1. To get more info about a function in python press shift+tab on the code

2. on str.split on datadrame add n=

df["Address"].str.split(",", n=1,expand=True)

3. on str.replace add regex=True

df['Phone_Number']=df['Phone_Number'].str.replace('[^a-zA-Z0-9]',",regex=True)

4. doing correlation add numeric=True

df.corr(numeric_only=True)

5.when doing aggregate add numeric=true as well

group_by_frame.mean(numeric_only=True)conda activate learn-env

df.groupby('Base Flavor').sum(numeric_only=True)

6.annot=True not working had to downgrade to matlib 3.7.3

pip install matplotlib==3.7.3 --user

7. intellisense press tab

8.Control +backslach to comment

9.Add column by looping though a dataset

#add flower name

df['flower_name']= df.target.apply(lambda x: iris.target_names[x])

df.head()

10. covert excel to a list of dictionaries

import pandas as pd

file_name = './cities.xlsx'

travel_df = pd.read_excel(file_name)

cities = travel_df.to_dict('records')

11. read json file

import json

```
with open("coffee_product_reviews.json") as f:
    reviews = json.load(f)
print(type(reviews))
```

12. Display images using ipython - will display image on that link

```
from IPython.display import Image
Image('https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png')
```

13. code. (opens in vs)

14.

Type pwd - this should show your home directory, the most basic of paths on your computer

Type cd Documents - this will change your directory, and move you into your Documents folder

Type mkdir Flatiron - this will create a new folder, called Flatiron, to keep all of your Flatiron repositories and files

Type cd Flatiron - this will change your directory, moving you into the new Flatiron folder you just created

15.a)creating Conda Virtual environment::on git run:

```
conda env create -f win_environment.yml
```

b)Activating the Conda Virtual Environment: To initiliaze a permanet shell which adds shell code to the startup scripts run:

```
conda init bash
```

Activate the environment run:

```
conda activate learn-env
```

c)To confirm that it worked, type the below: and confirm that the asterisk (*) is next to the learn-env environment.

```
 conda info --envs
```

16 commands for terminal

pwd

cd- takes you to home folder

cd .. one folder up

ls - list all files

ls -a (list all files including hidden files)

ls -l gives a long listing of files (including file size and last edit times)

You can also pass multiple parameters simultaneously, such as ls -al to produce a detailed listing of all files.

you wanted to list all files in the current working directory that begin with a, you could type ls a*

list all pdf files in the current working directory you can use ls *.pdf, or to list all text files, you can use ls *.txt

mkdir command, which stands for make directory. Try it out with mkdir NewFolderName. Afterward, use the ls command to see that there is indeed a new folder, and if you wish, move into the new folder using the cd commandr.

17. cmd + L to highlight the url bar

18. Always activate the virtual environment

conda activate learn-env

19.Add new folder to github

git remote add origin the_url_for_the_repo

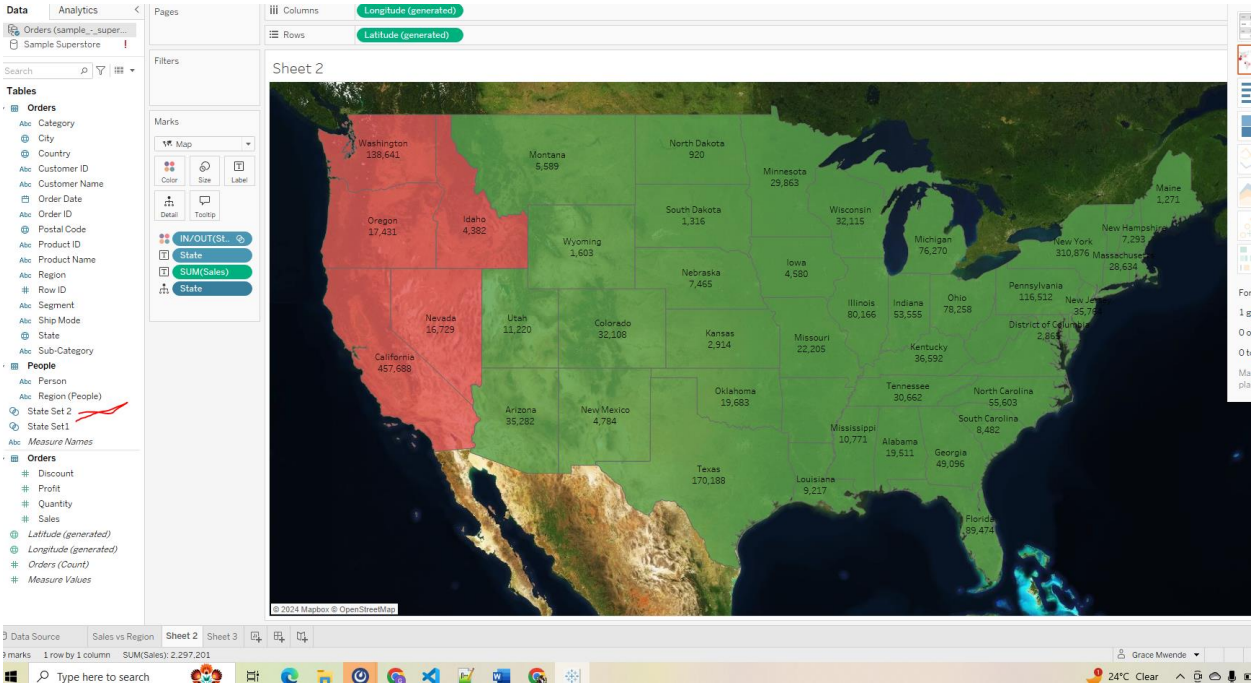
20.xtiks rotation

plt.xticks(rotation=90)

21.hist and kde plot together

sns.histplot(data['total_bill'],kde=True)

22.create set-shade different color on stuff



separate a region from the rest could use colors

23 . meetups

omdena

afterwork

eudureka

24..Change from exponential to numbers

pd.set_option('display.float_format','{:.2f}'.format)

25. **check for unique values in each column**

for column in df1:

unique_values = df1[column].unique()

print(f"Unique values in column '{column}','\n': {unique_values}","\n')

26. insert to a certain location

fifa.insert(loc=fifa.columns.get_loc('Contract')+1+i,column=new_cols[i], value=new_data[i])

27. convert all at once to float and strip

```
 df1[['missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',
      'dumb', 'mental', 'paralyzed', 'other']] = df1[['missing_hand', 'missing_foot',
'lame', 'blind', 'deaf',
      'dumb', 'mental', 'paralyzed', 'other']].apply(lambda x:
x.str.strip('%').astype(float)/100)
```

28. check outliers for all columns

removing outliers if all numeric

```
### if all numeric use
# dropping ouliers using interquantile method
Q1 = df1.quantile(0.25)
Q3 = df1.quantile(0.75)
IQR = Q3 - Q1
# removing outliers
df2 = df1[~((df1 < (Q1 - 1.5 * IQR)) | (df1 > (Q3 + 1.5 * IQR))).any(axis=1)]



# checking old shape
print("old shape:","\n",df1.shape)
print("****"*10)
print("new shape:","\n", df2.shape)
```

select numeric columns

```
numeric_columns = df1.select_dtypes(include=['int','float']).columns
Q1= df1[numeric_columns].quantile(0.25)
print(f'Q1: "\n" {Q1}')
```

Q3 = df1[numeric_columns].quantile(0.75)

print(f'Q3: "\n"{Q3}')

IQR = Q3- Q1

IQR

df2 = df1[~((df1[numeric_columns] < (Q1 - 1.5 * IQR)) | (df1[numeric_columns] > (Q3 + 1.5 * IQR))).any(axis=1)]

#checking shape

print('old shape',"\n",df1.shape)

print('new shape',"\n",df2.shape)

29. 27. Headers shows where the import shld start from(below imports starts from row 47

df = pd.read_excel('Zipcode_Demos.xlsx',header=48)

- can aslo be achieved by skipping rows

df2 = pd.read_excel('Zipcode_Demos.xlsx',skiprows=48)

df

28.We can also define the no of rows we want from the dataset and columns to use

df1 = pd.read_excel('Zipcode_Demos.xlsx',nrows=46,header=1,usecols=[0,1]) # import from index 1 and takes only column 0 and 1

29.loading corrupt files

df= pd.read_csv('Yelp_Reviews_Corrupt.csv',on_bad_lines='warn') #use warn or skip

30. Change index to a column in the df

df.set_index('linename')

change back by using reset_index(we remove index by using reset index)

df.reset_index()

31. Convering date to datetime type

df['date'] = pd.to_datetime(df['date'],format='%m/%d/%Y')

Code	Meaning	Example
%a	Weekday as locale's abbreviated name.	Mon
%A	Weekday as locale's full name.	Monday
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	1
%d	Day of the month as a zero-padded decimal number.	30
%-d	Day of the month as a decimal number. (Platform specific)	30
%b	Month as locale's abbreviated name.	Sep
%B	Month as locale's full name.	September
%m	Month as a zero-padded decimal number.	09
%-m	Month as a decimal number. (Platform specific)	9
%y	Year without century as a zero-padded decimal number.	13
%Y	Year with century as a decimal number.	2013
%H	Hour (24-hour clock) as a zero-padded decimal number.	07
%-H	Hour (24-hour clock) as a decimal number. (Platform specific)	7
%I	Hour (12-hour clock) as a zero-padded decimal number.	07
%-I	Hour (12-hour clock) as a decimal number. (Platform specific)	7
%p	Locale's equivalent of either AM or PM.	AM
%M	Minute as a zero-padded decimal number.	06
%-M	Minute as a decimal number. (Platform specific)	6

Get day of week from date from 0-6(The day of the week with Monday=0, Sunday=6.)

s.dt.dayofweek

df['day_of_week'] = df['date'].dt.day_of_week

get day name eg mon, tue etc

s.dt.day_name()

32.use value counts to understand the distribution of categorical data

33.Rename abbrevition to long format by using map

```
division_mapping = {
    'IRT':'Interborough Rapid Transit Company',
    'IND':'Independent Subway System',
    'BMT':'Brooklyn–Manhattan Transit Corporation',
    'PTH':'Port Authority Trans-Hudson (PATH)',
    'SRT':'Staten Island Rapid Transit',
    'RIT':'Roosevelt Island Tram'
}
df['DIVISION'] = df['DIVISION'].map(division_mapping)
```

output

```
Interborough Rapid Transit Company        72198
Independent Subway System                 69274
Brooklyn–Manhattan Transit Corporation    41727
Port Authority Trans-Hudson (PATH)        12788
Staten Island Rapid Transit                1386
Roosevelt Island Tram                       252
Name: DIVISION, dtype: int64
```

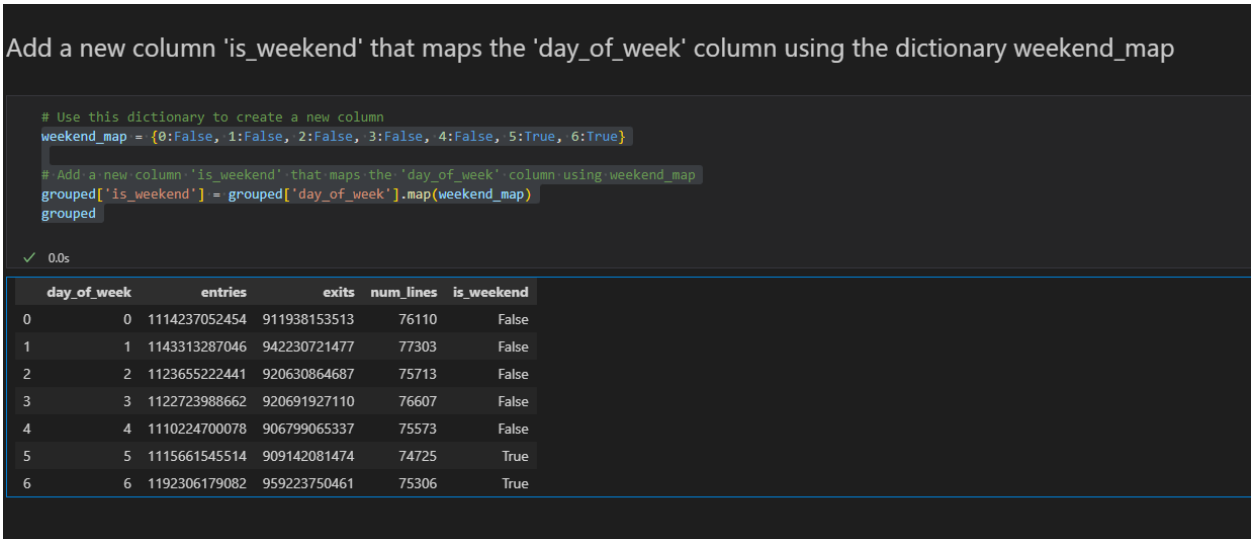
33.bWorks same as this

```
weekend_map = {0:False, 1:False, 2:False, 3:False, 4:False, 5:True, 6:True}
```

Add a new column 'is_weekend' that maps the 'day_of_week' column using weekend_map

grouped['is_weekend'] = grouped['day_of_week'].map(weekend_map)

grouped



	day_of_week	entries	exits	num_lines	is_weekend
0	0	1114237052454	911938153513	76110	False
1	1	1143313287046	942230721477	77303	False
2	2	1123655222441	920630864687	75713	False
3	3	1122723988662	920691927110	76607	False
4	4	1110224700078	906799065337	75573	False
5	5	1115661545514	909142081474	74725	True
6	6	1192306179082	959223750461	75306	True

34.Check n in column(check if sring contains

```
def contains_n(text):
    return 'N' in text


[df['LINENAME'].map(contains_n)]
df['LINENAME'].apply(contains_n)
#add a column that represents true and false
*df['On_N_Line'] = df['LINENAME'].map(contains_n)
or use map and lambda
*df['On_N_Line'] = df['LINENAME'].map(lambda x:'N' in x)
*df['On_N_Line'] = df['LINENAME'].str.contains('N',regex=False)
```

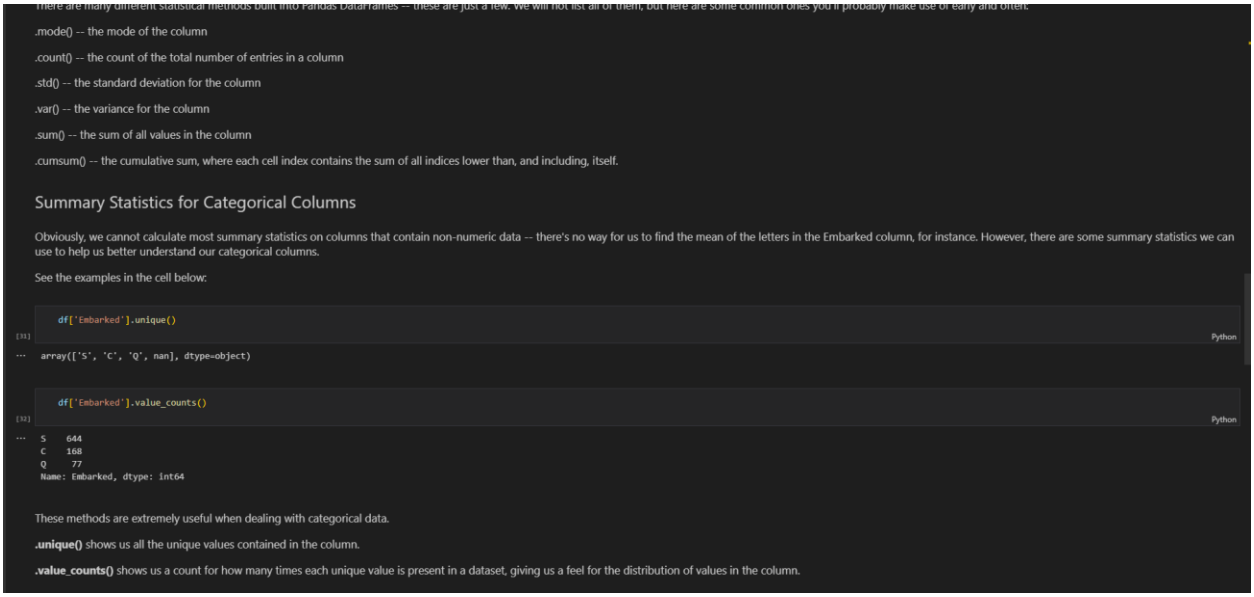
36.Convert a column to a specific data type

df['ENTRIES'] = df['ENTRIES'].astype(int) #converts to a sprcific dtype eg float,int etc

35.If you have index already in dataset and don’t want pandas to add its own

df= pd.read_csv('titanic.csv',index_col=0)

36.summary statics for categorical values is unique and value counts



37.covert whole data frame by using applymap eg convert all df to sring

string_df = df.applymap(lambda x: str(x))

string_df.info()

```
string_df = df.applymap(lambda x: str(x))
string_df.info()
```

[33]

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    object
 1   Survived     891 non-null    object
 2   Pclass       891 non-null    object
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    object
 6   SibSp        891 non-null    object
 7   Parch        891 non-null    object
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    object
 10  Cabin        891 non-null    object
 11  Embarked     891 non-null    object
dtypes: object(12)
memory usage: 90.5+ KB
```

-animals.applymap(type)

```
# This line will apply the base `type()` function to
# all entries of the DataFrame.
animals.applymap(type)
```

✓ 0.0s

	animal_id	name	datetime	monthyear	date
0	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
1	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
2	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
3	<class 'str'>	<class 'float'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
4	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
...	...	...	...	...	
995	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
996	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
997	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
998	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c
999	<class 'str'>	<class 'str'>	<class 'pandas._libs.tslibs.timestamps.Timesta...	<class 'str'>	<c

38. Whereas **.info()** provides statistics about the **DataFrame itself**, .**describe()** returns output containing **basic summary statistics** about the data contained with the DataFrame.

39.get total of unique by using nunique

df['play_star_rating'].nunique()

40.Styling matplotlib using stle function

plt.style.available

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (10,10)
plt.style.available
```

```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

plt.style.use('seaborn')

plt.style.use('fivethirtyeight')

41. matplotlib

a) Scatter plot

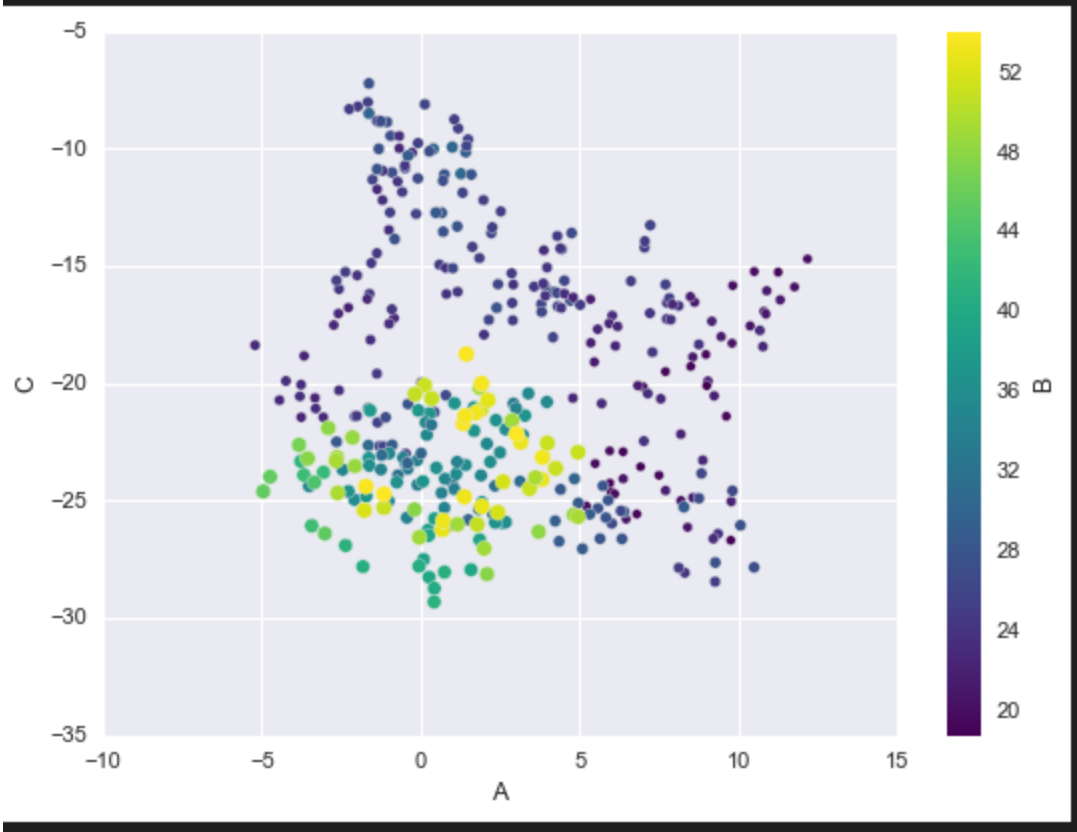
```
data.plot('A','B',kind='scatter')
```

or use

```
data.plot.scatter('A','C',
                  c ='B',
                  s = data['B'],
                  colormap = 'viridis'
                  );
```

Link to **colormaps**(https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html)

```
data.plot.scatter('A','C',
                  c ='B',
                  s = data['B'],
                  colormap = 'viridis'
                  );
```



b)box plot

data.plot.box();

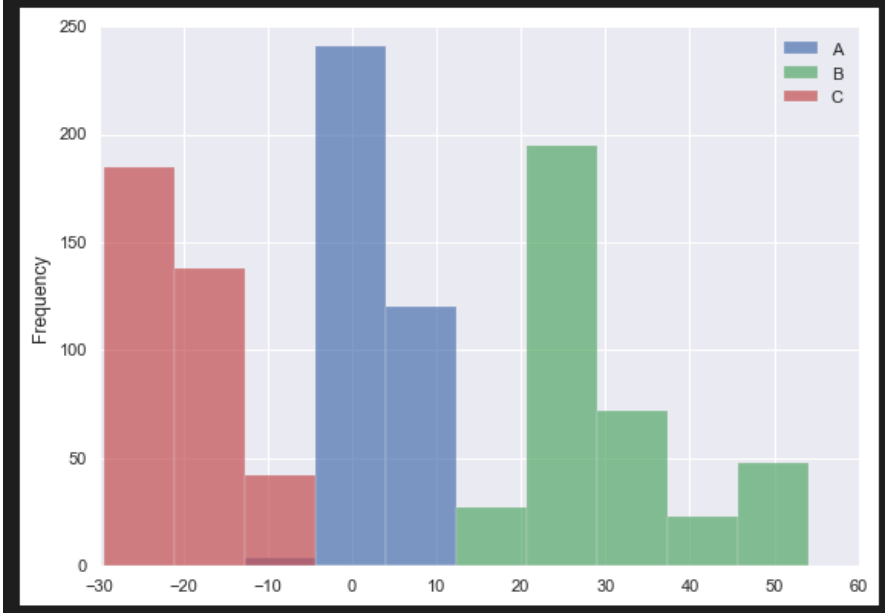
c)Histogram

data.plot.hist(alpha = 0.7);

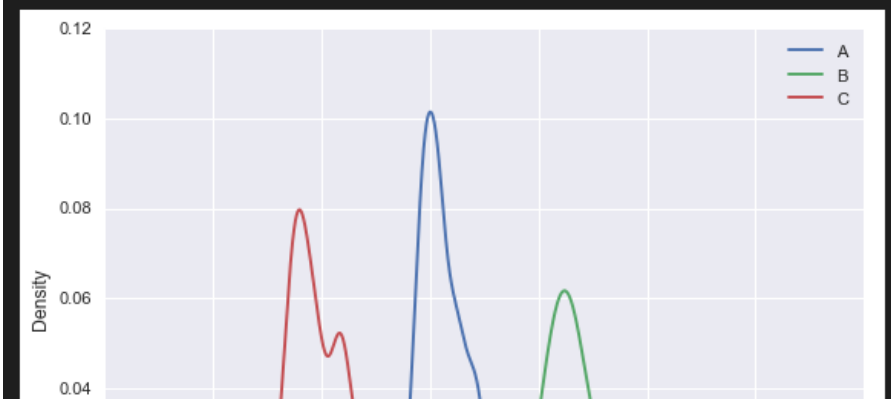
d)Kernel density plot(kde)

data.plot.kde();

```
data.plot.hist(alpha = 0.7);
```




```
#Kernel Density Estimate plots
#useful for visualizing an estimate of a variable's probability density function.
data.plot.kde();
```



e)Visualizing high dimension data (way too many columns/variables to inspect individually

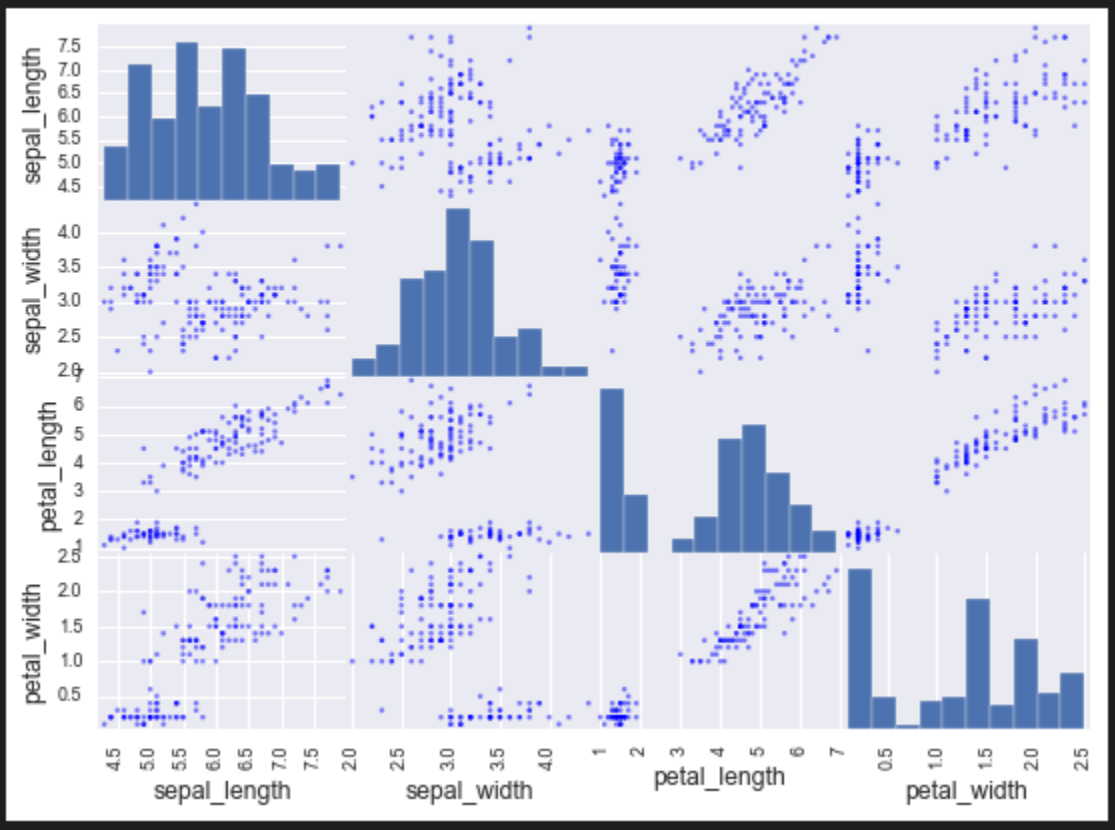
) multivariate with **Scatter matrix(way of comparing each column in a DataFrame to every other column in a pairwise fashion)**

The ****scatter matrix**** creates scatter plots between the different variables and histograms along the diagonals.

This allows us to quickly see some of the more obvious patterns in the dataset. Let's use it to visualize the iris DataFrame and see what insights we can gain from our data. We will use the method pd.tools.plotting.scatter_matrix() and pass in our dataset as an argument.

pd.plotting.scatter_matrix(iris);

```
pd.plotting.scatter_matrix(iris);
```



Looking at above scatter plots generated by scatter_matrix(), it appears that there are some distinct groupings of the values which could be indicative of clustering/ grouping etc. Such handy visual analytics allow us to better decide a course of action for in-depth predictive analysis.

f)parallel plots

Pandas includes a plotting tool for creating parallel coordinates plots which could be a great way to visualize multivariate data.

Parallel coordinate plots are a common way of visualizing high dimensional multivariate data. Each variable in the dataset corresponds to an equally-spaced, parallel, vertical line. The values of each variable are then connected by lines between for each individual observation.

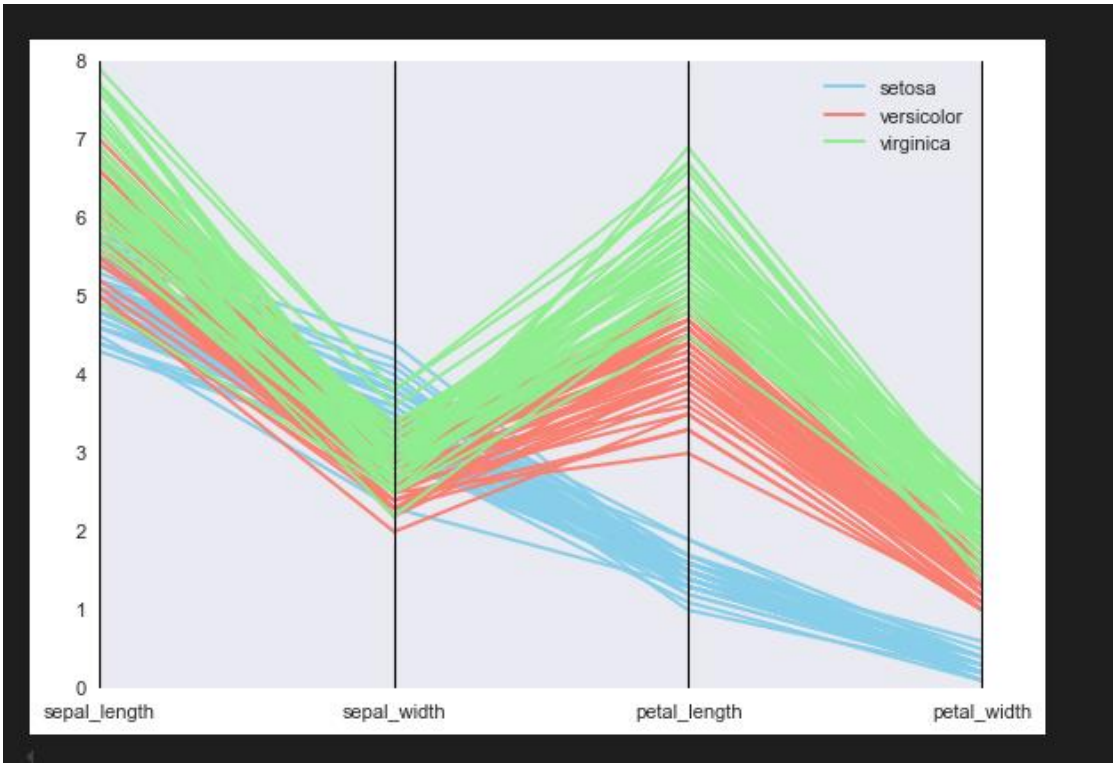
Let's create a parallel plot for the 4 predictor variables in the iris dataset and see if we can make any further judgments about the nature of data. We will use the pd.plotting.parellel_coordinates() function and pass in the iris dataset with the response column (species) as an argument, just like we saw above. Let's also apply some customizations.

Color the lines by class given in 'species' column (this will allow handy inspection to see any patterns).

set a colormap with 3 colors to show species

colormap = ('skyblue','salmon','lightgreen')

pd.plotting.parallel_coordinates(iris,'species',color=colormap)



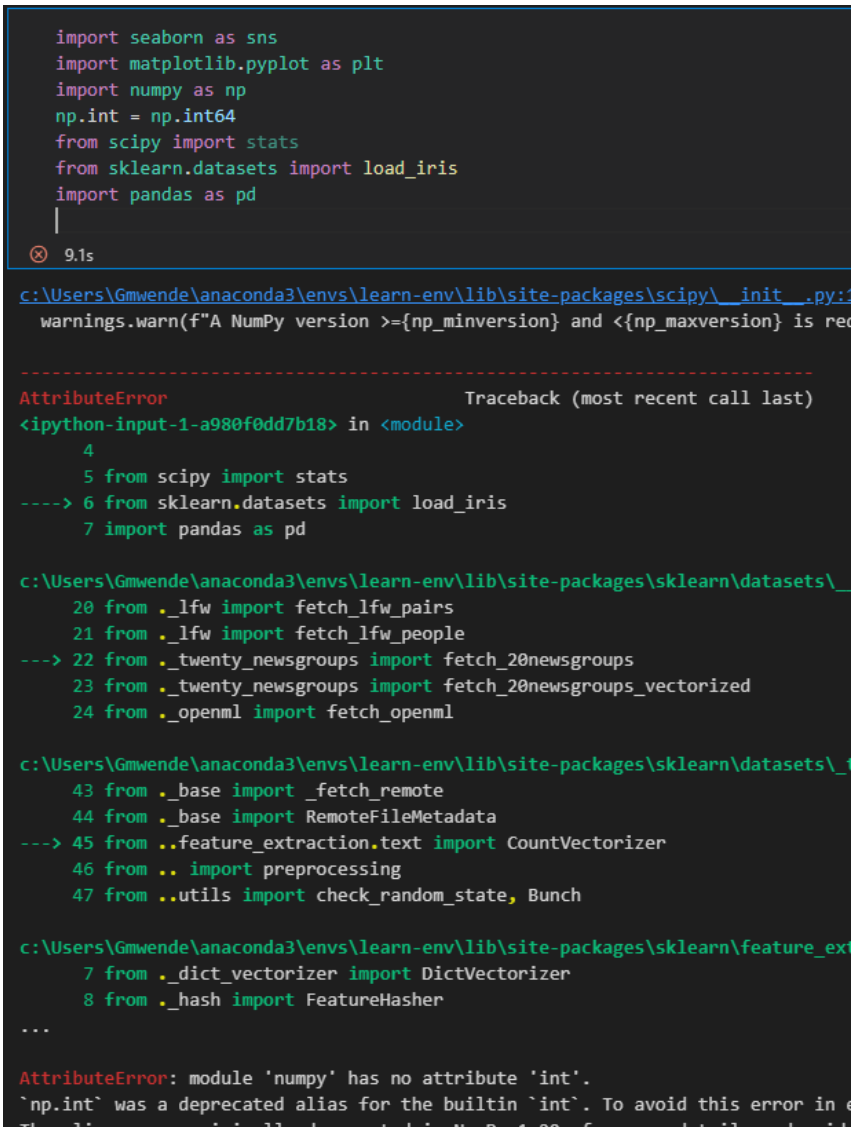
So, what do we learn from the parallel plot?

Looking at our parallel plot, we can see that the petal length and petal width are two variables that split the different species fairly clearly. Iris virginica has the longest and the widest petals among all flower types. Iris setosa has the shortest and narrowest petals etc.

These initial set of statistical observations go a long way in the field of data analytics. We may decide to apply extra pre-processing to the data, or decide which are the best predictor variables for our analysis - based on the results of quick visualizations in pandas.

42. Get error **numpy** has no attribute **int-** add np.int = np.int64

If there is scipy used make sure the code is above it as from the screenshot below



43. Saving figures

plt.savefig('Images/parabola.png')

44. Add line plot on a scatter plot

fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(X,y,s=100)

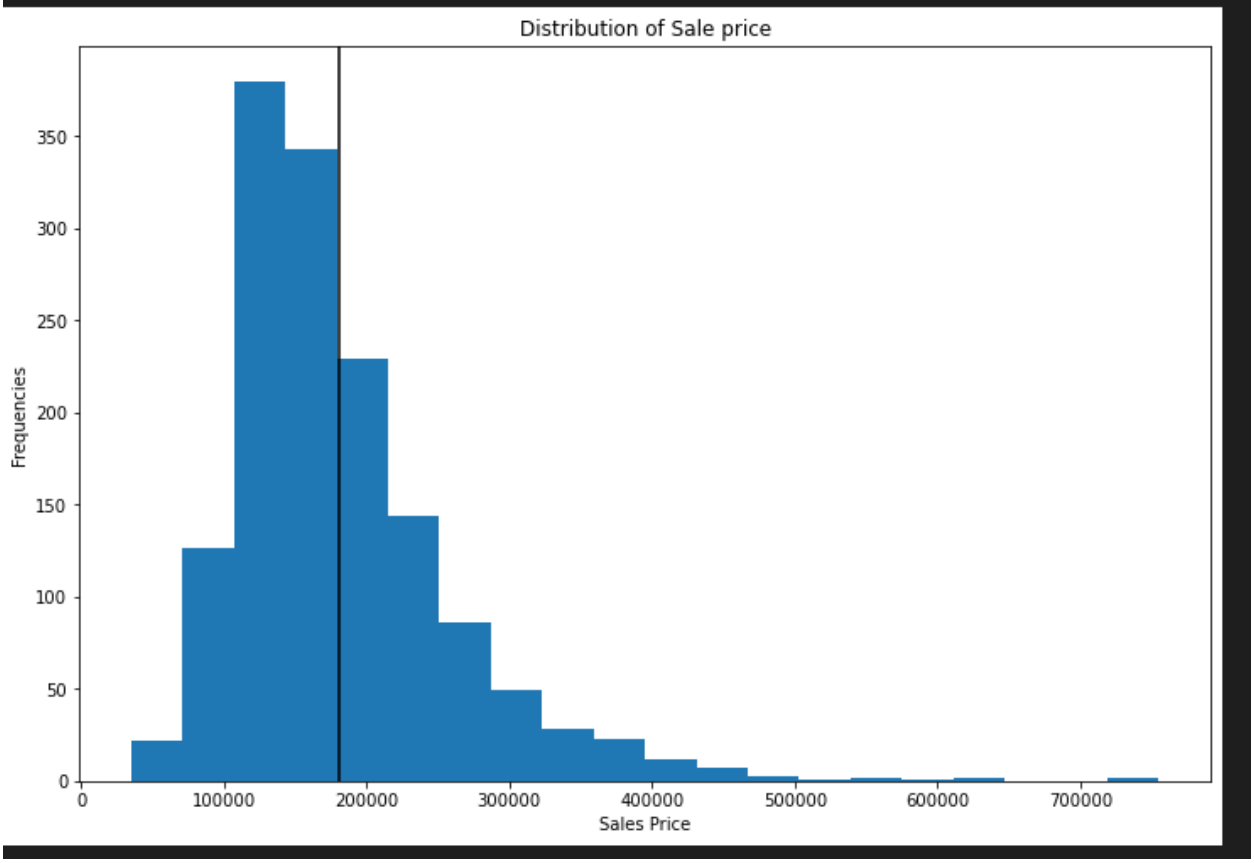
#add line plot ontop

ax.plot(X,10*X+50,c='orange');

45. Adding horizontal line on visualization

ax.axvline(x=np.mean(df['SalePrice']),c='k')(Adds horizontal for mean)

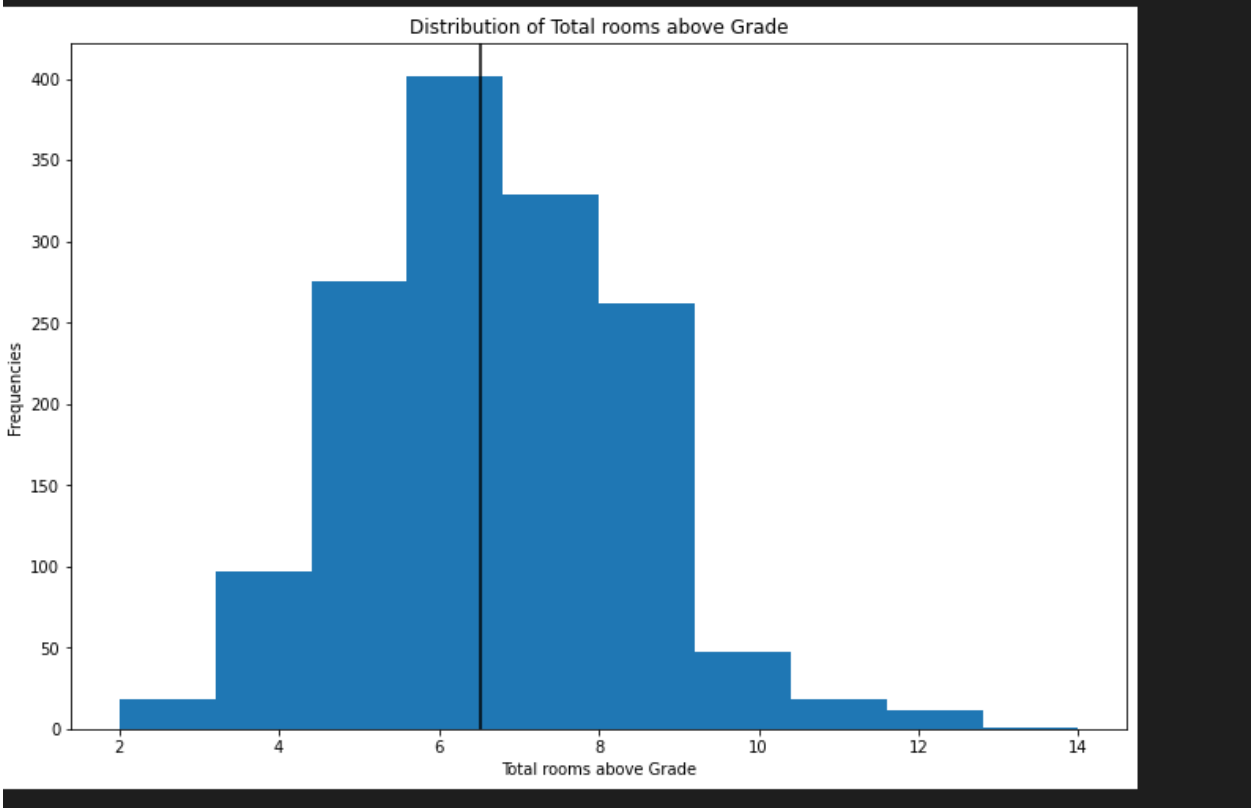
```
import numpy as np
fig, ax = plt.subplots(figsize=(12,8))
ax.hist(df['SalePrice'],bins=20)
ax.axvline(x=np.mean(df['SalePrice']),c='k')
ax.set_xlabel('Sales Price')
ax.set_ylabel('Frequencies')
ax.set_title('Distribution of Sale price');
```



Typical average sales is 180k , from the histogram we can see that the data is skewed and therefore

we can use median to conclude the central sale is 163k

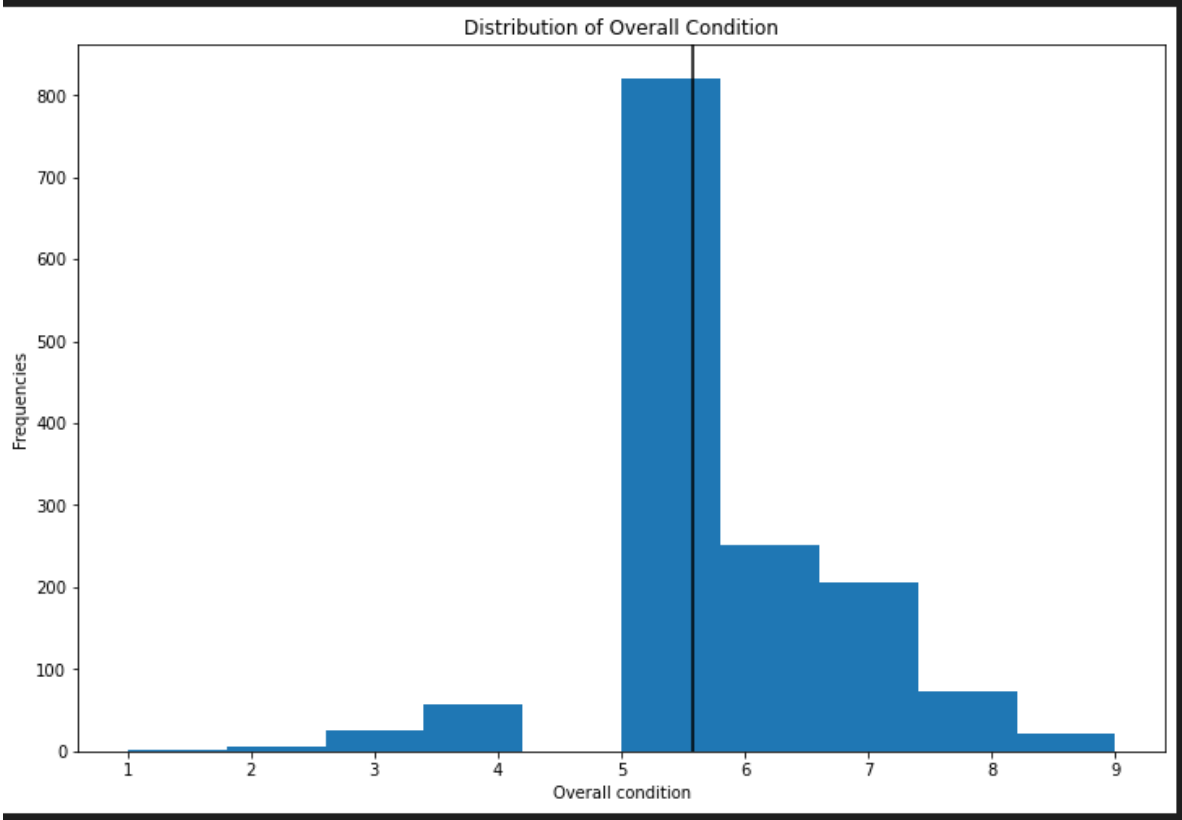
```
import numpy as np
fig, ax = plt.subplots(figsize=(12,8))
ax.hist(df['TotRmsAbvGrd'])
ax.axvline(x=np.mean(df['TotRmsAbvGrd']),c='k')
ax.set_xlabel('Total rooms above Grade')
ax.set_ylabel('Frequencies')
ax.set_title('Distribution of Total rooms above Grade');
```



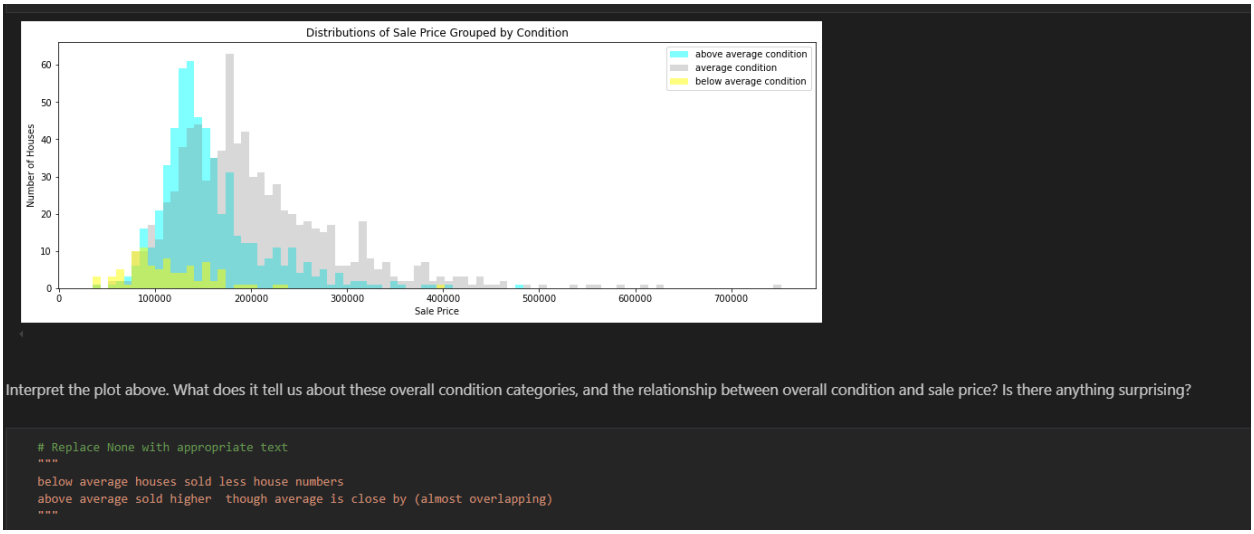
The data looks uniformly distributed therefore average Total rooms is 6.5

i'll use the mean since it known as average by most peopl

```
import numpy as np
fig, ax = plt.subplots(figsize=(12,8))
ax.hist(df['OverallCond'])
ax.axvline(x=np.mean(df['OverallCond']),c='k')
ax.set_xlabel('Overall condition')
ax.set_ylabel('Frequencies')
ax.set_title('Distribution of Overall Condition');
```



data looks uniformly distrubted and overall condition is 5.5.will use mean since its mostly known as the average



Interpret the plot above. What does it tell us about these overall condition categories, and the relationship between overall condition and sale price? Is there anything surprising?

```
# Replace None with appropriate text
"""
below average houses sold less house numbers
above average sold higher  though average is close by (almost overlapping)
"""
```

46.Exploring how a certain colun correlates to others

df.corr(numeric_only = True)['SalePrice'].sort_values()

#overallQual is the most postiviley correlated to price

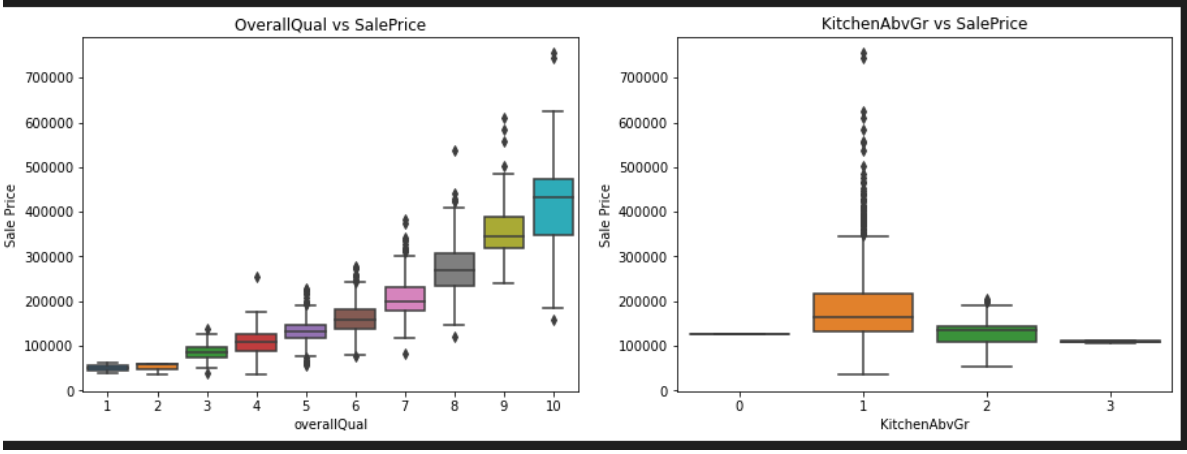
#KitchenAbvGr is the most negatively correlated to price

```
df.corr(numeric_only = True)['SalePrice'].sort_values()
#overallQual is the most postiviley correlated to price
```

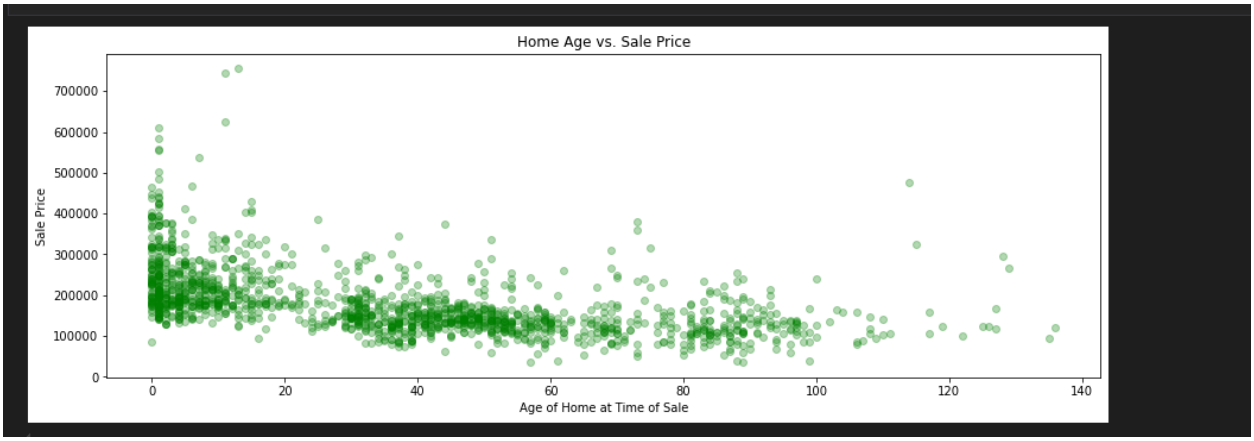
```
KitchenAbvGr    -0.135907
EnclosedPorch   -0.128578
MSSubClass      -0.084284
OverallCond     -0.077856
YrSold          -0.028923
LowQualFinSF    -0.025606
MiscVal         -0.021190
BsmtHalfBath    -0.016844
BsmtFinSF2      -0.011378
3SsnPorch        0.044584
MoSold           0.046432
PoolArea         0.092404
ScreenPorch      0.111447
BedroomAbvGr     0.168213
BsmtUnfSF        0.214479
BsmtFullBath     0.227122
LotArea          0.263843
HalfBath         0.284108
OpenPorchSF      0.315856
2ndFlrSF         0.319334
WoodDeckSF       0.324413
LotFrontage      0.351799
BsmtFinSF1       0.386420
Fireplaces       0.466929
MasVnrArea       0.477493
...
GarageCars       0.640409
GrLivArea        0.708624
OverallQual      0.790982
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

47.Box plot for most positively and most negatively related

```python
import seaborn as sns

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,5))

# Plot distribution of column with highest correlation
sns.boxplot(
    x=df['OverallQual'],
    y=df["SalePrice"],
    ax=ax1
)
# Plot distribution of column with most negative correlation
sns.boxplot(
    x=df['KitchenAbvGr'],
    y=df["SalePrice"],
    ax=ax2
)

# Customize labels
ax1.set_title('OverallQual vs SalePrice')
ax1.set_xlabel('overallQual')
ax1.set_ylabel("Sale Price")
ax2.set_title('KitchenAbvGr vs SalePrice')
ax2.set_xlabel('KitchenAbvGr')
ax2.set_ylabel("Sale Price");
```



sale price increases with overall quality and decreases with kitchen above grade



cluster where less ages sold more, middle abit less and the older the house the less the price

48.Using lamda functions

a)Checking length of words in a string

df['text'].map(lambda x: len(x.split())).head()

b)lambda fuctions with conditionals

df['text'].map(lambda x: 'Good' if any([word in x.lower() for word in ['awesome','love','good','great']]) else 'Bad')

c)select year from column(Slicing column using lambda)

df['date'].map(lambda x: x[:4])

- Can also be achived by

```
pd.to_datetime(df['date'],format='%Y-%m-
%d').dt.year.head()
```

d)also useful in sorting eg sorted on names will sort by first name wha if you wanted to sort by last name

sorted(names,key=lambda x: x.split()[1])

```
# Without a key
names = ['Miriam Marks','Sidney Baird','Elaine Barrera','Eddie Reeves','Marley Beard',
         'Jaiden Liu','Bethany Martin','Stephen Rios','Audrey Mayer','Kameron Davidson',
         'Carter Wong','Teagan Bennett']
sorted(names)
```

✓ 0.0s

```
['Audrey Mayer',
 'Bethany Martin',
 'Carter Wong',
 'Eddie Reeves',
 'Elaine Barrera',
 'Jaiden Liu',
 'Kameron Davidson',
 'Marley Beard',
 'Miriam Marks',
 'Sidney Baird',
 'Stephen Rios',
 'Teagan Bennett']
```

```
# sorting by last name
sorted(names,key=lambda x: x.split()[1])
```

✓ 0.0s

```
['Sidney Baird',
 'Elaine Barrera',
 'Marley Beard',
 'Teagan Bennett',
 'Kameron Davidson',
 'Jaiden Liu',
 'Miriam Marks',
 'Bethany Martin',
 'Audrey Mayer',
 'Eddie Reeves',
 'Stephen Rios',
 'Carter Wong']
```

- General approach to lambda is by first solving individual cases first
  Eg

```
example = df.text.iloc[0] #df['text'].iloc[0]
example
```

✓ 0.0s

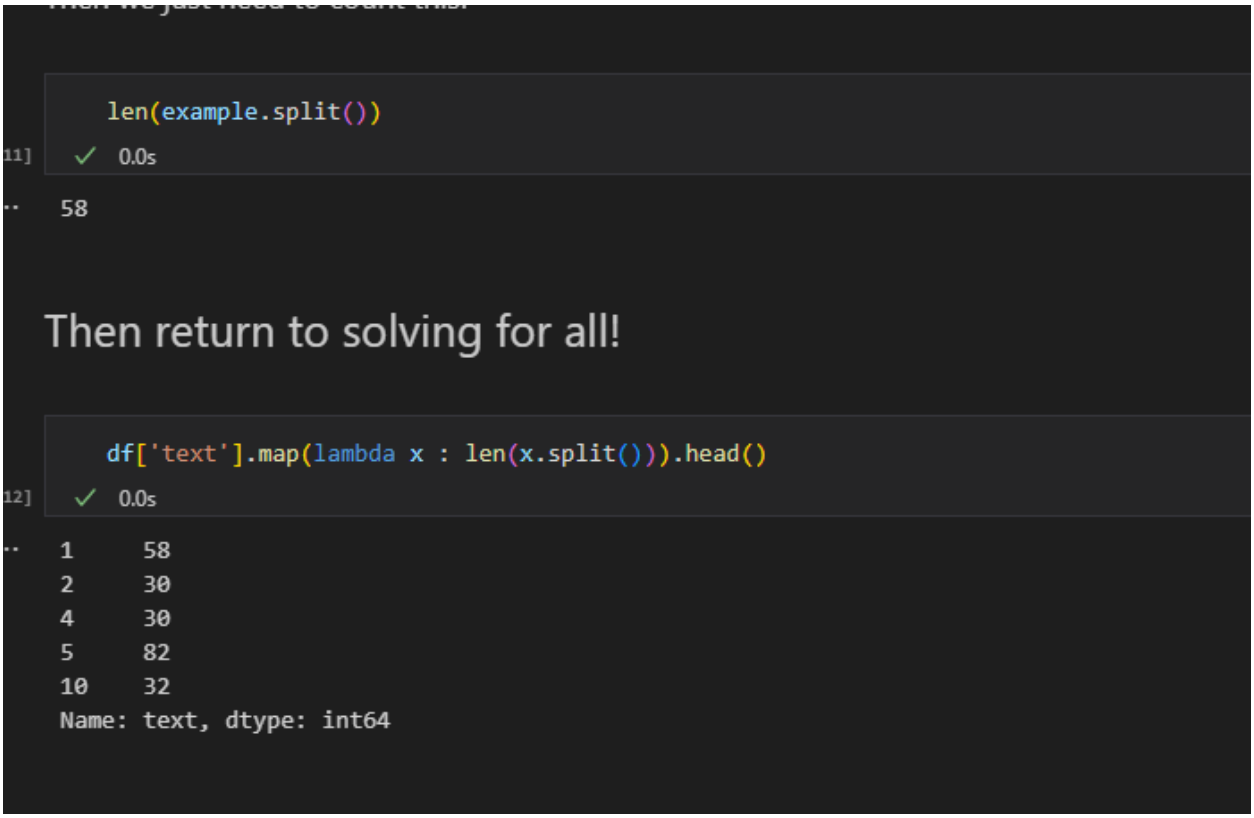
```
'I love this place! My fiance And I go here atleast once a week. Th
```

Then start writing the function for that example. For example, if we nee

```
example.split()
```

✓ 0.0s

```
['I',
 'love',
 'this',
 'place!',
 'My',
 'fiance',
 'And',
 'I',
 'go',
 'here',
 'atleast',
 'once',
 'a',
 'week.',
 'The',
 'portions',
 'are',
 'huge!',
 'Food',
 'is',
 'amazing.',
 'I',
 'love',
 'their',
 'carne',
...
 'sauce',
 'is',
 'different',
 'And',
```



48 b. Other Common Patterns: the % and // operators

Another common pattern that you find very useful is the modulus or remainder operator(%) as well

as the floor division operator(//).these are both useful when u want behavior such as 'every fourth element' or 'group of three consecutive elements'.Let's investigate a couple of examples.

The modulus operator (%)

Useful for queries such as 'every other element' or 'every fifth element' etc.

Combining % and //

Combining the two can be very useful, such as when creating subplots! Below we iterate through 12 elements arranging them into 3 rows and 4 columns.

fig, axes = plt.subplots(nrows=3,ncols=4,figsize=(10,10))

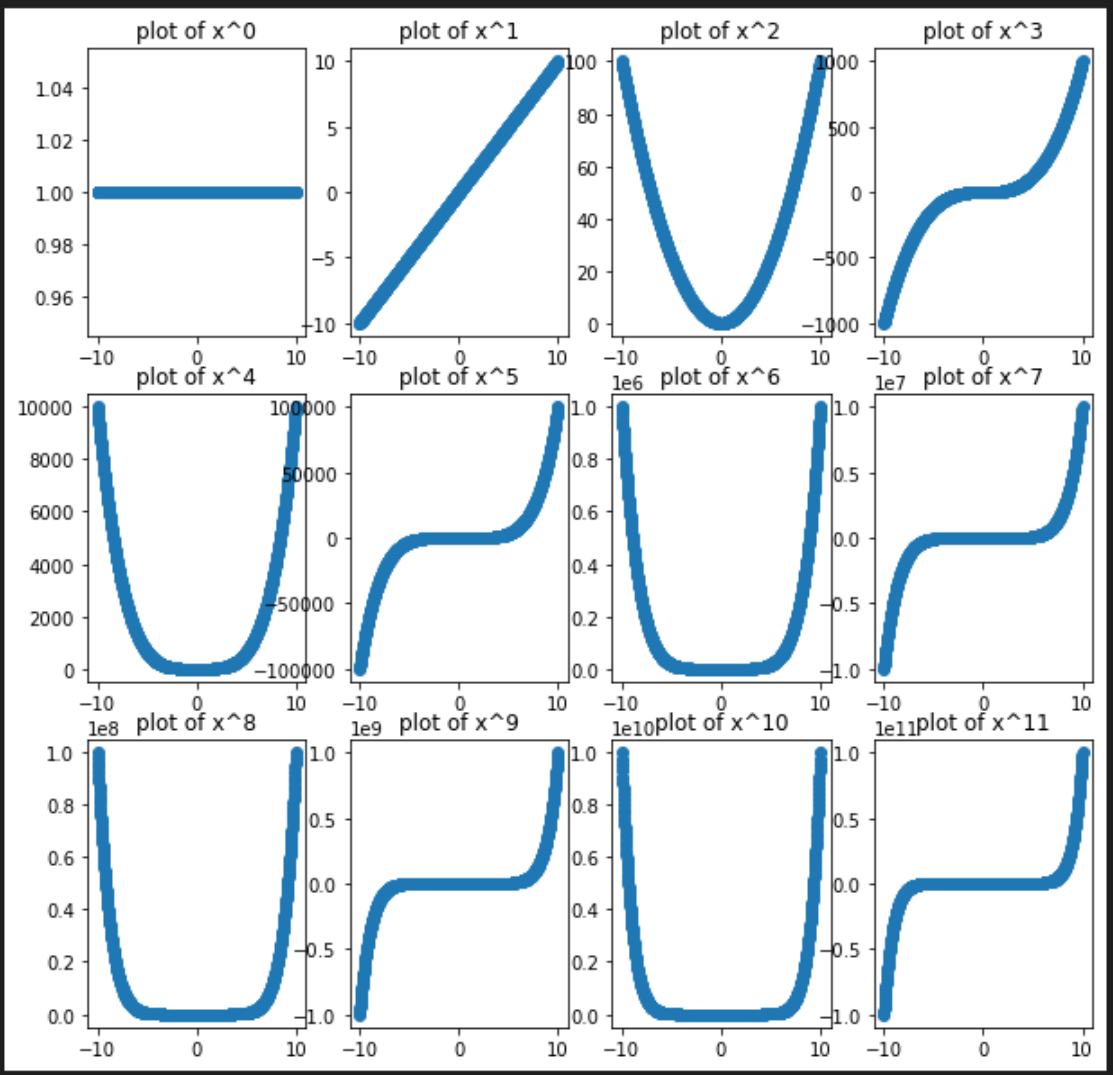
x = np.linspace(start=-10,stop=10,num=10*83)

for i in range(12):

```
    row = i//4
    col = i%4
    ax= axes[row,col] #ax= axes[row][col]
    ax.scatter(x,x**i)
    ax.set_title(f'plot of x^{i}')
plt.show()
```

```python
fig, axes = plt.subplots(nrows=3,ncols=4,figsize=(10,10))
x = np.linspace(start=-10,stop=10,num=10*83)
for i in range(12):
    row = i//4
    col = i%4
    ax= axes[row,col] #ax= axes[row][col]
    ax.scatter(x,x**i)
    ax.set_title(f'plot of x^{i}')
plt.show()
```

✓ 1.2s



c)Use a lambda function to create a new column called 'stars_squared' by squaring the stars column.

df['stars_squared'] = df['stars'].map(lambda x: x**2)

d)Get month from date(if date is string)

df['date'].map(lambda x: x[5:7]).head()

- **If date or you have converted to date**

```
- df['date'] = pd.to_datetime(df['date'],format='%Y-%m-
  %d')
- df['date'].map(lambda x: x.month)
```

e) What is the average number of words for a yelp review?

df['text'].map(lambda x: len(x.split())).mean()

f) Create a new column for the number of words in the review

df['no_of_words'] = df['text'].map(lambda x: len(x.split()))

g) Rewrite the following as a lambda function

```
def rewrite_as_lambda(value):
    if len(value) > 50:
        return 'Short'
    elif len(value) < 80:
        return 'Medium'
    else:
        return 'Long'
# Hint: nest your if, else conditionals
```

df['Review_length'] = df['Review_num_words'].map(lambda x: 'Short' if x < 50 else ('Medium' if x < 80 else 'Long'))

df['Review_length'].value_counts(normalize=True)

h) **Overwrite the 'date' column by reordering the month and day from YYYY-MM-DD to DD-MM-YYYY. Try to do this using a lambda function.**

If column in string

df['date'].map(lambda x : '{}-{}-{}'.format(x[-2:],x[5:7],x[:4]))

if column in date or have converted to date

df['date2'] = df['date'].map(lambda x : x.strftime('%d-%m-%Y'))

df[['date','date2']]

```
example = df['date'].iloc[0]
example.strftime('%d-%m-%Y')
```

✓ 0.0s

```
'13-11-2012'
```

```
df['date2'] = df['date'].map(lambda x : x.strftime('%d-%m-%Y'))
df[['date','date2']]
```

✓ 0.0s

	date	date2
1	2012-11-13	13-11-2012
2	2014-10-23	23-10-2014
4	2014-09-05	05-09-2014
5	2011-02-25	25-02-2011
10	2016-06-15	15-06-2016
...	...	...
689	2013-06-02	02-06-2013
4874	2016-08-14	14-08-2016
564	2016-06-14	14-06-2016
3458	2013-10-02	02-10-2013
4206	2016-08-15	15-08-2016

2610 rows × 2 columns

```
#Teachers code
df['date'].map(lambda x : '{}-{}-{}'.format(x[-2:],x[5:7],x[:4])) #if column not yet converte to date
```

50.Dealing with missing Data

On checking we have missing data on cabin(687), age(177) and embarked(2) on titanic dataset

a)Cabin Column

Now that we know how many missing values exist in each column, we can make some decisions about how to deal with them.

We'll deal with each column individually, and employ a different strategy for each.

- Determine what percentage of rows in this column contain missing values

df['Cabin'].isna().value_counts(normalize=True)

or

len(df[df['Cabin'].isna()])/len(df)

```
# Your code here
df['Cabin'].isna().value_counts(normalize=True)
```

```
True     0.771044
False    0.228956
Name: Cabin, dtype: float64
```

```
#Teachers code
len(df[df['Cabin'].isna()])/len(df)
```

```
0.7710437710437711
```

```
df['Cabin'].nunique()
```

```
147
```

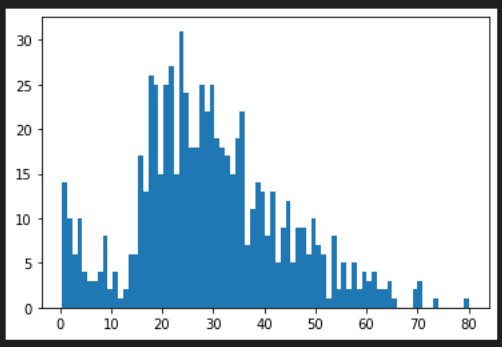
With this many missing values its best to drop the column completely

df.drop('Cabin',axis=1,inplace=True)

b)Age column

Plot a histogram of values in the `'Age'` column

```
# Your code
fig, ax = plt.subplots()
ax.hist(df['Age'],bins=80);
#teachers code
#df['Age'].plot(kind='hist', bins=80)
```




```
# Your code here
mean_age = df['Age'].mean() #np.mean(df['Age'])
print('mean_age',mean_age)
median_age = df['Age'].median() #np.median(df['Age'])
print('median_age',median_age)
```

```
mean_age 29.69911764705882
median_age 28.0
```

From the visualization above we see that the data has a slighltly positive skew therefore replace all Na with median

df['Age'].fillna(value=df['Age'].median)

or

median_age = df['Age'].median()

df['Age'].fillna(median_age,inplace=True)

c)**Embarked Column**

Dropping rows that contain missing values

Perhaps the most common solution to dealing with missing values is to simply drop any rows that contain them. Of course, this is only a good idea if the number dropped does *not constitute a significant portion of our dataset*. Often, you'll need to make the overall determination to see if dropping the values is an acceptable loss, or if it is a better idea to just drop an offending column (e.g. the `'Cabin'` column) or to impute placeholder values instead.

```
df[df['Embarked'].isna()] #checking missing rows

df.dropna(subset='Embarked',inplace=True)

#Teachers code

df = df.dropna()

df.isna().sum()
```

We've dealt with all the **_*obvious*_** missing values, but we should also take some time to make sure that there aren't symbols or numbers included that are meant to denote a missing value.

d)Checking for place holders

A common thing to see when working with datasets is missing values denoted with a preassigned code or symbol. Let's check to ensure that each categorical column contains only what we expect.

In the cell below, return the unique values in the `'Embarked'`, `'Sex'`, `'Pclass'`, and `'Survived'` columns to ensure that there are no values in there that we don't understand or can't account for

```
print('Embarked',df['Embarked'].unique())

print('Sex',df['Sex'].unique())

print('Pclass',df['Pclass'].unique())

print('Survived',df['Survived'].unique())
```

or put togther

```
for col in ['Embarked','Sex', 'Pclass','Survived']:
    print(f'Values for {col}:\n{df[col].unique()}\n\n')
```

```
    # Your code here
    print('Embarked',df['Embarked'].unique())
    print('Sex',df['Sex'].unique())
    print('Pclass',df['Pclass'].unique())
    print('Survived',df['Survived'].unique())
```

```
Embarked ['S' 'C' 'Q']
Sex ['male' 'female']
Pclass ['3' '1' '2' '?']
Survived [0 1]
```

```
    #Teachers code
    for col in ['Embarked','Sex', 'Pclass','Survived']:
        print(f'Values for {col}:\n{df[col].unique()}\n\n')
```

```
Values for Embarked:
['S' 'C' 'Q' nan]


Values for Sex:
['male' 'female']


Values for Pclass:
['3' '1' '2' '?']


Values for Survived:
[0 1]
```

It looks like the `'Pclass'` column contains some missing values denoted by a placeholder.

In the cell below, investigate how many placeholder values this column contains. Then, deal with these missing values using whichever strategy you believe is most appropriate in this case.

df.Pclass.value_counts(normalize=True)

```
#Teachers code
df.Pclass.value_counts(normalize=True)
```

```
3    0.526375
1    0.225589
2    0.193042
?    0.054994
Name: Pclass, dtype: float64
```

```
# Observation: account for 5% of the data
# Method: randomly select a class acccording to current distribution
rel_prob = [.53, .22, .19]
prob = [i/sum(rel_prob) for i in rel_prob]
def impute_pclass(value):
    if value == '?':
        return np.random.choice(['3','1','2'], p=prob)
    else:
        return value
df.Pclass = df.Pclass.map(lambda x: impute_pclass(x))
df.Pclass.value_counts(normalize=True)
```

```
3    0.558923
1    0.234568
2    0.206510
Name: Pclass, dtype: float64
```

What is the benefit of treating missing values as a separate valid category? What is the benefit of removing or replacing them? What are the drawbacks of each? Finally, which strategy did you choose? Explain your choice below.

By treating missing values as a separate category, information is preserved.

Perhaps there is a reason that this information is missing.

By removing or replacing missing information, we can more easily conduct mathematical analyses which require values for computation.

I chose to randomly replace for now. I could have just as easily removed the data.

Concerns include that I imputed the wrong value (indeed it was a random guess).

The strategy for dealing with missing data will depend on our desired application,

but regardless of the approach taken, the ramifications of how missing data are handled must be considered.

For example, imputing the median of our age reduces variance

and assumes that a new value would be close to the center of the distribution

(albeit this assumption is statistically likely

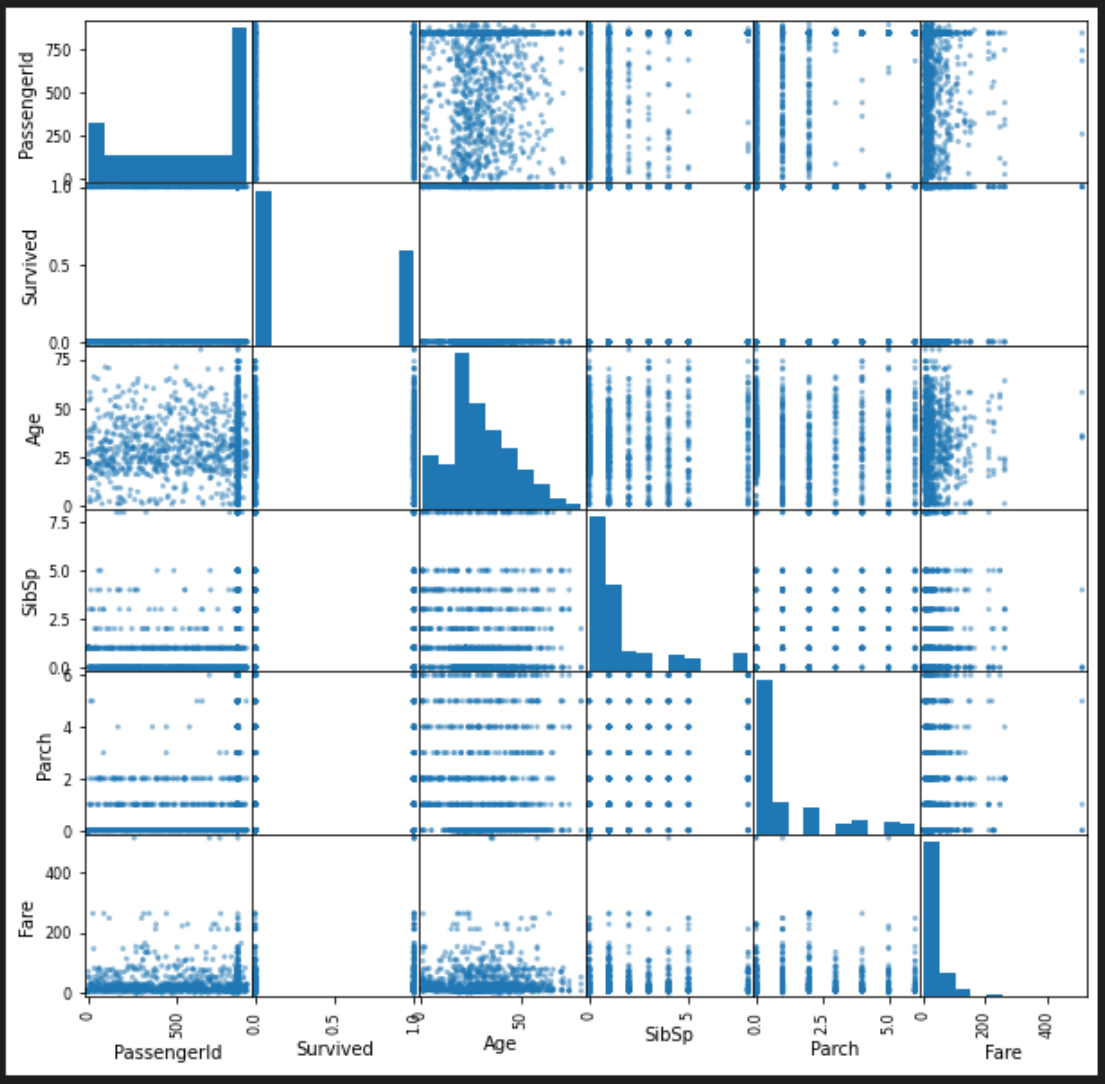
51.More on Missing Data

While imputing the mean or median methods of dealing with missing values, these standard methods do have caveats. For example, doing so will reduce the overall variance of yor dataset which should be taken into account when performing subsequent analyses or training a Ml algorithm on the dataset

-using the titanic dataset again here is the scatter matrix

pd.plotting.scatter_matrix(df,figsize=(10,10));

```
pd.plotting.scatter_matrix(df,figsize=(10,10));
```



1.Check for missing data

Typically, the first step in checking for missing data is to simply use the .info() method. This gives us various information about the columns including their data type and the number of non-missing values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1391 entries, 0 to 1390
Data columns (total 12 columns):
PassengerId    1391 non-null float64
Survived       1391 non-null float64
Pclass         1391 non-null object
Name           1391 non-null object
Sex            1391 non-null object
Age            1209 non-null float64
SibSp          1391 non-null float64
Parch          1391 non-null float64
Ticket         1391 non-null object
Fare           1391 non-null float64
Cabin          602 non-null object
Embarked       1289 non-null object
dtypes: float64(6), object(6)
memory usage: 130.5+ KB
```

As you can see, 'Age' and 'Cabin' have a substantial amount of missing values, and 'Embarked' has two extraneous missing values.

2. Check for duplicates

While df.info() is a good initial spot check for missing values, it may not catch more subtle anomalies in the data such as duplicates. While these values are populated, it is always worrisome if we have observation rows with identical data.

While df.info() is a good initial spot check for missing values, it may not catch more subtle anomalies in the data such as duplicates. While these values are populated, it is always worrisome if we have observation rows with identical data.

duplicates = df[df.duplicated()]

print(len(duplicates))

100

- df.duplicated().value_counts()

Similarly, if a feature such as 'PassengerId' can be assumed to be unique, we can further check if there are duplicate rows based on a subset of the DataFrame columns.

```
duplicates = df[df.duplicated(subset='PassengerId')]

print(len(duplicates))
```

500

3. **Check for extraneous values**

Sometimes, missing values are even further hidden within a dataset. For example, sometimes an entry such as 999999 is used for missing values, or an arbitrary date such as 12-01-1970 might be set for unknown dates. In general, doing a quick eyeball and previewing the top occurring values for each feature can help further tease out peculiarities in the dataset.

```
for col in df.columns:
    print(col, '\n', df[col].value_counts(normalize=True).head(), '\n\n')
```

```
for col in df.columns:
    print(col, '\n', df[col].value_counts(normalize=True).head(), '\n\n')
```

```
PassengerId
 839.0    0.288282
1.0      0.072610
881.0    0.000719
757.0    0.000719
195.0    0.000719
Name: PassengerId, dtype: float64


Survived
 0.0    0.618979
1.0    0.381021
Name: Survived, dtype: float64


Pclass
 3    0.475198
1    0.219267
2    0.199137
?    0.106398
Name: Pclass, dtype: float64


Name
 Braund, Mr. Owen Harris                     0.072610
Stone, Mrs. George Nelson (Martha Evelyn)    0.003595
Maioni, Miss. Roberta                        0.002876
Butler, Mr. Reginald Fenton                  0.002876
Markun, Mr. Johann                           0.002876
Name: Name, dtype: float64


Sex
 male      0.641265
female    0.358735
Name: Sex, dtype: float64
```

- You can see that we've uncovered another case of missing data that did not show up before. The 'Pclass' feature has ? for roughly 10% of the entries.

Choosing a methodology

How do you choose which method for dealing with missing data to use? The answer will depend on the scenario and specifics to the application itself. As a

general rule of thumb, we tend **towards imputing values rather than dropping them**, as we wish to use as much information as possible. That said, larger gaps where data is missing can pose more substantial problems, and thereby warrant alternative approaches. We'll take a look at specific cases below in more detail, but here's a quick table of your options.

	Continuous	Categorical
Delete	Delete rows (observations) Delete column (entire variable)	Delete rows (observations) Delete column (entire variable)
Replace	replace using median/mean	replace using mode
Keep	keep as NA (not possible for many ML algorithms)	NA category

Imputing values

Imputing values is often a go to option when dealing with missing data. For example, if we are building a machine learning model with the data, many algorithms cannot handle missing values. By imputing data, we still get to use the full extent of the data at hand without having to throw away data, which, as you know, is an easy option.

Considerations when imputing

When imputing missing values, keep in mind that you are influencing the distribution of this variable. For example, if you impute the mean, you will reduce the variance of that feature.

When to drop rows

Dropping rows is an appropriate choice if there are very few missing values to start with. After all, we do not wish to throw away troves of data if we have it, so cases in which there are larger occurrences of missing values, dropping all occurrences is typically inadvisable.

When to drop columns

Dropping columns is typically a last case resort. That said, if a feature does not add predictive value to the machine learning algorithm driving your application, dropping said feature has no cost.

A few simple lines such as this can easily subset your DataFrame:

cols_to_remove = ['col1', 'col2']

cols = [col for col in df.columns if col not in cols_to_remove]

subset = df[cols]

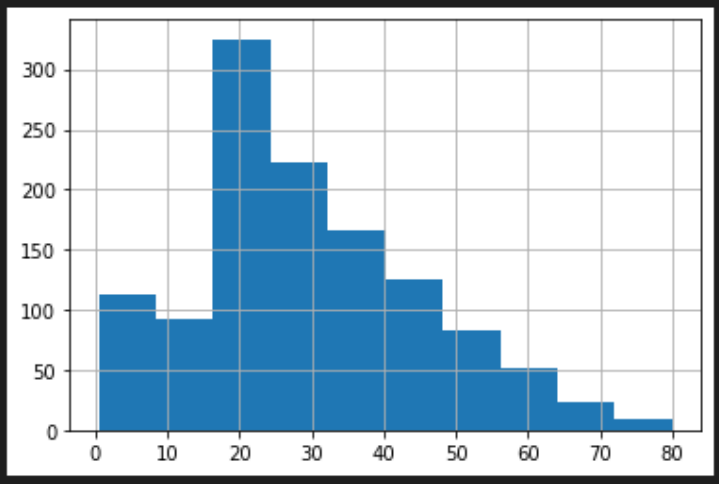
52. CHECKING IMPACT OF ON HOW I HANDLE MISSING VALUES

- check for mean, median and std for column

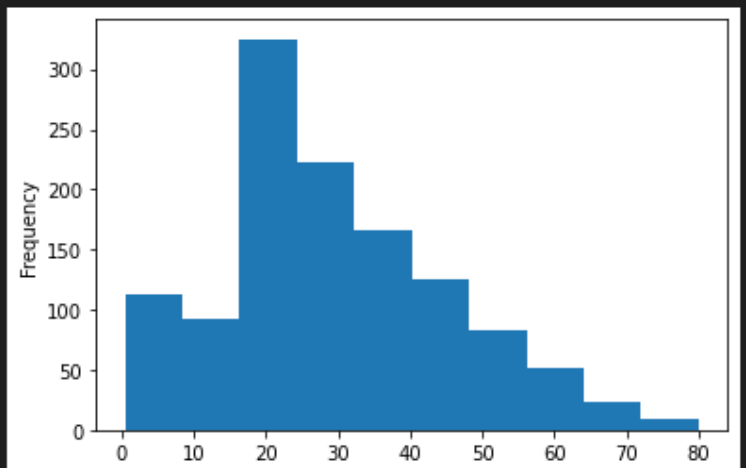
```
print(df['Age'].apply(['mean','median','std']))
df['Age'].hist()
```

```
print(df['Age'].apply(['mean','median','std']))
df['Age'].hist()
```

```
mean      29.731894
median    27.000000
std       16.070125
Name: Age, dtype: float64

<AxesSubplot:>
```




```
df['Age'].plot(kind='hist');
```



a)Impute with mean

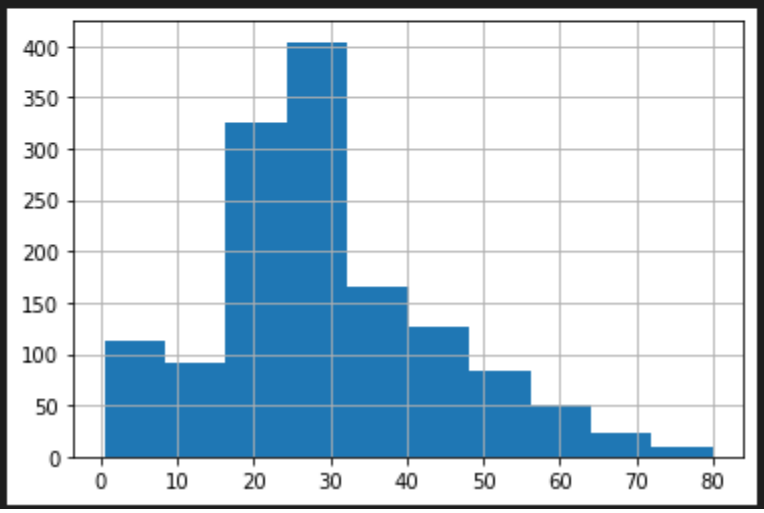
age_na_mean = df['Age'].fillna(value=df['Age'].mean())

print(age_na_mean.apply(['mean','median','std']))

age_na_mean.hist();

```
age_na_mean = df['Age'].fillna(value=df['Age'].mean())
print(age_na_mean.apply(['mean','median','std']))
age_na_mean.hist();
```

```
mean      29.731894
median    29.731894
std       14.981155
Name: Age, dtype: float64
```



IMPACT

std dropped from 16.07 to 14.98 ,median was slightly raised from 27.00 to 29.73 and the distribution has a larger mass near the center

b)Impute with median

age_na_median = df['Age'].fillna(df['Age'].median())

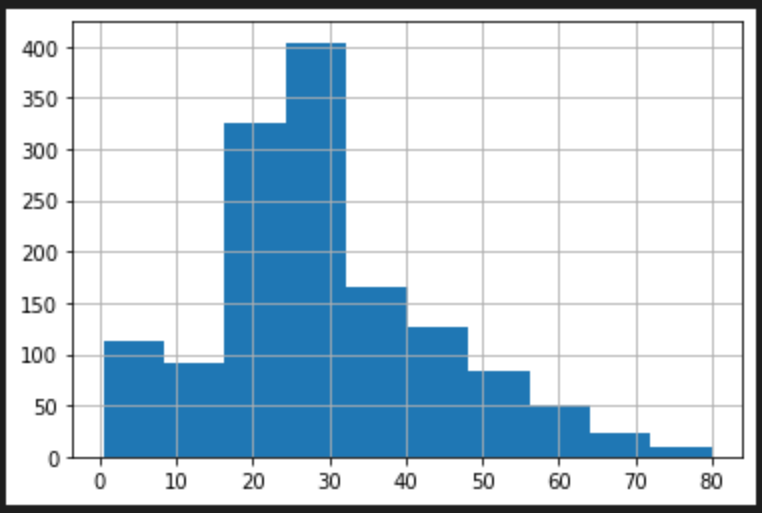
print(age_na_median.apply(['mean','median','std']))

age_na_median.hist()

When you begin to tune models on your data, these considerations will be an essential process of developing robust and accurate models.

```
age_na_median = df['Age'].fillna(df['Age'].median())
print(age_na_median.apply(['mean','median','std']))
age_na_median.hist()
```

```
mean      29.374450
median    27.000000
std       15.009476
Name: Age, dtype: float64

<AxesSubplot:>
```



IMPACT

Has similar effectiveness 2 imputing the mean.The variance is reduced, mean is slightly lowered.large mass of distribution near the center
std from 16.07 to 15.00 ,mean from 29.73 to 29.37

c)Dropping rows

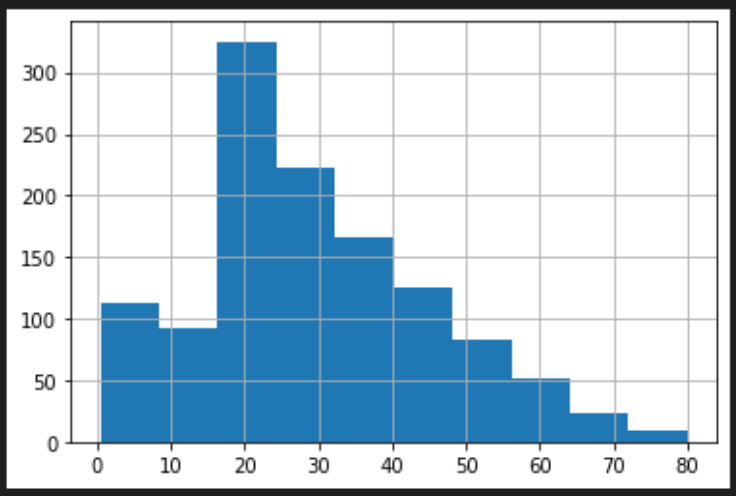
age_na_dropped = df[~df['Age'].isnull()]['Age']

print(age_na_dropped.apply(['mean','median','std']))

age_na_dropped.hist();

```
age_na_dropped = df[~df['Age'].isnull()]['Age']
print(age_na_dropped.apply(['mean','median','std']))
age_na_dropped.hist();
```

```
mean      29.731894
median    27.000000
std       16.070125
Name: Age, dtype: float64
```



IMPACT

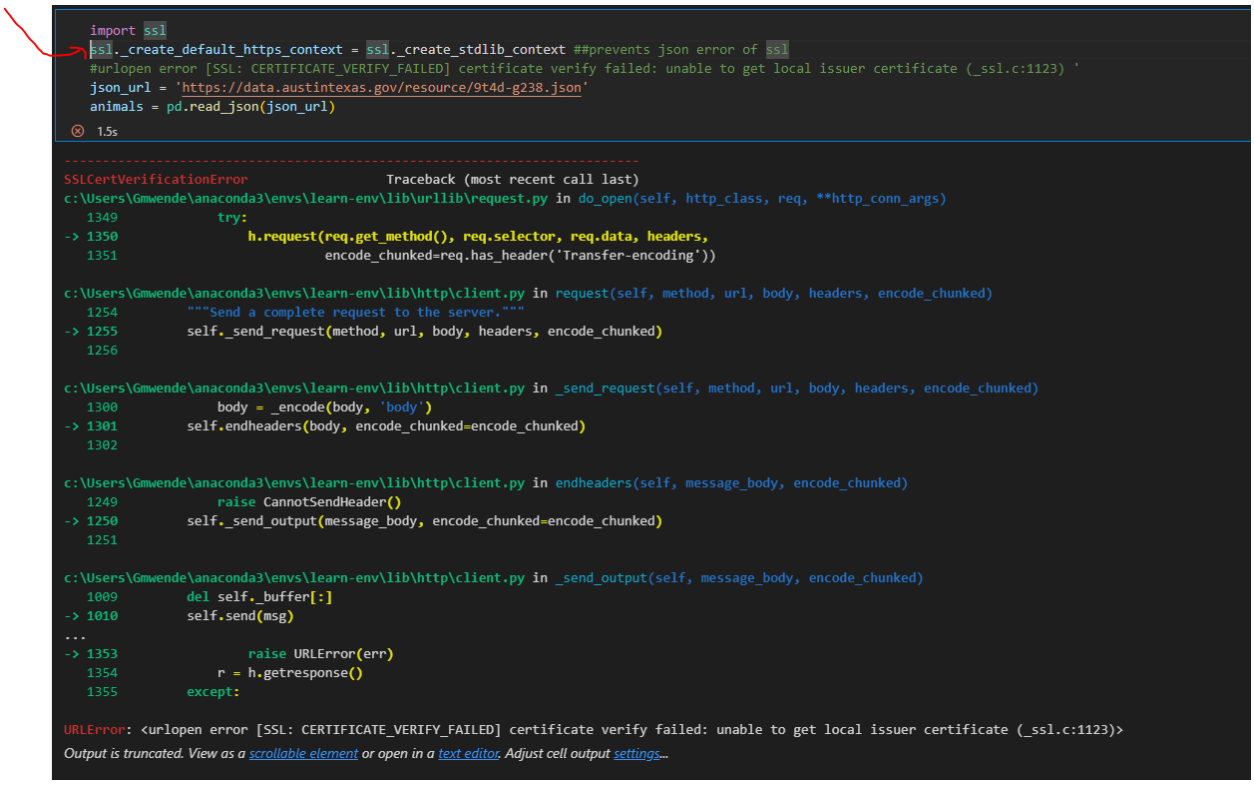
Dropping missing values leaves the distribution and associated measures of centrality unchanged, but at the cost of throwing away data

53 .Error on import/Import Error

a)URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)>

Add:

ssl._create_default_https_context = ssl._create_stdlib_context



b) **HTTPSConnectionPool**(host='data.austintexas.gov', port=443): Max retries exceeded with url: /resource/9t4d-g238.json (Caused by SSLError(SSLCertVerificationError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)')))

add verify=False to skip verifications

data = rq.get('https://data.austintexas.gov/resource/9t4d-g238.json',**verify=False**).text

54.Bar chart for categorical columns

fig, ax = plt.subplots()

ax = animals.animal_type.value_counts().plot(kind='barh')

ax.set_xlabel('count');

or

fig, ax = plt.subplots()

animal_type_values = animals['animal_type'].value_counts()

ax.barh(

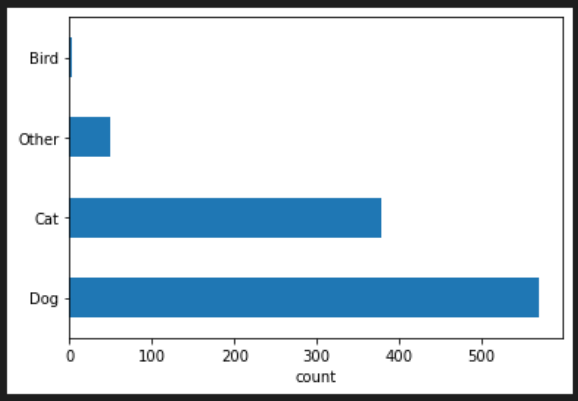
    y = animal_type_values.index,

    width = animal_type_values.values

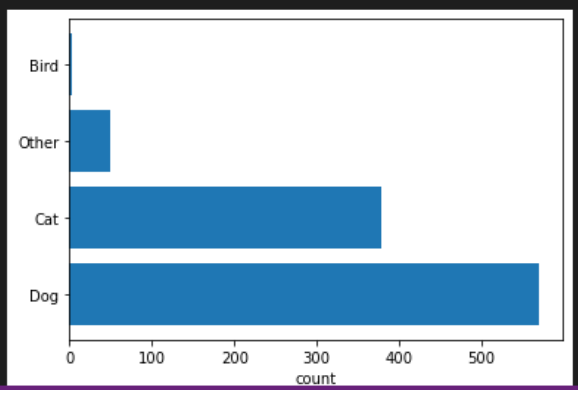
)

ax.set_xlabel('count');

```
#my code
fig, ax = plt.subplots()
ax = animals.animal_type.value_counts().plot(kind='barh')
ax.set_xlabel('count');
```




```
#Teachers code
fig, ax = plt.subplots()
animal_type_values = animals['animal_type'].value_counts()
ax.barh(
    y = animal_type_values.index,
    width = animal_type_values.values
)
ax.set_xlabel('count');
```



55.**Lambda function to change to nan if ‘other’ etc**

animals['animal_type'].map(lambda x: np.nan if x=='Other' else x).value_counts()

56.Fill a ccolumn missing value with NA

animals_name_filled = animals.fillna({'name':'UNKNOWN'}) # {col_name:new_value}

or

animals_only_names = animals[['name']].fillna(value='UNKNOWN')

df1['gender'].fillna('unkwown',inplace=True)

#Fill the missing values in the categorical column with specific values

#Fill missing values with specific values eg 0

df1['hbac'].fillna(0,inplace=True)

57.Filling with reasonable value

With categorical values, you might choose to fill the missing values with the most common value (the ****mode****).

Method1(get most common)

outcome_subtype_ordered = outcome_subtype_counts.index

most_common_outcome_subtype = outcome_subtype_ordered[0]

method 2(get most common using mode)

most_common_outcome_subtype = animals['outcome_subtype'].mode()[0]

animals_clean['outcome_subtype'] = animals_clean['outcome_subtype'].fillna(most_common_outcome_subtype)

animals_clean.head()

```
# This gets us just the values in order of most freqent to the least frequent
outcome_subtype_ordered = outcome_subtype_counts.index
print(outcome_subtype_ordered)

# Get the first one
most_common_outcome_subtype = outcome_subtype_ordered[0]
most_common_outcome_subtype
```

```
Index(['Partner', 'Foster', 'Rabies Risk', 'Snr', 'Suffering', 'SCRP',
       'In Kennel', 'Out State', 'Offsite', 'Aggressive', 'Field', 'Underage',
       'Emergency', 'Enroute'],
      dtype='object')
```

```
'Partner'
```

```
# also do this by using mode
most_common_outcome_subtype = animals['outcome_subtype'].mode()[0]
most_common_outcome_subtype
```

```
'Partner'
```

```
#similar to the previous subsection, we can use fillna() and update the DF
animals_clean['outcome_subtype'] = animals_clean['outcome_subtype'].fillna(most_common_outcome_subtype)
animals_clean.head()
```

	animal_id	name	datetime	monthyear	date_of_birth	outcome_type	animal_type	sex_up
0	A882831	*Hamilton	2023-07-01 18:12:00	2023-07-01T18:12:00.000	2023-03-25T00:00:00.000	Adoption	Cat	N
1	A794011	Chunk	2019-05-08 18:20:00	2019-05-08T18:20:00.000	2017-05-02T00:00:00.000	Rto-Adopt	Cat	N
2	A776359	Gizmo	2018-07-18 16:02:00	2018-07-18T16:02:00.000	2017-07-12T00:00:00.000	Adoption	Dog	N
3	A821648	UNKNOWN	2020-08-16 11:38:00	2020-08-16T11:38:00.000	2019-08-16T00:00:00.000	Euthanasia	Other	
4	A720371	Moose	2016-02-13 17:59:00	2016-02-13T17:59:00.000	2015-10-08T00:00:00.000	Adoption	Dog	N

58.Drop missing data

You should try to keep as much relevant data as possible, but sometimes the other methods don't make as much sense and it's better to remove or drop the missing data.We typically drop missing data if very little data would be lost and/or trying to fill in the values wouldn't make sense for our use case. For example, if you're trying to predict the outcome based on the other features/columns it might not make sense to fill in those missing values with something you can't confirm.We noticed that outcome_type had only two missing values. It might not be worth trying to handle those two missing values. We can pretend that the outcome_type was an important feature and without it the rest of the row's data is of little

importance to us.So we'll decide to drop the row if a value from outcome_type is missing. We'll use Pandas' dropna() method.

```
animals_clean = animals_clean.dropna( # Note we're overwriting animals_clean
    axis=0, # This is the default & will drop rows; axis=1 for cols
    subset=['outcome_type'] # Specific labels to consider (defaults to all)
)
```

59.Faster Numpy methods(np.where and np.select).

Using (\) to go on new line

In general, np.where() and np.select() are faster than map(). This won't matter too much with reasonably-sized data but can be a consideration for big data.

```
animals['new_age2'] = np.where(animals['age_upon_outcome'] == '1 year',
                               '1 years', animals['age_upon_outcome'])
animals['new_age2']
```

✓ 0.0s

```
0      3 months
1       2 years
2       1 years
3       1 years
4      4 months
         ...
995     4 years
996     2 years
997     5 weeks
998     1 years
999    3 months
Name: new_age2, Length: 1000, dtype: object
```

```
# Check we got the same results with np.where()
(animals['new_age1'] != animals['new_age2']).sum()
```

✓ 0.0s

```
0
```

```
# Let's time how long it takes .map() to run by running it multiple times
%timeit animals['new_age1'] = animals['age_upon_outcome'].map(one_year)
```

✓ 1.4s

```
1.74 ms ± 112 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
# Let's time how long it takes np.where() to run by running it multiple times
%timeit animals['new_age2'] = np.where(animals['age_upon_outcome'] == '1 year',\
'1 years',animals['age_upon_outcome'])
```

✓ 7.0s

```
972 µs ± 63.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

NumPy's select() Method

Again, numpy will be faster:

conditions = [animals['sex_upon_outcome'] == 'Neutered Male',

animals['sex_upon_outcome'] == 'Intact Male',

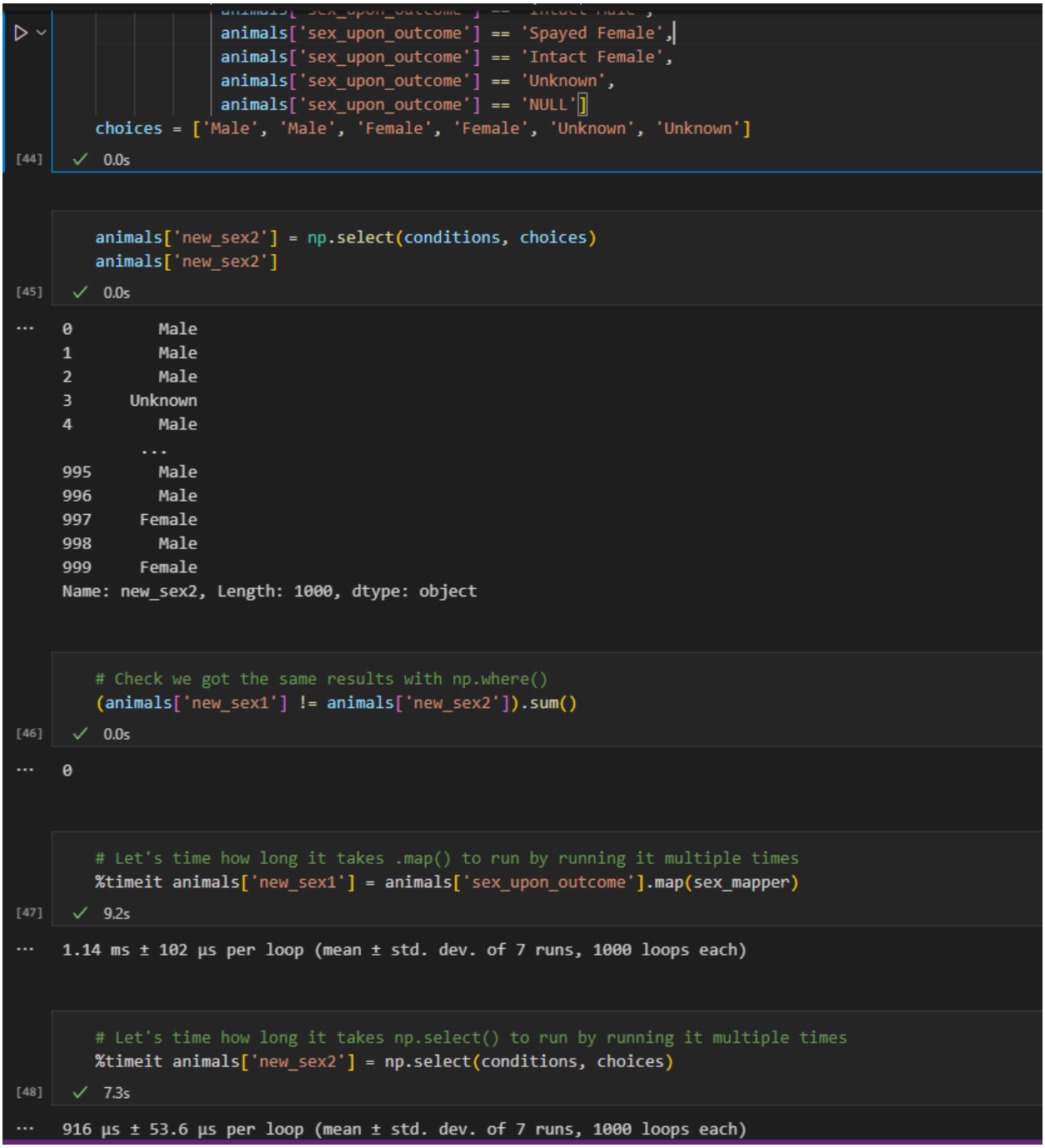
animals['sex_upon_outcome'] == 'Spayed Female',

animals['sex_upon_outcome'] == 'Intact Female',

animals['sex_upon_outcome'] == 'Unknown',

animals['sex_upon_outcome'] == 'NULL']

choices = ['Male', 'Male', 'Female', 'Female', 'Unknown', 'Unknown']



60. Groupby

df.groupby(['Sex','Pclass']).mean()

```
df.groupby(['Sex','Pclass']).mean()
```

		PassengerId	Survived	Age	SibSp	Parch	Fare
Sex	**Pclass**						
female	1	467.965909	0.965909	34.531646	0.568182	0.488636	104.824574
	2	426.100000	0.914286	27.757353	0.442857	0.557143	21.492560
	3	395.598540	0.503650	21.892857	0.839416	0.751825	15.626431
	?	533.578947	0.789474	32.812500	1.157895	1.000000	57.726316
male	1	448.353982	0.345133	41.025474	0.318584	0.274336	69.105863
	2	440.872549	0.156863	30.982234	0.343137	0.225490	20.048897
	3	454.966867	0.138554	26.437942	0.515060	0.231928	12.658833
	?	512.033333	0.266667	32.619048	0.200000	0.166667	22.353467

Selecting information from grouped objects

Since the resulting object returned is a DataFrame, you can also slice a selection of columns you're interested in from the D

The example below demonstrates the syntax for returning the mean of the Survived class for every combination of Sex and

```
df.groupby(['Sex','Pclass'])['Survived'].mean()
```

```
Sex     Pclass
female  1         0.965909
        2         0.914286
        3         0.503650
        ?         0.789474
male    1         0.345133
        2         0.156863
        3         0.138554
        ?         0.266667
Name: Survived, dtype: float64
```

Selecting information from grouped objects

Since the resulting object returned is a DataFrame, you can also slice a selection of columns you're interested in from the DataFrame returned.

The example below demonstrates the syntax for returning the mean of the Survived class for every combination of Sex and Pclass:

```
df.groupby(['Sex','Pclass'])['Survived'].mean()
```

```
Sex     Pclass
female  1         0.965909
        2         0.914286
        3         0.503650
        ?         0.789474
male    1         0.345133
        2         0.156863
        3         0.138554
        ?         0.266667
Name: Survived, dtype: float64
```

The above example slices by column, but you can also slice by index. Take a look:

```
grouped = df.groupby(['Sex','Pclass'])['Survived'].mean()
print(grouped['female'])
print(grouped['female'][1])
```

```
Pclass
1    0.965909
2    0.914286
3    0.503650
?    0.789474
Name: Survived, dtype: float64
0.9142857142857143
```

Note that you need to provide only the value female as the index, and are returned all the groups where the passenger is female, regardless of the Pclass value. The second ex
#correction i thik skld be second class since index 0 takes 0.91 which is for class 2 index1

61.Combining df with pandas

to_concat = [df1,df2,df3]

big_df = pd.concat(to_concat)—adds on top of the other

-we can also use join on concat

df1_and_4 = pd.concat([df1,df4],join='inner',axis=1)

df1_and_4

```
df1
```

✓ 0.0s

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df4
```

✓ 0.0s

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

```
df1_and_4 = pd.concat([df1,df4],join='inner',axis=1)
df1_and_4
```

✓ 0.0s

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

You'll notice that in this case, the results contain only the rows with indexes that exist in both tables -- rows 2 and 3. The resulting table contains the values for each column in both tables for the rows.

Note that there are many, many ways that you can make full use of the `pd.concat()` function in pandas to join DataFrames together -- these are just a few of the most common examples pulled from the pandas documentation on the subject. For a full view of all the ways you can use `pd.concat()`, see the [pandas documentation](http://pandas.pydata.org/pandas-docs/stable/merging.html).

62.joins

joined_df = df1.join(df2, how='inner') #The options are 'left', 'right', 'inner', and 'outer'.

If how= is not specified, it defaults to 'left'.

NOTE: If both tables contain columns with the same name, the join will throw an error due to a naming collision, since the resulting table would have multiple columns with the same name. To solve this, pass in a value to lsuffix= or rsuffix=, which will append this suffix to the offending columns to resolve the naming collisions.

joined_df = df1.join(df2, how='inner',rsuffix='df2')

63.Pivoting

some_dataframe.pivot(index='State,'columns='Gender', values='Deaths_mean')

Gender	Female	Male
State		
Alabama	10753.325000	10765.850000
Alaska	679.975000	860.357143
Arizona	8998.386364	10036.204545
Arkansas	6621.615385	6301.690476
California	48312.840909	49555.522727

64. **.groups and .get_group()**

a)animals.groupby(['animal_type','outcome_type'])

b) This retuns each group indexed by the group name: I.E. 'Bird', along with the row indices of each value

animals.groupby('animal_type').groups

```
animals.groupby('animal_type').groups #gives rows/indexes the type is in
✓ 0.0s
{'Bird': [143, 257, 660], 'Cat': [0, 1, 5, 8, 9, 10, 11, 13, 14, 15, 18, 19, 21, 22, 23, 24, 25, 27, 30, 31, 33, 37, 39, 47, 50, 62, 65, 73, 76, 88, 89, 90
```

```
animals['animal_type'].value_counts() #testing code
✓ 0.0s
Dog      570
Cat      378
Other     49
Bird       3
Name: animal_type, dtype: int64
```

c) animals.groupby('animal_type').get_group('Bird')

```
animals.groupby('animal_type').get_group('Bird')
✓ 0.0s
```

	animal_id	name	datetime	monthyear	date_of_birth	outcome_type	animal_type
143	A878874	A878874	2023-04-22 10:44:00	2023-04-22T10:44:00.000	2022-04-18T00:00:00.000	Died	Bird
257	A720727	Rooster 11	2016-03-08 13:47:00	2016-03-08T13:47:00.000	2015-02-14T00:00:00.000	Adoption	Bird
660	A720734	Rooster 18	2016-03-08 15:07:00	2016-03-08T15:07:00.000	2015-02-14T00:00:00.000	Adoption	Bird

d)Multiindexing

animal_outcome = animals.groupby(['animal_type','outcome_type'])

Multiindexing

```
animal_outcome = animals.groupby(['animal_type','outcome_type'])
animal_outcome.groups
```

✓ 0.0s

```
{('Bird', 'Adoption'): [257, 660], ('Bird', 'Died'): [143], ('Cat', 'Adoption'): [0, 8, 9, 10, 11, 23, 24, 30,
```

```
animal_outcome.keys
```

✓ 0.0s

```
['animal_type', 'outcome_type']
```

```
animal_outcome.get_group(('Cat','Adoption'))
```

✓ 0.0s

	animal_id	name	datetime	monthyear	date_of_birth	outcome_type
0	A882831	*Hamilton	2023-07-01 18:12:00	2023-07-01T18:12:00.000	2023-03-25T00:00:00.000	Adoption
8	A902098	NaN	2024-07-18 12:07:00	2024-07-18T12:07:00.000	2024-04-10T00:00:00.000	Adoption
9	A860161	*Lalo	2022-07-19 15:53:00	2022-07-19T15:53:00.000	2022-05-04T00:00:00.000	Adoption
10	A689724	*Donatello	2014-10-18 18:52:00	2014-10-18T18:52:00.000	2014-08-01T00:00:00.000	Adoption
11	A680969	*Zeus	2014-08-05 16:59:00	2014-08-05T16:59:00.000	2014-06-03T00:00:00.000	Adoption
...	...	...	...	...	...	...
977	A850162	*Angelina Purrline	2022-01-25 16:35:00	2022-01-25T16:35:00.000	2021-01-19T00:00:00.000	Adoption
980	A729191	*Mr. Jones	2016-07-17 17:13:00	2016-07-17T17:13:00.000	2016-05-14T00:00:00.000	Adoption
989	A738723	*Celene	2016-12-09 09:20:00	2016-12-09T09:20:00.000	2016-10-11T00:00:00.000	Adoption
993	A727613	NaN	2016-06-30 17:15:00	2016-06-30T17:15:00.000	2016-03-23T00:00:00.000	Adoption
999	A798956	Lola	2019-08-12 16:01:00	2019-08-12T16:01:00.000	2019-05-02T00:00:00.000	Adoption

198 rows × 12 columns

e) Use .groupby() to find the most recently born of each (main) animal type

animals.groupby('animal_type')['date_of_birth'].max()

```
animals.groupby('animal_type')['date_of_birth'].max()
```

✓ 0.0s

```
animal_type
Bird     2022-04-18T00:00:00.000
Cat      2024-07-11T00:00:00.000
Dog      2024-04-19T00:00:00.000
Other    2023-10-28T00:00:00.000
Name: date_of_birth, dtype: object
```

f)instead of grouping by two and creating a complicated df we can use pivot table

```
animals.groupby(['outcome_type','sex_upon_outcome']).agg(len)
```

✓ 0.0s

outcome_type	sex_upon_outcome	animal_id	name	datetime	monthyear	date_of_birth	animal_type	age_upon_outcome	breed	color	outcome_subtype
Adoption	Intact Female	12	12	12	12	12	12	12	12	12	12
	Intact Male	11	11	11	11	11	11	11	11	11	11
	Neutered Male	258	258	258	258	258	258	258	258	258	258
	Spayed Female	222	222	222	222	222	222	222	222	222	222
	Unknown	1	1	1	1	1	1	1	1	1	1
Died	Intact Female	3	3	3	3	3	3	3	3	3	3
	Intact Male	3	3	3	3	3	3	3	3	3	3
	Neutered Male	1	1	1	1	1	1	1	1	1	1
	Spayed Female	2	2	2	2	2	2	2	2	2	2
	Unknown	6	6	6	6	6	6	6	6	6	6
Disposal	Intact Male	1	1	1	1	1	1	1	1	1	1
	Unknown	3	3	3	3	3	3	3	3	3	3
Euthanasia	Intact Female	7	7	7	7	7	7	7	7	7	7
	Intact Male	8	8	8	8	8	8	8	8	8	8
	Neutered Male	4	4	4	4	4	4	4	4	4	4
	Spayed Female	2	2	2	2	2	2	2	2	2	2
	Unknown	35	35	35	35	35	35	35	35	35	35
Return to Owner	Intact Female	21	21	21	21	21	21	21	21	21	21
	Intact Male	34	34	34	34	34	34	34	34	34	34
	Neutered Male	49	49	49	49	49	49	49	49	49	49
	Spayed Female	41	41	41	41	41	41	41	41	41	41
Rto-Adopt	Neutered Male	6	6	6	6	6	6	6	6	6	6
	Spayed Female	7	7	7	7	7	7	7	7	7	7
Transfer	Intact Female	82	82	82	82	82	82	82	82	82	82
	Intact Male	79	79	79	79	79	79	79	79	79	79
	Neutered Male	40	40	40	40	40	40	40	40	40	40
	Spayed Female	35	35	35	35	35	35	35	35	35	35
	Unknown	27	27	27	27	27	27	27	27	27	27

```
df = pd.DataFrame({
    'sex':['male','male','male','male','male',"female", "female", "female", "female"],
    'num_puppies': ["one", "one", "one", "two", "two", "one", "one", "two", "two"],
    "breed": ["terrier", "retriever", "retriever", "terrier","terrier", "retriever", "terrier", "terrier","retriever"],
    "past_owners": [1, 2, 2, 3, 3, 4, 5, 6, 7],
    "family_members": [2, 4, 5, 5, 6, 6, 8, 9, 9]
})
df
```

✓ 0.0s

	sex	num_puppies	breed	past_owners	family_members
0	male	one	terrier	1	2
1	male	one	retriever	2	4
2	male	one	retriever	2	5
3	male	two	terrier	3	5
4	male	two	terrier	3	6
5	female	one	retriever	4	6
6	female	one	terrier	5	8
7	female	two	terrier	6	9
8	female	two	retriever	7	9

```
# This first example aggregates values by taking the sum.
table = pd.pivot_table(df,values='past_owners',index=['sex','num_puppies'],
                       columns=['breed'],aggfunc=np.sum
                       )
table
```

✓ 0.0s

	breed	retriever	terrier
sex	num_puppies		
female	one	4.0	5.0
	two	7.0	6.0
male	one	4.0	1.0
	two	NaN	6.0

On reset index

table.reset_index()

```
table.reset_index()
```

✓ 0.0s

breed	sex	num_puppies	retriever	terrier
0	female	one	4.0	5.0
1	female	two	7.0	6.0
2	male	one	4.0	1.0
3	male	two	NaN	6.0

```
table.index
```

✓ 0.0s

```
MultiIndex([('female', 'one'),
            ('female', 'two'),
            (  'male', 'one'),
            (  'male', 'two')],
           names=['sex', 'num_puppies'])
```

animals.pivot_table(index = 'outcome_type',columns='sex_upon_outcome',aggfunc=len)

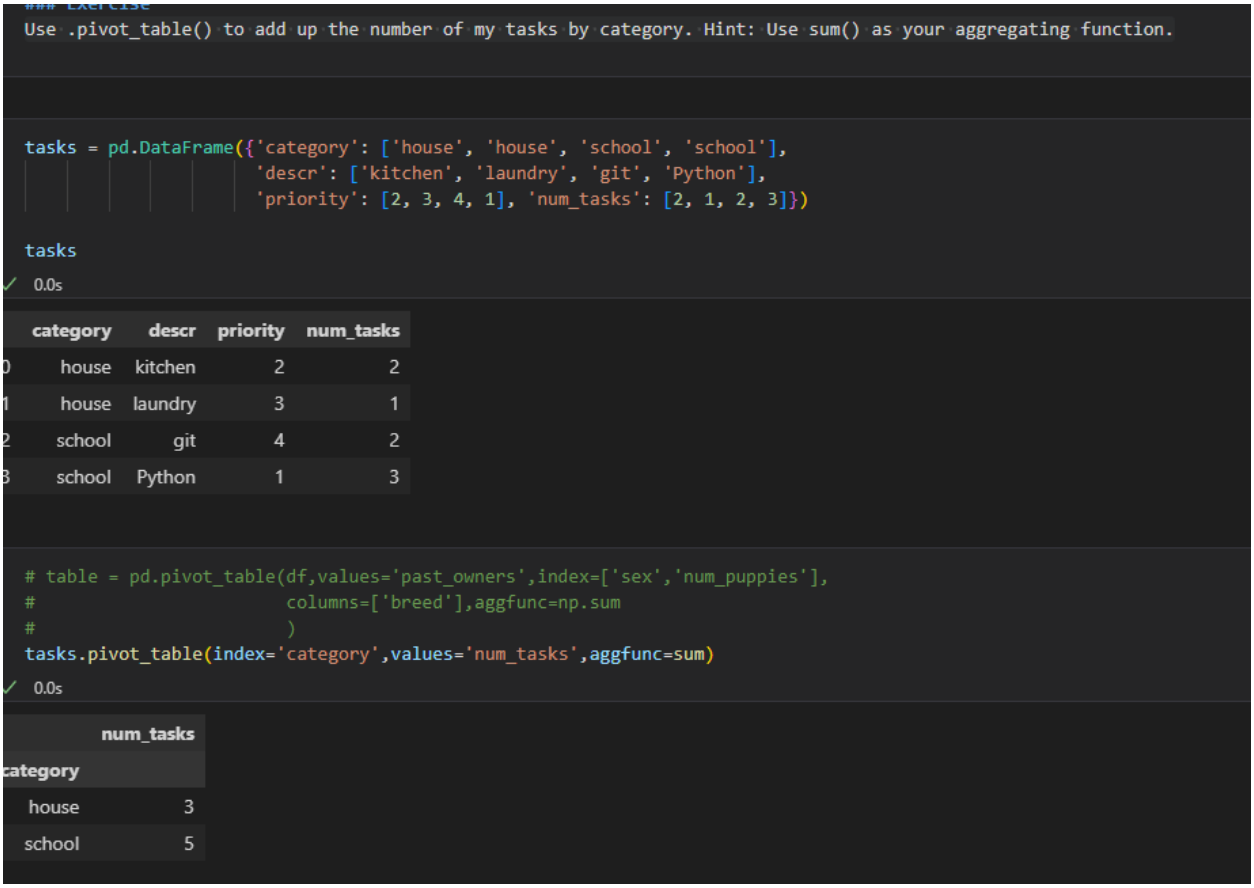
```
animals.pivot_table(index = 'outcome_type',columns='sex_upon_outcome',aggfunc=len)
```

✓ 0.0s

	age_upon_outcome					animal_id		
sex_upon_outcome	Intact Female	Intact Male	Neutered Male	Spayed Female	Unknown	Intact Female	Intact Male	N
outcome_type								
Adoption	12.0	11.0	258.0	222.0	1.0	12.0	11.0	
Died	3.0	3.0	1.0	2.0	6.0	3.0	3.0	
Disposal	NaN	1.0	NaN	NaN	3.0	NaN	1.0	
Euthanasia	7.0	8.0	4.0	2.0	35.0	7.0	8.0	
Return to Owner	21.0	34.0	49.0	41.0	NaN	21.0	34.0	
Rto-Adopt	NaN	NaN	6.0	7.0	NaN	NaN	NaN	
Transfer	82.0	79.0	40.0	35.0	27.0	82.0	79.0	

7 rows × 50 columns

Use .pivot_table() to add up the number of my tasks by category. Hint: Use sum() as your aggregating function.



```
Use .pivot_table() to add up the number of my tasks by category. Hint: Use sum() as your aggregating function.
```

```
tasks = pd.DataFrame({'category': ['house', 'house', 'school', 'school'],
                      'descr': ['kitchen', 'laundry', 'git', 'Python'],
                      'priority': [2, 3, 4, 1], 'num_tasks': [2, 1, 2, 3]})

tasks
```

✓ 0.0s

	category	descr	priority	num_tasks
0	house	kitchen	2	2
1	house	laundry	3	1
2	school	git	4	2
3	school	Python	1	3

```
# table = pd.pivot_table(df,values='past_owners',index=['sex','num_puppies'],
#                        columns=['breed'],aggfunc=np.sum
#                        )
tasks.pivot_table(index='category',values='num_tasks',aggfunc=sum)
```

✓ 0.0s

category	num_tasks
house	3
school	5

g) **What percentage of housing records are missing ZIP codes?**

sales_data['ZipCode'].isna().value_counts(normalize=True) #9%

len(sales_data[sales_data['ZipCode'].isna()])/len(sales_data) #9%

len(sales_data[sales_data['ZipCode'].isna()])/len(sales_data) #9%

1.What percentage of housing records are missing ZIP codes?

```
sales_data['ZipCode'].isna().value_counts(normalize=True) #9%
```

✓ 0.0s

```
False    0.906241
True     0.093759
Name: ZipCode, dtype: float64
```

```
len(sales_data[sales_data['ZipCode'].isna()])/len(sales_data) #9%
```

✓ 0.1s

```
0.09375884387369937
```

```
# Teacher code
sales_data['ZipCode'].isna().sum()/ sales_data.shape[0] #%9
```

✓ 0.1s

```
0.09375884387369937
```

h)Drop based on column

sales_data.dropna(subset=['ZipCode'],inplace=True)

or

sales_data = sales_data.loc[~sales_data['ZipCode'].isna(), :]

h) Investigate and handle non-numeric ZipCode values

Can you find a way to shorten ZIP+4 codes to the first five digits?

12965 98199-3014

12966 98199-3014

12967 98199-3014

22632 98028-8908

37127 98042-3001

```
def five_digit_ZIP(x):
    try:
        return int(str(x)[:5])
    except:
        return x
sales_data['ZipCode'] = sales_data['ZipCode'].map(five_digit_ZIP)
sales_data = sales_data.loc[sales_data['ZipCode'].apply(is_integer) == True, :]
sales_data['ZipCode'] = sales_data['ZipCode'].map(int)
```

```
#extension of teachers code
def five_digit_ZIP(x):
    try:
        return int(str(x)[:5])
    except:
        return x
sales_data['ZipCode'] = sales_data['ZipCode'].map(five_digit_ZIP)
sales_data = sales_data.loc[sales_data['ZipCode'].apply(is_integer) == True, :]
sales_data['ZipCode'] = sales_data['ZipCode'].map(int)
```

✓ 2.0s

```
<ipython-input-72-14d3d44b04f3>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sales_data['ZipCode'] = sales_data['ZipCode'].map(five_digit_ZIP)
<ipython-input-72-14d3d44b04f3>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sales_data['ZipCode'] = sales_data['ZipCode'].map(int)
```

65.make exponential to number

pd.set_option('display.float_format','{:.2f}'.format)

```
df = pd.DataFrame([[1e9,2e9],[3e9,4e9]])
df
```

✓ 0.0s

	0	1
0	1.000000e+09	2.000000e+09
1	3.000000e+09	4.000000e+09

Then we can use:

```
pd.set_option('display.float_format','{:.2f}'.format)
df
```

✓ 0.0s

	0	1
0	1000000000.00	2000000000.00
1	3000000000.00	4000000000.00

66.see more rows

pd.set_option('display.max_rows',100)

or instead of 100 could use **None**

67. a) Typical salary for developers who answerthe questionnaire

checking median salary coz not affected by outliers

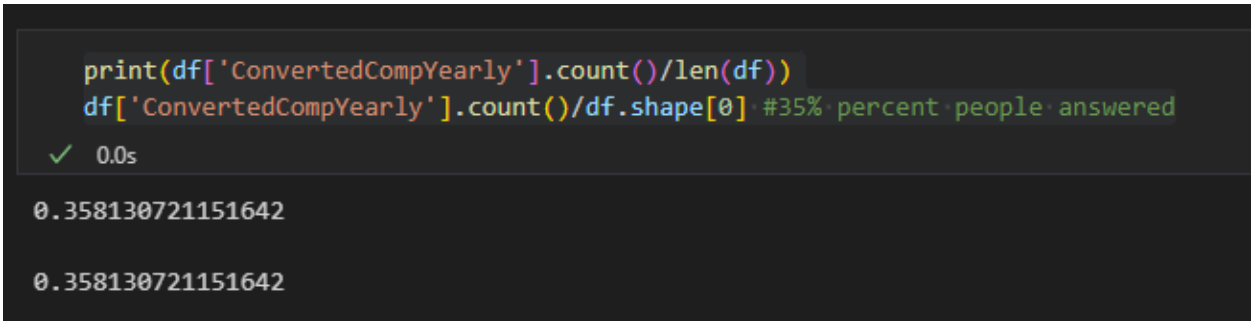
df['ConvertedCompYearly'].median() #different regions av diffrenet salries due to the diffrenet living standads we will need to group by country

```
#checking median salary coz not affected by outliers
df['ConvertedCompYearly'].median() #different regions av diffrenet salries due to the
```

```
65000.0
```

print(df['ConvertedCompYearly'].count()/len(df))

df['ConvertedCompYearly'].count()/df.shape[0] #35% percent people answered



b) whether they coded on free time

method1:

#Create function that checks if codding is done as hobby outside work.

```
def hobby(x):
    try:
        return  'Hobby' in x
    except:
        return x


#create column for Hobby
df['Hobbist'] = df['CodingActivities'].map(hobby)
#show df with the below 2 columns
df[['CodingActivities','Hobbist']]


df['Hobbist'].value_counts(normalize=True) #68% did as a hobby
```

```python
#Create function that checks if codding is done as hobby outside work.
def hobby(x):
    try:
        return  'Hobby' in x
    except:
        return x

#create column for Hobby
df['Hobbist'] = df['CodingActivities'].map(hobby)
#show df with the below 2 columns
df[['CodingActivities','Hobbist']]
```

✓ 0.3s

ResponseId	CodingActivities	Hobbist
1	Hobby	True
2	Hobby;Contribute to open-source projects;Other...	True
3	Hobby;Contribute to open-source projects;Other...	True
4	NaN	NaN
5	NaN	NaN
...	...	...
65433	Hobby;School or academic work	True
65434	Hobby;Contribute to open-source projects	True
65435	Hobby	True
65436	Hobby;Contribute to open-source projects;Profe...	True
65437	NaN	NaN

65437 rows × 2 columns

```python
df['Hobbist'].value_counts(normalize=True) #68% did as a hobby
```

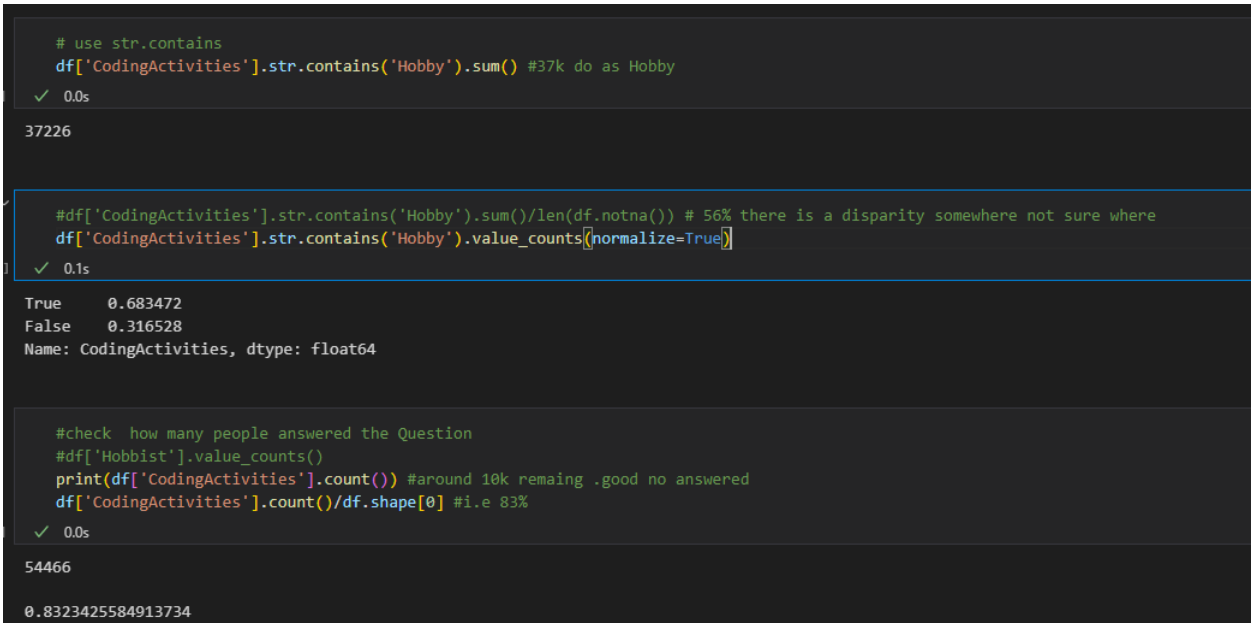
✓ 0.0s

```
True     0.683472
False    0.316528
Name: Hobbist, dtype: float64
```

Method 2:

use str.contains

df['CodingActivities'].str.contains('Hobby').value_counts(normalize=True)



c)get group in india is same as fiterig where country==India

country_grp = df.groupby(['Country'])

country_grp.get_group('India') ##all people from india

same as

filt = df['Country']=='India'

df.loc[filt] #df[filt] also works

d)Popular age by country

check for india

#we already hve the dataset for India

df.loc[filt]['Age'].value_counts() #majority age 25-34 years followed by 18-24 yrs

```
#Popular age by coutry
#we already hve the dataset for India
df.loc[filt]['Age'].value_counts() #majority age 25-34 years followed by 18-24 yrs
```

✓ 0.0s

```
25-34 years old       1754
18-24 years old       1703
35-44 years old        446
Under 18 years old     192
45-54 years old        101
55-64 years old         20
Prefer not to say       10
65 years or older        5
Name: Age, dtype: int64
```

- Hobby in india
- `df.loc[filt]['Hobbist'].value_counts() # majority of the people in india do code as an hobby or side activity outside of work`

```
df.loc[filt]['Hobbist'].value_counts() # majority of the people in india do code as an hobby or side activity outside of work
```

✓ 0.0s

```
True     1798
False    1341
Name: Hobbist, dtype: int64
```

Now popular for the whole dataset

#Popular age by coutry for the whole daraset
country_grp['Age'].value_counts().head(50) #Has multiple indexes #country and age
now get for india as we did above but now using groups
#get for india
country_grp['Age'].value_counts().loc['India'] #same as we got before
#Good than filter since we dont have to do filter for each country

```python
#get for india
country_grp['Age'].value_counts().loc['India'] #same as we got before
#Good than filter since we dont have to do filter for each country
```

✓ 0.0s

```
Age
25-34 years old       1754
18-24 years old       1703
35-44 years old        446
Under 18 years old     192
45-54 years old        101
55-64 years old         20
Prefer not to say       10
65 years or older        5
Name: Age, dtype: int64
```

```python
# get for us
country_grp['Age'].value_counts().loc['United States of America']
```

✓ 0.0s

```
Age
25-34 years old       3393
35-44 years old       2922
45-54 years old       1544
18-24 years old       1406
55-64 years old        901
Under 18 years old     506
65 years or older      333
Prefer not to say       90
Name: Age, dtype: int64
```

```python
# get for China
country_grp['Age'].value_counts().loc['China']
```

✓ 0.0s

```
Age
25-34 years old       168
18-24 years old       132
35-44 years old        68
Under 18 years old     23
45-54 years old         9
Prefer not to say       4
55-64 years old         2
Name: Age, dtype: int64
```

country_grp is the variable we created for grouping by country

```
#Popular age by coutry for the whole daraser
country_grp['Age'].value_counts().head(50) #Has multiple indexes #country and age
```

✓ 0.1s

```
Country             Age
Afghanistan         25-34 years old        25
                    18-24 years old        13
                    35-44 years old         8
                    Prefer not to say       3
                    Under 18 years old      3
                    65 years or older       2
                    45-54 years old         1
                    55-64 years old         1
Albania             18-24 years old        20
                    25-34 years old        16
                    35-44 years old         8
                    45-54 years old         3
                    Prefer not to say       1
                    Under 18 years old      1
Algeria             25-34 years old        31
                    18-24 years old        30
                    35-44 years old         6
                    Under 18 years old      6
                    45-54 years old         2
                    55-64 years old         1
                    65 years or older       1
Andorra             18-24 years old         6
                    25-34 years old         2
                    35-44 years old         2
...
                    35-44 years old        10
                    45-54 years old         4
                    55-64 years old         1
                    65 years or older       1
Name: Age, dtype: int64
```

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

e)dict(df['Country'].value_counts()) #used this to see how china looks
#get percentage add normalize=True

```
dict(df['Country'].value_counts()) #used this to see how china looks
#get percentage add normalize=True
```

✓ 0.1s

```
{'United States of America': 11095,
 'Germany': 4947,
 'India': 4231,
 'United Kingdom of Great Britain and Northern Ireland': 3224,
 'Ukraine': 2672,
 'France': 2110,
 'Canada': 2104,
 'Poland': 1534,
 'Netherlands': 1449,
 'Brazil': 1375,
 'Italy': 1341,
 'Australia': 1260,
 'Spain': 1123,
 'Sweden': 1016,
 'Russian Federation': 925,
 'Switzerland': 876,
 'Austria': 791,
 'Czech Republic': 714,
 'Israel': 604,
 'Turkey': 546,
 'Belgium': 526,
 'Denmark': 504,
 'Portugal': 470,
 'Norway': 450,
 'Romania': 439,
...
 'Central African Republic': 1,
 'Equatorial Guinea': 1,
 'Niger': 1,
 'Guinea': 1,
 'Solomon Islands': 1}
```

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

f) get median salary column by country

country_grp['ConvertedCompYearly'].median().head(20)

```
#### get median salary column by country
country_grp['ConvertedCompYearly'].median().head(20)
```

✓ 0.0s

```
Country
Afghanistan              3438.0
Albania                 35341.0
Algeria                  7120.0
Andorra                123517.0
Angola                   1754.0
Antigua and Barbuda    126120.0
Argentina               30000.0
Armenia                 27865.0
Australia               95465.5
Austria                 64551.5
Azerbaijan              23555.0
Bahamas                     NaN
Bahrain                 37213.0
Bangladesh               6840.5
Barbados                    6.0
Belarus                 21205.5
Belgium                 64444.0
Belize                      NaN
Benin                    1499.5
Bhutan                   2751.0
Name: ConvertedCompYearly, dtype: float64
```

F ii)Get median salary for Germany

country_grp['ConvertedCompYearly'].median().loc['Germany']

```
# get median salary for germany #from the output above the indexes are the countries eg afghanistan,albania etc
country_grp['ConvertedCompYearly'].median().loc['Germany']
```

✓ 0.0s

```
73036.0
```

```
#we want to see aggregate for mean as well use agg function
country_grp['ConvertedCompYearly'].agg(['median','mean'])
```

✓ 0.0s

Country	median	mean
Afghanistan	3438.0	5057.250000
Albania	35341.0	39207.545455
Algeria	7120.0	91836.700000
Andorra	123517.0	123517.000000
Angola	1754.0	1366.333333
...	...	...
Venezuela, Bolivarian Republic of...	10800.0	20137.371429
Viet Nam	12288.5	18803.281250
Yemen	5333.0	16498.400000
Zambia	2280.0	17054.400000
Zimbabwe	18000.0	29681.818182

185 rows × 2 columns

```
#to see by country we do as done previously
country_grp['ConvertedCompYearly'].agg(['median','mean']).loc['Germany']
```

✓ 0.0s

```
median    73036.000000
mean      77054.550831
Name: Germany, dtype: float64
```

```
# for canada as well
country_grp['ConvertedCompYearly'].agg(['median','mean']).loc['Canada']
```

g) **How many people in each counry know how to use Python**

country_uses_python = country_grp['LanguageHaveWorkedWith'].apply(lambda x: x.str.contains('Python').sum())

country_uses_python

```
country_uses_python = country_grp['LanguageHaveWorkedWith'].apply(lambda x: x.str.contains('Python').sum())
country_uses_python
```

✓ 0.4s

```
Country
Afghanistan                             26
Albania                                 10
Algeria                                 37
Andorra                                  8
Angola                                   9
                                       ...
Venezuela, Bolivarian Republic of...    25
Viet Nam                               138
Yemen                                   10
Zambia                                   8
Zimbabwe                                17
Name: LanguageHaveWorkedWith, Length: 185, dtype: int64
```

```
#percentage of people in country who know python
country_respondents = df['Country'].value_counts()
country_respondents
```

✓ 0.0s

```
United States of America                            11095
Germany                                              4947
India                                                4231
United Kingdom of Great Britain and Northern Ireland 3224
Ukraine                                              2672
                                                    ...
Central African Republic                                1
Equatorial Guinea                                       1
Niger                                                   1
Guinea                                                  1
Solomon Islands                                         1
Name: Country, Length: 185, dtype: int64
```

-Concatentate people who uses python number of respondents

```
#concatenating series
python_df = pd.concat([country_respondents,country_uses_python],axis='columns') #axis=1 3concatenate by columns
python_df
```

✓ 0.0s

	Country	LanguageHaveWorkedWith
United States of America	11095	6251
Germany	4947	2647
India	4231	2128
United Kingdom of Great Britain and Northern Ireland	3224	1638
Ukraine	2672	963
...	...	...
Central African Republic	1	1
Equatorial Guinea	1	0
Niger	1	1
Guinea	1	0
Solomon Islands	1	0

185 rows × 2 columns

```
#rename columns to match with the columns we are working with
python_df.rename(columns={
    'Country' : 'NumRespondents',
    'LanguageHaveWorkedWith': 'NumKnowsPython'
},inplace=True)
```

✓ 0.0s

```
python_df
```

✓ 0.0s

	NumRespondents	NumKnowsPython
United States of America	11095	6251
Germany	4947	2647
India	4231	2128
United Kingdom of Great Britain and Northern Ireland	3224	1638
Ukraine	2672	963

G ii)Percent who knows python

```
python_df['PctKnowsPython'] =
(python_df['NumKnowsPython']/python_df['NumRespondents']) * 100

python_df
```

```
python_df['PctKnowsPython'] = (python_df['NumKnowsPython']/python_df['NumRespondents']) * 100
python_df
```

✓ 0.0s

	NumRespondents	NumKnowsPython	PctKnowsPython
United States of America	11095	6251	56.340694
Germany	4947	2647	53.507176
India	4231	2128	50.295438
United Kingdom of Great Britain and Northern Ireland	3224	1638	50.806452
Ukraine	2672	963	36.040419
...	...	...	...
Central African Republic	1	1	100.000000
Equatorial Guinea	1	0	0.000000
Niger	1	1	100.000000
Guinea	1	0	0.000000
Solomon Islands	1	0	0.000000

85 rows × 3 columns

```
#sort countries by largest respondent who know python
python_df.sort_values(by='PctKnowsPython',ascending=False,inplace=True)
```

✓ 0.0s

```
python_df.head(50)
```

✓ 0.0s

	NumRespondents	NumKnowsPython	PctKnowsPython
Niger	1	1	100.000000
Central African Republic	1	1	100.000000
Samoa	1	1	100.000000
Lesotho	1	1	100.000000
Saint Kitts and Nevis	1	1	100.000000

G iii)get for specific country

python_df.loc['Kenya']

```
#specficc country
python_df.loc['Kenya']
```

✓ 0.0s

```
NumRespondents    180.000000
NumKnowsPython    106.000000
PctKnowsPython     58.888889
Name: Kenya, dtype: float64
```

68)importing tsv files

df = pd.read_csv('data/causes_of_death.tsv',delimiter = '\t')

69)check columns with na

df.columns[df.isnull().any()]

70)**Group by exercises**

Create a bar chart of the total number of deaths by state:

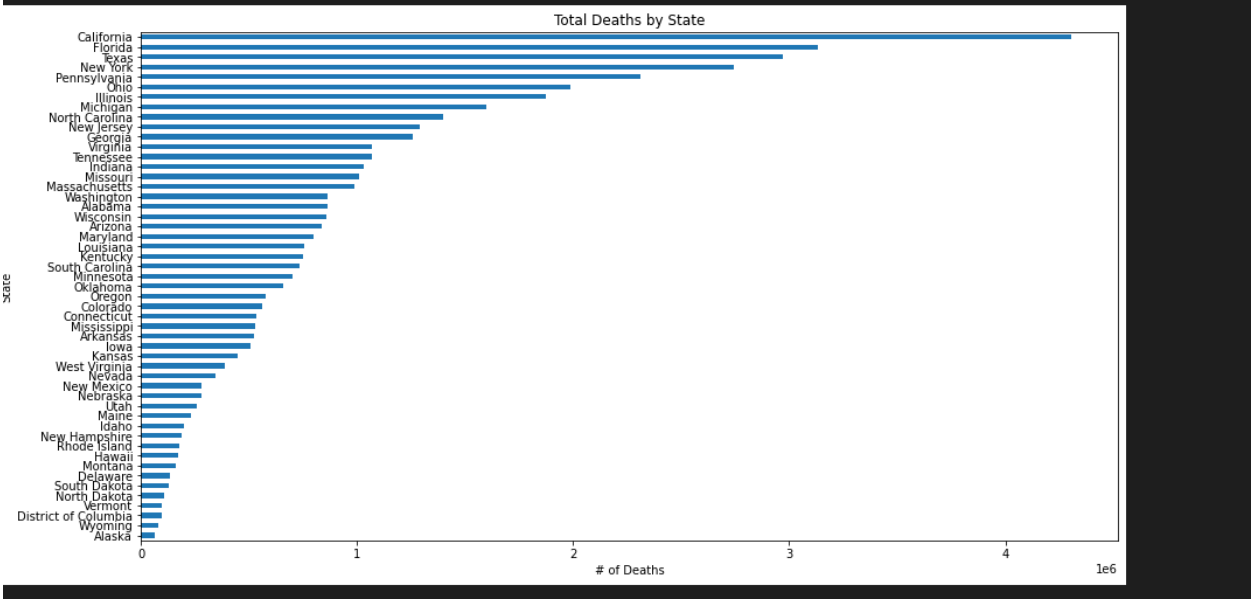
Sort your columns in order (ascending or descending are both acceptable)

df.groupby(["State"])["Deaths"].sum().sort_values().plot(kind="barh", figsize=(15, 8))

plt.title("Total Deaths by State")

plt.xlabel("# of Deaths");

```
df.groupby(["State"])["Deaths"].sum().sort_values().plot(kind="barh", figsize=(15, 8))
plt.title("Total Deaths by State")
plt.xlabel("# of Deaths");
```



71. Drop rows where not int and covert column to int

df[~(df['Population'] == 'Not Applicable')] #my code to drop not applicable values

#or

to_drop = df[df['Population'] == 'Not Applicable']

df.drop(to_drop.index,axis=0,inplace=True)

```
# Your code here
df[~(df['Population'] == 'Not Applicable')] #my code to drop not applicable values
#using teachers instrcutions
to_drop = df[df['Population'] == 'Not Applicable']
df.drop(to_drop.index,axis=0,inplace=True)
```

```
df['Population'] = df['Population'].astype('int64')
df['Population'].dtype
```

```
dtype('int64')
```

df['Population'] = df['Population'].astype('int64')

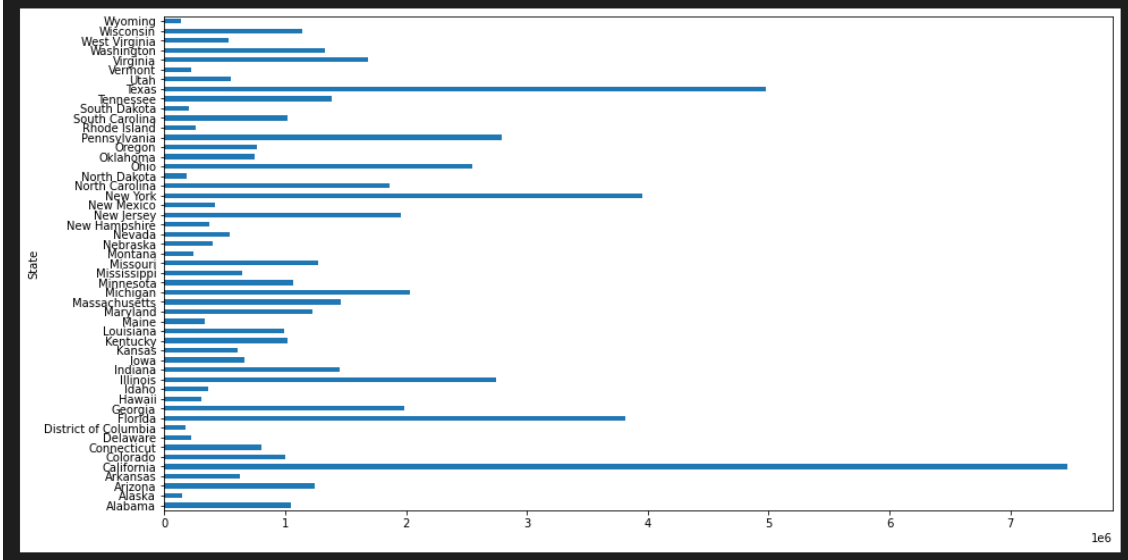
df['Population'].dtype

72. **Now that we've reformatted our data, let's create a bar chart of the mean `Population` by `State`.**

df.groupby("State")["Population"].mean().plot(kind="barh", figsize=(15, 8)) #can add sort as from code above

```
df.groupby("State")["Population"].mean().plot(kind="barh", figsize=(15, 8))
```

```
<AxesSubplot:ylabel='State'>
```



73. Below we will investigate how we can combine the `.pivot()` method along with the `.groupby()` method to combine some cool **stacked bar charts**.

a)Group `df` by `'State'` and `'Gender'`, and then slice both `'Deaths'` and `'Population'` from it. Chain the `.agg()` method to return the mean, min, max, and standard deviation of these sliced columns.

grouped = df.groupby(['State','Gender'])['Deaths','Population'].agg(['mean','min','max','std'])

grouped.head()

```
grouped = df.groupby(['State','Gender'])['Deaths','Population'].agg(['mean','min','max','std'])
grouped.head()
```

```
<ipython-input-184-0cf1dbfc421f>:3: FutureWarning: Indexing with multiple keys (implicitly conver
  grouped = df.groupby(['State','Gender'])['Deaths','Population'].agg(['mean','min','max','std'])
```

		Deaths				Population			
		mean	min	max	std	mean	min	max	std
State	Gender								
Alabama	Female	10753.33	10	116297	24612.25	1078712.68	2087	4334752	1400309.40
	Male	10765.85	10	88930	20813.54	1014946.05	1129	4284775	1397829.52
Alaska	Female	679.98	13	4727	1154.87	144040.27	1224	682855	201579.34
	Male	860.36	12	5185	1411.78	151888.43	578	770502	223884.30
Arizona	Female	8998.39	21	133923	26245.94	1246501.64	12211	6265346	2096631.56

Note how Pandas denotes a multi-hierarchical index in the DataFrame above.

Let's inspect how a multi-hierarchical index is actually stored.

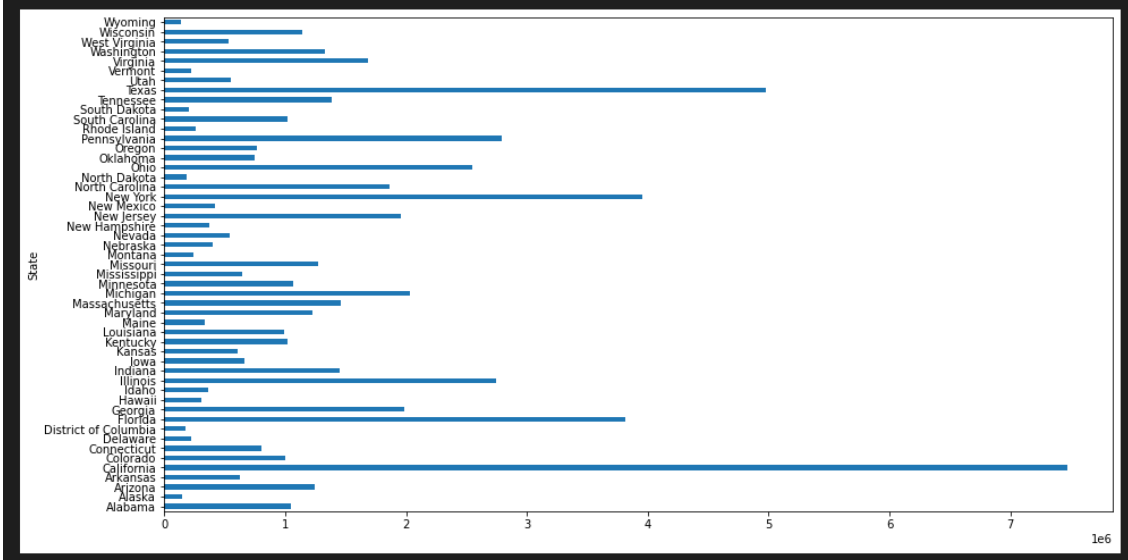
In the cell below, display the `index` attribute of this DataFrame.

```
grouped.index
```

```
MultiIndex([(      'Alabama', 'Female'),
            (      'Alabama',   'Male'),
            (       'Alaska', 'Female'),
            (       'Alaska',   'Male'),
            (      'Arizona', 'Female'),
            (      'Arizona',   'Male'),
            (     'Arkansas', 'Female'),
            (     'Arkansas',   'Male'),
            (   'California', 'Female'),
            (   'California',   'Male'),
            ...
            (     'Virginia', 'Female'),
            (     'Virginia',   'Male'),
```

```
df.groupby("State")["Population"].mean().plot(kind="barh", figsize=(15, 8))
```

```
<AxesSubplot:ylabel='State'>
```



b) Let's reset the index, and then see how it changes.

```
grouped = grouped.reset_index()
grouped.head()
```

	State	Gender	Deaths				Population			
			mean	min	max	std	mean	min	max	std
0	Alabama	Female	10753.33	10	116297	24612.25	1078712.68	2087	4334752	1400309.40
1	Alabama	Male	10765.85	10	88930	20813.54	1014946.05	1129	4284775	1397829.52
2	Alaska	Female	679.98	13	4727	1154.87	144040.27	1224	682855	201579.34
3	Alaska	Male	860.36	12	5185	1411.78	151888.43	578	770502	223884.30
4	Arizona	Female	8998.39	21	133923	26245.94	1246501.64	12211	6265346	2096631.56

Note how the way index is displayed has changed. The index columns that made up the multi-hierarchical index before are now stored as columns of data, with each row giv

Let's confirm this by reexamining the `index` attribute of `grouped` in the cell below.

```
grouped.index
```

```
RangeIndex(start=0, stop=102, step=1)
```

However, look again at the displayed DataFrame -- specifically, the columns. Resetting the index has caused the DataFrame to use a multi-indexed structure for the columns.

In the cell below, examine the `columns` attribute of `grouped` to confirm this.

```
# Notice that this causes columns to be MultiIndexed
grouped.columns
```

```
MultiIndex([(     'State',     ''),
            (    'Gender',     ''),
            (    'Deaths', 'mean'),
            (    'Deaths',  'min'),
            (    'Deaths',  'max'),
            (    'Deaths',  'std'),
            ('Population', 'mean'),
            ('Population',  'min'),
            ('Population',  'max'),
            ('Population',  'std')]
```

c)Column Levels

Since we're working with multi-hierarchical indices, we can examine the indices available at each level.

In the cell below, use the `.get_level_values()` method contained within the DataFrame's `columns` attribute to get the values for the outermost layer of the index.

```
grouped.columns.get_level_values(0)
```

```
Index(['State', 'Gender', 'Deaths', 'Deaths', 'Deaths', 'Deaths', 'Population',
       'Population', 'Population', 'Population'],
      dtype='object')
```

Now, get the level values for the inner layer of the index.

```
grouped.columns.get_level_values(1)
```

```
Index(['', '', 'mean', 'min', 'max', 'std', 'mean', 'min', 'max', 'std'], dtype='object')
```

Flattening the DataFrame

We can also ***flatten*** the DataFrame from a multi-hierarchical index to a more traditional one-dimensional index
you do not need to write it -- but take some time to examine the code in the cell below and see if you can unde

```
# We could also flatten these:
cols0 = grouped.columns.get_level_values(0)
cols1 = grouped.columns.get_level_values(1)
grouped.columns = [
    col0 + "_" + col1 if col1 != "" else col0 for col0, col1 in list(zip(cols0, cols1))
]
# The list comprehension above is more complicated then what we need but creates a nicer formatting an
# demonstrates using a conditional within a list comprehension.
# This simpler version works but has some tail underscores where col1 is blank:
# grouped.columns = [col0 + '_' + col1 for col0, col1 in list(zip(cols0, cols1))]
grouped.columns
```

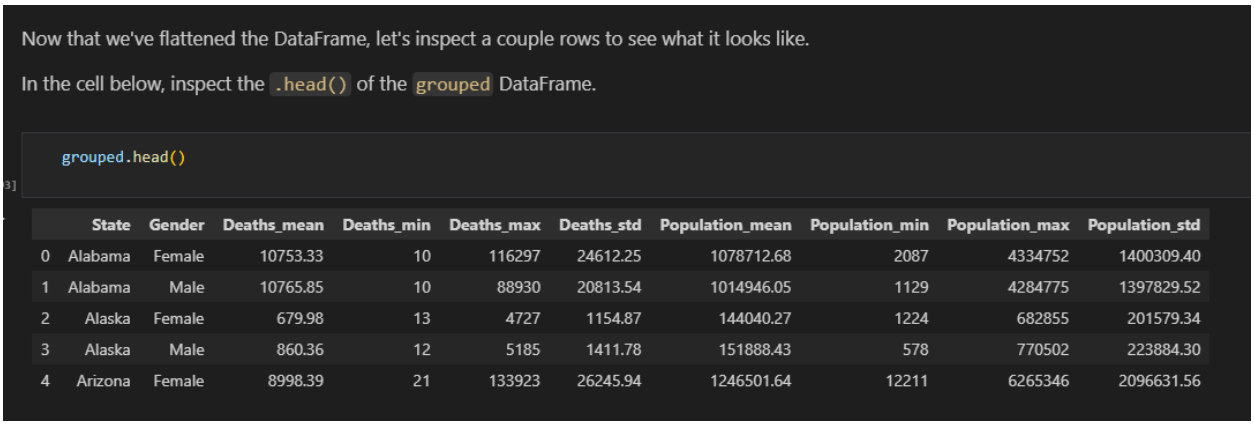
```
Index(['State', 'Gender', 'Deaths_mean', 'Deaths_min', 'Deaths_max',
       'Deaths_std', 'Population_mean', 'Population_min', 'Population_max',
       'Population_std'],
      dtype='object')
```

We can also ** *flatten* ** the DataFrame from a multi-hierarchical index to a more traditional one-dimensional index. We do this by creating each unique combination possible of every level of the multi-hierarchical index. Since this is a complex task, you do not need to write it -- but take some time to examine the code in the cell below and see if you can understand how it works

We could also flatten these:

cols0 = grouped.columns.get_level_values(0)

```
cols1 = grouped.columns.get_level_values(1)
grouped.columns = [
    col0 + "_" + col1 if col1 != "" else col0 for col0, col1 in list(zip(cols0, cols1))
]
# The list comprehension above is more complicated then what we need but creates a nicer formatting and
# demonstrates using a conditional within a list comprehension.
# This simpler version works but has some tail underscores where col1 is blank:
# grouped.columns = [col0 + '_' + col1 for col0, col1 in list(zip(cols0, cols1))]
grouped.columns
```



Now that we've flattened the DataFrame, let's inspect a couple rows to see what it looks like.

In the cell below, inspect the `.head()` of the `grouped` DataFrame.

```
grouped.head()
```

	State	Gender	Deaths_mean	Deaths_min	Deaths_max	Deaths_std	Population_mean	Population_min	Population_max	Population_std
0	Alabama	Female	10753.33	10	116297	24612.25	1078712.68	2087	4334752	1400309.40
1	Alabama	Male	10765.85	10	88930	20813.54	1014946.05	1129	4284775	1397829.52
2	Alaska	Female	679.98	13	4727	1154.87	144040.27	1224	682855	201579.34
3	Alaska	Male	860.36	12	5185	1411.78	151888.43	578	770502	223884.30
4	Arizona	Female	8998.39	21	133923	26245.94	1246501.64	12211	6265346	2096631.56

74.Using pivots

```
pivot = grouped.pivot(index='State',columns='Gender',values='Deaths_mean')
pivot
```

Now, we'll gain some practice using the DataFrame's built-in `.pivot()` method.

In the cell below, call the DataFrame's `.pivot()` method with the following paramete

- index = `'State'`
- columns = `'Gender'`
- values = `'Deaths_mean'`

Then, display the `.head()` of our new `pivot` DataFrame to see what it looks like.

```
# Now it's time to pivot
pivot = grouped.pivot(index='State',columns='Gender',values='Deaths_mean')
pivot
```

[194]

Gender	Female	Male
State		
Alabama	10753.33	10765.85
Alaska	679.98	860.36
Arizona	8998.39	10036.20
Arkansas	6621.62	6301.69
California	48312.84	49555.52
Colorado	6460.16	6442.50
Connecticut	7144.64	6315.30
Delaware	2000.03	1940.91
District of Columbia	1497.58	1534.81
Florida	36019.07	36771.34
Georgia	15372.32	14621.07
Hawaii	2182.94	2341.45
Idaho	2874.32	2693.42
Illinois	23432.93	21698.00
Indiana	13425.72	12700.27
Iowa	6419.71	5952.17

b) We've just created a pivot table.

Let's reset the index and see how it changes our pivot table.

We've just created a pivot table.

Let's reset the index and see how it changes our pivot table.

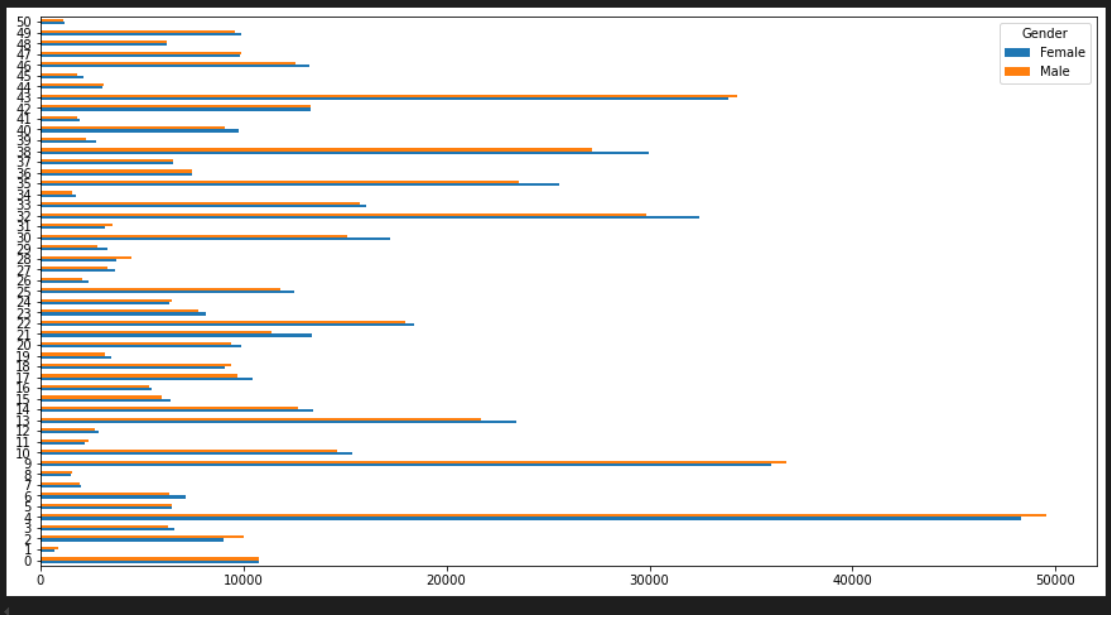
In the cell below, reset the index of the `pivot` object as we did previously. Then, display the `.head()` of the object to see if we can detect any changes.

```
# Again, notice the subtle difference of resetting the index:
pivot = pivot.reset_index()
pivot.head()
```

Gender	State	Female	Male
0	Alabama	10753.33	10765.85
1	Alaska	679.98	860.36
2	Arizona	8998.39	10036.20
3	Arkansas	6621.62	6301.69
4	California	48312.84	49555.52

75. **Visualizing Data With Pivot Tables**

```
# Now let's make a sweet bar chart
pivot.plot(kind='barh',figsize=(15,8));
```



Notice the Y-axis is currently just a list of numbers. That's because when we reset the index, it defaulted to assigning integers as the index for the DataFrame. Let's set the index back to `'State'`, and then recreate the visualization.

pivot.set_index('State').plot(kind='barh',figsize=(15,8));

Notice the Y-axis is currently just a list of numbers. That's because when we reset the index, it defaulted to assigning integers as the index for the DataFrame. Let's set the index back to `'State'`, and then recreate

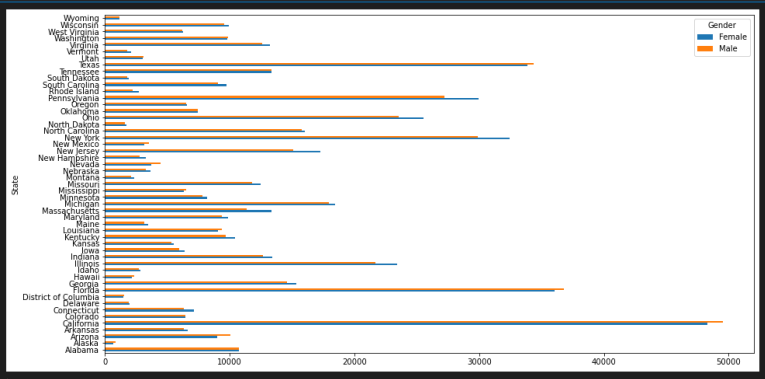
In the cell below:

- Use the `pivot` object's `.set_index()` method and set the index to `'State'`. Then, chain this with a `.plot()` call to recreate the visualization using the code we used in the cell above.

All the code in this cell should be done in a single line. Just call the methods -- do not rebind `pivot` to be equal to this line of code.

```
# Where are the states? Notice the y-axis is just a list of numbers.
# This is populated by the DataFrame's index.
# When we used the .reset_index() method, we created a new numbered index to name each row.
# Let's fix that by making state the index again.
pivot.set_index('State').plot(kind='barh',figsize=(15,8));
```

[197]



76.Stack Bar Chart

pivot = pivot.set_index('State')

pivot.plot(kind='barh',figsize=(15,8),stacked=True)

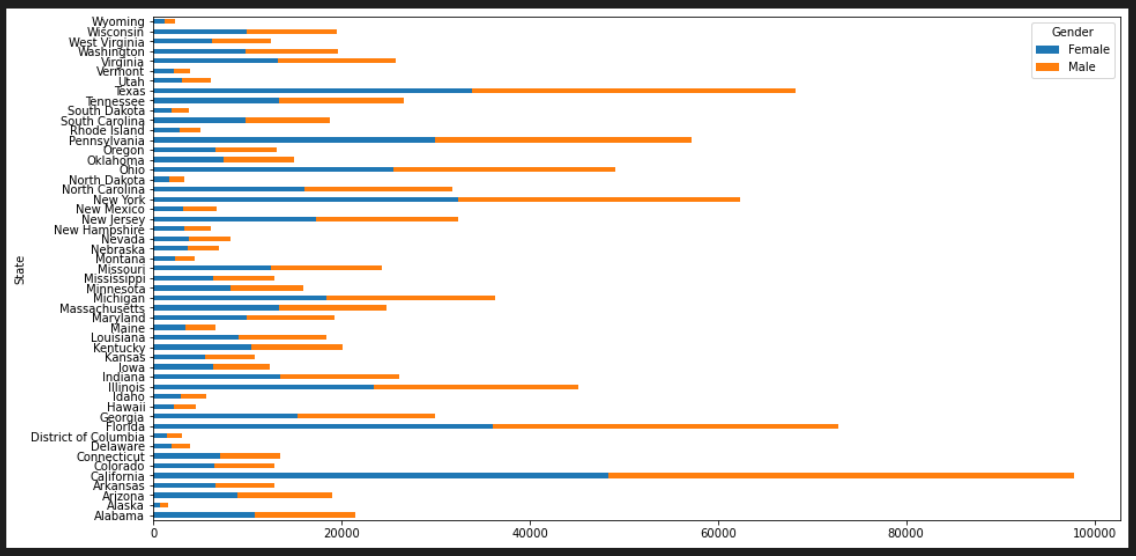
Finally, let's stack these bar charts to see how that looks.

In the cell below, recreate the visualization we did in the cell above, but this time, also pass in `stacked=True` as a parameter.

```
# Lastly, let's stack each of these bars for each state.
# Notice we don't have to worry about index here, because we've already set it above.
pivot.plot(kind='barh',figsize=(15,8),stacked=True)
```

[201]

```
<AxesSubplot:ylabel='State'>
```



77. Get Sample

```
data.sample(10,random_state=10)
```

78. (FIFA Dataset data cleaning)

a)remove leading space

```
fifa['Club'] = fifa['Club'].str.strip()
```

```
fifa['Club'].unique()
```

```
array(['\n\n\n\nFC Barcelona', '\n\n\n\nJuventus',
       '\n\n\n\nAtlético Madrid', '\n\n\n\nManchester City',
       '\n\n\n\nParis Saint-Germain', '\n\n\n\nFC Bayern München',
       '\n\n\n\nLiverpool', '\n\n\n\nReal Madrid', '\n\n\n\nChelsea',
       '\n\n\n\nTottenham Hotspur', '\n\n\n\nInter', '\n\n\n\nNapoli',
       '\n\n\n\nBorussia Dortmund', '\n\n\n\nManchester United',
       '\n\n\n\nArsenal', '\n\n\n\nLazio', '\n\n\n\nLeicester City',
       '\n\n\n\nBorussia Mönchengladbach', '\n\n\n\nReal Sociedad',
       '\n\n\n\nAtalanta', '\n\n\n\nOlympique Lyonnais', '\n\n\n\nMilan',
       '\n\n\n\nVillarreal CF', '\n\n\n\nRB Leipzig', '\n\n\n\nCagliari',
       '\n\n\n\nAjax', '\n\n\n\nSL Benfica', '\n\n\n\nAS Monaco',
       '\n\n\n\nWolverhampton Wanderers', '\n\n\n\nEverton',
       '\n\n\n\nFiorentina', '\n\n\n\nFC Porto', '\n\n\n\nRC Celta',
       '\n\n\n\nTorino', '\n\n\n\nSevilla FC', '\n\n\n\nGrêmio',
       '\n\n\n\nReal Betis', '\n\n\n\nRoma', '\n\n\n\nNewcastle United',
       '\n\n\n\nEintracht Frankfurt', '\n\n\n\nValencia CF',
       '\n\n\n\nMedipol Başakşehir FK', '\n\n\n\nInter Miami',
       '\n\n\n\nBayer 04 Leverkusen', '\n\n\n\nLevante UD',
       '\n\n\n\nCrystal Palace', '\n\n\n\nAthletic Club de Bilbao',
       '\n\n\n\nShanghai SIPG FC', '\n\n\n\nVfL Wolfsburg',
       '\n\n\n\nGuangzhou Evergrande Taobao FC', '\n\n\n\nAl Shabab',
       '\n\n\n\nOlympique de Marseille', '\n\n\n\nLos Angeles FC',
       '\n\n\n\nBeijing Sinobo Guoan FC', '\n\n\n\nGetafe CF',
       '\n\n\n\nSV Werder Bremen', '\n\n\n\nTSG 1899 Hoffenheim',
       '\n\n\n\nLOSC Lille', '\n\n\n\nDeportivo Alavés',
...
       '\n\n\n\nLlaneros de Guanare', '\n\n\n\nSligo Rovers',
       '\n\n\n\nZamora FC', "\n\n\n\nSt. Patrick's Athletic",
       '\n\n\n\nCork City', '\n\n\n\nShelbourne FC',
       '\n\n\n\nHarrogate Town', '\n\n\n\nWaterford FC',
       '\n\n\n\nFinn Harps'], dtype=object)
```

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

```
#remove leading spaces
fifa['Club'] = fifa['Club'].str.strip()
fifa['Club'].unique()
```

```
array(['FC Barcelona', 'Juventus', 'Atlético Madrid', 'Manchester City',
       'Paris Saint-Germain', 'FC Bayern München', 'Liverpool',
       'Real Madrid', 'Chelsea', 'Tottenham Hotspur', 'Inter', 'Napoli',
       'Borussia Dortmund', 'Manchester United', 'Arsenal', 'Lazio',
```

.b)get start and end date from column

```
    fifa['Contract'].unique()
```

```
array(['2004 ~ 2021', '2018 ~ 2022', '2014 ~ 2023', '2015 ~ 2023',
       '2017 ~ 2022', '2017 ~ 2023', '2018 ~ 2024', '2014 ~ 2022',
       '2018 ~ 2023', '2016 ~ 2023', '2013 ~ 2023', '2011 ~ 2023',
       '2009 ~ 2022', '2005 ~ 2021', '2011 ~ 2021', '2015 ~ 2022',
       '2017 ~ 2024', '2010 ~ 2024', '2012 ~ 2021', '2019 ~ 2024',
       '2015 ~ 2024', '2017 ~ 2025', '2020 ~ 2025', '2019 ~ 2023',
       '2008 ~ 2023', '2015 ~ 2021', '2020 ~ 2022', '2012 ~ 2022',
       '2016 ~ 2025', '2013 ~ 2022', '2011 ~ 2022', '2012 ~ 2024',
       '2016 ~ 2021', '2012 ~ 2023', '2008 ~ 2022', '2019 ~ 2022',
       '2017 ~ 2021', '2013 ~ 2024', '2020 ~ 2024', '2010 ~ 2022',
       '2020 ~ 2021', '2011 ~ 2024', '2020 ~ 2023', '2014 ~ 2024',
       '2013 ~ 2026', '2016 ~ 2022', '2010 ~ 2021', '2013 ~ 2021',
       '2019 ~ 2025', '2018 ~ 2025', '2016 ~ 2024', '2018 ~ 2021',
       '2009 ~ 2024', '2007 ~ 2022', 'Jun 30, 2021 On Loan',
       '2009 ~ 2021', '2019 ~ 2021', '2019 ~ 2026', 'Free', '2012 ~ 2028',
       '2010 ~ 2023', '2014 ~ 2021', '2015 ~ 2025', '2014 ~ 2026',
       '2012 ~ 2025', '2017 ~ 2020', '2002 ~ 2022', '2020 ~ 2027',
       '2013 ~ 2025', 'Dec 31, 2020 On Loan', '2019 ~ 2020',
       '2011 ~ 2025', '2016 ~ 2020', '2007 ~ 2021', '2020 ~ 2026',
       '2010 ~ 2025', '2009 ~ 2023', '2008 ~ 2021', '2020 ~ 2020',
       '2016 ~ 2026', 'Jan 30, 2021 On Loan', '2012 ~ 2020',
       '2014 ~ 2025', 'Jun 30, 2022 On Loan', '2015 ~ 2020',
       'May 31, 2021 On Loan', '2018 ~ 2020', '2014 ~ 2020',
       '2013 ~ 2020', '2006 ~ 2024', 'Jul 5, 2021 On Loan',
       'Dec 31, 2021 On Loan', '2004 ~ 2025', '2011 ~ 2020',
...
       'Jun 30, 2023 On Loan', '1998 ~ 2021', '2003 ~ 2022',
       '2007 ~ 2023', 'Jul 31, 2021 On Loan', 'Nov 22, 2020 On Loan',
       'May 31, 2022 On Loan', '2006 ~ 2020', 'Dec 30, 2020 On Loan',
       '2007 ~ 2025', 'Jan 4, 2021 On Loan', 'Nov 30, 2020 On Loan',
       '2004 ~ 2020', '2009 ~ 2025', 'Aug 1, 2021 On Loan'], dtype=object)
```

Check if format consistent

```
for index, row in fifa.iterrows():
    if ' On Loan' in row['Contract'] or 'Free' in row['Contract']:
        print(row['Contract'])
```

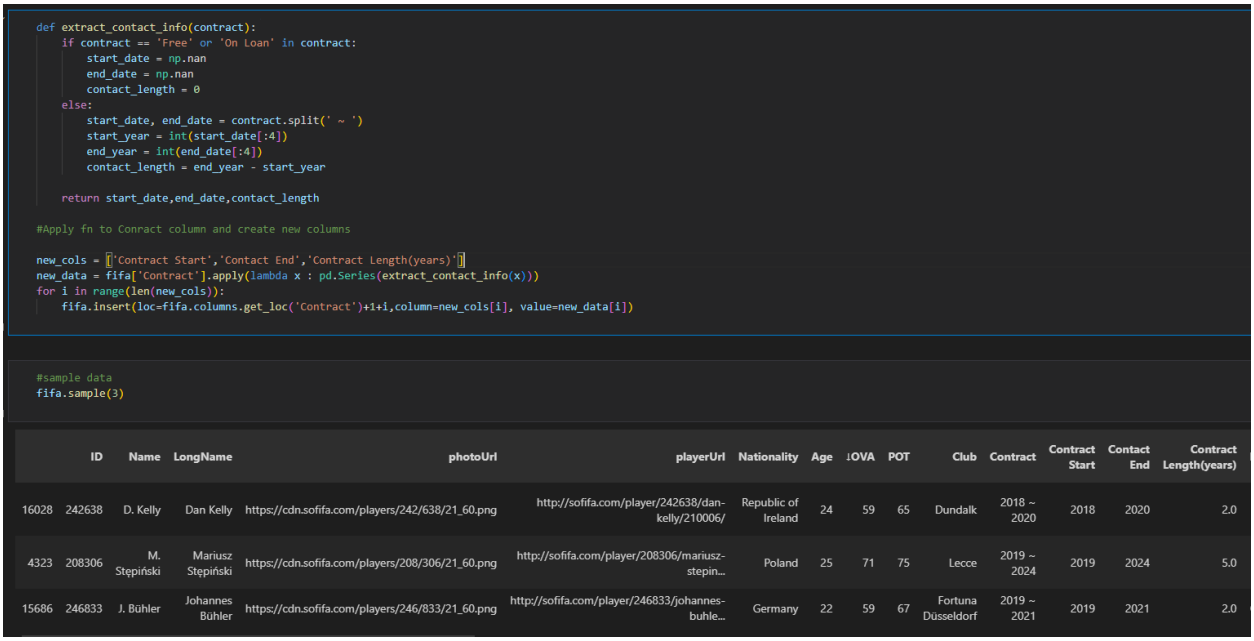
#Format is consistent 2 letter words on loan

```
    for index, row in fifa.iterrows():
        if ' On Loan' in row['Contract'] or 'Free' in row['Contract']:
            print(row['Contract'])

    #Format is consistent 2 letter  words on loan
```

```
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Free
Free
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Free
Free
Free
Free
Free
Free
Free
Free
Free
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Jun 30, 2021 On Loan
Free
Free
...
Jun 30, 2021 On Loan
Dec 31, 2020 On Loan
Dec 31, 2020 On Loan
Dec 31, 2020 On Loan
```

```
def extract_contact_info(contract):
    if contract == 'Free' or 'On Loan' in contract:
        start_date = np.nan
        end_date = np.nan
```

```
        contact_length = 0
    else:
        start_date, end_date = contract.split(' ~ ')
        start_year = int(start_date[:4])
        end_year = int(end_date[:4])
        contact_length = end_year - start_year

    return start_date,end_date,contact_length

#Apply fn to Conract column and create new columns

new_cols = ['Contract Start','Contact End','Contract Length(years)']
new_data = fifa['Contract'].apply(lambda x : pd.Series(extract_contact_info(x)))
for i in range(len(new_cols)):
    fifa.insert(loc=fifa.columns.get_loc('Contract')+1+i,column=new_cols[i],
value=new_data[i])
```



```
def extract_contact_info(contract):
    if contract == 'Free' or 'On Loan' in contract:
        start_date = np.nan
        end_date = np.nan
        contact_length = 0
    else:
        start_date, end_date = contract.split(' ~ ')
        start_year = int(start_date[:4])
        end_year = int(end_date[:4])
        contact_length = end_year - start_year

    return start_date,end_date,contact_length

#Apply fn to Conract column and create new columns

new_cols = ['Contract Start','Contact End','Contract Length(years)']
new_data = fifa['Contract'].apply(lambda x : pd.Series(extract_contact_info(x)))
for i in range(len(new_cols)):
    fifa.insert(loc=fifa.columns.get_loc('Contract')+1+i,column=new_cols[i], value=new_data[i])
```

```
#sample data
fifa.sample(3)
```

	ID	Name	LongName	photoUrl	playerUrl	Nationality	Age	OVA	POT	Club	Contract	Contract Start	Contact End	Contract Length(years)
16028	242638	D. Kelly	Dan Kelly	https://cdn.sofifa.com/players/242/638/21_60.png	http://sofifa.com/player/242638/dan-kelly/210006/	Republic of Ireland	24	59	65	Dundalk	2018 ~ 2020	2018	2020	2.0
4323	208306	M. Stępiński	Mariusz Stępiński	https://cdn.sofifa.com/players/208/306/21_60.png	http://sofifa.com/player/208306/mariusz-stepin...	Poland	25	71	75	Lecce	2019 ~ 2024	2019	2024	5.0
15686	246833	J. Bühler	Johannes Bühler	https://cdn.sofifa.com/players/246/833/21_60.png	http://sofifa.com/player/246833/johannes-buhle...	Germany	22	59	67	Fortuna Düsseldorf	2019 ~ 2021	2019	2021	2.0

c)Contract categories

```
def categorize_contract_status(contract_status):
    if contract_status =='Free':
        return 'Free'
    elif 'On Loan' in contract_status:
        return 'on Loan'
    else:
        return'Contract'
```

#Add Contact Status Column

```
fifa.insert(fifa.columns.get_loc('Contract Length(years)')+1,'Contract status', fifa['Contract'].apply(categorize_contract_status()))
```

```
#subset of the data frame
fifa[['Contract','Contract Start','Contact End','Contract Length(years)']].sample(5)
```

	Contract	Contract Start	Contact End	Contract Length(years)
1617	2016 ~ 2021	2016	2021	5.0
2446	2019 ~ 2023	2019	2023	4.0
9286	2018 ~ 2021	2018	2021	3.0
4309	2019 ~ 2023	2019	2023	4.0
13353	2019 ~ 2021	2019	2021	2.0

```
#Contact categories
def categorize_contract_status(contract_status):
    if contract_status =='Free':
        return 'Free'
    elif 'On Loan' in contract_status:
        return 'on Loan'
    else:
        return'Contract'

#Add Contact Status Column
fifa.insert(fifa.columns.get_loc('Contract Length(years)')+1,'Contract status', fifa['Contract'].apply(categorize_contract_status()))
```

```
#View sample of data to make sure it has been inserted
fifa.sample(5)
```

	ID	Name	LongName	photoUrl	playerUrl	Nationality	Age	OVA	POT	Club	Contract	Contract Start	Contact End	Contract Length(years)	Contract status	P
9066	224395	B. Bahn	Bentley Baxter Bahn	https://cdn.sofifa.com/players/224/395/21_60.png	http://sofifa.com/player/224395/bentley-baxter...	Germany	27	66	66	FC Hansa Rostock	2020 ~ 2022	2020	2022	2.0	Contract	C
15202	242060	A. Ashraf	Ahmed Ashraf	https://cdn.sofifa.com/players/242/060/21_60.png	http://sofifa.com/player/242060/ahmed-ashraf/2...	Saudi Arabia	27	60	60	Al Hilal	2018 ~ 2021	2018	2021	3.0	Contract	
9035	237708	M. Sylla	Moussa Sylla	https://cdn.sofifa.com/players/237/708/21_60.png	http://sofifa.com/player/237708/moussa-sylla/2...	France	20	66	77	FC Utrecht	2020 ~ 2024	2020	2024	4.0	Contract	

-The insert adds the columns after the Contract status column

```
fifa[['Contract','Contract Start','Contact End','Contract Length(years)','Contract status']].sample(5)
```

```
fifa[['Contract','Contract Start','Contact End','Contract Length(years)','Contract status']].sample(5)
```

	Contract	Contract Start	Contact End	Contract Length(years)	Contract status
3910	2012 ~ 2020	2012	2020	8.0	Contract
17183	Aug 1, 2021 On Loan	NaN	NaN	0.0	on Loan
11634	2020 ~ 2022	2020	2022	2.0	Contract
11040	2020 ~ 2021	2020	2021	1.0	Contract
3853	2017 ~ 2021	2017	2021	4.0	Contract

d)remove cm in heights and those in ft convert to cm

```
def convert_height(height):
    if "cm" in height:
        return int(height.strip("cm"))
    else:
        feet , inches = height.split("'")
        total_inches = int(feet) *12 + int(inches.strip("\""))
        return round(total_inches * 2.54)


#apply fn to height column
fifa['Height'] = fifa['Height'].apply(convert_height)
fifa['Height'] .unique()
#Worked but column doesnt tell us the height is in cm so we need to rename
```

```
fifa['Height'].unique()
```

```
array(['170cm', '187cm', '188cm', '181cm', '175cm', '184cm', '191cm',
       '178cm', '193cm', '185cm', '199cm', '173cm', '168cm', '176cm',
       '177cm', '183cm', '180cm', '189cm', '179cm', '195cm', '172cm',
       '182cm', '186cm', '192cm', '165cm', '194cm', '167cm', '196cm',
       '163cm', '190cm', '174cm', '169cm', '171cm', '197cm', '200cm',
       '166cm', '6\'2"', '164cm', '198cm', '6\'3"', '6\'5"', '5\'11"',
       '6\'4"', '6\'1"', '6\'0"', '5\'10"', '5\'9"', '5\'6"', '5\'7"',
       '5\'4"', '201cm', '158cm', '162cm', '161cm', '160cm', '203cm',
       '157cm', '156cm', '202cm', '159cm', '206cm', '155cm'], dtype=object)
```

```
def convert_height(height):
    if "cm" in height:
        return int(height.strip("cm"))
    else:
        feet , inches = height.split("'")
        total_inches = int(feet) *12 + int(inches.strip('"'))
        return round(total_inches * 2.54)

#apply fn to height column
fifa['Height'] = fifa['Height'].apply(convert_height)
fifa['Height'] .unique()
#Worked but column doesnt tell us the height is in cm so we need to rename
```

```
array([170, 187, 188, 181, 175, 184, 191, 178, 193, 185, 199, 173, 168,
       176, 177, 183, 180, 189, 179, 195, 172, 182, 186, 192, 165, 194,
       167, 196, 163, 190, 174, 169, 171, 197, 200, 166, 164, 198, 201,
       158, 162, 161, 160, 203, 157, 156, 202, 159, 206, 155], dtype=int64)
```

```
fifa = fifa.rename(columns={
    'Height':'Height(cm)'
})
fifa.sample(3)
```

e)weight in lbs and kg's(need to strip lbl and kg's and convert all to kgs;)

```
def convert_weight(weight):
    if 'kg' in weight:
        return int(weight.strip('kg'))
    else:
        pounds = int(weight.strip('lbs'))
        return round(pounds / 2.205)
convert_weight('181lbs')
#apply function to weight column
fifa['Weight'] = fifa['Weight'].apply(convert_weight)
```

```
fifa['Weight'].unique()
```

```
array(['72kg', '83kg', '87kg', '70kg', '68kg', '80kg', '71kg', '91kg',
       '73kg', '85kg', '92kg', '69kg', '84kg', '96kg', '81kg', '82kg',
       '75kg', '86kg', '89kg', '74kg', '76kg', '64kg', '78kg', '90kg',
       '66kg', '60kg', '94kg', '79kg', '67kg', '65kg', '59kg', '61kg',
       '93kg', '88kg', '97kg', '77kg', '62kg', '63kg', '95kg', '100kg',
       '58kg', '183lbs', '179lbs', '172lbs', '196lbs', '176lbs', '185lbs',
       '170lbs', '203lbs', '168lbs', '161lbs', '146lbs', '130lbs',
       '190lbs', '174lbs', '148lbs', '165lbs', '159lbs', '192lbs',
       '181lbs', '139lbs', '154lbs', '157lbs', '163lbs', '98kg', '103kg',
       '99kg', '102kg', '56kg', '101kg', '57kg', '55kg', '104kg', '107kg',
       '110kg', '53kg', '50kg', '54kg', '52kg'], dtype=object)
```

```
def convert_weight(weight):
    if 'kg' in weight:
        return int(weight.strip('kg'))
    else:
        pounds = int(weight.strip('lbs'))
        return round(pounds / 2.205)
convert_weight('181lbs')
#apply function to weight column
fifa['Weight'] = fifa['Weight'].apply(convert_weight)

#check if everything converted well
fifa['Weight'].unique()
#look at the weight and cant tell the units they are in
```

```
array([ 72,  83,  87,  70,  68,  80,  71,  91,  73,  85,  92,  69,  84,
        96,  81,  82,  75,  86,  89,  74,  76,  64,  78,  90,  66,  60,
        94,  79,  67,  65,  59,  61,  93,  88,  97,  77,  62,  63,  95,
       100,  58,  98, 103,  99, 102,  56, 101,  57,  55, 104, 107, 110,
        53,  50,  54,  52], dtype=int64)
```

```
#rename weight column
fifa =fifa.rename(columns={
    'Weight': 'Weight(Kg)'
})
fifa.sample(3)
```

f)just check if loan end date is for the people on loan and if the date matches

19. Loan Date End

Date when the player's loan ends (if On Loan)

```
[176]
#loan end date has many missing values since few were on loan
fifa['Loan Date End'].dtype
```

```
dtype('O')
```

```
[177]
fifa['Loan Date End'].unique()
```

```
array([nan, '30-Jun-21', '31-Dec-20', '30-Jan-21', '30-Jun-22',
       '31-May-21', '05-Jul-21', '31-Dec-21', '01-Jul-21', '01-Jan-21',
       '31-Aug-21', '31-Jan-21', '30-Dec-21', '23-Jun-21', '03-Jan-21',
       '27-Nov-21', '17-Jan-21', '30-Jun-23', '31-Jul-21', '22-Nov-20',
       '31-May-22', '30-Dec-20', '04-Jan-21', '30-Nov-20', '01-Aug-21'],
      dtype=object)
```

```
[190]
#check players on loan
on_loan = fifa[fifa['Contract status'] == 'on Loan']
#Check subset
on_loan[['Contract','Contract status','Loan Date End']] # 1013 rows as seen from the info
```

	Contract	Contract status	Loan Date End
205	Jun 30, 2021 On Loan	on Loan	30-Jun-21
248	Jun 30, 2021 On Loan	on Loan	30-Jun-21
254	Jun 30, 2021 On Loan	on Loan	30-Jun-21
302	Jun 30, 2021 On Loan	on Loan	30-Jun-21
306	Jun 30, 2021 On Loan	on Loan	30-Jun-21
...	...	...	...
18472	Aug 31, 2021 On Loan	on Loan	31-Aug-21
18571	Jun 30, 2021 On Loan	on Loan	30-Jun-21
18600	Dec 31, 2020 On Loan	on Loan	31-Dec-20
18622	Dec 31, 2020 On Loan	on Loan	31-Dec-20

g)remove the stars icon on rating

method1

fifa['W/F'] = fifa['W/F'].str.replace('★','')

```
fifa['W/F'].dtype
```

```
dtype('O')
```

```
fifa['W/F'].unique() # if you want for categorical remain as is but if we want for calculation  we can convert to int
```

```
array(['4 ★', '3 ★', '5 ★', '2 ★', '1 ★'], dtype=object)
```

```
# my code
#fifa['W/F'] = fifa['W/F'].map(lambda x: int(x[:1]))
fifa['W/F'] = fifa['W/F'].str.replace('★','')
fifa['W/F'].unique()
```

```
array(['4 ', '3 ', '5 ', '2 ', '1 '], dtype=object)
```

Method2

fifa['SM'] = fifa['SM'].map(lambda x: int(x[:1]))

```
fifa['SM'].dtype
```

```
dtype('O')
```

```
fifa['SM'].unique()
fifa['SM'] = fifa['SM'].map(lambda x: int(x[:1]))
fifa['SM'].unique()
```

```
array([4, 5, 1, 2, 3], dtype=int64)
```

79.Tree planting dataset

a)Select necessary columns from a dataset

tree_census_subset
= tree_census[['tree_id','tree_dbh','stump_diam','curb_loc','status','health', 'spc_latin','steward','sidewalk', 'problems', 'root_stone','root_grate', 'root_other', 'trunk_wire', 'trnk_light', 'trnk_other',

'brch_light', 'brch_shoe', 'brch_other']]

b)plot histogram for the data(be defaults plot only numeric values)

tree_census_subset.hist(bins=60,figsize=(20,10));

See most are less than 50

-Check how this values look for both?

treedbh

big_trees = tree_census_subset[tree_census_subset['tree_dbh'] > 50]

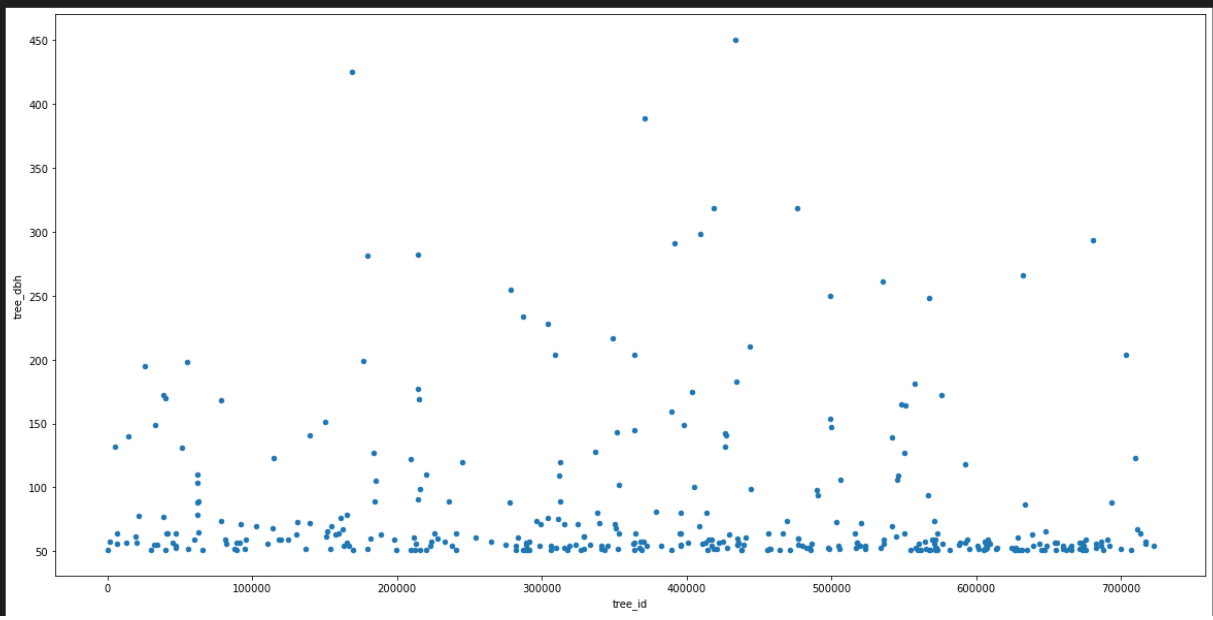
big_trees

import matplotlib.pyplot as plt

#plt.scatter(big_trees.tree_id,big_trees.tree_dbh);

big_trees[['tree_id','tree_dbh']].plot(kind='scatter',x='tree_id',y='tree_dbh',figsize=(20,10));

```
#my code
#visualize this
import matplotlib.pyplot as plt
#plt.scatter(big_trees.tree_id,big_trees.tree_dbh);
big_trees[['tree_id','tree_dbh']].plot(kind='scatter',x='tree_id',y='tree_dbh',figsize=(20,10));
```

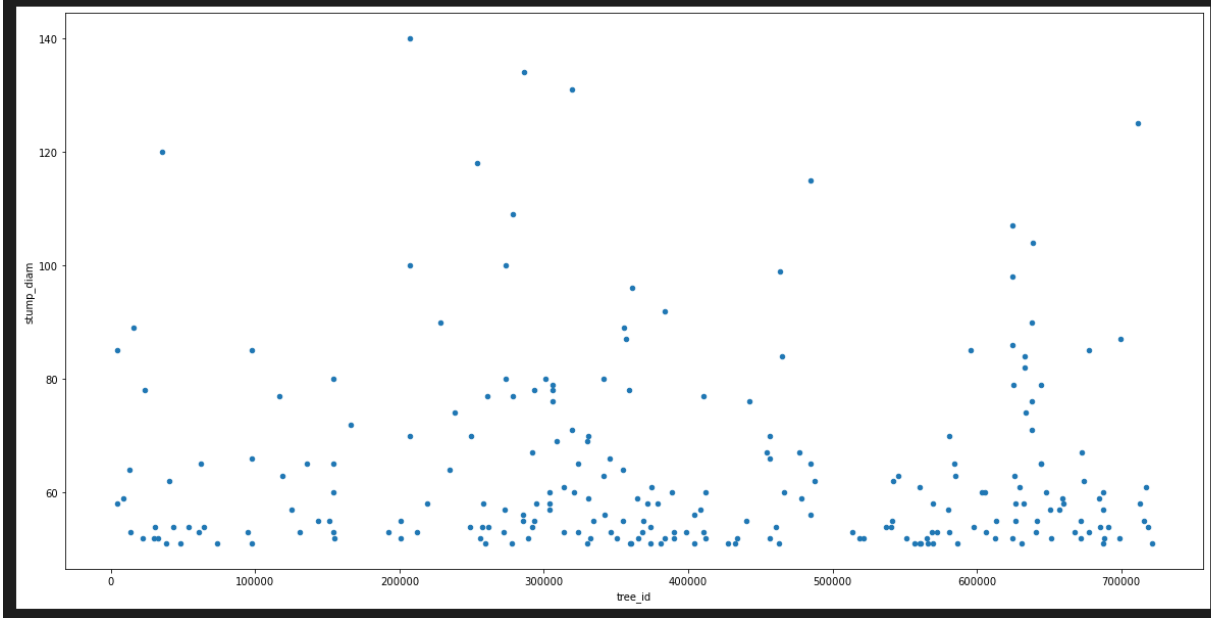


stump_diam

big_trees2 = tree_census_subset[tree_census_subset['stump_diam'] > 50]

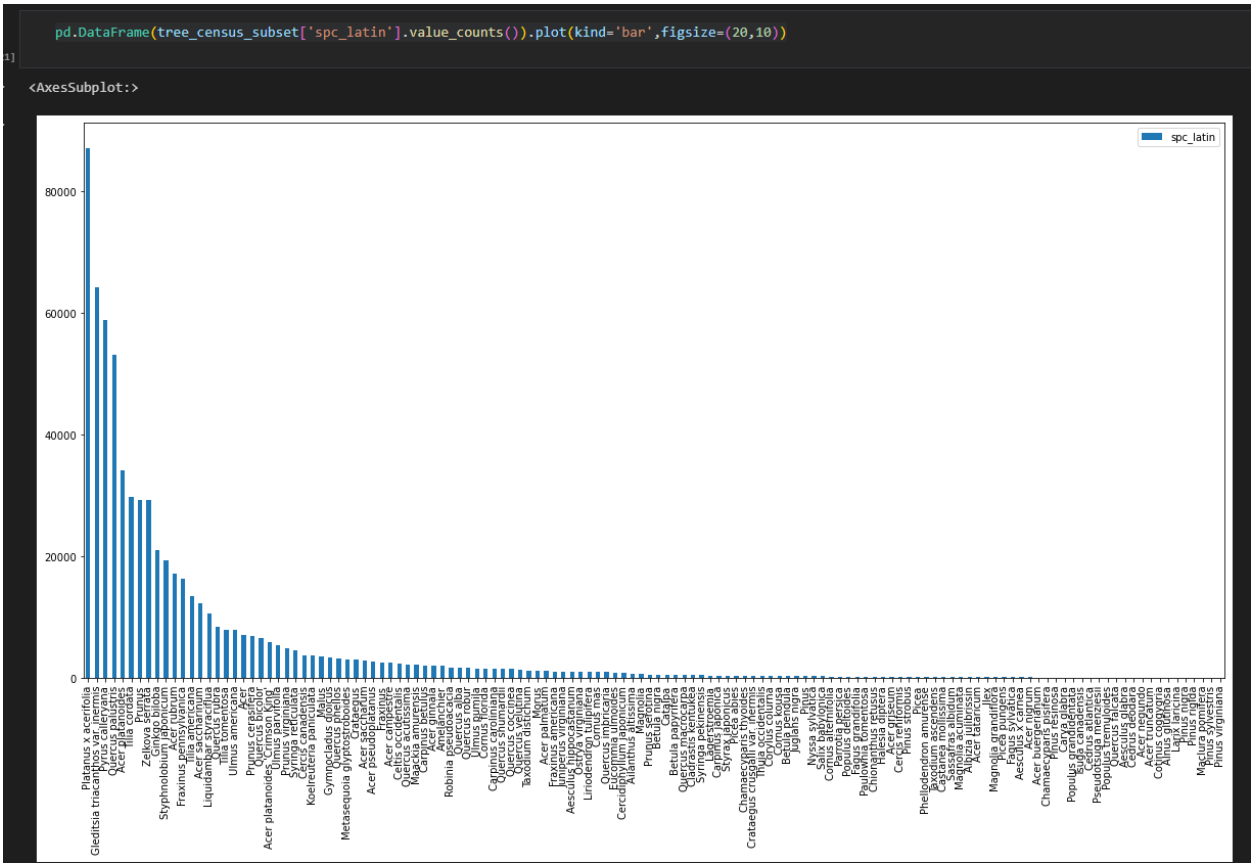
big_trees2

```
big_trees2[['tree_id','stump_diam']].plot(kind='scatter',x='tree_id',y='stump_diam',figsize=(20,10));
```



c)analyze name column

pd.DataFrame(tree_census_subset['spc_latin'].value_counts()).plot(kind='bar',figsize=(20,10))



d)check values in categorical columns

```
tree_census_subset['steward'].value_counts()
```

```
None       487823
1or2       143557
3or4        19183
4orMore      1610
Name: steward, dtype: int64
```

```
tree_census_subset['sidewalk'].value_counts()
```

```
NoDamage    464978
Damage      187194
Name: sidewalk, dtype: int64
```

```
tree_census_subset['status'].value_counts()
```

```
Alive    652173
Stump     17654
Dead      13961
Name: status, dtype: int64
```

```
tree_census_subset['curb_loc'].value_counts()
```

```
OnCurb            656896
OffsetFromCurb     26892
Name: curb_loc, dtype: int64
```

```
stumps = tree_census_subset[tree_census_subset['status'] =='Stump']
stumps
```

e)Apply value_counts to certain columns

tree_problems = tree_census_subset[['root_stone',

'root_grate', 'root_other', 'trunk_wire', 'trnk_light', 'trnk_other',

'brch_light', 'brch_shoe', 'brch_other']]

tree_problems

tree_problems.apply(pd.Series.value_counts) #apply value count to each of the series

#root_stones have many problems compared to the rest

```
tree_problems = tree_census_subset[['root_stone',
       'root_grate', 'root_other', 'trunk_wire', 'trnk_light', 'trnk_other',
       'brch_light', 'brch_shoe', 'brch_other']]
tree_problems
```

	root_stone	root_grate	root_other	trunk_wire	trnk_light	trnk_other	brch_light	brch_shoe	brch_other
0	No	No	No	No	No	No	No	No	No
1	Yes	No	No	No	No	No	No	No	No
2	No	No	No	No	No	No	No	No	No
3	Yes	No	No	No	No	No	No	No	No
4	Yes	No	No	No	No	No	No	No	No
...	...	...	...	...	...	...	...	...	...
683783	No	No	No	No	No	No	No	No	No
683784	No	No	No	No	No	No	No	No	No
683785	No	No	No	No	No	No	No	No	No
683786	No	No	No	No	No	No	No	No	No
683787	No	No	No	No	No	No	No	No	No

683788 rows × 9 columns

```
tree_problems.apply(pd.Series.value_counts) #apply value count to each of the series
#root_stones have many problems compared to the rest
```

	root_stone	root_grate	root_other	trunk_wire	trnk_light	trnk_other	brch_light	brch_shoe	brch_other
No	543789	680252	653466	670514	682757	651215	621423	683377	659433
Yes	139999	3536	30322	13274	1031	32573	62365	411	24355

f)fill health status with NoT Applicable where status -dump or dead

mask = ((tree_census_subset['status'] =='Stump') | (tree_census_subset['status'] =='Dead'))

replaces for only health columns

tree_census_subset.loc[mask, 'health'] = tree_census_subset.loc[mask, 'health'].fillna('Not Applicable')

For all columns

tree_census_subset.loc[mask] = tree_census_subset.loc[mask].fillna('Not Applicable')

```
mask = ((tree_census_subset['status'] =='Stump') | (tree_census_subset['status'] =='Dead'))
mask
```

✓ 0.0s

```
0         False
1         False
2         False
3         False
4         False
          ...
683783    False
683784    False
683785    False
683786    False
683787    False
Name: status, Length: 683788, dtype: bool
```

```
# replaces for only health columns
tree_census_subset.loc[mask, 'health'] = tree_census_subset.loc[mask, 'health'].fillna('Not Applicable')
# For all columns
tree_census_subset.loc[mask] = tree_census_subset.loc[mask].fillna('Not Applicable')
```

✓ 0.5s

```
tree_census_subset[tree_census_subset['status'] =='Stump']
```

✓ 0.1s

	tree_id	tree_dbh	stump_diam	curb_loc	status	health	spc_latin	steward	sidewalk	problems	root_stone
37	211205	0	16	OnCurb	Stump	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	No
239	203597	0	15	OnCurb	Stump	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	No
641	179766	0	30	OnCurb	Stump	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	No
644	180619	0	10	OnCurb	Stump	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	No
646	208828	0	15	OnCurb	Stump	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	No
...	...	...	...	...	...	...	...	...	...	...	...

On info

```
tree_census_subset.isna().sum()
```
✓ 3.3s

```
tree_id        0
tree_dbh       0
stump_diam     0
curb_loc       0
status         0
health         1
spc_latin      5
steward        0
sidewalk       1
problems      49
root_stone     0
root_grate     0
root_other     0
trunk_wire     0
trnk_light     0
trnk_other     0
brch_light     0
brch_shoe      0
brch_other     0
dtype: int64
```

tree_census_subset['problems'].fillna('None',inplace=True)

tree_census_subset['health'].fillna('Good',inplace=True)

tree_census_subset['sidewalk'].fillna('NoDamage',inplace=True)

tree_census_subset['spc_latin'].fillna('No Observation',inplace=True)

replaced with most common except the name

80.Convert columns to lower and strip spaces

df1.columns = df1.columns.str.lower().str.replace(' ','')

81.fill with mode

#impute using mode

gender_mode = df1.gender.mode()[0]

df1.gender.fillna(gender_mode,inplace=True) ## use bfill for backword fill, and ffill for forward fill

82.a)Check for outliers

sns.boxplot(df1.iloc[:,1:]) #sns.boxplot(df1)

b)outlier for certain column

sns.boxplot(df1,y='cr');

c)Remove outlier using maximum quantile

max_cr = df1['cr'].quantile(.995)

#Check the outliers

df1[df1['cr'] >max_cr]

```
#Remove outlier using maximum quantile
max_cr = df1['cr'].quantile(.995)
max_cr
```

```
401.0
```

```
#Check the outliers
df1[df1['cr'] >max_cr]
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
273	34325	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
283	24060	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
846	34325	M	56.00	20.80	800.00	9.00	4.60	2.00	1.20	2.50	0.90	35.00	Y
860	51623	M	60.00	20.80	800.00	9.00	2.30	1.10	0.90	0.90	0.50	33.00	Y

```
#also use .loc
df1.loc[df1['cr']> max_cr]
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
273	34325	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
283	24060	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
846	34325	M	56.00	20.80	800.00	9.00	4.60	2.00	1.20	2.50	0.90	35.00	Y
860	51623	M	60.00	20.80	800.00	9.00	2.30	1.10	0.90	0.90	0.50	33.00	Y

```
#use query
df1.query('cr > @max_cr')
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
273	34325	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
283	24060	M	58.00	20.80	800.00	9.10	6.60	2.90	1.10	4.30	1.30	33.00	Y
846	34325	M	56.00	20.80	800.00	9.00	4.60	2.00	1.20	2.50	0.90	35.00	Y
860	51623	M	60.00	20.80	800.00	9.00	2.30	1.10	0.90	0.90	0.50	33.00	Y

#remove

df1[df1['cr'] <=max_cr]

df1[~(df1['cr'] > max_cr)]

```
#remove the outlier
df1[df1['cr'] <=max_cr]
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
0	17975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
1	34221	M	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00	N
2	47975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
3	87656	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
4	34223	M	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...
997	87654	M	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40	Y
998	24004	M	38.00	5.80	59.00	6.70	5.30	2.00	1.60	2.90	14.00	40.50	Y
1000	454316	M	64.00	8.80	106.00	8.50	5.90	2.10	1.20	4.00	1.20	32.00	Y
1005	454316	M	55.00	4.80	88.00	8.00	5.70	4.00	0.90	3.30	1.80	30.00	Y
1007	454316	F	57.00	4.10	70.00	9.30	5.30	3.30	1.00	1.40	1.30	29.00	Y

990 rows × 13 columns

```
df1[~(df1['cr'] > max_cr)]
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
0	17975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
1	34221	M	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00	N
2	47975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
3	87656	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
4	34223	M	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...
997	87654	M	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40	Y
998	24004	M	38.00	5.80	59.00	6.70	5.30	2.00	1.60	2.90	14.00	40.50	Y
1000	454316	M	64.00	8.80	106.00	8.50	5.90	2.10	1.20	4.00	1.20	32.00	Y
1005	454316	M	55.00	4.80	88.00	8.00	5.70	4.00	0.90	3.30	1.80	30.00	Y
1007	454316	F	57.00	4.10	70.00	9.30	5.30	3.30	1.00	1.40	1.30	29.00	Y

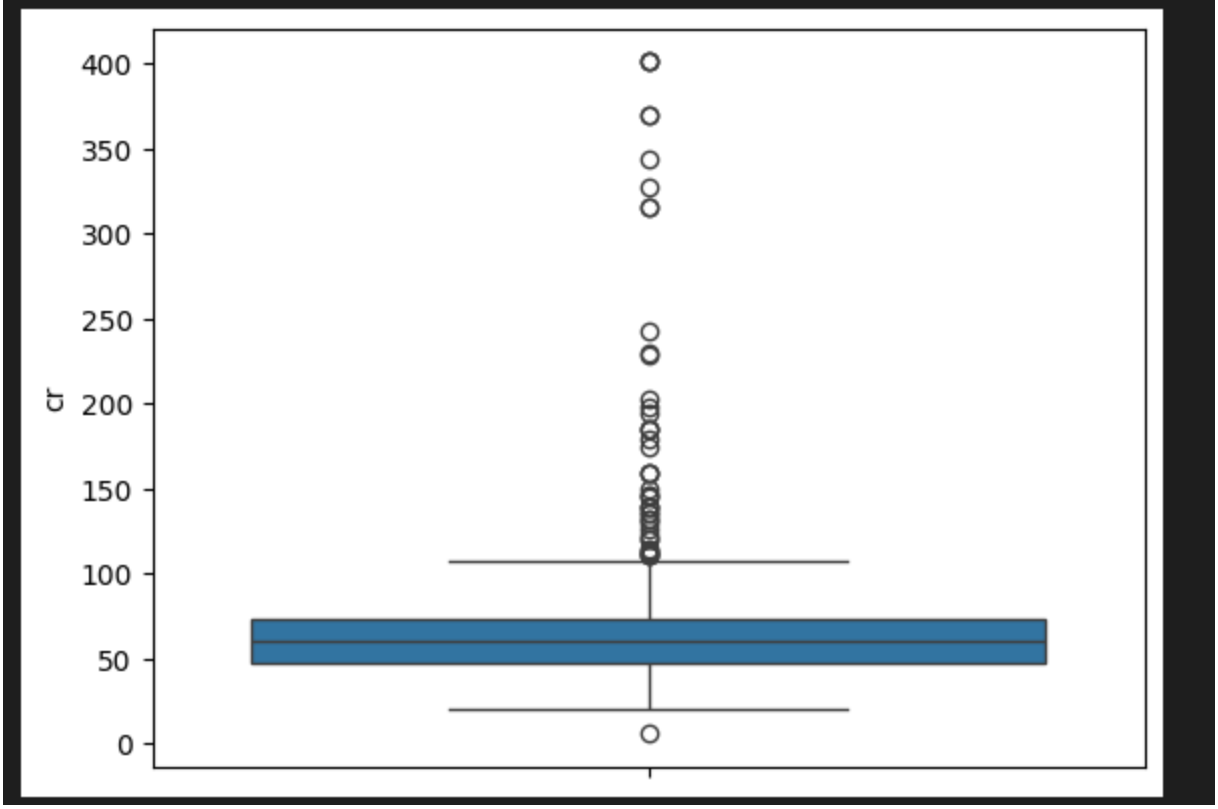
Or

df1 = df1.query('~(cr > @max_cr)')

d)Confirm outlier has been removed

```
#confirm outlier removal
sns.boxplot(df1,y='cr') #outlier removed
```

```
<Axes: ylabel='cr'>
```



83.Remove outlier using IQR

```
### remove oulier using   iqr
#calculate IQR
q1 = df2['cr'].quantile(0.25)
print(q1)
q3 = df2['cr'].quantile(0.75)
print(q3)
iqr =  q3-q1
iqr
```

```
48.0
73.0

25.0
```

```
#define lower and upper bounds for outliers
lower_bound = q1 - 1.5 *iqr
print(lower_bound)
upper_bound = q3 + 1.5 * iqr
upper_bound
```

```
10.5

110.5
```

```
# Filter the data to remove outliers
df2[(df2['cr'] >= lower_bound) & (df2['cr'] <= upper_bound)]
```

	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
0	17975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
1	34221	M	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00	N
2	47975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
3	87656	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
4	34223	M	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...
997	87654	M	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40	Y

Use query

df2 = df2.query('cr >= @lower_bound & cr <=@upper_bound')

df2

```
df2 = df2.query('cr >= @lower_bound & cr <=@upper_bound')
df2
```

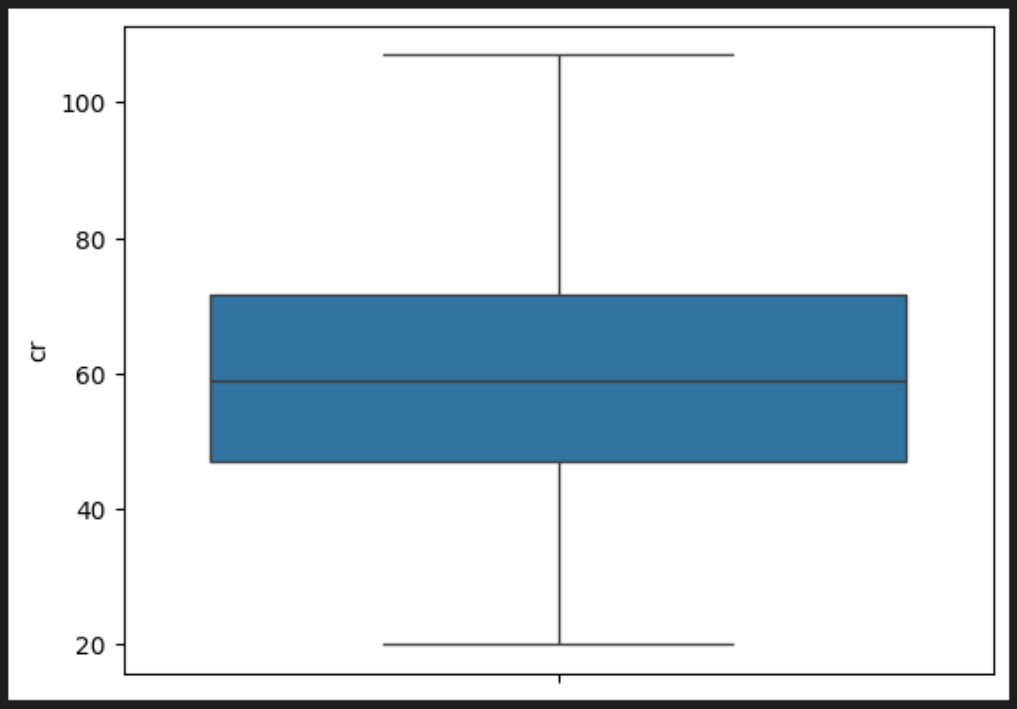
	patient_no	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	class
0	17975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
1	34221	M	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00	N
2	47975	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
3	87656	F	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00	N
4	34223	M	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...
997	87654	M	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40	Y
998	24004	M	38.00	5.80	59.00	6.70	5.30	2.00	1.60	2.90	14.00	40.50	Y
1000	454316	M	64.00	8.80	106.00	8.50	5.90	2.10	1.20	4.00	1.20	32.00	Y
1005	454316	M	55.00	4.80	88.00	8.00	5.70	4.00	0.90	3.30	1.80	30.00	Y
1007	454316	F	57.00	4.10	70.00	9.30	5.30	3.30	1.00	1.40	1.30	29.00	Y

943 rows × 13 columns

b)Check to see if outlier removed

```
#check boxplot again
sns.boxplot(df2, y='cr')
```

```
<Axes: ylabel='cr'>
```



84.EDA(Univariate analysis)

- ❖ **VISUALIZATION**

a)check missing values

data.isna().sum().any()

b)do a copy to a different dataframe

diabetes = data.copy()

diabetes

c)Check distribution of categorical variables we use countplot,piechart or barchart

for continuous we use histogram,kde(distplot)

84. a)Check gender variable

1)counplot(seaborn)

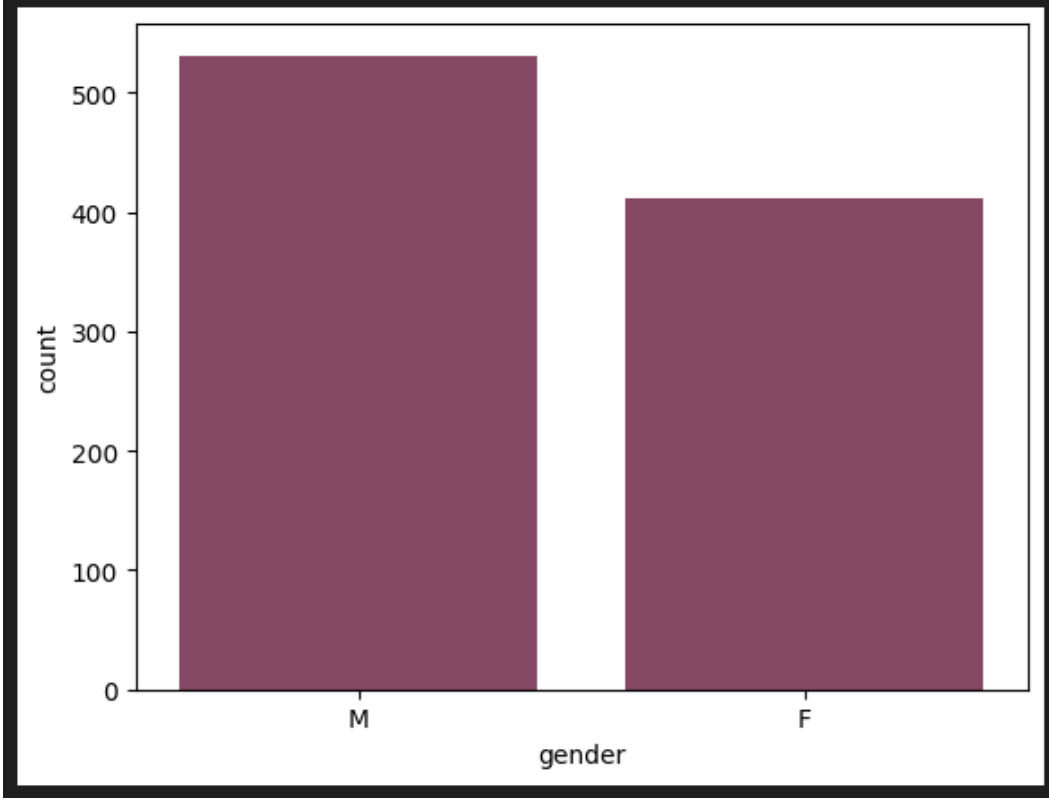
gender_counts = diabetes.gender.value_counts()

sns.countplot(diabetes, x='gender',order=gender_counts.index)

#sns.countplot(x=data['gender'], order=gender_counts.index, color='#8E3E63')

```
#using seaborn
sns.countplot(diabetes, x='gender',order=gender_counts.index)
#sns.countplot(x=data['gender'], order=gender_counts.index, color='#8E3E63')
```

```
<Axes: xlabel='gender', ylabel='count'>
```

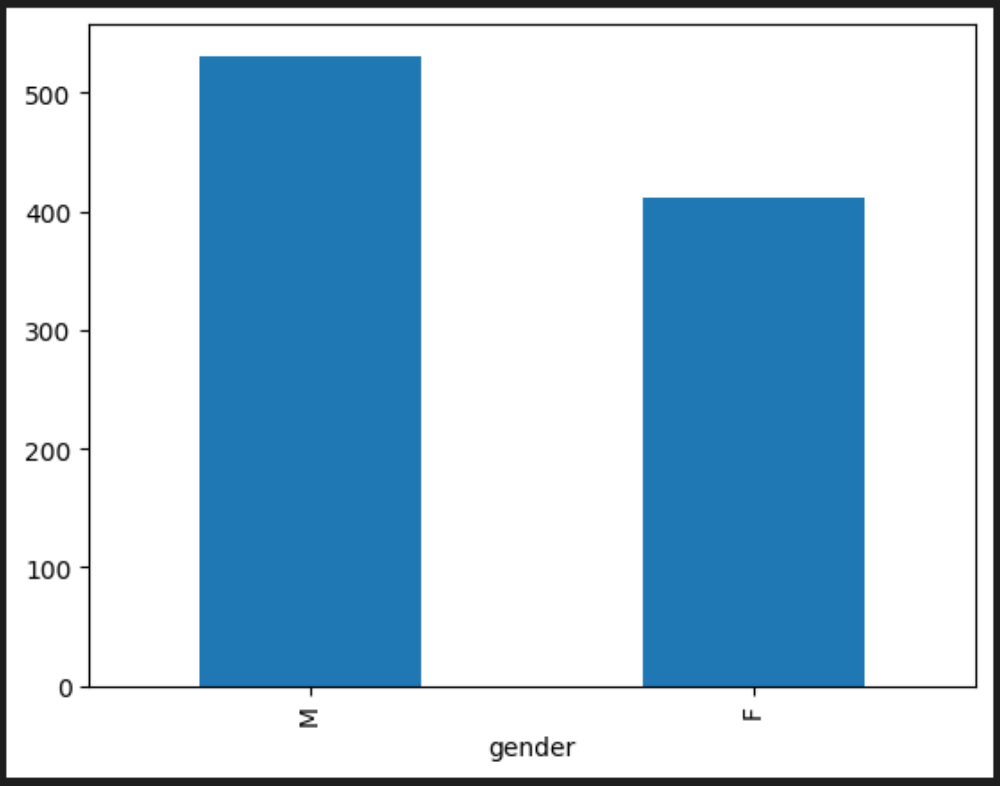


Matplotlib

diabetes.gender.value_counts().plot(kind='bar')

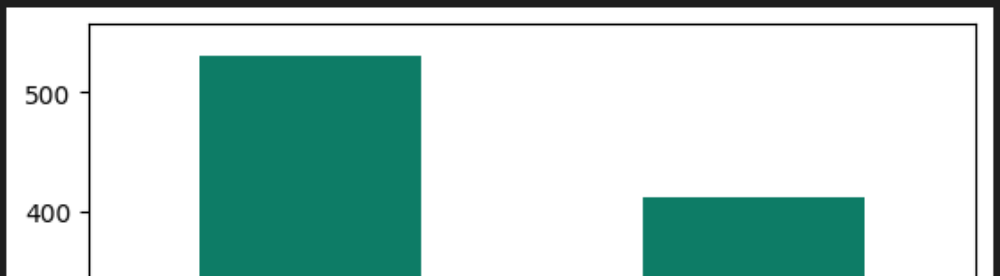
```
diabetes.gender.value_counts().plot(kind='bar')
```

```
<Axes: xlabel='gender'>
```




```
diabetes.gender.value_counts().plot(kind='bar',color='#0D7C66')
```

```
<Axes: xlabel='gender'>
```



Or

gender_counts = diabetes.gender.value_counts()

gender_counts.plot(kind='bar')

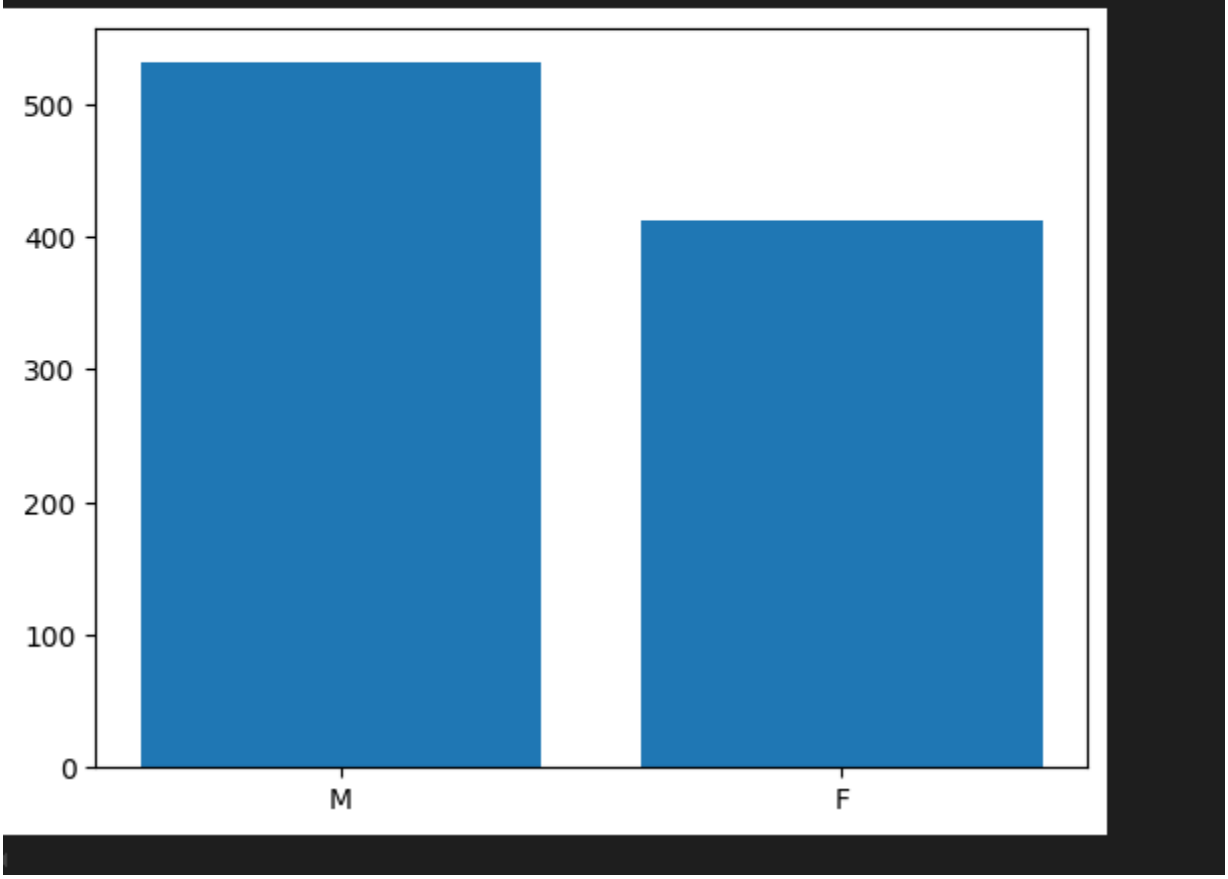
2)Barplot

<u>Matplotlib</u>

plt.bar(gender_counts.index,gender_counts.values)

```
#barplot
#using matplotlib
plt.bar(gender_counts.index,gender_counts.values)
```

```
BarContainer object of 2 artists>
```

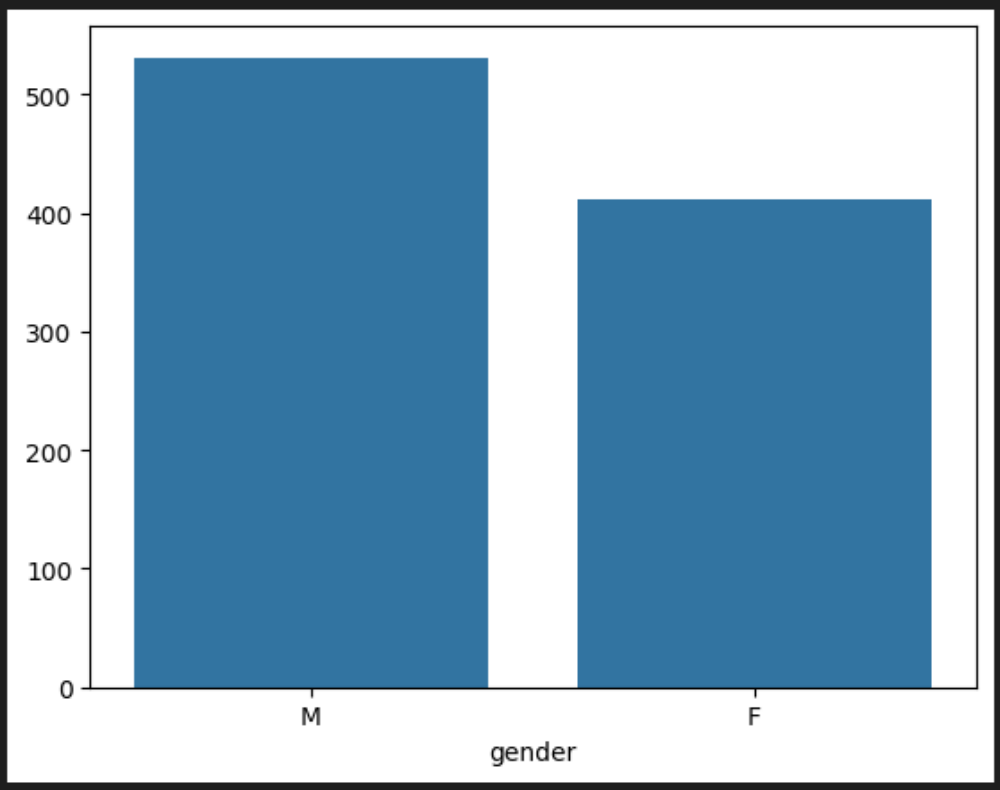


using seaborns

sns.barplot(x=gender_counts.index,y=gender_counts.values)

```
##barplot
#using seaborns
sns.barplot(x=gender_counts.index,y=gender_counts.values)
```

```
<Axes: xlabel='gender'>
```




```
gender_counts.values
```

```
array([531, 412])
```

Observation:males are more

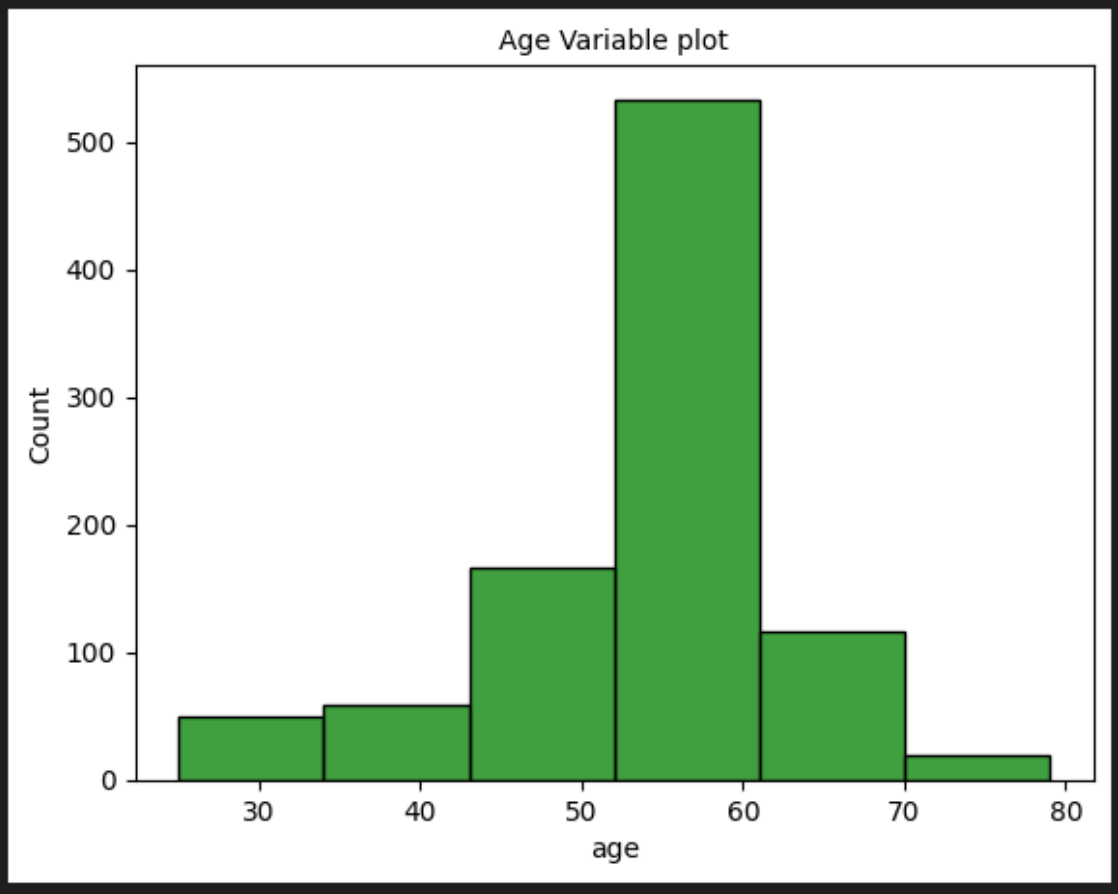
b)Check age variable

i)histplot

sns.histplot(diabetes['age'],bins=6,color='g')

plt.title('Age Variable plot', fontsize=10);

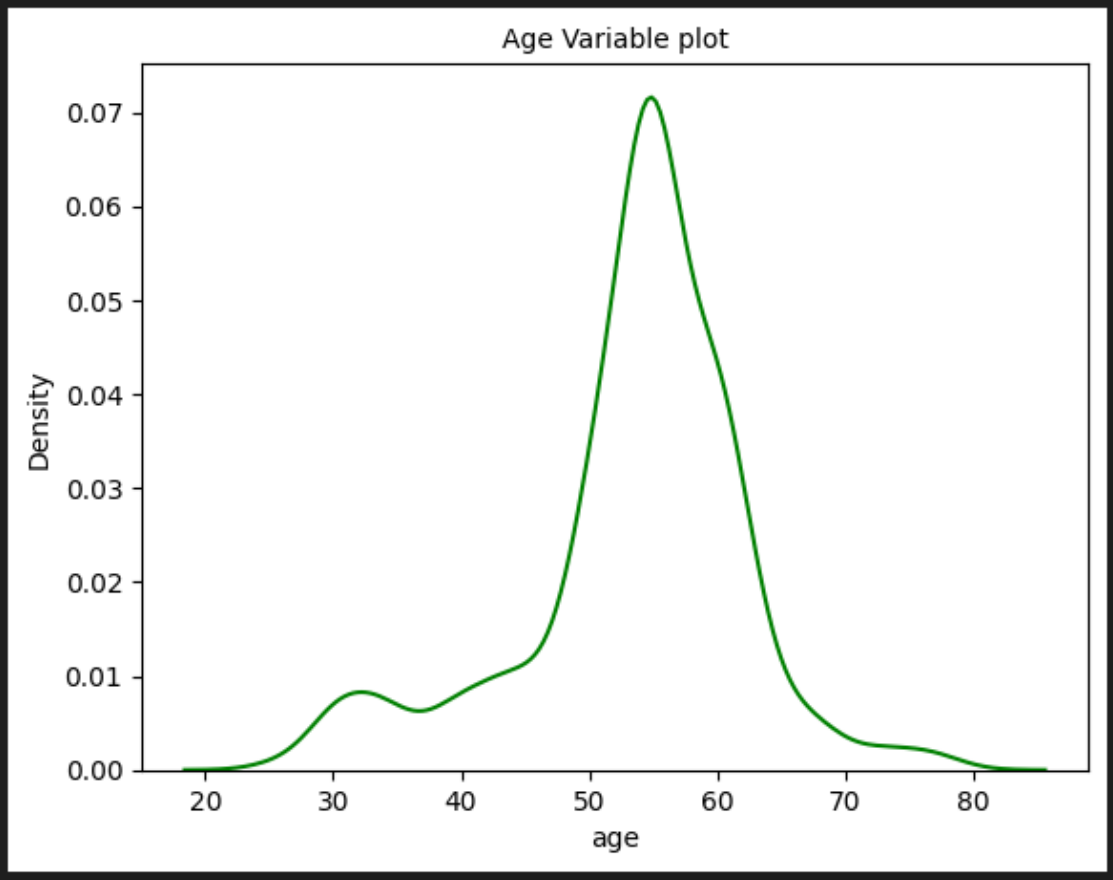
```
sns.histplot(diabetes['age'],bins=6,color='g')
plt.title('Age Variable plot', fontsize=10);
```



ii) sns.kdeplot(diabetes['age'],color='g')

plt.title('Age Variable plot', fontsize=10);

```
sns.kdeplot(diabetes['age'],color='g')
plt.title('Age Variable plot', fontsize=10);
```

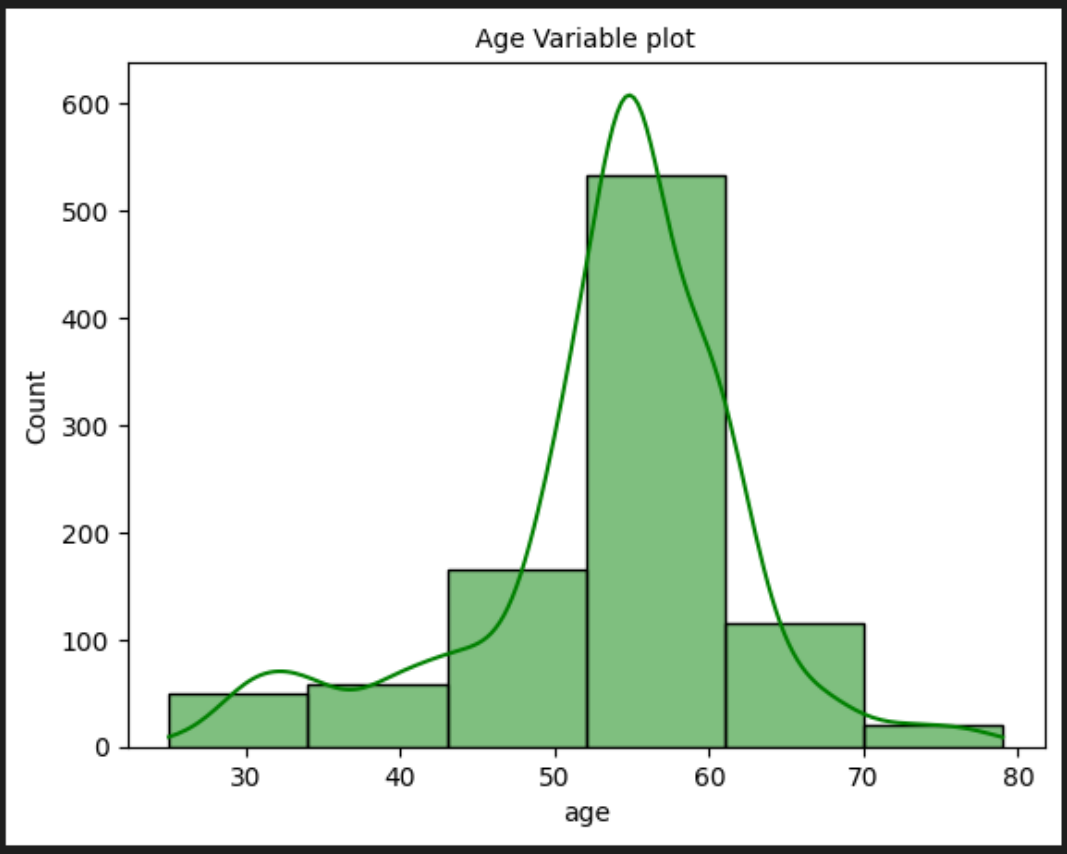


iii)Combine the two

sns.histplot(diabetes['age'],color='g',bins=6,kde=True)

plt.title('Age Variable plot', fontsize=10);

```
# combine the two
sns.histplot(diabetes['age'],color='g',bins=6,kde=True)
plt.title('Age Variable plot', fontsize=10);
```




```
diabetes['age'].skew()
```

```
-0.8158667725750883
```

Observation:Age between 50 and 60

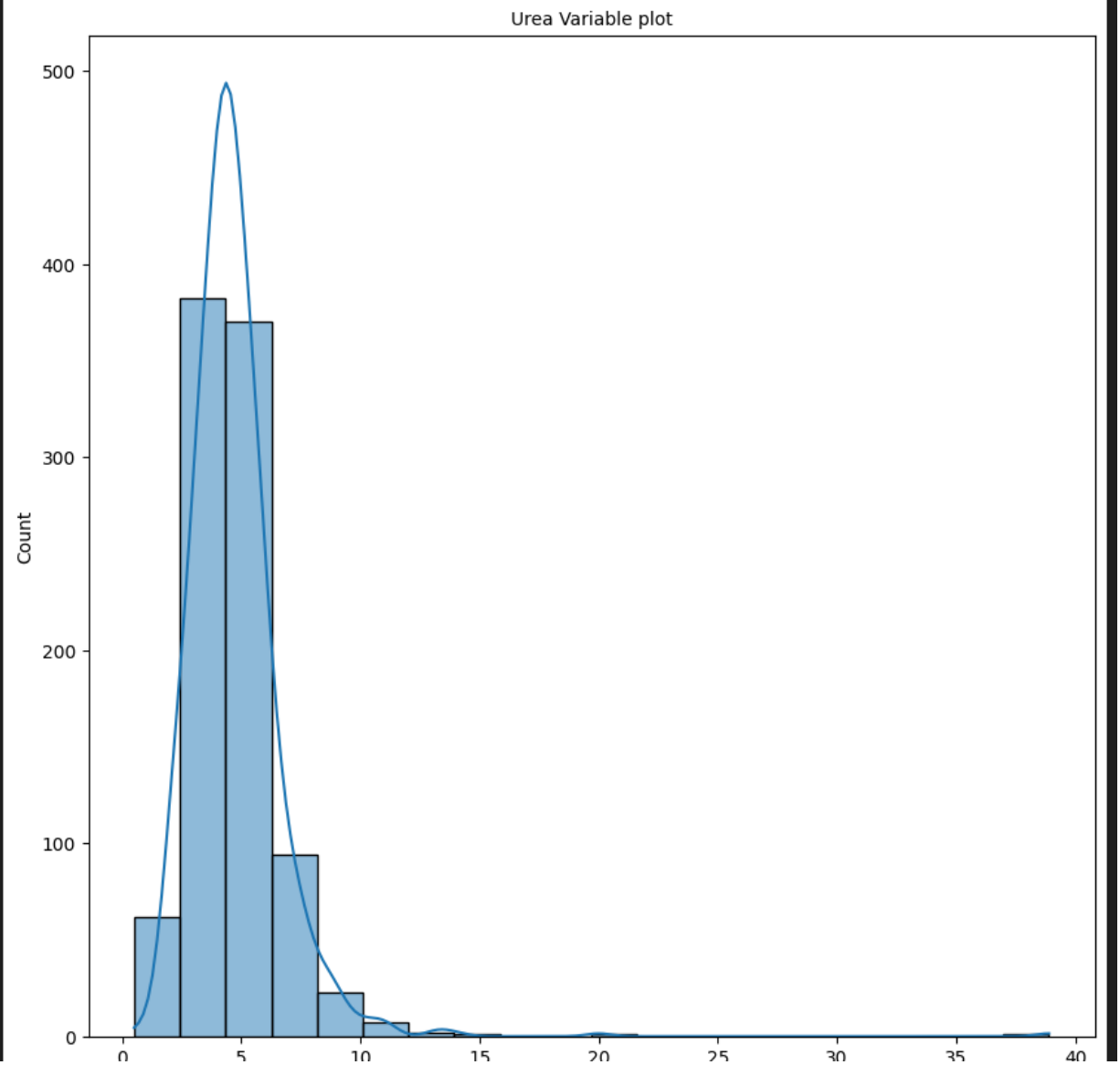
c)Check urea variable

plt.figure(figsize=(10,10))

sns.histplot(diabetes['urea'],bins=20,kde=True)

plt.title('Urea Variable plot', fontsize=10);

```
sns.histplot(diabetes['urea'],bins=20,kde=True)
plt.title('Urea Variable plot', fontsize=10);
```




```
diabetes['urea'].skew()
```

```
5.856406539153679
```

Observation:

d)Check class variable

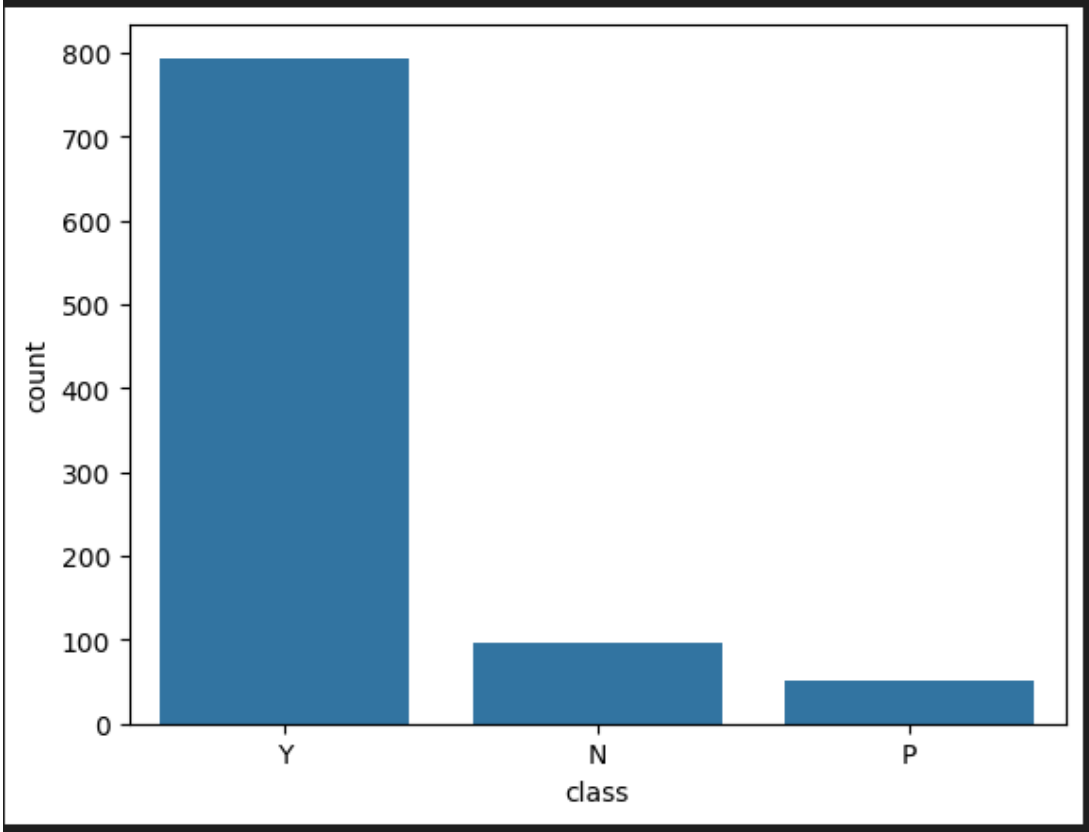
i)use count plot

class_value =diabetes['class'].value_counts().index

sns.countplot(x=diabetes['class'],order=class_value)

```
class_value =diabetes['class'].value_counts().index
sns.countplot(x=diabetes['class'],order=class_value)
```

```
Axes: xlabel='class', ylabel='count'>
```



ii)Use pie chart

diabetes_count = diabetes['class'].value_counts()

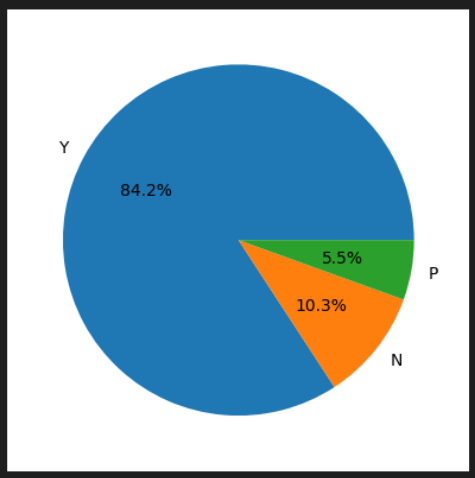
plt.pie(diabetes_count,labels=diabetes_count.index,autopct='%1.1f%%');

```
###use pie chart
diabetes_count = diabetes['class'].value_counts()
diabetes_count
```

class	count
Y	794
N	97
P	52

dtype: int64

```
plt.pie(diabetes_count,labels=diabetes_count.index,autopct='%1.1f%%');
#plt.pie(diabetis_count, labels=diabetis_count.index, autopct='%1.1f%%', startangle=90, explode=[0.1,0.0,0.0])
```

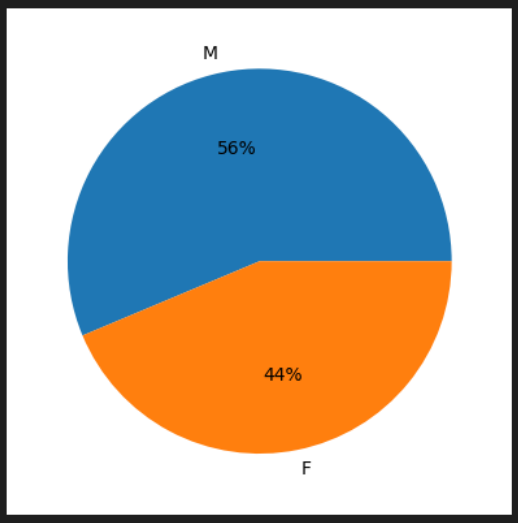


Observation:Y class is more

Pie chart for gender

plt.pie(gender_counts,labels=gender_counts.index,autopct='%1.0f%%');

```
plt.pie(gender_counts,labels=gender_counts.index,autopct='%1.0f%%');
```



❖ **Summary statistics**

```
diabetes.mean(numeric_only=True)
```

```
diabetes.mean(numeric_only=True)
```

	0
patient_no	284192.66
age	53.52
urea	4.72
cr	59.67
hba1c	8.29
chol	4.87
tg	2.33
hdl	1.21
ldl	2.61
vldl	1.89
bmi	29.55

dtype: float64

```
diabetes.columns
```

```
Index(['patient_no', 'gender', 'age', 'urea', 'cr', 'hba1c', 'chol', 'tg',
       'hdl', 'ldl', 'vldl', 'bmi', 'class'],
      dtype='object')
```

Do for all

df_num = diabetes.select_dtypes(['int','float']) # diabetes.select_dtypes([int,float])

df_num

df_num.agg(['mean','median','skew','kurtosis'])

```
df_num = diabetes.select_dtypes(['int','float']) # diabetes.select_dtypes([int,float])
df_num
```

	patient_no	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi
0	17975	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
1	34221	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00
2	47975	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
3	87656	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
4	34223	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00
...	...	...	...	...	...	...	...	...	...	...	...
938	87654	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40
939	24004	38.00	5.80	59.00	6.70	5.30	2.00	1.60	2.90	14.00	40.50
940	454316	64.00	8.80	106.00	8.50	5.90	2.10	1.20	4.00	1.20	32.00
941	454316	55.00	4.80	88.00	8.00	5.70	4.00	0.90	3.30	1.80	30.00
942	454316	57.00	4.10	70.00	9.30	5.30	3.30	1.00	1.40	1.30	29.00

43 rows × 11 columns

```
df_num.agg(['mean','median','skew','kurtosis'])
```

	patient_no	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi
mean	284192.66	53.52	4.72	59.67	8.29	4.87	2.33	1.21	2.61	1.89	29.55
median	34387.00	55.00	4.50	59.00	8.00	4.80	2.00	1.10	2.50	0.90	30.00
skew	18.99	-0.82	5.86	0.30	0.22	0.64	2.35	6.34	1.18	5.21	0.16
kurtosis	377.78	1.40	83.83	-0.23	-0.29	1.89	10.99	62.35	4.32	31.59	-0.25

85.EDA(Bivariate analysis)

Between categorical variables- counplot and add second variable as hue

Barchart, you can first do crosstab

Between categorical and numeric

Barchart

Beween numeric

Scatterplot, do correlation diagram using heatmap

a)Compare gender and class

<u>using barchart</u>

gender_class = pd.crosstab(diabetes['gender'],diabetes['class'])

print(gender_class)

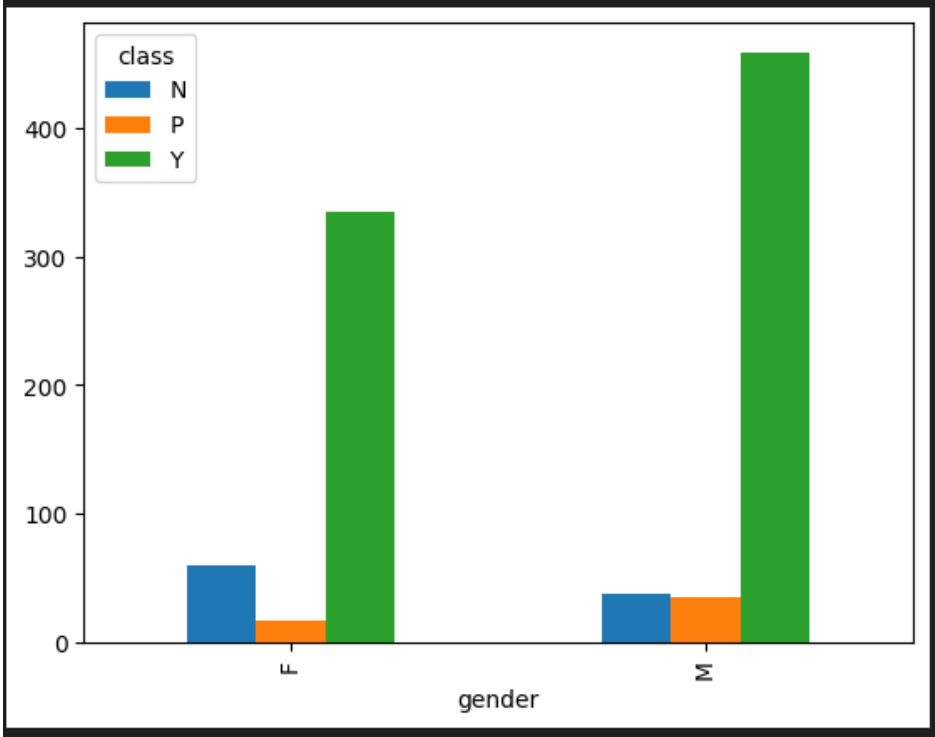
gender_class.plot(kind='bar')

```
#compare gender and class
gender_class = pd.crosstab(diabetes['gender'],diabetes['class'])
print(gender_class)
```

```
lass    N   P    Y
ender
       60  17  335
       37  35  459
```

```
gender_class.plot(kind='bar')
```

```
Axes: xlabel='gender'>
```



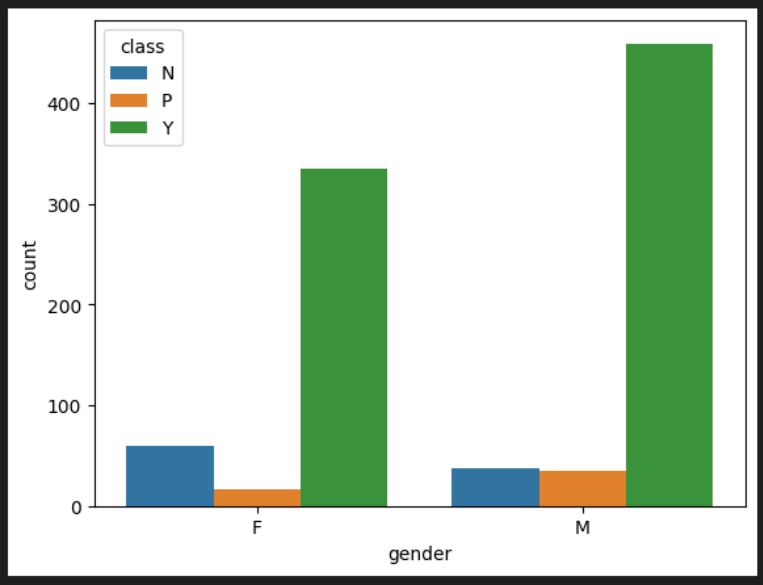
Using countplot and add hue

sns.countplot(diabetes, x='gender',hue='class')

#sns.countplot(x=diabetes['gender'], order=gender_counts.index, color='#8E3E63')

```
## use countplot and add hue
sns.countplot(diabetes, x='gender',hue='class')
#sns.countplot(x=diabetes['gender'], order=gender_counts.index, color='#8E3E63')
```

```
<Axes: xlabel='gender', ylabel='count'>
```



Observation:

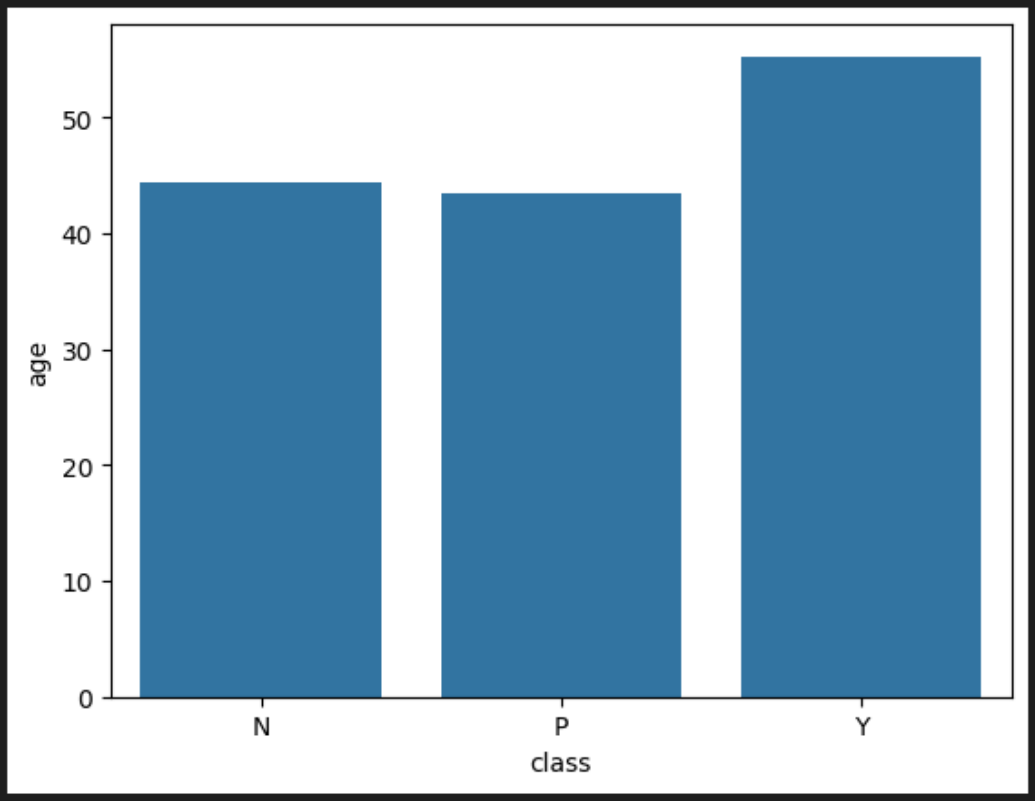
b) **age vs class**(numeric and categorical)#barchart

seaborn

sns.barplot(data=diabetes, y='age',x='class',errorbar=None)

```
##age vs class(numeric and categorical)#barchart
sns.barplot(data=diabetes, y='age',x='class',errorbar=None)
```

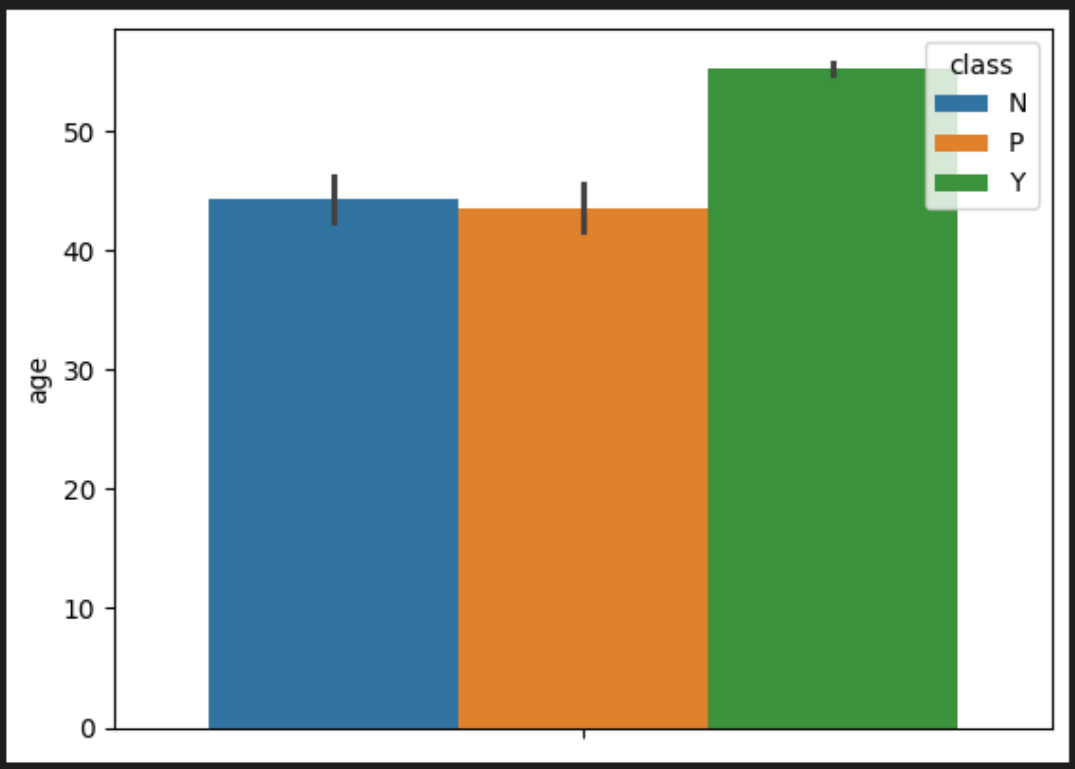
```
<Axes: xlabel='class', ylabel='age'>
```



sns.barplot(y=diabetes['age'], hue = diabetes['class']);

```
sns.barplot(y=diabetes['age'], hue = diabetes['class']);
```

```
<Axes: ylabel='age'>
```

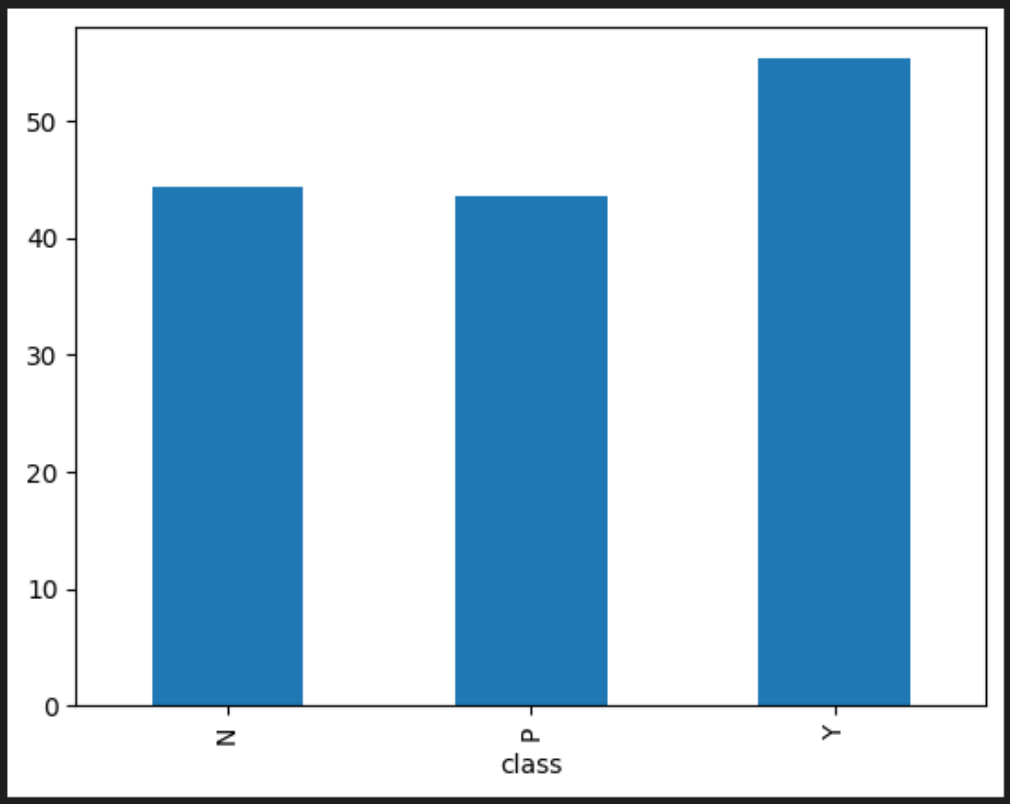


matplotlib

diabetes.groupby('class')['age'].mean().plot(kind='bar')

```
###matplotlib
diabetes.groupby('class')['age'].mean().plot(kind='bar')
```

```
<Axes: xlabel='class'>
```

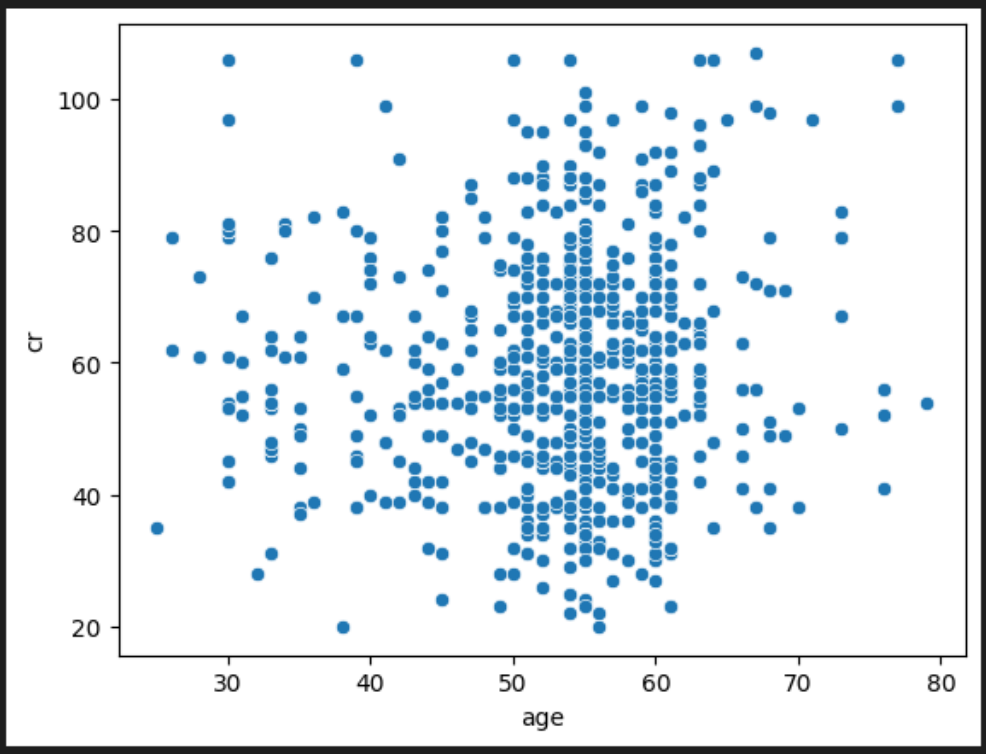


c) age vs cr (continous var) scatterplot

sns.scatterplot(data=diabetes,x='age',y='cr')

```
### age vs cr (continous var) scatterplot
sns.scatterplot(data=diabetes,x='age',y='cr')
```

```
<Axes: xlabel='age', ylabel='cr'>
```




```
np.corrcoef(diabetes['age'],diabetes['cr'])[0][1]
```

```
0.057700634886981884
```

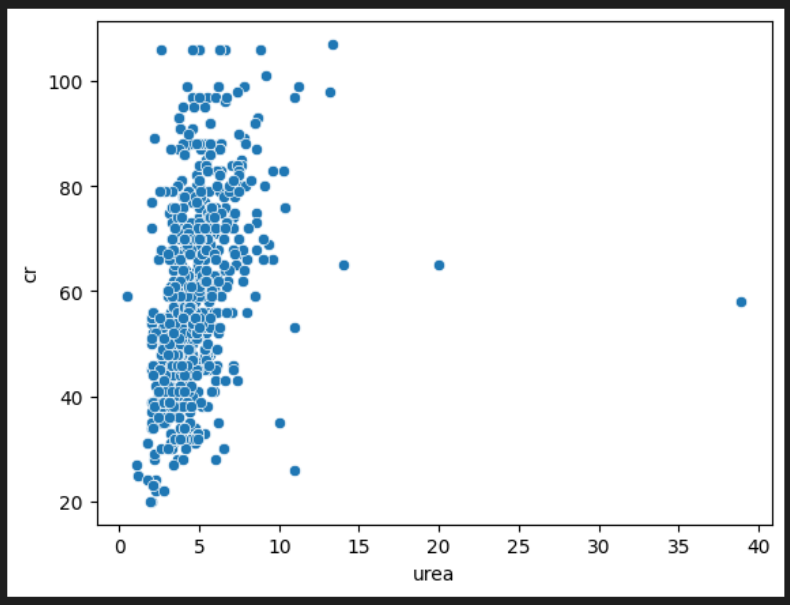
d) urea and cr

sns.scatterplot(data=diabetes,x='urea',y='cr')

np.corrcoef(diabetes['urea'],diabetes['cr'])[0][1]

```
### urea and cr
sns.scatterplot(data=diabetes,x='urea',y='cr')
```

```
<Axes: xlabel='urea', ylabel='cr'>
```



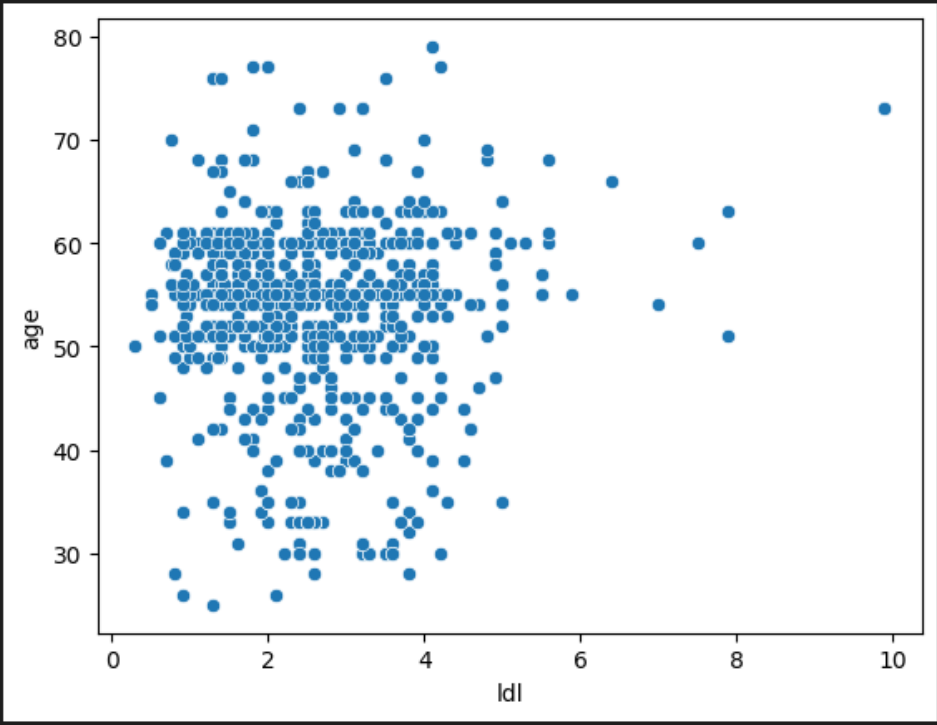

```
np.corrcoef(diabetes['urea'],diabetes['cr'])[0][1]
```

```
0.39380331568274707
```

e)ldl vs age

```
### ldl and age
sns.scatterplot(data=diabetes,x='ldl',y='age')
```

```
<Axes: xlabel='ldl', ylabel='age'>
```




```
np.corrcoef(diabetes['ldl'],diabetes['age'])[0][1]
```

```
0.017925105000133855
```

f)do all the corr at once

first select the columns

diabetes.select_dtypes(['int','float'])

diabetes.select_dtypes(['int','float']).columns

data_int = diabetes[['age', 'urea', 'cr', 'hba1c', 'chol', 'tg', 'hdl', 'ldl',
'vldl', 'bmi']]

data_int

```
diabetes.select_dtypes(['int','float'])
```

	patient_no	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi
0	17975	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
1	34221	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00
2	47975	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
3	87656	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
4	34223	33.00	7.10	46.00	4.90	4.90	1.00	0.80	2.00	0.40	21.00
...	...	...	...	...	...	...	...	...	...	...	...
938	87654	30.00	7.10	81.00	6.70	4.10	1.10	1.20	2.40	8.10	27.40
939	24004	38.00	5.80	59.00	6.70	5.30	2.00	1.60	2.90	14.00	40.50
940	454316	64.00	8.80	106.00	8.50	5.90	2.10	1.20	4.00	1.20	32.00
941	454316	55.00	4.80	88.00	8.00	5.70	4.00	0.90	3.30	1.80	30.00
942	454316	57.00	4.10	70.00	9.30	5.30	3.30	1.00	1.40	1.30	29.00

943 rows × 11 columns

```
diabetes.select_dtypes(['int','float']).columns
```

```
Index(['patient_no', 'age', 'urea', 'cr', 'hba1c', 'chol', 'tg', 'hdl', 'ldl',
       'vldl', 'bmi'],
      dtype='object')
```

```
data_int = diabetes[['age', 'urea', 'cr', 'hba1c', 'chol', 'tg', 'hdl', 'ldl',
       'vldl', 'bmi']]
data_int
```

	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi
0	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
1	26.00	4.50	62.00	4.90	3.70	1.40	1.10	2.10	0.60	23.00
2	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00
3	50.00	4.70	46.00	4.90	4.20	0.90	2.40	1.40	0.50	24.00

Then do corr

corr = data_int.corr()

corr

```
corr = data_int.corr()
corr
```

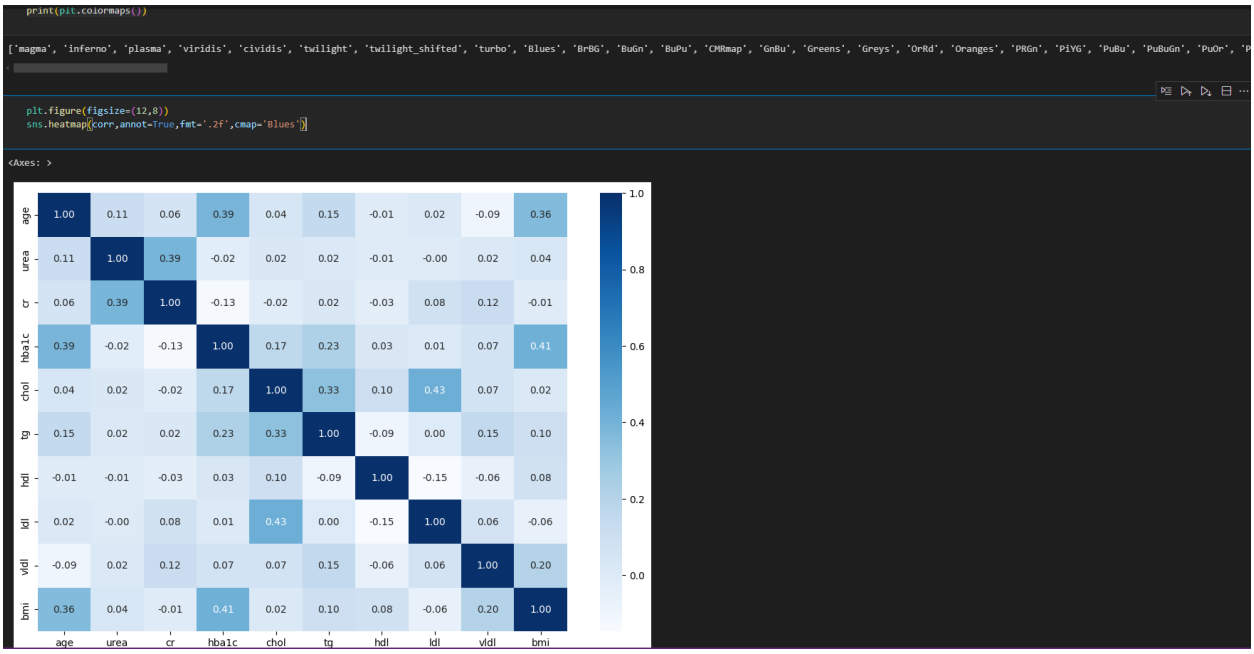
	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi
age	1.00	0.11	0.06	0.39	0.04	0.15	-0.01	0.02	-0.09	0.36
urea	0.11	1.00	0.39	-0.02	0.02	0.02	-0.01	-0.00	0.02	0.04
cr	0.06	0.39	1.00	-0.13	-0.02	0.02	-0.03	0.08	0.12	-0.01
hba1c	0.39	-0.02	-0.13	1.00	0.17	0.23	0.03	0.01	0.07	0.41
chol	0.04	0.02	-0.02	0.17	1.00	0.33	0.10	0.43	0.07	0.02
tg	0.15	0.02	0.02	0.23	0.33	1.00	-0.09	0.00	0.15	0.10
hdl	-0.01	-0.01	-0.03	0.03	0.10	-0.09	1.00	-0.15	-0.06	0.08
ldl	0.02	-0.00	0.08	0.01	0.43	0.00	-0.15	1.00	0.06	-0.06
vldl	-0.09	0.02	0.12	0.07	0.07	0.15	-0.06	0.06	1.00	0.20
bmi	0.36	0.04	-0.01	0.41	0.02	0.10	0.08	-0.06	0.20	1.00

Do heat map for easy visualization

print(plt.colormaps())

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

sns.heatmap(corr,annot=True,fmt='.2f',cmap='Blues')



86 .EDA(Multivariate analysis)

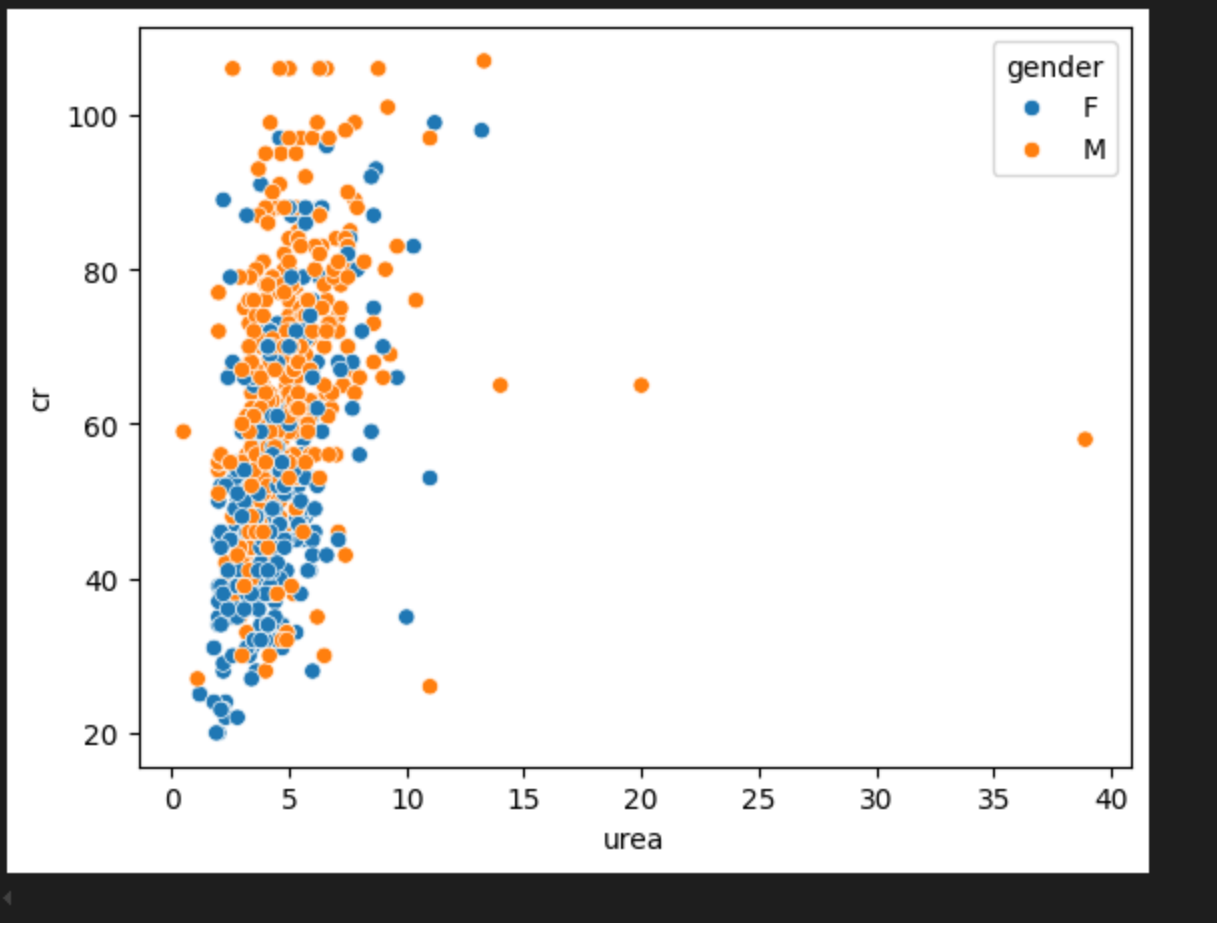
Multivariate analysis is a statistical technique used to describe and summarize the relationship between three or more variables

a)urea,cr,gender

sns.scatterplot(data=diabetes,x='urea',y='cr',hue='gender')

```
#urea,cr,gender
sns.scatterplot(data=diabetes,x='urea',y='cr',hue='gender')
```

```
<Axes: xlabel='urea', ylabel='cr'>
```

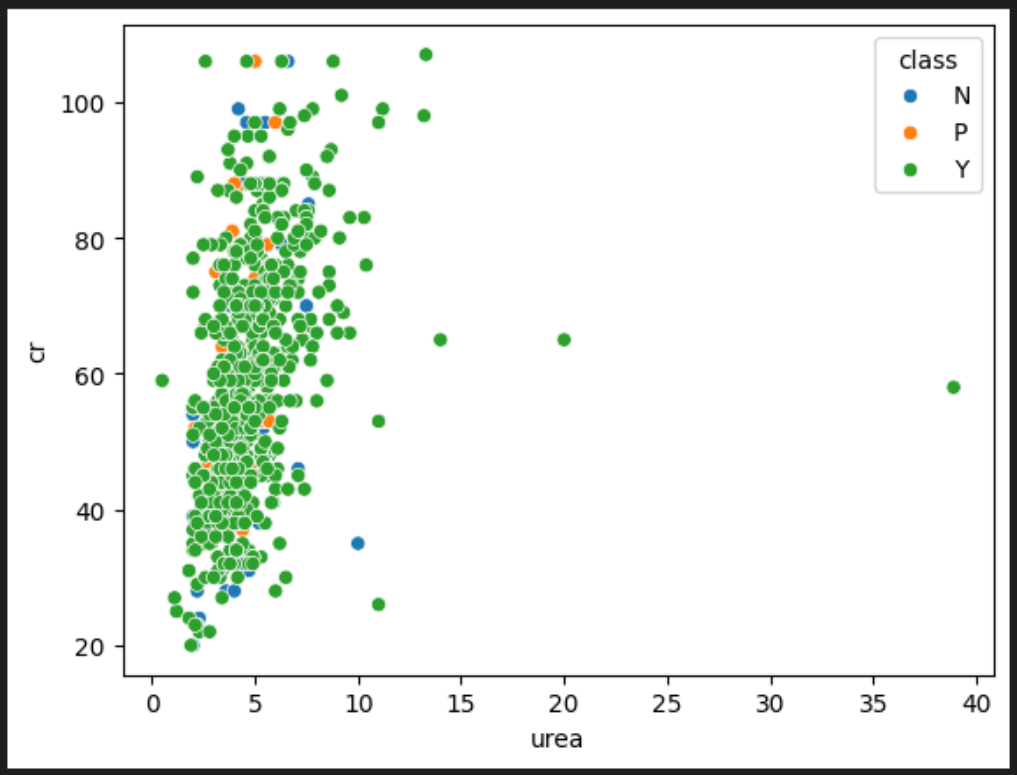


b) urea,cr,class

sns.scatterplot(data=diabetes,x='urea',y='cr',hue='class')

```
#urea,cr,class
sns.scatterplot(data=diabetes,x='urea',y='cr',hue='class')
```

```
<Axes: xlabel='urea', ylabel='cr'>
```



87. Remove columns/drop columns(We won't need it during Analysis. No question to be answered requires that column.)

df1.drop('Location_1',axis=1)

df1.drop(['Location_1','OBJECTID'],axis=1,inplace=True)

88.Update columns

df1.columns= df1.columns.str.strip().str.lower()

89.Remove (%) and make float(getting rid of % sign and coverting the datatype Explanation:for easy manupulation and analysis**)**

df1[['missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',

'dumb', 'mental', 'paralyzed', 'other']] = df1[['missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',

'dumb', 'mental', 'paralyzed', 'other']].apply(lambda x: x.str.strip('%').astype(float)/100)

	county	missing_hand	missing_foot	lame	blind	deaf	dumb	mental	paralyzed	other	total_count
0	Baringo	10.9%	2.8%	30.9%	2.6%	1%	3.8%	12.4%	54.2%	10.9%	6512.1
1	Bomet	0%	0%	29.2%	10%	8.1%	8.1%	14.2%	0.3%	38.7%	6538.0
2	Bungoma	12.3%	0%	49.2%	0%	2.3%	20.8%	12.7%	0%	21.9%	13170.6
3	Busia	0%	0%	14.3%	12.3%	4.6%	35.5%	4.8%	14.8%	31.4%	6655.5
4	Elgeyo Marakwet	0%	0%	31.7%	0%	35.7%	0%	20%	12.6%	7.4%	3599.9

```
# Procedure 4:
# Data Cleaning Action:getting rid of % sign and coverting the datatype
# Explanation:for easy manupulation and analysis
df1.columns
```

```
Index(['county', 'missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',
       'dumb', 'mental', 'paralyzed', 'other', 'total_count'],
      dtype='object')
```

```
df1[['missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',
       'dumb', 'mental', 'paralyzed', 'other']] = df1[['missing_hand', 'missing_foot', 'lame', 'blind', 'deaf',
       'dumb', 'mental', 'paralyzed', 'other']].apply(lambda x: x.str.strip('%').astype(float)/100)
```

```
df1.head()
```

	county	missing_hand	missing_foot	lame	blind	deaf	dumb	mental	paralyzed	other	total_count
0	Baringo	0.109	0.028	0.309	0.026	0.010	0.038	0.124	0.542	0.109	6512.1
1	Bomet	0.000	0.000	0.292	0.100	0.081	0.081	0.142	0.003	0.387	6538.0
2	Bungoma	0.123	0.000	0.492	0.000	0.023	0.208	0.127	0.000	0.219	13170.6
3	Busia	0.000	0.000	0.143	0.123	0.046	0.355	0.048	0.148	0.314	6655.5
4	Elgeyo Marakwet	0.000	0.000	0.317	0.000	0.357	0.000	0.200	0.126	0.074	3599.9

90.Remove outliers for all numeric

Q1 = df1.quantile(0.25)

Q3 = df1.quantile(0.75)

IQR = Q3 - Q1

removing outliers

df2 = df1[~((df1 < (Q1 - 1.5 * IQR)) | (df1 > (Q3 + 1.5 * IQR))).any(axis=1)]

or

```
df1.query('~(@df1 < (@Q1-1.5 *IQR) | @df1 >(@Q3+ 1.5* @IQR))')
```

```
# checking old shape
print("old shape:","\n",df1.shape)
print("****"*10)
print("new shape:","\n", df2.shape)
```

b)Remove outliers for only numeric columns

```
numeric_columns = df1.select_dtypes(include=['int','float']).columns
Q1= df1[numeric_columns].quantile(0.25)
print(f'Q1: "\n" {Q1}')
Q3 = df1[numeric_columns].quantile(0.75)
print(f'Q3: "\n"{Q3}')
IQR = Q3- Q1
IQR
df2 = df1[~((df1[numeric_columns] < (Q1 - 1.5 * IQR)) | (df1[numeric_columns] > (Q3 + 1.5 * IQR))).any(axis=1)]
#checking shape
print('old shape',"\n",df1.shape)
print('new shape',"\n",df2.shape)
```

91.make columns upper

```
df2.columns = df2.columns.str.upper()
df2.head()
#df2.columns = map(lambda x: str(x).upper(), df2.columns)
```

92.Population dataset

a)County with the highest no of registereddeaf people

1.population.groupby(['COUNTY'])['DEAF'].sum().sort_values(ascending=False)
2.population.sort_values(by='DEAF',ascending=False) #Tharaka Nithi

3. population.nlargest(1,'DEAF') #Tharaka Nithi

```
population.nlargest(1,'DEAF') #Tharaka Nithi
```
✓ 0.0s

	COUNTY	MISSING_HAND	MISSING_FOOT	LAME	BLIND	DEAF	DUMB	MENTAL	PARALYZED	OTHER	TOTAL_COUNT
19	Tharaka Nithi	0.0	0.0	0.0	0.0	0.293	0.293	0.414	0.0	0.293	420.1

```
(population.groupby('COUNTY')['DEAF'].sum().sort_values(ascending=False))[0:1] #TN
```
✓ 0.0s

```
COUNTY
Tharaka Nithi    0.293
Name: DEAF, dtype: float64
```

```
### Teachers code
county= population.groupby(["COUNTY"])['DEAF'].sum()
county.sort_values(ascending=False)[0:1]
```
✓ 0.0s

```
COUNTY
Tharaka Nithi    0.293
Name: DEAF, dtype: float64
```

b)smallest we use

county.nsmallest(1)

c) Which counties had no registerd blind persons nor deaf persons?

1. population.query('BLIND==0 & DEAF==0 ')['COUNTY']
2. population[(population['DEAF'] ==0) & (population['BLIND']==0)]["COUNTY"]

d) **Which disability was the most registered across all the counties?get sum per column**

```
b = population[['MISSING_HAND', 'MISSING_FOOT', 'LAME', 'BLIND', 'DEAF',
    'DUMB', 'MENTAL', 'PARALYZED', 'OTHER' ]].sum()
print(b.nlargest(1))
b.sort_values(ascending=False).head(1)
```

```
population.sum(numeric_only=True).sort_values(ascending=False)
```

✓ 0.0s

```
TOTAL_COUNT     233099.700
OTHER                8.364
LAME                 5.329
MENTAL               4.660
BLIND                2.540
PARALYZED            2.104
DUMB                 1.991
DEAF                 1.631
MISSING_FOOT         0.411
MISSING_HAND         0.078
dtype: float64
```

```
b = population[['MISSING_HAND', 'MISSING_FOOT', 'LAME', 'BLIND', 'DEAF',
       'DUMB', 'MENTAL', 'PARALYZED', 'OTHER' ]].sum()
print(b.nlargest(1))
b.sort_values(ascending=False).head(1)
```

✓ 0.0s

```
OTHER    8.364
dtype: float64

OTHER    8.364
dtype: float64
```

e) **Which three counties had least registered persons with disabilities?**

1.population.nsmallest(3,'TOTAL_COUNT')['COUNTY']

2.urban_population = population.groupby(['COUNTY'])["TOTAL_COUNT"].sum()

urban_population.nsmallest(3)

3. urban_population = population.groupby(['COUNTY'])["TOTAL_COUNT"].sum()

urban_population.sort_values(ascending=False).tail(3)

```
population.nsmallest(3,'TOTAL_COUNT')['COUNTY']
```

✓ 0.0s

```
19    Tharaka Nithi
8             Lamu
4          Kericho
Name: COUNTY, dtype: object
```

```
urban_population = population.groupby(['COUNTY'])["TOTAL_COUNT"].sum()
urban_population.nsmallest(3)
```

✓ 0.0s

```
COUNTY
Tharaka Nithi     420.1
Lamu              524.9
Kericho          3055.8
Name: TOTAL_COUNT, dtype: float64
```

```
urban_population = population.groupby(['COUNTY'])["TOTAL_COUNT"].sum()
urban_population.sort_values(ascending=False).tail(3)
```

✓ 0.0s

```
COUNTY
Kericho          3055.8
Lamu              524.9
Tharaka Nithi     420.1
Name: TOTAL_COUNT, dtype: float64
```

93.**a) Build a dict without errors**

plant_dict = defaultdict(int) #provides default value for a key that does not exist

for plant in plants:

  #print(plant['family_common_name'])

  name =plant['family_common_name']

  plant_dict[name] += 1

plant_dict

```
plant_dict = defaultdict(int) #provides default value for a key that does not exist
for plant in plants:
  #print(plant['family_common_name'])
  name =plant['family_common_name']
  plant_dict[name] += 1

plant_dict
```

```
defaultdict(int,
            {'Beech family': 10,
             'Nettle family': 2,
             'Grass family': 81,
             'Plantain family': 25,
             'Buttercup family': 21,
             'Pea family': 63,
             'Olive family': 5,
             'Rose family': 49,
             'Birch family': 7,
             'Rush family': 19,
             'Buckwheat family': 17,
             'Soapberry family': 7,
             'Pine family': 10,
             'Mint family': 45,
             'Madder family': 9,
             None: 113,
             'Pink family': 31,
             'Carrot family': 29
```

b)sort the dict

plant_dict_list = list(plant_dict.items())

#print(plant_dict_list)

sorted(plant_dict_list, key = lambda x: x[1])

```
plant_dict_list = list(plant_dict.items())
#print(plant_dict_list)
sorted(plant_dict_list, key = lambda x: x[1])
```

```
[('Bracken Fern family', 1),
 ('Iris family', 1),
 ('Holly family', 1),
 ('Water-plantain family', 1),
 ('Hemp family', 1),
 ('Buckbean family', 1),
 ('Ginseng family', 1),
 ('Sandalwood family', 1),
 ('Yam family', 1),
 ('Walnut family', 1),
 ('Boxwood family', 1),
 ('Melastome family', 1),
 ('Purslane family', 1),
 ('Pokeweed family', 1),
 ('Bayberry family', 1),
 ('Hornwort family', 1),
 ('Leadwort family', 1),
 ('Cucumber family', 1),
 ('Eel-grass family', 1),
 ('Flowering Rush family', 1),
 ('Spike-moss family', 1),
 ('Barberry family', 1),
```

c)Dictionary comprehension

plants_clean_above_ten = {key:value for key, value in plants_clean.items() if value>=10}

#use for loop for above

plants_clean_above_ten = {}

for key, value in plants_clean.items():

  if value >=10:

    plants_clean_above_ten[key]= value

plants_clean_above_ten

```
plants_clean_above_ten = {}
for key, value in plants_clean.items():
  if value >=10:
    plants_clean_above_ten[key]= value
plants_clean_above_ten
```

```
{'Beech family': 10,
 'Grass family': 81,
 'Plantain family': 25,
 'Buttercup family': 21,
 'Pea family': 63,
 'Rose family': 49,
 'Rush family': 19,
 'Buckwheat family': 17,
 'Pine family': 10,
 'Mint family': 45,
 'Pink family': 31,
 'Carrot family': 29,
 'Geranium family': 10,
 'Aster family': 80,
 'Heath family': 17,
 'Mustard family': 29,
 'Evening Primrose family': 10,
 'Sedge family': 41,
 'Borage family': 13,
 'Primrose family': 12,
 'Spurge family': 10,
 'Broom-rape family': 10,
 'Tape-grass family': 10}
```

```
len(plants_clean_above_ten)
```

```
23
```

```
[109] #use dictionary comprehension
plants_clean_above_ten = {key:value for key, value in plants_clean.items() if value>=10}
plants_clean_above_ten
```

```
{'Beech family': 10,
 'Grass family': 81,
 'Plantain family': 25,
 'Buttercup family': 21,
 'Pea family': 63,
```

94. There are two general strategies for dealing with missing values:

1. Fill in missing values (either using another value from the column, e.g. the mean or mode, or using some other value like "Unknown")

2. Drop rows with missing values

95. *What is the distribution of superheroes by publisher*

Since publisher is a categorical column just does do value counts

96.Rotate x axis for subplots

ax1.tick_params(axis="x", labelrotation=90)

ax2.tick_params(axis="x", labelrotation=45)

96.If asked the number of players in each country or number of superhoeros by publisher (**What is the distribution of superheroes by publisher**?)(**Create a list `top_10_countries` containing the names of the 10 countries with the most players (using the `'Nationality'` column)**

)just do a value count of country and publisher respectively

df['Nationality'].value_counts()

heroes_df['Publisher'].value_counts()