D195 Capstone Project:

A Damage Analysis of Dungeons and Dragons

In [1]:
```
#Importing necessary modules

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patheffects as pe
import seaborn as sns
import statsmodels.api as sm
```

In [2]:
```
# Importing Weapon Data

wpn = pd.read_csv('WeaponData.csv')
```

In [3]:
```
# This calculates the chance to hit an average monster.
# The chance is based on rolling a d20 and adding the
#   Proficiency and Attribute bonus scores
# If the number is equal or greater to the AC, the attack hits

# Proficiency for all Classes is 4 at level 10

Proficiency = 4

# Attribute bonus for the Standard Array stat distribution is 2
# The Attribute bonus is added for both Attack Chance and Damage

Attribute = 2

# Average AC for a Challenge Rating 10 monster is 14

AC = 14
Hit_Chance = ((20 + Proficiency + Attribute) - AC)/ 20

# Advantage is the ability to roll your hit dice twice
#   and take the better value

Advantage = 1 - (Hit_Chance - 1)**2

Hit_Chance, Advantage
```

Out[3]:
```
(0.6, 0.84)
```

In [4]:
```
# This calculates the Critical Strike damage average multiplier
# An attack automatically doubles its damage if the d20 rolls a natural 20

Crit = 1 + (1/20)*2
Crit
```

Out[4]:
```
1.1
```

In [5]:
```
# Calculates the average damage per weapon attack
# Uses the "n(n + 1) / 2n" formula to calculate the average dice roll value
# Then, multiply by the Crit value
# the Attribute bonus
```

```
wpn['Damage'] = Crit * (wpn['Dice Count'] * ((wpn['Dice Type'] * (wpn['Dice Type'] + 1))
```

In [6]:
```
wpn.sort_values(by=['Damage'], ascending = False)
```

Out[6]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
16	Maul	Melee	Martial	Heavy	No	6	2	9.70
11	Greatsword	Melee	Martial	Heavy	No	6	2	9.70
10	Greataxe	Melee	Martial	Heavy	No	12	1	9.15
13	Lance	Melee	Martial	Normal	No	12	1	9.15
15	Longsword 2h	Melee	Martial	Normal	No	10	1	8.05
29	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05
24	Warhammer 2h	Melee	Martial	Normal	No	10	1	8.05
7	Battleaxe 2h	Melee	Martial	Normal	No	10	1	8.05
9	Glaive	Melee	Martial	Heavy	No	10	1	8.05
12	Halberd	Melee	Martial	Heavy	No	10	1	8.05
18	Pike	Melee	Martial	Heavy	No	10	1	8.05
26	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95
23	Warhammer 1h	Melee	Martial	Normal	No	8	1	6.95
22	War Pick	Melee	Martial	Normal	No	8	1	6.95
19	Rapier	Melee	Martial	Normal	No	8	1	6.95
17	Morningstar	Melee	Martial	Normal	No	8	1	6.95
30	Longbow	Ranged	Martial	Normal	No	8	1	6.95
14	Longsword 1h	Melee	Martial	Normal	No	8	1	6.95
8	Flail	Melee	Martial	Normal	No	8	1	6.95
6	Battleaxe 1h	Melee	Martial	Normal	No	8	1	6.95
4	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95
2	Greatclub	Melee	Simple	Normal	No	8	1	6.95
20	Scimitar	Melee	Martial	Light	Yes	6	1	5.85
21	Shortsword	Melee	Martial	Light	Yes	6	1	5.85
3	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85
27	Shortbow	Ranged	Simple	Normal	No	6	1	5.85
28	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85
1	Dagger	Melee	Simple	Light	Yes	4	1	4.75
5	Sickle	Melee	Simple	Light	Yes	4	1	4.75
25	Whip	Melee	Martial	Normal	No	4	1	4.75
0	Club	Melee	Simple	Light	Yes	4	1	4.75

Barbarian Section

In [7]:
```
# Because all Barbarian damage is in the form of melee physical attacks
#   we will start by creating a Dataframe using weapons
```

```
barb_wpn = wpn[wpn['Type'] == 'Melee'].sort_values(by=['Damage'], ignore_index = True, a
```

In [8]:
```
barb_wpn
```

Out[8]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Greatsword	Melee	Martial	Heavy	No	6	2	9.70
1	Maul	Melee	Martial	Heavy	No	6	2	9.70
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15
3	Lance	Melee	Martial	Normal	No	12	1	9.15
4	Longsword 2h	Melee	Martial	Normal	No	10	1	8.05
5	Battleaxe 2h	Melee	Martial	Normal	No	10	1	8.05
6	Pike	Melee	Martial	Heavy	No	10	1	8.05
7	Glaive	Melee	Martial	Heavy	No	10	1	8.05
8	Halberd	Melee	Martial	Heavy	No	10	1	8.05
9	Warhammer 2h	Melee	Martial	Normal	No	10	1	8.05
10	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95
11	Rapier	Melee	Martial	Normal	No	8	1	6.95
12	Morningstar	Melee	Martial	Normal	No	8	1	6.95
13	Greatclub	Melee	Simple	Normal	No	8	1	6.95
14	Warhammer 1h	Melee	Martial	Normal	No	8	1	6.95
15	Longsword 1h	Melee	Martial	Normal	No	8	1	6.95
16	War Pick	Melee	Martial	Normal	No	8	1	6.95
17	Flail	Melee	Martial	Normal	No	8	1	6.95
18	Battleaxe 1h	Melee	Martial	Normal	No	8	1	6.95
19	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85
20	Scimitar	Melee	Martial	Light	Yes	6	1	5.85
21	Shortsword	Melee	Martial	Light	Yes	6	1	5.85
22	Club	Melee	Simple	Light	Yes	4	1	4.75
23	Dagger	Melee	Simple	Light	Yes	4	1	4.75
24	Sickle	Melee	Simple	Light	Yes	4	1	4.75
25	Whip	Melee	Martial	Normal	No	4	1	4.75

In [9]:
```
# Creation of the Barbarian Correlation dataframe
# To be used to capture data for the Linear Regression

barb_corr = pd.DataFrame(columns = {'Complexity Level', 'Damage Output'})
```

Barbarian Modifiers

In [10]:
```
# Class to add values to the Correlation dataframe

def add_to_barb_corr(max_data, complexity_level):
    for i in max_data.max().values:
        new_row = {'Complexity Level' : complexity_level,
```

```
                    'Damage Output' : i}
        barb_corr.loc[len(barb_corr)] = new_row
```

In [11]:
```
# Dual Wielding allows for two attack to occur
# The second attack will not benefit from the Attribute damage bonus

def dw(i):
    i = i * Hit_Chance + (i - Attribute) * Hit_Chance
    return i
```

In [12]:
```
# Rage allows +3 damage per hit

def rage(i):
    i = (i + 3) * Hit_Chance
    return i
```

In [13]:
```
# Reckless Attack allows an attack with Advantage for the first attack in combat

def reckless_attack(i):
    i = (i) * Advantage
    return i
```

In [14]:
```
# Extra Attack allows an additional Attack to occur

def extra_attack(i):
    i = (i * 2) * Hit_Chance
    return i
```

In [15]:
```
# Brutal Critical doubles the Critical Strike bonus

def brutal_critical(i):
    i = (i - Attribute)/Crit
    i = i * (1 + (Crit - 1) * 2) + Attribute * Hit_Chance
    return i
```

In [16]:
```
# Path of the Berserker: Frenzy allows an additional Attack to occur during Rage

def frenzy(i):
    i = (i * 2) * Hit_chance
    return i
```

In [17]:
```
# Path of the Totem Warrior: Totem Spirit: Wolf, allows Advantage during Rage

def wolf(i):
    i = (i) * Advantage
    return i
```

In [18]:
```
# Allows a Heavy Weapon user to take a -5 penalty to hit to add +10 damage

def great_weapon_master(i):
    i = (i * ((20 + Proficiency + Attribute - 5) - AC)/ 20) + 10
    return i
```

In [19]:
```
# Lucky is a Feat that allows an attack with Advantage 3 times during combat

def lucky(i):
    i = i * Advantage
    return i
```

In [20]:
```
# Savage Attacker is a Feat that allows damage dice to be re-rolled
# number between the initial roll and re-roll is taken
```

```
# The average increase in value is approximately 1.2 (based on a study using anydice.com

def savage_attacker(i):
    i = i * 1.2 * Hit_Chance
    return i
```

Barbarian Calculations

In [21]:
```
# Creation of the Baseline damage

barb_0 = barb_wpn.copy()
barb_0['Damage'] = barb_0['Damage'] * Hit_Chance
```

In [22]:
```
barb_0
```

Out[22]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Greatsword	Melee	Martial	Heavy	No	6	2	5.82
1	Maul	Melee	Martial	Heavy	No	6	2	5.82
2	Greataxe	Melee	Martial	Heavy	No	12	1	5.49
3	Lance	Melee	Martial	Normal	No	12	1	5.49
4	Longsword 2h	Melee	Martial	Normal	No	10	1	4.83
5	Battleaxe 2h	Melee	Martial	Normal	No	10	1	4.83
6	Pike	Melee	Martial	Heavy	No	10	1	4.83
7	Glaive	Melee	Martial	Heavy	No	10	1	4.83
8	Halberd	Melee	Martial	Heavy	No	10	1	4.83
9	Warhammer 2h	Melee	Martial	Normal	No	10	1	4.83
10	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	4.17
11	Rapier	Melee	Martial	Normal	No	8	1	4.17
12	Morningstar	Melee	Martial	Normal	No	8	1	4.17
13	Greatclub	Melee	Simple	Normal	No	8	1	4.17
14	Warhammer 1h	Melee	Martial	Normal	No	8	1	4.17
15	Longsword 1h	Melee	Martial	Normal	No	8	1	4.17
16	War Pick	Melee	Martial	Normal	No	8	1	4.17
17	Flail	Melee	Martial	Normal	No	8	1	4.17
18	Battleaxe 1h	Melee	Martial	Normal	No	8	1	4.17
19	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	3.51
20	Scimitar	Melee	Martial	Light	Yes	6	1	3.51
21	Shortsword	Melee	Martial	Light	Yes	6	1	3.51
22	Club	Melee	Simple	Light	Yes	4	1	2.85
23	Dagger	Melee	Simple	Light	Yes	4	1	2.85
24	Sickle	Melee	Simple	Light	Yes	4	1	2.85
25	Whip	Melee	Martial	Normal	No	4	1	2.85

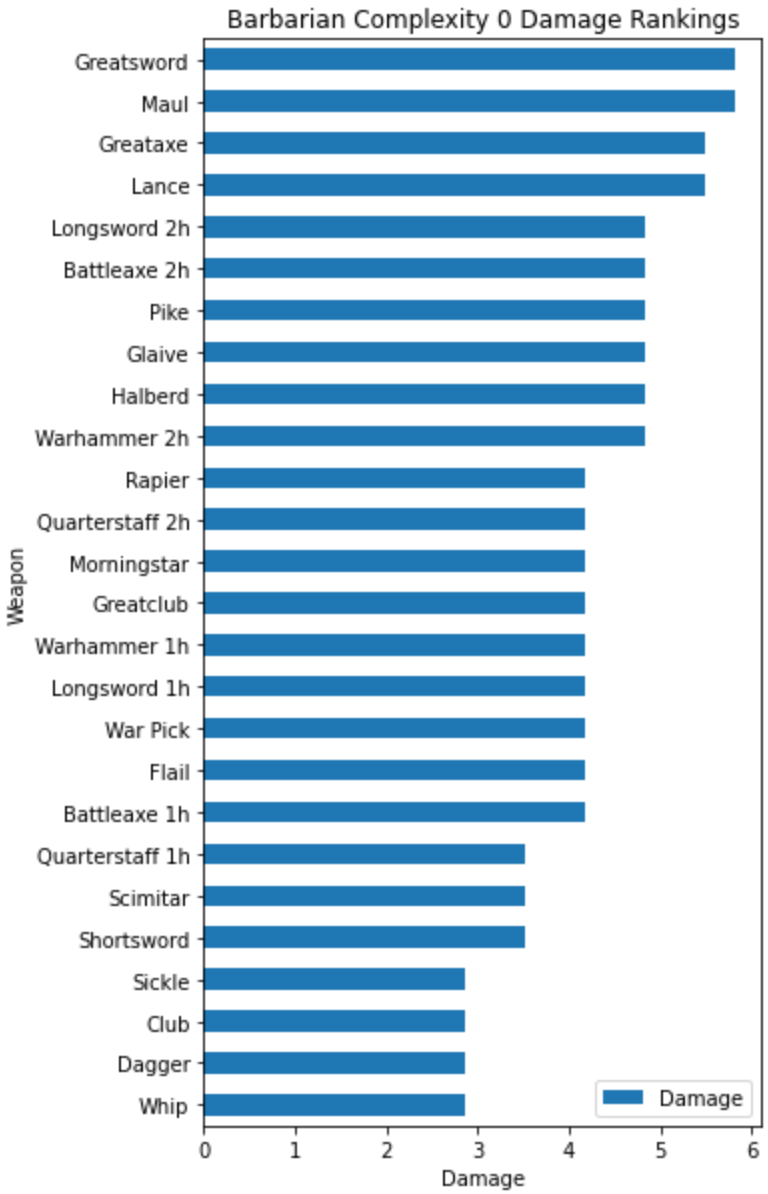
In [23]:
```
barb_0_raw = barb_0
```

In [24]:
```
barb_0_raw
```

Out[24]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Greatsword	Melee	Martial	Heavy	No	6	2	5.82
1	Maul	Melee	Martial	Heavy	No	6	2	5.82
2	Greataxe	Melee	Martial	Heavy	No	12	1	5.49
3	Lance	Melee	Martial	Normal	No	12	1	5.49
4	Longsword 2h	Melee	Martial	Normal	No	10	1	4.83
5	Battleaxe 2h	Melee	Martial	Normal	No	10	1	4.83
6	Pike	Melee	Martial	Heavy	No	10	1	4.83
7	Glaive	Melee	Martial	Heavy	No	10	1	4.83
8	Halberd	Melee	Martial	Heavy	No	10	1	4.83
9	Warhammer 2h	Melee	Martial	Normal	No	10	1	4.83
10	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	4.17
11	Rapier	Melee	Martial	Normal	No	8	1	4.17
12	Morningstar	Melee	Martial	Normal	No	8	1	4.17
13	Greatclub	Melee	Simple	Normal	No	8	1	4.17
14	Warhammer 1h	Melee	Martial	Normal	No	8	1	4.17
15	Longsword 1h	Melee	Martial	Normal	No	8	1	4.17
16	War Pick	Melee	Martial	Normal	No	8	1	4.17
17	Flail	Melee	Martial	Normal	No	8	1	4.17
18	Battleaxe 1h	Melee	Martial	Normal	No	8	1	4.17
19	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	3.51
20	Scimitar	Melee	Martial	Light	Yes	6	1	3.51
21	Shortsword	Melee	Martial	Light	Yes	6	1	3.51
22	Club	Melee	Simple	Light	Yes	4	1	2.85
23	Dagger	Melee	Simple	Light	Yes	4	1	2.85
24	Sickle	Melee	Simple	Light	Yes	4	1	2.85
25	Whip	Melee	Martial	Normal	No	4	1	2.85

