Final Project

Douglas Williams

12/8/2022

Overwatch is a competitive first-person shooter video game published by Blizzard Entertainment in 2016. The game has teams of 5 repeatedly fight and "eliminate" eachother to gain and maintain control of some objective area on a playing field. The different playing fields are called "maps", each having unique geometry and objectives. Players can select one of 30+ characters to play. Each character is unique with their own abilities and roles.

One such character is a man named Lucio. He has a low-damage gun that is difficult to aim with and the unique capacity to, at any given time, give teammates within a radius of him healing or increased movement speed. In 2016, when the game released, players used him supportively to provide healing and speed when necessary to teammates and use the radius to stay safe behind the team while still providing value.

In 2018, a gloabl franchised league was created called the Overwatch League, where players from around the world are contracted to play on franchised teams. The year is split into 5 "stages". Stages 1 - 4 are the regular season, with stage 5 being qualifiers and playoffs ending with the grand finals. When you get people together and put millions of dollars on the table, those people started becoming significantly better at whatever activity they are competing in.

From 2018 to 2022, Lucio has gone from a passive supportive character to an aggressive one. The best Lucio players in 2022 take advantage of his speed to move in with the team and assist them in chasing down kills. They also have spent the time to get better at aiming his difficult projectiles to deal more damage. From a teamplay perspective, we have seen teams spend the years focusing on how best to play around the utility Lucio can provide, so they can synchronize plays with assists from their Lucio player. This growth in individual aggression and team coordination can be seen in the data.

In this project, we will be examining how Lucio play has evolved during this professional period. We have 5 seasons of play to analysze: 2018, 2019, 2020, 2021, and 2022. Each with 5 stages of data. This is a daunting task, so we begin by looking at what percentage of time Lucio was played on each map during each stage to identify where to zoom in.

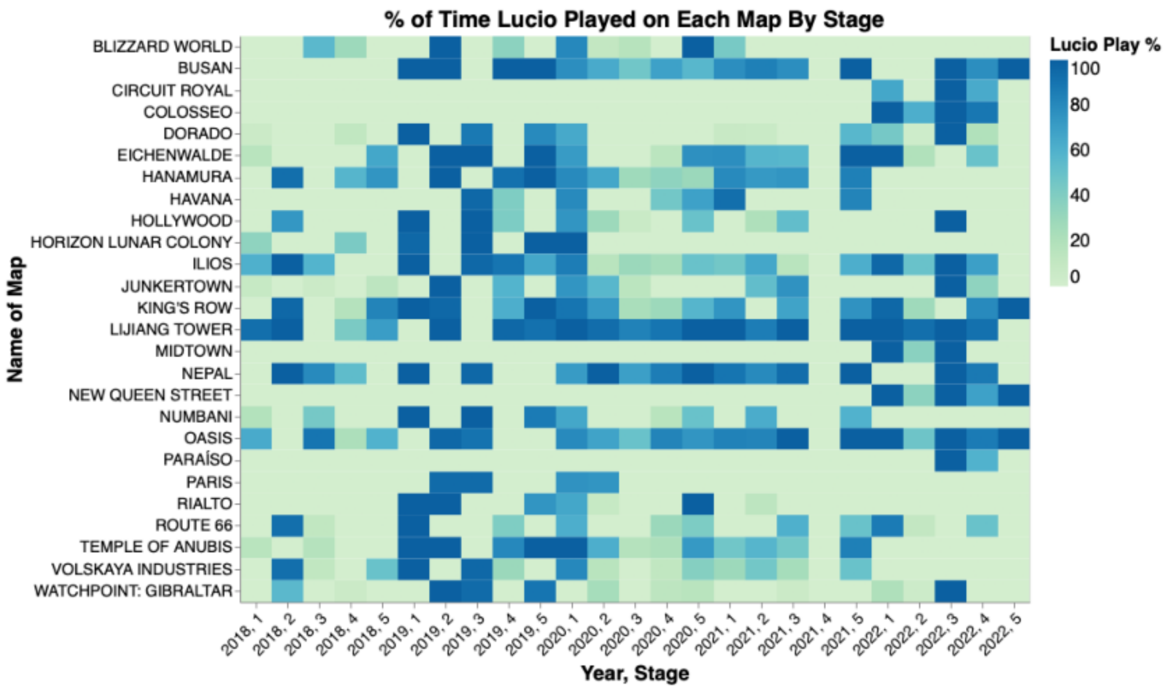


Figure 1: Percentage of Times Lucio was played on Each Map By Stage. We see Lucio played sparatically in 2018 across certain maps. But then in 2019 there is a surge in dark blue, indicating much more Lucio play across many different maps. From 2019 onward we see horizontal stripes of blue develop, showing the maps where Lucio was determined to be a good strategic pick as understanding of the optimal playstyle grew.

Zooming out we can glean insight from the horizontal and vertical stripes of blue. Horizontal stripes indicate maps where professional play has decided Lucio is a good strategic fit. Vertical stripes of blue indicate stages with large amounts of Lucio play due to the overarching best strategy for that stage including him as a key character. If we hope to understand how Lucio has evolved over the years, we can pick the stage from each year where Lucio was the most prevalent and compare those. We would expect Lucio play to evolve during these periods of high play, as evolution when Lucio is important to the dominant strategy leads to wins and money.

In 2018, Lucio was played on a splattering of maps quite a lot during stage 2.

Similarly, we see a gargantuan amount of lucio play in 2019. This is for reasons beyond the scope of this project but needless to say there was a lot of Lucio play and refinement. Stage 2 of 2019 had the highest amount of Lucio play so we will use it for this analysis.

Lucio saw a good amount of play across most maps in stage 1 of 2020, then fell off in use for the remainder of the season save a few key maps. The same pattern can be seen in 2021

Lastly, Lucio becomes a dominant pick in stage 3 of 2022.

For most of the folowing analysis we will be comparing stage 2 of 2018 and stage 3 of 2022 to get the best of Lucio play in each year. We will, however, look at minute shifts across time later on.

Let's start with this idea of individual aggression. We can measure this via the damage and eliminations (elims) a Lucio secures over the course of a map. To make this a fairer comparison across matches and years, we can normalize the damage and elims by dividing by the total amount of damage and elims earned by the Lucio player's team. This gives us what percentage of the team's damage and eliminations the Lucio player contributed to in a given map of play.

This graph sends a very clear message. We expect more aggressive Lucios to be in the top right and more passive in the bottom left. 2018 Lucios had a mean of about 8% and 27.7% of team damage and eliminations respectively. Compare this to 2022, with means of 12.2% and 55.7% of damage and eliminations. It is clear that 2022 Lucios are dealing a lot more damage and contributing many more of the team's eliminations than 2018 Lucios.

So it is clear that stylistically, Lucio play has become more aggressive. But is this reflected in their skill as well? Are they able to accomplish more without consequence or are they running headlong into the enemy? To explore this, let's take a peek at the percentage of team eliminations Lucio contributed to against the number of deaths they had that map. If Lucio players are more aggressive but not necessarily more skilled, we would see deaths rise in 2022. If the opposite were true, we should see deaths remain the same, indicating they are getting more aggressive value while avoiding punishment.

As we saw in the prior figure, 2022 Lucios contribute significantly more to their team's eliminations. Interestingly, along the y-axis we see the distribution of deaths looking approximately the same. We can use the horizonatal mean lines to investigate this further, and we see that

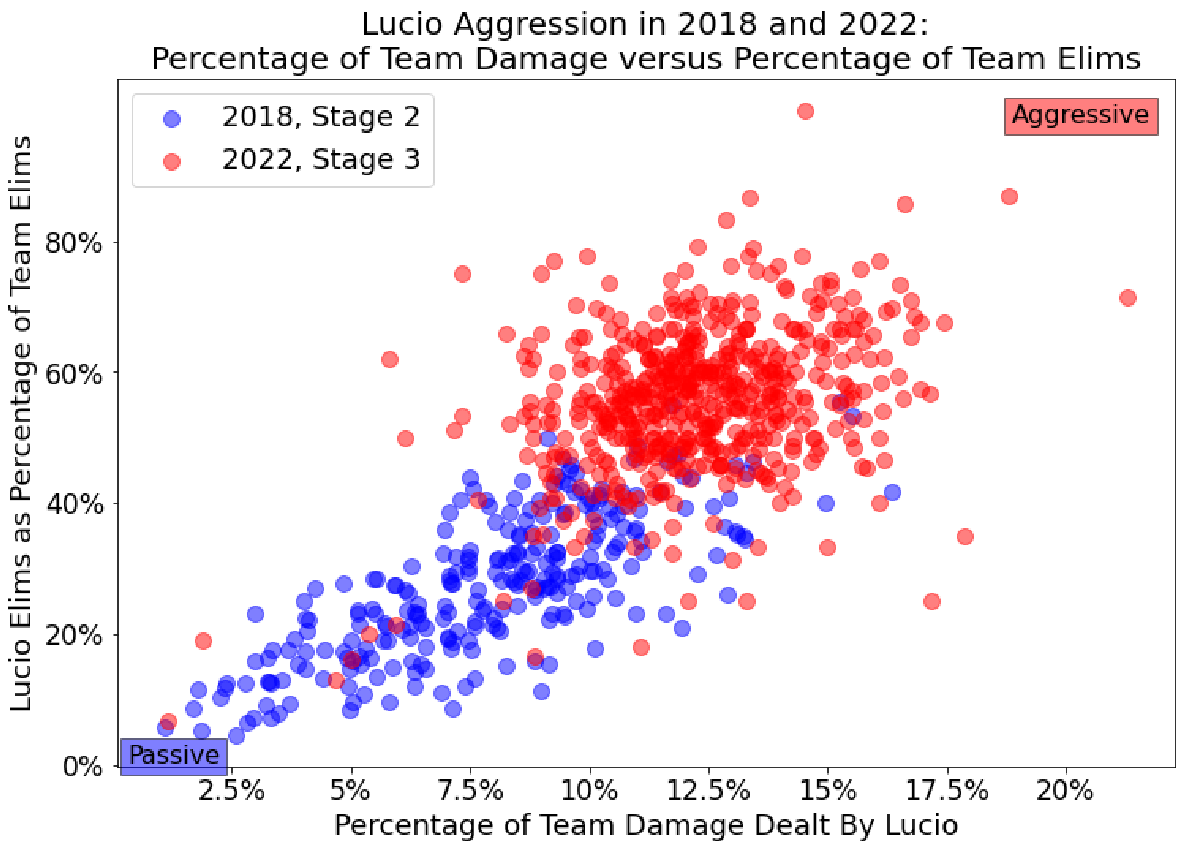


Figure 2: Percentage of Team Damage versus Percentage of Team Eliminations by Lucio in 2018 and 2022. Each dot represents a Lucio player over one map of play. Blue dots are from stage 2 of 2018 and red dots stage 3 of 2022. Dots in the top right indicate aggressive Lucios contributing to a larger percentage of team damage and eliminations while lower left dots are more passive. Despite variance in both years the 2022 players are clearly clustered up and to the right while 2018 players lie along the horizontal on the lower left.

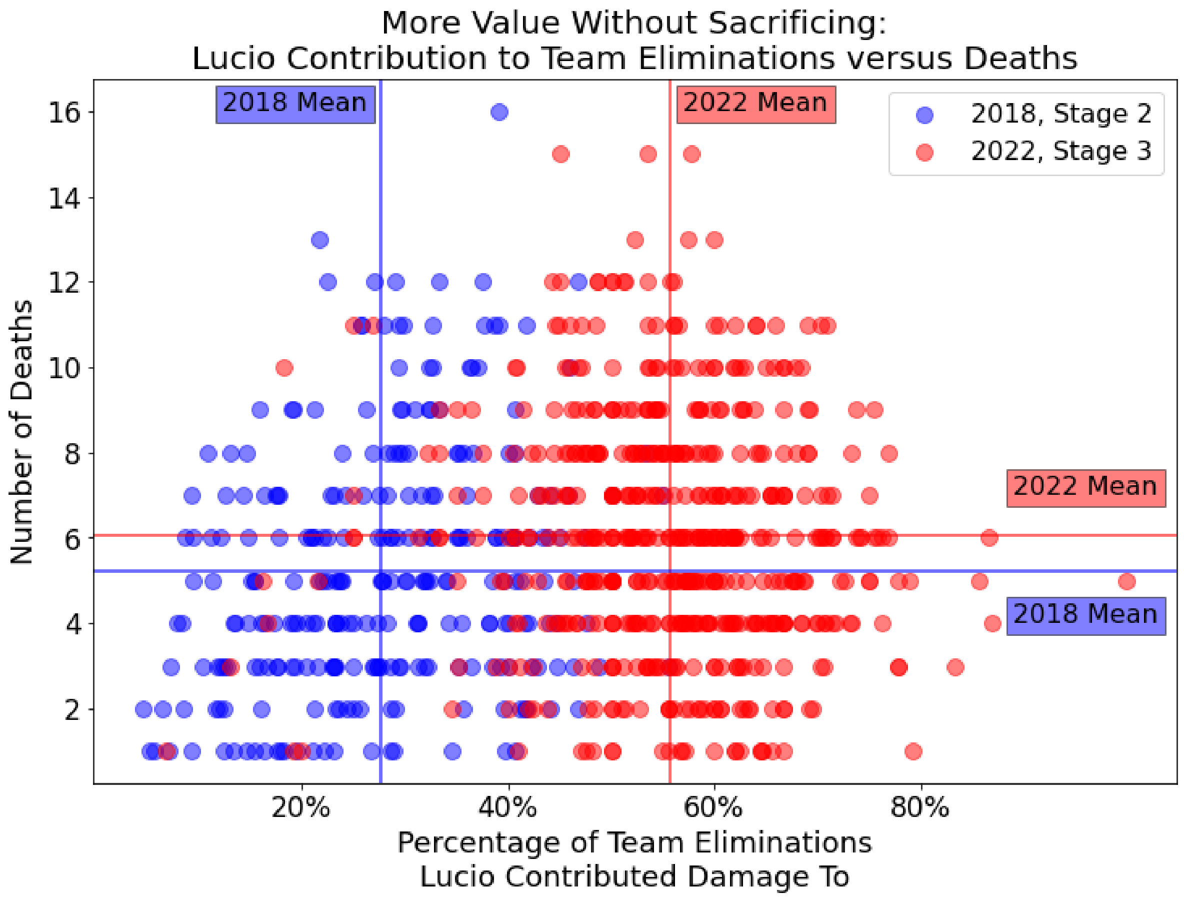


Figure 3: Lucio Contribution to Team Eliminations versus Deaths. The x-axis plots Percentage of Team Eliminations Lucio Contributed to against a y-axis of number of times the Lucio player died that map. Blue is data from stage 1 of 2018 and red is from stage 3 of 2022. The vertical lines show 2022 Lucios contributing to a higher percentage of their team's eliminations on average. While the horizontal lines show 2022 Lucios dying slightly more on average than 2018 Lucios.

2022 Lucios do die more often than 2018 Lucios, with an average of 6.051 compared to 5.235 deaths per map respectively.

That being said, the gap is less than 1 death per map while contributing to about double the proportion of eliminations. This dramatic imbalance between increase in deaths and increase in aggression shows that 2022 Lucios are not only more aggressive, but more skilled. Able to weave in-and-out to deal damage while avoiding punishment.

So we can definitely see that Lucios have improved in terms of individual aggression, but what about their supportive contribution to the team? Yes they deal more damage but what makes Lucio unique and valuable is his ability to heal and speed teammates at crucial moments.

We can measure each Lucio's offensive assists per map to get a sense of their contribution. In order to gain an offensive assist, Lucio must provide some offensive utility, either speed or displacement of an enemy, shortly before that enemy is eliminated by a teammate. To normalize this stat for comparison across years, we can divide it by the total number of eliminations. This gives us the percentage of the team's eliminations that Lucio gave an offensive assist to.

This stat is particularly powerful because it not only measures the supportive ability of the Lucio player, but also the overall team cohesion. In order to get the most value out of Lucio support, a team needs to coordinate plays around the Lucio's utility. Because of this added insight, let's look at the stages where Lucio was most prevalent in across all years of professional play.

```
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/163065500.py:55: UserWarning
  ax_objs[-1].set_xticklabels(x_labels)
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/163065500.py:71: UserWarning
  plt.tight_layout()
```

This last plot brings the evolution of play full circle. In 2018, there was a wide range of team assist contributions that ranged between just above 0% to around 12% with a mean of 6.5%. Then we see a dramatic jump in 2019 to a distribution with a mean of 12.7%. This doubling is likely the dramatic affect of the first season of professional play, with teams spending a year scrambling to define and work towards optimal play. But the trend doesn't stop there. In 2022, we see a distribution centered at a mean of 17.4%. The left tail in 2022 is dramatically sparser, showing that even "lower-end" Lucio matches were significantly higher than in previous years.

This confirms that individual Lucio players have grown to extract more inividual and team value out of their play without a substantial jump in punishment metrics like deaths. We also can see that overall team coordination around Lucio utility has skyrocketed as well. The leap from 2018 to 2019 shows that the professional league incentivizing refinement of individual and team play has had a powerful impact on the skill and coordination of professional players.

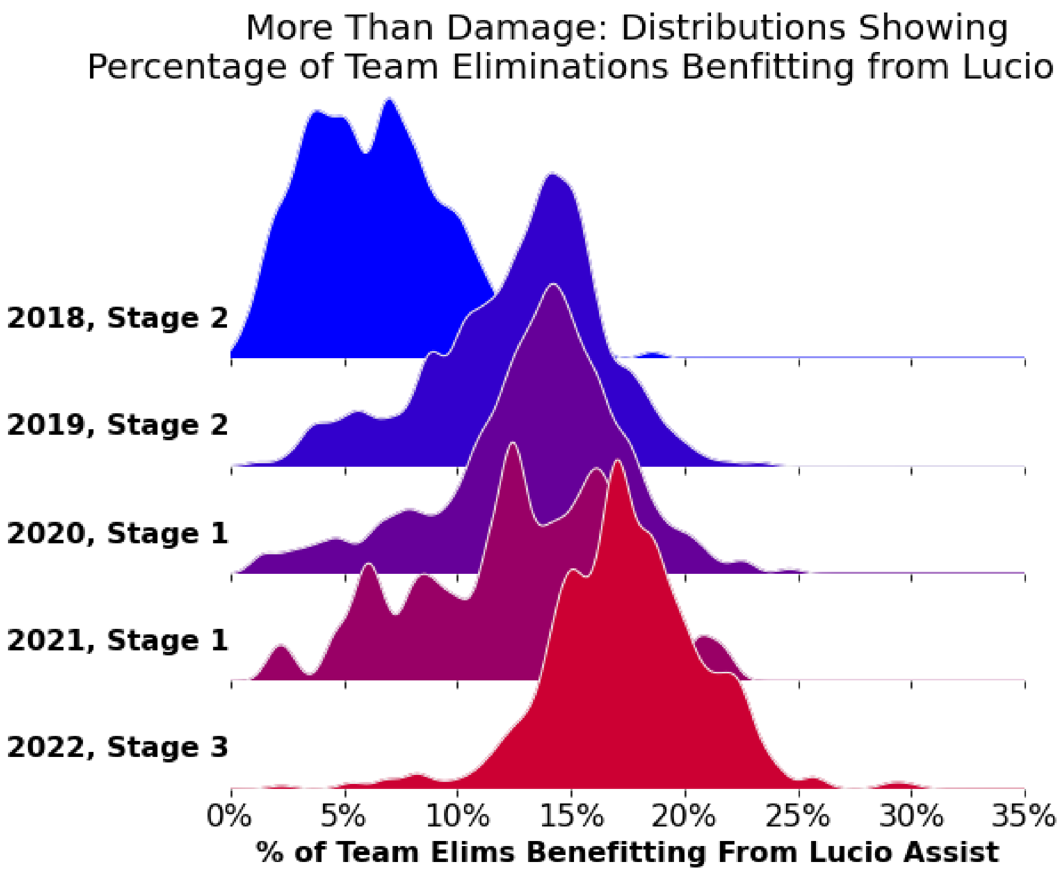


Figure 4: Percentage of Team Eliminations Benefitting From A Lucio Offensive Assist. The above ridge plot shows normalized distributions of team eliminations with a Lucio assist for stage 2, 2018; stage 2, 2019; stage 1, 2020; stage 1, 2021; and stage 3 2022. The distributions shift right over time, with a particularly stark jump between 2018 ad 2019.

This is just the tip of the iceberg, as there is even more detail to be found when examining the dominant strategies from each of these periods and the evolution of the other characters. It's exciting to be able to pinpoint improvement and I personally cannot wait to see how the game of Overwatch continues to evolve at the highest levels!

Critique

Below is the copy-pasted critique I recieved on earlier drafts of the plots. I ended up switching from Seaborn to Altair for the first plot and realized I had accidentally plotted the wrong stage for the 2022 season thanks to the feedback.

"Hey Doug,

I really appreciated the context provided for the 'lucio graphic', specifically I thought that the couching of the graphic in the general play of lucio and the development of the meta was well represented in the graph. It seems obvious that there was initial sparsity but the competitors clearly began experimenting with 'lucio' overtime and the really beautiful insightful part of the graphic is that it is clear that after the intial explosion of play and experimentation the maps in which lucio performed retained his play whereas he seemed to drop off in other over time. I'd be interested in some more granularity in teh data as it seems that the categorical blocking made some large partitions in the graphic (2020, 1&2) faded. My only major gripe with the first graphic is the muddled color scheme as a result of some weird background to the histogram (atleast it seems this way). I don't know if this is an issue with this type of histogram in seaborn or if this is a purposeful decision about the palette. The second graphic paints a clear picture that 'lucio' play got more aggressive over time, I would be interested in understanding why the data got more noisy. The third graphic shows that the 'aggression' metric going up was not congruent with a increase in deaths, which implies that the 'aggression' is more of an optimization than a style shift of the character (mean deaths seems similar vs. mean % team elimination contribution). The final graphic completes this picture and shows that 'lucio' seems to actually also provide value to a team, not just value as an individual character, which shows the improvement of 'lucio' as a character in terms of hte composition of a team. I really liked the overlaying of the ridge plot as it showed the dramatic shift after 2018 and the subsequent transformations of the initial outbreak of 'lucio' play in 2019 stage 2.

Cool project."

Sources

2018 - 2022 Player Stats. Overwatch League Stats Lab. https://overwatchleague.com/en-us/statslab. Updated 10/26/2022.

