MATPLOTLIB

```
height = [150,160,165,185]
weight = [70, 80, 90, 100]

# figure size in inches
plt.figure(figsize=(15,5))

# draw the plot
plt.plot(height, weight)
plt.plot(weight,  'y--')
```
yellow line and – means dashed line

```
# draw the plot
plt.plot(height,weight)

# add title
plt.title("Relationship between height and weight")

# label x axis
plt.xlabel("Height")

# label y axis
plt.ylabel("Weight")

# add legend in the lower right part of the figure
plt.legend(labels=['Calories Burnt', 'Weight'], loc='lower right')

# set labels for each of these persons
plt.xticks(ticks=[0,1,2,3], labels=['p1', 'p2', 'p3', 'p4']);
```

BARPLOT from a groupby summary:

```
sales_by_outlet_size = data_BM.groupby('Outlet_Size').Item_Outlet_Sales.mean()
x = sales_by_outlet_size.index.tolist()
y = sales_by_outlet_size.values.tolist()
plt.bar(x, y, color=['red', 'orange', 'magenta'])
```

SUBPLOTS

```
# create 2 plots
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6,6))

# plot on 0 row and 0 column
ax[0,0].plot(calories_burnt,'go')
```

```
# plot on 0 row and 1 column
ax[0,1].plot(weight)

# set titles for subplots
ax[0,0].set_title("Calories Burnt")
ax[0,1].set_title("Weight")

# set ticks for each of these persons
ax[0,0].set_xticks(ticks=[0,1,2,3]);
ax[0,1].set_xticks(ticks=[0,1,2,3]);

# set labels for each of these persons
ax[0,0].set_xticklabels(labels=['p1', 'p2', 'p3', 'p4']);
ax[0,1].set_xticklabels(labels=['p1', 'p2', 'p3', 'p4']);
```

SEABORN

import seaborn as sns

Lineplot:
sns.lineplot(data=aa[:100], x="Item_Weight", y="Item_MRP", hue="Outlet_Size")

BarPlot:
sns.barplot(x="Item_Type", y="Item_MRP", data=aa[:5])

Distribution plot (Bar plot with density distribution curve)
sns.distplot(aa['Item_MRP'])

BoxPlot:
sns.boxplot(aa['Item_Outlet_Sales'], orient='vertical')

ViolinPlot:
sns.violinplot(aa['Item_Outlet_Sales'], orient='vertical', color='yellow')

Lineplot and ScatterPlot: using relplot (kind='line / 'scatter')
sns.relplot(x="Item_MRP", y="Item_Outlet_Sales", col="Outlet_Size", hue='Item_Weight', kind='scatter', size="Visibility_Scaled", data=aa[:200])

Advanced Categorical Plots using catplot (only kind= changes): sns.catplot

Strip plot and Swarn Plot: (kind='strip' / kind='swarn')

- Strip overlaps points whereas swarn gives an idea of distribution density

sns.catplot(x="Outlet_Size", y="Item_Outlet_Sales", kind='strip', data=aa[:250])

Box plot:

sns.catplot(x="Outlet_Size", y="Item_Outlet_Sales", kind='box', data=aa)

Violin Plot:

sns.catplot(x="Outlet_Size", y="Item_Outlet_Sales", kind='violin', data=aa)

Bar Plot:

sns.catplot(x="Outlet_Size", y="Item_Outlet_Sales", kind='bar', data=aa)

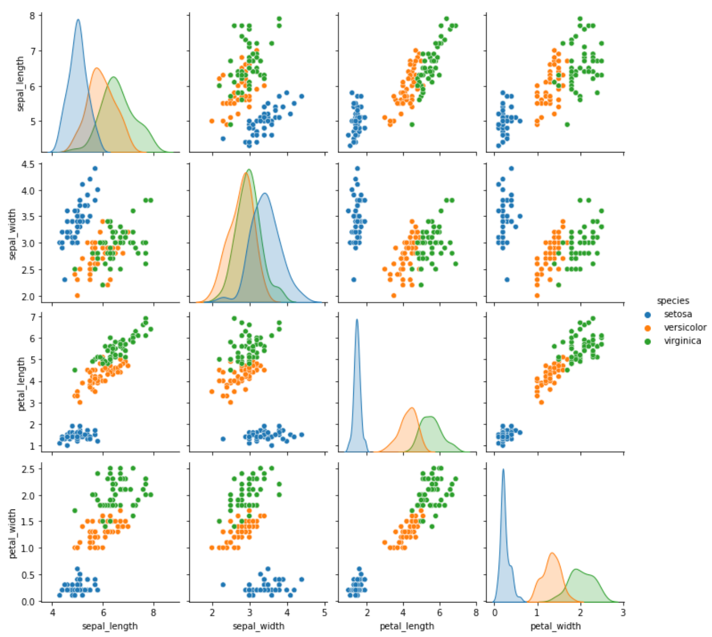
Density Plots : sns.kdeplot

sns.kdeplot(aa['Item_Visibility'], shade=True)

sns.distplot(aa['Item_Outlet_Sales'])

Pairplots: to give plots of all pair of variables

sns.pairplot(iris, hue='species')



PANDAS

Import pandas as pd
Import numpy as np

1. Reading a csv:
    a. df=pd.read_csv(‘data.csv’)
    b. df.dropna(how=’any’)
2. Reading an excel:
    a. To get to know the sheets of excel:
        i. xl = pd.ExcelFile('foo.xls')
           xl.sheet_names # see all sheet names in list format
           xl.parse(sheet_name)
        ii. df=pd.read_excel(‘fee.xlsx’, sheet_name=’def’)

Understanding the data:

aa.head()
aa.tail()
aa.shape
aa.columns
aa.describe()

- **aa.describe(include=[‘O’]**
- **aa.describe(percentiles=[.01,.05,.1,.25,......])**

Aa.iloc[:5] *rows*
Aa.iloc[:,:5] *Columns*
aa[(aa[‘Age’]==5) & (aa[‘Height’]<70)] *applying filters*
sort_values(by=’Date’, ascending=False, inplace=True)
sort_index()

Identify the elements with ***missing values*** in a particular row or column:

```
merged_left[ pd.isnull(merged_left.genus) ]
```

```
Sum of missing rows in dataframe:
ex5_pandas.isnull().sum()
```

aa.dropna()

- **Drops the missing values**

- **Parameters:**
  - **subset=[‘var1’,’var2’]** *Only drop missing values from scanning the subset*
  - **how= ‘all’ or ‘any’** *Drop when all values are missing or any value is missing*
  - **Axis =1 or 0** *0 removes rows containing null, 1 removes columns*
  - **Renaming the columns:**

    pmt5.rename(columns = {'Suit Filed':'Suit_Filed', 'post wo settled':'post_wo_settled', 'written off':'written_off'}, inplace = True)

  - **Transposing a data using melt()**

    ex2_pandas=ex1_pandas.melt(['Date','ticker'],var_name='cols',value_name='vals')

    Date, tickler wont be modified, whereas everything else will be transposed with col name in var_name column and their values in value_name column

Merging the Dataset:

1. concat() - appends the data

cnct= pd.concat([df1,df2,df3], keys=[‘a’,’b’,’c’]) cnct.loc[‘c’] returns values from df3

2. merge() - merges 2 datasets based on some key

```
merged_left = pd.merge(left=survey_sub, right=species_sub, how='left',
left_on='species_id', right_on='species_id', validate='one_to_one')
```

- how: One of 'left', 'right', 'outer', 'inner', 'cross'. Defaults to inner. See below for more detailed description of each method.

- **validate : string, default None. If specified, checks if merge is of specified type.**
    - **“one_to_one” or “1:1”: checks if merge keys are unique in both left and right datasets.**
    - **“one_to_many” or “1:m”: checks if merge keys are unique in left dataset.**
    - **“many_to_one” or “m:1”: checks if merge keys are unique in right dataset.**
    - **“many_to_many” or “m:m”: allowed, but does not result in checks.**

PIVOT TABLE

Pivot table has similar functionality as excel pivots

```
data.pivot_table(index=["race", "sex"], columns="salary",
values=[‘age’,’salary’], aggfunc={"age": ["mean",’median]
,"salary": lambda x: x.mode()[0]}, margins = True,
margins_name='Total')
```

Aggfunc can take as input the custom defined functions

GROUPBY

```
paygrp=pay.groupby(['Gender','Occupation','Religion']).agg({'loana
mount':['sum','mean','count','nunique'],'Tenure of
loan':'sum'}).unstack()
```

CONVERTING CONTINUOUS TO CATEGORICAL
Or
UPDATING A COLUMN

payments['Age_b'] = pd.cut(payments['Age'], bins = [-np.inf,0,20,30,40,50,60,70,np.inf])
payments['Age_b'] = pd.qcut(payments['Age'], q=10)

```
for dataset in combine:
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age']
= 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age']
= 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age']
= 3
    dataset.loc[ dataset['Age'] > 64, 'Age']
train_df.head()
```

CONVERTING CATEGORICAL VARIABLES INTO NUMERIC

```
dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2}
).astype(int)
```

RENAMING COLUMNS IN PANDAS DATAFRAME

```
df.rename(columns={'gender':'Gender','occupat':'Occupation'}, inplace=True)
```

APPLY FUNCTION / LAMBDA FUNCTION USE CASES

```
df = df.assign(Product=lambda x: (x['Field_1'] * x['Field_2']
* x['Field_3']))

df = df.apply(lambda x: np.square(x) if x.name == 'd' else x,
axis=1)
```

Using Lambda to apply regular expressions in a string:

```
data=pd.read_csv("titanic.csv")
def titles_ext(x):
    return re.findall('\w+[.]',x)[0]

titles=data['Name'].apply(lambda x: titles_ext(x))

titles.value_counts()
```

STRING TO DATE TIME CONVERSIONS:


```
from datetime import datetime

date_time_str = '18/09/19 01:55:19'

date_time_obj = datetime.strptime(date_time_str, '%d/%m/%y %H:%M:%S')


print ("The type of the date is now",  type(date_time_obj))
print ("The date is", date_time_obj)
```

EXAMPLE 1:

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
print(x.strftime("%B"))
x = datetime.datetime(2020, 5, 17,23,23,23)
print(x)
```

EXAMPLE 2:

```
from datetime import datetime
def dt(x):
    return datetime.strptime(x, '%d/%m/%Y')
df22['acq_dt']=df22['acq_dt'].apply(lambda x: dt(x))
df22.info()
```

RUNNING SQL QUERIES IN PYTHON PANDAS:

```
ex3_sql_query = """
SELECT
    date(Date) AS Date
    , ticker
    , closing_price
    , LAG(closing_price, 1) OVER(
        PARTITION BY ticker
        ORDER BY date(Date)
    ) AS previous_close
FROM
    prices
"""
ex3_sql = pd.read_sql(ex3_sql_query, con=conn)
Ex3_sql
```

WINDOW FUNCTIONS:

Example 1: (finding max price with transform function. (Equivalent to partition by in SQL)

```
ex3_pandas["previous_close"] =
(ex3_pandas.sort_values("Date").groupby("ticker")["closing_price"].transform("max"))
ex3_pandas
```

Example 2: (finding last day closing price using shift() function. (Equivalent to LAG() in SQL)

```
ex3_pandas["previous_close"] =
(ex3_pandas.sort_values("Date").groupby("ticker")["closing_price"].shift(1))
ex3_pandas
```

Example 3: (finding %gain using lambda and shift() function. (Equivalent to LAG() in SQL)

```
ex3_pandas["previous_close"] =
(ex3_pandas.sort_values("Date").groupby("ticker")["closing_price"].transform(lambda x: x/x.shift(1)-1)
ex3_pandas
```

DROPPING DUPLICATES IN PYTHON

```
Aa.drop_duplicates()
```