```
df = pd.read_excel('Valorant_Agent')
df['KDA'] = (df['Kill'] + df['Assist']) / df['Death']
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

	Name	Role	Map	KD	Win	Pick	AvgScore	Matches	Kill	Death	Assist	
0	Astra	controller	Split	1.06	51.0	0.9	215	2228	15.4	14.6	6.1	1.47
1	Killjoy	sentinel	Split	0.97	49.9	4.5	200	10637	13.8	14.2	4.1	1.26
2	Raze	duelist	Split	1.06	48.8	9.0	243	21377	16.4	15.4	4.9	1.38
3	Reyna	duelist	Split	1.14	48.1	13.2	247	31275	17.3	15.1	4.4	1.43
4	Skye	initiator	Split	0.92	47.7	3.6	197	8472	13.8	14.9	7.1	1.40

```
manova = MANOVA.from_formula('KD +Win+  Pick+ AvgScore + Kill +Death+ Assist+KDA  ~ Name + Map + (Name * Map)', data=df)
```

```
print(manova.mv_test())
```

```
                  Multivariate linear model
==================================================================
                                                                  
------------------------------------------------------------------
       Intercept         Value  Num DF   Den DF     F Value   Pr > F
------------------------------------------------------------------
          Wilks' lambda   0.0038 8.0000 3492.0000 114520.3557 0.0000
         Pillai's trace   0.9962 8.0000 3492.0000 114520.3557 0.0000
 Hotelling-Lawley trace 262.3605 8.0000 3492.0000 114520.3557 0.0000
    Roy's greatest root 262.3605 8.0000 3492.0000 114520.3557 0.0000
------------------------------------------------------------------
                                                                  
------------------------------------------------------------------
            Name          Value    Num DF    Den DF   F Value  Pr > F
------------------------------------------------------------------
            Wilks' lambda 0.0828 152.0000 25911.3595  68.0440 0.0000
           Pillai's trace 1.7606 152.0000 27992.0000  51.9648 0.0000
   Hotelling-Lawley trace 3.8122 152.0000 19163.3031  87.5387 0.0000
      Roy's greatest root 1.8652  19.0000  3499.0000 343.4883 0.0000
------------------------------------------------------------------
                                                                  
------------------------------------------------------------------
              Map           Value   Num DF   Den DF   F Value Pr > F
------------------------------------------------------------------
              Wilks' lambda 0.8787 48.0000 17186.1666  9.5361 0.0000
             Pillai's trace 0.1250 48.0000 20982.0000  9.3001 0.0000
     Hotelling-Lawley trace 0.1339 48.0000 11622.9008  9.7395 0.0000
        Roy's greatest root 0.0948  8.0000  3497.0000 41.4439 0.0000
------------------------------------------------------------------
                                                                  
------------------------------------------------------------------
           Name:Map        Value    Num DF   Den DF   F Value Pr > F
------------------------------------------------------------------
             Wilks' lambda 0.3213 912.0000 27892.6574  4.6749 0.0000
            Pillai's trace 0.9178 912.0000 27992.0000  3.9775 0.0000
    Hotelling-Lawley trace 1.4723 912.0000 25875.3009  5.6346 0.0000
       Roy's greatest root 1.0157 114.0000  3499.0000 31.1741 0.0000
==================================================================
```

```
fit1 = ols('KD ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	14.242020	0.749580	178.121061	0.0
Residual	3619.0	15.229698	0.004208	NaN	NaN

```
fit1 = ols('Win ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	8131.419582	427.969452	32.58218	2.754736e-109
Residual	3619.0	47535.845123	13.135077	NaN	NaN

```
fit1 = ols('KDA ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	18.671950	0.982734	130.981037	0.0
Residual	3619.0	27.152901	0.007503	NaN	NaN

```
fit1 = ols('Pick ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	31070.514763	1635.290251	285.539332	0.0
Residual	3619.0	20726.095344	5.727023	NaN	NaN

```
fit1 = ols('AvgScore ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	924597.631894	48663.033258	334.359076	0.0
Residual	3619.0	526713.733866	145.541236	NaN	NaN

```
fit1 = ols('Assist ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	7830.782536	412.146449	1027.424223	0.0
Residual	3619.0	1451.745021	0.401145	NaN	NaN

```
fit1 = ols('Kill ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	4541.443682	239.023352	156.283718	0.0
Residual	3619.0	5534.968854	1.529419	NaN	NaN

```
fit1 = ols('Death ~ Name', data = df).fit()
annova1 = sm.stats.anova_lm(fit1)
annova1
```

	df	sum_sq	mean_sq	F	PR(>F)
Name	19.0	820.192098	43.168005	186.168863	0.0
Residual	3619.0	839.157570	0.231876	NaN	NaN

```
# null hypothesis:group1 = group2 same map
# reject null group1 =! group 2 same map
tukey = pairwise_tukeyhsd(df["Assist"],groups = df["Map"]+df['Name'])
results_df = pd.DataFrame(data=tukey._results_table.data[1:], columns=tukey._results_table.data[0])
```

```
results_df
```

	group1	group2	meandiff	p-adj	lower	upper	reject
0	AscentAstra	AscentBreach	-0.3154	1.0000	-1.0303	0.3996	False
1	AscentAstra	AscentBrimstone	2.0038	0.0000	1.2889	2.7188	True
2	AscentAstra	AscentChamber	-3.8962	0.0000	-4.6111	-3.1812	True
3	AscentAstra	AscentCypher	-1.9846	0.0000	-2.6996	-1.2697	True
4	AscentAstra	AscentFade	-0.4769	0.9795	-1.1919	0.2380	False
...	...	...	...	...	...	...	...

```
results_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9730 entries, 0 to 9729
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   group1    9730 non-null   object 
 1   group2    9730 non-null   object 
 2   meandiff  9730 non-null   float64
 3   p-adj     9730 non-null   float64
 4   lower     9730 non-null   float64
 5   upper     9730 non-null   float64
 6   reject    9730 non-null   bool   
dtypes: bool(1), float64(4), object(2)
memory usage: 465.7+ KB
```

```
results_df= results_df[results_df['reject'] == True]
results_df.head()
```

	group1	group2	meandiff	p-adj	lower	upper	reject
1	AscentAstra	AscentBrimstone	2.0038	0.0	1.2889	2.7188	True
2	AscentAstra	AscentChamber	-3.8962	0.0	-4.6111	-3.1812	True
3	AscentAstra	AscentCypher	-1.9846	0.0	-2.6996	-1.2697	True
5	AscentAstra	AscentHarbor	-0.9731	0.0	-1.6880	-0.2581	True
6	AscentAstra	AscentJett	-3.2077	0.0	-3.9226	-2.4928	True

```
results_df[['Map1', 'Agent1']] = results_df['group1'].str.extract(r'([A-Z][a-z]*)([A-Z][a-z]*)')
results_df[['Map2', 'Agent2']] = results_df['group2'].str.extract(r'([A-Z][a-z]*)([A-Z][a-z]*)')
results_df = results_df.replace('K', 'KAY/O')
results_df
```

```
<ipython-input-47-f7b839b1e16f>:1: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  results_df[['Map1', 'Agent1']] = results_df['group1'].str.extract(r'([A-Z][a-z]*)([A-Z][a-z]*)')
<ipython-input-47-f7b839b1e16f>:1: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
results_df= results_df[results_df['Map1'] == results_df['Map2']]
results_df
```

	group1	group2	meandiff	p-adj	lower	upper	reject	Map1	Agent1	Map2	Agent:
1	AscentAstra	AscentBrimstone	2.0038	0.0	1.2889	2.7188	True	Ascent	Astra	Ascent	Brimston
2	AscentAstra	AscentChamber	-3.8962	0.0	-4.6111	-3.1812	True	Ascent	Astra	Ascent	Chambe
3	AscentAstra	AscentCypher	-1.9846	0.0	-2.6996	-1.2697	True	Ascent	Astra	Ascent	Cyphe
5	AscentAstra	AscentHarbor	-0.9731	0.0	-1.6880	-0.2581	True	Ascent	Astra	Ascent	Harbo
6	AscentAstra	AscentJett	-3.2077	0.0	-3.9226	-2.4928	True	Ascent	Astra	Ascent	Jet
...	...	...	...	...	...	...	...	...	...	...	..
9724	SplitSkye	SplitSova	-2.4000	0.0	-3.1149	-1.6851	True	Split	Skye	Split	Sova
9725	SplitSkye	SplitViper	-2.2846	0.0	-2.9996	-1.5697	True	Split	Skye	Split	Vipe
9726	SplitSkye	SplitYoru	-3.3769	0.0	-4.0919	-2.6620	True	Split	Skye	Split	Yoru
9728	SplitSova	SplitYoru	-0.9769	0.0	-1.6919	-0.2620	True	Split	Sova	Split	Yoru
9729	SplitViper	SplitYoru	-1.0923	0.0	-1.8072	-0.3774	True	Split	Viper	Split	Yoru

