In order to begin processing the League of Legends dataset, all of the csv files were imported into a jupyter notebook in order to determine what preprocessing steps would be needed to convert the data into a useable format. This was done using the package pandas. Upon looking at the head() method of each generated dataframe, It became apparent that only the “LeagueofLegends.csv” file would be needed for the purpose of creating a predictor model.

First, the `.head()`, `.describe()`, and `.info()` methods were called on the main dataframe. It was observed that some columns contained missing values. Upon further inspection, this was a result of the format of the specific tournament the data was referencing. Values such as a team name was missing. Because this dataset was fairly large (~8000 samples) and the number of missing values was quite small (~40), it was decided that the rows containing missing values could be dropped without much consequence. Afterwards, each column of dataframe was inspected in order to determine whether each column would need to be processed and how to process them. Figure 1 below describes how each column of the dataframe would be processed.

```
#1 league_df.League // dummy encode
#2 league_df.Year // dummy encode
#3 league_df.Season // dummy encode
#4 league_df.Type // dummy encode
#5 league_df.blueTeamTag // dummy encode
#6 league_df.bResult // will be used as label
#7 league_df.rResult // dropped because contains same information as bResult
#8 league_df.redTeamTag // dummy encode
#9 league_df.gameLength
#10 league_df.golddiff // calculate var
#11 league_df.goldblue // calculate var
#12 league_df.bKills // retrieve number of kills made by team, will use coordinate data later for plotting where kills occurred
#13 league_df.bTowers // going to convert to column for every turret, value will be time taken, if not taken, value = 0
#14 league_df.bInhibs // convert into number of inhibs taken, will use time for plots later on
#15 league_df.bDragons // create a score metric, based on number of dragons taken and type of dragon taken
#16 league_df.bBarons // create a score metric, similar to dragons
#17 league_df.bHeralds // take lenght to determine if herald was taken
#18 league_df.goldred // going to take total gold at specific time points based on max length of game
#19 league_df.rKills // retrieve number of kills made by team, will use coordinate data later for plotting where kills occurred
#20 league_df.rTowers // going to convert to column for every turret, value will be time taken, if not taken, value = 0
#21 league_df.rInhibs // convert into number of inhibs taken, will use time for plots later on
#22 league_df.rDragons // create a score metric, based on number of dragons taken and type of dragon taken
#23 league_df.rBarons // create a score metric, similar to dragons
#24 league_df.rHeralds // only one herald per game but there are games where herald isnt taken by anyone
#25 league_df.blueTop // dummy encode
#26 league_df.blueTopChamp // dummy encode
#27 league_df.goldblueTop // calculate var
#28 league_df.blueJungle // dummy encode
#29 league_df.blueJungleChamp // dummy encode
#30 league_df.goldblueJungle // calculate var
#31 league_df.blueMiddle // dummy encode
#32 league_df.blueMiddleChamp // dummy encode
#33 league_df.goldblueMiddle // calculate var
#34 league_df.blueADC // dummy encode
#35 league_df.blueADCChamp // dummy encode
#36 league_df.goldblueADC // calculate var
#37 league_df.blueSupport // dummy encode
#38 league_df.blueSupportChamp // dummy encode
#39 league_df.goldblueSupport // calculate var
#40 league_df.blueBans // dummy encode plus fill in empty values as 'empty'
#41 league_df.redTop // dummy encode
#42 league_df.redTopChamp // dummy encode
#43 league_df.goldredTop // calculate var
#44 league_df.redJungle // dummy encode
#45 league_df.redJungleChamp // dummy encodeb
#46 league_df.goldredJungle // calculate var
#47 league_df.redMiddle // dummy encode
#48 league_df.redMiddleChamp // dummy encode
#49 league_df.goldredMiddle // calculate var
#50 league_df.redADC // dummy encode
#51 league_df.redADCChamp // dummy encode
#52 league_df.goldredADC // calculate var
#53 league_df.redSupport // dummy encode
#54 league_df.redSupportChamp // dummy encode
#55 league_df.goldredSupport // calculate var
#56 league_df.redBans // dummy encode
```

Figure 1. Description of how each column in the League of Legends dataframe were to be processed.

