DataClean

December 1, 2019

1 Data Cleanning

```
In [1]: import numpy as np
        import pandas as pd
        import sklearn
```

```
In [2]: data = pd.read_csv("LeagueofLegends.csv")
        data.shape
```

```
Out[2]: (7620, 57)
```

```
In [3]: data.head()
```

```
Out[3]:   League  Year  Season    Type blueTeamTag  bResult  rResult redTeamTag  \
        0  NALCS  2015  Spring  Season         TSM        1        0         C9
        1  NALCS  2015  Spring  Season         CST        0        1        DIG
        2  NALCS  2015  Spring  Season         WFX        1        0         GV
        3  NALCS  2015  Spring  Season         TIP        0        1         TL
        4  NALCS  2015  Spring  Season         CLG        1        0         T8

           gamelength                                          golddiff  ...  \
        0          40  [0, 0, -14, -65, -268, -431, -488, -789, -494,...  ...
        1          38  [0, 0, -26, -18, 147, 237, -152, 18, 88, -242,...  ...
        2          40  [0, 0, 10, -60, 34, 37, 589, 1064, 1258, 913, ...  ...
        3          41  [0, 0, -15, 25, 228, -6, -243, 175, -346, 16, ...  ...
        4          35  [40, 40, 44, -36, 113, 158, -121, -191, 23, 20...  ...

          redMiddleChamp                                     goldredMiddle  \
        0           Fizz  [475, 475, 552, 842, 1178, 1378, 1635, 1949, 2...
        1           Azir  [475, 475, 552, 786, 1097, 1389, 1660, 1955, 2...
        2           Azir  [475, 475, 533, 801, 1006, 1233, 1385, 1720, 1...
        3           Lulu  [475, 475, 532, 771, 1046, 1288, 1534, 1776, 2...
        4           Lulu  [475, 475, 532, 807, 1042, 1338, 1646, 1951, 2...

                redADC redADCChamp  \
        0       Sneaky       Sivir
        1       CoreJJ       Corki
        2          Cop       Corki
```

```
3         KEITH      KogMaw
4  Maplestreet8       Corki

                                              goldredADC   redSupport  \
0  [475, 475, 532, 762, 1097, 1469, 1726, 2112, 2...  LemonNation
1  [475, 475, 532, 868, 1220, 1445, 1732, 1979, 2...      KiWiKiD
2  [475, 475, 533, 781, 1085, 1398, 1782, 1957, 2...   BunnyFuFuu
3  [475, 475, 532, 766, 1161, 1438, 1776, 1936, 2...      Xpecial
4  [475, 475, 532, 792, 1187, 1488, 1832, 2136, 2...        Dodo8

  redSupportChamp                                    goldredSupport  \
0          Thresh  [515, 515, 577, 722, 911, 1042, 1194, 1370, 14...
1           Annie  [515, 515, 583, 752, 900, 1066, 1236, 1417, 15...
2           Janna  [515, 515, 584, 721, 858, 1002, 1168, 1303, 14...
3           Janna  [515, 515, 583, 721, 870, 1059, 1205, 1342, 15...
4           Annie  [475, 475, 538, 671, 817, 948, 1104, 1240, 136...

                             redBans  \
0  ['Tristana', 'Leblanc', 'Nidalee']
1      ['RekSai', 'Janna', 'Leblanc']
2        ['Leblanc', 'Zed', 'RekSai']
3      ['RekSai', 'Rumble', 'LeeSin']
4       ['Rumble', 'Sivir', 'Rengar']

                                             Address
0  http://matchhistory.na.leagueoflegends.com/en/...
1  http://matchhistory.na.leagueoflegends.com/en/...
2  http://matchhistory.na.leagueoflegends.com/en/...
3  http://matchhistory.na.leagueoflegends.com/en/...
4  http://matchhistory.na.leagueoflegends.com/en/...

[5 rows x 57 columns]
```

1.1 Feature Engineering: Strings

1.1.1 Missing / Wrong String handling

Since R will automaticly one-hot encoding string, here we just need to check the quality of those strings.