In [25]:
```
# Create visual representation of the Complexity 0 Damage Rankings

barb_0_raw.sort_values('Damage').plot(x = 'Name', y = 'Damage', kind = 'barh', figsize =
plt.xlabel('Damage');
plt.ylabel('Weapon');
plt.title('Barbarian Complexity 0 Damage Rankings');
```



In [26]:
```
# Adding the damage values from the Complexity 0 dataframe to the
#   Correlation dataframe
# Because there is only one column, all values have been added

for i in barb_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        barb_corr.loc[len(barb_corr)] = new_row
```

In [27]:
```
barb_corr
```

Out[27]:

	Complexity Level	Damage Output
0	0	5.82
1	0	5.82
2	0	5.49
3	0	5.49
4	0	4.83
5	0	4.83
6	0	4.83
7	0	4.83
8	0	4.83
9	0	4.83
10	0	4.17
11	0	4.17
12	0	4.17
13	0	4.17
14	0	4.17
15	0	4.17
16	0	4.17
17	0	4.17
18	0	4.17
19	0	3.51
20	0	3.51
21	0	3.51
22	0	2.85
23	0	2.85
24	0	2.85
25	0	2.85

In [28]:

```
# Creation of the first complexity level Dataframe

barb_1 = barb_wpn.copy()

# Advantage does not stack, so only one column
#   is added to track the effect of Advantage

barb_1['DW'] = np.where(barb_1['Dual Wield'] == 'Yes',
                        dw(barb_1['Damage']),
                        None)
barb_1['Rage'] = rage(barb_1['Damage'])
barb_1['Advantage'] = lucky(barb_1['Damage'])
barb_1['Extra Attack'] = extra_attack(barb_1['Damage'])
barb_1['Brutal Critical'] = brutal_critical(barb_1['Damage'])
barb_1['Great Weapon Master'] = np.where(barb_1['Weight'] == 'Heavy',
                                         great_weapon_master(barb_1['Damage']),
                                         None)
barb_1['Savage Attacker'] = savage_attacker(barb_1['Damage'])
```

In [29]:
```
barb_1
```

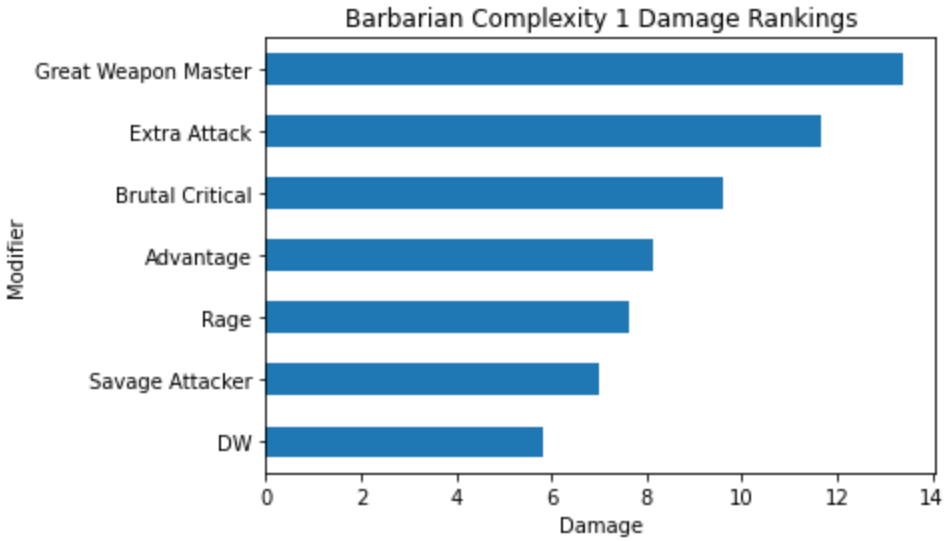
Out[29]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	DW	Rage	Advantage	Extra Attack	Bruta Critica
0	Greatsword	Melee	Martial	Heavy	No	6	2	9.70	None	7.62	8.148	11.64	9.
1	Maul	Melee	Martial	Heavy	No	6	2	9.70	None	7.62	8.148	11.64	9.
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15	None	7.29	7.686	10.98	9.
3	Lance	Melee	Martial	Normal	No	12	1	9.15	None	7.29	7.686	10.98	9.
4	Longsword 2h	Melee	Martial	Normal	No	10	1	8.05	None	6.63	6.762	9.66	7.
5	Battleaxe 2h	Melee	Martial	Normal	No	10	1	8.05	None	6.63	6.762	9.66	7.
6	Pike	Melee	Martial	Heavy	No	10	1	8.05	None	6.63	6.762	9.66	7.
7	Glaive	Melee	Martial	Heavy	No	10	1	8.05	None	6.63	6.762	9.66	7.
8	Halberd	Melee	Martial	Heavy	No	10	1	8.05	None	6.63	6.762	9.66	7.
9	Warhammer 2h	Melee	Martial	Normal	No	10	1	8.05	None	6.63	6.762	9.66	7.
10	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
11	Rapier	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
12	Morningstar	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
13	Greatclub	Melee	Simple	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
14	Warhammer 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
15	Longsword 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
16	War Pick	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
17	Flail	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
18	Battleaxe 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.97	5.838	8.34	6.
19	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85	None	5.31	4.914	7.02	5.
20	Scimitar	Melee	Martial	Light	Yes	6	1	5.85	5.82	5.31	4.914	7.02	5.
21	Shortsword	Melee	Martial	Light	Yes	6	1	5.85	5.82	5.31	4.914	7.02	5.
22	Club	Melee	Simple	Light	Yes	4	1	4.75	4.5	4.65	3.990	5.70	4.
23	Dagger	Melee	Simple	Light	Yes	4	1	4.75	4.5	4.65	3.990	5.70	4.
24	Sickle	Melee	Simple	Light	Yes	4	1	4.75	4.5	4.65	3.990	5.70	4.
25	Whip	Melee	Martial	Normal	No	4	1	4.75	None	4.65	3.990	5.70	4.

In [30]:
```
# The Max() function will show us which ability had the highest damage

barb_1_raw = barb_1.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})
barb_1_raw.max()
```

Out[30]:
```
DW                    5.82
Rage                  7.62
Advantage            8.148
Extra Attack         11.64
Brutal Critical        9.6
Great Weapon Master  13.395
Savage Attacker       6.984
dtype: object
```

In [31]:
```
# Creating a visual representation of the Complexity 1 Damage Rankings

barb_1_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 1 Damage Rankings');
```



In [32]:
```
add_to_barb_corr(barb_1_raw, 1)
```

In [33]:
```
# Because Great Weapon Master is the highest damage output, by far,
#   we will be using it as the basis for the Complexity Level 2 analysis
#   and only use Heavy weapons, which are required for the modifier
# Because Great Weapon Master is not compatible with Dual Wield,
#   we will be omitting it from further analysis

# Creation of Complexity Level 2

barb_2 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [34]:
```
# Defining a new class for Great Weapon Master with Advantage
#   because the calculations include addition in the initial class,
#   which throws off the nested formula

def gwm_advantage(i):
    i = i * ((1 - (20 + Proficiency + Attribute - 5 - AC)/ 20) ** 2) + 10
    return i
```

In [35]:
```
# Iterating through each combination of Great Weapon Master and
#   applicable modifiers
# Because Hit Chance is intrinsically present in the classes
#   it will have to be manually removed

barb_2['GWM + Rage'] = rage(great_weapon_master(barb_2['Damage']))/Hit_Chance
barb_2['GWM + Advantage'] = gwm_advantage(barb_2['Damage'])
barb_2['GWM + Extra Attack'] = extra_attack(great_weapon_master(barb_2['Damage']))/Hit_C
```

```
barb_2['GWM + Brutal Critical'] = brutal_critical(great_weapon_master(barb_2['Damage']))
barb_2['GWM + Savage Attacker'] = savage_attacker(great_weapon_master(barb_2['Damage']))
```

In [36]:
```
barb_2
```

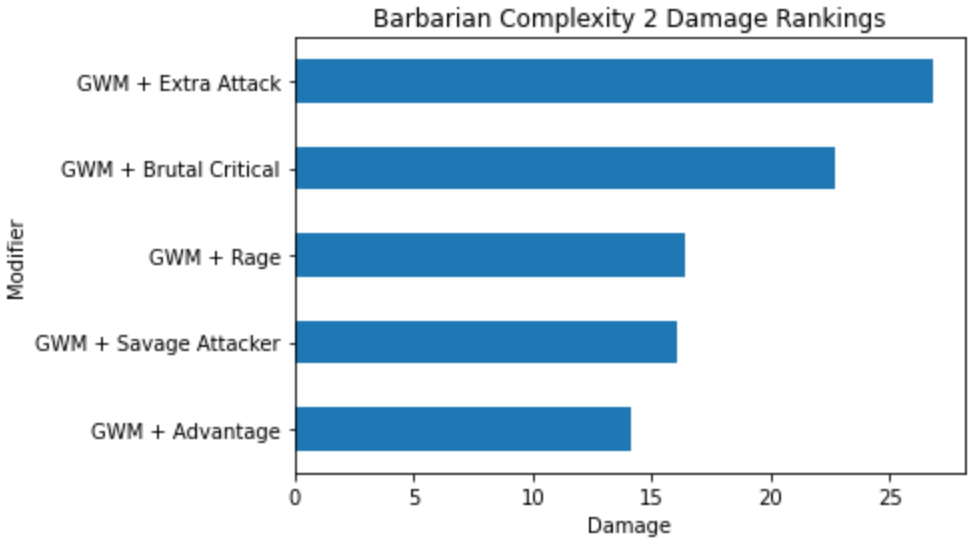
Out[36]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	GWM + Rage	GWM + Advantage	GWM + Extra Attack	GWM + Brutal Critical
0	Greatsword	Melee	Martial	Heavy	No	6	2	9.70	16.3950	14.098250	26.790	22.718182
1	Maul	Melee	Martial	Heavy	No	6	2	9.70	16.3950	14.098250	26.790	22.718182
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15	16.2025	13.865875	26.405	22.368182
6	Pike	Melee	Martial	Heavy	No	10	1	8.05	15.8175	13.401125	25.635	21.668182
7	Glaive	Melee	Martial	Heavy	No	10	1	8.05	15.8175	13.401125	25.635	21.668182
8	Halberd	Melee	Martial	Heavy	No	10	1	8.05	15.8175	13.401125	25.635	21.668182

In [37]:
```
barb_2_raw = barb_2.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})

barb_2_raw.max()
```

Out[37]:
```
GWM + Rage               16.395000
GWM + Advantage          14.098250
GWM + Extra Attack       26.790000
GWM + Brutal Critical    22.718182
GWM + Savage Attacker    16.074000
dtype: float64
```

In [38]:
```
# Creating a visual representation of the Complexity 2 Damage Rankings

barb_2_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 2 Damage Rankings');
```



In [39]:
```
add_to_barb_corr(barb_2_raw, 2)
```

In [40]:
```
# Because Great Weapon Master + Extra Attack is the highest damage output,
#   we will be using it as the basis for the Complexity Level 3 analysis
#   and only use Heavy weapons, which are required for the modifier
```

```
# Creation of Complexity Level 3

barb_3 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [41]:
```
# Iterating through each combination of Great Weapon Master
#   + Extra Attack and applicable modifiers
# Because Hit Chance is intrinsically present in the classes
#   it will have to be manually removed

barb_3['GWM + Extra Attack + Rage'] = rage(extra_attack(great_weapon_master(barb_3['Dama
barb_3['GWM + Extra Attack + Brutal Critical'] = brutal_critical(extra_attack(great_weap
barb_3['GWM + Extra Attack + Savage Attacker'] = savage_attacker(extra_attack(great_weap
barb_3['GWM + Extra Attack + Advantage'] = extra_attack(gwm_advantage(barb_3['Damage']))
```

