

## 1. CAST OBJECTS TO A DATA TYPE

```
SELECT customerNumber,
       COUNT(*) AS number_payments,
       MIN(CAST(amount AS INT)) AS min_purchase,
       MAX(CAST(amount AS INT)) AS max_purchase,
       AVG(CAST(amount AS INT)) AS avg_purchase,
       SUM(CAST(amount AS INT)) AS total_spent
  FROM payments
```

```
pd.read_sql("""
    select cast(round(priceEach) as INTEGER) as rounded_price_int
        from orderDetails
      """,conn)
```

## 2. Strip year or month from date as a string object

WHERE strftime('%Y',paymentDate)='2004'

```
pd.read_sql("""
    SELECT customerNumber,
           COUNT(*) AS number_payments,
           MIN(CAST(amount AS INT)) AS min_purchase,
           MAX(CAST(amount AS INT)) AS max_purchase,
           AVG(CAST(amount AS INT)) AS avg_purchase,
           SUM(CAST(amount AS INT)) AS total_spent
      FROM payments
     WHERE strftime('%Y',paymentDate)='2004'
   GROUP BY customerNumber
  """ ,conn)
```

✓ 0.0s

	customerNumber	number_payments	min_purchase	max_purchase	avg_purchase	total_spent
0	103	2	1676	6066	3871.0	7742
1	112	2	14191	33347	23769.0	47538
2	114	2	44894	82261	63577.5	127155
3	119	2	19501	47924	33712.5	67425
4	121	2	17876	34638	26257.0	52514
...	...	...	...	...	...	...
83	486	2	5899	45994	25946.5	51893
84	487	1	12573	12573	12573.0	12573
85	489	1	7310	7310	7310.0	7310
86	495	1	6276	6276	6276.0	6276
87	496	1	52166	52166	52166.0	52166

88 rows × 6 columns

```

pd.read_sql(
    select orderDate,
           strftime('%m',orderDate) AS month,
           strftime('%Y',orderDate) AS year,
           strftime('%d',orderDate) as day
    from orders;
    "",conn)

```

> apart a date or time value into different sub-parts. For example, here we extract the year, month and day from the order date:

1

```

pd.read_sql('''
    select orderDate,
           strftime('%m',orderDate) AS month,
           strftime('%Y',orderDate) AS year,
           strftime('%d',orderDate) as day
    from orders;
    ''',conn)

```

[109]

	orderDate	month	year	day
0	2003-01-06	01	2003	06
1	2003-01-09	01	2003	09
2	2003-01-10	01	2003	10
3	2003-01-29	01	2003	29
4	2003-01-31	01	2003	31
...	...	...	...	...
321	2005-05-29	05	2005	29
322	2005-05-30	05	2005	30
323	2005-05-30	05	2005	30
324	2005-05-31	05	2005	31
325	2005-05-31	05	2005	31

## Or use substr method

```

pd.read_sql("""
SELECT customerNumber,
       COUNT(*) AS number_payments,
       MIN(CAST(amount AS INT)) AS min_purchase,

```

```

MAX(CAST(amount AS INT)) AS max_purchase,
AVG(CAST(amount AS INT)) AS avg_purchase,
SUM(CAST(amount AS INT)) AS total_spent
FROM payments
WHERE substr(paymentDate,1,4) ='2004'
GROUP BY customerNumber
",conn)

```

```

pd.read_sql('''
SELECT customerNumber,
       COUNT(*) AS number_payments,
       MIN(CAST(amount AS INT)) AS min_purchase,
       MAX(CAST(amount AS INT)) AS max_purchase,
       AVG(CAST(amount AS INT)) AS avg_purchase,
       SUM(CAST(amount AS INT)) AS total_spent
  FROM payments
 WHERE substr(paymentDate,1,4) = '2004'
 GROUP BY customerNumber
''',conn)

```

✓ 0.0s

	customerNumber	number_payments	min_purchase	max_purchase	avg_purchase	total_spent
0	103	2	1676	6066	3871.0	7742
1	112	2	14191	33347	23769.0	47538
2	114	2	44894	82261	63577.5	127155
3	119	2	19501	47924	33712.5	67425
4	121	2	17876	34638	26257.0	52514
...	...	...	...	...	...	...
83	486	2	5899	45994	25946.5	51893
84	487	1	12573	12573	12573.0	12573
85	489	1	7310	7310	7310.0	7310
86	495	1	6276	6276	6276.0	6276
87	496	1	52166	52166	52166.0	52166

88 rows × 6 columns

### 3.Convert select statement to dataframe

```

import pandas as pd
cur.execute('select * from contactinfo')
df = pd.DataFrame(cur.fetchall())

```

```
df.columns = [x[0] for x in cur.description]
```

```
df
```

```
import pandas as pd
cur.execute('select * from contactinfo')
df = pd.DataFrame(cur.fetchall())
df.columns = [x[0] for x in cur.description]
df
```

	userId	firstName	lastName	role	telephone	street	city	state	zipcode
0	1	Christine	Holden	staff	2035687697	1672 Whitman Court	Stamford	CT	06995
1	2	Christopher	Warren	student	2175150957	1935 University Hill Road	Champaign	IL	61938
2	3	Linda	Jacobson	staff	4049446441	479 Musgrave Street	Atlanta	GA	30303
3	4	Andrew	Stepp	student	7866419252	2981 Lamberts Branch Road	Hialeah	FL	33012
4	5	Jane	Evans	student	3259909290	1461 Briarhill Lane	Abilene	TX	79602
5	6	Jane	Evans	student	3259909290	1461 Briarhill Lane	Abilene	TX	79602
6	7	Mary	Raines	student	9075772295	3975 Jerry Toth Drive	Ninilchik	AK	99639
7	8	Ed	Lyman	student	5179695576	3478 Be Sreet	Lansing	MI	48933

- 4. Highest **-altitude**
- Southern/northern – **latitude** eg northern-most airport is the highest latitude  
Southern-most – smallest latitude

## 5.Pandasql Error

```
----> 6 passenger_names = pysqldf(q)
```

**ImportError:** Unable to find a usable engine; tried using: 'sqlalchemy'.

A suitable version of sqlalchemy is required for sql I/O support.

Trying to import the above resulted in these errors:

- Pandas requires version '1.4.0' or newer of 'sqlalchemy' (version '1.3.19' currently installed).

**TO update use**

```
conda update sqlalchemy
```

Check version if updated

```
pip show sqlalchemy
```

6.

```

q = ...
SELECT *
FROM df
LIMIT 10;
...

passenger_names = pysqldf(q)
passenger_names
⊗ 0.0s Python

-----
AttributeError                               Traceback (most recent call last)
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\sqlalchemy\engine\base.py in execute(self, statement, parameters, execution_options)
    1411         try:
--> 1412             meth = statement._execute_on_connection
    1413         except AttributeError as err:

AttributeError: 'str' object has no attribute '_execute_on_connection'

The above exception was the direct cause of the following exception:

ObjectNotExecutableError                  Traceback (most recent call last)
<ipython-input-29-a8e6fa35812d> in <module>
      5 ...
      6
----> 7 passenger_names = pysqldf(q)
      8 passenger_names

<ipython-input-28-e40dd094d5eb> in <lambda>(q)
----> 1 pysqldf = lambda q: sqldf(q,locals())

c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\pandasql\sqldf.py in sqldf(query, env, db_uri)
    154     >>> sqldf("select avg(x) from df;", locals())
    155     """
--> 156     return PandaSQL(db_uri)(query, env)
...
--> 1414         raise exc.ObjectNotExecutableError(statement) from err
    1415     else:
    1416         return meth()

ObjectNotExecutableError: Not an executable object: '\nSELECT * \nFROM df\nLIMIT 10;\n'
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Works well in colab

## 7.Put dataframe in memory as to use conn

```
#put df to memory
```

```
import sqlite3
```

```
conn = sqlite3.connect(':memory:')
```

```
df.to_sql('df', conn, index=False, if_exists='replace')
```

```
# Execute SQL query using the connection
```

```
query = "SELECT * FROM df"
```

```
result = pd.read_sql(query, conn)
```

result

```
#put df to memory
import sqlite3
conn = sqlite3.connect(':memory:')
df.to_sql('df', conn, index=False, if_exists='replace')
|
# Execute SQL query using the connection
query = "SELECT * FROM df"
result = pd.read_sql(query, conn)

result
```

[93] 0.0s Python

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	A
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	None	S	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	None	S	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	None	S	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montville, Rev. Juozas	male	27.0	0	0	211536	13.0000	None	S	
887	888	1	1	Graham, Miss. Margaret	female	19.0	0	0	112053	30.0000	B42	S	

## 8.Get female and children that is female and male less than or equal to 15

```
df[(df['Sex'] == 'female') | (df['Age'] <= 15)]
```

## 9.Select everyone else other than 1

```
df[df['Pclass'] != '1']
```

## 10.calculate totals using counter(get frequency for each value)

```
counter = collections.Counter(x)
```

```
import collections
x = [1,1,1,1,2,2,2,2,3,3,4,5,5]
counter = collections.Counter(x)
|
✓ 0.0s
Counter({1: 4, 2: 4, 3: 2, 4: 1, 5: 2})
```

## 11. Create two vertical subplots sharing 15% and 85% of plot space

Create density instead of count on seaborn histogram

```
#Create two vertical subplots sharing 15% and 85% of plot space
#sharex allows sharing of axes i.e building multiple plots on the same axes
fig, (ax,ax2) = plt.subplots(2,sharex=True,gridspec_kw={'height_ratios':(.15,.85)},figsize=(10,8))
sns.histplot(data['Height'],
lw=2,
edgecolor='r',
alpha=0.4,
color='w',
label='Histogram',
stat='density',
ax=ax2
)
sns.kdeplot(data.Height,
lw=3,
color='b',
label='Kernerl Density Estimation plot',
alpha=0.7,
ax=ax2
)
mean = data.Height.mean()
```

```
std = data.Height.std()
parametric_dist = stats.norm(loc=mean, scale=std)
x=np.linspace(parametric_dist.ppf(0.01),parametric_dist.ppf(0.99),100)

ax2.plot(x,
          parametric_dist.pdf(x),
          color='g',
          alpha=0.7,
          lw=3,
          label = 'Parametric Fit'
         )
ax2.set_title('Density Estimations')

sns.boxplot(data=data,x='Height',ax=ax,color='r') #sns.boxplot(x=data.Height, ax = ax,color = 'red')
ax.set_title('Box and Whiskers Plot')
ax2.set(ylim=(0, 0.08))
plt.ylim(0, 0.11)
plt.legend();
```

```

#Create two vertical subplots sharing 15% and 85% of plot space
#sharex allows sharing of axes i.e building multiple plots on the same axes
fig, (ax,ax2) = plt.subplots(2,sharex=True,gridspec_kw={'height_ratios':(.15,.85)},figsize=(10,8))
sns.histplot(data['Height'],
             lw=2,
             edgecolor='r',
             alpha=0.4,
             color='w',
             label='Histogram',
             stat='density',
             ax=ax2
            )
sns.kdeplot(data.Height,
             lw=3,
             color='b',
             label='Kernel Density Estimation plot',
             alpha=0.7,
             ax=ax2
            )

mean = data.Height.mean()
std = data.Height.std()
parametric_dist = stats.norm(loc=mean, scale=std)
x=np.linspace(parametric_dist.ppf(0.01),parametric_dist.ppf(0.99),100)

ax2.plot(x,
          parametric_dist.pdf(x),
          color='g',
          alpha=0.7,
          lw=3,
          label = 'Parametric Fit'
         )
ax2.set_title('Density Estimations')

sns.boxplot(data=data,x='Height',ax=ax,color='r') #sns.boxplot(x=data.Height, ax = ax,color = 'red')
ax.set_title('Box and Whiskers Plot')
ax2.set(ylim=(0, 0.08))
plt.ylim(0, 0.11)
plt.legend();

```

✓ 0.4s

Python



```

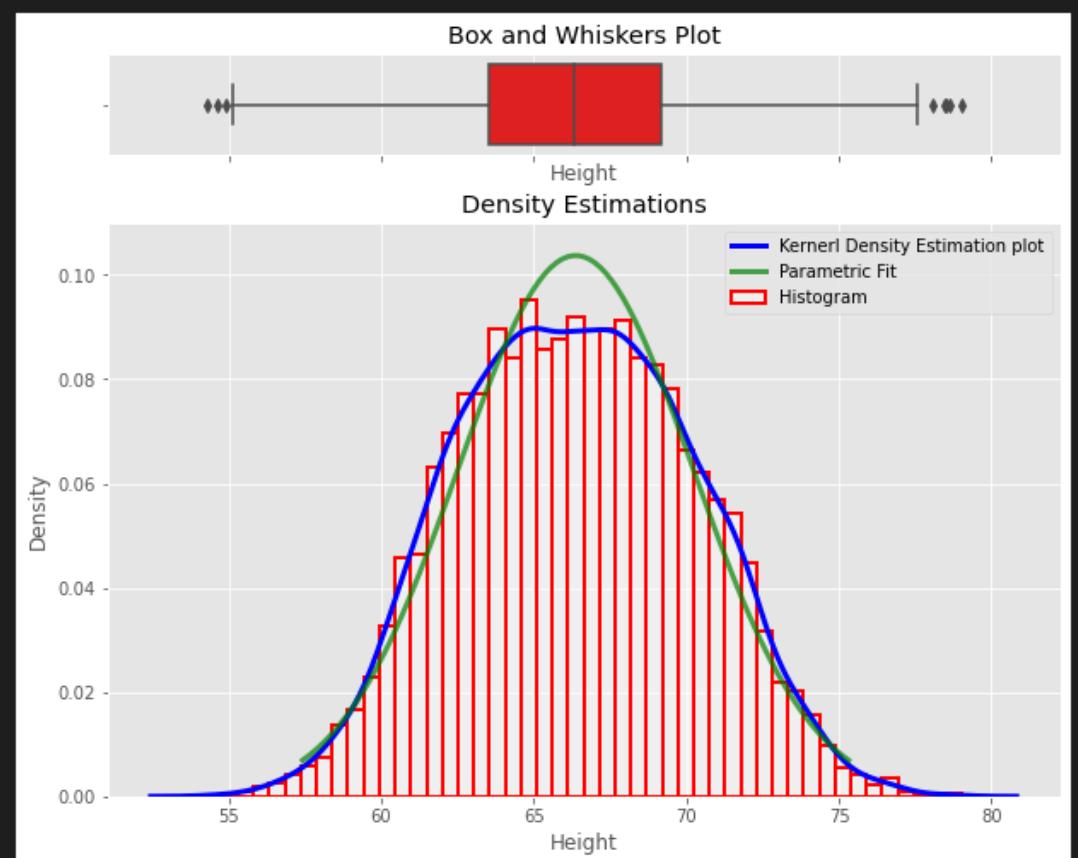
x=np.linspace(parametric_dist.ppf(0.01),parametric_dist.ppf(0.99),100)

ax2.plot(x,
          parametric_dist.pdf(x),
          color='g',
          alpha=0.7,
          lw=3,
          label = 'Parametric Fit'
         )
ax2.set_title('Density Estimations')

sns.boxplot(data=data,x='Height',ax=ax,color='r') #sns.boxplot(x=data.Height, ax = ax,color='r')
ax.set_title('Box and Whiskers Plot')
ax2.set(ylim=(0, 0.08))
plt.ylim(0, 0.11)
plt.legend();

```

✓ 0.4s



## 12. Add density (probability) instead of counts in matplotlib histogram

```

xtick_locations = range(1,6)

bins = np.arange(6) +0.5 #[0.5, 1.5, 2.5, 3.5, 4.5, 5.5]

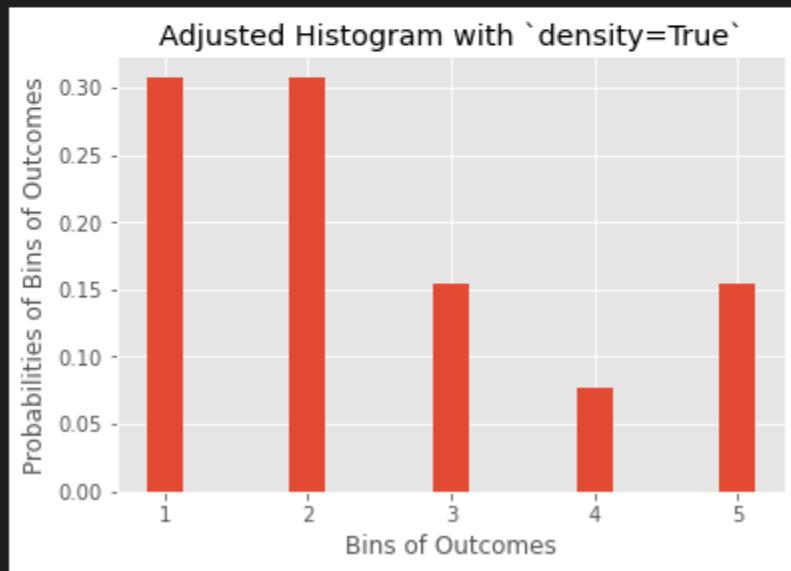
plt.hist(x,bins=bins,rwidth=0.25,density=True)

plt.xticks(ticks=xtick_locations)

```

```
plt.xlabel('Bins of Outcomes')
plt.ylabel('Probabilities of Bins of Outcomes')
plt.title("Adjusted Histogram with `density=True`");
```

```
xtick_locations = range(1,6)
bins = np.arange(6) +0.5 #[0.5, 1.5, 2.5, 3.5, 4.5, 5.5]
plt.hist(x,bins=rwidth=0.25,density=True)
plt.xticks(ticks=xtick_locations)
plt.xlabel('Bins of Outcomes')
plt.ylabel(['Probabilities of Bins of Outcomes'])
plt.title("Adjusted Histogram with `density=True`");
```



### 13. ttest

```
t= (x_bar-mu)/(sample_std/np.sqrt(25))
print('t: ',t)

critical_t_value = stats.t.ppf(1-alpha,24)
print('critical_t_value: ',critical_t_value)

p_value = stats.t.sf(t,df)
print("p-value:",p_value)
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Code Cell (Top):** Contains Python code for hypothesis testing. It calculates a t-statistic, critical t-value, and p-value. The output shows the t-statistic is approximately 13.64, which is much larger than the critical value of about 2.797, leading to a very low p-value (around 4.24e-13). The code cell has a status bar indicating it took 0.0s to run.
- Output Cell (Bottom):** Shows the results of the code execution. The output includes the calculated t-value, critical t-value, and p-value.
- Toolbar:** Includes standard Jupyter Notebook controls like cell selection, execution, and deletion.
- Section Header:** A "► Answer" section header is present below the notebook cells.
- Text Cell:** A "Make a decision." section with a note explaining that the t-value is greater than the critical value, so the null hypothesis is rejected.
- Text Cell (Bottom):** An "▼ Answer" section with a concluding statement about rejecting the null hypothesis due to the low p-value.

b) Example 2# one tailed left tail

```
sample =[20, 30, 30, 50, 75, 25, 30, 30, 40, 80]
```

```
x_bar = np.mean(sample)
```

```
sample_std = np.std(sample,ddof=1)
```

n=len(sample)

$$df=n-1$$

$$mu = 58$$

t\_stat1 =

```
print('t_stat1:' t_stat1[0])
```

```
print('alpha: ', stat1[1]/2)
```

$$t\_stat2 = (x\_bar - mu) / (s / \sqrt{n})$$

```
print(t.stat2d,t.stat2)
```

1112 *Journal of Health Politics*

$\rightarrow (1, 1, 1, 2, 1, 1, 2)$

```
print(alpha.z., alpha.z.)
```

```
t_crit = stats.t.ppf(0.1,df)
```

```
print('t_crit:', t_crit)
```

```
#null: there is no difference in prices in pants-atorium and pants-R-us  
#alternative: price in pants-atorium store is cheaper than in pants-R-us
```

Python

### ▼ Answer

Null: The pants at Pants-a-torium are not cheaper than the pants at Pants-R-Us. Alternative: The pants at Pants-a-torium are cheaper than the pants at Pants-R-Us.

### Perform the test.

```
#one tailed  
sample =[20, 30, 30, 50, 75, 25, 30, 30, 40, 80]  
x_bar = np.mean(sample)  
sample_std = np.std(sample,ddof=1)  
n=len(sample)  
df=n-1  
mu =58  
t_stat1 = stats.ttest_1samp(a=sample,popmean=58)  
print('t_stat1:', t_stat1[0])  
print('alpha:', t_stat1[1]/2)  
t_stat2 = (x_bar-mu)/(sample_std/np.sqrt(n))  
print('t_stat2:', t_stat2)  
alpha2 = stats.t.cdf(t_stat2, df) #used cdf since it is left tail(checking less)  
print('alpha2:', alpha2)  
t_crit = stats.t.ppf(0.1,df)  
print('t_crit:', t_crit)
```

✓ 0.0s

Python

```
t_stat1: -2.56934288148538  
alpha: 0.015110178147891276  
t_stat2: -2.56934288148538  
alpha2: 0.015110178147891276  
t_crit: -1.3830287383964925
```

### ► Answer

### Make a decision.

```
# p-value is small so we reject the null that is pants in p-atorium are cheaper  
#using t-critic since this is a left-tailed test if t_stat < t_critic we reject the null hypothesis
```

### c)two-sample t-test

```
delivery_times_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]
```

```
mean_A = np.mean(delivery_times_A)
```

```
std_A = np.std(delivery_times_A)
```

```
nobs_A = len(delivery_times_A)
```

**mean\_B = 26.8**

**std\_B = 2.6**

**nobs\_B = 10**

```
stats.ttest_ind_from_stats(mean1=mean_A, std1=std_A, nobs1=nobs_A,
                           mean2=mean_B, std2=std_B, nobs2=nobs_B,
                           equal_var=False)
```

**Perform the test.**

```
[78] delivery_times_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]
     mean_A = np.mean(delivery_times_A)
     std_A = np.std(delivery_times_A)
     nobs_A = len(delivery_times_A)
     mean_B = 26.8
     std_B = 2.6
     nobs_B = 10
     stats.ttest_ind_from_stats(mean1=mean_A, std1=std_A, nobs1=nobs_A,
                                mean2=mean_B, std2=std_B, nobs2=nobs_B,
                                equal_var=False)
```

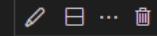
✓ 0.0s

Python

```
... Ttest_IndResult(statistic=1.8206924435070326, pvalue=0.08535597189429299)
```

► Answer

**Make a decision.**



▼ Answer

The p-value is greater than our threshold value of 5%, so we can't reject the null hypothesis that the two neighborhoods have the same restaurant delivery times.

c) 2 sample again two tailed

```
high_protein = [134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97, 123]
```

```
low_protein = [70, 118, 101, 85, 107, 132, 94]
```

```
stats.ttest_ind(a=high_protein, b=low_protein)
```

Perform the test. Do we reject the null hypothesis?

```
high_protein = [134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97, 123]
low_protein = [70, 118, 101, 85, 107, 132, 94]
mean_high = np.mean(high_protein)
mean_low = np.mean(low_protein)
std_high = np.std(high_protein, ddof=1)
std_low = np.std(low_protein, ddof=1)
n_high = len(high_protein)
n_low = len(low_protein)

stats.ttest_ind_from_stats(mean1=mean_high, std1=std_high, nobs1=n_high,
                           mean2=mean_low, std2=std_low, nobs2=n_low,
                           equal_var=False
                           )
#fail to reject at a significance level of 0.05 and say no difference in weights
```

[86] ✓ 0.0s Python

... Ttest\_indResult(statistic=1.9107001042454415, pvalue=0.07820704092145601)

```
high_protein = [134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97, 123]
low_protein = [70, 118, 101, 85, 107, 132, 94]
stats.ttest_ind(a=high_protein, b=low_protein)
```

[87] ✓ 0.0s Python

... Ttest\_indResult(statistic=1.89143639744233, pvalue=0.07573012895667763)

▼ Answer

```
high_protein = [134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97, 123] low_protein = [70,
118, 101, 85, 107, 132, 94] stats.ttest_ind(a=high_protein, b=low_protein)
```

We fail to reject the null hypothesis at a significance level of  $\alpha = 0.05$ .

## d) 2 sample one tailed

```
h_bar = np.mean(high_protein)
```

```
l_bar = np.mean(low_protein)
```

```
h_df = len(high_protein) - 1
```

```
l_df = len(low_protein) - 1
```

```
pooled_var = (h_df*np.var(high_protein) + l_df*np.var(low_protein)) / (h_df + l_df)
```

```
t_stat = (h_bar - l_bar) / np.sqrt(pooled_var * (1/len(high_protein) + 1/len(low_protein)))
```

```
t_stat
```

```
stats.t(df=h_df+l_df).cdf(t_stat)
```

```

h_bar = np.mean(high_protein)
l_bar = np.mean(low_protein)
h_df = len(high_protein) - 1
l_df = len(low_protein) - 1
pooled_var = (h_df*np.var(high_protein) + l_df*np.var(low_protein)) / (h_df + l_df)
t_stat = (h_bar - l_bar) / np.sqrt(pooled_var * (1/len(high_protein) + 1/len(low_protein)))
t_stat

```

Slide Type: notes Python

✓ 0.0s  
1.9974844476064675

#### ▼ Answer

```

h_bar = np.mean(high_protein) l_bar = np.mean(low_protein) h_df = len(high_protein) - 1 l_df
= len(low_protein) - 1 pooled_var = (h_df*np.var(high_protein) + l_df*np.var(low_protein)) /
(h_df + l_df) t_stat = (h_bar - l_bar) / np.sqrt(pooled_var * (1/len(high_protein) +
1/len(low_protein))) t_stat

```

Can we reject?

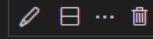
```

stats.t(df=h_df+l_df).cdf(t_stat)

```

Python

✓ 0.0s  
0.9689827301771085



#### ▼ Answer

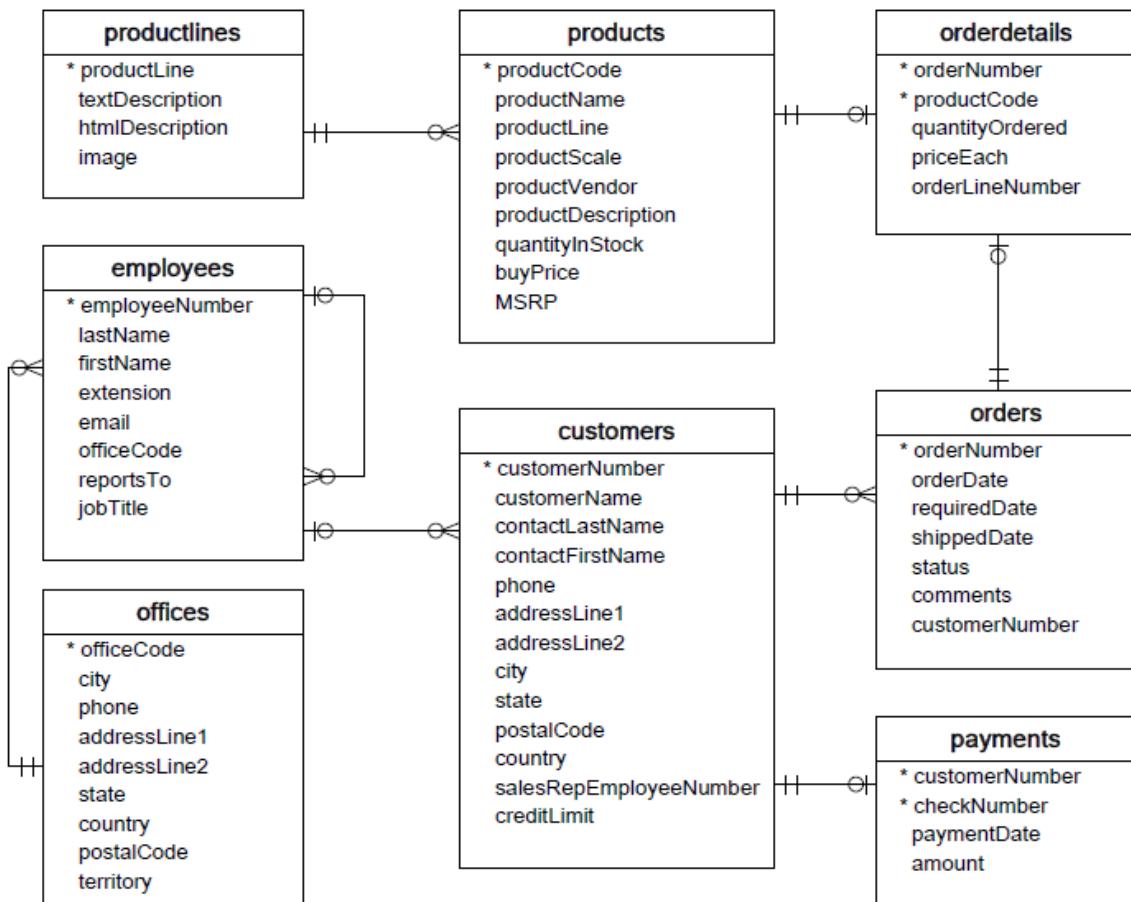
```
stats.t(df=h_df+l_df).cdf(t_stat)
```

Yes, we can reject the null hypothesis!

</details>

## SQL SUMMARY

- We will primarily use SQLite in these lessons because it is lightweight and portable (and therefore useful for educational purposes)
- SQLite is a C library that provides lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
- Some applications can use SQLite for internal data storage. Its also possible to prototype an application using sqlite and then port the code to a larger database such as PostgreSQL or Oracle



1. **ERD (Entity Relationship Diagram)**- shows the relationship between tables. It does not give us any information about the specific data stored in the database, but rather the metadata

## 2. Connect to sqlite 3

```
import sqlite3
conn = sqlite3.connect('data.sqlite')
```

## 3. Select Records

```
q="SELECT * FROM EMPLOYEES"
pd.read_sql(q,conn)
```

	employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo
0	1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	
1	1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002
2	1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002
3	1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056
4	1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056
5	1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056
6	1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143
7	1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143
8	1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143
9	1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143
10	1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143
11	1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143
12	1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102
13	1370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1102
14	1401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1102
15	1501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1102
16	1504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1102
17	1611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1088
18	1612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1088
19	1619	King	Tom	x103	tking@classicmodelcars.com	6	1088
20	1621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1056
21	1625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1621
22	1702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1102

4. Check table in a database (Note that the .execute() method didn't actually return our data. The data is now just available in our cursor object. We'll use the .fetchall() method to get all the rows from our query.)

```
cur = conn.cursor()
cur.execute("""SELECT name FROM sqlite_master WHERE type = 'table';""")
table_names = cur.fetchall()
table_names
```

```
cur = conn.cursor()
cur.execute("""SELECT name FROM sqlite_master WHERE type = 'table';""")
table_names = cur.fetchall()
table_names
```

```
[('orderdetails',),
 ('payments',),
 ('offices',),
 ('customers',),
 ('orders',),
 ('productlines',),
 ('products',),
 ('employees',)]
```

5. Select \* using cursor or can use 'curr.execute(" select \* from offices").fetchall(),'

```
curr.execute(" select * from offices ")
curr.fetchall()
```

```
curr.execute('' select * from offices ''')
curr.fetchall()
```

```
[('1',
 'San Francisco',
 '+1 650 219 4782',
 '100 Market Street',
 'Suite 300',
 'CA',
 'USA',
 '94080',
 'NA'),
 ('2',
 'Boston',
 '+1 215 837 0825',
 '1550 Court Place',
 'Suite 102',
 'MA',
 'USA',
 '02107',
 'NA'),
 ('3',
 'NYC',
 '+1 212 555 3000',
 '523 East 53rd Street',
 'apt. 5A',
 'NY',
 'USA',
 ...
 'Level 7',
 '',
 'UK',
 'EC2N 1HN',
 'EMEA')]
```

6. curr.execute(" select \* from offices").fetchall()[0][2]

```
curr.execute('' select * from offices '').fetchall()[0][2]
```

```
+1 650 219 4782
```

7. Viewing column names(

Looks like we got some data, but it's not clear what each element represents. We can view the column names in the cursor's description attribute.)

```
curr.description
```

```
curr.description
```

```
(('officeCode', None, None, None, None, None, None),  
 ('city', None, None, None, None, None, None),  
 ('phone', None, None, None, None, None, None),  
 ('addressLine1', None, None, None, None, None, None),  
 ('addressLine2', None, None, None, None, None, None),  
 ('state', None, None, None, None, None, None),  
 ('country', None, None, None, None, None, None),  
 ('postalCode', None, None, None, None, None, None),  
 ('territory', None, None, None, None, None, None))
```

8. Create a dataframe with column names from the records/from a table

```
import pandas as pd  
pd.DataFrame(  
    data = curr.execute(" select * from offices;").fetchall(),  
    columns =[x[0] for x in curr.description]  
)
```

The screenshot shows a Jupyter Notebook cell with the following content:

```
import pandas as pd  
pd.DataFrame(  
    data = curr.execute('' select * from offices;'').fetchall(),  
    columns =[x[0] for x in curr.description]  
)
```

[48]

Python

The resulting DataFrame is displayed below:

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
0	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
1	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
2	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
3	4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans			France	75017	EMEA
4	5	Tokyo	+81 33 224 5000	4-1 Kioicho		Chiyoda-Ku	Japan	102-8578	Japan
5	6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2		Australia	NSW 2010	APAC
6	7	London	+44 20 7877 2041	25 Old Broad Street	Level 7		UK	EC2N 1HN	EMEA

9. Retrieving a subset of columns

```
pd.read_sql("
```

```
select lastName,FirstName from employees  
'',conn).head()
```

#### 10. Use Aliases (AS keyword) to change column names

```
> pd.read_sql(  
    "select FirstName as Name from employees  
    ''',conn).head()
```

```
[ ]
```

```
..
```

	Name
0	Diane
1	Mary
2	Jeff
3	William
4	Gerard

#### 11. Using SQL case statements

```
pd.read_sql(  
    "select Firstname,LastName,jobTitle,  
        CASE  
            WHEN jobTitle = 'Sales Rep' then 'Sales Rep'  
            ELSE 'Not Sales Rep'  
        END AS role  
    From employees  
    ''',conn)
```

#### 12. Cases to make Human Readable

```
pd.read_sql(  
    "select FirstName,lastName,officeCode,  
        CASE officeCode  
            WHEN '1' then 'San Francisco, CA'  
            WHEN '2' then 'Boston, MA'  
            WHEN '3' then 'New York, NY'  
            WHEN '4' then 'Paris, France '  
            WHEN '5' then 'Tokyo, Japan'  
        END as office  
    from employees  
    ''',conn).head(10)
```

#### 13. a)Check length

```
pd.read_sql(  
    "select length(firstName) as name_length  
        from employees  
    ",conn).head(5)
```

b)Convert to upper

```
pd.read_sql(  
    "select upper(firstName) as name_in_all_caps  
        from employees;  
    ",conn).head(5)
```

## length

Returns the no of characters.if we wanted to find the length of the firstname  
Run and Debug (Ctrl+Shift+D) this:

```
pd.read_sql(''  
    select length(firstName) as name_length  
    |      from employees  
    ''',conn).head(5)
```

[ ]

...

### name\_length

0	5
1	4
2	4
3	7
4	6

## Upper

return all employee names in all caps

▷ ▾

```
pd.read_sql(''  
    select upper(firstName) as name_in_all_caps  
    |      from employees;  
    ''',conn).head[5]
```

[ ]

...

### name\_in\_all\_caps

0	DIANE
1	MARY
2	JEFF
3	WILLIAM
4	GERARD

### c) Slicing

```
select substr(firstname,1,1) as first_initial  
from employees;
```

```
select substr(firstName,1,1) || '.' as first_initial --use || as concatenate operator  
from employees
```

```
pd.read_sql(''  
    select substr(firstname,1,1) as first_initial  
    from employees;  
    ''',conn).head(5)  
    ✓ 0.0s
```

	first_initial
0	D
1	M
2	J
3	W
4	G

```
pd.read_sql(''  
    select substr(firstName,1,1) || '.' as first_initial --use || as concatenate operator  
    from employees  
    ''',conn)  
    ✓ 0.0s
```

	first_initial
0	D.
1	M.
2	J.
3	W.
4	G.

d) We can also **combine multiple column values**, not just string literals

```
pd.read_sql("'  
    select firstname || ' ' || lastname as name  
    from employees  
    ''',conn).head()
```

We can also combine multiple column values, not just string literals.

lets combine the first and last name:

```
pd.read_sql(''  
    select firstname || ' ' || lastname as name  
    from employees  
''',conn).head()
```

	name
0	Diane Murphy
1	Mary Patterson
2	Jeff Firrelli
3	William Patterson
4	Gerard Bondur

e)Aggregations eg count

```
pd.read_sql("'  
    select COUNT() as "Number of Airports"  
    from airports  
    ''',con)
```

Some functions will aggregate your data and return a table ✓

```
pd.read_sql(''  
    select COUNT() as "Number of Airports"  
    from airports  
    ''',con)  
3] ✓ 0.0s
```

Number of Airports
0
8107

NB: in AGGREGATIONS ensure that's it's the correct datatype for the function to avoid unexpected results eg

## Testing Datatype Compatibility

Make sure your column is the right datatype for the function to avoid unexpected results

[24]

```
pd.read_sql(''  
    select name AS "Airport Name",  
           MAX(alitude) AS "Altitude(ft)"  
        from airports  
'',con) ### did max on string and 9 comes at the end
```

✓ 0.0s

...

	Airport Name	Altitude(ft)
--	--------------	--------------

0	Dauphin Barker	999
---	----------------	-----

## CAST

You could fix this using the CAST() function

[25]

```
pd.read_sql(''  
    select name AS "Airport Name",  
           MAX(CAST(alitude as INT)) AS "Altitude(ft)"  
        from airports  
'',con)
```

✓ 0.0s

...

	Airport Name	Altitude(ft)
--	--------------	--------------

0	Yading Daocheng	14472
---	-----------------	-------

## 14. Built in SQL for Math Functions

### round

```
▷ pd.read_sql('''
    select round(priceEach) as rounded_price
    from orderDetails
    ''',conn)
```

[ ]

...

	rounded_price
0	136.0
1	55.0
2	75.0
3	35.0
4	108.0
...	...
2991	128.0
2992	32.0
2993	84.0
2994	50.0
2995	95.0

2996 rows × 1 columns

### CAST

The previous result looks ok, but it's returning floating point numbers. What if we want integers instead?

```
pd.read_sql('''
    select cast(round(priceEach) as INTEGER) as rounded_price_int
    from orderDetails
    ''',conn)
```

[ ]

...

	rounded_price_int
0	136

## 15. Built-in SQL Functions for Date and Time Operations-

(JulianDay and Date)

What if we wanted to know how many days there are between the requiredDate and the orderDate for each order?

```
[ ] pd.read_sql('''
    select julianday(requiredDate)- julianday(orderDate) AS days_from_order_to_required
    from orders;
    |   |   ''',conn)
```

Python

...  
days\_from\_order\_to\_required

0	7.0
1	9.0
2	8.0
3	9.0
4	9.0
...	...
321	8.0
322	12.0
323	6.0
324	8.0
325	7.0

326 rows × 1 columns

If we wanted to select the order dates as well as dates 1 week after the order dates,

```
[19] pd.read_sql('''
    select orderDate, date(orderDate,'+7 days') AS one_week_later
    from orders;
    |   |   ''',conn)
```

Python

...  
orderDate one\_week\_later

0	2003-01-06	2003-01-13
1	2003-01-09	2003-01-16
2	2003-01-10	2003-01-17

## 16. strftime function

```
pd.read_sql(""
select orderDate,
       strftime('%m',orderDate) AS month,
       strftime('%Y',orderDate) AS year,
       strftime('%d',orderDate) as day
from orders;
'',conn)
```

You can also use the strftime function, which is very similar to the Python version. This is useful if you want to split apart a date or time value into different sub-parts. For example, here we extract the year, month, and day of month from the order date:

```
pd.read_sql('''
    select orderDate,
           strftime('%m',orderDate) AS month,
           strftime('%Y',orderDate) AS year,
           strftime('%d',orderDate) as day
      from orders;
''',conn)
```

```
[ ] Python
```

... 

	orderDate	month	year	day
0	2003-01-06	01	2003	06
1	2003-01-09	01	2003	09
2	2003-01-10	01	2003	10
3	2003-01-29	01	2003	29
4	2003-01-31	01	2003	31
...	...	...	...	...
321	2005-05-29	05	2005	29
322	2005-05-30	05	2005	30
323	2005-05-30	05	2005	30
324	2005-05-31	05	2005	31
325	2005-05-31	05	2005	31

  
326 rows × 4 columns

Now that we are finished with our queries, we can close the database connection.

```
[ ] conn.close()
```

13. If we really wanted to, we could just use those same SQLite terminal commands directly in a Jupyter Notebook using magic commands.

```
%%script sqlite3 data/data.sqlite --out offices_data
select* from offices;
```

```
%%script sqlite3 data/data.sqlite --out offices_data  
select* from offices;
```

✓ 0.0s

Python

```
print(offices_data)
```

✓ 0.0s

Python

```
1|San Francisco|+1 650 219 4782|100 Market Street|Suite 300|CA|USA|94080|NA  
2|Boston|+1 215 837 0825|1550 Court Place|Suite 102|MA|USA|02107|NA  
3|NYC|+1 212 555 3000|523 East 53rd Street|apt. 5A|NY|USA|10022|NA  
4|Paris|+33 14 723 4404|43 Rue Jouffroy D'abbans|||France|75017|EMEA  
5|Tokyo|+81 33 224 5000|4-1 Kioicho||Chiyoda-Ku|Japan|102-8578|Japan  
6|Sydney|+61 2 9264 2451|5-11 Wentworth Avenue|Floor #2||Australia|NSW 2010|APAC  
7|London|+44 20 7877 2041|25 Old Broad Street|Level 7||UK|EC2N 1HN|EMEA
```

```
%%script sqlite3 data/data.sqlite --out offices_schema  
.schema offices  
.quit|
```

✓ 0.0s

Python

```
print(offices_schema)
```

✓ 0.0s

Python

```
CREATE TABLE `offices` (`officeCode`, `city`, `phone`, `addressLine1`, `addressLine2`, `state`, `country`, `pc
```

```
%%script sqlite3 data.sqlite --out tables  
.tables  
.quit
```

```

%%script sqlite3 data.sqlite --out tables
.tables
.quit

print(tables)

customers      offices      orders      productlines
employees      orderdetails  payments    products

type(tables),type(offices_data),type(offices_schema)

(str, str, str)

```

## 17. Connecting to the database using the terminal

```

$ sqlite3 data/data.sqlite
.tables
customers      offices      orders      productlines
employees      orderdetails  payments    products
select * from offices;
1|San Francisco|+1 650 219 4782|100 Market Street|Suite 300|CA|USA|94080|NA
2|Boston|+1 215 837 0825|1550 Court Place|Suite 102|MA|USA|02107|NA
3|NYC|+1 212 555 3000|523 East 53rd Street|apt. 5A|NY|USA|10022|NA
4|Paris|+33 14 723 4404|43 Rue Jouffroy D'abbans|||France|75017|EMEA
5|Tokyo|+81 33 224 5000|4-1 Kioicho||Chiyoda-Ku|Japan|102-8578|Japan
6|Sydney|+61 2 9264 2451|5-11 Wentworth Avenue|Floor #2||Australia|NSW 2010|APAC
7|London|+44 20 7877 2041|25 Old Broad Street|Level 7||UK|EC2N 1HN|EMEA
.schema offices
CREATE TABLE `offices` (`officeCode`, `city`, `phone`, `addressLine1`, `addressLine2`, `state`, `country`, `postalCode`, `territory`);
.quit
(learn-env)

```

## 18. Relational databases

typically have multiple tables containing data, and the tables have defined relationships

Database **Schema** – defines the structure of the database, including the tables and relationships between tables

**Primary key** – uniquely identifies each row in a table

**Foreign key** - used in one table to refer to the primary key of another table

## 19. Access schema of lets say the first table

```
print(schema_df['sql'].iloc[0]) #print(schema_df.iloc[0]['sql'])
```

```
schema_df = pd.read_sql('select * from sqlite_master',con)
schema_df
```

✓ 0.0s

	<b>type</b>	<b>name</b>	<b>tbl_name</b>	<b>rootpage</b>	<b>sql</b>
0	table	airports	airports	2	CREATE TABLE airports ([index] INTEGER,\n [id] TEXT,\n [name] TEXT,\n [city] TEXT,\n [country] TEXT,\n [code] TEXT,\n [icao] TEXT,\n [latitude] TEXT,\n [longitude] TEXT,\n [altitude] TEXT,\n [offset] TEXT,\n [dst] TEXT,\n [timezone] TEXT
1	index	ix_airports_index	airports	3	CREATE INDEX ix_airports_index ON airports ([index])
2	table	airlines	airlines	945	CREATE TABLE airlines ([index] INTEGER,\n [id] TEXT,\n [name] TEXT,\n [code] TEXT,\n [iata] TEXT,\n [icao] TEXT,\n [callsign] TEXT,\n [country] TEXT,\n [frequency] TEXT,\n [lat] REAL,\n [lon] REAL,\n [elevation] REAL,\n [wkt] TEXT)
3	index	ix_airlines_index	airlines	946	CREATE INDEX ix_airlines_index ON airlines ([index])
4	table	routes	routes	1393	CREATE TABLE routes ([index] INTEGER,\n [id] TEXT,\n [origin] TEXT,\n [destination] TEXT,\n [airline] TEXT,\n [distance] REAL,\n [duration] REAL,\n [stops] INTEGER,\n [lat] REAL,\n [lon] REAL,\n [elevation] REAL,\n [wkt] TEXT)
5	index	ix_routes_index	routes	1394	CREATE INDEX ix_routes_index ON routes ([index])

It looks like there are three tables in our database: airports, airlines, and routes. Each table also has an index used to optimize queries for large datasets.

The column names and datatypes for each table are defined in the schema in the sql column.

```
print(schema_df['sql'].iloc[0]) #print(schema_df.iloc[0]['sql'])
```

✓ 0.0s

```
CREATE TABLE airports (
[index] INTEGER,
[id] TEXT,
[name] TEXT,
[city] TEXT,
[country] TEXT,
[code] TEXT,
[icao] TEXT,
[latitude] TEXT,
[longitude] TEXT,
[altitude] TEXT,
[offset] TEXT,
[dst] TEXT,
[timezone] TEXT
)
```

## 20. Between

```
pd.read_sql("""
SELECT name AS "Airport Name",
       CAST(latitude AS int) AS "Airport Latitude",
       CAST(altitude AS int) AS "Altitude (ft)"
FROM airports
WHERE "Airport Latitude" between 20 AND 50 AND "Altitude (ft)">10000
```

```
","",con)

▷ pd.read_sql('''
    SELECT name AS "Airport Name",
           CAST(latitude AS int) AS "Airport Latitude",
           CAST(altitude AS int) AS "Altitude (ft)"
      FROM airports
     WHERE "Airport Latitude" between 20 AND 50 AND "Altitude (ft)">10000
      ''',con)
[28] ✓ 0.0s
```

	Airport Name	Airport Latitude	Altitude (ft)
0	Leh	34	10682
1	Hongyuan Airfield	32	11500
2	Lhasa-Gonggar	29	13136
3	Manang	28	11000
4	Jiuzhaigou Huanglong	32	11311
5	Qamdo Bangda Airport	30	14219
6	Maiwa	33	11500
7	Syangboche	27	12309
8	Yushu Batang	32	13000
9	Gunsa	32	13780
10	Shigatse Peace Airport	29	12408
11	Kangding Airport	30	14042
12	Yading Daocheng	29	14472
13	Gannan	34	10466

## 21. IS – Useful when working with NULL values

```
pd.read_sql("'
SELECT name AS "Airport Name",
       code AS "Airport Code"
  FROM airports
 WHERE "Airport Code" IS NOT NULL
  "",con)
```

```

pd.read_sql('''
SELECT name AS "Airport Name",
       code as "Airport Code"
FROM airports
WHERE "Airport Code" IS NOT NULL
''',con)

```

✓ 0.0s

	Airport Name	Airport Code
0	Goroka	GKA
1	Madang	MAG
2	Mount Hagen	HGU
3	Nadzab	LAE
4	Port Moresby Jacksons Intl	POM
...	...	...
5875	Mansons Landing Water Aerodrome	YMU
5876	Port McNeill Airport	YMP
5877	Sullivan Bay Water Aerodrome	YTG
5878	Deer Harbor Seaplane	DHB
5879	San Diego Old Town Transit Center	OLT

5880 rows × 2 columns

## 22. ORDER BY

```

pd.read_sql("
SELECT name AS "Aiport Name",
       CAST(latitude AS int) AS "Airport Latitude",
       CAST(altitude AS int) AS "Altitude(ft)"

FROM airports
WHERE "Altitude(ft)" >=10000

ORDER BY "Airport Latitude" DESC,
         "Altitude(ft)" DESC
      ",con)

```

```

pd.read_sql("""
SELECT name AS "Aiport Name",
       CAST(latitude AS int) AS "Airport Latitude",
       CAST(altitude AS int) AS "Altitude(ft)"
    |
FROM airports
WHERE "Altitude(ft)" >=10000

ORDER BY "Airport Latitude" DESC,
         "Altitude(ft)" DESC
""",con)

```

✓ 0.0s

	Aiport Name	Airport Latitude	Altitude(ft)
0	Summit Camp	72	11000
1	Summit Camp	72	10552
2	Irkutsk-2	52	13411
3	Leh	34	10682
4	Gannan	34	10466
5	Maiwa	33	11500
6	Gunsa	32	13780
7	Yushu Batang	32	13000
8	Hongyuan Airfield	32	11500
9	Jiuzhaigou Huanglong	32	11311
10	Qamdo Bangda Airport	30	14219
11	Kangding Airport	30	14042
12	Yading Daocheng	29	14472
13	Lhasa-Gonggar	29	13136
14	Shigatse Peace Airport	29	12408
15	Manang	28	11000
16	Syangboche	27	12309
17	Francisco Carle	-11	11034
18	Poroy Train Station	-13	11710
19	Andahuaylas	-13	11300
20	Teniente Alejandro Velasco Astete Intl	-13	10860
21	Julianca	-15	12552

## 23. LIMIT

## LIMIT Number of Results

Specify the maximum number of results you want

```
pd.read_sql(''  
SELECT name AS "Aiport Name",  
          CAST(latitude AS int) AS "Airport Latitude",  
          CAST(altitude AS int) AS "Altitude(ft)"  
  
         FROM airports  
        WHERE "Altitude(ft)" >=10000  
  
        ORDER BY "Airport Latitude" DESC,  
                 | "Altitude(ft)" DESC  
        LIMIT 5  
        ''',con)  
31]    ✓ 0.0s
```

	Aiport Name	Airport Latitude	Altitude(ft)
0	Summit Camp	72	11000
1	Summit Camp	72	10552
2	Irkutsk-2	52	13411
3	Leh	34	10682
4	Gannan	34	10466

24. a)Filtering by price

```

pd.read_sql('''
SELECT *,CAST(round(priceEach) AS INTEGER) AS price_int
FROM orderdetails
WHERE price_int=30

''',conn)

```

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	price_int
0	10104	S24_2840	44	30.41	10	30
1	10173	S24_1937	31	29.87	9	30
2	10184	S24_2840	42	30.06	7	30
3	10280	S24_1937	20	29.87	12	30
4	10332	S24_1937	45	29.87	6	30
5	10367	S24_1937	23	29.54	13	30
6	10380	S24_1937	32	29.87	4	30

```

pd.read_sql('''
SELECT *
FROM orderdetails
where CAST(round(priceEach) AS INT) =30
''',conn)

```

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
0	10104	S24_2840	44	30.41	10
1	10173	S24_1937	31	29.87	9
2	10184	S24_2840	42	30.06	7
3	10280	S24_1937	20	29.87	12
4	10332	S24_1937	45	29.87	6
5	10367	S24_1937	23	29.54	13
6	10380	S24_1937	32	29.87	4

b)Filter by Date eg select all January orders

## Selecting Orders Based on Date

We can use the strftime function to select all orders placed in January of any year:

```
pd.read_sql("""  
    SELECT * ,strftime('%m',orderDate) as Month  
    FROM orders  
    WHERE Month='01'  
""",conn)
```

Python

	orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber	Month
0	10100	2003-01-06	2003-01-13	2003-01-10	Shipped		363	01
1	10101	2003-01-09	2003-01-18	2003-01-11	Shipped	Check on availability.	128	01
2	10102	2003-01-10	2003-01-18	2003-01-14	Shipped		181	01
3	10103	2003-01-29	2003-02-07	2003-02-02	Shipped		121	01
4	10104	2003-01-31	2003-02-09	2003-02-01	Shipped		141	01

### C)Filter late orders

```
pd.read_sql(  
    "SELECT julianday(shippedDate) - julianday(requiredDate) as days_late ,*  
    FROM orders  
    WHERE days_late >0  
    ",conn)
```

Python

	days_late	orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber
0	56.0	10165	2003-10-22	2003-10-31	2003-12-26	Shipped	This order was on hold because customers's cre...	148

### d)Like – select cats starting with ‘M’ or ‘m’

```
pd.read_sql('''
SELECT *
FROM cats
WHERE name like 'm%'
''', conn)
```

✓ 0.0s

	<b>id</b>	<b>name</b>	<b>age</b>	<b>breed</b>	<b>owner_id</b>
0	1	Maru	3	Scottish Fold	1.0
1	4	Moe	10	Tabby	NaN

e) select all cats with four-letter names where the second letter was "a", we could use `_a__`

```
pd.read_sql(""
SELECT *
FROM cats
WHERE name like '_a__'
",conn)
```

[21] ✓ 0.0s

	<b>id</b>	<b>name</b>	<b>age</b>	<b>breed</b>	<b>owner_id</b>
0	1	Maru	3	Scottish Fold	1
1	2	Hana	1	Tabby	1

Python

Again, we could have done this using length and substr, although it would be much less concise:

[22] ✓ 0.0s

	<b>id</b>	<b>name</b>	<b>age</b>	<b>breed</b>	<b>owner_id</b>
0	1	Maru	3	Scottish Fold	1
1	2	Hana	1	Tabby	1

Python

25. Select from 2 tables

```
pd.read_sql(""
select cats.name,dogs.name
from cats,dogs;
",conn)
```

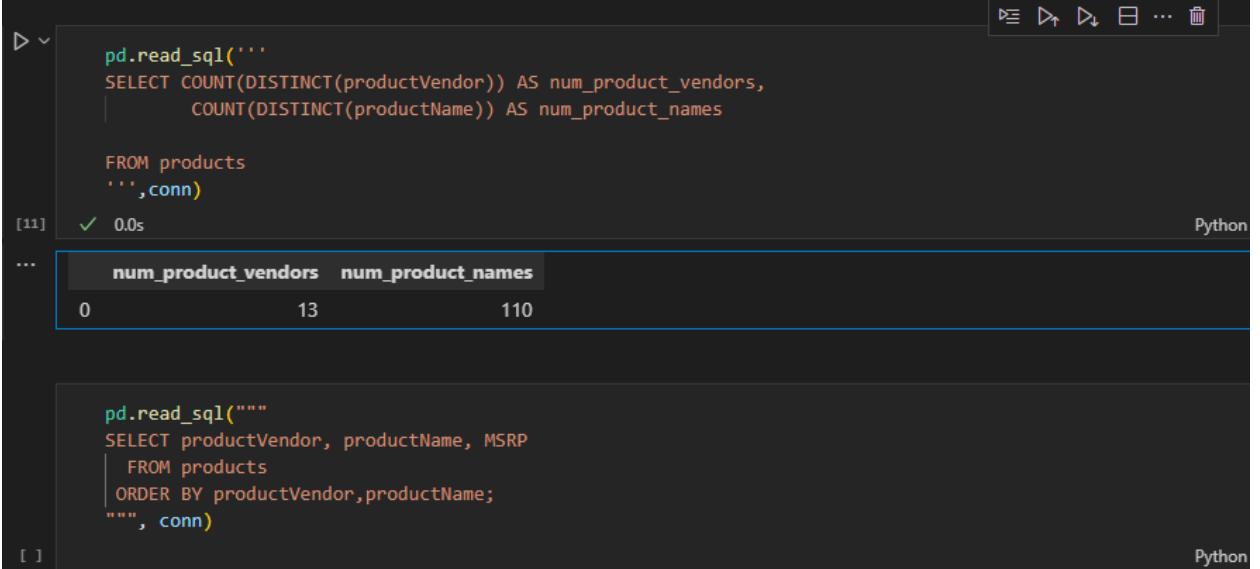
```
pd.read_sql(''  
    select cats.name,dogs.name  
    from cats  
    left join dogs  
    on cats.id = dogs.cat_id  
    where cats.breed = "Domestic Shorthair"  
    order by cats.name, dogs.name  
', conn)
```

	name	name
0	Maru	Clifford
1	Maru	Jack
2	Maru	Lily
3	Maru	Clifford
4	Maru	Jack
5	Maru	Lily
6	Hana	Clifford
7	Hana	Jack
8	Hana	Lily
9	Hana	Clifford
10	Hana	Jack
11	Hana	Lily
12	Lil' Bub	Clifford
13	Lil' Bub	Jack
14	Lil' Bub	Lily
15	Lil' Bub	Clifford
16	Lil' Bub	Jack

26. way of thinking about which column should come first after ORDER BY, is that the fewer unique values the column has, the earlier it should appear

Another way of thinking about which column should come first after ORDER BY, is that the fewer unique values the column has, the earlier it should appear.

To find out how many unique values there are in a column, we can use the DISTINCT keyword in addition to COUNT. Below are some examples:



```
pd.read_sql('''
SELECT COUNT(DISTINCT(productVendor)) AS num_product_vendors,
       COUNT(DISTINCT(productName)) AS num_product_names
FROM products
''', conn)
[11]: 0.0s
```

	num_product_vendors	num_product_names
0	13	110

```
pd.read_sql(""""
SELECT productVendor, productName, MSRP
FROM products
ORDER BY productVendor,productName;
""", conn)
```

This result aligns with what we observed earlier. There are many more unique product names than product vendors so it makes sense to sort by vendors first, then names.

## 27. Create database and tables

```
import sqlite3
conn = sqlite3.connect('pets_database.db')
cur = conn.cursor()
```

a)Create Cats table and insert records

#Creating the cats table

```
cur.execute("")
```

```
CREATE TABLE cats(
```

```
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    breed TEXT
)
```

```
")
```

```
""")
```

```
===== cur.execute("")
```

```
INSERT INTO cats(name,age,breed)
```

```
    VALUES('Maru',3,'Scottish Fold'),
           ('Lil Bub',2,'Sheperd cat'),
           ('Hannah',4,'German cats')
```

```
""")
```

```
#Creating the cats table
cur.execute('''
CREATE TABLE cats(
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    breed TEXT
)
...''')
41] .. <sqlite3.Cursor at 0x1fd53568c70>
```

## Populating Tables

in order to populate a table, you can use the `INSERT INTO` command,followed by the name of the table to which we want to add data. Then in parentheses, we type the column names that we want to fill the data with. This followed by the `VALUES` Keyword, which is accompanied by a parentheses enclosed list of the values that correspond to each column name

**Important:** Note that you don't have to specify the "id" column name or value. Primary Key columns are auto-incrementing. Therefore, since the cats table has an "id" column whose type is INTEGER PRIMARY KEY, you don't have to specify the id column values when you insert data. As long as you have defined an id column with a data type of INTEGER PRIMARY KEY, a newly inserted row's id column will be automatically given the correct value.

Okay, let's start storing some cats.

### Code Along I: INSERT INTO

To insert a record with values, type the following:

```
# insert Maru into the pet_database.db here
cur.execute('''
INSERT INTO cats(name,age,breed)
VALUES('Maru',3,'Scottish Fold'),
      ('Lil Bub',2,'Sheperd cat'),
      ('Hannah',4,'German cats')
...'''')
```

## 28. Altering(update) and deleting data

## Altering a Table

You can also update a table like this: `cursor.execute(" ALTER TABLE CATS ADD COLUMN notes text")` The general pattern is `ALTER TABLE table_name ADD COLUMN column_name column_type`

```
▷ 
  cur.execute('''
    UPDATE cats
      SET name = 'Hana'
    WHERE name='Hannah'
  ''')
[43] Python
... <sqlite3.Cursor at 0x1fd53568c70>
```

## Deleting Data

You can use the `DELETE` keyword to delete table rows similar to the `UPDATE` keyword. The `DELETE` keyword uses a `WHERE` clause to select rows. A boilerplate DELETE statement looks like this:

```
curr.execute( '''DELETE FROM [table name]
  WHERE [column name] = [value]
  ...
)'''
```

## Code Along III: DELETE

Let's go ahead and delete Lil' Bub from our cats table (sorry Lil' Bub):

```
cur.execute('''
  DELeTE FROM cats
  WHERE id=2
  ''')
[45] Python
```

29. We need to save the changes otherwise when we create a new connection and access record we find that it's empty

## Saving changes

Source Control (Ctrl+Shift+G)

While everything may look well and good, if you were to connect to the database from another Jupyter notebook (or elsewhere) the database would appear blank! That is, while the changes are reflected in your current session connection to the database you have yet to commit those changes to the master database so that other users and connections can also view the updates.

Before you commit the changes, let's demonstrate this concept.

First, preview the results of the table:

```
[48] #Preview the table via the current cursor/connection
      cur.execute('''
      select * from cats
      ''').fetchall()
...
[(1, 'Maru', 3, 'Scottish Fold'), (3, 'Hana', 4, 'German cats')]
```

Python

Now, to demonstrate that these changes aren't reflected to other connections to the database create a 2nd connection/cursor and run the same preview:

```
[49] conn2 = sqlite3.connect('pets_database.db')
      cur2 = conn2.cursor()
      cur2.execute('''SELECT * FROM cats''').fetchall()
```

Python

...

As you can see, the second connection doesn't currently display any data in the cats table! To make the changes universally accessible commit the changes.

In this case:

```
[51] # Commit your changes to the database
      conn.commit()
```

Python

## 30. Insert into table from dictionary

```

# Load the list of dictionaries just from this file
import pickle

with open('contact_list.pickle', 'rb') as f:
    contacts = pickle.load(f)

for contact in contacts:
    print(contact)

```

Python

```

{'firstName': 'Christine', 'lastName': 'Holden', 'role': 'staff', 'telephone ': 2035687697, 'street': '1672
{'firstName': 'Christopher', 'lastName': 'Warren', 'role': 'student', 'telephone ': 2175150957, 'street': '1
{'firstName': 'Linda', 'lastName': 'Jacobson', 'role': 'staff', 'telephone ': 4049446441, 'street': '479 Mus
{'firstName': 'Andrew', 'lastName': 'Stepp', 'role': 'student', 'telephone ': 7866419252, 'street': '2981 La
{'firstName': 'Jane', 'lastName': 'Evans', 'role': 'student', 'telephone ': 3259909290, 'street': '1461 Bria
{'firstName': 'Jane', 'lastName': 'Evans', 'role': 'student', 'telephone ': 3259909290, 'street': '1461 Bria
{'firstName': 'Mary', 'lastName': 'Raines', 'role': 'student', 'telephone ': 9075772295, 'street': '3975 Jen
{'firstName': 'Ed', 'lastName': 'Lyman', 'role': 'student', 'telephone ': 5179695576, 'street': '3478 Be Sre

```

# Iterate over the contact list and populate the contactInfo table here

```

for contact in contacts:
    firstName = contact['firstName']
    lastName = contact['lastName']
    role = contact['role']
    telephone = contact['telephone ']
    street = contact['street']
    city = contact['city']
    state = contact['state']
    zipcode = contact['zipcode ']

    cur.execute('''
        INSERT INTO contactinfo(firstName,lastName,role,telephone,street,city,state,zipcode)
        VALUES('{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}');
    '''.format(firstName,lastName,role,telephone,street,city,state,zipcode))

```

Python

### Query the Table to Ensure it is populated

#### 31. Create table with dual key

```

cur.execute("""
CREATE TABLE Grades(
    userid INTEGER NOT NULL,
    courseid INTEGER NOT NULL,
    grade INTEGER,
    PRIMARY KEY(userid,courseid)
)
""")

```

```
# Create the grades table
cur.execute('''
CREATE TABLE Grades(
    userid INTEGER NOT NULL,
    courseid INTEGER NOT NULL,
    grade INTEGER,
    PRIMARY KEY(userid,courseid)
)''')
```

Python

```
<sqlite3.Cursor at 0x18da49abc70>
```

## Remove Duplicate Entries

An analyst just realized that there is a duplicate entry in the contactInfo table! Find and remove it.

```
# Find the duplicate entry
cur.execute('''
SELECT firstName ,lastName , telephone,COUNT(*)
FROM contactinfo
GROUP BY firstName ,lastName , telephone
HAVING
    COUNT(*) >1
''').fetchall()
```

Python

```
[('Jane', 'Evans', 3259909290, 2)]
```

```
# Delete the duplicate entry
cur.execute('''
DELETE FROM contactinfo where id=5
''')
```

Python

**32. Typing** – the practice of explicitly declaring a type. Exercises some level of control over our data. Without typing our data can become complicated and messy and it would be difficult to ask the database questions about large sets of data

- TEXT -str
- INTEGER- int
- REAL -float
- BLOB

## 33. JOINS

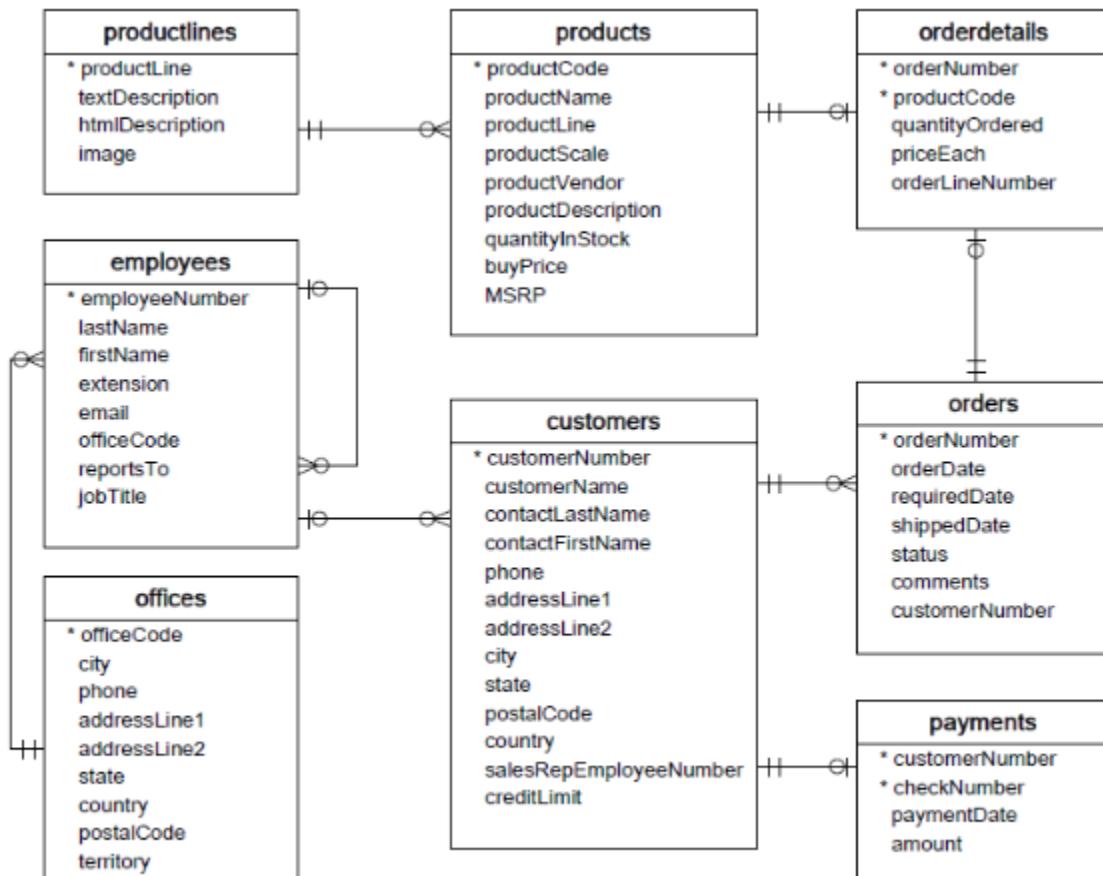
```
q= ""
select *
FROM orderdetails
JOIN products
ON orderdetails.productCode = products.productCode
LIMIT 10
""
```

```
pd.read_sql(q,conn)
```

### b) Columns have the same name

```
#IF COLUMNS HAVE THE SAME NAME
q= ""
select *
FROM orderdetails
JOIN products
USING (productCode)
LIMIT 10
""
pd.read_sql(q,conn)
```

**34. JOIN IF table is related by joining another table eg checking how many customers are there per office yet customer does relate to offices but related through employee**



q = "'''"

```
SELECT
o.officeCode,
o.city,
COUNT(c.customerNumber) AS n_customers
```

```

FROM offices AS o
JOIN employees AS e
  USING(officeCode)
JOIN customers AS c
  ON e.employeeNumber = c.salesRepEmployeeNumber
GROUP BY officeCode
;
"""

```

```

pd.read_sql(q, conn)
#teacher's code
q = """
SELECT
    o.officeCode,
    o.city,
    COUNT(c.customerNumber) AS n_customers
FROM offices AS o
JOIN employees AS e
  USING(officeCode)
JOIN customers AS c
  ON e.employeeNumber = c.salesRepEmployeeNumber
GROUP BY officeCode
;
"""

pd.read_sql(q, conn)

```

✓ 0.0s

	<b>officeCode</b>	<b>city</b>	<b>n_customers</b>
0	1	San Francisco	12
1	2	Boston	12
2	3	NYC	15
3	4	Paris	29
4	5	Tokyo	5
5	6	Sydney	10
6	7	London	17

### 35. Cast for numeric purposes

```

q=""
SELECT
-- Note we have to cast to a numerical value first
MAX(CAST(longitude as REAL)) AS max_longitude
From airports
"""
pd.read_sql(q,conn)

```

```
q='''  
SELECT  
-- Note we have to cast to a numerical value first  
MAX(CAST(longitude as REAL)) AS max_longitude  
From airports  
...  
pd.read_sql(q,conn)  
✓ 0.0s
```

### max\_longitude

0	179.951
---	---------

```
q = '''  
SELECT longitude  
FROM airports  
ORDER BY CAST(longitude AS REAL) desc  
LIMIT 1  
'''  
pd.read_sql(q,conn)  
✓ 0.0s
```

### longitude

0	179.951
---	---------

```
q = '''  
SELECT longitude  
FROM airports  
WHERE longitude = (  
SELECT MAX(CAST(longitude as REAL))  
From airports  
)  
'''  
pd.read_sql(q,conn)
```

✓ 0.0s
--------

### longitude

0	179.951
---	---------

## 36. Using eval to create new columns

```
df = df.eval('Age_x_Fare= Age * Fare')  
df.head()
```

## 37. Querying dataframes with sql(using pandassql)- easier to write than from dataframes

```
pip install pandasql
from pandasql import sqldf
pysqldf = lambda q: sqldf(q,globals())
q = ""
SELECT name
FROM df
LIMIT 10;
 ""

passenger_names = pysqldf(q)
passenger_names
```

```

q =
SELECT name
FROM df
LIMIT 10;
...

passenger_names = pymysqldf(q)
passenger_names

```

	Name
0	Braund, Mr. Owen Harris
1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina
2	Futrelle, Mrs. Jacques Heath (Lily May Peel)
3	Allen, Mr. William Henry
4	Moran, Mr. James
5	McCarthy, Mr. Timothy J
6	Palsson, Master. Gosta Leonard
7	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
8	Nasser, Mrs. Nicholas (Adele Achem)
9	

eat! Now, for a harder one:

the cell below, query the DataFrame for names and fares of any male passengers that survived, limit 30.

```

q2 = '''
SELECT Name,Fare FROM
df where Sex="male" AND Survived=1
LIMIT 30
'''

sql_surviving_males = pymysqldf(q2)
sql_surviving_males

```

	Name	Fare
0	Williams, Mr. Charles Eugene	13.0000
1	Beesley, Mr. Lawrence	13.0000

**38. Statistical Distribution** – Representation of the frequencies of potential events or the percentage of time each event occurs.

**a. Discrete Distribution-** Known number of possible outcomes(describes random variables that can take on a countable number of distinct values).use PMF

Eg

- ✓ Rolling a dice(possible outcomes 1,2,3,4,5,6)
- ✓ Number of defective items in a batch
- ✓ Count of customers arriving at the store in an hour

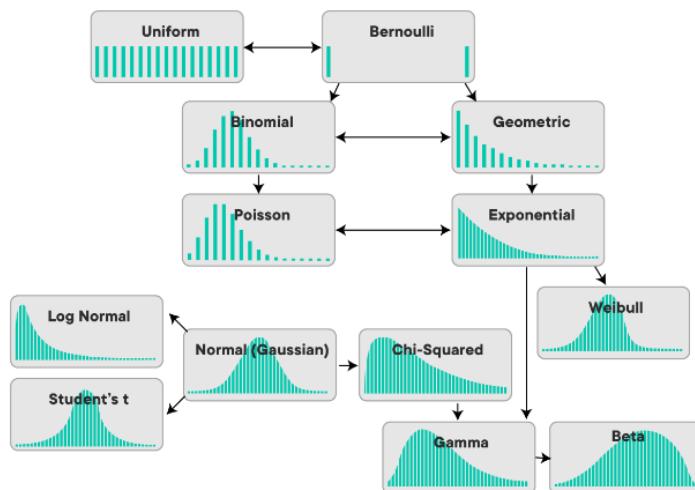
**b. Continuous Distributions** – infinite no of possible values(describes random variables that can take on an infinite number of values within a given range)

Eg

- ✓ Heights of individuals
- ✓ Time taken to complete a task
- ✓ Temperature measurements

## Common Distributions

In this image, you can see the general shapes of some common distributions. The horizontal axis in each chart represents the set of possible numeric outcomes. The vertical axis describes the probability of the respective outcomes.



You'll get a more in-depth overview of some important distributions in the next few lessons, but to give you an initial idea of some applications, we'll give you a quick overview below. Let's quickly talk about some common distributions and their use cases below.

## c. Common distributions and their use cases

### Examples of Discrete Distributions

- ✓ **The Bernoulli Distribution** – deals with a series of Boolean events eg coin toss
- ✓ **Poisson Distribution** – rep the likelihood of a given no of successes over a given period of time. A typical example is pieces of mail. If your overall mail received is constant, the number of items received on a single day (or month) follows a Poisson distribution. Other examples might include visitors to a website, or customers arriving at a store, or clients waiting to be served in a queue.
- ✓ **Uniform distribution** - Occurs when all possible outcomes are equally likely eg dice
- ✓ **Binomial distribution** – Probability of observing a specific no of successes (Bernoulli trials) in a specific no of trials
- ✓ **Geometric**

### Examples of Continuous Distributions

- ✓ **Normal/Gaussian Distribution** – follows a bell shape and is a foundational distribution for many models and theories un statistics and data science. A normal distribution turns up very often when dealing with real-world data including heights, weights of different people, errors in some measurement or grades on a test. Our temperature example above follows a normal distribution as well!
- ✓ **Exponential** – rep the amt of time it takes before an event occurs

✓ **Continous uniform**

**39. Population** -Whole set of possible outcomes(universal set)

**40. Sample** – subset of the population

**Reasons we cant observe a whole group/population**

- ✓ Expensive(not enough fuding)
- ✓ Unrealistic (grp too large)
- ✓ We don't need a whole population to gain insights
- ✓ People not willing to give info

- A large enough sample has mean close to the population mean,small samples can be misleading
- Sample means with multiple samples has mean close to the population mean

**41. Descriptive statistics** – summary metrics that describe and summarize the main features of a dataset.

They help in understanding the basic aspects of the data by providing quantitative descriptions eg measures of central tendency(mean,mode, median),measures of dispersion/spread eg(variance. Std,IQR) and measures of shape(skweness and kurtosis)

-We use visualizations such as histograms and boxplots to understand the shape of the distributions

**Population staticstics-** when descriptive statistics are applied to populations

**Point estimates-** descriptive staticstics applied to samples

- In mathematical notation, we often use different symbols for a given statistic depending on whether it applies to the population or a sample. Some examples of those differences are listed below:
  - Number
    - $N$  is the number of individuals/cases in a population
    - $n$  is the number of individuals/cases in sample
  - Mean
    - $\mu$  is the population mean (pronounced "mu")
    - $\bar{x}$  is the sample mean (pronounced "x bar")
  - Standard Deviation
    - $\sigma$  is the population standard deviation (pronounced "sigma")
    - $s$  is the sample standard deviation

**42. Probability Mass Function(PMF)** – Used for discrete random variables.It gives the probability that a discrete random variable is exacly equal to a specific value.

**PMF Intuition** - Probability is a number in the range [0,1] that is calculated as the frequency expressed as fraction of the sample size

-In order to convert any random variables frequency into a probability we need to perform the following steps

- ✓ Get the frequency of each possible value in te dataset
- ✓ Divide the frequency of ach value by the total number of values(length of the dataset)
- ✓ Get the probability of each value

**EXAMPLE1**

**Simple toy example**

```
import collections
x = [1,1,1,1,2,2,2,2,3,3,4,5,5]
counter = collections.Counter(x)
print(counter)
print(len(x))
```



```
Counter({1: 4, 2: 4, 3: 2, 5: 2, 4: 1})
13
```

```
pmf = []
for k, v in counter.items():
    pmf.append(round(v/len(x),2))
print(pmf)
print(counter.keys(),pmf)
```

```
[0.31, 0.31, 0.15, 0.08, 0.15]
dict_keys([1, 2, 3, 4, 5]) [0.31, 0.31, 0.15, 0.08, 0.15]
```

You notice that the PMF is normalized so the total probability is 1.

```
import numpy as np
np.array(pmf).sum()
```

```
1.0
```

If we want, we can write this as an actual Python function, which is "trained" using the global variables `counter` we have already declared.

```
def p(i):
    frequency = counter[i]
    total_number = len(x)
    return frequency/total_number
print('P(1) = ',p(1))
print('P(3) = ',p(2))
```

```
P(1) =  0.3076923076923077
P(3) = 0.3076923076923077
```

## Visualizing a PMF

You can inspect the probability mass function of a discrete variable by visualizing the distribution using `matplotlib`. You can use a simple bar graph to show the probability mass function using the probabilities calculated above.

Here's the code:



-Can use histogram as well but include density =True  
plt.hist(x,bins=bins,rwidth=0.25,density=True)



## Why Compare PMF Bar Graph vs. Histogram?

The idea here is to help you understand the key distinctions between a PMF bar graph and a typical histogram for showing the probabilities of categorical data.

Because a PMF is designed to work with discrete (categorical) data in the first place, no binning, counting, or normalization is needed to display the probability of each possible x value.

Histograms are typically used with continuous data to show frequencies, although they can be adapted to show similar information to PMFs. The most important thing to customize is the `density=True` argument, which tells the histogram to normalize the values and display probabilities rather than frequencies.

## EXAMPLE 2(Titanic dataset)

```

# choose column
df_gender = df['Sex'].dropna()

✓ 0.0s

# Calculate counts for each category in the column column
gender_counts = df_gender.value_counts()
gender_counts

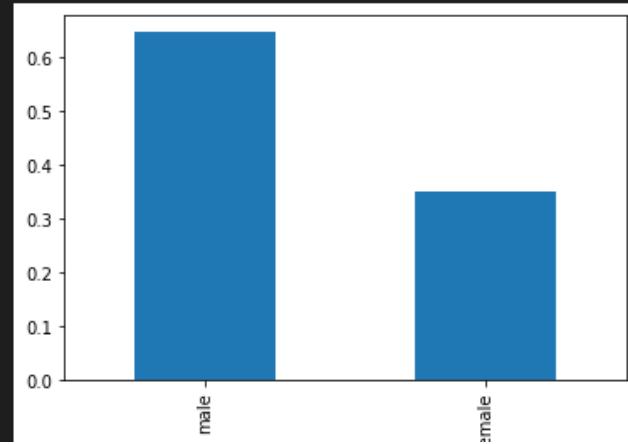
male      577
female    314
Name: Sex, dtype: int64

pmf = gender_counts/df_gender.shape[0]
pmf

male      0.647587
female    0.352413
Name: Sex, dtype: float64

# Plotting the PMF with a bar graph
pmf.plot(kind='bar');

```



**43. Probability Density Function(PDF)** – used for continuous random variables. It describes the likelihood of the variable falling within a particular range of values , rather than taking on a specific value

### PDF Intuition-

#### 1. Concept of Density

**\*\*Continuous Variables\*\*:** Unlike discrete variables, continuous variables can take on an infinite number of values within a given range (e.g., heights, weights). Therefore, we can't assign a probability to a specific value (e.g., the probability of someone being exactly 170 cm tall is zero).

Density: Instead, the PDF represents the density of probabilities. Higher values of the PDF at a point indicate a higher likelihood of the random variable being near that value.

## 2. Area Under the Curve

**\*\*Probability as Area:\*\*** The PDF itself does not give probabilities directly; instead, probabilities are found by calculating the area under the curve of the PDF over a specific interval. For instance, the probability that a random variable X falls between a and b is given by:

$$P(a < X < b) = \int_a^b f(x) dx$$

## 3. Total Area Equals One

**\*\*Normalization\*\*:** The total area under the PDF curve is always equal to 1. This reflects the fact that the random variable must take on some value within the range of possible values.

## Visualizing Probability Density Functions

---

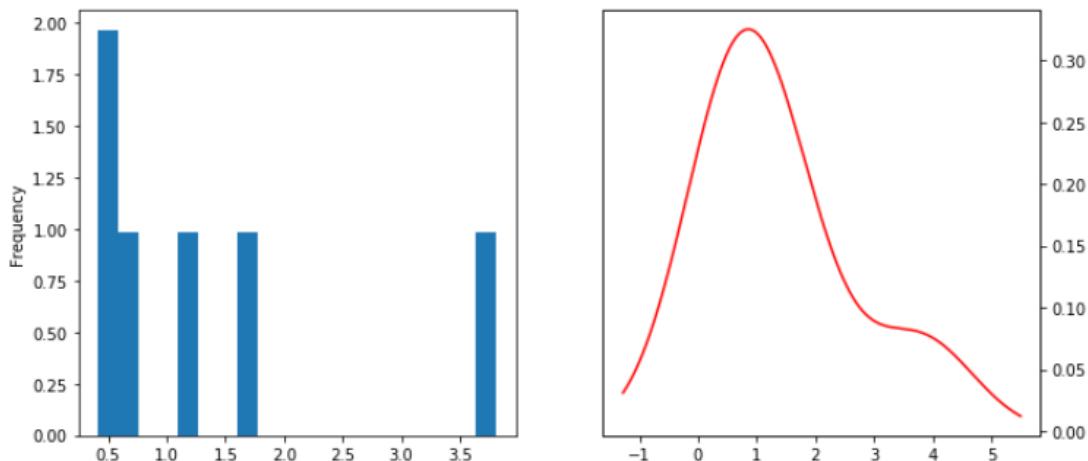
PDFs can be visualized using histograms and density plots. You've had quite a bit of practice on histograms. Now, you'll learn how to plot a density plot for a distribution in Python.

### Density Estimation and Plotting

As you've seen before, a density plot is a "smoothed" version of a histogram estimated from the observations. To estimate a density function from given continuous data, you can use parametric or non-parametric methods.

**Parametric methods** use parameters like mean and standard deviation of given data and attempt to work out the shape of the distribution that the data belongs to. These may implement maximum likelihood methods to fit a distribution to the given data. You'll learn more about this later.

**Kernel density estimation** or KDE is a common non-parametric estimation technique to plot a curve (the kernel) at every individual data point. These curves are then added to plot a smooth density estimation. The kernel most often used is a Gaussian (which produces a bell curve at each data point). Other kernels can be used in special cases when the underlying distribution is not normal.



In the image above, the histogram (left) and kernel density estimate (right) are constructed using the same data.

### EXAMPLE:TITANIC DATA SET

```
# imports
from scipy.stats import gaussian_kde
from scipy.integrate import trapezoid
#from scipy.integrate import trapz
✓ 0.0s Python
```

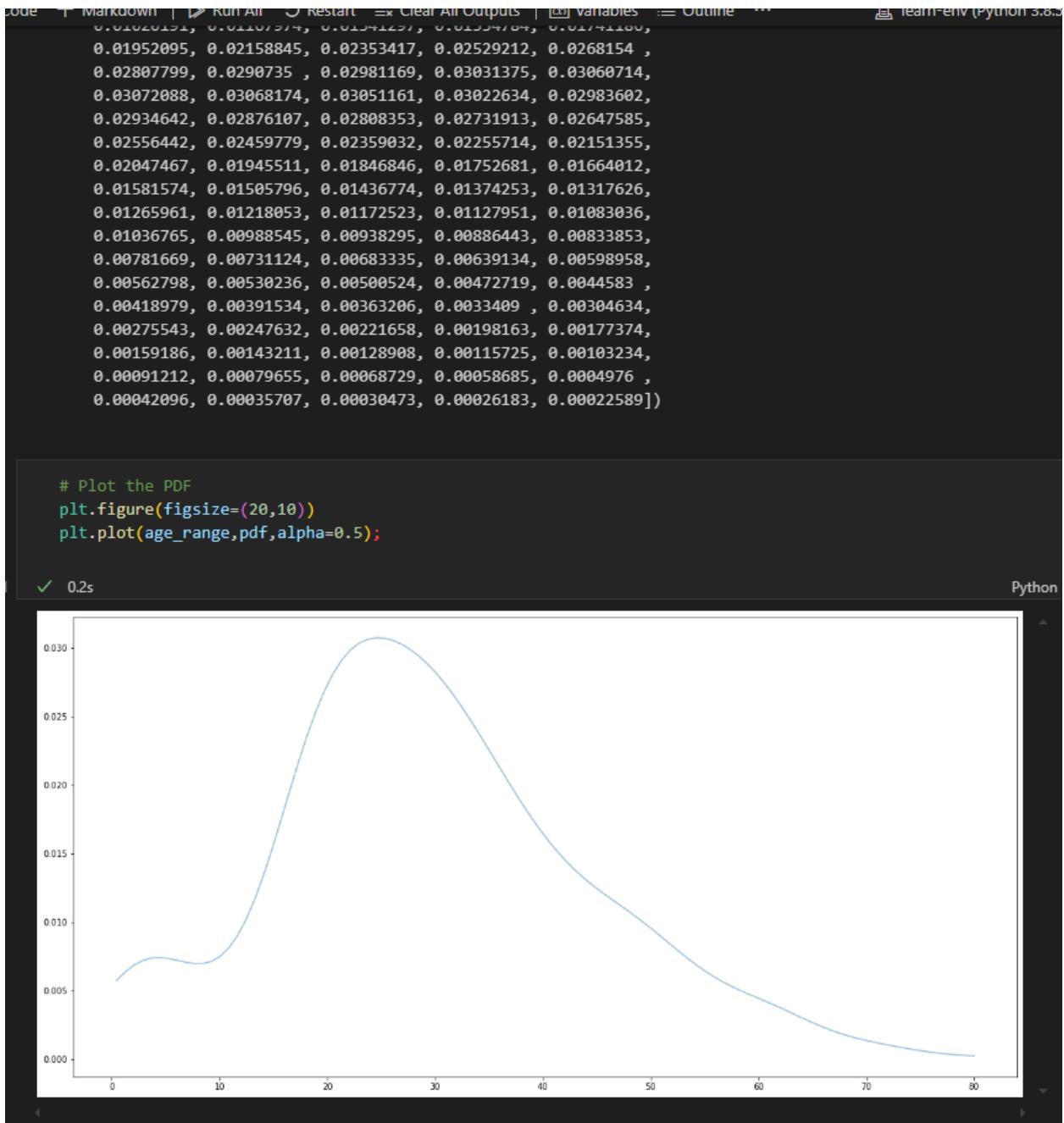
```
# choose a column from the dataset
df_age = df['Age'].dropna()
✓ 0.0s Python
```

```
# generate the estimator
kde = gaussian_kde(df_age)
✓ 0.0s Python
```

```
# Create a range of ages for the PDF
age_range = np.linspace(df_age.min(), df_age.max(), 100)
# generate the pdf values
pdf = kde(age_range)
pdf
```

```
✓ 0.0s Python
```

```
array([0.00570943, 0.00633591, 0.00683045, 0.00717142, 0.00735571,
       0.007399 , 0.00733342, 0.00720349, 0.00706162, 0.00696412,
       0.00696798, 0.00712832, 0.00749574, 0.00811294, 0.00901047,
       0.01020191, 0.01167974, 0.01341297, 0.01534784, 0.01741186,
       0.01952095, 0.02158845, 0.02353417, 0.02529212, 0.0268154 ,
       0.02807799, 0.0290735 , 0.02981169, 0.03031375, 0.03060714,
       0.03072088, 0.03068174, 0.03051161, 0.03022634, 0.02983602,
       0.02934642, 0.02876107, 0.02808353, 0.02731913, 0.02647585,
       0.02556442, 0.02459779, 0.02359032, 0.02255714, 0.02151355,
       0.02047467, 0.01945511, 0.01846846, 0.01752681, 0.01664012,
       0.01581574, 0.01505796, 0.01436774, 0.01374253, 0.01317626,
       0.01265961, 0.01218053, 0.01172523, 0.01127951, 0.01083036,
       0.01036765, 0.00988545, 0.00938295, 0.00886443, 0.00833853,
       0.00781669, 0.00731124, 0.00683335, 0.00639134, 0.00598958,
```



**44. CUMULATIVE DISTRIBUTION FUNCTION (CDF)-** provides a way to describe the probability distribution of a random variable

Gives the probability that variable X is less than or equal to a certain possible value of x

#### Properties of CDF

- ✓ Non-decreasing – CDF is always non-decreasing; as x increases, f(x) either increases or stays the same
- ✓ Range- CDF takes values in the range [0,1]
- ✓ Right-continuous- CDF approaches its limit from the right

#### EXAMPLE 1: CONT from titanic dataset

1. sort data

# Calculating CDF

```
# Calculate CDF continuos data
sorted_data = np.sort(df_age)
sorted_data

[37]
...
array([ 0.42,  0.67,  0.75,  0.75,  0.83,  0.83,  0.92,  1. ,  1. ,
       1. ,  1. ,  1. ,  1. ,  2. ,  2. ,  2. ,  2. ,  2. ,  2. ,
       2. ,  2. ,  2. ,  2. ,  2. ,  2. ,  3. ,  3. ,  3. ,
       3. ,  3. ,  3. ,  4. ,  4. ,  4. ,  4. ,  4. ,  4. ,
       4. ,  4. ,  4. ,  4. ,  5. ,  5. ,  5. ,  5. ,  5. ,
       5. ,  6. ,  6. ,  7. ,  7. ,  7. ,  8. ,  8. ,  8. ,
       8. ,  9. ,  9. ,  9. ,  9. ,  9. ,  9. ,  9. ,  9. ,
       9. ,  9. ,  10. ,  10. ,  10. ,  11. ,  11. ,  11. ,
       11. ,  11. ,  11. ,  12. ,  12. ,  13. ,  13. ,
       13. ,  13. ,  14. ,  14. ,  14. ,  14.5 ,  15. ,
       15. ,  15. ,  15. ,  16. ,  16. ,  16. ,  16. ,
       16. ,  16. ,  16. ,  16. ,  16. ,  16. ,  16. ,
       16. ,  16. ,  16. ,  16. ,  16. ,  16. ,  16. ,
       16. ,  16. ,  17. ,  17. ,  17. ,  17. ,  17. ,
       17. ,  17. ,  17. ,  17. ,  17. ,  18. ,  18. ,
       18. ,  18. ,  18. ,  18. ,  18. ,  18. ,  18. ,
       18. ,  18. ,  18. ,  18. ,  18. ,  18. ,  18. ,
       18. ,  18. ,  18. ,  18. ,  19. ,  19. ,  19. ,
       19. ,  19. ,  19. ,  19. ,  19. ,  19. ,  19. ,
       19. ,  19. ,  19. ,  19. ,  19. ,  19. ,  19. ,
       19. ,  19. ,  20. ,  20. ,  20. ,  20. ,  20. ,
       20. ,  20. ,  20. ,  20. ,  20. ,  20. ,  20. ,
       21. ,  21. ,  21. ,  21. ,  21. ,  21. ,  21. ,
       21. ,  21. ,  21. ,  21. ,  21. ,  21. ,  21. ,
       21. ,  21. ,  21. ,  21. ,  21. ,  22. ,  22. ,
       22. ,  22. ,  22. ,  22. ,  22. ,  22. ,  22. ,
       22. ,  22. ,  22. ,  22. ,  22. ,  22. ,  22. ,
       ...
       56. ,  56. ,  56. ,  56. ,  57. ,  57. ,  58. ,
       58. ,  58. ,  59. ,  59. ,  60. ,  60. ,  60. ,
       61. ,  61. ,  62. ,  62. ,  62. ,  62. ,  63. ,
       64. ,  65. ,  65. ,  65. ,  66. ,  70. ,  70. ,
       71. ,  74. ,  80. ])
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

2. Do a cumulative sum and calculate cdf

```

# cumsum
cum_sum = np.arange(1,len(sorted_data)+1) #add +1 since arrange doesn't count the last item
#generate cumsum from the length of our data
cum_sum| Python
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
       14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
       53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
       66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
       79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
       92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
       105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
       118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
       131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
       144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
       157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
       170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
       183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
       196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
       209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
       222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
       235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
       248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
       261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273,
       274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286,
       287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299,
       300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312,
       313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325,
       ...
       651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663,
       664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676,
       677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689,
       690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
       703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

---

```

# calculate cdf
cdf = cum_sum/len(sorted_data)
cdf| Python
array([0.00140056, 0.00280112, 0.00420168, 0.00560224, 0.0070028 ,
       0.00840336, 0.00980392, 0.01120448, 0.01260504, 0.0140056 ...]
```

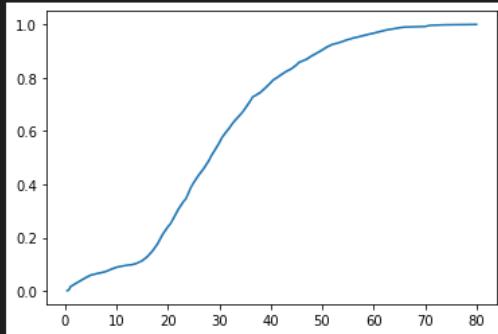
### 3.Plot

```
# plot
import seaborn as sns
sns.lineplot(x=sorted_data,y=cdf)
```

49]

Python

.. <AxesSubplot:>



```
# example calculations
u_index = np.searchsorted(sorted_data,80) #search index of 80 in our sorted dataset
u_index
```

51]

Python

.. 713

```
cdf[713] #prob of getting 80 and below
```

53]

Python

.. 1.0

```
u_index = np.searchsorted(sorted_data,1) #search index of 1 in our sorted dataset  
u_index
```

[54]

... 7

```
cdf[7]
```

[56]

... 0.011204481792717087

```
u_index = np.searchsorted(sorted_data,0.87)  
u_index
```

[82]

... 6

```
np.searchsorted(sorted_data,79)
```

[83]

... 713

```
#position of lowerbound  
u_index = np.searchsorted(sorted_data,80)  
l_index = np.searchsorted(sorted_data,79)  
cdf[u_index- l_index]
```

[63]

... 0.0014005602240896359

# An example with discrete Data

```
# select a column
```

```
df_class = df["Pclass"]
```

[64]

Python

```
# Count frequencies of each class
```

```
count = df_class.value_counts().sort_index()  
count
```

[66]

Python

```
... 1 201  
2 172  
3 469  
? 49  
Name: Pclass, dtype: int64
```

```
# get the sum
```

```
cum_sum2 = count.cumsum()  
cum_sum2
```

[68]

Python

```
... 1 201  
2 373  
3 842  
? 891  
Name: Pclass, dtype: int64
```

```
# calculate the values
```

```
cdf2 = cum_sum2/len(df_class)  
cdf2
```

[69]

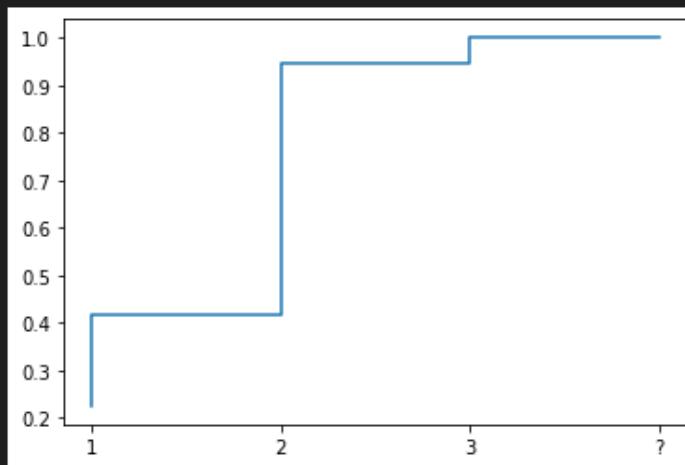
Python

```
... 1 0.225589  
2 0.418631  
3 0.945006  
? 1.000000  
Name: Pclass, dtype: float64
```

```
?      1.000000  
Name: Pclass, dtype: float64
```

```
[72] # plot using step ploter  
plt.step(y=cdf2.values,x=cdf2.index)
```

```
[72] ... [ <matplotlib.lines.Line2D at 0x1fb7e8d5e0>]
```



```
cdf2['1']
```

```
[79] ... 0.2255892255892256
```

```
cdf2['?']
```

```
[80] ... 1.0
```

```
[78] # Example calc  
cdf2['?'] - cdf2['1']
```

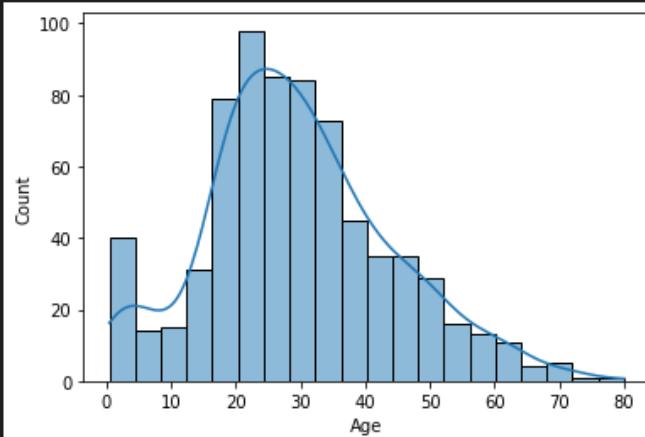
```
[78] ... 0.7744107744107744
```

In most cases when only interested in the distribution kde plot from sns is enough e.g

```
# check the distribution of the using histplot  
sns.histplot(df_age,kde=True)
```

[86]

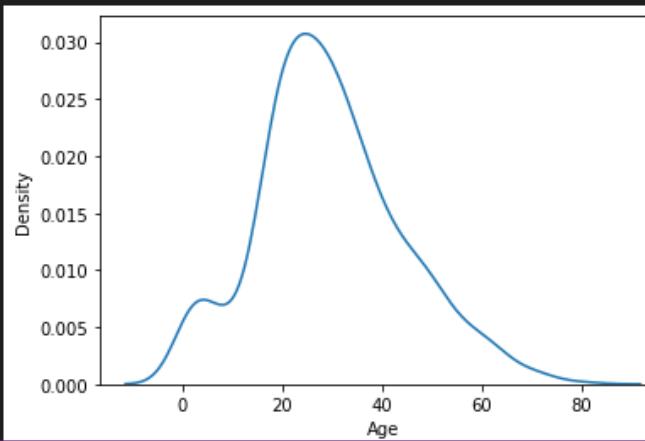
```
... <AxesSubplot:xlabel='Age', ylabel='Count'>
```



```
# check using kde plot  
sns.kdeplot(df_age)
```

[85]

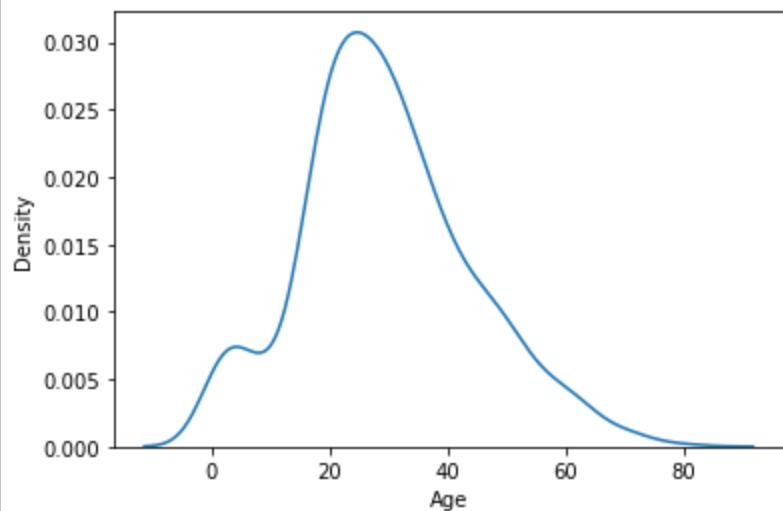
```
... <AxesSubplot:xlabel='Age', ylabel='Density'>
```



```
sns.kdeplot(df_age)
```

```
90]
```

```
.. <AxesSubplot:xlabel='Age', ylabel='Density'>
```

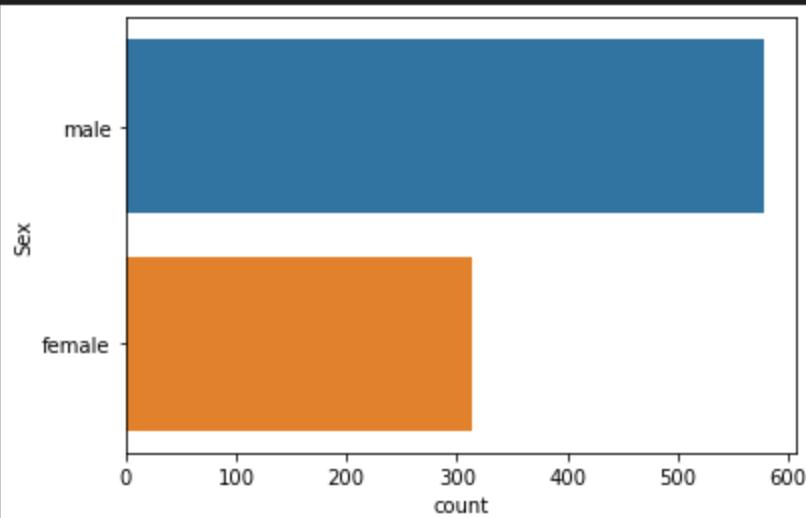


```
#male and female
```

```
#Discrete data
```

```
sns.countplot(y=df_gender);
```

```
93]
```



## Order Totals CDF & PDF

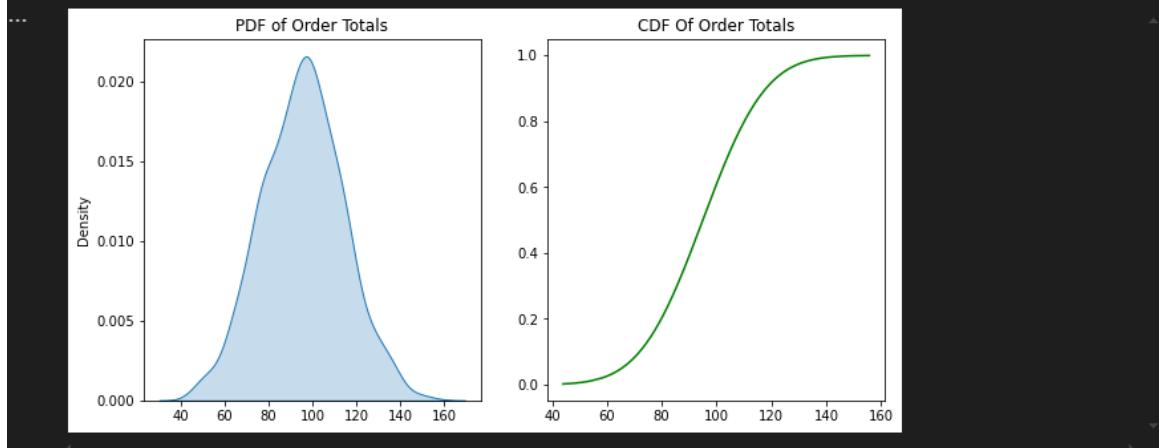
Imagine that an online clothing business gets orders with totals that are normally distributed with an average of \\$95 and a standard deviation of \\$18. **Graph the PDF and CDF** for the orders for this business.

```
orders = sorted(stats.norm.rvs(loc=95,scale=18,size=1000))
orders_cdf = stats.norm.cdf(orders,loc=95,scale=18)
fig,(ax1,ax2) = plt.subplots(1,2,figsize=(10,5))
sns.kdeplot(orders,ax=ax1,shade=True)
ax1.set_title('PDF of Order Totals')

ax2.plot(orders,orders_cdf,color='g')
ax2.set_title('CDF Of Order Totals');
```

[89]

Python



## Order Totals Observations

After graphing, **write 1-3 observations** about the distributions of order totals based on these graphs.

### Your Observations

Observation 1: max value 155

Observation 2: min 44

Observation 3: values are infinite but not zero

```
orders[0]
```

[2]

Python

## HISTOGRAM AND CDF

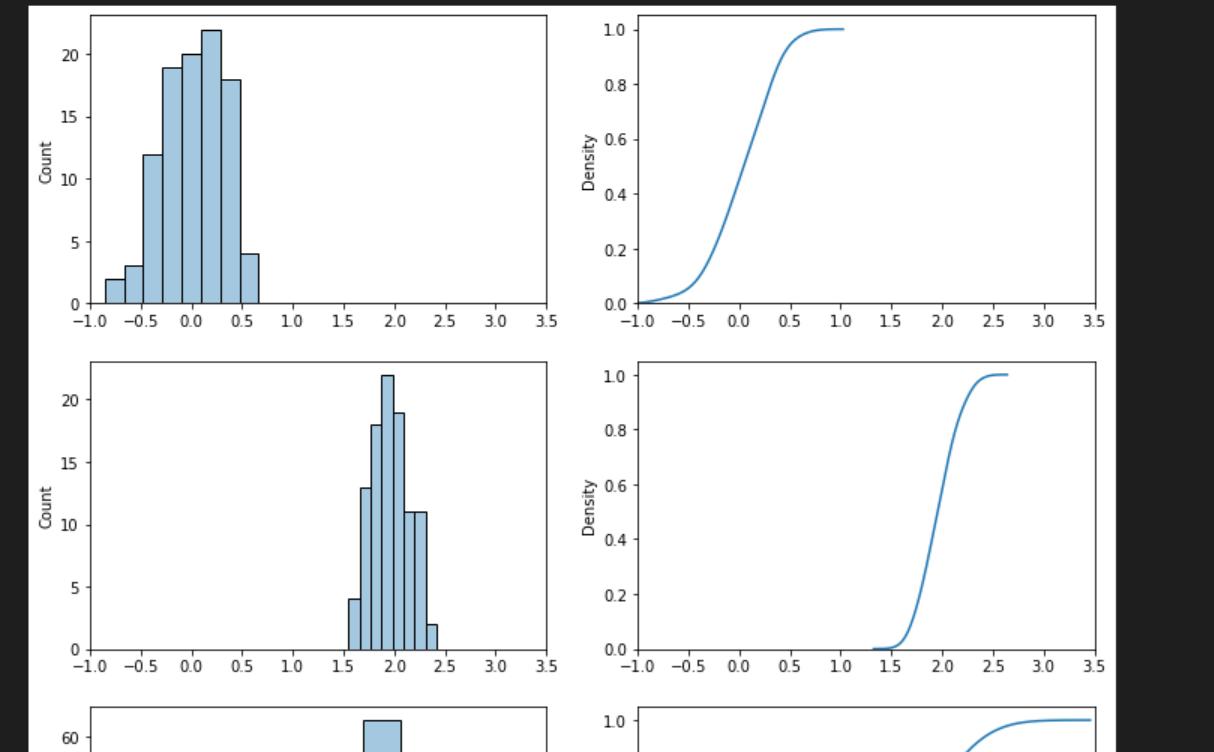
```

f,(ax0,ax1,ax2) = plt.subplots(3,2, figsize=(12,12))
# Histogram
ax=sns.histplot(x=norm_dist0, alpha=0.4, ax=ax0[0])
ax.set_xlim(-1,3.5)
ax=sns.histplot(x=norm_dist1, alpha=0.4, ax=ax1[0])
ax.set_xlim(-1,3.5)
ax=sns.histplot(x=two_dist, alpha=0.4, ax=ax2[0])
ax.set_xlim([-1,3.5])

# CDF
ax=sns.kdeplot(x=norm_dist0, alpha=0.4, ax=ax0[1], cumulative=True)
ax.set_xlim(-1,3.5)
ax=sns.kdeplot(x=norm_dist1, alpha=0.4, ax=ax1[1], cumulative=True)
ax.set_xlim(-1,3.5)
ax=sns.kdeplot(x=two_dist, alpha=0.4, ax=ax2[1], cumulative=True)
ax.set_xlim([-1,3.5])

```

(-1.0, 3.5)



**45. Normal Distributions/Gaussian Distribution** - is defined by two parameters: the mean ( $\mu$ ) and the standard deviation ( $\sigma$ ). The mean determines the center of the distribution, while the standard deviation measures the dispersion or spread of the data points around the mean. The probability density function

- The curve is symmetric about the mean, meaning that approximately 68% of the data lies within one standard deviation ( $\sigma$ ) from the mean, about 95% within two standard deviations, and about 99.7% within three standard deviations. This is often referred to as the empirical rule or the 68-95-99.7 rule.

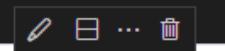
#### Importance in Statistics

- The normal distribution holds a central place in statistics for several reasons:

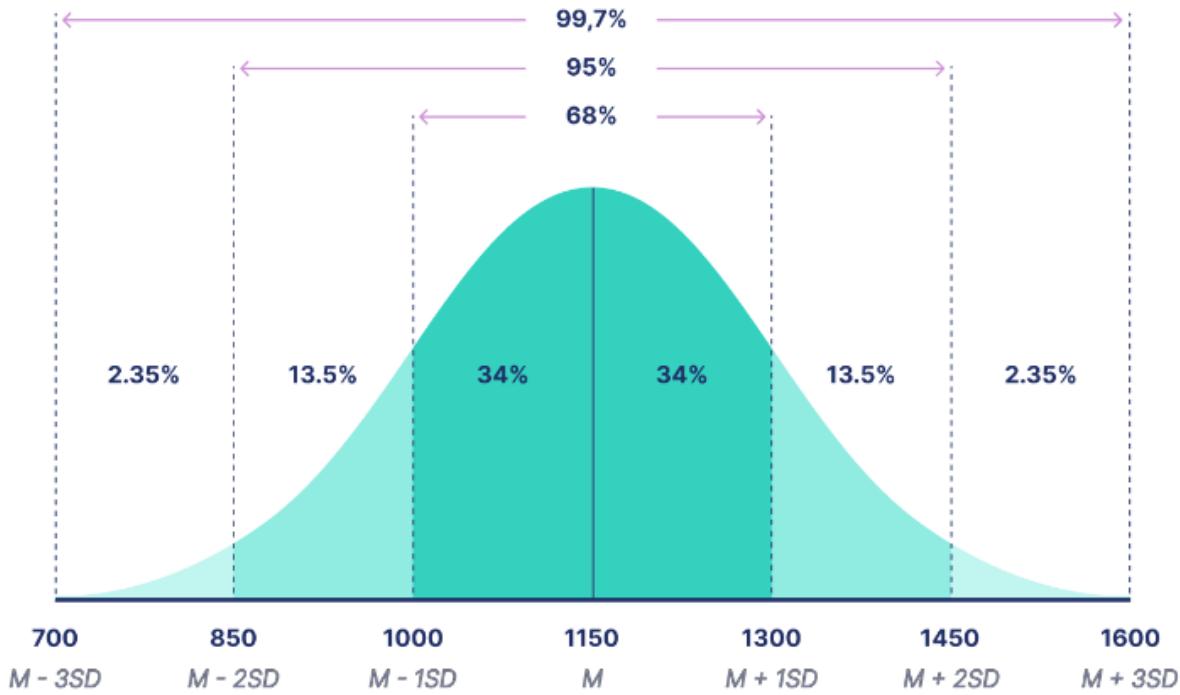
- Central Limit Theorem: One of the most significant reasons for the prominence of the normal distribution is the Central Limit Theorem (CLT). The CLT states that the distribution of the sample means will tend to be normally distributed, regardless of the original distribution of the data, as long as the sample size is sufficiently large. This makes the normal distribution a key tool for inferential statistics.

- Statistical Inference: Many statistical tests and methods, including t-tests, ANOVA, and regression analysis, assume that the underlying data is normally distributed. This assumption allows for the application of parametric tests, which can provide more powerful results than non-parametric alternatives.

- Natural Phenomena: Numerous real-world measurements—such as heights, weights, test scores, and measurement errors—tend to follow a normal distribution. This natural occurrence in various fields, including psychology, biology, and economics, makes the normal distribution a useful model for analyzing data



### Using the empirical rule in a normal distribution



### X-Stics of Normal Distribution

**1. Symmetry-** Bell-Shaped Curve: The normal distribution is symmetric about the mean ( $\mu$ ). This means that the left side of the curve is a mirror image of the right side. Mean = Median = Mode: In a normal distribution, the mean, median, and mode are all located at the center of the distribution.

**2. Defined by Mean and Standard Deviation-** Mean ( $\mu$ ): The average value around which the data points are distributed. It determines the center of the distribution.

Standard Deviation ( $\sigma$ ): This measures the spread or dispersion of the data. A smaller standard deviation results in a steeper curve, while a larger standard deviation produces a flatter curve.

**3. Empirical Rule (68-95-99.7 Rule)-** 68% of Data within 1 Standard Deviation: Approximately 68% of the data falls within one standard deviation ( $\sigma$ ) of the mean ( $\mu$ ).

95% within 2 Standard Deviations: About 95% of the data lies within two standard deviations.

99.7% within 3 Standard Deviations: Nearly all (99.7%) of the data is within three standard deviations from the mean.

The screenshot shows a Jupyter Notebook interface with several code cells and a resulting figure. The code cells are numbered [106], [107], [108], and [109]. The first cell contains a header: "99.7% within 3 Standard Deviations: Nearly all (99.7%) of the data is within three standard deviations from the mean." The second cell imports numpy and seaborn, sets a random seed, and defines mu and sigma. The third cell generates data using np.random.normal. The fourth cell uses sns.kdeplot to create a density plot. The resulting figure is a bell-shaped curve on a coordinate system where the x-axis ranges from -60 to 80 and the y-axis (Density) ranges from 0.000 to 0.025. The peak of the curve is at approximately x=10.

```
[106]: 99.7% within 3 Standard Deviations: Nearly all (99.7%) of the data is within three standard deviations from the mean.

[107]: import numpy as np
import seaborn as sns

# set seed for reproducibility
np.random.seed(42)

[108]: # define mu & sigma
mu, sigma = 10, 15

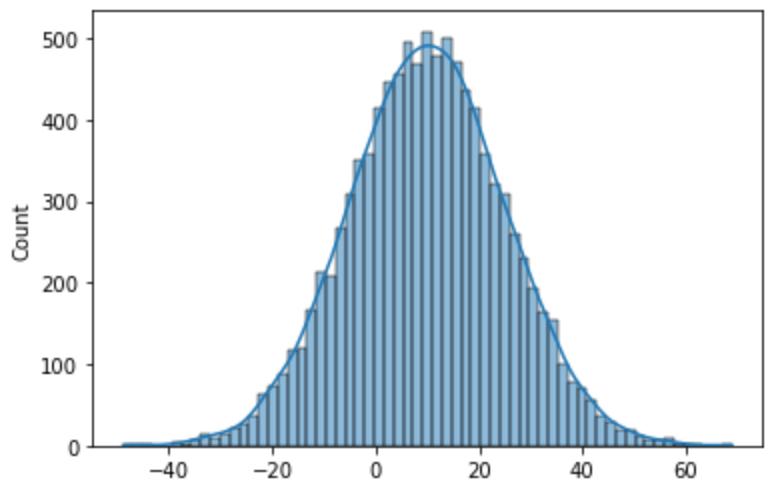
[109]: # generate data
s = np.random.normal(mu, sigma, 10000)

sns.kdeplot(s);
```

Density

The figure is a density plot titled 'Density'. The x-axis is labeled from -60 to 80 with major ticks every 20 units. The y-axis is labeled from 0.000 to 0.025 with major ticks every 0.005 units. A single blue bell-shaped curve is plotted, centered at x=10, reaching its maximum density of about 0.025 at x=10.

```
# plot the data
sns.histplot(s,kde=True);
```



```
# cheking the mean mode median
from scipy import stats
print(s.mean())
print(np.median(s))
stats.mode(s)
```

```
9.967960249473606
```

```
9.961075363106836
```

```
ModeResult(mode=array([-48.83600377]), count=array([1]))
```

**46. Standard Normal Distribution-** Special case of the normal distribution that has been standardized to simplify analysis and interpretation. Its crucial in statistics due to its properties and its role in various statistical methods.

Special case of normal distribution with a mean of 0 and std of 1

The standard normal distribution is defined as a normal distribution with:

Mean ( $\mu$ ) = 0

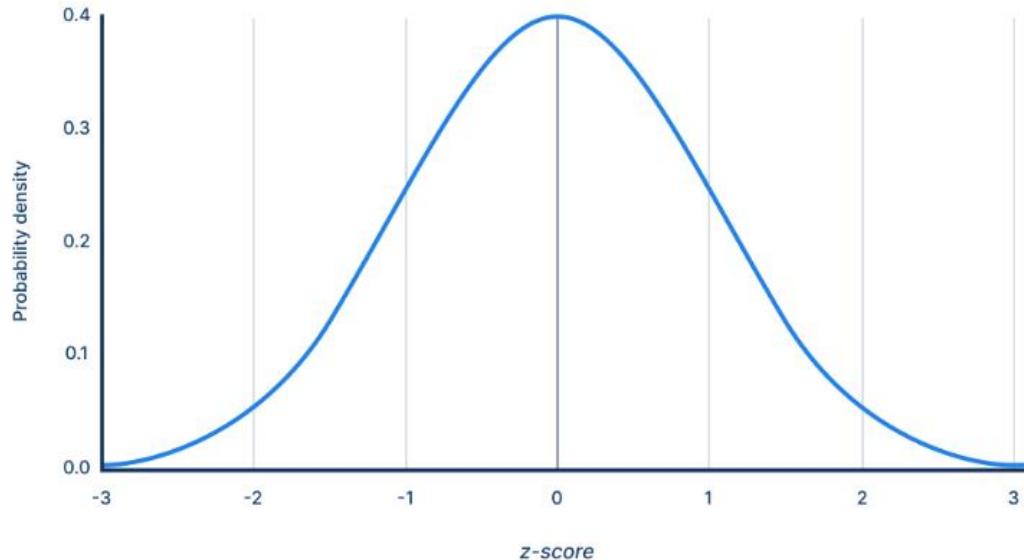
Standard Deviation ( $\sigma$ ) = 1

+ Code

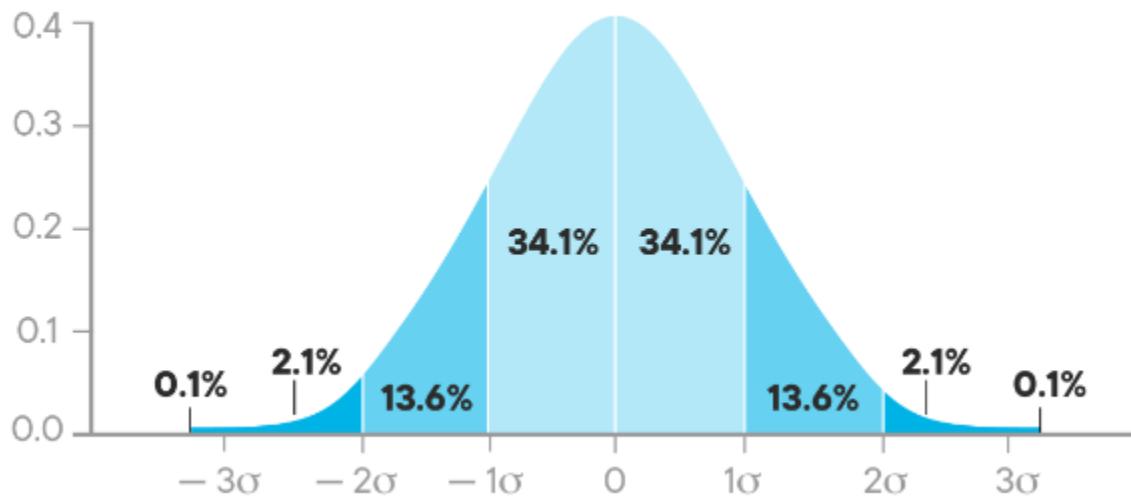
+ Markdown



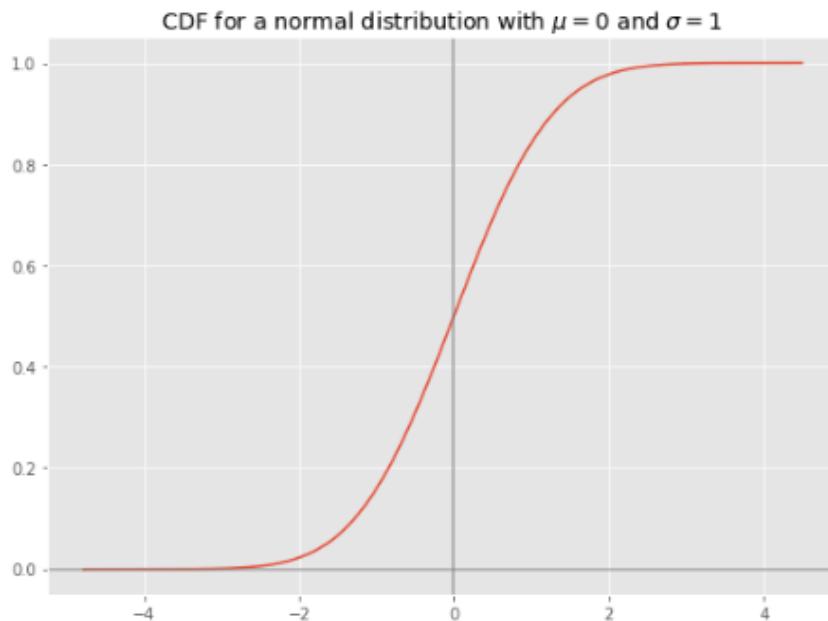
## Standard normal distribution



mean of 0 and a standard deviation of 1.



Plotting a continuous cumulative distribution function for the standard normal distribution, the CDF would look like this:



Thinking back to the standard deviation rule for normal distributions:

- 68 of the area lies in the interval of 1 standard deviation from the mean, or mathematically speaking, 68 is in the interval  $[\mu - \sigma, \mu + \sigma]$
- 95 of the area lies in the interval of 2 standard deviations from the mean, or mathematically speaking, 95 is in the interval  $[(\mu - 2\sigma), (\mu + 2\sigma)]$
- 99 of the area lies in the interval of 3 standard deviations from the mean, or mathematically speaking, 99 is in the interval  $[(\mu - 3\sigma), (\mu + 3\sigma)]$

With a  $\mu = 0$  and  $\sigma = 1$ , this means that for the standard normal distribution:

- 68 of the area lies between -1 and 1.
- 95 of the area lies between -2 and 2.
- 99 of the area lies between -3 and 3.

This simplicity makes a standard normal distribution very desirable to work with. The exciting news is that you can very easily transform any normal distribution to a standard normal distribution!

---

a) **Standard score/z score** – Statistical measurement that describes a value's relation to the mean of a group of values. It indicates how many standard deviations a data point is from the mean of the dataset  
$$z = (X - \mu) / \sigma$$

Where:

$X$  is the value in question.

$\mu$  is the mean of the dataset.

$\sigma$  is the standard deviation of the dataset.

## An example

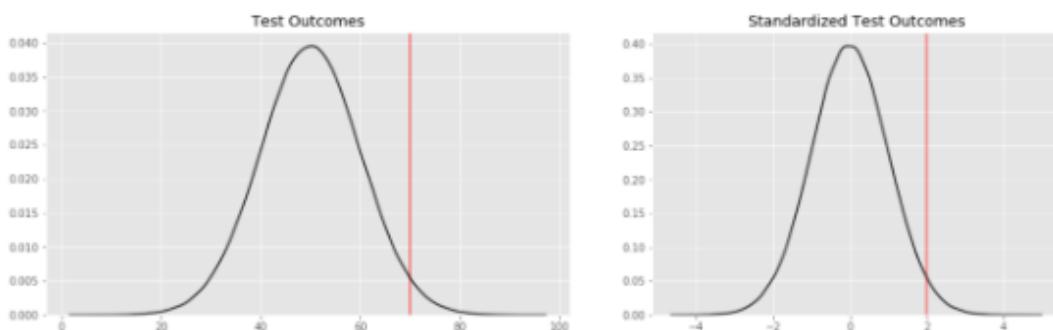
Imagine some test results follow a normal distribution with a mean score of 50 and a standard deviation of 10. One of the students scored a 70 on the test.

Using this information and  $z$ -scores makes it easy to tell how she performed in terms of standard deviations from the mean. Converting a test score of 70 to a  $z$ -score, an  $x$  of 70 would be, in this case:

$$z = \frac{70 - 50}{10} = 2$$

By transforming the test result of 70 to a  $z$ -score of 2, we now know that the student's original score was 2 standard deviations above the mean score. Note that the  $z$  distribution will only be a normal distribution if the original distribution of  $x$  was normal.

In summary, calculating the  $z$ -score gives us quick and easy access to understanding how **extreme** a certain result is. Looking at the original distribution ( $\mu = 50, \sigma = 10$ ) and the standard normal distribution ( $\mu = 0, \sigma = 1$ ) while highlighting  $x = 70$  and  $z = 2$  gives the following result:



Visually, the idea is that the area under the curve, left and right from the vertical red line, are identical in the left plot and the right plot!

Thinking along these lines, you can also convert a  $z$ -score back to an original score  $x$  by using the same formula as:

$$x = \mu + z\sigma$$

For the above example, this would work out as:

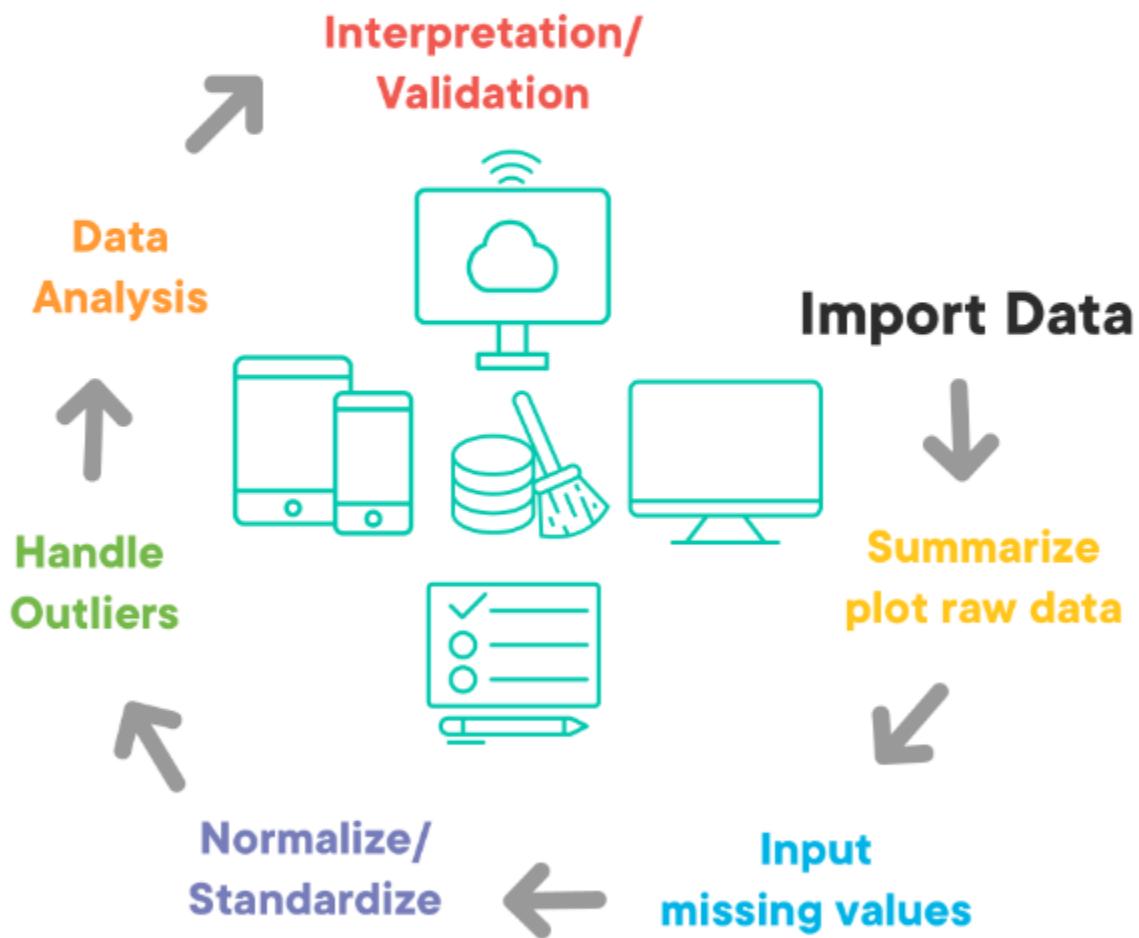
$$x = 50 + 2 * 10 = 70$$

**b. Data Standardization** – common data preprocessing skill, which is used to compare a number of observations belonging to the different normal distributions which may have distinct means and standard deviations

## Data Standardization

Data standardization is a common data preprocessing skill, which is used to compare a number of observations belonging to different normal distributions which may have distinct means and standard deviations.

Standardization applies a  $z$ -score calculation, as shown above, on each element of the distribution. The output of this process is a  **$z$ -distribution** or a **standard normal distribution**.

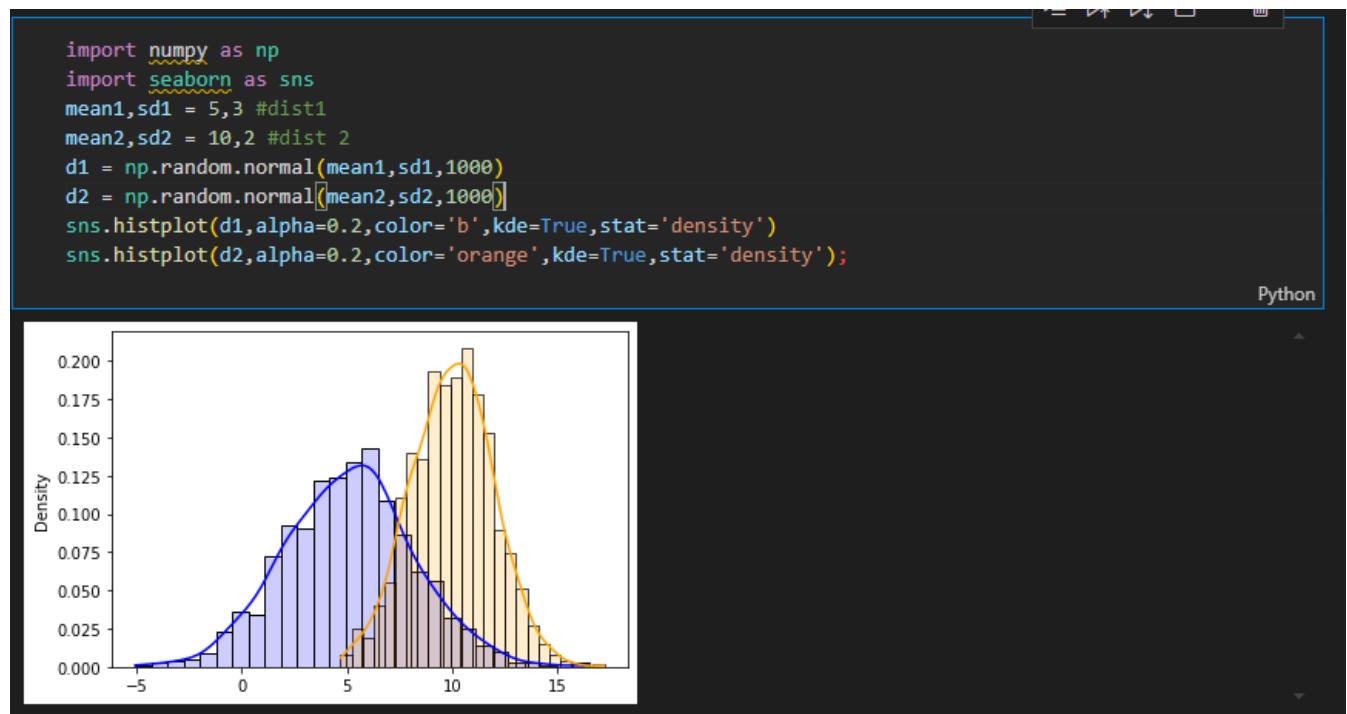


Let's look at a quick example. First, we'll randomly generate two normal distributions with different means and standard deviations. Let's generate 1000 observations for each. Next, we'll use `seaborn` to plot the results, where the output may look different than this based on your machine.

```

import numpy as np
import seaborn as sns
mean1,sd1 = 5,3 #dist1
mean2,sd2 = 10,2 #dist 2
d1 = np.random.normal(mean1,sd1,1000)
d2 = np.random.normal(mean2,sd2,1000)
sns.histplot(d1,alpha=0.2,color='b',kde=True,stat='density')
sns.histplot(d2,alpha=0.2,color='orange',kde=True,stat='density');

```



You can see that these distributions differ from each other and are not directly comparable.

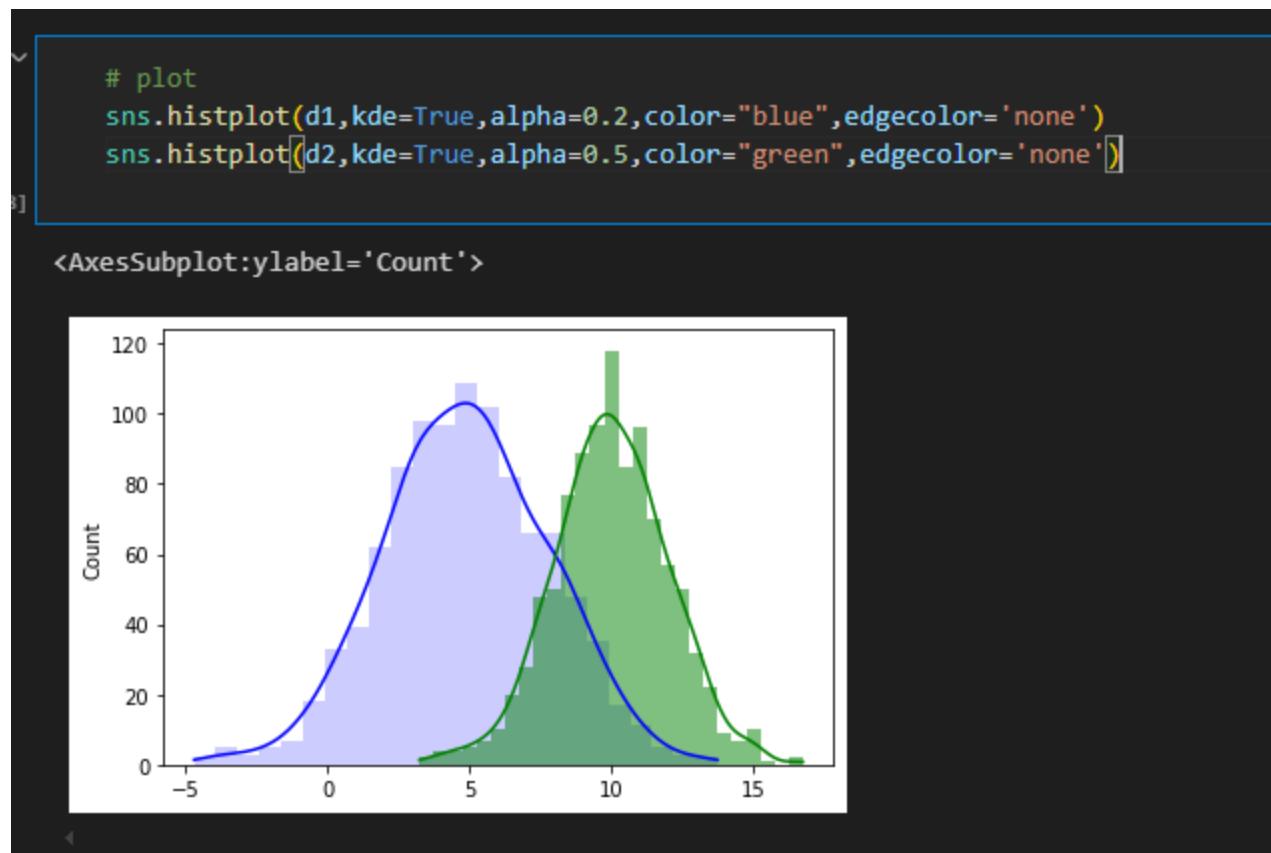
For a number of machine learning algorithms and data visualization techniques, it is important that the effect of the scale of the data is removed before you start thinking about building your model. Standardization allows for this by converting the distributions into a  $z$ -distribution, bringing them to a common scale (with  $\mu = 0, \sigma = 1$ ). Let's standardize the above distributions and look at the effect.

### TEACHERS CODE using kde instead of stat

```

sns.histplot(d1,kde=True,alpha=0.2,color="blue",edgecolor='none')
sns.histplot(d2,kde=True,alpha=0.5,color="green",edgecolor='none')

```



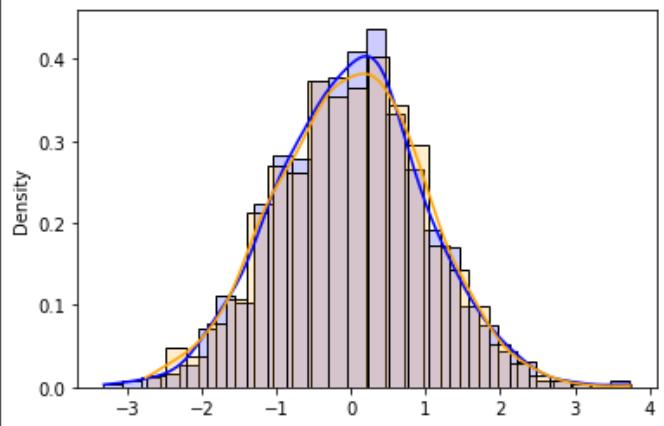
## AFTER STANDARDIZATION

```
# Standardizing and visualizing distributions

sns.histplot([(x - d1.mean())/d1.std() for x in d1],alpha=0.2,color='b',kde=True,stat='density');
sns.histplot([(x - d2.mean())/d2.std() for x in d2],alpha=0.2,color='orange',kde=True,stat='density');
```

```
# Standardizing and visualizing distributions  
|  
sns.histplot([(x - d1.mean())/d1.std() for x in d1],alpha=0.2,color='b',kde=True,stat='density');  
sns.histplot([(x - d2.mean())/d2.std() for x in d2],alpha=0.2,color='orange',kde=True,stat='density');
```

Pyt



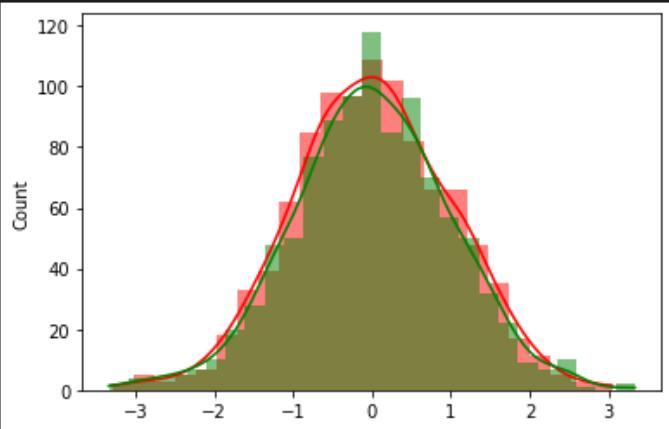
You see that both distributions are directly comparable on a common standard scale. As mentioned earlier, this will come in handy with analytics experiments while training machine learning algorithms.

### Teachers code without stat

## After Stardadization

```
# stardadization by calculating the z-score  
  
d1_s = [(x - d1.mean())/d1.std() for x in d1]  
d2_s = [(x - d2.mean())/d2.std() for x in d2]
```

```
119]  
  
# plot  
sns.histplot(d1_s,kde=True,color="red",edgecolor="none")  
sns.histplot(d2_s,kde=True,color="green",edgecolor="none")  
120]  
.. <AxesSubplot:ylabel='Count'>
```



```
121]  
.. np.mean(d1_s)  
.. -3.375077994860476e-17
```

47. The **z-score** can be used to understand how extreme a certain result is

48. **Skewness** and **kurtosis** can be used to measure how different a given distribution is from a normal distribution

### Examples of z scores

# Height Empirical Rule

Use the empirical rule and the information above to determine about how many people are between **62 inches and 74 inches**.

The empirical rule (also known as the 68-95-99.7 rule) is a guideline for normal distributions, stating:

- About 68% of the data falls within 1 standard deviation of the mean.
- About 95% falls within 2 standard deviations.
- About 99.7% falls within 3 standard deviations.

```
(62-66)/4 #62 is 1 standard deviation below the mean  
(74-66)/4 #74 is 2 standard deviations above the mean  
68+13.5 #81.5% of the people i.e from -1 to 2
```

[13] ✓ 0.0s

Python

... 81.5

# Height Percentile

Assuming the above distribution of people's heights in the United States is approximately normal, what percent of people have a height less than **75 inches**?

```
(75 - 66)/4  
#Now that we have a z-score of 2.25, we can use a z-table or standard normal distribution to find the  
#percentage of people below this z-score. A z-score of 2.25 corresponds to a cumulative probability  
# of approximately 0.9878, or 98.78%. from the z-score table  
  
#Thus, about 98.78% of people have a height less than 75 inches.
```

[17] ✓ 0.0s

... 2.25

Python

stats.norm.cdf(2.25)

```
stats.norm.cdf(2.25)
```

✓ 0.0s

0.9877755273449553

## Bonus

Assuming the above distribution of people's heights in the United States is approximately normal, what range of heights contain the **middle 50% of values**, also known as the *interquartile range* (IQR)?

```
#Middle 50 % is IQR which is 75th -25th percentile
#get z_score for 25%
from scipy.stats import norm
#norm.ppf(p) gives you the z-score where the cumulative probability is p.
z_25th = norm.ppf(0.25) #z score of approximately -0.674
z_75th = norm.ppf(0.75) #z score of approximately 0.674
z_75th # 0.674
[18] ✓ 0.0s
... 0.6744897501960817
Python
```

```
#use the z-score formula to find the height
#z= (X-μ)/σ
#X = z*σ+μ
#For the 25th percentile (Q1):
x1=z_25th*4+66
x2= z_75th*4+66
print(f'25th percentile(Q1): {x1}')
#For the 75th percentile (Q3):
print(f'75th percentile(Q1): {x2}')
#IQR
print(f'IQR {x2-x1}')
#Conclusion:
...
The middle 50% of people's heights in the United States falls between 63.3 inches and 68.7 inches,
| with an interquartile range (IQR) of 5.4 inches.''
```

```
[ ] Python
... 25th percentile(Q1): 63.30204099921567
75th percentile(Q1): 68.69795900078432
IQR 5.395918001568653
```

49. **Inferential statistics** – drawing conclusions beyond just describing statistics of your sample data. Allows us to make claims abt data that we don't have access to.
50. **Central Limit theorem** – states that given a sufficiently large sample size, the sampling distribution of the sample mean will be approximately normally distributed regardless of the original population's distribution.  
allows us to treat non-normal distributions as normal distributions and provides ways for us to estimate parameters about a population  
**CENTRAL LIMIT THEOREM**
  - ✓ Plot the data to check **distribution** eg sns.kdeplot(bimodal\_data) OR sns.distplot()

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Set the random seed for reproducibility
np.random.seed(42)
```

[105] ✓ 0.0s

```
# Generate two different normal distributions

data1 = np.random.normal(loc=5, scale=1, size=500)
data2 = np.random.normal(loc=10, scale=1, size=500)

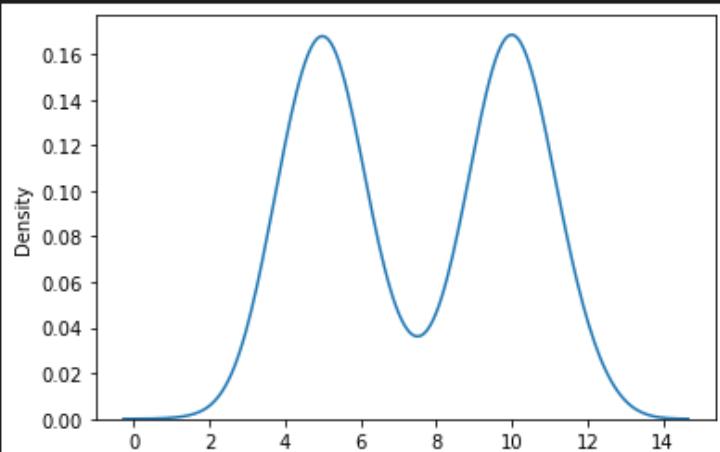
# Combine the two datasets
bimodal_data = np.concatenate([data1, data2])
```

[106] ✓ 0.0s

```
sns.kdeplot(bimodal_data)
```

[107] ✓ 0.2s

... <AxesSubplot:ylabel='Density'>

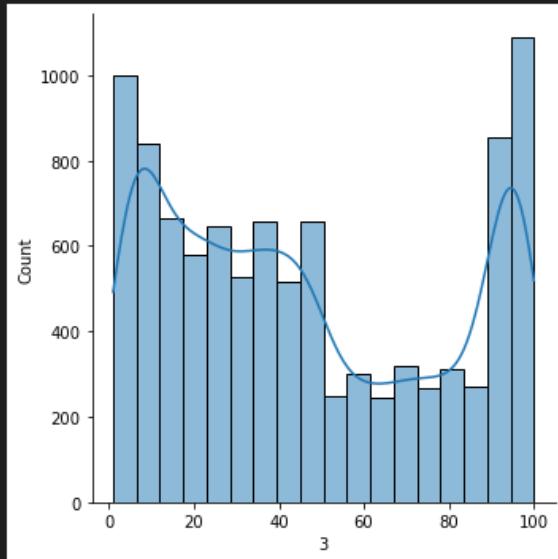


In the cell below, use `seaborn`'s `distplot` method to visualize a histogram of the distribution overlaid with a probability density curve.

```
▶   sns.distplot(data,kde=True)
```

[29] ✓ 0.2s

```
... <seaborn.axisgrid.FacetGrid at 0x25ad21eddf0>
```



As expected, this dataset is not normally distributed.

For a more formal way to check if a dataset is normally distributed or not, we can make use of a statistical test. There are many different statistical tests that can be used to check for normality, but we'll keep it simple and make use of the `normaltest()` function from `scipy.stats`, which we imported as `st` -- see the documentation for more information. If you have questions about how to use this method,

In the cell below, use `normaltest()` to check if the dataset is normally distributed.

- Also use a **statistical test** to check normality

```
stat,p_value = st.normaltest(data)
print(stat)
print(p_value)

alpha = 0.05
#interpret the result
if p_value > alpha:
    print('Fail to reject the null hypothesis(data is likely normally distributed)')
else:
    print('Reject the null hypothesis(data is likely not normally distributed)')
```

```

#my code
stat,p_value = st.normaltest(data)
alpha = 0.05
#interpret the result
if p_value > alpha:
    print('Fail to reject the null hypothesis(data is likely normally distributed)')
else:
    print('Reject the null hypothesis(data is likely not normally distributed)')

✓ 0.0s
Python
43432.811126532004
0.0
Reject the null hypothesis(data is likely not normally distributed)

#Teacher code
st.normaltest(data)
✓ 0.0s
Python
NormaltestResult(statistic=43432.811126532004, pvalue=0.0)

```

## Explanation:

`stats.normaltest()`: This function performs the D'Agostino and Pearson's test for normality, which combines tests for skewness and kurtosis.

- Null Hypothesis: The data follows a normal distribution.
- Alternative Hypothesis: The data does not follow a normal distribution. The test returns:

`stat`: The test statistic.

`p_value`: The p-value, which indicates the likelihood that the data is normally distributed.

## Interpretation:

- If the p-value is greater than 0.05 (commonly chosen significance level), we fail to reject the null hypothesis, meaning the data is likely normally distributed.
- If the p-value is less than 0.05, we reject the null hypothesis, meaning the data is likely not

Since our dataset is non-normal, that means we'll need to use the \*\*Central Limit Theorem\*\*.

- ✓ We sample our data

```

sample_30 = np.random.choice(bimodal_data,30)
sample_30.mean()
✓ Create a sampling distribution of sample means

```

```
sample_means = []
for i in range(100):
    sample_30 = np.random.choice(bimodal_data,30)
    sample_means.append(sample_30.mean())

sample_means
```

```
# calculate sample_means
sample_means = []
for i in range(100):
    sample_30 = np.random.choice(bimodal_data,30)
    sample_means.append(sample_30.mean())

sample_means
```

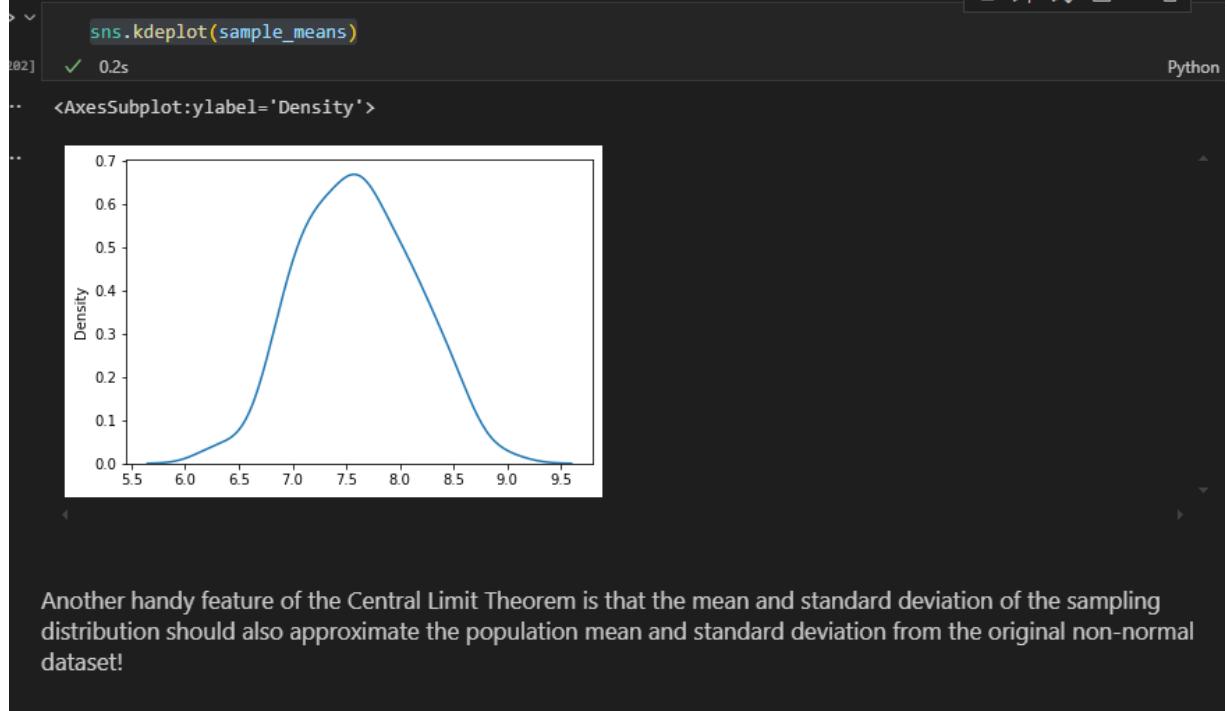
.62] ✓ 0.0s

```
[7.1531191237244025,
 8.108632954678022,
 8.169603659772104,
 7.771772666661061,
 7.254995979323588,
 7.631428341563319,
 7.731167609783466,
 6.372475511459524,
 6.782172500789712,
 7.923808465831867,
 7.494155951129795,
 8.183026498998263,
```

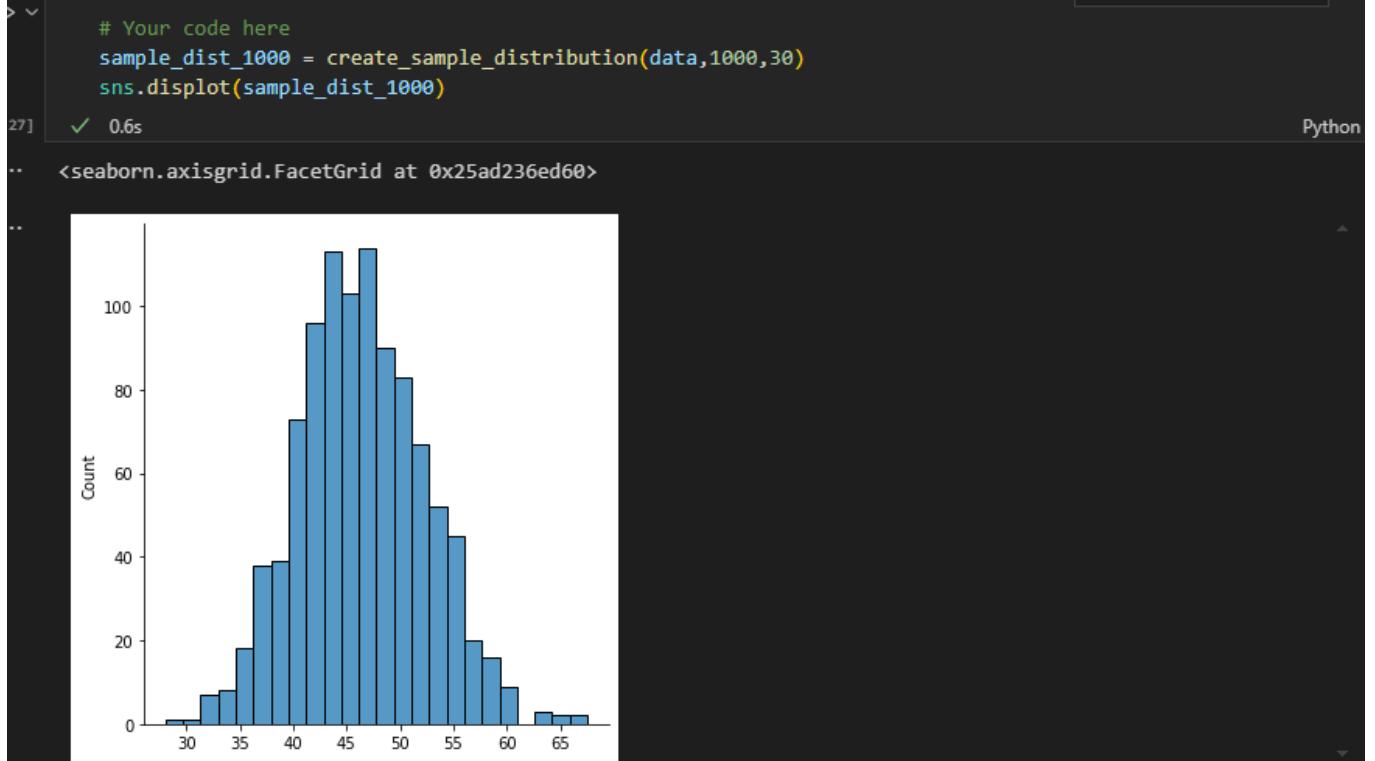
✓ Plot the sample means

```
sns.kdeplot(sample_means)
```

## Plotting

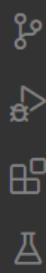


- Data almost look normal



Great! As you can see, the dataset *approximates* a normal distribution. It isn't pretty, but it's generally normal enough that we can use it to answer statistical questions using  $z$ -scores and p-values.

Another handy feature of the Central Limit Theorem is that the mean and standard deviation of the sampling distribution should also approximate the population mean and standard deviation from the original non-normal dataset! Although it's outside the scope of this lab, we could also use the same sampling methods seen here to approximate other parameters from any non-normal distribution, such as the median or mode!



## 2. Concepts to Learn Before the Central Limit Theorem:

- Population vs. Sample: A population includes every member of a specified group. A sample is a subset of that group. For instance, all residents of a town would be the population, whereas a group of 100 residents selected from that town would be a sample. Mean: The average value. For instance, the average height of a group of people.
- Standard Deviation: A measure of how spread out the numbers in a data set are. If everyone in a group is about the same height, the standard deviation is small. If their heights are all over the place, the standard deviation is large.
- Probability: The measure of the likelihood that an event will occur.
- Probability Distribution: A mathematical function that provides the probabilities of occurrence of different possible outcomes. For example, in a dice throw, there's a 1/6 chance for each number from 1 to 6.
- Sampling Distribution: The probability distribution of a statistic (like the mean) based on many samples from the same population.
- Normal Distribution (or Bell Curve): A specific probability distribution that is symmetric and has a bell-shaped curve. Many natural phenomena follow this distribution.

## 3. What is the Central Limit Theorem?

In simple terms, the Central Limit Theorem (CLT) states that regardless of the original distribution of the population, the sampling distribution of the sample mean approaches a normal distribution as the sample size increases. This is true when the sample size is sufficiently large, typically considered as  $n > 30$ .

### 3.1. Why is the Central Limit Theorem Important?

The central limit theorem is considered helpful for various reasons:

- A Foundational Principle in Inferential Statistics: CLT forms the foundation for many statistical procedures and tests, especially when the original data doesn't follow a normal distribution. To draw inferences: By knowing that the sampling distribution will be approximately normal, we can use the properties of a normal distribution to make inferences about population parameters.
- Applicability: As many natural phenomena are described by non-normal distributions, the CLT provides a way to make robust inferences based on the normality of sample means.



## 3.2. Assumptions and Requirements

There are few assumptions you need to consider when applying the central limit theorem.

- Random Sampling: Samples should be drawn randomly from the population. Independence: The sampled observations should be independent. This means that the occurrence of one event does not affect the probability of the other. This is often met if the sample is less than 10% of the population.
- Sample Size: Typically, a sample size above 30 is considered sufficient for the CLT to hold, though for largely skewed data, a larger sample might be necessary.

## 3.3. Applications of Central Limit Theorem

Central limit theorem is a foundational concept in statistics and is helpful in various other concepts / techniques as well:

- Confidence Intervals: CLT allows us to create confidence intervals around our sample mean for the population mean, even for non-normally distributed data.
- Hypothesis Testing: Many tests, like the t-test, assume that data is normally distributed. Even if the original data isn't normal, CLT ensures the sampling distribution is, allowing us to use such tests.
- Quality Control: In industries, especially manufacturing, the CLT is used to understand variations and bring products within quality specifications.

## 3.4. Limitations of Central Limit Theorem

While the Central Limit Theorem is widely applicable, it is not a magic bullet. For very skewed data or data with heavy tails, a larger sample size might be required. Also, it doesn't apply to median or mode, only the mean.

## 4. Central Limit Theorem in Action

- Population Distribution: Imagine a dice, whose outcomes (1, 2, 3, 4, 5, or 6) do not form a normal distribution. If we were to plot the probabilities, we would get a flat distribution since each outcome has an equal probability.
- Sample Distribution: Let's consider taking a sample of 2 dice rolls and calculating the average. The possible means range from 1 (if both dice show 1) to 6 (if both show 6). If you were to plot the frequencies of these sample means, the resulting shape would start to look a bit like a bell curve, though not perfectly normal.
- Increasing Sample Size: As the sample size increases, say 10 dice rolls, the shape becomes more and more like a perfect bell curve. This is the magic of the Central Limit Theorem in action!

51. **Confidence intervals** – range of values above and below the point estimate that captures the true population parameter at some predetermined confidence level

range of values surrounding an estimated parameter eg 12-18 inches is the height of chairs required for 12 yr-olds

(If we pulled 100 samples and constructed confidence intervals in the same manner, we would expect that 95 of the intervals would contain the true mean of population age. )

**Sample Distribution** - The sample distribution is the distribution of values in a single sample taken from a population.

**Sampling Distribution** - The sampling distribution is the distribution of a sample statistic (e.g., the mean, proportion, or variance) across multiple samples taken from the same population

- **Sample Distribution** is the distribution of raw data within a single sample.
- **Sampling Distribution** is the distribution of a statistic (like the mean) across many samples from the population, allowing us to make inferences about the population based on sample statistics.

Calculate the Confidence Interval: Using the formula:

$$CI = \bar{x} \pm z * SE$$

**Margin of Error** =  $z * \sigma / \sqrt{n}$  # where  $SE = \sigma / \sqrt{n}$  meaning MOE =  $z * SE$

Example:

```

#teacher code
np.random.seed(12)

# Select the sample size
sample_size = 1000

# Initialize lists to store interval and mean values
intervals = []
sample_means = []

# Run a for loop for sampling 25 times and calculate + store confidence interval and sample mean values

for sample in range(25):
    # Take a random sample of chosen size
    sample = np.random.choice(a= population_ages, size = sample_size)
    sample_mean = sample.mean()
    sample_means.append(sample_mean)

    z_critical = stats.norm.ppf(q = 0.975) # Get the z-critical value*
    
    # Calculate z_critical, margin_of_error, confidence_interval from function above
    # z_critical, margin_of_error, confidence_interval = conf_interval(population_ages, sample)

    pop_stdev = population_ages.std() # Get the population standard deviation

    stats.norm.ppf(q = 0.025)

    margin_of_error = z_critical * (pop_stdev/math.sqrt(sample_size))

    confidence_interval = (sample_mean - margin_of_error,
                           sample_mean + margin_of_error)

    intervals.append(confidence_interval)

    # Calculate and append sample means and conf intervals for each iteration
intervals

```

✓ 0.0s

Python

```

[(42.07615146777586, 43.71384853222415),
 (42.21315146777585, 43.85084853222414),
 (43.17115146777586, 44.80884853222415),
 (41.97015146777586, 43.607848532224146),
 (41.66015146777585, 42.306848532224144)

```

If we pulled 100 samples and constructed confidence intervals in the same manner, we would expect that 95 of the intervals would contain the true mean of population age.

**Example 2:**

```
# find the critical z value
z = stats.norm.ppf(0.95)
z

13] ✓ 0.0s Python
.. 1.6448536269514722

#population std
std = bimodal_data.std()
std

14] ✓ 0.0s Python
.. 2.6963633464167818

# get the standard error
SE = std/np.sqrt(len(sample_50))
SE

15] ✓ 0.0s Python
.. 0.38132336135883166

# calculate the margin of error
ME = z * SE
ME

16] ✓ 0.0s Python
.. 0.6272211139724011

# calculate the confidence interval
CI = (sample_50.mean() - ME, sample_50.mean() + ME)
CI #only 5 would have mean outside this range

17] ✓ 0.0s Python
.. (6.747389560245873, 8.001831788190675)
```

## With library

```
stats.norm.interval(alpha= 0.95,           # Confidence level
                    loc = sample_50.mean(),      # Sample mean
                    scale = SE)
```

# With Libraries

```
stats.norm.interval(alpha= 0.95,
                     loc = sample_50.mean(),
                     scale = SE)
# Confidence level
# Sample mean
# standard error
9] ✓ 0.0s
· (6.627230619491211, 8.121990728945336) Python
```



Confidence intervals make a statement of probability about the confidence interval range that could contain the true value.

## Zcore for a two-tailed test

$90\% = \text{norm.ppf}(95)$

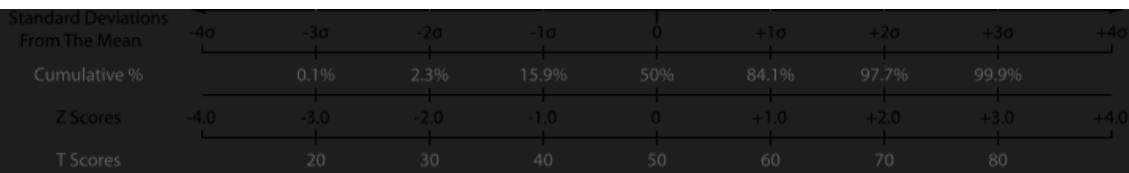
`Stats.norm.ppf()`

$z_{80} = \text{stats.norm.ppf}(0.9)$

## **calculate p-value from z score**

`stats.norm.cdf(z)`

`stats.norm.ppf(1-alpha)`



If we want our confidence level — i.e., how confident we are that the true value of the parameter lies within the confidence interval — to be:

- **90%**: The z-score multiplier should be  **$z = 1.645$** , because 90% of the area under the  $Z \sim N(0, 1)$  normal distribution lies between -1.645 and 1.645.
- **95%**: The z-score multiplier should be  **$z = 1.96$** , because 95% of the area under the  $Z \sim N(0, 1)$  normal distribution lies between -1.96 and 1.96.
- **99%**: The z-score multiplier should be  **$z = 2.575$** , because 99% of the area under the  $Z \sim N(0, 1)$  normal distribution lies between -2.575 and 2.575.

It is more suitable to get z-critical values with `stats.norm.ppf()` as the results are more accurate.

`stats.norm.ppf(q, loc=0, scale=1)` is a percent point function (inverse of cdf — percentiles).

Create a function to input population and sample data to calculate the confidence intervals:

```
from scipy.stats import norm

# Find the z-score for a 90% confidence level
z_90 = norm.ppf(0.95) # For 90%, we need 95% of the cumulative area
print(z_90) # Output: 1.645

# Find the z-score for a 95% confidence level
z_95 = norm.ppf(0.975) # For 95%, we need 97.5% of the cumulative area
print(z_95) # Output: 1.96

# Find the z-score for a 99% confidence level
z_99 = norm.ppf(0.995) # For 99%, we need 99.5% of the cumulative area
print(z_99) # Output: 2.575
```

[12] ✓ 0.0s

Python

... 1.6448536269514722  
1.959963984540054  
2.575093025482994

## Confidence Levels and Cumulative Area

### 1. 90% Confidence Level (Two-Tailed Test):

- To achieve a **90% confidence level** in a two-tailed test, we need 90% of the data within the confidence interval, leaving 10% in the two tails (5% in each tail).
- Thus, we want the z-score where **95% of the cumulative area** lies to the left, corresponding to `norm.ppf(0.95)`.
- This value, `z_90 = 1.645`, means that 1.645 standard deviations above the mean covers 95% of the cumulative area from the left.

### 2. 95% Confidence Level (Two-Tailed Test):

- For a **95% confidence level**, we need 95% of the area within the interval, leaving 5% in the tails (2.5% in each).
- So, we find the z-score with **97.5% cumulative area** to the left, corresponding to `norm.ppf(0.975)`.
- The result, `z_95 = 1.96`, means that  $\pm 1.96$  standard deviations from the mean captures 95% of the data in the normal distribution.

### 3. 99% Confidence Level (Two-Tailed Test):

- For a **99% confidence level**, we want 99% within the interval, leaving 1% in the tails (0.5% in each).
- We therefore calculate the z-score with **99.5% cumulative area** to the left, using `norm.ppf(0.995)`.
- The result, `z_99 = 2.575`, means that  $\pm 2.575$  standard deviations from the mean capture 99% of the distribution.



## CONFIDENCE INTERVALS WITH T DISTRIBUTION

- Used in statistics particularly when dealing with small sample sizes or when the population standard deviation is unknown

### Example 1:

Stats.t.ppf(0.95,ddof) where ddof = len(sample)-1 (get t-value) for 90%CI

Construct the Confidence Interval:

```
Confidence Interval=(x̄-ME,x̄+ME)
```

```
#calculate the sample mean  
sample_mean = sample_50.mean()  
sample_mean
```

20] ✓ 0.0s

.. 7.374610674218274

```
# get the t value  
t = stats.t.ppf(0.95,len(sample_50)-1)  
t
```

21] ✓ 0.0s

.. 1.6765508919142629

```
# get the SE  
SE = sample_50.std()/np.sqrt(len(sample_50))  
SE
```

22] ✓ 0.0s

.. 0.35343332276053774

```
ME = t * SE  
ME
```

23] ✓ 0.0s

.. 0.592548952506401

```
# calculate the confidence intervals  
CI = (sample_50.mean()-ME,sample_50.mean()+ME)  
CI
```

24] ✓ 0.0s

.. (6.782061721711873, 7.9671596267246745)

```
0.592548952500401

# calculate the confidence intervals
CI = (sample_50.mean()-ME,sample_50.mean()+ME)
CI

[24] ✓ 0.0s                                         Python
...
(6.782061721711873, 7.9671596267246745)

# check the population mean
bimodal_data.mean()

[25] ✓ 0.0s                                         Python
...
7.519332055822325
```

## Calculating Confidence intervals using libraries

```
> 
  stats.t.interval(alpha= 0.95,
                    df= len(sample_50)-1,
                    loc = sample_50.mean(),
                    scale = SE)           # Confidence level
                                         # Degrees of freedom
                                         # Sample mean
                                         # Estimated standard error of the mean

[26] ✓ 0.0s                                         Python
```

### EXAMPLE 2:

NB: added ddof when calculating standard deviation

We pass the parameter `ddof = 1` to `np.std` to make sure we correctly compute the standard deviation of the sample.

```
x_bar = np.mean(sample_chol_levels)
s = np.std(sample_chol_levels, ddof = 1)
se = s/np.sqrt(len(sample_chol_levels))
print(x_bar, s, se)
```

[3]

Python

... 62.45 18.72291376896235 4.186570792426661

```
import math
s/math.sqrt(len(sample_chol_levels))
```

[10]

Python

... 4.186570792426661

We then calculate our interval estimate using a t-distribution and our various parameters. The t-distribution requires 4 parameters:

- The sample mean
- The sample standard deviation
- The degrees of freedom (this is 1 less than the number of items in the sample)
- The confidence level we wish to have in our estimate

```
import scipy.stats as stats
```

[4]

Python

... c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\scipy\\_init\_.py:138: UserWarning: A NumPy version warning: A NumPy version >={np\_minversion} and <{np\_maxversion} is required for this version of "

```
stats.t.interval(alpha = 0.95,                      # Confidence level
                  df= len(sample_chol_levels)-1,      # Degrees of freedom
                  loc = x_bar,                      # Sample mean
                  scale = se)                      # Estimated standard error of the mean
```

[6]

Python

... (53.68740662596794, 71.21259337403207)

### Example 3: Using libraries

Note we pass the parameter `ddof = 1` to `np.std` to make sure we correctly compute the standard deviation of the sample.

```
[12] sample_chol_levels = np.random.normal(loc=54, scale=17, size=1000)
```

Python

```
[13] # With the randomized sample, the code is the same as above, but the output will be different.  
x_bar = np.mean(sample_chol_levels)  
s = np.std(sample_chol_levels, ddof = 1)  
se = s/np.sqrt(len(sample_chol_levels))  
print('Sample mean:', x_bar)  
print('Sample standard deviation:', s)  
print('Estimated standard error:', se)
```

Python

```
... Sample mean: 54.234458056469535  
Sample standard deviation: 17.122957029786754  
Estimated standard error: 0.5414754449131776
```

```
[16] #Min and Max of Confidence Interval  
stats.t.interval(alpha = 0.95,  
                  df= len(sample_chol_levels)-1,  
                  loc = x_bar,  
                  scale = se)
```

Python

```
... (53.17189834073073, 55.29701777220834)
```

## Summary

In this lesson, we investigated the more common method for calculating confidence intervals, as we will rarely know the population's standard deviation. As a result, we use the t-distribution, allowing us to find estimates for the population mean even when not knowing any specific parameters concerning the population.

### Example 4: Using a real life scenario

$n = 30$

$\text{mean} = 4.8$

$sd = 0.4$

```
t_value = stats.t.ppf(0.95, n-1)
```

```
margin_error = t_value * sd/(n**0.5)
```

```
confidence_interval = (mean - margin_error, mean + margin_error)
```

```
confidence_interval
```

## Scenario

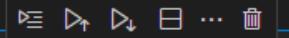
You are inspecting a hardware factory and want to construct a 90% confidence interval of acceptable screw lengths. You draw a sample of 30 screws and calculate their mean length as 4.8 centimeters and the standard deviation as 0.4 centimeters. What are the bounds of your confidence interval?

```
n = 30
mean = 4.8
sd = 0.4
t_value = stats.t.ppf(0.95, n-1)
margin_error = t_value * sd/(n**0.5)
confidence_interval = (mean - margin_error, mean + margin_error)

confidence_interval
```

Python

```
(4.6759133066001235, 4.924086693399876)
```



```
stats.t(loc=4.8, scale=0.4/(30**0.5), df=n-1).interval(alpha=0.9)
```

Python

```
(4.6759133066001235, 4.924086693399876)
```

## RECAP ON CENRTAL LIMIT THEOREM AND CONFIDENCE LEVEL

# Central Limit Theorem and Confidence Intervals - Recap

## Introduction

This short lesson summarizes the topics we covered in this section and why they'll be important to you as a data scientist.

## Key Takeaways

This section was all about building further on your statistics foundations by introducing the Central Limit Theorem and confidence intervals. Some of the key takeaways include:

- The Central Limit Theorem states that often, independent random variables summed together will converge to a normal distribution as the number of variables increases
- Using the Central Limit Theorem, we can work with non-normally distributed data sets as if they were normally distributed
- The Standard Error is a measure of spread - it is the standard deviation of samples from the sample mean
- If you take repeated samples and compute the 95% confidence interval for a given parameter for each sample, 95% of the intervals would contain the population parameter.
- The  $z$ -critical value is the number of standard deviations you'd have to go from the mean of the normal distribution to capture the proportion of the data associated with the desired confidence level.
- If you don't know the standard deviation for a population, you need to use t-distributions to compute the margin of error for calculating a confidence interval.

**52. Z distribution** – requires that we know the standard deviation of the population while **t-distribution** allows us to work with samples where the population standard deviation is unknown (as well as smaller samples) in order to form confidence intervals

**53. Standard Error** – measure of the variability or dispersion of a sample mean estimate relative to the true population mean. It quantifies how much the sample mean is expected to fluctuate from the true population mean due to sampling variability

## So what is standard error?

The **Standard Error (SE)** is very similar to the standard deviation. Both are measures of spread. The higher the number, the more spread out your data is. To put it simply, the two terms are essentially equal — but there is one important difference. While the standard error uses statistics (sample data), standard deviations use parameters (population data). We achieve this by dividing the standard deviation by the square root of the sample size.

The calculation for the standard error of the sample mean is:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \approx \frac{s}{\sqrt{n}}$$

Here,  $\sigma$  is the population standard deviation (which we will approximate with the sample standard deviation  $s$ ) and  $n$  is the sample size.

Example:

## Formula for Standard Error

The standard error of the sample mean is calculated using the formula:

Where:

```
s = standard deviation of the sample  
n = number of observations in the sample
```

```
# e.g with a population size of 50  
sample_50 = np.random.choice(bimodal_data, 50)  
  
sample_50.mean()  
✓ 0.0s
```

7.374610674218274

```
# population mean  
bimodal_data.mean()  
  
✓ 0.0s
```

7.519332055822325

```
# standard error  
SE = sample_50.std()/np.sqrt(len(sample_50))  
SE  
✓ 0.0s
```

0.35343332276053774

### 54. Generate data points using poisson distribution

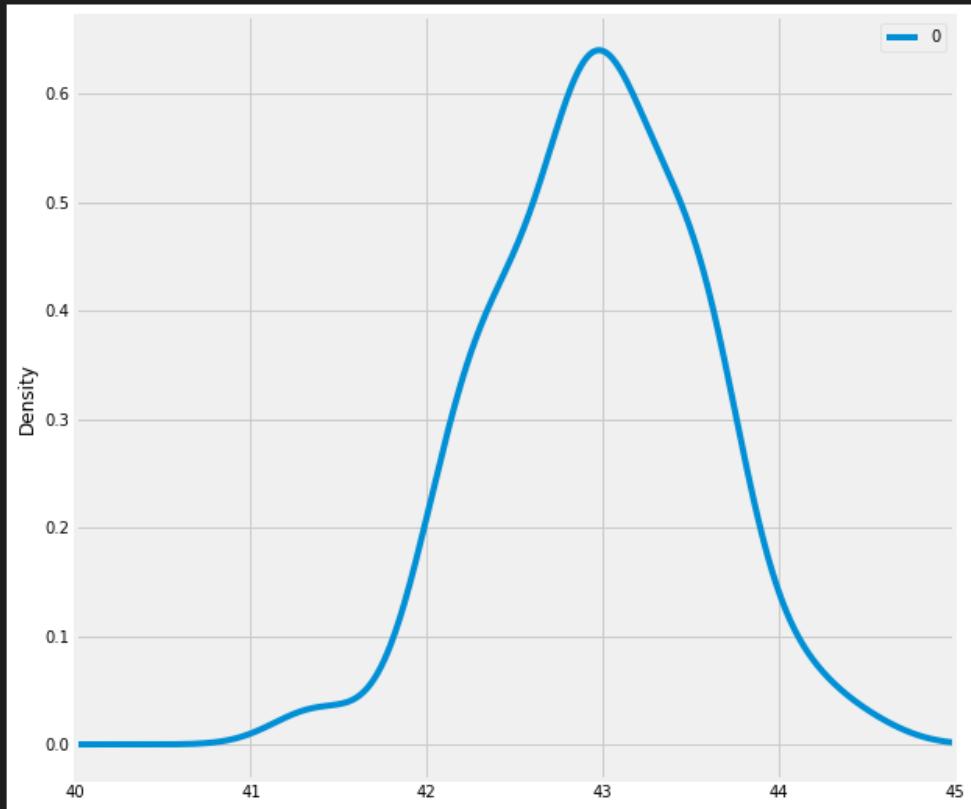
```
population_ages1 = stats.poisson.rvs(loc=18, mu=35, size=150000)  
population_ages2 = stats.poisson.rvs(loc=18, mu=10, size=100000)  
population_ages = np.concatenate((population_ages1, population_ages2))
```

### 55. Create normal distribution data from scipy

```
population = list(stats.norm.rvs(size=1000,  
                                 random_state=42))
```

56. Convert single column to a dataframe and plot

```
pd.DataFrame(point_estimates).plot(kind='density',# Plot sample mean density
                                         figsize=(9,9),
                                         xlim=(40,45)
                                         )
pd.DataFrame(point_estimates).plot(kind='density',# Plot sample mean density
                                         figsize=(9,9),
                                         xlim=[40,45]
                                         )
<AxesSubplot:ylabel='Density'>
```



```
sns.kdeplot(point_estimates, color='blue', fill=True, label='Density Plot (KDE)')
<AxesSubplot:ylabel='Density'>
```



57. Adding mean formula labels(x-bar using code)

```
import matplotlib.pyplot as plt
```

```

import seaborn as sns
fig, ax = plt.subplots()
ax = sns.kdeplot(sample, ax=ax, label='Sample PDF')
ax.axvline(sample_mean, color="red", label=r'$\bar{x}$')
ax.legend();

```

We can find their average (point estimate) like this:

```

sample_mean = sample.mean()
sample_mean
[2]   ✓  0.0s
...    14.9794176958

```

Python

In other words, our sample indicates that about 15 inches is the best chair back height.

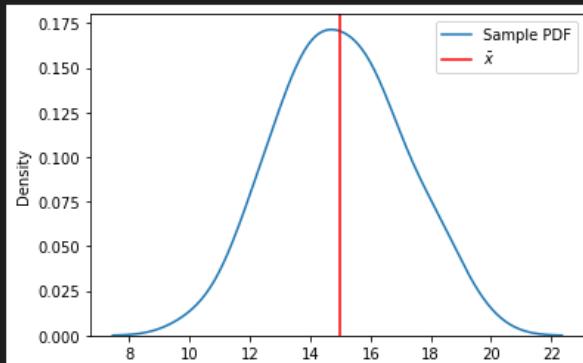
We can also plot our sample like this:

```

import matplotlib.pyplot as plt
import seaborn as sns
fig, ax = plt.subplots()
ax = sns.kdeplot(sample, ax=ax, label='Sample PDF')
ax.axvline(sample_mean, color="red", label=r'$\bar{x}$')
ax.legend();
[3]   ✓  1.7s
...  c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\scipy\_init_.py:138: UserWarning: A NumPy version
      warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "

```

Python



58. Add mu and sigma label using code

59. f, ax = plt.subplots()

60. ax = sns.kdeplot(population, ax=ax, label='Population PDF')

61. plt.axvline(pop\_mean, ls='--', c='r', label=r'\$\mu\$')

62.

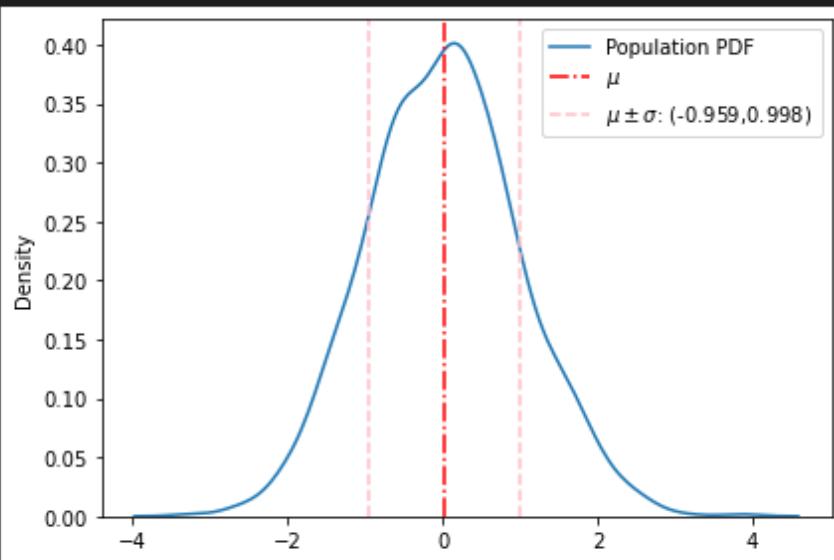
63. std\_label = f'\$\mu \pm \sigma\$: ({pop\_mean-pop\_std:.3f}, {pop\_mean+pop\_std:.3f})'

64. plt.axvline(pop\_mean-pop\_std, ls='--', c='pink')

65. plt.axvline(pop\_mean+pop\_std, ls='--', c='pink', label=std\_label)

```
66. ax.legend()  
67. plt.tight_layout()
```

- ```
# Visualizing population (approximately normal)  
f, ax = plt.subplots()  
ax = sns.kdeplot(population, ax=ax, label='Population PDF')  
plt.axvline(pop_mean, ls='--', c='r', label='$\mu$')  
  
std_label = f'$\mu \pm \sigma$: ({pop_mean-pop_std:.3f},{pop_mean+pop_std:.3f})'  
plt.axvline(pop_mean-pop_std, ls='--', c='pink')  
plt.axvline(pop_mean+pop_std, ls='--', c='pink', label=std_label)  
ax.legend()  
plt.tight_layout()
```



68. **Statistical significance** – determination about whether the observed effect in the data is unlikely to have occurred by random chance alone.

69. **Significance level** = alpha eg **0.05(5%)** 95% confidence level or **0.01(1%)** 99% confidence level

## 70. HYPOTHESIS TESTING

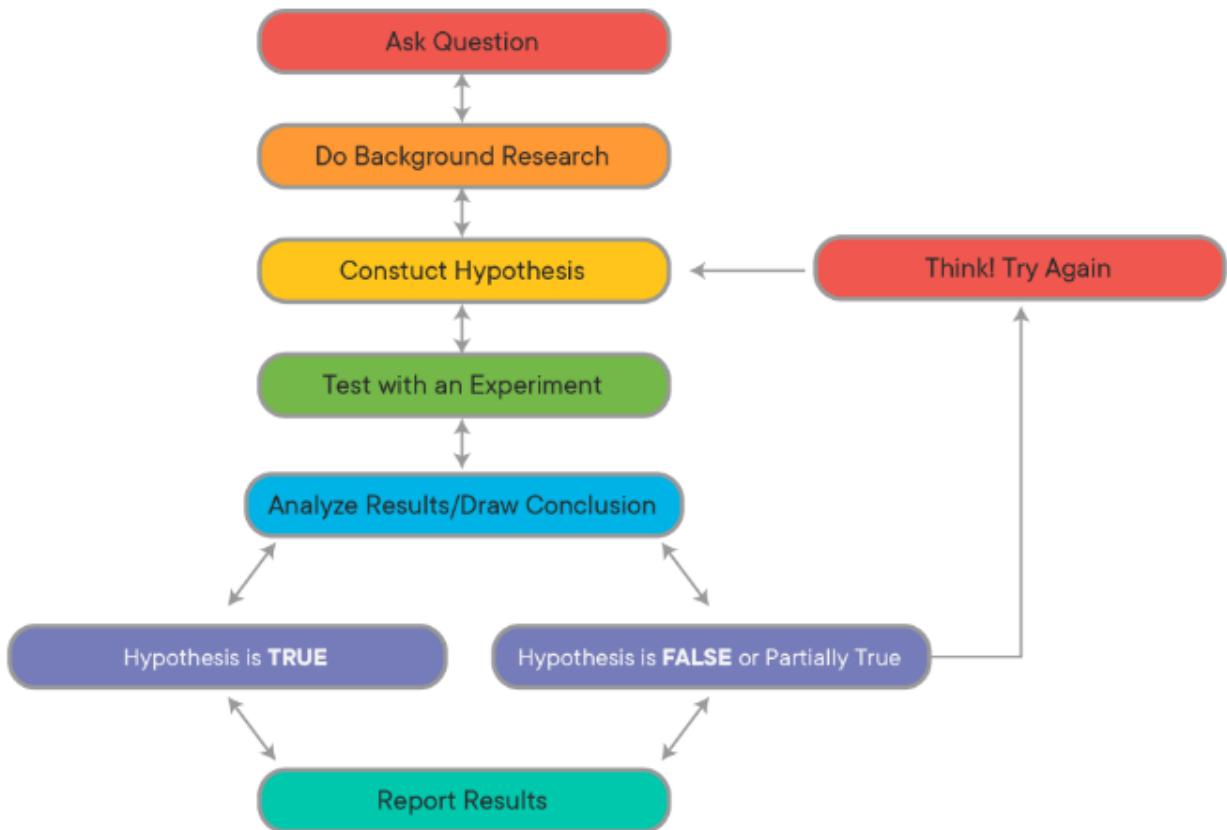
### a).General structure of an experiment

- **Make an Observation**- observe sth you want to test. During this step, you must observe phenomena to help refine the question that you want to answer. This might be anything from "does this drug have an effect on headaches?" to "does the color of this button affect the number of sales a website makes in a day?". Before testing these ideas, you need to observe that there might be some phenomena occurring and then come up with a specific question to answer.
- **Examine the research** - Good data scientists work smart before they work hard. In the case of the scientific method, this means seeing what research already exists that may help you answer your question, directly or indirectly. It could be that someone else has already done an experiment that answers your question--if that's the case, you should be aware of that

experiment before starting your own, as it could inform your approach to structuring your experiment, or maybe even answer your question outright!

- **Form an hypothesis** - you'll formulate 2 hypotheses to test--your educated guess about the outcome is called the *Alternative Hypothesis*, while the opposite of it is called the *Null Hypothesis*.
- **Conduct an experiment** - This step is the part of the scientific method that will be the focus of this section. You can only test a hypothesis by gathering data from a well-structured experiment. A well-structured experiment is one that accounts for all of the mistakes and randomness that could give you false signals relating to the effect of an intervention. Just because you're running an experiment doesn't prove that A causes B, or that there's even a relationship between A and B! A poorly designed experiment will lead to false conclusions that you haven't considered or controlled for. A well-designed experiment leaves you no choice but to acknowledge that the effects seen in a dependent variable are related to an independent variable. The world is messy and random. You have to account for this messiness and randomness in experiments so that you can filter it out and be left only with the things you're actively trying to measure.
- **Analyze experimental Results**- All the work you've done with statistics is usually in service of this goal--looking at the data and understanding what happened. During this step, you will tease out relationships, filter out noise, and try to determine if something that happened is *statistically significant* or not.
- **Draw conclusions** - This step is the logical endpoint for an experiment. You've asked a question, looked at experimental results from others that could be related to your question, made an educated guess, designed an experiment, collected data, and analyzed the results. The reality of this step is that you use your analysis of the data to do one of two things: either *reject the null hypothesis or fail to reject the null hypothesis*

# The Scientific Method



## The Foundations of a Sound Experiment

### b). Foundations of a sound experiment

All experiments are not created equal--simply following the steps outlined above does not guarantee that the results of any experiment will be meaningful. For instance, there's nothing stopping a person from testing the hypothesis that "wearing a green shirt will make it rain tomorrow!", seeing rain the next day, and rejecting the null hypothesis, thereby incorrectly "proving" that their choice of wardrobe affected the weather. Good experiments demonstrate that independent variables {X} have an effect on the dependent variables {Y} because you control for all the other things that could be affecting {Y}, until you are forced to conclude that the only thing that explains what happened to {Y} is {X}!

Although there are many different kinds of experiments, there are some fundamental aspects of experimental design that all experiments have:

#### ❖ Control group/Random controlled Trials -

One of the most important aspects of a sound experiment is the use of a **Control Group**. A Control Group is a cohort that receives no treatment or intervention--for them, it's just business as usual. In a medical test, this might be a *placebo*, such as a sugar pill. In the example of testing the color of a button on a website, this would be customers that are shown a version of the website with the button color unchanged. Using a control group allows researchers to

compare the results of doing nothing (the control) with the effects of doing something (the ***intervention***). Without a control group, you have no way of knowing how much of the results you see can be attributed to the intervention, and how much would have happened anyway.

**NB: A control group is only a control group if they are sampled from the same population as the treatment groups!**

The main way scientists deal with this is through ***Random Controlled Trials***. In a Random Controlled Trial, there is a control group and an intervention (also called treatment) group, where subjects are ***randomly assigned to each***.

A ***Single-Blind*** or ***Blind Trial*** is one where the participant does not know if they are receiving the treatment or a placebo.

A ***Double-Blind Trial*** is one where the participant does not know if they are receiving the treatment or a placebo, and neither does the person administering the experiment (because their bias could affect the outcomes, too!). Instead, knowing whether someone received the treatment or a placebo is kept hidden from everyone until after the experiment is over (obviously, *someone* has to know for recordkeeping purposes, but that person stays away from the actual experiment to avoid contaminating it with bias from that knowledge).

### ❖ **Appropriate Sampling techniques and sample size**

When data scientists are performing experiments, they rarely have the opportunity to work with an entire population of data. Rather, they must obtain a sample that is representative of the population. In order to get a high quality sample, you should follow these four assumptions related to sampling techniques and sample size.

- **Sample is independent**

Independence means the value of one observation does not influence or affect the value of other observations. Independent data items are not connected with one another in any way (unless you account for it in your model). This includes the observations in both the “between” and “within” groups of your sample. Non-independent observations introduce bias and can make your statistical test give too many false positives.

- **Sample is collected randomly**

A sample is random when each data point in your population has an equal chance of being included in the sample; therefore, the selection of any individual observation happens by chance, rather than by choice. This reduces the chance that differences in materials or conditions strongly bias results. Random samples are more likely to be representative of the population; therefore, you can be more confident with your statistical inferences with a random sample.

- **The sample is approximately normally distributed**

The normal distribution assumption is that the sampling distribution of the mean is normal. That is, if you took a sample, calculated its mean, and then you took another (independent) sample (from the same population) and got its mean (and repeated this an infinite number of times), then the distribution of the values that you wrote down would always be a perfect bell curve. This is the principle behind the Central Limit Theorem, and it is this idea that allows us to perform hypothesis tests. While maybe surprising, this assumption turns out to be relatively uncontroversial, at least when each of the samples is large, such as  $N \geq 30$ .

- **Appropriate Sample Size**

Randomness is a big problem in experiments. It can lead you to false conclusions by making you think that something doesn't matter when it does, or vice versa. Small sample sizes make experiments susceptible to the problem of randomness; whereas, large sample sizes protect experiments from it. The following scenario illustrates this point:

A person tells you that they can predict the outcome of a fair coin flip. You flip a coin, they call "tails", and they are correct. Is this enough evidence to accept or reject this person's statement? What if they got it right 2 times in a row? 5 times in a row? 55 times out of 100?

This situation illustrates two things that are important for us to understand and acknowledge:

1. No matter how large your sample size, there's always a chance that your results can be attributed to randomness or luck.
2. At some point, you would cross a threshold where random chance is small enough that you'd say "this probably isn't random", and are okay with accepting the results as the result of something other than randomness or luck.

With the situation above, you probably wouldn't assume that this person can predict coin flips after only seeing them get 1 correct. However, if this person got 970 out of 1000 correct, you would probably believe very strongly that this person *can* predict coin flips because the odds of guessing randomly and getting 970/1000 correct are very, very small--but not 0!

Large sample sizes protect us from randomness and variance. A more realistic example would be testing a treatment for HIV. Less than 1% of the global population carries a protective mutation that makes them resistant to HIV infection. If you took a randomly selected sample of 1 person from the population, there is a ~1% chance that you may mistakenly attribute successful prevention to the drug you're testing, when the results really happened because you randomly selected a person with this mutation. However, if your sample size was 100 people per sample, your odds of randomly selecting 100 people with that mutation are  $0.01^{100}$ . The larger your sample size, the more unlikely it is that you randomly draw people that happen to affect your study in a way that is not reflected by the general population.

### ❖ Reproducibility

- ➊ This one is a big one, and it represents a bit of a crisis in some parts of the scientific community right now. Good scientific experiments have **Reproducible Results!** This means that if someone else follows the steps you outline for your experiment and performs it themselves, they should get pretty much the same results as you did (allowing for natural variance and randomness). If many different people try reproducing your experiment and don't get the same results, this might suggest that your results are due to randomness, or to a **lurking variable** that was present in your samples that wasn't present in others. Either way, a lack of reproducibility often casts serious doubts on the results of a study or experiment.
- ➋ This is less of a problem for data scientists, since reproducibility usually just means providing the dataset you worked with and the corresponding Jupyter notebook. However, this isn't always the case! Luckily, you can use code to easily run your experiments multiple times and show reproducibility. When planning experiments, consider running them multiple times to ensure to really help show that your results are sound, and not due to randomness!

**71. Hypothesis testing** – statistical method used to make inferences or draw conclusions about a population based on sample data. It involves two competing hypothesis

- ✓ **Null hypothesis** – default assumption that there is no effect or no difference. It represents the status quo or a statement of no change
- ✓ **Alternative hypothesis** – This is what you want to prove. It suggests that there is an effect, a difference or a relationship

### Applications of hypothesis in real life

#### # Practicability of Hypothesis Testing in Real Life

##### 1. Finance

- **Investment Decisions**: Investors use hypothesis testing to evaluate whether a new investment strategy will outperform existing strategies. For example, a null hypothesis might state that the new strategy does not yield higher returns than the traditional strategy.
- **Risk Assessment**: Financial analysts may test whether changes in market conditions significantly impact asset prices. For instance, they might test if a particular economic indicator affects stock prices ( $H_0$ : no effect vs.  $H_1$ : significant effect).
- **Credit Risk**: Banks use hypothesis testing to determine whether a particular demographic group is more likely to default on loans, which can influence lending policies.

##### 2. Education

- **Program Evaluation**: Educators may use hypothesis testing to assess the effectiveness of new teaching methods or curricula. For example, they might test whether students taught with a new method score higher on standardized tests compared to those taught with traditional methods ( $H_0$ : no difference in scores).
- **Student Performance Analysis**: Schools can analyze whether there is a significant difference in performance between different groups of students (e.g., those receiving tutoring vs. those who do not).
- **Survey Analysis**: Educational institutions often conduct surveys to gather feedback on courses. Hypothesis testing can help determine if a new course format leads to higher student satisfaction levels compared to the previous format.

##### 3. Manufacturing

- **Quality Control**: Manufacturers use hypothesis testing to maintain product quality. For example, they might test whether a new production process results in a lower defect rate compared to the old process ( $H_0$ : defect rates are equal).
- **Process Improvement**: Companies may test hypotheses about the effects of changes in manufacturing processes on output. For instance, they might evaluate whether increasing the temperature during production improves product quality.

- **Supply Chain Decisions**: Businesses can test whether changes in supplier performance metrics significantly affect overall production efficiency, aiding in vendor selection and contract negotiations.

- - Chemistry - do inputs from two different barley fields produce different yields?
- - Astrophysics - do star systems with near-orbiting gas giants have hotter stars?
- - Medicine - BMI vs. Hypertension, etc.
- - Business - which ad is more effective given engagement?

## Examples of null and alternative hypothesis

**1. A drug manufacturer claims that a drug increases memory. It designs an experiment where both control and experimental groups are shown a series of images, and records the number of correct recollections until an error is made for each group.**

- **Null:** People who took the drug don't have more correct correlations than people who didn't take the drug
- **Alternative:** people who took the drug do have more correct recollections than people who didn't take the drug

**2. An online toystore claims that putting a 5 minute timer on the checkout page of its website decreases conversion rate. It sets up two versions of its site, one with a timer and one with no timer.**

- **Null:** putting a timer on the checkout page does not decrease conversion rate
- **Alternative:** putting a timer on the checkout page decreases conversion rate

**3. The Kansas City public school system wants to test whether the scores of students who take standardized tests under the supervision of teachers differ from the scores of students who take them in rooms with school administrators.**

- **Null:** There is no difference on scores of students under the supervision of teachers and that of school administrators
- **Alternative:** There is a difference on scores of students under the supervision of teachers and that of school administrators

### a).Hypothesis Testing Using the t-Distribution

- You construct a test statistic from the measured data and use the value of the statistic to decide whether to reject the null hypothesis.

**Test statistic**- lower-dimensional summary of the data but still maintains the discriminatory power necessary to determine whether a result is statistically significant for a given significance level

**t-test-** (also called Student's t-test) statistical test used to determine if there is a significant difference between the means of two groups

You should run a t-test when you either:

- Don't know the population standard deviation- used as it accounts for the additional uncertainty introduced by estimating the standard deviation from a small sample

- You have a small sample size -less than 30.Z-tests are generally used for larger samples, where the Central Limit theorem applies, allowing the use of normal distribution
- Normality- data should be approximately normally distributed.For small sample sizes this is critical

Like a z-test, the t-test also tells you how significant the differences are i.e. it lets you know if those differences could have happened by chance

## 72. Types of t-test

- ✓ **One sample t test** – Compares the mean of your sample data to a known value.eg you might want to know how your sample mean compares to a known population mean

*Suppose you are interested in determining whether a bakery production line produces cakes with a weight of exactly 2 pounds. To test this hypothesis, you could collect a sample of cakes from the production line, measure their weights, and compare the sample with a value of 2 using a one-sample t-test.*

### Sample Question 1

Acme Ltd. wants to improve sales performance. Past sales data indicate that the average sale was 100 dollars per transaction. After training the sales force, recent sales data (from a random sample of 25 salesmen) is shown below.\*

\*\*\*

[122.09, 100.64, 125.77, 120.32, 118.25,  
 96.47, 111.4 , 80.66, 110.77, 111.14,  
 102.9, 114.54, 88.09, 98.59, 87.07,  
 110.43, 101.9 , 123.89, 97.03, 116.23,  
 108.3, 112.82, 119.57, 131.38, 128.39]

```
import numpy as np
from scipy import stats
import math

# For visualizing distributions - optional
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data listed above
sample = np.array([122.09, 100.64, 125.77, 120.32, 118.25, 96.47, 111.4 , 80.66,
  110.77, 111.14, 102.9 , 114.54, 88.09, 98.59, 87.07, 110.43,
```

```
101.9 , 123.89, 97.03, 116.23, 108.3 , 112.82, 119.57, 131.38,
128.39])
#population mean
#(Known from past data)
mu = 100
mu
# Sample mean ( $\bar{x}$ ) using NumPy array method mean()
x_bar = sample.mean()
x_bar
# Sample standard deviation (sigma) using NumPy function std()
sigma = np.std(sample,ddof=1)
sigma
# Sample size (n)
n = len(sample)
n
#degrees of freedom
df = n-1
df
#Difference between sample and population mean
diff = x_bar - mu
diff
```

## Descriptive Statistics

Before completing the hypothesis test, let's calculate some summary statistics to see if the mean of the sample Run and Debug (Ctrl+Shift+D) is significantly different from the population. After, you can check to ensure that the data is relatively normal.

- **The population mean ( $\mu$ ):** Given as 100 (from past data).
- **The sample mean ( $\bar{x}$ ):** Calculate from the sample data
- **The sample standard deviation ( $s$ ):** Calculate from sample data
- **Number of observations ( $n$ ):** 25 as given in the question. This can also be calculated from the sample data.
- **Degrees of Freedom ( $df$ ):** Calculate from the sample as  $df = \text{total no. of observations} - 1$

```
[6] #import packages
import numpy as np
from scipy import stats
import math

# For visualizing distributions - optional
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data listed above
sample = np.array([122.09, 100.64, 125.77, 120.32, 118.25, 96.47, 111.4 , 80.66,
    110.77, 111.14, 102.9 , 114.54, 88.09, 98.59, 87.07, 110.43,
    101.9 , 123.89, 97.03, 116.23, 108.3 , 112.82, 119.57, 131.38,
    128.39])

... c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\scipy\_init_.py:138: UserWarning: A NumPy version warning.warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
    #population mean
    #(Known from past data)
    mu = 100
    mu
[7] 100

... # Sample mean ( $\bar{x}$ ) using NumPy array method mean()
```

```
mu
```

Python

100

```
# Sample mean ( $\bar{x}$ ) using NumPy array method mean()  
x_bar = sample.mean()  
x_bar
```

Python

109.5456

```
# Sample standard deviation (sigma) using NumPy function std()  
sigma = np.std(sample,ddof=1)  
sigma
```

Python

13.338774643871902

```
# Sample size (n)  
n = len(sample)  
n
```

Python

25

```
#degrees of freedom  
df = n-1  
df
```

Python

24

```
#Difference between sample and population mean  
diff = x_bar - mu  
diff
```

Python

9.545599999999993

Now, before we actually conduct a hypothesis test, we can summarize what we know about the data. This is a step that accidentally gets skipped sometimes, but it's actually really important!

It's useful to know what values you have found *before* trying to demonstrate that the values are statistically significant.

```
# Print the findings
print(f"""
The sample contains {n} observations, having a mean of {x_bar}
and a standard deviation (sigma) of {round(sigma,3)}, with {df} degrees of freedom.

The difference between sample and population means is {round(diff, 2)}.
""") | |
# The sample contains 25 observations, having a mean of 109.5456
# and a standard deviation (sigma) of 13.339, with 24 degrees of freedom.

# The difference between sample and population means is 9.55.
```

[13] Python

...  
The sample contains 25 observations, having a mean of 109.5456  
and a standard deviation (sigma) of 13.339, with 24 degrees of freedom.  
The difference between sample and population means is 9.55.

Ok, so we have found that the sample mean is about \$9.55 higher than the population mean. This indicates that, at least superficially, the training program appears to have increased sales by about \$9.55 per sale on average.

But, is that increase statistically significant?

If we want to investigate that question using a t-test, first we want to check that the sample is roughly normally distributed.

```
sns.set(color_codes=True)
sns.histplot(sample, kde=True, bins=5, color='darkblue');
```

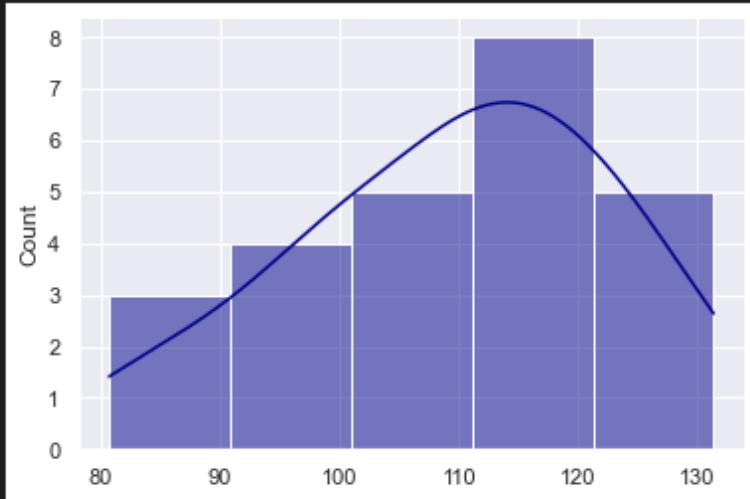
[33]

Python

```
sns.set(color_codes=True)
```

```
sns.histplot(sample, kde=True, bins=5, color='darkblue');
```

```
sns.set(color_codes=True)  
sns.histplot(sample, kde=True, bins=5, color='darkblue');
```



Let's say that looks close enough. Now we'll actually set up and run the hypothesis test.

Step1:Establish null and Alternative hypothesis

## Step 1: Establish Null and Alternative Hypotheses

As you are trying to monitor a change in the sales performance after the training, the null hypothesis represents what should be your default assumption — that the training did not have the hypothesized impact.

$H_0$ : *The null hypothesis is that the training did not increase sales, so:*

$$H_0 : \mu \leq 100$$

The alternative hypothesis is the one that you are testing. Our alternative hypothesis should address the expected change in the sales performance i.e. the sales performance has increased and the mean of sales post-training is greater than 100.

$H_a$ : *The alternative hypothesis is that there is a positive change (i.e. the mean sales increased), so:*

$$H_a : \mu > 100$$

**Why are these hypotheses written with  $\mu$  rather than  $\bar{x}$ ?**

Hypotheses are always written in terms of population parameters like  $\mu$  rather than sample statistics like  $\bar{x}$ , because our hypothesis is about a population, not a sample.

What we are really trying to understand here is whether these new sample data come from a *different population* than the one that produced the original  $\mu$  of 100.

So, another way we could write the alternative hypothesis would be that  $\mu > \mu_0$ , where  $\mu_0$  is the original population mean (100) and  $\mu$  is the population mean of those who had the new training.

We are using  $\bar{x}$  in our calculations determining whether we can reject the null hypothesis, but  $\bar{x}$  is not part of the hypotheses.

## Step 2: Choose a Significance Level (Alpha)

Here we'll detour slightly to think some more about what it means to choose a given alpha value. The graphs below are from a different example (not the Acme Ltd. sales data) and are used for illustration purposes.

The significance level, also denoted as alpha or  $\alpha$ , is the probability of rejecting the null hypothesis when it is true. For example, a significance level of 0.05 indicates a 5% risk of concluding that a difference exists when there is no actual difference. Look at the following graphs for a better understanding:

## Step 3: Calculate the t-statistic

Assuming that we are fulfilling the three requirements for a t-test mentioned above (i.e. normality, independence, and randomness), we are ready to calculate our t statistic using the formula for one-sample t-test given as:

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}$$

(The *t-statistic* is also known as the *t-value*.)

Using the formula given above, calculate the t-statistic in Python:

```
7] t= (x_bar - mu)/(sigma/np.sqrt(n))  
t
```

Python

## Step 4: Calculate Critical Value (Find Rejection Region)

Note that a positive t-value indicates that the sample mean is greater than the population mean. This means that the sample's average sales performance post-training is greater than average population sales performance. (We already knew this from our descriptive analysis when we found that  $\bar{x} - \mu$  was a positive value, and the t-statistic is just that divided by  $\frac{s}{\sqrt{n}}$ .)

This sounds like good news, **BUT** is the increase high enough to reject the null hypothesis, which says that there is no significant increase?

We'll answer this question by calculating a critical t-value. It's possible to calculate a critical t-value with a t-table or by using Python `scipy.stats` module.

The critical value approach involves determining "likely" or "unlikely", by determining whether or not the observed test statistic is more extreme than would be expected if the null hypothesis were true. This involves comparing the observed test statistic to some cutoff value, called the "**critical value**".

If the test statistic is more extreme than the critical value, then the null hypothesis is rejected in favor of the alternative hypothesis. If the test statistic is not as extreme as the critical value, then the null hypothesis is not rejected.

You need two values to find this:

The **alpha level**: given as 5% in the question.

You use this alpha level to determine what p-value to look up in this table. In the current example, we are executing a one-tailed t-test because we are only concerned with whether sales *increased*, not whether they were *different* in any way (which would be a two-tailed test). Therefore our p-value is just 0.05, from our 5% alpha.

**Degrees of freedom**, which is the number of items in the sample ( $n$ ) minus 1:  $25 - 1 = 24$ .

In the table below, look up the df of 24 and p of 0.05:

| t distribution critical values |                            |       |       |       |       |       |       |       |       |       |       |       |
|--------------------------------|----------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| df                             | Upper-tail probability $p$ |       |       |       |       |       |       |       |       |       |       |       |
|                                | .25                        | .20   | .15   | .10   | .05   | .025  | .02   | .01   | .005  | .0025 | .001  | .0005 |
| 1                              | 1.000                      | 1.376 | 1.963 | 3.078 | 6.314 | 12.71 | 15.89 | 31.82 | 63.66 | 127.3 | 318.3 | 636.6 |
| 2                              | 0.816                      | 1.061 | 1.386 | 1.886 | 2.920 | 4.303 | 4.849 | 6.965 | 9.925 | 14.09 | 22.33 | 31.60 |
| 3                              | 0.765                      | 0.978 | 1.250 | 1.638 | 2.353 | 3.182 | 3.482 | 4.541 | 5.841 | 7.453 | 10.21 | 12.92 |
| 4                              | 0.741                      | 0.941 | 1.190 | 1.533 | 2.132 | 2.776 | 2.999 | 3.747 | 4.604 | 5.598 | 7.173 | 8.610 |
| 5                              | 0.727                      | 0.920 | 1.156 | 1.476 | 2.015 | 2.571 | 2.757 | 3.365 | 4.032 | 4.773 | 5.893 | 6.869 |

|       |       |       |       |       |       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 1.000 | 1.376 | 1.963 | 3.078 | 6.314 | 12.71 | 15.89 | 31.82 | 63.66 | 127.3 | 318.3 | 636.6 |
| 2     | 0.816 | 1.061 | 1.386 | 1.886 | 2.920 | 4.303 | 4.849 | 6.965 | 9.925 | 14.09 | 22.33 | 31.60 |
| 3     | 0.765 | 0.978 | 1.250 | 1.638 | 2.353 | 3.182 | 3.482 | 4.541 | 5.841 | 7.453 | 10.21 | 12.92 |
| 4     | 0.741 | 0.941 | 1.190 | 1.533 | 2.132 | 2.776 | 2.999 | 3.747 | 4.604 | 5.598 | 7.173 | 8.610 |
| 5     | 0.727 | 0.920 | 1.156 | 1.476 | 2.015 | 2.571 | 2.757 | 3.365 | 4.032 | 4.773 | 5.893 | 6.869 |
| 6     | 0.718 | 0.906 | 1.134 | 1.440 | 1.943 | 2.447 | 2.612 | 3.143 | 3.707 | 4.317 | 5.208 | 5.959 |
| 7     | 0.711 | 0.896 | 1.119 | 1.415 | 1.895 | 2.365 | 2.517 | 2.998 | 3.499 | 4.029 | 4.785 | 5.408 |
| 8     | 0.706 | 0.889 | 1.108 | 1.397 | 1.860 | 2.306 | 2.449 | 2.896 | 3.355 | 3.833 | 4.501 | 5.041 |
| 9     | 0.703 | 0.883 | 1.100 | 1.383 | 1.833 | 2.262 | 2.398 | 2.821 | 3.250 | 3.690 | 4.297 | 4.781 |
| 10    | 0.700 | 0.879 | 1.093 | 1.372 | 1.812 | 2.228 | 2.359 | 2.764 | 3.169 | 3.581 | 4.144 | 4.587 |
| 11    | 0.697 | 0.876 | 1.088 | 1.363 | 1.796 | 2.201 | 2.328 | 2.718 | 3.106 | 3.497 | 4.025 | 4.437 |
| 12    | 0.695 | 0.873 | 1.083 | 1.356 | 1.782 | 2.179 | 2.303 | 2.681 | 3.055 | 3.428 | 3.930 | 4.318 |
| 13    | 0.694 | 0.870 | 1.079 | 1.350 | 1.771 | 2.160 | 2.282 | 2.650 | 3.012 | 3.372 | 3.852 | 4.221 |
| 14    | 0.692 | 0.868 | 1.076 | 1.345 | 1.761 | 2.145 | 2.264 | 2.624 | 2.977 | 3.326 | 3.787 | 4.140 |
| 15    | 0.691 | 0.866 | 1.074 | 1.341 | 1.753 | 2.131 | 2.249 | 2.602 | 2.947 | 3.286 | 3.733 | 4.073 |
| 16    | 0.690 | 0.865 | 1.071 | 1.337 | 1.746 | 2.120 | 2.235 | 2.583 | 2.921 | 3.252 | 3.686 | 4.015 |
| 17    | 0.689 | 0.863 | 1.069 | 1.333 | 1.740 | 2.110 | 2.224 | 2.567 | 2.898 | 3.222 | 3.646 | 3.965 |
| 18    | 0.688 | 0.862 | 1.067 | 1.330 | 1.734 | 2.101 | 2.214 | 2.552 | 2.878 | 3.197 | 3.611 | 3.922 |
| 19    | 0.688 | 0.861 | 1.066 | 1.328 | 1.729 | 2.093 | 2.205 | 2.539 | 2.861 | 3.174 | 3.579 | 3.883 |
| 20    | 0.687 | 0.860 | 1.064 | 1.325 | 1.725 | 2.086 | 2.197 | 2.528 | 2.845 | 3.153 | 3.552 | 3.850 |
| 21    | 0.686 | 0.859 | 1.063 | 1.323 | 1.721 | 2.080 | 2.189 | 2.518 | 2.831 | 3.135 | 3.527 | 3.819 |
| 22    | 0.686 | 0.858 | 1.061 | 1.321 | 1.717 | 2.074 | 2.183 | 2.508 | 2.819 | 3.119 | 3.505 | 3.792 |
| 23    | 0.685 | 0.858 | 1.060 | 1.319 | 1.714 | 2.069 | 2.177 | 2.500 | 2.807 | 3.104 | 3.485 | 3.768 |
| 24    | 0.685 | 0.857 | 1.059 | 1.318 | 1.711 | 2.064 | 2.172 | 2.492 | 2.797 | 3.091 | 3.467 | 3.745 |
| 25    | 0.684 | 0.856 | 1.058 | 1.316 | 1.708 | 2.060 | 2.167 | 2.485 | 2.787 | 3.078 | 3.450 | 3.725 |
| 26    | 0.684 | 0.856 | 1.058 | 1.315 | 1.706 | 2.056 | 2.162 | 2.479 | 2.779 | 3.067 | 3.435 | 3.707 |
| 27    | 0.684 | 0.855 | 1.057 | 1.314 | 1.703 | 2.052 | 2.158 | 2.473 | 2.771 | 3.057 | 3.421 | 3.690 |
| 28    | 0.683 | 0.855 | 1.056 | 1.313 | 1.701 | 2.048 | 2.154 | 2.467 | 2.763 | 3.047 | 3.408 | 3.674 |
| 29    | 0.683 | 0.854 | 1.055 | 1.311 | 1.699 | 2.045 | 2.150 | 2.462 | 2.756 | 3.038 | 3.396 | 3.659 |
| 30    | 0.683 | 0.854 | 1.055 | 1.310 | 1.697 | 2.042 | 2.147 | 2.457 | 2.750 | 3.030 | 3.385 | 3.646 |
| 40    | 0.681 | 0.851 | 1.050 | 1.303 | 1.684 | 2.021 | 2.123 | 2.423 | 2.704 | 2.971 | 3.307 | 3.551 |
| 50    | 0.679 | 0.849 | 1.047 | 1.299 | 1.676 | 2.009 | 2.109 | 2.403 | 2.678 | 2.937 | 3.261 | 3.496 |
| 60    | 0.679 | 0.848 | 1.045 | 1.296 | 1.671 | 2.000 | 2.099 | 2.390 | 2.660 | 2.915 | 3.232 | 3.460 |
| 80    | 0.678 | 0.846 | 1.043 | 1.292 | 1.664 | 1.990 | 2.088 | 2.374 | 2.639 | 2.887 | 3.195 | 3.416 |
| 100   | 0.677 | 0.845 | 1.042 | 1.290 | 1.660 | 1.984 | 2.081 | 2.364 | 2.626 | 2.871 | 3.174 | 3.390 |
| 1000  | 0.675 | 0.842 | 1.037 | 1.282 | 1.646 | 1.962 | 2.056 | 2.330 | 2.581 | 2.813 | 3.098 | 3.300 |
| $z^*$ | 0.674 | 0.841 | 1.036 | 1.282 | 1.645 | 1.960 | 2.054 | 2.326 | 2.576 | 2.807 | 3.091 | 3.291 |
|       | 50%   | 60%   | 70%   | 80%   | 90%   | 95%   | 96%   | 98%   | 99%   | 99.5% | 99.8% | 99.9% |

Hopefully you found the right value, 1.711! This is our critical t-value.

You can double-check that this is the right answer using the CDF (cumulative distribution function) of the t-distribution from `scipy.stats`. The CDF gives the probability that the t-statistic will be less than or equal to a given value. In this case, we are plugging in 1.711, our critical t-value:

```
stats.t.cdf(1.711, df=24)
```

[13] ✓ 0.0s

Python

... 0.9500110459177917

`stats.t.cdf(1.711, df=24)`

That checks out! It says that there is a 95% probability that the t-statistic will be less than or equal to 1.711, which is 1 minus our chosen alpha.

More often as a data scientist you will use code to find the critical t-value, rather than looking it up in a table. Conveniently, there is another function from `scipy.stats` that is just the inverse of the CDF shown above, called PPF. This stands for "percent point function". Given a probability, we use the PPF to compute the corresponding value — in this case, the critical t-value!

(Note that we need to pass in 1 minus alpha, not alpha, because we are finding the t-value where there is a 95% probability we are "right" and 5% probability we are "wrong", not a 5% probability we are "right".)

```
t_crit = stats.t.ppf(1 - 0.05, df=24)
t_crit
[14] 0.0s
... 1.7108820799094275
```

This is the same critical t-value we found in the table! With more digits, so it looks a little bit different.

The critical t-value marks the boundary of the **rejection region**. Because this is a one-tailed test, we can simply say that the rejection region is anything greater than the critical t-statistic. If our sample t-statistic falls into the rejection region, we can reject the null hypothesis!

`t_crit = stats.t.ppf(1 - 0.05, df=24)` right tail, greater than one sample

`t_crit`

## Step 5: Compare sample t-value with critical t-value

```
fig, ax = plt.subplots(figsize=(8,5))

#plot the PDF as a line graph
# (x and y were created in previous plotting code)
ax.plot(x,y,color='darkblue',label='t-distribution PDF')
#plot a vertical line for our critical t-value
ax.axvline(t_crit,color='g',ls='--',lw=4,label='critical t-value')
#fill area under line graph where x value is greater than critical t
ax.fill_betweenx(y,x,t_crit,where= x > t_crit,color='gray',label='rejection region')
#plot a vertical line for our measured difference in sales t-statistic
ax.axvline(t,color='r',ls='--',lw=5,label='observed-t-statistic')

ax.legend()
```

```

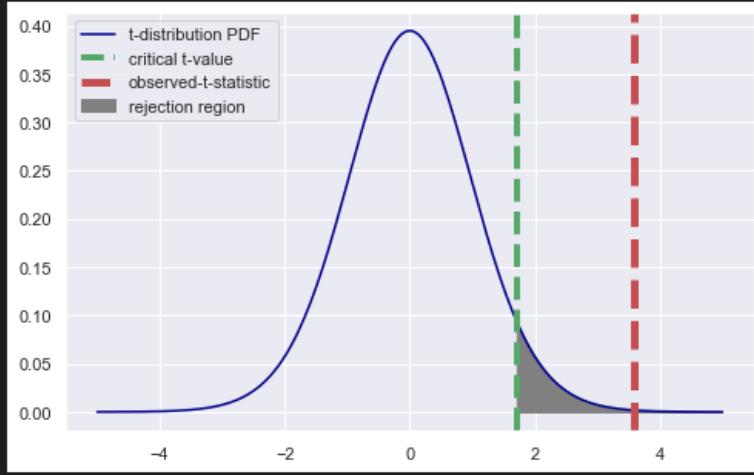
# (x and y were created in previous plotting code)
ax.plot(x,y,color='darkblue',label='t-distribution PDF')
#plot a vertical line for our critical t-value
ax.axvline(t_crit,color='g',ls='--',lw=4,label='critical t-value')
#fill area under line graph where x value is greater than critical t
ax.fill_betweenx(y,x,t_crit,where= x > t_crit,color='gray',label='rejection region')
#plot a vertical line for our measured difference in sales t-statistic
ax.axvline(t,color='r',ls='--',lw=5,label='observed-t-statistic')

ax.legend()

```

✓ 0.1s Python

<matplotlib.legend.Legend at 0x12362a4f7c0>



### **Is our observed t-statistic in the rejection region?**

Yes! It's hard to even see the rejection region shading at this extreme part of the distribution, but we are in it. That means we have completed the hypothesis test, and now we only need to interpret the result.

To break down the points:

- Our observed t-statistic is greater than the critical t-value,
- which means that it is in the rejection region,
- which means that we can reject the null hypothesis at a significance level of 0.05,
- which means that **there was a statistically significant increase in sales performance for those who received the training** (at our specified significance level)

**Use p\_value to reject(calculate p-value in t tests)**

p\_value = stats.t.sf(t,df=24)

p\_value

**p\_value for left tail**

stats.t.cdf(t\_statistic, df)

Although it isn't strictly necessary because we already have our answer, we could also calculate a p-value for this test. Recall that the p-value is the probability that we would observe at least this extreme of a t-statistic given that the null hypothesis is actually correct.

One way we could do this would be finding the highest value in the  $df = 24$  row of the lookup table above that is still below our t-statistic, then reporting that column. This tends to be fairly imprecise. It appears that our p-value is somewhere between 0.001 and 0.0005, since our t-statistic is between 3.467 and 3.745. If the details aren't important, you could just report this as  $p < 0.001$ .

Alternatively, there is functionality within `scipy.stats` to find a more precise p-value using the "survival function" SF:

```
p_value = stats.t.sf(t,df=24)
p_value
0.0s
```

Python

So, our p-value is about 0.00076, meaning that if the null hypothesis is correct (that there was no increase in sales performance with the training), we would expect it to produce results like we observed 0.076% of the time — very improbable!

Given our alpha of 0.05, this p-value is smaller than our alpha, therefore we can again say that we reject the null hypothesis at a significance level of 0.05.

## Sample Question 2

*Educators may use hypothesis testing to assess the effectiveness of new teaching methods or curricula. For example, they might test whether students taught with a new method score higher on standardized tests compared to those taught with traditional methods ( $H_0$ : no difference in scores).*

Hypothesis:

- Null Hypothesis ( $H_0$ ): There is no difference in scores between students taught using the new method and the traditional average score of 75.
- Alternative Hypothesis ( $H_1$ ): Students taught with the new method score higher than 75.



```
# Step 2: Calculate Sample Mean (X_bar)
X_bar = np.mean(scores)
X_bar

# Step 3: Calculate Sample Standard Deviation (s)
s = np.std(scores, ddof=1) # Use ddof=1 for sample standard deviation
s
diff = X_bar - population_mean
print('diff: there is a difference yes but is it statistically significant?', diff)
# Step 4: Calculate T-Statistic
t_statistic = (X_bar - population_mean) / (s / np.sqrt(n))
t_statistic
```

```
# Step 5: Calculate Degrees of Freedom
df = n - 1
df

# Step 6: Calculate Critical T-Value (one-tailed) for alpha = 0.05
alpha = 0.05
t_critical = 1.65

# Step 8: Output Results
print(f"Sample Mean (X_bar): {X_bar:.2f}")
print(f"Sample Standard Deviation (s): {s:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"Degrees of Freedom (df): {df}")
#print(f"T-Critical Value ( $\alpha=0.05$ ): {t_critical:.4f}")
#print(f"P-Value: {p_value:.4f}")

# Step 9: Conclusion
if t_statistic > t_critical:
    print("Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher than 75.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in scores.")
import scipy.stats as st
print(st.t.ppf(1-0.05,19)) #from ttable use df as 19 and 0.05 to give 1.729#check greater (right tail)
```

```

X_bar = np.mean(scores)
X_bar

# Step 3: Calculate Sample Standard Deviation (s)
s = np.std(scores, ddof=1) # Use ddof=1 for sample standard deviation
s
diff = X_bar - population_mean
print('diff:there is a difference yes but is it statistically significant?',diff)
# Step 4: Calculate T-Statistic
t_statistic = (X_bar - population_mean) / (s / np.sqrt(n))
t_statistic

# Step 5: Calculate Degrees of Freedom
df = n - 1
df

# Step 6: Calculate Critical T-Value (one-tailed) for alpha = 0.05
alpha = 0.05
t_critical = 1.65

# Step 8: Output Results
print(f"Sample Mean (X_bar): {X_bar:.2f}")
print(f"Sample Standard Deviation (s): {s:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"Degrees of Freedom (df): {df}")
#print(f"T-Critical Value (a=0.05): {t_critical:.4f}")
#print(f"P-Value: {p_value:.4f}")

# Step 9: Conclusion
if t_statistic > t_critical:
    print("Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in scores.")
import scipy.stats as st
print(st.t.ppf(1-0.05,19)) #from ttable use df as 19 and 0.05 to give 1.729#check greater (right tail)

```

✓ 0.0s

Python

```

diff:there is a difference yes but is it statistically significant? 8.950000000000003
Sample Mean (X_bar): 83.95
Sample Standard Deviation (s): 5.49
T-Statistic: 7.2888
Degrees of Freedom (df): 19
Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher than 75.
1.729132811521367

```

## Using stats Library ttest\_1samp sample question 1

stats.ttest\_1samp gives tstatic / t\_value

t\_critic = stats.t.ppf(1-0.05/2,df=19) for two tailed tests

t\_critic = stats.t.ppf(1-0.05,df=19) for one tailed tests

```

results = stats.ttest_1samp(
    a=sample, #entire array-like sample
    popmean = 100 #mean u'r testing the sample against
)

```

```
results
alpha = 0.05
t_crit = stats.t.ppf(1-alpha,df=len(sample)-1)

if results.statistic > t_crit:
    print('Null hypothesis rejected')
    print('The increase in sales performance is statistically significant at the',alpha,'level')
else:
    print('Failed to reject the null hypothesis.')
    print('The increase in sales performance is not statistically significant at the', alpha,'level')
```

```
results = stats.ttest_1samp(  
    a=sample, #entire array-like sample  
    popmean = 100 #mean u'r testing the sample against  
)  
results|  
✓ 0.0s
```

Python

```
Ttest_1sampResult(statistic=3.578139767278185, pvalue=0.0015178945415114085)
```

This `results` variable contains both the t-statistic and the p-value. Below we compare the t-statistic from this calculation to the one we calculated earlier:

```
print(t)  
print(results.statistic)|  
✓ 0.0s
```

Python

```
3.578139767278185  
3.578139767278185
```

We can either use the t-statistic and rejection region approach next, or we can just use the p-value!

## Rejection Region Approach with `ttest_1samp`

One approach would be to compute the critical t-value, then compare the t-value from our `ttest_1samp` results to that critical t-value.

```
alpha = 0.05  
t_crit = stats.t.ppf(1-alpha,df=len(sample)-1)  
  
if results.statistic > t_crit:  
    print('Null hypothesis rejected')  
    print('The increase in sales performance is statisctically significant at the',alpha,'level')  
else:  
    print('Failed to rject the null hypothesis.')  
    print('The increase in sales performance is not statistically at the', alpha,'level')|  
✓ 0.0s
```

Python

```
Null hypothesis rejected  
The increase in sales performance is statisctically significant at the 0.05 level
```

Using p-value we divide by two since since ttest assumes you are performing two sided one-sample t-test

```
print(p_value)  
print(results.pvalue / 2)
```

Let's look at that more closely:

```
[21]    print(p_value)
          print(results.pvalue)

[21]    ✓ 0.0s
```

Python

```
...   0.0007589472707557043
      0.0015178945415114085
```

The p-value from `results` looks like it's twice as large as the one we calculated! In fact, it's exactly twice as large:

```
[22]    print(p_value)
          print(results.pvalue / 2)

[22]    ✓ 0.0s
```

Python

```
...   0.0007589472707557043
      0.0007589472707557043
```

In this case with very small p-values it doesn't affect our answer either way — both 0.0015 and 0.00076 are smaller than 0.05.

But in other cases this difference in p-values could mean the difference between rejecting and failing to reject the null hypothesis!

So, what should we do?

Simply **divide the p-value result from `ttest_1samp` by 2 if you are using it for a one-sided t-test**. If you are using it for a two-sided t-test, no further action is required.

In conclusion, we can use the p-value from `ttest_1samp` to answer our research question like this:

```
if (results.pvalue / 2) < alpha:
    print("Null hypothesis rejected.")
    print("The increase in sales performance is statistically significant at the", alpha, "level.")
else:
    print("Failed to reject the null hypothesis.")
    print("The increase in sales performance is not statistically significant at the", alpha, "level.")
```

```
[23]    ✓ 0.0s
```

Python

```
print("results.statistic:", results.statistic)
print("results[0]:    ", results[0])
print("results.pvalue: ", results.pvalue)
print("results[1]:    ", results[1])
```

That was easy! This p-value approach meant that we never had to calculate the critical t, we just handed the work over to SciPy. It was conceptually a bit more challenging but the code was shorter.

Unless you are specifically instructed to use one approach or the other (rejection region or p-value), either one works well.

Also, note that you might also see a couple different techniques for extracting information from the t-test results object. It is designed so you can use dot notation or indexing notation to retrieve the same values:

```
print("results.statistic:", results.statistic)
print("results[0]:      ", results[0])
print("results.pvalue:   ", results.pvalue)
print("results[1]:      ", results[1])
✓ 0.0s
```

Python

```
results.statistic: 3.578139767278185
results[0]:      3.578139767278185
results.pvalue:   0.0015178945415114085
results[1]:      0.0015178945415114085
```

## Using stats Library ttest\_1samp sample question 2

## Using Stats Library

```
import numpy as np
from scipy import stats

# Data
scores = np.array([78, 82, 85, 80, 90, 76, 88, 91, 84, 79, 87, 92, 77, 81, 89, 93, 86, 75, 82, 84])

# One-sample t-test
t_statistic, p_value = stats.ttest_1samp(scores, population_mean)

# Output results
print(f"T-statistic: {t_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation
alpha = 0.05
if t_statistic > t_critical:
    print("Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in scores.")
```

✓ 0.0s

Python

```
T-statistic: 7.2888
P-value: 0.0000
Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher than 75.
```

```
if p_value/2 < alpha:
    print("Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in scores.")
```

✓ 0.0s

Python

```
Reject the null hypothesis ( $H_0$ ): Students taught with the new method score significantly higher than 75.
```

- ✓ Two sample t test – used to compare if two population means are equal

Types:

- Paired -useful for determining how different a sample is affected by a treatment  
In other words, the individual items/people in the sample will **remain the same** and researchers are comparing how they change after treatment. Here is an example of a scenario where a two-sample paired t-test could be applied:

*The US Olympic weightlifting team is trying out a new workout technique to in an attempt to improve everyone's powerlifting abilities. Did the program have an effect at a 95% significance level?*

Because we are looking at how specific individuals were affected by a treatment, we would use the paired t-test.

- Independent- Independent two-sample t-tests are for when we are comparing two different, unrelated samples to one another. Unlike paired t-tests, we are not taking paired differences because there is no way

to pair two unrelated samples! Here is an example of a scenario where a two-sample independent t-test could be applied:

- ✚ Agricultural scientists are trying to compare the difference in soybean yields in two different counties in Mississippi.

### **Sample Question for paired two tests**

## ^ 2. Paired Samples T-Test (Dependent T-Test) #two sample mean

### **Explanation**

A paired samples t-test compares the means of two related groups. This test is useful when the same subjects are measured under two different conditions.

The formula for the paired samples t-test is:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}$$

Where:

- $\bar{d}$  = mean of the differences between pairs
- $s_d$  = standard deviation of the differences
- $n$  = number of pairs

### **Example**

- *the effectiveness of a teaching method is evaluated before and after the intervention on the same group of students.*

Hypothesis:

- Null Hypothesis ( $H_0$ ): There is no difference in test scores before and after the teaching intervention.
- Alternative Hypothesis ( $H_1$ ): There is a significant difference in test scores before and after the teaching intervention.

# Step 1: Data

```
scores_before = np.array([70, 75, 68, 80, 78, 74, 73, 81, 69, 77,
    72, 76, 79, 74, 82, 71, 75, 78, 70, 80,
    76, 72, 79, 74, 81])
```

```
scores_after = np.array([85, 90, 82, 88, 87, 84, 83, 89, 81, 86,
    80, 85, 89, 83, 91, 78, 84, 87, 81, 88,
```

```
85, 80, 88, 83, 90])
```

```
# Step 2: Calculate differences
```

```
differences = scores_after - scores_before  
print(differences)
```

```
# Step 3: Calculate the mean of the differences (d_bar)
```

```
mean_difference = np.mean(differences)
```

```
# Step 4: Calculate the standard deviation of the differences (s_d)
```

```
std_difference = np.std(differences, ddof=1) # Use ddof=1 for sample standard deviation
```

```
# Step 5: Calculate the number of pairs (n)
```

```
n = len(differences)
```

```
# Step 6: Calculate the T-Statistic
```

```
t_statistic = mean_difference / (std_difference / np.sqrt(n))
```

```
# Step 7: Calculate Degrees of Freedom (df)
```

```
df = n - 1
```

```
# Step 8: Calculate P-Value
```

```
p_value = stats.ttest_rel(scores_before, scores_after).pvalue
```

```
# Step 9: Output Results
```

```
print(f"Mean of Differences (d_bar): {mean_difference:.2f}")  
print(f"Standard Deviation of Differences (s_d): {std_difference:.2f}")  
print(f"T-Statistic: {t_statistic:.4f}")  
print(f"Degrees of Freedom (df): {df}")
```

```
print(f"P-Value: {p_value}") #print(f"P-Value: {p_value:.4f}")

# Step 10: Conclusion
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in test scores.")
```

```
# Step 1: Data
scores_before = np.array([70, 75, 68, 80, 78, 74, 73, 81, 69, 77,
                         72, 76, 79, 74, 82, 71, 75, 78, 70, 80,
                         76, 72, 79, 74, 81])
scores_after = np.array([85, 90, 82, 88, 87, 84, 83, 89, 81, 86,
                        80, 85, 89, 83, 91, 78, 84, 87, 81, 88,
                        85, 80, 88, 83, 90])

# Step 2: Calculate differences
differences = scores_after - scores_before
print(differences)

# Step 3: Calculate the mean of the differences (d_bar)
mean_difference = np.mean(differences)

# Step 4: Calculate the standard deviation of the differences (s_d)
std_difference = np.std(differences, ddof=1) # Use ddof=1 for sample standard deviation

# Step 5: Calculate the number of pairs (n)
n = len(differences)

# Step 6: Calculate the T-Statistic
t_statistic = mean_difference / (std_difference / np.sqrt(n))
|
# Step 7: Calculate Degrees of Freedom (df)
df = n - 1

# Step 8: Calculate P-Value
p_value = stats.ttest_rel(scores_before, scores_after).pvalue

# Step 9: Output Results
print(f"Mean of Differences (d_bar): {mean_difference:.2f}")
print(f"Standard Deviation of Differences (s_d): {std_difference:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"Degrees of Freedom (df): {df}")
print(f"P-Value: {p_value}") #print(f"P-Value: {p_value:.4f}")

# Step 10: Conclusion
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in test scores.")
```

```

# Step 10: Conclusion
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in test scores.")

✓ 0.0s

15 15 14 8 9 10 10 8 12 9 8 9 10 9 9 7 9 9 11 8 9 8 9 9
[9]
ean of Differences (d_bar): 9.72
tandard Deviation of Differences (s_d): 2.13
t-Statistic: 22.8007
egrees of Freedom (df): 24
t-Value: 8.972244521459368e-18
ject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.

np.mean(scores_after) - np.mean(scores_before)
✓ 0.0s
.7199999999999999

stats.ttest_rel(scores_before, scores_after)

test_relResult(statistic=-22.800741386563786, pvalue=8.972244521459368e-18)

```

## Using stats library

```

import numpy as np
from scipy import stats

# Step 1: Data
scores_before = np.array([70, 75, 68, 80, 78, 74, 73, 81, 69, 77,
                         72, 76, 79, 74, 82, 71, 75, 78, 70, 80,
                         76, 72, 79, 74, 81])
scores_after = np.array([85, 90, 82, 88, 87, 84, 83, 89, 81, 86,
                        80, 85, 89, 83, 91, 78, 84, 87, 81, 88,
                        86, 82, 88, 85, 89, 87, 84, 86, 83, 85])

```

```
85, 80, 88, 83, 90])  
  
# Step 2: Use the stats library to perform the paired t-test  
t_statistic, p_value = stats.ttest_rel(scores_after, scores_before)  
  
# Step 3: Output Results  
print(f"T-Statistic: {t_statistic:.4f}")  
print(f"P-Value: {p_value:.4f}")  
  
# Step 4: Conclusion  
alpha = 0.05  
if p_value < alpha:  
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.")  
else:  
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in test scores.")
```

```

t_critic= 2.064#(24,0.05 on two tails)
stats.t.ppf(1-alpha/2,df)

[20]    ✓  0.0s
...    2.0638985616280205

```

Using stats library

```

import numpy as np
from scipy import stats

# Step 1: Data
scores_before = np.array([70, 75, 68, 80, 78, 74, 73, 81, 69, 77,
                         72, 76, 79, 74, 82, 71, 75, 78, 70, 80,
                         76, 72, 79, 74, 81])
scores_after = np.array([85, 90, 82, 88, 87, 84, 83, 89, 81, 86,
                        80, 85, 89, 83, 91, 78, 84, 87, 81, 88,
                        85, 80, 88, 83, 90])

# Step 2: Use the stats library to perform the paired t-test
t_statistic, p_value = stats.ttest_rel(scores_after, scores_before)

# Step 3: Output Results
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Step 4: Conclusion
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in test scores.")

[ ]

```

[ ]

... T-Statistic: 22.8007  
P-Value: 0.0000  
Reject the null hypothesis ( $H_0$ ): There is a significant difference in test scores.

### #test assuming identical variances for two sample

stats.ttest\_ind(experimental, control)

**stats.ttest\_ind** (Independent T-Test):

**stats.ttest\_rel** (Paired T-Test):

### stats.ttest\_ind (Independent T-Test):

- Used for comparing the means of two independent, unrelated groups.
- Assumes that the two groups are from populations with similar variances (homogeneity of variances). If this assumption doesn't hold, you can use equal\_var=False in the function to apply Welch's t-test, which does not assume equal variances.
- Example use case: Testing if two different groups (e.g., treatment and control) have different means on a continuous variable, like comparing test scores between two separate groups of students.

### stats.ttest\_rel (Paired T-Test):

- Used for comparing the means of two related groups, typically the same group measured at two different times (e.g., before and after treatment) or under two different conditions.
- Assumes the samples are paired, meaning each data point in one group has a corresponding data point in the other group.
- Example use case: Testing if there is a significant difference in pre-test and post-test scores for the same group of individuals.

### Sample Question for independent two tests

An independent samples t-test compares the means of two independent groups to determine if they are significantly different from each other.

# Step 1: Data

```
strategy_a = np.array([250, 300, 275, 400, 320, 275, 310, 290, 330, 350, 275, 360, 310, 295, 340])  
strategy_b = np.array([220, 250, 300, 280, 270, 240, 250, 310, 290, 265, 230, 275, 280, 260, 240])
```

# Step 2: Use the stats library to perform the independent t-test

```
t_statistic, p_value = stats.ttest_ind(strategy_a, strategy_b) #used when we have both samples  
if we don't have all samples we use
```

```
stats.ttest_ind_from_stats(mean1=mean_A, std1=std_A, nobs1=nobs_A,  
                           mean2=mean_B, std2=std_B, nobs2=nobs_B,  
                           equal_var=False)
```

# Step 3: Output Results

```
print(f"T-Statistic: {t_statistic:.4f}")  
print(f"P-Value: {p_value:.4f}")
```

# Step 4: Conclusion

```
alpha = 0.05
```

```
if p_value < alpha:
```

```
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in average sales.")  
else:
```

```
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in average sales.")
```

# Using Stats

```
# Step 1: Data
strategy_a = np.array([250, 300, 275, 400, 320, 275, 310, 290, 330, 350, 275, 360, 310, 295, 340])
strategy_b = np.array([220, 250, 300, 280, 270, 240, 250, 310, 290, 265, 230, 275, 280, 260, 240])

# Step 2: Use the stats library to perform the independent t-test
t_statistic, p_value = stats.ttest_ind(strategy_a, strategy_b)

# Step 3: Output Results
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Step 4: Conclusion
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis ( $H_0$ ): There is a significant difference in average sales.")
else:
    print("Fail to reject the null hypothesis ( $H_0$ ): No significant difference in average sales.")
```

Python

```
T-Statistic: 3.9539
P-Value: 0.0005
Reject the null hypothesis ( $H_0$ ): There is a significant difference in average sales.
```

## t-critical formulas(summary notes)

if one\_tailed:

```
    t_critical = stats.t.ppf(1 - alpha, df)#mostly used
```

```
    critical_region = (t_critical, np.inf)
```

else:

```
    t_critical_left = stats.t.ppf(alpha / 2, df)
```

```
    t_critical_right = stats.t.ppf(1 - alpha / 2, df)
```

```
    critical_region = (t_critical_left, t_critical_right)
```

Z-test  
calculate Z-statistic using populations standard deviation ( $\sigma$ ).

$$z = \frac{x - \mu}{\sigma/\sqrt{n}}$$

p-value

- T-distribution as more probability in 2 tails. As samples increase size, this decreases t & distribution more closely resembles Z, or standard normal distribution by sample  $n=100$

they're virtually indistinguishable from each other

Critical value (state t.ppt ( $1-\alpha, df$ ) right tail) greater  
crit t.ppt ( $\alpha, df$ ) left tail few (another)  
state ppt ( $1-\alpha/2, df$ ) right tail 3 2 sample test  
( $\alpha/2, df$ ) left critical

The below 2 are for two tail not 2 sample

t-value / t-statistic

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

(statistic > t-crit (one-tail),  
P-value < alpha)

check P-value by 2 if using one-sided t-test, no  
further action + 2-sided t-test

t-stat > crit value then we reject null hypothesis

P-value

cstat > t-crit (tstat > t-crit) right-tailed greater

If P-value less than alpha we reject null hypothesis  
(tstat < t-crit (tstat < t-crit) left-tail less)

for a left-tailed

we reject if tstat < critic  
right-tailed

tstat > critic

-value for 2 sample

stat = ttest\_1nd (mean1 = mean A, std1 = std A, nobs1 =

mean2 = mean B, std2 = std B, nobs2 =

1 = sample 1 nobs = no of observations

2 = sample 2

)

stat = ttest\_1nd (a=sample 1, b=sample 2)

sample one-tailed

AB Testing

test design parameters include effect size, sample size

simulated questions

using appropriate parameters

## 73. P-VALUES AND NULL HYPOTHESIS

### Understanding the Null Hypothesis

As stated previously, scientific experiments actually have 2 hypotheses:

**Null Hypothesis:** There is no relationship between A and B

Example: "There is no relationship between this flu medication and a *reduced* recovery time from the flu."

The *null hypothesis* is usually denoted as  $H_0$

**Alternative Hypothesis:** The hypothesis traditionally thought of when creating a hypothesis for an experiment

Example: "This flu medication reduces recovery time for the flu."

The *alternative hypothesis* is usually denoted as  $H_1$  or  $H_a$

An easy way to differentiate between the null hypothesis and the alternative hypothesis is that the null hypothesis is the more conservative choice. It always assumes that there is no difference between some variables of interest, and therefore when it is represented mathematically, it should always contain an equals sign ( $=$ ,  $\geq$ , or  $\leq$ ).

The alternative hypothesis is whatever claim you are trying to prove with an experiment. It posits that there is some difference between some variables of interest, and therefore when it is represented mathematically it should never contain an equals sign, but rather some symbol representing inequality ( $\neq$ ,  $<$ , or  $>$ ).

Whenever you are performing a statistical test, you are determining whether you have enough evidence to reject the null hypothesis. If you do not have enough evidence, you fail to reject the null hypothesis.

### p-Values and Alpha Values

No matter what you're experimenting on, statistical tests come down to one question: Is your p-value less than your alpha value? Let's dive into what each of these values represents, and why they're so important to experimental design.

**p-value:** The probability of observing a test statistic at least as large as the one observed, by random chance, assuming that the null hypothesis is true.

If you calculate a p-value and it comes out to 0.03, you can interpret this as saying "There is a 3% chance of obtaining the results I'm seeing when the null hypothesis is true."

A low p-value means that either *the null hypothesis is false* or *the null hypothesis is true and a highly improbable event has occurred*. It is impossible to know which of these is what actually happened! So the best practice is to set an  $\alpha$  prior to conducting the experiment, to establish what level of this particular kind of incorrectness ~~you are willing to tolerate~~.

A low p-value means that either *the null hypothesis is false* or *the null hypothesis is true and a highly improbable event has occurred*. It is impossible to know which of these is what actually happened! So the best practice is to set an  $\alpha$  prior to conducting the experiment, to establish what level of this particular kind of incorrectness you are willing to tolerate.

$\alpha$  (**alpha value**): The marginal threshold at which you're okay with agreeing that the null hypothesis is implausible enough to be rejected.

If you set an alpha value of  $\alpha = 0.05$ , you're essentially saying "I'm okay with accepting my alternative hypothesis as true when we expect that the null hypothesis would randomly cause the results I'm seeing less than 5% of the time."

When you conduct an experiment, your goal is to calculate a p-value and compare it to the alpha value. If  $p < \alpha$ , then you **reject the null hypothesis** because the idea that there is "no relationship" between the variables of interest has become too implausible. Note that any good scientist will admit that this doesn't prove that there is a *direct causal relationship* between some chosen "independent" and "dependent" variables, just that they now have enough evidence to the contrary to show that they no longer believe that there is no relationship between them.

In simple terms:

$p < \alpha$ : Reject the *null hypothesis* and in favor of the *alternative hypothesis*

$p \geq \alpha$ : Fail to reject the *null hypothesis*.

So for example, if you set an alpha of 0.05 and your statistical test produces a p-value of 0.02, you reject the null hypothesis. (You would say *we reject the null hypothesis at a significance level of 0.05*.) If you set an alpha of 0.05 and your statistical test produces a p-value of 0.06, you fail to reject the null hypothesis. (You would say *we fail to reject the null hypothesis at a significance level of 0.05*.)

**Whether something is statistically significant depends on the specified alpha value.** Statistical significance is not a purely objective measure; it is dependent on the design choices you make as you set up your experiment.

So for example, if you set an alpha of 0.05 and your statistical test produces a p-value of 0.02, you reject the null hypothesis. (You would say *we reject the null hypothesis at a significance level of 0.05*.) If you set an alpha of 0.05 and your statistical test produces a p-value of 0.06, you fail to reject the null hypothesis. (You would say *we fail to reject the null hypothesis at a significance level of 0.05*.)

## NONE AND ALTERNATIVE HYPOTHESIS EXAMPLES

There are many different ways that you can structure a hypothesis statement, but they always come down to some statement of equality (null hypothesis) and some statement of inequality (alternative hypothesis).

It is important that these two hypotheses are mutually exclusive (cannot both occur at the same time, so the intersection of  $H_0$  and  $H_a$  is the empty set) and exhaustive (all possible scenarios are represented by the union of  $H_0$  and  $H_a$ ). (You will see some contexts where they are not defined in an exhaustive way, but for our statistical tests they will need to be.)

In all of the following examples, we will focus on **statistical tests that compare means ( $\mu$  values) of normally distributed data**. This is only one kind of statistical test! There are numerous other statistical tests for comparing categorical data, comparing variances of data, comparing an observed distribution to a theoretical distribution, comparing a sample to a population, etc. that we are not digging into just yet.

## One-Tail and Two-Tail Tests

In normally distributed data, you calculate p-values from t-statistics (or z-scores if the population parameters are known). One of the main considerations is whether to perform a **one-tailed** or a **two-tailed** test.

### Example One-Tail Hypotheses

A **one-tailed test** is when you want to know if a parameter from the treatment group is greater than (or less than) a corresponding parameter from the control group.

$H_1 : \mu_{control} < \mu_{treatment}$  The treatment group given this weight loss drug will lose more weight on average than the control group that was given a competitor's weight loss drug

$H_0 : \mu_{control} \geq \mu_{treatment}$  The treatment group given this weight loss drug will not lose more weight on average than the control group that was given a competitor's weight loss drug".

In general,  $<$  or  $>$  in the alternative hypothesis and  $\geq$  or  $\leq$  in the null hypothesis indicates that this is a one-tailed test.

### Example Two-Tail Hypotheses

A **two-tailed test** is for when you want to test if a parameter falls between (or outside of) a range of two given values.

$H_1 : \mu_{control} \neq \mu_{treatment}$  "People in the experimental group that are administered this drug will not lose the same amount of weight as the people in the control group. They will be heavier or lighter".

$H_0 : \mu_{control} = \mu_{treatment}$  "People in the experimental group that are administered this drug will lose the same amount of weight as the people in the control group."

In general,  $\neq$  in the alternative hypothesis and  $=$  in the null hypothesis indicate that this is a two-tailed test.

## SAMPLES VS TAILS

## Samples vs. Tails

Note that we now have two different labels where "one" and "two" appear repeatedly. Let's make sure the difference is clear!

Previously we learned about *one-tail* and *two-tail* tests. A one-tail test means that the alternative hypothesis contains something like  $>$  or  $<$ , which means that we are only looking at the area under the curve on one side. Whereas a two-tail test means that the alternative hypothesis contains something like  $\neq$  (which could mean greater than *or* less than), which means that we are looking at the area under the curve in two places, one on each side.

One-sample tests can be one-tail or two-tail tests, as can two-sample tests.

Some quick examples:

- One-sample one-tail:  $H_a: \mu < 3$
- One-sample two-tail:  $H_a: \mu \neq 3$
- Two-sample one-tail:  $H_a: \mu_1 < \mu_2$
- Two-sample two-tail:  $H_a: \mu_1 \neq \mu_2$

## 74. TYPE 1 AND TYPE II ERRORS

- ✓ **Alpha and TYPE I (False positive)** – probability that you reject the null hypothesis when its actually true. If a researcher was set to alpha 0.05 this indicates there is a chance that you will reject the null hypothesis when its actually true
- Could also be explained as when we reject the null hypothesis(favouring the alternative hypothesis) but the 'truth' we should have failed to reject (favouring the null hypothesis)  
**Consequences** - False claims fake alarms, in medical research it could mean approving a treatment that has no real benefit which could be harmful or costly  
**Minimize** – Choose a lower alpha value such as 0.01 instead of 0.05 however this makes its more stringent which increases the risk of a type II error.
  - ✓ **Beta and TYPE II Errors (False negative)** – probability that you fail to reject the null hypothesis given that it is actually false
- Could also be explained as when we fail to reject the null hypothesis(favouring the null hypothesis) but the truth is that we should have rejected the hypothesis(favouring the alternative hypothesis) I,e we conclude that there is no effect or difference when in reality there is one.  
Eg if a drug actually works but you fail to reject the null hypothesis and conclude that it doesn't work.

**Consequences** – missing a real effect, which could delay discoveries or prevent effective treatment from being recognized. In scientific research it could mean overlook true differences or relationships potentially wasting time and resources

**Minimize**- increase sample size.larger sample provide more reliable estimates and increases tests statisctical power

- Increase significance level-using a higher alpha(0.1) instead of 0.05.increaes the likelihood of rejecting the null hypothesis which reduces type II but increases type I error

### **(Hypothesis examples)**

#### **1. Two tailed example**

are consistent before and after their move. They buy a new machine and hire a new barista. In Manhattan, lattes are made with 4 oz of espresso. A random sample of 25 lattes made in their new store in Brooklyn shows a mean of 4.6 oz and standard deviation of 0.22 oz. Are their lattes different now that they've relocated to Brooklyn? Use a significance level of `alpha = 0.01`.

**Null:** The Brooklyn lattes are not different in size from the Manhattan lattes.

**Alternative:** The Brooklyn lattes are different in size from the Manhattan lattes.

```
t_stat = (4.6 - 4) / (0.22 / np.sqrt(25))
```

```
# This is a two-tailed test, so we want 1/2 of the 1% for
```

```
# the right tail and 1/2 for the left tail.
```

```
t_crit = stats.t.ppf(0.995, df=24) #stats.t.ppf(1-0.01/2,24)
```

```
print("t-statistic: ", t_stat)
```

```
print("critical t-value: ", t_crit)
```

```
print("p-value: ", stats.t(df=24).sf(t_stat))
```

```
# Because the p-value is lower than our threshold of 1% (or, equivalently, because our t-statistic is larger than the critical t-statistic), we should reject the null hypothesis. The Brooklyn lattes are different!
```

```
t_stat = (4.6 - 4) / (0.22 / np.sqrt(25))
# This is a two-tailed test, so we want 1/2 of the 1% for
# the right tail and 1/2 for the left tail.
t_crit = stats.t.ppf(0.995, df=24) #stats.t.ppf(1-0.01/2,24)
print("t-statistic: ", t_stat)
print("critical t-value: ", t_crit)
print("p-value: ", stats.t(df=24).sf(t_stat))

# Because the p-value is lower than our threshold of 1% (or, equivalently, because our t-statistic is larger than the critical t-statistic), we should reject the null hypothesis. The Brooklyn lattes are different!
✓ 0.0s

t-statistic: 13.63636363636363
critical t-value: 2.796939504772804
p-value: 4.242714627951655e-13
```

#### **2. Left tail example(one-tail)**

I'm buying jeans from Pants-a-torium. I know nothing about their inventory other than prices after looking at some random jean prices:

```
50, 75, 25, 30, 30, 40, 80]
```

However, I know that my typical pants store (Pants-R-Us) sells jeans at an average price of \$58 with a standard deviation of \$18. (Yes, I do descriptive statistics while I shop. Why do you ask?)

Should I go just to one store for a less expensive pair of jeans? I'm pretty apprehensive about my decision, so `alpha = 0.1$`.

**Null:** The pants at Pants-a-torium are not cheaper than the pants at Pants-R-Us.

**Alternative:** The pants at Pants-a-torium are cheaper than the pants at Pants-R-Us.

```
alpha = 0.1
sample = [20, 30, 30, 50, 75, 25, 30, 30, 40, 80]
df = len(sample)-1
t_statistic,p_value = stats.ttest_1samp(sample,58)
print('t_statistic',t_statistic,'p_value ',p_value/2) #since this formula is for two tail
t_crtitical = stats.t.ppf(alpha,df) #one tail(left) less than
print('t_crtitical',t_crtitical)
stats.t.cdf(t_statistic, df)
# p-value is small so we reject the null that is pants in p-atorium are cheaper
#using t-critic since this is a left-tailed test if t_stat < t_critic we reject the null hypothesis
```

```
alpha = 0.1
sample = [20, 30, 30, 50, 75, 25, 30, 30, 40, 80]
df = len(sample)-1
t_statistic,p_value = stats.ttest_1samp(sample,58)
print('t_statistic',t_statistic,'p_value ',p_value/2) #since this formula is for two tail
t_crtitical = stats.t.ppf(alpha,df) #one tail(left) less than
print('t_crtitical',t_crtitical)
stats.t.cdf(t_statistic, df)
# p-value is small so we reject the null that is pants in p-atorium are cheaper
#using t-critic since this is a left-tailed test if t_stat < t_critic we reject the null hypothesis
✓ 0.0s

t_statistic -2.56934288148538 p_value  0.015110178147891276
t_crtitical -1.3830287383964925

0.015110178147891276
```

### 3. Two sample independent test

You measure the delivery times of ten different restaurants in two different neighborhoods. You want to know if restaurants in the different neighborhoods have the same delivery times. Set your significance threshold to 0.05.

We measured this for neighborhood A:

```
delivery_times_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]
```

For neighborhood B, someone already reported the values as a mean time of 26.8 minutes and a standard deviation of 2.6 minutes.

**Null:** The delivery times for the restaurants in Neighborhood A are the same as the times for the restaurants in Neighborhood B

**Alternative:** The delivery times for the restaurants in Neighborhood A are NOT the same as the times for the restaurants in Neighborhood B.

#This is a TWO-SAMPLE problem. So we'll use stats.ttest\_ind\_from\_stats(). The two sets don't have exactly the same variance, so we'll perform Welch's test by setting the equal\_var parameter to False.

```
delivery_times_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]
mean_A = np.mean(delivery_times_A)
std_A = np.std(delivery_times_A)
nobs_A = len(delivery_times_A)
mean_B = 26.8
std_B = 2.6
nobs_B = 10
stats.ttest_ind_from_stats(mean1=mean_A, std1=std_A, nobs1=nobs_A,
                           mean2=mean_B, std2=std_B, nobs2=nobs_B,
                           equal_var=False)
```

# The p-value is greater than our threshold value of 5%, so we can't reject the null hypothesis that the two neighborhoods have the same restaurant delivery times.

```
#This is a TWO-SAMPLE problem. So we'll use stats.ttest_ind_from_stats(). The two sets don't have exactly the same variance, so we'll perform Welch's test by setting the equal_var parameter to False.
delivery_times_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]
mean_A = np.mean(delivery_times_A)
std_A = np.std(delivery_times_A)
nobs_A = len(delivery_times_A)
mean_B = 26.8
std_B = 2.6
nobs_B = 10
stats.ttest_ind_from_stats(mean1=mean_A, std1=std_A, nobs1=nobs_A,
                           mean2=mean_B, std2=std_B, nobs2=nobs_B,
                           equal_var=False)
# The p-value is greater than our threshold value of 5%, so we can't reject the null hypothesis that the two neighborhoods have the same restaurant delivery times.
✓ 0.0s
Ttest_IndResult(statistic=1.8206924435070326, pvalue=0.08535597189429299)
```

## 75. AB TESTING- general methodology when u want to test a new feature and/or product(especially for online products)

- > A/B testing is best suited for informing us what improvements we can make on something we already have. It's not really meant to tell us what \_new direction\_ we should go.
- >
- > An analogy: A/B testing is meant to tell us how to climb up the mountain we're already on, but not necessarily \_what\_ mountain we should be on.

A/B is essentially hypothesis testing but applied to a business problem

1. **Experimental testing** – p-value by itself is prone to misinterpretation if not presented with other relevant design parameters such as effect size, sample size, and alpha.
- Before conducting A/B testing, it's important to ensure **“well-designed experiments”** that can provide meaningful results.

### Key Design Principles:

- **Well-Formulated Questions\*\*:** Specific and measurable to avoid unintended consequences.
- **Selecting Appropriate Parameters\*\*:** Balance between alpha ( $\alpha$ ), power, sample size, and effect size.
- **Preprocessing Data\*\*:** Handle anomalies and outliers before running statistical tests.

**\*\*Goodheart's Law\*\*:** When a measure becomes a target, it ceases to be a good measure.

## 2. Effect size and p-values

**Effect size** quantifies how large the difference is between two groups, providing practical insights beyond mere statistical significance.

### Relationship between Effect size and P-value

- P-value indicates if the difference is statistically significant
- Effect size shows how meaningful the difference is

### Why do we need to calculate effect size

- Communicate the practical significance of the results. An effect might be statistically significant, but does it matter in practical scenarios?
- Effect size calculation and interpretation allows you to draw meta-analytical conclusions i.e combining and comparing estimates from different studies conducted on different samples. This allows you to group together a number of existing studies, calculate the meta-analytic effect size and get best estimate of the effect size of the population.
- Perform power analysis which helps determine the number of participants (sample size) that a study requires to achieve a certain probability of finding a true effect – if there is any

#### Example 1: Height difference between males and females

Explore the effect size of height between males and females in the US

```
# Import required libraries
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
from statsmodels.stats.power import TTestIndPower

%matplotlib inline
✓ 3.0s
```

```
# Define mean and standard deviation for male and female heights
male_mean, male_sd = 178, 7.7
female_mean, female_sd = 163, 7.3

# Create normal distributions for heights
male_height = stats.norm(male_mean, male_sd) #The result male_height is a SciPy rv object
which
#represents a normal continuous random variable.
female_height = stats.norm(female_mean, female_sd)

# Function to evaluate the PDF
```

```
def evaluate_PDF(rv, x=4):
    '''Input: a random variable object, standard deviation
    output : x and y values for the normal distribution
    '''
    # Identify the mean and standard deviation of random variable
    mean, std = rv.mean(), rv.std()
    # Use numpy to calculate evenly spaced numbers over the specified interval (4 sd) and
    # generate 100 samples.
    xs = np.linspace(mean - x * std, mean + x * std, 100)
    # Calculate the peak of normal distribution i.e. probability density.
    ys = rv.pdf(xs)
    return xs, ys

# Plot PDFs for both distributions to visualize the effect size
xs, ys = evaluate_PDF(male_height)
plt.plot(xs, ys, label='Male', color='blue')

xs, ys = evaluate_PDF(female_height)
plt.plot(xs, ys, label='Female', color='orange')
plt.xlabel('Height (cm)')
plt.legend()
plt.title("PDF of Male and Female Heights")
plt.show()
```

```

female_mean, female_sd = 163, 7.3

# Create normal distributions for heights
male_height = stats.norm(male_mean, male_sd) #The result male_height is a SciPy rv object which
#represents a normal continuous random variable.
female_height = stats.norm(female_mean, female_sd)

# Function to evaluate the PDF
def evaluate_PDF(rv, x=4):
    '''Input: a random variable object, standard deviation
    output : x and y values for the normal distribution
    ...
    # Identify the mean and standard deviation of random variable
    mean, std = rv.mean(), rv.std()
    # Use numpy to calculate evenly spaced numbers over the specified interval (4 sd) and
    # generate 100 samples.
    xs = np.linspace(mean - x * std, mean + x * std, 100)
    # Calculate the peak of normal distribution i.e. probability density.
    ys = rv.pdf(xs)
    return xs, ys

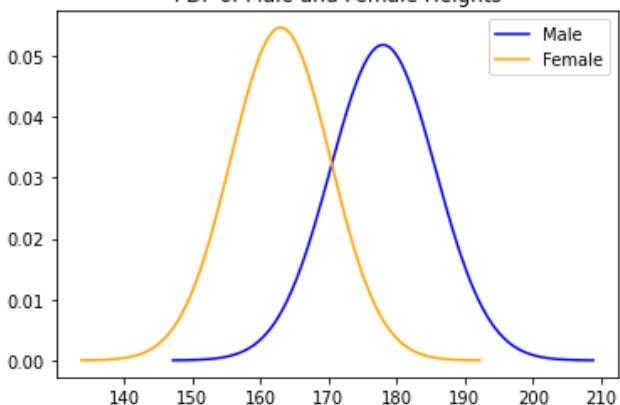
# Plot PDFs for both distributions to visualize the effect size
xs, ys = evaluate_PDF(male_height)
plt.plot(xs, ys, label='Male', color='blue')

xs, ys = evaluate_PDF(female_height)
plt.plot(xs, ys, label='Female', color='orange')
plt.xlabel('Height (cm)')
plt.legend()
plt.title("PDF of Male and Female Heights")
plt.show()

```

Python

PDF of Male and Female Heights



# Interpretations of Results

## Shape of Distributions

- **Distribution:** Bell-shaped and normal.
- **Males:** Generally taller (mean: **178 cm**).
- **Females:** Generally shorter (mean: **163 cm**).

## Spread

- **Males:** More variability (SD: **7.7 cm**).
- **Females:** Less variability (SD: **7.3 cm**).

## Probability Density

- Higher peaks near the mean.
- Areas under the curves sum to **1**.

## Comparative Analysis

- Overlap shows that some females can be taller than shorter males, illustrating height variability.

### 3. **CALCULATING EFFECT SIZE (Unstandardized Effect size)**

# 3. Calculating Effect Size (Unstandardized and Standardized)

## 3.1 Unstandardized Effect Size

The difference in means provides an **unstandardized effect size**.

```
# Generate samples from the distributions
male_sample = male_height.rvs(1000)
female_sample = female_height.rvs(1000)

# Calculate the difference in means
difference_in_means = male_sample.mean() - female_sample.mean()
print(f"Difference in Means: {difference_in_means:.2f} cm")
```

[15] ... Difference in Means: 14.89 cm

Python

## Interpretation of Results

### Significance of Difference

A difference of **14.50 cm** suggests that, on average, the male sample is significantly taller than the female sample.

### Relevance

This difference can be significant in various applications, such as ergonomics, clothing design, and health assessments. For example, understanding average height differences can guide product design to ensure that items like chairs, desks, and clothing fit appropriately for different genders.

## CALCULATING EFFECT SIZE (Standardized Effect size) Cohen's d

Cohen's d is one of the most common ways to measure effect size. As an effect size, Cohen's d is typically used to represent the **magnitude of differences between two (or more) groups** on a given variable, with larger values representing a greater differentiation between the two groups on that variable.

The basic formula to calculate Cohen's d is:

$$d = \text{effect size (difference of means)} / \text{pooled standard deviation}$$

The denominator is the standardiser, and it is important to select the most appropriate one for a given dataset. The pooled standard deviation is the average spread of all data points around their group mean (not the overall mean).

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s}.$$

Cohen defined  $s$ , the pooled standard deviation, as (for two independent samples):<sup>[9]:67</sup>

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

```
# Function to compute Cohen's d
def Cohen_d(group1, group2):
    diff = group1.mean() - group2.mean()
    n1, n2 = len(group1), len(group2)
    pooled_var = ((n1 - 1) * group1.var() + (n2 - 1) * group2.var()) / (n1 + n2 - 2)
    return diff / np.sqrt(pooled_var)

# Calculate and display Cohen's d
cohen_d_value = Cohen_d(male_sample, female_sample)
print(f"Cohen's d: {cohen_d_value:.2f}")
```

```

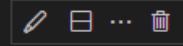
# Function to compute Cohen's d
def Cohen_d(group1, group2):
    diff = group1.mean() - group2.mean()
    n1, n2 = len(group1), len(group2)
    pooled_var = ((n1 - 1) * group1.var() + (n2 - 1) * group2.var()) / (n1 + n2 - 2)
    return diff / np.sqrt(pooled_var)

# Calculate and display Cohen's d
cohen_d_value = Cohen_d(male_sample, female_sample)
print(f"Cohen's d: {cohen_d_value:.2f}")

```

[6] ✓ 0.0s Python

... Cohen's d: 1.95



## Interpretation of Cohen's d

### Understanding Cohen's d

Cohen's d provides a measure of the effect size, which can help in understanding the practical significance of the difference between groups. The value of Cohen's d can be interpreted as follows:

- **0.2**: Small effect size
- **0.5**: Medium effect size
- **0.8**: Large effect size

A Cohen's d of **1.90** significantly exceeds the threshold for a large effect, suggesting that the difference in means between the two groups (males and females) is substantial. This indicates a strong practical significance in the difference observed, highlighting that the difference in heights is not only statistically significant but also meaningful in real-world contexts.

### Further Analysis

This measure can be used in conjunction with statistical tests (like t-tests) to provide a more comprehensive view of the data. In reports or presentations, Cohen's d can be used to substantiate claims about differences between groups, helping stakeholders understand the relevance of findings.

4. **Statistical power** - is the probability of rejecting the null hypothesis when it is false (avoiding a Type II error)

#### Factors Affecting Power:

- **Alpha ( $\alpha$ )**: The significance level. lower alpha lower power and vice versa
- **Sample Size**: Larger samples increase power.
- **Effect Size**: Larger effect sizes make it easier to detect differences.

We can visualize the power curve using `statsmodels`.

- # Plot power curves for different effect sizes

```
# import matplotlib.pyplot as plt
# from statsmodels.stats.power import TTestIndPower
power_analysis = TTestIndPower()#This initializes the TTestIndPower class, which is used to
compute statistical power for a two-sample independent t-test (e.g., comparing two groups). The
class provides methods to calculate power, effect size, sample size, and more.
effect_sizes = [0.2, 0.5, 0.8]

plt.figure(figsize=(10, 6))

nobs_range = np.arange(5, 500)

for es in effect_sizes:
    power = [power_analysis.solve_power(effect_size=es, nobs1=n, alpha=0.05) for n in
nobs_range]
    plt.plot(nobs_range, power, label=f'Effect Size: {es}')

plt.title("Power Curves for Different Effect Sizes")
plt.xlabel("Sample Size")
plt.ylabel("Power")
plt.ylim(0, 1)
plt.axhline(y=0.8, color='r', linestyle='--', label='Power = 0.8')
plt.legend()
plt.grid()
plt.show()
```

```

# Plot power curves for different effect sizes
# import matplotlib.pyplot as plt
# from statsmodels.stats.power import TTestIndPower, TTestPower
power_analysis = TTestIndPower()#This initializes the TTestIndPower class, which is used to compute statistics
effect_sizes = [0.2, 0.5, 0.8]

plt.figure(figsize=(10, 6))

nobs_range = np.arange(5, 500)

for es in effect_sizes:
    power = [power_analysis.solve_power(effect_size=es, nobs1=n, alpha=0.05) for n in nobs_range]
    plt.plot(nobs_range, power, label=f'Effect Size: {es}')

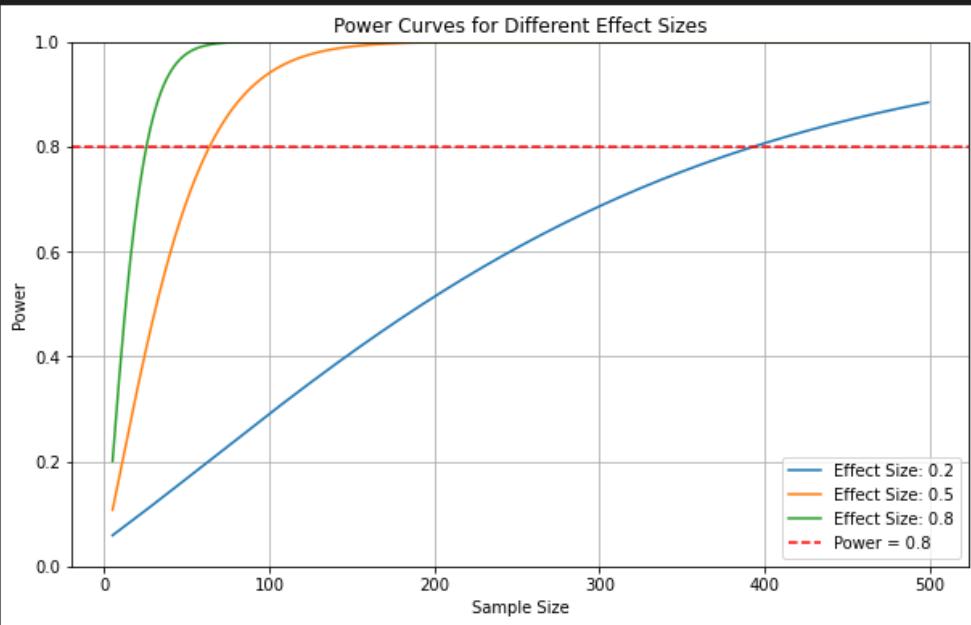
plt.title("Power Curves for Different Effect Sizes")
plt.xlabel("Sample Size")
plt.ylabel("Power")
plt.ylim(0, 1)
plt.axhline(y=0.8, color='r', linestyle='--', label='Power = 0.8')
plt.legend()
plt.grid()
plt.show()

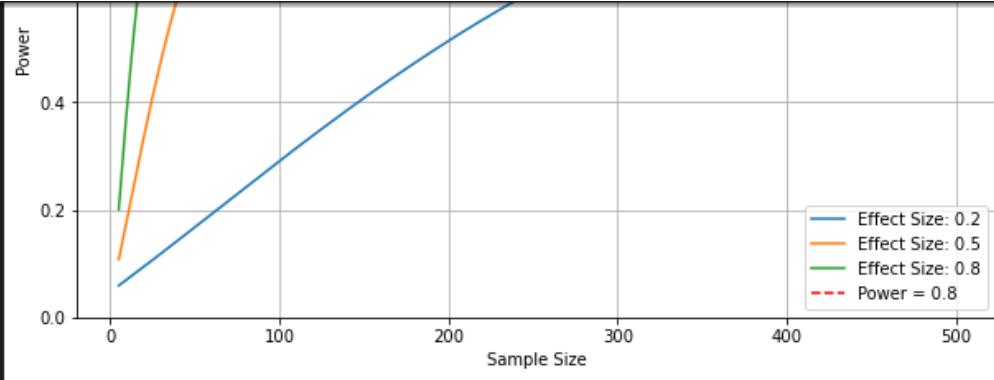
```

[8]

✓ 0.6s

Python





## Interpretation of the Power Curves

### Understanding Power

Power is the probability of correctly rejecting a false null hypothesis (avoiding a Type II error). A higher power (close to 1) indicates a greater likelihood of detecting an effect when one exists.

### Effect of Sample Size

- As expected, the curves show that increasing sample size leads to an increase in power for all effect sizes.
- For a given effect size, larger sample sizes provide more reliable estimates and increase the likelihood of detecting true effects.

### Comparison of Effect Sizes

The power curves illustrate that:

- A small effect size (0.2) requires a much larger sample size to achieve a power of 0.8 compared to medium (0.5) and large (0.8) effect sizes.
- Medium and large effect sizes achieve a power of 0.8 with fewer samples.

These observations highlight the importance of considering both effect size and sample size when designing experiments to ensure that the study is adequately powered to detect meaningful differences.

**In addition to plotting a full curve, you can also calculate specific values. Simply don't specify one of the four parameters.**

We can also draw the following conclusion from the plot above: For a specific sample size ( $x$ -axis), power increases ( $y$ -axis) as effect size increases.

Power\_analysis.solve\_power

```
# Calculate power  
power_analysis.solve_power(effect_size=.2, nobs1=80, alpha=.05)
```

```
0.24175778678474175
```

```
# Calculate sample size required  
power_analysis.solve_power(effect_size=.2, alpha=.05, power=.8)
```

```
393.4056989990348
```

```
# Calculate minimum effect size to satisfy desired alpha and power :  
power_analysis.solve_power(nobs1=25, alpha=.05, power=.8)
```

```
0.808707788668041
```

```
# Calculate alpha (less traditional)  
power_analysis.solve_power(nobs1=25, effect_size=.3, power=.8)
```

```
0.6613634273431549
```

## 5. Ethical considerations in A/B Testing

When conducting A/B tests, especially with real users, it is important to consider ethics

### Key ethical considerations

-**Informed Consent:** participants must be aware of the tests

-**Assesment of Risks and benefits-** weigh potential risks against benefits

-**Fair selection of subjects-** Avoid targeting vulnerable populations

## Example 2:A/B TESTING EXAMPLE

## The Scenario

You've been tasked with designing an experiment to test whether a new email template will be more effective for your company's marketing team. The current template has a 5% response rate (with standard deviation .0475), which has outperformed numerous other templates in the past. The company is excited to test the new design that was developed internally but nervous about losing sales if it is not to work out. As a result, they are looking to determine how many individuals they will need to serve the new email template in order to detect a 1% performance increase.

**NULL:** The probability of success of the new email template is < 0.06

**Alternative:** The probability of success for the new email template >=0.06

a) Now define what alpha and beta you believe might be appropriate for this scenario.

To start, arbitrarily set \$\alpha\$ to 0.05. From this, calculate the required sample size to detect a .01 response rate difference at a power of .8.

> Note: Be sure to calculate a normalized effect size using Cohen's d from the raw response rate

```
# Calculate the required sample size
from statsmodels.stats.power import TTestIndPower
power_analysis = TTestIndPower()
mean_difference = 0.01
std = .0475
alpha=.05
power=.8
effect_size = 0.01/std #(mean_difference/std)cohens d
sample_size =power_analysis.solve_power(effect_size =
effect_size,alpha=alpha,power=power,alternative='larger')
sample_size
```

#  **alternative='two-sided'**: The test is two-tailed, meaning it checks if the mean difference is either significantly greater than or less than zero (i.e., the means of the two groups are different in either direction).

**alternative='larger'**: The test is one-tailed, meaning it only checks if the mean of the test group is significantly *greater* than the mean of the control group.

**alternative='smaller'**: The test is also one-tailed, but it checks if the mean of the test group is significantly *less* than the mean of the control group.

```

# Calculate the required sample size
from statsmodels.stats.power import TTestIndPower
power_analysis = TTestIndPower()
mean_difference = 0.01
std = .0475
alpha=.05
power=.8
effect_size = 0.01/std #(mean_difference/std)cohens d
sample_size = power_analysis.solve_power(effect_size = effect_size, alpha=alpha, power=power, alternative='larger')
sample_size

✓ 0.0s
279.6667468021841

```

### b). Step 4: Plot Power Curves for Alternative Experiment Formulations

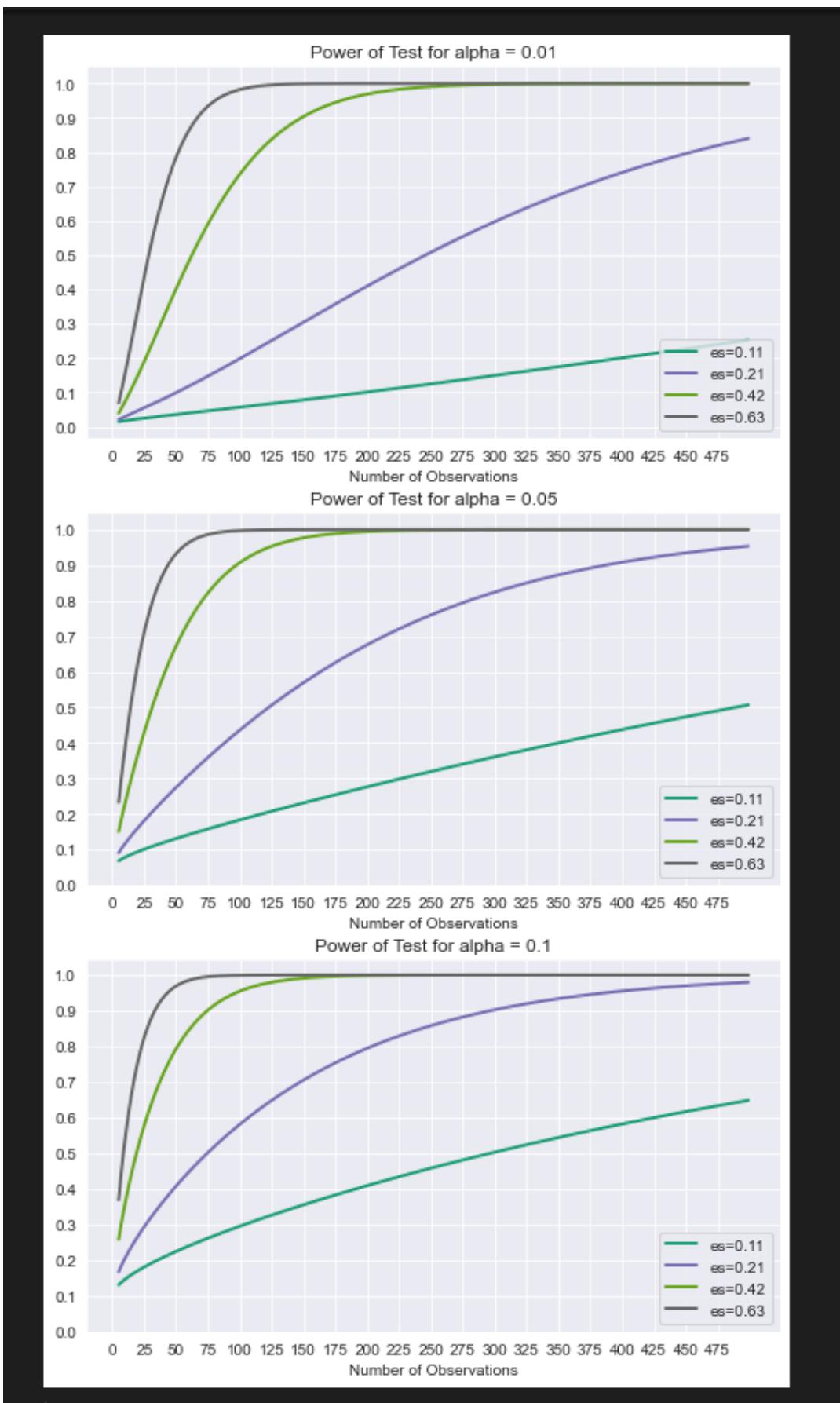
While you now know how many observations you need in order to run a t-test for the given formulation above, it is worth exploring what sample sizes would be required for alternative test formulations. For example, how much does the required sample size increase if you put the more stringent criteria of alpha=.01? Or what is the sample size required to detect a .03 response rate difference at the same alpha and power thresholds? To investigate this, plot power vs sample size curves for alpha values of .01, .05 and .1 along with varying response rate differences of .005, .01, .02 and .03.

```

#Your code; plot power curves for the various alpha and effect size combinations
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline

sd = 0.0475
e_sizes = [mu_delta/sd for mu_delta in [.005,.01,.02,.03]]
fig, axes = plt.subplots(ncols=1, nrows=3, figsize=(8,15))
for n, alpha in enumerate([.01, .05, .1]):
    print(type(n), alpha)
    ax = axes[n]
    power_analysis.plot_power(dep_var="nobs",
                               nobs = np.array(range(5,500)),
                               effect_size=e_sizes,
                               alpha=alpha,
                               ax=ax,
                               alternative='larger')
    ax.set_title('Power of Test for alpha = {}'.format(alpha))
    ax.set_xticks(list(range(0,500,25)))
    ax.set_yticks(np.linspace(0,1,11))

```



## Step 5: Propose a Final Experimental Design

Finally, now that you've explored some of the various sample sizes required for statistical tests of varying power, effect size and type I errors, propose an experimental design to pitch to your boss and some of the accompanying advantages or disadvantages with it.

Your answer here

```
"""
Answers will vary. It seems that a minimum sample size 100,
to detect all but the largest effect sizes with a reasonable balance of alpha and power.
After the initial roll-out, there should be sufficient evidence to determine whether
further investigation is warranted.
"""

[6]  ✓ 0.0s                                     Python
...  '\nAnswers will vary. It seems that a minimum sample size 100, \nto detect all but the largest effect sizes wi
```

## Summary

In this lab, you practiced designing an initial experiment and then refined the parameters of the experiment based on an initial sample to determine feasibility.

**76. CHI SQUARE-** non-parametric test for analyzing frequencies in categorical data.it helps determine if there is a significant association between categorical variables.

- Unlike t-tests, which examine population parameters(e,g mean), chi square tests focus on observed vs expected frequencies, making them ideal for categorical data
- parametric eg mean compare with pop mean,std, variability egt-test,ztest
- non\_parametric- checking correlation no paramters eg does gender affect employment to military,,nothing like mean here\*\*

### Comparing t-tests and Chi-square Tests

- T-tests are parametric tests used for continuous variables,typically analyzing sample means to test hypotheses about population parameters.A common use is comparing means between two groups, using the t-distribution for significance testing.
- In contrast , chi-square tests are non-parametric and focus on categorical data, examining the association between variables rather than means.This distinction makes each tests suitable for different types of data analysis.
- **T-test example**

```
# Generate sample data
sample_data = stats.norm.rvs(loc=20, scale=2, size=50, random_state=5)

# Perform one-sample t-test with a hypothetical mean of 21
result = stats.ttest_1samp(sample_data, 21)
print(result)

Ttest_1sampResult(statistic=-3.335711380689097, pvalue=0.001628519936938842)
```

## Chi-square Test Introduction

Chi-square tests is applicable for discrete variables that can be represented by a **Probability mass function**, which allows us to understand the data in terms of the **frequencies** of each outcome. here are several different kinds of chi-square tests depending on the question being asked, but we'll focus on *Pearson's chi-square test* and how it is applied for goodness of fit, independence, and homogeneity.

**Goodness of Fit:** compare observed frequencies to expected frequencies under a theoretical distribution.

**Test for Independence** – tests if two categorical variables are independent of each other.

**Test for Homogeneity:** Compares distribution frequencies across different groups

### **1. Chi-Square Test for Goodness of fit**

We compare observed frequencies in data to expected frequencies under a specific hypothesis.eg we could test if a coin is fair by comparing observed counts of heads and tail

## EXAMPLE:CHI SQUARE

## Example: Super Bowl Coin Toss Outcomes

Assume a fair coin toss should produce heads and tails equally often. We analyze data from Super Bowl coin tosses and perform a chi-square test for goodness of fit.

### Hypotheses for the chi-square test

- Null Hypothesis  $H_0$ : The coin toss is fair (50% heads, 50% tails).
- Alternative Hypothesis  $H_1$ : The coin toss is not fair (unequal outcomes).

1. **Observed Frequencies:** Count the actual outcomes of heads and tails from the coin tosses.

2. **Expected Frequencies:** For a fair coin, we would expect an equal number of heads and tails.

3. **Chi-Square Calculation:** Use the formula to determine the chi-square statistic:

$$\chi^2 = \frac{\sum(O_i - E_i)^2}{E_i}$$

- where  $O_i$  is the observed frequency
- $E_i$  is the expected frequency.

4. **Interpretation:** Compare the calculated chi-square value to the critical value from the chi-square distribution table based on the degrees of freedom. If the calculated value exceeds the critical value, we reject the null hypothesis, suggesting that the coin may not be fair.

```
import pandas as pd
from scipy import stats

# Load Super Bowl coin toss data
sb_data = pd.read_csv("superbowl.csv")
coin_toss_counts = sb_data["Coin Toss Outcome"].value_counts()
print('coin_toss_counts:', coin_toss_counts)

# Observed and expected frequencies
observed = coin_toss_counts.values
print('observed:', observed)
expected = [sum(coin_toss_counts) / 2] * 2 # Assuming a fair coin
print('expected:', expected)

# Perform chi-square test
result = stats.chisquare(observed, expected)

# Print the result
print(result)
```

```

import pandas as pd
from scipy import stats

# Load Super Bowl coin toss data
sb_data = pd.read_csv("superbowl.csv")
coin_toss_counts = sb_data["Coin Toss Outcome"].value_counts()
print('coin_toss_counts:', coin_toss_counts)

# Observed and expected frequencies
observed = coin_toss_counts.values
print('observed:', observed)
expected = [sum(coin_toss_counts) / 2] * 2 # Assuming a fair coin
print('expected:', expected)

# Perform chi-square test
result = stats.chisquare(observed, expected)

# Print the result
print(result)

```

✓ 0.0s Python

```

coin_toss_counts: Tails    29
Heads    26
Name: Coin Toss Outcome, dtype: int64
observed: [29 26]
expected: [27.5, 27.5]
Power_divergenceResult(statistic=0.16363636363636364, pvalue=0.6858304344516056)

```

```

sb_data.head()
# sum(sb_data["Coin Toss Outcome"].value_counts()) / 2 * 2
# expected = [sum(coin_toss_counts) / 2] * 2 # Assuming a fair coin
# expected

```

Python

| Super Bowl | Coin Toss Outcome | Coin Toss Winner | Game Winner |
|------------|-------------------|------------------|-------------|
| 0          | 1                 | Heads            | Home Team   |
| 1          | 2                 | Tails            | Home Team   |
| 2          | 3                 | Heads            | Home Team   |
| 3          | 4                 | Tails            | Away Team   |
| 4          | 5                 | Tails            | Home Team   |

```
[28] # expected
...
    Super Bowl  Coin Toss Outcome  Coin Toss Winner  Game Winner
0            1           Heads     Home Team     Home Team
1            2          Tails     Home Team   Away Team
2            3           Heads     Home Team     Home Team
3            4          Tails   Away Team     Home Team
4            5          Tails     Home Team   Away Team
```

```
[4] # Significance level (alpha)
alpha = 0.05

# Decision based on p-value
if result.pvalue < alpha:
    print("Reject the null hypothesis: The coin toss is not fair.")
else:
    print("Fail to reject the null hypothesis: The coin toss is fair.")
##chi square 55,0.05- around 70 so 0.16< 70 so we fail to reject
```

[4] Fail to reject the null hypothesis: The coin toss is fair.

## Conclusion

1. **Chi-Square Statistic:** 0.1636
2. **p-value:** 0.6858

Based on the results of the Chi-Square Test for Goodness of Fit, the p-value (0.6858) is significantly greater than the common significance level of 0.05. Therefore, we **fail to reject the null hypothesis**.

This suggests that there is no significant evidence to conclude that the coin toss outcomes deviate from what we would expect from a fair coin (50% heads and 50% tails). In other words, the data supports the claim that the coin is fair.

2. **Chi- Square Test for Independence** – tests whether two categorical variables are independent. e.g. we might examine whether winning a Super Bowl game is related to winning the coin toss.  
**NULL:** winning the super Bowl game is independent of winning the coin toss.  
**Alternative:** winning the Super Bowl game is not independent of winning the coin toss

### Example: Super Bowl Game and Coin Toss Winners

```
# Create contingency table
independence_table = pd.crosstab(sb_data["Coin Toss Winner"], sb_data["Game Winner"])
```

```
# Perform chi-square test for independence
chi2, p, dof, ex = stats.chi2_contingency(independence_table)
print("Chi-square statistic:", chi2)
print("p-value:", p)
##chi square 55,0.05- around 70 so 0.59< 70 so we fail to reject
```

```
# Create contingency table
independence_table = pd.crosstab(sb_data["Coin Toss Winner"], sb_data["Game Winner"])

# Perform chi-square test for independence
chi2, p, dof, ex = stats.chi2_contingency(independence_table)
print("Chi-square statistic:", chi2)
print("p-value:", p)
##chi square 55,0.05- around 70 so 0.59< 70 so we fail to reject
```

Python

```
Chi-square statistic: 0.5920138888888885
p-value: 0.44164141533080714
```

```
independence_table
#stats.chi2_contingency(independence_table)
```

Python

| Game Winner             | Away Team | Home Team |
|-------------------------|-----------|-----------|
| <b>Coin Toss Winner</b> |           |           |
| Away Team               | 15        | 15        |
| Home Team               | 16        | 9         |

```
sb_data['Coin Toss Winner'].value_counts()
```

Python

```
Away Team    30
Home Team    25
Name: Coin Toss Winner, dtype: int64
```

```
sb_data['Game Winner'].value_counts()
```

Python

```
Away Team    31
Home Team    24
Name: Game Winner, dtype: int64
```

# Interpretation

- **Chi-square Statistic:** 0.5920
- **p-value:** 0.4416

# Conclusion

- **Comparison with Alpha (0.05):** Since the p-value (0.4416) is greater than the significance level ( $\alpha = 0.05$ ), you fail to reject the null hypothesis.
- **Final Decision:** "Winning the Super Bowl game is independent of winning the coin toss." This suggests that there is no significant relationship between the outcomes of the coin toss and the results of the Super Bowl games based on the data you analyzed.

3. **Chi-square Test for Homogeneity** – compares the distributions of categorical variable across multiple groups. It answers questions like whether winning the NFL games differ before and after a specific period.

## EXAMPLE: NFL Home-Field advantage Before and After 2020

- **Null Hypothesis (H0):** The distribution of NFL game winners is the same before (2002-2019) and after (2020 onward) the specified period. (no homogeneity)
- **Alternative Hypothesis (H1):** The distribution of NFL game winners differs between the two periods (before and after 2020). (heterogeneous)

## #### How

1. **Contingency Table:** We create a table of game winners across the two periods.
2. **Chi-Square Test:** Calculates if observed winner distributions deviate from expected distributions (assuming no change over time).
3. **Interpret p-value:**
  - Low p-value ( $< 0.05$ ) → Significant change in winning patterns.
  - High p-value → No significant change.

```
nfl_data = pd.read_csv('nfl_games.csv')
nfl_data.head()
```

```
# Create Game Winner column
nfl_data["Game Winner"] = nfl_data.apply(lambda row: row["team1"]
   if row["result1"] == 1.0 else row["team2"], axis=1)
nfl_data.head()
```

```
nfl_data = pd.read_csv('nfl_games.csv')
nfl_data.head()
```

Python

|   | date       | season | neutral | playoff | team1 | team2 | elo1     | elo2     | elo_prob1 | score1 | score2 | result1 |
|---|------------|--------|---------|---------|-------|-------|----------|----------|-----------|--------|--------|---------|
| 0 | 1920-09-26 | 1920   | 0       | 0       | RII   | STP   | 1503.947 | 1300.000 | 0.824651  | 48     | 0      | 1.0     |
| 1 | 1920-10-03 | 1920   | 0       | 0       | AKR   | WHE   | 1503.420 | 1300.000 | 0.824212  | 43     | 0      | 1.0     |
| 2 | 1920-10-03 | 1920   | 0       | 0       | RCH   | ABU   | 1503.420 | 1300.000 | 0.824212  | 10     | 0      | 1.0     |
| 3 | 1920-10-03 | 1920   | 0       | 0       | DAY   | COL   | 1493.002 | 1504.908 | 0.575819  | 14     | 0      | 1.0     |
| 4 | 1920-10-03 | 1920   | 0       | 0       | RII   | MUN   | 1516.108 | 1478.004 | 0.644171  | 45     | 0      | 1.0     |

```
# Create Game Winner column
nfl_data["Game Winner"] = nfl_data.apply(lambda row: row["team1"]
   if row["result1"] == 1.0 else row["team2"], axis=1)
nfl_data.head()
```

Python

|   | date       | season | neutral | playoff | team1 | team2 | elo1     | elo2     | elo_prob1 | score1 | score2 | result1 | Game Winner |
|---|------------|--------|---------|---------|-------|-------|----------|----------|-----------|--------|--------|---------|-------------|
| 0 | 1920-09-26 | 1920   | 0       | 0       | RII   | STP   | 1503.947 | 1300.000 | 0.824651  | 48     | 0      | 1.0     | RII         |
| 1 | 1920-10-03 | 1920   | 0       | 0       | AKR   | WHE   | 1503.420 | 1300.000 | 0.824212  | 43     | 0      | 1.0     | AKR         |
| 2 | 1920-10-03 | 1920   | 0       | 0       | RCH   | ABU   | 1503.420 | 1300.000 | 0.824212  | 10     | 0      | 1.0     | RCH         |
| 3 | 1920-10-03 | 1920   | 0       | 0       | DAY   | COL   | 1493.002 | 1504.908 | 0.575819  | 14     | 0      | 1.0     | DAY         |
| 4 | 1920-10-03 | 1920   | 0       | 0       | RII   | MUN   | 1516.108 | 1478.004 | 0.644171  | 45     | 0      | 1.0     | RII         |

This code adds a "Game Winner" column to nfl\_data, labeling each game with the winning team:

- If result1 == 1.0, team1 won, so "Game Winner" is set to team1.
- Otherwise, it assigns team2 as the winner.
- This column is useful for analyzing team performance over time.

```
[38] # Subset data for analysis
nfl_data_subset = nfl_data[nfl_data["season"] >= 2002].copy() # Use .copy() to avoid SettingWithCopyWarning
nfl_data_subset.head()
```

Python

|       |            | date | season | neutral | playoff | team1 | team2    | elo1     | elo2     | elo_prob1 | score1 | score2 | result1 | G<br>Win |
|-------|------------|------|--------|---------|---------|-------|----------|----------|----------|-----------|--------|--------|---------|----------|
| 11735 | 2002-09-05 | 2002 | 0      | 0       | NYG     | SF    | 1485.669 | 1561.242 | 0.484789 | 13        | 16     | 0.0    |         |          |
| 11736 | 2002-09-08 | 2002 | 0      | 0       | CLE     | KC    | 1446.242 | 1475.100 | 0.551826 | 39        | 40     | 0.0    |         |          |
| 11737 | 2002-09-08 | 2002 | 0      | 0       | CHI     | MIN   | 1565.787 | 1452.964 | 0.735679 | 27        | 23     | 1.0    |         |          |
| 11738 | 2002-09-08 | 2002 | 0      | 0       | CAR     | BAL   | 1370.952 | 1572.940 | 0.312477 | 10        | 7      | 1.0    |         |          |
| 11739 | 2002-09-08 | 2002 | 0      | 0       | CIN     | LAC   | 1417.743 | 1407.844 | 0.606149 | 6         | 34     | 0.0    |         |          |

```
nfl_data_subset.loc[nfl_data_subset["season"] <= 2019, "Period"] = "Before"
nfl_data_subset.loc[nfl_data_subset["season"] > 2019, "Period"] = "After"
```

```
nfl_data_subset.head()
```

Python

|       |            | date | season | neutral | playoff | team1 | team2    | elo1     | elo2     | elo_prob1 | score1 | score2 | result1 | G<br>Win |
|-------|------------|------|--------|---------|---------|-------|----------|----------|----------|-----------|--------|--------|---------|----------|
| 11735 | 2002-09-05 | 2002 | 0      | 0       | NYG     | SF    | 1485.669 | 1561.242 | 0.484789 | 13        | 16     | 0.0    |         |          |
| 11736 | 2002-09-08 | 2002 | 0      | 0       | CLE     | KC    | 1446.242 | 1475.100 | 0.551826 | 39        | 40     | 0.0    |         |          |
| 11737 | 2002-09-08 | 2002 | 0      | 0       | CHI     | MIN   | 1565.787 | 1452.964 | 0.735679 | 27        | 23     | 1.0    |         |          |
| 11738 | 2002-09-08 | 2002 | 0      | 0       | CAR     | BAL   | 1370.952 | 1572.940 | 0.312477 | 10        | 7      | 1.0    |         |          |
| 11739 | 2002-09-08 | 2002 | 0      | 0       | CIN     | LAC   | 1417.743 | 1407.844 | 0.606149 | 6         | 34     | 0.0    |         |          |

```
# Create contingency table
homogeneity_table = pd.crosstab(nfl_data_subset["Period"], nfl_data_subset["Game Winner"])
homogeneity_table
# Perform chi-square test for homogeneity
chi2, p, dof, ex = stats.chi2_contingency(homogeneity_table)
print("Chi-square statistic:", chi2)
print("p-value:", p)
```

```
# Create contingency table
homogeneity_table = pd.crosstab(nfl_data_subset["Period"], nfl_data_subset["Game Winner"])
homogeneity_table
```

0] Python

| Game Winner | ARI | ATL | BAL | BUF | CAR | CHI | CIN | CLE | DAL | DEN | ... | NYG | NYJ | OAK | PHI | PIT | SEA | SF  | TB  |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Period      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| After       | 8   | 4   | 12  | 15  | 5   | 8   | 5   | 12  | 6   | 5   | ... | 6   | 2   | 8   | 4   | 12  | 12  | 6   | 15  |
| Before      | 132 | 160 | 178 | 126 | 158 | 142 | 135 | 89  | 161 | 170 | ... | 147 | 135 | 105 | 180 | 198 | 182 | 138 | 122 |

2 rows × 32 columns

1] Python

```
# Perform chi-square test for homogeneity
chi2, p, dof, ex = stats.chi2_contingency(homogeneity_table)
print("Chi-square statistic:", chi2)
print("p-value:", p)
```

1] Chi-square statistic: 65.73475199088233  
p-value: 0.00026989544985461225

## Conclusions

1. **Chi-Square Statistic:** 65.73
2. **p-value:** 0.00027

The p-value (0.00027) is significantly less than the significance level ( $\alpha = 0.05$ ).

## Final Decision

**Reject the Null Hypothesis ( $H_0$ ):** The distribution of NFL game winners differs significantly between the periods before and after 2020. This suggests that there has been a notable change in game outcomes, possibly indicating variations in home-field advantage or other influencing factors affecting game results during these periods.

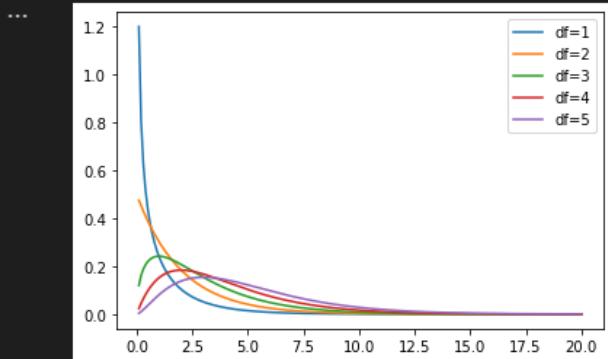
# Visualizing Chi-Square Distribution

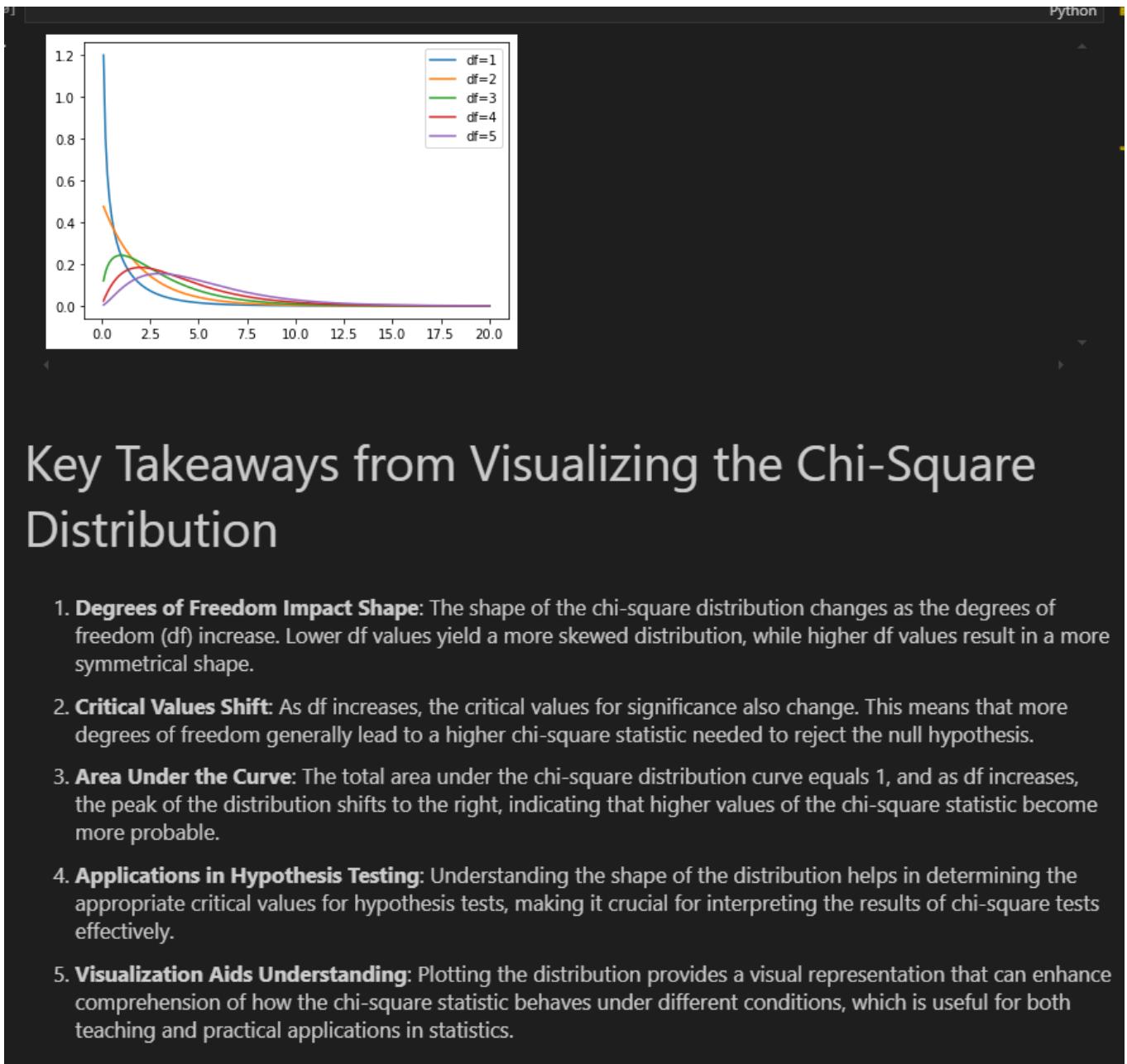
The shape of the chi-square distribution varies based on degrees of freedom (df), which are determined by the number of categories minus one.

```
# Plot chi-square distribution for various degrees of freedom
fig, ax = plt.subplots()
x = np.linspace(0.1, 20, 200)
for df in range(1, 6):
    ax.plot(x, stats.chi2.pdf(x, df), label=f'df={df}')
ax.legend()
plt.show()
```

[48]

Python





# Real-Life Examples of Chi-Square Applications with Test Types

## 1. Market Research:

- **Test Type:** Chi-Square Test for Independence
- Companies use chi-square tests to analyze consumer preferences. They survey customers about their preferred product features and use chi-square tests to determine if preferences vary by demographic groups like age or gender.

## 2. Medical Studies:

- **Test Type:** Chi-Square Test for Independence
- In clinical trials, researchers may use chi-square tests to determine if there is an association between treatment types and patient outcomes (e.g., recovery vs. no recovery). This helps assess whether the treatment effectiveness differs among demographic groups.

## 3. Election Polls:

- **Test Type:** Chi-Square Test for Independence
- Political analysts often use chi-square tests to evaluate whether voting patterns are independent of demographic factors such as age, education, or ethnicity, aiding in understanding how different groups vote.

## 4. Sports Analytics:

- **Test Type:** Chi-Square Test for Independence
- Sports analysts might use chi-square tests to evaluate whether winning rates are independent of home-field advantage. For example, they can analyze win-loss records of teams before and after changes in stadium capacity or crowd presence.

## 5. Education Research:

- **Test Type:** Chi-Square Test for Independence
- Educators might use chi-square tests to analyze test scores across different teaching methods or curricula to see if there are significant differences in outcomes among groups of students.

## 6. Quality Control:

- **Test Type:** Chi-Square Test for Homogeneity
- In manufacturing, companies can use chi-square tests to determine if defect rates differ across different production lines or shifts, helping identify areas that need improvement.

vote.

#### 4. Sports Analytics:

- **Test Type:** Chi-Square Test for Independence
- Sports analysts might use chi-square tests to evaluate whether winning rates are independent of home-field advantage. For example, they can analyze win-loss records of teams before and after changes in stadium capacity or crowd presence.

#### 5. Education Research:

- **Test Type:** Chi-Square Test for Independence
- Educators might use chi-square tests to analyze test scores across different teaching methods or curricula to see if there are significant differences in outcomes among groups of students.

#### 6. Quality Control:

- **Test Type:** Chi-Square Test for Homogeneity
- In manufacturing, companies can use chi-square tests to determine if defect rates differ across different production lines or shifts, helping identify areas that need improvement.

#### 7. Genetics:

- **Test Type:** Chi-Square Goodness of Fit Test
- In genetics studies, chi-square tests are used to determine if observed genetic trait distributions fit expected Mendelian ratios, helping researchers understand inheritance patterns.

#### 8. Customer Satisfaction Surveys:

- **Test Type:** Chi-Square Test for Homogeneity
- Businesses can analyze survey results to see if satisfaction levels differ by customer segments, allowing for targeted improvements in service or product offerings.

#### 9. Social Science Research:

- **Test Type:** Chi-Square Test for Independence
- Researchers may study social behaviors, such as whether the frequency of certain social activities (e.g., attending events) is independent of factors like income level or education.

#### 10. Public Health Studies:

- **Test Type:** Chi-Square Test for Independence
- Chi-square tests can be applied to examine relationships between health behaviors (e.g., smoking status) and health outcomes (e.g., incidence of lung disease) across different population groups.

### EXAMPLE1:: A/B TEST AND CHI SQUARE

We've been hired on by a company looking to see if they can change out their UI to get more website visitors to create an account.

Their innovative idea? Modify their sign-up button from pink to slightly more pink!

They've tasked us to figure out if it's worth them making the change. They say their developers really don't want to put in the effort unless we're confident it has an effect.

The company says if they have an absolute increase in the conversion rate of **\*\*just 2%**, it'd be worth making the change for the whole site! there's an **8\% chance** a visitor viewing the page will sign-up.

### What was decided:

If we want to see an increase of \$2\%\$ from \$8\%\$ and we choose a typical power \$0.8\$ and a conservative alpha=0.01, we can do a power analysis to find the minimum number of samples needed is about 4,700 samples.

- Since we're running for just a month and we have about 40,000 visitors per day, we probably can sample a decent number of visitors without changing a lot of visitor's UI.
- We'll have two groups; a control group that will have no change and an experiment group that will have the updated sign-up button.

```
from statsmodels.stats.power import TTestIndPower
import statsmodels.stats.api as sms
power_analysis = TTestIndPower()
# n=4700
# m1=0.08
# m2=0.1
# p=m2
# std =np.sqrt(n*p*(1-p))
# print(std)
# d = (m1*n-m2*n)/std
# print(d)

effect_size = sms.proportion_effectsize(0.08, 0.1)
power_analysis.solve_power(alpha=0.01,power=.8, effect_size=effect_size)
```

```
from statsmodels.stats.power import TTestIndPower
import statsmodels.stats.api as sms
power_analysis = TTestIndPower()
# n=4700
# m1=0.08
# m2=0.1
# p=m2
# std =np.sqrt(n*p*(1-p))
# print(std)
# d = (m1*n-m2*n)/std
# print(d)

effect_size = sms.proportion_effectsize(0.08, 0.1)
power_analysis.solve_power(alpha=0.01,power=.8, effect_size=effect_size)
```

✓ 0.0s

4770.219081581475

# Experiment Time!

+ Code + Markdown

Let's pretend we already collected about a month's worth of data for the control and experiment groups.

We have the data aggregated in separate files for the two groups. In the file, we have a new day on each line where we recorded the number of pageviews (for the visitors assigned to a group) and the number of conversions (sign-ups).

```
[36] # Load the data  
df_control = pd.read_csv('data/control.csv')  
df_experiment = pd.read_csv('data/experiment.csv')
```

Python

```
[32] df_control.head()
```

Python

```
...  
views conversions  
0 7779 696  
1 9150 790  
2 10564 917  
3 9923 847  
4 10065 842
```

```
[38] df_experiment.head()
```

Python

```
...  
views conversions  
0 7766 716  
1 9338 815  
2 10529 914  
3 9917 857  
4 9844 862
```

## VISUALIZE OUR DATA

Lets look at the data and see if we can notice anything visually

```
f, (ax0,ax1) = plt.subplots(nrows=2, figsize=(10,6))  
  
# Views  
ax0.set_title('Views')
```

```
sns.kdeplot(data=df_control.views, ax=ax0, label='Control')
sns.kdeplot(data=df_experiment.views, ax=ax0, label='Experiment')
ax0.legend()

# Conversions
ax1.set_title('Conversions')
sns.kdeplot(data=df_control.conversions, ax=ax1, label='Control')
sns.kdeplot(data=df_experiment.conversions, ax=ax1, label='Experiment')
ax1.legend()

plt.tight_layout()
```

## Visualize Our Data

Let's look at the data and see if we can notice anything visually

```
f, (ax0,ax1) = plt.subplots(nrows=2, figsize=(10,6))

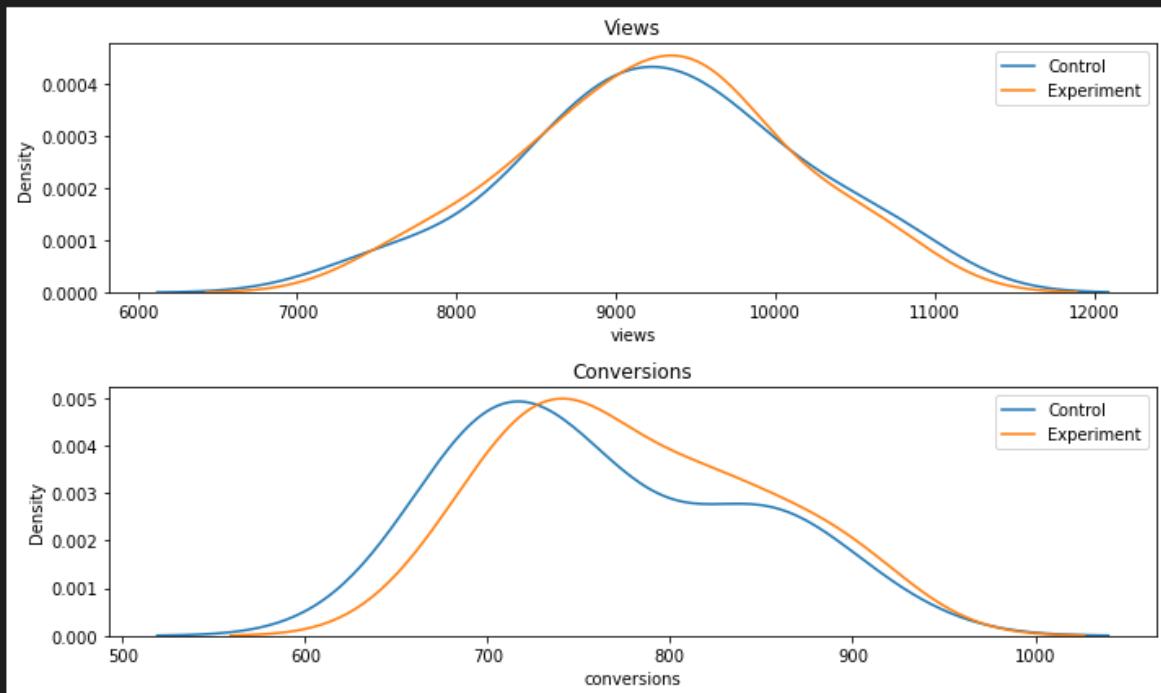
# Views
ax0.set_title('Views')
sns.kdeplot(data=df_control.views, ax=ax0, label='Control')
sns.kdeplot(data=df_experiment.views, ax=ax0, label='Experiment')
ax0.legend()

# Conversions
ax1.set_title('Conversions')
sns.kdeplot(data=df_control.conversions, ax=ax1, label='Control')
sns.kdeplot(data=df_experiment.conversions, ax=ax1, label='Experiment')
ax1.legend()

plt.tight_layout()
```

[37]

Python



## PERFORM STATISCAL TEST

Since we are looking at the frequency if ocnversions from views, we can use the chi square goodness-of -fit test.

So the first step is to get our data into a format of ‘observed’ (experiment) vs ‘expected’ (control)

```
# We'll just sum all the days together to see an overall change
control_views = sum(df_control.views)
control_conv = sum(df_control.conversions)
```

```
experiment_views = sum(df_experiment.views)
experiment_conv = sum(df_experiment.conversions)
print('control_views',control_views)
print('control_conv',control_conv)
print('experiment_views',experiment_views)
print('experiment_conv',experiment_conv)
```

## Perform Statistical test

+ Code + Markdown

Since we are looking at the **frequency of conversions from views**, we can use the  $\chi^2$  goodness-of-fit test.

So the first step is to get our data into a format of "observed" (experiment) vs "expected" (control)

```
# We'll just sum all the days together to see an overall change
control_views = sum(df_control.views)
control_conv = sum(df_control.conversions)

experiment_views = sum(df_experiment.views)
experiment_conv = sum(df_experiment.conversions)
print('control_views',control_views)
print('control_conv',control_conv)
print('experiment_views',experiment_views)
print('experiment_conv',experiment_conv)
```

Python

```
control_views 213327
control_conv 17531
experiment_views 212514
experiment_conv 17950
```

```
# This should be "converted" and "not converted"
observations = np.array([experiment_conv, experiment_views - experiment_conv])

expectations = np.array([control_conv, control_views - control_conv])

print('OBSERVED (experiment):', observations)
print('EXPECTED (control):', expectations)
```

Python

```
OBSERVED (experiment): [ 17950 194564]
EXPECTED (control): [ 17531 195796]
```

```
stats.chisquare(f_obs=observations, f_exp=expectations)
```

Python

```
Power_divergenceResult(statistic=17.766385974161416, pvalue=2.4975705608443254e-05)
```

```
- # This should be "converted" and "not converted"
- observations = np.array([experiment_conv, experiment_views - experiment_conv])
-
- expectations = np.array([control_conv, control_views - control_conv])
-
- print('OBSERVED (experiment):', observations)
- print('EXPECTED (control):', expectations)
- stats.chisquare(f_obs=observations, f_exp=expectations)
```

Is it statistically significant - would it be worth making the change based on the observed effect?

```
experiment_percent = experiment_conv/experiment_views*100
print(f'Percent Experiment Converted: {experiment_percent:.5}%')
control_percent = control_conv/control_views*100
print(f'Percent Control Converted: {control_percent:.5}%')

print(f'Difference between experiment & control {experiment_percent-
control_percent:.3}%')
```

Is it statistically significant?

Would it be worth making the change based on the observed effect?

```
experiment_percent = experiment_conv/experiment_views*100
print(f'Percent Experiment Converted: {experiment_percent:.5}%')
control_percent = control_conv/control_views*100
print(f'Percent Control Converted: {control_percent:.5}%')

print(f'Difference between experiment & control {experiment_percent-control_percent:.3}%')
13] ✓ 0.0s Python
.. Percent Experiment Converted: 8.4465%
Percent Control Converted: 8.2179%
Difference between experiment & control 0.229%
```



## Conclusion?

We got a significant result with 99% confidence! But although we're certain the effect isn't large enough for the company to make the change (at least based on what they told us).

We might break the news like this:

*We're very confident that there was an observable effect in conversions by changing the buttons color. However, the observed effect was smaller than what was stated to make the change site-wide valuable.*

*The difference in button color was observed to increase sign-ups by an absolute amount of about 0.2%. Perhaps this change can still be made valuable since we are confident that the effect was real.*

## USING FISHERS TEST

At this point, you likely can see how this can be used for the above example.

```
[17] # We'll use our observations (experiment group) & expectations (control group) as  
# defined earlier in the lecture  
contingency_table = np.array([observations, expectations])  
contingency_table  
[17]: ✓ 0.0s Python  
... array([[ 17950, 194564],  
       [ 17531, 195796]])
```

Using SciPy's function:

```
[18] # Note the slowness of the method (due to large factorials)  
result = stats.fisher_exact(contingency_table)  
_, p = result  
result  
[18]: ✓ 2.3s Python  
... (1.0303839663755003, 0.007041963060184108)  
  
p  
[16] ✓ 0.0s Python  
... 0.007041963060184108
```

We still find that we still find a significant result at our significance level.

## EXAMPLE2:: A/B TEST AND CHI SQUARE

*We have data about whether customers completed sales transactions, segregated by the type of ad banners to which the customers were exposed.*

*The question we want to answer is whether there was any difference in sales "conversions" between desktop customers who saw the sneakers banner and desktop customers who saw the accessories banner in the month of May 2019.*

What would we need to consider when designing our experiment?  
68@gmail.com) is signed in

Might include:

- Who is it that we're including in our test?
- How big of an effect would make it "worth" us seeing?
  - This can affect sample size
  - This can give context of a statistically significant result
- Other biases or "gotchas"

## Loading the Data

First let's download the data from [kaggle](#) via the release page of this repo: [https://github.com/flatiron-school/ds-ab\\_testing/releases](https://github.com/flatiron-school/ds-ab_testing/releases)

The code below will load it into our DataFrame:

```
[4] !l download the data from online so it can take some time (but relatively small download)
!ad_csv('https://github.com/flatiron-school/ds-ab_testing/releases/download/v1.2/products_small.csv')
# download the data from online so it can take some time (but relatively small download)
!_csv('data/products_small.csv')

[4] ✓ 3.0s
```

Python

Let's take a look while we're at it

```
[5] df.head()
[5] ✓ 0.0s
```

Python

|     | user_id                    | page_id                          | product     | site_version | time                | title       | target |
|-----|----------------------------|----------------------------------|-------------|--------------|---------------------|-------------|--------|
| ... | 967eb72656e86059ec6f208092 | 2fdc16a09e0016555dd4da4a3fe84414 | accessories | desktop      | 2019-03-06 08:42:47 | banner_show | 0      |
|     | ab1f004eabba25aacb7f0e5484 | 6b0a902b9b73d5a158d0119d6feb38ac | sneakers    | mobile       | 2019-04-19 10:50:15 | banner_show | 0      |

# Some Exploration to Better Understand our Data

Lets's look at the different banner types:

```
[7] df['product'].value_counts() Python
✓ 0.0s
```

```
... clothes 210996
company 203020
sneakers 201298
sports_nutrition 193200
accessories 191486
Name: product, dtype: int64
```

```
[38] df.groupby('product')['target'].value_counts() Python
✓ 0.2s
```

```
... product target
accessories 0 186127
             1 5359
clothes 0 197720
         1 13276
company 0 203020
sneakers 0 193375
         1 7923
sports_nutrition 0 190363
                  1 2837
Name: target, dtype: int64
```

Let's look at the range of time-stamps on these data:

```
[39] df['time'].min() Python
✓ 0.0s
... '2019-01-01 00:00:25'
```

```
[40] df['time'].max() Python
✓ 0.0s
```

```

... 2019-01-01 00:00:25

    df['time'].max()
[48] ✓ 0.0s
... '2019-05-31 23:59:21'

Python

```

Let's check the counts of the different site\_version values:

```

    df['site_version'].value_counts()
[11] ✓ 0.0s
... mobile      718521
desktop     281479
Name: site_version, dtype: int64

Python

```

```

    df['title'].value_counts()
[12] ✓ 0.0s
... banner_show      872275
banner_click      98330
order            29395
Name: title, dtype: int64

Python

```

```

    df.groupby('title').agg({'target': 'mean'})
[13] ✓ 0.0s
...      target
       title
       banner_click    0.0
       banner_show     0.0
       order          1.0

Python

```

**EXPERIMENTAL SETUP:** We need to filter by site\_version, time, and product:

```

df_AB = df[(df['site_version'] == 'desktop') &
            (df['time'] >= '2019-05-01') &
            ((df['product'] == 'accessories') | (df['product'] == 'sneakers'))].reset_index(drop
= True)
df_AB
SAME AS

```

```

df_AB2 = df.query('site_version=="desktop" & product in["accessories","sneakers"] &
time>="2019-05-01"]').reset_index(drop=True)
df_AB2.tail()

```

+ Code + Markdown

query('site\_version=="desktop" & product in["accessories","sneakers"] & time>="2019-05-01"').reset\_index(dr)

✓ 0.2s Python

|   | user_id                    | page_id                          | product     | site_version | time                | title       | target |
|---|----------------------------|----------------------------------|-------------|--------------|---------------------|-------------|--------|
| 1 | ea5798ad6bd3f5a667ae4c217a | 644ad50dd900d6735d8ceb15b71d50f8 | accessories | desktop      | 2019-05-01 23:01:05 | banner_show | 0      |
| 2 | ia4db2babcfde1c4095c805632 | a3d2de7675556553a5f08e4c88d2c228 | sneakers    | desktop      | 2019-05-26 11:19:09 | order       | 1      |
| 3 | 8941b89c4716c4f389e9b8bd7  | a3d2de7675556553a5f08e4c88d2c228 | sneakers    | desktop      | 2019-05-11 09:04:46 | order       | 1      |
| 4 | 4843c70c41c0db294ee479582  | 13f395d7c86ef276f7ce52a557fd3491 | accessories | desktop      | 2019-05-09 09:17:24 | banner_show | 0      |
| 5 | 69971fb5e08ae0316f58e3f434 | ef38d906f15db1efde867c68f1336946 | sneakers    | desktop      | 2019-05-20 17:58:16 | banner_show | 0      |

## What Test Would Make Sense?

Since we're comparing the frequency of conversions of customers who saw the "sneakers" banner against those who saw the "accessories" banner, we can use a  $\chi^2$  test.

Note there are other hypothesis tests we can use but this should be fine since it should fit our criteria.

## The Hypotheses

$H_0$ : Customers who saw the sneakers banner were no more or less likely to buy than customers who saw the accessories banner.

$H_1$ : Customers who saw the sneakers banner were more or less likely to buy than customers who saw the accessories banner.

## Setting a Threshold

We'll set a false-positive rate of  $\alpha = 0.05$ .

## $\chi^2$ Test

## Setup the Data

We need our contingency table: the numbers of people who did or did not submit orders, both for the accessories banner and the sneakers banner.

```
# We convert have two groups
df_A = df_AB[df_AB['product'] == 'accessories']
df_B = df_AB[df_AB['product'] == 'sneakers']
```

[17]

✓ 0.0s

Python

We need our contingency table: the numbers of people who did or did not submit orders, both for the accessories banner and the sneakers banner.

```
[17] # We convert have two groups
      df_A = df_AB[df_AB['product'] == 'accessories']
      df_B = df_AB[df_AB['product'] == 'sneakers']

      ✓ 0.0s
```

Python

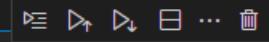
```
[18] ▶ accessories_orders = sum(df_A['target'])
      sneakers_orders = sum(df_B['target'])

      accessories_orders, sneakers_orders

      ✓ 0.0s
```

Python

... (496, 799)



To get the numbers of people who didn't submit orders, we get the total number of people who were shown banners and then subtract the numbers of people who did make orders.

```
[19] accessories_total = sum(df_A['title'] == 'banner_show')
      sneakers_total = sum(df_B['title'] == 'banner_show')

      accessories_no_orders = accessories_total - accessories_orders
      sneakers_no_orders = sneakers_total - sneakers_orders

      accessories_no_orders, sneakers_no_orders

      ✓ 0.0s
```

Python

... (11219, 11055)

```
[20] contingencies_table = np.array([
      (accessories_orders, accessories_no_orders),
      (sneakers_orders, sneakers_no_orders)
    ])

    contingencies_table

    ✓ 0.0s
```

Python

... array([[ 496, 11219],
 [ 799, 11055]])

## Calculation

```
[21] stats.chi2_contingency(contingency_table)
    ✓ 0.0s   Python
...
(70.80332433558804,
 3.946714706061366e-17,
 1,
 array([[ 643.68131868, 11071.31868132],
       [ 651.31868132, 11202.68131868]]))
```

This extremely low  $p$ -value suggests that these two groups are genuinely performing differently. In particular, the desktop customers who saw the sneakers banner in May 2019 bought at a higher rate than the desktop customers who saw the accessories banner in May 2019.

## Interpretation

```
[22] contingency_table
    ✓ 0.0s   Python
...
array([[ 496, 11219],
       [ 799, 11055]])
```

```
[23] # Find the difference in conversion rate
      accessory_CR, sneaker_CR = contingency_table[:,0]/contingency_table[:,1]
      print(accessory_CR,sneaker_CR)
    ✓ 0.0s   Python
...
0.044210713967376775 0.07227498869289914
```

```
[23] ...     array([[ 496, 11219],
   ...           [ 799, 11055]])
```

```
[23] # Find the difference in conversion rate
accessory_CR, sneaker_CR = contingency_table[:,0]/contingency_table[:,1]
print(accessory_CR,sneaker_CR)
[23] ✓ 0.0s
[23] Python
... 0.044210713967376775 0.07227498869289914
```

```
[24] print(f'Conversion Rate for accessory banner:\n\t{100*accessory_CR:.3f}%')
print(f'Conversion Rate for sneaker banner:\n\t{100*sneaker_CR:.3f}%')
print('')
print(f'Absolute difference of CR: {100*(sneaker_CR-accessory_CR):.3f}%')
[24] ✓ 0.0s
[24] Python
... Conversion Rate for accessory banner:
      4.421%
Conversion Rate for sneaker banner:
      7.227%

Absolute difference of CR: 2.806%
```

So we can say:

- There was a statistically significant difference at the alpha (confidence level)
- The difference was about 2.8% in favor of the sneaker banner!

```

import numpy as np
import scipy.stats as stats

# Contingency Table
# Rows: Gender (Male, Female)
# Columns: Likes Product, Dislikes Product
data = np.array([[30, 20],      # Male
                [25, 25]])    # Female

# Perform the Chi-Square Test of Independence
chi2, p, dof, expected = stats.chi2_contingency(data)

# Output the results
print(f"Chi-Square Statistic: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis (There is a significant association between gender \
          and product preference).")
else:
    print("Fail to reject the null hypothesis (No significant association between gender\
          and product preference).")

```

✓ 0.0s

Python

```

Chi-Square Statistic: 0.6464646464646464
P-value: 0.4213795037428696
Degrees of Freedom: 1
Expected Frequencies:
[[27.5 22.5]
 [27.5 22.5]]
Fail to reject the null hypothesis (No significant association between gender and product preference).

```

## 77. ANOVA(Analysis of Variance)

Helps determine If there are statistically significant differences between the means of three or more independent groups.

### Use Case Examples:

- Testing different teaching methods' effectiveness
- Comparing means of sales across regions
- Medical studies comparing drug effects across multiple groups

### Key Concepts in Anova

- Hypotheses:

**Null hypothesis:** all group means are equal

**Alternative hypothesis:** At least one group mean is different

- Assumptions of Anova

1. Independence of observations
2. Normally distributed groups- (can use Shapiro test here)
3. Homogeneity of variances across groups – same var-levene

## 78. EXAMPLE 1:Anova using f\_oneway and OLS

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import f_oneway, levene, shapiro
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Generate example data
np.random.seed(42)
group_A = np.random.normal(50, 10, 30)
group_B = np.random.normal(55, 10, 30)
group_C = np.random.normal(60, 10, 30)

# Combine into DataFrame
data = pd.DataFrame({
    'Score': np.concatenate([group_A, group_B, group_C]),
    'Group': ['A']*30 + ['B']*30 + ['C']*30
})
data.head(20)
```

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import f_oneway, levene, shapiro
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Generate example data
np.random.seed(42)
group_A = np.random.normal(50, 10, 30)
group_B = np.random.normal(55, 10, 30)
group_C = np.random.normal(60, 10, 30)

# Combine into DataFrame
data = pd.DataFrame({
    'Score': np.concatenate([group_A, group_B, group_C]),
    'Group': ['A']*30 + ['B']*30 + ['C']*30
})
data.head(20)

```

✓ 0.0s

Python

|    | Score     | Group |
|----|-----------|-------|
| 0  | 54.967142 | A     |
| 1  | 48.617357 | A     |
| 2  | 56.476885 | A     |
| 3  | 65.230299 | A     |
| 4  | 47.658466 | A     |
| 5  | 47.658630 | A     |
| 6  | 65.792128 | A     |
| 7  | 57.674347 | A     |
| 8  | 45.305256 | A     |
| 9  | 55.425600 | A     |
| 10 | 45.365823 | A     |
| 11 | 45.342702 | A     |
| 12 | 52.419623 | A     |
| 13 | 30.867198 | A     |
| 14 | 32.750822 | A     |

## Descriptive statistics and Visualization

```

data.groupby('Group').describe()
# Visualization
sns.boxplot(x='Group', y='Score', data=data)
plt.title("Box Plot of Scores by Group")
plt.show()

```

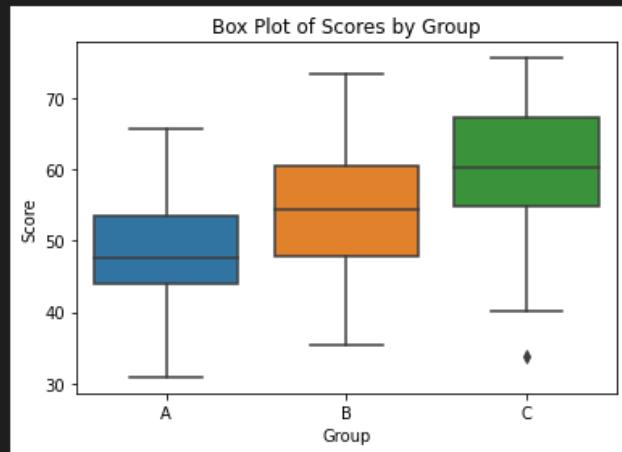
```
[2] # Descriptive Statistics  
data.groupby('Group').describe()
```

Python

| Group | Score |           |          |           |           |           |           |           |  |
|-------|-------|-----------|----------|-----------|-----------|-----------|-----------|-----------|--|
|       | count | mean      | std      | min       | 25%       | 50%       | 75%       | max       |  |
| A     | 30.0  | 48.118531 | 9.000064 | 30.867198 | 44.089491 | 47.658548 | 53.603353 | 65.792128 |  |
| B     | 30.0  | 53.788375 | 9.311022 | 35.403299 | 47.908863 | 54.354272 | 60.446618 | 73.522782 |  |
| C     | 30.0  | 60.128848 | 9.919830 | 33.802549 | 54.858581 | 60.256105 | 67.377112 | 75.646437 |  |

```
[3] # Visualization  
sns.boxplot(x='Group', y='Score', data=data)  
plt.title("Box Plot of Scores by Group")  
plt.show()
```

Python



**Interpretation:** The box plot allows us to visually inspect the distribution of scores across each group. If the medians and spreads vary significantly, it may indicate a difference in group means, which we can test with ANOVA.

Check for pvalue and statistic

```
Results= f_oneway(data[data['Group'] == 'A']['Score'],  
                  data[data['Group'] == 'B']['Score'],  
                  data[data['Group'] == 'C']['Score'])
```

Results

This also works

```

f_oneway(data.query('Group=="A"')['Score'],
         data.query('Group=="B"')['Score'],
         data.query('Group=="C"')['Score'],
         )

F_stat, p_value = Results[0], Results[1]
print("F-statistic:", F_stat)
print("p-value:", p_value)

```

```

Results= f_oneway(data[data['Group'] == 'A']['Score'],
                  data[data['Group'] == 'B']['Score'],
                  data[data['Group'] == 'C']['Score'])

```

Results

✓ 0.0s

```
F_onewayResult(statistic=12.20952551797281, pvalue=2.1200748140507065e-05)
```

```

f_oneway(data.query('Group=="A"')['Score'],
         data.query('Group=="B"')['Score'],
         data.query('Group=="C"')['Score'],
         )

```

✓ 0.0s

```
F_onewayResult(statistic=12.20952551797281, pvalue=2.1200748140507065e-05)
```

```

F_stat, p_value = Results[0], Results[1]
print("F-statistic:", F_stat)
print("p-value:", p_value)

```

✓ 0.0s

```
F-statistic: 12.20952551797281
p-value: 2.1200748140507065e-05
```

## Performing ANOVA using OLS(Ordinary Least Squares) method

- # Performing ANOVA using OLS (Ordinary Least Squares) method
 

```
import statsmodels.api as sm
```

```

from statsmodels.formula.api import ols

# Example OLS model for One-Way ANOVA
formula = 'Score ~ C(Group)'

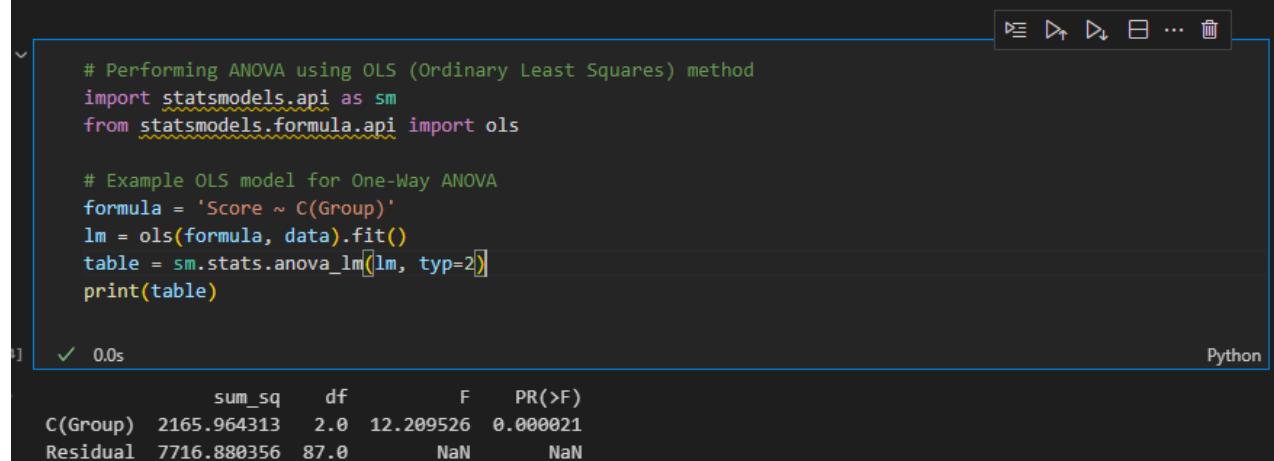
lm = ols(formula, data).fit()

table = sm.stats.anova_lm(lm, typ=2)

print(table)

```

## Performing ANOVA using OLS (Ordinary Least Squares) method



```

# Performing ANOVA using OLS (Ordinary Least Squares) method
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Example OLS model for One-Way ANOVA
formula = 'Score ~ C(Group)'
lm = ols(formula, data).fit()
table = sm.stats.anova_lm(lm, typ=2)
print(table)

```

✓ 0.0s

|          | sum_sq      | df   | F         | PR(>F)   |
|----------|-------------|------|-----------|----------|
| C(Group) | 2165.964313 | 2.0  | 12.209526 | 0.000021 |
| Residual | 7716.880356 | 87.0 | NaN       | NaN      |

Python

### Interpretation:

- The **F-statistic** represents the ratio of variance between the groups to the variance within the groups. A higher F-statistic suggests greater disparity between group means.
- The **p-value** indicates the likelihood that the observed differences are due to random chance. A p-value below our significance threshold (e.g., 0.05) leads us to reject the null hypothesis, concluding that at least one group mean differs significantly.

```
#two way anova #manova #ancova #one way anova
```

- You can see we have the same p\_value and d\_statistic we got from one way
- Once we see there is a differences between the group scores we can use turkeys, dunnets test among others to see the exact values for the groups

### 79. EXAMPLE 2: Effect of Teaching Methods on Student Performance

# Hypothetical Scenario

In this example, we are testing the effectiveness of three different teaching methods on student exam scores.

The three teaching methods are:

- **Group A:** Lecture-only
- **Group B:** Interactive activities
- **Group C:** Mixed (lectures and activities)

We want to know if there are statistically significant differences in exam scores across these three groups.

## Objective

To use ANOVA to check if the teaching method has a significant effect on exam scores.

## Dependencies

Let's start by importing the necessary libraries.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure plots display within the notebook
%matplotlib inline

# Seed for reproducibility
np.random.seed(0)

# Create random scores for each teaching method
scores_A = np.random.normal(70, 10, 30) # Lecture-only group
scores_B = np.random.normal(75, 10, 30) # Interactive activities group
scores_C = np.random.normal(80, 10, 30) # Mixed methods group
```

```
# Combine data into a DataFrame
data = pd.DataFrame({
    'Score': np.concatenate([scores_A, scores_B, scores_C]),
    'Method': ['Lecture-only'] * 30 + ['Interactive'] * 30 + ['Mixed'] * 30
})

# Display the first few rows of the dataset
data.head(50)
```

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure plots display within the notebook
%matplotlib inline
```

[5] ✓ 0.0s

Python

## Generate a Random Dataset

```
# Seed for reproducibility
np.random.seed(0)

# Create random scores for each teaching method
scores_A = np.random.normal(70, 10, 30) # Lecture-only group
scores_B = np.random.normal(75, 10, 30) # Interactive activities group
scores_C = np.random.normal(80, 10, 30) # Mixed methods group

# Combine data into a DataFrame
data = pd.DataFrame({
    'Score': np.concatenate([scores_A, scores_B, scores_C]),
    'Method': ['Lecture-only'] * 30 + ['Interactive'] * 30 + ['Mixed'] * 30
})

# Display the first few rows of the dataset
data.head(50)
```

[2] ✓ 0.0s

Python

|   | Score     | Method       |
|---|-----------|--------------|
| 0 | 87.640523 | Lecture-only |
| 1 | 74.001572 | Lecture-only |
| 2 | 79.787380 | Lecture-only |
| 3 | 92.408932 | Lecture-only |
| 4 | 88.675580 | Lecture-only |
| 5 | 60.227221 | Lecture-only |

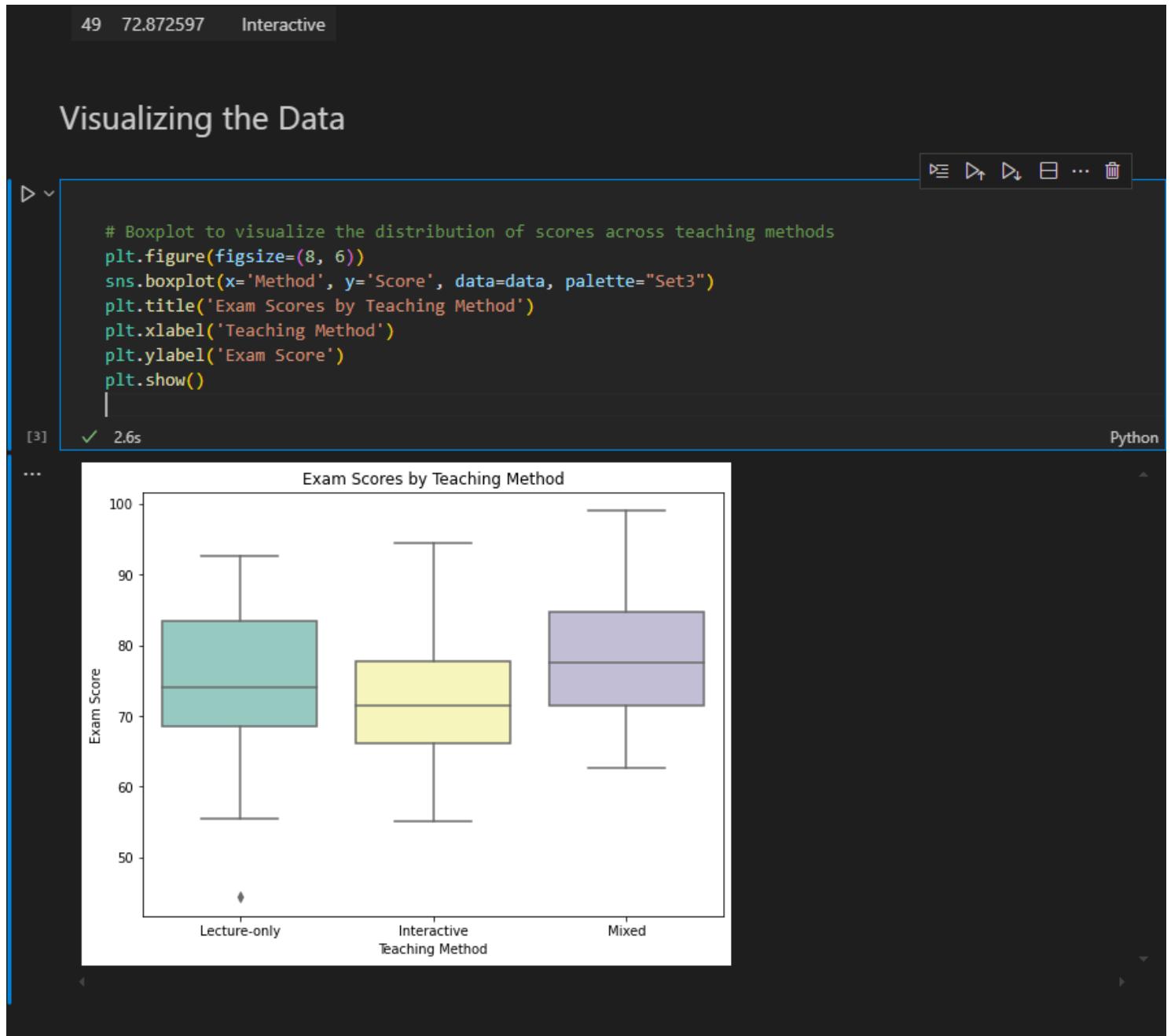
## Visualizing the Data

```
# Boxplot to visualize the distribution of scores across teaching methods
plt.figure(figsize=(8, 6))

sns.boxplot(x='Method', y='Score', data=data, palette="Set3")

plt.title('Exam Scores by Teaching Method')
```

```
plt.xlabel('Teaching Method')
plt.ylabel('Exam Score')
plt.show()
```



## Perform Anova

```
# Define the formula and fit the model
formula = 'Score ~ C(Method)'
```

```
model = ols(formula, data).fit()

# Generate the ANOVA table
anova_table = sm.stats.anova_lm(model, typ=2)
anova_table
```

```
# Define the formula and fit the model
formula = 'Score ~ C(Method)'
model = ols(formula, data).fit()

# Generate the ANOVA table
anova_table = sm.stats.anova_lm(model, typ=2)
anova_table
```

✓ 2.3s

Python

|           | sum_sq      | df   | F        | PR(>F)   |
|-----------|-------------|------|----------|----------|
| C(Method) | 663.361108  | 2.0  | 3.341461 | 0.039981 |
| Residual  | 8635.806619 | 87.0 |          | NaN      |

## Interpreting the Results

In the ANOVA table, focus on the **PR(>F)** column (p-value). If this value is below our significance threshold (commonly 0.05), we can reject the null hypothesis.

### Interpretation:

- If the p-value for the teaching method is **less than 0.05**, we conclude that there are significant differences in exam scores based on the teaching method.
- If the p-value is **greater than 0.05**, we fail to reject the null hypothesis, meaning we don't have enough evidence to claim differences in scores between groups.

## Summary

In this example, we used ANOVA to test the impact of different teaching methods on exam performance. We looked at a dataset of exam scores, split by three teaching methods, and checked if the differences in mean scores were statistically significant.

This analysis is essential in educational research where the effectiveness of teaching styles is often analyzed.

#other tests to check which has the highest mean#use post hoc tests eg turke tests,dunnet tests

### 80. EXAMPLE 3: Check IT salaries

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
df = pd.read_csv('data/IT_salaries.csv')
```

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

✓ 0.0s

Python

## Loading the data

As usual, we start by loading in a dataset of our sample observations. This particular table is of salaries in IT and has 4 columns:

- S - the individuals salary
- X - years of experience
- E - education level (1-Bachelors, 2-Masters, 3-PHD)
- M - management (0-no management, 1-yes management)

```
df = pd.read_csv('data/IT_salaries.csv')
df.head()
```

✓ 0.0s

Python

|   | S     | X | E | M |
|---|-------|---|---|---|
| 0 | 13876 | 1 | 1 | 1 |
| 1 | 11608 | 1 | 3 | 0 |
| 2 | 18701 | 1 | 3 | 1 |
| 3 | 11283 | 1 | 2 | 0 |
| 4 | 11767 | 1 | 3 | 0 |

## Generating the Anova table

```
formula = 'S ~ C(E) + C(M) + X'
lm = ols(formula, df).fit()
table = sm.stats.anova_lm(lm, typ=2)
print(table)
```

```
3 11283 1 2 0  
4 11767 1 3 0
```

## Generating the ANOVA table

In order to generate the ANOVA table, you first fit a linear model and then generate the table from this object. Our formula will be written as:

```
Control_Column ~ C(factor_col1) + factor_col2 + C(factor_col3) + ... + X
```

We indicate categorical variables by wrapping them with `C()`.

The screenshot shows a Jupyter Notebook cell with the following content:

```
formula = 'S ~ C(E) + C(M) + X'  
lm = ols(formula, df).fit()  
table = sm.stats.anova_lm(lm, typ=2)  
print(table)
```

[3] 0.0s

... sum\_sq df F PR(>F)  
C(E) 9.152624e+07 2.0 43.351589 7.672450e-11  
C(M) 5.075724e+08 1.0 480.825394 2.901444e-24  
X 3.380979e+08 1.0 320.281524 5.546313e-21  
Residual 4.328072e+07 41.0 NaN NaN

Python

## Interpreting the table

For now, simply focus on the outermost columns. On the left, you can see our various groups, and on the right, the probability that the factor is indeed influential. Values less than 0.05 (or whatever we set  $\alpha$  to) indicate rejection of the null hypothesis. In this case, notice that all three factors appear influential, with management being the potentially most significant, followed by years experience, and finally, educational degree.

## Summary

In this lesson, you examined the ANOVA technique to generalize testing methods to multiple groups and factors.

### 81. EXAMPLE 4::Tooth growth

```
import pandas as pd  
from statsmodels.formula.api import ols  
import statsmodels.api as sm  
tooth_growth = pd.read_csv('./data/ToothGrowth.csv')  
tooth_growth.head()  
formula = 'len ~ C(supp) + C(dose)'  
lm = ols(formula,tooth_growth).fit()  
table = sm.stats.anova_lm(lm,typ=2)  
print(table)
```

```
# Your code here
import pandas as pd
from statsmodels.formula.api import ols
import statsmodels.api as sm
tooth_growth = pd.read_csv('./data/ToothGrowth.csv')
tooth_growth.head()
✓ 0.0s
```

|   | len  | supp | dose |
|---|------|------|------|
| 0 | 4.2  | VC   | 0.5  |
| 1 | 11.5 | VC   | 0.5  |
| 2 | 7.3  | VC   | 0.5  |
| 3 | 5.8  | VC   | 0.5  |
| 4 | 6.4  | VC   | 0.5  |

## Generate the ANOVA table

Now generate an ANOVA table in order to analyze the influence of the medication and dosage:

```
# Your code here
formula = 'len ~ C(supp) + C(dose)'
lm = ols(formula,tooth_growth).fit()
table = sm.stats.anova_lm(lm,typ=2)
print(table)
✓ 0.0s
```

|          | sum_sq      | df   | F         | PR(>F)       |
|----------|-------------|------|-----------|--------------|
| C(supp)  | 205.350000  | 1.0  | 14.016638 | 4.292793e-04 |
| C(dose)  | 2426.434333 | 2.0  | 82.810935 | 1.871163e-17 |
| Residual | 820.425000  | 56.0 | NaN       | NaN          |

## Interpret the output

Make a brief comment regarding the statistics and the effect of supplement and dosage on tooth length:

```
# Both dose and supplement type are impactful.
# At first glance, dosage seems to be the more impactful of the two.
```

## Compare to t-tests

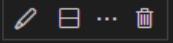
Now that you've had a chance to generate an ANOVA table, it's interesting to compare the results to those from the t-tests you were working with earlier. With that, start by breaking the data into two samples: those given the OJ supplement, and those given the VC supplement. Afterward, you'll conduct a t-test to compare the tooth length of these two different samples:

```
# Your code here
OJ_supp = tooth_growth.query('supp == "OJ"')['len']
VS_supp = tooth_growth.query('supp == "VC"')['len']
```

19]

✓ 0.0s

Python



Now run a t-test between these two groups and print the associated two-sided p-value:

```
# Calculate the 2-sided p-value for a t-test comparing the two supplement groups
import numpy as np
from scipy import stats
pvalue = stats.ttest_ind(OJ_supp, VS_supp)[1]
pvalue
```

24]

✓ 0.0s

Python

·

0.06039337122412849

## A 2-Category ANOVA F-test is equivalent to a 2-tailed t-test!

Now, recalculate an ANOVA F-test with only the supplement variable. An ANOVA F-test between two categories is the same as performing a 2-tailed t-test! So, the p-value in the table should be identical to your calculation above.

Note: there may be a small fractional difference (>0.001) between the two values due to a rounding error between implementations.

```
# Your code here; conduct an ANOVA F-test of the oj and vc supplement groups.  
# Compare the p-value to that of the t-test above.  
# They should match (there may be a tiny fractional difference due to rounding errors in varying implement  
# Your code here  
formula = 'len ~ C(supp)'  
lm = ols(formula,tooth_growth).fit()  
table = sm.stats.anova_lm(lm,typ=2)  
print(table)
```

✓ 0.0s

Python

|          | sum_sq      | df   | F        | PR(>F)   |
|----------|-------------|------|----------|----------|
| C(supp)  | 205.350000  | 1.0  | 3.668253 | 0.060393 |
| Residual | 3246.859333 | 58.0 | NaN      | NaN      |

**82 . STATISTICAL MODELLING** - mathematical approach used to represent and analyze the relationship between variables in data. It involves constructing a model that captures the underlying patterns and structures within a dataset, often with the goal of making predictions or understanding the data better

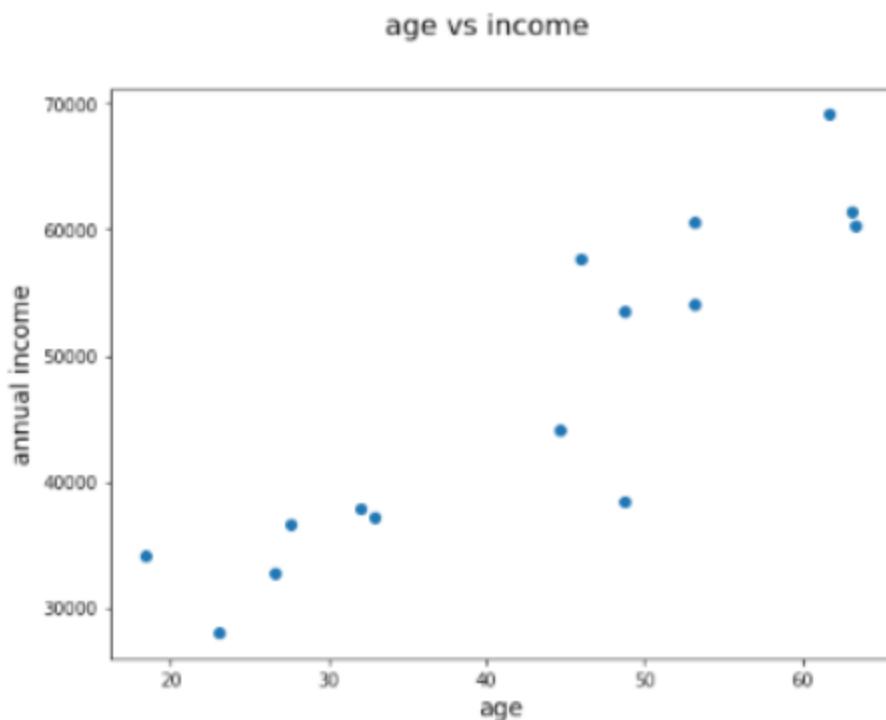
- One of the first steps of building a model is setting up the independent and dependent variables  
**Independent variables**- variables that will affect (or lead to a change) in the dependent variables(s)  
Independent variables are also known as predictor variables, input variables, explanatory variables, features.  
**Dependent variables** – also known as outcome variables, target variables, response variables

Independent variables goes to the x-axis

## Plotting Independent and Dependent Variables

Independent and dependent variables are normally shown on a graph under a standardized approach. This makes it easy for you to quickly see which variable is independent and which is dependent when looking at a graph or chart.

Conventionally, the independent variable goes on the x-axis, or the horizontal axis. Let's consider another example, one where we look at someone's income depending on their age. Below, you see a scatter plot where age is the independent variable, and income is the dependent variable. In this setting, we want to study if age has some effect on annual income.



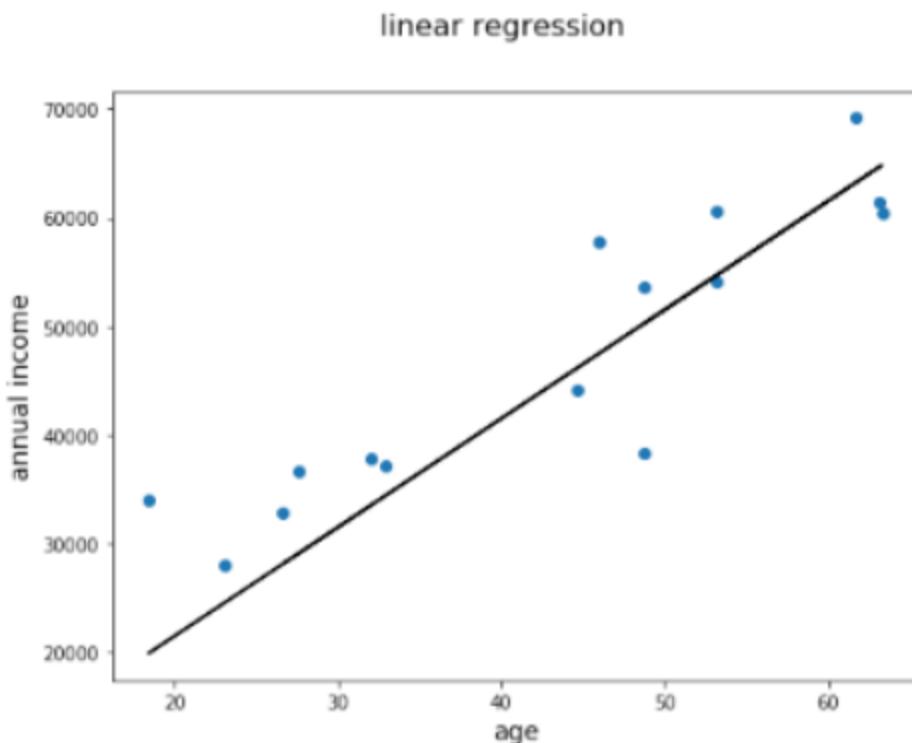
Statistical model can be thought of some kind of transformation that helps us express dependent variables as a function of one or more independent variables.

## Our First Statistical Model

A statistical model can be thought of as some kind of a transformation that helps us express dependent variables as a function of one or more independent variables.

A statistical model defines a **relationship** between a dependent and an independent variable.

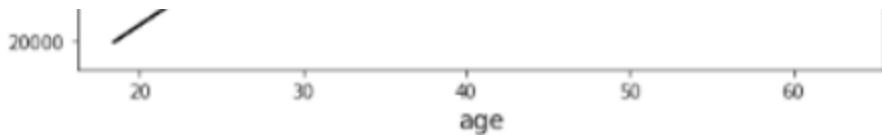
For the plot we see above, the relationship between age and income can be shown using a **straight line** connecting all the individual observations in the data. So this line here would be our **model** as shown in the image below.



We can define and **fit** such a straight line to our data following a straight line equation:

$$y = mx + c$$

-The line here would be our model



We can define and **fit** such a straight line to our data following a straight line equation:

$$y = mx + c$$

You'll often come across Greek letters talking about models like this. Another common way of writing a linear equation is ( $\beta$  is the Greek letter "beta"):

$$y = \beta_0 + \beta_1 x$$

$\beta_0$  has the same role as  $c$  in the first expression and denotes the *intercept with the y-axis*.  $\beta_1$  has the same role as  $m$  in the first expression and denotes the *slope of the line*. More on this below.

Looking at this line above, we can define it as **Income = 1500 + 1000 \* Age**, based on slope ( $m$  or  $\beta_1$ ) and intercept ( $c$  or  $\beta_0$ ) values.

So this is our simple model for the relationship. Of course, we can use more sophisticated models for a better fit, and you may see this later on if you dig into more advanced modeling. This would be our **linear model**, which can help us work out an income value for a given age.

## Statistical model parameters

In the example above, where  $\text{Income} = 1500 + 1000 * \text{Age}$ , we have modeled Income as the dependent variable and Age as the independent variable.

The other pieces of the equation, 1500 and 1000, are the model *parameters*.

The step of 'fitting' a model means finding the best value for these parameters. For linear regression in particular, these parameters are **coefficients** and we will use the terms "parameters" and "coefficient" fairly interchangeably.

## MODEL LOSS

A loss function evaluates how well your model represents the relationships between data variables. If the model is unable to identify the relationship between the independent and dependent variable(s) the loss function will output a very high number

## Model Loss

A loss function evaluates how well your model represents the relationship between data variables.

If the model is unable to identify the underlying relationship between the independent and dependent variable(s), the loss function will output a very high number. Consider the age vs. income example above. You can see that the linear model is not exactly touching each data point because these points do not exist in a line. The individual distance of each point from the line is the **loss** that the model exhibits.



These individual losses, which is essentially the **vertical distance between the individual data points and the line** are taken into account to calculate the overall model loss.

If the relationship is well modeled, the loss will be low. As we change the parameters of our model to try and improve results, our loss function is our best friend, telling us if we are on the right track.

‘Error’ of a linea regression model - the vertical offset between the fit line and the actual observation

### Simple Linear Regression from scratch- code along

In this codealong, you'll get some hands-on practice developing a simple linear regression model. In practice, you would typically use a code library rather than writing linear regression code from scratch, but this is an exercise designed to help you see what is happening "under the hood"

## Simple Linear Regression Recap

Remember that the **data** for a simple linear regression consists of  $y$  (the *dependent variable*) and  $x$  (the *independent variable*). Then the model **parameters** are the slope of the line, denoted as  $m$  or  $\beta_1$ , and the intercept ( $y$  value of the line when  $x$  is 0), denoted as  $c$  or  $\beta_0$ .

Thus the overall model notation is

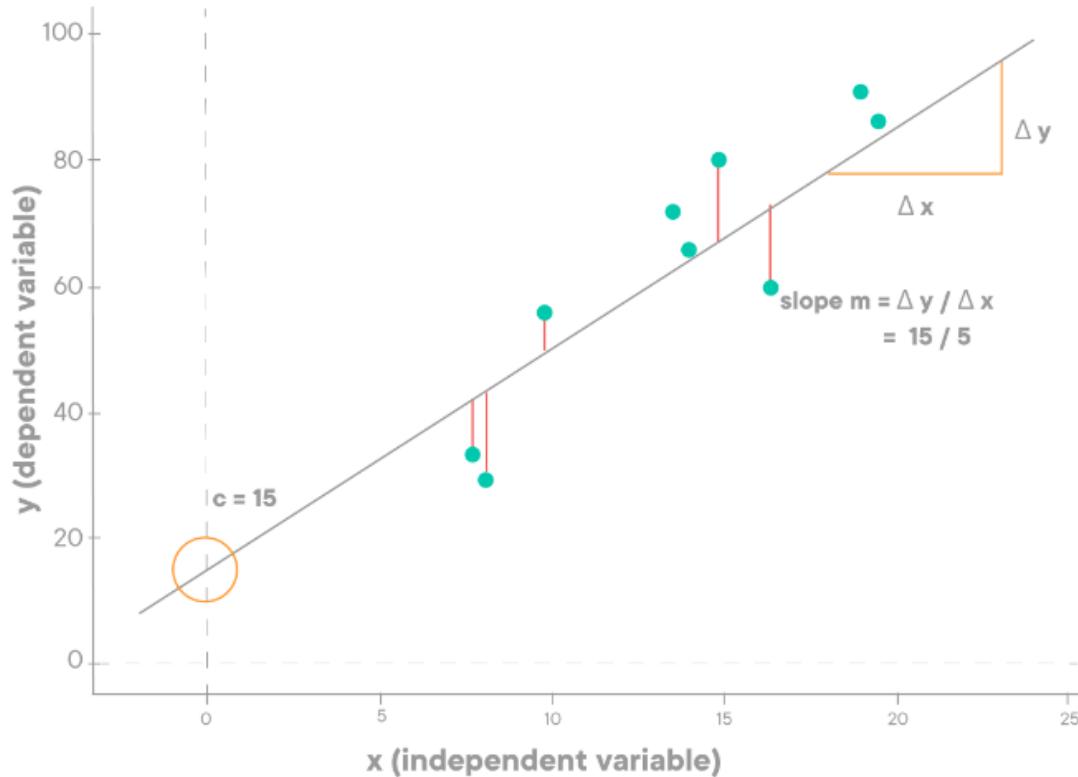
$$y = mx + c$$

or, alternatively

$$y = \beta_0 + \beta_1 x$$

In the example below,  $c$  is equal to 15 and  $m$  is equal to 3.

In other words, the overall equation is  $y = 3x + 15$ .



## Finding Model Parameters

## Finding Model Parameters

---

If you think back to the basic algebra formulas, you might remember that slope can be calculated between two points by finding the change in  $y$  over the change in  $x$ , i.e.  $\Delta y / \Delta x$ . But now you are dealing with messy data rather than perfect abstractions, so your regression line is not going to represent the relationship perfectly (i.e. there is going to be some amount of *error*). The question is how to find the **best fit** line, rather than just calculating  $\Delta y / \Delta x$ .

Because these are **estimations**, we'll use the "hat" notation for the variables, i.e.

$$\hat{y} = \hat{m}x + \hat{c}$$

or

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Everything in these equations represented with a "hat" (e.g.  $\hat{y}$  rather than just  $y$ ) means that it is an estimate or an approximation. The only part that is not using this notation is  $x$ , because we have the actual data values for the independent variable.

So, how do you find the line with the best fit? You may think that you have to try lots and lots of different lines to see which one fits best. Fortunately, this task is not as complicated as it may seem. Given some data points, **the best-fit line always has a distinct slope and y-intercept that can be calculated using simple linear algebraic approaches**.

**The Least Squares Method-** an algorithm to find the best-fit line that minimizes the squared error

# Calculating M

## The Least-Squares Method

We can calculate  $\hat{m}$  (the slope of the best-fit line) using this formula:

$$\hat{m} = \rho \frac{S_y}{S_x}$$

Breaking down those components, we have:

- $\hat{m}$ : the estimated slope
- $\rho$ : the Pearson correlation, represented by the Greek letter "Rho"
- $S_y$ : the standard deviation of the y values
- $S_x$ : the standard deviation of the x values

# Calculating C

Then once we have the slope value ( $\hat{m}$ ), we can put it back into our formula ( $\hat{y} = \hat{m}x + \hat{c}$ ) to calculate the intercept. The idea is that

$$\bar{y} = \hat{c} + \hat{m}\bar{x}$$

so

$$\hat{c} = \bar{y} - \hat{m}\bar{x}$$

Breaking down those components, we have:

- $\hat{c}$ : the estimated intercept
- $\bar{y}$ : the mean of the y values
- $\hat{m}$ : the estimated slope
- $\bar{x}$ : the mean of the x values

## EXAMPLE:

```
# imports

# Run this cell without changes

# import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
```

```

style.use('ggplot')

# Initialize arrays X and Y with given values
# X = Independent Variable
X = np.array([1,2,3,4,5,6,8,8,9,10], dtype=np.float64)
# Y = Dependent Variable
Y = np.array([7,7,8,9,9,10,10,11,11,12], dtype=np.float64)

```

## Step 1: Scatter plot

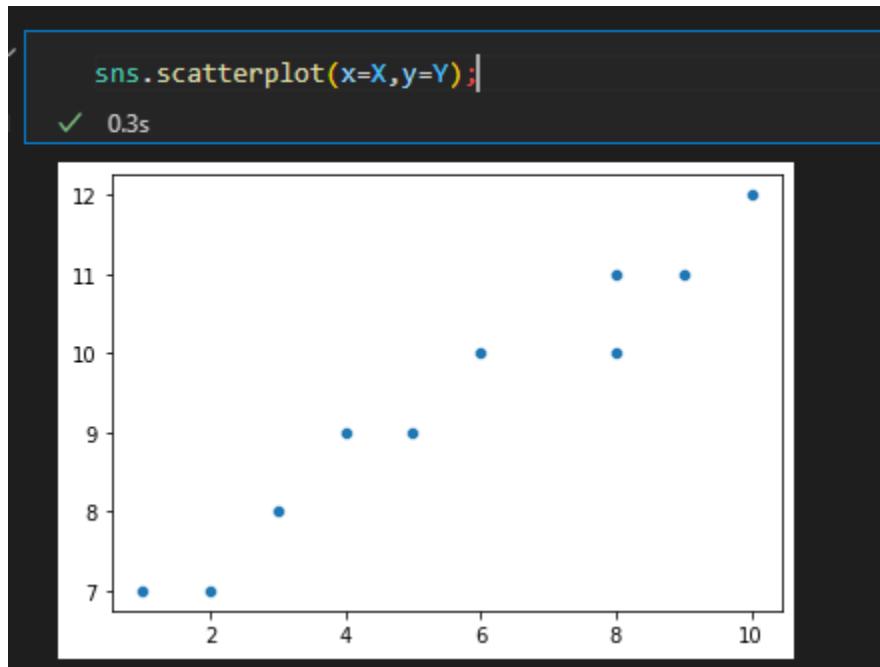
Before performing a linear regression analysis, it's a best practice to look at scatter plot of the independent variables vs dependent variable. Linear regression is only appropriate if there is a linear relationship between them. In the cell below, create a quick scatter plot showing x vs. y.

```

plt.scatter(X, Y);

sns.scatterplot(x=X,y=Y);

```



Based on the plot above, does linear regression analysis seem appropriate? Yes the relationship is very linear but not perfectly linear. The best line of fit should be able to explain this relationship with a very low error

## Step 2: Calculate slope

The formula is:

$$\hat{m} = \rho \frac{S_y}{S_x}$$

```

std_y = Y.std()#np.std(Y)
std_x= X.std()#np.std(X)
p_corr = np.corrcoef(X,Y)[0][1]
m = p_corr * (std_y/std_x)
m

```

Do the same using a function

```

def get_slope(X,Y):
    # corr_coef
    p_corr = np.corrcoef(X,Y)[0][1]
    #std y
    std_y = np.std(Y)
    #std X
    std_x = X.std()
    #m
    m = p_corr*(std_y/std_x)
    return m

get_slope(X,Y)

```

### Step 3: Calculate Intercept

```

# calculate the intercept
Y_mean = Y.mean()#np.mean()
X_mean = X.mean()
print('Y_mean',Y_mean)
print('X_mean',X_mean)
c = Y_mean - m*(X_mean)
c

```

Using a function

```

def get_c(X,Y):
    #Y_mean
    Y_mean = np.mean(Y)
    #X_mean
    X_mean = np.mean(X)
    m = m= get_slope(X,Y)
    #c
    c = Y_mean - m*(X_mean)
    return c
get_c(X,Y)

```

# Calculating the Intercept

Now that we have our estimated slope  $\hat{m}$ , we can calculate the estimated intercept  $\hat{c}$ .

As a reminder, the calculation for the best-fit line's y-intercept is:

$$\hat{c} = \bar{y} - \hat{m}\bar{x}$$

Write a function `calc_intercept` that returns  $\hat{c}$  for a given  $\hat{m}$ ,  $x$ , and  $y$ .

```
# calculate the intercept
Y_mean = Y.mean()#np.mean()
X_mean = X.mean()
print('Y_mean',Y_mean)
print('X_mean',X_mean)
c = Y_mean - m*(X_mean)
c
```

[7]

✓ 0.0s

Python

```
... Y_mean 9.4
... X_mean 5.6
```

```
... 6.37962962962963
```

```
def get_c(X,Y):
    #Y_mean
    Y_mean = np.mean(Y)
    #X_mean
    X_mean = np.mean(X)
    m = m= get_slope(X,Y)
    #c
    c = Y_mean - m*(X_mean)
    return c
```

```
get_c(X,Y)
```

[8]

✓ 0.0s

Python

```
... 6.37962962962963
```

## Step 4: Predicting a new data point

```
y = 3* get_slope(X,Y) + get_c(X,Y)
y
```

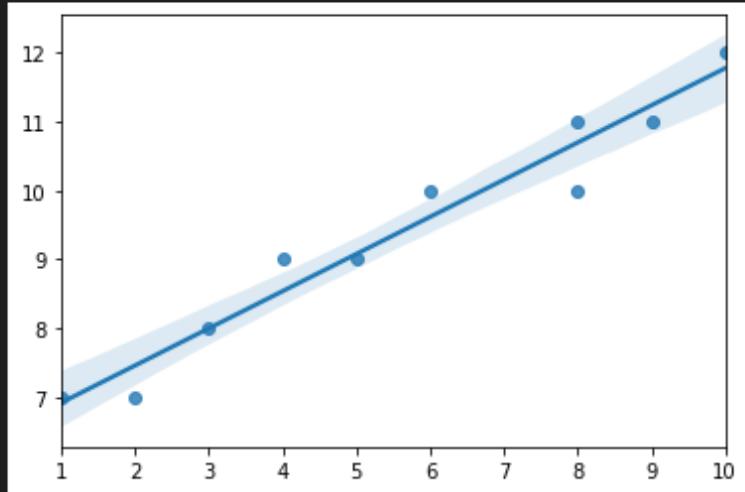
# Predicting a datapoint



```
# to predict a data point
# X = np.array([1,2,3,4,5,6,8,8,9,10], dtype=np.float64)
# # Y = Dependent Variable
# Y = np.array([7,7,8,9,9,10,10,11,11,12], dtype=np.float64)
y = 3* get_slope(X,Y) + get_c(X,Y)
y
✓ 0.0s
```

7.997685185185185

```
sns.regplot(x=X,y=Y);
✓ 0.5s
```



## Step 5: Bringing it all together

```
import matplotlib.pyplot as plt
def best_fit(x_vals, y_vals):
    # Create a scatter plot of x vs. y
    plt.scatter(x_vals,y_vals)

    # Calculate and print coefficient and intercept
    m = get_slope(x_vals,y_vals)
    c = get_c(x_vals,y_vals)
    print('m',m)
    print('c',c)
    y= m*x_vals +c
```

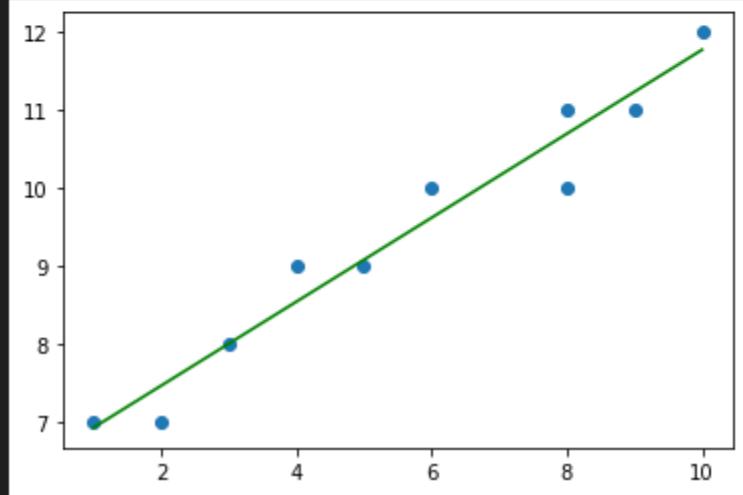
```
# Plot line created by coefficient and intercept  
plt.plot(x_vals,y,color='g')
```

```
best_fit(X, Y)
```

```
import matplotlib.pyplot as plt  
def best_fit(x_vals, y_vals):  
    # Create a scatter plot of x vs. y  
    plt.scatter(x_vals,y_vals)  
  
    # Calculate and print coefficient and intercept  
    m = get_slope(x_vals,y_vals)  
    c = get_c(x_vals,y_vals)  
    print('m',m)  
    print('c',c)  
    y= m*x_vals +c  
    # Plot line created by coefficient and intercept  
    plt.plot(x_vals,y,color='g')  
  
best_fit(X, Y)
```

✓ 0.2s

```
m 0.5393518518518519  
c 6.37962962962963
```



## Step 6: Describe your Model Mathematically and in Words

The overall formula is  $y=0.53x+6.37$ .

The intercept (where the line crosses the y-axis) is at 6.37. This means that if x is equal to 0, the value of y would be 6.37.

The slope of the line is 0.53. This means that every increase of 1 in the value of x is associated with an increase of 0.53 in the value of y.

## **LINEAR REGRESSION CONNECTIONS TO OTHER STATISTICAL MEASURES**

Linear models are closely related to other statistical measures: t-test, Pearson correlation, ANOVA, Chi-square, etc

Lets see how linear regression is related to t-tests, confidence intervals and Pearson Correlation

### **Our Linear Regression Example**

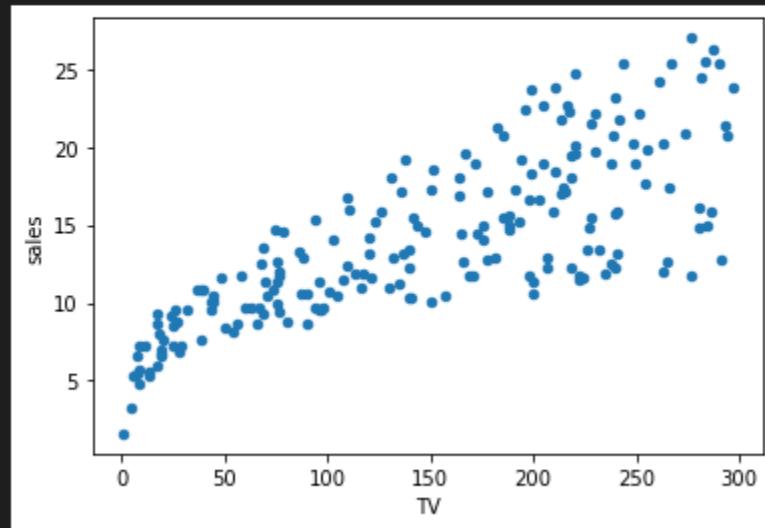
In the examples below we'll be using the Advertising dataset, using TV spending to predict sales.

```
import pandas as pd

data = pd.read_csv("data/advertising.csv", index_col=0)
data.plot(x="TV", y="sales", kind="scatter");
```

```
import pandas as pd

data = pd.read_csv("data/advertising.csv", index_col=0)
data.plot(x="TV", y="sales", kind="scatter");
```



The code below builds a linear regression model using StatsModels and displays the params(i.e coefficients)

```
# https://www.statsmodels.org/devel/generated/statsmodels.regression.linear_model.OLS.html
import statsmodels.api as sm

results = sm.OLS(data["sales"], sm.add_constant(data[["TV"]])).fit()
results.params
```

```

# https://www.statsmodels.org/devel/generated/statsmodels.regression.linear_model.OLS.html
import statsmodels.api as sm

results = sm.OLS(data["sales"], sm.add_constant(data[["TV"]])).fit()
results.params|
```

Python

```

c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\scipy\_init_.py:138: UserWarning: A NumPy version
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: Ir
  x = pd.concat(x[::-order], 1)

const      7.032594
TV         0.047537
dtype: float64
```

The code above is saying that we have set `TV` (money spent on television advertising) as the independent variable, and `sales` as the dependent variable. And the result of this linear regression model is that we have an intercept of about 7.033 and a `TV` coefficient of about 0.048.

In other words, our regression model is `sales` = 7.033 + 0.048 \* `TV`.

Lets plot that line along with the data

```

import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots();
data.plot(x="TV", y="sales", kind="scatter", ax=ax)

# set up line data representing linear regression model
x = np.linspace(data["TV"].min(), data["TV"].max())
y = x * results.params["TV"] + results.params["const"]

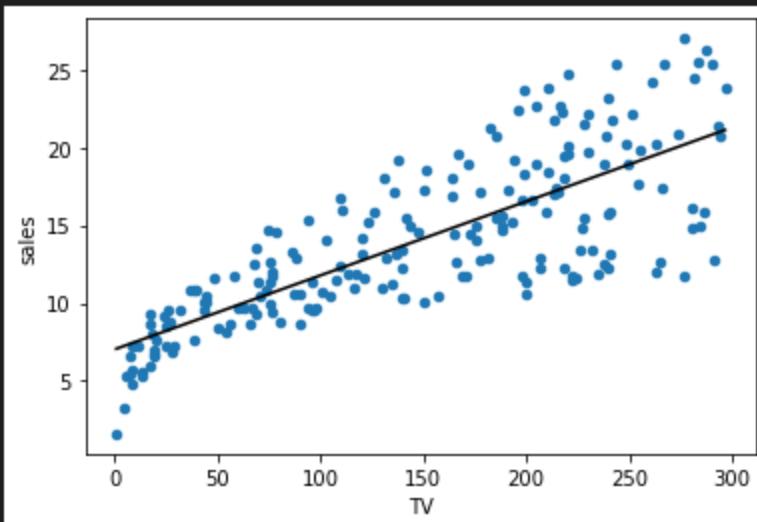
# plot model on the same axes as the data
ax.plot(x, y, color="black");
```

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots();
data.plot(x="TV", y="sales", kind="scatter", ax=ax)

# set up line data representing linear regression model
x = np.linspace(data["TV"].min(), data["TV"].max())
y = x * results.params["TV"] + results.params["const"]

# plot model on the same axes as the data
ax.plot(x, y, color="black");
```



## Linear Regression and t-test

## Linear Regression and t-Test

Most examples of a t-test that you might have seen are in terms of two different means. For example, the null hypothesis might be that  $\mu_1 = \mu_2$  while the alternative hypothesis might be  $\mu_1 \neq \mu_2$ .

**In the context of linear regression, the t-test that we use is for whether a given coefficient is equal to zero.**

The null hypothesis is that  $\beta = 0$  and alternative hypothesis is that  $\beta \neq 0$ .

Luckily, we don't have to perform any additional data manipulation, or import or run any additional functions, to see the results of this t-test. It is just run as part of the linear regression fitting we already did!

We can get the t-test p-values from our linear regression result like this:

```
results.pvalues
```

Python

```
const      1.406300e-35
TV         1.467390e-42
dtype: float64
```

Compared to a standard alpha of 0.05, these are quite small p-values, which means that there is a statistically significant difference between our coefficients and 0. In other words, both our intercept and our slope are statistically significant.

## LINEAR REGRESSION AND CONFIDENCE INTERVALS

## Linear Regression and Confidence Intervals

Just like you can use the t-distribution to calculate confidence intervals about a mean, you can also use the t-distribution to calculate confidence intervals for linear regression coefficients.

**In the context of linear regression, we use confidence intervals to express uncertainty about our coefficients.**

In particular, you can call the `conf_int` method to get the 95% confidence interval for the coefficients:

```
print(results.conf_int())
```

Python

```
          0      1
const  6.129719  7.935468
TV     0.042231  0.052843
```

This means that our 95% confidence interval for `const` (the intercept) is about 6.13 to 7.94, and the 95% confidence interval for the `TV` coefficient is about 0.042 to 0.053.

## LINEAR REGRESSION VS CORRELATION

## Linear Regression vs. Correlation

You might be thinking back to correlation, which is similar to linear regression in that it also describes a relationship between two numeric variables. What is the difference between correlation and linear regression, and why might you use one or the other?

Correlation is a single measurement that describes the strength of the relationship between the variables:

```
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html
from scipy.stats import pearsonr

correlation, p_value = pearsonr(data["sales"], data["TV"])
correlation
```

[7]

Python

```
... 0.7822244248616065
```

That first number, 0.78, means that this is a fairly strong, positive correlation. But it does not tell us the change in `sales` that we might expect for a given change in `TV` spending, nor does it give us anything to plot.

How is it related to the linear regression?

First, they have the same p-value (within rounding error difference):

```
print(p_value) # from pearsonr
print(results.pvalues["TV"])
```

[8]

Python

```
... 1.467389700194781e-42
1.4673897001946964e-42
```

```
[8] Python
...
...     1.467389700194781e-42
     1.4673897001946964e-42

Second, the correlation is the same as the coefficient if we scale sales and TV so that they both have a standard deviation of 1:

[9] Python
D v
print(round(correlation, 10))
print(round(results.params["TV"] / data["sales"].std() * data["TV"].std(), 10))

[9] Python
...
...     0.7822244249
     0.7822244249
```

However the correlation does not give you an intercept value, or describe the relationship in terms of a model.

## Linear Regression vs. Correlation: The Bottom Line

Correlation and linear regression are very similar calculations. Both are comparing two numeric variables against each other. And in fact, you can calculate correlation from a linear regression coefficient (or vice versa) by scaling the data.

However, linear regression is more interpretable because it produces coefficients that can be used to describe the relationship between the variables. **In general you should use linear regression rather than correlation if you believe that there is a linear association between an independent variable and a dependent variable.**

Unlike with t-tests and confidence intervals, where that knowledge will be incorporated into your linear regression interpretation, most likely you will replace any analysis that previously used correlation with linear regression analysis.

## 82. REGRESSION MODEL EVALUATION

For any given set of X and Y values, we can build a linear regression model using the least squares method to find the best fit line. Just because this model exists does not necessarily mean it is a particularly appropriate or useful way to describe the relationship between the two variables.

-Here we will see techniques to evaluate a regression model. First we use F-tests to test whether the model overall is **statistically significant**. Then we'll use the coefficient of determination(also known as 'R squared') to measure the **goodness of fit**

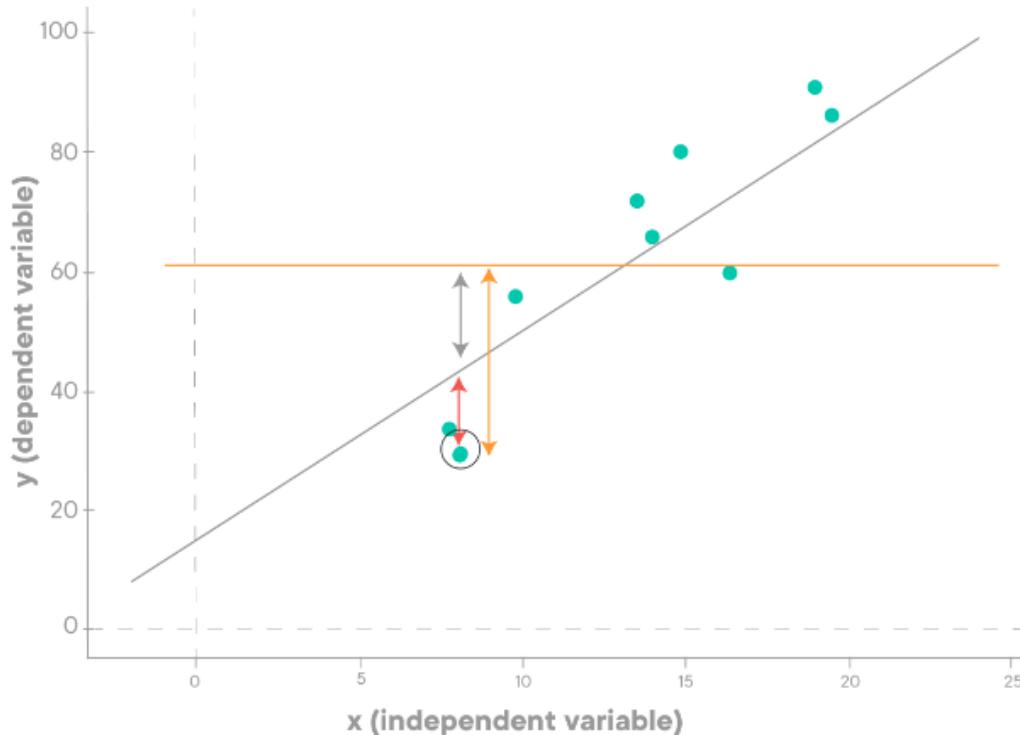
### The intercept only Model

There is no universal, objective standard for evaluating models, because every set of x and y values is different. Therefore we measure the fit of our models against a baseline model known as **intercept-only model**

**Intercept-only model** does not make use of any independent variables to predict the value of dependent variable y. Instead it uses the **mean** of the observed responses of the dependent variable y and always predicts this mean as the value of y for any value of x. It is also known as the 'mean model', 'baseline model' or 'restricted model'

An intercept-only model does not make use of any independent variables to predict the value of dependent variable  $y$ . Instead, it uses the **mean** of the observed responses of the dependent variable  $y$  and always predicts this mean as the value of  $y$  for any value of  $x$ . It is also known as the "mean model", "baseline model", or "restricted model".

In the image below, this model is given by the straight orange line.



You can see that, in this plot, the intercept-only model (orange) always predicts the mean of  $y$  irrespective of the value of the  $x$ . The gray line, however, is our fitted regression line which makes use of  $x$  values to predict the values of  $y$ .

In order to evaluate the regression model, we ask: is our **fitted regression line better than an intercept-only model?**

We will determine this answer in terms of both statistical significance and goodness-of-fit by comparing the **errors** made by our model (red vertical arrow) to the errors made by the intercept-only model (orange vertical arrow) and determining the difference between them (gray vertical arrow).

## TOY DATASET EXAMPLE

```
#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
# Initialize arrays X and Y with given values
# X = Independent Variable
X = np.array([1,2,3,4,5,6,8,8,9,10],dtype=np.float64)
Y = np.array([7,7,8,9,9,10,10,11,11,12],dtype = np.float64)
```

```
plt.scatter(X,Y)
```

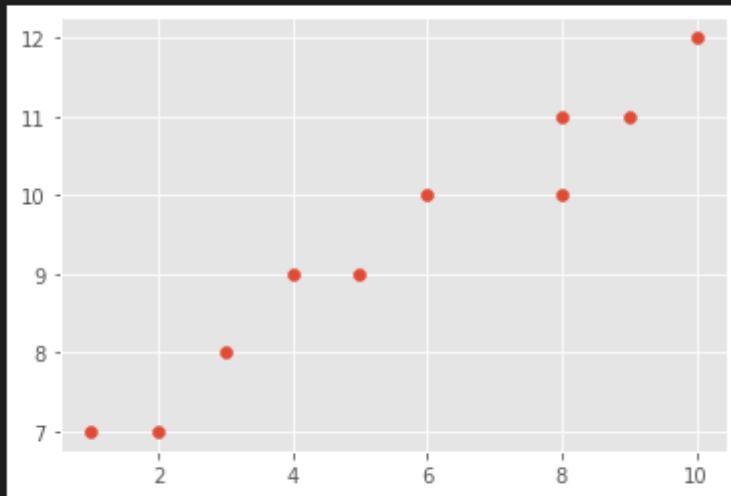
## Toy Dataset Example

Let's return to our toy dataset and fitted regression model:



```
#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
# Initialize arrays X and Y with given values
# X = Independent Variable
X = np.array([1,2,3,4,5,6,8,8,9,10],dtype=np.float64)
Y = np.array([7,7,8,9,9,10,10,11,11,12],dtype = np.float64)
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x235c6888160>
```



```
def calc_slope(x_vals, y_vals):
    # setting up components of formula
    rho = np.corrcoef(x_vals, y_vals)[0][1]
    s_y = y_vals.std()
    s_x = x_vals.std()
```

```
# calculating slope estimate
m = rho * s_y / s_x
return m

def calc_intercept(m, x_vals, y_vals):
    # setting up components of formula
    y_mean = y_vals.mean()
    x_mean = x_vals.mean()

    # calculating intercept estimate
    c = y_mean - m * x_mean
    return c

fig, ax = plt.subplots()
ax.scatter(X, Y, color='#003F72', label="Data points")

m = calc_slope(X,Y)
c = calc_intercept(m,X,Y)
print(f'Our model is: y = {round(m,3)}x + {round(c,4)}')

regression_line = m*X+c
ax.plot(X,regression_line,label='Regression Line')

intercept_only_line = np.array([Y.mean() for x in X])
print(intercept_only_line)
ax.plot(X,intercept_only_line,label='Intercept-Only Line')

ax.legend();
```

```

# calculating slope estimate
m = rho * s_y / s_x
return m

def calc_intercept(m, x_vals, y_vals):
    # setting up components of formula
    y_mean = y_vals.mean()
    x_mean = x_vals.mean()

    # calculating intercept estimate
    c = y_mean - m * x_mean
    return c

fig, ax = plt.subplots()
ax.scatter(X, Y, color="#003F72", label="Data points")

m = calc_slope(X,Y)
c = calc_intercept(m,X,Y)
print(f'Our model is: y = {round(m,3)}x + {round(c,4)}')

regression_line = m*X+c
ax.plot(X,regression_line,label='Regression Line')

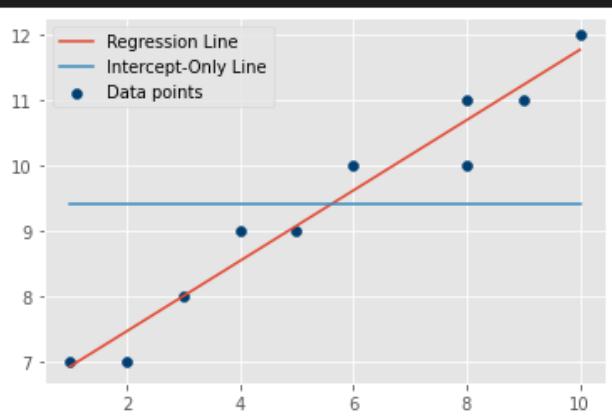
intercept_only_line = np.array([Y.mean() for x in X])
print(intercept_only_line)
ax.plot(X,intercept_only_line,label='Intercept-Only Line')

ax.legend();

```

Python

Our model is:  $y = 0.539x + 6.3796$   
 $[9.4 \ 9.4 \ 9.4 \ 9.4 \ 9.4 \ 9.4 \ 9.4 \ 9.4 \ 9.4]$



## F-Test for Statistical Significance

So, is our overall model statistically significant? Let's frame this in terms of a null and alternative hypothesis:

- H0 (**null hypothesis**): the intercept-only model fits the data just as well as (or better than) our model
- Ha (**alternative hypothesis**): our model fits the data better than the intercept-only model

Then we'll follow the typical hypothesis test structure of computing a test statistic, calculating the p-value, then determining whether we can reject the null hypothesis.

```
import statsmodels.api as sm
model = sm.OLS(endog=Y, exog=sm.add_constant(X))
results = model.fit()
results.f_pvalue, results.fvalue
```

```
import statsmodels.api as sm
model = sm.OLS(endog=Y, exog=sm.add_constant(X))
results = model.fit()
results.f_pvalue, results.fvalue
```

✓ 0.0s

(1.4755820678821598e-06, 158.79780621572215)

## Is our Model statistically significant

We have f statistic of 158 but what does it mean. Our p\_value is below the typical alpha of 0.05. We can reject the null hypothesis and say that our model is statistically significant!

## Measuring Goodness of Fit with R Squared(R2 helps to understand the performance of your model compared to the baseline model). How well the model fits the data

Now that we know our model is statistically significant, we can go one step further and quantify how much of the variation in the dependent variable is explained by our model. This measure is called R2 or coefficient of determination

```
results.rsquared
```

```
results.rsquared
```

✓ 0.0s

0.9520377384960719

## Interpreting R Squared

We have a value of about 0.95 as our coefficient of determination (R-squared). What does this mean?

Means that the proportion of the variance explained by our model out of the total variance is about 0.95, or 95%. We can also say that, **95% of the variation in y is explained by our model**

For a least-squares regression model, R-Squared can take a value between 0 and 1 where values closer to 0 represent a poor fit and values closer to 1 represent an (almost) perfect fit

An R Squared of 0 would mean that we are explaining 0% of the variation in y. This is especially likely to occur if your dataset is not actually suited to linear regression modeling, like the image below.

Eg r squared of 61 % with predictor as tv advert for sales- we cld say that **61 %of the variance in the target variable can be explained by TV spending**

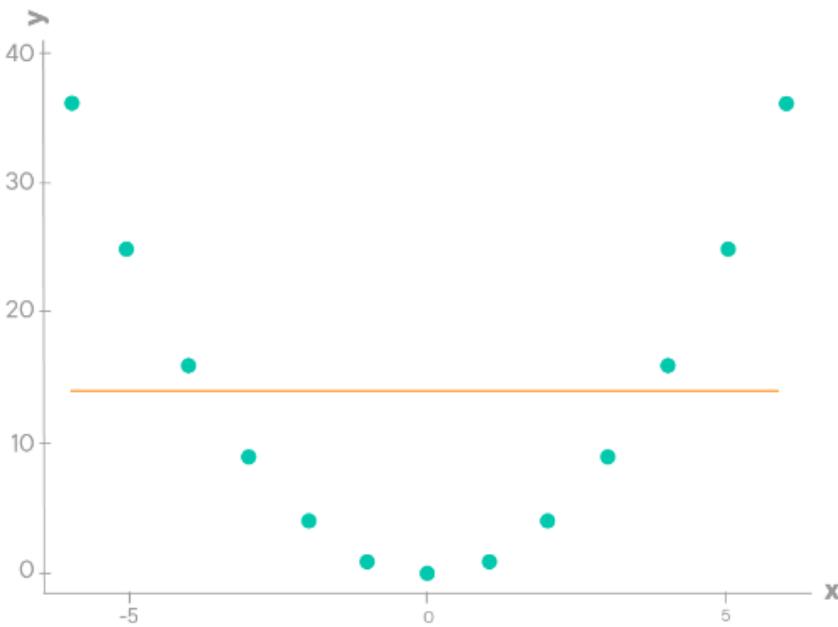
## Interpreting R-Squared

We have a value of about 0.95 as our coefficient of determination (R-squared). What does this mean?

This means that the proportion of the variance *explained* by our model out of the *total* variance is about 0.95, or 95%. You can also say, **95% of the variation in \$y\$ is explained by our model.**

For a least-squares regression model, R-Squared can take a value between 0 and 1 where values closer to 0 represent a poor fit and values closer to 1 represent an (almost) perfect fit

An R-Squared of 0 would mean that we are explaining 0% of the variation in y. This is especially likely to occur if your dataset is not actually suited to linear regression modeling, like the image below.

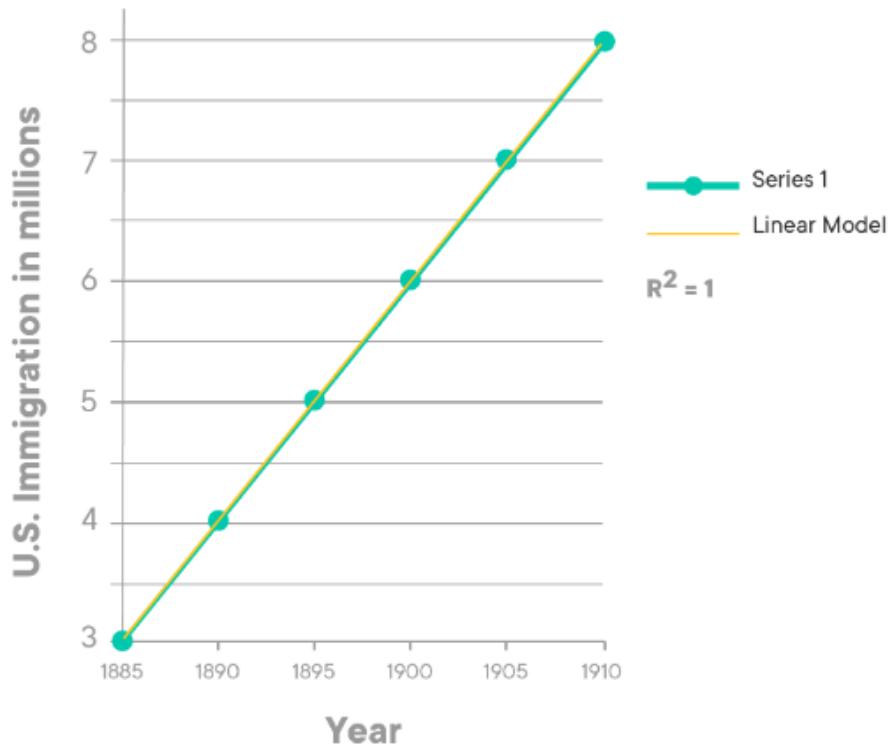


An R-Squared of 1 means that you are explaining 100% of the variation. This is very unlikely to see with linear regression on real-world data, so you probably want to double-check that you have set up your variables properly.

The image below shows what it might look like if you had an R-Squared of 1:

An R-Squared of 1 means that you are explaining 100% of the variation. This is very unlikely to see with linear regression on real-world data, so you probably want to double-check that you have set up your variables properly.

The image below shows what it might look like if you had an R-Squared of 1:



### What is a ‘good’ R-Squared Value?

If you doing *inferential modelling*, i.e the part of the model you are mostly interested is in the value of the parameters , then as long as the model is statistically significant overall,you don’t need to worry about R-Squared

If you are doing *predictive modelling* i.e you are most interested in making predictions about unknown values of y,then a higher R-squared is important

- **Goal:** Inferential models aim to explain relationships, while predictive models aim to forecast outcomes.
- **Interpretability:** Inferential models are often more interpretable and focused on significance testing, while predictive models prioritize accuracy and may use complex algorithms with less interpretability.
- **Data Requirements:** Predictive models generally require larger datasets to achieve high accuracy, while inferential models can work well with smaller, well-sampled data.

## What Is a "Good" R-Squared Value?

Like many other things in data science, the answer is "it depends"!

If you are doing *inferential* modeling, i.e. the part of the model you are most interested in is the value of the parameters, then as long as the model is statistically significant overall, you don't need to worry too much about R-Squared.

If you are doing *predictive* modeling, i.e. you are most interested in making predictions about unknown values of  $y$ , then a higher R-Squared is very important.

Either way, R-Squared provides important context for how you communicate about your model to others. If your model is only explaining a tiny amount of the variation in  $y$ , then there are potentially-important contributing factors that aren't included in your analysis. Just be sure to communicate your findings (and their limitations) clearly!

A good template to use is:

*(R-Squared \* 100)% of the variation in the dependent variable \$y\$ is explained by our model.*

Other evaluation Metrics Model parameters:

```
[60] # y=mx+c  
# m,c  
results.params #tv=m #mis negative for a negative correlation  
... const    7.032594  
TV        0.047537  
dtype: float64
```

Python

Model parameter p-values:

```
[61] # p-values  for m & c  
results.pvalues  
... const    1.406300e-35  
TV        1.467390e-42  
dtype: float64
```

Python

Model Confidence intervals returns value ranges at 95% percent confidence intervals

## confidence intervals

```
[62] results.conf_int() #give ranges instead of single values  
...  
0           1  
const  6.129719  7.935468  
TV    0.042231  0.052843
```

Python

# confidence intervals

```
results.conf_int() #give ranges instead of single values
```

[62]

Python

...

0 1

```
const 6.129719 7.935468
TV 0.042231 0.052843
```

```
#y = mx+c
lower_bound = 0.04223*230.1 + 6.129719
print('lower_bound',lower_bound)
higher_bound = 0.052843*230.1 + 7.935468
print('higher_bound',higher_bound)
```

[63]

Python

...

```
lower_bound 15.846841999999999
higher_bound 20.0946423
```

```
lower_bound,higher_bound
```

[64]

Python

...

```
(15.84684199999999, 20.0946423)
```

```
lower_bound = 0.04223*44.5 + 6.129719
print('lower_bound',lower_bound)
higher_bound = 0.052843*44.5 + 7.935468
print('higher_bound',higher_bound)
```

[65]

Python

...

```
lower_bound 8.008954
higher_bound 10.2869815
```

□ Use **covariance** to understand the direction of the relationship.

□ Use **correlation** to understand both the direction and the strength of the relationship.

\_Why\_ does it happen that variables correlate? It \_may\_ be that one is the cause of the other. A city having a high population, for example, probably does have some causal effect on the number of buses that the city has. But this \_need not\_ be the case, and that is why statisticians are fond of saying that 'correlation is not causation'. An alternative possibility, for example, is that high values of X and Y are \_both\_ caused by high values of some third factor Z. The size of children's feet, for example, is correlated with their ability to spell, but this is of course NOT because either is a cause of the other. Rather, BOTH are caused by the natural maturing and development of children. As they get older, both their feet and their spelling abilities grow!

- **Data Handling:** sm.OLS expects arrays or matrices, while sm.formula.ols works with DataFrames and formula strings.
- **Ease of Use:** sm.formula.ols is more user-friendly for typical regression tasks with categorical variables and complex relationships.
- **Automatic Handling:** sm.formula.ols can automatically create dummy variables for categorical data and includes an intercept by default.

In short, use sm.OLS if you need fine-grained control and sm.formula.ols for a more intuitive, formula-based approach, especially when working with DataFrames.

### **83. LINEAR REGRESSION IN STATS MODELS**

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('data/heightWeight.csv')
df.plot.scatter(x="height", y="weight");
```

Going forward, `sm` refers to StatsModels.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')
```

[22]

Python

## Loading the Data

Let's load a simple dataset for the purpose of understanding the process.

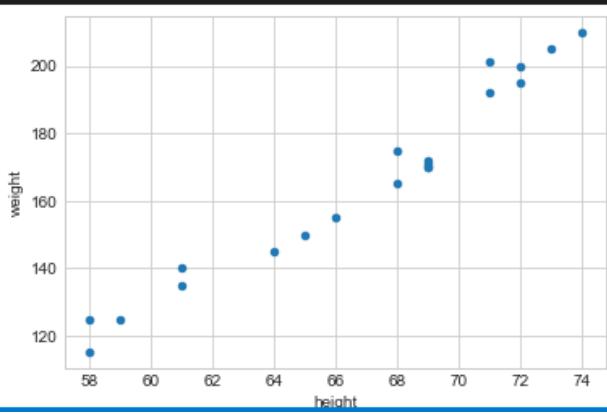
The dataset contains weight and height, and we'll set height as the independent variable ( $x$ ) and weight as the dependent variable.

We'll start out by plotting them against each other in a scatter plot:

```
df = pd.read_csv('data/heightWeight.csv')
df.plot.scatter(x="height", y="weight");
```

[23]

Python



-Looks like a linear relationship

## Building our Linear Regression

```
rho = np.corrcoef(df["height"], df["weight"])[0][1]
s_y = df["weight"].std()
s_x = df["height"].std()
m = rho * s_y / s_x
```

```
mean_y = df["weight"].mean()
mean_x = df["height"].mean()
c = mean_y - m * mean_x

print(f"Our regression line is: y = {round(m, 5)}x + {round(c, 5)})")
```

That works, in a way. But now if we wanted to calculate the F-statistic, or R-Squared, or find the confidence intervals for the parameters, that would be a lot of additional calculations.

If we use **StatsModels instead**, all of the calculations are done for us! We just have to set up the appropriate x and y variables and plug them into a model.

## Step 1: Determining X and y variables

```
X = df[["height"]]
y = df["weight"]
```

### Determining **X** and **y** Variables

Technically you can subset a pandas dataframe in the same line where you create the model, but it tends to be easier if you specify your variables first.

```
X = df[["height"]]
y = df["weight"]
```

Python

You might notice that **X** is capitalized and **y** is not. This is not a mistake! The idea is to indicate that **X** is a 2D matrix, and **y** is a 1D array. Currently **X** only has one value in it (height) but this will change when we get to multiple regression.

## Step 2: Creating and fit the model

```
model = sm.OLS(endog=y, exog=sm.add_constant(X))
model
results = model.fit()
results
```

## Creating the Model

With StatsModels, the "model" is the result of calling the `OLS` constructor function.

We will also use the `add_constant` method because StatsModels expects a column of constant values if there should be a constant in the resulting regression.

```
model = sm.OLS(endog=y, exog=sm.add_constant(x))
model
<statsmodels.regression.linear_model.OLS at 0x1cf0b404c0>
```

Python

## Fitting the Model

Once we have a model, we call the `fit` method, which returns a results object.

```
results = model.fit()
results
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1cf0b400a0>
```

Python

### Step3: Evaluating and interpreting the model

#### a)F statistic and f statistic p value

`results.fvalue, results.f_pvalue`

## Evaluating and Interpreting the Model

Now we can get all kinds of different information out of the model results!

#### ***F-statistic and F-statistic p-value***

```
results.fvalue, results.f_pvalue
(384.774028857076, 1.3461445985228957e-13)
```

**Interpretation:** our model is statistically significant, with a p-value below the standard alpha of 0.05

#### b)R-Squared (coefficient of determination)

```
results.rsquared
```

**R-Squared** (coefficient of determination):

```
results.rsquared
```

```
✓ 0.0s
```

```
0.9553099288673668
```

**Interpretation:** Our model explains about 95.5% of the variance in weight, the dependent variable

c) Model parameters:

```
results.params
```

```
results.params
```

```
✓ 0.0s
```

```
const -204.483436
```

```
height 5.539019
```

```
dtype: float64
```

**Interpretation:** For a height of 0, our model would predict a weight of about -204.5. An increase of 1 in height is associated with an increase of about 5.5 in weight

```
results.params
```

```
✓ 0.0s
```

```
const -204.483436
```

```
height 5.539019
```

```
dtype: float64
```

Python

**Interpretation:** For a height of 0, our model would predict a weight of about -204.5. An increase of 1 in height is associated with an increase of about 5.5 in weight.

Notes:

1. These are the same values we got from our "by hand" regression earlier! `m` corresponds to `height` in this output, and `c` corresponds to `const`
2. Intercept values are often nonsensical if a value of 0 in the independent variable is nonsensical. So don't worry too much about making sense of what a height of -204.5 for a weight of 0 means

d) Model parameters p-values

```
results.pvalues
```

```
results.pvalues
```

```
✓ 0.0s
```

```
const    2.688182e-09
height   1.346145e-13
dtype: float64
```

**Interpretation :** both our model parameters (coefficient for height and intercept) are statistically significant , with p-values well below the standard alpha of 0.05

#### e)Model parameter and confidence interval

```
print(results.conf_int())
```

##### ***Model parameter confidence intervals:***

```
print(results.conf_int())
```

```
✓ 0.0s
```

|        | 0           | 1           |
|--------|-------------|-------------|
| const  | -244.252410 | -164.714462 |
| height | 4.945766    | 6.132272    |

**Interpretation:** our 95% CI for the intercept is abt -244.3 to -164.7.Our 95% CI for the coefficient of height is abt 4.9 to abt 6.1

#### f)Results summary

# The Results Summary

Instead of extracting these individual values from the results object, there is also a method `summary` that will print out all of this information at once. It can get overwhelming, but take a look and see if you can find all of the information extracted above:

- F-statistic and F-statistic p-value
- R-Squared (coefficient of determination)
- Parameters
- Parameter p-values
- Parameter confidence intervals

## ► Hints (click to reveal)

```
print(results.summary())
```

[14] ✓ 0.0s

Python

```
...                                OLS Regression Results
=====
Dep. Variable:          weight    R-squared:         0.955
Model:                 OLS      Adj. R-squared:      0.953
Method:                Least Squares   F-statistic:     384.8
Date:        Thu, 07 Nov 2024   Prob (F-statistic): 1.35e-13
Time:             12:45:29      Log-Likelihood:   -64.112
No. Observations:      20      AIC:                  132.2
Df Residuals:          18      BIC:                  134.2
Df Model:                   1
Covariance Type:    nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
const    -204.4834    18.929    -10.802      0.000    -244.252    -164.714
height      5.5390     0.282     19.616      0.000      4.946      6.132
=====
Omnibus:           2.588    Durbin-Watson:       2.053
Prob(Omnibus):      0.274    Jarque-Bera (JB):    1.245
Skew:               0.202    Prob(JB):          0.537
Kurtosis:           1.846    Cond. No.          902.
=====
```

This summary contains a **lot** of information. If you're curious about all of the fields, feel free to expand explanation:

#### ▼ Regression results explanation (click to expand)

The left part of the first table gives some specifics on the data and the model:

- **Dep. Variable:** Singular. Which variable is the point of interest of the model
- **Model:** Technique used, an abbreviated version of Method (see methods for more).
- **Method:** The loss function optimized in the parameter selection process. Least Squares since it picks the parameters that reduce the training error. This is also known as Mean Squared Error (MSE).
- **No. Observations:** The number of observations used by the model, or size of the training data.
- **Df Residuals:** Degrees of freedom of the residuals, which is the number of observations minus the number of parameters. Intercept is a parameter. The purpose of Degrees of Freedom is to reflect the impact of descriptive/summarizing statistics in the model, which in regression is the coefficient. Since the observations must "live up" to these parameters, they only have so many free observations, and the rest must be reserved to "live up" to the parameters' prophecy. This internal mechanism ensures that there are enough observations to match the parameters.
- **Df Model:** The number of parameters in the model (not including the constant/intercept term if present)
- **Covariance Type:** Robust regression methods are designed to be not overly affected by violations of assumptions by the underlying data-generating process. Since this model is Ordinary Least Squares, it is non-robust and therefore highly sensitive to outliers.

The right part of the first table shows the goodness of fit:

- **R-squared:** The coefficient of determination, the Explained Sum of Squares divided by Total Sum of Squares. This translates to the percent of variance explained by the model. The remaining percentage represents the variance explained by error, the E term, the part that model and predictors fail to grasp.
- **Adj. R-squared:** Version of the R-Squared that penalizes additional independent variables.
- **F-statistic:** A measure of how significant the fit is. The mean squared error of the model divided by the mean squared error of the residuals. Feeds into the calculation of the P-Value.
- **Prob (F-statistic):** The probability that a sample like this would yield the above statistic, and whether the model's verdict on the null hypothesis will consistently represent the population. Does not measure effect magnitude, instead measures the integrity and consistency of this test on this group of data.
- **Log-likelihood:** The log of the likelihood function.
- **AIC:** The Akaike Information Criterion. Adjusts the log-likelihood based on the number of observations and the complexity of the model. Penalizes the model selection metrics when more independent variables are added.
- **BIC:** The Bayesian Information Criterion. Similar to the AIC, but has a higher penalty for models with more parameters. Penalizes the model selection metrics when more independent variables are added.

- complexity of the model. Penalizes the model selection metrics when more independent variables are added.
- **BIC**: The Bayesian Information Criterion. Similar to the AIC, but has a higher penalty for models with more parameters. Penalizes the model selection metrics when more independent variables are added.

The second table shows the coefficient report:

- **coef**: The estimated value of the coefficient. By how much the model multiplies the independent variable by.
- **std err**: The basic standard error of the estimate of the coefficient. Average distance deviation of the points from the model, which offers a unit relevant way to gauge model accuracy.
- **t**: The t-statistic value. This is a measure of how statistically significant the coefficient is.
- **P > |t|**: P-value that the null-hypothesis that the coefficient = 0 is true. If it is less than the confidence level, often 0.05, it indicates that there is a statistically significant relationship between the term and the response.
- **[95.0% Conf. Interval]**: The lower and upper values of the 95% confidence interval. Specific range of the possible coefficient values.

The third table shows information about the residuals, autocorrelation, and multicollinearity:

- **Skewness**: A measure of the symmetry of the data about the mean. Normally-distributed errors should be symmetrically distributed about the mean (equal amounts above and below the line). The normal distribution has 0 skew.
- **Kurtosis**: A measure of the shape of the distribution. Compares the amount of data close to the mean with those far away from the mean (in the tails), so model "peakiness". The normal distribution has a Kurtosis of 3, and the greater the number, the more the curve peaks.
- **Omnibus**: Provides a combined statistical test for the presence of skewness and kurtosis.
- **Prob(Omnibus)**: The above statistic turned into a probability
- **Jarque-Bera**: A different test of the skewness and kurtosis
- **Prob (JB)**: The above statistic turned into a probability
- **Durbin-Watson**: A test for the presence of autocorrelation (that the errors are not independent), which is often important in time-series analysis
- **Cond. No**: A test for multicollinearity (if in a fit with multiple parameters, the parameters are related to each other).

# Other Regression Statistics

What else do we have in this report?

- **F-statistic:** The F-test measures the significance of your model relative to a model in which all coefficients are 0, i.e. relative to a model that says there is no correlation whatever between the predictors and the target.
- **Log-Likelihood:** The probability in question is the probability of seeing these data points, *given* the model parameter values. The higher this is, the more our data conform to our model and so the better our fit. AIC and BIC are related to the log-likelihood; we'll talk about those later.
- **coef:** These are the betas as calculated by the least-squares regression. We also have p-values and 95%-confidence intervals.
- **Omnibus:** This is a test for error normality. The probability is the chance that the errors are normally distributed.
- **Durbin-Watson:** This is a test for autocorrelation. We'll return to this topic in a future lecture.
- **Jarque-Bera:** This is another test for error normality.
- **Cond. No.:** The condition number tests for independence of the predictors. Lower scores are better. When the predictors are *not* independent, we can run into problems of multicollinearity. For more on the condition number, see [here](#).
  - from another data- The condition number is large, 1.15e+03. This might indicate that there are strong multicollinearity or other numerical problems.

g) Visualizing our model

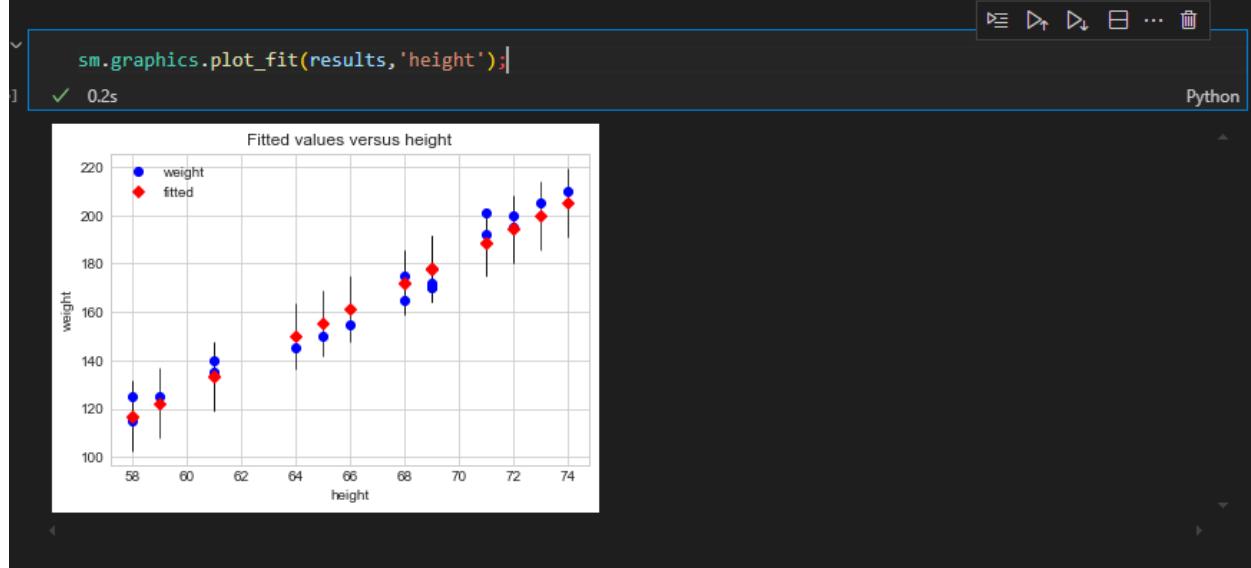
➤ Plot fitting

```
sm.graphics.plot_fit(results,'height');
```

## Plotting Fit

StatsModels also comes with some plotting utilities that are particularly useful for statistical modeling. You can find the full list [here](#).

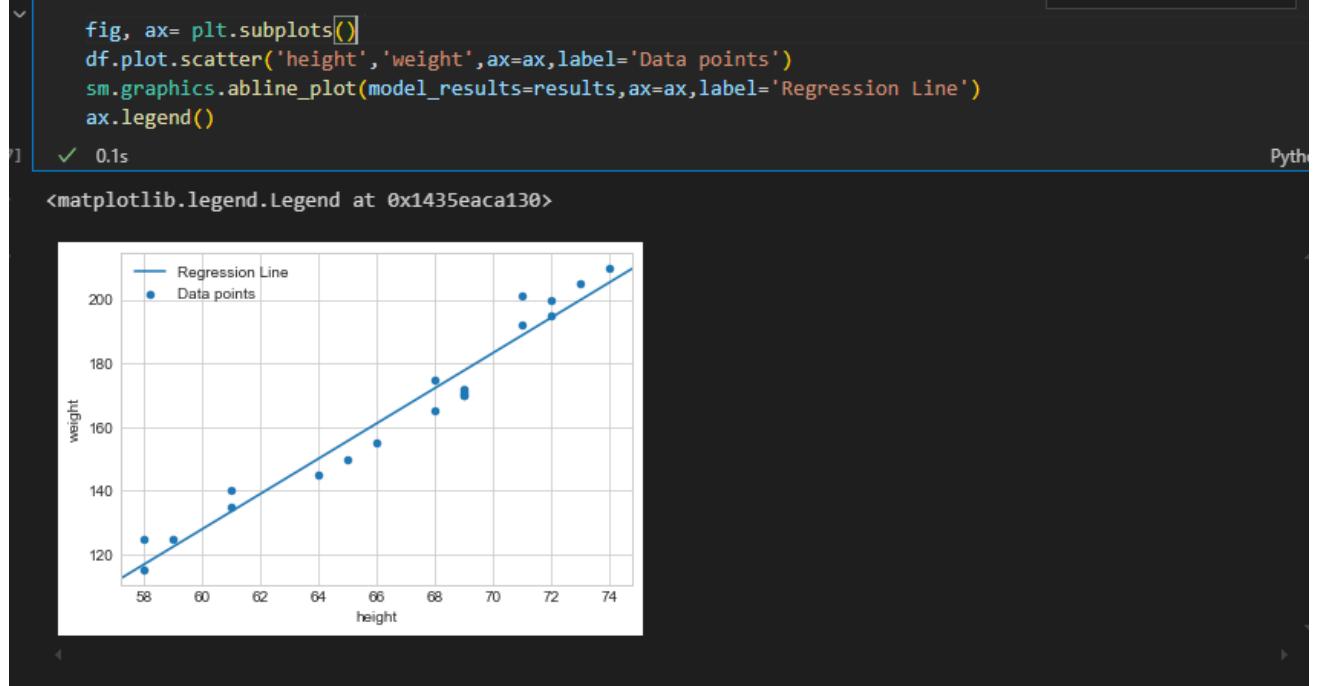
This method shows the scatter plot of actual values as well as points indicating the values predicted by the model. The black vertical lines represent the confidence intervals for each prediction. (These confidence intervals use the t-distribution.)



- For a simpler visualization we can use abline\_plot

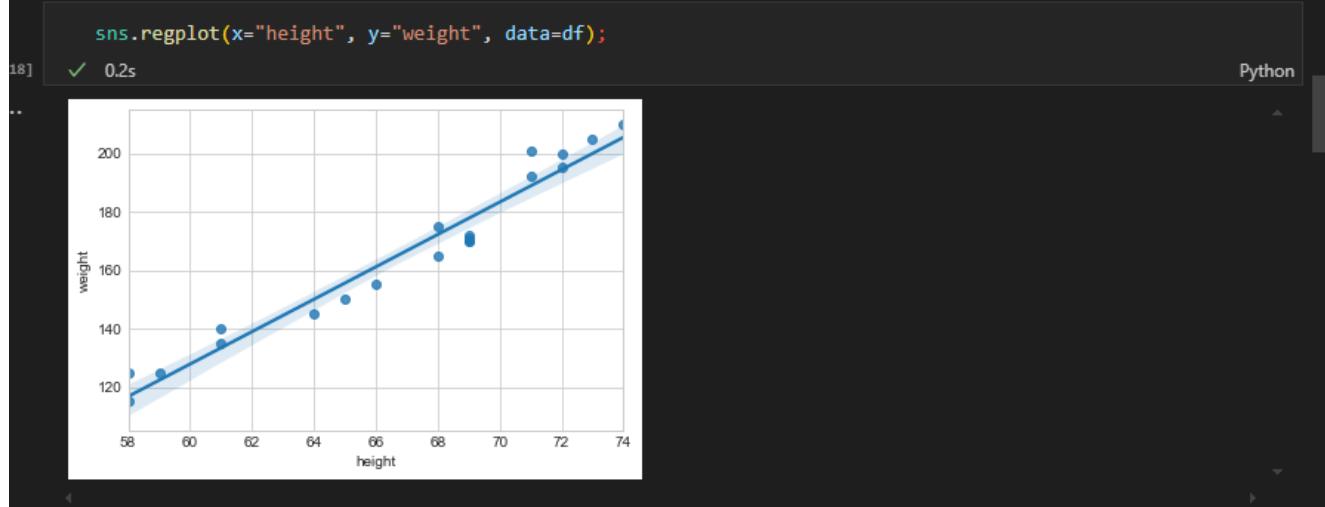
```
fig, ax= plt.subplots()
df.plot.scatter('height','weight',ax=ax,label='Data points')
sm.graphics.abline_plot(model_results=results,ax=ax,label='Regression Line')
ax.legend()
```

For a simpler visualization, you can use `abline_plot` from StatsModels to add a best-fit line to a regular scatter plot.



#### ➤ Seaborn regplot- visualizing original data alongside the regression line

There is also a Seaborn function, `regplot` ([documentation here](#)), that can be useful for visualizing the original data alongside the regression line. In this case the shaded region also represents the confidence interval, which is computed using a different technique called bootstrapping.



#### h) Plotting residuals

**Model residuals** – Differences between the true values and the values predicted by the model. We can get them easily from the model results using the `resid` attribute.

```
results.resid
0    0.0s
0    -7.169872
1    12.213070
2     6.603263
3    -7.708891
4     3.213070
5     8.220320
6     0.674051
7     5.135032
8    -1.779680
9     4.596012
10    1.603263
11    2.681301
12    -5.708891
13    2.830128
14    -5.013795
15    -7.708891
16     5.674051
17    -6.091834
18    -5.552814
19    -6.708891
dtype: float64
```

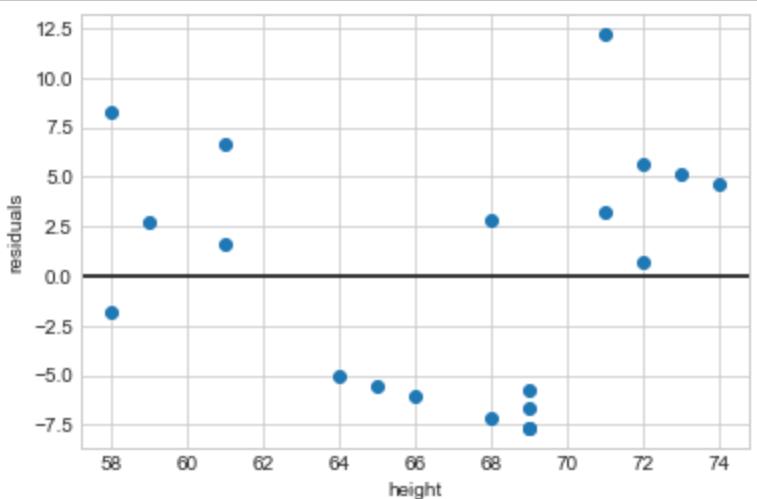
Python

As you can see, model residuals can be negative or positive depending on whether the model guessed too high or too low of a value.

```
fig, ax = plt.subplots()
ax.scatter(df['height'], results.resid)
ax.axhline(y=0, c='k')
ax.set_xlabel("height")
ax.set_ylabel("residuals");
```

```
fig, ax = plt.subplots()
ax.scatter(df['height'], results.resid)
ax.axhline(y=0, c='k')
ax.set_xlabel("height")
ax.set_ylabel("residuals");
```

✓ 0.1s

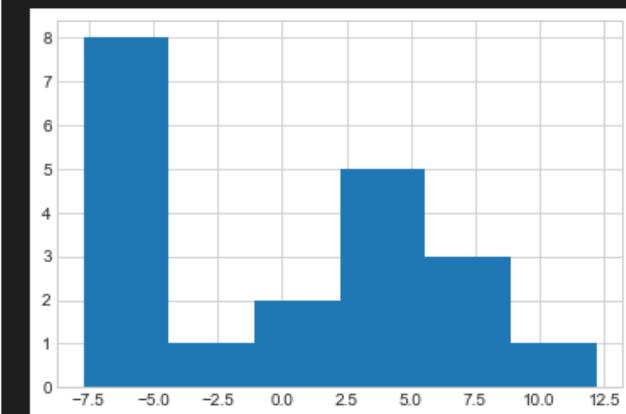


## ➤ Histogram to check distribution of the residuals

```
plt.hist(results.resid,bins='auto')
```

```
plt.hist(results.resid,bins='auto')
✓ 0.1s

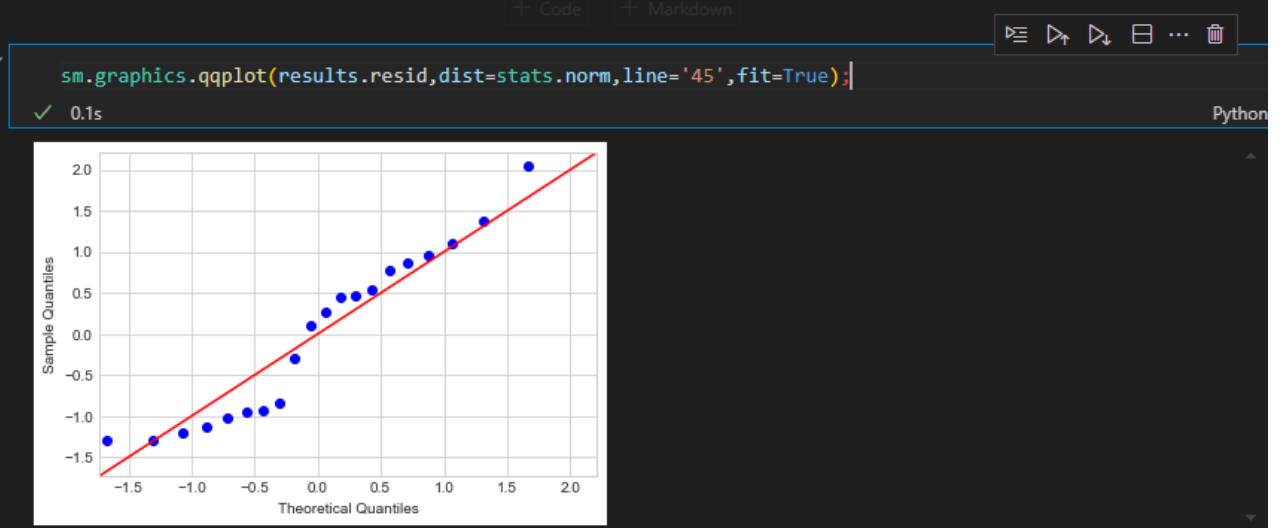
(array([8., 1., 2., 5., 3., 1.]),
 array([-7.0889135, -4.38856443, -1.06823751,  2.25208942,  5.57241634,
        8.89274326, 12.21307018]),
 <BarContainer object of 6 artists>)
```



- **Q-Q plot** – It compares the quantiles of the residuals to the quantiles of theoretical normal distribution. The farther from the line that the data point appear, the farther from a normal distribution they are

```
sm.graphics.qqplot(results.resid,dist=stats.norm,line='45',fit=True);
```

Another tool we might use to visualize the distribution of the residuals is called a Q-Q plot. It compares the quantiles of the residuals to the quantiles of a theoretical normal distribution. The farther from the line that the data points appear, the farther from a normal distribution they are.



## Why Visualize Residuals?

### Why visualize residuals

The main purpose of visualizing the residuals is to help you better understand where your model is performing well, and where it is performing poorly. For example, some models perform better on smaller values than larger values.

You might use this information to improve the model, or simply to communicate transparently about its strengths and weaknesses. These plots will also become relevant when investigating whether the model assumptions are being met.

## 84. R-Style Regression Formulas

We have been using the primary (`api`) interface to StatsModels. There is also an interface (`formula.api`) that uses R-style formulas. R-style meaning that the formulas for the models are written in the same style as they would be written in the programming language [R]([https://en.wikipedia.org/wiki/R\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/R_(programming_language))).

Unlike Python, R has built-in functionality for linear modeling, and so you will find that some foundational statistics and data science resources and papers used R rather than Python. StatsModels includes this R-style interface to be more convenient for people who learned R first. You can read more about it [here]([https://www.statsmodels.org/stable/example\\_formulas.html](https://www.statsmodels.org/stable/example_formulas.html)).

Below we provide an example of the same model above, except developed using an R-style formula.

```

import statsmodels.formula.api as smf
rstyle_model = smf.ols(formula='weight ~ height', data=df)
rstyle_results = rstyle_model.fit()
print(rstyle_results.summary())

```

Below we provide an example of the same model above, except developed using an R-style formula.

```

import statsmodels.formula.api as smf
rstyle_model = smf.ols(formula='weight ~ height', data=df)
rstyle_results = rstyle_model.fit()
print(rstyle_results.summary())

```

[23] ✓ 0.0s Python

...

OLS Regression Results

| Dep. Variable:    | weight           | R-squared:          | 0.955             |       |          |          |
|-------------------|------------------|---------------------|-------------------|-------|----------|----------|
| Model:            | OLS              | Adj. R-squared:     | 0.953             |       |          |          |
| Method:           | Least Squares    | F-statistic:        | 384.8             |       |          |          |
| Date:             | Thu, 07 Nov 2024 | Prob (F-statistic): | 1.35e-13          |       |          |          |
| Time:             | 12:45:30         | Log-Likelihood:     | -64.112           |       |          |          |
| No. Observations: | 20               | AIC:                | 132.2             |       |          |          |
| Df Residuals:     | 18               | BIC:                | 134.2             |       |          |          |
| Df Model:         | 1                |                     |                   |       |          |          |
| Covariance Type:  | nonrobust        |                     |                   |       |          |          |
|                   | coef             | std err             | t                 | P> t  | [0.025   | 0.975]   |
| Intercept         | -204.4834        | 18.929              | -10.802           | 0.000 | -244.252 | -164.714 |
| height            | 5.5390           | 0.282               | 19.616            | 0.000 | 4.946    | 6.132    |
| Omnibus:          |                  | 2.588               | Durbin-Watson:    |       | 2.053    |          |
| Prob(Omnibus):    |                  | 0.274               | Jarque-Bera (JB): |       | 1.245    |          |
| Skew:             |                  | 0.202               | Prob(JB):         |       | 0.537    |          |
| Kurtosis:         |                  | 1.846               | Cond. No.         |       | 902.     |          |

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The only difference in the results between using the R-style formula `ols` and the `OLS` we used previously is that one of the parameters is called `Intercept` rather than `const`. This is because `ols` automatically adds a constant to the `X` values, whereas with `OLS` you need to use `sm.add_constant`. Both forms result in an intercept term, they just have different names.

In general, we recommend using the `OLS` interface rather than the R-style formula interface because building the formula can get increasingly complicated as we move from simple linear regression to multiple regression. But you may see examples using either, so it's useful to be able to interpret both forms.

As a stakeholder holder what are we looking at variance or more sales

## Regression Without Error Model in statsmodels

```
sm.formula.ols(formula = "y ~ x", data = test_df).fit().summary()
```

Experiment: Playing with regression line

# Regression Without Error in `statsmodels`

```
D ▾
[17] sm.formula.ols(formula = "y ~ x", data = test_df).fit().summary()
```

...

OLS Regression Results

|                   |                  |                     |           |
|-------------------|------------------|---------------------|-----------|
| Dep. Variable:    | y                | R-squared:          | 1.000     |
| Model:            | OLS              | Adj. R-squared:     | 1.000     |
| Method:           | Least Squares    | F-statistic:        | 3.321e+31 |
| Date:             | Thu, 07 Nov 2024 | Prob (F-statistic): | 7.48e-274 |
| Time:             | 10:58:12         | Log-Likelihood:     | 611.51    |
| No. Observations: | 20               | AIC:                | -1219.    |
| Df Residuals:     | 18               | BIC:                | -1217.    |
| Df Model:         | 1                |                     |           |

Covariance Type: nonrobust

|           | coef   | std err  | t        | P> t  | [0.025 | 0.975] |
|-----------|--------|----------|----------|-------|--------|--------|
| Intercept | 5.0000 | 5.79e-15 | 8.64e+14 | 0.000 | 5.000  | 5.000  |
| x         | 3.0000 | 5.21e-16 | 5.76e+15 | 0.000 | 3.000  | 3.000  |

Omnibus: 43.228 Durbin-Watson: 0.004

Prob(Omnibus): 0.000 Jarque-Bera (JB): 3.335

Skew: 0.201 Prob(JB): 0.189

Kurtosis: 1.040 Cond. No. 21.5

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

# Regression with Error in statsmodels

Now let's add a little noise:

```
x = np.arange(20)
y = np.array([3*pt + 5 + gauss(mu=0, sigma=5) for pt in x])

df2 = pd.DataFrame(columns=['x', 'y'])

df2['x'] = x
df2['y'] = y

df2.head(10)
```

[18]

Python

...

|   | x | y         |
|---|---|-----------|
| 0 | 0 | 3.258875  |
| 1 | 1 | 9.581259  |
| 2 | 2 | 5.902383  |
| 3 | 3 | 14.512547 |
| 4 | 4 | 19.312524 |
| 5 | 5 | 21.504258 |
| 6 | 6 | 26.446434 |
| 7 | 7 | 25.862669 |
| 8 | 8 | 28.466996 |
| 9 | 9 | 27.834786 |

▷

```
model = sm.formula.ols(formula='y~x', data=df2).fit()
|
model.summary()
```

[19]

Python

## Introduction to Linear Regression - Recap

# Introduction to Linear Regression - Recap



## Introduction

In this section you learned about a statistical model, linear regression, that can be used to model the relationship between an independent and dependent variable.

## Key Takeaways

- **Statistical modeling** combines statistics and data to draw conclusions about real-world relationships
  - When defining a statistical model, you use data understanding to define an **independent variable** (x) and a **dependent variable** (y)
- A **simple linear regression** model defines the relationship between x and y as a straight line,  $y = mx + c$ 
  - The fitted model **parameters** are the best-fit values for m (slope) and c (intercept)
  - The **least-squares** method allows you to calculate a "closed-form" best fit by minimizing the model error
- You can **evaluate** linear regression models overall, as well as their parameters
  - The **F-statistic** indicates whether the model is statistically significant overall
  - **R-Squared** (coefficient of determination) describes the amount of variance in the dependent variable that is explained by the model
  - Parameters have **coefficient** values that describe the change in the dependent variable associated with a unit change in the independent variable, as well as **p-values** and **confidence intervals** created using the t-distribution
- **StatsModels** is a useful library for fitting and evaluating linear regression models, using the **OLS** class
  - You can also use StatsModels to find the model **residuals**, i.e. the differences between the true values and the values predicted by the model

## 85. ASSUMPTIONS OF LINEAR REGRESSION

Like any other statistical technique, the coefficients and p-values of a linear regression model are based on underlying assumptions about the variables and the relationships between them.

### REGRESSION ASSUMPTIONS AND MODEL DIAGNOSTICS

When you fit a regression model, the algorithm will find a best fit line **according to how you have defined the model**. If the model is ill-defined, you will still get coefficients and p-values, but they might not actually describe the true relationship between the underlying variables.

```
import numpy as np
import statsmodels.api as sm

X=np.linspace(1,10).reshape(-1,1)
y= np.exp(X)

model = sm.OLS(y,sm.add_constant(X))
results = model.fit()

print(f'''
model p-value : {results.f_pvalue}
model R-Squared: {results.rsquared}
coef p-value : {results.pvalues[1]}
coef value : {results.params[1]}
''' )
```

For example, let's say you fit this model:

```
import numpy as np
import statsmodels.api as sm

X=np.linspace(1,10).reshape(-1,1)
y= np.exp(X)

model = sm.OLS(y,sm.add_constant(X))
results = model.fit()

print(f'''
```

```
model p-value : {results.f_pvalue}
model R-Squared: {results.rsquared}
coef p-value : {results.pvalues[1]}
coef value : {results.params[1]}
''')
```

-We are explaining about 51% of the variance in y(r squared),and for each increase of 1 in X you see an associated increase of about 1.3 k in y.The model overall ad coefficients of X are statistically significant.

But what's the problem here.lets look at a graph?

```
import matplotlib.pyplot as plt

fig,ax = plt.subplots()
ax.scatter(X,y,label='data points')
ax.plot(X,results.predict(sm.add_constant(X)),label='best fit line')
ax.legend();
```

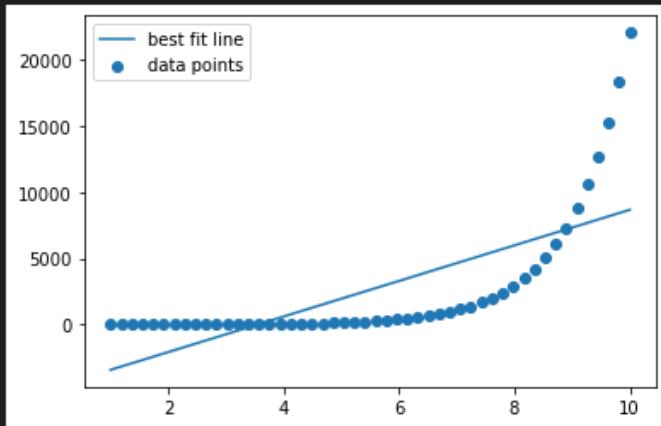
Well, let's look at the graph:

```
import matplotlib.pyplot as plt

fig,ax = plt.subplots()
ax.scatter(X,y,label='data points')
ax.plot(X,results.predict(sm.add_constant(X)),label='best fit line')
ax.legend();
```

✓ 0.1s

Python



Just by looking at that graph, you can tell that this model is not actually a good fit for the data. The ordinary least squares algorithm found a line that would maximize the explained error, but that doesn't mean you would actually want to use this model for inference or prediction.

Linear regression assumptions formalize the ways that a model can be "off" and give you tools beyond just visualization in order to diagnose the problem!

"What are the assumptions of linear regression?" is also a common technical interview question, so we recommend that you use the acronym LINE to remember them: Linearity, Independence, Normality, Equal Variance (Homoscedasticity).

By looking at the graph, you can tell that this model is not actually a good fit for the data. OLS (ordinary least squares) algorithm found a line that would maximize the explained error, but that doesn't mean you would actually want to use this model for inference or prediction.

**Linear Regression assumptions** formalize the ways that a model can be 'off' and give you tools beyond just visualization in order to diagnose the problem.

**MODEL ASSUMPTIONS** – In the context of statistical modelling, especially in linear regression, model assumptions refer to the set of conditions that the underlying data and model need to satisfy for the results to be valid and reliable. These assumptions are critical because if they are violated, the estimates obtained from the model may be biased, inefficient, or misleading.

**Rule of Thumb:**

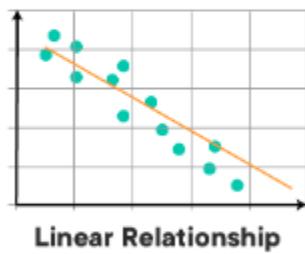
If the p-value is **less than 0.05**, you reject the null hypothesis ( $H_0$ ).

If the p-value is **greater than or equal to 0.05**, you fail to reject the null hypothesis.

- ✓ **LINEARITY**- requires that there is a linear relationship between the response variable (y) and predictor (X). Linear means that the change in y by 1-unit change in X, is constant

## 1. Linearity

The linearity assumption requires that there is a **linear relationship** between the response variable (y) and predictor (X). Linear means that the change in y by 1-unit change in X, is constant.



# Linearity

At this point this formula should be familiar:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_n x_n$$

And so should this type of interpretation of a linear regression coefficient:

an increase of 1 unit of independent variable  $x$  is associated with a change of coefficient  $\beta$  in the dependent variable  $y$

In both formula and interpretation, we are assuming that it's valid to fit a **straight line** with a given slope and intercept through our independent and dependent variable. This is true for both simple linear regression, where we have a single slope and intercept, and for multiple regression, where we have straight lines in multiple dimensions, each with their own slopes, forming a hyperplane.

**Linear regression will assume that there is a linear relationship and give you coefficient values regardless of whether there is actually a linear relationship** between the variables. So to ensure the validity of your model, you'll want to investigate linearity.

## Visualizing Linearity

Visualizations are a key component of model diagnostics because they can not only indicate that there is a *problem* but also can help guide you towards a *solution*.

If a relationship is not linear, you should be thinking of ways that you might try to transform the variables so that the relationship becomes linear. For example, you might want to apply a **log transformation, or a polynomial transformation**.

### Simple Linear Regression

Visualizing linearity with a simple linear regression is very straightforward. Simply plot the independent variable on the x-axis and the dependent variable on the y-axis.

For example, here is a plot of synthetic data where we know that the underlying relationship is  $y = -5x + 2$  (plus some noise):

#### 1. Using scatter plot

```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('fivethirtyeight')

np.random.seed(2)
x=np.linspace(0,10,300)
y = -5 * x + 2 + np.random.normal(scale=5,size=300)

fig, ax = plt.subplots()
```

```
ax.scatter(x, y, color="green")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Linear Relationship X vs. Y");
np.corrcoef(x,y)[0,1]
```

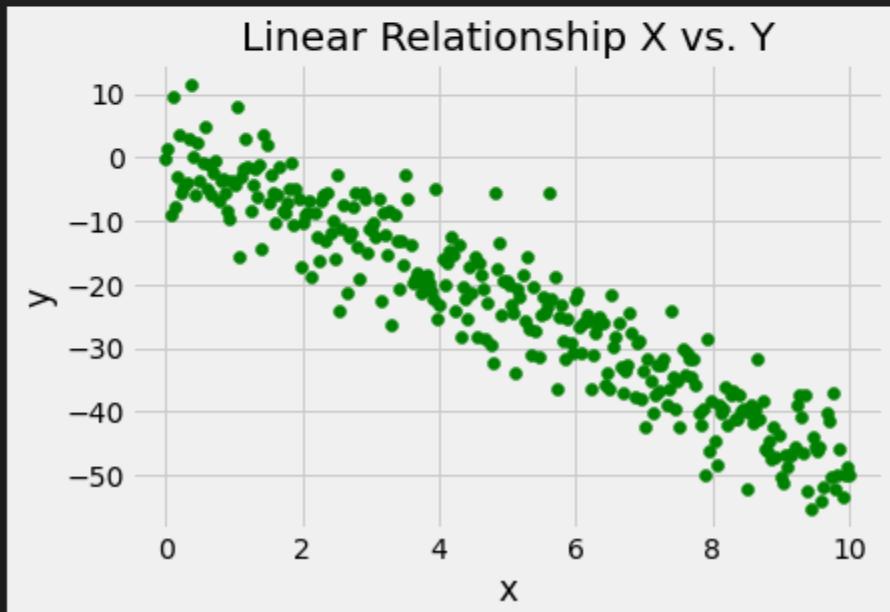
```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('fivethirtyeight')

np.random.seed(2)
x=np.linspace(0,10,300)
y = -5 * x + 2 + np.random.normal(scale=5,size=300)

fig, ax = plt.subplots()
ax.scatter(x, y, color="green")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Linear Relationship X vs. Y");
np.corrcoef(x,y)[0,1]
```

-0.9411928236109894



Linear relationship between these two axis is clearly visible

Lets compare this to the below plot, which uses weight and mpg from the Auto MPG Dataset

```
import pandas as pd
import statsmodels.api as sm

data = pd.read_csv("data/auto-mpg.csv")

y = data["mpg"]
x = data["weight"]

fig, ax = plt.subplots()
ax.scatter(x, y)
ax.set_xlabel("weight")
ax.set_ylabel("mpg")
ax.set_title("Not So Linear Relationship X vs. Y");
np.corrcoef(x,y)[0,1]
```

Compare this to the below plot, which uses `weight` and `mpg` from the Auto MPG dataset:

```
import pandas as pd
import statsmodels.api as sm

data = pd.read_csv("data/auto-mpg.csv")

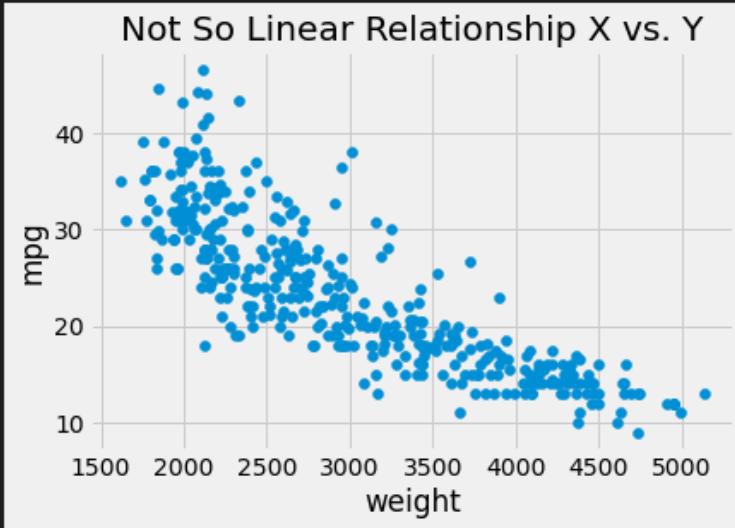
y = data["mpg"]
x = data["weight"]

fig, ax = plt.subplots()
ax.scatter(x, y)
ax.set_xlabel("weight")
ax.set_ylabel("mpg")
ax.set_title("Not So Linear Relationship X vs. Y");
np.corrcoef(x,y)[0,1]
```

✓ 2.9s Python

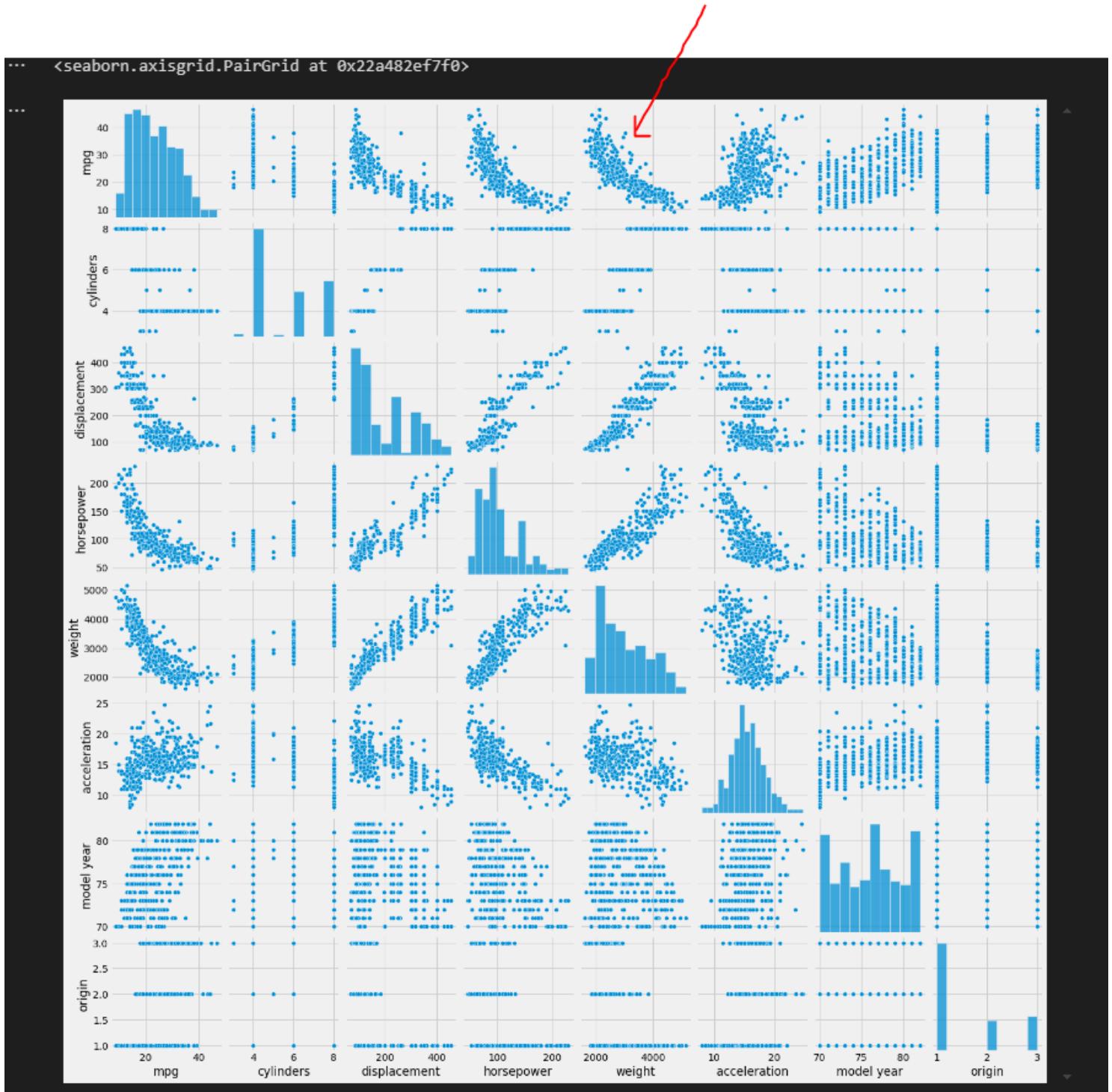
```
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning
from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning
from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

-0.8322442148315754



- We can also use pairplot

```
import seaborn as sns
sns.pairplot(data)
```



Note the curvature in the **weight X mpg** plot compared to the previous one. It looks like the slope is steeper for lower values of 'weight', then less steep as 'weight' increases. This inconsistent slope is an indication that the relationship might not be linear.

### Multiple linear regression

```
np.random.seed(1)
x1 = np.linspace(start=0, stop=10, num=300) + np.random.normal(size=300)
```

```
x2 = np.linspace(start=0, stop=5, num=300) + np.random.normal(size=300)
y = -5 * x1 + 10 * x2 + 2 + np.random.normal(size=300)

fig, axes = plt.subplots(ncols=2, figsize=(15,5))

axes[0].scatter(x1, y, color="orange")
axes[0].set_xlabel("x1")
axes[0].set_ylabel("y")

axes[1].scatter(x2, y, color="orange")
axes[1].set_xlabel("x2")
axes[1].set_ylabel("y")

fig.suptitle("Don't Do This to Evaluate Linearity: Individual X vs. Y for Multiple Regression");
```

## Multiple Linear Regression

Visualizing linearity in multiple regression is a bit more complicated than with simple regression, because the model you are fitting can't be expressed in a 2D plot.

In fact, using a 2D plot for a multiple regression model can actually make you think that a relationship isn't linear when it actually is!

Let's say the underlying relationship is  $y = -5x_1 + 10x_2 + 2$



You can somewhat see the linear relationship with  $x_2$  but  $x_1$  vs  $y$  appears to be a random distribution of points. And yet we know there is a linear relationship between  $x_1$  and  $y$ , because we defined the  $y$  variable ourselves.

-The two ways to avoid this issue are **plotting model residuals** and **partial regression plots**. Note that both of these approaches mean that you need to create a model before you can evaluate the linearity of the relationship.

### a) Plotting Model residuals

If the model is correctly specified, then we should not see curvature in the residuals. Let's plot the residuals for the  $y = -5x_1 + 10x_2 + 2$  data generated in the previous example.

```
X = pd.DataFrame()
X["x1"] = x1
X["x2"] = x2
linear_model = sm.OLS(y, sm.add_constant(X))
linear_results = linear_model.fit()

fig, ax = plt.subplots()

ax.scatter(y, linear_results.resid, color="green")
ax.axhline(y=0, color="black")
ax.set_xlabel("y")
ax.set_ylabel("residuals")
ax.set_title("Linear Relationship Residual Plot");
```

## Plotting Model Residuals

If the model is correctly specified, then **we should not see curvature in the residuals**. Let's plot the residuals for the  $y = -5x_1 + 10x_2 + 2$  data generated in the previous example.

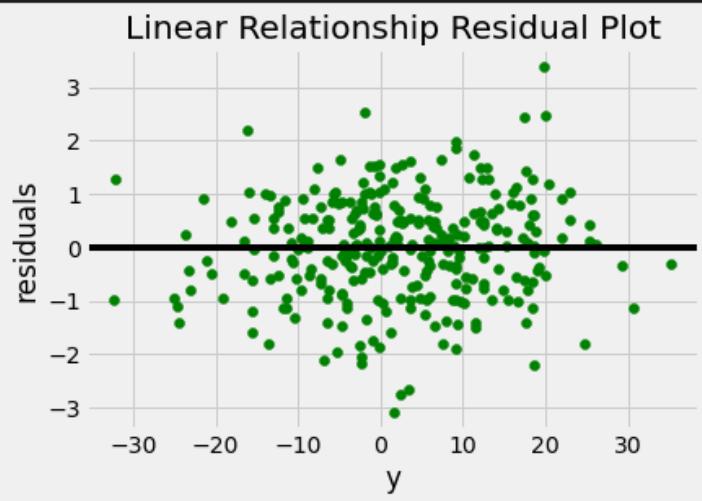
```
X = pd.DataFrame()
X["x1"] = x1
X["x2"] = x2
linear_model = sm.OLS(y, sm.add_constant(X))
linear_results = linear_model.fit()

fig, ax = plt.subplots()

ax.scatter(y, linear_results.resid, color="green")
ax.axhline(y=0, color="black")
ax.set_xlabel("y")
ax.set_ylabel("residuals")
ax.set_title("Linear Relationship Residual Plot");
```

[20] ✓ 0.1s Python

... c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: Ir x = pd.concat(x[:,order], 1)



By comparison, here is a residual plot for a model that predicts mpg using weight and model year.

By comparison, here is a residual plot for a model that predicts `mpg` using `weight` and `model year`:

```
y = data["mpg"]
X = data[["weight", "model year"]]

non_linear_model = sm.OLS(y, sm.add_constant(X))
non_linear_results = non_linear_model.fit()

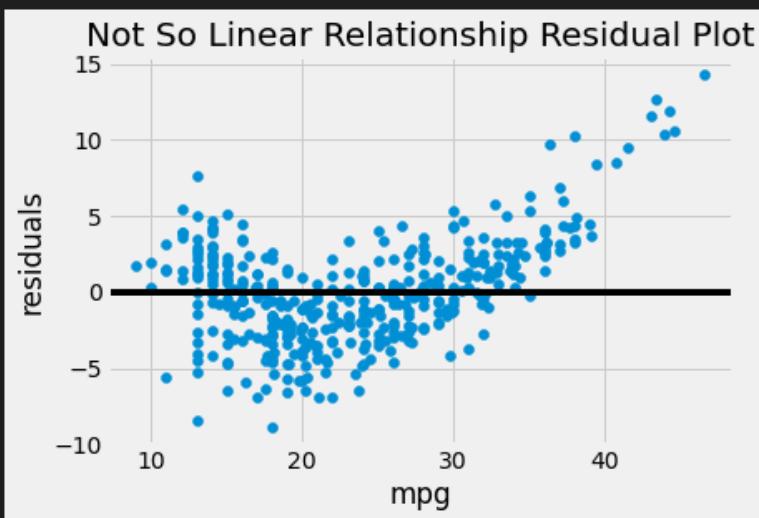
fig, ax = plt.subplots()

ax.scatter(y, non_linear_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("mpg")
ax.set_ylabel("residuals")
ax.set_title("Not So Linear Relationship Residual Plot");

```

[7] ✓ 0.1s Python

... c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: I...  
x = pd.concat(x[::-order], 1)



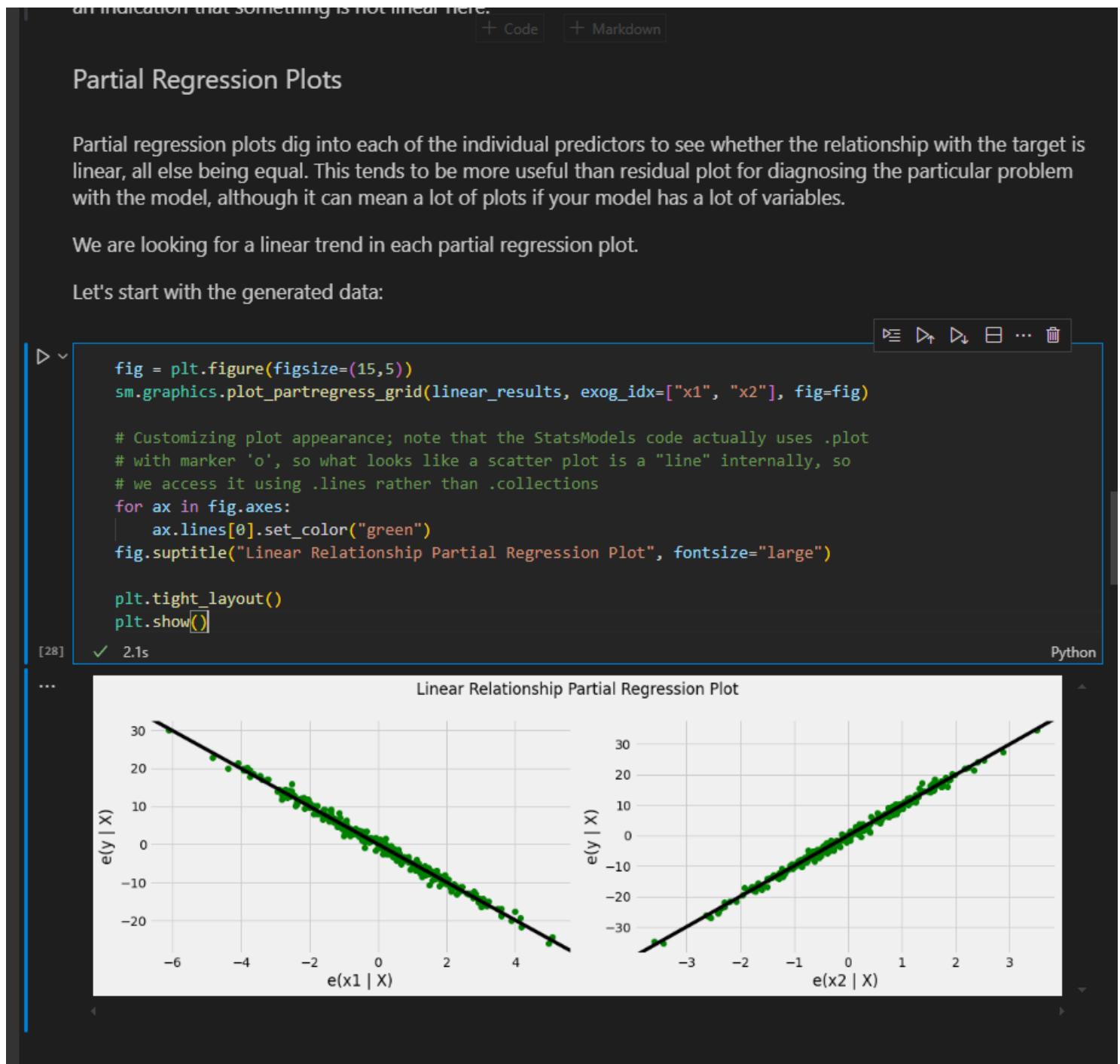
You can see curvature in this plot, just like we saw in the simple linear regression using `weight`. Once again this is an indication that something is not linear here.

## b) Partial Regression Plots

Partial regression plots dig into each of the individual predictors to see whether the relationship with the target is linear, all else being equal. This tends to be more useful than residual plot for diagnosing the particular problem with the model, although it can mean a lot of plots if your model has a lot of variables.

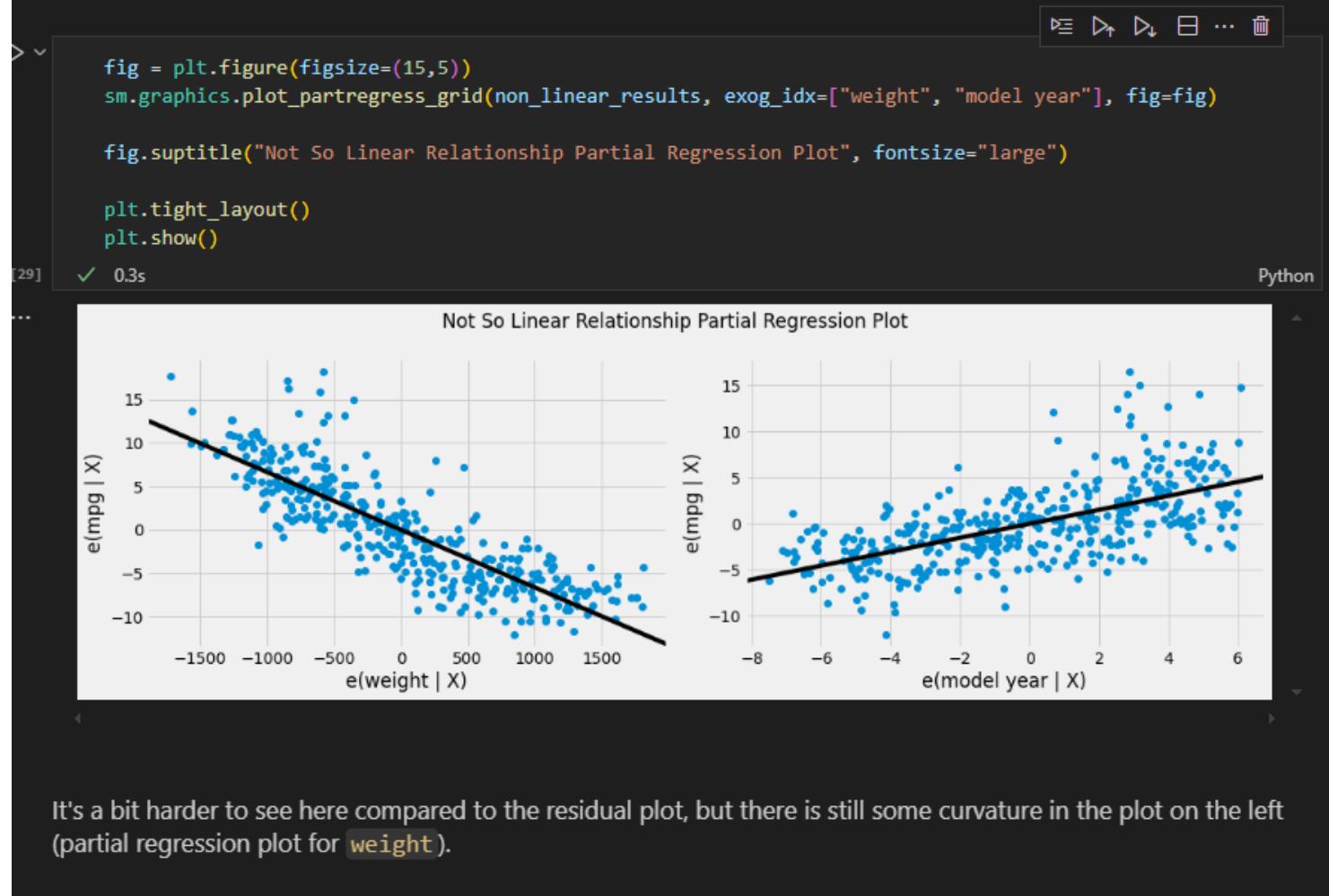
We are looking for a linear trend in each partial regression plot.

Let's start with the generated data:



Compared to when we plotted them individually, now we can clearly see the linear relationships! Let's do the same for the Auto MPG model.

Compared to when we plotted them individually, now we can clearly see the linear relationships! Let's do the same for the Auto MPG model.



It's a bit harder to see here compared to the residual plot, but there is still some curvature in the plot on the left (partial regression plot for `weight`).

## Statistical Test for Linearity

**a) Rainbow test-** linear rainbow is a function that takes in OLS results object and returns a test statistic and a p-value

**Null:** There is no significant difference between the subset model and the full model I,e the relationship is linear

Linear relationship is correctly specified I,e(the relationship between the predictors and the target variable is linear)

**Alternative:** linear model is misspecified, and there is a non-linear relationship between the predictors and the dependent variable

```
from statsmodels.stats.diagnostic import linear_rainbow
linear_rainbow(linear_results)
linear_rainbow(non_linear_results)
```

`linear_rainbow` is a function that takes an OLS results object and returns a test statistic and a p-value.

Below we demonstrate the same test coded above, this time using StatsModels:

```
▶ 
  from statsmodels.stats.diagnostic import linear_rainbow
  linear_rainbow(linear_results)
[35] ✓ 0.0s
... (0.8888211560283967, 0.7634250654991737) Python
```

And here is the output for our Auto MPG model:

```
▶ 
  linear_rainbow(non_linear_results)
[36] ✓ 0.0s
... (1.907684636312535, 4.178078921300679e-06) Python
```

## Interpreting Rainbow Test Results

Just like any other statistical test, there is a null hypothesis, and a low enough p-value indicates that we can reject that null hypothesis.

In the case of the rainbow test, **the null hypothesis** is that there is no significant difference between the subset model and the full model, i.e. that **the relationship is linear**.

A sufficiently low p-value means that we reject the null hypothesis, meaning that the relationship is not linear.

For the two models presented above:

- The first model (generated data) has a **p-value of about 0.76**. This is much higher than the standard alpha of 0.05, so we fail to reject the null hypothesis and can **consider the relationship to be linear**
- The second model (Auto MPG data) has a **p-value of about 0.000004**. This is much lower than the standard alpha of 0.05, so we reject the null hypothesis and **do not consider the relationship to be linear**

Note that this is "opposite" of the p-values you are used to looking at for model significance or coefficient significance. A low p-value in the rainbow test is a "bad" outcome, indicating a *problem* with the model.

\* The first model (generated data) has a **\*\*p-value of about 0.76\*\***. This is much higher than the standard alpha of 0.05, so we fail to reject the null hypothesis and can **\*\*consider the relationship to be linear\*\***

\* The second model (Auto MPG data) has a **\*\*p-value of about 0.000004\*\***. This is much lower than the standard alpha of 0.05, so we reject the null hypothesis and **\*\*do not consider the relationship to be linear\*\***

Note that this is "opposite" of the p-values you are used to looking at for model significance or coefficient significance. A low p-value in the rainbow test is a "bad" outcome, indicating a problem with the model.

## Treating Linearity issues

If your visualization or statistical test is indicating that your modeled relationship is not actually linear, you should consider transforming the feature and/or target in order to create a linear relationship. This is going to be different for every dataset, but it will always need to be a **\*\*non-linear transformation\*\*** (e.g. log transformation, adding interaction terms, etc.). A linear transformation (e.g. standardization) will not resolve linearity issues.

Use a function :

```
def make_model(X,Y):
    model = sm.OLS(endog=Y ,exog=sm.add_constant(X))
    results = model.fit()
    return results
results = make_model(df_auto[ "horsepower"],df_auto[ "displacement"])

linear_rainbow(results)
```

- ✓ **INDEPENDENCE -** The independence assumption has two parts:Independence of features and independence of errors

**Independence of features** - we want to avoid collinearity between features in a multiple regression

**Collinearity** means that the features can be used to predict each other, which causes numerical problems for the regression algorithm and leads to unstable coefficients

## Identifying multicollinearity

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('data/auto-mpg.csv')
data['horsepower'].astype(str).astype(int)
data.head()
```

```
data_pred = data.iloc[:,1:8]
data_pred.head()
```

To illustrate ways to identify multicollinearity, let's have a look at the "auto-mpg" data again.

+ Code + Markdown

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('data/auto-mpg.csv')
data['horsepower'].astype(str).astype(int)
data.head()
```

[5]

Python

|   | mpg  | cylinders | displacement | horsepower | weight | acceleration | model year | origin | car name                  |
|---|------|-----------|--------------|------------|--------|--------------|------------|--------|---------------------------|
| 0 | 18.0 | 8         | 307.0        | 130        | 3504   | 12.0         | 70         | 1      | chevrolet chevelle malibu |
| 1 | 15.0 | 8         | 350.0        | 165        | 3693   | 11.5         | 70         | 1      | buick skylark 320         |
| 2 | 18.0 | 8         | 318.0        | 150        | 3436   | 11.0         | 70         | 1      | plymouth satellite        |
| 3 | 16.0 | 8         | 304.0        | 150        | 3433   | 12.0         | 70         | 1      | amc rebel sst             |
| 4 | 17.0 | 8         | 302.0        | 140        | 3449   | 10.5         | 70         | 1      | ford torino               |

To understand the correlation structure of the predictors, you'll take a copy of the data but this time without the target variable (mpg) in it. It's also advisable to remove the 'car name' column as keeping it in won't lead to meaningful results.

```
data_pred = data.iloc[:,1:8]
data_pred.head()
```

[6]

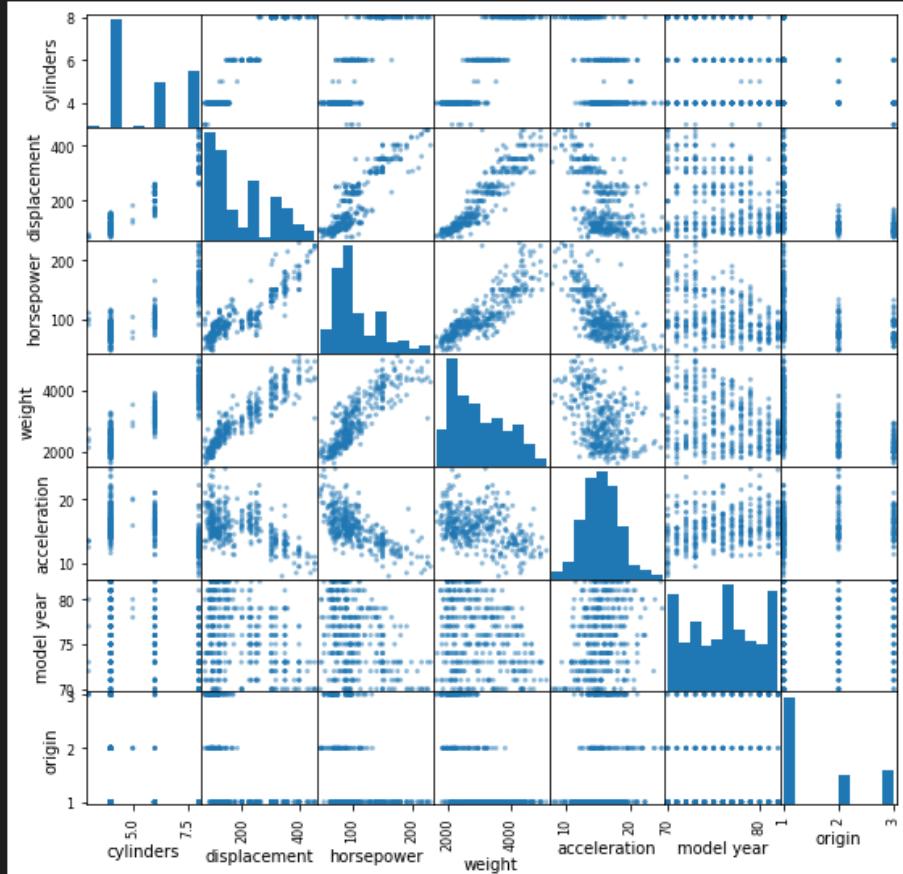
Python

|   | cylinders | displacement | horsepower | weight | acceleration | model year | origin |
|---|-----------|--------------|------------|--------|--------------|------------|--------|
| 0 | 8         | 307.0        | 130        | 3504   | 12.0         | 70         | 1      |
| 1 | 8         | 350.0        | 165        | 3693   | 11.5         | 70         | 1      |
| 2 | 8         | 318.0        | 150        | 3436   | 11.0         | 70         | 1      |
| 3 | 8         | 304.0        | 150        | 3433   | 12.0         | 70         | 1      |
| 4 | 8         | 302.0        | 140        | 3449   | 10.5         | 70         | 1      |

For an initial idea on how the predictors relate, you can take a look at scatterplots between predictors. You can use Pandas to generate a scatter matrix as follows:

```
pd.plotting.scatter_matrix(data_pred,figsize = [9, 9]);  
plt.show()
```

Python



This matrix has the cool feature that it returns scatterplots for relationships between two predictors, and histograms for a single feature on the diagonal. Have a quick look at this. When talking about correlation, what sort of scatter plots will catch our eye? Probably the ones with scatter plots that reveal some sort of linear relationship. For example, weight and displacement seem to be highly correlated. Weight and horsepower as well, and (not surprisingly) displacement and horsepower. This is nice, but it would be hard to examine each plot in detail when having a ton of features. Let's look at the correlation matrix instead. Instead of returning scatter plots and histograms, a correlation matrix returns pairwise correlations. Recall that correlations take a value between -1 and 1, -1 being a perfectly negative linear relationship, and +1 a perfectly positive linear relationship.

```
data_pred.corr()  
abs(data_pred.corr()) > 0.75
```

Histograms, a correlation matrix returns pairwise correlations. Recall that correlations take a value between -1 and +1, -1 being a perfectly negative linear relationship, and +1 a perfectly positive linear relationship.

[8]

data\_pred.corr()

Python

|              | cylinders | displacement | horsepower | weight    | acceleration | model year | origin    |
|--------------|-----------|--------------|------------|-----------|--------------|------------|-----------|
| cylinders    | 1.000000  | 0.950823     | 0.842983   | 0.897527  | -0.504683    | -0.345647  | -0.568932 |
| displacement | 0.950823  | 1.000000     | 0.897257   | 0.932994  | -0.543800    | -0.369855  | -0.614535 |
| horsepower   | 0.842983  | 0.897257     | 1.000000   | 0.864538  | -0.689196    | -0.416361  | -0.455171 |
| weight       | 0.897527  | 0.932994     | 0.864538   | 1.000000  | -0.416839    | -0.309120  | -0.585005 |
| acceleration | -0.504683 | -0.543800    | -0.689196  | -0.416839 | 1.000000     | 0.290316   | 0.212746  |
| model year   | -0.345647 | -0.369855    | -0.416361  | -0.309120 | 0.290316     | 1.000000   | 0.181528  |
| origin       | -0.568932 | -0.614535    | -0.455171  | -0.585005 | 0.212746     | 0.181528   | 1.000000  |

Note that correlations on the diagonal are automatically equal to one as they represent correlations between a variable and the variable itself. So when would you consider a correlation "high"? Generally, a correlation with an absolute value around 0.7-0.8 or higher is considered a high correlation. If we take 0.75 as a cut-off, how many high correlations do we have?

[9]

abs(data\_pred.corr()) > 0.75

Python

|              | cylinders | displacement | horsepower | weight | acceleration | model year | origin |
|--------------|-----------|--------------|------------|--------|--------------|------------|--------|
| cylinders    | True      | True         | True       | True   | False        | False      | False  |
| displacement | True      | True         | True       | True   | False        | False      | False  |
| horsepower   | True      | True         | True       | True   | False        | False      | False  |
| weight       | True      | True         | True       | True   | False        | False      | False  |
| acceleration | False     | False        | False      | False  | True         | False      | False  |
| model year   | False     | False        | False      | False  | False        | True       | False  |
| origin       | False     | False        | False      | False  | False        | False      | True   |

It seems like the variables 'cylinder', 'displacement', 'horsepower', and 'weight' are all pretty highly correlated among each other. Gaining that information was easy from a small correlation matrix, but what if it was larger?

Because corr returns a data frame, a neat fix is to use stack and a subset to return only the highly correlated pairs.

```
# save absolute value of correlation matrix as a data frame
# converts all values to absolute value
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to separate columns
# sort values. 0 is the column automatically generated by the stacking
```

```
df=data_pred.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_1 by default) in a new
# column named "pairs"
df['pairs'] = list(zip(df.level_0, df.level_1))

# set index to pairs
df.set_index(['pairs'], inplace = True)

#drop level columns
df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df.columns = ['cc'] # df.rename(columns=
                 #{0:'cc'},
                 #inplace=True)

# drop duplicates. This could be dangerous if you have variables perfectly correlated with
variables other than themselves.
# for the sake of exercise, kept it in.
df.drop_duplicates(inplace=True)
df[(df.cc>.75) & (df.cc <1)] #df.query('cc >.75 & cc<1')
```

```

# save absolute value of correlation matrix as a data frame
# converts all values to absolute value
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to separate columns
# sort values. 0 is the column automatically generated by the stacking

df=data_pred.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_1 by default) in a new column nam
df['pairs'] = list(zip(df.level_0, df.level_1))

# set index to pairs
df.set_index(['pairs'], inplace = True)

#drop level columns
df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfectly correlated with variables other
# for the sake of exercise, kept it in.
df.drop_duplicates(inplace=True)

```

14] ✓ 0.0s

Python

Now use a similar subset as done previously.

```
df[(df.cc>.75) & (df.cc <1)]
```

7] ✓ 0.0s

Python

**pairs**

|                            | cc       |
|----------------------------|----------|
| (cylinders, displacement)  | 0.950823 |
| (weight, displacement)     | 0.932994 |
| (weight, cylinders)        | 0.897527 |
| (horsepower, displacement) | 0.897257 |
| (horsepower, weight)       | 0.864538 |
| (horsepower, cylinders)    | 0.842983 |

△ 9 ⌂ 0

Spaces: 4 CRLF Cell 12 of 18 ⌂ {}

uv Extreme UV ⌂ ⌂ ⌂ ⌂ 12:25 PM  
11/8/2024

With the variables `cylinder`, `displacement`, `horsepower`, and `weight` so highly correlated, you would typically remove three of them in order to remove collinear features.

A few lines of code, cleverly using pandas, produces a readable list of variable pairs and their correlation. The code can be scaled in case our predictor base grows (sometimes models have 100s of predictors!).

**NOTE** There will be cases when an analyst *should* see all the highly correlated variables, including variables that are correlated 1 to 1. The code can be adjusted to those conditions.

With the variables `'cylinder'`, `'displacement'`, `'horsepower'`, and `'weight'` so highly correlated, you would typically remove *three* of them in order to remove collinear features.

Another option is to use a heatmap to render the correlation matrix as a visualization.



Lets says we found correlation like the following in housing dataset.What variables are highly correlated?

| cc                                 |
|------------------------------------|
| pairs                              |
| (GarageArea, GarageCars) 0.882475  |
| (TotRmsAbvGrd, GrLivArea) 0.825489 |
| (1stFlrSF, TotalBsmtSF) 0.819530   |

Which variables are highly correlated in the Ames Housing data set?

#There are three sets of variables that are highly correlated.

#GarageArea with GarageCars

#Total rooms above ground with Total square feet of living space above ground

#First floor square feet with Total Basement square foot

**Now that we know which variables are correlated with each other, which would you drop from the dataset?**

#\_\_SOLUTION\_\_

"""

Since three different pairs of variables are highly correlated, the correct approach would be to drop one variable from each pair.

One approach would be to drop Garage Area, Total Rooms, and Total Basement Square Feet.

Garage Area: The size of the garage is dependent on how many cars are in it. If you wanted to still keep the information captured by Garage Area, you could create a new variable "Average space per car" before dropping Garage Area.

Total Rooms: There are other variables that count the number of kitchens, bathrooms, bedrooms, etc.

Total Basement Square Feet: The first floor of a building is usually built upon the foundation, which contains the basement. To keep the information that there *\*is\** a basement, you could create a variable "HasBasement", when TotalBsmtSF >0, before deleting the original variable.

"""

Using code we remove columns as

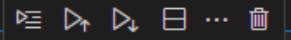
```
ames_preprocessed['AreaPerCar'] = ames_preprocessed['GarageArea']/ames_preprocessed['GarageCars']
ames_preprocessed.drop(columns=['GarageArea','TotRmsAbvGrd','TotalBsmtSF'],inplace=True)
```

ames\_preprocessed

## Address the colinearity

Remove the chosen variables from ames\_preprocessed.

+ Code + Markdown



```
# write code here
ames_preprocessed['AreaPerCar'] = ames_preprocessed['GarageArea']/ames_preprocessed['GarageCars']
ames_preprocessed.drop(columns=['GarageArea', 'TotRmsAbvGrd', 'TotalBsmtSF'], inplace=True)
ames_preprocessed
```

Python

**Independence of errors** - we want to avoid autocorrelation of error. Autocorrelation means that a variable is correlated with itself, so that later values can be predicted based on previous values

- ✓ **NORMALITY ASSUMPTION** – model residuals should follow a normal distribution

Recall that a normal distribution looks something like this:

# Normality

Recall that a normal distribution looks something like this:

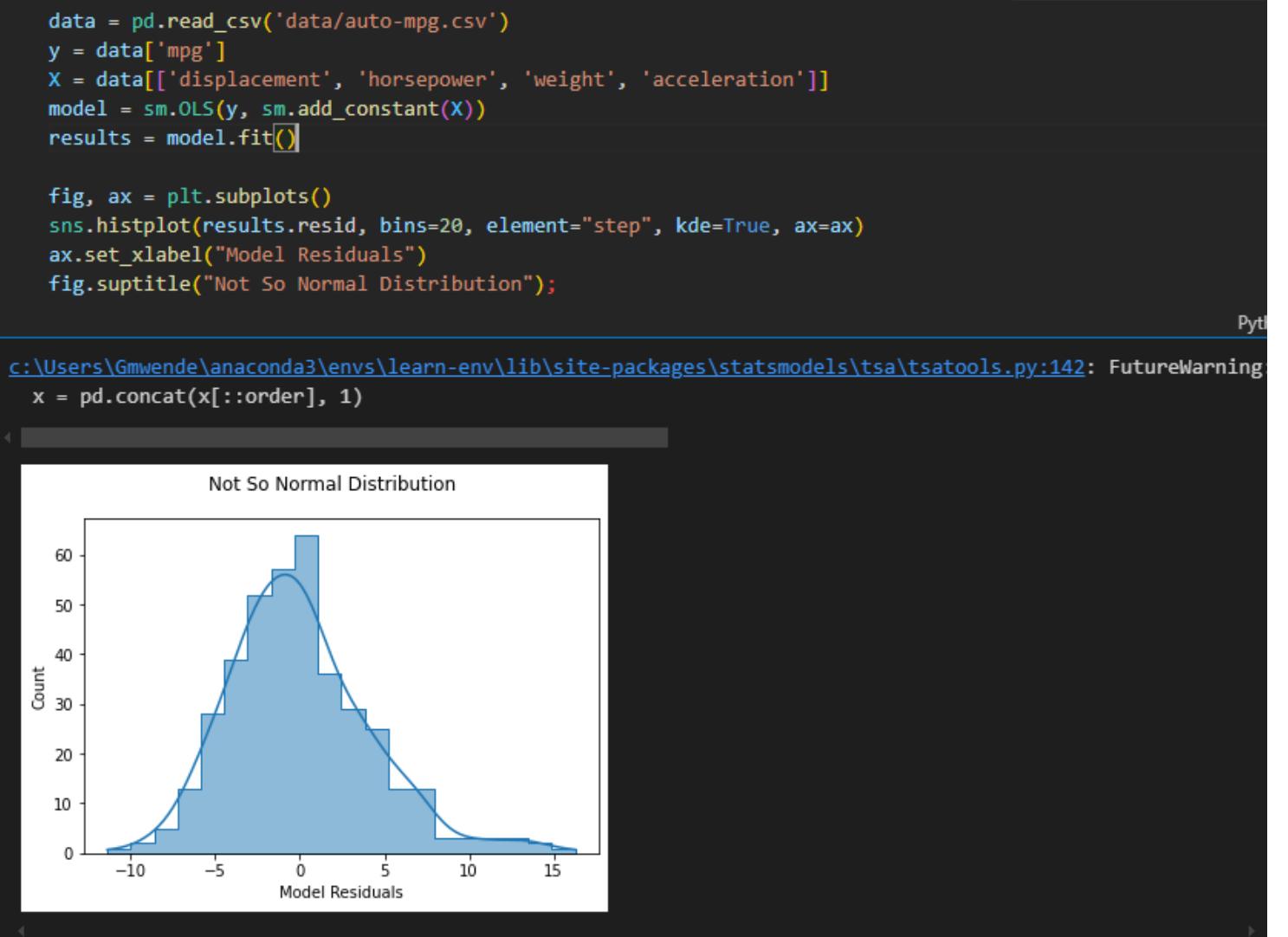


One of the assumptions of linear regression is that the model **residuals** should follow a normal distribution.

## Visualizing Normality

### -Histogram and Q-Q plots

```
data = pd.read_csv('data/auto-mpg.csv')  
y = data['mpg']  
X = data[['displacement', 'horsepower', 'weight', 'acceleration']]  
model = sm.OLS(y, sm.add_constant(X))  
results = model.fit()  
  
fig, ax = plt.subplots()  
sns.histplot(results.resid, bins=20, element="step", kde=True, ax=ax)  
ax.set_xlabel("Model Residuals")  
fig.suptitle("Not So Normal Distribution");
```



As you can see, the residuals from the actual model are a bit skewed, not quite normal.

-Also use Kde plot sns.kdeplot(results.resid)

## Q-Q plot

Below is for normal data

In statistics, a **Q-Q (quantile-quantile) plot** is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other.

We will use the Q-Q plot to compare the *actual* data to *theoretical* quantiles. In this case the theoretical quantiles would be the normal distribution (using the SciPy `stats.norm` function).

When inspecting a Q-Q plot you are looking for the data points to follow the diagonal line as closely as possible.

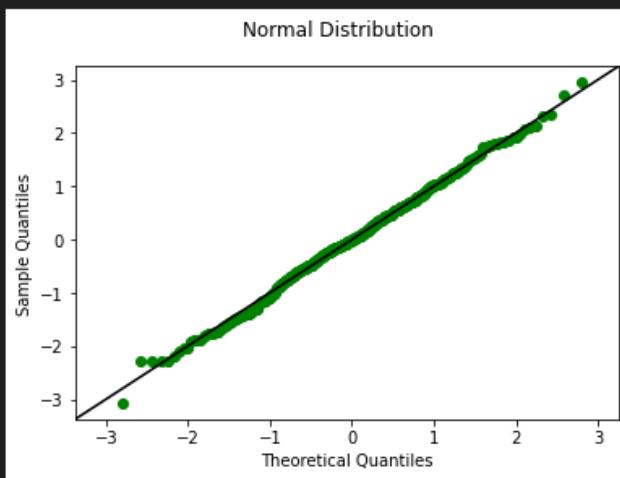
First, here's an example using the generated normal data:

```
# Use qqplot function from StatsModels
fig, ax = plt.subplots()
sm.graphics.qqplot(generated_data, dist=stats.norm, line='45', fit=True, ax=ax)

# Customize plot appearance
scatter = ax.lines[0] ## Scatter points from the Q-Q plot
line = ax.lines[1] ## Reference 45-degree line
scatter.set_markeredgewidth(1)
scatter.set_markeredgecolor("green")
scatter.set_markerfacecolor("green")
line.set_color("black")
fig.suptitle("Normal Distribution");
```

[23]

Python

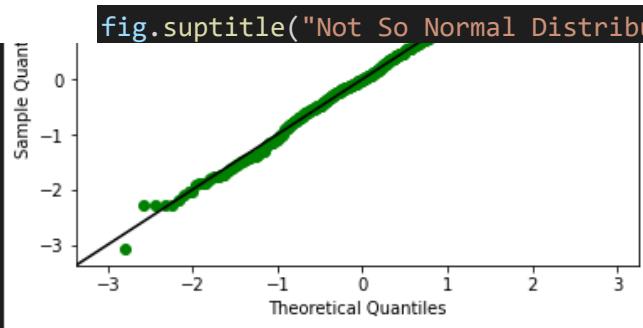


There are a couple of places where the scatter plot diverges from the diagonal line because this data has some pseudo-random noise, but in general the points and the line are very close together.

Lets compare above to the auto mpg model residuals

```
fig, ax = plt.subplots()
sm.graphics.qqplot(results.resid, dist=stats.norm, line='45', fit=True, ax=ax)

# Customize plot appearance
line = ax.lines[1]
line.set_color("black")
```



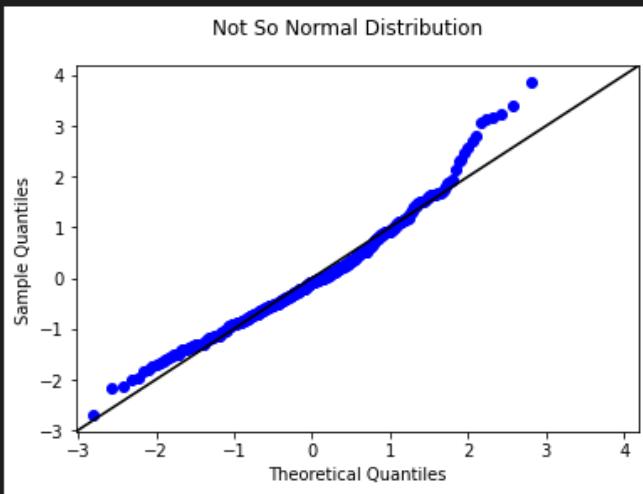
There are a couple of places where the scatter plot diverges from the diagonal line because this data has some pseudo-random noise, but in general the points and the line are very close together.

Compare that to the plot for the auto MPG model residuals:

```
# Use qqplot function from StatsModels
fig, ax = plt.subplots()
sm.graphics.qqplot(results.resid, dist=stats.norm, line='45', fit=True, ax=ax)

# Customize plot appearance
line = ax.lines[1]
line.set_color("black")
fig.suptitle("Not So Normal Distribution");
```

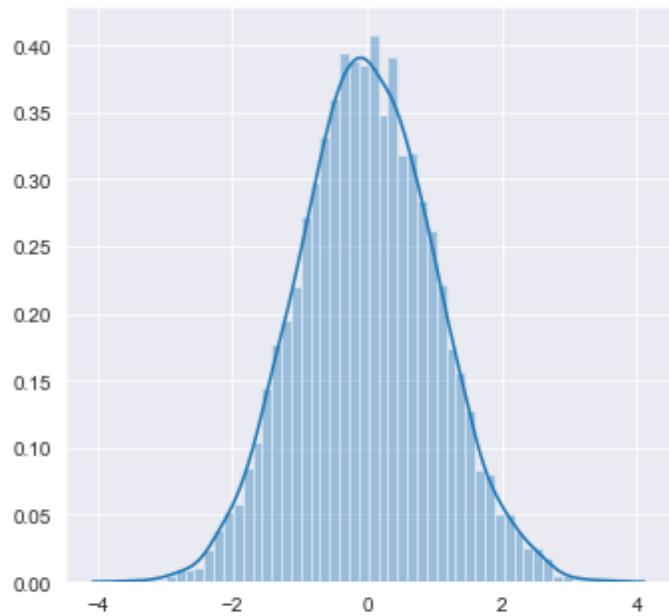
Python



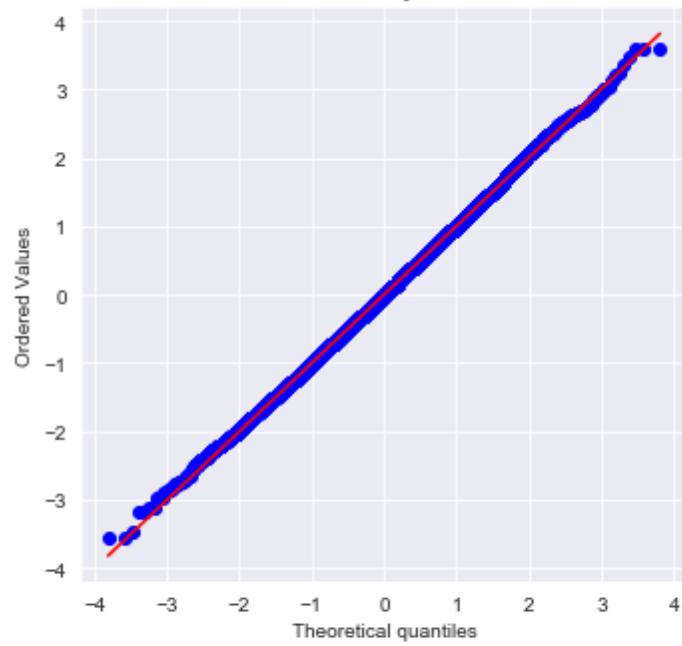
We see that the middle looks ok, but the ends, especially the higher end, are diverging from a normal distribution. This aligns with the skew that we observed from the histogram

**More examples of histograms and their associated Q-Q plots**

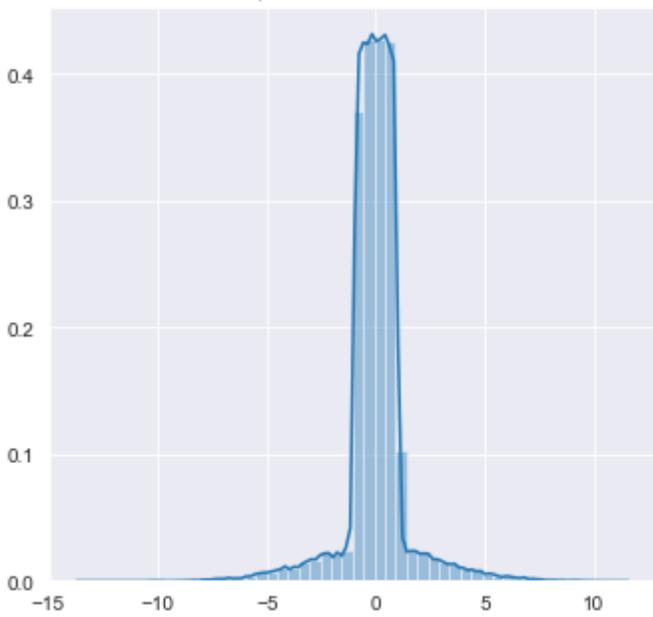
The Standard Normal Distribution



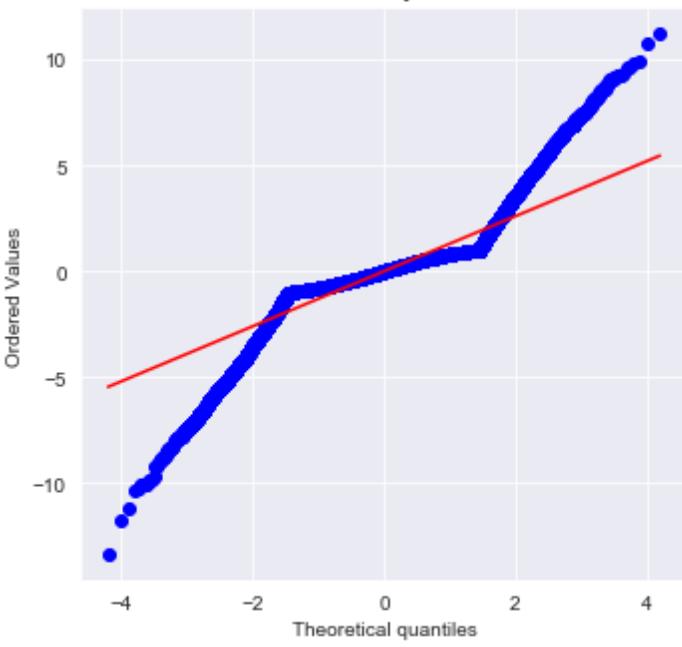
Probability Plot

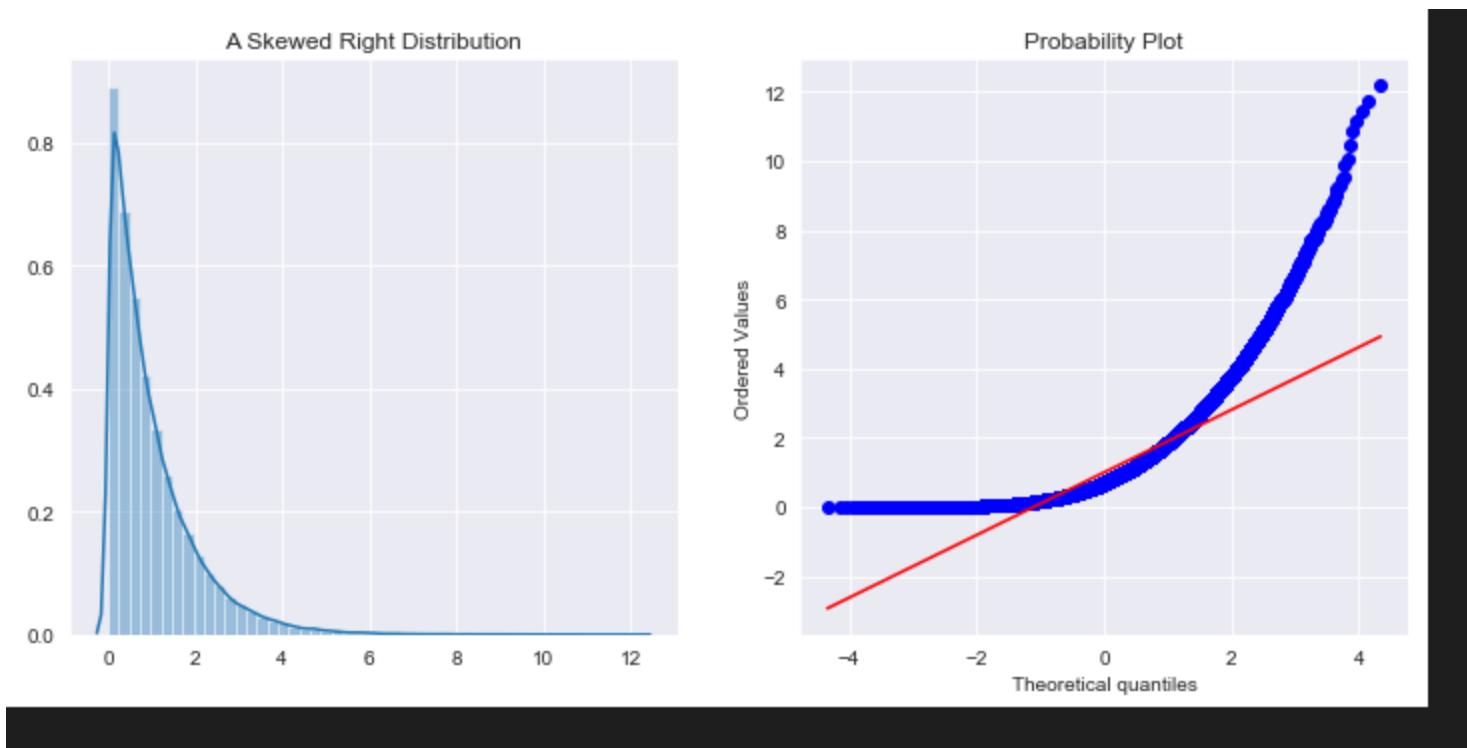


Nonnormal; Data too Peaked in Middle



Probability Plot





## Statistical Testing for Normality

### i) Jarque-Bera test

**NULL:** the data follows a normal distribution

**Alternative:** The data does not follow a normal distribution

```
from statsmodels.stats.stattools import jarque_bera
jarque_bera(generated_data)
jarque_bera(results.resid)
```

```
from statsmodels.stats.stattools import jarque_bera
```

Python

+ Code + Markdown

```
jarque_bera(generated_data)
```

Python

```
(0.4519188967633713,  
 0.7977504510815616,  
 -0.0332236549490571,  
 2.849335177253925)
```

☰ ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋

```
jarque_bera(results.resid)
```

Python

```
(51.332550412287, 7.133099908986785e-12, 0.7146664114306294, 4.048721550463704)
```

The four values returned are:

1. JB test statistic
2. The p-value for JB
3. Skew
4. Kurtosis

You can also view these values directly within a model summary. For example, these are the results for our auto MPG model. If you look in the lower right area you will see Jarque-Bera (JB) and Prob(JB):

```
print(results.summary())
```

Python

```
print(results.summary())
```

Python

### OLS Regression Results

```
=====
Dep. Variable:          mpg   R-squared:      0.707
Model:                 OLS   Adj. R-squared:  0.704
Method:                Least Squares   F-statistic:    233.4
Date: Fri, 18 Oct 2024   Prob (F-statistic):  9.63e-102
Time: 12:50:02           Log-Likelihood:   -1120.6
No. Observations:      392   AIC:            2251.
Df Residuals:          387   BIC:            2271.
Df Model:               4
Covariance Type:       nonrobust
=====
              coef    std err        t      P>|t|      [0.025]     [0.975]
-----
const      45.2511    2.456    18.424      0.000     40.422     50.080
displacement -0.0060    0.007   -0.894      0.372     -0.019      0.007
horsepower  -0.0436    0.017   -2.631      0.009     -0.076     -0.011
weight      -0.0053    0.001   -6.512      0.000     -0.007     -0.004
acceleration -0.0231    0.126   -0.184      0.854     -0.270      0.224
=====
Omnibus:             38.359   Durbin-Watson:    0.861
Prob(Omnibus):       0.000   Jarque-Bera (JB):  51.333
Skew:                  0.715   Prob(JB):       7.13e-12
Kurtosis:             4.049   Cond. No.       3.56e+04
=====
...
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.56e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

### ### Interpreting Jarque-Bera Test Results

In the case of the Jarque-Bera test the **\*\*null hypothesis is that the distribution is normal\*\***. A sufficiently low p-value means we reject the null hypothesis, i.e. that the distribution is not normal.

For the two datasets presented above:

\* The generated data's **\*\*p-value is about 0.8\*\***. This is much higher than the standard alpha of 0.05, so we fail to reject the null hypothesis and can **\*\*consider the distribution to be normal\*\***

\* The model residuals' **\*\*p-value is about 0.00000000007\*\***. This is much lower than the standard alpha of 0.05, so we reject the null hypothesis and **\*\*do not consider the distribution to be normal\*\***

Like with some other model assumption checks, this is "opposite" of the p-values you want for model significance or coefficient significance. A low p-value in the Jarque-Bera test is a "bad" outcome, indicating a problem with the model.

## Other statistical tests

As mentioned previously, there are several different options to test for normality, each of which uses a slightly different approach. Below are some additional tests you might want to consider:

### #### 1. Kolmogorov-Smirnov Test

```
stats.kstest(generated_data, "norm")  
stats.kstest(results.resid, "norm")
```

#### Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test compares the CDF (cumulative distribution function) of the two distributions. The null hypothesis is that the distributions are the same. There is an implementation in SciPy ([documentation here](#)):

```
stats.kstest(generated_data, "norm")
```



Python

```
KstestResult(statistic=0.03437796417014605, pvalue=0.7184091664903246)
```

```
stats.kstest(results.resid, "norm")
```

Python

```
KstestResult(statistic=0.32698123251877126, pvalue=8.001138817172193e-38)
```

### 2. Lilliefors Test

The [Lilliefors test]([https://en.wikipedia.org/wiki/Lilliefors\\_test](https://en.wikipedia.org/wiki/Lilliefors_test)) is based on the Kolmogorov-Smirnov test and uses estimated parameters. The null hypothesis is that the specified data comes from a normal distribution. There is an implementation in StatsModels ([documentation here]([https://www.statsmodels.org/devel/generated/statsmodels.stats.diagnostic.kstest\\_normal.html](https://www.statsmodels.org/devel/generated/statsmodels.stats.diagnostic.kstest_normal.html))), which returns the test statistic and the p-value:

```
from statsmodels.stats.diagnostic import kstest_normal  
kstest_normal(generated_data)  
kstest_normal(results.resid)
```

## Lilliefors Test

The [Lilliefors test](#) is based on the Kolmogorov-Smirnov test and uses estimated parameters. The null hypothesis is that the specified data comes from a normal distribution. There is an implementation in StatsModels ([documentation here](#)), which returns the test statistic and the p-value:

```
from statsmodels.stats.diagnostic import kstest_normal
```

Python

```
| kstest_normal(generated_data)
```



Python

```
| (0.02646141843017441, 0.7280229807121467)
```

```
1 kstest_normal(results.resid)
```

Python

```
| (0.06804544956822389, 0.000999999999998899)
```

### 3. ##### Anderson-Darling Test

The [Anderson-Darling test]([https://en.wikipedia.org/wiki/Anderson%20%20%20Darling\\_test](https://en.wikipedia.org/wiki/Anderson%20%20%20Darling_test)) also compares the CDF of the two distributions. The null hypothesis is that the specified data comes from a normal distribution. There is an implementation in StatsModels ([documentation here]([https://www.statsmodels.org/devel/generated/statsmodels.stats.diagnostic.normal\\_ad.html](https://www.statsmodels.org/devel/generated/statsmodels.stats.diagnostic.normal_ad.html))), which returns the test statistic and p-value:

```
from statsmodels.stats.diagnostic import normal_ad
```

```
normal_ad(generated_data)
```

```
normal_ad(results.resid)
```

```
|from statsmodels.stats.diagnostic import normal_ad
```

```
normal_ad(generated_data)
```

```
(0.306208049572092, 0.5639537893049426)
```

```
normal_ad(results.resid)
```

```
(2.5714252349764024, 1.688944543036666e-06)
```

#### 4.Omnibus Test

for normality tests for deviations that result from skewness or kurtosis. The null hypothesis is that the specified data comes from a normal distribution. There is an implementation in StatsModels ([documentation here]([https://www.statsmodels.org/devel/generated/statsmodels.stats.stattools.omni\\_normtest.html](https://www.statsmodels.org/devel/generated/statsmodels.stats.stattools.omni_normtest.html))):

```
from statsmodels.stats.stattools import omni_normtest  
omni_normtest(generated_data)  
omni_normtest(results.resid)
```

```
from statsmodels.stats.stattools import omni_normtest
```

Python

```
omni_normtest(generated_data)
```

Python

```
NormaltestResult(statistic=0.3290035852108211, pvalue=0.8483162362467015)
```



```
omni_normtest(results.resid)
```

Python

```
NormaltestResult(statistic=38.35880421385135, pvalue=4.68264785079494e-09)
```

This result is also printed out in the model summary, in the lower left. Look for `Omnibus` and `Prob(Omnibus)`:

```
print(results.summary())
```

Python

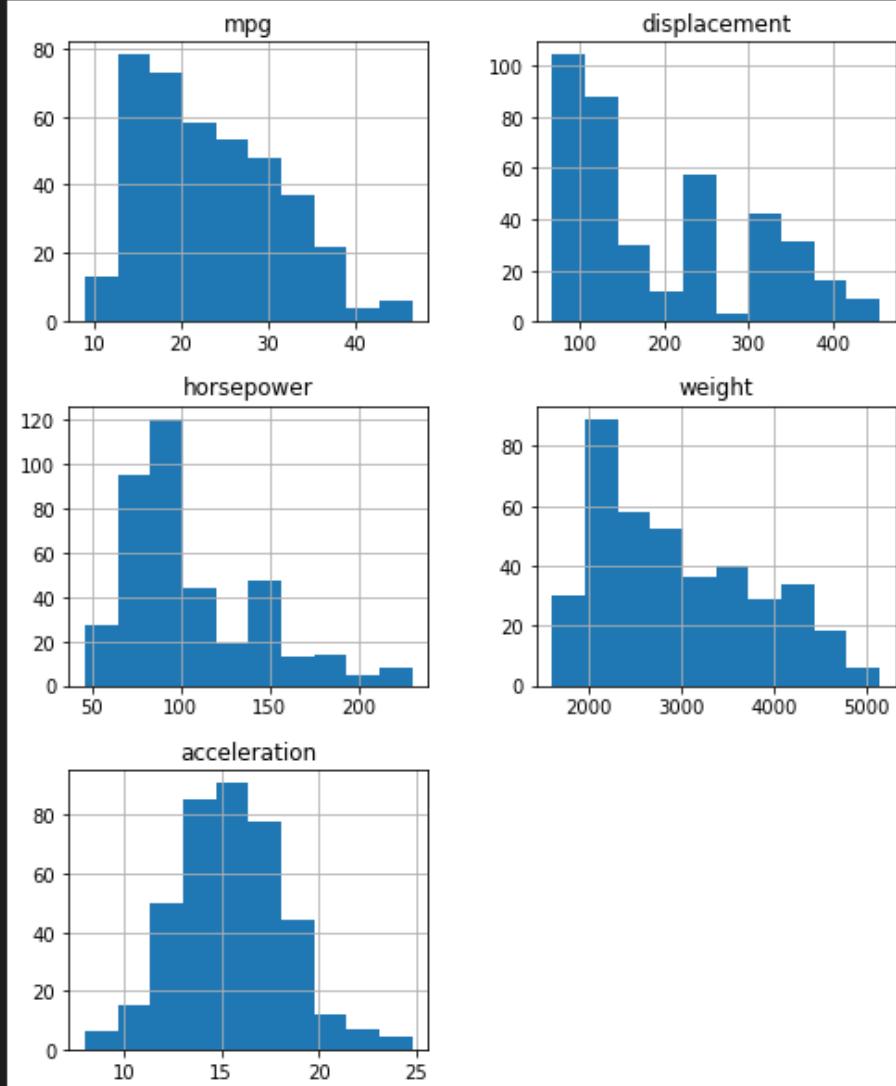
### OLS Regression Results

```
=====
Dep. Variable:          mpg    R-squared:       0.707
Model:                 OLS    Adj. R-squared:   0.704
Method:                Least Squares F-statistic:     233.4
Date:      Fri, 17 Jun 2022   Prob (F-statistic):  9.63e-102
Time:          15:28:49    Log-Likelihood:   -1120.6
No. Observations:      392    AIC:             2251.
Df Residuals:          387    BIC:             2271.
Df Model:                  4
Covariance Type:    nonrobust
=====
            coef    std err          t      P>|t|      [0.025    0.975]
-----
const      45.2511    2.456     18.424      0.000     40.422    50.080
displacement  -0.0060    0.007     -0.894      0.372     -0.019     0.007
horsepower    -0.0436    0.017     -2.631      0.009     -0.076     -0.011
weight      -0.0053    0.001     -6.512      0.000     -0.007     -0.004
acceleration  -0.0231    0.126     -0.184      0.854     -0.270     0.224
=====
Omnibus:           38.359    Durbin-Watson:     0.861
Prob(Omnibus):    0.000    Jarque-Bera (JB):  51.333
Skew:              0.715    Prob(JB):        7.13e-12
Kurtosis:          4.040    Cond. No.:      2.56e+04
=====
```

## Treating Normality Issues

Log transformations tend to be the most helpful when it comes to normality issues. Non-normality in the features or target is often associated with non-normality in the residuals. Below we plot the distributions of all data in the auto MPG model:

```
data[['mpg', 'displacement', 'horsepower', 'weight',
      'acceleration']].hist(figsize=(8,10));
```

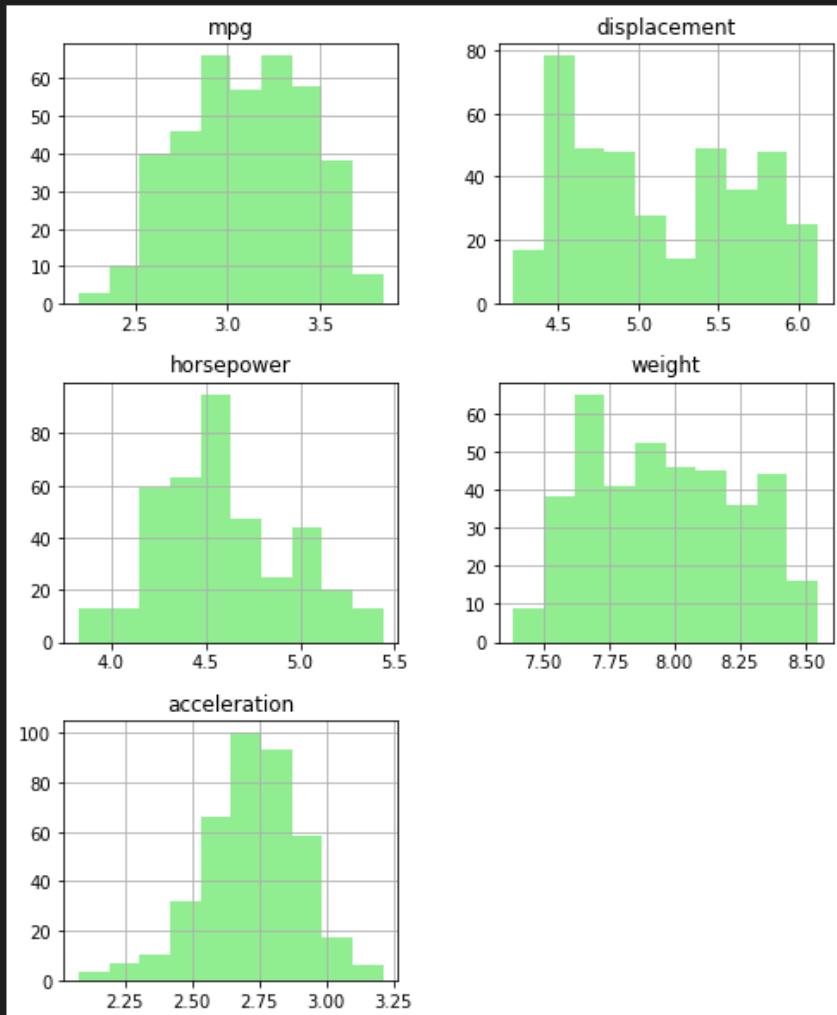


Same distributions after being log transformed

```
np.log(data[['mpg', 'displacement', 'horsepower', 'weight',
            'acceleration']]).hist(figsize=(8,10),color='lightgreen');
```

```
np.log(data[['mpg', 'displacement', 'horsepower', 'weight',
|   'acceleration']]).hist(figsize=(8,10),color='lightgreen');
```

Python



As you can see, for most variables, they still aren't fully normal, but they're closer. However `acceleration` looks more skewed now than before. So let's log transform everything except `acceleration`.

Then if we build a model using those log-transformed features and target, we see some improvement in the distribution of the residuals:

```
# Build log transformed model
y_log = np.log(data["mpg"])
X_log = pd.concat([np.log(data[['displacement', 'horsepower', 'weight']]),
data[['acceleration']]], axis=1)
log_model = sm.OLS(y_log, sm.add_constant(X_log))
log_results = log_model.fit()
```

```
# Set up plot and properties of two models
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12,10))
resids = [results.resid, log_results.resid]
labels = ["Original Model", "Log Transformed Model"]
colors = ["blue", "lightgreen"]

# Plot histograms
for index, ax in enumerate(axes[0]):
    sns.histplot(resids[index], bins=20, element="step", kde=True,
color=colors[index], ax=ax)
    ax.set_xlabel("Model Residuals")
    ax.set_title(labels[index])

# Plot Q-Q plots
for index, ax in enumerate(axes[1]):
    sm.graphics.qqplot(resids[index], dist=stats.norm, line='45', fit=True, ax=ax)
    scatter = ax.lines[0]
    line = ax.lines[1]
    scatter.set_markeredgecolor(colors[index])
    scatter.set_markerfacecolor(colors[index])
    line.set_color("black")
    ax.set_title(labels[index])

fig.tight_layout()
```

```

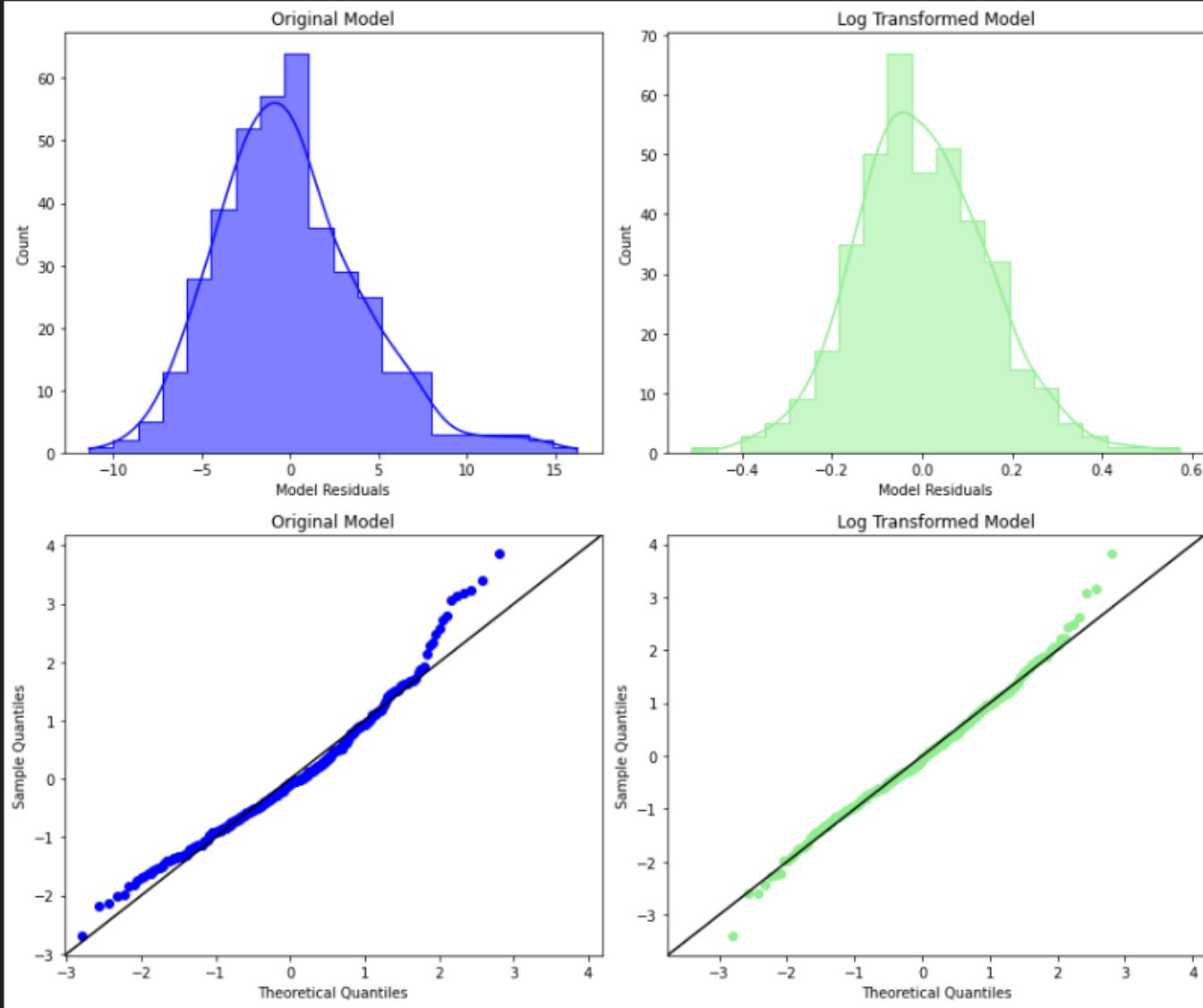
        scatter.set_markerfacecolor(colors[index])
        line.set_color("black")
        ax.set_title(labels[index])

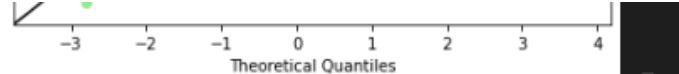
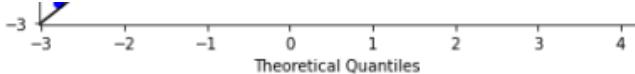
fig.tight_layout()

```

Python

c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: Ir  
x = pd.concat(x[::-order], 1)





Looking at the p-values from the Jarque-Bera test, you can see we're getting a lot closer:

```
27] print(F"Original model: {jarque_bera(results.resid)[1]}")
print(F"Log transformed model: {jarque_bera(log_results.resid)[1]}")
```

Python

```
.. Original model: 7.133099908986785e-12
Log transformed model: 0.0036669351258854627
```

And using some other measures, we actually can't reject the null hypothesis!

```
28] print(F"Original model: {kstest_normal(results.resid)[1]}")
print(F"Log transformed model: {kstest_normal(log_results.resid)[1]}")
```

Python

```
.. Original model: 0.000999999999998899
Log transformed model: 0.1610763758916161
```

```
29] print(F"Original model: {normal_ad(results.resid)[1]}")
print(F"Log transformed model: {normal_ad(log_results.resid)[1]}")
```

Python

```
.. Original model: 1.688944543036666e-06
Log transformed model: 0.08748354038810166
```

Note that because the different tests assess normality in different ways, you might make different decisions depending on the statistic you are using.

Also keep in mind that log transformations are fundamentally changing the nature of the linear regression model to be multiplicative with respect to the original units rather than additive. So there may be a trade-off between model interpretability and meeting the normality assumption.

## 86. QQ Plots

```
import statsmodels.api as sm
from seaborn import load_dataset
import numpy as np
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
```

[3]

```
diamonds = load_dataset('diamonds')
diamonds.head()
```

[4]

...

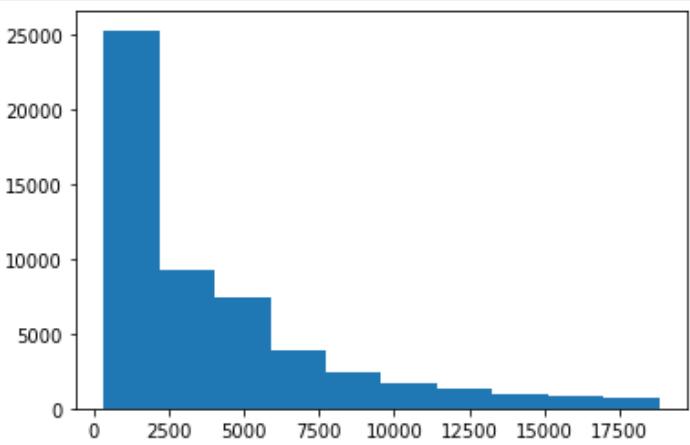
|   | carat | cut     | color | clarity | depth | table | price | x    | y    | z    |
|---|-------|---------|-------|---------|-------|-------|-------|------|------|------|
| 0 | 0.23  | Ideal   | E     | SI2     | 61.5  | 55.0  | 326   | 3.95 | 3.98 | 2.43 |
| 1 | 0.21  | Premium | E     | SI1     | 59.8  | 61.0  | 326   | 3.89 | 3.84 | 2.31 |
| 2 | 0.23  | Good    | E     | VS1     | 56.9  | 65.0  | 327   | 4.05 | 4.07 | 2.31 |
| 3 | 0.29  | Premium | I     | VS2     | 62.4  | 58.0  | 334   | 4.20 | 4.23 | 2.63 |
| 4 | 0.31  | Good    | J     | SI2     | 63.3  | 58.0  | 335   | 4.34 | 4.35 | 2.75 |

## QQPlots and Log Transformation

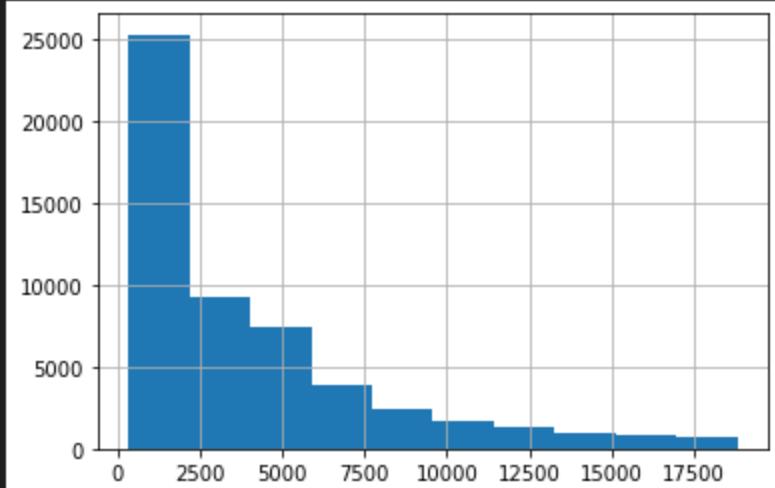
```
#check histogram of the target
plt.hist(diamonds['price']);
```

[6]

...



```
diamonds['price'].hist();
```

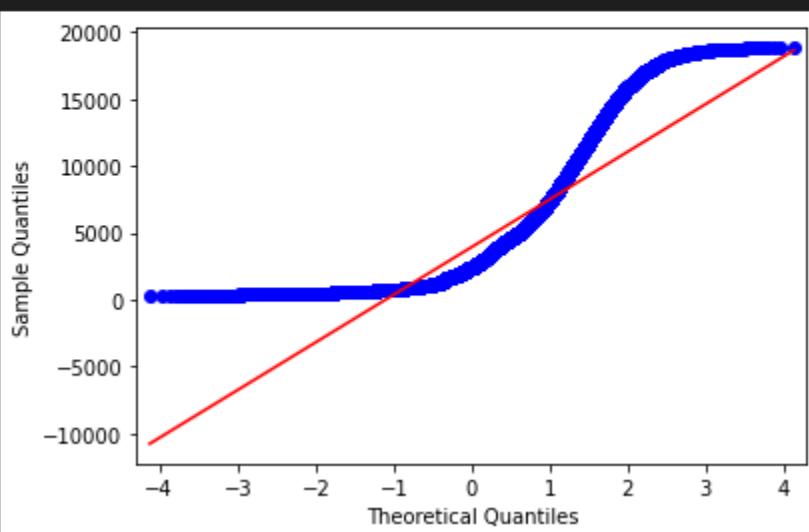


### Check qqplot against normal

```
sm.qqplot(diamonds['price'],line='r');
```

```
# check qqplot against normal
sm.qqplot(diamonds['price'],line='r');
```

✓ 0.2s

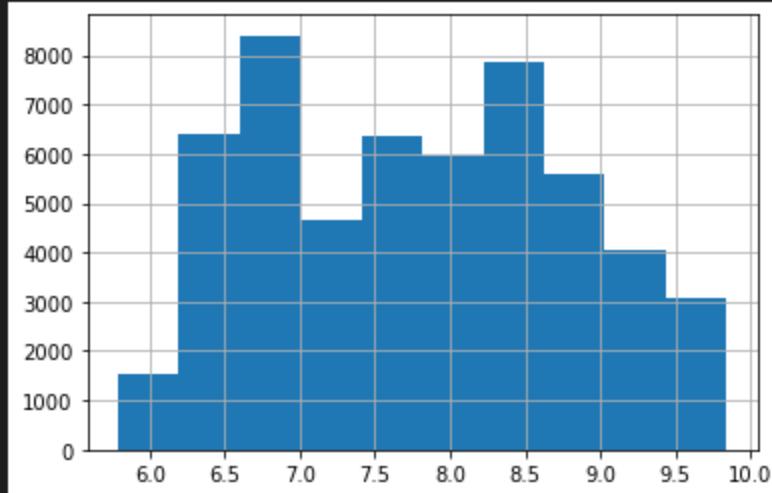


### Histogram of log-transformed target

```
np.log(diamonds['price']).hist();
```

```
#Histogram of log-transformed target  
np.log(diamonds['price']).hist();
```

✓ 0.1s

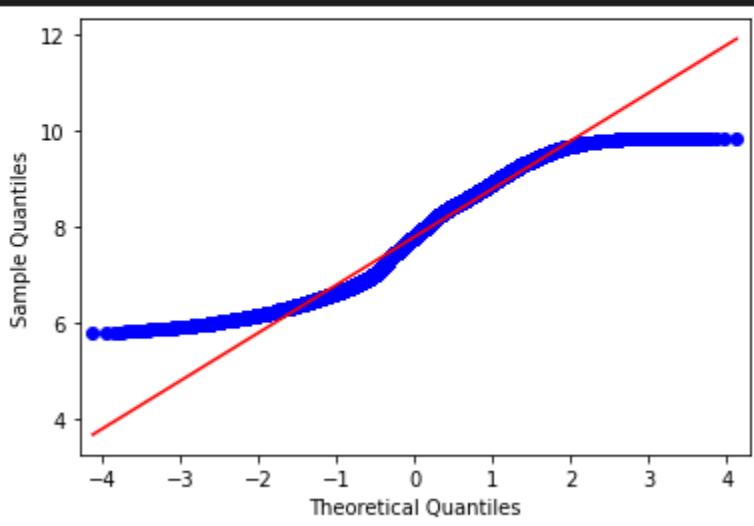


## QQ plot of log-transformed target

```
sm.qqplot(np.log(diamonds['price']),line='r'); #not perfect but better
```

```
##QQ plot of log-transformed target  
sm.qqplot(np.log(diamonds['price']),line='r'); #not perfect but better
```

✓ 0.2s



## QQ plot For model residuals

```
#set up X, y and lr
X = diamonds[['carat']]
y = diamonds['price']
lr = LinearRegression().fit(X,y)

#store model predictins
preds = lr.predict(X)
#check the histogram of the residuals
resids = y -preds
#plt.hist(resids)
fig, ax = plt.subplots()
ax.hist(resids,bins=30);

sm.qqplot(resids,line='r');
```

```
#set up X, y and lr
X = diamonds[['carat']]
y = diamonds['price']
lr = LinearRegression().fit(X,y)
```

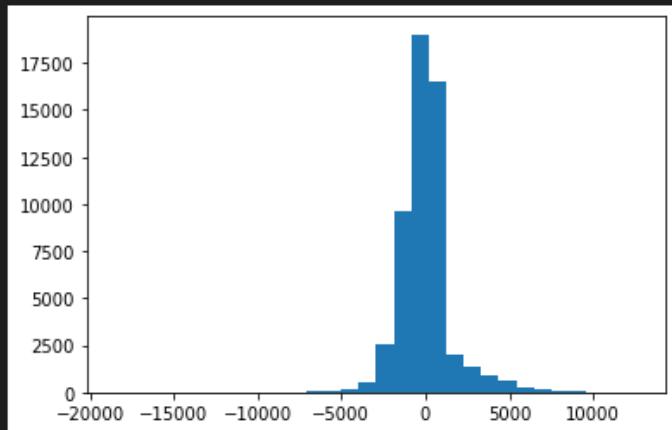
✓ 0.2s

Python

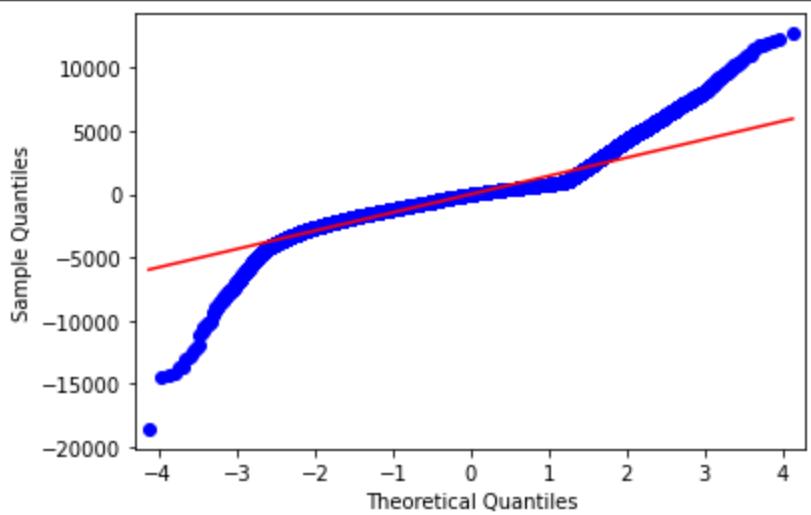
```
#store model predictions
preds = lr.predict(X)
#check the histogram of the residuals
resids = y -preds
#plt.hist(resids)
fig, ax = plt.subplots()
ax.hist(resids,bins=30);
```

✓ 0.3s

Python



```
#check the QQ Plot  
sm.qqplot(resids,line='r');  
✓ 0.2s
```



## BUILD MODEL ON TRANSFORMED TARGET

```
log_target = np.log(y)  
log_lr = LinearRegression().fit(X,log_target)  
#store predictions  
log_preds = log_lr.predict(X)  
#new QQ plot  
sm.qqplot(log_target - log_preds, line='r');
```

```
#Build model on transformed target
log_target = np.log(y)
log_lr = LinearRegression().fit(X, log_target)
✓ 0.0s

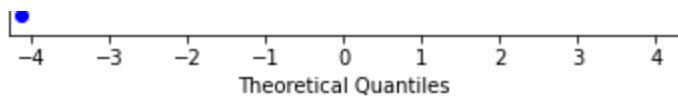
#store predictions
log_preds = log_lr.predict(X)
✓ 0.0s

#new QQ plot
sm.qqplot(log_target - log_preds, line='r');
✓ 0.2s
```

A Quantile-Quantile (QQ) plot comparing sample quantiles against theoretical quantiles. The x-axis is labeled "Theoretical Quantiles" and ranges from -4 to 4. The y-axis is labeled "Sample Quantiles" and ranges from -6 to 2. Blue dots represent the sample quantiles, which deviate significantly from the red diagonal line at lower values (around -4 to -1) and then follow it closely for higher values (above 0).

## Check shape of our predictor

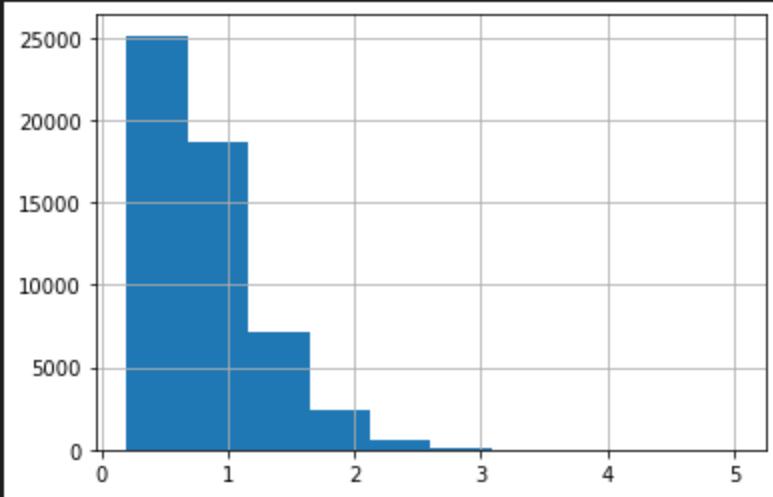
```
diamonds['carat'].hist()
```



```
#what about the shape of our predictor  
diamonds['carat'].hist()
```

[14] ✓ 0.1s

... <AxesSubplot:>



```
#log transform the predictor
```

## Log transform our predictor

```
#log transform the predictor
```

```
log_carat = np.log(X)
```

```
two_logs_lr = LinearRegression().fit(log_carat,log_target)
```

```
##QQ plot of double log-transformed data
```

```
two_logs_predict = two_logs_lr.predict(log_carat)
```

```
sm.qqplot(log_target-two_logs_predict,line='r');
```

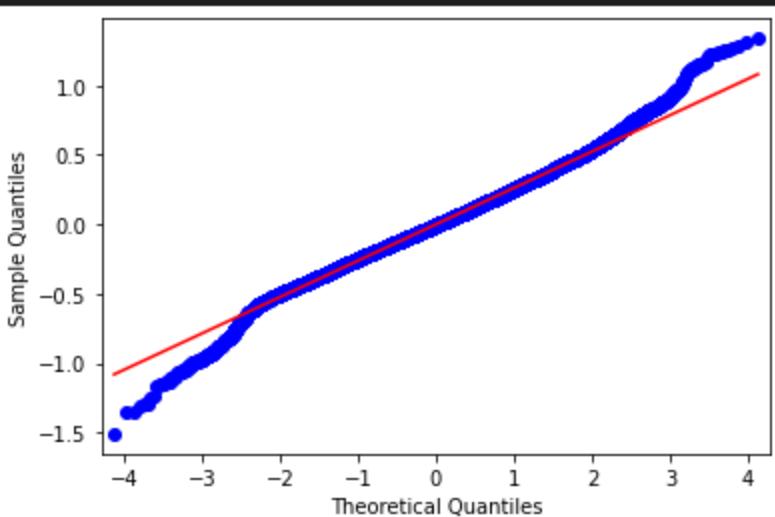
```
#sm.qqplot(np.log(y)-two_logs_predict,line='r');
```

```
#log transform the predictor  
log_carat = np.log(X)  
two_logs_lr = LinearRegression().fit(log_carat,log_target)
```

✓ 0.0s

```
##QQ plot of double log-transformed data  
two_logs_predict = two_logs_lr.predict(log_carat)  
sm.qqplot(log_target-two_logs_predict,line='r')  
#sm.qqplot(np.log(y)-two_logs_predict,line='r');
```

✓ 0.2s



- ✓ [HOMOSCEDASTICITY ASSUMPTION: EQUAL VARIANCE](#)([https://github.com/GraceMwende/PHASE\\_2-MS\\_DS/blob/main/Flatiron/week5/The%20Homoscedasticity%20Assumption.ipynb](https://github.com/GraceMwende/PHASE_2-MS_DS/blob/main/Flatiron/week5/The%20Homoscedasticity%20Assumption.ipynb))

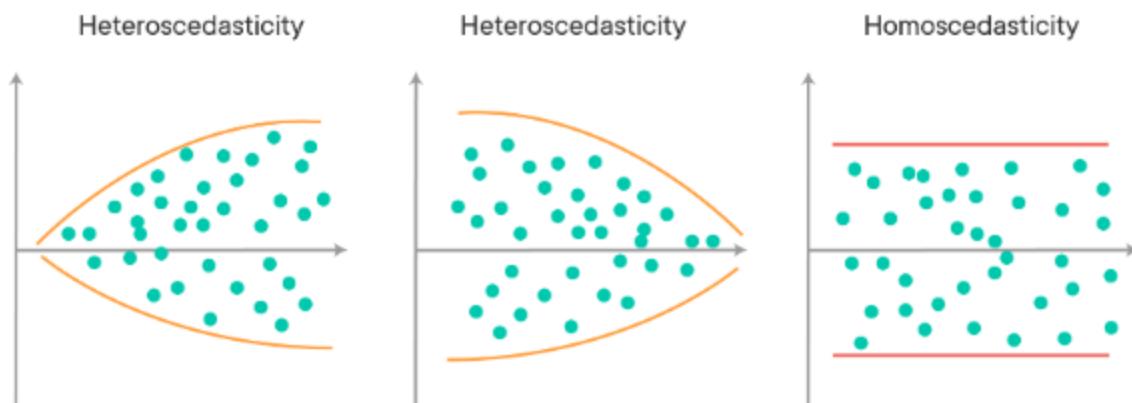
States that we want to avoid **heteroscedasticity** of the error.heteroscedasticity refers to the circumstance in which a variable's variability is unequal across the range of values of the predictor(s)

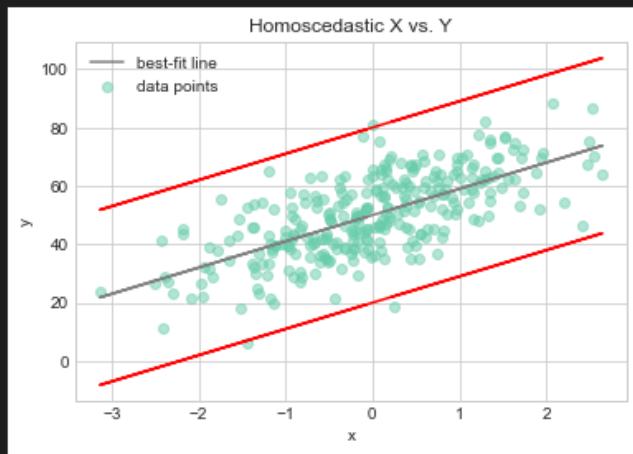
The equal variance (homoscedasticity) assumption states that we want to avoid heteroscedasticity of the errors

Heteroscedasticity (also spelled heteroskedasticity) refers to the circumstance in which a variable's variability is unequal across the range of values of the predictor(s).

When there is heteroscedasticity in the data, a scatter plot of these variables will often create a cone-like shape. The scatter of the errors widens or narrows as the value of the independent variable increases.

The inverse of heteroscedasticity is *homoscedasticity*, which indicates that a variable's variability is equal across values of the independent variable.





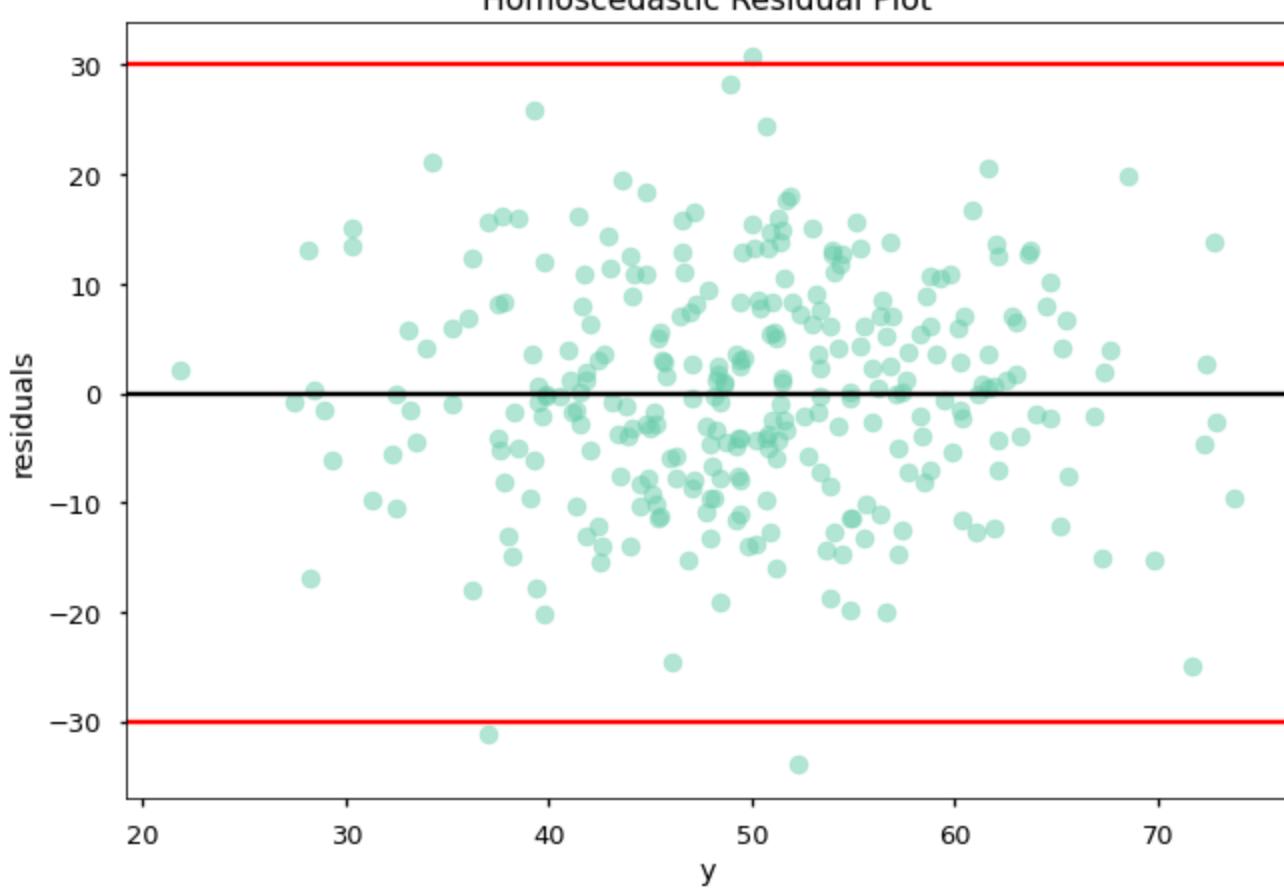
You can tell that this is *homoscedastic* (has equal variance) because the red lines (showing the distribution of the predictions vs. the real data points) are parallel to the best-fit line.

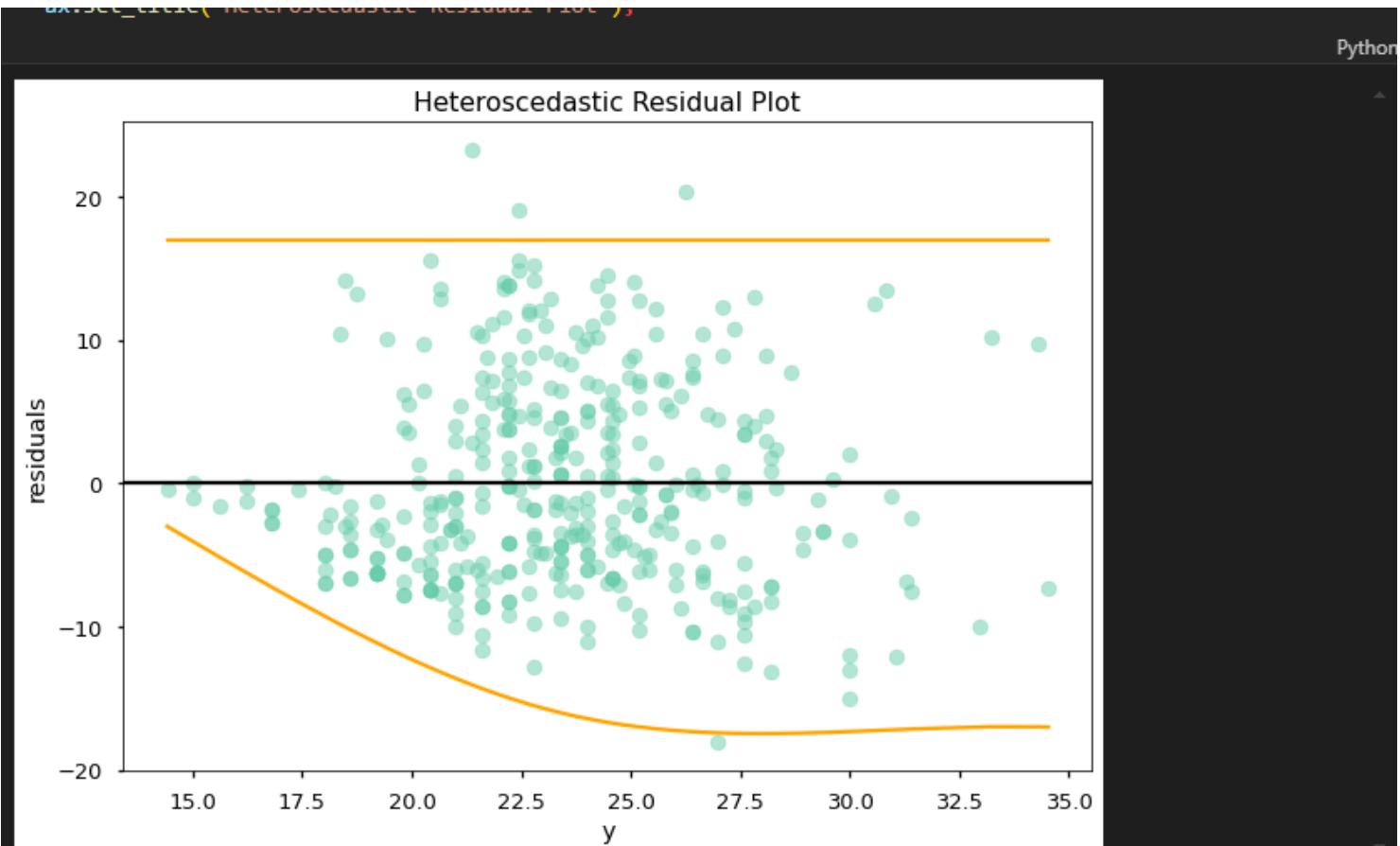


You can tell that this is *heteroscedastic* because of the cone shape on the bottom. The top looks fairly linear, but the fact that the top and bottom don't match is an indication that something is wrong with the model specification (e.g. a non-linear relationship).

## Residual Plot(Simple or Multiple Linear Regression)

Homoscedastic Residual Plot





Here the "cone" shape on the bottom is easier to see. Because of this shape (meaning that the variance in the residuals is increasing as  $y$  increases, rather than being consistent) we can tell that this is *heteroscedastic*.

### Statistical tests for Homoscedasticity

**Null :** The variance of the residuals is constant(i.e the error terms are homoscedastic)

**Alternative:** The variance of the residuals is not constant(i.e the error terms are heteroscedastic)

#### 1.Goldfeld-Quandt Test

divides the dataset into two groups, then finds the MSE of the residuals for each group. The ratio of the second group's `mse\_resid` divided by the first group's `mse\_resid` becomes a statistic that can be compared to the f-distribution to find a p-value.

There is a StatsModels implementation ([documentation here]([https://www.statsmodels.org/stable/generated/statsmodels.stats.diagnostic.het\\_goldfeldquandt.html](https://www.statsmodels.org/stable/generated/statsmodels.stats.diagnostic.het_goldfeldquandt.html))) that returns:

1. Goldfeld-Quandt test statistic
2. Goldfeld-Quandt test p-value
3. Ordering

(Ordering relates to the direction of the heteroscedasticity you want to be able to detect. By default it is "increasing" but you can also specify "decreasing" or "two-sided".)

This implementation actually creates models and computes residuals for each subset of the dataset, so you pass in the target and features, not the residuals.

```
from statsmodels.stats.diagnostic import het_goldfeldquandt
```

*If the p-value from the test is small (typically less than 0.05) reject the null hypothesis*

```
from statsmodels.stats.diagnostic import het_goldfeldquandt
# out puts
# Goldfeld-Quandt test statistic
# Goldfeld-Quandt test p-value
# Ordering

het_goldfeldquandt(df_ads["sales"], sm.add_constant(df_ads["radio"])) #fail to reject pvalue of 0.25
```

Python

```
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: Ir
x = pd.concat(x[::-order], 1)
```

```
(1.1409213847001907, 0.2576335266276604, 'increasing')
```

```
het_goldfeldquandt(df_ads["sales"], sm.add_constant(df_ads["TV"]))
```

Python

```
c:\Users\Gmwende\anaconda3\envs\learn-env\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: Ir
x = pd.concat(x[::-order], 1)
```

```
(1.2071212974713172, 0.17652851936962768, 'increasing')
```

2. Breusch-Pagan Test

## Breusch-Pagan Test

Run and Debug (Ctrl+Shift+D) Statistical test for homoscedasticity is the [Breusch-Pagan test](#). It is a type of  $\chi^2$  test based on the [Lagrange multiplier test](#) and the underlying concept is that you are trying to see whether you can linearly predict the residuals using the provided features.

There is a StatsModels implementation ([documentation here](#)) that returns:

1. Lagrange multiplier statistic
2. Lagrange multiplier p-value
3. Breusch-Pagan test statistic
4. Breusch-Pagan test p-value

This implementation expects you to pass in the residuals as the first argument and the features (including a constant) as the second argument.

```
[8] from statsmodels.stats.diagnostic import het_breushpagan
```

Python

```
[9]     het_breushpagan(sm.OLS(y_generated, sm.add_constant(x_generated)).fit().resid,
```

Python

```
... (0.1956987661898224,
 0.6582153107631223,
 0.1945209994805444,
 0.6594999728629689)
```

```
[10]     het_breushpagan(results.resid, sm.add_constant(x_mpg))
```

Python

```
... (6.023194926772518,
 0.014119064285488344,
 6.085977163823662,
 0.014054760063763048)
```

One again the null hypothesis is homoscedasticity, which we fail to reject for the generated data and reject for the auto MPG data.