In [1]:
```
"""
For section 1: You need to write your code in order to regenerate the exact re
For section 2: You need to generate the answers based on the questions without

"""

print(__doc__)
```

```
For section 1: You need to write your code in order to regenerate the exact re
sult as shown under each question
For section 2: You need to generate the answers based on the questions without
having the sample results.
```

Section 1

Note

- Instructions have been included for each segment. You do not have to follow them exactly, but they are included to help you think through the steps.

In [2]:
```
# Dependencies and Setup
import pandas as pd
import numpy as np

# File to Load (Remember to Change These)
file_to_load = "purchase_data.csv"

# Read Purchasing File and store into Pandas data frame
purchase_data = pd.read_csv(file_to_load)

purchase_data.head()
```

Out[2]:

	Purchase ID	SN	Age	Gender	Item ID	Item Name	Price
0	0	Lisim78	20	Male	108	Extraction, Quickblade Of Trembling Hands	3.53
1	1	Lisovynya38	40	Male	143	Frenzied Scimitar	1.56
2	2	Ithergue48	24	Male	92	Final Critic	4.88
3	3	Chamassasya86	24	Male	100	Blindscythe	3.27
4	4	Iskosia90	23	Male	131	Fury	1.44

Q1 - check for missing data

In [3]:
```
purchase_data.notnull().sum()
```

Out[3]:
```
Purchase ID    780
SN             780
Age            780
Gender         780
```

```
Item ID        780
Item Name      780
Price          780
dtype: int64
```

Player Count

Q2 - Display the total number of players

In [4]:
```
NumPlayer = purchase_data[['SN']]
NumPlayer = NumPlayer.drop_duplicates()
NumPlayer = NumPlayer.shape[0]
DisplayNumPlayer = pd.DataFrame({'Total # of Players': [NumPlayer]})
DisplayNumPlayer
```

Out[4]:

	Total # of Players
0	576

Q3- Purchasing Analysis (Total)

- Obtain number of unique items, average price, etc.
- Create a summary data frame to hold the results
- Optional: give the displayed data cleaner formatting
- Display the summary data frame

In [5]:
```
#Create a subset from purchase_data
TotalPurchase = purchase_data[['Item ID','Price']]

#Calculate Unique Items, Average Price, Total Purchases, Total Revenue
UniqueValue = TotalPurchase['Item ID'].nunique()
AveragePrice = round(TotalPurchase['Price'].mean(),2)
TotalPurchases = TotalPurchase['Item ID'].count()
TotalRevenue = TotalPurchase['Price'].sum()

#Create data frame
TotalPurchase = pd.DataFrame({'Unique items':[UniqueValue],'Average Price':[A

#Apply $ sign
TotalPurchase['Average Price'] = TotalPurchase['Average Price'].map('${:,.2f}
TotalPurchase['Total Revenue'] = TotalPurchase['Total Revenue'].map('${:,.2f}

#Display result
TotalPurchase
```

Out[5]:

	Unique items	Average Price	Total Purchases	Total Revenue
0	183	$3.05	780	$2,379.77

Q4- Gender Demographics

- Percentage and Count of Male Players
- Percentage and Count of Female Players
- Percentage and Count of Other / Non-Disclosed

In [6]:
```
#Subset on SN and Gender
GenderDemo = purchase_data[['SN','Gender']]

#Remove Duplicates
GenderDemo = GenderDemo.drop_duplicates()

#Apply Group By to Count
GenderDemo = GenderDemo.groupby(['Gender']).size().to_frame('Count').reset_inc

#Calculate Percentage
GenderDemo['Percent of Total Players'] = GenderDemo['Count']/ GenderDemo['Cou

#Reformat, add % sign to 'Percent of Total Players' column

GenderDemo['Percent of Total Players'] = GenderDemo['Percent of Total Players

#Sort by Percent of Total Players
GenderDemo = GenderDemo.sort_values(by = ['Percent of Total Players'], ascend

#Clear column name for Gender
GenderDemo = GenderDemo.rename(columns = {"Gender":""})

#Set index that column
GenderDemo.set_index('')
```

Out[6]:

	Count	Percent of Total Players
Male	484	84.03%
Female	81	14.06%
Other / Non-Disclosed	11	1.91%

Q5-Purchasing Analysis (Gender)

- Run basic calculations to obtain purchase count, avg. purchase price, avg. purchase total per person etc. by gender
- Create a summary data frame to hold the results
- Optional: give the displayed data cleaner formatting

In [7]:
```
#Create subset
Gender_Purchase = purchase_data[['Gender','Price']]

#Rename Price column
Gender_Purchase = Gender_Purchase.rename(columns = {"Price":"Amount Purchase"

#Apply group by
Gender_Purchase = Gender_Purchase.groupby('Gender').agg({'Amount Purchase': '

#Apply $ sign for Price-related column
Gender_Purchase['Amount Purchase'] = Gender_Purchase['Amount Purchase'].map('

#Display value
Gender_Purchase
```

Out[7]:

Gender	Amount Purchase
Female	$361.94
Male	$1,967.64
Other / Non-Disclosed	$50.19

Q6-Display the summary data frame

In [8]:
```
pd.options.mode.chained_assignment = None  # default='warn'

#Create subset
GenderSummary1 = purchase_data[['Gender','SN','Price']]

#Duplicate Price column
GenderSummary1['Price2'] = GenderSummary1['Price']

#Apply Group By
GenderSummary1 = GenderSummary1.groupby('Gender').agg({'SN': 'count','Price':

#Rename, round numbers, apply $ sign column
GenderSummary1 = GenderSummary1.rename(columns = {"SN":"Purchase Count","Price
GenderSummary1['Avg Purchase Price'] = round(GenderSummary1['Avg Purchase Pri
GenderSummary1['Avg Purchase Price'] = GenderSummary1['Avg Purchase Price'].ma
GenderSummary1['Total Revenue'] = GenderSummary1['Total Revenue'].map('${:,.2

#Calculate number of players
GenderSummary2 = purchase_data[['Gender','SN']]
GenderSummary2 = GenderSummary2.drop_duplicates()
GenderSummary2 = GenderSummary2.groupby('Gender').agg({'SN':'count'})
GenderSummary2 = GenderSummary2.rename(columns = {"SN": "Number of Players"})

#Merge 2 Data Frames
GenderSummary = pd.merge(GenderSummary2,GenderSummary1, left_index = True, ri

#Format index
GenderSummary = GenderSummary.reset_index()
GenderSummary = GenderSummary.rename(columns = {"Gender":""})
GenderSummary.set_index('')
```

Out[8]:

	Number of Players	Purchase Count	Avg Purchase Price	Total Revenue
Female	81	113	$3.20	$361.94
Male	484	652	$3.02	$1,967.64
Other / Non-Disclosed	11	15	$3.35	$50.19

Q7- Age Demographics

- Establish bins for ages
- Categorize the existing players using the age bins. Hint: use pd.cut()
- Calculate the numbers and percentages by age group
- Create a summary data frame to hold the results
- Optional: round the percentage column to two decimal points
- Display Age Demographics Table

In [9]:

```
#Subset on SN and Age
AgeDemo = purchase_data[['Age','SN']]

#Drop duplicates
AgeDemo = AgeDemo.drop_duplicates()

#Drop column SN since it's irrelevant now
AgeDemo = AgeDemo[['Age']]

#Categorize Age Group
bins = [0,10,15,20,25,30,35,40,100]
labels = ['<10','10-14','15-19','20-24','25-29','30-34','35-39','40+']
AgeDemo['age_group'] = pd.cut(AgeDemo['Age'], bins = bins, labels = labels, r

#Apply group by to count
AgeDemo = AgeDemo.groupby(by = "age_group").count()

#Rename column Age to Total Count
AgeDemo = AgeDemo.rename(columns = {"Age":"Total Count"})

#Calculate Percentage of Players
AgeDemo['Percentage of Players'] = round(AgeDemo['Total Count']/ AgeDemo['Tota

#Display Result
AgeDemo
```

Out[9]:

age_group	Total Count	Percentage of Players
<10	17	2.95
10-14	22	3.82

age_group	Total Count	Percentage of Players
15-19	107	18.58
20-24	258	44.79
25-29	77	13.37
30-34	52	9.03

Q8- Purchasing Analysis (Age)

- Bin the purchase_data data frame by age
- Run basic calculations to obtain purchase count, avg. purchase price, avg. purchase total per person etc. in the table below
- Create a summary data frame to hold the results
- Optional: give the displayed data cleaner formatting
- Display the summary data frame

In [10]:
```
import warnings #To ignore warning
warnings.filterwarnings("ignore")
```

In [11]:
```
#Subset on Age and Price
AgeAnalysis = purchase_data[['Age','Price']]

#Categorize Age Group by bins and labels set above
AgeAnalysis['age_group'] = pd.cut(AgeAnalysis['Age'], bins = bins, labels = la

#Drop column Age since it's irrelevant now
AgeAnalysis = AgeAnalysis[['age_group','Price']]

#Duplicate Price columns for group by computation
AgeAnalysis['Price2'] = AgeAnalysis['Price']
AgeAnalysis['Price3'] = AgeAnalysis['Price']
AgeAnalysis = AgeAnalysis.groupby('age_group').agg({'Price': 'count','Price2'

#Rename columns
AgeAnalysis = AgeAnalysis.rename(columns = {"Price":"Purchase Count","Price2"

#Round Average Purchase Price Column
AgeAnalysis['Average Purchase Price'] = round(AgeAnalysis['Average Purchase Pr

#Apply $ to Average Purchase Price and Total Purchase Value
dollar_dict = {'Average Purchase Price': '${:,.2f}','Total Purchase Value': '$
AgeAnalysis.style.format(dollar_dict)
```

