunter', ' Rogue', ' Paladin', ' Death Knight']:
        return 10
    elif x in [' Shaman', ' Warlock', ' Druid', ' Mage', ' Priest']:
        return 11
```

Step 4: Create Frequency Flag by creating a rule for each frequency

```python
a = df3.PatternCount.quantile(0.33)
b = df3.PatternCount.quantile(0.66)
```

```python
def addfrequencyflag(x):
    if x>=1 and x<=a:
        return 18
    elif x>a and x<=b:
        return 19
    else:
        return 20
```

```python
df3 = df2[['GuildFlag','LevelFlag','RaceFlag','ClassFlag','PatternCount']]
```

```python
df3 = df3.groupby(by = ['GuildFlag','LevelFlag','RaceFlag','ClassFlag'])['PatternCount'].sum().reset_index()
```

```python
df3['FrequencyFlag'] = df3.apply(lambda col: addfrequencyflag(col['PatternCount']), axis = 1)
```

Step 5: Create Pattern ID for each unique pattern

```python
df3['PatternID'] = df3.index+1
```

```python
df3
```

	GuildFlag	LevelFlag	RaceFlag	ClassFlag	PatternCount	FrequencyFlag	PatternID
0	0	2	5	10	516572	20	1
1	0	2	5	11	484618	20	2
2	0	2	6	10	257647	19	3
3	0	2	6	11	75869	18	4
4	0	2	7	10	217354	19	5
5	0	2	7	11	293998	19	6
6	0	2	8	10	420970	20	7
7	0	2	8	11	166371	18	8
8	0	2	9	10	235052	19	9
9	0	2	9	11	462959	20	10

Step 6: Identify the number of unique patterns in the dataset

```
import pandas as pd
import numpy as np
#cols = ['GuildFlag','LevelFlag','RaceFlag','ClassFlag','PatternID']
data = pd.read_csv("test1.csv")
data = data.sample(1000)
```

```
data.head(5)
```

	GuildFlag	LevelFlag	RaceFlag	ClassFlag	FrequencyFlag	PatternID
735	1	4	9	11	20	48
8099	1	4	9	11	20	48
31171	1	3	7	10	20	35
32234	1	3	9	10	20	39
44265	0	4	9	11	19	24

Step 7: Normalize the data and create Train & Test data

```
#Normalized
df_norm = data[['GuildFlag','LevelFlag','RaceFlag','ClassFlag']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
target = data[['PatternID']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
target.PatternID.nunique()
```

```
46
```

```
df = pd.concat([df_norm, target], axis=1)
df.sample(n=4)
```

	GuildFlag	LevelFlag	RaceFlag	ClassFlag	PatternID
21861	1.0	1.0	1.000000	1.0	1.000000
32025	1.0	0.5	0.333333	0.0	0.723404
33893	1.0	1.0	0.000000	0.0	0.851064
28537	1.0	1.0	0.333333	0.0	0.893617

```
X = np.array(np.array(df.iloc[:,0:4].values),dtype=int)
X
```

```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 0, 0, 0],
       ...,
       [1, 1, 1, 0],
       [1, 0, 0, 1],
       [1, 1, 1, 0]])
```

```
y = np.array(df.iloc[:,4],dtype=int)[np.newaxis].T
```

Step 8: Provide inputs to Multiple Back Propagation neural networks and predict patterns (Input Layer = 4, Hidden Layer = 7 and Output Layer = 1)

```
class Neural_Network(object):
  def __init__(self):
    #parameters
    self.inputSize = 4
    self.outputSize = 1
    self.hiddenSize = 7

    #weights
    self.W1 = np.random.randn(self.inputSize, self.hiddenSize) # (3x2) weight matrix from input to hidden layer
    self.W2 = np.random.randn(self.hiddenSize, self.outputSize) # (3x1) weight matrix from hidden to output layer

  def forward(self, X):
    #forward propagation through our network
    self.z = np.dot(X, self.W1) # dot product of X (input) and first set of 3x2 weights
    self.z2 = self.sigmoid(self.z) # activation function
    self.z3 = np.dot(self.z2, self.W2) # dot product of hidden layer (z2) and second set of 3x1 weights
    o = self.sigmoid(self.z3) # final activation function
    return o
```

Step 9: Use sigmoid function to activate the neurons

```
def sigmoid(self, s):
  # activation function
  return 1/(1+np.exp(-s))

def sigmoidPrime(self, s):
  #derivative of sigmoid
  return s * (1 - s)

def backward(self, X, y, o):
  # backward propgate through the network
  self.o_error = y - o # error in output
  self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to error

  self.z2_error = self.o_delta.dot(self.W2.T) # z2 error: how much our hidden layer weights contributed to output error
  self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error

  self.W1 += X.T.dot(self.z2_delta) # adjusting first set (input --> hidden) weights
  self.W2 += self.z2.T.dot(self.o_delta) # adjusting second set (hidden --> output) weights

def train (self, X, y):
  o = self.forward(X)
  self.backward(X, y, o)
```

Step 10: Predict Pattern id and Loss on Test Data

```
NN = Neural_Network()
for i in range(100): # trains the NN 1,000 times
  print ("Input: \n" + str(X) )
  print ("Actual Output: \n" + str(y) )
  print ("Predicted Output: \n" + str(NN.forward(X)) )
  print ("Loss: \n" + str(np.mean(np.square(y - NN.forward(X))))) # mean sum squared loss
  print ("\n")
  NN.train(X, y)
```

```
Loss:
0.126
```

The results of the tests with the neural networks shows how it can be trained to recognize groups of players by their characteristics and learn the unique patterns of Guild, Level, Race, Class combinations of these groups in the game.

2. TIME SERIES FORECASTING TO PREDICT NUMBER OF PLAYERS

Another algorithm applied so far on this dataset is ARIMA model **Auto-Regressive Integrated Moving Averages** to predict number of players expected on a future time series

Step1: Load and handle Time series data in Python

```
df1.head()
```

	QueryTime	AvatarID
0	12/31/05 23:59:46	0
1	12/31/05 23:59:46	1
2	12/31/05 23:59:52	2
3	12/31/05 23:59:52	3
4	12/31/05 23:59:52	4

```
df2 = df1.groupby(['QueryTime']).size().reset_index()

df2.rename(columns = {0: 'PlayersCount'},  inplace=True)

dateparse = lambda dates: pd.to_datetime(dates, dayfirst=True)

#Strip until date with .str[:-9]
df2['Date'] = df2['QueryTime'].str[:-9]

df2 = df2[['Date','PlayersCount']]

df3 = df2.groupby(['Date']).sum().reset_index()
```

```
dateparse = lambda dates: pd.to_datetime(dates, dayfirst=True)

df3['DateTime'] = df3.apply(lambda col: dateparse(col['Date']), axis = 1)

data = df3[['DateTime', 'PlayersCount']].set_index(['DateTime'])

data.index

DatetimeIndex(['2005-12-31', '2006-01-01', '2006-01-02', '2006-01-03',
               '2006-01-04', '2006-01-05', '2006-01-06', '2006-01-07',
               '2006-01-08', '2006-01-09',
               ...
               '2009-01-01', '2009-02-01', '2009-03-01', '2009-04-01',
               '2009-05-01', '2009-06-01', '2009-07-01', '2009-08-01',
               '2009-09-01', '2009-10-01'],
              dtype='datetime64[ns]', name=u'DateTime', length=1083, freq=None)

data.sort_index(inplace = True)
```

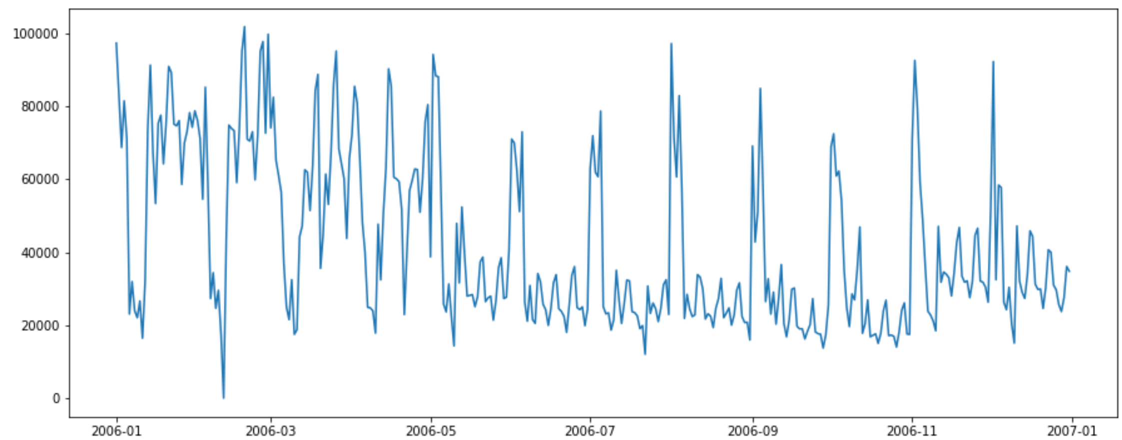
```
ts = data['PlayersCount']
ts.head(10)
```

```
DateTime
2005-12-31       26
2006-01-01    97322
2006-01-02    82668
2006-01-03    68720
2006-01-04    81508
2006-01-05    71252
2006-01-06    23068
2006-01-07    31981
2006-01-08    23990
2006-01-09    22086
Name: PlayersCount, dtype: int64
```

Step 2: Check stationarity of a Time series

```
plt.plot(ts['2006'])
```

```
[<matplotlib.lines.Line2D at 0x510cdb70>]
```



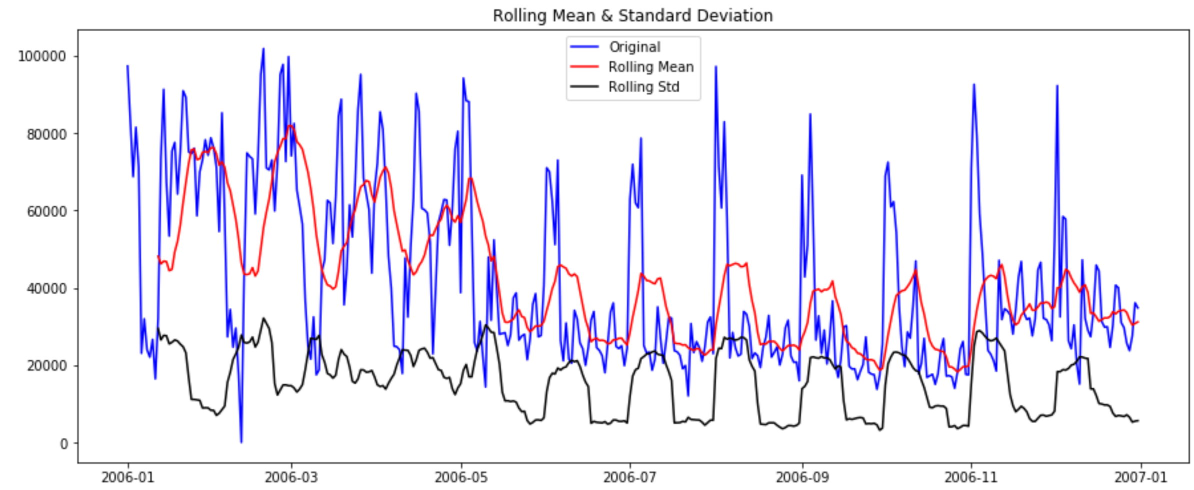

```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(window=12,center=False).mean()
    rolstd = timeseries.rolling(window=12,center=False).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

```
test_stationarity(ts['2006'])
```



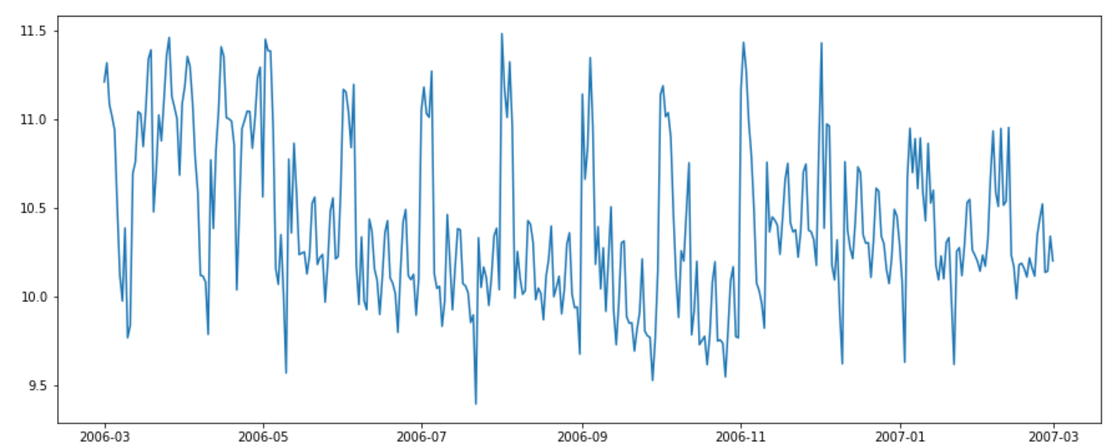

```
Results of Dickey-Fuller Test:
Test Statistic                  -4.586999
p-value                          0.000136
#Lags Used                       6.000000
Number of Observations Used    357.000000
Critical Value (1%)             -3.448801
Critical Value (5%)             -2.869670
Critical Value (10%)            -2.571101
dtype: float64
```

Step 3: Make time series stationary

```
Estimating & Eliminating Trend
```

```
#ts_log = np.log(ts['2006'])
ts_log = np.log(ts['2006-03-01':'2007-03-01'])
plt.plot(ts_log)
```

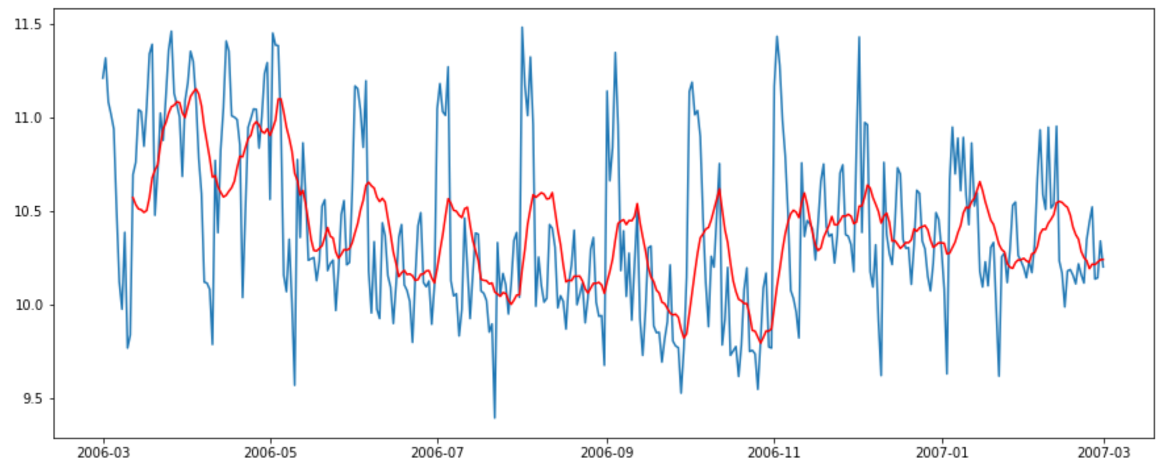
```
[<matplotlib.lines.Line2D at 0x5228e908>]
```



Moving average

```
moving_avg = ts_log.rolling(window=12,center=False).mean()
plt.plot(ts_log)
plt.plot(moving_avg, color='red')
```

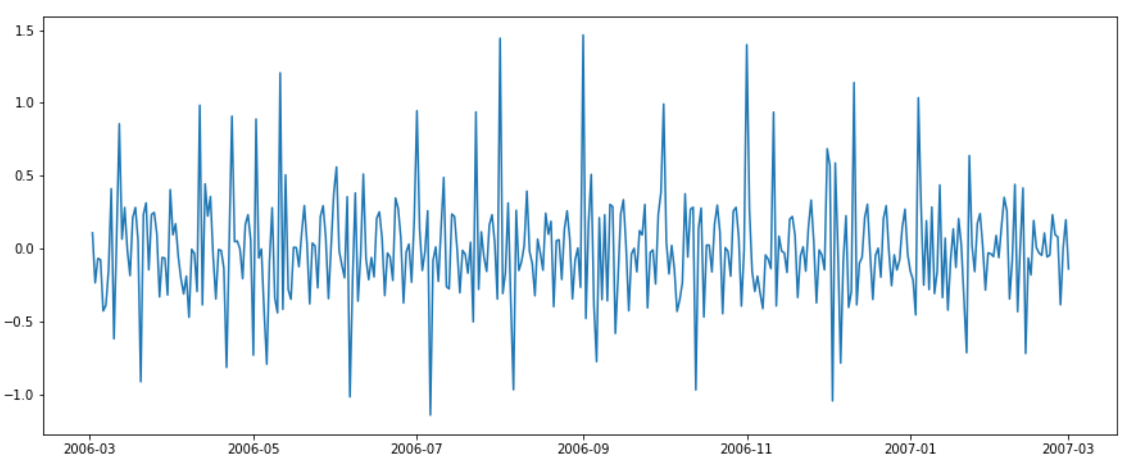
```
[<matplotlib.lines.Line2D at 0x52852550>]
```



Eliminating Trend and Seasonality

Differencing – taking the differece with a particular time lag

```
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)
```

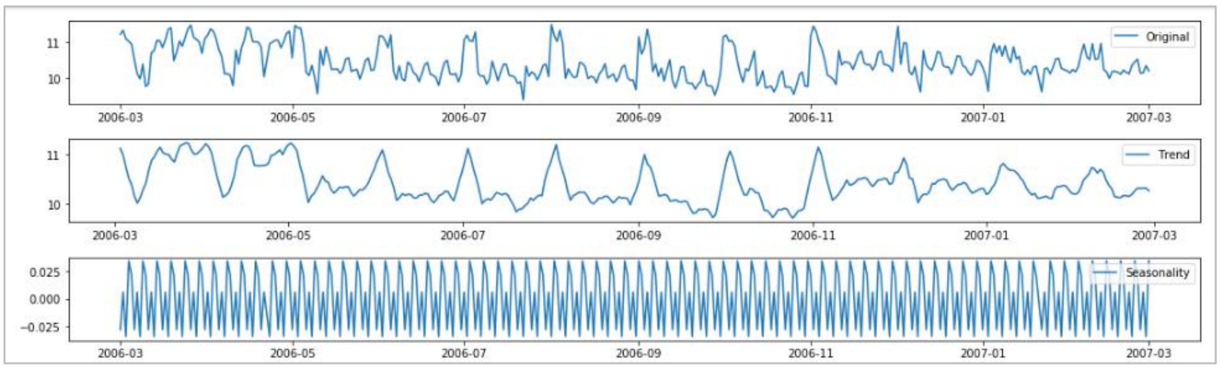


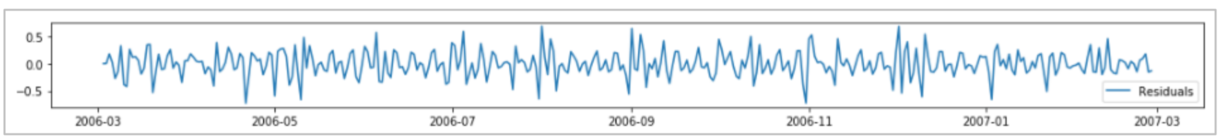

```
Decomposition – modeling both trend and seasonality and removing them from the model
```

```python
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_log, freq = 5)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



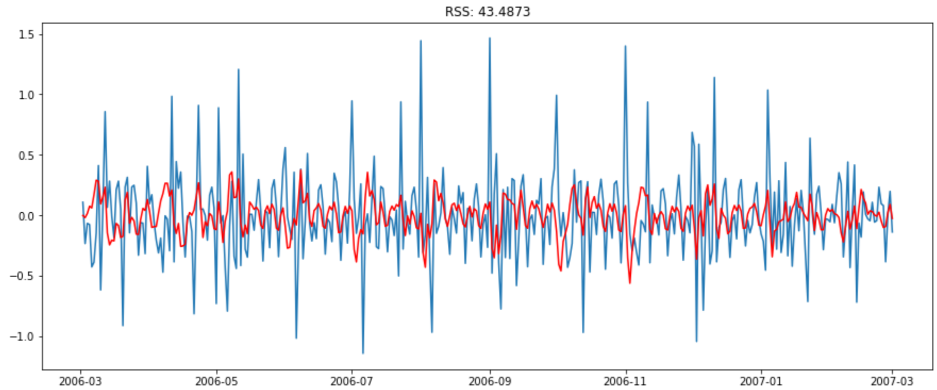


Step 4: Forecast on Time series

Applying ARIMA model

```python
model = ARIMA(ts_log, order=(0, 1, 2))
results_ARIMA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```

```
Text(0.5,1,'RSS: 43.4873')
```




```python
predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
print (predictions_ARIMA_diff.head())
```

```
DateTime
2006-03-02   -0.002225
2006-03-03   -0.019941
2006-03-04    0.014380
2006-03-05    0.074339
2006-03-06    0.060022
dtype: float64
```

```python
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print (predictions_ARIMA_diff_cumsum.head())
```

