Munachimso Aneke

Lab 4

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LinearRegression
import scipy as sci
```

Question 1

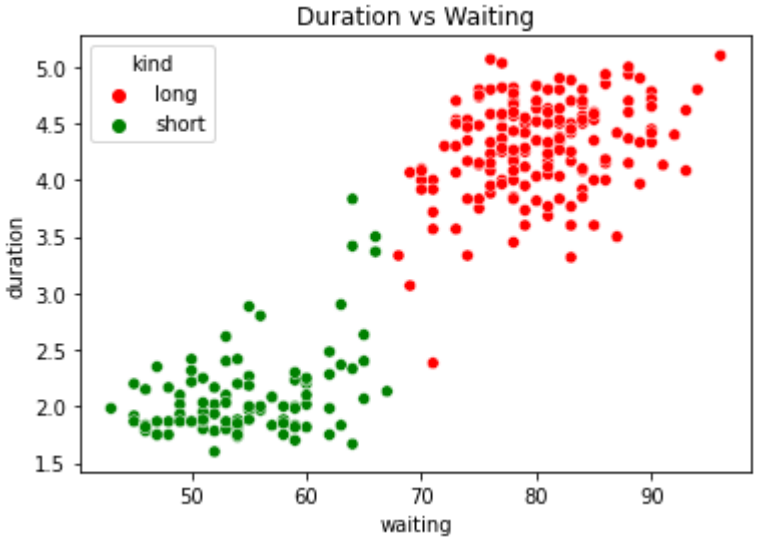
This question will center around analysis of the `geyser` dataset in base `seaborn` package. First download the dataset and name as `geyser_df`.

In [ ]:
```
geyser_df = sns.load_dataset('geyser')
```

Part A

Create a scatterplot of the `geyser_df` that plots `waiting` on the x-axis and `duration` on the y-axis. Make the color of the dot depend on the `kind` variable, use non-base coloring, and include a legend. Be sure to label the x- and y- axes and give your graph a title.

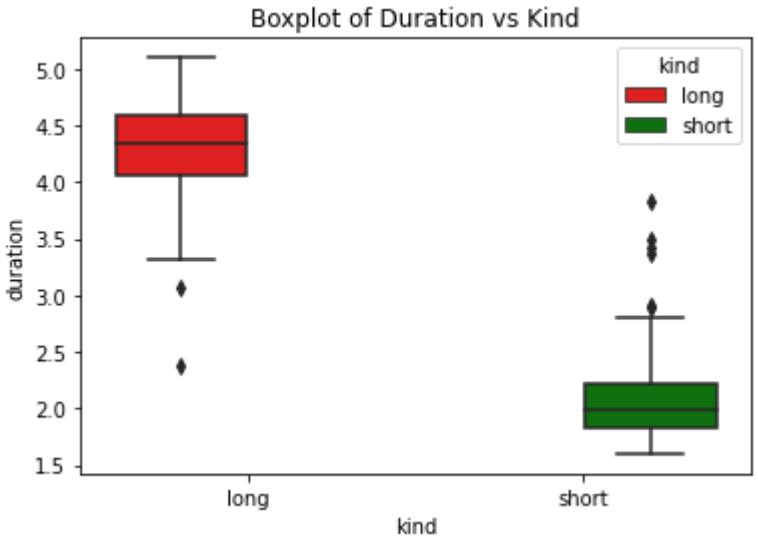
In [ ]:
```
ax_scatter = sns.scatterplot(x='waiting', y='duration', data=geyser_df, hue= 'kind', palette=['red', 'green'])
ax_scatter.set(title='Duration vs Waiting')
plt.show()
```



Part B

Based on the plot above, you see that there is a vast difference in the `duration` of long and short geyser events. Before you conduct an ANOVA test, you want to compare side-by-side boxplots. Plot `kind` on the x-axis and `duration` on the y-axis. Use `kind` as hue of the boxplot. Name the legend and place it in a spot that does not overlap any of the data.