In [42]:
```
barb_3
```

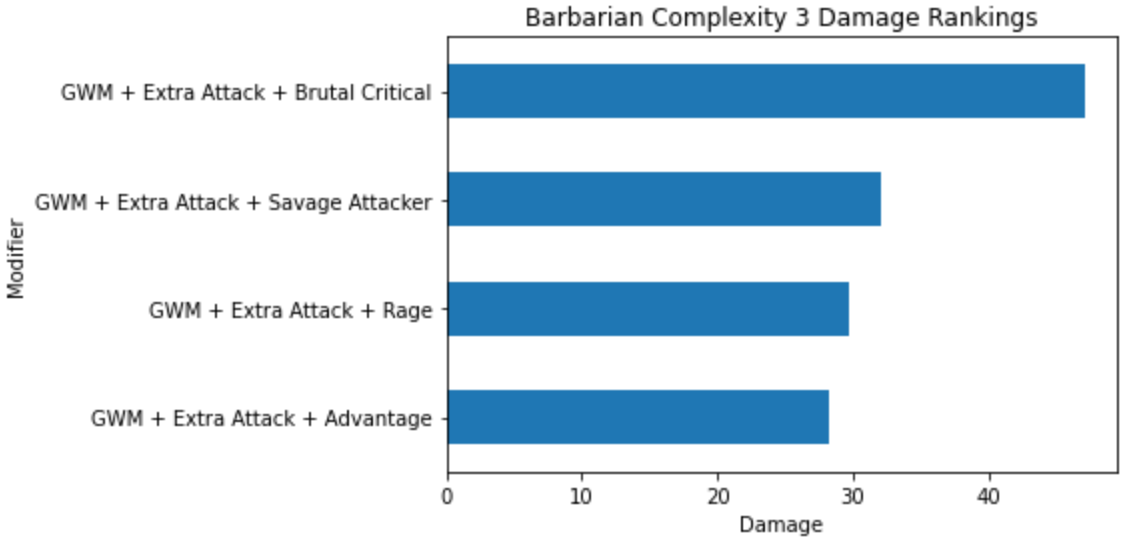
Out[42]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	GWM + Extra Attack + Rage	GWM + Extra Attack + Brutal Critical	GWM + Extra Attack + Savage Attacker	GWM + Extra Attack + Advantage
0	Greatsword	Melee	Martial	Heavy	No	6	2	9.70	29.790	47.072727	32.148	28.19650
1	Maul	Melee	Martial	Heavy	No	6	2	9.70	29.790	47.072727	32.148	28.19650
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15	29.405	46.372727	31.686	27.73175
6	Pike	Melee	Martial	Heavy	No	10	1	8.05	28.635	44.972727	30.762	26.80225
7	Glaive	Melee	Martial	Heavy	No	10	1	8.05	28.635	44.972727	30.762	26.80225
8	Halberd	Melee	Martial	Heavy	No	10	1	8.05	28.635	44.972727	30.762	26.80225

In [43]:
```
barb_3_raw = barb_3.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})

barb_3_raw.max()
```

Out[43]:
```
GWM + Extra Attack + Rage              29.790000
GWM + Extra Attack + Brutal Critical   47.072727
GWM + Extra Attack + Savage Attacker   32.148000
GWM + Extra Attack + Advantage         28.196500
dtype: float64
```

In [44]:
```
# Creating a visual representation of the Complexity 3 Damage Rankings

barb_3_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 3 Damage Rankings');
```



In [45]:
```
add_to_barb_corr(barb_3_raw, 3)
```

In [46]:
```
# Because Great Weapon Master + Extra Attack + Brutal Critical is the highest damage out
#   we will be using it as the basis for the Complexity Level 4 analysis
#   and only use Heavy weapons, which are required for the modifier
# We will also re-visit Rage as an additional modifier, as with complexity level 4,
#   we can add 2 additional attack via Extra Attack and Frenzy (which requires Rage)
# As a note, we will need to add stacking attacks, as they are not multiplicative with e

# Creation of Complexity Level 4

barb_4 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [47]:
```
# Iterating through modifier combinations

barb_4['GWM + Rage + Extra Attack + Frenzy'] = 3 * rage(great_weapon_master(barb_4['Dama
barb_4['GWM + Extra Attack + Brutal Critical + Rage'] = brutal_critical(extra_attack(rag
barb_4['GWM + Extra Attack + Brutal Critical + Advantage'] = brutal_critical(extra_attac
barb_4['GWM + Extra Attack + Brutal Critical + Savage Attacker'] = savage_attacker(extra
```

In [48]:
```
barb_4
```

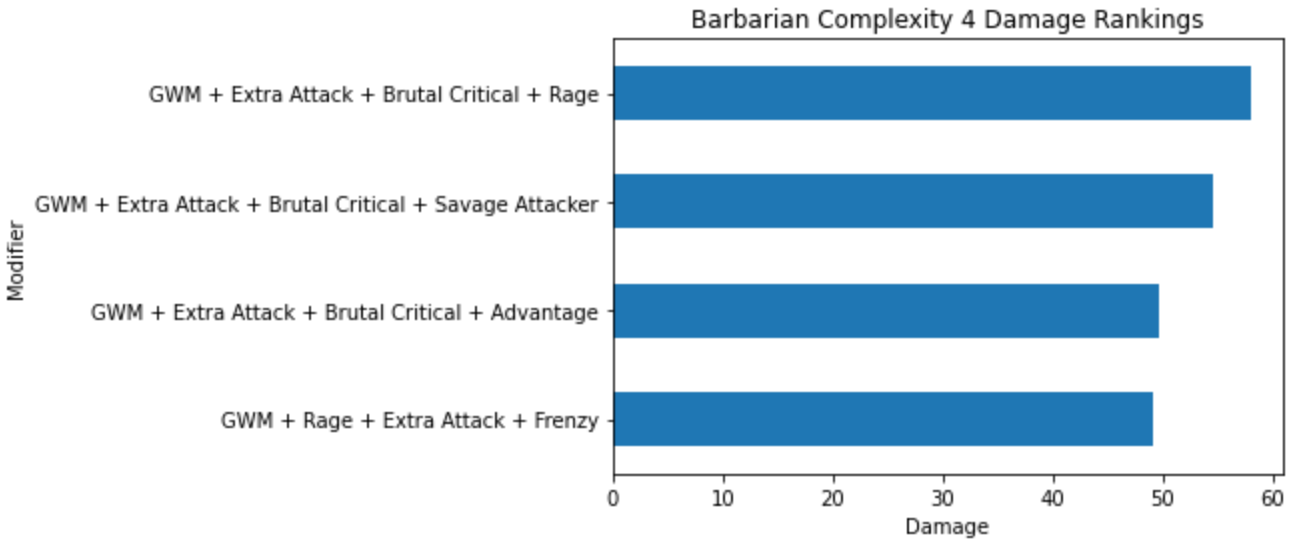
Out[48]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	GWM + Rage + Extra Attack + Frenzy	GWM + Extra Attack + Brutal Critical + Rage	GWM + Extra Attack + Brutal Critical + Advantage	GWM Extr Attack Bruta Critical Savag Attacke
0	Greatsword	Melee	Martial	Heavy	No	6	2	9.70	49.1850	57.981818	49.630	54.52363
1	Maul	Melee	Martial	Heavy	No	6	2	9.70	49.1850	57.981818	49.630	54.52363
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15	48.6075	57.281818	48.785	53.68363
6	Pike	Melee	Martial	Heavy	No	10	1	8.05	47.4525	55.881818	47.095	52.00363
7	Glaive	Melee	Martial	Heavy	No	10	1	8.05	47.4525	55.881818	47.095	52.00363
8	Halberd	Melee	Martial	Heavy	No	10	1	8.05	47.4525	55.881818	47.095	52.00363

In [49]:
```
barb_4_raw = barb_4.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})
ax()
```

Out[49]:
```
GWM + Rage + Extra Attack + Frenzy                    49.185000
GWM + Extra Attack + Brutal Critical + Rage           57.981818
GWM + Extra Attack + Brutal Critical + Advantage      49.630000
GWM + Extra Attack + Brutal Critical + Savage Attacker    54.523636
dtype: float64
```

In [50]:
```
# Creating a visual representation of the Complexity 4 Damage Rankings
barb_4_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 4 Damage Rankings');
```



In [51]:
```
add_to_barb_corr(barb_4_raw, 4)
```

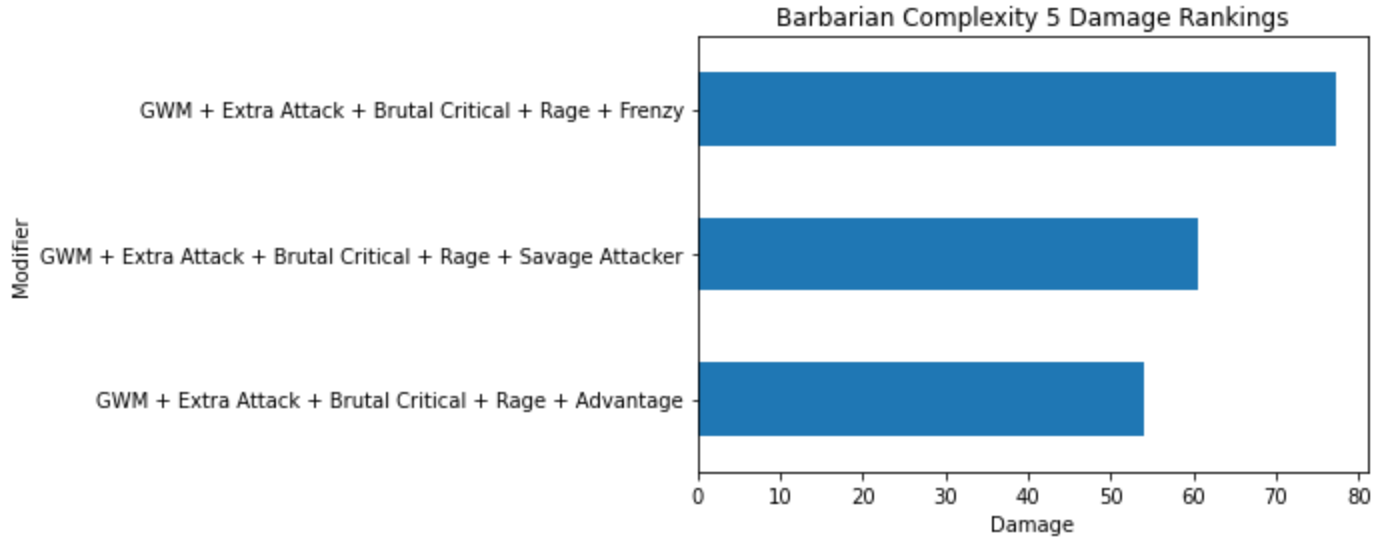
In [52]:
```
# Because Great Weapon Master + Extra Attack + Brutal Critical + Rage is the highest dam
#   we will be using it as the basis for the Complexity Level 5 analysis
#   and only use Heavy weapons, which are required for the modifier

# Creation of Complexity Level 5

barb_5 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [53]:
```
# Iterating through modifier combinations

barb_5['GWM + Extra Attack + Brutal Critical + Rage + Frenzy'] = 3 * rage(brutal_critica
barb_5['GWM + Extra Attack + Brutal Critical + Rage + Savage Attacker'] = extra_attack(r
barb_5['GWM + Extra Attack + Brutal Critical + Rage + Advantage'] = extra_attack(rage(br
```

In [54]:
```
barb_5_raw = barb_5.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})
barb_5_raw.max()
```

Out[54]:
```
GWM + Extra Attack + Brutal Critical + Rage + Frenzy          77.154545
GWM + Extra Attack + Brutal Critical + Rage + Savage Attacker    60.523636
GWM + Extra Attack + Brutal Critical + Rage + Advantage       53.993636
dtype: float64
```

In [55]:
```
# Creating a visual representation of the Complexity 5 Damage Rankings

barb_5_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 5 Damage Rankings');
```



In [56]:
```
add_to_barb_corr(barb_5_raw, 5)
```

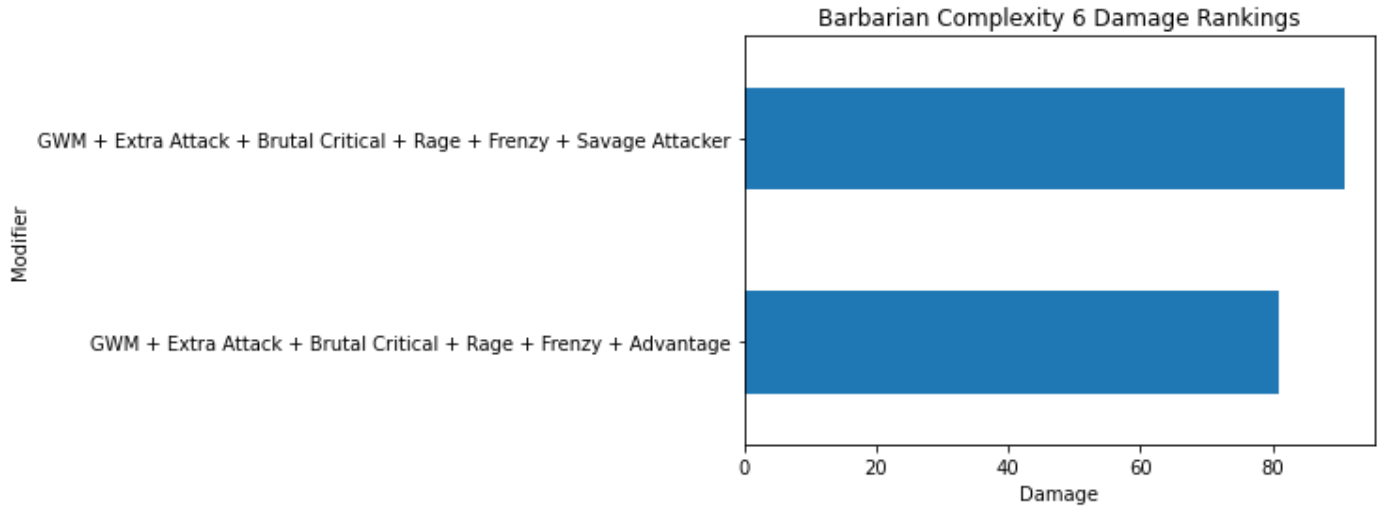
In [57]:
```
# Because Great Weapon Master + Extra Attack + Brutal Critical + Rage + Frenzy is the hi
#   we will be using it as the basis for the Complexity Level 6 analysis
#   and only use Heavy weapons, which are required for the modifier

# Creation of Complexity Level 6

barb_6 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [58]:
```
# Iterating through modifier combinations

barb_6['GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Savage Attacker'] = 3 * r
barb_6['GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Advantage'] = 3 * rage(br
```

In [59]:
```
barb_6_raw = barb_6.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})
barb_6_raw.max()
```

Out[59]:
```
GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Savage Attacker    90.785455
GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Advantage          80.990455
dtype: float64
```

In [60]:
```
# Creating a visual representation of the Complexity 6 Damage Rankings