1023 rows × 11 columns

9724 SplitSkye SplitSova -2.4000 0.0 -3.1149 -1.6851 True Split Skye Split Sova

```
import scipy.stats as stats

p_values = []
f_statistics = []

for _, row in results_df.iterrows():
    agent1 = df[(df['Name'] == row['Agent1']) & (df['Map'] == row['Map1'])]['Assist']
    agent2 = df[(df['Name'] == row['Agent2']) & (df['Map'] == row['Map2'])]['Assist']
    t_statistic, p_value = stats.ttest_ind(agent1, agent2, equal_var=True, alternative='two-sided')
    p_values.append(p_value)
    f_statistics.append(t_statistic)

results_df['p_value'] = p_values
results_df['f_statistic'] = f_statistics
results_df
```

```
<ipython-input-50-7a2583d7c214>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
import pandas as pd
results_df = results_df[results_df['p_value'] < 0.05]
results_df['operator'] = results_df['f_statistic'].apply(lambda x: '>' if x > 0 else '<')
results_df
```

```
<ipython-input-51-2be0e0c991b2>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  results_df['operator'] = results_df['f_statistic'].apply(lambda x: '>' if x > 0 else '<')
```

	group1	group2	meandiff	p-adj	lower	upper	reject	Map1	Agent1	Map2	Agent
1	AscentAstra	AscentBrimstone	2.0038	0.0	1.2889	2.7188	True	Ascent	Astra	Ascent	Brimston
2	AscentAstra	AscentChamber	-3.8962	0.0	-4.6111	-3.1812	True	Ascent	Astra	Ascent	Chambe
3	AscentAstra	AscentCypher	-1.9846	0.0	-2.6996	-1.2697	True	Ascent	Astra	Ascent	Cyphe
5	AscentAstra	AscentHarbor	-0.9731	0.0	-1.6880	-0.2581	True	Ascent	Astra	Ascent	Harbo
6	AscentAstra	AscentJett	-3.2077	0.0	-3.9226	-2.4928	True	Ascent	Astra	Ascent	Jet
...	...	...	...	...	...	...	...	...	...	...	..
9724	SplitSkye	SplitSova	-2.4000	0.0	-3.1149	-1.6851	True	Split	Skye	Split	Sov
9725	SplitSkye	SplitViper	-2.2846	0.0	-2.9996	-1.5697	True	Split	Skye	Split	Vipe

```
df1 = results_df.reindex(columns=['group1', 'group2', 'meandiff',   'p-adj',    'lower',    'upper','reject', 'p_value', 'f_statistic', '
df1
```

	group1	group2	meandiff	p-adj	lower	upper	reject	p_value	f_statistic	Map1
1	AscentAstra	AscentBrimstone	2.0038	0.0	1.2889	2.7188	True	2.151390e-06	-5.355900	Ascent
2	AscentAstra	AscentChamber	-3.8962	0.0	-4.6111	-3.1812	True	1.247316e-21	16.284827	Ascent
3	AscentAstra	AscentCypher	-1.9846	0.0	-2.6996	-1.2697	True	1.999427e-11	8.598854	Ascent
5	AscentAstra	AscentHarbor	-0.9731	0.0	-1.6880	-0.2581	True	4.032376e-04	3.792307	Ascent
6	AscentAstra	AscentJett	-3.2077	0.0	-3.9226	-2.4928	True	9.424875e-19	13.858793	Ascent
...	...	...	...	...	...	...	...	...	...	...
9724	SplitSkye	SplitSova	-2.4000	0.0	-3.1149	-1.6851	True	2.269762e-28	23.139696	Split
9725	SplitSkye	SplitViper	-2.2846	0.0	-2.9996	-1.5697	True	1.352052e-26	21.149641	Split

```
x = input('Enter the Map: ')
df1 = df1[results_df['Map1'] == x]
```

```
Enter the Map: Split
```

```
import pandas as pd

# create an empty dictionary to store the rankings
rankings = {}

# loop through the rows of the dataframe
for index, row in df1.iterrows():
    # extract the relevant data from the row
    agent1 = row['Agent1']
    operator = row['operator']
    agent2 = row['Agent2']
    # update the rankings dictionary
    if operator == '>':
        if agent1 not in rankings:
            rankings[agent1] = {'greater': 0, 'less': 0}
        if agent2 not in rankings:
            rankings[agent2] = {'greater': 0, 'less': 0}
        rankings[agent1]['greater'] += 1
        rankings[agent2]['less'] += 1
    elif operator == '<':
        if agent1 not in rankings:
            rankings[agent1] = {'greater': 0, 'less': 0}
        if agent2 not in rankings:
            rankings[agent2] = {'greater': 0, 'less': 0}
        rankings[agent1]['less'] += 1
        rankings[agent2]['greater'] += 1

# sort the rankings by the number of times each agent was ranked greater than another agent
sorted_rankings = sorted(rankings.items(), key=lambda x: x[1]['greater'], reverse=True)

# print out the rankings
print('    Agent Rankings:')
print('--------------------------')
for i, (agent, counts) in enumerate(sorted_rankings):
    greater_count = counts['greater']
    less_count = counts['less']
    total_count = greater_count + less_count
    if total_count == 0:
        rank = 'N/A'
    else:
        rank = str(i+1)
        print(f'{rank}.{agent},{greater_count} wins, {less_count} losses')
```

```
    Agent Rankings:
--------------------------
1.KAY/O,18 wins, 0 losses
2.Brimstone,17 wins, 0 losses
3.Skye,16 wins, 1 losses
4.Sage,14 wins, 2 losses
5.Omen,13 wins, 3 losses
6.Breach,12 wins, 3 losses
7.Astra,11 wins, 4 losses
8.Fade,10 wins, 5 losses
9.Harbor,8 wins, 6 losses
10.Viper,7 wins, 7 losses
11.Sova,6 wins, 8 losses
12.Raze,5 wins, 8 losses
13.Cypher,3 wins, 9 losses
14.Phoenix,2 wins, 10 losses
15.Jett,1 wins, 14 losses
16.Killjoy,1 wins, 13 losses
17.Neon,1 wins, 12 losses
18.Reyna,1 wins, 11 losses
19.Yoru,1 wins, 12 losses
20.Chamber,0 wins, 19 losses
```

This code assumes that the dataframe df contains the columns Agent1, operator, Agent2, and reject, as well as the rows for each comparison. It also assumes that the reject column indicates whether each comparison was rejected or not, and that the operator column indicates whether the comparison found that agent 1 was greater than (>) or less than (<) agent 2. The code outputs a ranking of the agents based on the number of times they were ranked higher than another agen

```
import pandas as pd

# Create a dictionary to store the rankings
rankings = {}

# Iterate over each row in the DataFrame
for _, row in df1.iterrows():
    # Get the names of the two agents and the comparison operator
    agent1, agent2 = row['Agent1'], row['Agent2']
    operator = row['operator']

    # Update the rankings based on the comparison operator
    if operator == '>':
        rankings[agent1] = rankings.get(agent1, 0) + 1
        rankings[agent2] = rankings.get(agent2, 0) - 1
    elif operator == '<':
        rankings[agent1] = rankings.get(agent1, 0) - 1
        rankings[agent2] = rankings.get(agent2, 0) + 1

# Create a new DataFrame with the rankings
```

```
df2 = pd.DataFrame({'Agent': list(rankings.keys()), 'Points': list(rankings.values())})

# Sort the DataFrame by ranking in descending order
df2 = df2.sort_values('Points', ascending=False)
df2['Rank'] = df2['Points'].rank(ascending=False).astype(int)
df2 = df2[['Rank', 'Agent', 'Points']]
df2 = df2.reset_index(drop=True)
print('Agent Rankings:')
df2
```

```
Agent Rankings:
      Rank      Agent  Points
  0      1      KAY/O      18
  1      2  Brimstone      17
  2      3       Skye      15
  3      4       Sage      12
  4      5       Omen      10
  5      6     Breach       9
  6      7      Astra       7
  7      8       Fade       5
  8      9     Harbor       2
  9     10      Viper       0
 10     11       Sova      -2
 11     12       Raze      -3
 12     13     Cypher      -6
 13     14    Phoenix      -8
 14     15      Reyna     -10
 15     16       Neon     -11
 16     16       Yoru     -11
 17     18    Killjoy     -12
 18     19       Jett     -13
 19     20    Chamber     -19
```

```
# Remove duplicates from the 'Name' column
df4 = df.drop_duplicates(subset=['Name'])
df4 = df4.drop(columns=['Map', 'KD', 'Win', 'Pick', 'AvgScore', 'Matches', 'Kill', 'Death', 'Assist'])
# Merge the 'df2' and 'df3' DataFrames based on the 'Agent' and 'Name' columns, respectively
df2 = pd.merge(df2, df4, left_on='Agent', right_on='Name')
# Drop the 'Name' column and reorder the remaining columns
df2 = df2.drop(columns=['Name'])
df2 = df2[['Rank', 'Agent', 'Role', 'Points']]
df2
```

	Rank	Agent	Role	Points
0	1	KAY/O	initiator	18
1	2	Brimstone	controller	17
2	3	Skye	initiator	15
3	4	Sage	sentinel	12
4	5	Omen	controller	10
5	6	Breach	initiator	9
6	7	Astra	controller	7
7	8	Fade	initiator	5
8	9	Harbor	controller	2
9	10	Viper	controller	0
10	11	Sova	initiator	-2
11	12	Raze	duelist	-3
12	13	Cypher	sentinel	-6
13	14	Phoenix	duelist	-8
14	15	Reyna	duelist	-10
15	16	Neon	duelist	-11
16	16	Yoru	duelist	-11
17	18	Killjoy	sentinel	-12

✓ 0s completed at 2:55 PM