Introduction to Pandas

by Heidari

Pandas

Pandas is a fast, powerful, flexible and easy to use Python library for data manipulation and analysis.

Check Pandas documentation: https://pandas.pydata.org/docs/getting_started/index.html#getting-started (https://pandas.pydata.org/docs/getting_started/index.html#getting-started)

In [1]:
```
#Install pandas
#%pip install pandas
```

In [2]:
```
#Import required libraries
import numpy as np
import pandas as pd
```

Pandas Series

In [3]:
```
#Demand for four different products
s1 = pd.Series([1750, 1200, 450, 2000])
s1
```

Out[3]:
```
0    1750
1    1200
2     450
3    2000
dtype: int64
```

In [4]:
```
type(s1)
```

Out[4]:
```
pandas.core.series.Series
```

In [5]:
```
#without name
s1.name
```

In [6]:
```
#Assign new name
s1.name = 'product_demand'
s1
```

Out[6]:
```
0    1750
1    1200
2     450
3    2000
Name: product_demand, dtype: int64
```

In [7]:
```
s1.values
```

Out[7]:
```
array([1750, 1200,  450, 2000], dtype=int64)
```

In [8]:
```
type(s1.values)
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
s1.index
```

Out[9]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [10]:
```
#Extract an item from Series with index
s1[0]
```

Out[10]:
```
1750
```

In [11]:
```
s1.index = ['p1', 'p2', 'p3', 'p4']
s1.index
```

Out[11]:
```
Index(['p1', 'p2', 'p3', 'p4'], dtype='object')
```

In [12]:
```
s1['p1']
```

Out[12]:
```
1750
```

In [13]:
```
#Modifying Series
s1['p1'] = 1000
s1
```

Out[13]:
```
p1    1000
p2    1200
p3     450
p4    2000
Name: product_demand, dtype: int64
```

In [14]:
```
s1['p5'] = 1250
s1
```

Out[14]:
```
p1    1000
p2    1200
p3     450
p4    2000
p5    1250
Name: product_demand, dtype: int64
```

In [15]:
```
#Series Operations
s1 > 1000
```

Out[15]:
```
p1    False
p2     True
p3    False
p4     True
p5     True
Name: product_demand, dtype: bool
```

In [16]:
```
s1[s1 > 1000]
```

Out[16]:
```
p2    1200
p4    2000
p5    1250
Name: product_demand, dtype: int64
```

In [17]:
```
s1 * 1.10
```

Out[17]:
```
p1    1100.0
p2    1320.0
p3     495.0
p4    2200.0
p5    1375.0
Name: product_demand, dtype: float64
```

In [18]:
```
s1 = s1 * 1.10
s1
```

Out[18]:
```
p1    1100.0
p2    1320.0
p3     495.0
p4    2200.0
p5    1375.0
Name: product_demand, dtype: float64
```

In [19]:
```
s1.mean()
```

Out[19]:
```
1298.0
```

In [20]:
```
s1.std()
```

Out[20]:
```
613.1945042154243
```

In [21]:
```
#  &: and
#  |: or
#  ~: not

s1[(s1 < s1.mean()) & (s1 > 1000)]
```

Out[21]:
```
p1    1100.0
Name: product_demand, dtype: float64
```

In [22]:
```
s1[s1 <= 1000] = 0
s1
```

Out[22]:
```
p1    1100.0
p2    1320.0
p3       0.0
p4    2200.0
p5    1375.0
Name: product_demand, dtype: float64
```

Dataframes

In [23]:
```
df = pd.DataFrame({'name'   : ['p1', 'p2', 'p3', 'p4'],
                   'demand': [1750, 1200, 450, 2000],
                   'brand' : ['x', 'x', 'y', 'z'],
                   'weight': [150, 200, 1500, 200]})
df
```

Out[23]:

	name	demand	brand	weight
0	p1	1750	x	150
1	p2	1200	x	200
2	p3	450	y	1500
3	p4	2000	z	200

In [24]:
```
type(df)
```

Out[24]:
```
pandas.core.frame.DataFrame
```

In [25]:
```
df.shape
```

Out[25]:
```
(4, 4)
```

In [26]:
```
df.columns
```

Out[26]:
```
Index(['name', 'demand', 'brand', 'weight'], dtype='object')
```

In [27]:
```
df.index
```

Out[27]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [28]:
```
df.index = ['p1', 'p2', 'p3', 'p4']
df
```

