In [ ]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as pplt
import seaborn as sbn
%matplotlib inline
```

In [ ]:
```
# Creating from a csv file
mcEnrollmentDf = pd.read_csv ('Montgomery_College_Enrollment_Data.csv')
```

In [ ]:
```
# View data
mcEnrollmentDf.head (1)
```

In [ ]:
```
# Statistical information on numeric columns
mcEnrollmentDf.describe ()
```

In [ ]:
```
# Size
mcEnrollmentDf.shape
```

In [ ]:
```
mcEnrollmentDf ['Age Group'].value_counts ()
```

In [ ]:
```
# Counts of values in one column
x = mcEnrollmentDf ['Age Group'].value_counts ()
print (x.index, x.values)
```

pyplot

In [ ]:
```
# Plot line y vs x
pplt.plot (x.index, x.values)
pplt.show ()
```

In [ ]:
```
# Plot markers y vs x
pplt.plot (x.index, x.values, 'r+')
pplt.show ()
```

In [ ]:
```
# Plot line y vs x
pplt.plot (mcEnrollmentDf ['Age Group'], mcEnrollmentDf ['Attending Germantown'])
pplt.show ()
```

In [ ]:
```
# Create other y vs x plots using some test data
xt = np.arange (0, 2, 0.1)
yt = np.sin (xt)
pplt.plot (xt, yt, '*')

t = np.arange (0.0, 2.0, 0.01)
s = 1 + np.sin (2 * np.pi * t)

pplt.plot (t, s)
pplt.grid ()
pplt.xlabel ('Time (sec)')
pplt.ylabel ('Voltage (mV)')
```

In [ ]:

```
# Create and save plots
fig, ax = pplt.subplots ()
ax.plot (t, s)
ax.grid ()
pplt.xlabel ('Time (sec)')
pplt.ylabel ('Voltage (mV)')
pplt.savefig ("test.png")
```

In [ ]:

```
# Load a builtin dataset
tipsDf = sbn.load_dataset ("tips")
```

In [ ]:

```
tipsDf.head (1)
```

In [ ]:

```
tipsDf.shape
```

In [ ]:

```
# Create a scatter plot showing tips and total bills
pplt.scatter (tipsDf.total_bill, tipsDf.tip, s = tipsDf.tip * 5, c = 'r')
pplt.grid ()
pplt.xlabel ('Total Bill')
pplt.ylabel ('Tip')
```

In [ ]:

```
pplt.figure (figsize = (9, 3))

pplt.subplot (131)
pplt.bar (x.index, x.values)
pplt.subplot (132)
pplt.scatter (x.index, x.values)
pplt.subplot (133)
pplt.scatter (tipsDf.total_bill, tipsDf.tip)
pplt.show ()
```

In [ ]:

```
pplt.figure (figsize = (9, 3))

pplt.subplot (131)
pplt.barh (x.index, x.values)
pplt.xticks (rotation = -45)
pplt.grid ()
pplt.subplot (132)
pplt.scatter (x.index, x.values)
pplt.xticks (rotation = -45)
pplt.grid ()
pplt.subplot (133)
pplt.scatter (tipsDf.total_bill, tipsDf.tip)
pplt.xticks (rotation = -45)
pplt.grid ()
pplt.show ()
```

In [ ]:

```
# Pie chart
pplt.pie (x.values, labels = x.index, autopct='%1.1f%%')
pplt.show ()
```

seaborn

In [ ]:

```
# Relational plots using seaborn
sbn.relplot (x = "total_bill", y = "tip", col = "time",
             hue = "smoker", style = "smoker", size = "size",
             data = tipsDf);
```

In [ ]:

```
# Create a scatter plot showing tips and total bills using seaborn
sbn.lmplot (x = 'total_bill', y = 'tip', data = tipsDf,
            fit_reg = False, #remove regression line
            hue = 'size')    #color by size of the guests in party
```

In [ ]:

```
# Create a scatter plot showing tips and total bills using seaborn
sbn.lmplot (x = 'total_bill', y = 'tip', data = tipsDf,
            fit_reg = True)
```

In [ ]:

```
# Create a scatter plot showing tips and total bills using seaborn
sbn.lmplot (x = 'total_bill', y = 'tip', data = tipsDf,
            fit_reg = True, #Default is true to add regression line
            hue = 'smoker')    #color by size of the guests in party
```

In [ ]:

```
# Counts of values in one column
mcEnrollmentDf ['Age Group'].value_counts ()
```

In [ ]:

```
# Create a bar plot of Age Group counts
sbn.countplot (x = 'Age Group',
               data = mcEnrollmentDf)
```

In [ ]:

```
# Create a bar plot of Age Group counts comparing Student Status
sbn.countplot (x = 'Age Group', hue = 'Student Status',
               data = mcEnrollmentDf)

pplt.ylabel ('Age Group Counts')
pplt.xticks (rotation = -45)
```

In [ ]:

```
# Create a bar plot of Age Group counts, sorted, comparing Student Status
sbn.countplot (x = 'Age Group', hue = 'Student Status',
               data = mcEnrollmentDf.sort_values (by = 'Age Group'))

pplt.ylabel ('Age Group Counts')
pplt.xticks (rotation = -45)
pplt.title ('Enrollment by Age Groups')
```

In [ ]:

```
# Create a bar plot of Age Group counts sorted, comparing Student Type
sbn.countplot (x = 'Age Group', hue = 'Student Type',
               data = mcEnrollmentDf.sort_values (by = 'Age Group'))

pplt.ylabel ('Age Group Counts by Student Type')
pplt.xticks (rotation = -45)
pplt.title ('Enrollment by Age Groups, Compared by Student Type')
```

In [ ]:

```
# Box plot showing quartiles
sbn.boxplot (x = "day", y = "tip", data = tipsDf)
```

In [ ]:

```
# Violin plot showing density
sbn.violinplot (x = "day", y = "tip", data = tipsDf)
```

In [ ]:

```
# Swarm plot showing non-overlapping points
sbn.swarmplot (x = "day", y = "tip", data = tipsDf)
```

In [ ]:

```
# Get correlation among various numeric variables in the dataframe
corr = tipsDf.corr ()

sbn.heatmap (corr, vmin = -1, annot = True)
```

In [ ]:

```
# Create a histogram of the distribution of total_bill
sbn.distplot (tipsDf ['total_bill'])
```

In [ ]:

```
# Standard Deviation Method
meanbill = tipsDf ['total_bill'].mean ()
stdbill = tipsDf ['total_bill'].std ()
toprange = meanbill + stdbill * 1.96
botrange = meanbill - stdbill * 1.96

tipsStdDf = tipsDf.copy() #to not mess up the original df
tipsStdDf = tipsStdDf.drop (tipsStdDf [tipsStdDf ['total_bill'] > toprange].index)
tipsStdDf = tipsStdDf.drop (tipsStdDf [tipsStdDf ['total_bill'] < botrange].index)

tipsStdDf.head ()
```

In [ ]:

```
tipsStdDf.shape
```

In [ ]:

```
corr = tipsStdDf.corr ()

sbn.heatmap (corr, vmin = -1, annot = True)
```

In [ ]:

```
# Create a histogram of the distribution of total_bill
sbn.distplot (tipsStdDf ['total_bill'])
```

In [ ]:

```
tipsDf ['total_bill'].min ()
```

In [ ]:

```
# Now let us make buckets of total_bill ranges to know how the total_bill is distri
buted
# and to support plotting where these properties are
totalBillBins = [0, 20, 40, 100]

# Create names for the as many groups as needed per 'totalBillBins'
totalBillBinsNames = ['<20', '20 - <40', '40+']

tipsDfCopy = tipsDf.copy ()

# Now use 'tipsDfCopy' to make a new column
tipsDfCopy ['total_bill_group'] = pd.cut (tipsDfCopy ['total_bill'],
                                          totalBillBins,
                                          labels = totalBillBinsNames)
tipsDfCopy.to_csv ('tips with bill category.csv')

#tipsDfCopy.head()
#tipsDfCopy.count()
print
("-----------------------------------------------------------------------------------
-------------------")
print ("-->Values in the total_bill column and the new total_bill_group column :\
n",
       tipsDfCopy [['total_bill', 'total_bill_group']])
```

In [ ]:

```
# Load an built-in dataset from one of the libraries i.e. sklearn
from sklearn.datasets import load_boston
bostonDs = load_boston ()
```

In [ ]:

```
from sklearn.datasets import load_diabetes
diabetesDs = load_diabetes ()
```

In [ ]:

```
#data = dataset, target = dependent variable, feature_names = column headers, DESCR
= data dictionary
bostonDs.keys()
```

In [ ]:

```
bostonDs.data.shape
```

In [ ]:

```
print (bostonDs.DESCR)
```

In [ ]:

```
# load the boston data into a dataframe
bosDf = pd.DataFrame (bostonDs.data)
bosDf.head ()
```

In [ ]:

```
bosDf.columns = bostonDs.feature_names
bosDf.head ()
```

In [ ]:

```
#assigning dependent variable to column named "Price"
bosDf ['PRICE'] = bostonDs.target
```

In [ ]:

```
bosDf.head ()
```

In [ ]:

```
# Setup linear regression for use in this exercise
from sklearn.linear_model import LinearRegression

# Remove the price column from boston data and assign the new dataset to X
X = bosDf.drop ('PRICE', axis = 1)

# Assign linear regression function to a variable
linReg = LinearRegression ()
linReg
```

In [ ]:

```
# Now fit your data in a linear regression model
linReg.fit (X, bosDf.PRICE)
```

In [ ]:

```
# Now plot price vs no. of rooms
pplt.scatter (bosDf.RM, bosDf.PRICE)
pplt.xlabel ("Average number of rooms per dwelling (RM)")
pplt.ylabel ("Housing Price")
pplt.title ("Relationship between RM and Price")
pplt.show ()
```

In [ ]:

```
# Now plot the actual price vs. predicted price
pplt.scatter (bosDf.PRICE, linReg.predict (X))
pplt.xlabel ("Price")
pplt.ylabel ("Predicted Price")
pplt.title ("Price vs Predicted Price")
pplt.show ()
```

In [ ]:

```
# Now compare the plot of actual price vs no. of rooms against the plot of predicted price vs no. of rooms
pplt.figure (figsize = (9, 2))

pplt.subplot (121)
pplt.scatter (bosDf.RM, bosDf.PRICE)
pplt.grid ()

pplt.subplot (122)
pplt.scatter (bosDf.RM, linReg.predict (X))
pplt.grid ()
pplt.show ()
```

In [ ]: