Data Inspection

Understand how to examine your dataset using functions like head(), tail(), info(), describe(), and shape to get a quick overview of the data.

.head(): returns the first few rows (the “head” of the DataFrame). .info() shows information on each of the columns, such as the data type and number of missing values. .shape returns the number of rows and columns of the DataFrame. .describe() calculates a few summary statistics for each column.

In [2]:

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 22, 27, 24],
    'City': ['New York', 'San Francisco', 'Los Angeles', 'Chicago', 'Miami'
    'Salary': [50000, 60000, 45000, 55000, 52000]
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age           City  Salary
0    Alice   25       New York   50000
1      Bob   30  San Francisco   60000
2  Charlie   22    Los Angeles   45000
3    David   27        Chicago   55000
4      Eva   24          Miami   52000
```

In [6]:
```
print(df.head())
print('INFO')
print(df.info())
print('SHAPE')
print(df.shape)
print('DESCRIBE')
print(df.describe())
```

```
      Name  Age           City  Salary
0    Alice   25       New York   50000
1      Bob   30  San Francisco   60000
2  Charlie   22    Los Angeles   45000
3    David   27        Chicago   55000
4      Eva   24          Miami   52000
INFO
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Name    5 non-null      object
 1   Age     5 non-null      int64 
 2   City    5 non-null      object
 3   Salary  5 non-null      int64 
dtypes: int64(2), object(2)
memory usage: 288.0+ bytes
None
SHAPE
(5, 4)
DESCRIBE
             Age        Salary
count   5.00000      5.000000
mean   25.60000  52400.000000
std     3.04959   5594.640292
min    22.00000  45000.000000
25%    24.00000  50000.000000
50%    25.00000  52000.000000
75%    27.00000  55000.000000
max    30.00000  60000.000000
```

Sort

In [10]:
```
print(df_age)
```

```
      Name  Age           City  Salary
2  Charlie   22    Los Angeles   45000
4      Eva   24          Miami   52000
0    Alice   25       New York   50000
3    David   27        Chicago   55000
1      Bob   30  San Francisco   60000
```

Subset

In [12]:
```
df_name=df['Name']

print(df_name)
```

In [4]:
```
df_young=df[df['Age']<25]

print(df_young)
```

```
      Name  Age         City  Salary
2  Charlie   22  Los Angeles   45000
4      Eva   24        Miami   52000
```

In [7]:
```
df_name_salary=df[['Name','Salary']]
print(df_name_salary)
```

```
      Name  Salary
0    Alice   50000
1      Bob   60000
2  Charlie   45000
3    David   55000
4      Eva   52000
```

In [5]:
```
#sort by multiple variables
canu = ["Los Angeles", "Miami"]
df_city = df.isin(canu)

print(df_city)
```

```
    Name    Age   City  Salary
0  False  False  False   False
1  False  False  False   False
2  False  False   True   False
3  False  False  False   False
4  False  False   True   False
```

Summary Statistics

In [4]:
```
print(df['Salary'].mean())
print(df['Salary'].median())
```

```
52400.0
52000.0
```

In [6]:
```
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)
print(df["Salary"].agg(iqr))
```

```
5000.0
```

In [7]:
```
print(df['Salary'].cumsum())
```

```
0     50000
1    110000
2    155000
3    210000
4    262000
Name: Salary, dtype: int64
```

In [9]:
```
df.drop_duplicates(subset="Age")
ptt= df.pivot_table(values='Age',index='Name')
```

In [10]:
```
print(ptt)
```

```
         Age
Name
Alice     25
Bob       30
Charlie   22
David     27
Eva       24
```

Explicit indexes

Setting & removing indexes

In [4]:
```
df_ind = df.set_index('Name')
print(df_ind)
```

```
         Age           City  Salary
Name
Alice     25       New York   50000
Bob       30  San Francisco   60000
Charlie   22    Los Angeles   45000
David     27        Chicago   55000
Eva       24          Miami   52000
```

In [5]:
```
df_ind.reset_index()
```

Out[5]:

	Name	Age	City	Salary
0	Alice	25	New York	50000
1	Bob	30	San Francisco	60000
2	Charlie	22	Los Angeles	45000
3	David	27	Chicago	55000
4	Eva	24	Miami	52000

In [10]:
```
df_ind.loc['Eva']
```

Out[10]:
```
Age          24
City      Miami
Salary    52000
Name: Eva, dtype: object
```

In [12]:
```
df_ind.sort_index(level=['Age','Salary'])
```

Out[12]:

Name	Age	City	Salary
Alice	25	New York	50000
Bob	30	San Francisco	60000
Charlie	22	Los Angeles	45000
David	27	Chicago	55000
Eva	24	Miami	52000

Slicing index values

In [6]:
```
df_ind=df.set_index('Name')
print(df_ind.loc['Alice':'Charlie'])
```

```
         Age           City  Salary
Name
Alice     25       New York   50000
Bob       30  San Francisco   60000
Charlie   22    Los Angeles   45000
```

Slicing in both direction

In [13]:
```
df_inds = df.set_index(['Name', 'Age'])
print(df_inds.loc[('Alice', 25):('Charlie', 22)])
```

```
                     City  Salary
Name    Age
Alice   25        New York   50000
Bob     30   San Francisco   60000
Charlie 22     Los Angeles   45000
```

In [15]:
```
print(df.iloc[1:3])
```

```
      Name  Age           City  Salary
1      Bob   30  San Francisco   60000
2  Charlie   22    Los Angeles   45000
```

Pivot tables

In [19]:
```
data1 = {
    'Date': ['2023-07-01', '2023-07-01', '2023-07-02', '2023-07-02'],
    'Product': ['A', 'B', 'A', 'B'],
    'Sales': [100, 150, 120, 130]
}

sales_df = pd.DataFrame(data1)

# Create a pivot table to summarize sales by product and date
pivot_table = sales_df.pivot_table(index='Date', columns='Product', values=
print(pivot_table)
```

```
Product       A    B
Date
2023-07-01  100  150
2023-07-02  120  130
```

Visualizing DataFrames

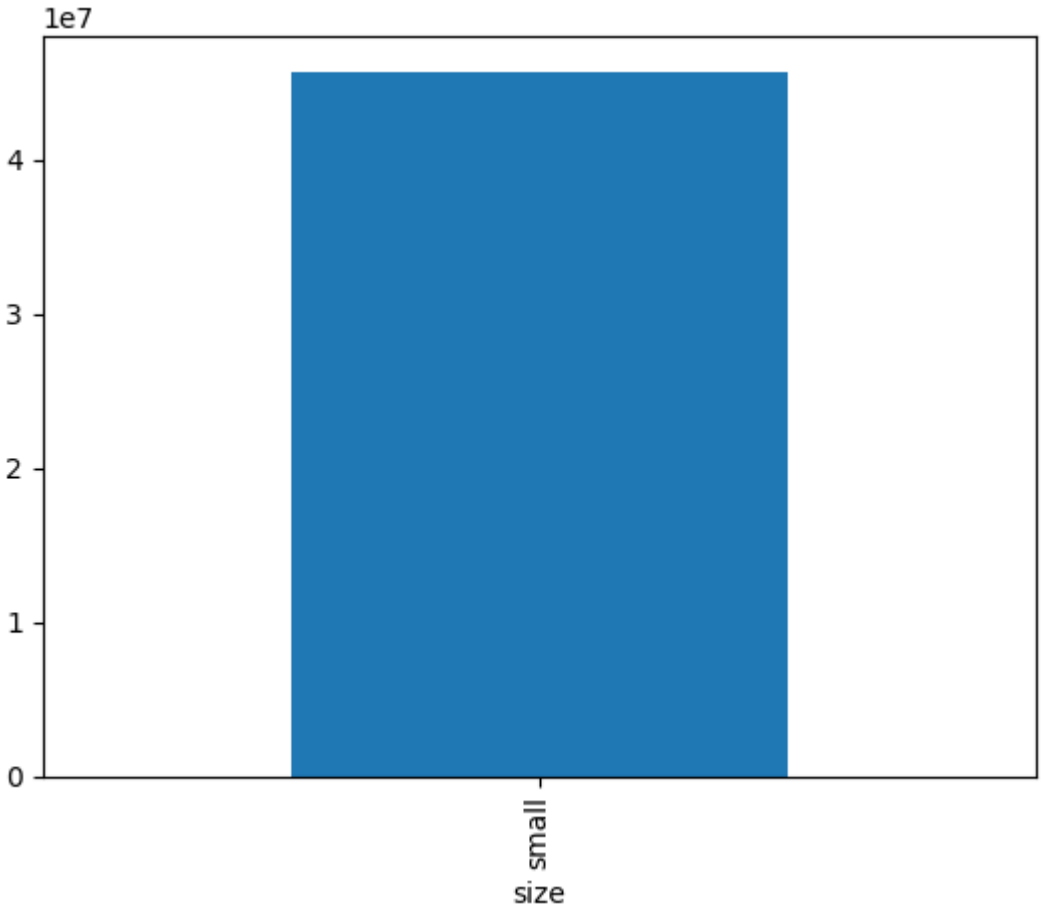
In [1]:
```
import pandas as pd

data = {
    'date': ['2015-12-27', '2015-12-20', '2015-12-13', '2015-12-06', '2015-
    'type': ['conventional', 'conventional', 'conventional', 'conventional'
    'year': [2015, 2015, 2015, 2015, 2015],
    'avg_price': [0.95, 0.98, 0.93, 0.89, 0.99],
    'size': ['small', 'small', 'small', 'small', 'small'],
    'nb_sold': [9627000, 8710000, 9855000, 9405000, 8095000]
}

avocado_df = pd.DataFrame(data)
print(avocado_df)
```

```
         date          type  year  avg_price   size  nb_sold
0  2015-12-27  conventional  2015       0.95  small  9627000
1  2015-12-20  conventional  2015       0.98  small  8710000
2  2015-12-13  conventional  2015       0.93  small  9855000
3  2015-12-06  conventional  2015       0.89  small  9405000
4  2015-11-29  conventional  2015       0.99  small  8095000
```

In [3]:
```
import matplotlib.pyplot as plt
sold_by_size = avocado_df.groupby('size')['nb_sold'].sum()
```

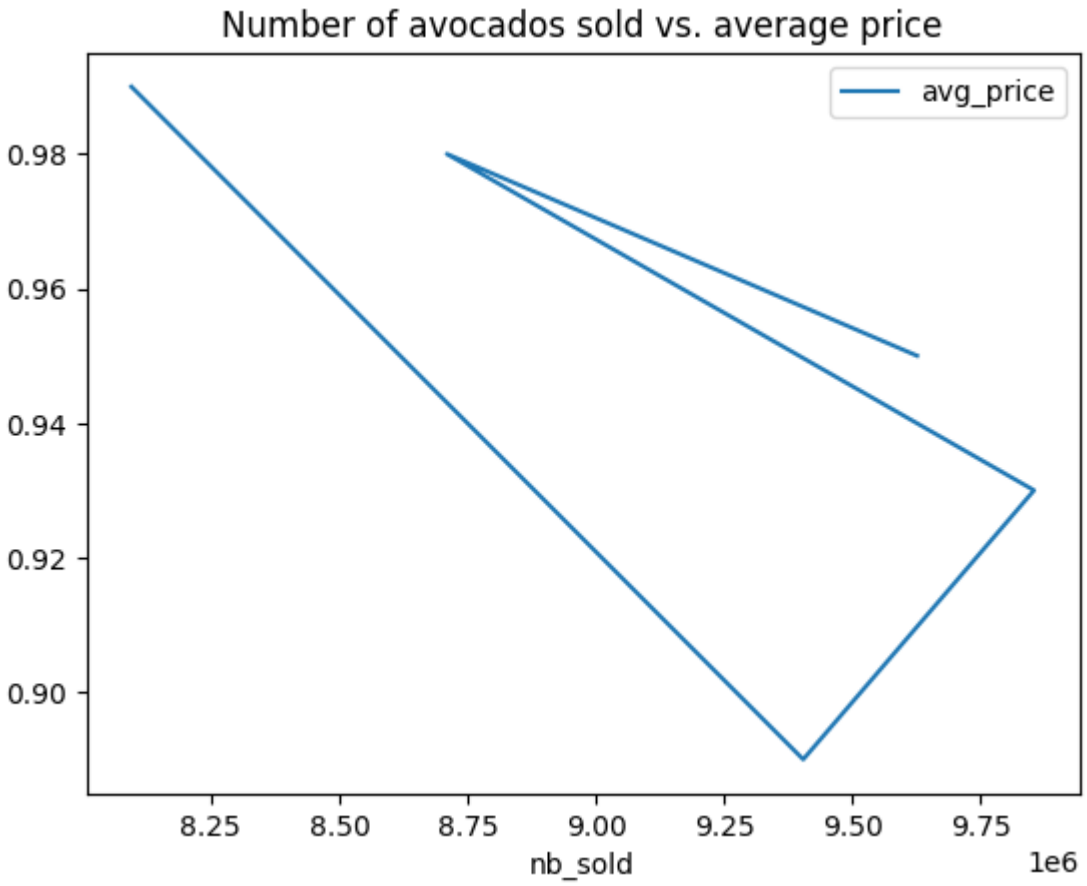
In [5]:

```
sold_by_size.plot(kind='bar')
plt.show()
```

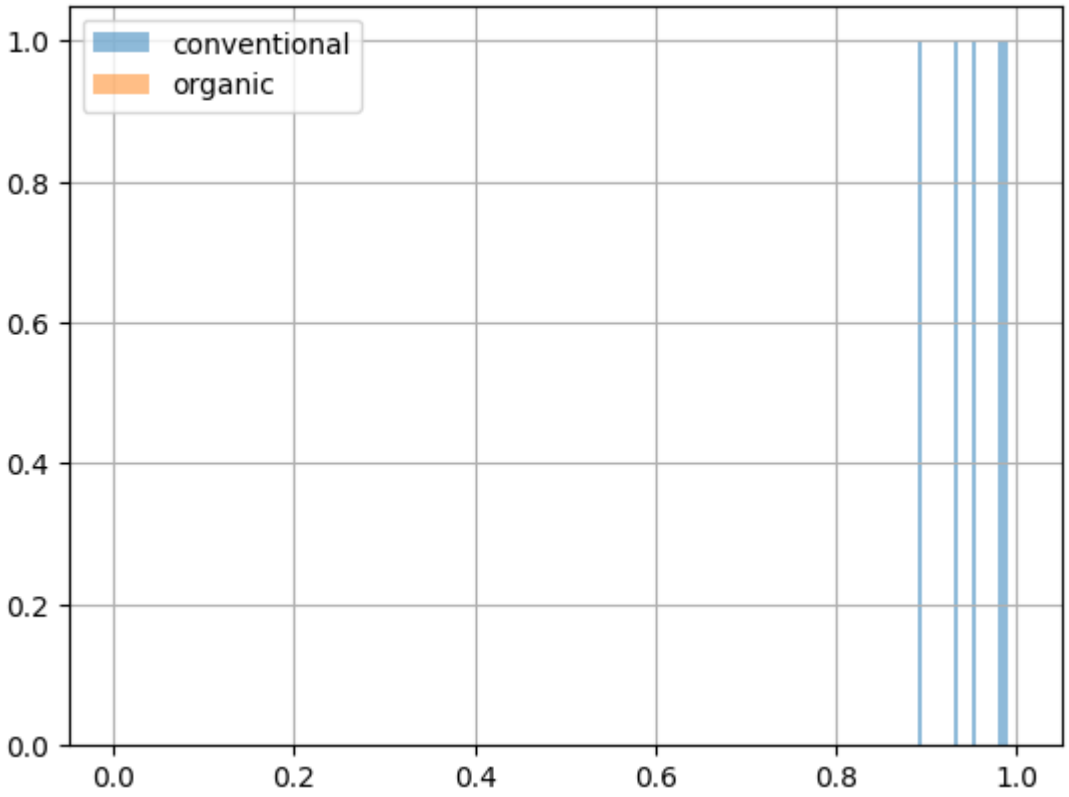


In [9]:

```
avocado_df.plot(x='nb_sold', y='avg_price', kind='line',title="Number of av

plt.show()
```



In [11]:
```
# # Histogram of conventional avg_price
avocado_df[avocado_df["type"] == "conventional"]["avg_price"].hist(bins=20,

# Histogram of organic avg_price
avocado_df[avocado_df["type"] == "organic"]["avg_price"].hist(bins=20, alph

# Add a legend
plt.legend(["conventional", "organic"])

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



Missing values

In [15]:
```
avocado_df.isna()
```

Out[15]:

	date	type	year	avg_price	size	nb_sold
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False

In [16]:
```
avocado_df.isna().any()  #checks each column
```

Out[16]:
```
date          False
type          False
year          False
avg_price     False
size          False
nb_sold       False
dtype: bool
```

In [17]:
```
avocado_df.dropna()   #Remove Nan
avocado_df.fillna(0)      #Fill Nan with 0
```

Out[17]:

	date	type	year	avg_price	size	nb_sold
0	2015-12-27	conventional	2015	0.95	small	9627000
1	2015-12-20	conventional	2015	0.98	small	8710000
2	2015-12-13	conventional	2015	0.93	small	9855000
3	2015-12-06	conventional	2015	0.89	small	9405000
4	2015-11-29	conventional	2015	0.99	small	8095000

Reading and writing CSVs

In [ ]:
```
pd.read_csv("file_name")
```

In [ ]:
```
dateframe.to_csv("name of the CSV file you want to create")
```

Joining Data with Pandas

In [15]:
```
import pandas as pd

data1 = {
    'student_id': [1, 2, 3, 4, 5],
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'age': [20, 21, 19, 22, 20]
}

students_df = pd.DataFrame(data1)

data2 = {
    'student_id': [1, 2, 3, 4, 5],
    'math_grade': [85, 90, 78, 95, 88],
    'science_grade': [92, 88, 79, 97, 85]
}

grades_df = pd.DataFrame(data2)

data3 = {
    'teacher_id': [1, 2, 3, 4, 5],
    'math_grade': [85, 90, 78, 95, 88],

}

teacher_df = pd.DataFrame(data3)
```

In [13]:
```
student_grades=students_df.merge(grades_df,on='student_id',suffixes=('_own'
print(student_grades)
```

	student_id	name	age	math_grade	science_grade
0	1	Alice	20	85	92
1	2	Bob	21	90	88
2	3	Charlie	19	78	79
3	4	David	22	95	97
4	5	Eve	20	88	85

In [17]:
```
std_grad_teacher=students_df.merge(grades_df,on='student_id').merge(teacher
print(std_grad_teacher)
```

	student_id	name	age	math_grade	science_grade	teacher_id
0	1	Alice	20	85	92	1
1	2	Bob	21	90	88	2
2	3	Charlie	19	78	79	3
3	4	David	22	95	97	4
4	5	Eve	20	88	85	5

In [6]:
```
import pandas as pd

data = {
    'product': ['A', 'B', 'A', 'B'],
    'date': ['2023-07-01', '2023-07-01', '2023-07-02', '2023-07-02'],
    'sales': [100, 150, 120, 130]
}

sales_df = pd.DataFrame(data)

merged_sales = pd.merge(sales_df, sales_df, on='product', suffixes=('_left'
print(merged_sales)
```

```
  product   date_left  sales_left  date_right  sales_right
0       A  2023-07-01         100  2023-07-01          100
1       A  2023-07-01         100  2023-07-02          120
2       A  2023-07-02         120  2023-07-01          100
3       A  2023-07-02         120  2023-07-02          120
4       B  2023-07-01         150  2023-07-01          150
5       B  2023-07-01         150  2023-07-02          130
6       B  2023-07-02         130  2023-07-01          150
7       B  2023-07-02         130  2023-07-02          130
```

In [19]:
```
data1 = {
    'key': ['A', 'B', 'C'],
    'value1': [10, 20, 30]
}

df1 = pd.DataFrame(data1)
df1 = df1.set_index('key')

data2 = {
    'value2': [100, 200, 300]
}

df2 = pd.DataFrame(data2)

merged_with_index = df1.merge(df2, left_index=True, right_index=True)
print(merged_with_index)
```

```
Empty DataFrame
Columns: [value1, value2]
Index: []
```

.query()

In [7]:
```
filtered_sales = sales_df.query('sales > 130')
print(filtered_sales)
```

```
  product        date  sales
1       B  2023-07-01    150
```

.melt()

The .melt() function in pandas is used to transform or reshape a DataFrame from a wide format to a long format, making the data more suitable for analysis or visualization.

In [8]:

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Jan': [100, 150, 120],
    'Feb': [110, 160, 130],
    'Mar': [120, 170, 140]
}

wide_data = pd.DataFrame(data)

long_data = wide_data.melt(id_vars='Name', var_name='Month', value_name='Va
print(long_data)

#DataFrame.melt(
#    id_vars=None,
#    value_vars=None,
#    var_name=None,
#    value_name='value',
#    col_level=None
#)
```

```
      Name Month  Value
0    Alice   Jan    100
1      Bob   Jan    150
2  Charlie   Jan    120
3    Alice   Feb    110
4      Bob   Feb    160
5  Charlie   Feb    130
6    Alice   Mar    120
7      Bob   Mar    170
8  Charlie   Mar    140
```

merge_ordered

In [1]:
```
import pandas as pd

data1 = {'date': ['2023-01-01', '2023-02-01', '2023-03-01'],
         'value_df1': [10, 20, 30]}

data2 = {'date': ['2023-02-01', '2023-03-01', '2023-04-01'],
         'value_df2': [200, 300, 400]}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

merged_ordered = pd.merge_ordered(df1, df2, on='date')
print(merged_ordered)
```

```
         date  value_df1  value_df2
0  2023-01-01       10.0        NaN
1  2023-02-01       20.0      200.0
2  2023-03-01       30.0      300.0
3  2023-04-01        NaN      400.0
```

merge_asof()

merge_ordered() maintains the order of values in the specified column, which is important for time-series data.

merge_asof() finds the nearest values in the specified column, allowing for merging of data with some degree of mismatch.

In [10]:

```
data3 = {'date': ['2023-01-01', '2023-02-01', '2023-03-01', '2023-04-01'],
         'value_df3': [100, 200, 300, 400]}

data4 = {'date': ['2023-02-15', '2023-03-01', '2023-04-10'],
         'value_df4': [250, 350, 450]}

df3 = pd.DataFrame(data3)
df4 = pd.DataFrame(data4)

# Convert the 'date' column to datetime format
df3['date'] = pd.to_datetime(df3['date'])
df4['date'] = pd.to_datetime(df4['date'])

# Merge using pd.merge_asof
merged_asof = pd.merge_asof(df3, df4, on='date')
print(merged_asof)
```

```
        date  value_df3  value_df4
0 2023-01-01        100        NaN
1 2023-02-01        200        NaN
2 2023-03-01        300      350.0
3 2023-04-01        400      350.0
```