barb_6_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 6 Damage Rankings');
```



In [61]:
```
add_to_barb_corr(barb_6_raw, 6)
```

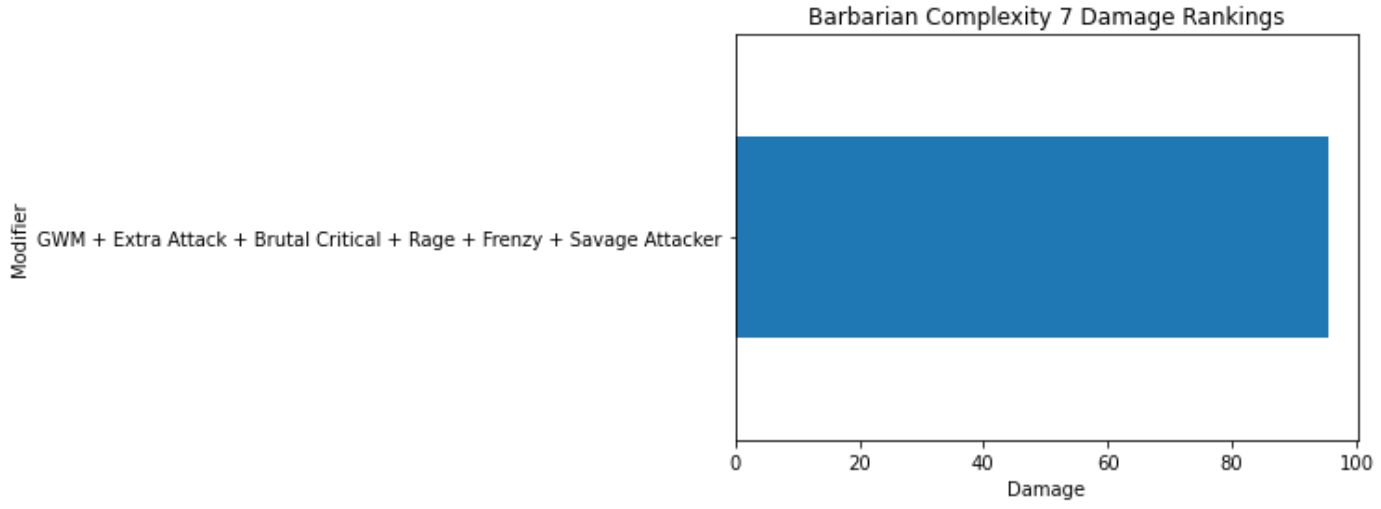
In [62]:
```
# As there is only one more possible combination of modifiers,
#   Complexity Level 7 will add all modifiers together

# Creation of Complexity Level 7

barb_7 = barb_wpn.loc[barb_wpn['Weight'] == 'Heavy'].copy()
```

In [63]:
```
# Creating modifier combo

barb_7['GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Savage Attacker'] = 3 * r
```

In [64]:
```
barb_7_raw = barb_7.drop(columns = {'Name', 'Type', 'Difficulty',
                                    'Weight', 'Dual Wield',
                                    'Dice Type', 'Dice Count',
                                    'Damage'})
barb_7_raw.max()
```

Out[64]:
```
GWM + Extra Attack + Brutal Critical + Rage + Frenzy + Savage Attacker    95.388545
dtype: float64
```

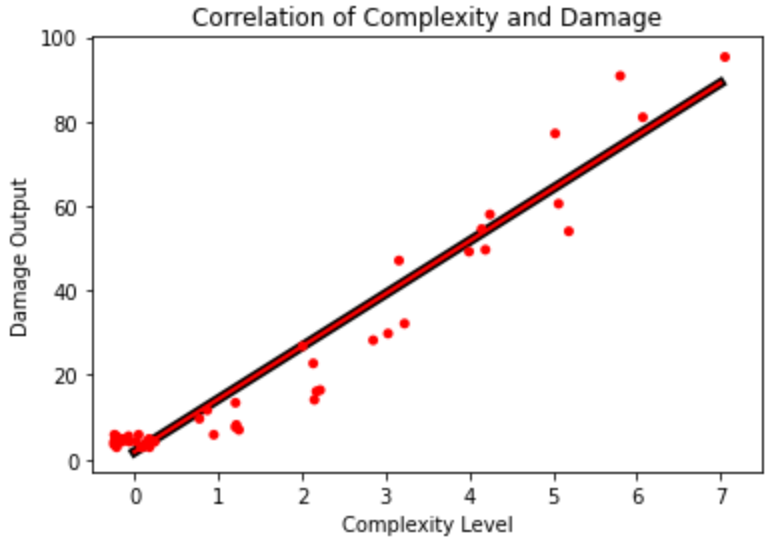
In [65]:
```
# Creating a visual representation of the Complexity 7 Damage Rankings

barb_7_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Barbarian Complexity 7 Damage Rankings');
```



In [66]:
```
add_to_barb_corr(barb_7_raw, 7)
```

In [67]:
```
# Creation of the Regression Line using the Least Squares Regression Algorithm

b, a = np.polyfit(barb_corr['Complexity Level'],
                  barb_corr['Damage Output'],
                  deg = 1)

xseq_barb = np.linspace(0, barb_corr['Complexity Level'].max(), num=100)
```

In [68]:
```
# Creation of the Correlation Plot

plt.plot(xseq_barb, a + b * xseq_barb, linewidth = 2, color = 'red', path_effects=[pe.St
sns.stripplot(data = barb_corr,
              x = 'Complexity Level',
              y = 'Damage Output',
              jitter = 0.25,
              color = 'red');
plt.xlabel('Complexity Level');
plt.ylabel('Damage Output');
plt.title('Correlation of Complexity and Damage');
```



In [69]:
```
# Finding the Correlation Coefficient

print("The Correlation Coefficient is", barb_corr.corr().iloc[0,1])
```

```
The Correlation Coefficient is 0.9736012947496059
```

Ranger Section

In [70]:
```
# Creating the Ranger Weapon dataframe

ranger_wpn = wpn.sort_values(by=['Damage'], ignore_index = True, ascending = False)
```

In [71]:
```
ranger_wpn
```

Out[71]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Maul	Melee	Martial	Heavy	No	6	2	9.70
1	Greatsword	Melee	Martial	Heavy	No	6	2	9.70
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15
3	Lance	Melee	Martial	Normal	No	12	1	9.15
4	Longsword 2h	Melee	Martial	Normal	No	10	1	8.05
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05
6	Warhammer 2h	Melee	Martial	Normal	No	10	1	8.05
7	Battleaxe 2h	Melee	Martial	Normal	No	10	1	8.05
8	Glaive	Melee	Martial	Heavy	No	10	1	8.05
9	Halberd	Melee	Martial	Heavy	No	10	1	8.05
10	Pike	Melee	Martial	Heavy	No	10	1	8.05
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95
12	Warhammer 1h	Melee	Martial	Normal	No	8	1	6.95
13	War Pick	Melee	Martial	Normal	No	8	1	6.95
14	Rapier	Melee	Martial	Normal	No	8	1	6.95
15	Morningstar	Melee	Martial	Normal	No	8	1	6.95
16	Longbow	Ranged	Martial	Normal	No	8	1	6.95
17	Longsword 1h	Melee	Martial	Normal	No	8	1	6.95
18	Flail	Melee	Martial	Normal	No	8	1	6.95
19	Battleaxe 1h	Melee	Martial	Normal	No	8	1	6.95
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95
21	Greatclub	Melee	Simple	Normal	No	8	1	6.95
22	Scimitar	Melee	Martial	Light	Yes	6	1	5.85
23	Shortsword	Melee	Martial	Light	Yes	6	1	5.85
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85
27	Dagger	Melee	Simple	Light	Yes	4	1	4.75
28	Sickle	Melee	Simple	Light	Yes	4	1	4.75
29	Whip	Melee	Martial	Normal	No	4	1	4.75
30	Club	Melee	Simple	Light	Yes	4	1	4.75

In [72]:
```
# Creation of the Ranger Correlation dataframe
# To be used to capture data for the Linear Regression

ranger_corr = pd.DataFrame(columns = {'Complexity Level', 'Damage Output'})
```

Ranger Modifiers

In [73]:
```
# Class to add values ot the Correlation dataframe

def add_to_ranger_corr(max_data, complexity_level):
```

```
    for i in max_data.max().values:
        new_row = {'Complexity Level' : complexity_level,
                   'Damage Output' : i}
        ranger_corr.loc[len(ranger_corr)] = new_row
```

In [74]:
```
# Dual  Wielding allows for two attacks to occur
# The second attack will not benefit from the Attribute damage bonus

def dw(i):
    i = i * Hit_Chance + (i - Attribute) * Hit_Chance
    return i
```

In [75]:
```
# Fighting Style: Archery allows for a +2 bonus to attack rolls,
#   affecting chance to hit

def fs_archery(i):
    i = (i / Hit_Chance) *  (((20 + Proficiency + Attribute + 2) - AC)/ 20)
    return i
```

In [76]:
```
# Fighting Style: Dueling allows for +2 damage on hit,
#   when using one-handed melee weapons

def fs_dueling(i):
    i = (i + 2) * Hit_Chance
    return i
```

In [77]:
```
# Fighting Style: Two-Weapon Fighting allows for the Attribute
#   damage bonus to be applied when dual wielding
# Normally, the Attribute bonus is removed

def fs_two(i):
    i = 2 * (i * Hit_Chance)
    return i
```

In [78]:
```
# Extra Attack allows an additional Attack to occur

def extra_attack(i):
    i = (i * 2) * Hit_Chance
    return i
```

In [79]:
```
# Hunter's Prey: Collossus Slayer allows for an extra 1d8 damage
#   to be added to a successful attack
# This follows the same dice roll average as a normal roll,
#   which adds an average value of 4.5 before Hit Chance

def hp_slayer(i):
    i = (i + 4.5) * Hit_Chance
    return i
```

In [80]:
```
# Hunter's Prey: Horde Breaker allows for an additional attack
#   against another creature
# For damage calculation purposes, this is essentially an additional attack

def hp_breaker(i):
    i = (i * 2) * Hit_Chance
    return i
```

In [81]:
```
# Ranger's Companion: Panther
# The Panther pet can make a Claw attack at 1d6 + 2 damage,
#   where the dice roll average is 3.85 with Crit, and a second
#   Bite attack at 1d4 + 2 damage (If Claw hits),
```

```
#   where the dice roll average is 2.75 with Crit, if a Strength Saving
#   throw is made, which is about a 50% chance
# This is independent of weapon type

def pet_panther():
    i = (5.85 * Hit_Chance) + (4.75 * Hit_Chance * Hit_Chance * 0.5)
    return i
```

In [82]:
```
# Ranger's Companion: Wolf
# The Wolf pet can make a Bite attack at 2d4 + 2 damage,
#   where the dice roll average is 2.75 with Crit, and make the
#   attack with Advantage
# This is independent of weapon type

def pet_wolf():
    i = 7.5 * Advantage
    return i
```

In [83]:
```
# Lightning Arrow replaces a ranged attack and does 4d8
#   damage on a successful hit (where the average dice
#   roll is 4.5) or half damage if missed
# Each surrounding creature also takes 2d8 damage
#   or half damage if a Dexterity Saving Throw is made,
# For the sake of calculations, we are treating the field
#   as if at least 1 additional creature is present

def lightning_arrow(i):
    i = (4.5 * 4 * Hit_Chance) + (4.5 * 4 * (1 - Hit_Chance))
    i = i + (4.5 * 2 * 0.5) + (4.5 * 0.5)
    return (i)
```

In [84]:
```
# Hail of Thorns adds 3d10 damage to a successful ranged
#   attack (where the average dice roll is 5.5),
#   or half damage if a Dexterity Saving Throw is made,
#   which is about a 50% chance

def hail_of_thorns(i):
    i = (i + (16.5) * 0.5 + (8.25) * 0.5) * Hit_Chance
    return i
```

Ranger Calculations

In [85]:
```
# Creation of the Baseline damage