Code for Data Wrangling and Plots

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import altair as alt
import matplotlib.image as mpimg
```

```python
stg1_2018 = pd.read_csv("2018/phs_2018_stage_1.csv")
stg2_2018 = pd.read_csv("2018/phs_2018_stage_2.csv")
stg3_2018 = pd.read_csv("2018/phs_2018_stage_3.csv")
stg4_2018 = pd.read_csv("2018/phs_2018_stage_4.csv")
playoffs_2018 = pd.read_csv("2018/phs_2018_playoffs.csv")

stg1_2019 = pd.read_csv("2019/phs_2019_stage_1.csv")
stg2_2019 = pd.read_csv("2019/phs_2019_stage_2.csv")
stg3_2019 = pd.read_csv("2019/phs_2019_stage_3.csv")
stg4_2019 = pd.read_csv("2019/phs_2019_stage_4.csv")
playoffs_2019 = pd.read_csv("2019/phs_2019_playoffs.csv")

pt1_2020 = pd.read_csv("2020/phs_2020_1.csv")
pt2_2020 = pd.read_csv("2020/phs_2020_2.csv")

full_2021 = pd.read_csv("phs_2021_1.csv")

full_2022 = pd.read_csv("phs-2022.csv")
```

```python
stg1_2019.columns
```

```
Index(['pelstart_time', 'match_id', 'stage', 'map_type', 'map_name', 'player',
       'team', 'stat_name', 'hero', 'stat_amount'],
      dtype='object')
```

```python
# GET DATAFRAMES WITH ONLY ALL STAT NAMES
stat_names_2018 = playoffs_2018.stat_name.unique()

stat_names_2019 = playoffs_2019.stat_name.unique()

stat_names_2020 = pt2_2020.stat_name.unique()
```

```
stat_names_2021 = full_2021.stat_name.unique()

stat_names_2022 = full_2022.stat_name.unique()
```

```
# GET DATAFRAMES WITH ONLY LUCIO STAT NAMES
Lucio_stat_names_2018 = playoffs_2018[(playoffs_2018["hero"] == "Lúcio") | (playoffs_2018[

Lucio_stat_names_2019 = playoffs_2019[(playoffs_2019["hero"] == "Lúcio") | (playoffs_2019[

Lucio_stat_names_2020 = pt2_2020[(pt2_2020["hero_name"] == "Lúcio") | (pt2_2020["hero_name

Lucio_stat_names_2021 = full_2021[(full_2021["hero_name"] == "Lúcio") | (full_2021["hero_n

Lucio_stat_names_2022 = full_2022[(full_2022["hero_name"] == "Lúcio") | (full_2022["hero_n
```

```
def stats_pivot(frame):

    column_names = {'tournament_title': 'stage',
                     'player': 'player_name',
                     'team': 'team_name',
                     'hero': 'hero_name',
                   'stat': 'stat_amount',
                   'esports_match_id': 'match_id',
                   'pelstart_time': 'start_time',
                   'amount': 'stat_amount'}

    frame = frame.rename(column_names, axis=1)
    pivot_rough = frame.pivot_table('stat_amount', \
                                    ['start_time', 'stage', 'map_type', \
                                     'map_name', 'player_name', \
                                     'match_id', 'team_name', \
                                     'hero_name'], ['stat_name'])
    pivot_clean = pivot_rough.reset_index().rename_axis(['index']).rename_axis(None, axis

    return pivot_clean
```

```
stg1_2018_stats = stats_pivot(stg1_2018)
stg2_2018_stats = stats_pivot(stg2_2018)
stg3_2018_stats = stats_pivot(stg3_2018)
```

```
stg4_2018_stats = stats_pivot(stg4_2018)
playoffs_2018_stats = stats_pivot(playoffs_2018)

stg1_2019_stats = stats_pivot(stg1_2019)
stg2_2019_stats = stats_pivot(stg2_2019)
stg3_2019_stats = stats_pivot(stg3_2019)
stg4_2019_stats = stats_pivot(stg4_2019)
playoffs_2019_stats = stats_pivot(playoffs_2019)

pt1_2020_stats = stats_pivot(pt1_2020)
pt2_2020_stats = stats_pivot(pt2_2020)

full_2021_stats = stats_pivot(full_2021)

full_2022_stats = stats_pivot(full_2022)
```

Add stage and season tag

```
def insert_consistent_col(frame, val, col_index, col_name):
    n, p = frame.shape
    new_col = np.full((n,), val)
    frame.insert(col_index, col_name, new_col)
    return frame

def insert_varied_col(frame, new_col, col_index, col_name):
    frame.insert(col_index, col_name, new_col)
    return frame

def replace_consistent_col(frame, val, col_name):
    n, p = frame.shape
    new_col = np.full((n,), val)
    frame[col_name] = new_col
    return frame

def replace_varied_col(frame, new_col, col_name):
    frame[col_name] = new_col
    return frame
```

```
stg1_2018_stats = insert_consistent_col(stg1_2018_stats, 2018, 1, 'season_year')
stg1_2018_stats = replace_consistent_col(stg1_2018_stats, 1, 'stage')

stg2_2018_stats = insert_consistent_col(stg2_2018_stats, 2018, 1, 'season_year')
stg2_2018_stats = replace_consistent_col(stg2_2018_stats, 2, 'stage')

stg3_2018_stats = insert_consistent_col(stg3_2018_stats, 2018, 1, 'season_year')
stg3_2018_stats = replace_consistent_col(stg3_2018_stats, 3, 'stage')

stg4_2018_stats = insert_consistent_col(stg4_2018_stats, 2018, 1, 'season_year')
stg4_2018_stats = replace_consistent_col(stg4_2018_stats, 4, 'stage')

playoffs_2018_stats = insert_consistent_col(playoffs_2018_stats, 2018, 1, 'season_year')
playoffs_2018_stats = replace_consistent_col(playoffs_2018_stats, 5, 'stage')

stg1_2019_stats = insert_consistent_col(stg1_2019_stats, 2019, 1, 'season_year')
stg1_2019_stats = replace_consistent_col(stg1_2019_stats, 1, 'stage')

stg2_2019_stats = insert_consistent_col(stg2_2019_stats, 2019, 1, 'season_year')
stg2_2019_stats = replace_consistent_col(stg2_2019_stats, 2, 'stage')

stg3_2019_stats = insert_consistent_col(stg3_2019_stats, 2019, 1, 'season_year')
stg3_2019_stats = replace_consistent_col(stg3_2019_stats, 3, 'stage')

stg4_2019_stats = insert_consistent_col(stg4_2019_stats, 2019, 1, 'season_year')
stg4_2019_stats = replace_consistent_col(stg4_2019_stats, 4, 'stage')

playoffs_2019_stats = insert_consistent_col(playoffs_2019_stats, 2019, 1, 'season_year')
playoffs_2019_stats = replace_consistent_col(playoffs_2019_stats, 5, 'stage')

pt1_2020_stats = insert_consistent_col(pt1_2020_stats, 2020, 1, 'season_year')
pt2_2020_stats = insert_consistent_col(pt2_2020_stats, 2020, 1, 'season_year')

full_2021_stats = insert_consistent_col(full_2021_stats, 2021, 1, 'season_year')

full_2022_stats = insert_consistent_col(full_2022_stats, 2022, 1, 'season_year')
```

```
def clean_date(start_time):
    split = start_time.split(' ')
    full_date = split[0]
    full_date = full_date.replace('-', '/')
    full_date = full_date.split('/')
    return full_date

def pull_date(start_time):
    full_date = clean_date(start_time)
    date = tuple([int(num) for num in full_date if len(num) <= 2])
    return date

def pull_year(start_time):
    full_date = clean_date(start_time)
    year = [int(num) for num in full_date if len(num) > 2][0]
    return year

def pull_all_time(start_time):
    full_date = clean_date(start_time)
    date = tuple([int(num) for num in full_date if len(num) <= 2])
    year = [int(num) for num in full_date if len(num) > 2][0]
    return date, year
```

```
def match_dates(range_start, range_end, date): #ASSUMES ALL DATES ARE IN SAME YEAR
    month_range = (range_start[0], range_end[0])
    day_range = (range_start[1], range_end[1])

    q_month = date[0]
    q_day = date[1]

    if not month_range[0] <= q_month <= month_range[1]:
        return False
    if q_month == month_range[0]:
        if q_day < day_range[0]:
            return False
    if q_month == month_range[1]:
        if q_day > day_range[1]:
            return False
    return True

def modern_stage_token(start_time): # 4 STAGES, POST-SEASON COUNTED AS STAGE 5
    date, year = pull_all_time(start_time)

    year_stage_ranges = {2020: {((2, 8), (4, 26)): 1,
                                ((5, 2), (5, 25)): 2,
                                ((6, 13), (7, 6)): 3,
                                ((7, 17), (8, 9)): 4,
                                ((8, 14), (10, 10)): 5
                               },
                         2021: {((4, 16), (5, 9)): 1,
                                ((5, 21), (6, 13)): 2,
                                ((6, 25), (7, 18)): 3,
                                ((7, 30), (8, 22)): 4,
                                ((9, 4), (9, 26)): 5}}

    stage_dict = year_stage_ranges[year]
    for key in stage_dict.keys():
        range_start, range_end = key
        if match_dates(range_start, range_end, date) == True:
            return stage_dict[key]
    raise ValueError(f"Could not classify stage for {date}")

def clean_2022_stage(raw_label): # 4 STAGES, POST-SEASON COUNTED AS STAGE 5
    stage_dict = {'Kickoff Clash': 1,
                 'Midseason Madness': 2,
                 'Summer Showdown': 3,
                 'Countdown Cup': 4,
                 'Postseason': 5}

    tourney = raw_label.split(':')[0]
    stage = stage_dict[tourney]
    return stage

def map_to_col(frame, function, col_name):
    vfunc = np.vectorize(function)
    col = frame[col_name]
    new_col = vfunc(col)
    return new_col

def map_replace_col(frame, function, col_map, col_replace):
    new_col = map_to_col(frame, function, col_map)
    new_frame = replace_varied_col(frame, new_col, col_replace)
    return new_frame
```

```
pt1_2020_stats = map_replace_col(pt1_2020_stats, modern_stage_token, 'start_time', 'stage'
pt2_2020_stats = map_replace_col(pt2_2020_stats, modern_stage_token, 'start_time', 'stage'

full_2021_stats = map_replace_col(full_2021_stats, modern_stage_token, 'start_time', 'stag

full_2022_stats = map_replace_col(full_2022_stats, clean_2022_stage, 'stage', 'stage')
```

```
full_2020_stats = pd.concat([pt1_2020_stats, pt2_2020_stats], axis=0).reset_index(drop=Tru
full_2019_stats = pd.concat([stg1_2019_stats, stg2_2019_stats, stg3_2019_stats, stg4_2019_
full_2018_stats = pd.concat([stg1_2018_stats, stg2_2018_stats, stg3_2018_stats, stg4_2018_
```

Now clean to have a version with just Lucio data

```
# CLEAN EACH DATAFRAME TO ONLY CONTAIN LUCIO PLAY DATA

Lucio_2018 = full_2018_stats[(full_2018_stats["hero_name"] == "Lúcio") | (full_2018_stats[

Lucio_2019 = full_2019_stats[(full_2019_stats["hero_name"] == "Lúcio") | (full_2019_stats[

Lucio_2020 = full_2020_stats[(full_2020_stats["hero_name"] == "Lúcio") | (full_2020_stats[

Lucio_2021 = full_2021_stats[(full_2021_stats["hero_name"] == "Lúcio") | (full_2021_stats[

Lucio_2022 = full_2022_stats[(full_2022_stats["hero_name"] == "Lúcio") | (full_2022_stats[
```

Now merge into one mega-frame

```
Lucio_stat_names_2022 = list(Lucio_stat_names_2022)
meta_stat_names = ['start_time', 'season_year', 'stage', 'map_type', 'map_name', 'player_n
```

```
Lucio_stats = pd.concat([Lucio_2018, Lucio_2019, Lucio_2020, Lucio_2021, Lucio_2022], axis
```