Out[28]:

	name	demand	brand	weight
p1	p1	1750	x	150
p2	p2	1200	x	200
p3	p3	450	y	1500
p4	p4	2000	z	200

In [29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, p1 to p4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   name    4 non-null      object
 1   demand  4 non-null      int64 
 2   brand   4 non-null      object
 3   weight  4 non-null      int64 
dtypes: int64(2), object(2)
memory usage: 160.0+ bytes
```

Subsetting Dataframes

In [30]:
```
df
```

Out[30]:

	name	demand	brand	weight
p1	p1	1750	x	150
p2	p2	1200	x	200
p3	p3	450	y	1500
p4	p4	2000	z	200

In [31]:
```
df.demand
```

Out[31]:
```
p1    1750
p2    1200
p3     450
p4    2000
Name: demand, dtype: int64
```

In [32]:
```
df['demand']
```

Out[32]:
```
p1    1750
p2    1200
p3     450
p4    2000
Name: demand, dtype: int64
```

In [33]:
```
df[['name', 'demand']]
```

Out[33]:

	name	demand
p1	p1	1750
p2	p2	1200
p3	p3	450
p4	p4	2000

In [34]:
```
#.iloc[ ] allows us to retrieve rows and columns by position.
df.iloc[0, 1]
```

Out[34]:
```
1750
```

In [35]:
```
#Extract information of first product
df.iloc[0, :]
```

Out[35]:
```
name        p1
demand    1750
brand        x
weight     150
Name: p1, dtype: object
```

In [36]:
```
#Extract brand column
df.iloc[:, 2]
```

Out[36]:
```
p1    x
p2    x
p3    y
p4    z
Name: brand, dtype: object
```

In [37]:
```
#Extract name & brand columns for p1, p3, and p4
df.iloc[[0, 2, 3], [0, 2]]
```

Out[37]:

	name	brand
p1	p1	x
p3	p3	y
p4	p4	z

In [38]:
```
#.loc[] selects data by the label of the rows and columns.
#Extract information of first product
df.loc['p1', :]
```

Out[38]:
```
name        p1
demand    1750
brand        x
weight     150
Name: p1, dtype: object
```

In [39]:
```
#Extract brand column
df.loc[:, 'brand']
```

Out[39]:
```
p1    x
p2    x
p3    y
p4    z
Name: brand, dtype: object
```

In [40]:
```
#Extract name & brand columns for p1, p3, and p4
df.loc[['p1', 'p3', 'p4'], ['name', 'brand']]
```

Out[40]:

	name	brand
p1	p1	x
p3	p3	y
p4	p4	z

In [41]:
```
#Check if demand below 1500
df['demand'] < 1500
```

Out[41]:
```
p1    False
p2     True
p3     True
p4    False
Name: demand, dtype: bool
```

In [42]:
```
#Identify products with demand below 1500
df.loc[df['demand'] < 1500, 'name']
```

Out[42]:
```
p2    p2
p3    p3
Name: name, dtype: object
```

Modifying Dataframes

In [43]:
```
#Add new column
df['price'] = [20, 15, 50, 10]
df
```

Out[43]:

	name	demand	brand	weight	price
p1	p1	1750	x	150	20
p2	p2	1200	x	200	15
p3	p3	450	y	1500	50
p4	p4	2000	z	200	10

In [44]:
```
#Add new row
df.loc['p5', :] = pd.Series({'name': 'p5',
                             'demand': 1000,
                             'brand': 'x',
                             'weight': 500,
                             'price': 60})
df
```

Out[44]:

	name	demand	brand	weight	price
p1	p1	1750.0	x	150.0	20.0
p2	p2	1200.0	x	200.0	15.0
p3	p3	450.0	y	1500.0	50.0
p4	p4	2000.0	z	200.0	10.0
p5	p5	1000.0	x	500.0	60.0

In [45]:
```
#Change demand of 'p3' into 700
df.loc['p3', 'demand'] = 700
df
```

Out[45]:

	name	demand	brand	weight	price
p1	p1	1750.0	x	150.0	20.0
p2	p2	1200.0	x	200.0	15.0
p3	p3	700.0	y	1500.0	50.0
p4	p4	2000.0	z	200.0	10.0
p5	p5	1000.0	x	500.0	60.0

In [46]:
```
#Calculate total revenue for each product
df['revenue'] = df['demand'] * df['price']
df
```

Out[46]:

	name	demand	brand	weight	price	revenue
p1	p1	1750.0	x	150.0	20.0	35000.0
p2	p2	1200.0	x	200.0	15.0	18000.0
p3	p3	700.0	y	1500.0	50.0	35000.0
p4	p4	2000.0	z	200.0	10.0	20000.0
p5	p5	1000.0	x	500.0	60.0	60000.0

In [47]:
```
#Modifying dataframe using conditional selection
#Question: increase price of products below 20 by 5% and
# recalculate monthly revenue
df.loc[df['price'] < 20, 'price'] = 1.05 * df.loc[df['price'] < 20, 'price']
df['revenue'] = df['demand'] * df['price']
df
```

Out[47]:

	name	demand	brand	weight	price	revenue
p1	p1	1750.0	x	150.0	20.00	35000.0
p2	p2	1200.0	x	200.0	15.75	18900.0
p3	p3	700.0	y	1500.0	50.00	35000.0
p4	p4	2000.0	z	200.0	10.50	21000.0
p5	p5	1000.0	x	500.0	60.00	60000.0

Import CSV Data File into a Pandas Dataframe

In [48]:
```
#Get work directory
import os
os.getcwd()
```

Out[48]:
```
'C:\\Users\\FarzadM'
```

In [49]:
```
#Read from work directory
data = pd.read_csv('sample_data.csv')
```

In [50]:
```
#Read from desktop
data = pd.read_csv('C:\\Users\\FarzadM\\Desktop\\sample_data.csv')
```

In [51]:
```
type(data)
```

Out[51]:
```
pandas.core.frame.DataFrame
```

In [52]:
```
data
```

Out[52]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state_of_res
0	2068	F	False	11300.0	Married	True	Homeowner free and clear	False	2.0	49.0	Michigan
1	2073	F	False	0.0	Married	True	Rented	True	3.0	40.0	Florida
2	2848	M	True	45000.0	Never Married	False	Rented	True	3.0	29.0	Georgia
3	5641	M	True	20000.0	Never Married	False	Occupied with no rent	False	0.0	22.0	New Mexico
4	6369	F	True	42000.0	Never Married	True	Rented	True	1.0	31.0	Florida
...	...	...	...	...	...	...	...	...	...	...	...
495	685994	F	True	21100.0	Married	False	Homeowner free and clear	False	2.0	63.0	Maryland
496	688580	M	True	48000.0	Married	True	Rented	True	2.0	22.0	Florida
497	688736	F	NaN	0.0	Married	True	Rented	False	2.0	34.0	Iowa
498	692445	M	True	140000.0	Married	True	Homeowner with mortgage/loan	False	5.0	48.0	Illinois
499	693235	M	True	36200.0	Married	True	Homeowner free and clear	False	2.0	43.0	Michigan

500 rows × 11 columns

Example: CRM database

In [53]:
```
data.head()
```

Out[53]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state_of_res
0	2068	F	False	11300.0	Married	True	Homeowner free and clear	False	2.0	49.0	Michigan
1	2073	F	False	0.0	Married	True	Rented	True	3.0	40.0	Florida
2	2848	M	True	45000.0	Never Married	False	Rented	True	3.0	29.0	Georgia
3	5641	M	True	20000.0	Never Married	False	Occupied with no rent	False	0.0	22.0	New Mexico
4	6369	F	True	42000.0	Never Married	True	Rented	True	1.0	31.0	Florida

In [54]:
```
data.tail()
```

Out[54]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state_of_res
495	685994	F	True	21100.0	Married	False	Homeowner free and clear	False	2.0	63.0	Maryland
496	688580	M	True	48000.0	Married	True	Rented	True	2.0	22.0	Florida
497	688736	F	NaN	0.0	Married	True	Rented	False	2.0	34.0	Iowa
498	692445	M	True	140000.0	Married	True	Homeowner with mortgage/loan	False	5.0	48.0	Illinois
499	693235	M	True	36200.0	Married	True	Homeowner free and clear	False	2.0	43.0	Michigan

In [55]:
```
data.shape
```

Out[55]:
```
(500, 11)
```

In [56]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   id                500 non-null    int64  
 1   sex               500 non-null    object 
 2   is_employed       435 non-null    object 
 3   income            493 non-null    float64
 4   marital_status    500 non-null    object 
 5   health_insurance  500 non-null    bool   
 6   housing_type      466 non-null    object 
 7   recent_move       466 non-null    object 
 8   num_vehicles      466 non-null    float64
 9   age               500 non-null    float64
 10  state_of_res      500 non-null    object 
dtypes: bool(1), float64(3), int64(1), object(6)
memory usage: 39.7+ KB
```

In [57]:
```
#Descriptive statistics
data.describe()
```

Out[57]:

	id	income	num_vehicles	age
count	500.000000	493.000000	466.000000	500.000000
mean	339421.530000	53899.584178	1.845494	51.919626
std	203464.742214	64585.040596	1.031624	18.374129
min	2068.000000	0.000000	0.000000	18.000000
25%	153033.250000	15500.000000	1.000000	38.000000
50%	345339.500000	34300.000000	2.000000	49.000000
75%	513477.750000	70000.000000	2.000000	64.000000
max	693235.000000	412000.000000	6.000000	137.700030

In [58]:
```
#Missing values?
data.isnull()
```

Out[58]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state_of_res
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
495	False	False	False	False	False	False	False	False	False	False	False
496	False	False	False	False	False	False	False	False	False	False	False
497	False	False	True	False	False	False	False	False	False	False	False
498	False	False	False	False	False	False	False	False	False	False	False
499	False	False	False	False	False	False	False	False	False	False	False

500 rows × 11 columns

In [59]:
```
#Missing values?
np.sum(data.isnull())
```

Out[59]:
```
id                   0
sex                  0
is_employed         65
income               7
marital_status       0
health_insurance     0
housing_type        34
recent_move         34
num_vehicles        34
age                  0
state_of_res         0
dtype: int64
```

In [60]:
```
#Id
data['id'].unique()
```

Out[60]:
```
array([  2068,   2073,   2848,   5641,   6369,   8322,   8521,  12195,
        14989,  15917,  16551,  17134,  17946,  18487,  20383,  22295,
        25863,  26057,  26355,  27214,  27445,  27761,  27928,  30768,
        31710,  33651,  33828,  36825,  37224,  38827,  40132,  40449,
        42374,  46099,  46791,  47009,  48479,  49496,  50220,  51723,
        52197,  52382,  52420,  52436,  53186,  53214,  53759,  54177,
        55873,  55992,  56040,  62999,  65004,  66778,  67776,  68013,
        68221,  69062,  72741,  74447,  76182,  77312,  78476,  79069,
        80274,  80549,  82503,  84636,  84879,  85398,  86711,  86786,
        90303,  90863,  92706,  94743,  96964,  97247,  98086,  99068,
       100475, 102269, 103389, 104048, 104506, 106395, 106726, 106956,
       107458, 108042, 110024, 110699, 111870, 112116, 113390, 114806,
       115100, 116171, 116481, 117491, 117900, 117911, 118561, 120705,
       122231, 124968, 125155, 126507, 127965, 131301, 133268, 136206,
       138865, 139771, 141492, 145172, 146457, 147984, 149429, 149480,
       150055, 150721, 151170, 151678, 151876, 153419, 155762, 156520,
       158530, 159107, 159573, 164576, 166246, 167154, 167498, 169519,
       173493, 174248, 175033, 175572, 177122, 177145, 178178, 178887,
       179481, 180009, 181385, 181754, 182072, 182146, 183889, 184170,
       184506, 184686, 186475, 186808, 191018, 191536, 194740, 195539,
       196613, 196828, 199788, 203816, 204591, 206122, 209604, 210241,
       210427, 211330, 211424, 213854, 214083, 215876, 217315, 218283,
       220135, 220142, 220205, 223271, 223773, 224356, 226840, 227318,
       228513, 236067, 236880, 237048, 249492, 250108, 251365, 252100,
       253015, 254969, 258722, 258948, 259807, 262234, 264292, 265299,
       268667, 269836, 272748, 274182, 276306, 277784, 278995, 279538,
       281529, 283521, 283607, 284973, 287882, 291564, 291599, 291608,
       297117, 298205, 298782, 303624, 304200, 307655, 308107, 310588,
       311095, 314099, 315265, 316205, 316280, 316753, 317166, 318105,
       318348, 319774, 323508, 324029, 330543, 331868, 332292, 333355,
       333982, 334135, 335080, 335174, 337683, 338635, 341507, 342460,
       343235, 344685, 345994, 348319, 349344, 351129, 352275, 354269,
       356073, 356688, 356763, 359997, 360674, 361561, 361946, 361968,
       363008, 364328, 364991, 368645, 373976, 374461, 374930, 376508,
       382338, 382614, 382908, 383409, 383455, 385318, 388567, 390273,
       394696, 394790, 395888, 396116, 396256, 396675, 397230, 399150,
       399930, 403690, 406984, 408221, 409728, 411385, 415060, 415381,
       415575, 415741, 416144, 416624, 419751, 419951, 421248, 421666,
       423279, 423910, 426154, 427431, 427589, 430516, 431729, 432526,
       433755, 435651, 438304, 439284, 440475, 441221, 442608, 445622,
       445937, 445985, 449907, 450221, 450776, 456859, 458094, 458212,
       458265, 459429, 459910, 459959, 460642, 460925, 461021, 461429,
       462569, 463808, 464082, 465818, 467335, 468344, 468553, 474507,
       478228, 478502, 479327, 482611, 487124, 487369, 487556, 489718,
       491280, 491389, 491848, 492072, 492789, 496465, 497970, 498048,
       499492, 499637, 499807, 501259, 502705, 502764, 503412, 504889,
       505318, 506042, 509386, 510349, 510555, 511732, 512594, 516129,
       517253, 517696, 518076, 518899, 519446, 520030, 520390, 523769,
       523990, 525942, 527928, 528994, 532940, 532971, 535114, 536463,
       536879, 537484, 540956, 542384, 543309, 545826, 546208, 547258,
       548303, 549068, 550733, 551584, 556935, 559163, 559955, 560007,
       560783, 561588, 562962, 565628, 565740, 568892, 569414, 571834,
       572341, 573573, 575660, 576799, 578596, 580304, 581657, 583093,
       583249, 583934, 584570, 588705, 588711, 589826, 591244, 592496,
       595225, 595796, 595944, 596523, 597817, 598008, 601247, 601778,
       603304, 604216, 604839, 605624, 606108, 606307, 607766, 608789,
       609449, 612775, 612779, 613391, 616498, 618036, 618059, 618500,
       620116, 620452, 622713, 624593, 628305, 634710, 635515, 635574,
       638458, 638760, 642727, 643754, 645268, 646061, 646808, 647206,
       648220, 649827, 650803, 650961, 655940, 661353, 665669, 667156,
       668199, 668750, 668777, 673010, 674271, 679084, 679364, 680475,
       680503, 682848, 683320, 683867, 684640, 684701, 685624, 685994,
       688580, 688736, 692445, 693235], dtype=int64)
```