In [ ]:
```
ax_box = sns.boxplot(x = 'kind', y= 'duration', hue='kind', data=geyser_df, palette=['red', 'green'])
ax_box.set(title='Boxplot of Duration vs Kind')
plt.show()
```



Part C

Using Pandas "groupby" option, group the dataset `geyser_df` on `kind` and then find the mean, median, standard deviation and variance of the duration for both kinds of geyser events.

In [ ]:
```
df_grouped = geyser_df.groupby('kind')
summary = df_grouped.duration.describe()[['mean', '50%', 'std']]
summary.rename(columns={'mean':'Mean','50%': 'Median', 'std': 'Standard Deviation'}, inplace=True)
summary['Variance'] = summary['Standard Deviation']**2
summary
```

Out[ ]:

kind	Mean	Median	Standard Deviation	Variance
long	4.29793	4.350	0.422677	0.178656
short	2.09433	1.983	0.394762	0.155837

Part D

Suppose you want to test:

H_0: Mean duration time of long and short geyser events are the same
H_a: Mean duration time of long and short geyser events are not the same

Conduct the ANOVA test at the 5% level of significance. Include a p-value in your response.

In [ ]:
```
fvalue, pvalue = sci.stats.f_oneway(geyser_df.duration[geyser_df.kind == 'long'],
                                    geyser_df.duration[geyser_df.kind == 'short'])
print('pvalue =',format(pvalue, '.2e'))
```

```
pvalue = 1.60e-121
```

Part E

Is there evidence of a difference between the long and short geyser event means?

Answer:

At a significance level of 5% (alpha = 0.05), we will reject the null hypothesis because the pvalue(1.60e-121) is less than 0.05. ***This means that there is evidence of a difference between the long and short geyser event means.***

Question 2

This question will center around analysis of the car crashes dataset in base seaborn package. First download the dataset and name as `carcrash_data`.

You can read about the variables here: https://www.kaggle.com/fivethirtyeight/fivethirtyeight-bad-drivers-dataset. The variables reported include:

- `total` : Number of drivers involved in fatal collisions per billion miles
- `speeding` : Percentage Of Drivers Involved In Fatal Collisions Who Were Speeding
- `alcohol` : Percentage Of Drivers Involved In Fatal Collisions Who Were Alcohol-Impaired
- `not_distracted` : Percentage Of Drivers Involved In Fatal Collisions Who Were Not Distracted
- `no_previous` : Percentage Of Drivers Involved In Fatal Collisions Who Had Not Been Involved In Any Previous Accidents
- `ins_premium` : Car Insurance Premiums in dollars
- `ins_losses` : Losses incurred by insurance companies for collisions per insured driver in dollars
- `abbrev` : State

In [ ]:
```
carcrash_data = sns.load_dataset('car_crashes')
```

Part A

Create a new column variable, `region`, which will classify the state based on the following definitions:

- `Northeast`: ME, NH, VT, MA, RI, CT, NY, NJ, and PA
- `Midwest`: OH, MI, IN, WI, IL, MN, IA, MO, ND, SD, NE, and KS
- `South`: DE, MD, VA, WV, KY, NC, SC, TN, GA, FL, AL, MS, AR, LA, TX, and OK
- `West`: MT, ID, WY, CO, NM, AZ, UT, NV, CA, OR, WA, AK, and HI

Create a column variable, `premium`, that reports "Above average" if the `ins_premium` value is above the average of the `ins_premium` and "Below average" elsewise.

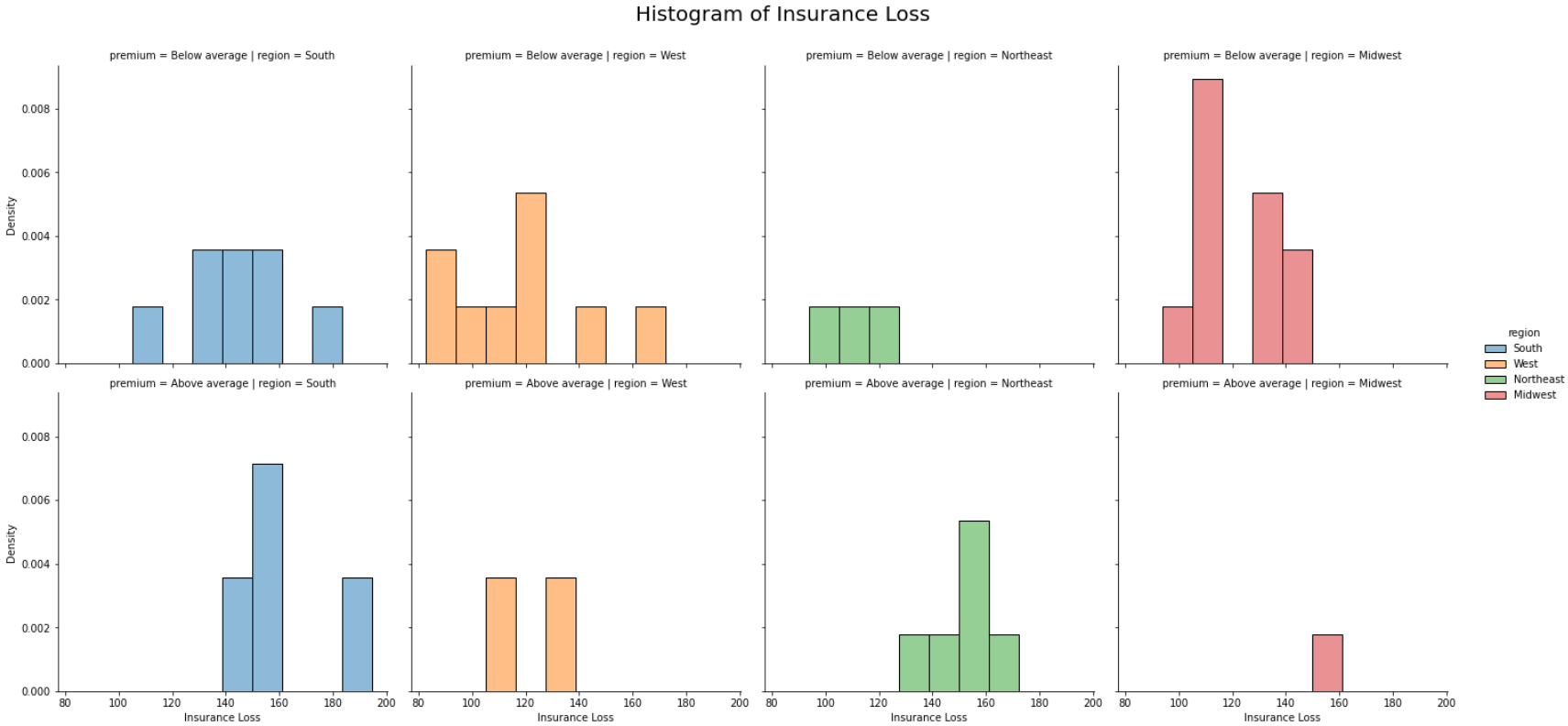
Remove the row corresponding to "DC".

In [ ]:
```
# create dictionary
region_dict = {}
region_dict.update(region_dict.fromkeys(['ME', 'NH', 'VT', 'MA', 'RI', 'CT', 'NY', 'NJ', 'PA' ],'Northeast')) # Northea
region_dict.update(region_dict.fromkeys(['OH', 'MI', 'IN', 'WI', 'IL', 'MN', 'IA', 'MO', 'ND', 'SD', 'NE', 'KS'],
                                        'Midwest')) # Midwest def
region_dict.update(region_dict.fromkeys(['DE', 'MD', 'VA', 'WV', 'KY', 'NC', 'SC', 'TN', 'GA', 'FL', 'AL', 'MS',
                                        'AR', 'LA', 'TX', 'OK'],'South')) # South def
region_dict.update(region_dict.fromkeys(['MT', 'ID', 'WY', 'CO', 'NM', 'AZ', 'UT', 'NV', 'CA', 'OR', 'WA', 'AK',
                                        'HI'],'West')) # West def

# apply dictionary and create 'region'
carcrash_data['region'] = carcrash_data['abbrev'].map(region_dict)

# create 'premium'
premium_avg = carcrash_data.ins_premium.mean() # avg ~= 886.957647
carcrash_data['premium'] = np.where(carcrash_data.ins_premium > premium_avg, 'Above average', 'Below average')

# remove 'DC'
carcrash_data = carcrash_data[carcrash_data.abbrev != 'DC']
carcrash_data = carcrash_data.reset_index()
del carcrash_data['index']
```

Part B

Graph facetted histograms based on `ins_losses` variable. Include 10 bins and color the histograms based on `region`. The rows should represent `premium` and the columns should represent `region`, both variables created in part a. On the y-axis, showcase the density.

In [ ]:
```
ax_facet_hist = sns.displot(data = carcrash_data, bins=10, x='ins_losses', row = 'premium', col = 'region',
                            hue = 'region', stat ='density')
#facet_kws=dict(margin_titles=True)
ax_facet_hist.set_axis_labels("Insurance Loss") # set x-axis label
ax_facet_hist.fig.subplots_adjust(top=0.9)
ax_facet_hist.fig.suptitle('Histogram of Insurance Loss', fontsize= 20)
plt.show()
```

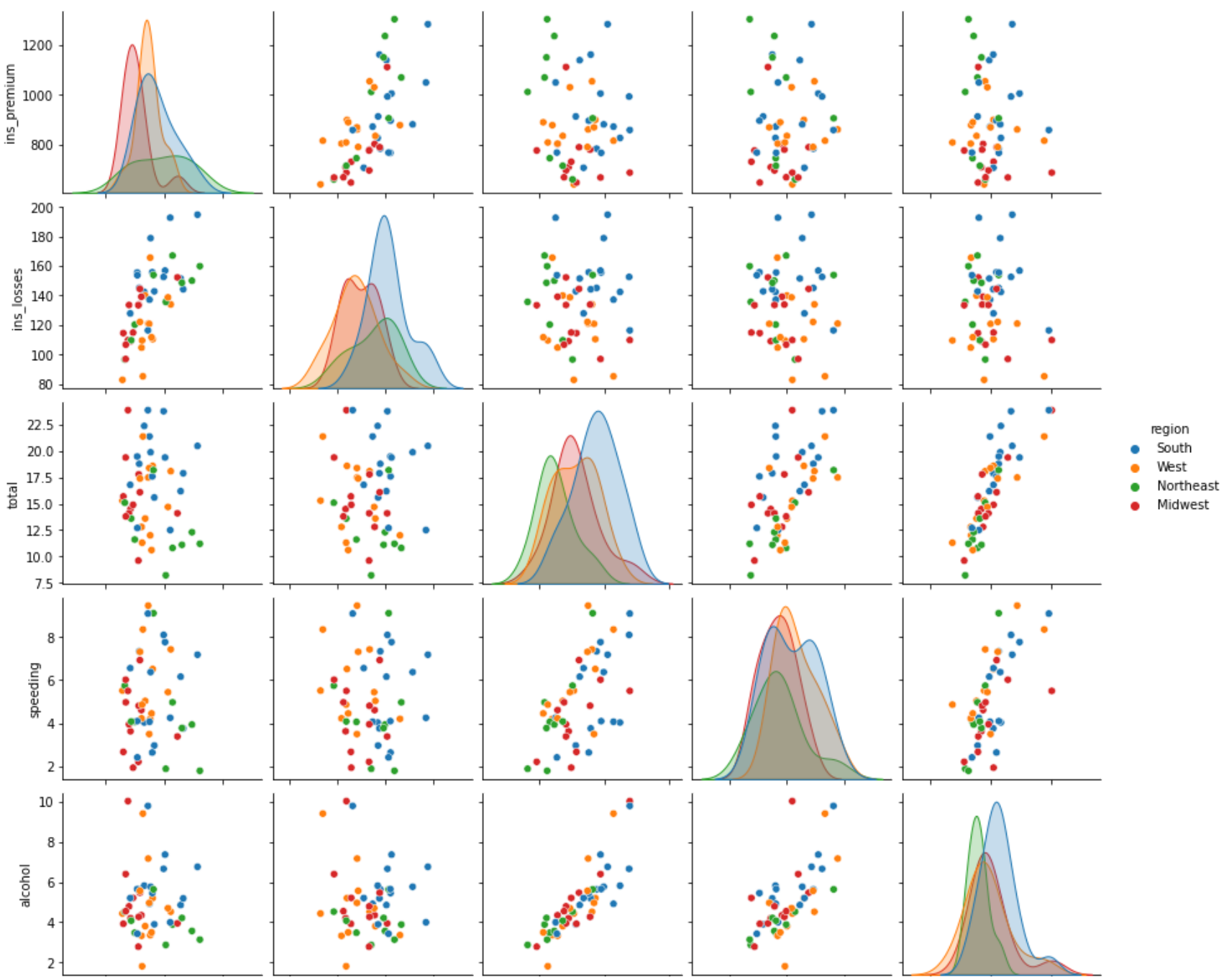


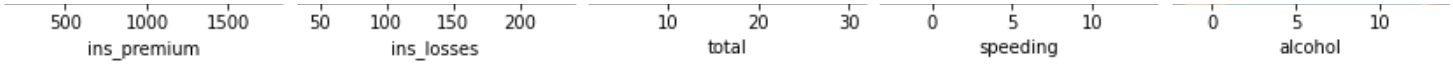
Part C

Create a pair plot of `carcrash_df` comparing `ins_premium`, `ins_losses`, `total`, `speeding`, `alcohol`. Colorize based on `region`.

In [ ]:

```
ax_pair_plot = sns.pairplot(data=carcrash_data, vars=['ins_premium', 'ins_losses', 'total', 'speeding', 'alcohol'],
                            hue='region')
ax_pair_plot.fig.subplots_adjust(top = 0.9)
ax_pair_plot.fig.suptitle("Scatter Pair Plots of ins_premium, ins_losses, total, speeding, alcohol", fontsize = 20)
plt.show()
```


Scatter Pair Plots of ins_premium, ins_losses, total, speeding, alcohol
ins_premium
ins_losses
total
speeding
alcohol
region
South
West
Northeast
Midwest



Part D

Analyzing the pairplot above, which variables do you think are highly correlated? Does the region seem to have any difference on the relationship between the variables?

Ans:

The highly correlated variables are

- `ins_premium` and `ins_losses`
- `total` and `alcohol`
- `speeding` and `alcohol`
- `speeding` and `total`

Region does not seem to have much difference

Question 3

The following questions will be analyzing the `planets` data in Seaborn package. Import the data using the code below:

In [ ]:
```
planets_data = sns.load_dataset('planets')
sns.set_theme(style = "ticks", palette = "pastel")
planets_data
```

Out[ ]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

Part A

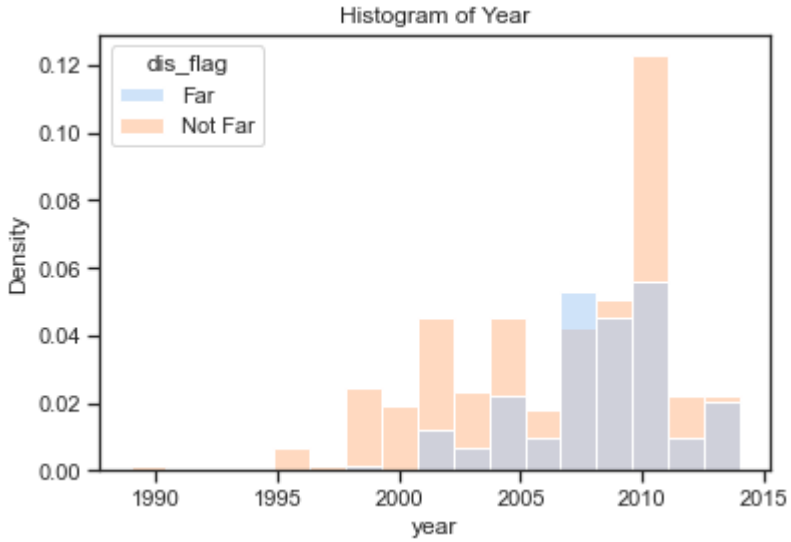
First, clean the data so that none of the columns have NaN response values. Name this new dataset `planets_df`. Then add a variable called `dis_flag` that returns "Far" if the distance is more then the average and "Not Far" if the distance is less than the average.

In [ ]:
```
planets_df = pd.DataFrame(planets_data.dropna())
planets_df = planets_df.reset_index()
del planets_df['index']
#print(planets_df.isnull().sum())
distance_avg = planets_df.distance.mean() # avg ~= 52.06821
planets_df['dis_flag'] = np.where(planets_df.distance > distance_avg, 'Far', 'Not Far')
```

Part B

Using a layered histogram, plot the density distribution of `year` with the overlapping bars representing the `dis_flag` variable. Include a title, legend, axis labels.

In [ ]:
```
ax_layered_hist = sns.histplot(data=planets_df, x='year', stat = 'density', multiple='layer', hue ='dis_flag')
ax_layered_hist.set(title = 'Histogram of Year')
plt.show()
```



Part C

Does it appear there is a trend between year and distance of the planet? When are "far" planets mostly found?

Ans: There appears to be no trend betweeen year and distance. By estimate, "far" planets are mostly found in around 2010 (probably between 2008-2011)

Question 4

Anscombe's quartet comprises four data sets that have nearly identical simple descriptive statistics, yet have very different distributions and appear very different when graphed. Each dataset consists of eleven (x,y) points. They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it and the effect of outliers and other influential observations on statistical properties. He described the article as being intended to counter the impression among statisticians that "numerical calculations are exact, but graphs are rough."

In [ ]:
```
anscombe_df = sns.load_dataset("anscombe")
```

Part A

Find the mean and variance of `x` and `y` grouping on the `dataset`. Then, find the mean and variance of `x` and `y` without any grouping.

In [ ]:
```
# Grouped Mean and Variance
anscombe_grouped = anscombe_df.groupby('dataset')
anscombe_summary = anscombe_grouped.describe().iloc[:,[1,2,9,10]]
anscombe_summary.iloc[:,1] = pow(anscombe_summary.iloc[:,1],2)
anscombe_summary.iloc[:,3] = pow(anscombe_summary.iloc[:,3],2)
anscombe_summary.rename(columns={'mean':'Mean', 'std': 'Variance'}, inplace=True)
anscombe_summary
```

Out[ ]:

	x		y	
	Mean	Variance	Mean	Variance
dataset				
I	9.0	11.0	7.500909	4.127269
II	9.0	11.0	7.500909	4.127629
III	9.0	11.0	7.500000	4.122620
IV	9.0	11.0	7.500909	4.123249

In [ ]:
```
# Non-Grouped Mean and Variance
m_v_data = {'Mean':[anscombe_df.x.mean(), anscombe_df.y.mean()],
            'Variance':[anscombe_df.x.var(),anscombe_df.y.var()]}
m_v_df = pd.DataFrame(m_v_data, index=['x','y'])
m_v_df
```

Out[ ]:

	Mean	Variance
x	9.000000	10.232558
y	7.500682	3.837388

Part B

Find the correlation between `x` and `y` grouped on `dataset`. Then, find the correlation between `x` and `y` without any grouping. Display your answers as correlation matrices.

In [ ]:
```
# Grouped Correlation
anscombe_grouped.corr()
```

Out[ ]:

		x	y
dataset			
I	**x**	1.000000	0.816421
	y	0.816421	1.000000
II	**x**	1.000000	0.816237
	y	0.816237	1.000000
III	**x**	1.000000	0.816287
	y	0.816287	1.000000
IV	**x**	1.000000	0.816521
	y	0.816521	1.000000

In [ ]:
```
# Non-Grouped Correlation
anscombe_df.corr()
```

Out[ ]:

	x	y
x	1.000000	0.816366
y	0.816366	1.000000

Part C

For each of the datasets, find the parameter estimates for the slope and y-intercept using `np.polyfit(x =, y =, d =)` where `d` is the degree. Since we are plotting simple linear regression models, the degree we want is 1. The output of `np.polyfit()` will be of the form: `array([a, b])`. `a` output will be the slope and `b` output will be y-intercept.

Display your calculated slope and y-intercept values in a dataframe with the dataset as rows and slope and y-intercept as columns. Include labeling.

In [ ]:
```
p1 = np.polyfit(x=anscombe_df.x[anscombe_df['dataset'] == 'I'], y=anscombe_df.y[anscombe_df['dataset'] == 'I'],
                deg=1)
p2 = np.polyfit(x=anscombe_df.x[anscombe_df['dataset'] == 'II'], y=anscombe_df.y[anscombe_df['dataset'] == 'II'],
                deg=1)
p3 = np.polyfit(x=anscombe_df.x[anscombe_df['dataset'] == 'III'], y=anscombe_df.y[anscombe_df['dataset'] == 'III'],
                deg=1)
p4 = np.polyfit(x=anscombe_df.x[anscombe_df['dataset'] == 'IV'], y=anscombe_df.y[anscombe_df['dataset'] == 'IV'],
                deg=1)

poly_data = {'slope':[p1[0], p2[0], p3[0], p4[0]], 'y-intercept':[p1[1], p2[1], p3[1], p4[1]]}
poly_frame = pd.DataFrame(poly_data, index= ['I', 'II', 'III', 'IV'])
poly_frame.index.name ='dataset'
poly_frame
```

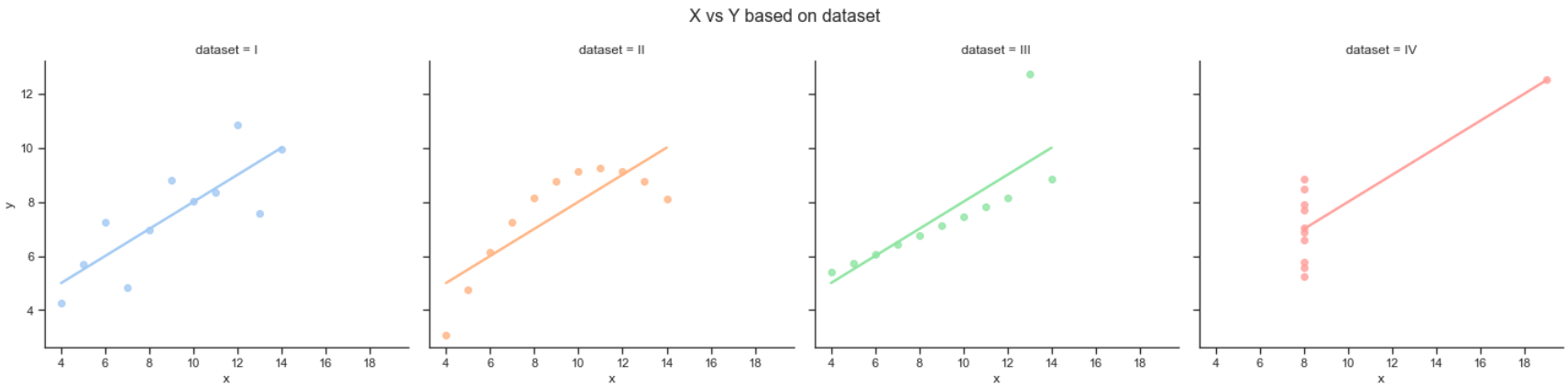
Out[ ]:

dataset	slope	y-intercept
I	0.500091	3.000091
II	0.500000	3.000909
III	0.499727	3.002455
IV	0.499909	3.001727

Part D

Use facet graphing to graph a scatterplot of `x` versus `y` for each dataset. Colorize each of the scatterplots based on the `dataset`. Include the linear regression equation on each of the graphs but do not include the confidence bands.

In [ ]:
```
ax_reg_plot = sns.lmplot(data=anscombe_df, x='x', y='y', hue='dataset', col='dataset', ci=None)
ax_reg_plot.fig.subplots_adjust(top= 0.85)
ax_reg_plot.fig.suptitle("X vs Y based on dataset", fontsize=16)
plt.show()
```



Part E

What does this exercise show you about relying on numeric outputs for statistical analysis?

Ans: This exercise shows that relying only on numeric outputs for statistical analysis can be misleading. In the graph, the scatterplot of dataset 2 and 4 shows a quadratic and constant relationship respectively. Dataset 3 has one big outlier that skews the regression line. Yet, the `numpy.polyfit()` function produced linear models for them which misleading on it's own.