```
Lucio_stats.shape
```

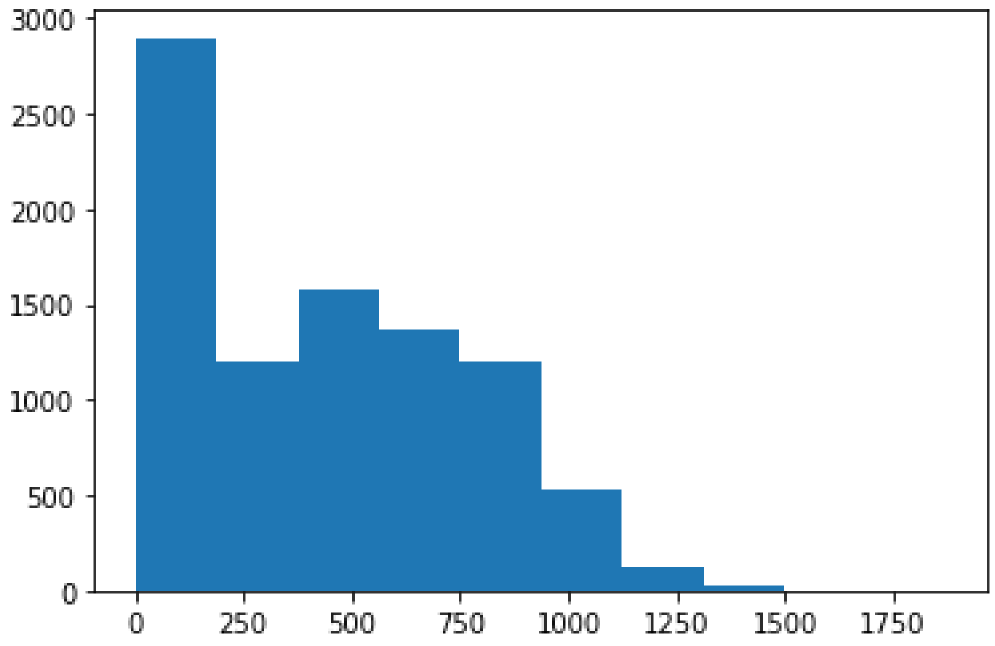
```
(8993, 69)
```

Cut Lucio data from < 2 min of playtime

```
Lucio_stats["Time Played"].describe()
```

```
count    8903.000000
mean      426.033541
std       341.509248
min         0.970016
25%        48.675742
50%       427.255614
75%       691.119101
max      1873.745082
Name: Time Played, dtype: float64
```

```
# WANT TO ESTABLISH A TIME PLAYED CUTOFF TO ONLY INCLUDE MATCHES WITH SUBSTANTIVE LUCIO PL
# Need to exclude quick swaps, boost swaps, and most stall swaps
# Also want to exclude matches that were complete stomps, as the stats are heavily skewed
# not reflective of typical play

plt.hist(Lucio_stats["Time Played"])
plt.show()
```



The large bump near zero is probably due to a combination of stall and early speed swaps. When a match is close and you need to get back to the fight quickly, people often swap to Lucio to rush back. An early speed swap is a common trend where, at the beginning of a match, as teams first leave their spawn they use a Lucio speed amp then quickly switch off. The goal is to gain a tiny advantage in reaching the objective / favorable positions at the start of the match.

```
# If Lucio is played for 2-minutes in a match, the team / player made a substantive
# attempt to use Lucio in their play. Lower means they probably had a few stall swaps
# or swapped off quickly because they were getting rolled.

def time_cutoff(frame, min_cut):
    cutoff = min_cut * 60
    cut_frame = frame[frame["Time Played"] >= cutoff]
    return cut_frame

# CLEANS STAT TO GIVE RATE PER X MINUTES OF PLAY
def per_2_col(stat, time, min_agg):
    time_2  = time / (60 * min_agg) # covert time corresponding to stat obversation from s
    per_2 = stat / time_2 # convert stat to stat per 2-minutes
    return per_2

# CLEAN STATS TO ONLY INCLUDE MAPS WHERE LUCIO MADE UP 90%+ OF THE PLAYTIME
```

```
Lucio_stats_2min = time_cutoff(Lucio_stats, 2).reset_index(drop=True)
Lucio_stats_2min.shape, Lucio_stats.shape
```

```
((6291, 69), (8993, 69))
```

Get counts of how many times a map was played (leverage match IDs) per stage

```
np.unique(np.array(list(full_2022_stats.map_name) + list(full_2019_stats.map_name)))
```

```
array(['Blizzard World', 'Busan', 'Circuit royal', 'Colosseo', 'Dorado',
       'Eichenwalde', 'Hanamura', 'Havana', 'Hollywood',
       'Horizon Lunar Colony', 'Ilios', 'Junkertown', "King's Row",
       'Lijiang Tower', 'Midtown', 'Nepal', 'New Queen Street', 'Numbani',
       'Oasis', 'Paraíso', 'Paris', 'Rialto', 'Route 66',
       'Temple of Anubis', 'Volskaya Industries', 'Watchpoint: Gibraltar'],
      dtype='<U21')
```

```
np.unique(np.array(list(full_2022_stats.map_type) + list(full_2019_stats.map_type)))
```

```
array(['ASSAULT', 'CONTROL', 'HYBRID', 'PAYLOAD', 'control', 'hybrid',
       'payload', 'push'], dtype='<U7')
```

```
def Merge(dict1, dict2):
    return dict1 | dict2

def capitalize_vals(frame, col=None, index=False):
    if index == True:
        frame.index = frame.index.map(lambda x: x.upper())
    else:
        frame[col] = frame[col].map(lambda x: x.upper())
    return frame

def pull_map_type_dict(frame):
    frame['map_type'] = frame['map_type'].map(lambda x: x.upper())
    frame['map_name'] = frame['map_name'].map(lambda x: x.upper())

    map_type_df = frame[['map_name', 'map_type']].drop_duplicates().reset_index(drop=True)
    map_type_dict = dict(zip(map_type_df.map_name, map_type_df.map_type))
    return map_type_dict

def build_map_type_dict(df_lst):
    map_type_dict = pull_map_type_dict(df_lst[0])
    for frame in df_lst[1:]:
        temp_dict = pull_map_type_dict(frame)
        map_type_dict = Merge(map_type_dict, temp_dict)

    del_keys = []
    for key, value in map_type_dict.items():
        if value == 'UNKNOWN':
            del_keys.append(key)

    for key in del_keys:
        del map_type_dict[key]
    return map_type_dict

def total_map_appearance_counts(full_frame, stage): # returns dictionary where keys are ma
    full_counts = full_frame[full_frame.stage == stage][['match_id', 'stage', 'map_type', 
    full_counts = capitalize_vals(full_counts, index=True)
    full_counts = dict(zip(full_counts.index, full_counts.values))
    return full_counts

def lucio_map_appearance_counts(lucio_frame, year, stage): # returns dictionary where keys
    lucio_counts = lucio_frame[(lucio_frame.stage == stage) & (lucio_frame.season_year == 
    lucio_counts = capitalize_vals(lucio_counts, index=True)
    lucio_counts = dict(zip(lucio_counts.index, lucio_counts.values))
    return lucio_counts

def sort_map_type(map_name, map_type_dict):
    map_type = map_type_dict[map_name.upper()]
    return map_type

def pct_lucio_apps(full_counts, lucio_counts, map_type_dict, year, stage): # FOR EACH POSS
    pct_l_apps_df = pd.DataFrame({'season_year':[], 'stage':[],
                                  'map_type':[], 'map_name':[],
                                  'appearances':[], 'Lucio_appearances':[],
                                 'Lucio_play_pct':[]})
    for map_name in map_type_dict.keys():
        map_type = sort_map_type(map_name, map_type_dict)

        try:
            full_count = full_counts[map_name]
        except:
            full_count = 0

        try:
            l_count = lucio_counts[map_name]
        except:
            l_count = 0

        try:
            lucio_pct = l_count / full_count
        except:
            lucio_pct = 0
        df2 = pd.DataFrame({'season_year': year, 'stage': stage,
               'map_type': map_type, 'map_name': map_name,
               'appearances': full_count, 'Lucio_appearances': l_count,
              'Lucio_play_pct': lucio_pct}, index=[0])

        pct_l_apps_df = pd.concat([pct_l_apps_df, df2], ignore_index = True)
    return pct_l_apps_df

def lucio_pct_single(full_frame, lucio_frame, map_type_dict):
    year = full_frame.season_year[0]
    pct_lucio_apps_df = pd.DataFrame({'season_year':[], 'stage':[],
                              'map_type':[], 'map_name':[],
                              'appearances':[], 'Lucio_appearances':[],
                             'Lucio_play_pct':[]})
    for stage in range(1, 6):
        full_counts = total_map_appearance_counts(full_frame, stage)
        lucio_counts = lucio_map_appearance_counts(lucio_frame, year, stage)
        pct_lucio_stg = pct_lucio_apps(full_counts, lucio_counts, map_type_dict, year, sta
        pct_lucio_apps_df = pd.concat([pct_lucio_apps_df, pct_lucio_stg], ignore_index = T

    return pct_lucio_apps_df

def build_full_lucio_pct_df(full_frame_lst, lucio_frame, map_type_dict):
    lucio_apps_pct_df = pd.DataFrame({'season_year':[], 'stage':[],
                              'map_type':[], 'map_name':[],
                              'appearances':[], 'Lucio_appearances':[],
                             'Lucio_play_pct':[]})

    for full_frame in full_frame_lst:
        year = full_frame.season_year[0]
        lucio_year_frame = lucio_frame[lucio_frame.season_year == year].reset_index(drop=T
        tmp_frame = lucio_pct_single(full_frame, lucio_year_frame, map_type_dict)
        lucio_apps_pct_df = pd.concat([lucio_apps_pct_df, tmp_frame], ignore_index = True)

    lucio_apps_pct_df.season_year = pd.to_numeric(lucio_apps_pct_df.season_year, downcast=
    lucio_apps_pct_df.stage = pd.to_numeric(lucio_apps_pct_df.stage, downcast='integer')

    time_ordering = (lucio_apps_pct_df.season_year * 10) + lucio_apps_pct_df.stage
    lucio_apps_pct_df.insert(2, 'time_ordering', time_ordering)
    lucio_apps_pct_df = lucio_apps_pct_df.sort_values(['map_type', 'map_name', 'season_yea
    return lucio_apps_pct_df
```

Make DataFrame with columns of order: season_year, stage, map_type, map_name, count

Loop over tables and stages, create for each year / stage, and append vertically

Combine season and stage into one number (for auto sorting in sns.histplot) and make function to generate label string

Convert map names into categorical and order to group by map type - df['Item'] = pd.Categorical(df['Item'], ['Up','Down','Left','Right'])

```
full_frame_lst = [full_2018_stats, full_2019_stats, full_2020_stats, full_2021_stats, full
map_type_dict = build_map_type_dict(frame_lst)
lucio_apps_pct_df = build_full_lucio_pct_df(full_frame_lst, Lucio_stats_2min, map_type_dic
```

```
lucio_apps_pct_df['Lucio_play_pct'] = (lucio_apps_pct_df['Lucio_play_pct'] * 100).astype(i
lucio_apps_pct_df
```

	season_year	stage	time_ordering	map_type	map_name	appearances	Lucio_app
0	2018	1	20181	ASSAULT	HANAMURA	0.0	0.0
1	2018	2	20182	ASSAULT	HANAMURA	31.0	29.0
2	2018	3	20183	ASSAULT	HANAMURA	0.0	0.0
3	2018	4	20184	ASSAULT	HANAMURA	31.0	18.0
4	2018	5	20185	ASSAULT	HANAMURA	4.0	3.0
...	...	...	...	...	...	...	...
645	2022	1	20221	PUSH	NEW QUEEN STREET	32.0	32.0
646	2022	2	20222	PUSH	NEW QUEEN STREET	33.0	12.0
647	2022	3	20223	PUSH	NEW QUEEN STREET	21.0	21.0
648	2022	4	20224	PUSH	NEW QUEEN STREET	23.0	16.0
649	2022	5	20225	PUSH	NEW QUEEN STREET	1.0	1.0

```
time_labels = []
for row in range(len(lucio_apps_pct_df)):
    row = lucio_apps_pct_df.iloc[row]
    year = str(row.season_year)
    stage = str(row.stage)
    time_lbl = year + ', ' + stage
    time_labels.append(time_lbl)
```

```
lucio_apps_pct_df["time_labels"] = time_labels
```

```
lucio_apps_pct_df.map_name.unique()
```

```
array(['HANAMURA', 'HORIZON LUNAR COLONY', 'PARIS', 'TEMPLE OF ANUBIS',
       'VOLSKAYA INDUSTRIES', 'BUSAN', 'ILIOS', 'LIJIANG TOWER', 'NEPAL',
       'OASIS', 'BLIZZARD WORLD', 'EICHENWALDE', 'HOLLYWOOD',
       "KING'S ROW", 'MIDTOWN', 'NUMBANI', 'PARAÍSO', 'CIRCUIT ROYAL',
       'DORADO', 'HAVANA', 'JUNKERTOWN', 'RIALTO', 'ROUTE 66',
       'WATCHPOINT: GIBRALTAR', 'COLOSSEO', 'NEW QUEEN STREET'],
      dtype=object)
```

```
lucio_apps_pct_df['time_labels'] = pd.Categorical(lucio_apps_pct_df['time_labels'],
                                   ['2018, 1', '2018, 2', '2018, 3', '2018, 4', '2018, 5',
       '2019, 2', '2019, 3', '2019, 4', '2019, 5', '2020, 1', '2020, 2',
       '2020, 3', '2020, 4', '2020, 5', '2021, 1', '2021, 2', '2021, 3',
       '2021, 4', '2021, 5', '2022, 1', '2022, 2', '2022, 3', '2022, 4',
       '2022, 5'])

lucio_apps_pct_df['map_name'] = pd.Categorical(lucio_apps_pct_df['map_name'],
                                   ['HANAMURA', 'HORIZON LUNAR COLONY', 'PARIS', 'TEMPLE O
       'VOLSKAYA INDUSTRIES', 'BUSAN', 'ILIOS', 'LIJIANG TOWER', 'NEPAL',
       'OASIS', 'BLIZZARD WORLD', 'EICHENWALDE', 'HOLLYWOOD',
       "KING'S ROW", 'MIDTOWN', 'NUMBANI', 'PARAÍSO', 'CIRCUIT ROYAL',
       'DORADO', 'HAVANA', 'JUNKERTOWN', 'RIALTO', 'ROUTE 66',
       'WATCHPOINT: GIBRALTAR', 'COLOSSEO', 'NEW QUEEN STREET'])

lucio_apps_pct_df = lucio_apps_pct_df.rename({'Lucio_play_pct': 'Lucio Play %',
                                              'map_name': 'Name of Map',
                                              'time_labels': 'Year, Stage'}, axis=1)
```

```
alt.Chart(lucio_apps_pct_df).mark_rect().encode(
    x=alt.X('Year, Stage:N', axis=alt.Axis(labelAngle=-45)),
    y='Name of Map:N',
    color=alt.Color('Lucio Play %:Q', scale=alt.Scale(scheme="greenblue"))
).properties(
    width=700,
    height=500,
    title="% of Time Lucio Played on Each Map By Stage"
    ).configure_axis(
    labelFontSize=14,
    titleFontSize=18
    ).configure_legend(
    titleFontSize=16,
    labelFontSize=16,
    ).configure_title(
    fontSize=20)
```

```
alt.Chart(...)
```

Get team sums of all damage done and final blows

Filter to only all heroes per player per team then group by team and match_id and sum damage and final blows

```
def pull_full_aggression_stats(frame):
    cut_frame = frame[['season_year', 'stage', 'map_type',
                     'map_name', 'match_id', 'team_name',
                     'hero_name', 'All Damage Done', 'Final Blows', 'Eliminations']]
    cut_frame = cut_frame[cut_frame["hero_name"] == 'All Heroes']
    full_aggro_stats = cut_frame.groupby(['season_year', 'stage',
                                         'match_id', 'map_name',
                                         'team_name']).sum().reset_index()[['season_year'
                                                                            'match_id', '
                                                                            'team_name',
                                                                            'Final Blows'

    return full_aggro_stats
```

Take Lucio data for % of team final blows and damage from each stage per season with the highest amount of Lucio played

1. Reduces size of data for plotting
2. Selects eras where Lucio is a mainstay so fewer matches where he only has a couple minutes playtime and makes up a tiny proportion of stats

The relevant stages are as follows:

1. 2018 stage 2
2. 2019 stage 1
3. 2020 stage 1
4. 2021 stage 1
5. 2022 stage 1

```
def pull_relevant_lucio_stg(lucio_frame, year, stage):
    relevant_data = lucio_frame[(lucio_frame.season_year == year) & (lucio_frame.stage ==
    return relevant_data

def merge_aggression_stats(full_stats, lucio_stats):
    lucio_frame = lucio_stats[['map_name', 'match_id', 'team_name',
                               'All Damage Done', 'Final Blows',
                               'Eliminations', 'Offensive Assists', 'Deaths']].rename({'Al
                                                                                     'Final Blows':
                                                                                     'Eliminations'
                                                                                    'Offensive Assi
                                                                                             'Dea
    full_frame = full_stats.drop(['stage', 'season_year'], axis=1)
    aggro_stats = lucio_frame.merge(full_frame, how='inner', on=['map_name', 'match_id', '
    return aggro_stats

def stat_over_stat(num_stat, denom_stat, frame):
    divided_stats = frame[num_stat] / frame[denom_stat]
    return divided_stats

def compare_aggro_frame(full_frame, lucio_frame, stage):
    year = full_frame.season_year[0]
    full_frame = full_frame[full_frame.stage == stage]
    full_aggro_stats = pull_full_aggression_stats(full_frame)
    lucio_aggro_stats = pull_relevant_lucio_stg(lucio_frame, year, stage)

    compare_aggro_stats = merge_aggression_stats(full_aggro_stats, lucio_aggro_stats)
    compare_aggro_stats['pct_Lucio_Final_Blows'] = stat_over_stat('Lucio Final Blows', 'Fi
    compare_aggro_stats['pct_Lucio_Damage'] = stat_over_stat('Lucio Damage Done', 'All Dam
    compare_aggro_stats['pct_Lucio_Elims'] = stat_over_stat('Lucio Eliminations', 'Final B
    compare_aggro_stats['pct_Lucio_Assist'] = stat_over_stat('Lucio Offensive Assists', 'E

    return compare_aggro_stats
```

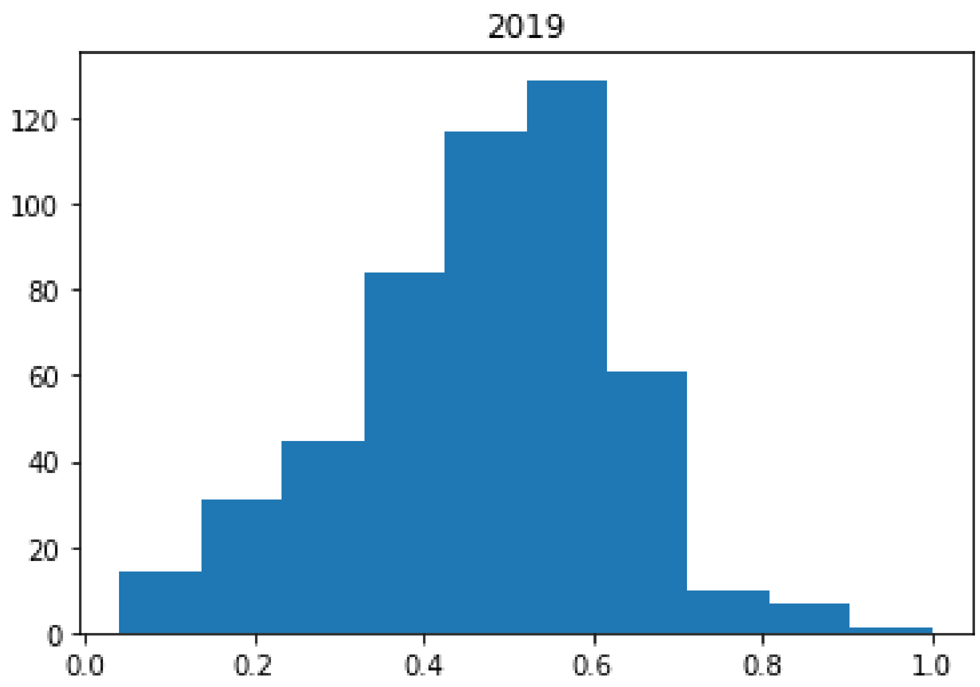
```
def compare_aggro_arrays(full_frame, lucio_frame, stage):
    year = full_frame.season_year[0]
    full_frame = full_frame[full_frame.stage == stage]
    full_aggro_stats = pull_full_aggression_stats(full_frame)
    lucio_aggro_stats = pull_relevant_lucio_stg(lucio_frame, year, stage)

    compare_aggro_stats = merge_aggression_stats(full_aggro_stats, lucio_aggro_stats)
    pct_Lucio_Final_Blows = stat_over_stat('Lucio Final Blows', 'Final Blows', compare_agg
    pct_Lucio_Damage = stat_over_stat('Lucio Damage Done', 'All Damage Done', compare_aggr
    pct_Lucio_Elims = stat_over_stat('Lucio Eliminations', 'Final Blows', compare_aggro_st
    pct_Lucio_Assist = stat_over_stat('Lucio Offensive Assists', 'Eliminations', compare_a
    lucio_deaths = compare_aggro_stats['Lucio Deaths']

    return pct_Lucio_Final_Blows, pct_Lucio_Damage, pct_Lucio_Elims, pct_Lucio_Assist, luc
```

```
pct_Lucio_Final_Blows, pct_Lucio_Damage, pct_Lucio_Elims, pct_Lucio_Assist = compare_aggro
pct_Lucio_Final_Blows.mean(), pct_Lucio_Damage.mean(), pct_Lucio_Elims.mean(), pct_Lucio_A
```

```
(0.08625052108920443,
 0.10040981040604773,
 0.4720926801202522,
 0.1269564087357047)
```

```
plt.hist(pct_Lucio_Elims)
plt.title('2019')
```

```
Text(0.5, 1.0, '2019')
```

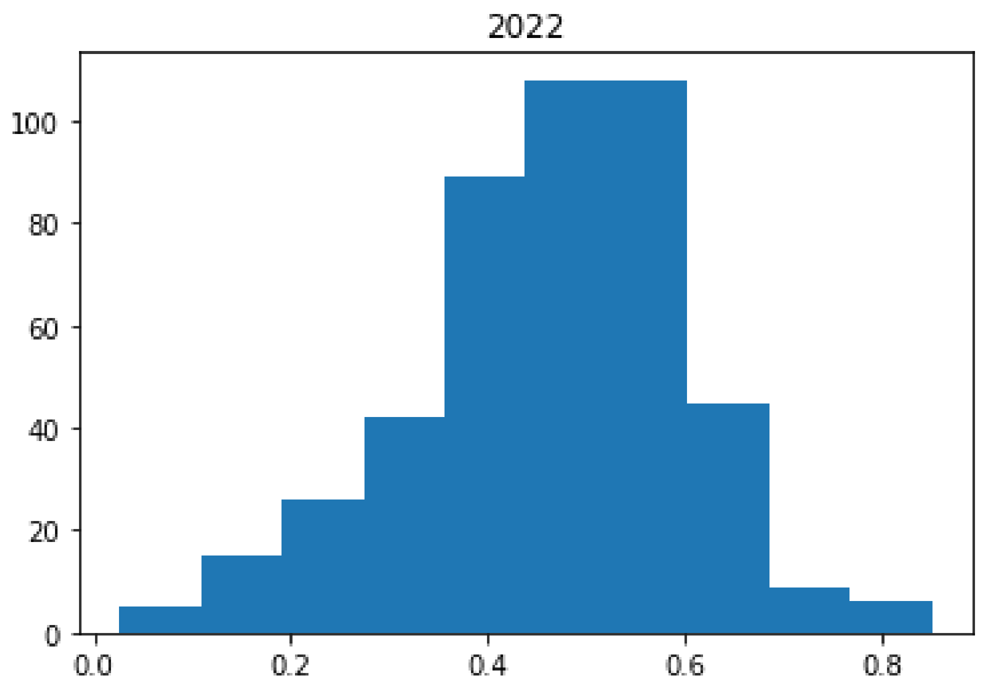


```
pct_Lucio_Final_Blows, pct_Lucio_Damage, pct_Lucio_Elims, pct_Lucio_Assist = compare_aggro
pct_Lucio_Final_Blows.mean(), pct_Lucio_Damage.mean(), pct_Lucio_Elims.mean(), pct_Lucio_A
```