ranger_0 = ranger_wpn.copy()
ranger_0['Damage'] = ranger_0['Damage'] * Hit_Chance
```

In [86]:
```
ranger_0
```

Out[86]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Maul	Melee	Martial	Heavy	No	6	2	5.82
1	Greatsword	Melee	Martial	Heavy	No	6	2	5.82
2	Greataxe	Melee	Martial	Heavy	No	12	1	5.49
3	Lance	Melee	Martial	Normal	No	12	1	5.49
4	Longsword 2h	Melee	Martial	Normal	No	10	1	4.83
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	4.83
6	Warhammer 2h	Melee	Martial	Normal	No	10	1	4.83
7	Battleaxe 2h	Melee	Martial	Normal	No	10	1	4.83
8	Glaive	Melee	Martial	Heavy	No	10	1	4.83
9	Halberd	Melee	Martial	Heavy	No	10	1	4.83
10	Pike	Melee	Martial	Heavy	No	10	1	4.83
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	4.17
12	Warhammer 1h	Melee	Martial	Normal	No	8	1	4.17
13	War Pick	Melee	Martial	Normal	No	8	1	4.17
14	Rapier	Melee	Martial	Normal	No	8	1	4.17
15	Morningstar	Melee	Martial	Normal	No	8	1	4.17
16	Longbow	Ranged	Martial	Normal	No	8	1	4.17
17	Longsword 1h	Melee	Martial	Normal	No	8	1	4.17
18	Flail	Melee	Martial	Normal	No	8	1	4.17
19	Battleaxe 1h	Melee	Martial	Normal	No	8	1	4.17
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	4.17
21	Greatclub	Melee	Simple	Normal	No	8	1	4.17
22	Scimitar	Melee	Martial	Light	Yes	6	1	3.51
23	Shortsword	Melee	Martial	Light	Yes	6	1	3.51
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	3.51
25	Shortbow	Ranged	Simple	Normal	No	6	1	3.51
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	3.51
27	Dagger	Melee	Simple	Light	Yes	4	1	2.85
28	Sickle	Melee	Simple	Light	Yes	4	1	2.85
29	Whip	Melee	Martial	Normal	No	4	1	2.85
30	Club	Melee	Simple	Light	Yes	4	1	2.85

In [87]:
```
ranger_0_raw = ranger_0
```

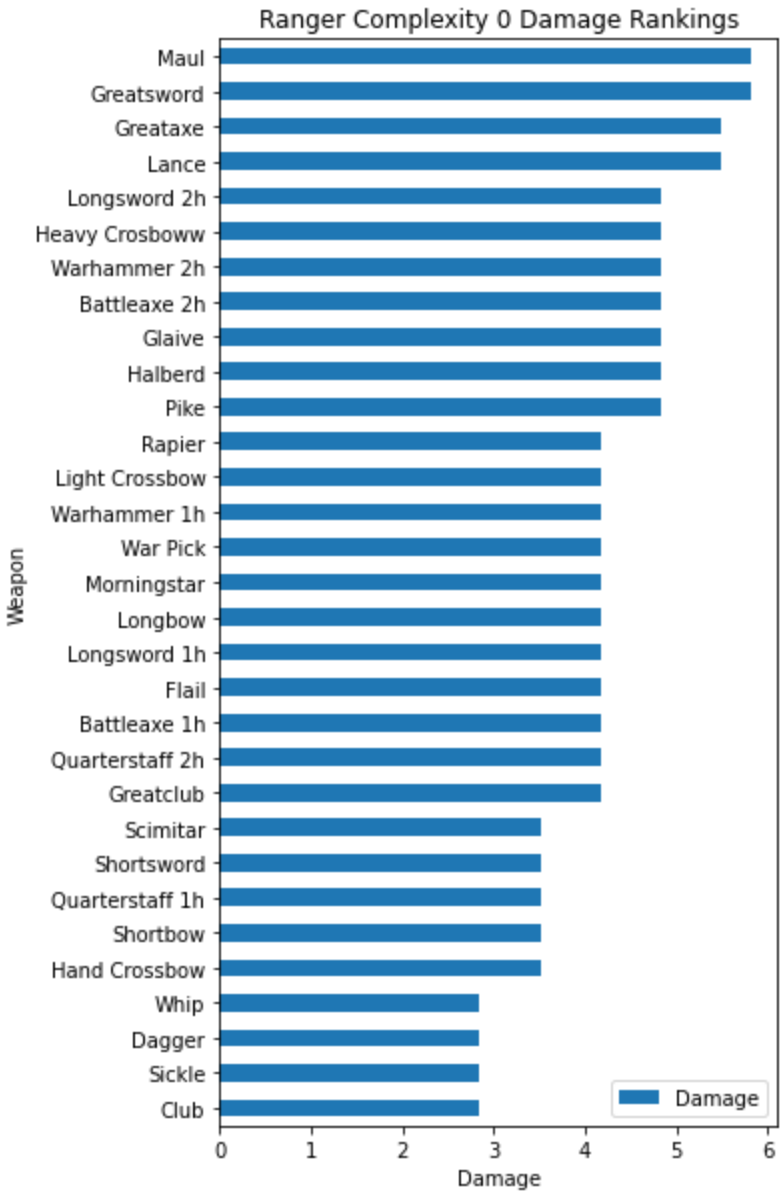
In [88]:
```
ranger_0_raw
```

Out[88]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
0	Maul	Melee	Martial	Heavy	No	6	2	5.82
1	Greatsword	Melee	Martial	Heavy	No	6	2	5.82
2	Greataxe	Melee	Martial	Heavy	No	12	1	5.49
3	Lance	Melee	Martial	Normal	No	12	1	5.49
4	Longsword 2h	Melee	Martial	Normal	No	10	1	4.83
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	4.83
6	Warhammer 2h	Melee	Martial	Normal	No	10	1	4.83
7	Battleaxe 2h	Melee	Martial	Normal	No	10	1	4.83
8	Glaive	Melee	Martial	Heavy	No	10	1	4.83
9	Halberd	Melee	Martial	Heavy	No	10	1	4.83
10	Pike	Melee	Martial	Heavy	No	10	1	4.83
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	4.17
12	Warhammer 1h	Melee	Martial	Normal	No	8	1	4.17
13	War Pick	Melee	Martial	Normal	No	8	1	4.17
14	Rapier	Melee	Martial	Normal	No	8	1	4.17
15	Morningstar	Melee	Martial	Normal	No	8	1	4.17
16	Longbow	Ranged	Martial	Normal	No	8	1	4.17
17	Longsword 1h	Melee	Martial	Normal	No	8	1	4.17
18	Flail	Melee	Martial	Normal	No	8	1	4.17
19	Battleaxe 1h	Melee	Martial	Normal	No	8	1	4.17
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	4.17
21	Greatclub	Melee	Simple	Normal	No	8	1	4.17
22	Scimitar	Melee	Martial	Light	Yes	6	1	3.51
23	Shortsword	Melee	Martial	Light	Yes	6	1	3.51
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	3.51
25	Shortbow	Ranged	Simple	Normal	No	6	1	3.51
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	3.51
27	Dagger	Melee	Simple	Light	Yes	4	1	2.85
28	Sickle	Melee	Simple	Light	Yes	4	1	2.85
29	Whip	Melee	Martial	Normal	No	4	1	2.85
30	Club	Melee	Simple	Light	Yes	4	1	2.85

In [89]:

```
# Create visual representation of the Complexity 0 Damage Rankings

ranger_0_raw.sort_values('Damage').plot(x = 'Name', y = 'Damage', kind = 'barh', figsize
plt.xlabel('Damage');
plt.ylabel('Weapon');
plt.title('Ranger Complexity 0 Damage Rankings');
```



In [90]:
```
# Adding the damage values from the Complexity 0 dataframe to the
#   Correlation dataframe
# Because there is only one column, all values have been added

for i in ranger_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        ranger_corr.loc[len(ranger_corr)] = new_row
```

In [91]:
```
ranger_corr
```

Out[91]:

	Complexity Level	Damage Output
0	0	5.82
1	0	5.82
2	0	5.49
3	0	5.49
4	0	4.83
5	0	4.83
6	0	4.83
7	0	4.83
8	0	4.83
9	0	4.83
10	0	4.83
11	0	4.17
12	0	4.17
13	0	4.17
14	0	4.17
15	0	4.17
16	0	4.17
17	0	4.17
18	0	4.17
19	0	4.17
20	0	4.17
21	0	4.17
22	0	3.51
23	0	3.51
24	0	3.51
25	0	3.51
26	0	3.51
27	0	2.85
28	0	2.85
29	0	2.85
30	0	2.85

In [92]:

```
# Creation of the first complexity level Dataframe
# Because Dual Wielding automatically implies the use of the
#   Two Weapon Fighting Style, we will omit Dual Wielding

ranger_1 = ranger_wpn.copy()

ranger_1['FS Archery'] = np.where(ranger_1['Type'] == 'Ranged',
                                  fs_archery(ranger_1['Damage']),
                                  None)
ranger_1['FS Dueling'] = np.where(np.logical_and(~ranger_1['Name'].str.contains('2h'), r
                                 fs_dueling(ranger_1['Damage']),
```

```
                                None)
ranger_1['FS Two Weapon'] = np.where(ranger_1['Dual Wield'] == 'Yes',
                            fs_two(ranger_1['Damage']),
                            None)
ranger_1['Extra Attack'] = extra_attack(ranger_1['Damage'])
ranger_1['Panther'] = ranger_1['Damage'] * Hit_Chance + pet_panther()
ranger_1['Wolf'] = ranger_1['Damage'] * Hit_Chance + pet_wolf()
ranger_1['Lightning Arrow'] = np.where(ranger_1['Type'] == 'Ranged',
                                lightning_arrow(ranger_1['Damage']),
                                None)
ranger_1['Hail of Thorns'] = np.where(ranger_1['Type'] == 'Ranged',
                                hail_of_thorns(ranger_1['Damage']),
                                None)
ranger_1['Collossus Slayer'] = hp_slayer(ranger_1['Damage'])
ranger_1['Horde Breaker'] = hp_breaker(ranger_1['Damage'])
```

In [93]:

```
ranger_1
```

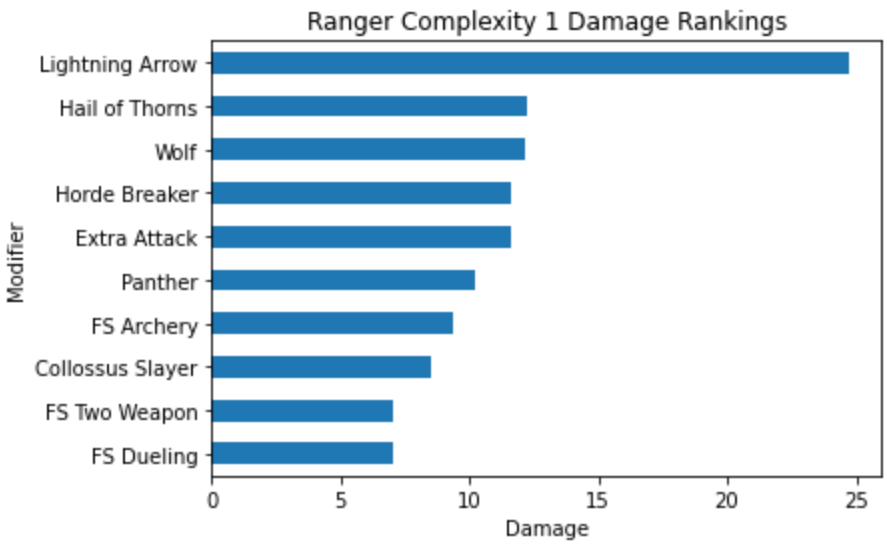
Out[93]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	FS Archery	FS Dueling	FS Two Weapon	Extra Attack	I
0	Maul	Melee	Martial	Heavy	No	6	2	9.70	None	7.02	None	11.64	
1	Greatsword	Melee	Martial	Heavy	No	6	2	9.70	None	7.02	None	11.64	
2	Greataxe	Melee	Martial	Heavy	No	12	1	9.15	None	6.69	None	10.98	
3	Lance	Melee	Martial	Normal	No	12	1	9.15	None	6.69	None	10.98	
4	Longsword 2h	Melee	Martial	Normal	No	10	1	8.05	None	None	None	9.66	
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05	9.391667	None	None	9.66	
6	Warhammer 2h	Melee	Martial	Normal	No	10	1	8.05	None	None	None	9.66	
7	Battleaxe 2h	Melee	Martial	Normal	No	10	1	8.05	None	None	None	9.66	
8	Glaive	Melee	Martial	Heavy	No	10	1	8.05	None	6.03	None	9.66	
9	Halberd	Melee	Martial	Heavy	No	10	1	8.05	None	6.03	None	9.66	
10	Pike	Melee	Martial	Heavy	No	10	1	8.05	None	6.03	None	9.66	
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95	8.108333	None	None	8.34	
12	Warhammer 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
13	War Pick	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
14	Rapier	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
15	Morningstar	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
16	Longbow	Ranged	Martial	Normal	No	8	1	6.95	8.108333	None	None	8.34	
17	Longsword 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
18	Flail	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
19	Battleaxe 1h	Melee	Martial	Normal	No	8	1	6.95	None	5.37	None	8.34	
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95	None	None	None	8.34	
21	Greatclub	Melee	Simple	Normal	No	8	1	6.95	None	5.37	None	8.34	
22	Scimitar	Melee	Martial	Light	Yes	6	1	5.85	None	4.71	7.02	7.02	
23	Shortsword	Melee	Martial	Light	Yes	6	1	5.85	None	4.71	7.02	7.02	
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85	None	4.71	None	7.02	
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85	6.825	None	None	7.02	
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85	6.825	None	7.02	7.02	
27	Dagger	Melee	Simple	Light	Yes	4	1	4.75	None	4.05	5.7	5.70	
28	Sickle	Melee	Simple	Light	Yes	4	1	4.75	None	4.05	5.7	5.70	
29	Whip	Melee	Martial	Normal	No	4	1	4.75	None	4.05	None	5.70	
30	Club	Melee	Simple	Light	Yes	4	1	4.75	None	4.05	5.7	5.70	

In [94]:
```
# The Max() function will show us which ability had the highest damage

ranger_1_raw = ranger_1.drop(columns = {'Name', 'Type', 'Difficulty',
                                        'Weight', 'Dual Wield',
                                        'Dice Type', 'Dice Count',
                                        'Damage'})
ranger_1_raw.max()
```

Out[94]:
```
FS Archery          9.391667
FS Dueling              7.02
FS Two Weapon           7.02
Extra Attack           11.64
Panther               10.185
Wolf                   12.12
Lightning Arrow        24.75
Hail of Thorns        12.255
Collossus Slayer        8.52
Horde Breaker          11.64
dtype: object
```

In [95]:
```
# Creating a visual representation of the Complexity 1 Damage Rankings

ranger_1_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 1 Damage Rankings');
```



In [96]:
```
add_to_ranger_corr(ranger_1_raw, 1)
```

In [97]:
```
# Because Lightning Arrow is the highest damage output, by far,
#   we will be using it as the basis for Complexity Level 2 analysis
# Additionally, we will be omitting FS Dueling and FS Two Weapon,
#   as they overall weaker compared to FS Archery, and they are mutually
#   exclusive with each other
# The same exists for the omission of the Panther, as the Wolf pet
#   provides a better overall damage output and pets are also
#   mutually exclusive
# The same exists for the omossion of Collossus Slayer, as it and
#   Horde Breaker are mutually exclusive as Hunter's Prey skills
# Furthermore we cannot utilize Hail of Thorns and Lightning Arrow together,
#   as they are magical effects and are mutually exclusive