League, Year, Season, Type

```
In [4]: print("All unique League:")
        print(data.groupby("League")["Address"].nunique())
        print()
        print("All unique Year:")
        print(data.groupby("Year")["Address"].nunique())
        print()
```

```
        print("All unique Season:")
        print(data.groupby("Season")["Address"].nunique())
        print()
        print("All unique Type:")
        print(data.groupby("Type")["Address"].nunique())
```

```
All unique League:
League
CBLoL     301
CLS       175
EULCS    1099
IEM       138
LCK      1445
LCL       281
LJL       258
LLN       242
LMS       778
MSI       111
NALCS    1272
OPL       458
RR        101
TCL       653
WC        308
Name: Address, dtype: int64

All unique Year:
Year
2014      78
2015    1496
2016    2494
2017    3311
2018     241
Name: Address, dtype: int64

All unique Season:
Season
Spring    3512
Summer    4108
Name: Address, dtype: int64

All unique Type:
Type
International     658
Playoffs          775
Promotion         391
Regional          143
Season           5653
Name: Address, dtype: int64
```

We can see the quality of those data are proper. Do not need to be clean

Team

```
In [5]: print("All blue unique Team:")
        print(data.groupby("blueTeamTag")["Address"].nunique().to_string())
        data.groupby("blueTeamTag")["Address"].nunique().count()
```

```
All blue unique Team:
blueTeamTag
100          3
17A          4
7h          35
A           17
AE          38
AF           4
AFR          2
AFs        111
AHQ         31
ALL          3
ANC         26
ANX         19
APX         28
AS          13
ASC          4
AUR         67
AV          59
B2K         12
BC           9
BE          10
BGG          7
BJK         54
BKT          3
BMR          2
BPI          7
C9         153
C9C          1
CA           7
CF           1
CG           3
CGE         16
CHF         67
CJ          23
CJE         80
CLB          1
CLG        154
```

CLK	35
CNB	34
COL	3
COW	3
CREW	7
CRJ	11
CRW	58
CST	20
CW	24
Crew	1
D9	28
DD	2
DF	7
DFM	39
DH	2
DIG	91
DLY	1
DP	80
DW	64
DoR	6
EDG	25
EEW	29
EG	2
EL	27
EMF	8
EMP	19
ESC	37
EUN	6
EVR	2
F5	7
FAC	5
FB	79
FG	40
FH	1
FIS	1
FLY	61
FNA	8
FNC	151
FOX	88
FSN	11
FW	124
G2	114
GAL	33
GAM	12
GCU	8
GET	21
GG	27
GGS	3

GIA	96
GMB	50
GRX	2
GS	14
GV	24
GlA	2
H2K	146
H2k	3
HAF	23
HKA	26
HKE	69
HKES	9
HLN	21
HMA	3
HR	12
HWA	76
IFG	13
IG	3
IM	19
IMG	4
IMT	94
INF	42
INTZ	1
ISG	25
ITZ	47
JAG	137
JSA	16
JST	14
JT	54
JTH	34
JTM	3
KBM	30
KDM	59
KLG	26
KOO	36
KST	26
KSV	5
KT	12
KZ	4
LGC	61
LGD	4
LGS	14
LK	25
LMQ	3
LOG	6
LYN	39
LZ	119
Longzhu	4

M17	83
M19	23
MAD	2
MC	1
MFA	4
MIL	10
MKZ	1
MM	14
MOU	2
MSE	52
MSF	60
MSK	3
MVP	73
MYM	10
Mac	7
NGU	5
NH	2
NIP	20
NJE	27
NJF	11
NME	11
NR1	25
NRG	32
NTR	1
NV	111
NWS	6
OG	78
OHM	10
OMG	9
ONE	19
OPK	25
OPT	2
ORD	2
P1	81
P3P	18
PDS	32
PNG	39
PRG	9
PRO	2
Prime	3
QG	6
RBE	1
RBT	26
RED	43
REM	7
REN	11
RG	37
RGC	11

RJ	31
RNG	25
ROC	94
ROX	137
RPG	40
SO4	31
SBENU	1
SBK	3
SCW	14
SHR	11
SIN	68
SK	32
SKP	3
SKT	195
SPA	1
SPY	69
SSB	50
SSG	167
SSW	8
SUP	73
SZ	12
T8	22
TDK	14
TIP	45
TL	133
TLA	1
TM	54
TPA	42
TRC	2
TRI	12
TS	1
TSM	146
TSW	7
TT	45
TTC	2
TY	5
UOL	122
USG	38
VEG	38
VFK	20
VIT	57
VK	2
VP	24
VS	34
WE	24
WFX	11
WOR	1
WS	26

```
Winners      2
X5          11
XG          47
Xenics       2
YC           4
ZEN          1
ZONE         7
ZTG         29
ahq         96
as           1
bbq         50
g3x          7
kt         163
yoeFW        8
```

```
Out[5]: 242
```

```
In [6]: print("All red unique Team:")
        print(data.groupby("redTeamTag")["Address"].nunique().to_string())
        data.groupby("redTeamTag")["Address"].nunique().count()
```

```
All red unique Team:
redTeamTag
100          1
17A          5
7h          39
A           16
AE          37
AF           6
AFR          3
AFs        109
AHQ         30
ALL          3
ANC         24
ANX         13
APX         25
AS          14
ASC          2
AUR         63
AV          60
B2K         12
BC          14
BE          11
BGG          9
BJK         52
BKT          3
BMR          2
```

BPI	7
C9	152
C9C	2
CA	7
CF	1
CG	3
CGE	15
CHF	64
CJ	26
CJE	79
CLB	1
CLG	141
CLK	35
CNB	36
COL	2
COW	2
CREW	7
CRJ	11
CRW	53
CST	20
CW	23
CWA	3
Crew	1
D9	29
DD	2
DF	7
DFM	40
DIG	89
DLY	1
DP	93
DW	60
DoR	5
EDG	30
EEW	26
EG	1
EL	27
EMF	10
EMP	16
ESC	32
EUN	11
EVR	2
F5	6
FAC	7
FB	71
FG	31
FH	2
FIS	1
FLY	50

FNA	4
FNC	130
FOX	72
FSN	10
FW	120
G2	112
GAL	35
GAM	12
GCU	14
GET	17
GG	27
GGS	1
GIA	95
GMB	49
GRX	2
GS	14
GV	26
GlA	3
H2K	112
H2k	3
HAF	23
HKA	22
HKE	74
HKES	8
HLN	22
HMA	5
HR	8
HWA	67
IFG	14
IG	3
IM	18
IMT	98
INF	41
INTZ	1
ISG	40
ITZ	43
JAG	150
JSA	17
JST	15
JT	60
JTH	39
JTM	3
KBM	28
KDM	63
KLG	24
KOO	32
KST	18
KSV	4

KT	10
KZ	4
LGC	67
LGD	4
LGS	14
LGT	2
LK	26
LMQ	3
LOG	8
LYN	36
LZ	112
Longzhu	5
M17	78
M19	26
MAD	3
MC	2
MFA	4
MIL	3
MKZ	2
MM	22
MOU	1
MSE	49
MSF	50
MSK	1
MVP	78
MYM	9
Mac	7
NGU	8
NH	2
NIP	19
NJE	32
NJF	14
NME	14
NR1	27
NRG	37
NTR	1
NV	120
NWS	4
OG	84
OHM	10
OMG	9
ONE	9
OPK	24
OPT	2
ORD	2
P1	85
P3P	16
PDS	31

PNG	50
PRG	10
PRO	2
Prime	2
QG	10
RBE	4
RBT	24
RED	47
REM	7
REN	11
RG	37
RGC	11
RJ	19
RNG	23
ROC	94
ROX	136
RPG	41
SO4	31
SBENU	1
SBK	1
SCW	14
SHR	9
SIN	68
SK	28
SKP	4
SKT	208
SPA	1
SPY	86
SSB	51
SSG	155
SSW	9
SUP	81
SZ	9
T8	25
TDK	17
TIP	43
TL	128
TLA	4
TM	54
TPA	44
TPB	4
TRC	1
TRI	12
TS	1
TSM	172
TSW	7
TT	49
TTC	2

```
TY           2
UC           2
UOL        139
USG         43
VEG         52
VFK         17
VIT         65
VK           2
VP          21
VS          39
WE          19
WFX         11
WOR          3
WS          44
Winners      3
X5          11
XG          45
Xenics       3
YC           1
ZEN          1
ZONE         7
ZTG         29
ahq         90
bbq         60
g3x          9
kt         144
yoeFW        7
```

```
Out[6]: 243
```