Ultimately, due to the structures of the data of the columns, a total of eight functions were needed to completely process all 56 columns of the dataframe.

```
def bans_to_encodable_cols(df, cols, col_names):
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(ast.literal_eval)
        encodable_cols = pd.DataFrame(df[col].values.tolist(), index=league_df.index, columns=names)
        df = pd.concat([df.drop(col, axis=1), encodable_cols], axis = 1)

    return df
```

The first function takes in a dataframe, columns, and column names as arguments. This function was responsible for converting the “blueBans” and “redBans” columns into five columns, one for each possible ban within the game. This was done so that these columns could be encoded.

```
def delete_col_get_dummies(df, cols):

    for col in cols:
        if 'blue' in col:
            if 'Top' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bTop')], axis=1) # add prefix or suffix
            elif 'Jungle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bJungle')], axis=1) # add prefix or suffix
            elif 'Middle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bMiddle')], axis=1) # add prefix or suffix
            elif 'ADC' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bADC')], axis=1) # add prefix or suffix
            elif 'Support' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bSupport')], axis=1) # add prefix or suffi
            else:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='b')], axis=1) # add prefix or suffix
        elif 'red' in col:
            if 'Top' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rTop')], axis=1) # add prefix or suffix
            elif 'Jungle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rJungle')], axis=1) # add prefix or suffix
            elif 'Middle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rMiddle')], axis=1) # add prefix or suffix
            elif 'ADC' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rADC')], axis=1) # add prefix or suffix
            elif 'Support' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rSupport')], axis=1) # add prefix or suffi
            else:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='r')], axis=1) # add prefix or suffix
        else:
            df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col])], axis=1) # add prefix or suffix

    return df
```

The second function is responsible for processing the columns needed to be converted into dummy variables. If-statements were used to add specific prefixes to the names of each column.

```
def var_of_column(df, cols):
    for col in cols:
        df[col] = df[col].apply(ast.literal_eval).apply(np.var)

    return df
```

The third function was responsible for calculating the variance of the numerical columns. This function was most often used on columns with gold values.

```
def count_num(df, cols):

    for col in cols:
        df[col] = df[col].apply(ast.literal_eval).apply(len)

    return df
```

The fourth function was used to count the number of objectives each team retrieved. This was done by converting the column data type from a string to a literal and then applying the length function on each list.

```
def get_drag_type(drag_lst):
    return [drag[1] for drag in drag_lst]

def process_dragons(df, cols, col_names):

    drag_df_lists = []
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(ast.literal_eval).apply(get_drag_type)
        df['num_of_drags'] = df[col].apply(len)

        dict_list = df[col].apply(Counter).tolist()
        drag_df = pd.DataFrame.from_dict(dict_list)
        drag_df.fillna(0, inplace=True)
        drag_df.columns = names
        drag_df.reset_index(drop=True, inplace=True)
        drag_df_lists.append(drag_df)

    df.reset_index(drop=True, inplace=True)
    df = pd.concat([df.drop(cols, axis=1)]+drag_df_lists, axis=1)

    return df
```

The fifth function and sixth function are used to process the dragon columns. First the columns are converted from a string into a list, and then a list of the dragon types are generated. Another column of how many dragons were killed by each team was generated by taking the length of this list. Finally, the Counter class was used to retrieve a count of how many of each type of dragon was taken, and these counter objects were converted into separate columns.

```
tower_dict = {'TOP_LANE': {'OUTER_TURRET': 0, 'INNER_TURRET': 1, 'BASE_TURRET': 2},
              'MID_LANE': {'OUTER_TURRET': 3, 'INNER_TURRET': 4, 'BASE_TURRET': 5},
              'BOT_LANE': {'OUTER_TURRET': 6, 'INNER_TURRET': 7, 'BASE_TURRET': 8}}

def convert_to_tower_list(tower_str):

    tower_list = [0,0,0,0,0,0,0,0,0]
    for towers_destroyed in ast.literal_eval(tower_str):
        if len(towers_destroyed) > 0:
            try:
                index = tower_dict[towers_destroyed[1]][towers_destroyed[2]]
                tower_list[index] = towers_destroyed[0]
            except:
                pass

    return tower_list


def process_towers(df, cols, col_names):

    tower_df_lists = []
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(convert_to_tower_list)
        towers_df = pd.DataFrame(df[col].values.tolist(), columns=names)
        towers_df.head()
        towers_df.reset_index(drop=True, inplace=True)
        tower_df_lists.append(towers_df)

    df.reset_index(drop=True, inplace=True)
    df=pd.concat([df.drop(cols, axis=1)] + tower_df_lists, axis=1)

    return df
```

The seventh and eight functions were used to process the tower columns. The purpose of these functions were to generate a set of columns containing information on when each teams' towers were destroyed. This was done by first creating a dictionary containing information on what index each tower would be located at, then creating a function which would take the list within each cell of each column and assign the times each tower was taken to a specific index using that dictionary. Finally, this was used on the tower columns using the `.apply()` method.

In terms of outliers, the only true outliers present were certain games whose game lengths were almost two times the average length. It is difficult to determine whether these sets of data will heavily affect the results. Games are often determined by hundreds of different factors, so these samples were kept. Should the results of the model be heavily affected by outliers, then they shall be removed.