In [61]:
```
data['id'].nunique()
```

Out[61]:
```
500
```

In [62]:
```
#Sex
data['sex'].value_counts()
```

Out[62]:
```
M    274
F    226
Name: sex, dtype: int64
```

In [63]:
```
#Missing values?
np.sum(data['sex'].isnull())
```

Out[63]:
```
0
```

In [64]:
```
#Extract  'sex', 'age', and 'income' columns
#         for female customers
data.loc[data['sex'] == 'F', ['sex', 'income', 'age']]
```

Out[64]:

	sex	income	age
0	F	11300.0	49.0
1	F	0.0	40.0
4	F	42000.0	31.0
5	F	NaN	40.0
9	F	24000.0	70.0
...	...	...	...
486	F	22800.0	55.0
489	F	59000.0	58.0
490	F	70200.0	32.0
495	F	21100.0	63.0
497	F	0.0	34.0

226 rows × 3 columns

In [65]:
```
#What percentage of our customers are female?
np.sum(data['sex'] == 'F') / data.shape[0] * 100
```

Out[65]:
```
45.2
```

In [66]:
```
#is.employed
data['is_employed'].value_counts()
```

Out[66]:
```
True     331
False    104
Name: is_employed, dtype: int64
```

In [67]:
```
#Missing values?
np.sum(data['is_employed'].isnull())
```

Out[67]:
```
65
```

In [68]:
```
#Percentage of missing values in is_employed?
np.sum(data['is_employed'].isna()) / data.shape[0] * 100
```

Out[68]:
```
13.0
```

In [69]:
```
#What percentage of customers are employed?
sum(data['is_employed'] == True) / sum(data['is_employed'].notnull()) * 100
```

Out[69]:
```
76.0919540229885
```

In [70]:
```
#Income
data['income'].describe()
```

Out[70]:
```
count       493.000000
mean      53899.584178
std       64585.040596
min           0.000000
25%       15500.000000
50%       34300.000000
75%       70000.000000
max      412000.000000
Name: income, dtype: float64
```

In [71]:
```
np.sum(data['income'].isnull())
```

Out[71]:
```
7
```

In [72]:
```
#Zero?
np.sum(data['income'] == 0) / data.shape[0] * 100
```

Out[72]:
```
9.4
```

In [73]:
```
#Age
data['age'].describe()
```

Out[73]:
```
count    500.000000
mean      51.919626
std       18.374129
min       18.000000
25%       38.000000
50%       49.000000
75%       64.000000
max      137.700030
Name: age, dtype: float64
```