We can see the data are really messy, we can try the following:

```
In [7]: data["blueTeamTag"] = data["blueTeamTag"].apply(str).apply(str.upper)
        print("Current Unique Blue Teams: ", data.groupby("blueTeamTag")["Address"].nunique().
        data["redTeamTag"] = data["redTeamTag"].apply(str).apply(str.upper)
        print("Current Unique Red Teams: ", data.groupby("redTeamTag")["Address"].nunique().cou
```

```
Current Unique Blue Teams:  238
Current Unique Red Teams:  240
```

```
In [8]: data["blueTeamTag"][~data["blueTeamTag"].isin(data["redTeamTag"])]
```

```
Out[8]: 7262    IMG
        7263    IMG
        7264    IMG
        7265    IMG
        7525     DH
```

```
7530     DH
Name: blueTeamTag, dtype: object
```

```
In [9]: data["redTeamTag"][~data["redTeamTag"].isin(data["blueTeamTag"])]
```

```
Out[9]: 7247    CWA
        7248    CWA
        7249    CWA
        7481    TPB
        7482    TPB
        7483    TPB
        7484    TPB
        7524    LGT
        7525     UC
        7528     UC
        7530    LGT
        Name: redTeamTag, dtype: object
```

Since those teams did exist, they will be kept.

Even though there are still some other problems for those data (e.g. C9C change their name to C9), we do not check here.

```
In [10]: # f0 = lambda x: x[1:-1].split(", ")[0]
         # f1 = lambda x: x[1:-1].split(", ")[1]
         # f2 = lambda x: x[1:-1].split(", ")[2]
         # data["blueBans0"] = data["blueBans"].apply(f0)
         # data["blueBans1"] = data["blueBans"].apply(f1)
         # data["blueBans2"] = data["blueBans"].apply(f2)
         # data["redBans0"] = data["redBans"].apply(f0)
         # data["redBans1"] = data["redBans"].apply(f1)
         # data["redBans2"] = data["redBans"].apply(f2)
```

Since those names seem to be valid, we do not modify them.

1.2 Feature Engineering: Arrays:

1.2.1 Golds:

We decide to engineer all golds by following methods: 1. Calculate the golds for all two teams for all five positions for 10min, 20min, and 30 min. 2. If the games ends before 30min, we will set the gold to be the amount at the end of gold.

```
In [11]: gold_columns = ["goldblueTop","goldredTop",\
                         "goldblueJungle","goldredJungle",\
                         "goldblueMiddle","goldredMiddle",\
                         "goldblueADC","goldredADC",\
                         "goldblueSupport","goldredSupport",]
```

```
        def counting_gold(columns, data = data, periods = [10, 20, 30]):
            def modify(s, sep = ", "):
                s = s[1:-1]
                result = []
                l = s.split(sep)
                for p in periods:
                    if (p < len(l)):
                        result.append(l[p])
                    else:
                        result.append(l[-1])
                return ",".join(result)
            def modify2(s):
                s = s.apply(modify)
                return s
            rdf = pd.DataFrame()
            df = data[columns]
            df = df.apply(modify2)
            f0 = lambda s: s.split(",")[0]
            f1 = lambda s: s.split(",")[1]
            f2 = lambda s: s.split(",")[2]
            for c in df.columns:
                rdf[c+"10"] = df[c].apply(f0).apply(int)
                rdf[c+"20"] = df[c].apply(f1).apply(int)
                rdf[c+"30"] = df[c].apply(f2).apply(int)
            return rdf

        counting_gold(gold_columns).to_excel("tmpgold.xlsx", index = False)
```

