```
#Calculator using operators and if else statement
print("Calculator")
print("1.Add")
print("2.Substract")
print("3.Multiply")
print("4.Divide")
#calculator code
calculator=int(input("Enter Choice(6-9): "))

if calculator==6:
    a=int(input("Enter A:"))
    b=int(input("Enter B:"))
    c=a+b
    print("Sum = ",c)
elif calculator==7:
    a=int(input("Enter C:"))
    b=int(input("Enter D:"))
    c=a-b
    print("Difference = ",c)
elif  calculator==8:
    a=int(input("Enter E:"))
    b=int(input("Enter F:"))
    c=a*b
    print("Product = ",c)
elif calculator==9:
    a=int(input("Enter G:"))
    b=int(input("Enter H:"))
    c=a/b
    print("Quotient = ",c)
else:
    print("None")
```

```
Calculator
1.Add
2.Substract
3.Multiply
4.Divide
Enter Choice(6-9): 8
Enter E:77
Enter F:7
Product =  539
```

```
#new
```

```
                                          #Sales Analysis
                                          #Task No_2
""" Importing the libraries """
import numpy as np
import pandas as pd
```

```
!pwd
```

```
/content
```

Double-click (or enter) to edit

```
!mkdir my_project
```

```
cd my_project
```

```
/content/my_project
```

New Section

```
from google.colab import files
upload=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving annex1.csv to annex1.csv
```

```
from google.colab import files
upload=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving annex2.csv to annex2.csv
```

```
from google.colab import files
upload=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving annex3.csv to annex3.csv
```

```
from google.colab import files
upload=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving annex4.csv to annex4.csv
```

```
cd ..
```

```
/content
```

```
cd my_projects
```

```
[Errno 2] No such file or directory: 'my_projects'
/content
```

```
cd my_project
```

```
/content/my_project
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
item_category = pd.read_csv("/content/my_project/annex1.csv")
```

```
sales = pd.read_csv("/content/my_project/annex2.csv")
wholesale = pd.read_csv("/content/my_project/annex3.csv")
loss_rate = pd.read_csv("/content/my_project/annex4.csv")
```

```
"""from google.colab import drive
drive.mount('/content/drive')"""
```

```
'from google.colab import drive\ndrive.mount('/content/drive')'
```

```
item_category.head() # head means --- it give the 1st five rows of the table
```

	Item Code	Item Name	Category Code	Category Name
0	102900005115168	Niushou Shengcai	1011010101	Flower/Leaf Vegetables
1	102900005115199	Sichuan Red Cedar	1011010101	Flower/Leaf Vegetables
2	102900005115625	Local Xiaomao Cabbage	1011010101	Flower/Leaf Vegetables
3	102900005115748	White Caitai	1011010101	Flower/Leaf Vegetables
4	102900005115762	Amaranth	1011010101	Flower/Leaf Vegetables

```
print("Checking rows & columns, Rows={}, Columns={}".format(item_category.shape[0],item_category.shape[1]))
print(item_category.nunique()) # Finding the number of unique elements present in iteam_category
```

```
Checking rows & columns, Rows=251, Columns=4
Item Code        251
Item Name        247
Category Code      6
Category Name      6
dtype: int64
```

```
item_category.isnull().sum() # checking the null values in all the columns
```

```
Item Code        0
Item Name        0
Category Code    0
Category Name    0
dtype: int64
```

```
item_category.duplicated().sum() # count the dupicated numbers
```

```
0
```

```
# finding the unique elements from all columns
for i in item_category.columns:
    print(i)
    print(item_category[i].unique())
    print("************************************************************************************")
```

```
Item Code
[102900005115168 102900005115199 102900005115625 102900005115748
 102900005115762 102900005115779 102900005115786 102900005115793
 102900005115816 102900005115823 102900005115854 102900005115861
 102900005115878 102900005115885 102900005115908 102900005115946
 102900005115960 102900005115977 102900005115984 102900005116639
 102900005116776 102900005116790 102900005116806 102900005118572
 102900005118817 102900005118831 102900005119975 102900005122654
 102900005128748 102900011000175 102900011000571 102900011002414
 102900011006689 102900011006948 102900011006955 102900011007464
 102900011007471 102900011007495 102900011008133 102900011008164
 102900011008485 102900011008492 102900011008515 102900011008522
 102900011008676 102900011015384 102900011015391 102900011021644
 102900011022849 102900011022924 102900011023464 102900011026502
 102900011026618 102900011027462 102900011027615 102900011029688
 102900011030042 102900011030059 102900011030097 102900011030103
 102900011030110 102900011030134 102900011030141 102900011030158
 102900011030400 102900011030417 102900011030905 102900011031216
 102900011032176 102900011032282 102900011032480 102900011032589
 102900011032787 102900011033081 102900011033173 102900011033234
 102900011033241 102900011033531 102900011033562 102900011033586
 102900011033906 102900011033920 102900011034200 102900011034217
 102900011034224 102900011034231 102900011034316 102900011034323
 102900011034354 102900011035481 102900011035764 102900011035771
 102900011035849 102900011036686 102900051000890 102900051009220
 102900051010455 102900051010790 106971563780002 106972776821582
 102900005116714 102900011000632 102900011009970 102900011033913
 102900011034026 102900005116042 102900005116899 102900005118824
 102900011001561 102900011001691 102900011007969 102900011009277
 102900011010891 102900011018132 102900011021842 102900011023976
 102900011024010 102900011032114 102900011032732 102900011034569
 102900011035511 102900011035962 102900051000944 102900051006229
 102900005116257 102900005116509 102900011000335 102900011009444
 102900011016909 102900011022764 102900011033975 102900011033982
 102900011033999 102900051000463 102900005116219 102900005116226
 102900005116233 102900005116905 102900005116943 102900005117056
 102900005117209 102900005119968 102900005123880 102900005125808
 102900011000328 102900011000861 102900011001219 102900011009772
 102900011016701 102900011022030 102900011023648 102900011027479
 102900011028407 102900011029176 102900011029275 102900011029299
 102900011029305 102900011031100 102900011031582 102900011031735
 102900011031742 102900011031759 102900011032022 102900011032145
 102900011032206 102900011032213 102900011032220 102900011032237
 102900011032244 102900011032251 102900011032343 102900011032350
 102900011032367 102900011032848 102900011034262 102900011034439
 102900011035078 102900011036242 102900051004294 102900005115250
 102900005116530 102900005116547 102900005116837 102900005116912
 102900005117353 102900005119098 102900005119104 102900005119944
 102900005125815 102900011001806 102900011001813 102900011007044
 102900011008577 102900011009246 102900011010563 102900011011058
 102900011011546 102900011011669 102900011011782 102900011012482
 102900011012871 102900011012994 102900011013274 102900011018095
 102900011021675 102900011021699 102900011023075 102900011026793
 102900011030561 102900011030608 102900011030615 102900011030622
 102900011030639 102900011030912 102900011030929 102900011031599
 102900011031841 102900011031858 102900011031926 102900011031995
 102900011032619 102900011032626 102900011032633 102900011032640
```

```
     102900011033937 102900011033944 102900011033968 102900011034330
```

```
# checking the data type of item category table
item_category.dtypes
```

```
     Item Code         int64
     Item Name        object
     Category Code     int64
     Category Name    object
     dtype: object
```

```
# It will give stats of the numberical values columns
item_category.describe().astype(int)
```

	Item Code	Category Code
count	251	251
mean	103190825064436	1011010414
std	1048400107277	291
min	102900005115168	1011010101
25%	102900011001626	1011010101
50%	102900011029275	1011010501
75%	102900011033746	1011010801
max	106973990980123	1011010801

```
from google.colab import drive
drive.mount("/content/drive")
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#" Exploring the Sales table"
```

```
sales.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)
0	2020-07-01	09:15:07.924	102900005117056	0.396	7.6	sale	No
1	2020-07-01	09:17:27.295	102900005115960	0.849	3.2	sale	No
	2020-						

```
sales.dtypes
# we have to change the wrong datatype to right data typesales.dtypes
```

```
     Date                            object
     Time                            object
     Item Code                        int64
     Quantity Sold (kilo)           float64
     Unit Selling Price (RMB/kg)    float64
     Sale or Return                  object
     Discount (Yes/No)               object
     dtype: object
```

```
sales["Date"]=pd.to_datetime(sales["Date"]) # converting the datatype to datetime
sales["Time"]=pd.to_datetime(sales["Time"]) # converting the datatype to datetime
```

```
sales["Date"]=pd.to_datetime(sales["Date"]) # converting the datatype to datetime
sales["Time"]=pd.to_datetime(sales["Time"]) # converting the datatype to datetime
```

```
sales.dtypes
```

```
Date                           datetime64[ns]
Time                           datetime64[ns]
Item Code                               int64
Quantity Sold (kilo)                  float64
Unit Selling Price (RMB/kg)           float64
Sale or Return                         object
Discount (Yes/No)                      object
dtype: object
```

```
sales.isnull().sum()
```

```
Date                           0
Time                           0
Item Code                      0
Quantity Sold (kilo)           0
Unit Selling Price (RMB/kg)    0
Sale or Return                 0
Discount (Yes/No)              0
dtype: int64
```

```
sales.duplicated().sum() # check the number of duplicated rows in sales table
```

```
0
```

```
print("Checking rows & columns, Rows={}, Columns={}".format(sales.shape[0], sales.shape[1]))
print(sales.nunique()) # Finding the number of unique elements present in sales
```

```
Checking rows & columns, Rows=878503, Columns=7
Date                              1085
Time                            848701
Item Code                          246
Quantity Sold (kilo)              2794
Unit Selling Price (RMB/kg)        264
Sale or Return                       2
Discount (Yes/No)                    2
dtype: int64
```

```
sales.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No
	2020-	2023-12-22					

```
#" Exploring the wholesale "
```

```
wholesale.head()
```

	Date	Item Code	Wholesale Price (RMB/kg)
0	2020-07-01	102900005115762	3.88
1	2020-07-01	102900005115779	6.72
2	2020-07-01	102900005115786	3.19
3	2020-07-01	102900005115793	9.24
4	2020-07-01	102900005115823	7.03

```
# checking the data type of whole sale table
wholesale.dtypes
```

```
Date                          object
Item Code                      int64
Wholesale Price (RMB/kg)     float64
dtype: object
```

```
wholesale.isnull().sum() # checking the null
```

```
Date                        0
Item Code                   0
Wholesale Price (RMB/kg)    0
dtype: int64
```

```
wholesale["Date"]=pd.to_datetime(wholesale["Date"]) # converting the date column from
```

```
wholesale.dtypes # after converting the Date column from object data type into datetime
```

```
Date                        datetime64[ns]
Item Code                            int64
Wholesale Price (RMB/kg)           float64
dtype: object
```

```
print("Checking rows & columns, Rows={}, Columns={}".format(wholesale.shape[0], wholesale.shape[1]))
print(wholesale.nunique()) # Finding the number of unique elements present in item_category
```

```
Checking rows & columns, Rows=55982, Columns=3
Date                        1091
Item Code                    251
Wholesale Price (RMB/kg)    2380
dtype: int64
```

```
wholesale.duplicated().sum()
```

```
0
```

```
# finding the unique elements from all columns
for i in wholesale.columns:
    print(i)
    print(wholesale[i].unique())
    print("*******************************************************************************************************")
```

```
Date
['2020-07-01T00:00:00.000000000' '2020-07-02T00:00:00.000000000'
 '2020-07-03T00:00:00.000000000' ... '2023-06-28T00:00:00.000000000'
 '2023-06-29T00:00:00.000000000' '2023-06-30T00:00:00.000000000']
*******************************************************************************************************
Item Code
[102900005115762 102900005115779 102900005115786 102900005115793
 102900005115823 102900005115908 102900005115946 102900005115960
 102900005115984 102900005116226 102900005116233 102900005116257
 102900005116509 102900005116530 102900005116547 102900005116714
 102900005116790 102900005116912 102900005116943 102900005117056
 102900005117209 102900005118817 102900005118824 102900005118831
 102900005119944 102900005119975 102900005123880 102900005125808
 102900005125815 102900011001219 102900011006948 102900011008522
 102900011009970 102900051000944 102900051004294 102900051010455
 102900011000328 102900011006689 102900011001813 102900011009444
 102900005116837 102900005115816 102900005116899 102900005115861
 106956146480203 106956146480197 102900011011546 102900011001806
 102900011001561 102900005116219 102900011000175 102900011007969
 102900011009246 102900011012994 102900011013274 102900005115885
 102900011001691 102900011008164 102900011010891 102900051000463
 102900051009336 102900005119968 102900011016909 102900005116905
 102900011016701 102900011007464 102900005115878 102900005115250
 102900005119098 102900011012871 102900011010563 102900011007044
 106931885000035 102900005118572 102900005115854 102900011018095
 102900005122654 102900005115168 102900011011782 102900011008676
 102900005115199 102900011018132 102900011008577 102900005115977
 102900005115748 102900011011669 102900011009277 102900011000571
 106949711300068 106930274220092 102900011021699 102900005119104
 102900011021842 106958851400125 102900011000632 102900011022030
 102900011022764 106973223300667 102900011008133 102900011022849
 102900011002414 102900051010790 102900051009220 102900011022924
 102900005116639 102900011008515 102900011015391 102900011007471
```

```
102900011023075 102900005116776 102900011006955 102900011024010
102900005117353 102900011026502 102900005116806 102900011008485
102900011000861 102900011026793 102900011027479 102900011000335
102900011008492 102900011027615 102900011027462 102900051000890
102900011007495 102900005128748 102900011023464 102900011028407
106949711300167 102900011029176 106971533450003 102900011029688
102900011029275 102900011030097 102900011030103 102900011030110
102900011030134 102900011030141 102900011030158 102900011030042
102900011030059 102900051006229 102900011031100 102900011021644
102900011030561 102900011030615 102900011030622 102900011030608
102900011029305 102900011030639 102900011031599 102900011031582
102900011031759 106957634300010 106957634300058 106971533455008
102900011031742 106949711300259 102900011031735 102900011031926
102900011032022 102900011032282 102900011031995 102900011009772
102900011032237 102900011030912 102900011031841 102900011032206
102900011032213 102900011032244 102900011032114 102900011032220
102900011032251 102900011032350 102900011032367 102900011031216
102900011032176 102900011015384 102900011032343 102900011032589
102900011032626 102900011032633 102900011032640 102900011032787
102900011032619 102900011032732 102900011032848 102900011021675
102900011033081 102900011033234 102900011033241 102900011032145
102900011033562 102900011033586 102900011033531 102900011033173
102900011033906 102900011033968 102900011034200 102900011034231
102900011033920 102900011033937 102900011033944 102900011034217
102900011034224 102900011033913 102900005116042 106931885000356
```

```
wholesale.describe().astype(int)
```

	Item Code	Wholesale Price (RMB/kg)
count	55982	55982
mean	103044945365014	5
std	752792703273	5
min	102900005115168	0
25%	102900005116547	2
50%	102900011000328	4
75%	102900011030141	7
max	106973990980123	141

```
                                #"Exploring the Loss rate"
loss_rate.head()
```

	Item Code	Item Name	Loss Rate (%)
0	102900005115168	Niushou Shengcai	4.39
1	102900005115199	Sichuan Red Cedar	10.46
2	102900005115250	Xixia Black Mushroom (1)	10.80
3	102900005115625	Local Xiaomao Cabbage	0.18
4	102900005115748	White Caitai	8.78

```
loss_rate.dtypes
```

```
Item Code          int64
Item Name         object
Loss Rate (%)    float64
dtype: object
```

```
loss_rate.isnull().sum()
```

```
Item Code        0
Item Name        0
Loss Rate (%)    0
dtype: int64
```

```
loss_rate.describe().astype(int)
```

	Item Code	Loss Rate (%)
count	251	251
mean	103190825064436	9
std	1048400107277	5
min	102900005115168	0
25%	102900011001626	8
50%	102900011029275	9

```
                                        #Merging / combing the data
sales.head(2)
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)
0	2020-	2023-12-22	102900005117056	0.396	7.6	sale	No

```
wholesale.head(2)
```

	Date	Item Code	Wholesale Price (RMB/kg)
0	2020-07-01	102900005115762	3.88
1	2020-07-01	102900005115779	6.72

```
item_category.head(2)
```

	Item Code	Item Name	Category Code	Category Name
0	102900005115168	Niushou Shengcai	1011010101	Flower/Leaf Vegetables
1	102900005115199	Sichuan Red Cedar	1011010101	Flower/Leaf Vegetables

```
loss_rate.head(2)
```

	Item Code	Item Name	Loss Rate (%)
0	102900005115168	Niushou Shengcai	4.39
1	102900005115199	Sichuan Red Cedar	10.46

```
sales_wholesale_combine_data = pd.merge(sales,wholesale,how="left",on=["Item Code","Date"])
sales_wholesale_combine_data.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No	2.10
2	2020-07-01	2023-12-22 09:17:33.905	102900005117056	0.409	7.6	sale	No	4.32
3	2020-07-01	2023-12-22 09:19:45.450	102900005115823	0.421	10.0	sale	No	7.03
4	2020-	2023-12-22	102900005115908	0.539	8.0	sale	No	4.60

```
sales_wholesale_combine_data.isnull().sum()
```

```
Date                           0
Time                           0
Item Code                      0
```

```
Quantity Sold (kilo)          0
Unit Selling Price (RMB/kg)   0
Sale or Return                0
Discount (Yes/No)             0
Wholesale Price (RMB/kg)      0
dtype: int64
```

```
sales_wholesale_category = pd.merge(sales_wholesale_combine_data,item_category,how="left",on="Item Code")
sales_wholesale_category.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No	2.10
2	2020-07-01	2023-12-22 09:17:33.905	102900005117056	0.409	7.6	sale	No	4.32
3	2020-07-01	2023-12-22 09:19:45.450	102900005115823	0.421	10.0	sale	No	7.03
4	2020-	2023-12-22	102900005115908	0.539	8.0	sale	No	4.60

```
final_data = pd.merge(sales_wholesale_category,loss_rate,how="left",on=["Item Code","Item Name"])
final_data.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No	2.10
2	2020-07-01	2023-12-22 09:17:33.905	102900005117056	0.409	7.6	sale	No	4.32
3	2020-07-01	2023-12-22 09:19:45.450	102900005115823	0.421	10.0	sale	No	7.03
4	2020-	2023-12-22	102900005115908	0.539	8.0	sale	No	4.60

```
final_data.columns
```

```
Index(['Date', 'Time', 'Item Code', 'Quantity Sold (kilo)',
       'Unit Selling Price (RMB/kg)', 'Sale or Return', 'Discount (Yes/No)',
       'Wholesale Price (RMB/kg)', 'Item Name', 'Category Code',
       'Category Name', 'Loss Rate (%)'],
      dtype='object')
```

```
final_data["total_sales"]=final_data["Quantity Sold (kilo)"]*final_data["Unit Selling Price (RMB/kg)"]
```

```
final_data["total_sales"]=final_data["Quantity Sold (kilo)"]*final_data["Unit Selling Price (RMB/kg)"]
```

```
final_data.head()
```

	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32

```
final_data.isnull().sum()
```

```
Date                           0
Time                           0
Item Code                      0
Quantity Sold (kilo)           0
Unit Selling Price (RMB/kg)    0
Sale or Return                 0
Discount (Yes/No)              0
Wholesale Price (RMB/kg)       0
Item Name                      0
Category Code                  0
Category Name                  0
Loss Rate (%)                  0
total_sales                    0
dtype: int64
```

```
final_data["Item Name"].nunique()
final_data["Category Name"].nunique()
```

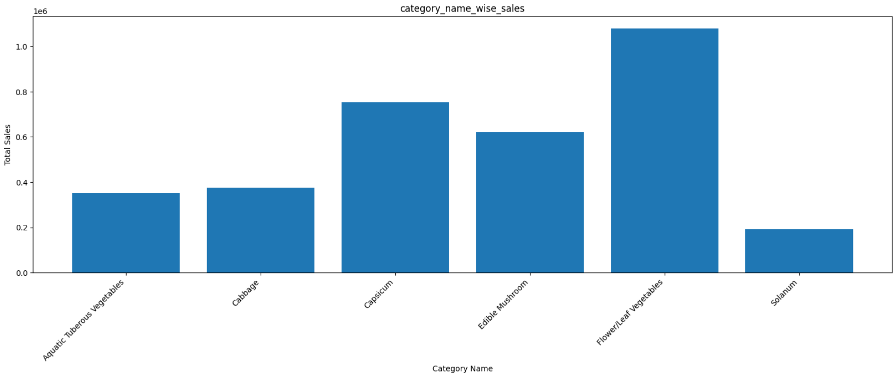
```
6
```

```
                                                         #Charts
# make a group of category name and check the total sales done by category
category_name_wise_sales = final_data.groupby(["Category Name"])["total_sales"].sum().reset_index()
#category_name_wise_sales["total_sales"]=category_name_wise_sales["total_sales"].astype(int)
```

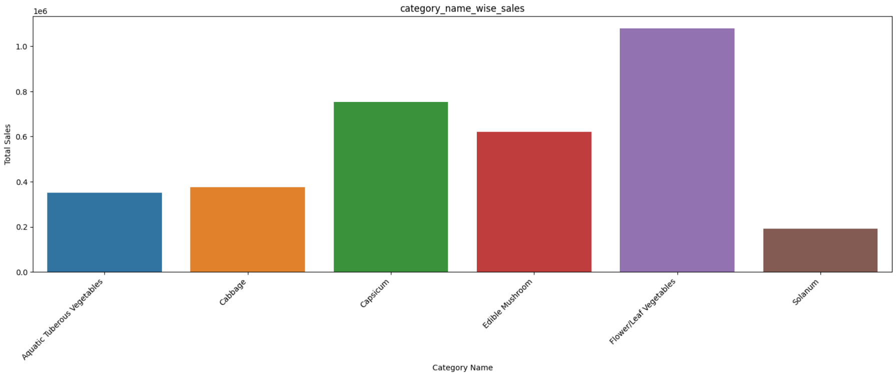
```
category_name_wise_sales
```

	Category Name	total_sales
0	Aquatic Tuberous Vegetables	3.500899e+05
1	Cabbage	3.757517e+05
2	Capsicum	7.541330e+05
3	Edible Mushroom	6.195978e+05
4	Flower/Leaf Vegetables	1.079070e+06
5	Solanum	1.911243e+05