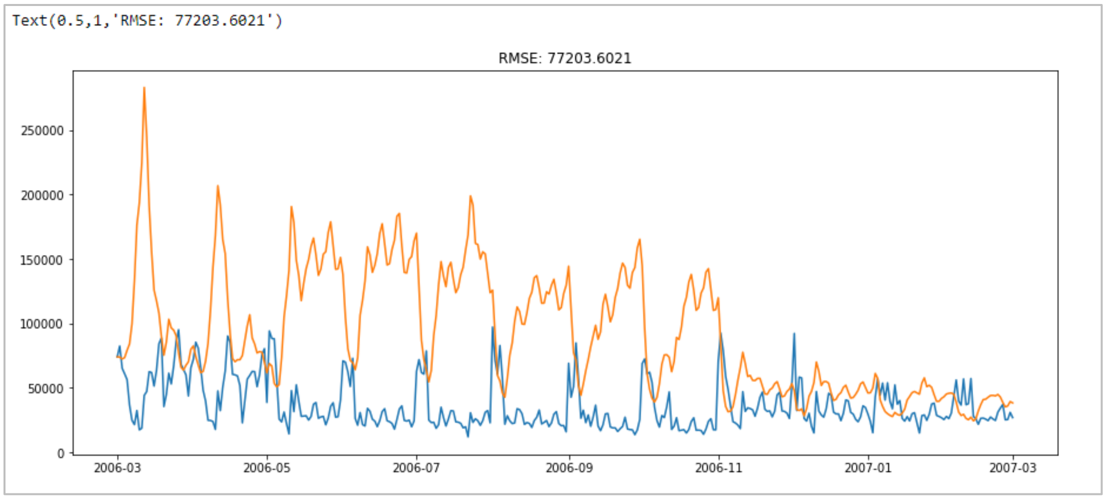
```
DateTime
2006-03-02   -0.002225
2006-03-03   -0.022166
2006-03-04   -0.007786
2006-03-05    0.066553
2006-03-06    0.126575
dtype: float64
```

```python
predictions_ARIMA_log = pd.Series(ts_log.ix[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
predictions_ARIMA_log.head()
```

```
DateTime
2006-03-01    11.213063
2006-03-02    11.210838
2006-03-03    11.190897
2006-03-04    11.205277
2006-03-05    11.279616
dtype: float64
```

```python
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(ts['2006-03-01':'2007-03-01'])
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-ts['2006-03-01':'2007-03-01'])**2)/len(ts['2006-03-01':'2007-03-01'])))
```

```
Text(0.5,1,'RMSE: 77203.6021')
```



VIII. CHALLENGES

- Data size of 438,163,764 data points is too time consuming to process in Python
- Data set had to be sampled into smaller chunks to apply models
- Handling data in Tableau for exploratory data analysis is also more time consuming but complete set of data needs to be used for Exploratory analysis and sampling will not provide appropriate insights
- Neural networks or Matrix Factorization is unable to handle all data points
- In Time series forecasting – ARIMA Model: RMSE value is very high and the model needs to be revisited.

IX. NEXT STEPS

Problem Statement	Next Steps	Models to be used
2.1 Forecast the number of players expected in future time point	To revisit ARIMA model and data for improving Accuracy and decreasing RMSE value	ARIMA / Neural Networks / Random Forest
2.2 Predict player churning	Features to be encoded to become suitable input for Churn prediction models	Decision Tree Classifier
2.3 Recommend guilds to players for effective gaming	Convert data into a format suitable for building a Recommendation Engine	Matrix Factorization, KNN
Data Visualization	Construct more visualizations and create story board	
Big Data Environment	Move data to Hadoop Environment and apply models on the same	

X. References

i. **Big Data Analytics Using Neural networks**
Author: *Chetan Sharma*
Master's Thesis: *San Jose State University*

ii. **Game Analytics - Maximizing the Value of Player Data**
Authors: *Magy Seif El-Nasr, Anders Drachen and Alessandro Canossa*
Publisher: *Springer*

iii. **Big Data Analytics in Cloud Gaming**
Authors: *Victor Perazzolo Barros and Pollyana Notargiacomo*
Paper: 2016 IEEE International Conference on Big Data

iv. **Setting Players' Behaviors in World of Warcraft through Semi-Supervised Learning**
Authors: *Marcelo Souza Nery, Victor do Nascimento Silva, Roque Anderson S. Teixeira and Adriano Alonso Veloso*