TERM PROJECT

January 20, 2019

0.1 INTRODUCTION TO PYTHON PROGRAMMING - TERM PROJECT

by Mercan KARACABEY

```
In [4]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib.pyplot as plt
        import seaborn as sns  # visualization tool
```

```
In [6]: data = pd.read_csv('/Users/macboookair/Downloads/csgo-small.csv')
```

```
In [10]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1127 entries, 0 to 1126
Data columns (total 37 columns):
win_ratio                    1127 non-null float64
total_accuracy               1127 non-null float64
kill_to_death_ratio          1127 non-null float64
total_wins_per_hour          1127 non-null float64
mvp_per_round                1127 non-null float64
total_headshots_per_round    1127 non-null float64
accuracy_ssg08               1127 non-null float64
accuracy_awp                 1127 non-null float64
accuracy_deagle              1127 non-null float64
accuracy_aug                 1127 non-null float64
accuracy_scar20              1127 non-null float64
accuracy_m4a1                1127 non-null float64
accuracy_ak47                1127 non-null float64
accuracy_bizon               1120 non-null float64
accuracy_elite               1104 non-null float64
accuracy_famas               1123 non-null float64
accuracy_fiveseven           1100 non-null float64
accuracy_g3sg1               1068 non-null float64
accuracy_galilar             1123 non-null float64
accuracy_glock               1126 non-null float64
accuracy_hkp2000             1127 non-null float64
accuracy_m249                1100 non-null float64
accuracy_mac10               1095 non-null float64
```

```
accuracy_mag7                1110 non-null float64
accuracy_mp7                 1118 non-null float64
accuracy_mp9                 1090 non-null float64
accuracy_negev               1108 non-null float64
accuracy_nova                1119 non-null float64
accuracy_p250                1110 non-null float64
accuracy_p90                 1125 non-null float64
accuracy_sawedoff            1088 non-null float64
accuracy_sg556               1122 non-null float64
accuracy_tec9                1088 non-null float64
accuracy_ump45               1114 non-null float64
accuracy_xm1014              1109 non-null float64
total_games_owned            1127 non-null int64
VACBanned                    1127 non-null int64
dtypes: float64(35), int64(2)
memory usage: 325.9 KB
```

```
In [11]: data.head(10)
```

Out[11]:

	win_ratio	total_accuracy	kill_to_death_ratio	total_wins_per_hour	\
0	0.51	0.17	0.83	15.43	
1	0.51	0.24	1.82	19.36	
2	0.60	0.19	1.42	26.85	
3	0.54	0.19	1.64	19.18	
4	0.70	0.19	2.41	8.48	
5	0.58	0.20	1.35	16.48	
6	0.81	0.19	1.78	118.90	
7	0.55	0.22	0.76	51.92	
8	0.54	0.21	1.91	10.80	
9	0.70	0.17	2.02	31.15	

	mvp_per_round	total_headshots_per_round	accuracy_ssg08	accuracy_awp	\
0	0.10	0.59	0.25	0.34	
1	0.17	0.97	0.36	0.47	
2	0.17	0.46	0.31	0.37	
3	0.12	1.19	0.24	0.41	
4	0.29	5.53	0.46	0.45	
5	0.14	0.78	0.42	0.44	
6	0.14	0.36	0.30	0.32	
7	0.07	0.43	0.45	0.40	
8	0.21	2.61	0.20	0.41	
9	0.16	0.71	0.23	0.43	

	accuracy_deagle	accuracy_aug	...	accuracy_nova	accuracy_p250	\
0	0.20	0.17	...	0.16	0.13	
1	0.28	0.26	...	0.24	0.27	
2	0.43	0.31	...	0.17	NaN	

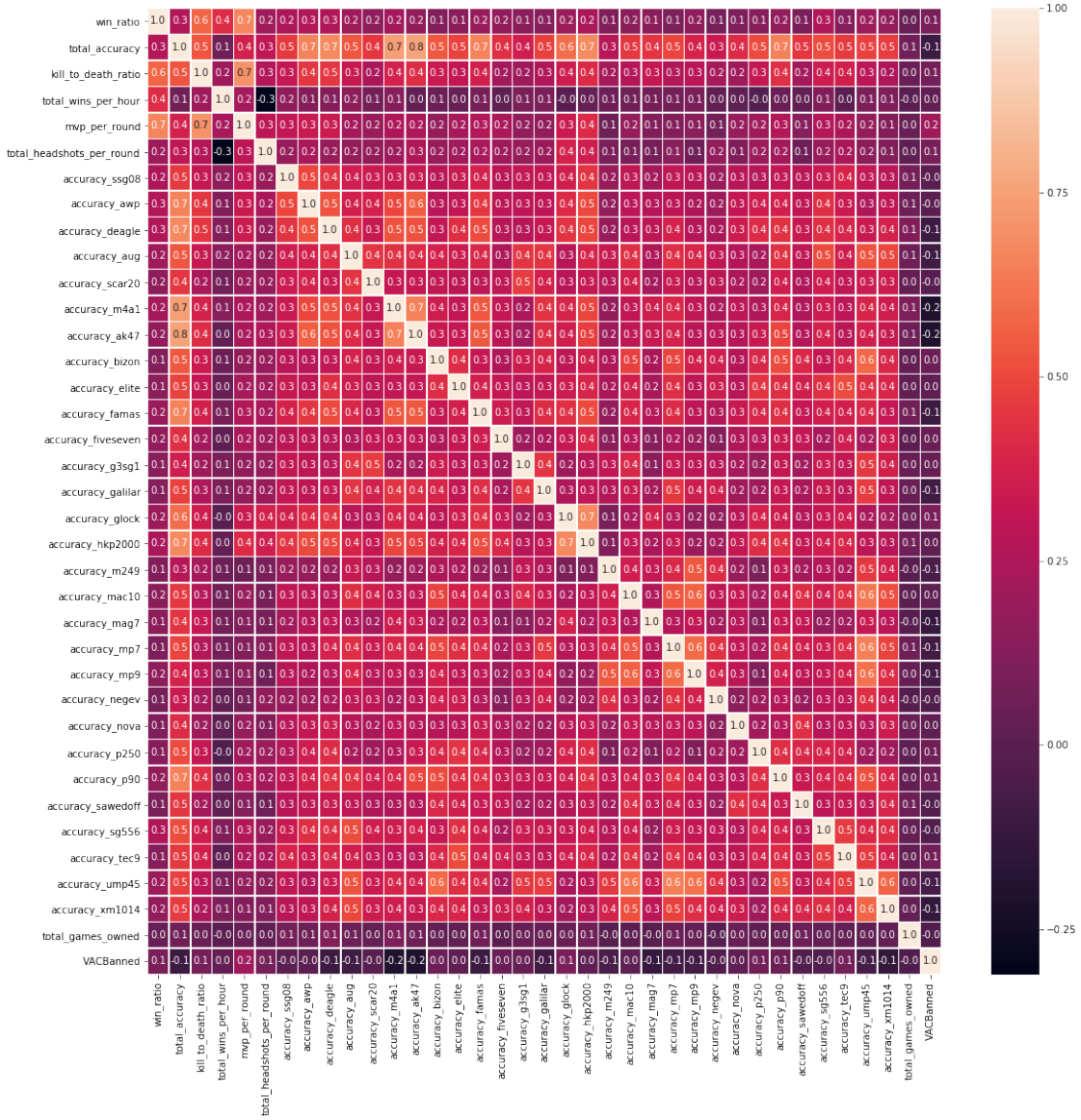
```
3              0.25          0.27    ...               0.17          0.19
4              0.27          0.22    ...               0.22          0.21
5              0.30          0.17    ...               0.17          0.23
6              0.22          0.40    ...               0.24          0.20
7              0.22          0.38    ...               0.18          0.24
8              0.27          0.18    ...               0.20          0.24
9              0.21          0.43    ...               0.33          0.23

   accuracy_p90  accuracy_sawedoff  accuracy_sg556  accuracy_tec9  \
0          0.20               0.14            0.10           0.20
1          0.21               0.20            0.23           0.23
2          0.21               0.15            0.21            NaN
3          0.18               0.26            0.22           0.25
4          0.15               0.15            0.22           0.13
5          0.16               0.17            0.14           0.13
6          0.24               0.17            0.21           0.26
7          0.23               0.16            0.35           0.28
8          0.18               0.18            0.18           0.21
9          0.14                NaN            0.07           0.18

   accuracy_ump45  accuracy_xm1014  total_games_owned  VACBanned
0            0.08             0.22                 70          1
1            0.19             0.23                 83          1
2            0.20             0.33                 99          1
3            0.18             0.22                 39          1
4            0.16             0.19                 44          1
5            0.23             0.24                 11          1
6            0.29             0.28                 20          1
7            0.29             0.23                 35          1
8            0.24             0.18                 33          1
9            0.16             0.18                 98          1

[10 rows x 37 columns]
```

```
In [6]: data.describe()
```

```
Out[6]:              1000001             10              0              3  \
count   5.375760e+05  537576.000000  537576.000000  537576.000000
mean    1.002992e+06       8.082706       0.408798       5.295551
std     1.714389e+03       6.524125       0.491612       3.750703
min     1.000001e+06       0.000000       0.000000       1.000000
25%     1.001495e+06       2.000000       0.000000       1.000000
50%     1.003031e+06       7.000000       0.000000       5.000000
75%     1.004417e+06      14.000000       1.000000       8.000000
max     1.006040e+06      20.000000       1.000000      18.000000

          Unnamed: 9    Unnamed: 10           8370
count  370591.000000  164278.000000  537576.000000
```

```
mean          9.842144      12.669840    9333.861646
std           5.087259       4.124341    4981.026592
min           2.000000       3.000000     185.000000
25%           5.000000       9.000000    5866.000000
50%           9.000000      14.000000    8062.000000
75%          15.000000      16.000000   12073.000000
max          18.000000      18.000000   23961.000000
```

In [12]:
```
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(data.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```



0.1.1 Correlation Win Ratio and Other Factors

Without accuracy data, we can look the affect of these variables to WIN_RATIO.

```
In [7]: data['kill_to_death_ratio'].corr(data['win_ratio'])
```

```
Out[7]: 0.5722237166867538
```

```
In [10]: data['win_ratio'].corr(data['total_headshots_per_round'])
```

```
Out[10]: 0.20498015474143996
```

```
In [11]: data['win_ratio'].corr(data['mvp_per_round'])
```

```
Out[11]: 0.6683842375032701
```

0.2 Linear Regression

Biggest correlation value and WIN RATIO regression was investigated.

STATISTICALLY ANALYSIS

```
In [14]: ## Without a constant

         import statsmodels.api as sm

         X = data["win_ratio"]
         y = data["mvp_per_round"]

         # Note the difference in argument order
         model = sm.OLS(y, X).fit()
         predictions = model.predict(X) # make the predictions by the model

         # Print out the statistics
         model.summary()
```

```
Out[14]: <class 'statsmodels.iolib.summary.Summary'>
         """
                                     OLS Regression Results
         ==============================================================================
         Dep. Variable:          mvp_per_round   R-squared:                       0.816
         Model:                            OLS   Adj. R-squared:                  0.816
         Method:                 Least Squares   F-statistic:                     4995.
         Date:                Sun, 20 Jan 2019   Prob (F-statistic):               0.00
         Time:                        16:05:07   Log-Likelihood:                1579.7
         No. Observations:                1127   AIC:                            -3157.
         Df Residuals:                    1126   BIC:                            -3152.
         Df Model:                           1
         Covariance Type:            nonrobust
         ==============================================================================
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
win_ratio      0.2388      0.003     70.678      0.000       0.232       0.245
==============================================================================
Omnibus:                      689.729   Durbin-Watson:                   1.845
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8304.164
Skew:                           2.641   Prob(JB):                         0.00
Kurtosis:                      15.204   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

RESULT - STATISTICALLY We can look R-Squared value. If R-Squared value close to 1, between these data(win_ratio & mvp_per_round) are significantly meaningful.

USING MACHINE LEARNING / ANALYSIS

In [74]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score


# Use only one feature
MVP_PER_ROUND = data["mvp_per_round"]
```

In [72]:
```
MVP_PER_ROUND_TRAIN = MVP_PER_ROUND[:-30]
MVP_PER_ROUND_TEST = MVP_PER_ROUND[-30:]
```

In [71]:
```
WIN_RATIO = data["win_ratio"]
```

In [73]:
```
WIN_RATIO_TRAIN = WIN_RATIO[:-30]
WIN_RATIO_TEST = WIN_RATIO[-30:]
```

In [36]:
```
WIN_RATIO_TRAIN= WIN_RATIO_TRAIN.values.reshape(-1, 1)
MVP_PER_ROUND_TRAIN= MVP_PER_ROUND_TRAIN.values.reshape(-1, 1)
```

In [37]:
```
# x from 0 to 30


x = WIN_RATIO_TRAIN

# y = a*x + b with noise
y = 0.5 * x + 1.0 + np.random.normal(size=x.shape)
```

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
```

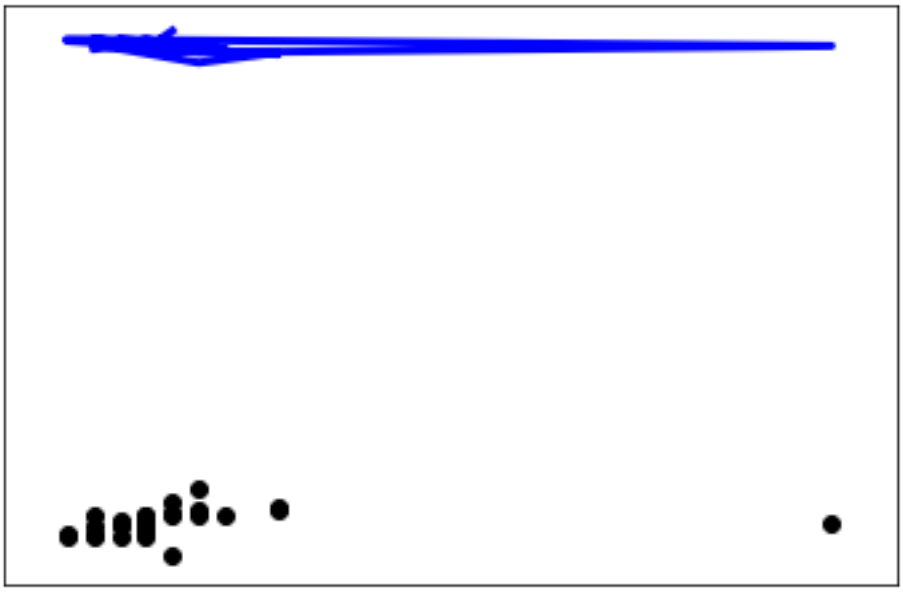
```
        from sklearn.linear_model import LinearRegression
        # create a linear regression model
        model = LinearRegression()
        model.fit(x, y)
```

```
Out[38]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [39]: MVP_PER_ROUND_TEST = MVP_PER_ROUND_TEST.values.reshape(-1,1)

        # Make predictions using the testing set
        MVP_Pred = model.predict(MVP_PER_ROUND_TEST)
```