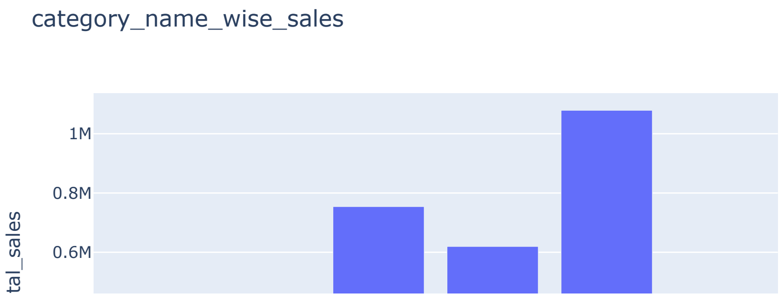
```
#" Creating the charts with the help of metplotlib, seaborn, plotly"
#Matplotlib bar chart
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 6))
plt.bar(category_name_wise_sales['Category Name'], category_name_wise_sales['total_sales'])
plt.xlabel('Category Name')
plt.ylabel('Total Sales')
plt.title('category_name_wise_sales')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
#Seaborn bar chart
import seaborn as sns
plt.figure(figsize=(20, 6))
sns.barplot(x='Category Name', y='total_sales', data=category_name_wise_sales)
plt.xlabel('Category Name')
plt.ylabel('Total Sales')
plt.title('category_name_wise_sales')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
#Plotly bar chart
import plotly.express as px
fig = px.bar(category_name_wise_sales, x='Category Name', y='total_sales', title='category_name_wise_sales')
fig.update_layout(xaxis=dict(tickangle=45))
fig.show()
```

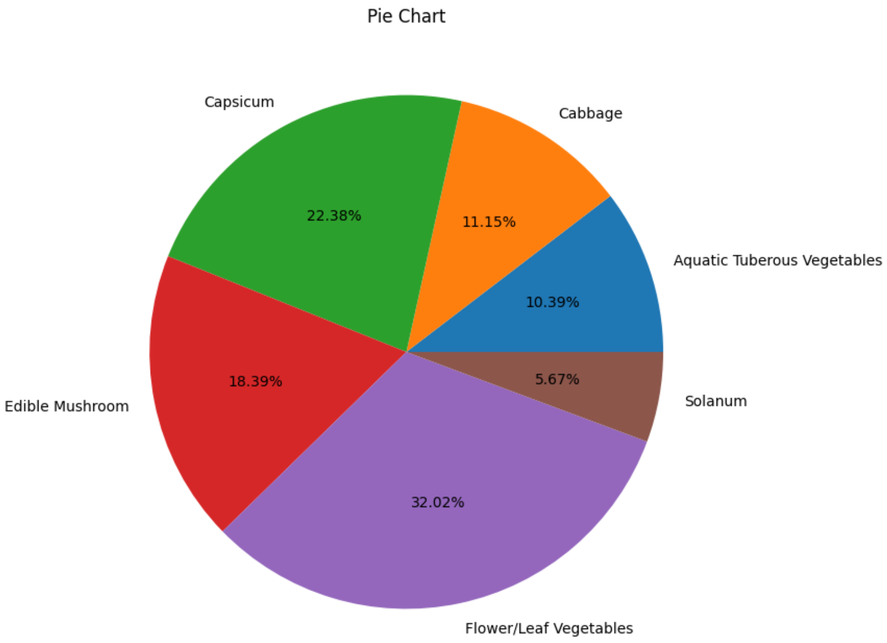


```
#Piechart
```

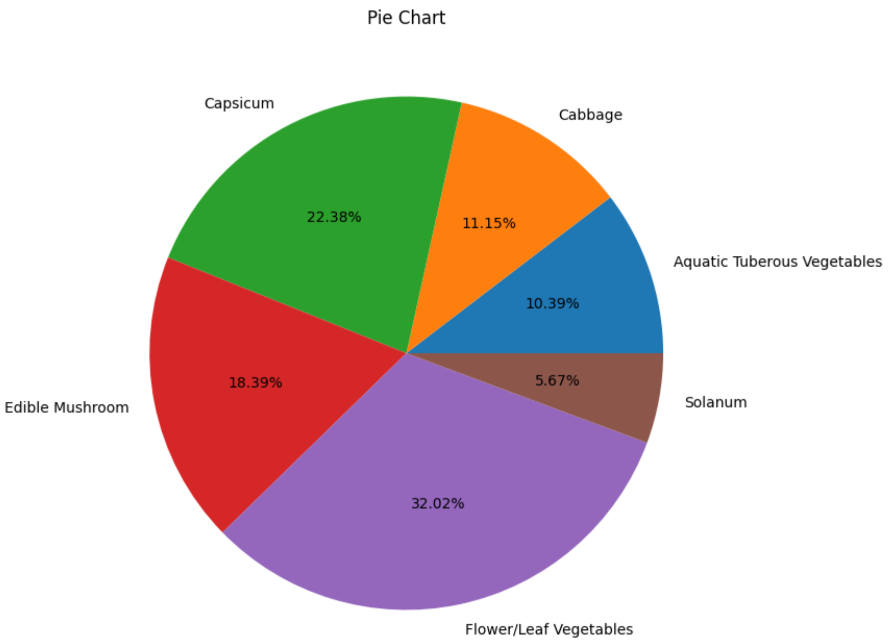
```
category_name_wise_sales
```

	Category Name	total_sales
0	Aquatic Tuberous Vegetables	3.500899e+05
1	Cabbage	3.757517e+05
2	Capsicum	7.541330e+05
3	Edible Mushroom	6.195978e+05
4	Flower/Leaf Vegetables	1.079070e+06
5	Solanum	1.911243e+05

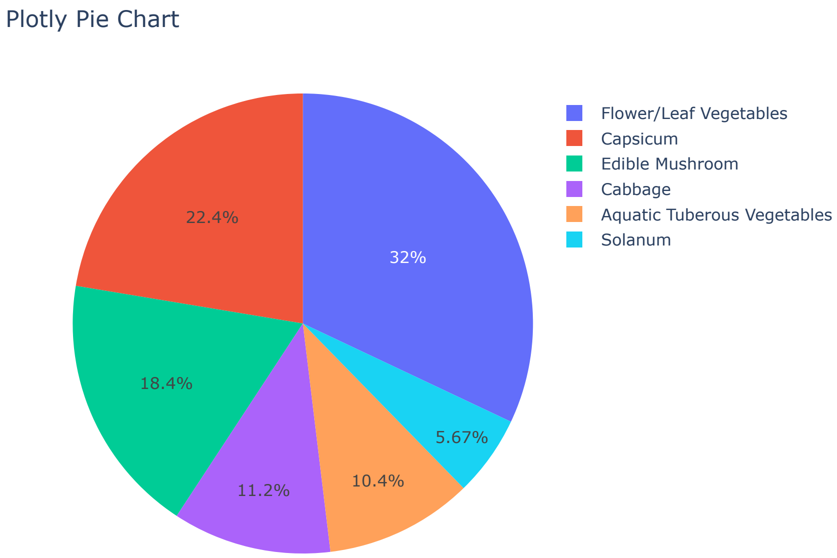
```
plt.figure(figsize=(8, 8))
plt.pie(category_name_wise_sales['total_sales'], labels=category_name_wise_sales['Category Name'], autopct='%1.2f%%')
plt.title('Pie Chart')
plt.show()
```



```
#using seaborn (same as matplot lib but we can edit it more by ading visualizations or colors)
plt.figure(figsize=(8, 8))
plt.pie(category_name_wise_sales['total_sales'], labels=category_name_wise_sales['Category Name'], autopct='%1.2f%%')
plt.title('Pie Chart')
plt.show()
```



```
#using plotly
fig = px.pie(category_name_wise_sales, values='total_sales', names='Category Name', title='Plotly Pie Chart')
fig.show()
```



```
# Extract Year and Month
final_data['Year'] = final_data['Date'].dt.year
final_data['Month'] = final_data['Date'].dt.month_name()
```

```
final_data
```

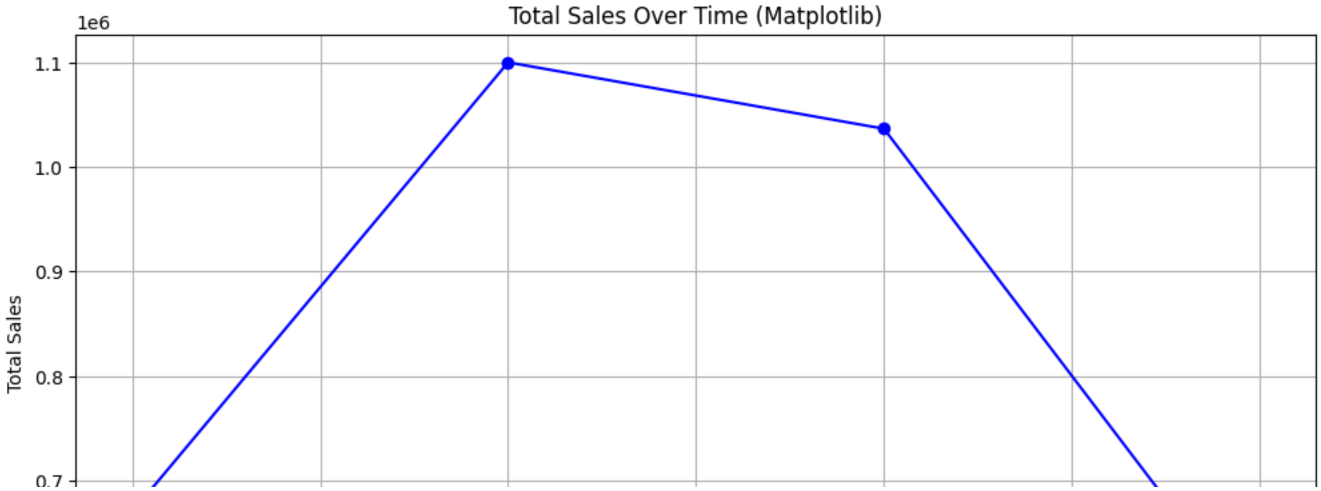
	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)	Item Name	Category Code	Category Nam
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32	Paopaojiao (Jingpin)	1011010504	Capsicur
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No	2.10	Chinese Cabbage	1011010101	Flower/Leaf Vegetable
2	2020-07-01	2023-12-22 09:17:33.905	102900005117056	0.409	7.6	sale	No	4.32	Paopaojiao (Jingpin)	1011010504	Capsicur
3	2020-07-01	2023-12-22 09:19:45.450	102900005115823	0.421	10.0	sale	No	7.03	Shanghaiqing	1011010101	Flower/Leaf Vegetable
4	2020-07-01	2023-12-22 09:20:23.686	102900005115908	0.539	8.0	sale	No	4.60	Caixin	1011010101	Flower/Leaf Vegetable
...	...	...	...	...	...	...	...	...	...	...	.
878498	2023-06-30	2023-12-22 21:35:13.264	102900005115250	0.284	24.0	sale	No	15.60	Xixia Black Mushroom (1)	1011010801	Edible Mushroor
878499	2023-06-30	2023-12-22 21:35:14.358	102900011022764	0.669	12.0	sale	No	7.00	Changxianqie	1011010501	Solanur
878500	2023-06-30	2023-12-22 21:35:20.264	102900005115250	0.125	24.0	sale	No	15.60	Xixia Black Mushroom (1)	1011010801	Edible Mushroor
878501	2023-06-30	2023-12-22 21:35:21.509	102900011016701	0.252	5.2	sale	No	3.63	Wuhu Green Pepper (1)	1011010504	Capsicur
878502	2023-06-30	2023-12-22 21:40:48.248	102900011022764	0.803	12.0	sale	No	7.00	Changxianqie	1011010501	Solanur

878503 rows × 15 columns

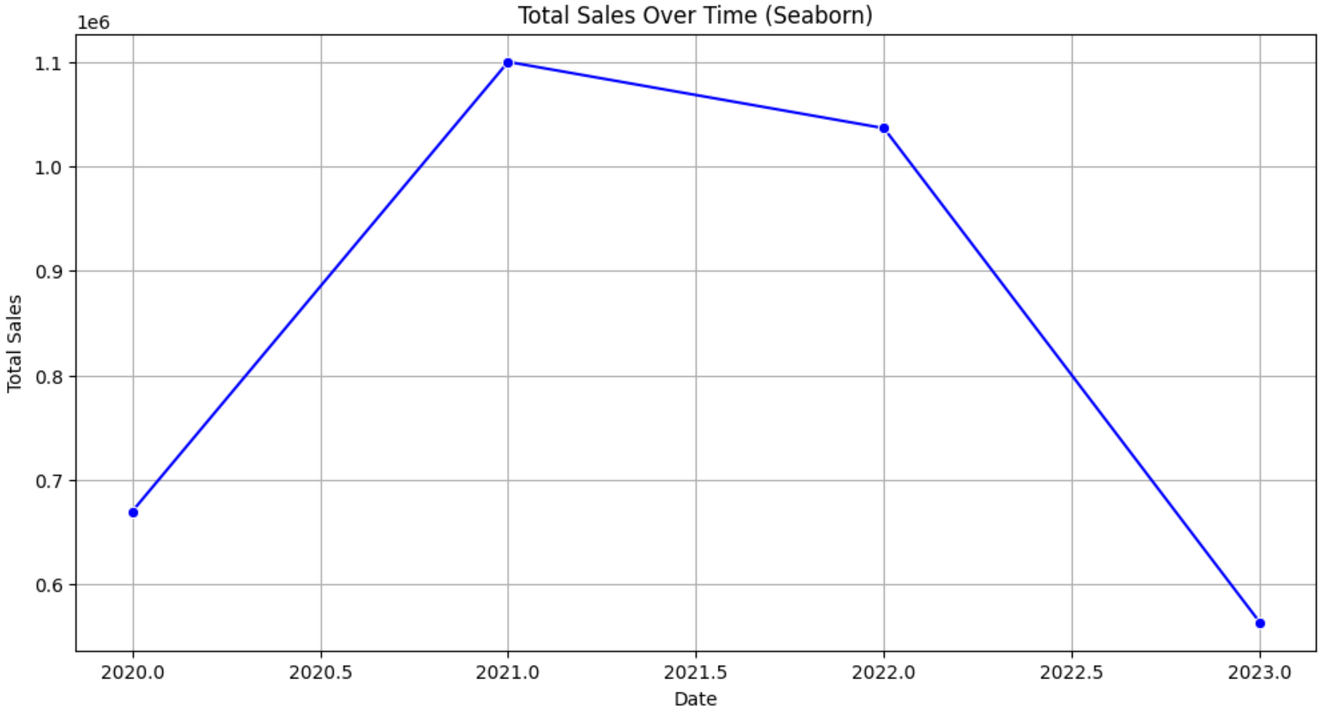
```
sales_trend = final_data.groupby(final_data['Date'].dt.year)['total_sales'].sum().astype(int).reset_index()
sales_trend
```

	Date	total_sales
0	2020	669529
1	2021	1100362
2	2022	1036772
3	2023	563102

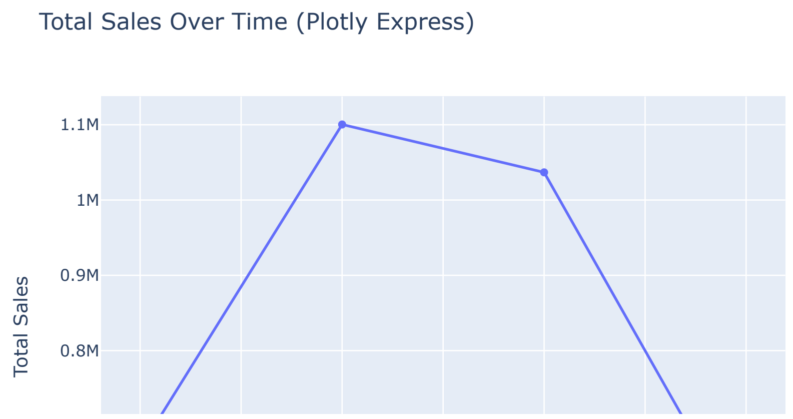
```
#Matplotlib
plt.figure(figsize=(12, 6))
plt.plot(sales_trend['Date'], sales_trend['total_sales'], marker='o', linestyle='-', color='b')
plt.title('Total Sales Over Time (Matplotlib)')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```



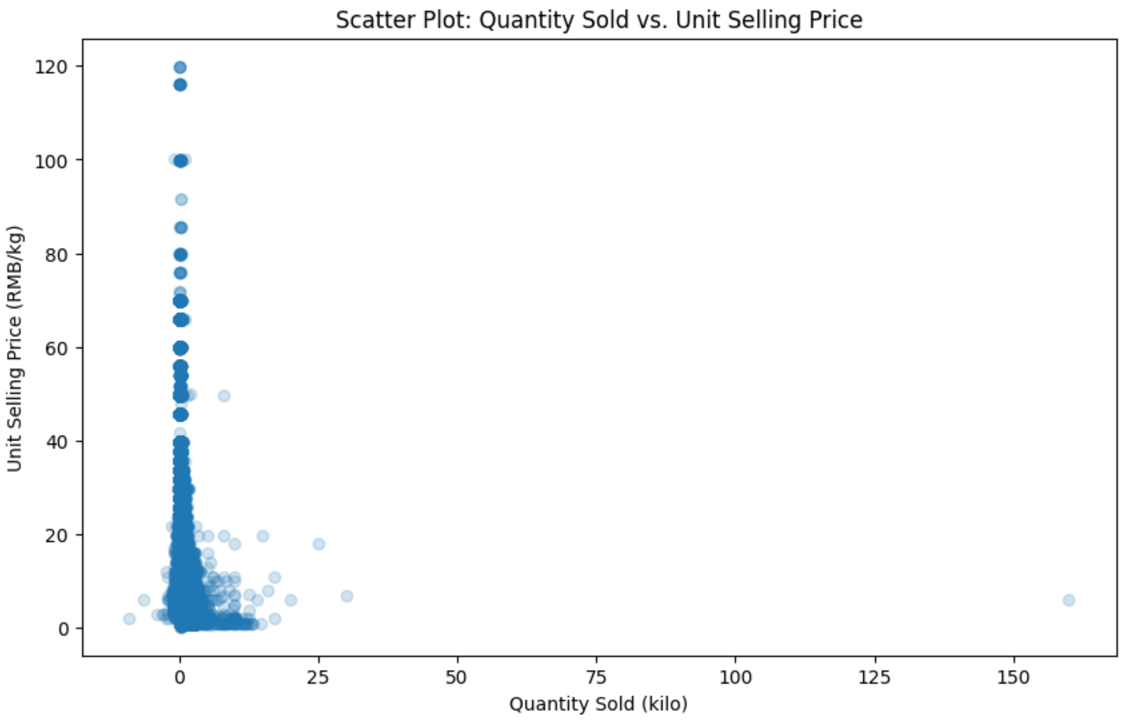
```
#Seaborn
plt.figure(figsize=(12, 6))
sns.lineplot(data=sales_trend, x='Date', y='total_sales', marker='o', color='b')
plt.title('Total Sales Over Time (Seaborn)')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```



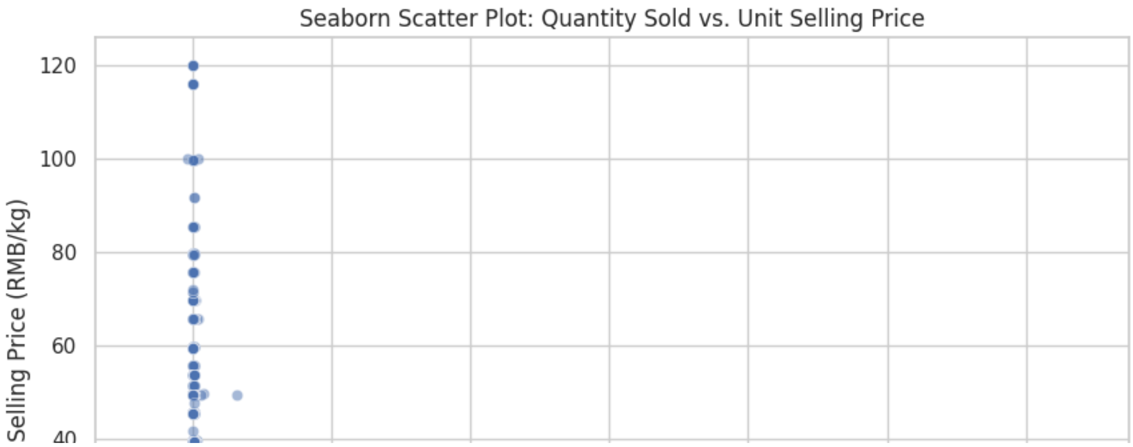
```
#plotly
fig_line = px.line(sales_trend, x='Date', y='total_sales', title='Total Sales Over Time (Plotly Express)', markers=True, line_shape='linear')
fig_line.update_layout(xaxis_title='Date', yaxis_title='Total Sales')
fig_line.show()
```