Out[11]:

age_group	Purchase Count	Average Purchase Price	Total Purchase Value
<10	23	$3.35	$77.13

age_group	Purchase Count	Average Purchase Price	Total Purchase Value
10-14	28	$2.96	$82.78
15-19	136	$3.04	$412.89
20-24	365	$3.05	$1,114.06
25-29	101	$2.90	$293.00
30-34	73	$2.93	$214.00

Q9-Top Spenders

- Run basic calculations to obtain the results in the table below
- Create a summary data frame to hold the results
- Sort the total purchase value column in descending order
- Optional: give the displayed data cleaner formatting
- Display a preview of the summary data frame

In [12]:
```
#Subset on SN and Price
TopSpenders = purchase_data[['SN','Price']]

#Perform group by computations
TopSpenders['Price2'] = TopSpenders['Price']
TopSpenders['Price3'] = TopSpenders['Price']
TopSpenders =  TopSpenders.groupby('SN').agg({'Price': 'count','Price2': 'mea

#Rename column
TopSpenders = TopSpenders.rename(columns = {"Price":"Purchase Count","Price2"

#Round Average Purchase Price Column
TopSpenders['Average Purchase Price'] = round(TopSpenders['Average Purchase P

#Sort by Percent of Total Players
TopSpenders = TopSpenders.sort_values(by = ['Total Purchase Value'], ascendin

#Apply $ sign
TopSpenders['Average Purchase Price'] = TopSpenders['Average Purchase Price']
TopSpenders['Total Purchase Value'] = TopSpenders['Total Purchase Value'].map

#Format index
TopSpenders = TopSpenders.reset_index()
TopSpenders = TopSpenders.rename(columns = {"SN":""})
TopSpenders = TopSpenders.set_index('')
TopSpenders.head(5)
```

Out[12]:

	Purchase Count	Average Purchase Price	Total Purchase Value
Lisosia93	5	$3.79	$18.96

	Purchase Count	Average Purchase Price	Total Purchase Value
Idastidru52	4	$3.86	$15.45
Chamjask73	3	$4.61	$13.83
Iral74	4	$3.40	$13.62

Q10-Most Popular Items

- Retrieve the Item ID, Item Name, and Item Price columns
- Group by Item ID and Item Name. Perform calculations to obtain purchase count, item price, and total purchase value
- Create a summary data frame to hold the results
- Sort the purchase count column in descending order
- Optional: give the displayed data cleaner formatting
- Display a preview of the summary data frame

In [13]:
```
#Create subset and duplicate columns
PopularItems = purchase_data[['Item ID','Item Name','Price']]
PopularItems['Price2'] = PopularItems['Price']
PopularItems['Price3'] = PopularItems['Price']

#Group By to find count, price, and total value
PopularItems = PopularItems.groupby(['Item ID','Item Name']).agg({'Price':'cou

#Rename column for presentation
PopularItems = PopularItems.rename(columns = {"Price":"Purchase Count","Price2

#Sort column and apply $ sign
SortPopularItems = PopularItems.sort_values(by = ['Purchase Count'], ascending
SortPopularItems['Item Price'] = PopularItems['Item Price'].map('${:,.2f}'.for
SortPopularItems['Total Purchase Value'] = PopularItems['Total Purchase Value

#Display value
SortPopularItems.head(5)
```

Out[13]:

Item ID	Item Name	Purchase Count	Item Price	Total Purchase Value
178	**Oathbreaker, Last Hope of the Breaking Storm**	12	$4.23	$50.76
145	**Fiery Glass Crusader**	9	$4.58	$41.22
108	**Extraction, Quickblade Of Trembling Hands**	9	$3.53	$31.77
82	**Nirvana**	9	$4.90	$44.10

Purchase	Item	Total Purchase

Q11-Most Profitable Items

- Sort the above table by total purchase value in descending order
- Optional: give the displayed data cleaner formatting
- Display a preview of the data frame

In [14]:
```
#Sort column and apply $ sign
SortProfitableItems = PopularItems.sort_values(by = ['Total Purchase Value'],
SortProfitableItems['Item Price'] = SortProfitableItems['Item Price'].map('${
SortProfitableItems['Total Purchase Value'] = SortProfitableItems['Total Purcl

#Display value
SortProfitableItems.head(5)
```

Out[14]:

Item ID	Item Name	Purchase Count	Item Price	Total Purchase Value
178	**Oathbreaker, Last Hope of the Breaking Storm**	12	$4.23	$50.76
82	**Nirvana**	9	$4.90	$44.10
145	**Fiery Glass Crusader**	9	$4.58	$41.22
92	**Final Critic**	8	$4.88	$39.04
103	**Singed Scalpel**	8	$4.35	$34.80

Section 2

Q12: Import the covid-19 dataset provided and set the state as the index of the dataframe.

In [15]:
```
covid = pd.read_csv("us_states_covid19_daily.csv", index_col = 'state')
covid.head(5)
```

Out[15]:

state	death	hospitalized	negative	pending	positive	recovered	total
WY	50	272.0	95212	NaN	5948	4791.0	101160
NE	478	2315.0	410594	NaN	45044	33198.0	455638
ND	191	859.0	219646	NaN	21846	17938.0	241492
NC	3532	NaN	2824239	NaN	210632	184422.0	3034871
MT	180	717.0	330087	NaN	13071	9256.0	343158

Q13: Replace all the NaN values in 'pending' column with the string 'None'.

In [16]:
```
covid['pending'].fillna('None', inplace = True)
covid.tail(10)
```

Out[16]:

state	death	hospitalized	negative	pending	positive	recovered	total
CA	15792	NaN	13894577	None	810625	NaN	14705202
WA	2124	7483.0	1767357	None	87042	NaN	1854399
AZ	5650	22119.0	1243641	None	218507	35261.0	1462148
WI	1337	7300.0	1423247	190	129123	99925.0	1552370
AS	0	NaN	1571	None	0	NaN	1571
WV	350	NaN	545720	None	15848	11507.0	561568
AR	1369	5354.0	949107	None	83697	75312.0	1032804
MS	2969	5844.0	638843	None	98190	89737.0	737033
AL	2540	17257.0	994475	None	154701	67948.0	1149176
AK	56	NaN	448427	None	8780	4555.0	457207

Q14: Replace the NaN values in the 'hospitalized' columns with the mean value of the column.

In [17]:
```
hospitalized = covid['hospitalized']
hospitalized_mean = hospitalized.mean()
covid['hospitalized'].fillna(hospitalized_mean, inplace = True)
covid.head(10)
```

Out[17]:

state	death	hospitalized	negative	pending	positive	recovered	total
WY	50	272.000000	95212	None	5948	4791.0	101160
NE	478	2315.000000	410594	None	45044	33198.0	455638
ND	191	859.000000	219646	None	21846	17938.0	241492
NC	3532	11044.567568	2824239	None	210632	184422.0	3034871
MT	180	717.000000	330087	None	13071	9256.0	343158
NH	439	738.000000	265447	None	8266	7522.0	273713
MP	2	4.000000	15112	None	70	29.0	15182
MO	2118	11044.567568	1178272	None	126113	NaN	1304385
MN	2089	7701.000000	1315567	None	99134	89392.0	1414701
MI	7083	11044.567568	3509848	None	138014	95051.0	3647862

Q15: Find the death percentage (round to 2 decimal points) of covid positive

patients for each state and add it as a column in the dataset. (death% = death/positive)

In [18]:
```
covid['death percentage'] = round(covid['death']/covid['positive']*100,2)
covid['death percentage'] = covid['death percentage'].map('{:,.2f}%'.format)
covid['death percentage']
```

Out[18]:
```
state
WY    0.84%
NE    1.06%
ND    0.87%
NC    1.68%
MT    1.38%
NH    5.31%
MP    2.86%
MO    1.68%
MN    2.11%
MI    5.13%
ME    2.62%
NJ    7.86%
MD    3.17%
NM    3.00%
MA    7.16%
NV    2.00%
LA    3.29%
NY    5.56%
KY    1.71%
OH    3.12%
KS    1.13%
OK    1.18%
IN    3.03%
OR    1.67%
IL    3.01%
PA    5.12%
ID    1.12%
PR    1.36%
IA    1.57%
RI    4.50%
HI    1.07%
SC    2.28%
GU    1.97%
SD    1.00%
GA    2.21%
TN    1.25%
FL    2.05%
TX    2.10%
DE    3.09%
UT    0.63%
DC    4.09%
VA    2.16%
CT    7.83%
VI    1.52%
CO    2.79%
VT    3.31%
CA    1.95%
WA    2.44%
AZ    2.59%
WI    1.04%
AS     nan%
WV    2.21%
AR    1.64%
MS    3.02%
AL    1.64%
AK    0.64%
Name: death percentage, dtype: object
```

Q16: Print the total number of negative, positive, and recovered cases in the US.

In [19]:
```
print('Total Number of negative cases: ',covid['negative'].sum())
print('Total Number of positive cases: ',covid['positive'].sum())
print('Total Number of positive cases: ',covid['recovered'].sum())
```

```
Total Number of negative cases:  92955835
Total Number of positive cases:  7198622
Total Number of positive cases:  2840747.0
```

Q17: What percent of the covid positive patients from the state of NY were hospitalized (round the value to 2 decimal places)?