#Creation of Complexity Level 2

ranger_2 = ranger_wpn.loc[ranger_wpn['Type'] == 'Ranged'].copy()
```

In [98]:
```
# Iterating through each combination of Lightning Arrow


ranger_2['L.Arrow + Extra Attack'] = lightning_arrow(ranger_2['Damage']) + (ranger_2['Da
ranger_2['L.Arrow + Horde Breaker'] = lightning_arrow(ranger_2['Damage']) + (ranger_2['D
ranger_2['L.Arrow + Wolf'] = lightning_arrow(ranger_2['Damage']) + pet_wolf()
ranger_2['L.Arrow + FS Archery'] = fs_archery(lightning_arrow(ranger_2['Damage']))
```

In [99]:
```
ranger_2
```

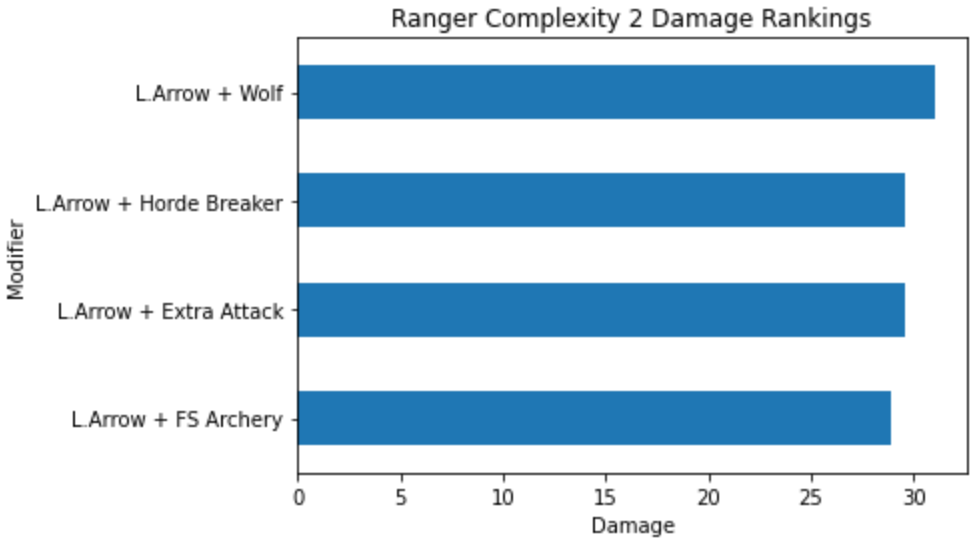
Out[99]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	L.Arrow + Extra Attack	L.Arrow + Horde Breaker	L.Arrow + Wolf	L.Arrow + FS Archery
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05	29.58	29.58	31.05	28.875
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95	28.92	28.92	31.05	28.875
16	Longbow	Ranged	Martial	Normal	No	8	1	6.95	28.92	28.92	31.05	28.875
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85	28.26	28.26	31.05	28.875
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85	28.26	28.26	31.05	28.875

In [100…
```
ranger_2_raw = ranger_2.drop(columns = {'Name', 'Type', 'Difficulty',
                                        'Weight', 'Dual Wield',
                                        'Dice Type', 'Dice Count',
                                        'Damage'})
ranger_2_raw.max()
```

Out[100]:
```
L.Arrow + Extra Attack     29.580
L.Arrow + Horde Breaker    29.580
L.Arrow + Wolf             31.050
L.Arrow + FS Archery       28.875
dtype: float64
```

In [101…
```
# Creating a visual representation of the Complexity 2 Damage Rankings

ranger_2_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 2 Damage Rankings');
```



In [102…
```
add_to_ranger_corr(ranger_2_raw,2)
```

In [103…
```
# Because Lightning Arrow and Wolf were very slightly ahead in damage,
#   we will be using it as the basis for the Complexity Level 3 analysis

# Creation of Complexity Level 3

ranger_3 = ranger_wpn.loc[ranger_wpn['Type'] == 'Ranged'].copy()
```

In [104…
```
# Iterating through each combination of Lightning Arrow
# + Wolf and applicable modifiers

ranger_3['L.Arrow + Wolf + Horde Breaker'] = lightning_arrow(ranger_3['Damage']) + pet_w
ranger_3['L.Arrow + Wolf + Extra Attack'] = lightning_arrow(ranger_3['Damage']) + pet_wo
ranger_3['L.Arrow + Wolf + FS Archery'] = fs_archery(lightning_arrow(ranger_2['Damage'])
```

In [105…
```
ranger_3
```

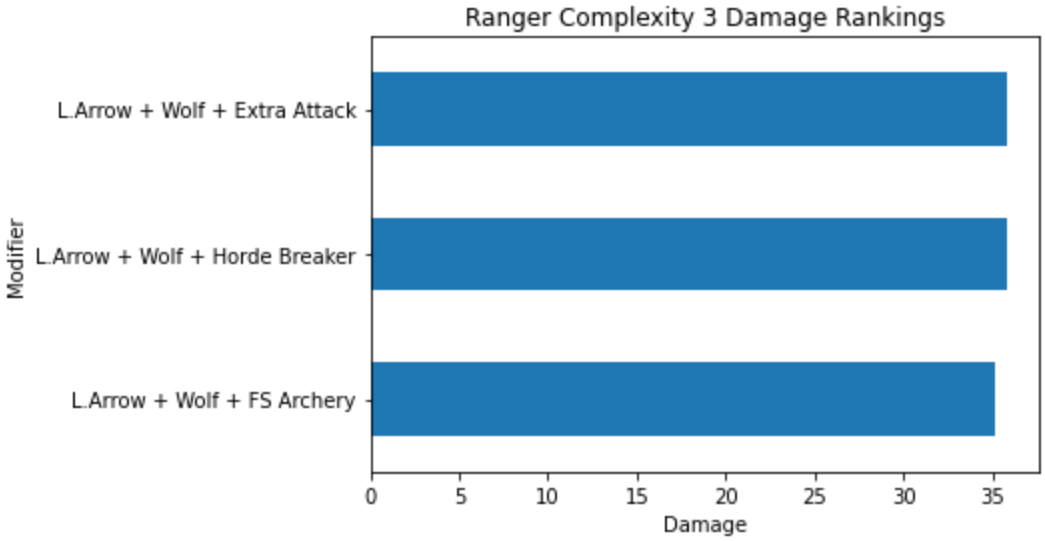
Out[105]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	L.Arrow + Wolf + Horde Breaker	L.Arrow + Wolf + Extra Attack	L.Arrow + Wolf + FS Archery
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05	35.88	35.88	35.175
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95	35.22	35.22	35.175
16	Longbow	Ranged	Martial	Normal	No	8	1	6.95	35.22	35.22	35.175
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85	34.56	34.56	35.175
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85	34.56	34.56	35.175

In [106…
```
ranger_3_raw = ranger_3.drop(columns = {'Name', 'Type', 'Difficulty',
                                        'Weight', 'Dual Wield',
                                        'Dice Type', 'Dice Count',
                                        'Damage'})
ranger_3_raw.max()
```

Out[106]:
```
L.Arrow + Wolf + Horde Breaker    35.880
L.Arrow + Wolf + Extra Attack     35.880
L.Arrow + Wolf + FS Archery       35.175
dtype: float64
```

In [107…
```
# Creating a visual representation of the Complexity 3 Damage Rankings

ranger_3_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 3 Damage Rankings');
```



In [108…
```
add_to_ranger_corr(ranger_3_raw,3)
```

In [109…
```
# Because there was a tie, we will use which is alphabetically first
#   as the basis for Complexity Level 3, which was adding Extra Attack

# Creation of Complexity Level 4

ranger_4 = ranger_wpn.loc[ranger_wpn['Type'] == 'Ranged'].copy()
```

In [110…
```
# Iterating through each combination

ranger_4['L.Arrow + Wolf + Extra Attack + Horde Breaker'] = lightning_arrow(ranger_3['Da
ranger_4['L.Arrow + Wolf + Extra Attack + FS Archery'] = fs_archery(lightning_arrow(rang
```

In [111…
```
ranger_4
```

Out[111]:

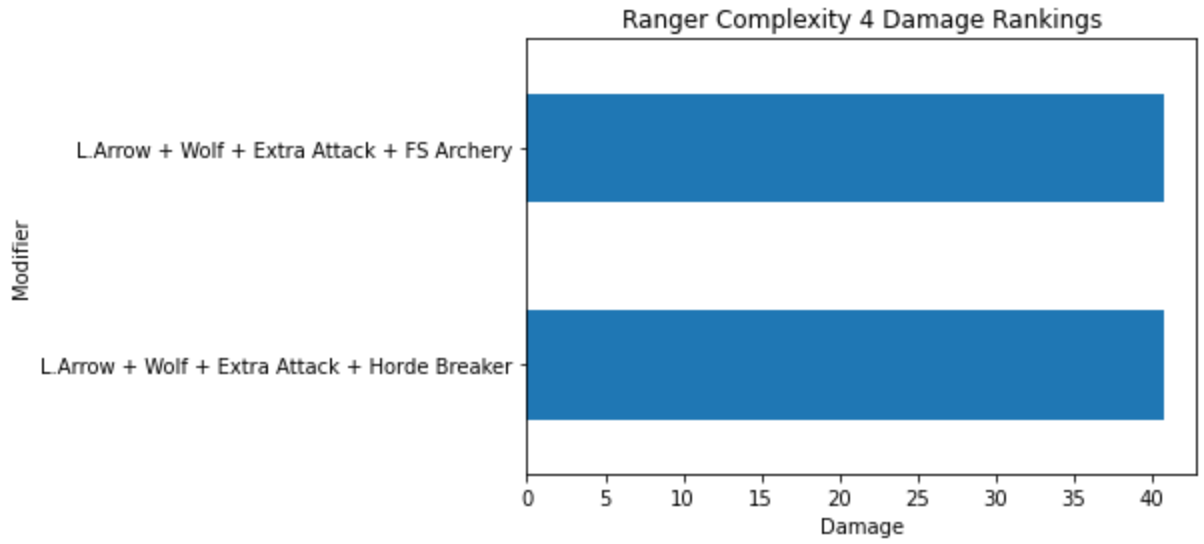
	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage	L.Arrow + Wolf + Extra Attack + Horde Breaker	L.Arrow + Wolf + Extra Attack + FS Archery
5	Heavy Crosboww	Ranged	Martial	Normal	No	10	1	8.05	40.71	40.81
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95	39.39	40.04
16	Longbow	Ranged	Martial	Normal	No	8	1	6.95	39.39	40.04
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85	38.07	39.27
26	Hand Crossbow	Ranged	Martial	Light	Yes	6	1	5.85	38.07	39.27

In [112…
```
ranger_4_raw = ranger_4.drop(columns = {'Name', 'Type', 'Difficulty',
                                        'Weight', 'Dual Wield',
                                        'Dice Type', 'Dice Count',
                                        'Damage'})
ranger_4_raw.max()
```

Out[112]:
```
L.Arrow + Wolf + Extra Attack + Horde Breaker    40.71
L.Arrow + Wolf + Extra Attack + FS Archery       40.81
dtype: float64
```

In [113…
```
# Creating a visual representation of the Complexity 4 Damage Rankings

max().sort_values().plot.barh()
```

```
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 4 Damage Rankings');
```



In [114…
```
add_to_ranger_corr(ranger_4_raw,4)
```

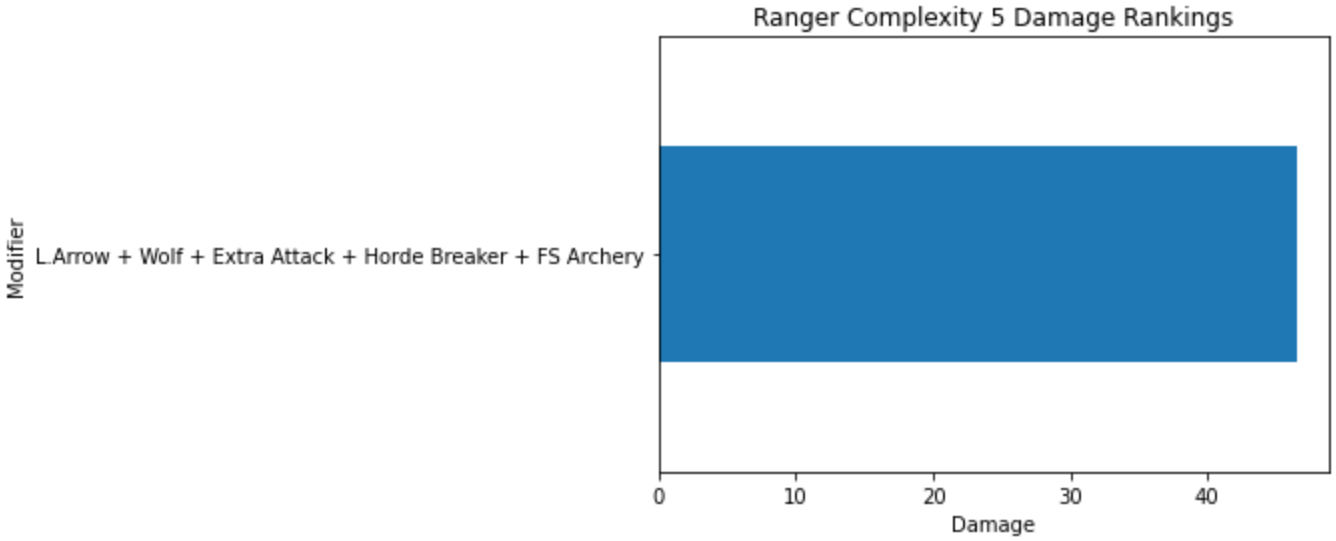
In [115…
```
# As there is only one more possible combination of modifiers,
#   Complexity Level 5 will add all applicable modifiers together

ranger_5 = ranger_wpn.loc[ranger_wpn['Type'] == 'Ranged'].copy()
```

In [116…
```
# Creating modifier combo

ranger_5['L.Arrow + Wolf + Extra Attack + Horde Breaker + FS Archery'] = fs_archery(ligh
```

In [117…
```
ranger_5_raw = ranger_5.drop(columns = {'Name', 'Type', 'Difficulty',
                                        'Weight', 'Dual Wield',
                                        'Dice Type', 'Dice Count',
                                        'Damage'})
ranger_5_raw.max()
```

Out[117]:
```
L.Arrow + Wolf + Extra Attack + Horde Breaker + FS Archery    46.445
dtype: float64
```

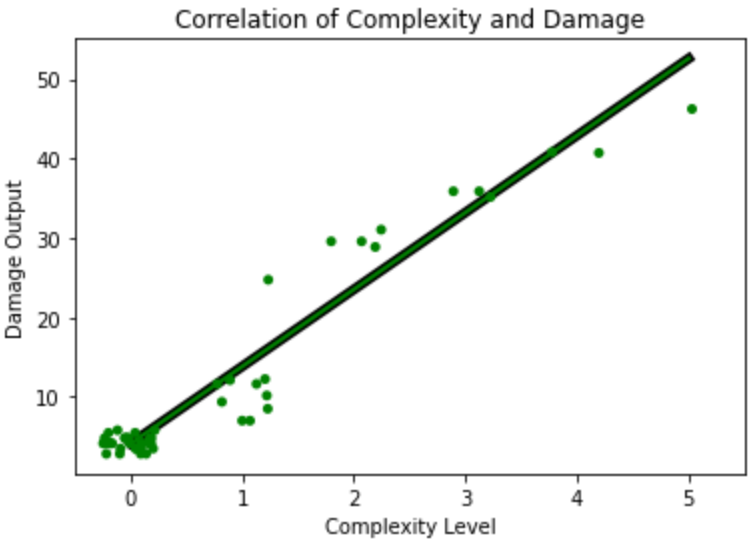
In [118…
```
# Creating a visual representation of the Complexity 5 Damage Rankings

ranger_5_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 5 Damage Rankings');
```



In [119…
```
add_to_ranger_corr(ranger_5_raw,5)
```

In [120…
```
# Creation of the Regression Line using the Least Squares Regression Algorithm

b, a = np.polyfit(ranger_corr['Complexity Level'],
                  ranger_corr['Damage Output'],
                  deg = 1)

xseq_ranger = np.linspace(0, ranger_corr['Complexity Level'].max(), num=100)
```

In [121…
```
# Creation of the Correlation Plot

plt.plot(xseq_ranger, a + b * xseq_ranger, linewidth = 2, color = 'green', path_effects=
sns.stripplot(data = ranger_corr,
              x = 'Complexity Level',
              y = 'Damage Output',
              jitter = 0.25,
              color = 'green');
plt.xlabel('Complexity Level');
plt.ylabel('Damage Output');
plt.title('Correlation of Complexity and Damage');
```



In [122…
```
# Finding the Correlation Coefficient

print("The Correlation Coefficient is", ranger_corr.corr().iloc[0,1])
```

```
The Correlation Coefficient is 0.9656866216013598
```

Warlock Section

In [123…
```
# Creating the Warlock Weapon dataframe

warlock_wpn = wpn.sort_values(by=['Damage'], ignore_index = True, ascending = False)
warlock_wpn = warlock_wpn.loc[warlock_wpn['Difficulty'] == 'Simple']
```

In [124…
```
warlock_wpn
```

Out[124]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95
21	Greatclub	Melee	Simple	Normal	No	8	1	6.95
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85
27	Dagger	Melee	Simple	Light	Yes	4	1	4.75
28	Sickle	Melee	Simple	Light	Yes	4	1	4.75
30	Club	Melee	Simple	Light	Yes	4	1	4.75

In [125…
```
# Creation of the Warlock Correlation dataframe
# To be used to capture data for the Linear Regression

warlock_corr = pd.DataFrame(columns = {'Complexity Level', 'Damage Output'})
```

Warlock Modifiers

In [126…
```
# Class to add values ot the Correlation dataframe

def add_to_warlock_corr(max_data, complexity_level):
    for i in max_data.max().values:
        new_row = {'Complexity Level' : complexity_level,
                   'Damage Output' : i}
        warlock_corr.loc[len(warlock_corr)] = new_row
```

In [127…
```
# Cantrip
# Eldritch Blast deals two separate 1d10 beams
#   with an average dice roll value of 5.5

def eldritch_blast():
    i = (5.5 * Hit_Chance + 5.5 * Hit_Chance) * Crit
    return i
```

In [128…
```
# Cantrip
# Chill Touch deals 2d8 damage with an average dice roll
#   value of 4.5

def chill_touch():
    i = 4.5 * 2 * Hit_Chance * Crit
    return i
```

In [129…
```
# Cantrip
# Poison Spray deals 2d12 poison damage with an average
#   dice roll value of 6.5, but a target can make a
```

```
#   Constitution Saving Throw to avoid damage, which
#   occurs 6 times out of 20
# Does not take Hit Chance into account

def poison_spray():
    i = 6.5 * 2 * Crit * (6 / 20)
    return i
```

In [130…
```
# True Strike allows an attack to occur with Advantage

def true_strike(i):
    i = (i / Hit_Chance) * Advantage
    return i
```

In [131…
```
# Hellish Rebuke deals 6d10 fire damage with an average
#   dice roll value of 5.5, but a target can make a
#   Dexterity Saving Throw to take half damage, which
#   occurs about 6 times out of 20
# Does not take Hit Chance or Crit into account

def hellish_rebuke():
    i = (5.5 * 10 * ( 6 / 20)) + (5.5 * 10 * (14/20))/2
    return i
```

In [132…
```
# Witch Bolt deals 5d12 Lightning Damage with an average
#   dice roll value of 6.5
#   Can be sustained over multiple rounds

def witch_bolt():
    i = 6.5 * 12 * Hit_Chance * Crit
    return i
```

In [133…
```
# Cloud of Daggers deals 10d4 Slashing Damage with an average
#   dice roll value of 2.5
# Does not take Hit Chance into account

def cloud_of_daggers():
    i = 2.5 * 10
    return i
```

In [134…
```
# Shatter deals 6d8 Sonic damage with an average
#   dice roll value of 4.5, but a target can make a
#   Constitution Saving Throw to take half damage, which
#   occurs about 6 times out of 20
# Does not take Hit Chance into account

def shatter():
    i = i = (4.5 * 6 * ( 6 / 20)) + (4.5 * 6 * (14/20))/2
    return i
```

In [135…
```
# Vampiric Touch deals 4d6 Necrotic Damage with an average
#   dice roll value of 3.5
#   Heals for half of the amount, which is effectively a combat
#   swing in favor of the Warlock, from an HP/Damage perspective
#   As such, we will count that life swing as "Damage"
# Can be sustained over multiple rounds

def vampiric_touch():
    i = (3.5 * 6 * Hit_Chance * Crit) + (3.5 * 6 * Hit_Chance * Crit)/2
    return i
```

In [136…
```
# Blight deals 9d8 Necrotic Damage with an average
#   dice roll value of 4.5, but a target can make a
#   Constitution Saving Throw to take half damag,e which
#   occurs about 6 times out of 20
# Does not take Hit Chance into account

def blight():
    i = i = (4.5 * 8 * ( 6 / 20)) + (4.5 * 8 * (14/20))/2
    return i
```

In [137…
```
# Agonizing Blast adds the Ability Modifier to the
#   damage calculations of each of the beams cast in
#   the Eldritch Blast Cantrip

def agonizing_blast(i):
    i = i + Attribute * 2
    return i
```

Warlock Calculations

In [138…
```
# Creation of the Baseline Damage

warlock_0 = warlock_wpn.copy()
warlock_0['Damage'] = warlock_0['Damage'] * Hit_Chance
```

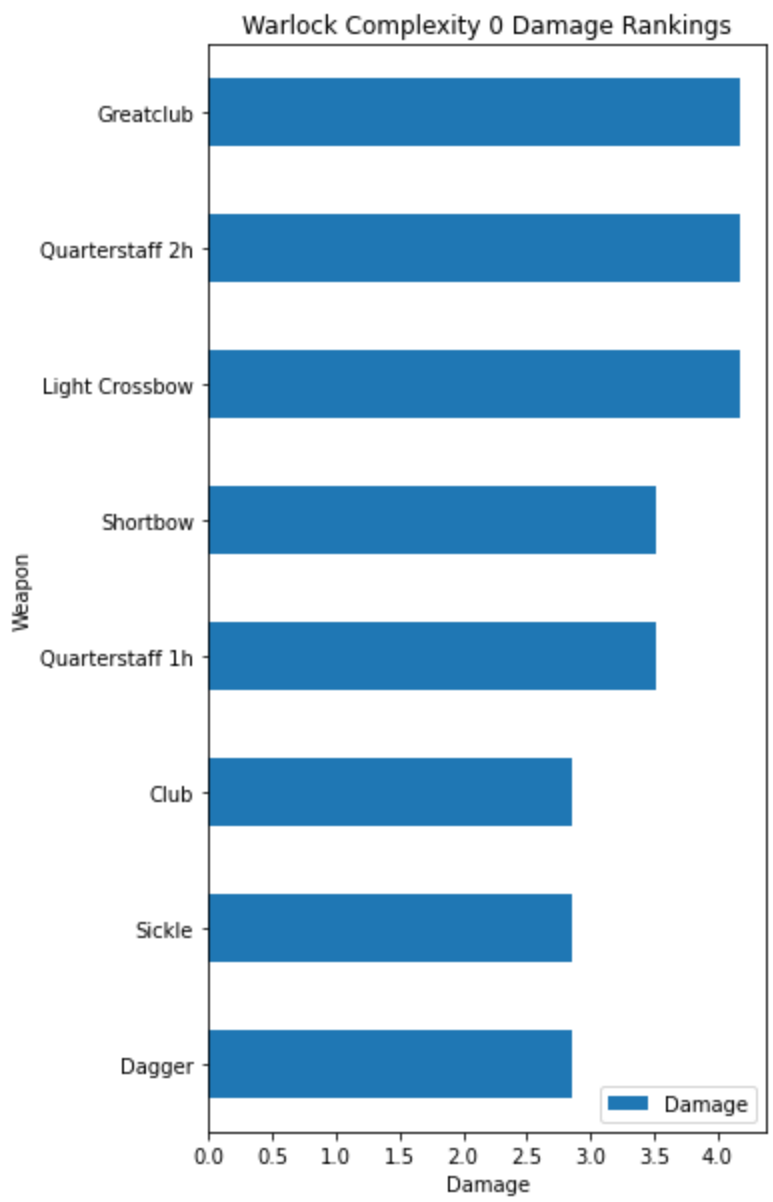
In [139…
```
warlock_0
```

Out[139]:

	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	4.17
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	4.17
21	Greatclub	Melee	Simple	Normal	No	8	1	4.17
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	3.51
25	Shortbow	Ranged	Simple	Normal	No	6	1	3.51
27	Dagger	Melee	Simple	Light	Yes	4	1	2.85
28	Sickle	Melee	Simple	Light	Yes	4	1	2.85
30	Club	Melee	Simple	Light	Yes	4	1	2.85

In [140…
```
warlock_0_raw = warlock_0
```

In [141…
```
# Create visual representation of the Complexity 0 Damage Rankings

warlock_0_raw.sort_values('Damage').plot(x = 'Name', y = 'Damage', kind = 'barh', figsiz
plt.xlabel('Damage');
plt.ylabel('Weapon');
plt.title('Warlock Complexity 0 Damage Rankings');
```



In [142…

```
# Adding the damage values from the Complexity 0 dataframe to the
#   Correlation dataframe
# Because there is only one column, all values have been added

for i in warlock_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        warlock_corr.loc[len(warlock_corr)] = new_row
```

In [143…

```
warlock_corr
```

Out[143]:

	Complexity Level	Damage Output
0	0	4.17
1	0	4.17
2	0	4.17
3	0	3.51
4	0	3.51
5	0	2.85
6	0	2.85
7	0	2.85

In [144…

```
# Creation of the first Complexity Level dataframe

warlock_1 = warlock_wpn.copy()

warlock_1['Cantrip: Eldritch Blast'] = eldritch_blast()
warlock_1['Cantrip: Chill Touch'] = chill_touch()
warlock_1['Cantrip: Poison Spray'] = poison_spray()
warlock_1['True Strike'] = true_strike(warlock_1['Damage'] * Hit_Chance)
warlock_1['Hellish Rebuke'] = hellish_rebuke()
warlock_1['Witch Bolt'] = witch_bolt()
warlock_1['Cloud of Daggers'] = cloud_of_daggers()
warlock_1['Shatter'] = shatter()
warlock_1['Vampiric Touch'] = vampiric_touch()
warlock_1['Blight'] = blight()
```

In [145…

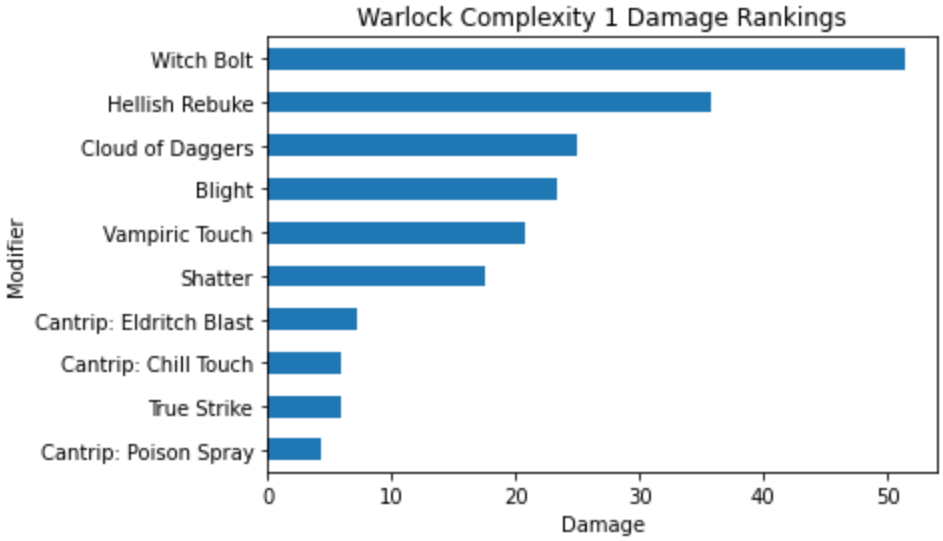
```
# The Max() function will show us which ability had the highest damage

warlock_1_raw = warlock_1.drop(columns = {'Name', 'Type', 'Difficulty',
                                          'Weight', 'Dual Wield',
                                          'Dice Type', 'Dice Count',
                                          'Damage'})
warlock_1_raw.max()
```

Out[145]:

```
Cantrip: Eldritch Blast     7.260
Cantrip: Chill Touch        5.940
Cantrip: Poison Spray       4.290
True Strike                 5.838
Hellish Rebuke             35.750
Witch Bolt                 51.480
Cloud of Daggers           25.000
Shatter                    17.550
Vampiric Touch             20.790
Blight                     23.400
dtype: float64
```

In [146…

```
#Creating a visual representation of the Complexity 1 Damage Rankings

warlock_1_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Warlock Complexity 1 Damage Rankings');
```



In [147…
```
add_to_warlock_corr(warlock_1_raw, 1)
```

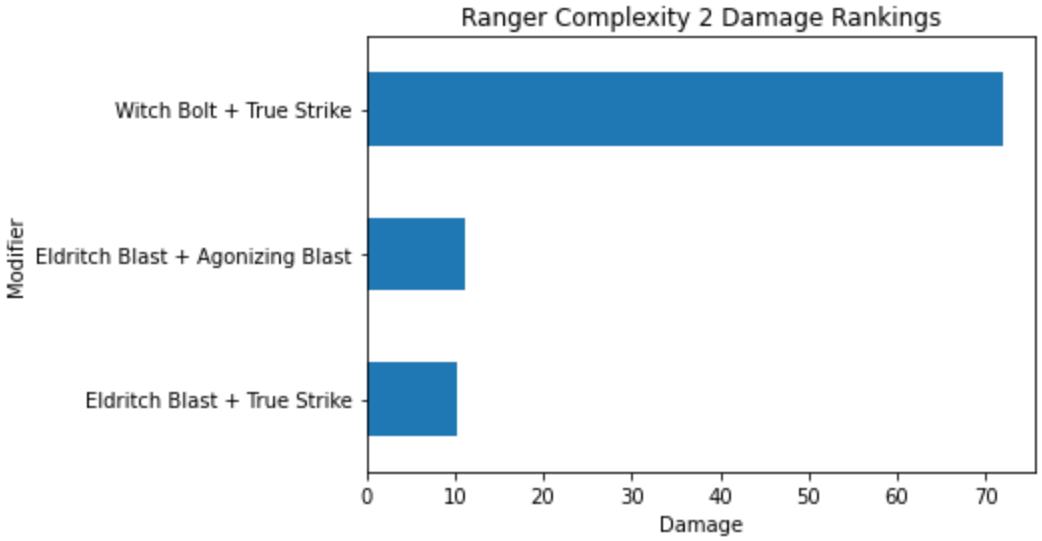
In [148…
```
# Because Witch Bolt was the damage leader by a considerable
#   amount, we will use it as the baseline
# However, Warlocks can only utilize two major spells per
#   engagement, we will also be looking at the "Cantrip" spells
#   and how they are affected by modifiers
# Because Eldritch Blast was the highest damage Cantrip, that
#   is the one we will be using to complete calculations

# Creation of Complexity Level 2

warlock_2 = warlock_wpn.copy()
```

In [149…
```
# Iterating through all modifier combos
# Because most of the Warlock abilities are flat damage,
#   there is very little to iterate on

warlock_2['Witch Bolt + True Strike'] = true_strike(witch_bolt())
warlock_2['Eldritch Blast + True Strike'] = true_strike(eldritch_blast())
warlock_2['Eldritch Blast + Agonizing Blast'] = agonizing_blast(eldritch_blast())
```

In [150…
```
warlock_2_raw = warlock_2.drop(columns = {'Name', 'Type', 'Difficulty',
                                          'Weight', 'Dual Wield',
                                          'Dice Type', 'Dice Count',
                                          'Damage'})
warlock_2_raw.max()
```

Out[150]:
```
Witch Bolt + True Strike           72.072
Eldritch Blast + True Strike       10.164
Eldritch Blast + Agonizing Blast   11.260
dtype: float64
```

In [151…
```
# Creating a visual representation of the Complexity 2 Damage Rankings

warlock_2_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 2 Damage Rankings');
```



In [152…
```
add_to_warlock_corr(warlock_2_raw, 2)
```

In [153…
```
# As there is only one more possibility of modifiers
#   for the cantrips, Complexity Level 3 will add
#   all modifiers together

warlock_3 = warlock_wpn.copy()
```

In [154…
```
warlock_3
```

Out[154]:

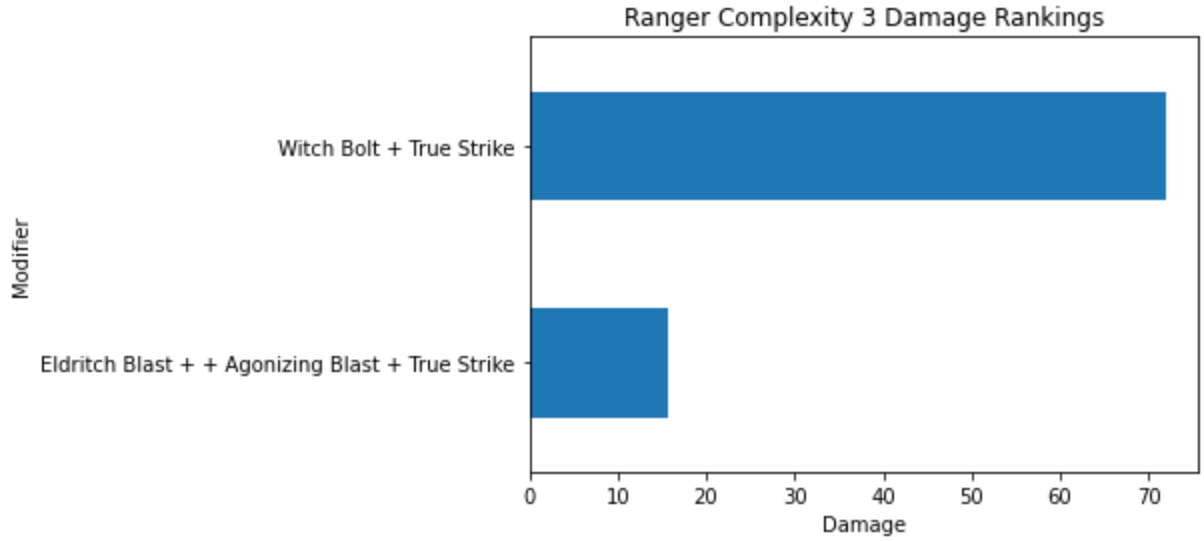
	Name	Type	Difficulty	Weight	Dual Wield	Dice Type	Dice Count	Damage
11	Light Crossbow	Ranged	Simple	Normal	No	8	1	6.95
20	Quarterstaff 2h	Melee	Simple	Normal	No	8	1	6.95
21	Greatclub	Melee	Simple	Normal	No	8	1	6.95
24	Quarterstaff 1h	Melee	Simple	Normal	No	6	1	5.85
25	Shortbow	Ranged	Simple	Normal	No	6	1	5.85
27	Dagger	Melee	Simple	Light	Yes	4	1	4.75
28	Sickle	Melee	Simple	Light	Yes	4	1	4.75
30	Club	Melee	Simple	Light	Yes	4	1	4.75

In [155…
```
warlock_3['Witch Bolt + True Strike'] = true_strike(witch_bolt())
warlock_3['Eldritch Blast + + Agonizing Blast + True Strike'] = true_strike(agonizing_bl
```

In [156…
```
warlock_3_raw = warlock_3.drop(columns = {'Name', 'Type', 'Difficulty',
                                          'Weight', 'Dual Wield',
                                          'Dice Type', 'Dice Count',
                                          'Damage'})
warlock_3_raw.max()
```

Out[156]:
```
Witch Bolt + True Strike                          72.072
Eldritch Blast + + Agonizing Blast + True Strike    15.764
dtype: float64
```

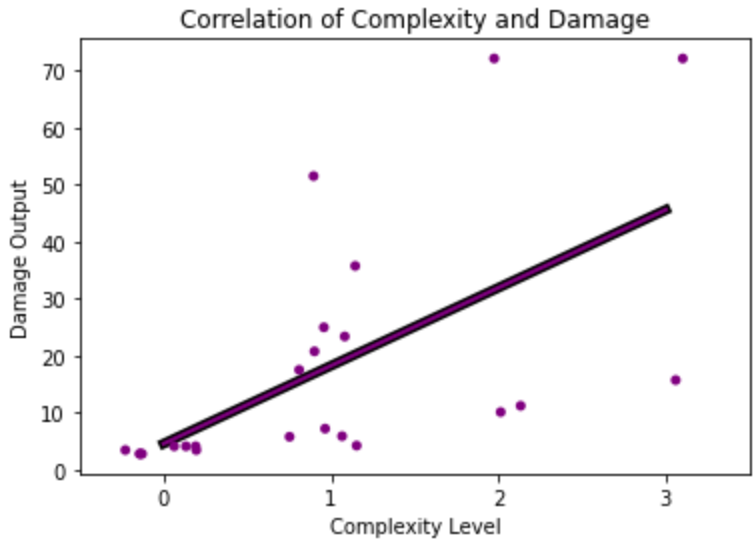
In [157…
```
# Creating a visual representation of the Complexity 2 Damage Rankings

warlock_3_raw.max().sort_values().plot.barh()
plt.xlabel('Damage');
plt.ylabel('Modifier');
plt.title('Ranger Complexity 3 Damage Rankings');
```



In [158…
```
add_to_warlock_corr(warlock_3_raw, 3)
```

In [159…
```
# Creation of the Regression Line using the Least Squares Regression Algorithm

b, a = np.polyfit(warlock_corr['Complexity Level'],
                  warlock_corr['Damage Output'],
                  deg = 1)

xseq_warlock = np.linspace(0, warlock_corr['Complexity Level'].max(), num=100)
```

In [160…
```
# Creation of the Correlation Plot

plt.plot(xseq_warlock, a + b * xseq_warlock, linewidth = 2, color = 'purple', path_effec
sns.stripplot(data = warlock_corr,
              x = 'Complexity Level',
              y = 'Damage Output',
              jitter = 0.25,
              color = 'purple');
plt.xlabel('Complexity Level');
plt.ylabel('Damage Output');
plt.title('Correlation of Complexity and Damage');
```



In [161…
```
# Finding the Correlation Coefficient

print("The Correlation Coefficient is", warlock_corr.corr().iloc[0,1])
```

```
The Correlation Coefficient is 0.6016615568225196
```

Combined Section

In [162…
```
# Creates a combined dataframe for all correlation data

combined_corr = pd.DataFrame(columns = {'Complexity Level', 'Damage Output'})
```

In [163…
```
# Creating Class to add Complexity Level 1+ to the combined dataframe

def add_to_combined_corr(max_data, complexity_level):
    for i in max_data.max().values:
        new_row = {'Complexity Level' : complexity_level,
                   'Damage Output' : i}
        combined_corr.loc[len(combined_corr)] = new_row
```

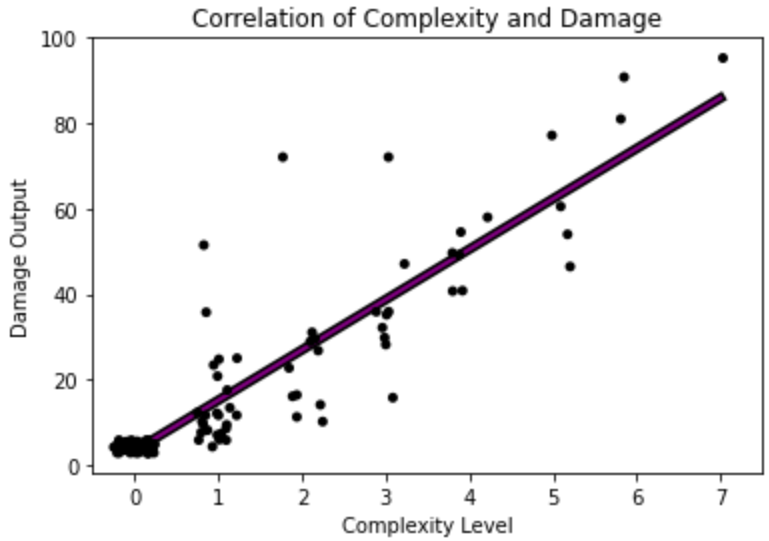
In [164…
```
# Adding the Complexity Level 0 data to the combined dataframe

for i in barb_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        combined_corr.loc[len(combined_corr)] = new_row

for i in ranger_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        combined_corr.loc[len(combined_corr)] = new_row

for i in warlock_0_raw['Damage']:
        new_row = {'Complexity Level' : 0,
                   'Damage Output' : i}
        combined_corr.loc[len(combined_corr)] = new_row
```

In [165…
```
# Adding all Complexity Levels 1+ to the combined dataframe

add_to_combined_corr(barb_1_raw, 1)
add_to_combined_corr(barb_2_raw, 2)
add_to_combined_corr(barb_3_raw, 3)
add_to_combined_corr(barb_4_raw, 4)
add_to_combined_corr(barb_5_raw, 5)
add_to_combined_corr(barb_6_raw, 6)
add_to_combined_corr(barb_7_raw, 7)
add_to_combined_corr(ranger_1_raw, 1)
add_to_combined_corr(ranger_2_raw, 2)
add_to_combined_corr(ranger_3_raw, 3)
add_to_combined_corr(ranger_4_raw, 4)
add_to_combined_corr(ranger_5_raw, 5)
add_to_combined_corr(warlock_1_raw, 1)
add_to_combined_corr(warlock_2_raw, 2)
add_to_combined_corr(warlock_3_raw, 3)
```

In [166…
```
# Creation of the Regression Line using the Least Squares Regression Algorithm

b, a = np.polyfit(combined_corr['Complexity Level'],
                  combined_corr['Damage Output'],
                  deg = 1)

xseq_combined = np.linspace(0, combined_corr['Complexity Level'].max(), num=100)
```

In [167…
```
# Creation of the correlation plot

plt.plot(xseq_combined, a + b * xseq_combined, linewidth = 2, color = 'purple', path_eff
```

```
sns.stripplot( data = combined_corr,
              x = 'Complexity Level',
              y = 'Damage Output',
              jitter = 0.25,
              color = 'black');
plt.xlabel('Complexity Level');
plt.ylabel('Damage Output');
plt.title('Correlation of Complexity and Damage');
```



In [168…

```
# Finding the Correlation Coefficient

print("The Correlation Coefficient is", combined_corr.corr().iloc[0,1])
```

```
The Correlation Coefficient is 0.9085075300321661
```

In [ ]: