Statistics

Two types of statistics:
Descriptive - describe and summarize
Inferential - use a sample of data to make inferences about a larger population

Types of data in statistics:
Numeric - broken down into continuous(measured) or discrete (counted)
Categorical - nominal (unordered) or ordinal (ordered)

Data type matters because it drives the plots and summary statistics that you will use

Right data skew is on the left hand side
Left data skew is on the right hand side
Median is to the right of the mean in right data skew
Median is to the left of the mean in left data skew
In both left and right data skew median is a more accurate measure of central tendency

Spread describes how spread apart or close together the data points are

Variance is the average distance from each data point to the data's mean
calculating variance
dists = df['column'] - np.mean(df['col'])
sq_dists = dists ** 2
sum_sq_dists = np.sum(sq_dists)
variance = sum_sq_dists / (# of data points - 1)
** the higher the variance the more spread out are the data points
** remember units of variance are squared
can get all of these steps in one line of code
np.var(df['col'], ddof=1)
*without ddof=1, population variance is calculated instead of sample variance
delta degrees of freedom

Standard deviation
np.sqrt(np.var(df['col'], ddof=1))
or
np.std(df['col'], ddof=1)
** remember not squared

Mean absolute deviation
dists = df['col'] - mean(df$col)
then np.mean(np.abs(dists))
$ is a placeholder

Standard deviation vs mean absolute deviation
SD squares distance penalizing outliers
MAD penalizes each distance equally
One isn't better than the other, SD is just used more frequently

Quantiles
np.quantile(df['col'], 0.5)
0.5 can be manipulated and represents the percent
0.5 is 50% and would represent the median
can add in a list of desired quantiles
np.quantile(df['col'], [0, 0.1, 0.25, 0.5, 0.75, 0.9, 1])
can also use
np.linspace(start, stop, num) within np.quantile
np.quantile(df['col'], np.linspace(0, 1, 5))
start number, stop number, and how many intervals

Boxplots
the boxes represent quartiles

IQR interquartile range
range between 75% and 25%
np.quantile(df['col'], 0.75) - np.quantile(df['col'], 0.25)
or
from scipy.stats import iqr
iqr(df['col'])

Outliers
How to define?
general rule: data < Q1 - 1.5 x IQR
or data > Q3 + 1.5 x IQR
finding outliers
iqr = iqr(df['col']
lower_threshold = np.quantile(df['col'], 0.25) - 1.5 * iqr
higher_threshold = np.quantile(df['col'], 0.75 + 1.5 * iqr
now subset your data
df[(df['col'] < lower_threshold) | (df['col'] > higher_threshold)]

All of these statistics in one line of code

df['col'].describe()

How to measure?
Probability
P(event) = #ways event can happen / total # of possible outcomes
example
P(heads) = 1 way to get heads / 2 possible outcomes = 1/2 = 50%
sampling with/without replacement
sales_counts.sample(5, replace=True)
5 is the number of options
replace ask if with or without replacement

**Independent events
Two events are independent if the probability of the second event isn't affected by the outcome of the first event
in general when sampling with replacement each pick is independent

**Dependent events
Two events are dependent if the probability of the second even is affected by the outcome of the first event
in general when sampling without replacement each pick is dependent

Discrete distributions
Probability distribution - describes the probability of each possible outcome in a scenario
Expected value: mean of a probability distribution
Expected value of a fair die roll = (1x1/6)+(2x1/6)+(3x1/6)+(4x1/6)+(5x1/6)+(6x1/6) = 3.5
Bar plot is a good way to lay out a probability distribution
Probability = area
P(die roll) <=2 = 1/6 + 1/6 = 1/3
Expected value of an uneven die roll (ie a die with 2 3s and no 2) = (1x1/6)+(2x0)+(3x1/3)+(4x1/6)+(5x1/6)+(6x1/6) = 3.67

discrete uniform distribution - when all outcomes have the same probability

Visualizing a sample
df['col'].hist(bins=np.linspace())

Law of large numbers
as the size of your sample increases, the sample mean will approach the expected value

Continuous uniform distribution
probability still = area
from spicy.stats import uniform
uniform.cdf(units of distribution solving for, distribution start #, distribution end #)
uniform.rvs(minimum value, maximum value, #random values we want to generate)

The binomial distribution
probability distribution of the number of successes in a sequence of independent trials
described by n: total number of trials and p: probability of success
binary outcomes = 2 possible outcomes
from spicy.stats import binom
binom.rvs(# of coins, probability of heads/success, size=# of trials
example
binom.rvs(1, 0.5, size=8)
1 coin, with 50% chance of success, flipped 8x)
n is represented by the 3rd argument in binom.rvs
p is represented by the 2nd argument in binom.rvs

binom.pmf(num heads, num trials, prob of heads)
binom.pmf(7, 10, 0.5)
what are the chances that we get 70% heads with 10 flips in a fair coin

binom.cdf gives the probability of getting a number of successes less than or equal to the first argument
1-binon.cdf() to get the probability of getting a number of successes greater than the first argument

Expected value of binomial distribution = n x p
example
Expected number of heads out of 10 flips = 10 x 0.5 = 5

**for the binomial distribution to apply, each trial must be independent, so the outcome of one trial shouldn’t have an effect on the next

Normal distributions
lots of populations when put into a histogram look like normal distributions
continuous
area under curve = 1
tails never come to 0
68% of distribution falls within 1 STD
95% within 2 STD
99.7% within 3 STD

often call the 68-95-99.7 rule

Standard normal distribution
special distribution
mean = 0 and STD = 1

from scipy.stats import norm
norm.cdf(number of interest, mean, std)
1-norm.cdf(number of interest, mean, std) tells you the area to the right of your request

get an in between area
norm.cdf(number of interest, mean, std) - norm.cdf(2nd number of interest, mean, std)

to calculate percentages use .ppf
norm.ppf(percent desired, mean, std)
to get the remaining percentage
norm.ppf(1-initial percent chosen, mean, std)

generate random numbers with a normal distribution
norm.rvs(mean, std, size=)

The central limit theorem
sampling distribution of a statistic becomes closer to the normal distribution as the number of trials increases
** only applies when samples are random and independent
example using a for loop

```
sample_means = [ ]
for i in range(10):
    samp_5 = die.sample(5, replace=True)
    sample_means.append(np.mean(samp_5))
print(sample_means)
```

this is rolling the dice 5x, taking the mean, appending it to means in list, and repeating 10x
** a distribution of a summary statistic like this is called a sampling distribution
if you were to continue to increase the number in the range from 10 to 100 to 1000 you would see the sample come closer to a norm distribution

can do CLT with std as well:

```
sample_std = [ ]
for i in range(10):
    sample_std.append(np.std(die.sample(5, replace=True)))
```

also can do CLT with proportions:
example
sales_team = pd.Series([‘Amir’, ‘Brian’, ‘Claire’, ‘Damian’])
sales_team.sample(10, replace=True)

```
# Set seed to 321
np.random.seed(321)

sample_means = []
# Loop 30 times to take 30 means
for i in range(30):
  # Take sample of size 20 from num_users col of all_deals with replacement
  cur_sample = all_deals['num_users'].sample(20, replace=True)
  # Take mean of cur_sample
  cur_mean = np.mean(cur_sample)
  # Append cur_mean to sample_means
  sample_means.append(np.mean(cur_mean))

# Print mean of sample_means
print(sample_means)

# Print mean of num_users in amir_deals
print(np.mean(amir_deals['num_users']))
```

Poisson processes
events appear to happen at a certain rate, but actually are happening completely at random

Poisson distribution
CLT applies
probability of some # of events occurring over a fixed period of time
described by lambda
lambda = average number of events per time interval
lambda is the distribution’s peak

```
from scipy.stats import poisson
poisson.pmf(# of events testing, mean events)
```

for less than or equal to use poisson.cdf
poisson.cdf(# of events testing, mean events)
for greater than

1 - poisson.cdf(# of events testing, mean events)

for random sampling
poisson.rvs(mean, size=)

Exponential distribution
probability of time between Poisson events
also uses lambda's (synonymous with rate)
continuous
example of lambda
on average 1 customer service ticket every 2 minutes
lambda = 0.5 tickets per minute

Expected value of exponential distribution
in terms of rate (Poisson - how frequently the events occur)

from scipy.stats import expon
#for less than
expon.cdf(parameter we are checking probability on, scale=)
**scale is actual rate, not lambda
for greater than
1 - expon.cdf()
for in between
expon.cdf(higher#) - expon.cdf(lower#)

(student's) t-distribution
similar shape to normal distribution
t-distribution tails are thicker which means that observations are more likely to fall further from the mean
has parameter degrees of freedom (df) which affects the thickness of the tails
lower df = thicker tails which in turn means higher std
as df gets higher the distribution comes closer and closer to a normal distribution

Log-normal distribution
variable whose logarithm is normally distributed
this results in distributions that are skewed

Correlation
relationships between two variables
x = explanatory/independent variable
y = response/dependent variable

Correlation coefficient

quantifies the linear relationship between two variables
number is between -1 and 1
magnitude corresponds to strength of relationship (closer to 1 or -1 denotes a strong relationship, to 0 is weak)
what 0 tells us is that x tells us nothing about y
sign (+ or -) corresponds to direction of relationship
+ tells us that as x increases y increases
minus tells us that as x increases y decreases

scatter plot is a nice way to visualize relationships
import seaboard as sos
sns.scatterplot(x='', y='', data=df)
plt.show()

adding a trend line
sns.lmplot(x=,y=,data=,ci=None)
plt.show()

Computing correlation
df['col1'.corr(df['col2'])
** can input columns in any order > correlation will be the same
this example used Pearson product-moment correlation (denoted as 'r')
this is the common way to compute correlation
r = sum of sample where each observation is calculated (xi - bar)(yi - year) / stdx * stdy

Non-linear relationships
ie quadratic (the data points are in the shape of a 'U')
correlation only works on linear relationships

when data is highly skewed a log transformation should be applied
df['col_log'] = np.log(df['col'])

other transformations:
square root (sqrt(x)) or reciprocal (1/x)

**transformations can be used on just x or y variable or different transformations on each variable

Design of Experiments
in general
Experiment aims to answer: What is the effect of the treatment on the response
Treatment is x, explanatory/independent variable

Response is y, response/dependent variable

Controlled experiments
A/B testing
one sees treatment and other does not
groups should be otherwise comparable
if not, confounding or bias will formulate

Tools to help eliminate bias
Randomized controlled trial
Placebo
Double-blind trial (administrator also doesn't know if the treatment is placebo or real

Observational studies
participants assign themselves
not random
**establish association, not causation
- effects can be confounded by factors that got certain people into the control or treatment group

Longitudinal study
participants are followed over a period of time to examine effect of treatment on response

Cross-sectional study
data on participants is collected from a single snapshot in time