1.2.2 Counting Legendary Monsters: Dragon

We decide to engineer dragon by following methods: 1. Count the first dragon slained for both sides, and the total number of dragons for both sides. 2. If one side does not slained any dragon, record the time as the end of game. 3. There are elements dragon in 2017 and 2018, we ingnore them here for now.

```
In [12]: dragon_columns = ["bDragons","rDragons"]
```

```
        def counting_dragon(columns, data = data, end_column = "gamelength"):
            def modify(s, sep = ", "):
                s = s[1:-1]
                l = s.split(sep)
                if l[0] != '':
                    return str(l[0][1:].split(", ")[0]) + ":" + str(len(l)/2)
                else:
                    return "100:0"
            def modify2(s):
                s = s.apply(modify)
```

```
            return s
        rdf = pd.DataFrame()
        df = data[columns]
        df = df.apply(modify2)
        f0 = lambda s: s.split(":")[0]
        f1 = lambda s: s.split(":")[1]
        for c in df.columns:
            rdf[c[0] + "FirstDragon"] = df[c].apply(f0).apply(float)
            rdf[c[0] + "FirstDragon"] = rdf[c[0] + "FirstDragon"].combine(data[end_column]
            rdf[c[0] + "NumofDragon"] = df[c].apply(f1).apply(float)
        return rdf

counting_dragon(dragon_columns).to_excel("tmpdragon.xlsx", index = False)
```

1.2.3 Counting Legendary Monsters: Baron

We decide to engineer baron by following methods: 1. Count the first baron slained for both sides.

```
In [13]: baron_columns = ["bBarons","rBarons"]

         def counting_baron(columns, data = data):
             def modify(s, sep = ", "):
                 s = s[1:-1]
                 l = s.split(sep)
                 if l[0] != '':
                     return len(l)
                 else:
                     return "0"
             def modify2(s):
                 s = s.apply(modify)
                 return s
             rdf = pd.DataFrame()
             df = data[columns]
             df = df.apply(modify2)
             for c in df.columns:
                 rdf[c[0] + "NumofBaron"] = df[c].apply(float)
             return rdf

         counting_baron(baron_columns).to_excel("tmpBaron.xlsx", index = False)
```

1.2.4 Counting Legendary Monsters: Herald

We decide to engineer herald by following methods: 1. Count the number of heralds slained for both sides.

```
In [14]: herald_columns = ["bHeralds","rHeralds"]

         def counting_herald(columns, data = data):
             def modify(s, sep = ", "):
```

```
            s = s[1:-1]
            l = s.split(sep)
            if l[0] != '':
                return len(l)
            else:
                return "0"
        def modify2(s):
            s = s.apply(modify)
            return s
        rdf = pd.DataFrame()
        df = data[columns]
        df = df.apply(modify2)
        for c in df.columns:
            rdf[c[0] + "NumofHerald"] = df[c].apply(float)
        return rdf

counting_herald(herald_columns).to_excel("tmpHerald.xlsx", index = False)
```

1.2.5 Counting Towers: Tower

We decide to engineer tower by following methods: 1. Count the first tower destroyed for both sides, and the total number of towers for both sides. 2. If one side does not destroy any tower, record the time as the end of game.

```
In [15]: tower_columns = ["bTowers","rTowers"]

def counting_tower(columns, data = data, end_column = "gamelength"):
    def modify(s, sep = ", "):
        s = s[1:-1]
        l = s.split(sep)
        if l[0] != '':
            return str(l[0][1:].split(", ")[0]) + ":" + str(len(l)/3)
        else:
            return "100:0"
    def modify2(s):
        s = s.apply(modify)
        return s
    rdf = pd.DataFrame()
    df = data[columns]
    df = df.apply(modify2)
    f0 = lambda s: s.split(":")[0]
    f1 = lambda s: s.split(":")[1]
    for c in df.columns:
        rdf[c[0] + "FirstTower"] = df[c].apply(f0).apply(float)
        rdf[c[0] + "FirstTower"] = rdf[c[0] + "FirstTower"].combine(data[end_column],
        rdf[c[0] + "NumofTower"] = df[c].apply(f1).apply(float)
    return rdf
```

```
counting_tower(tower_columns).to_excel("tmptower.xlsx", index = False)
```

1.2.6 Counting Towers: Inhibitors

We decide to engineer inhibitor by following methods: 1. Count the first inhibitor destroyed for both sides, and the total number of inhibitors for both sides. 2. If one side does not destroy any inhibitor, record the time as the end of game.

In [16]:
```
inhib_columns = ["bInhibs","rInhibs"]

def counting_inhib(columns, data = data, end_column = "gamelength"):
    def modify(s, sep = ", "):
        s = s[1:-1]
        l = s.split(sep)
        if l[0] != '':
            return str(l[0][1:].split(", ")[0]) + ":" + str(len(l)/2)
        else:
            return "100:0"
    def modify2(s):
        s = s.apply(modify)
        return s
    rdf = pd.DataFrame()
    df = data[columns]
    df = df.apply(modify2)
    f0 = lambda s: s.split(":")[0]
    f1 = lambda s: s.split(":")[1]
    for c in df.columns:
        rdf[c[0] + "FirstInhib"] = df[c].apply(f0).apply(float)
        rdf[c[0] + "FirstInhib"] = rdf[c[0] + "FirstInhib"].combine(data[end_column],
        rdf[c[0] + "NumofInhib"] = df[c].apply(f1).apply(float)
    return rdf

counting_inhib(inhib_columns).to_excel("tmpinhib.xlsx", index = False)
```

1.2.7 Counting Kills

We decide to engineer kills by following methods: 1. Count the first kill for both sides, and the total number of kills for both sides. 2. If one side does not kill, record the time as the end of game.

In [17]:
```
kill_columns = ["bKills","rKills"]

def counting_kill(columns, data = data, end_column = "gamelength"):
    def modify(s, sep = "["):
        s = s[2:-1]
        l = s.split(sep)
        if l[0] != '':
            return str(l[0].split(", ")[0]) + ":" + str(len(l)/2)
        else:
            return "100:0"
```

```
        def modify2(s):
            s = s.apply(modify)
            return s
        rdf = pd.DataFrame()
        df = data[columns]
        df = df.apply(modify2)
        f0 = lambda s: s.split(":")[0]
        f1 = lambda s: s.split(":")[1]
        for c in df.columns:
            rdf[c[0] + "FirstKill"] = df[c].apply(f0).apply(float)
            rdf[c[0] + "FirstKill"] = rdf[c[0] + "FirstKill"].combine(data[end_column], m
            rdf[c[0] + "NumofKill"] = df[c].apply(f1).apply(float)
        return rdf

    counting_kill(kill_columns).to_excel("tmpkill.xlsx", index = False)
```

1.3 Columns Dropping:

We will drop Address, Bans, and players for now.

1.4 Final Step: merge data:

```
In [18]: target = data[["bResult","League","Year","Season","Type",\
                        "blueTeamTag","redTeamTag","gamelength",\
                        "blueTopChamp","blueJungleChamp","blueMiddleChamp","blueADCChamp","blue
                        "redTopChamp","redJungleChamp","redMiddleChamp","redADCChamp","redSuppo
         target.columns = ["bResult","League","Year","Season","Type",\
                        "blueTeamTag","redTeamTag","gamelength",\
                        "blueTopChamp","blueJungleChamp","blueMiddleChamp","blueADCChamp","blue
                        "redTopChamp","redJungleChamp","redMiddleChamp","redADCChamp","redSuppo
         df1 = pd.read_excel("tmpgold.xlsx")
         df2 = pd.read_excel("tmpdragon.xlsx")
         df3 = pd.read_excel("tmpBaron.xlsx")
         df4 = pd.read_excel("tmpHerald.xlsx")
         df5 = pd.read_excel("tmptower.xlsx")
         df6 = pd.read_excel("tmpinhib.xlsx")
         df7 = pd.read_excel("tmpkill.xlsx")

In [21]: target = pd.concat([target, df1, df2, df3, df4, df5, df6, df7], axis= 1, ignore_index=
         #target
         target.to_csv("LOL.csv", sep = ",", index = False)

In [ ]:
```