1. TV, halftime shows, and the Big Game

Whether or not you like football, the Super Bowl is a spectacle. There's a little something for everyone at your Super Bowl party. Drama in the form of blowouts, comebacks, and controversy for the sports fan. There are the ridiculously expensive ads, some hilarious, others gut-wrenching, thought-provoking, and weird. The half-time shows with the biggest musicians in the world, sometimes riding giant mechanical tigers or leaping from the roof of the stadium. It's a show, baby. And in this notebook, we're going to find out how some of the elements of this show interact with each other. After exploring and cleaning our data a little, we're going to answer questions like:

- What are the most extreme game outcomes?
- How does the game affect television viewership?
- How have viewership, TV ratings, and ad cost evolved over time?
- Who are the most prolific musicians in terms of halftime show performances?

Left Shark Steals The Show. Katy Perry performing at halftime of Super Bowl XLIX. Photo by Huntley Paton. Attribution-ShareAlike 2.0 Generic (CC BY-SA 2.0).

The dataset we'll use was scraped and polished from Wikipedia. It is made up of three CSV files, one with game data, one with TV data, and one with halftime musician data for all 52 Super Bowls through 2018. Let's take a look, using `display()` instead of `print()` since its output is much prettier in Jupyter Notebooks.

In [153]:

```
# Import pandas
import pandas as pd
```

```python
# Load the CSV data into DataFrames
super_bowls = pd.read_csv('datasets/super_bowls.csv')
tv = pd.read_csv('datasets/tv.csv')
halftime_musicians = pd.read_csv('datasets/halftime_musicians.csv')

# Display the first five rows of each DataFrame
display(super_bowls.head())
display(tv.head())
display(halftime_musicians.head())
```

	date	super_bowl	venue	city	state	attendance	team_winner	winning_pts	qb_winner_1	qb_winner_2
0	2018-02-04	52	U.S. Bank Stadium	Minneapolis	Minnesota	67612	Philadelphia Eagles	41	Nick Foles	NaN
1	2017-02-05	51	NRG Stadium	Houston	Texas	70807	New England Patriots	34	Tom Brady	NaN
2	2016-02-07	50	Levi's Stadium	Santa Clara	California	71088	Denver Broncos	24	Peyton Manning	NaN
3	2015-02-01	49	University of Phoenix Stadium	Glendale	Arizona	70288	New England Patriots	28	Tom Brady	NaN
4	2014-02-02	48	MetLife Stadium	East Rutherford	New Jersey	82529	Seattle Seahawks	43	Russell Wilson	NaN

	super_bowl	network	avg_us_viewers	total_us_viewers	rating_household	share_household	rating_18_49	share_18_49	ad_
0	52	NBC	103390000	NaN	43.1	68	33.4	78.0	500
1	51	Fox	111319000	172000000.0	45.3	73	37.1	79.0	500
2	50	CBS	111864000	167000000.0	46.6	72	37.7	79.0	500
3	49	NBC	114442000	168000000.0	47.5	71	39.1	79.0	450
4	48	Fox	112191000	167000000.0	46.7	69	39.3	77.0	400

	super_bowl	musician	num_songs
0	52	Justin Timberlake	11.0
1	52	University of Minnesota Marching Band	1.0
2	51	Lady Gaga	7.0
3	50	Coldplay	6.0
4	50	Beyoncé	3.0

2. Taking note of dataset issues

For the Super Bowl game data, we can see the dataset appears whole except for missing values in the backup quarterback columns (`qb_winner_2` and `qb_loser_2`), which make sense given most starting QBs in the Super Bowl (`qb_winner_1` and `qb_loser_1`) play the entire game.

From the visual inspection of TV and halftime musicians data, there is only one missing value displayed, but I've got a hunch there are more. The Super Bowl goes all the way back to 1967, and the more granular columns (e.g. the number of songs for halftime musicians) probably weren't tracked reliably over time. Wikipedia is great but not perfect.

An inspection of the `.info()` output for `tv` and `halftime_musicians` shows us that there are multiple columns with null values.

In [155]:

```
# Summary of the TV data to inspect
tv.info()

print('\n')

# Summary of the halftime musician data to inspect
halftime_musicians.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53 entries, 0 to 52
Data columns (total 9 columns):
super_bowl          53 non-null int64
network             53 non-null object
avg_us_viewers      53 non-null int64
total_us_viewers    15 non-null float64
rating_household    53 non-null float64
share_household     53 non-null int64
rating_18_49        15 non-null float64
share_18_49         6 non-null float64
ad_cost             53 non-null int64
dtypes: float64(4), int64(4), object(1)
memory usage: 3.8+ KB


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134 entries, 0 to 133
Data columns (total 3 columns):
super_bowl    134 non-null int64
musician      134 non-null object
num_songs     88 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 3.2+ KB
```

3. Combined points distribution

For the TV data, the following columns have missing values and a lot of them:

- `total_us_viewers` **(amount of U.S. viewers who watched at least some part of the broadcast)**
- `rating_18_49` **(average % of U.S. adults 18-49 who live in a household with a TV that were watching for the entire broadcast)**
- `share_18_49` **(average % of U.S. adults 18-49 who live in a household with a TV *in use* that were watching for the entire broadcast)**

For the halftime musician data, there are missing numbers of songs performed (`num_songs` **) for about a third of the performances.**

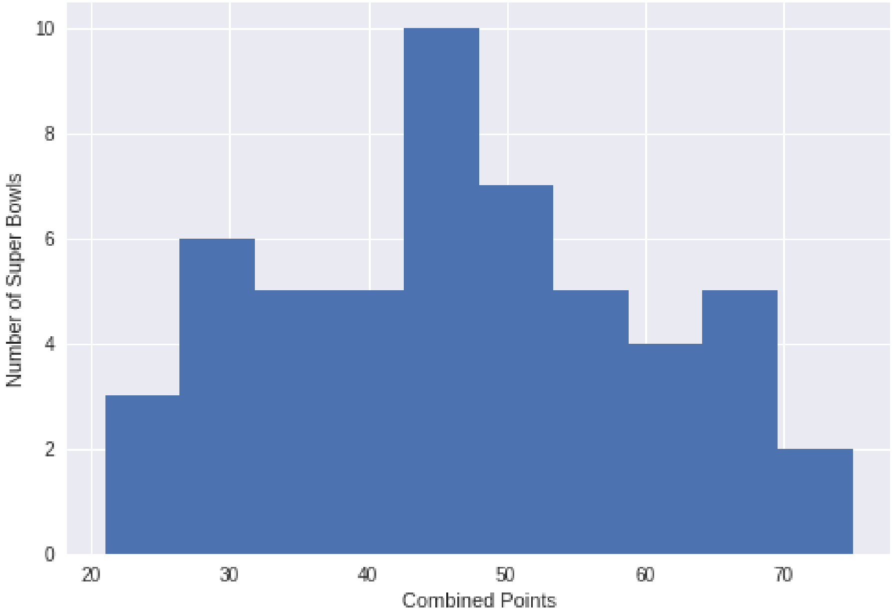
There are a lot of potential reasons for these missing values. Was the data ever tracked? Was it lost in history? Is the research effort to make this data whole worth it? Maybe. Watching every Super Bowl halftime show to get song counts would be pretty fun. But we don't have the time to do that kind of stuff now! Let's take note of where the dataset isn't perfect and start uncovering some insights.

Let's start by looking at combined points for each Super Bowl by visualizing the distribution. Let's also pinpoint the Super Bowls with the highest and lowest scores.

In [157]:

```
# Import matplotlib and set plotting style
from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('seaborn')

# Plot a histogram of combined points
plt.hist(super_bowls['combined_pts'])
plt.xlabel('Combined Points')
plt.ylabel('Number of Super Bowls')
plt.show()
```

```
# Display the Super Bowls with the highest and lowest combined scores
display(super_bowls[super_bowls['combined_pts'] > 70])
display(super_bowls[super_bowls['combined_pts'] < 25])
```



	date	super_bowl	venue	city	state	attendance	team_winner	winning_pts	qb_winner_1	qb_winner_2	c
0	2018-02-04	52	U.S. Bank Stadium	Minneapolis	Minnesota	67612	Philadelphia Eagles	41	Nick Foles	NaN	
23	1995-01-29	29	Joe Robbie Stadium	Miami Gardens	Florida	74107	San Francisco 49ers	49	Steve Young	NaN	

	date	super_bowl	venue	city	state	attendance	team_winner	winning_pts	qb_winner_1	qb_winner_2	coac
43	1975-01-12	9	Tulane Stadium	New Orleans	Louisiana	80997	Pittsburgh Steelers	16	Terry Bradshaw	NaN	Cl
45	1973-01-14	7	Memorial Coliseum	Los Angeles	California	90182	Miami Dolphins	14	Bob Griese	NaN	D
49	1969-01-12	3	Orange Bowl	Miami	Florida	75389	New York Jets	16	Joe Namath	NaN	

4. Point difference distribution

Most combined scores are around 40-50 points, with the extremes being roughly equal distance away in opposite directions. Going up to the highest combined scores at 74 and 75, we find two games featuring dominant quarterback performances. One even happened recently in 2018's Super Bowl LII where Tom Brady's Patriots lost to Nick Foles' underdog Eagles 41-33 for a combined score of 74.

Going down to the lowest combined scores, we have Super Bowl III and VII, which featured tough defenses that dominated. We also have Super Bowl IX in New Orleans in 1975, whose 16-6 score can be attributed to inclement weather. The field was slick from overnight rain, and it was cold at 46 °F (8 °C), making it hard for the Steelers and Vikings to do much offensively. This was the second-coldest Super Bowl ever and the last to be played in inclement weather for over 30 years. The NFL realized people like points, I guess.

UPDATE: In Super Bowl LIII in 2019, the Patriots and Rams broke the record for the lowest-scoring Super Bowl with a combined score of 16 points (13-3 for the Patriots).

Let's take a look at point *difference* now.

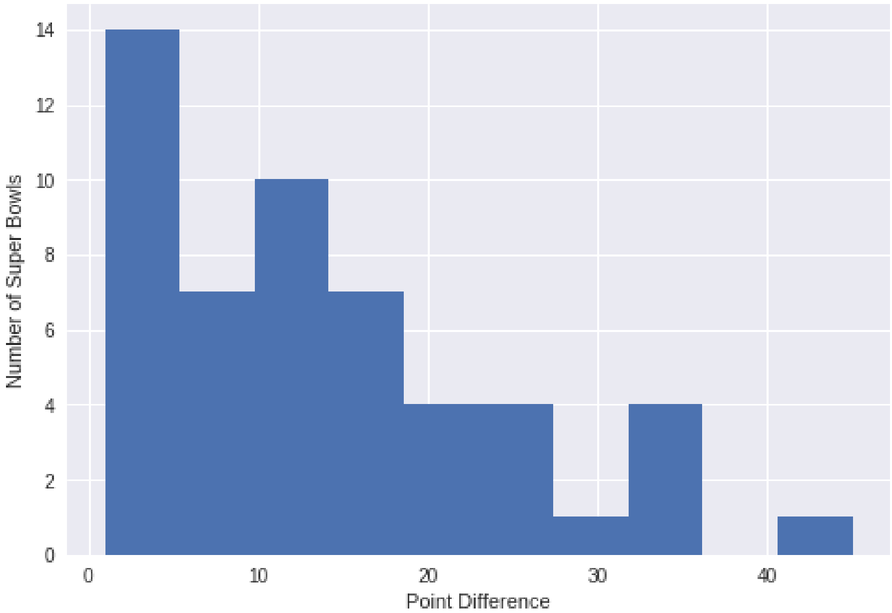
In [159]:

```
# Plot a histogram of point differences
```

```
# Plot a histogram of point differences
plt.hist(super_bowls.difference_pts)
plt.xlabel('Point Difference')
plt.ylabel("Number of Super Bowls")

# Display the closest game(s) and biggest blowouts
display(super_bowls[super_bowls['difference_pts'] == 1])
display(super_bowls[super_bowls['difference_pts'] >= 35])
```

	date	super_bowl	venue	city	state	attendance	team_winner	winning_pts	qb_winner_1	qb_winner_2	coach_wi
27	1991-01-27	25	Tampa Stadium	Tampa	Florida	73813	New York Giants	20	Jeff Hostetler	NaN	Bill Parc

	date	super_bowl	venue	city	state	attendance	team_winner	winning_pts	qb_winner_1	qb_winner_2
4	2014-02-02	48	MetLife Stadium	East Rutherford	New Jersey	82529	Seattle Seahawks	43	Russell Wilson	NaN
25	1993-01-31	27	Rose Bowl	Pasadena	California	98374	Dallas Cowboys	52	Troy Aikman	NaN
28	1990-01-28	24	Louisiana Superdome	New Orleans	Louisiana	72919	San Francisco 49ers	55	Joe Montana	NaN
32	1986-01-26	20	Louisiana Superdome	New Orleans	Louisiana	73818	Chicago Bears	46	Jim McMahon	NaN



5. Do blowouts translate to lost viewers?

The vast majority of Super Bowls are close games. Makes sense. Both teams are likely to be deserving if they've made it this far. The closest game ever was when the Buffalo Bills lost to the New York Giants by 1 point in 1991, which was best remembered for Scott Norwood's last-second missed field goal attempt that went *wide right*, kicking off four Bills Super Bowl losses in a row. Poor Scott. The biggest point discrepancy ever was 45 points (!) where Hall of Famer Joe Montana's led the San Francisco 49ers to victory in 1990, one year before the closest game ever.

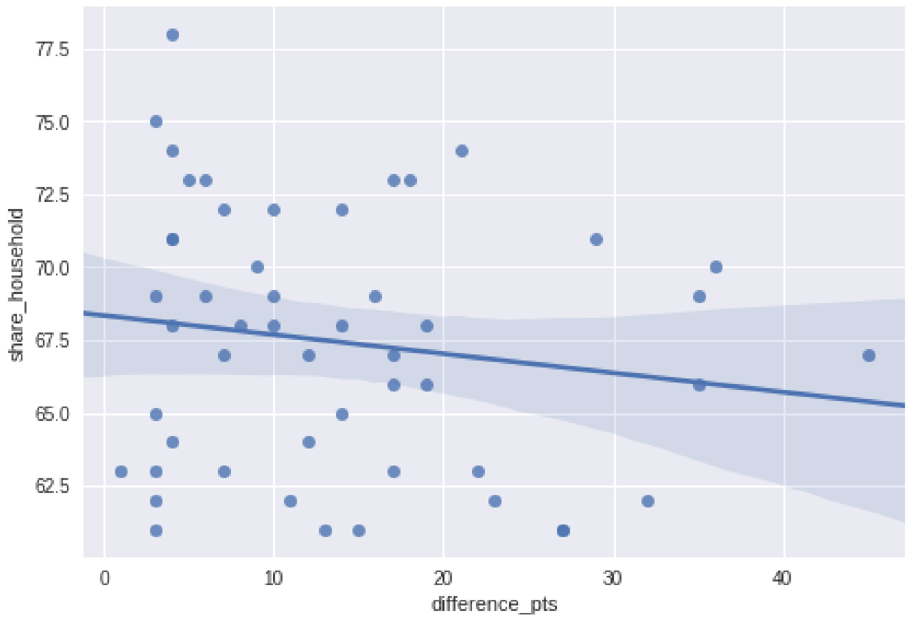
I remember watching the Seahawks crush the Broncos by 35 points (43-8) in 2014, which was a boring experience in my opinion. The game was never really close. I'm pretty sure we changed the channel at the end of the third quarter. Let's combine our game data and TV to see if this is a universal phenomenon. Do large point differences translate to lost viewers? We can plot household share *(average percentage of U.S. households with a TV in use that were watching for the entire broadcast)* vs. point difference to find out.

In [161]:

```python
# Join game and TV data, filtering out SB I because it was split over two networks
games_tv = pd.merge(tv[tv['super_bowl'] > 1], super_bowls, on='super_bowl')

# Import seaborn
import seaborn as sns

# Create a scatter plot with a linear regression model fit
sns.regplot(x='difference_pts', y='share_household', data=games_tv)
```

Out[161]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f67a4306908>
```



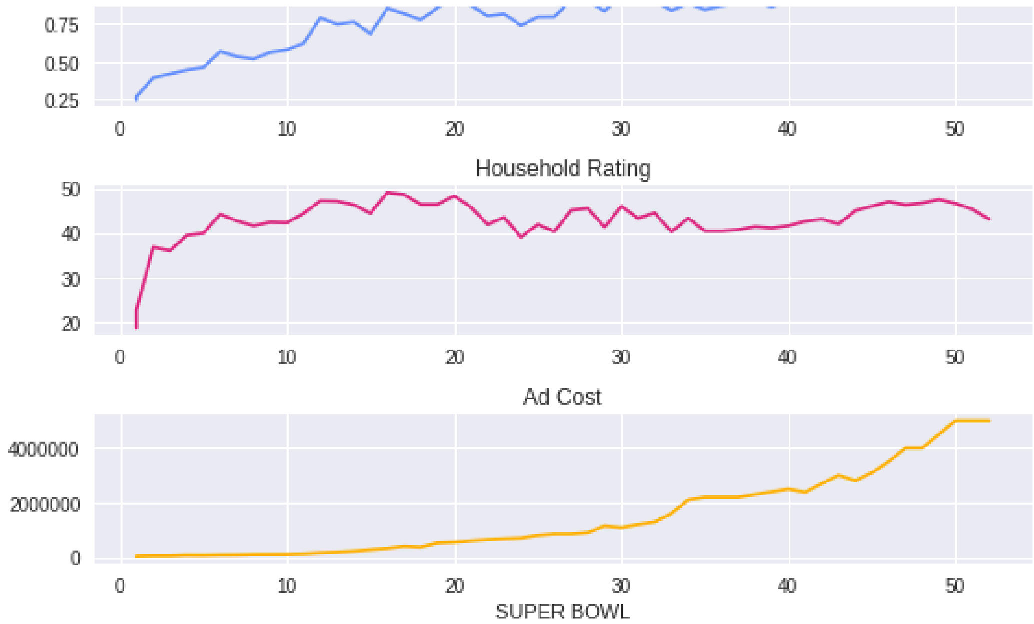
6. Viewership and the ad industry over time

The downward sloping regression line and the 95% confidence interval for that regression *suggest* that bailing on the game if it is a blowout is common. Though it matches our intuition, we must take it with a grain of salt because the linear relationship in the data is weak due to our small sample size of 52 games.

Regardless of the score though, I bet most people stick it out for the halftime show, which is good news for the TV networks and advertisers. A 30-second spot costs a pretty [$5 million] now, but has it always been that way? And how have number of viewers and household ratings trended alongside ad cost? We can find out using line plots that share a "Super Bowl" x-axis.

In [163]:

```python
# Create a figure with 3x1 subplot and activate the top subplot
plt.subplot(3, 1, 1)
plt.plot(tv.super_bowl, tv.avg_us_viewers, color='#648FFF')
plt.title('Average Number of US Viewers')

# Activate the middle subplot
plt.subplot(3, 1, 2)
plt.plot(tv.super_bowl, tv.rating_household, color='#DC267F')
plt.title('Household Rating')

# Activate the bottom subplot
plt.subplot(3, 1, 3)
plt.plot(tv.super_bowl, tv.ad_cost, color='#FFB000')
plt.title('Ad Cost')
plt.xlabel('SUPER BOWL')

# Improve the spacing between subplots
plt.tight_layout()
```



7. Halftime shows weren't always this great

We can see viewers increased before ad costs did. Maybe the networks weren't very data savvy and were slow to react? Makes sense since DataCamp didn't exist back then.

Another hypothesis: maybe halftime shows weren't that good in the earlier years? The modern spectacle of the Super Bowl has a lot to do with the cultural prestige of big halftime acts. I went down a YouTube rabbit hole and it turns out the old ones weren't up to today's standards. Some offenders:

- **Super Bowl XXVI in 1992: A Frosty The Snowman rap performed by children.**
- **Super Bowl XXIII in 1989: An Elvis impersonator that did magic tricks and didn't even sing one Elvis song.**
- **Super Bowl XXI in 1987: Tap dancing ponies. (Okay, that's pretty awesome actually.)**

It turns out Michael Jackson's Super Bowl XXVII performance, one of the most watched events in American TV history, was when the NFL realized the value of Super Bowl airtime and decided they needed to sign big name acts from then on out. The halftime shows before MJ indeed weren't that impressive, which we can see by filtering our `halftime_musician` data.

In [165]:

```
# Display all halftime musicians for Super Bowls up to and including Super Bowl XXVII
halftime_musicians[halftime_musicians.super_bowl <= 27]
```

Out[165]:

	super_bowl	musician	num_songs
80	27	Michael Jackson	5.0
81	26	Gloria Estefan	2.0
82	26	University of Minnesota Marching Band	NaN
83	25	New Kids on the Block	2.0
84	24	Pete Fountain	1.0
85	24	Doug Kershaw	1.0
86	24	Irma Thomas	1.0
87	24	Pride of Nicholls Marching Band	NaN
88	24	The Human Jukebox	NaN
89	24	Pride of Acadiana	NaN
90	23	Elvis Presto	7.0
91	22	Chubby Checker	2.0
92	22	San Diego State University Marching Aztecs	NaN

	super_bowl	musician	num_songs
93	22	Spirit of Troy	NaN
94	21	Grambling State University Tiger Marching Band	8.0
95	21	Spirit of Troy	8.0
96	20	Up with People	NaN
97	19	Tops In Blue	NaN
98	18	The University of Florida Fightin' Gator March...	7.0
99	18	The Florida State University Marching Chiefs	7.0
100	17	Los Angeles Unified School District All City H...	NaN
101	16	Up with People	NaN
102	15	The Human Jukebox	NaN
103	15	Helen O'Connell	NaN
104	14	Up with People	NaN
105	14	Grambling State University Tiger Marching Band	NaN
106	13	Ken Hamilton	NaN
107	13	Gramacks	NaN
108	12	Tyler Junior College Apache Band	NaN
109	12	Pete Fountain	NaN
110	12	Al Hirt	NaN
111	11	Los Angeles Unified School District All City H...	NaN
112	10	Up with People	NaN
113	9	Mercer Ellington	NaN
114	9	Grambling State University Tiger Marching Band	NaN
115	8	University of Texas Longhorn Band	NaN
116	8	Judy Mallett	NaN
117	7	University of Michigan Marching Band	NaN
118	7	Woody Herman	NaN
119	7	Andy Williams	NaN
120	6	Ella Fitzgerald	NaN
121	6	Carol Channing	NaN
122	6	Al Hirt	NaN
123	6	United States Air Force Academy Cadet Chorale	NaN
124	5	Southeast Missouri State Marching Band	NaN
125	4	Marguerite Piazza	NaN
126	4	Doc Severinsen	NaN
127	4	Al Hirt	NaN
128	4	The Human Jukebox	NaN
129	3	Florida A&M University Marching 100 Band	NaN
130	2	Grambling State University Tiger Marching Band	NaN
131	1	University of Arizona Symphonic Marching Band	NaN
132	1	Grambling State University Tiger Marching Band	NaN
133	1	Al Hirt	NaN

8. Who has the most halftime show appearances?

Lots of marching bands. American jazz clarinetist Pete Fountain. Miss Texas 1973 playing a violin. Nothing against those performers, they're just simply not [Beyoncé](). To be fair, no one is.

Let's see all of the musicians that have done more than one halftime show, including their performance counts.

In [167]:

```
# Count halftime show appearances for each musician and sort them from most to least
halftime_appearances = halftime_musicians.groupby('musician').count()['super_bowl'].rese
t_index()
halftime_appearances = halftime_appearances.sort_values('super_bowl', ascending=False)

# Display musicians with more than one halftime show appearance
halftime_appearances[halftime_appearances.super_bowl > 1]
```

Out[167]:

	musician	super_bowl
28	Grambling State University Tiger Marching Band	6
104	Up with People	4
1	Al Hirt	4
83	The Human Jukebox	3
76	Spirit of Troy	2
25	Florida A&M University Marching 100 Band	2
26	Gloria Estefan	2
102	University of Minnesota Marching Band	2
10	Bruno Mars	2
64	Pete Fountain	2
5	Beyoncé	2
36	Justin Timberlake	2
57	Nelly	2
44	Los Angeles Unified School District All City H...	2

9. Who performed the most songs in a halftime show?

The world famous [Grambling State University Tiger Marching Band]() takes the crown with six appearances. Beyoncé, Justin Timberlake, Nelly, and Bruno Mars are the only post-Y2K musicians with multiple appearances (two each).

From our previous inspections, the `num_songs` column has lots of missing values:

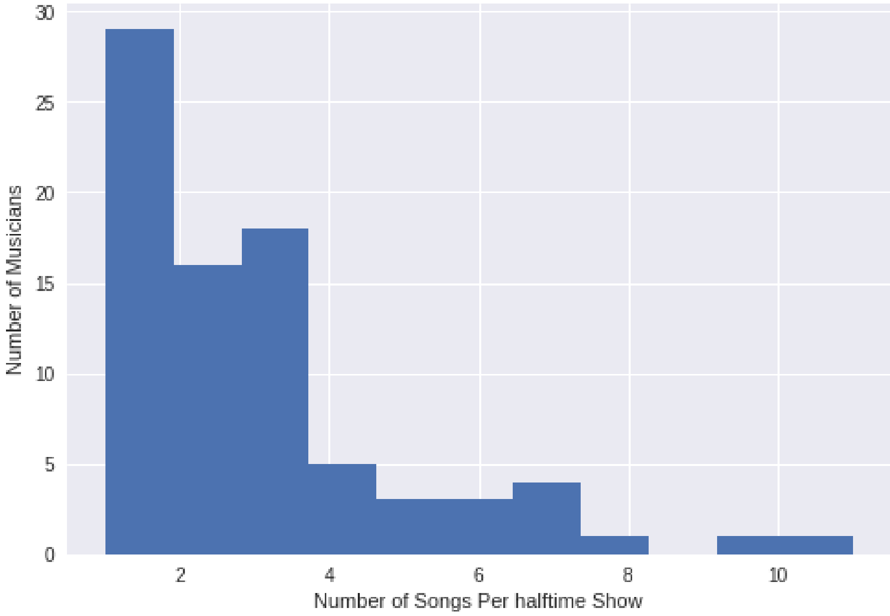
- **A lot of the marching bands don't have `num_songs` entries.**
- **For non-marching bands, missing data starts occurring at Super Bowl XX.**

Let's filter out marching bands by filtering out musicians with the word "Marching" in them and the word "Spirit" (a common naming convention for marching bands is "Spirit of [something]"). Then we'll filter for Super Bowls after Super Bowl XX to address the missing data issue, *then* let's see who has the most number of songs.

In [169]:

```
# Filter out most marching bands
no_bands = halftime_musicians[~halftime_musicians.musician.str.contains('Marching')]
no_bands = no_bands[~no_bands.musician.str.contains('Spirit')]

# Plot a histogram of number of songs per performance
```

```
most_songs = int(max(no_bands['num_songs'].values))
plt.hist(no_bands.num_songs.dropna(), bins=most_songs)
plt.xlabel("Number of Songs Per halftime Show")
plt.ylabel('Number of Musicians')
plt.show()

# Sort the non-band musicians by number of songs per appearance...
no_bands = no_bands.sort_values('num_songs', ascending=False)
# ...and display the top 15
display(no_bands.head(15))
```



	super_bowl	musician	num_songs
0	52	Justin Timberlake	11.0
70	30	Diana Ross	10.0
10	49	Katy Perry	8.0
2	51	Lady Gaga	7.0
90	23	Elvis Presto	7.0
33	41	Prince	7.0
16	47	Beyoncé	7.0
14	48	Bruno Mars	6.0
3	50	Coldplay	6.0
25	45	The Black Eyed Peas	6.0
20	46	Madonna	5.0
30	44	The Who	5.0
80	27	Michael Jackson	5.0
64	32	The Temptations	4.0
36	39	Paul McCartney	4.0

10. Conclusion

So most non-band musicians do 1-3 songs per halftime show. It's important to note that the duration of the halftime show is fixed (roughly 12 minutes) so songs per performance is more a measure of how many hit songs you have. JT went off in 2018, wow. 11 songs! Diana Ross comes in second with 10 in her medley in 1996.

In this notebook, we loaded, cleaned, then explored Super Bowl game, television, and halftime show data. We visualized the distributions of combined points, point differences, and halftime show performances using histograms. We used line plots to see how ad cost increases lagged behind viewership increases. And we discovered that blowouts do appear to lead to a drop in viewers.

This year's Big Game will be here before you know it. Who do you think will win Super Bowl LIII?

UPDATE:* [*Spoiler alert*]*.

In [171]:

```
# 2018-2019 conference champions
patriots = 'New England Patriots'
rams = 'Los Angeles Rams'

# Who will win Super Bowl LIII?
super_bowl_LIII_winner = rams
print('The winner of Super Bowl LIII will be the', super_bowl_LIII_winner)
```

```
The winner of Super Bowl LIII will be the Los Angeles Rams
```

1. The Statcast revolution

This is Aaron Judge. Judge is one of the physically largest players in Major League Baseball standing 6 feet 7 inches (2.01 m) tall and weighing 282 pounds (128 kg). He also hit the [hardest home run]() ever recorded. How do we know this? **Statcast.**

Statcast is a state-of-the-art tracking system that uses high-resolution cameras and radar equipment to measure the precise location and movement of baseballs and baseball players. Introduced in 2015 to all 30 major league ballparks, Statcast data is revolutionizing the game. Teams are engaging in an "arms race" of data analysis, hiring analysts left and right in an attempt to gain an edge over their competition. This [video]() describing the system is incredible.

In this notebook, we're going to wrangle, analyze, and visualize Statcast data to compare Mr. Judge and another (extremely large) teammate of his. Let's start by loading the data into our Notebook. There are two CSV files, `judge.csv` and `stanton.csv`, both of which contain Statcast data for 2015-2017. We'll use pandas DataFrames to store this data. Let's also load our data visualization libraries, matplotlib and seaborn.

In [97]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Load Aaron Judge's Statcast data
judge = pd.read_csv('datasets/judge.csv')

# Load Giancarlo Stanton's Statcast data
stanton = pd.read_csv("datasets/stanton.csv")
```

2. What can Statcast measure?

The better question might be, what can't Statcast measure?

> Starting with the pitcher, Statcast can measure simple data points such as velocity. At the same time, Statcast digs a whole lot deeper, also measuring the release point and spin rate of every pitch.
>
> Moving on to hitters, Statcast is capable of measuring the exit velocity, launch angle and vector of the ball as it comes off the bat. From there, Statcast can also track the hang time and projected distance that a ball travels.

Let's inspect the last five rows of the `judge` DataFrame. You'll see that each row represents one pitch thrown to a batter. You'll also see that some columns have esoteric names. If these don't make sense now, don't worry. The relevant ones will be explained as necessary.

In [99]:

```
# Display all columns (pandas will collapse some columns if we don't set this option)
pd.set_option('display.max_columns', None)

# Display the last five rows of the Aaron Judge file
```

```
judge.tail()
```

Out[99]:

	pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_name	batter	pitcher	events	c
3431	CH	2016-08-13	85.6	-1.9659	5.9113	Aaron Judge	592450	542882	NaN	
3432	CH	2016-08-13	87.6	-1.9318	5.9349	Aaron Judge	592450	542882	home_run	hit_into_
3433	CH	2016-08-13	87.2	-2.0285	5.8656	Aaron Judge	592450	542882	NaN	
3434	CU	2016-08-13	79.7	-1.7108	6.1926	Aaron Judge	592450	542882	NaN	
3435	FF	2016-08-13	93.2	-1.8476	6.0063	Aaron Judge	592450	542882	NaN	ca

3. Aaron Judge and Giancarlo Stanton, prolific sluggers

This is Giancarlo Stanton. He is also a very large human being, standing 6 feet 6 inches tall and weighing 245 pounds. Despite not wearing the same jersey as Judge in the pictures provided, in 2018 they will be teammates on the New York Yankees. They are similar in a lot of ways, one being that they hit a lot of home runs. Stanton and Judge led baseball in home runs in 2017, with [59] and [52], respectively. These are exceptional totals - the player in third "only" had 45 home runs.

Stanton and Judge are also different in many ways. One is [batted ball events], which is any batted ball that produces a result. This includes outs, hits, and errors. Next, you'll find the counts of batted ball events for each player in 2017. The frequencies of other events are quite different.

In [101]:

```
# All of Aaron Judge's batted ball events in 2017
judge_events_2017 = judge.loc[judge["game_year"] == 2017].events
print("Aaron Judge batted ball event totals, 2017:")
print(judge_events_2017.value_counts())

# All of Giancarlo Stanton's batted ball events in 2017
stanton_events_2017 = stanton.loc[stanton["game_year"] ==2017].events

print("\nGiancarlo Stanton batted ball event totals, 2017:")
print(stanton_events_2017.value_counts())
```

```
Aaron Judge batted ball event totals, 2017:
strikeout                    207
field_out                    146
walk                         116
single                        75
home_run                      52
double                        24
grounded_into_double_play     15
force_out                     11
intent_walk                   11
```

```
intent_walk                   11
hit_by_pitch                   5
sac_fly                        4
field_error                    4
fielders_choice_out            4
triple                         3
strikeout_double_play          1
Name: events, dtype: int64

Giancarlo Stanton batted ball event totals, 2017:
field_out                    239
strikeout                    161
single                        77
walk                          72
home_run                      59
double                        32
grounded_into_double_play     13
intent_walk                   13
force_out                      7
hit_by_pitch                   7
field_error                    5
sac_fly                        3
strikeout_double_play          2
fielders_choice_out            2
pickoff_1b                     1
Name: events, dtype: int64
```

4. Analyzing home runs with Statcast data

So Judge walks and strikes out more than Stanton. Stanton flies out more than Judge. But let's get into their hitting profiles in more detail. Two of the most groundbreaking Statcast metrics are launch angle and exit velocity:

- **Launch angle: the vertical angle at which the ball leaves a player's bat**
- **Exit velocity: the speed of the baseball as it comes off the bat**

This new data has changed the way teams value both hitters and pitchers. Why? As per the Washington Post:

> **Balls hit with a high launch angle are more likely to result in a hit. Hit fast enough and at the right angle, they become home runs.**

Let's look at exit velocity vs. launch angle and let's focus on home runs only (2015-2017). The first two plots show data points. The second two show smoothed contours to represent density.

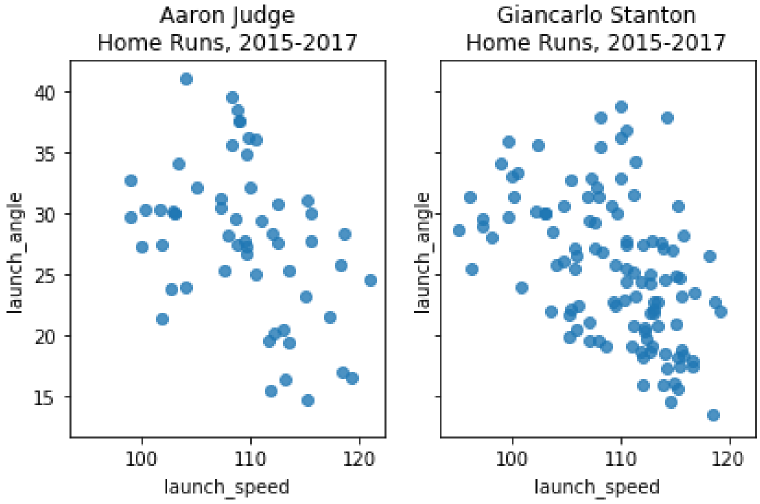
In [103]:

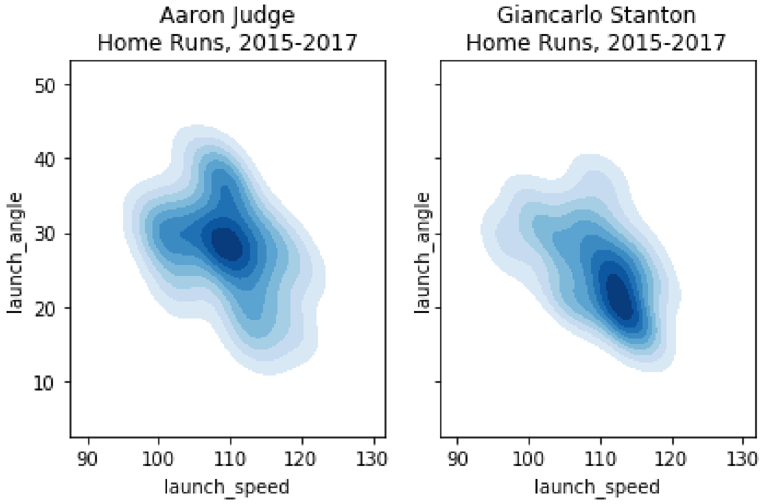
```
# Filter to include home runs only



judge_hr = judge.loc[judge["events"] == 'home_run' ]
stanton_hr = stanton.loc[stanton["events"] == 'home_run']




# Create a figure with two scatter plots of launch speed vs. launch angle, one for each p
layer's home runs
fig1, axs1 = plt.subplots(ncols=2, sharex=True, sharey=True)
sns.regplot(x='launch_speed', y='launch_angle', fit_reg=False, color='tab:blue', data=ju
dge_hr, ax=axs1[0]).set_title('Aaron Judge\nHome Runs, 2015-2017')
sns.regplot(x='launch_speed', y='launch_angle', fit_reg=False, color='tab:blue', data=st
anton_hr, ax=axs1[1]).set_title('Giancarlo Stanton\nHome Runs, 2015-2017')

# Create a figure with two KDE plots of launch speed vs. launch angle, one for each playe
r's home runs
fig2, axs2 = plt.subplots(ncols=2, sharex=True, sharey=True)
sns.kdeplot(judge_hr['launch_speed'], judge_hr['launch_angle'], cmap="Blues", shade=True,
```

```
shade_lowest=False, ax=axs2[0]).set_title('Aaron Judge\nHome Runs, 2015-2017')
sns.kdeplot(stanton_hr['launch_speed'], stanton_hr['launch_angle'], cmap="Blues", shade=
True, shade_lowest=False, ax=axs2[1]).set_title('Giancarlo Stanton\nHome Runs, 2015-2017
')
```

Out[103]:

```
Text(0.5,1,'Giancarlo Stanton\nHome Runs, 2015-2017')
```





5. Home runs by pitch velocity

It appears that Stanton hits his home runs slightly lower and slightly harder than Judge, though this needs to be taken with a grain of salt given the small sample size of home runs.

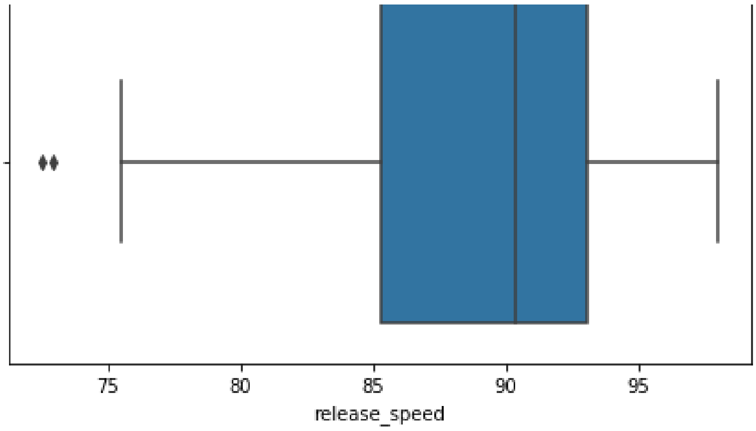
Not only does Statcast measure the velocity of the ball coming off of the bat, it measures the velocity of the ball coming out of the pitcher's hand and begins its journey towards the plate. We can use this data to compare Stanton and Judge's home runs in terms of pitch velocity. Next you'll find box plots displaying the five-number summaries for each player: minimum, first quartile, median, third quartile, and maximum.

In [105]:

```
# Combine the Judge and Stanton home run DataFrames for easy boxplot plotting
judge_stanton_hr = pd.concat([judge_hr, stanton_hr])


# Create a boxplot that describes the pitch velocity of each player's home runs
sns.boxplot(x=judge_stanton_hr['release_speed'], color='tab:blue').set_title('Home Runs,
2015-2017')
```

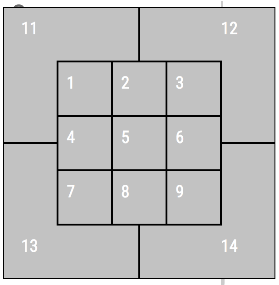
Out[105]:

```
Text(0.5,1,'Home Runs, 2015-2017')
```



6. Home runs by pitch location (I)

So Judge appears to hit his home runs off of faster pitches than Stanton. We might call Judge a fastball hitter. Stanton appears agnostic to pitch speed and likely pitch movement since slower pitches (e.g. curveballs, sliders, and changeups) tend to have more break. Statcast *does* track pitch movement and type but let's move on to something else: pitch location. Statcast tracks the zone the pitch is in when it crosses the plate. The zone numbering looks like this (from the catcher's point of view):



We can plot this using a 2D histogram. For simplicity, let's only look at strikes, which gives us a 9x9 grid. We can view each zone as coordinates on a 2D plot, the bottom left corner being (1,1) and the top right corner being (3,3). Let's set up a function to assign x-coordinates to each pitch.

In [107]:

```
def assign_x_coord(row):
    """
    Assigns an x-coordinate to Statcast's strike zone numbers. Zones 11, 12, 13,
    and 14 are ignored for plotting simplicity.
    """
    # Left third of strike zone
    if row.zone in [1, 4, 7]:
        return 1
    # Middle third of strike zone
    if row.zone in [2, 5, 8]:
        return 2
    # Right third of strike zone
    if row.zone in [3, 6, 9]:
        return 3
```

7. Home runs by pitch location (II)

And let's do the same but for y-coordinates.

In [109]:

```
def assign_y_coord(row):
    """
    Assigns a y-coordinate to Statcast's strike zone numbers. Zones 11, 12, 13,
    and 14 are ignored for plotting simplicity.
    """
    if row.zone in [1, 2, 3]:
        return 3
```

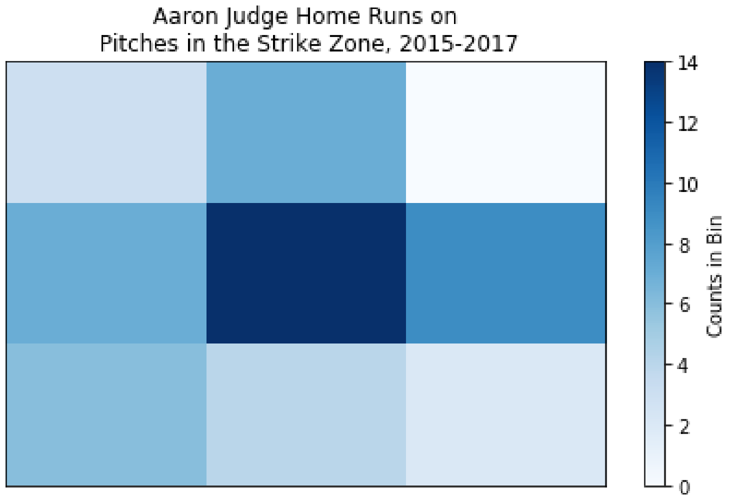
```
    # Middle third of strike zone
    if row.zone in [4, 5, 6]:
        return 2
    # Lower third of strike zone
    if row.zone in [7, 8, 9]:
        return 1
```

8. Aaron Judge's home run zone

Now we can apply the functions we've created then construct our 2D histograms. First, for Aaron Judge (again, for pitches in the strike zone that resulted in home runs).

In [111]:

```
# Zones 11, 12, 13, and 14 are to be ignored for plotting simplicity
judge_strike_hr = judge_hr.copy().loc[judge_hr.zone <= 9]

# Assign Cartesian coordinates to pitches in the strike zone for Judge home runs
judge_strike_hr['zone_x'] = judge_strike_hr.apply(assign_x_coord, axis=1)
judge_strike_hr['zone_y'] = judge_strike_hr.apply(assign_y_coord, axis=1)


# Plot Judge's home run zone as a 2D histogram with a colorbar
plt.hist2d(x='zone_x', y='zone_y', data=judge_strike_hr, bins = 3, cmap='Blues')
plt.title('Aaron Judge Home Runs on\n Pitches in the Strike Zone, 2015-2017')
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
cb = plt.colorbar()
cb.set_label('Counts in Bin')
```



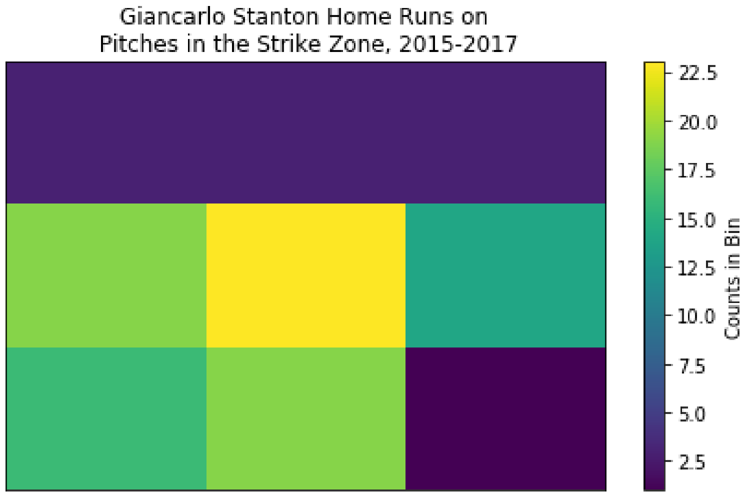
9. Giancarlo Stanton's home run zone

And now for Giancarlo Stanton.

In [113]:

```
# Zones 11, 12, 13, and 14 are to be ignored for plotting simplicity
stanton_strike_hr = stanton_hr.copy().loc[stanton_hr.zone <= 9]

# Assign Cartesian coordinates to pitches in the strike zone for Stanton home runs
stanton_strike_hr['zone_x'] = stanton_strike_hr.apply(assign_x_coord, axis=1)
stanton_strike_hr['zone_y'] = stanton_strike_hr.apply(assign_y_coord, axis=1)

# Plot Stanton's home run zone as a 2D histogram with a colorbar
# ... YOUR CODE FOR TASK 9 ...
plt.hist2d(x='zone_x', y='zone_y', data=stanton_strike_hr, bins = 3)
plt.title('Giancarlo Stanton Home Runs on\n Pitches in the Strike Zone, 2015-2017')
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
cb = plt.colorbar()
```

```
cb.set_label('Counts in Bin')
```



10. Should opposing pitchers be scared?

A few takeaways:

- **Stanton does not hit many home runs on pitches in the upper third of the strike zone.**
- **Like pretty much every hitter ever, both players love pitches in the horizontal and vertical middle of the plate.**
- **Judge's least favorite home run pitch appears to be high-away while Stanton's appears to be low-away.**
- **If we were to describe Stanton's home run zone, it'd be middle-inside. Judge's home run zone is much more spread out.**

The grand takeaway from this whole exercise: Aaron Judge and Giancarlo Stanton are not identical despite their superficial similarities. In terms of home runs, their launch profiles, as well as their pitch speed and location preferences, are different.

Should opposing pitchers still be scared?

In [115]:

```
# Should opposing pitchers be wary of Aaron Judge and Giancarlo Stanton
should_pitchers_be_scared = True
```