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

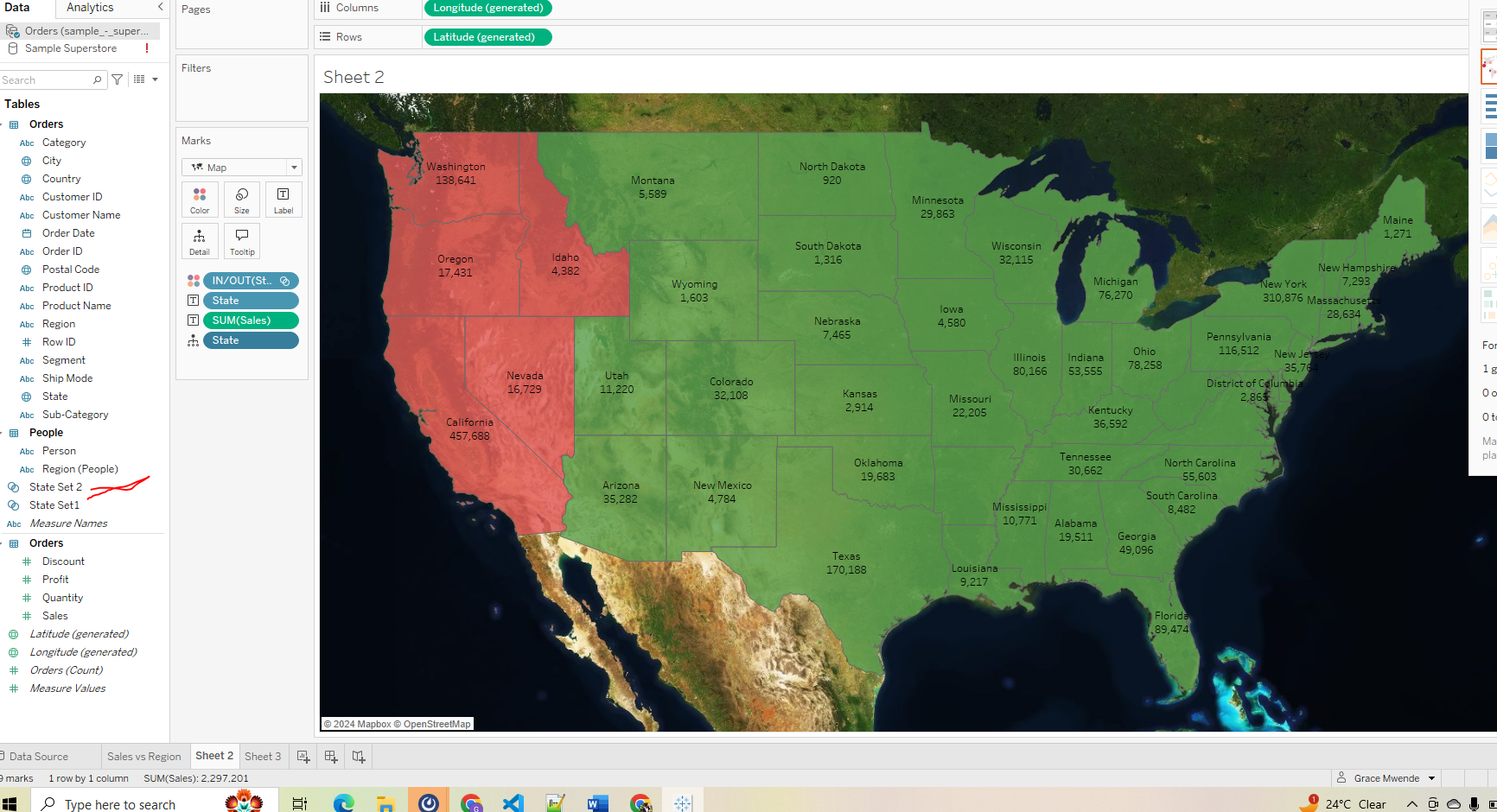
20.xtiks rotation

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**

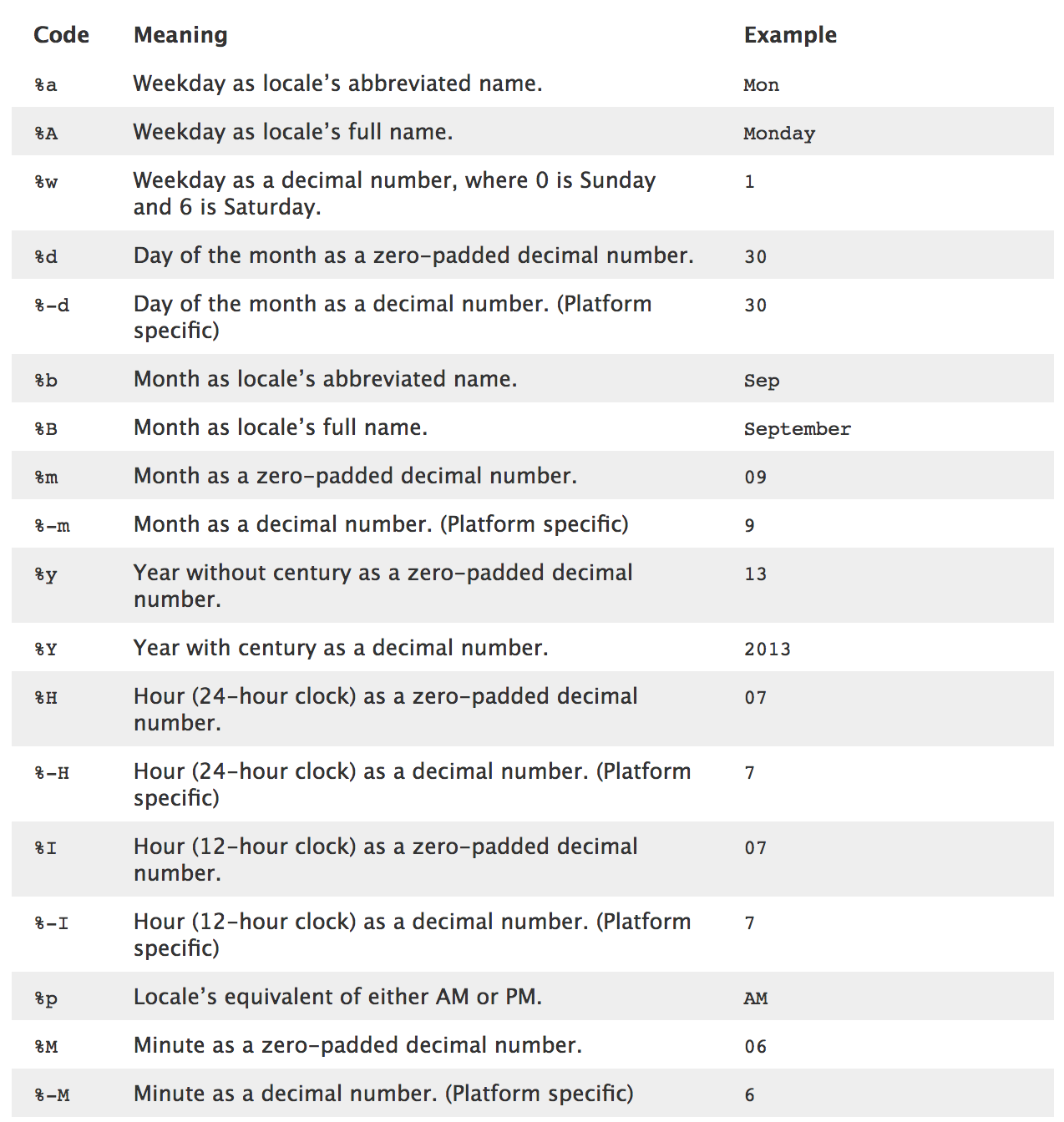
df.set\_index('linename')

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')



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

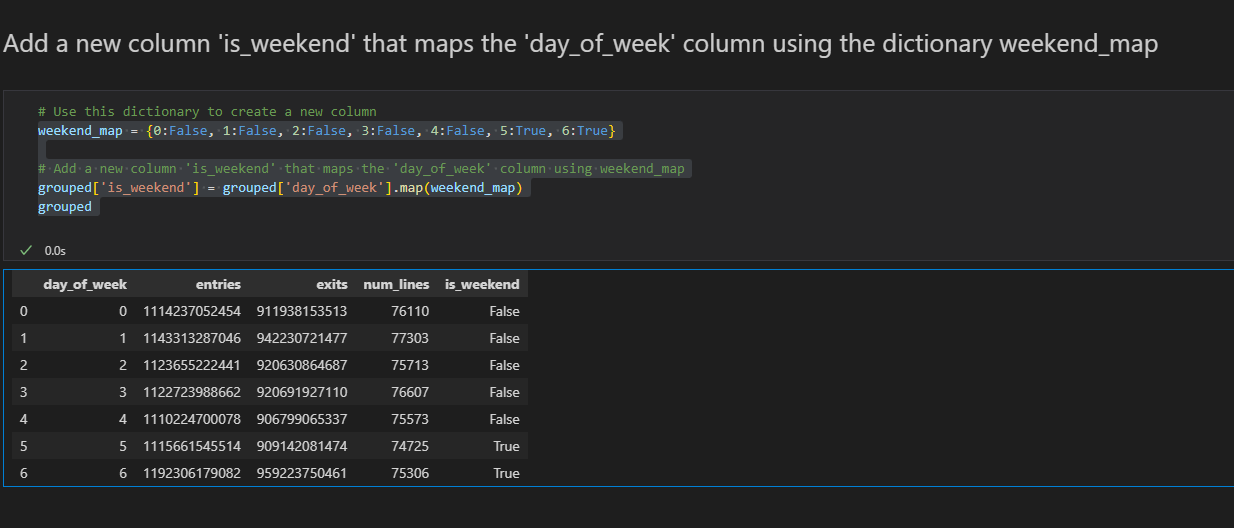
**33.bWorks same as this**

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



**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)

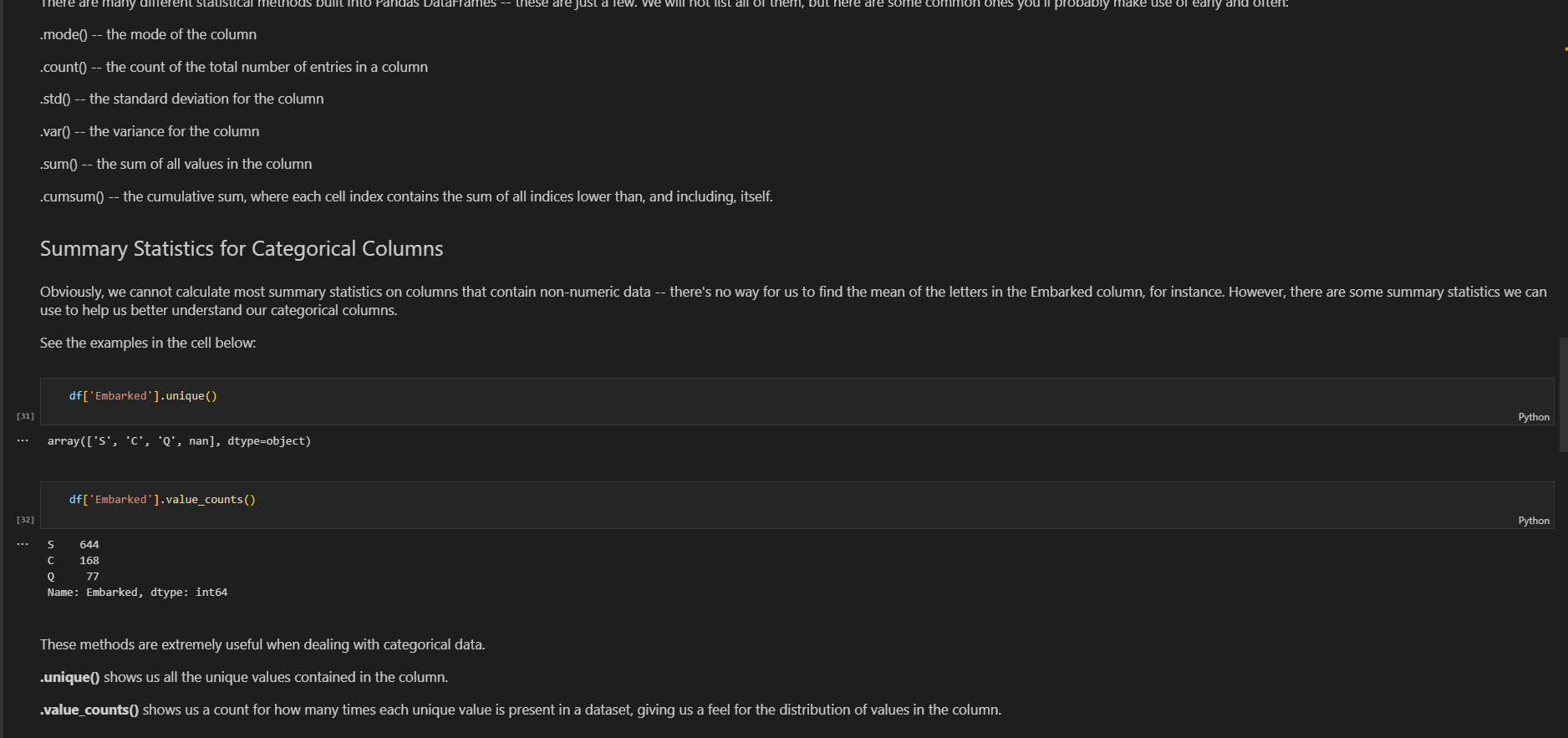
36.Convert a column to a specific data type

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)