In [20]:
```
NewYork = covid.loc['NY']
Q17 = round(NewYork.hospitalized/NewYork.positive*100,2)
print('Percentage of covid positive patitents from the state of NY were hospit
```

```
Percentage of covid positive patitents from the state of NY were hospitalized:
19.62 %
```

Q18: Return the state name with the maximum number of covid positive cases

In [21]:
```
maxpositive = covid['positive'].max()
Q18 = covid[covid['positive'] == maxpositive]
Q18 = Q18.index.tolist()
print(str(Q18).strip('[]'))
```

```
'CA'
```

Q19: Return the number of states with more than the mean value of positive cases

In [22]:
```
#Find average positive cases
positive = covid['positive']
positive_mean = positive.mean()

#Count numbers of state with more than mean value of positive cases
count = 0
for i in positive:
    if i > positive_mean:
        count = count+1

#Display value
print('Number of states with more than the mean value of positive cases: ',cou
```

```
Number of states with more than the mean value of positive cases:  19
```

Q20: Print the data of those states whose no of negative cases is greater than the number of negative cases for MT. Then sort the dataframe wrt the total column values in descending order.

In [23]:
```
MT = covid._get_value('MT','negative')
Q20 = covid[covid['negative'] > MT]
Q20.sort_values(by = ['total'], ascending = False)
```

Out[23]:

state	death	hospitalized	negative	pending	positive	recovered	total	death percentage
CA	15792	11044.567568	13894577	None	810625	NaN	14705202	1.95%
NY	25479	89995.000000	10288664	None	458649	76754.0	10747313	5.56%

state	death	hospitalized	negative	pending	positive	recovered	total	death percentage
TX	15711	11044.567568	5488190	None	748967	664883.0	6237157	2.10%
IL	8916	11044.567568	5331548	None	295763	NaN	5627311	3.01%
FL	14488	44608.000000	4594668	3980	706516	NaN	5301184	2.05%
MI	7083	11044.567568	3509848	None	138014	95051.0	3647862	5.13%
NJ	16126	23439.000000	3407757	None	205275	34859.0	3613032	7.86%
OH	4804	15516.000000	3011948	None	153987	132980.0	3165935	3.12%
NC	3532	11044.567568	2824239	None	210632	184422.0	3034871	1.68%
GA	7021	28522.000000	2615653	None	318026	NaN	2933679	2.21%
TN	2454	8733.000000	2688217	None	196139	179322.0	2884356	1.25%
LA	5511	11044.567568	2151146	None	167458	154163.0	2318604	3.29%
MA	9456	12686.000000	2104449	None	132116	113768.0	2236565	7.16%
VA	3208	18541.000000	1909374	314	148271	17633.0	2057645	2.16%
PA	8142	11044.567568	1879127	None	158967	130352.0	2038094	5.12%
WA	2124	7483.000000	1767357	None	87042	NaN	1854399	2.44%
MD	3949	15530.000000	1493282	None	124725	7536.0	1618007	3.17%
CT	4508	11560.000000	1536443	None	57550	9310.0	1593993	7.83%
WI	1337	7300.000000	1423247	190	129123	99925.0	1552370	1.04%
AZ	5650	22119.000000	1243641	None	218507	35261.0	1462148	2.59%
MN	2089	7701.000000	1315567	None	99134	89392.0	1414701	2.11%
KY	1174	5279.000000	1315769	None	68840	11840.0	1384609	1.71%
IN	3632	12814.000000	1259831	None	120019	94816.0	1379850	3.03%
SC	3378	9160.000000	1161156	None	147942	71691.0	1309098	2.28%
MO	2118	11044.567568	1178272	None	126113	NaN	1304385	1.68%
OK	1031	6449.000000	1116829	4926	87199	73100.0	1204028	1.18%
AL	2540	17257.000000	994475	None	154701	67948.0	1149176	1.64%
AR	1369	5354.000000	949107	None	83697	75312.0	1032804	1.64%
NM	875	3475.000000	885829	None	29157	16565.0	914986	3.00%
CO	1952	7558.000000	837127	None	70025	6500.0	907152	2.79%
UT	459	3847.000000	756928	None	73042	55141.0	829970	0.63%
IA	1344	11044.567568	679271	None	85596	68456.0	764867	1.57%
MS	2969	5844.000000	638843	None	98190	89737.0	737033	3.02%
NV	1600	11044.567568	618116	None	79980	2220.0	698096	2.00%

state	death	hospitalized	negative	pending	positive	recovered	total	death percentage
OR	559	2598.000000	652364	None	33509	5720.0	685873	1.67%
WV	350	11044.567568	545720	None	15848	11507.0	561568	2.21%
KS	678	2917.000000	461701	None	59749	2095.0	521450	1.13%

Q21: Create a new column called recovery_rate which classifies a state as-

a) 'High Recovery' when recovered% > 50
b) 'Modearate Recovery' when recovered% is between 25 and 50
c) 'Low Recovery' when recovered% is less than 25

where recovered% = recovered / positive for each state

In [24]:
```
#Find in misssing recover with 0
covid['recovered'].fillna(0, inplace = True)

#Create recover% = recovered / positive for each state
covid['recover%'] = round(covid['recovered']/covid['positive']*100,2)

#Categorize recover% group to recovery_rate
bins2 = [0,25,50,100]
labels2 = ['Low Recovery','Moderate Recovery','High Recovery']
covid['recovery_rate'] = pd.cut(covid['recover%'], bins = bins2, labels = labe

#Display value
covid[['recovered','positive','recover%','recovery_rate']].head(20)
```

Out[24]:

state	recovered	positive	recover%	recovery_rate
WY	4791.0	5948	80.55	High Recovery
NE	33198.0	45044	73.70	High Recovery
ND	17938.0	21846	82.11	High Recovery
NC	184422.0	210632	87.56	High Recovery
MT	9256.0	13071	70.81	High Recovery
NH	7522.0	8266	91.00	High Recovery
MP	29.0	70	41.43	Moderate Recovery
MO	0.0	126113	0.00	Low Recovery
MN	89392.0	99134	90.17	High Recovery
MI	95051.0	138014	68.87	High Recovery
ME	4678.0	5391	86.77	High Recovery
NJ	34859.0	205275	16.98	Low Recovery
MD	7536.0	124725	6.04	Low Recovery

state	recovered	positive	recover%	recovery_rate
NM	16565.0	29157	56.81	High Recovery
MA	113768.0	132116	86.11	High Recovery
NV	2220.0	79980	2.78	Low Recovery
LA	154163.0	167458	92.06	High Recovery
NY	76754.0	458649	16.73	Low Recovery

Plotting pandas using IRIS dataset

Note: This problem makes use of the iris data set and depends upon your having completed the previous problem, so please do that first.

In [25]:
```
import matplotlib.pyplot as plt
import seaborn as sns
iris = pd.read_csv('iris.csv')
iris['Name'] = iris['Name'].str.replace("Iris-","") #Remove iris-
iris.head()
```

Out[25]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

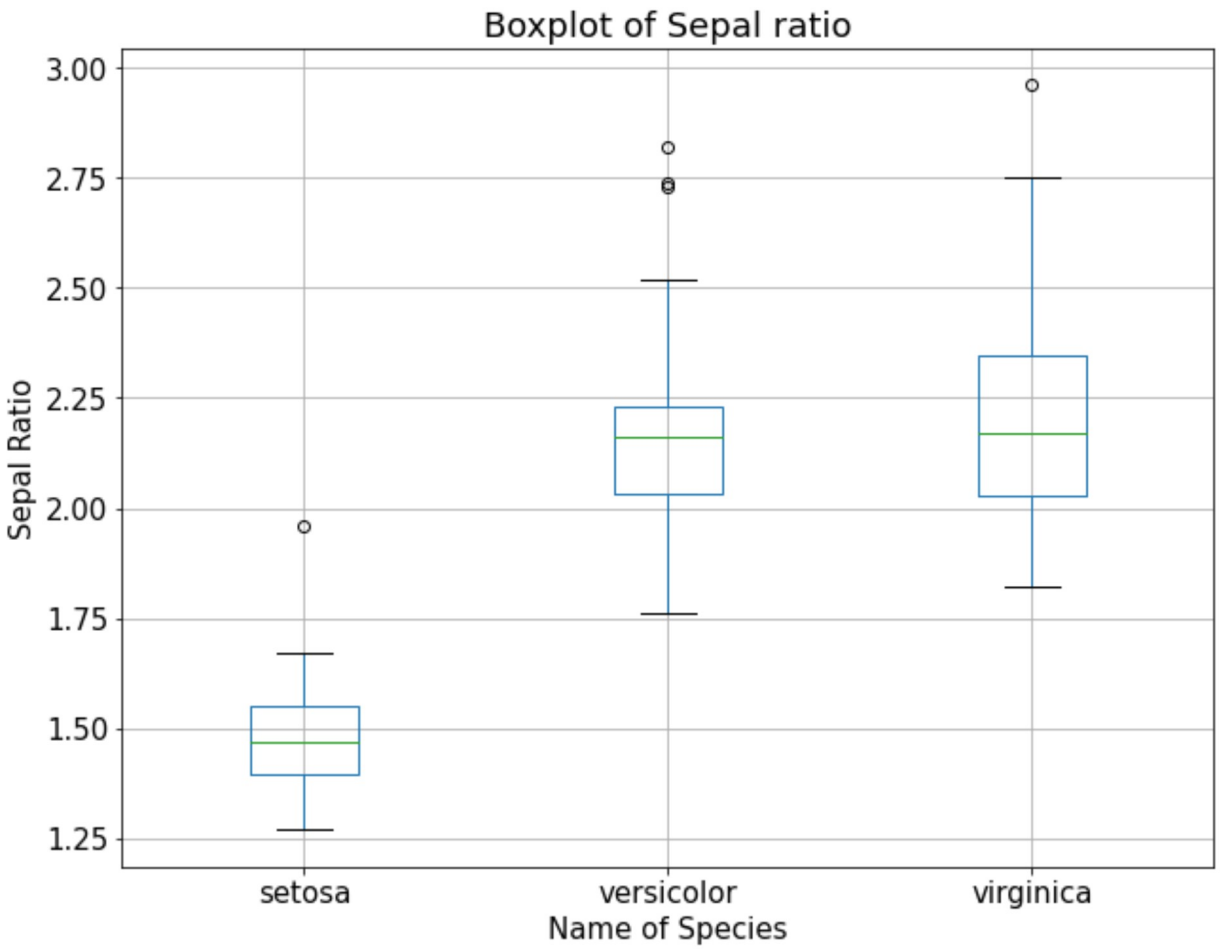
Q22: Use the built-in pandas plotting tools to make a box-and-whisker plot showing the distribution of petal ratio and sepal ratio for each of the three species. Your plot should have two subplots, one for petal ratio and one for sepal ratio. You may choose the details of your plots (i.e., how to handle outliers, displaying mean vs median, etc) however you think is best. Please include labels on your x- and y-axes and give an appropriate title to your plot.

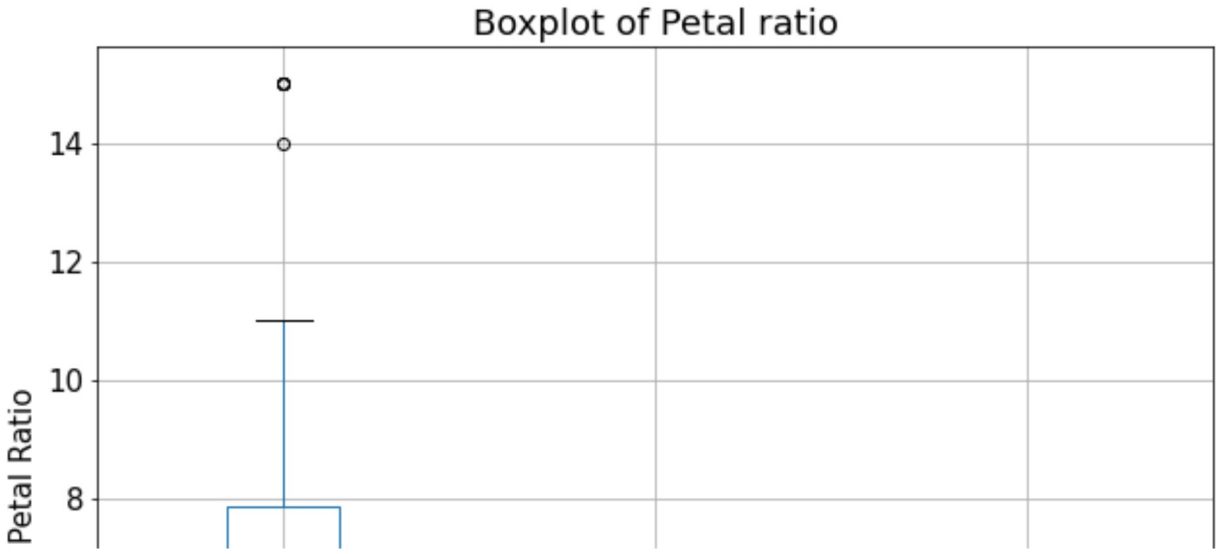
In [26]:
```
iris['SepalRatio'] = round(iris['SepalLength']/iris['SepalWidth'],2)
iris['PetalRatio'] = round(iris['PetalLength']/iris['PetalWidth'],2)
iris.head()
```

Out[26]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name	SepalRatio	PetalRatio
0	5.1	3.5	1.4	0.2	setosa	1.46	7.0
1	4.9	3.0	1.4	0.2	setosa	1.63	7.0
2	4.7	3.2	1.3	0.2	setosa	1.47	6.5
3	4.6	3.1	1.5	0.2	setosa	1.48	7.5
4	5.0	3.6	1.4	0.2	setosa	1.39	7.0

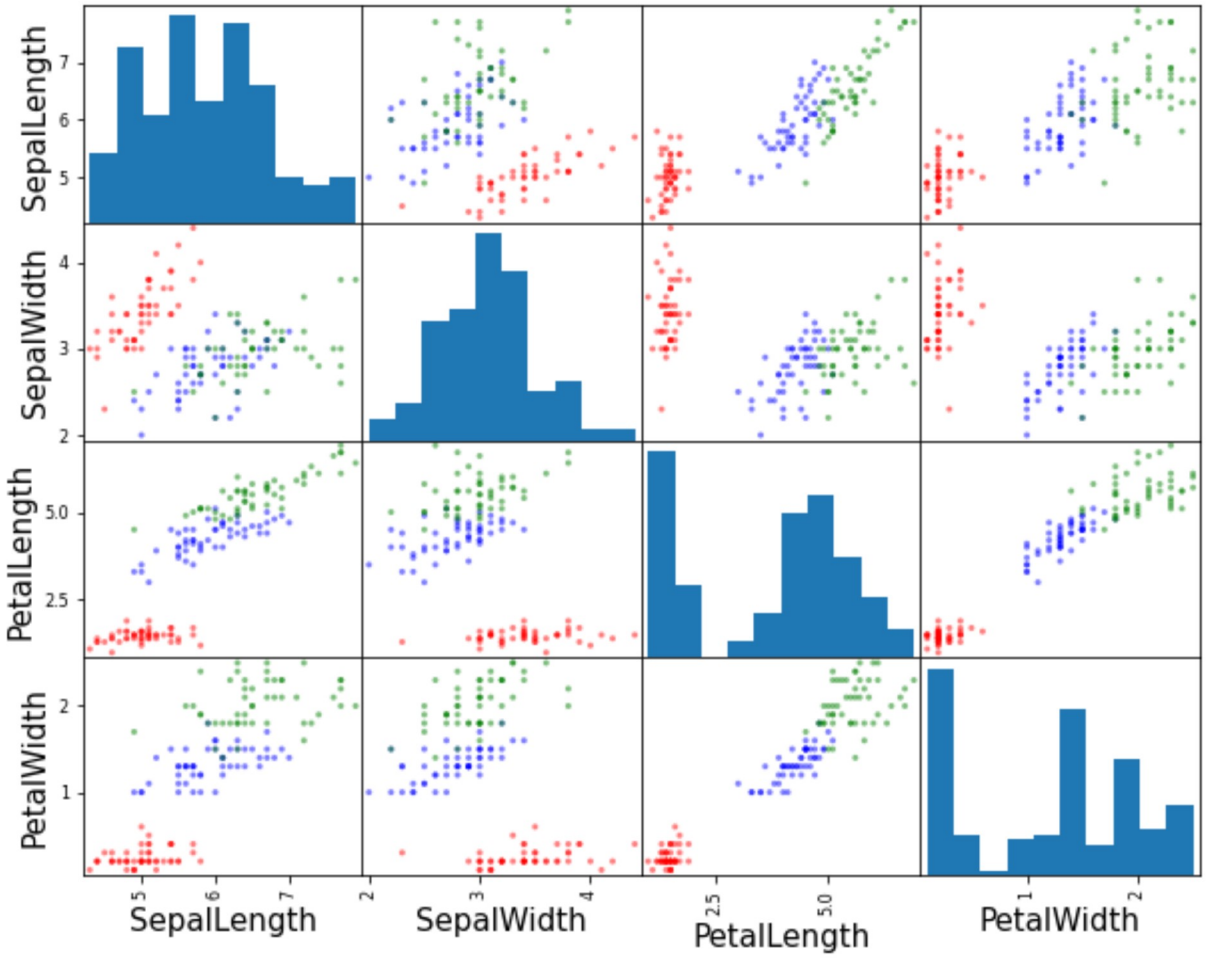
In [27]:
```
plt.rcParams.update({'font.size':15, 'figure.figsize':(10,8)})
ax1 = iris.boxplot(by = 'Name', column = ['SepalRatio'])
plt.title("Boxplot of Sepal ratio")
plt.suptitle("")
ax1.set_xlabel('Name of Species')
ax1.set_ylabel('Sepal Ratio')
plt.show()

ax2 = iris.boxplot(by = 'Name', column = ['PetalRatio'])
plt.title("Boxplot of Petal ratio")
plt.suptitle("")
ax2.set_xlabel('Name of Species')
ax2.set_ylabel('Petal Ratio')
plt.show()
```





Q23: Use the built-in pandas plotting tools to make a scatter matrix plot for the four original features (petal width, petal length, sepal width and sepal length). Each point in the scatter plot should be colored according to its species. Hint: see the documentation at https://pandas.pydata.org/pandas-docs/stable/visualization.html# scatter-matrix-plot to get started.

In [28]:
```
from pandas.plotting import scatter_matrix
iris = pd.read_csv('iris.csv')
iris['Name'] = iris['Name'].str.replace("Iris-","") #Remove iris-
colors = iris['Name'].replace({'setosa':'red', 'virginica': 'green', 'versicol
pd.plotting.scatter_matrix(iris, c = colors, cmap='viridis');
```



Load the stockprice dataset

In [29]:
```
stock = pd.read_csv("timeseries_stockprice.csv", header=0) #add header
stock.head()
```

Out[29]:

	Date	Amazon	Google	Facebook
0	5/1/18	927.800	901.94	151.74
1	5/2/18	946.645	909.62	153.34
2	5/3/18	946.000	914.86	153.60
3	5/4/18	944.750	926.07	150.17
4	5/7/18	940.520	933.54	151.45

Q24-Plotting a single series

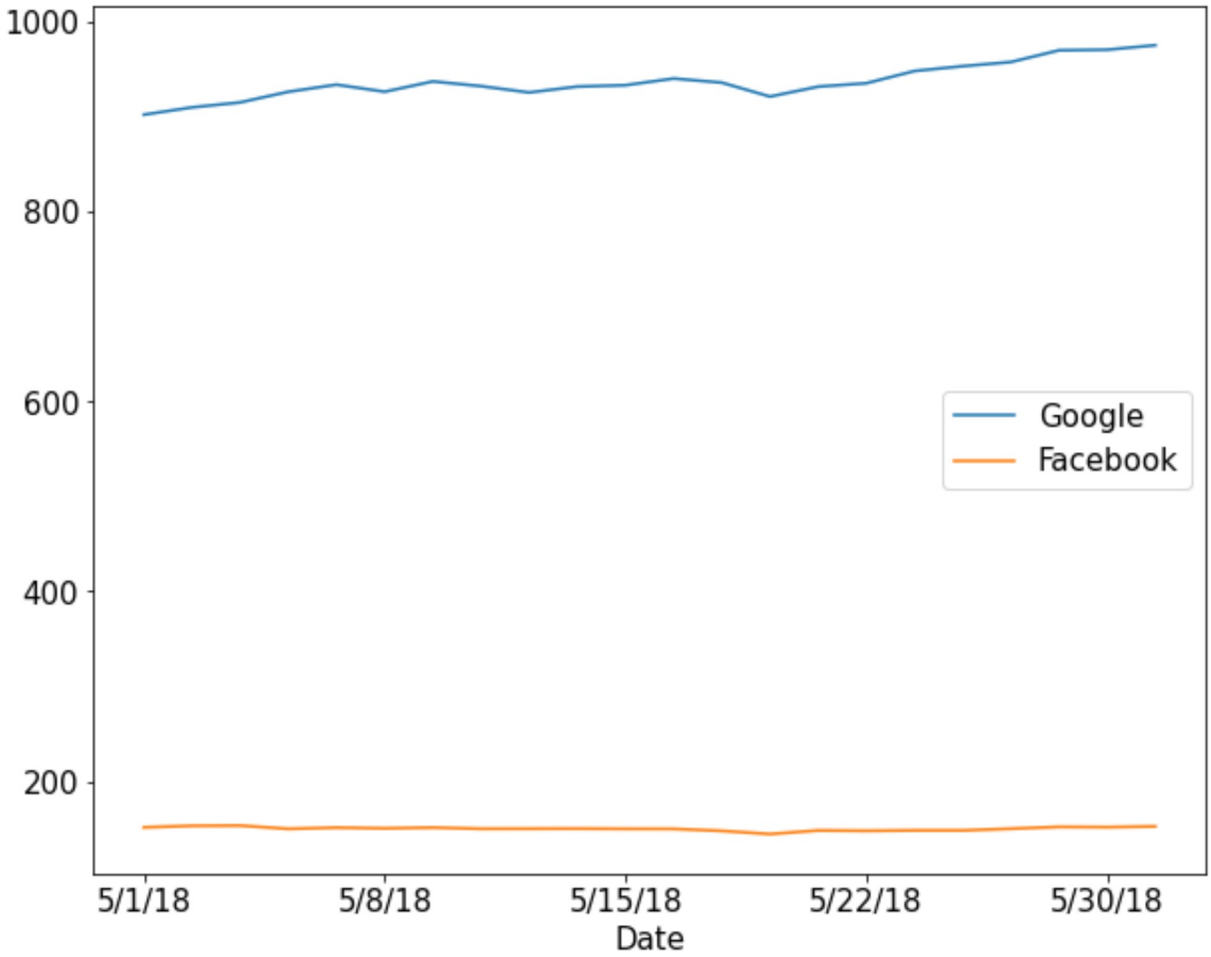
To plot a single series, such as the closing stock price for Amazon for each date, you can simple designate the x-axis values as Date and the y-axis values as Amazon and set the kind parameter to line.

In [30]:
```
stock.plot.line(x= "Date", y= "Amazon");
```

Q25-Plotting two series

When plotting two series, set the y-axis values to a list containing the two column names, each being a data series.

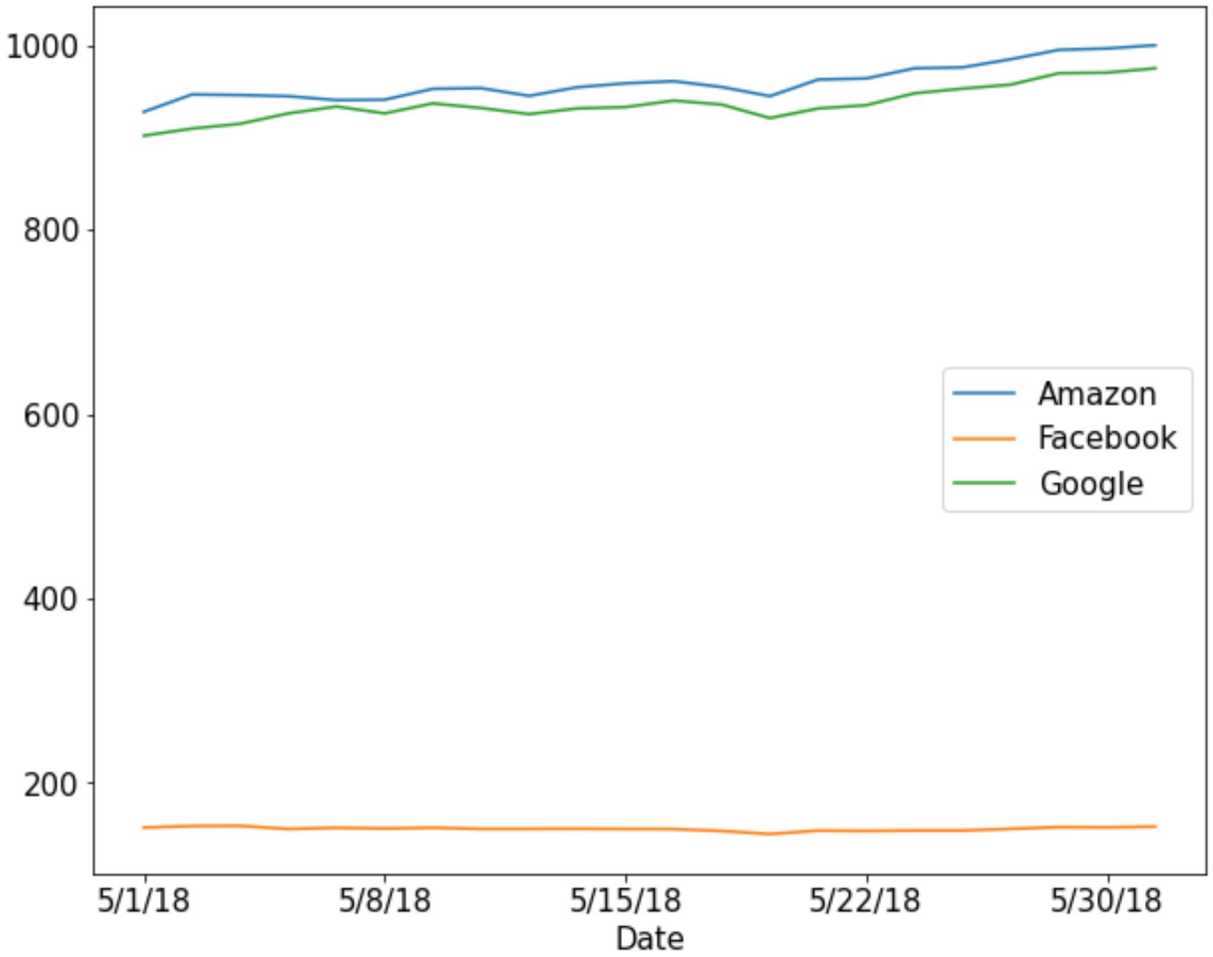
In [31]:
```
stock.plot(kind = "line", x = "Date", y = ["Google", "Facebook"]);
```



Q26- Plotting three series

When plotting three series, set the y-axis values to a list containing the three column names, each being a data series.

In [32]:
```
stock.plot(kind = "line", x = "Date", y = ["Amazon","Facebook","Google"]);
```



Load the following dataset

In [33]:
```
import pandas as pd
mydata = pd.read_csv("mba.csv", header=0, index_col ="School") #add header an
mydata.head()
```

Out[33]:

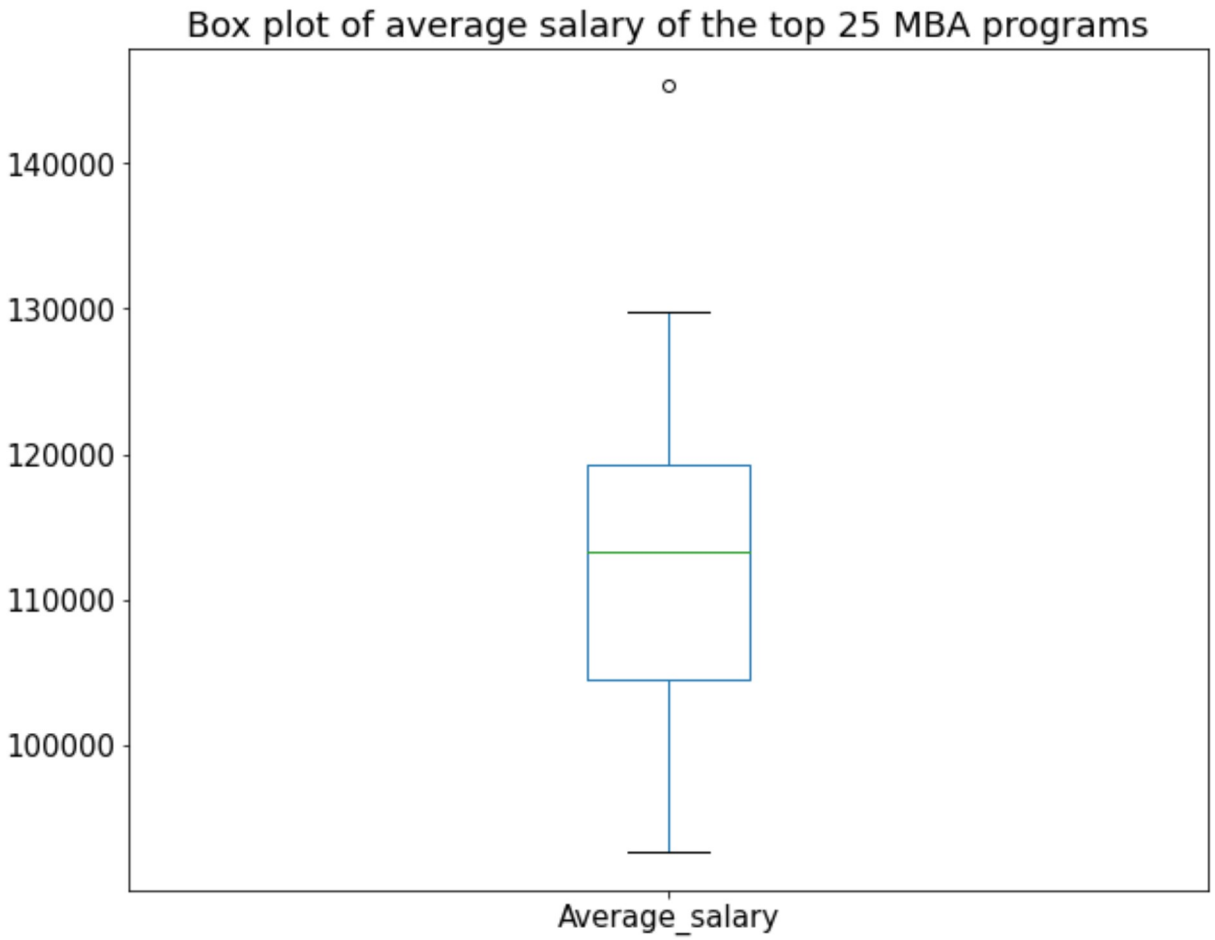
School	Rank	Country	Average_salary	Pre_Salary	Grad_Jobs	PhD	Avg_Age_Students	Avg_Wo
Chicago (Booth)	1	US	113217	63	93	96	27	
Dartmouth (Tuck)	2	US	115143	62	97	100	28	
Virginia (Darden)	3	US	104478	67	95	97	27	
Harvard	4	US	121785	44	97	94	27	
Columbia	5	US	113340	55	98	97	28	

Q27-Plot the Box plot using both approach available in pandas:

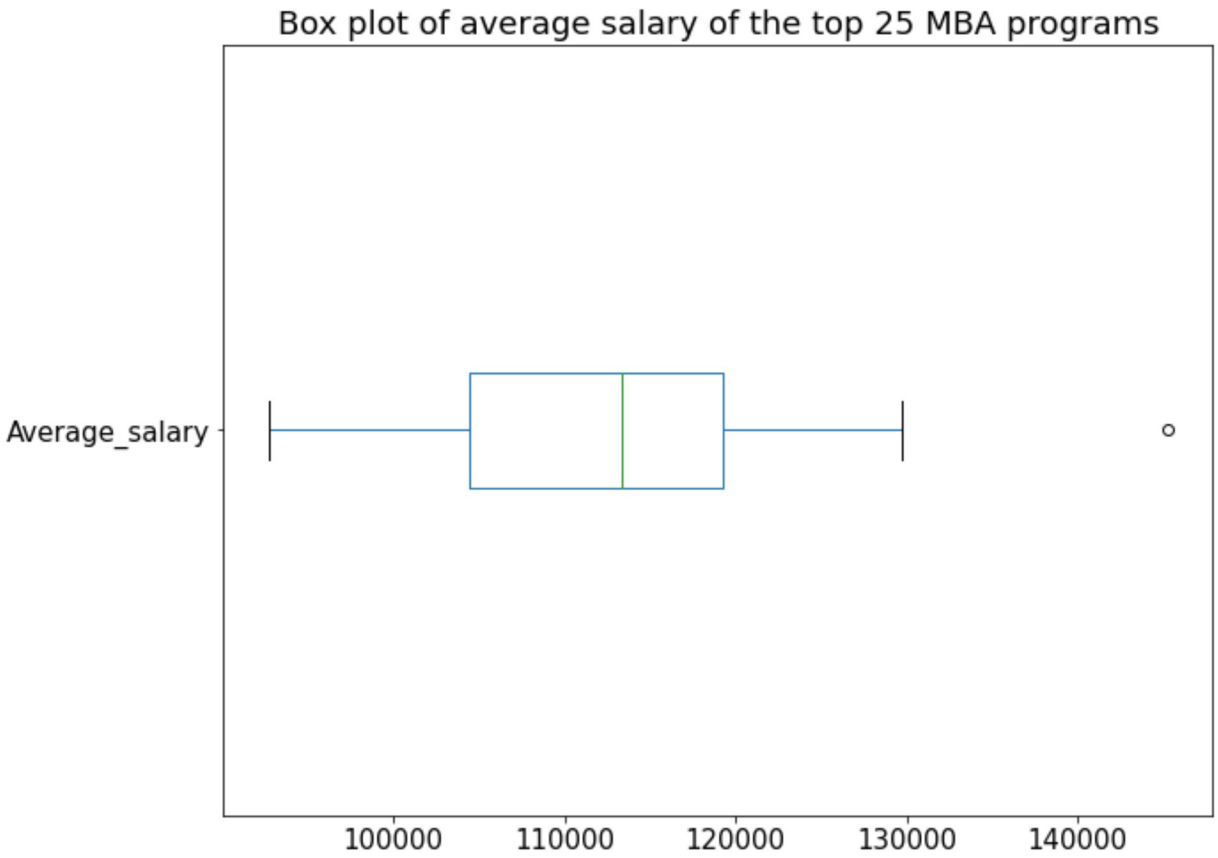
* .plot

* boxplot

In [34]:
```
mydata['Average_salary'].plot(kind= "box", title="Box plot of average salary
```

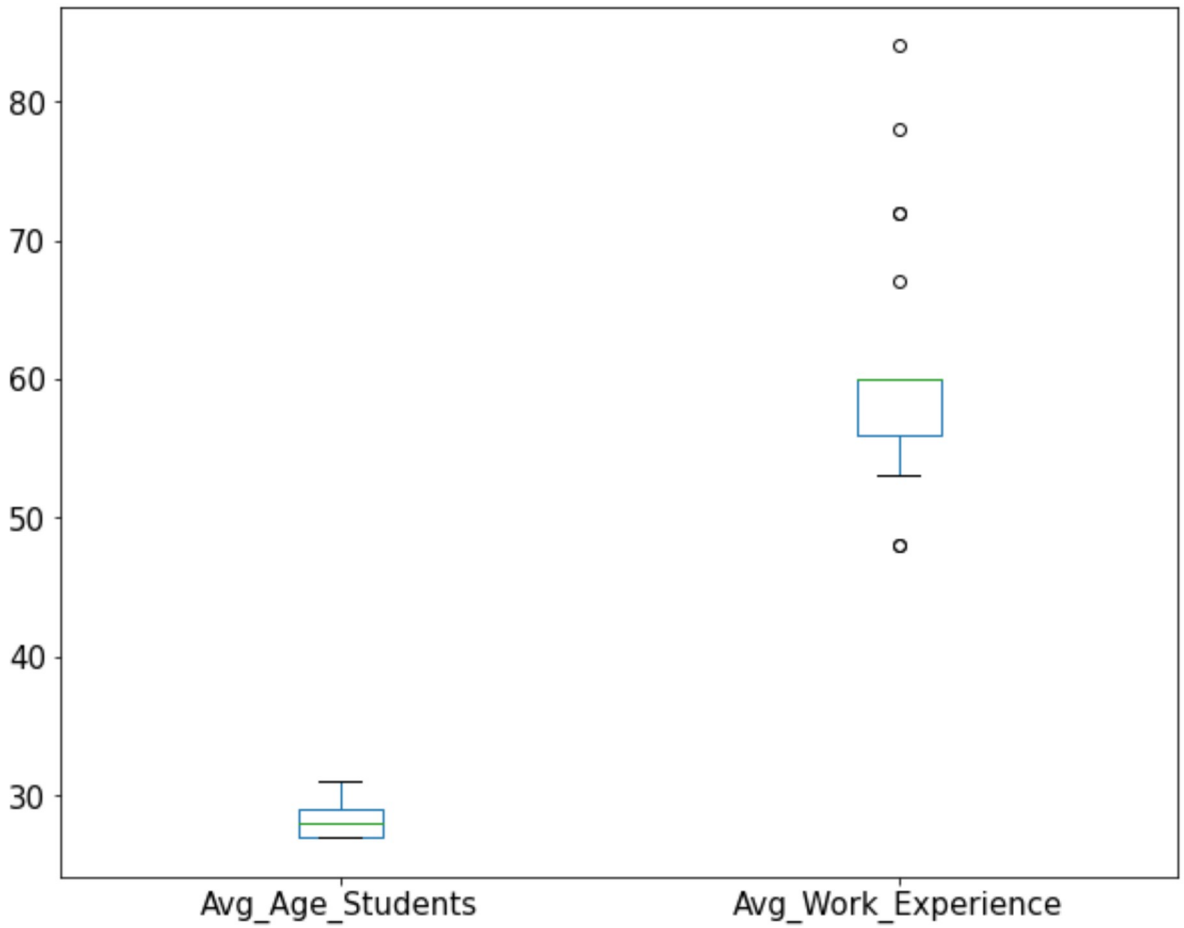


In [35]:
```
mydata['Average_salary'].plot.box(vert = False, title = "Box plot of average
```



Q28- creat a box plot of two variables: "the average student" and "average work experience" in one graph (side by side)

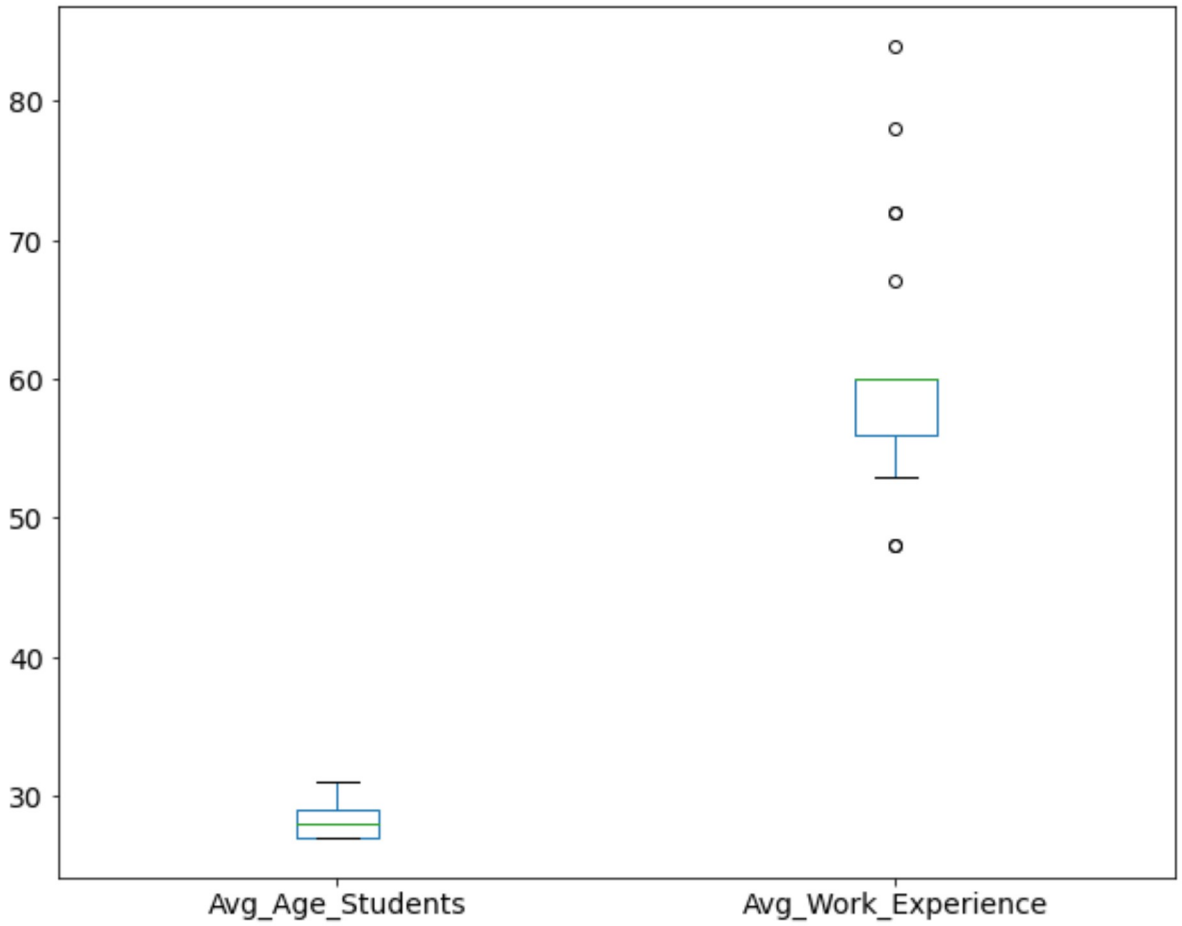
In [36]:
```
mydata[['Avg_Age_Students','Avg_Work_Experience']].plot.box();
```



Q29- Formatting with .boxplot().

change the following parameters for the previous question as grid=False and fontsize=14:

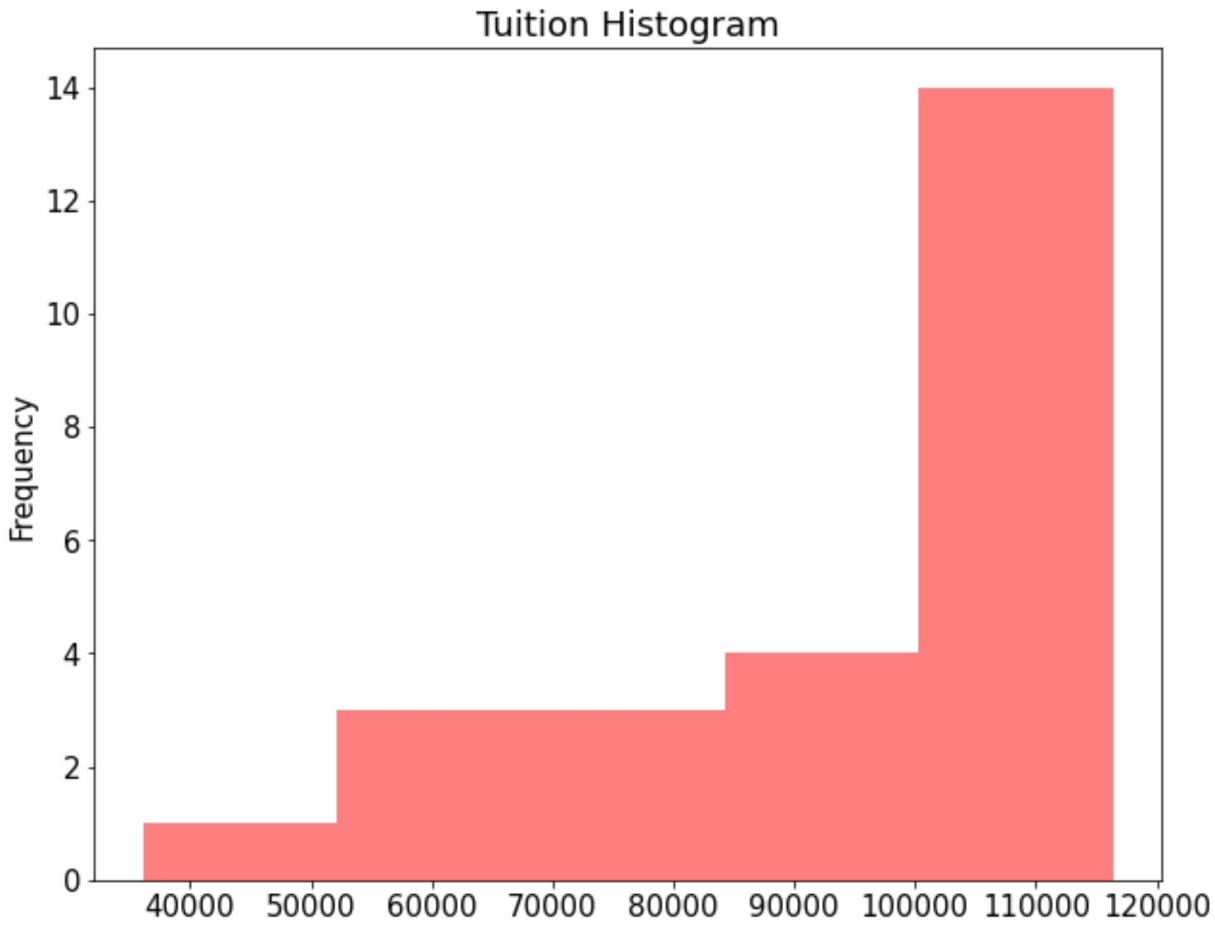
In [37]:
```
mydata[['Avg_Age_Students','Avg_Work_Experience']].plot.box(grid=False, fonts
```