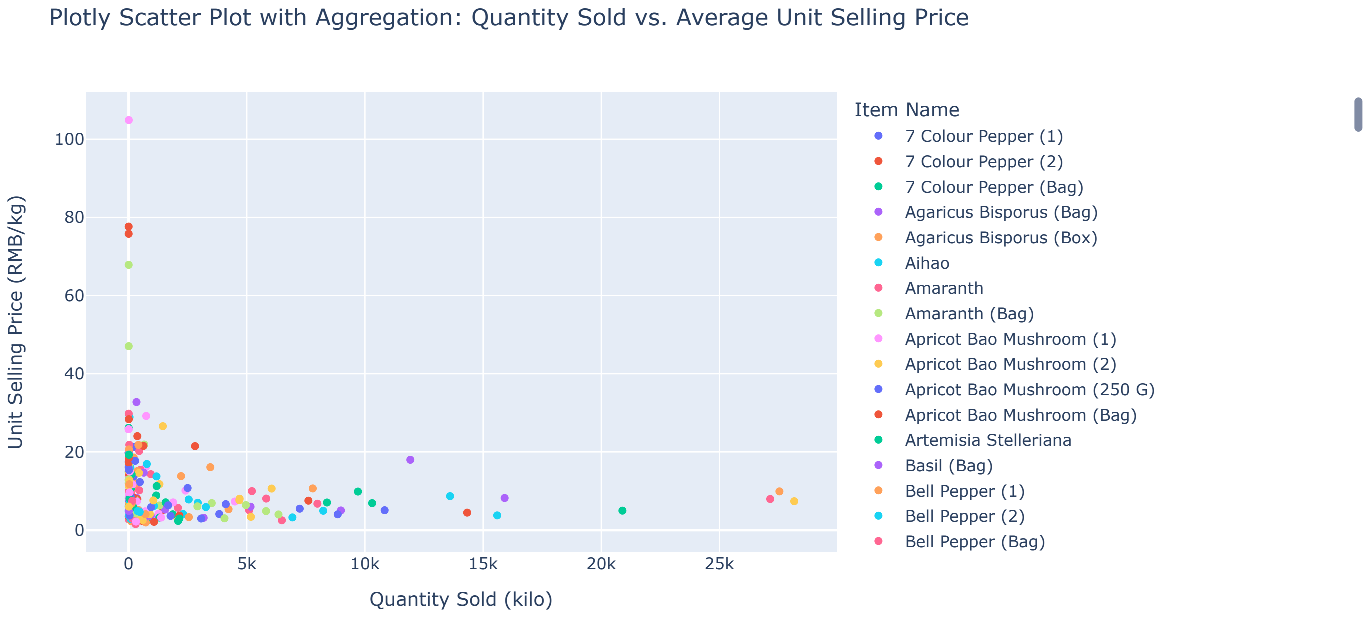
```
#Scatter Plot
#Quantity VS Unit Selling Price
plt.figure(figsize=(10, 6))
plt.scatter(final_data['Quantity Sold (kilo)'], final_data['Unit Selling Price (RMB/kg)'], alpha=0.2)
plt.title('Scatter Plot: Quantity Sold vs. Unit Selling Price')
plt.xlabel('Quantity Sold (kilo)')
plt.ylabel('Unit Selling Price (RMB/kg)')
plt.show()
```



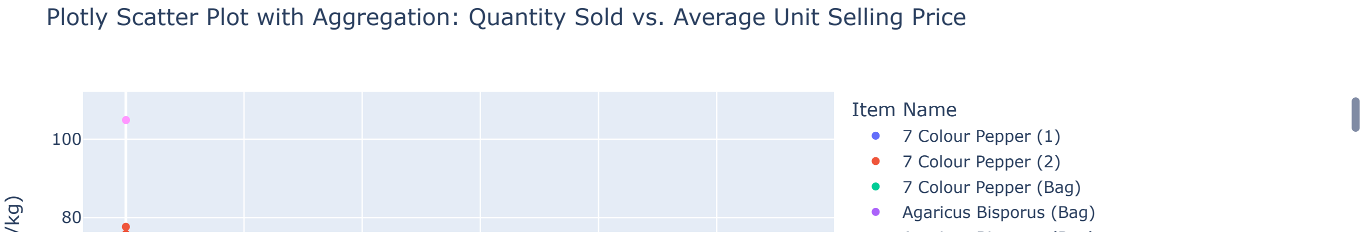
```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Quantity Sold (kilo)', y='Unit Selling Price (RMB/kg)', data=final_data, alpha=0.5)
plt.title('Seaborn Scatter Plot: Quantity Sold vs. Unit Selling Price')
plt.show()
```



```
agg_data = final_data.groupby('Item Name').agg({'Quantity Sold (kilo)': 'sum', 'Unit Selling Price (RMB/kg)': 'mean'}).reset_index()
# Scatter plot using Plotly with aggregation
fig = px.scatter(agg_data, x='Quantity Sold (kilo)', y='Unit Selling Price (RMB/kg)', color='Item Name',
title='Plotly Scatter Plot with Aggregation: Quantity Sold vs. Average Unit Selling Price')
fig.show()
```



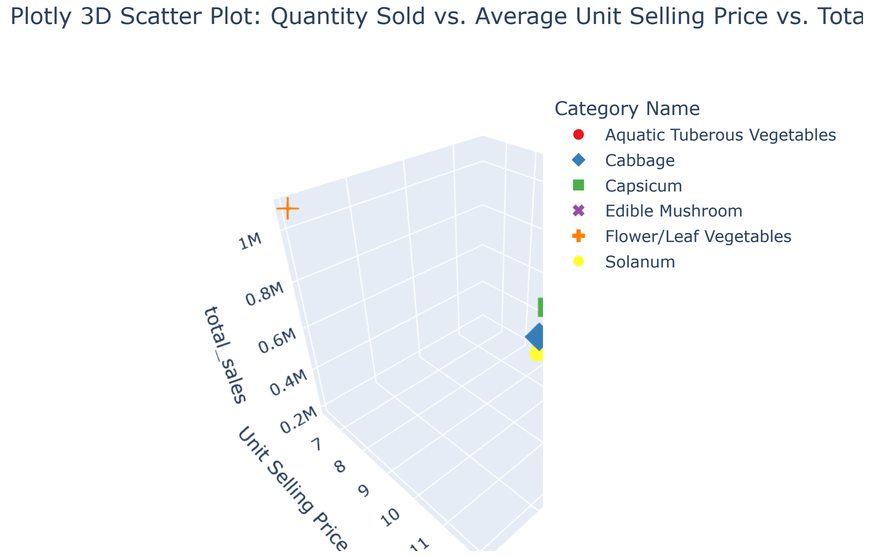
```
agg_data = final_data.groupby('Item Name').agg({'Quantity Sold (kilo)': 'sum', 'Unit Selling Price (RMB/kg)': 'mean'}).reset_index()
# Scatter plot using Plotly with aggregation
fig = px.scatter(agg_data, x='Quantity Sold (kilo)', y='Unit Selling Price (RMB/kg)', color='Item Name',
title='Plotly Scatter Plot with Aggregation: Quantity Sold vs. Average Unit Selling Price')
fig.show()
```



```
final_data.head(2)
```

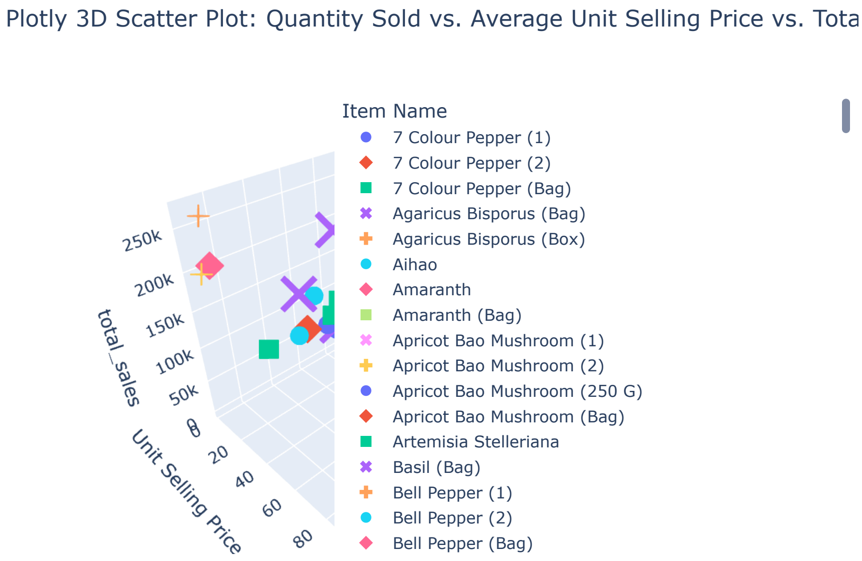
	Date	Time	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Sale or Return	Discount (Yes/No)	Wholesale Price (RMB/kg)	Item Name	Category Code	Category Name	Loss Rate (%)
0	2020-07-01	2023-12-22 09:15:07.924	102900005117056	0.396	7.6	sale	No	4.32	Paopaojiao (Jingpin)	1011010504	Capsicum	7.08
1	2020-07-01	2023-12-22 09:17:27.295	102900005115960	0.849	3.2	sale	No	2.10	Chinese Cabbage	1011010101	Flower/Leaf Vegetables	22.27

```
#Scatter 3D Plot Plotly
#Based on Category Name
agg_data = final_data.groupby('Category Name').agg({
'Quantity Sold (kilo)': 'sum',
'Unit Selling Price (RMB/kg)': 'mean',
'total_sales': 'sum' # Assuming 'total_sales' is a column you want to sum
}).reset_index()
fig = px.scatter_3d(agg_data, x='Quantity Sold (kilo)', y='Unit Selling Price (RMB/kg)', z='total_sales',
title='Plotly 3D Scatter Plot: Quantity Sold vs. Average Unit Selling Price vs. Total Sales',
color='Category Name', symbol='Category Name',
color_discrete_sequence=px.colors.qualitative.Set1)
fig.show()
```



```
#Based on Item Name
agg_data = final_data.groupby('Item Name').agg({
'Quantity Sold (kilo)': 'sum',
'Unit Selling Price (RMB/kg)': 'mean',
'total_sales': 'sum' # Assuming 'total_sales' is a column you want to sum
}).reset_index()
```

```
fig = px.scatter_3d(agg_data, x='Quantity Sold (kilo)', y='Unit Selling Price (RMB/kg)', z='total_sales',
title='Plotly 3D Scatter Plot: Quantity Sold vs. Average Unit Selling Price vs. Total Sales',
color='Item Name', symbol='Item Name')
fig.show()
```