```
(0.11028789629079079,
 0.12476103473236944,
 0.4625469930057494,
 0.1607406865047156)
```

```
plt.hist(pct_Lucio_Elims)
plt.title('2022')
```

```
Text(0.5, 1.0, '2022')
```

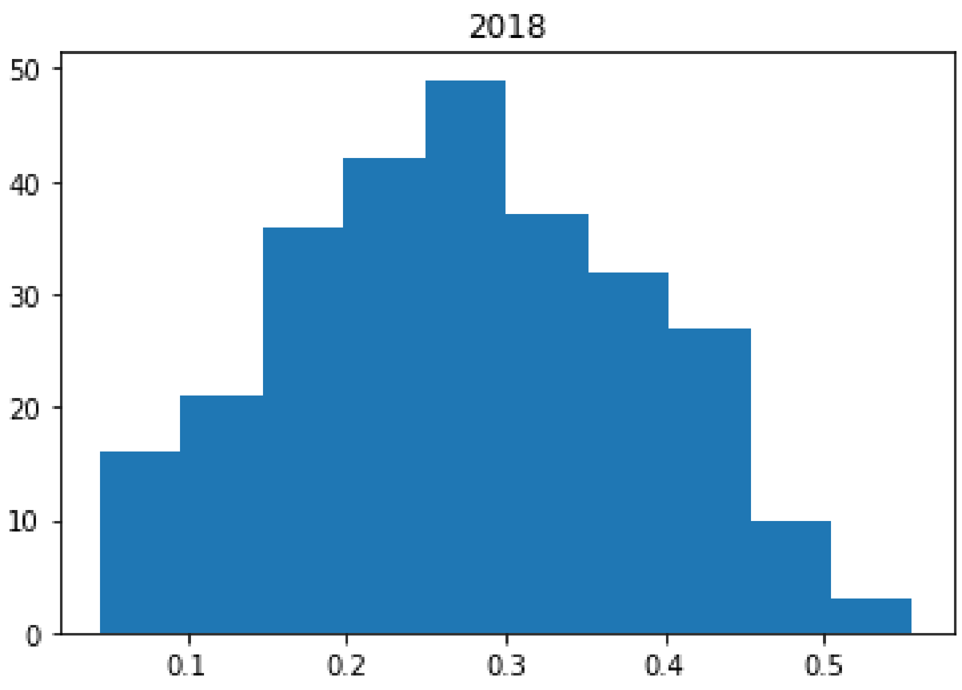


```
pct_Lucio_Final_Blows, pct_Lucio_Damage, pct_Lucio_Elims, pct_Lucio_Assist = compare_aggro
pct_Lucio_Final_Blows.mean(), pct_Lucio_Damage.mean(), pct_Lucio_Elims.mean(), pct_Lucio_A
```

```
(0.062142696738577736,
 0.07917352040053428,
 0.27521941087478163,
 0.06476252794904397)
```

```
plt.hist(pct_Lucio_Elims)
plt.title('2018')
```

```
Text(0.5, 1.0, '2018')
```



Get total number of assists (offensive + defensive), amps (heal + speed), and aura time (heal + speed)

```
Lucio_stats_2min.head()
```

	start_time	season_year	stage	map_type	map_name	player_name	match_id
0	1/11/2018 1:11	2018	1	CONTROL	ILIOS	dhaK	10223
1	1/11/2018 22:29	2018	1	ASSAULT	TEMPLE OF ANUBIS	Zebbosai	10226
2	1/11/2018 2:25	2018	1	ASSAULT	TEMPLE OF ANUBIS	BigG00se	10224
3	1/11/2018 2:52	2018	1	CONTROL	ILIOS	BigG00se	10224
4	1/11/2018 3:46	2018	1	PAYLOAD	JUNKERTOWN	tobi	10225

```
Lucio_stats_2min.columns
```

```
Index(['start_time', 'season_year', 'stage', 'map_type', 'map_name',
       'player_name', 'match_id', 'team_name', 'hero_name', 'All Damage Done',
       'Amped Heal Activations', 'Amped Speed Activations', 'Assists',
       'Average Time Alive', 'Barrier Damage Done', 'Critical Hit Accuracy',
       'Critical Hit Kills', 'Critical Hits', 'Damage - Quick Melee',
       'Damage - Weapon Primary', 'Damage - Weapon Secondary', 'Damage Taken',
       'Deaths', 'Defensive Assists', 'Eliminations', 'Environmental Kills',
       'Final Blows', 'Games Played', 'Games Played Plus Won', 'Games Won',
       'Heal Song Time Elapsed', 'Healing - Healing Boost',
       'Healing - Healing Boost Amped', 'Healing Done', 'Healing Received',
       'Hero Damage Done', 'Knockback Kills', 'Objective Kills',
       'Objective Time', 'Offensive Assists', 'Players Knocked Back',
       'Quick Melee Accuracy', 'Quick Melee Hits', 'Quick Melee Ticks',
       'Self Healing', 'Self Healing Percent of Damage Taken', 'Shots Fired',
       'Shots Hit', 'Shots Missed', 'Sound Barrier Casts',
       'Sound Barrier Efficiency', 'Sound Barriers Provided',
       'Soundwave Kills', 'Speed Song Time Elapsed', 'Time Alive',
       'Time Building Ultimate', 'Time Elapsed per Ultimate Earned',
       'Time Holding Ultimate', 'Time Played', 'Ultimates Earned - Fractional',
       'Ultimates Used', 'Weapon Accuracy', 'Environmental Deaths',
       'Time Discorded', 'Melee Final Blows', 'Multikills', 'Solo Kills',
       'Time Hacked', 'Turrets Destroyed'],
      dtype='object')
```

```
def lucio_util_stats_frame(lucio_frame):
    cut_frame = lucio_frame[['start_time', 'season_year', 'stage', 'map_type',
                 'map_name', 'player_name', 'match_id', 'team_name',
                 'hero_name', 'Amped Heal Activations',
                 'Amped Speed Activations', 'Offensive Assists', 'Defensive Assists',
                 'Assists', 'Heal Song Time Elapsed', 'Speed Song Time Elapsed',
                          'Average Time Alive', 'Time Played']]

    cut_frame['Total_Amps'] = cut_frame['Amped Heal Activations'] + cut_frame['Amped Speed
    cut_frame['pct_speed_amps'] = stat_over_stat('Amped Speed Activations', 'Total_Amps',

    cut_frame['Total_Song_Time'] = cut_frame['Heal Song Time Elapsed'] + cut_frame['Speed
    cut_frame['pct_speed_song'] = stat_over_stat('Speed Song Time Elapsed', 'Total_Song_Ti

    cut_frame['Total_Assists'] = cut_frame['Offensive Assists'] + cut_frame['Defensive Ass
    cut_frame['pct_offensive_assists'] = stat_over_stat('Offensive Assists', 'Total_Assist

    cut_frame['pct_avg_time_alive'] = stat_over_stat('Average Time Alive', 'Time Played',

    return cut_frame
```

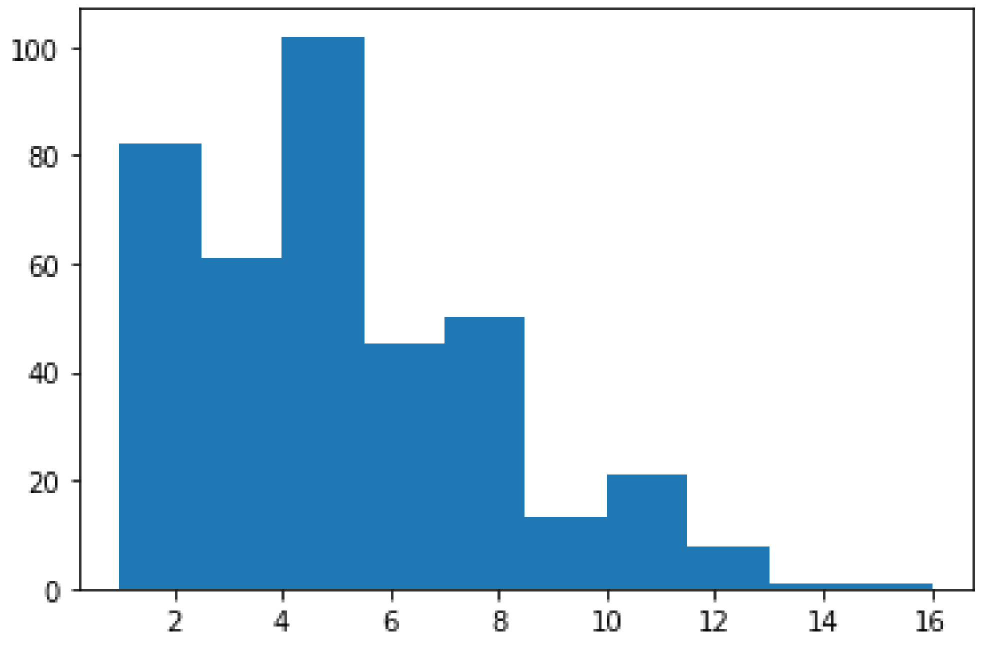
```
util_stats = lucio_util_stats_frame(Lucio_stats_2min)
```

```
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:9: SettingWit
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_guid
  cut_frame['Total_Amps'] = cut_frame['Amped Heal Activations'] + cut_frame['Amped Speed Acti
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:10: SettingWi
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_guid
  cut_frame['pct_speed_amps'] = stat_over_stat('Amped Speed Activations', 'Total_Amps', cut_
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:12: SettingWi
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_guid
  cut_frame['Total_Song_Time'] = cut_frame['Heal Song Time Elapsed'] + cut_frame['Speed Song
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:13: SettingWi
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_guid
  cut_frame['pct_speed_song'] = stat_over_stat('Speed Song Time Elapsed', 'Total_Song_Time',
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:15: SettingWi
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_guid
  cut_frame['Total_Assists'] = cut_frame['Offensive Assists'] + cut_frame['Defensive Assists
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:16: SettingWi
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_guid
  cut_frame['pct_offensive_assists'] = stat_over_stat('Offensive Assists', 'Total_Assists', 
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/2087416982.py:18: SettingWi
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_guid
  cut_frame['pct_avg_time_alive'] = stat_over_stat('Average Time Alive', 'Time Played', cut_
```

```
plt.hist(Lucio_stats_2min[(Lucio_stats_2min.season_year == 2018) & (Lucio_stats_2min.stage
```

```
(array([ 82.,  61., 102.,  45.,  50.,  13.,  21.,   8.,   1.,   1.]),
 array([ 1. ,  2.5,  4. ,  5.5,  7. ,  8.5, 10. , 11.5, 13. , 14.5, 16. ]),
 <BarContainer object of 10 artists>)
```

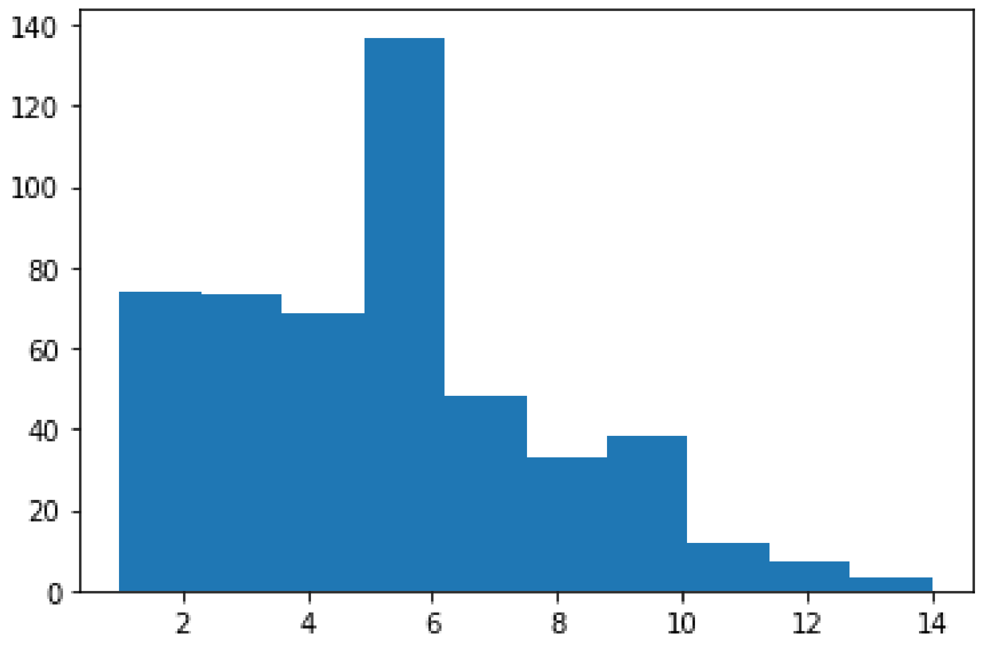


```
Lucio_stats_2min[(Lucio_stats_2min.season_year == 2018) & (Lucio_stats_2min.stage == 2)]['
```

```
(4.8515625, 5.232793522267206)
```

```
plt.hist(Lucio_stats_2min[(Lucio_stats_2min.season_year == 2022) & (Lucio_stats_2min.stage
```

```
(array([ 74.,  73.,  69., 137.,  48.,  33.,  38.,  12.,   7.,   3.]),
 array([ 1. ,  2.3,  3.6,  4.9,  6.2,  7.5,  8.8, 10.1, 11.4, 12.7, 14. ]),
 <BarContainer object of 10 artists>)
```



Create columns by dividing the following stats by arrays generated above:

1. % of team damage done
2. elims out of total team final blows (% of kills where Lucio did damage to target)
3. % of assists that were offensive
4. % of team final blows where Lucio provided an offensive assist (amp speed)
5. % of amp uses that were for speed
6. % of song time that was speed

```
util_stats.head(1)
```

	start_time	season_year	stage	map_type	map_name	player_name	match_id	team_name
0	1/11/2018 1:11	2018	1	CONTROL	ILIOS	dhaK	10223	San Francis

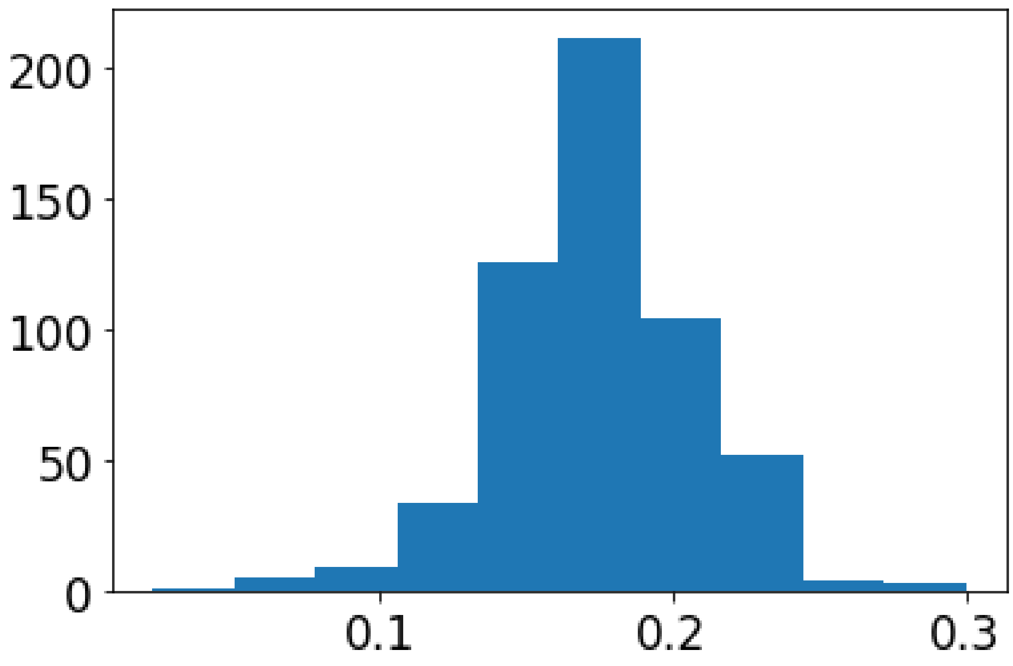
```
time_labels = []
for row in range(len(util_stats)):
    row = util_stats.iloc[row]
    year = str(row.season_year)
    stage = str(row.stage)
    time_lbl = year + ', ' + stage
    time_labels.append(time_lbl)

util_stats['time_labels'] = time_labels
util_stats['time_labels'] = pd.Categorical(util_stats['time_labels'],
                                   ['2018, 1', '2018, 2', '2018, 3', '2018, 4', '2018, 5',
       '2019, 2', '2019, 3', '2019, 4', '2019, 5', '2020, 1', '2020, 2',
       '2020, 3', '2020, 4', '2020, 5', '2021, 1', '2021, 2', '2021, 3',
       '2021, 4', '2021, 5', '2022, 1', '2022, 2', '2022, 3', '2022, 4',
       '2022, 5'])
```

```
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/3559008931.py:9: SettingWit
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_guid
  util_stats['time_labels'] = time_labels
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/3559008931.py:10: SettingWi
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_guid
  util_stats['time_labels'] = pd.Categorical(util_stats['time_labels'],
```

```
pct_Lucio_Final_Blows_2018, pct_Lucio_Damage_2018, pct_Lucio_Elims_2018, pct_Lucio_Assist_
pct_Lucio_Final_Blows_2019, pct_Lucio_Damage_2019, pct_Lucio_Elims_2019, pct_Lucio_Assist_
pct_Lucio_Final_Blows_2020, pct_Lucio_Damage_2020, pct_Lucio_Elims_2020, pct_Lucio_Assist_
pct_Lucio_Final_Blows_2021, pct_Lucio_Damage_2021, pct_Lucio_Elims_2021, pct_Lucio_Assist_
pct_Lucio_Final_Blows_2022, pct_Lucio_Damage_2022, pct_Lucio_Elims_2022, pct_Lucio_Assist_
```

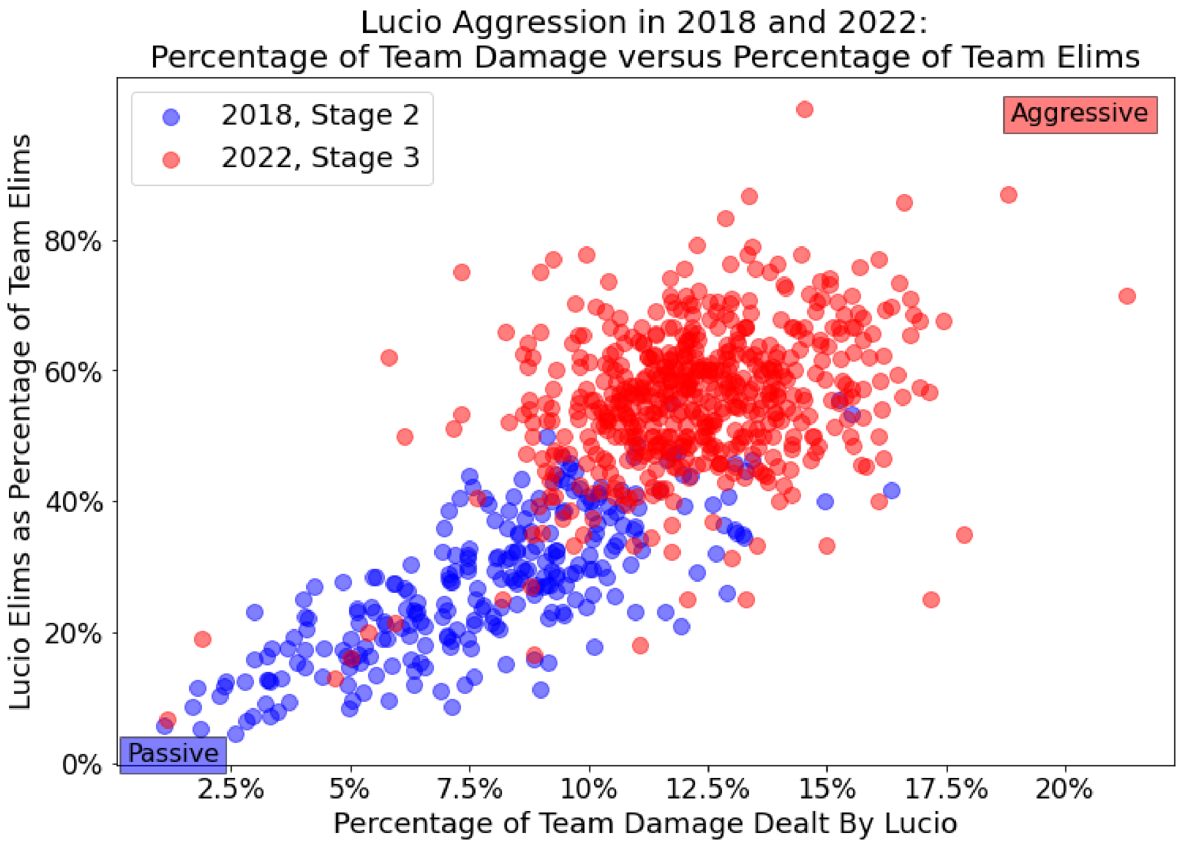
```
plt.hist(pct_Lucio_Assist_2022)
```

```
(array([  1.,   5.,   9.,  34., 126., 212., 104.,  52.,   4.,   3.]),
 array([0.02255639, 0.05030075, 0.07804511, 0.10578947, 0.13353383,
        0.1612782 , 0.18902256, 0.21676692, 0.24451128, 0.27225564,
        0.3       ]),
 <BarContainer object of 10 artists>)
```

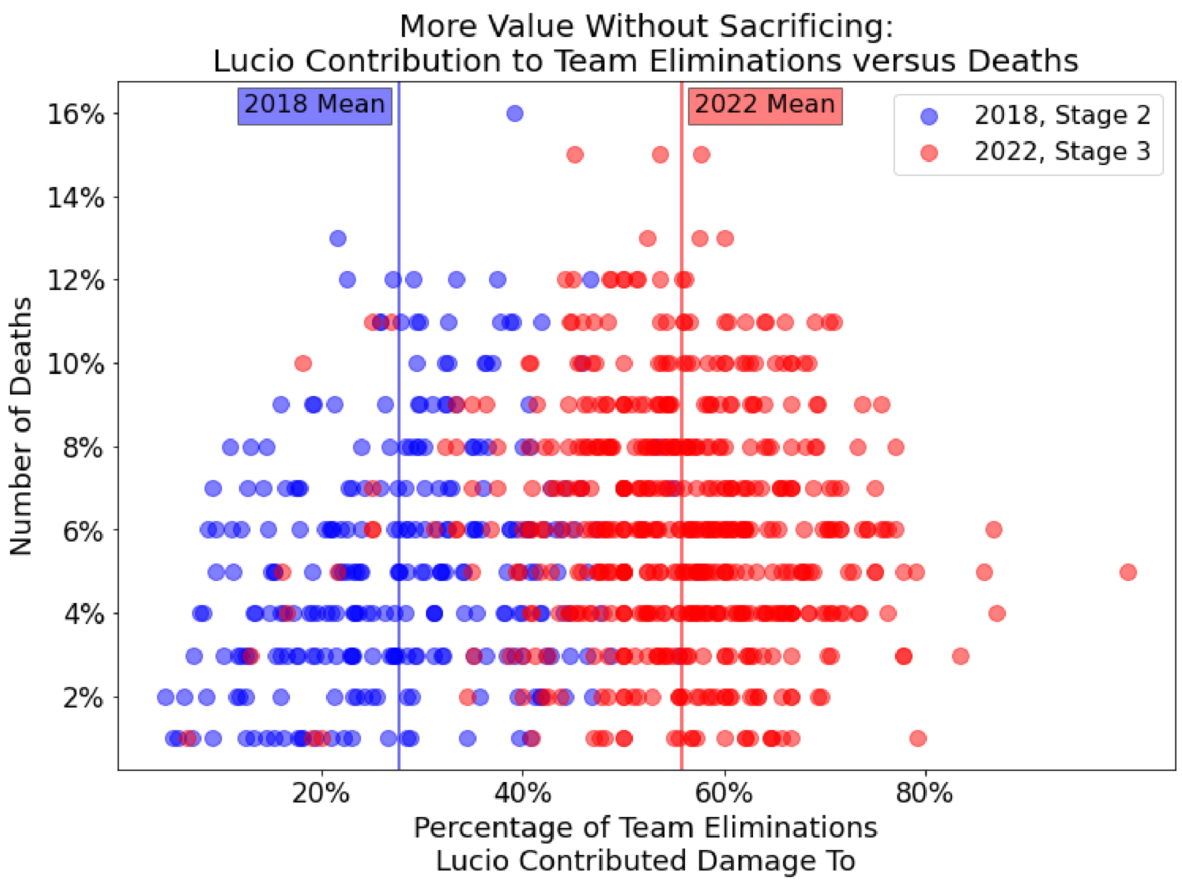


```
plt.figure(figsize=(12,8))

plt.scatter(pct_Lucio_Damage_2018*100, pct_Lucio_Elims_2018*100, c='blue', alpha=0.5, labe
plt.scatter(pct_Lucio_Damage_2022*100, pct_Lucio_Elims_2022*100, c='red', alpha=0.5, label

plt.xlabel('Percentage of Team Damage Dealt By Lucio', fontsize=18)
plt.ylabel('Lucio Elims as Percentage of Team Elims', fontsize=18)

plt.text(18.85, 98, 'Aggressive', fontsize = 16,
         bbox = dict(facecolor = 'red', alpha = 0.5))
plt.text(0.3, 0.3, 'Passive', fontsize = 16,
         bbox = dict(facecolor = 'blue', alpha = 0.5))

plt.title('Lucio Aggression in 2018 and 2022:\nPercentage of Team Damage versus Percentage
plt.legend(loc='upper left', prop={'size': 18})

y_ticks = [0, 20, 40, 60, 80]
y_labels = [str(x) + '%' for x in y_ticks]

x_ticks = [2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20]
x_labels = [str(x) + '%' for x in x_ticks]
```

```
plt.xticks(x_ticks, x_labels)
plt.yticks(y_ticks, y_labels)

plt.xticks()

plt.show()
```



```
plt.figure(figsize=(12,8))

plt.axvline(x = pct_Lucio_Elims_2018.mean()*100, color = 'b', alpha=0.75)
plt.axvline(x = pct_Lucio_Elims_2022.mean()*100, color = 'r', alpha=0.75)

plt.scatter(pct_Lucio_Elims_2018*100, lucio_deaths_2018, c='blue', alpha=0.5, s=100, label
plt.scatter(pct_Lucio_Elims_2022*100, lucio_deaths_2022, c='red', alpha=0.5, s=100, label=
plt.text(12.3, 16, '2018 Mean', fontsize = 16,
         bbox = dict(facecolor = 'blue', alpha = 0.5))
plt.text(57, 16, '2022 Mean', fontsize = 16,
         bbox = dict(facecolor = 'red', alpha = 0.5))

plt.legend(prop={'size': 16})

plt.ylabel('Number of Deaths', fontsize=18)
plt.xlabel('Percentage of Team Eliminations\nLucio Contributed Damage To', fontsize=18)

y_ticks = [2, 4, 6, 8, 10, 12, 14, 16]
y_labels = [str(x) + '%' for x in y_ticks]

x_ticks = [20, 40, 60, 80]
x_labels = [str(x) + '%' for x in x_ticks]

plt.xticks(x_ticks, x_labels)
plt.yticks(y_ticks, y_labels)

plt.xticks()

plt.title('More Value Without Sacrificing:\nLucio Contribution to Team Eliminations versus

plt.show()
```

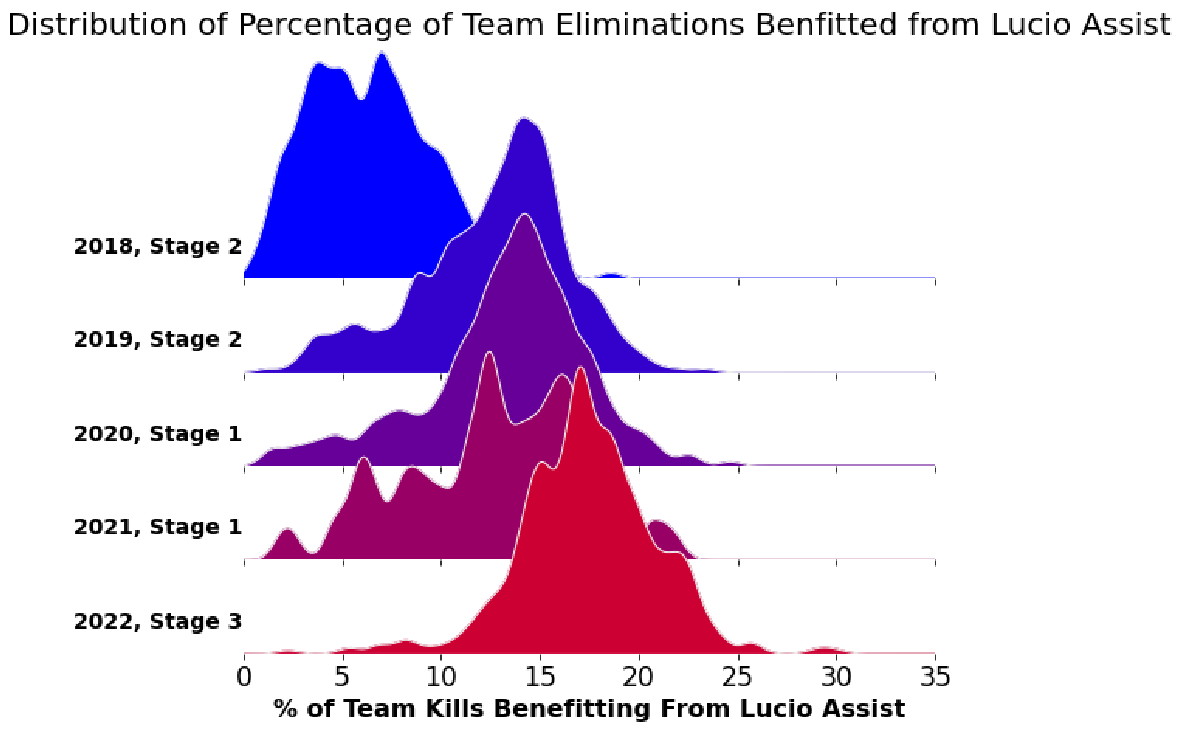


```
from sklearn.neighbors import KernelDensity
import matplotlib as mpl
import matplotlib.gridspec as grid_spec
import matplotlib.pylab as pylab
```

```
seasons = ['2018, Stage 2', '2019, Stage 2', '2020, Stage 1', '2021, Stage 1', '2022, Stag
x_arrays = [pct_Lucio_Assist_2018, pct_Lucio_Assist_2019, pct_Lucio_Assist_2020, pct_Lucio
colors = ['#0000ff', '#3300cc', '#660099', '#990066', '#cc0033']

gs = grid_spec.GridSpec(len(seasons),1)
fig = plt.figure(figsize=(8,10))

i = 0

#plt.tick_params(left = False)
```

```
params = {'xtick.labelsize':'xx-large',
         'ytick.labelsize':'xx-large'}
pylab.rcParams.update(params)

ax_objs = []
for i in range(len(seasons)):
    season = seasons[i]
    x = np.array(x_arrays[i]) * 100
    x_d = np.linspace(0,100, 1000)

    kde = KernelDensity(bandwidth=0.5, kernel='gaussian')
    kde.fit(x[:, None])

    logprob = kde.score_samples(x_d[:, None])

    # creating new axes object
    ax_objs.append(fig.add_subplot(gs[i:i+1, 0:]))

    # plotting the distribution
    ax_objs[-1].plot(x_d, np.exp(logprob),color="#f0f0f0",lw=1)
    ax_objs[-1].fill_between(x_d, np.exp(logprob), alpha=1,color=colors[i])


    # setting uniform x and y lims
    ax_objs[-1].set_xlim(0,35)
    ax_objs[-1].set_ylim(0,0.25)

    # make background transparent
    rect = ax_objs[-1].patch
    rect.set_alpha(0)

    # remove borders, axis ticks, and labels
    ax_objs[-1].set_yticklabels([])
    ax_objs[-1].get_yaxis().set_visible(False)

    if i == len(seasons)-1:
        ax_objs[-1].set_xlabel("% of Team Kills Benefitting From Lucio Assist", fontsize=1
        ax_objs[-1].set_xlabel("% of Team Kills Benefitting From Lucio Assist", fontsize=1
    else:
        ax_objs[-1].set_xticklabels([])
    spines = ["top","right","left","bottom"]
    for s in spines:
        ax_objs[-1].spines[s].set_visible(False)

    adj_season = season + "\n"
    ax_objs[-1].text(-0.02,0, adj_season, fontweight="bold", fontsize=14, ha="right")

gs.update(hspace=-0.8)

#fig.text(-0.1,0.85,"Distribution of Percentage of Team Kills Benfitted from Lucio Assist"
plt.title("Distribution of Percentage of Team Eliminations Benfitted from Lucio Assist", f

plt.tight_layout()
plt.show()
```

```
/var/folders/q9/v0k4_bgj7d31b7zlsnys9hsh0000gn/T/ipykernel_75351/1143940732.py:64: UserWarnin
  plt.tight_layout()
```


Distribution of Percentage of Team Eliminations Benfitted from Lucio Assist
2018, Stage 2
2019, Stage 2
2020, Stage 1
2021, Stage 1
2022, Stage 3
0
5
10
15
20
25
30
35
% of Team Kills Benefitting From Lucio Assist