Q30-Histograms

plot the histogram of Tuitions with proper bins

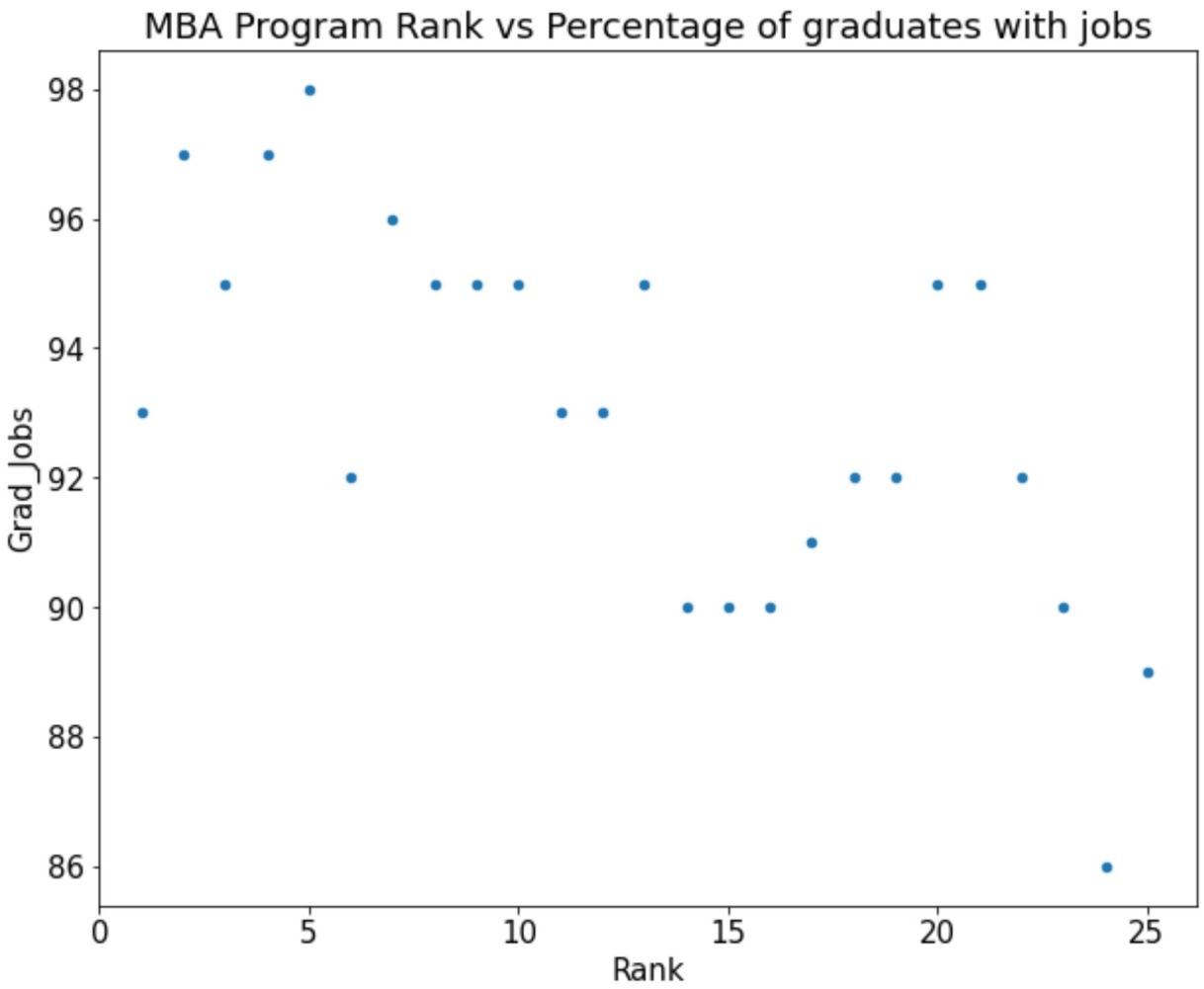
In [38]:
```
mydata['Tuition'].plot(kind = 'hist', title = 'Tuition Histogram', color = 'r
```



Q31-Scatter plots

plot the scatter plot using Rank (x-axis) and Grad_job (y-axis) and set the proper xlim

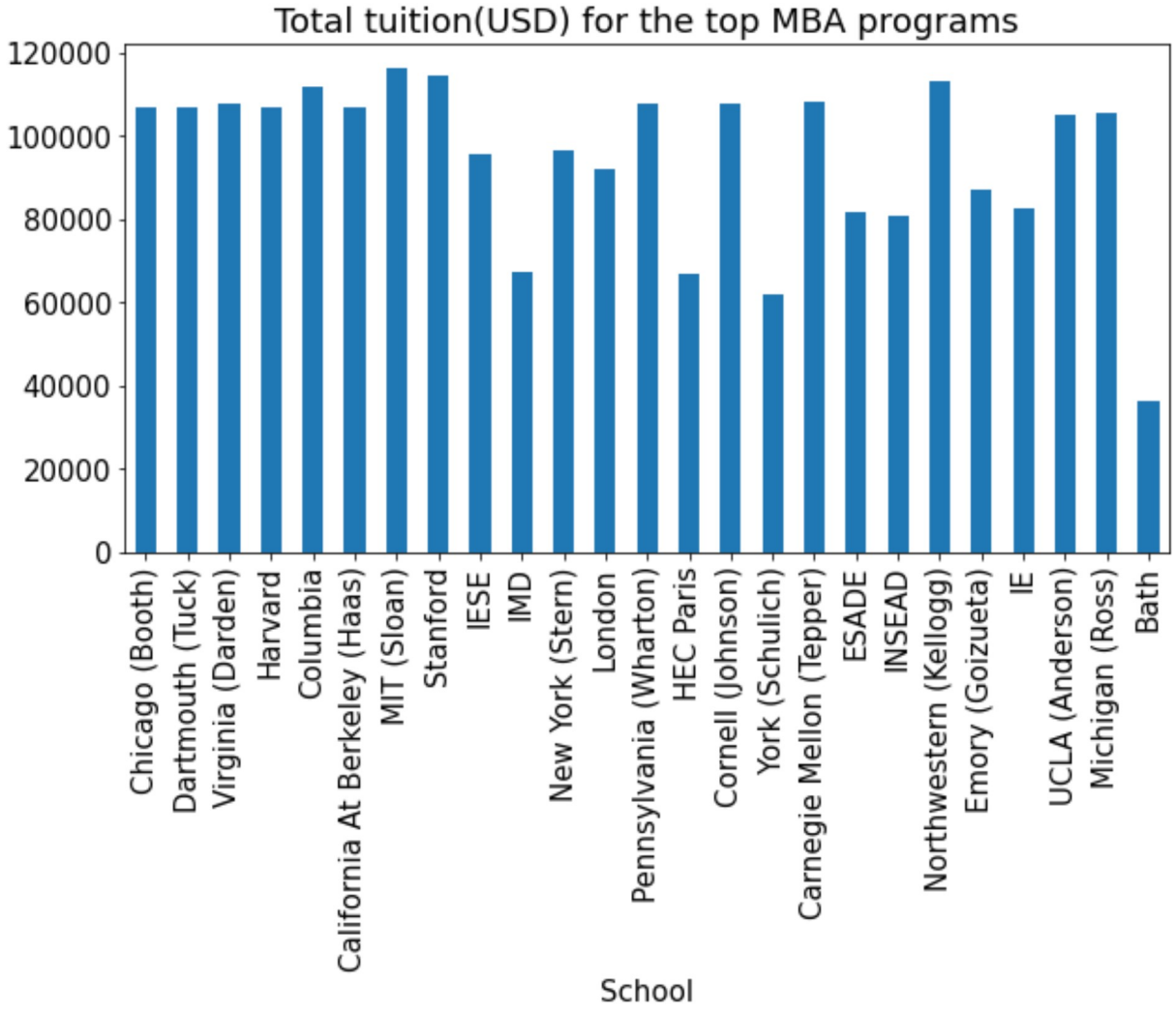
In [39]:
```
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
mydata.plot.scatter(x = 'Rank', y = 'Grad_Jobs',
                    title='MBA Program Rank vs Percentage of graduates with j
```



Q32-Bar charts

plot horrizontal bar graph for Tuition and assign a proper title for your graph

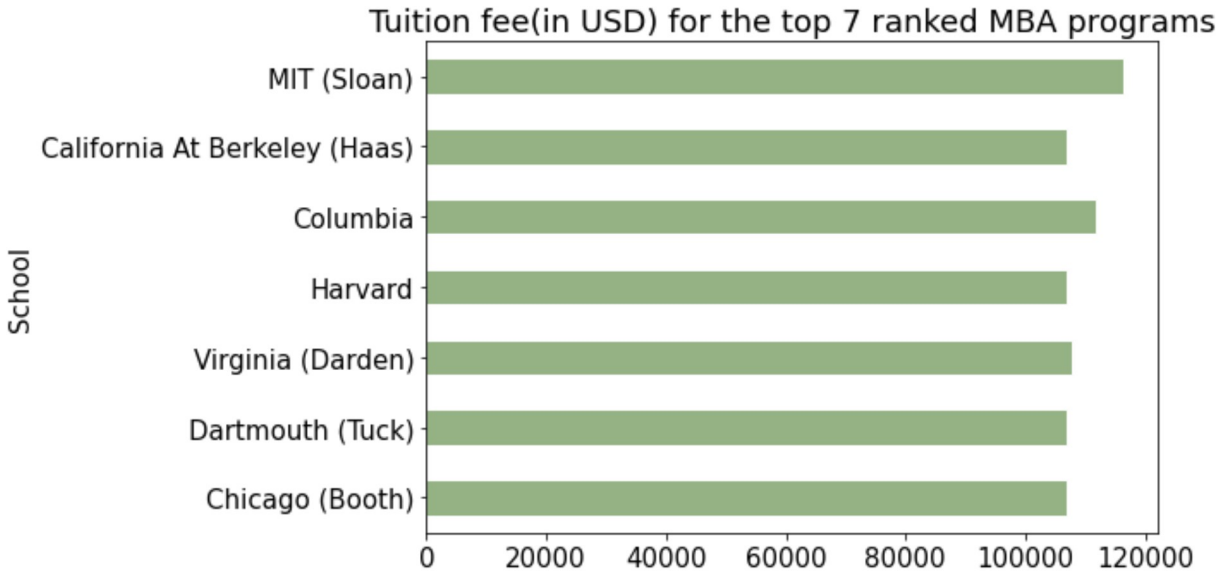
In [42]:
```
plt.rcParams["figure.figsize"]=10,5
mydata['Tuition'].plot.bar(title='Total tuition(USD) for the top MBA programs
```



Q33-Refine the bar chart with a few aesthetic elements.

- The legend parameter is set to False to remove the redundant legend.
- The color parameter can set the color values using RGB. RGB is a way of making colors. You have to provide an amount of red, green and blue + the transparency and it returns a color.
- The edgecolor parameter allows you to set the border of the bars.

In [43]:
```
from matplotlib import rcParams
rcParams.update({'figure.autolayout': True}) # keeps labels cutoff
mydata.head(7).plot(kind='barh', y = 'Tuition', title='Tuition fee(in USD) fo
                    legend=False, color=(0.3, 0.5, 0.2, 0.6), edgecolor='white
```


Tuition fee(in USD) for the top 7 ranked MBA programs
School
MIT (Sloan)
California At Berkeley (Haas)
Columbia
Harvard
Virginia (Darden)
Dartmouth (Tuck)
Chicago (Booth)
0
20000
40000
60000
80000
100000
120000