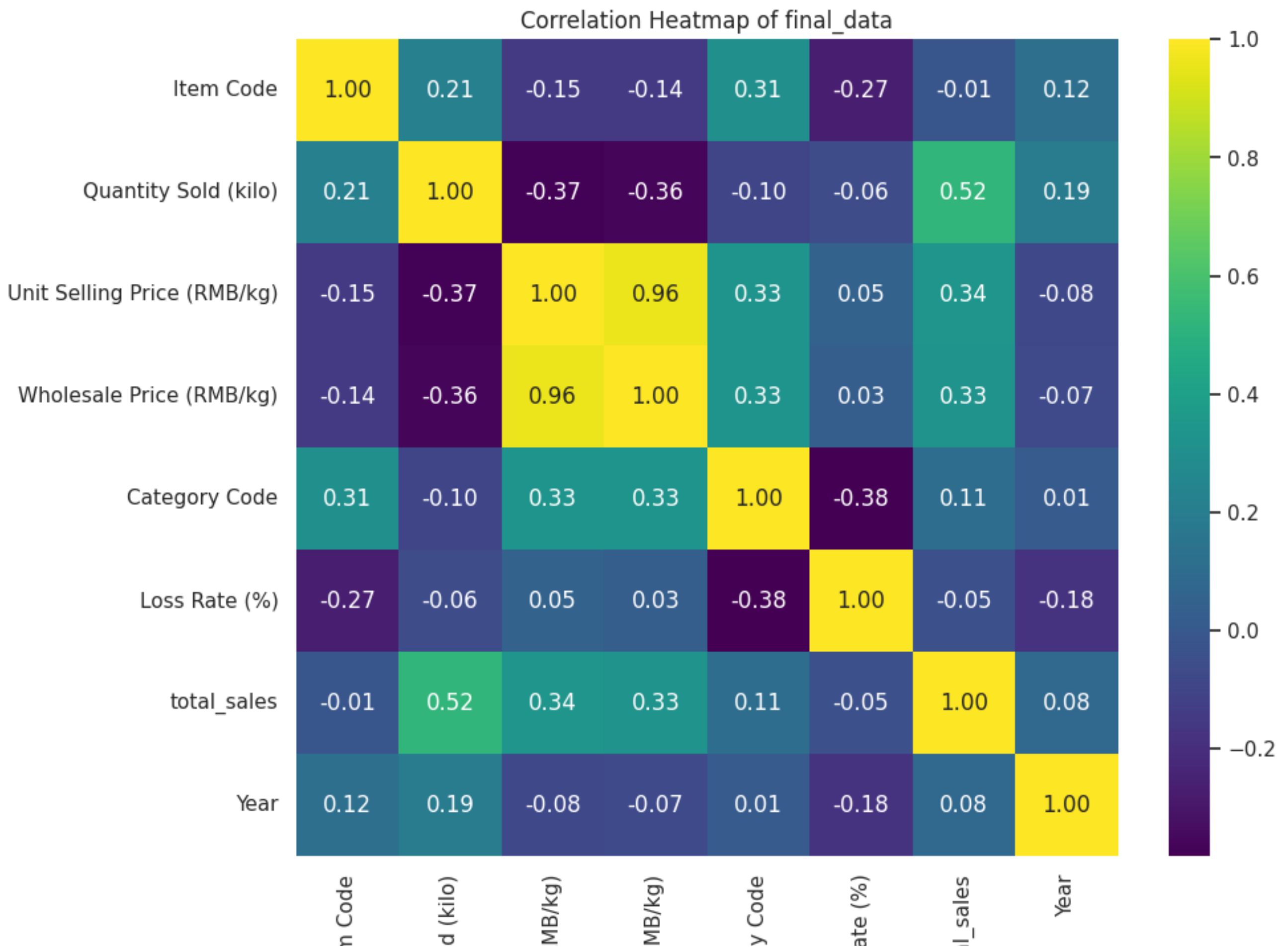


```
correlation_matrix = final_data.corr()
correlation_matrix
```

```
<ipython-input-114-e8d538bc49c9>:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid colu
```

	Item Code	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Wholesale Price (RMB/kg)	Category Code	Loss Rate (%)	total_sales	Year
Item Code	1.000000	0.212891	-0.152789	-0.144599	0.307052	-0.266643	-0.012147	0.119615
Quantity Sold (kilo)	0.212891	1.000000	-0.374163	-0.363975	-0.097333	-0.064141	0.524797	0.188677
Unit Selling Price (RMB/kg)	-0.152789	-0.374163	1.000000	0.958943	0.331646	0.046217	0.339107	-0.079342
Wholesale Price (RMB/kg)	-0.144599	-0.363975	0.958943	1.000000	0.333595	0.025472	0.325845	-0.070093
Category Code	0.307052	-0.097333	0.331646	0.333595	1.000000	-0.381446	0.105203	0.009351
Loss Rate (%)	-0.266643	-0.064141	0.046217	0.025472	-0.381446	1.000000	-0.045221	-0.176838
total_sales	-0.012147	0.524797	0.339107	0.325845	0.105203	-0.045221	1.000000	0.081466
Year	0.119615	0.188677	-0.079342	-0.070093	0.009351	-0.176838	0.081466	1.000000

```
#Create a heatmap using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt='.2f')
plt.title('Correlation Heatmap of final_data')
plt.show()
```



	n Code	d (kilo)	MB/kg)	MB/kg)	y Code	ate (%)	l_sales	Year
Item Code	1.00	0.21	-0.15	-0.14	0.31	-0.27	-0.01	0.12
Quantity Sold (kilo)	0.21	1.00	-0.37	-0.36	-0.10	-0.06	0.52	0.19
Unit Selling Price (RMB/kg)	-0.15	-0.37	1.00	0.96	0.33	0.05	0.34	-0.08
Wholesale Price (RMB/kg)	-0.14	-0.36	0.96	1.00	0.33	0.03	0.33	-0.07
Category Code	0.31	-0.10	0.33	0.33	1.00	-0.38	0.11	0.01
Loss Rate (%)	-0.27	-0.06	0.05	0.03	-0.38	1.00	-0.05	-0.18
total_sales	-0.01	0.52	0.34	0.33	0.11	-0.05	1.00	0.08
Year	0.12	0.19	-0.08	-0.07	0.01	-0.18	0.08	1.00

```
final_data[["Quantity Sold (kilo)","Unit Selling Price (RMB/kg)","Wholesale Price (RMB/kg)"]].corr()
```

	Quantity Sold (kilo)	Unit Selling Price (RMB/kg)	Wholesale Price (RMB/kg)
Quantity Sold (kilo)	1.000000	-0.374163	-0.363975
Unit Selling Price (RMB/kg)	-0.374163	1.000000	0.958943
Wholesale Price (RMB/kg)	-0.363975	0.958943	1.000000

```
#placement_Data
```

```
import numpy as np
import pandas as pd
```

```
pwd!
```

```
'/content'
```

```
!mkdir placement_data
```

```
cd placement_data
```

```
/content/placement_data
```

```
from google.colab import files
upload=files.upload()
```

Choose Files Placement_Data.csv

- **Placement_Data.csv**(text/csv) - 18183 bytes, last modified: 12/22/2023 - 100% done

```
Saving Placement_Data.csv to Placement_Data.csv
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
placement_data=pd.read_csv('/content/placement_data/Placement_Data.csv')
```

```
"""from google.colab import drive
drive.mount('/content/drive')"""
```

```
'from google.colab import drive\ndrive.mount('/content/drive')'
```

```
#Check the first four rows of the dataframe
placement_data.head()
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN

```
#Get all information names
placement_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sl_no           215 non-null    int64
 1   gender          215 non-null    object
 2   ssc_p           215 non-null    float64
 3   ssc_b           215 non-null    object
 4   hsc_p           215 non-null    float64
 5   hsc_b           215 non-null    object
 6   hsc_s           215 non-null    object
 7   degree_p        215 non-null    float64
 8   degree_t        215 non-null    object
 9   workex          215 non-null    object
 10  etest_p         215 non-null    float64
 11  specialisation  215 non-null    object
 12  mba_p           215 non-null    float64
 13  status          215 non-null    object
 14  salary          148 non-null    float64
dtypes: float64(6), int64(1), object(8)
memory usage: 25.3+ KB
```

```
#Find the number of records and columns
print("Checking rows & columns, Rows={}, Columns={}".format(placement_data.shape[0], placement_data.shape[1]))
print(placement_data.nunique())
```

```
Checking rows & columns, Rows=215, Columns=15
sl_no             215
gender              2
ssc_p             103
ssc_b               2
hsc_p              97
hsc_b               2
hsc_s               3
degree_p           89
degree_t            3
workex              2
etest_p           100
specialisation      2
mba_p             205
status              2
salary             45
dtype: int64
```

```
#Use the .info() method to find the number of Non Null entries and Data Type of each feature
placement_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
 ---  ------          --------------  -----
  0   sl_no           215 non-null    int64
  1   gender          215 non-null    object
  2   ssc_p           215 non-null    float64
  3   ssc_b           215 non-null    object
  4   hsc_p           215 non-null    float64
  5   hsc_b           215 non-null    object
  6   hsc_s           215 non-null    object
  7   degree_p        215 non-null    float64
  8   degree_t        215 non-null    object
  9   workex          215 non-null    object
  10  etest_p         215 non-null    float64
  11  specialisation  215 non-null    object
  12  mba_p           215 non-null    float64
  13  status          215 non-null    object
  14  salary          148 non-null    float64
dtypes: float64(6), int64(1), object(8)
memory usage: 25.3+ KB
```

```
#What is the average Secondary Education percentage - 10th Grade
placement_data["ssc_p"].mean()
```

```
67.30339534883721
```

```
#What is the max Secondary Education percentage - 10th Grade
placement_data["ssc_p"].max()
```

```
89.4
```

```
#How many toppers where there in 10th Grade?
import pandas as pd
# Filter the data for students in the 10th grade
grade_10_students = placement_data[placement_data['ssc_p'] == 10]

# Find the maximum grade in the 10th grade
max_grade_10th = grade_10_students['ssc_p'].max()

# Filter the 10th-grade students who achieved the maximum grade (toppers)
toppers_10th_grade = grade_10_students[grade_10_students['ssc_p'] == max_grade_10th]

# Get the count of toppers in the 10th grade
num_toppers_10th_grade = toppers_10th_grade.shape[0]

print(f"The number of toppers in 10th grade based on grades is: {num_toppers_10th_grade}")
```

```
The number of toppers in 10th grade based on grades is: 0
```

```
import pandas as pd
import numpy as np
secondary_education = placement_data[placement_data['ssc_p'] == '10th']

if not secondary_education.empty:
    # Find the student with the highest Secondary Education percentage
    max_percentage = secondary_education['status'].max()

    # Retrieve the details of the student(s) with the highest percentage
    top_students = secondary_education[secondary_education['status'] == max_percentage]

    if not top_students.empty:
        # Comparing two columns (Grade and Percentage) to infer placement status
        if top_students['ssc_p'].iloc[0] == '10th':
            print("The student with the highest Secondary Education percentage is placed.")
        else:
            print("The student with the highest Secondary Education percentage is not placed.")
    else:
        print("No data found for Secondary Education (10th grade) in the dataset.")
else:
    print("No data found for Secondary Education (10th grade) in the dataset.")
```

```
No data found for Secondary Education (10th grade) in the dataset.
```

```
#How many students are placed or unplaced?
placement_data.value_counts("status")
```

```
status
Placed        148
```

```
    Not Placed     67
    dtype: int64
```

```
#How many unique degrees are there in the dataset?
placement_data["degree_t"].unique()
```

```
    array(['Sci&Tech', 'Comm&Mgmt', 'Others'], dtype=object)
```

```
#Is there a correlation between 10th and 12th percentage
placement_data["ssc_p"].corr(placement_data["hsc_p"], method="spearman")
```

```
    0.49002773975076686
```

```
Find the correlation matrix?
correlation_matrix = placement_data.corr()

print(correlation_matrix)
```

```
                 sl_no     ssc_p     hsc_p  degree_p   etest_p     mba_p    salary
    sl_no     1.000000 -0.078155 -0.085711 -0.088281  0.063636  0.022327  0.063764
    ssc_p    -0.078155  1.000000  0.511472  0.538404  0.261993  0.388478  0.035330
    hsc_p    -0.085711  0.511472  1.000000  0.434206  0.245113  0.354823  0.076819
    degree_p -0.088281  0.538404  0.434206  1.000000  0.224470  0.402364 -0.019272
    etest_p   0.063636  0.261993  0.245113  0.224470  1.000000  0.218055  0.178307
    mba_p     0.022327  0.388478  0.354823  0.402364  0.218055  1.000000  0.175013
    salary    0.063764  0.035330  0.076819 -0.019272  0.178307  0.175013  1.000000
    <ipython-input-60-2e296c166804>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
      correlation_matrix = placement_data.corr()
```

```
#Identify the column which can be removed?(only 1)
placement_data.drop("ssc_b", axis=1)
```

	sl_no	gender	ssc_p	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN
4	5	M	85.80	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Placed	400000.0
211	212	M	58.00	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Placed	275000.0
212	213	M	67.00	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Placed	295000.0
213	214	F	74.00	66.00	Others	Commerce	58.00	Comm&Mgmt	No	70.0	Mkt&HR	60.23	Placed	204000.0
214	215	M	62.00	58.00	Others	Science	53.00	Comm&Mgmt	No	89.0	Mkt&HR	60.22	Not Placed	NaN

215 rows × 14 columns

```
placement_data.drop('workex', axis=1)
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	75.0	Mkt&Fin	57.80	Placed	250000.0

```
#Check number of null values in each column
placement_data.isnull().sum()
```

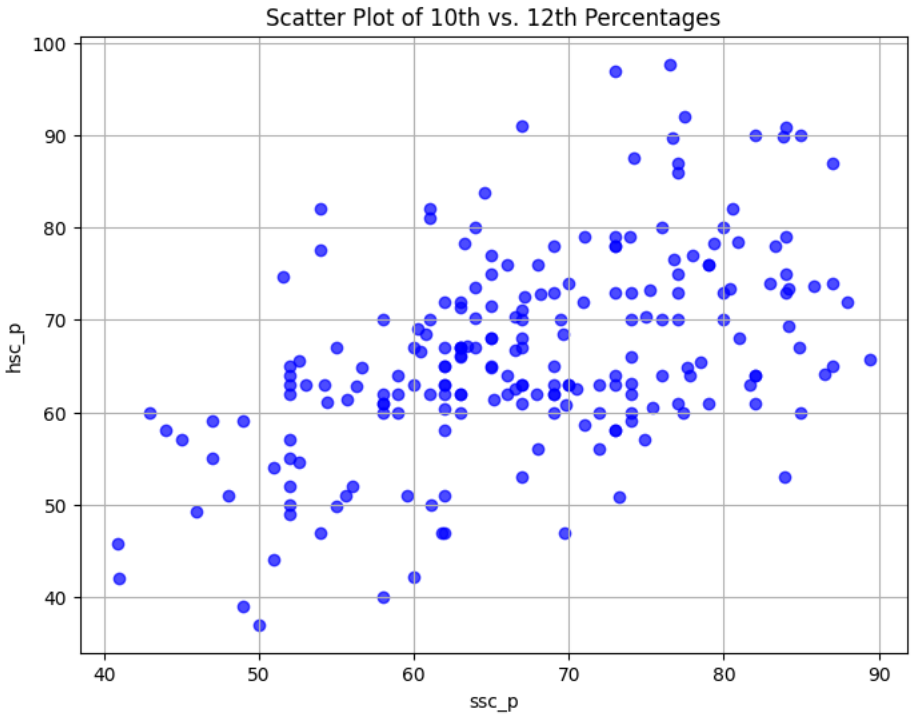
```
sl_no              0
gender             0
ssc_p              0
ssc_b              0
hsc_p              0
hsc_b              0
hsc_s              0
degree_p           0
degree_t           0
workex             0
etest_p            0
specialisation     0
mba_p              0
status             0
salary            67
dtype: int64
```

```
#Fill the missing values with appropriate values and check number of null values in each column again
# To insert the mean value of each column into its missing rows:
placement_data.fillna(placement_data.mean(numeric_only=True).round(1), inplace=True)

# For median:
placement_data.fillna(placement_data.median(numeric_only=True).round(1), inplace=True)

print(placement_data)
```

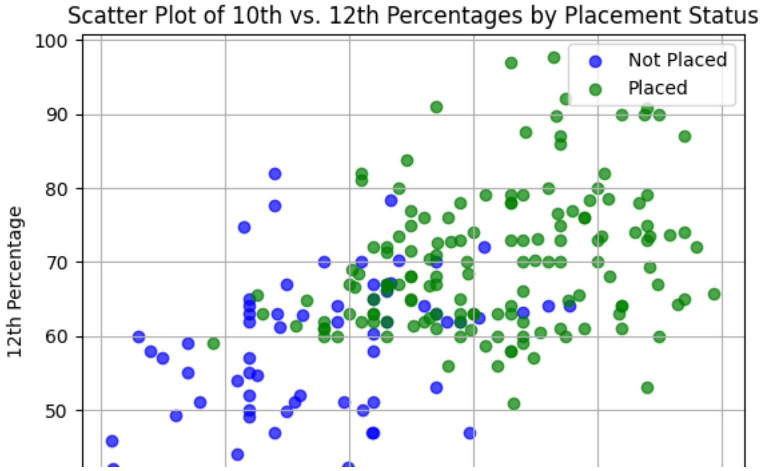
```
     sl_no gender  ssc_p    ssc_b  hsc_p    hsc_b     hsc_s  degree_p  \
0        1      M  67.00   Others  91.00   Others  Commerce     58.00   
1        2      M  79.33  Central  78.33   Others   Science     77.48   
2        3      M  65.00  Central  68.00  Central      Arts     64.00   
3        4      M  56.00  Central  52.00  Central   Science     52.00   
4        5      M  85.80  Central  73.60  Central  Commerce     73.30   
..     ...    ...    ...      ...    ...      ...       ...       ...   
210    211      M  80.60   Others  82.00   Others  Commerce     77.60   
211    212      M  58.00   Others  60.00   Others   Science     72.00   
212    213      M  67.00   Others  67.00   Others  Commerce     73.00   
213    214      F  74.00   Others  66.00   Others  Commerce     58.00   
214    215      M  62.00  Central  58.00   Others   Science     53.00   

      degree_t workex  etest_p specialisation  mba_p      status    salary  
0     Sci&Tech     No     55.0         Mkt&HR  58.80      Placed  270000.0  
1     Sci&Tech    Yes     86.5        Mkt&Fin  66.28      Placed  200000.0  
2    Comm&Mgmt     No     75.0        Mkt&Fin  57.80      Placed  250000.0  
3     Sci&Tech     No     66.0         Mkt&HR  59.43  Not Placed       0.0  
4    Comm&Mgmt     No     96.8        Mkt&Fin  55.50      Placed  425000.0  
..         ...    ...      ...            ...    ...         ...       ...  
210  Comm&Mgmt     No     91.0        Mkt&Fin  74.49      Placed  400000.0  
211   Sci&Tech     No     74.0        Mkt&Fin  53.62      Placed  275000.0  
212  Comm&Mgmt    Yes     59.0        Mkt&Fin  69.72      Placed  295000.0  
213  Comm&Mgmt     No     70.0         Mkt&HR  60.23      Placed  204000.0  
214  Comm&Mgmt     No     89.0         Mkt&HR  60.22  Not Placed       0.0  

[215 rows x 15 columns]
```

```
#Draw a scatter plot between 10th and 12th percentage with labels and title
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Create the scatter plot
plt.figure(figsize=(8, 6))  # Adjust the figure size as needed
plt.scatter(data["ssc_p"], data["hsc_p"], c='blue', alpha=0.7)  # Customize colors and transparency

# Add labels and title
```

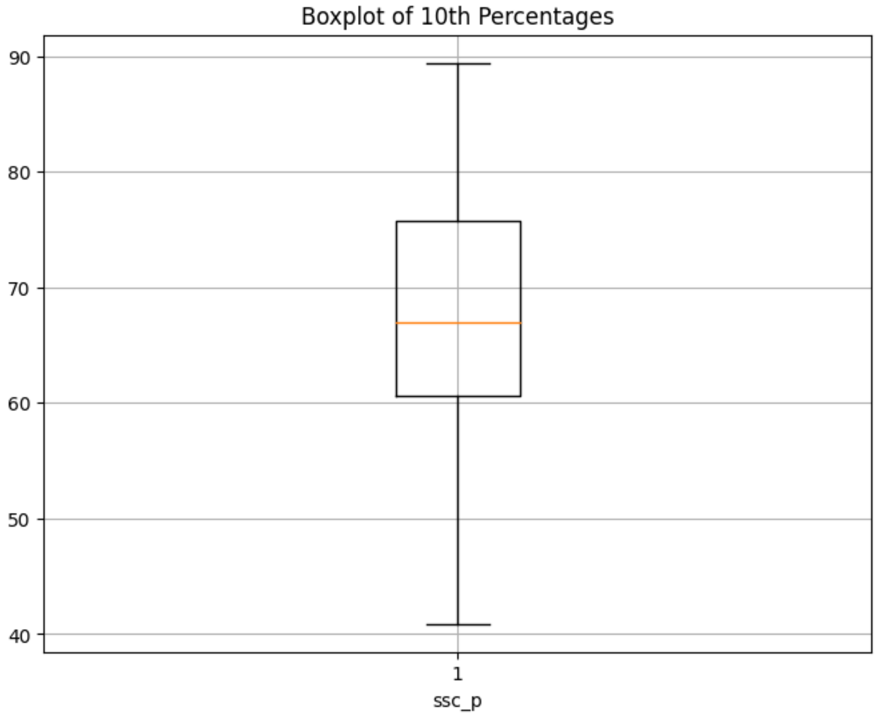
```
plt.xlabel("ssc_p")
plt.ylabel("hsc_p")
plt.title("Scatter Plot of 10th vs. 12th Percentages")

# Show the plot
plt.grid(True)
plt.show()
```



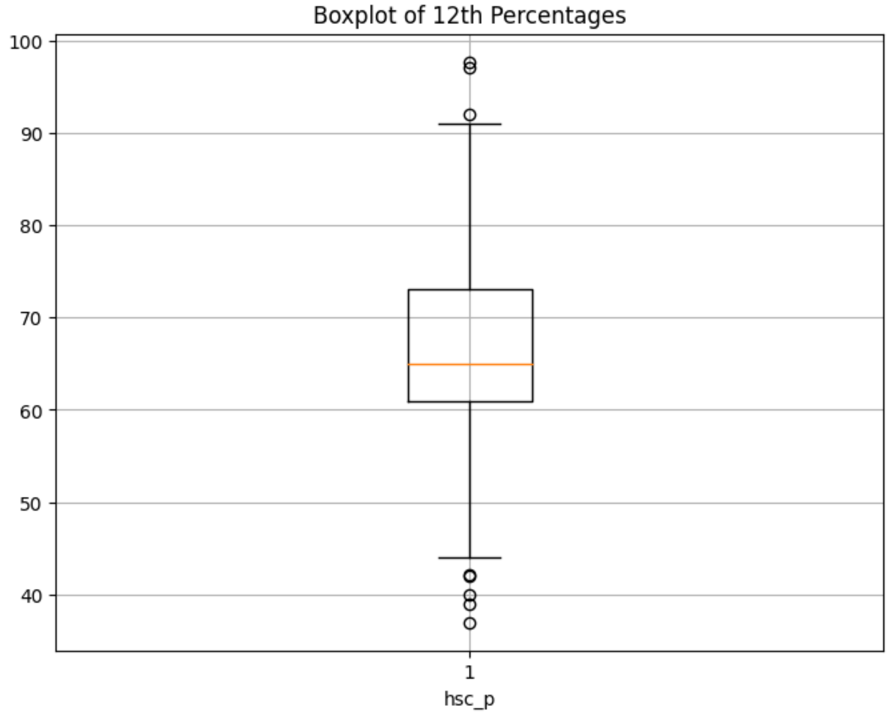
```
#Draw the scatter plot between 10th and 12th class percentage of students grouped based on placement data
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Group the data based on placement status
grouped_data = data.groupby("status")

# Create the scatter plot for each group
colors = ["blue", "green"]  # Assign colors for each group
for i, (status, group_data) in enumerate(grouped_data):
    plt.scatter(
        group_data["ssc_p"],
        group_data["hsc_p"],
        c=colors[i],
        alpha=0.7,
        label=status,
    )

# Add labels and title
plt.xlabel("10th Percentage")
plt.ylabel("12th Percentage")
plt.title("Scatter Plot of 10th vs. 12th Percentages by Placement Status")

# Add legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



```
#Draw a boxplot for 10th percentage of the students
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Create the boxplot
plt.figure(figsize=(8, 6))  # Adjust figure size as needed
plt.boxplot(data["ssc_p"])

# Add labels and title
plt.xlabel("ssc_p")
plt.title("Boxplot of 10th Percentages")

# Show the plot
plt.grid(True)
plt.show()
```



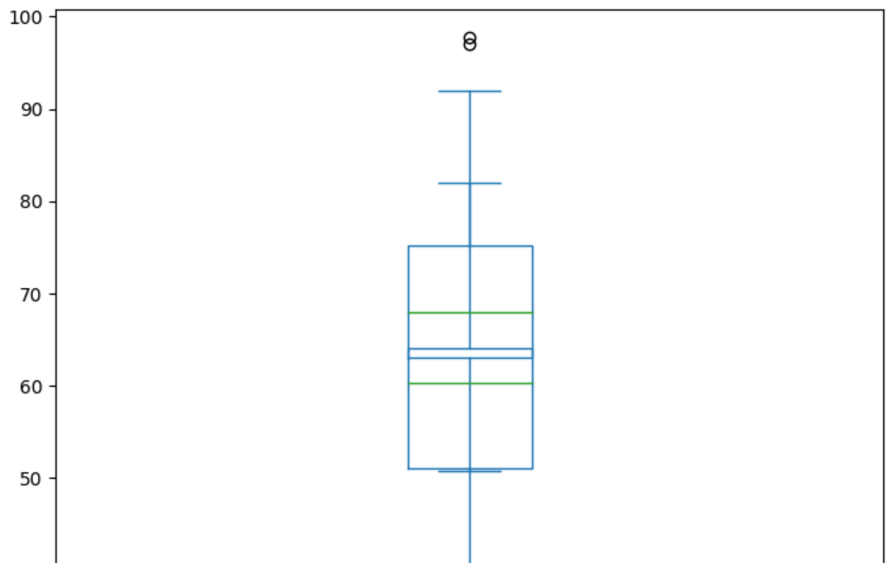
```
#Draw a boxplot for 12th percentage of the students
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Create the boxplot
plt.figure(figsize=(8, 6))  # Adjust figure size as needed
plt.boxplot(data["hsc_p"])

# Add labels and title
plt.xlabel("hsc_p")
plt.title("Boxplot of 12th Percentages")

# Show the plot
plt.grid(True)
plt.show()
```



```
#Draw a boxplot for 12th percentage of the students for placed and unplaced students
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Group the data based on placement status
grouped_data = data.groupby("status")

# Create the boxplots for each group
plt.figure(figsize=(8, 6))  # Adjust figure size as needed
grouped_data["hsc_p"].plot(kind="box")

# Add labels and title
plt.xlabel("status")
plt.ylabel("hsc_p")
plt.title("Boxplots of 12th Percentages by Placement Status")

# Show the plot
plt.grid(True)
plt.show()
```

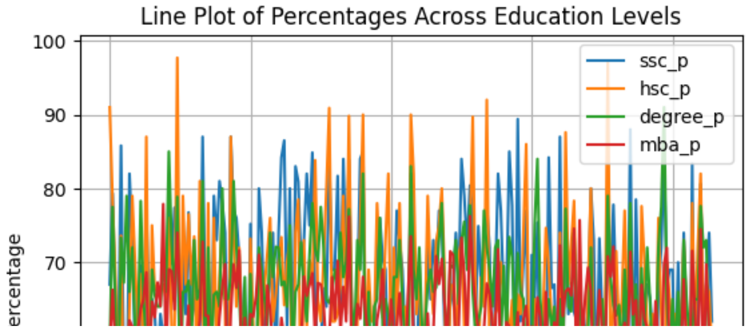
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-3ad38287f7e5> in <cell line: 13>()
     11 # Create the boxplots for each group
     12 plt.figure(figsize=(8, 6))  # Adjust figure size as needed
---> 13 grouped_data["hsc_p"].plot(kind="box")
     14 
     15 # Add labels and title

                              13 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/axis.py in set_ticklabels(self, labels, minor, fontdict, **kwargs)
   1967             # remove all tick labels, so only error for > 0 labels
   1968             if len(locator.locs) != len(labels) and len(labels) != 0:
-> 1969                 raise ValueError(
   1970                     "The number of FixedLocator locations"
   1971                     f" ({len(locator.locs)}), usually from a call to"

ValueError: The number of FixedLocator locations (2), usually from a call to set_ticks, does not match the number of labels (1).
```

EXPLAIN ERROR



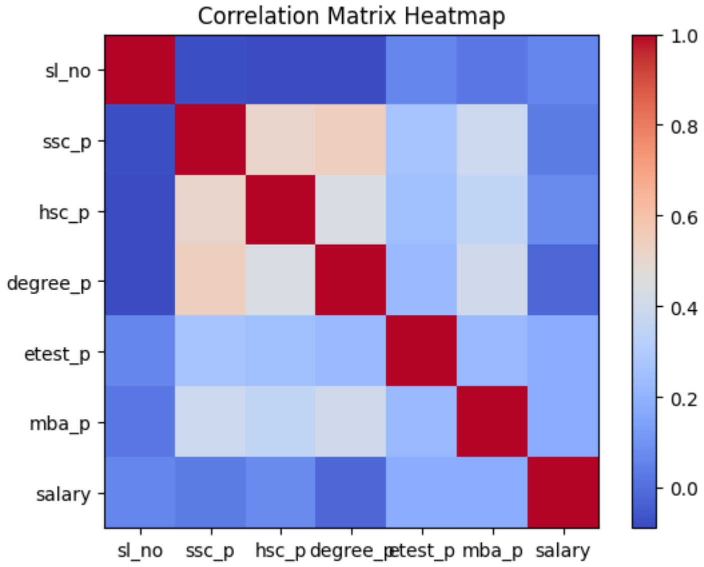
```
#Draw lineplot for 10th, 12th, degree and MBA percentage
import matplotlib.pyplot as plt
import pandas as pd

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")

# Extract the relevant columns
percentages = data[["ssc_p", "hsc_p", "degree_p", "mba_p"]]

# Create the line plot
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
percentages.plot(kind="line")

# Add labels and title
plt.xlabel("Education Level")
plt.ylabel("Percentage")
plt.title("Line Plot of Percentages Across Education Levels")

# Show the plot
plt.grid(True)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



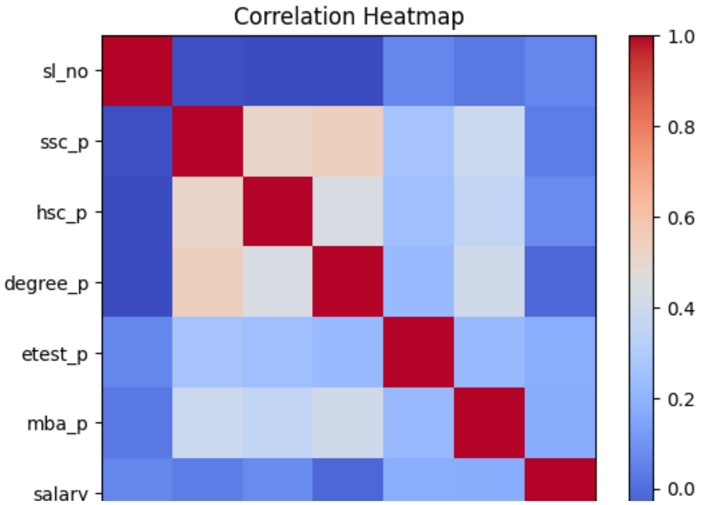
```
#Find correlation between continous columns
import pandas as pd
data = pd.read_csv("/content/placement_data/Placement_Data.csv")
correlation_matrix = data.corr()
print(correlation_matrix)
import matplotlib.pyplot as plt
plt.imshow(correlation_matrix, cmap='coolwarm')  # Use a colormap to visualize correlations
plt.colorbar()  # Add a colorbar to interpret the intensity of correlations
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)  # Label the x-axis with column names
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)  # Label the y-axis with column names
plt.title("Correlation Matrix Heatmap")
plt.show()
```

```
<ipython-input-36-da2c1551279d>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  correlation_matrix = data.corr()
             sl_no     ssc_p     hsc_p  degree_p   etest_p     mba_p    salary
sl_no     1.000000 -0.078155 -0.085711 -0.088281  0.063636  0.022327  0.063764
ssc_p    -0.078155  1.000000  0.511472  0.538404  0.261993  0.388478  0.035330
hsc_p    -0.085711  0.511472  1.000000  0.434206  0.245113  0.354823  0.076819
degree_p -0.088281  0.538404  0.434206  1.000000  0.224470  0.402364 -0.019272
etest_p   0.063636  0.261993  0.245113  0.224470  1.000000  0.218055  0.178307
mba_p     0.022327  0.388478  0.354823  0.402364  0.218055  1.000000  0.175013
salary    0.063764  0.035330  0.076819 -0.019272  0.178307  0.175013  1.000000
```



```
#Draw heatmap of correlation
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the CSV file
data = pd.read_csv("/content/placement_data/Placement_Data.csv")  # Replace "your_data.csv" with the actual filename

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Create the heatmap
plt.imshow(correlation_matrix, cmap='coolwarm')  # Use a colormap to visualize correlations
plt.colorbar()  # Add a colorbar to interpret the intensity of correlations
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=45, ha='right')  # Label the x-axis with column name
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)  # Label the y-axis with column names
plt.title("Correlation Heatmap")
plt.show()
```

```
<ipython-input-37-0319b388cd8d>:9: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  correlation_matrix = data.corr()
```


Correlation Heatmap
sl_no
ssc_p
hsc_p
degree_p
etest_p
mba_p
salary