In [74]:
```
#How many people are above 100 years old!?
np.sum(data['age'] > 100)
```

Out[74]:
```
3
```

In [75]:
```
#Who are they?
data.loc[data['age'] > 100, ]
```

Out[75]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state
212	287882	M	True	60000.0	Married	True	Rented	True	2.0	137.700030	
266	364991	F	NaN	12000.0	Divorced/Separated	True	Homeowner with mortgage/loan	False	1.0	123.061023	
286	397230	F	True	31200.0	Married	True	Rented	False	2.0	136.052160	

In [76]:
```
#Remove people with age above 100
data = data[data['age'] < 100]
```

In [77]:
```
data.shape
```

Out[77]:
```
(497, 11)
```

In [78]:
```
data['age'].describe()
```

Out[78]:
```
count    497.000000
mean      51.434608
std       17.323542
min       18.000000
25%       38.000000
50%       49.000000
75%       63.000000
max       93.000000
Name: age, dtype: float64
```

In [79]:
```
#Extract  'sex', 'age', and 'income' columns
#         for male customers who are above 50 years old
data.loc[(data['age'] > 50) & (data['sex'] == 'M'),
         ['sex', 'age', 'income']]
```

Out[79]:

	sex	age	income
15	M	54.0	34400.0
16	M	70.0	41000.0
19	M	68.0	18800.0
22	M	58.0	75000.0
30	M	66.0	22000.0
...	...	...	...
484	M	54.0	0.0
485	M	88.0	29200.0
491	M	62.0	82000.0
492	M	70.0	18200.0
493	M	63.0	12600.0

126 rows × 3 columns

In [80]:
```
#What percentage of customers are between 25 and 35?
round(np.sum((data['age'] > 25) &
             (data['age'] < 35)) / data.shape[0] * 100, 2)
```

Out[80]: 14.69

In [81]:
```
#What percentage of male customers are above 30?
round(np.sum((data['sex'] == 'M') &
             (data['age'] > 30)) / np.sum(data['sex'] == 'M') * 100, 2)
```

Out[81]: 89.38

In [82]:
```
#Extract those customers from Florida
#        who are older than 75% of all customers
data.loc[(data['state_of_res'] == 'Florida') &
         (data['age'] > np.quantile(data['age'], 0.75)), :]
```

Out[82]:

	id	sex	is_employed	income	marital_status	health_insurance	housing_type	recent_move	num_vehicles	age	state_of_res
40	52197	M	False	65100.0	Married	True	Homeowner free and clear	False	2.0	66.0	Florida
78	98086	M	True	52100.0	Married	True	Homeowner with mortgage/loan	True	2.0	69.0	Florida
88	107458	M	True	182500.0	Married	True	Homeowner with mortgage/loan	True	2.0	66.0	Florida
159	195539	M	False	65700.0	Married	True	Rented	False	2.0	69.0	Florida
161	196828	M	True	24800.0	Married	True	Homeowner with mortgage/loan	False	3.0	65.0	Florida
244	337683	F	True	23100.0	Widowed	True	Rented	False	1.0	65.0	Florida
354	491848	F	NaN	16700.0	Widowed	True	Rented	False	1.0	84.0	Florida
361	499637	F	NaN	12000.0	Never Married	True	NaN	NaN	NaN	93.0	Florida

In [83]:
```
#3rd Quantile of age
np.quantile(data['age'], 0.75)
```

Out[83]: 63.0

Assignment

Q1: Create a dataframe which contains information of a company's products:

prod_name	price	monthly_demand	brand
g1	45	1200	z
g2	25	2500	x
c1	75	200	x
c2	35	1850	y
c3	89	120	z
f1	20	3500	y
f2	55	545	z

a: Add a column to calculate monthly revenue of each product

b: Calculate total monthly revenue generated by brand z

c: Create a column called revenue share which includes monthly revenue share of each product

d: Which brand does have the largest revenue share?

Q2: Consider CRM dataset and answer these questions:

a: Extract data of married customers with income above average.

b: What percentag of female customers are below 26?

c: What percentage of female customers above 30 are employed?

d: What percentage of cutomers who have health insurance are employed with income above average income?

e: What percentage of cutomers who have health insurance are married males?

f: What percentage of married customers have health insurance?

g: Identify our top customers? (income > average , homeowner, and own at least one vehicle)

End of Code