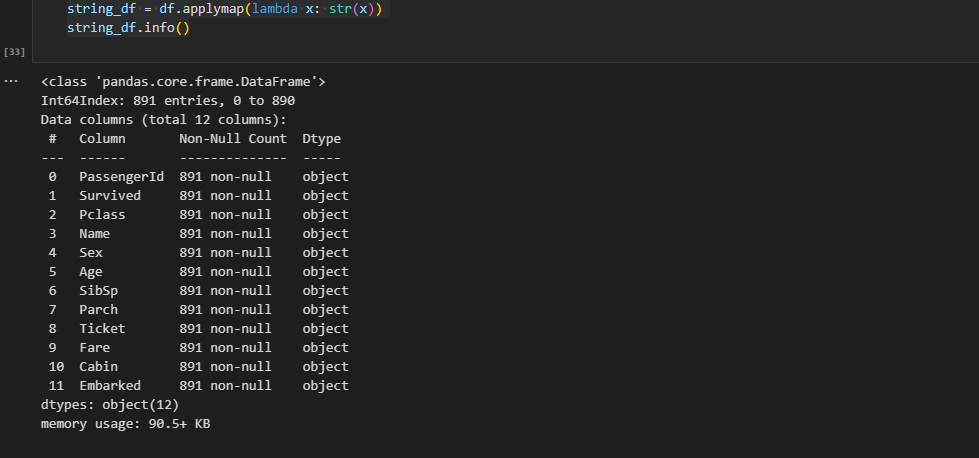
**36.summary statics for categorical values is unique and value counts**



**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()



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



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',

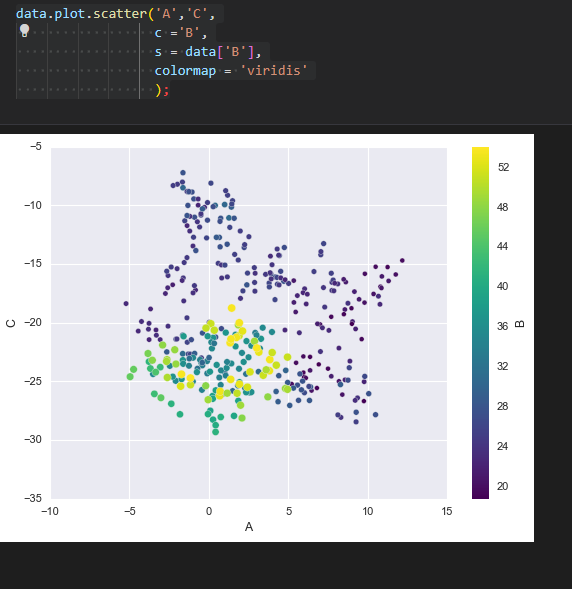
                  c ='B',

                  s = data['B'],

                  colormap = 'viridis'

                  );

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



**b)box plot**

data.plot.box();

**c)Histogram**

data.plot.hist(alpha = 0.7);

**d)Kernel density plot(kde)**

data.plot.kde();



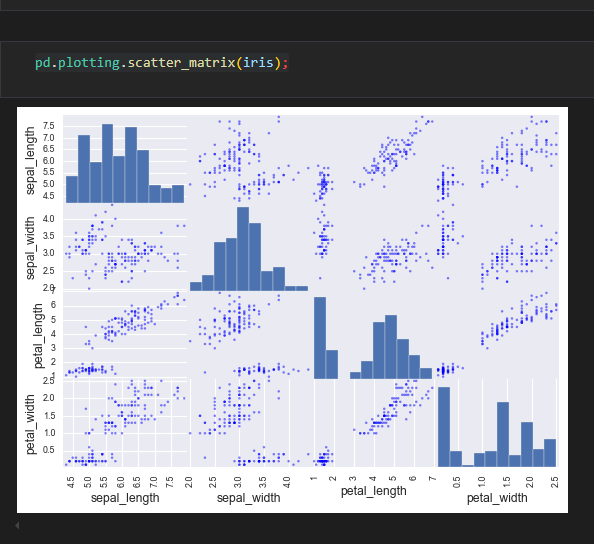
**e)Visualizing high dimension data** (way too many columns/variables to inspect individually

) 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);



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.

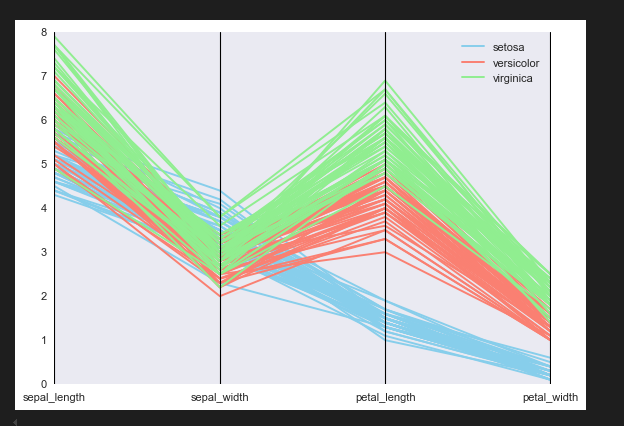
Let's create a parallel plot for the 4 predictor variables in the iris dataset and see if we can make any further judgments about the nature of data. We will use the pd.plotting.parellel\_coordinates() function and pass in the iris dataset with the response column (species) as an argument, just like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



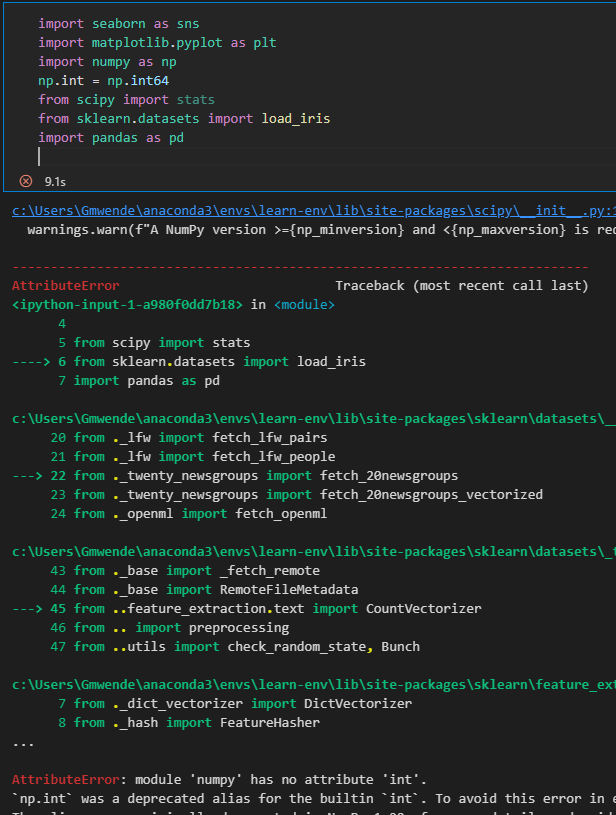
**### So, what do we learn from the parallel plot?**

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))

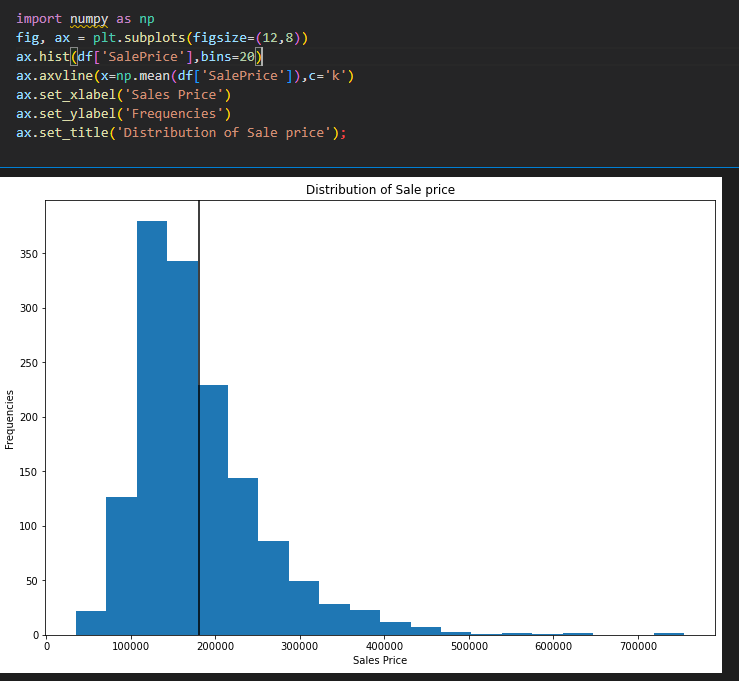
ax.scatter(X,y,s=100)

#add line plot ontop

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

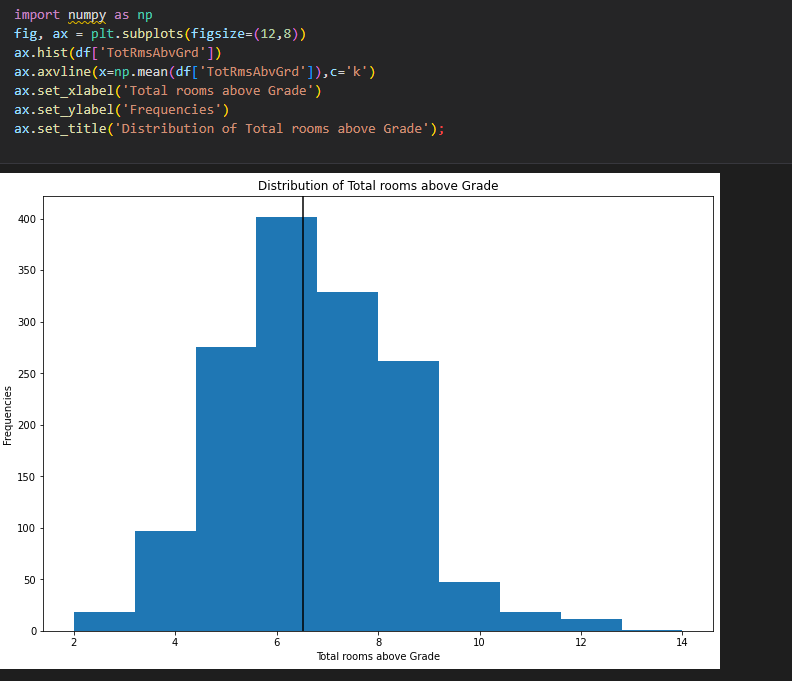
**45. Adding horizontal line on visualization**

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



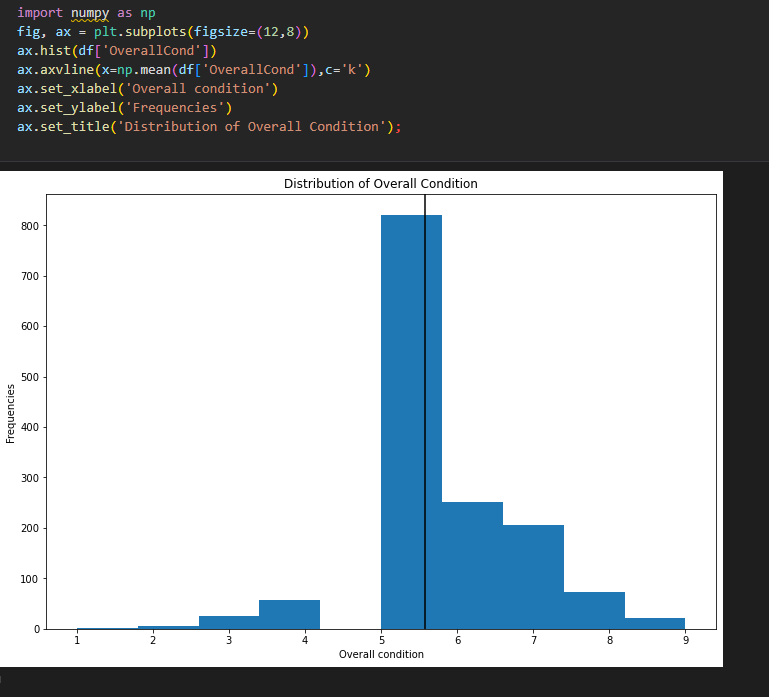
Typical average sales is 180k , from the histogram we can see that the data is skewed and therefore

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

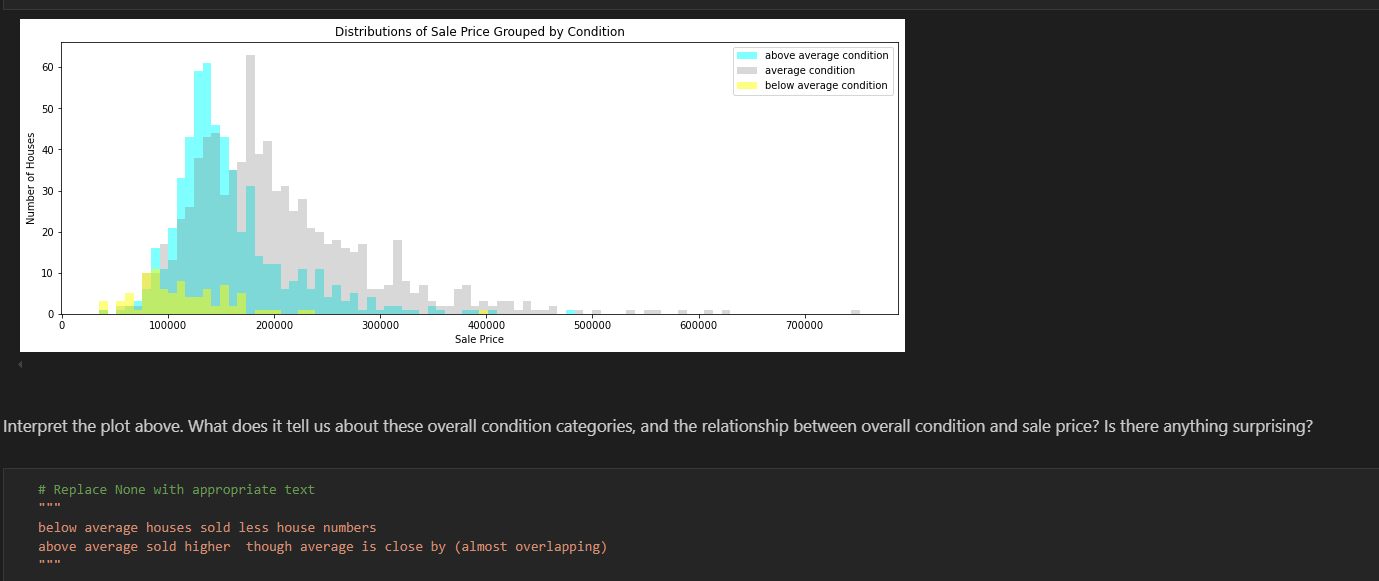


The data looks uniformly distributed  therefore average Total rooms is 6.5

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



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

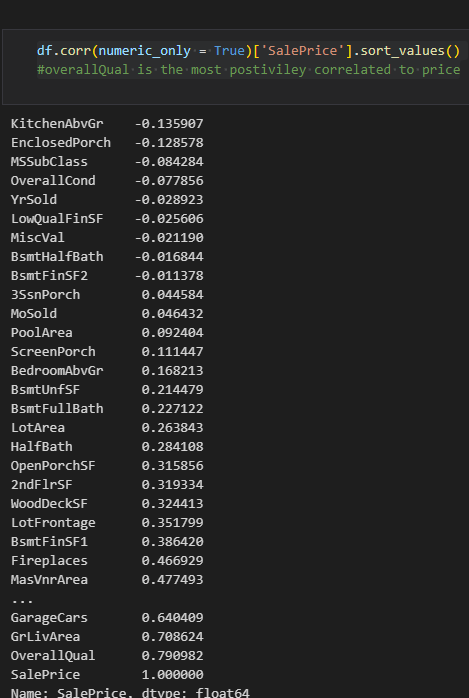


**46.Exploring how a certain colun correlates to others**

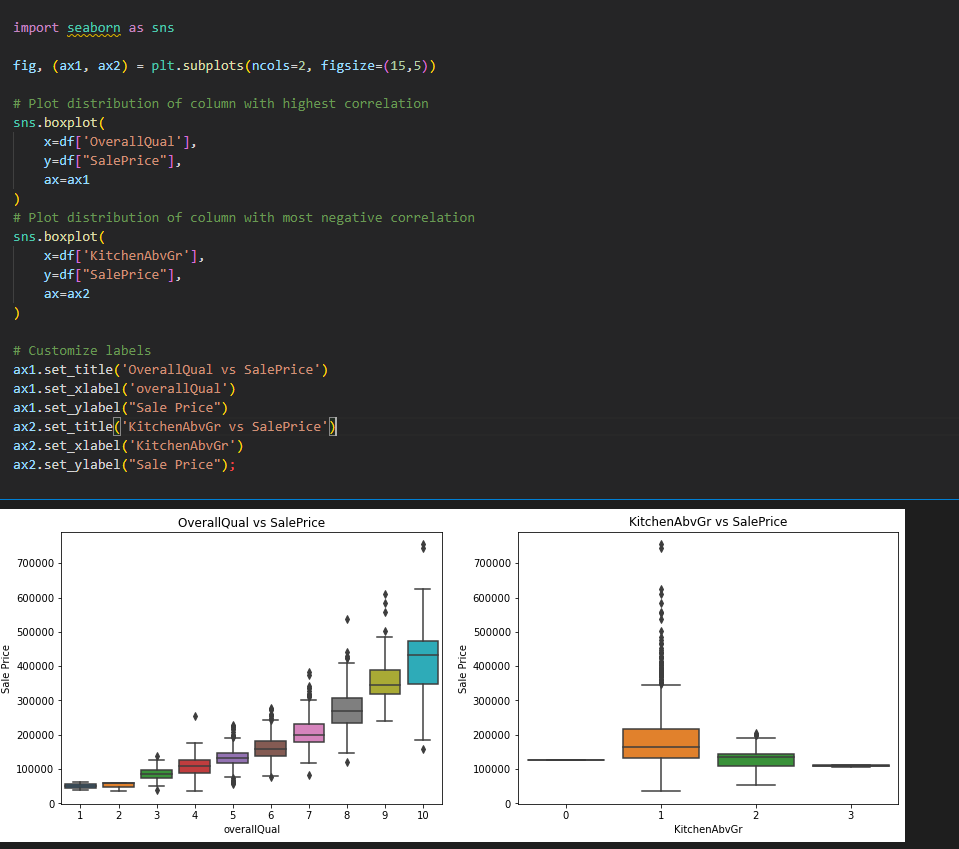
df.corr(numeric\_only = True)['SalePrice'].sort\_values()

#overallQual is the most postiviley correlated to price

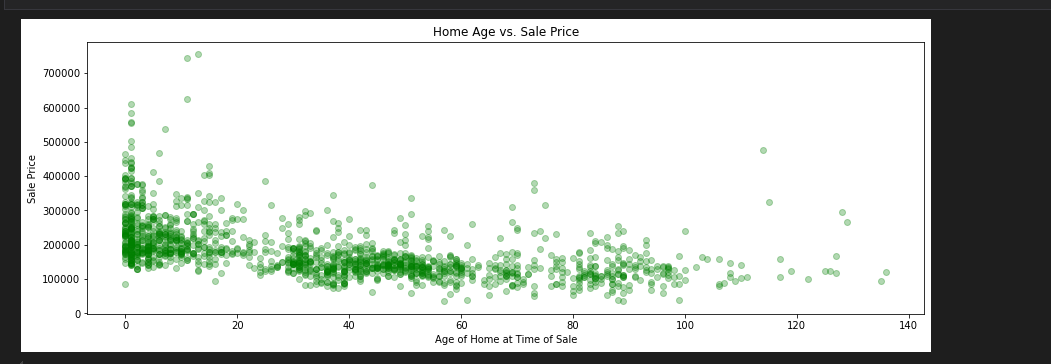
#KitchenAbvGr is the most negatively correlated to price



**47.Box plot for most positively and most negatively related**



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)**

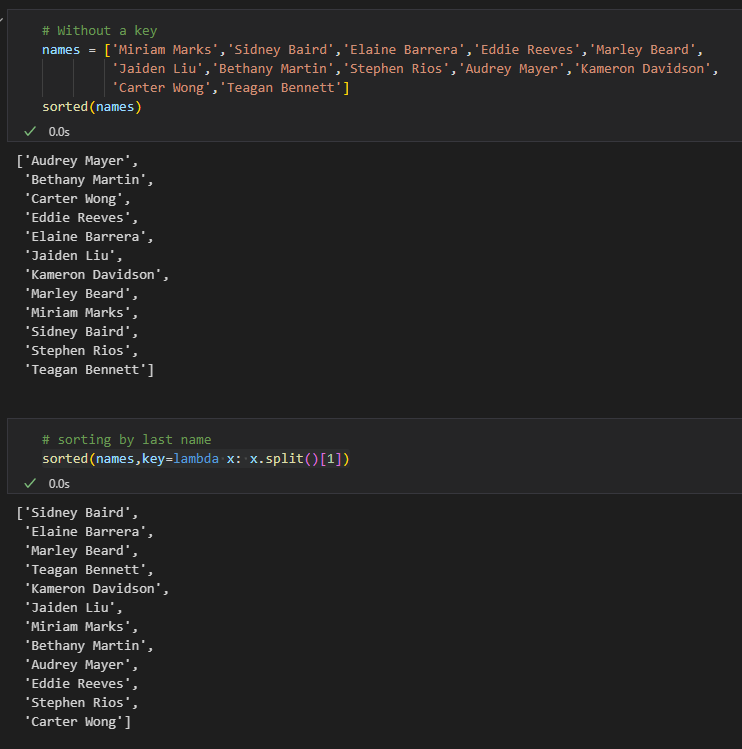
df['date'].map( lambda x: x[:4])

* Can also be achived by

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

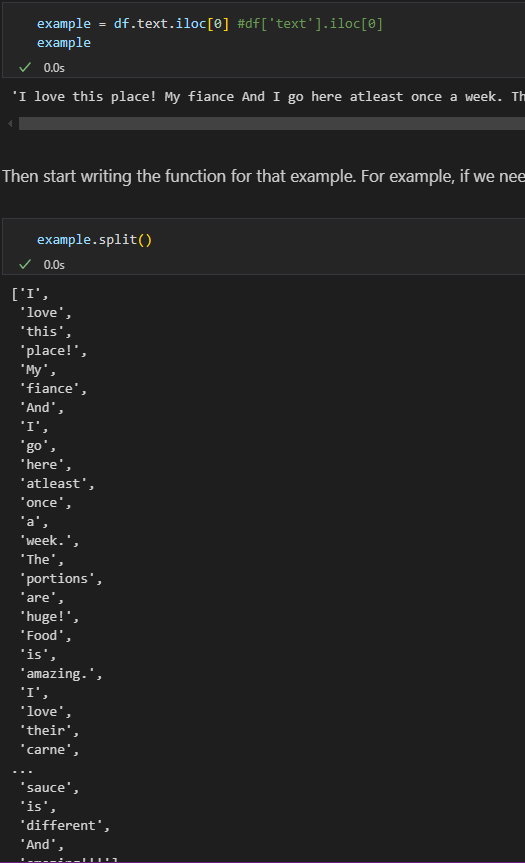
**d)also useful in sorting eg sorted on names will sort by first name wha if you wanted to sort by last name**

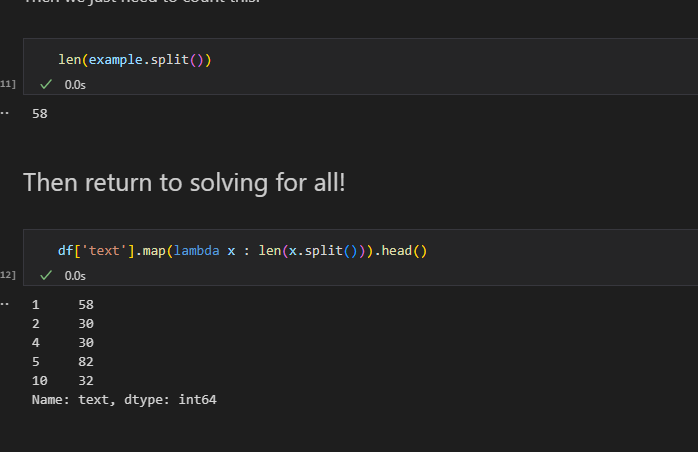
sorted(names,key=lambda x: x.split()[1])



* General approach to lambda is by first solving individual cases first

Eg





**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

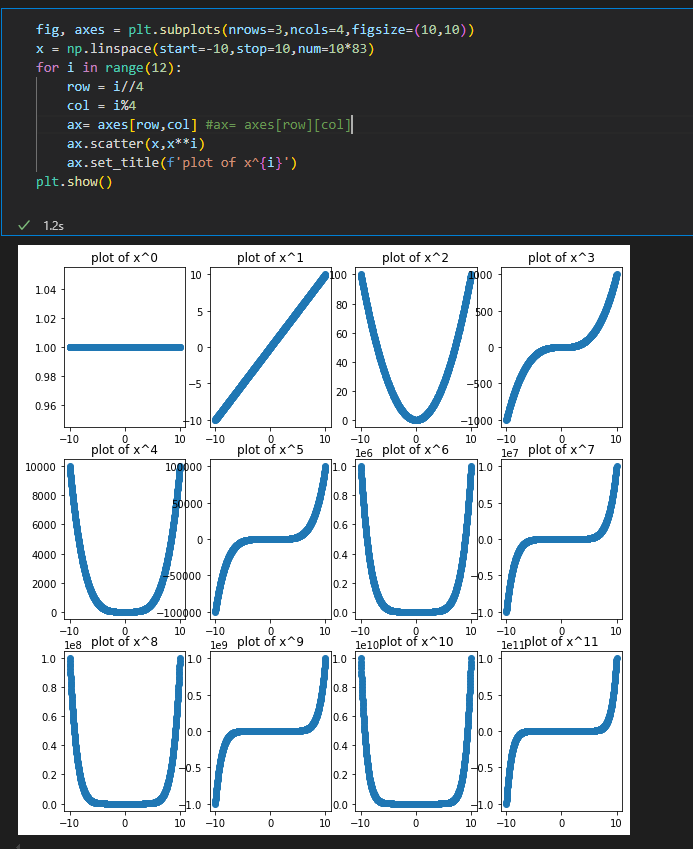
    col = i%4

    ax= axes[row,col] #ax= axes[row][col]

    ax.scatter(x,x\*\*i)

    ax.set\_title(f'plot of x^{i}')

plt.show()



**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.**

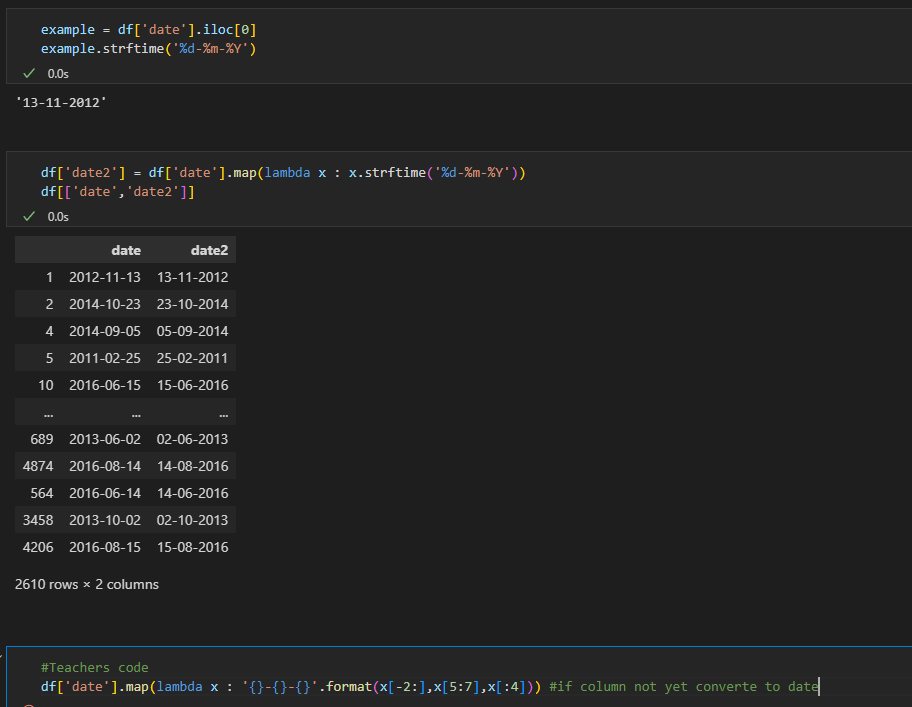
If column in string

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']]



**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.

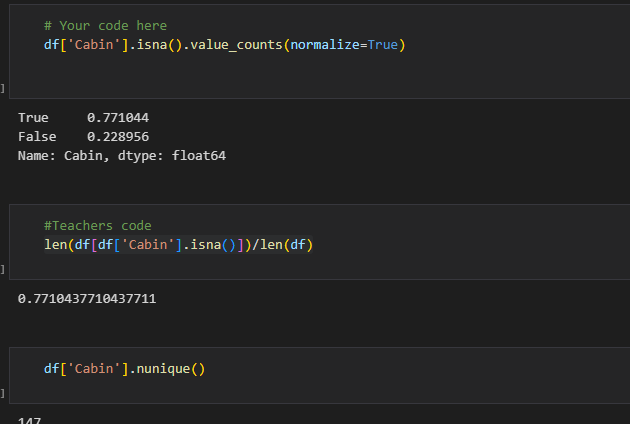
We'll deal with each column individually, and employ a different strategy for each.

- 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)



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



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 datase*t.  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())

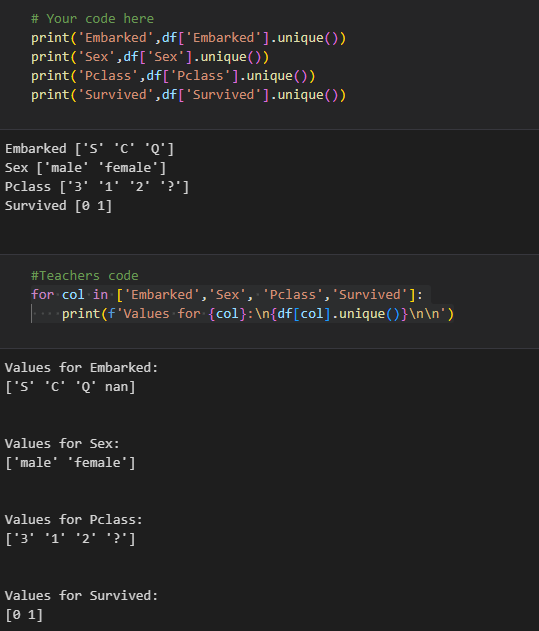
print('Pclass',df['Pclass'].unique())

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

or put togther

for col in ['Embarked','Sex', 'Pclass','Survived']:

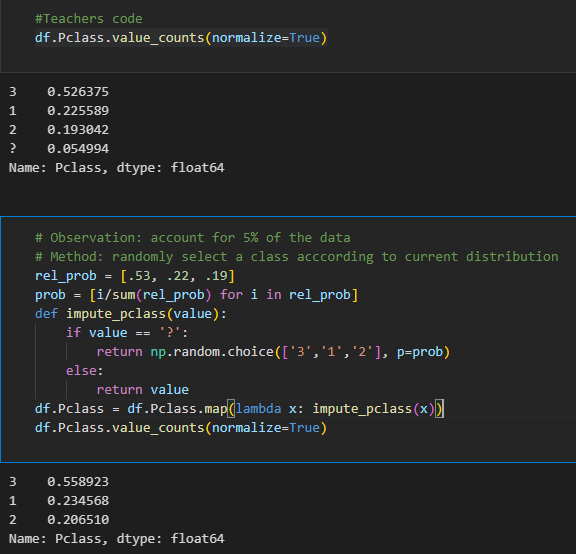
    print(f'Values for {col}:\n{df[col].unique()}\n\n')



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)



**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

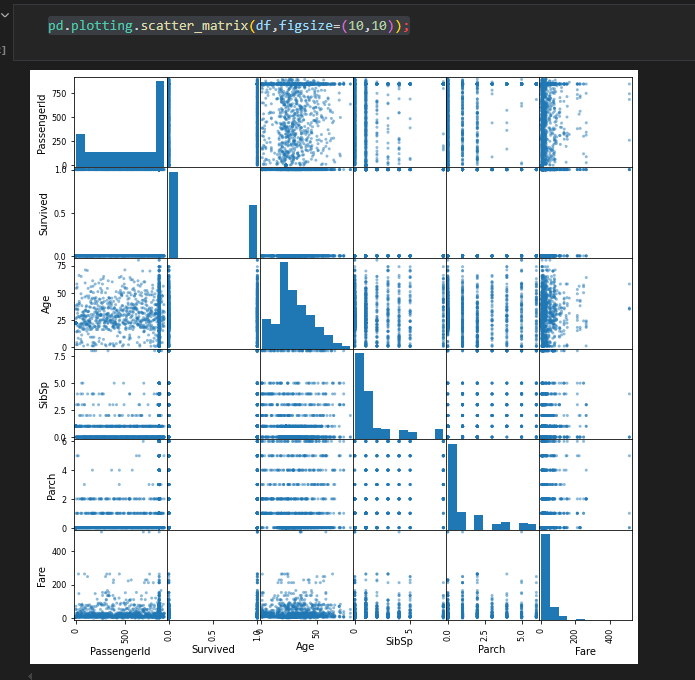
(albeit this assumption is statistically likely

**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));



**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.



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))

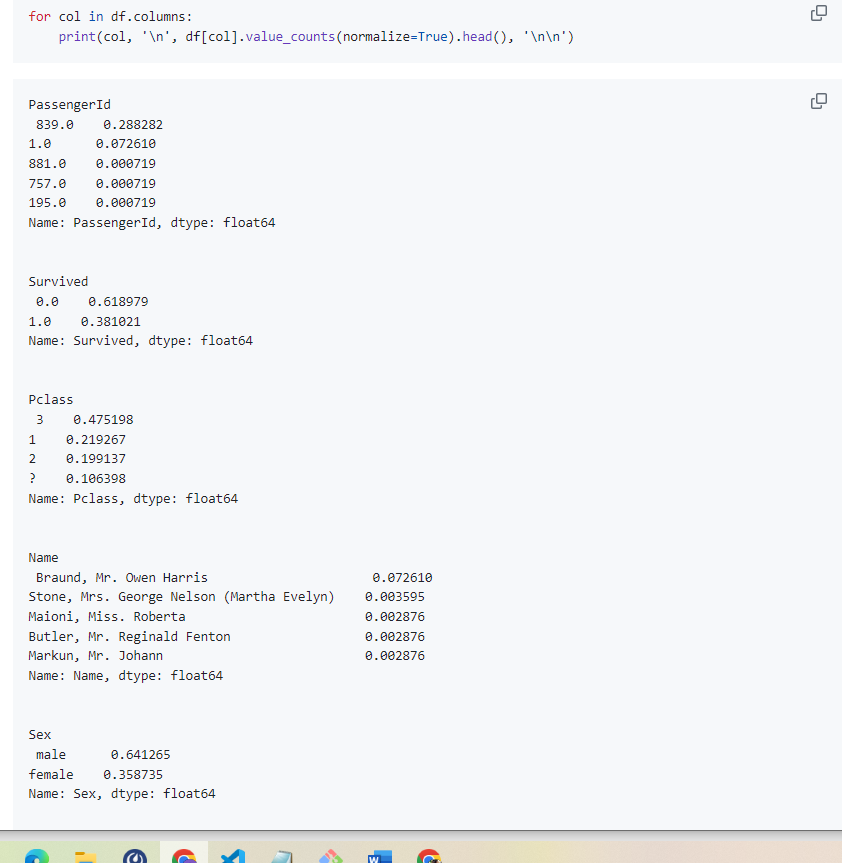
500

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:

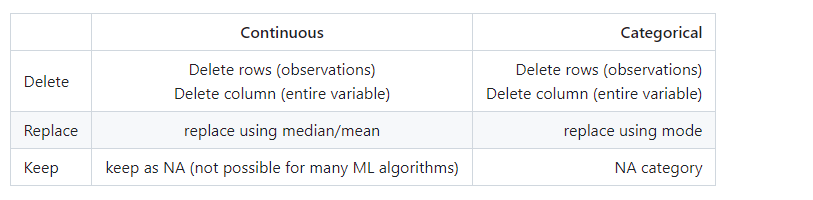
print(col, '\n', df[col].value\_counts(normalize=True).head(), '\n\n')



- 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.



**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]

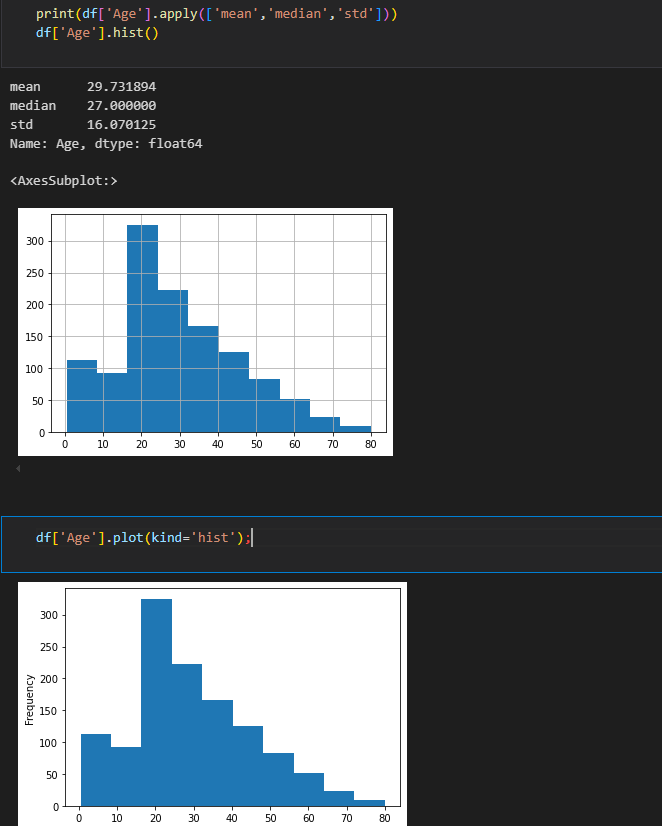
subset = df[cols]

**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()

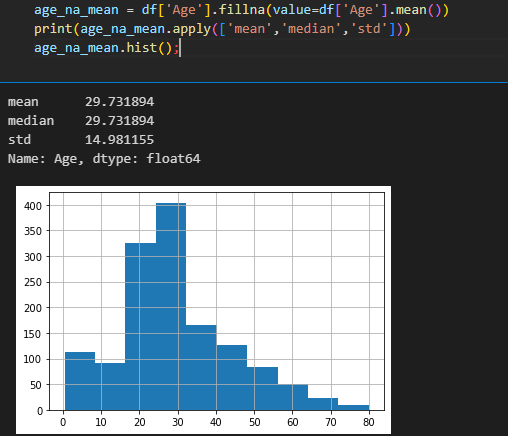
* 

**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();



**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

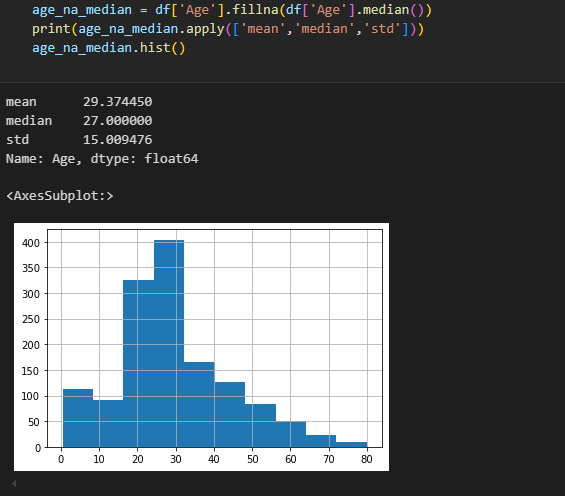
**b)Impute with median**

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.



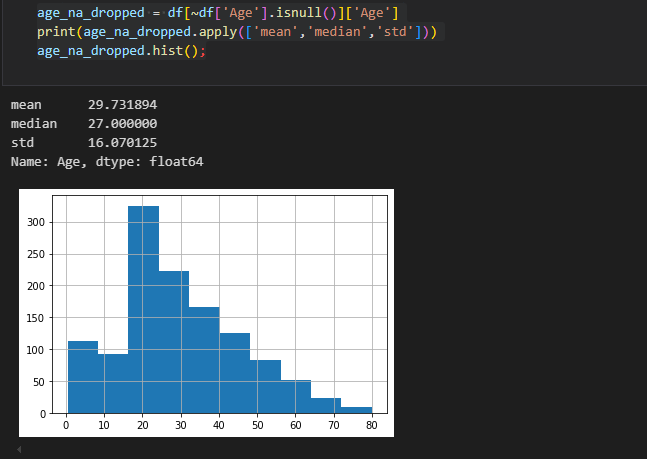
**IMPACT**Has similar effectiveness 2 imputing the mean.The variance is reduced, mean is slightly lowered.large mass of distribution near the center  
std from 16.07 to 15.00 ,mean from 29.73 to 29.37

**c)Dropping rows**

age\_na\_dropped = df[~df['Age'].isnull()]['Age']

print(age\_na\_dropped.apply(['mean','median','std']))

age\_na\_dropped.hist();



**IMPACT**

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