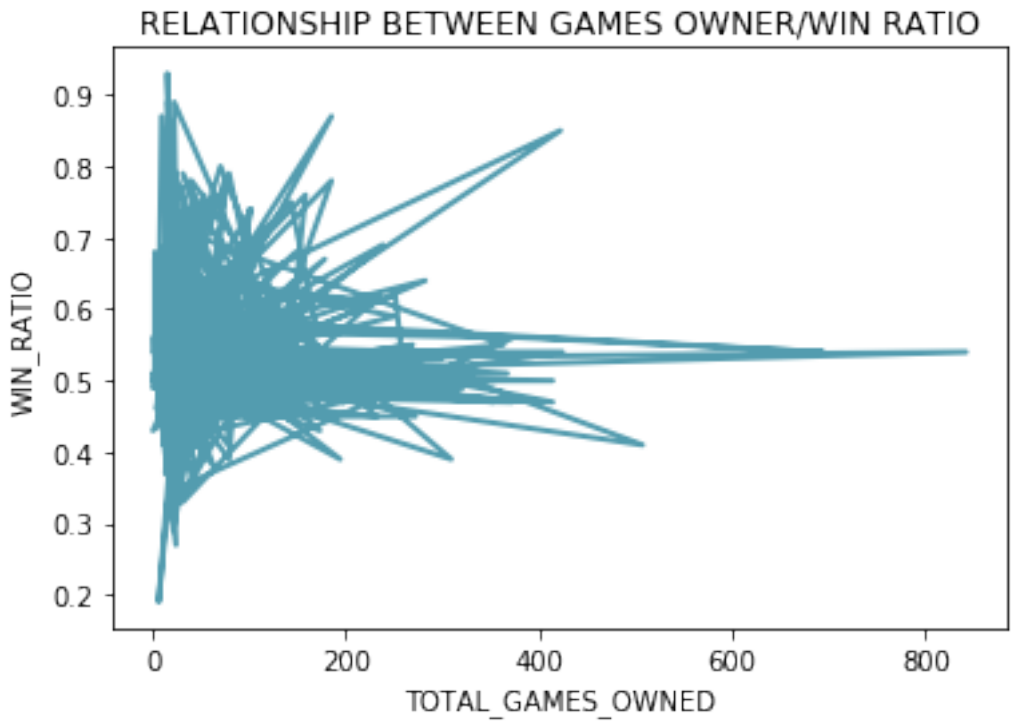
```
In [45]: # The coefficients
        print('Coefficients: \n', model.coef_)
        # The mean squared error
        print("Mean squared error: %.2f"
              % mean_squared_error(MVP_PER_ROUND_TEST, MVP_Pred))
        # Explained variance score: 1 is perfect prediction
        print('Variance score: %.2f' % r2_score(MVP_PER_ROUND_TEST, MVP_Pred))

        # Plot outputs
        plt.scatter(WIN_RATIO_TEST, MVP_PER_ROUND_TEST,  color='black')
        plt.plot(WIN_RATIO_TEST, MVP_Pred, color='blue', linewidth=3)

        plt.xticks(())
        plt.yticks(())

        plt.show()
```

```
Coefficients:
 [[-0.48699953]]
Mean squared error: 1.89
Variance score: -1438.05
```

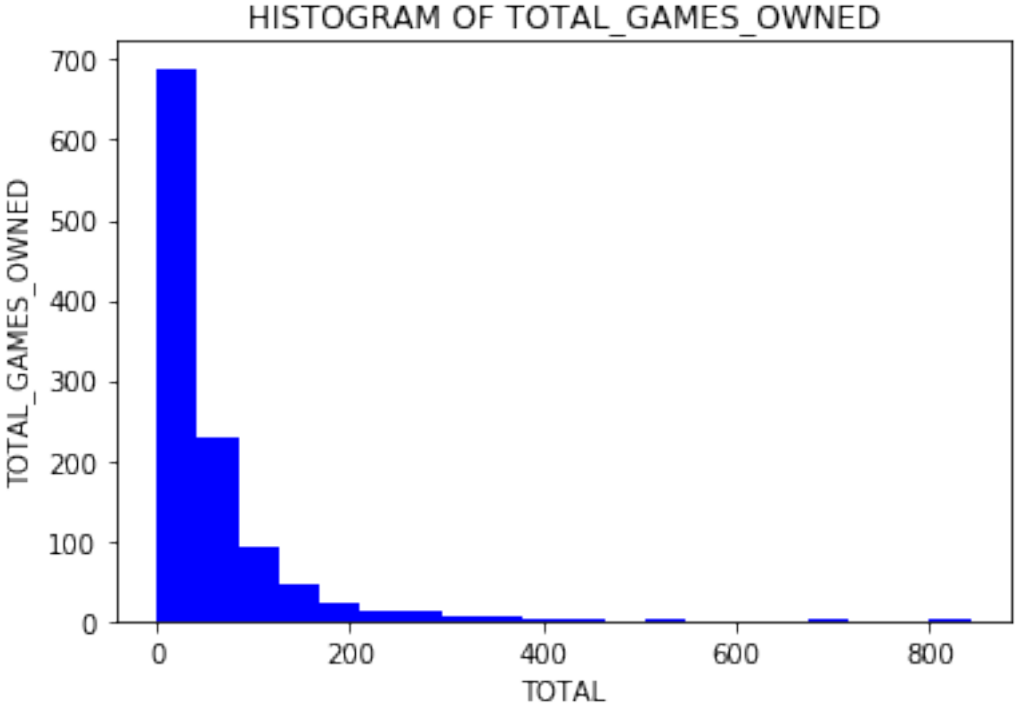


Test&train dataset was not enough. So, I couldnt show the regression of two variables.

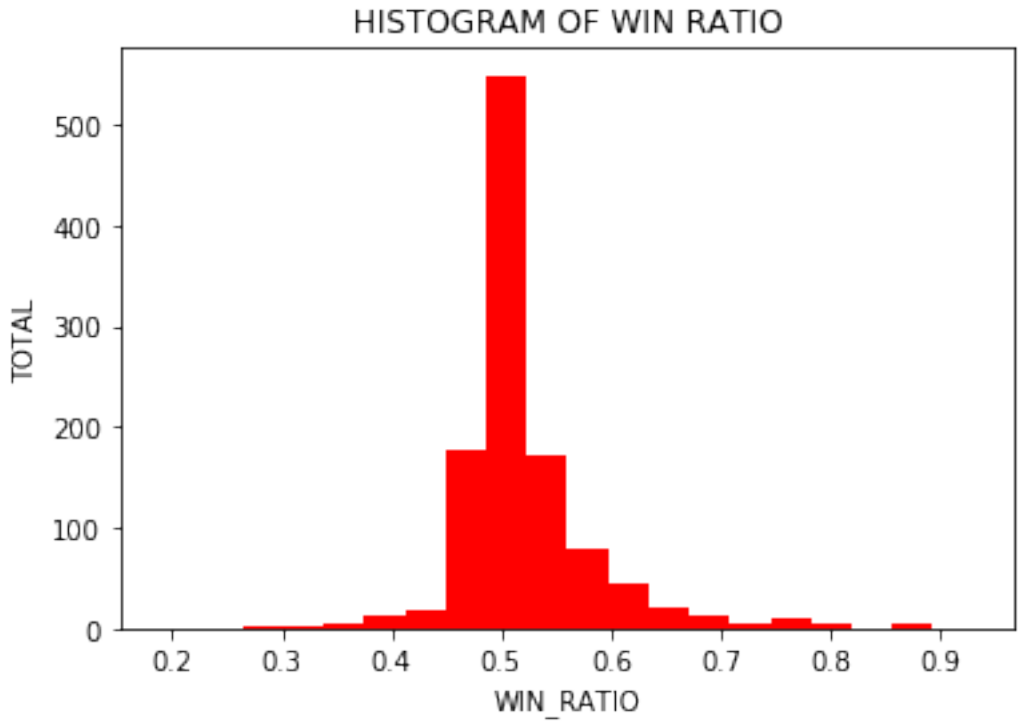
```
In [49]: def lineplot(x_data, y_data, x_label="", y_label="", title=""):
             # Create the plot object
             _, ax = plt.subplots()

             # Plot the best fit line, set the linewidth (lw), color and
             # transparency (alpha) of the line
             ax.plot(x_data, y_data, lw = 2, color = '#539caf', alpha = 1)

             # Label the axes and provide a title
             ax.set_title(title)
             ax.set_xlabel(x_label)
             ax.set_ylabel(y_label)

In [50]: lineplot(data['total_games_owned'],data['win_ratio'],"TOTAL_GAMES_OWNED","WIN_RATIO",'
```




```
In [70]: total_owner_games = data['total_games_owned']
         histogram(total_owner_games,20,False,"TOTAL","TOTAL_GAMES_OWNED","HISTOGRAM OF TOTAL_
```



```
In [69]: def histogram(data, n_bins, cumulative=False, x_label = "", y_label = "", title = "")
             _, ax = plt.subplots()
             ax.hist(data, bins = n_bins, cumulative = cumulative, color = 'BLUE')
             ax.set_ylabel(y_label)
             ax.set_xlabel(x_label)
             ax.set_title(title)

In [65]: win_ratio_data  = data['win_ratio']

In [68]: histogram(win_ratio_data,20,False,"WIN_RATIO","TOTAL","HISTOGRAM OF WIN RATIO")
```




```
In [76]: # Overlay 2 histograms to compare them
         def overlaid_histogram(data1, data2, n_bins = 0, data1_name="", data1_color="#539caf"
             # Set the bounds for the bins so that the two distributions are fairly compared
             max_nbins = 10
             data_range = [min(min(data1), min(data2)), max(max(data1), max(data2))]
             binwidth = (data_range[1] - data_range[0]) / max_nbins


             if n_bins == 0:
                     bins = np.arange(data_range[0], data_range[1] + binwidth, binwidth)
```

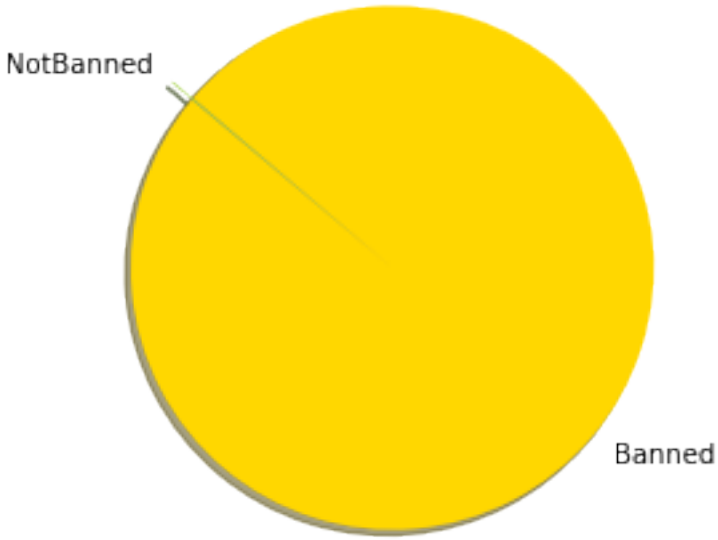
```
        else:
                bins = n_bins

        # Create the plot
        _, ax = plt.subplots()
        ax.hist(data1, bins = bins, color = data1_color, alpha = 1, label = data1_name)
        ax.hist(data2, bins = bins, color = data2_color, alpha = 0.75, label = data2_name)
        ax.set_ylabel(y_label)
        ax.set_xlabel(x_label)
        ax.set_title(title)
        ax.legend(loc = 'best')
```

OVERLAID FUNCTION didnt work ,because values didnt match.
VACBanned Analysis

In [92]:
```
countBanned = 0
countNotBanned = 1
ban = data['VACBanned']
for i in ban:
    if ban[i] == 1:
        countBanned = countBanned +1
    else:
        countNotBanned = countNotBanned + 1
```

In [93]:
```
dataBanned = [countBanned,countNotBanned]
dframe = pd.DataFrame(dataBanned)
```

In [94]:
```
# Data to plot
labels = 'Banned', 'NotBanned'
sizes = [countBanned, countNotBanned]
colors = ['gold', 'yellowgreen']
explode = (0.1, 0)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors, shadow=True, startangle=

plt.axis('equal')
plt.show()
```


NotBanned
Banned