

Elements Of Data Science - F2023

Week 3: Pandas, Data Exploration and Visualization

9/25/2023

TODOs

- Practical Statistics for Data Scientists, Chapter 3 [EBSCO](#)
- An introduction to seaborn <https://seaborn.pydata.org/tutorial/introduction.html>
- (Optional) Data Science From Scratch, Chapter 5,6,7 [EBSCO](#)
- Complete Week 3 Quiz
- HW1 out this week, includes questions on Hypothesis Testing

TODAY

- Pandas
- Data Exploration
- Visualization

Questions?

Environment Setup

Environment Setup

```
In [1]: 1 import numpy as np
```

Intro to Pandas

Intro to Pandas



Pandas is an open source, BSD-licensed library providing:

- **high-performance, easy-to-use data structures and**
- **data analysis tools**

Intro to Pandas



Pandas is an open source, BSD-licensed library providing:

- high-performance, easy-to-use data structures and
- data analysis tools

```
In [2]: 1 # usually imported using the alias 'pd'
        2 import pandas as pd
```


Intro to Pandas



Pandas is an open source, BSD-licensed library providing:

- high-performance, easy-to-use data structures and
- data analysis tools

```
In [2]: 1 # usually imported using the alias 'pd'
        2 import pandas as pd
```

- Primary datastructures:
 - **Series:** 1D array with a flexible index
 - **Dataframe:** 2D matrix with flexible index and column names

Pandas Series

Pandas Series

- 1D array of data (any numpy datatype) plus an associated **index** array

Pandas Series

- 1D array of data (any numpy datatype) plus an associated **index** array

```
In [3]: 1 s = pd.Series(np.random.rand(4))  
        2 s
```

```
Out[3]: 0    0.826651  
        1    0.336333  
        2    0.211941  
        3    0.481098  
        dtype: float64
```

Pandas Series

- 1D array of data (any numpy datatype) plus an associated **index** array

```
In [3]: 1 s = pd.Series(np.random.rand(4))  
        2 s
```

```
Out[3]: 0    0.826651  
        1    0.336333  
        2    0.211941  
        3    0.481098  
        dtype: float64
```

```
In [4]: 1 # return the values of the series  
        2 s.values
```

```
Out[4]: array([0.8266513 , 0.3363326 , 0.21194121, 0.48109781])
```

Pandas Series

- 1D array of data (any numpy datatype) plus an associated **index** array

```
In [3]: 1 s = pd.Series(np.random.rand(4))  
        2 s
```

```
Out[3]: 0    0.826651  
        1    0.336333  
        2    0.211941  
        3    0.481098  
        dtype: float64
```

```
In [4]: 1 # return the values of the series  
        2 s.values
```

```
Out[4]: array([0.8266513 , 0.3363326 , 0.21194121, 0.48109781])
```

```
In [5]: 1 # return the index of the series  
        2 s.index
```

```
Out[5]: RangeIndex(start=0, stop=4, step=1)
```

Pandas Series Cont.

Pandas Series Cont.

- index is flexible, can be anything hashable (integers, strings, ...)

Pandas Series Cont.

- index is flexible, can be anything hashable (integers, strings, ...)

```
In [6]: 1 # create Series from array and set index
        2 s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
        3 s1
```

```
Out[6]: house_a    1.0
        2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

Pandas Series Cont.

- index is flexible, can be anything hashable (integers, strings, ...)

```
In [6]: 1 # create Series from array and set index
        2 s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
        3 s1
```

```
Out[6]: house_a    1.0
        2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

```
In [7]: 1 s1['house_a'] # access a single value via index label
```

```
Out[7]: 1.0
```

Pandas Series Cont.

- index is flexible, can be anything hashable (integers, strings, ...)

```
In [6]: 1 # create Series from array and set index
        2 s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
        3 s1
```

```
Out[6]: house_a    1.0
        2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

```
In [7]: 1 s1['house_a'] # access a single value via index label
```

```
Out[7]: 1.0
```

```
In [8]: 1 s1[[2,'house c']] # access multiple values via index label
```

```
Out[8]: 2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

Pandas Series Cont.

- index is flexible, can be anything hashable (integers, strings, ...)

```
In [6]: 1 # create Series from array and set index
        2 s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
        3 s1
```

```
Out[6]: house_a    1.0
        2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

```
In [7]: 1 s1['house_a'] # access a single value via index label
```

```
Out[7]: 1.0
```

```
In [8]: 1 s1[[2,'house c']] # access multiple values via index label
```

```
Out[8]: 2          2.0
        house c    3.0
        Name: NumRooms, dtype: float64
```

```
In [9]: 1 s1.house_a # dot notation (How do we get "house c"?)
```

```
Out[9]: 1.0
```

Pandas Series Cont.

Pandas Series Cont.

- accessing other Series attributes

```
In [10]: 1 s1
```

```
Out[10]: house_a    1.0  
         2         2.0  
         house c    3.0  
         Name: NumRooms, dtype: float64
```

Pandas Series Cont.

- accessing other Series attributes

```
In [10]: 1 s1
```

```
Out[10]: house_a    1.0  
         2          2.0  
         house c    3.0  
         Name: NumRooms, dtype: float64
```

```
In [11]: 1 #print(f'{s.index  = :}')  
         2 #print(f'{s.values = :}')  
         3 print(f'{s1.name   = :}')  
         4 print(f'{s1.dtype  = :}')  
         5 print(f'{s1.shape  = :}')
```

```
s1.name   = NumRooms  
s1.dtype  = float64  
s1.shape  = (3,)
```

Pandas Series Cont.

Pandas Series Cont.

```
In [12]: 1 # Can create series with index from a dictionary
          2 s2 = pd.Series({'a':1, 'b':2, 'c':3, 'd':4})
          3 s2
```

```
Out[12]: a    1
          b    2
          c    3
          d    4
          dtype: int64
```

Pandas Series Cont.

```
In [12]: 1 # Can create series with index from a dictionary
          2 s2 = pd.Series({'a':1, 'b':2, 'c':3, 'd':4})
          3 s2
```

```
Out[12]: a    1
          b    2
          c    3
          d    4
          dtype: int64
```

```
In [13]: 1 print(f'{s2.index = :}')
          2 print(f'{s2.values = :}')
```

```
s2.index = Index(['a', 'b', 'c', 'd'], dtype='object')
s2.values = [1 2 3 4]
```

Pandas DataFrame

Pandas DataFrame

- tabular datastructure
- each column a single datatype
- contains both row and column indices
- single column == Series

Pandas DataFrame Cont.

Pandas DataFrame Cont.

```
In [14]: 1 df = pd.DataFrame({'Year':[2017,2018,2018,2019],  
2                               'Semester':['Fall','Fall','Spring','Fall'],  
3                               'Measure_1':[2.1,3.0,2.4,1.9]  
4                               })
```

Pandas DataFrame Cont.

```
In [14]: 1 df = pd.DataFrame({'Year':[2017,2018,2018,2019],  
2                               'Semester':['Fall','Fall','Spring','Fall'],  
3                               'Measure_1':[2.1,3.0,2.4,1.9]  
4                               })
```

```
In [15]: 1 df
```

Out[15]:

	Year	Semester	Measure_1
0	2017	Fall	2.1
1	2018	Fall	3.0
2	2018	Spring	2.4
3	2019	Fall	1.9

Pandas DataFrame Cont.

```
In [14]: 1 df = pd.DataFrame({'Year':[2017,2018,2018,2019],  
2                               'Semester':['Fall','Fall','Spring','Fall'],  
3                               'Measure_1':[2.1,3.0,2.4,1.9]  
4                               })
```

```
In [15]: 1 df
```

Out[15]:

	Year	Semester	Measure_1
0	2017	Fall	2.1
1	2018	Fall	3.0
2	2018	Spring	2.4
3	2019	Fall	1.9

```
In [16]: 1 print(df)
```

```
   Year Semester  Measure_1  
0  2017      Fall         2.1  
1  2018      Fall         3.0  
2  2018    Spring         2.4  
3  2019      Fall         1.9
```


Pandas DataFrame Cont.

```
In [14]: 1 df = pd.DataFrame({'Year':[2017,2018,2018,2019],
2                               'Semester':['Fall','Fall','Spring','Fall'],
3                               'Measure_1':[2.1,3.0,2.4,1.9]
4                               })
```

```
In [15]: 1 df
```

Out[15]:

	Year	Semester	Measure_1
0	2017	Fall	2.1
1	2018	Fall	3.0
2	2018	Spring	2.4
3	2019	Fall	1.9

```
In [16]: 1 print(df)
```

```
   Year Semester  Measure_1
0  2017      Fall         2.1
1  2018      Fall         3.0
2  2018   Spring         2.4
3  2019      Fall         1.9
```

```
In [17]: 1 display(df)
```

	Year	Semester	Measure_1
0	2017	Fall	2.1
1	2018	Fall	3.0
2	2018	Spring	2.4
3	2019	Fall	1.9

Pandas DataFrame Cont.

Pandas DataFrame Cont.

```
In [18]: 1 data = [[2017, 'Fall', 2.1],  
2             [2018, 'Fall', 3.0],  
3             [2018, 'Spring', 2.4],  
4             [2019, 'Fall', 1.9]]
```

Pandas DataFrame Cont.

```
In [18]: 1 data = [[2017, 'Fall', 2.1],  
2             [2018, 'Fall', 3.0],  
3             [2018, 'Spring', 2.4],  
4             [2019, 'Fall', 1.9]]
```

```
In [19]: 1 df = pd.DataFrame(data,  
2             columns=['Year', 'Semester', 'Measure_1'],  
3             index=['001', '002', '003', '004'])  
4 df.shape
```

```
Out[19]: (4, 3)
```

Pandas DataFrame Cont.

```
In [18]: 1 data = [[2017, 'Fall', 2.1],  
2             [2018, 'Fall', 3.0],  
3             [2018, 'Spring', 2.4],  
4             [2019, 'Fall', 1.9]]
```

```
In [19]: 1 df = pd.DataFrame(data,  
2             columns=['Year', 'Semester', 'Measure_1'],  
3             index=['001', '002', '003', '004'])  
4 df.shape
```

Out[19]: (4, 3)

```
In [20]: 1 df
```

Out[20]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Attributes

Pandas Attributes

- Get shape of DataFrame : `shape`

Pandas Attributes

- Get shape of DataFrame : `shape`

```
In [21]: 1 df.shape # rows, columns
```

```
Out[21]: (4, 3)
```


Pandas Attributes

- Get shape of DataFrame : `shape`

```
In [21]: 1 df.shape # rows, columns
```

```
Out[21]: (4, 3)
```

- Get index values : `index`

Pandas Attributes

- Get shape of DataFrame : `shape`

```
In [21]: 1 df.shape # rows, columns
```

```
Out[21]: (4, 3)
```

- Get index values : `index`

```
In [22]: 1 df.index
```

```
Out[22]: Index(['001', '002', '003', '004'], dtype='object')
```

Pandas Attributes

- Get shape of DataFrame : `shape`

```
In [21]: 1 df.shape # rows, columns
```

```
Out[21]: (4, 3)
```

- Get index values : `index`

```
In [22]: 1 df.index
```

```
Out[22]: Index(['001', '002', '003', '004'], dtype='object')
```

- Get column values : `columns`

Pandas Attributes

- Get shape of DataFrame : `shape`

```
In [21]: 1 df.shape # rows, columns
```

```
Out[21]: (4, 3)
```

- Get index values : `index`

```
In [22]: 1 df.index
```

```
Out[22]: Index(['001', '002', '003', '004'], dtype='object')
```

- Get column values : `columns`

```
In [23]: 1 df.columns
```

```
Out[23]: Index(['Year', 'Semester', 'Measure_1'], dtype='object')
```

Pandas Indexing/Selection

Pandas Indexing/Selection

Select by label:

- `.loc[]`

Pandas Indexing/Selection

Select by label:

- `.loc[]`

```
In [24]: 1 df.loc['001']
```

```
Out[24]: Year      2017  
         Semester  Fall  
         Measure_1  2.1  
         Name: 001, dtype: object
```

Pandas Indexing/Selection

Select by label:

- `.loc[]`

```
In [24]: 1 df.loc['001']
```

```
Out[24]: Year      2017  
Semester    Fall  
Measure_1    2.1  
Name: 001, dtype: object
```

```
In [25]: 1 df.loc['001', 'Measure_1']
```

```
Out[25]: 2.1
```


Pandas Indexing/Selection Cont.

Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

```
In [26]: 1 df.iloc[0]
```

```
Out[26]: Year      2017  
         Semester  Fall  
         Measure_1    2.1  
         Name: 001, dtype: object
```

Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

```
In [26]: 1 df.iloc[0]
```

```
Out[26]: Year      2017  
         Semester  Fall  
         Measure_1  2.1  
         Name: 001, dtype: object
```

```
In [27]: 1 df.iloc[0,2]
```

```
Out[27]: 2.1
```

Pandas Indexing/Selection Cont.

Pandas Indexing/Selection Cont.

Selecting multiple rows/columns: use list (fancy indexing)

Pandas Indexing/Selection Cont.

Selecting multiple rows/columns: use list (fancy indexing)

```
In [28]: 1 df.loc[['002', '004']]
```

Out[28]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
004	2019	Fall	1.9

Pandas Indexing/Selection Cont.

Selecting multiple rows/columns: use list (fancy indexing)

```
In [28]: 1 df.loc[['002', '004']]
```

Out[28]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
004	2019	Fall	1.9

```
In [29]: 1 df.loc[['002', '004'], ['Year', 'Measure_1']]
```

Out[29]:

	Year	Measure_1
002	2018	3.0
004	2019	1.9

Pandas Slicing

Pandas Slicing

```
In [30]: 1 # Get last two rows
          2 df.iloc[-2:]
```

Out[30]:

	Year	Semester	Measure_1
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Slicing

```
In [30]: 1 # Get last two rows
        2 df.iloc[-2:]
```

Out[30]:

	Year	Semester	Measure_1
003	2018	Spring	2.4
004	2019	Fall	1.9

```
In [31]: 1 # Get first two rows and first two columns
        2 df.iloc[:2,:2]
```

Out[31]:

	Year	Semester
001	2017	Fall
002	2018	Fall

Pandas Slicing

```
In [30]: 1 # Get last two rows
          2 df.iloc[-2:]
```

Out[30]:

	Year	Semester	Measure_1
003	2018	Spring	2.4
004	2019	Fall	1.9

```
In [31]: 1 # Get first two rows and first two columns
          2 df.iloc[:2,:2]
```

Out[31]:

	Year	Semester
001	2017	Fall
002	2018	Fall

NOTE: `.iloc` is exclusive (start:end+1)

Pandas Slicing Cont.

Pandas Slicing Cont.

Can also slice using labels:

Pandas Slicing Cont.

Can also slice using labels:

```
In [32]: 1 df.loc['002':'004']
```

Out[32]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Slicing Cont.

Can also slice using labels:

```
In [32]: 1 df.loc['002':'004']
```

Out[32]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

```
In [33]: 1 df.loc['002':'004',: 'Measure_1']
```

Out[33]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Slicing Cont.

Can also slice using labels:

```
In [32]: 1 df.loc['002':'004']
```

Out[32]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

```
In [33]: 1 df.loc['002':'004',: 'Measure_1']
```

Out[33]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

NOTE: `.loc` is inclusive

Pandas Slicing Cont.

Pandas Slicing Cont.

How to indicate all rows or all columns? :

Pandas Slicing Cont.

How to indicate all rows or all columns? :

```
In [34]: 1 df.loc[:, 'Measure_1']
```

```
Out[34]: 001    2.1  
         002    3.0  
         003    2.4  
         004    1.9  
         Name: Measure_1, dtype: float64
```

Pandas Slicing Cont.

How to indicate all rows or all columns? :

```
In [34]: 1 df.loc[:, 'Measure_1']
```

```
Out[34]: 001    2.1  
         002    3.0  
         003    2.4  
         004    1.9  
         Name: Measure_1, dtype: float64
```

```
In [35]: 1 df.iloc[2:,:]
```

```
Out[35]:
```

	Year	Semester	Measure_1
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Indexing Cont.

Pandas Indexing Cont.

Shortcut for indexing:

Pandas Indexing Cont.

Shortcut for indexing:

```
In [36]: 1 df['Semester']
```

```
Out[36]: 001      Fall
          002      Fall
          003    Spring
          004      Fall
          Name: Semester, dtype: object
```


Pandas Indexing Cont.

Shortcut for indexing:

```
In [36]: 1 df['Semester']
```

```
Out[36]: 001      Fall
         002      Fall
         003    Spring
         004      Fall
         Name: Semester, dtype: object
```

```
In [37]: 1 # can use dot notation if there is no space in label
         2 df.Semester
```

```
Out[37]: 001      Fall
         002      Fall
         003    Spring
         004      Fall
         Name: Semester, dtype: object
```

Panda Selection Chaining

Panda Selection Chaining

Get 'Year' and 'Measure_1' for first 3 rows:

Panda Selection Chaining

Get 'Year' and 'Measure_1' for first 3 rows:

```
In [38]: 1 df.iloc[:3].loc[:,['Year', 'Measure_1']]
```

Out[38]:

	Year	Measure_1
001	2017	2.1
002	2018	3.0
003	2018	2.4

Panda Selection Chaining

Get 'Year' and 'Measure_1' for first 3 rows:

```
In [38]: 1 df.iloc[:3].loc[:, ['Year', 'Measure_1']]
```

Out[38]:

	Year	Measure_1
001	2017	2.1
002	2018	3.0
003	2018	2.4

For records '001' and '003' get last two columns

Panda Selection Chaining

Get 'Year' and 'Measure_1' for first 3 rows:

```
In [38]: 1 df.iloc[:3].loc[:,['Year','Measure_1']]
```

Out[38]:

	Year	Measure_1
001	2017	2.1
002	2018	3.0
003	2018	2.4

For records '001' and '003' get last two columns

```
In [39]: 1 df.loc[['001','003']].iloc[:, -2:]
```

Out[39]:

	Semester	Measure_1
001	Fall	2.1
003	Spring	2.4

Pandas Selection Chaining Cont.

Pandas Selection Chaining Cont.

For record '002' get last two columns?:

Pandas Selection Chaining Cont.

For record '002' get last two columns?:

```
In [40]: 1 # reduce the amount of error information printed  
        2 %xmode Minimal
```

Exception reporting mode: Minimal

Pandas Selection Chaining Cont.

For record '002' get last two columns?:

```
In [40]: 1 # reduce the amount of error information printed
        2 %xmode Minimal
```

Exception reporting mode: Minimal

```
In [41]: 1 # Note: add 'raises-exception' tag to cell to continue running after exception
        2
        3 df.loc['002'].iloc[:, -2:] # row with label '002', then all rows, last two columns?
```

IndexingError: Too many indexers

Pandas Selection Chaining Cont.

For record '002' get last two columns?:

```
In [40]: 1 # reduce the amount of error information printed
        2 %xmode Minimal
```

Exception reporting mode: Minimal

```
In [41]: 1 # Note: add 'raises-exception' tag to cell to continue running after exception
        2
        3 df.loc['002'].iloc[:, -2:] # row with label '002', then all rows, last two columns?
```

IndexingError: Too many indexers

```
In [42]: 1 df.loc['002']
```

```
Out[42]: Year          2018
         Semester      Fall
         Measure_1      3.0
         Name: 002, dtype: object
```

Pandas Selection Chaining Cont.

For record '002' get last two columns?:

```
In [40]: 1 # reduce the amount of error information printed
         2 %xmode Minimal
```

Exception reporting mode: Minimal

```
In [41]: 1 # Note: add 'raises-exception' tag to cell to continue running after exception
         2
         3 df.loc['002'].iloc[:, -2:] # row with label '002', then all rows, last two columns?
```

IndexingError: Too many indexers

```
In [42]: 1 df.loc['002']
```

```
Out[42]: Year          2018
         Semester      Fall
         Measure_1      3.0
         Name: 002, dtype: object
```

```
In [43]: 1 df.loc['002'].iloc[-2:] # row with label '002', last two elements of Series
```

```
Out[43]: Semester      Fall
         Measure_1      3.0
         Name: 002, dtype: object
```

Pandas **head** and **tail**

Pandas `head` and `tail`

Get a quick view of the first or last rows in a DataFrame

Pandas `head` and `tail`

Get a quick view of the first or last rows in a DataFrame

```
In [44]: 1 df.head() # first 5 rows by default
```

Out[44]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas **head** and **tail**

Get a quick view of the first or last rows in a DataFrame

```
In [44]: 1 df.head() # first 5 rows by default
```

Out[44]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0
003	2018	Spring	2.4
004	2019	Fall	1.9

```
In [45]: 1 df.tail(2) # only print last 2 rows
```

Out[45]:

	Year	Semester	Measure_1
003	2018	Spring	2.4
004	2019	Fall	1.9

Pandas Boolean Mask

Pandas Boolean Mask

```
In [46]: 1 # Which rows have Semester of 'Fall'?  
        2 df.loc[:, 'Semester'] == 'Fall'
```

```
Out[46]: 001      True  
        002      True  
        003     False  
        004      True  
        Name: Semester, dtype: bool
```

Pandas Boolean Mask

```
In [46]: 1 # Which rows have Semester of 'Fall'?  
2 df.loc[:, 'Semester'] == 'Fall'
```

```
Out[46]: 001      True  
002      True  
003     False  
004      True  
Name: Semester, dtype: bool
```

```
In [47]: 1 # Get all data for rows with with Semester 'Fall'  
2 df.loc[df.Semester == 'Fall']
```

```
Out[47]:
```

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0
004	2019	Fall	1.9

Pandas Boolean Mask

```
In [46]: 1 # Which rows have Semester of 'Fall'?  
2 df.loc[:, 'Semester'] == 'Fall'
```

```
Out[46]: 001      True  
002      True  
003     False  
004      True  
Name: Semester, dtype: bool
```

```
In [47]: 1 # Get all data for rows with with Semester 'Fall'  
2 df.loc[df.Semester == 'Fall']
```

```
Out[47]:
```

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0
004	2019	Fall	1.9

```
In [48]: 1 # Get Measure_1 for all records for Semester 'Fall'  
2 df.loc[df.Semester == 'Fall', 'Measure_1']
```

```
Out[48]: 001      2.1  
002      3.0  
004      1.9  
Name: Measure_1, dtype: float64
```

Pandas Boolean Mask Cont.

Pandas Boolean Mask Cont.

Get all records Fall Semester prior to 2019

Pandas Boolean Mask Cont.

Get all records Fall Semester prior to 2019

```
In [49]: 1 # make sure to use parentheses with comparisons!
          2 df.loc[(df.Semester == 'Fall') & (df.Year < 2019)]
```

Out[49]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0

Pandas Boolean Mask Cont.

Get all records Fall Semester prior to 2019

```
In [49]: 1 # make sure to use parentheses with comparisons!
          2 df.loc[(df.Semester == 'Fall') & (df.Year < 2019)]
```

Out[49]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0

```
In [50]: 1 # or use comparison functions: .eq, .ne, .gt, .ge, .lt, .le
          2 df.loc[df.Semester.eq('Fall') & df.Year.lt(2019)]
```

Out[50]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
002	2018	Fall	3.0

Pandas Boolean Mask Cont.

Pandas Boolean Mask Cont.

Get all records belonging to a set with `.isin:`

Pandas Boolean Mask Cont.

Get all records belonging to a set with `.isin:`

```
In [51]: 1 df.loc[df.Year.isin([2017,2019])]
```

Out[51]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
004	2019	Fall	1.9

Pandas Selection Review

Pandas Selection Review

- `.loc[]`
- `.iloc[]`
- Fancy Indexing
- Slicing
- Chaining
- `head` and `tail`
- Boolean Mask
- `.isin`

Pandas Sorting

Pandas Sorting

```
In [52]: 1 df.sort_values(by=[ 'Measure_1' ]).head(3)
```

Out[52]:

	Year	Semester	Measure_1
004	2019	Fall	1.9
001	2017	Fall	2.1
003	2018	Spring	2.4

Pandas Sorting

```
In [52]: 1 df.sort_values(by=[ 'Measure_1' ]).head(3)
```

Out[52]:

	Year	Semester	Measure_1
004	2019	Fall	1.9
001	2017	Fall	2.1
003	2018	Spring	2.4

```
In [53]: 1 df.sort_values(by=[ 'Measure_1' ],ascending=False).head(3)
```

Out[53]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
001	2017	Fall	2.1

Pandas Sorting

```
In [52]: 1 df.sort_values(by=[ 'Measure_1' ]).head(3)
```

Out[52]:

	Year	Semester	Measure_1
004	2019	Fall	1.9
001	2017	Fall	2.1
003	2018	Spring	2.4

```
In [53]: 1 df.sort_values(by=[ 'Measure_1' ],ascending=False).head(3)
```

Out[53]:

	Year	Semester	Measure_1
002	2018	Fall	3.0
003	2018	Spring	2.4
001	2017	Fall	2.1

```
In [54]: 1 df.sort_values(by=[ 'Year', 'Measure_1' ]).head(3)
```

Out[54]:

	Year	Semester	Measure_1
001	2017	Fall	2.1
003	2018	Spring	2.4
002	2018	Fall	3.0

Questions?

Exploratory Data Analysis

Exploratory Data Analysis

For a new set of data, would like to know:

- amount of data (rows, columns)
- range (min, max)
- counts of discrete values
- central tendencies (mean, median)
- dispersion or spread (variance, IQR)
- skew
- covariance and correlation ...

Yellowcab Dataset

- Records of Yellowcab Taxi trips from January 2017
- more info: <https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

Loading Datasets from CSV (Comma Separated Values)

- columns separated by delimiter, eg. comma, tab (\t), pipe (|)
- one row per record, observation
- often, strings quoted
- often, first row contains column headings
- often, comment rows starting with #

Loading Datasets from CSV (Comma Separated Values)

- columns separated by delimiter, eg. comma, tab (\t), pipe (|)
- one row per record, observation
- often, strings quoted
- often, first row contains column headings
- often, comment rows starting with #

```
In [55]: 1 !head ../data/yellowcab_demo_withdaycategories.csv
```

```
# A sample of yellocab taxi trip data from Jan 2017
pickup_datetime,dropoff_datetime,trip_distance,fare_amount,tip_amount,payment_type,day_of_week,is_weekend
2017-01-05 14:49:04,2017-01-05 14:53:53,0.89,5.5,1.26,Credit card,3,False
2017-01-15 01:07:22,2017-01-15 01:26:47,2.7,14.0,0.0,Cash,6,True
2017-01-29 09:55:00,2017-01-29 10:04:43,1.41,8.0,0.0,Cash,6,True
2017-01-10 05:40:12,2017-01-10 05:42:22,0.4,4.0,0.0,Cash,1,False
2017-01-06 17:02:48,2017-01-06 17:16:10,2.3,11.0,0.0,Cash,4,False
2017-01-14 19:03:14,2017-01-14 19:08:41,0.8,5.5,,Credit card,5,False
2017-01-06 18:51:52,2017-01-06 18:55:45,0.2,4.5,0.0,Cash,4,False
2017-01-04 20:47:30,2017-01-04 21:01:24,2.68,11.5,,Credit card,2,False
```

Loading Datasets with Pandas

Loading Datasets with Pandas

```
In [56]: 1 import pandas as pd
          2 df_taxi = (
          3     pd.read_csv('../data/yellowcab_demo_withdaycategories.csv',
          4                   sep=',',
          5                   header=1,
          6                   parse_dates=['pickup_datetime', 'dropoff_datetime'],
          7                   )
          8 )
```

Loading Datasets with Pandas

```
In [56]: 1 import pandas as pd
          2 df_taxi = (
          3     pd.read_csv('../data/yellowcab_demo_withdaycategories.csv',
          4                   sep=',',
          5                   header=1,
          6                   parse_dates=['pickup_datetime', 'dropoff_datetime'],
          7                   )
          8 )
```

```
In [57]: 1 # display first 5 rows
          2 df_taxi.head(5)
```

Out[57]:

	pickup_datetime	dropoff_datetime	trip_distance	fare_amount	tip_amount	payment_type	day_of_week	is_weekend
0	2017-01-05 14:49:04	2017-01-05 14:53:53	0.89	5.5	1.26	Credit card	3	False
1	2017-01-15 01:07:22	2017-01-15 01:26:47	2.70	14.0	0.00	Cash	6	True
2	2017-01-29 09:55:00	2017-01-29 10:04:43	1.41	8.0	0.00	Cash	6	True
3	2017-01-10 05:40:12	2017-01-10 05:42:22	0.40	4.0	0.00	Cash	1	False
4	2017-01-06 17:02:48	2017-01-06 17:16:10	2.30	11.0	0.00	Cash	4	False

Get Size of Dataset

Get Size of Dataset

```
In [58]: 1 df_taxi.shape
```

```
Out[58]: (1000, 8)
```

Get Size of Dataset

```
In [58]: 1 df_taxi.shape
```

```
Out[58]: (1000, 8)
```

```
In [59]: 1 # number of rows  
2 f'{df_taxi.shape[0]} rows'
```

```
Out[59]: '1000 rows'
```

Get Size of Dataset

```
In [58]: 1 df_taxi.shape
```

```
Out[58]: (1000, 8)
```

```
In [59]: 1 # number of rows  
2 f'{df_taxi.shape[0]} rows'
```

```
Out[59]: '1000 rows'
```

```
In [60]: 1 # number of columns  
2 f'{df_taxi.shape[1]} columns'
```

```
Out[60]: '8 columns'
```

Get Size of Dataset

```
In [58]: 1 df_taxi.shape
```

```
Out[58]: (1000, 8)
```

```
In [59]: 1 # number of rows  
2 f'{df_taxi.shape[0]} rows'
```

```
Out[59]: '1000 rows'
```

```
In [60]: 1 # number of columns  
2 f'{df_taxi.shape[1]} columns'
```

```
Out[60]: '8 columns'
```

```
In [61]: 1 'number of rows: {}, number of columns: {}'.format(*df_taxi.shape)
```

```
Out[61]: 'number of rows: 1000, number of columns: 8'
```

Aside: Argument Unpacking with *

Aside: Argument Unpacking with *

- * in when calling a function unpacks an iterable, passing each value as an argument
- want `format(2, 8)` instead of the `format((2, 8))`

Aside: Argument Unpacking with *

- * in when calling a function unpacks an iterable, passing each value as an argument
- want `format(2, 8)` instead of the `format((2, 8))`

```
In [62]: 1 print(*df_taxi.shape)
```

```
1000 8
```

Aside: Argument Unpacking with *

- * in when calling a function unpacks an iterable, passing each value as an argument
- want `format(2, 8)` instead of the `format((2, 8))`

```
In [62]: 1 print(*df_taxi.shape)
```

```
1000 8
```

```
In [63]: 1 # call .format( (2,8) )
2 'number of rows: {}, number of columns: {}'.format(df_taxi.shape)
```

```
IndexError: Replacement index 1 out of range for positional args tuple
```

Aside: Argument Unpacking with *

- * in when calling a function unpacks an iterable, passing each value as an argument
- want `format(2, 8)` instead of the `format((2, 8))`

```
In [62]: 1 print(*df_taxi.shape)
```

```
1000 8
```

```
In [63]: 1 # call .format( (2,8) )
2 'number of rows: {}, number of columns: {}'.format(df_taxi.shape)
```

```
IndexError: Replacement index 1 out of range for positional args tuple
```

```
In [64]: 1 # call .format(2,8)
2 'number of rows: {}, number of columns: {}'.format(*df_taxi.shape)
```

```
Out[64]: 'number of rows: 1000, number of columns: 8'
```

What are the column names?

What are the column names?

```
In [65]: 1 df_taxi.columns
```

```
Out[65]: Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',  
               'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],  
              dtype='object')
```

What are the column names?

```
In [65]: 1 df_taxi.columns
```

```
Out[65]: Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',  
              'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],  
              dtype='object')
```

```
In [66]: 1 df_taxi.columns.values
```

```
Out[66]: array(['pickup_datetime', 'dropoff_datetime', 'trip_distance',  
              'fare_amount', 'tip_amount', 'payment_type', 'day_of_week',  
              'is_weekend'], dtype=object)
```

What are the column names?

```
In [65]: 1 df_taxi.columns
```

```
Out[65]: Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',  
              'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],  
              dtype='object')
```

```
In [66]: 1 df_taxi.columns.values
```

```
Out[66]: array(['pickup_datetime', 'dropoff_datetime', 'trip_distance',  
              'fare_amount', 'tip_amount', 'payment_type', 'day_of_week',  
              'is_weekend'], dtype=object)
```

```
In [67]: 1 df_taxi.columns.tolist()
```

```
Out[67]: ['pickup_datetime',  
          'dropoff_datetime',  
          'trip_distance',  
          'fare_amount',  
          'tip_amount',  
          'payment_type',  
          'day_of_week',  
          'is_weekend']
```


What are the column datatypes?

What are the column datatypes?

```
In [68]: 1 df_taxi.dtypes
```

```
Out[68]: pickup_datetime    datetime64[ns]  
dropoff_datetime          datetime64[ns]  
trip_distance              float64  
fare_amount                float64  
tip_amount                 float64  
payment_type               object  
day_of_week                int64  
is_weekend                 bool  
dtype: object
```

What are the column datatypes?

```
In [68]: 1 df_taxi.dtypes
```

```
Out[68]: pickup_datetime    datetime64[ns]  
dropoff_datetime          datetime64[ns]  
trip_distance              float64  
fare_amount                float64  
tip_amount                 float64  
payment_type               object  
day_of_week                int64  
is_weekend                  bool  
dtype: object
```

```
In [69]: 1 type(df_taxi.dtypes)
```

```
Out[69]: pandas.core.series.Series
```

Get Summary Info for DataFrame

Get Summary Info for DataFrame

```
In [70]: 1 df_taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   pickup_datetime       1000 non-null   datetime64[ns]
1   dropoff_datetime      1000 non-null   datetime64[ns]
2   trip_distance         1000 non-null   float64
3   fare_amount           1000 non-null   float64
4   tip_amount            910 non-null    float64
5   payment_type          1000 non-null   object
6   day_of_week           1000 non-null   int64
7   is_weekend            1000 non-null   bool
dtypes: bool(1), datetime64[ns](2), float64(3), int64(1), object(1)
memory usage: 55.8+ KB
```

Get Summary Info for DataFrame

```
In [70]: 1 df_taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   pickup_datetime       1000 non-null   datetime64[ns]
1   dropoff_datetime      1000 non-null   datetime64[ns]
2   trip_distance         1000 non-null   float64
3   fare_amount           1000 non-null   float64
4   tip_amount            910 non-null    float64
5   payment_type          1000 non-null   object
6   day_of_week           1000 non-null   int64
7   is_weekend            1000 non-null   bool
dtypes: bool(1), datetime64[ns](2), float64(3), int64(1), object(1)
memory usage: 55.8+ KB
```

- number of rows
- number of columns
- column names, number of filled values, datatypes
- number of each datatype seen
- size of dataset in memory

Variable (Observation) Types

Variable (Observation) Types

- **Numeric** (eg. weight, temperature)
 - usually has a zero value
 - describes magnitude

Variable (Observation) Types

- **Numeric** (eg. weight, temperature)
 - usually has a zero value
 - describes magnitude
- **Categorical** (eg. class, variety)
 - usually a finite set
 - no order

Variable (Observation) Types

- **Numeric** (eg. weight, temperature)
 - usually has a zero value
 - describes magnitude
- **Categorical** (eg. class, variety)
 - usually a finite set
 - no order
- **Ordinal** (eg. Likert scale, education level, etc.)
 - usually a finite set
 - has order
 - usually missing zero
 - difference between levels may not be the same

Numeric: Data Ranges

Numeric: Data Ranges

```
In [71]: 1 df_taxi.trip_distance.min()
```

```
Out[71]: 0.0
```

Numeric: Data Ranges

```
In [71]: 1 df_taxi.trip_distance.min()
```

```
Out[71]: 0.0
```

```
In [72]: 1 df_taxi.trip_distance.max()
```

```
Out[72]: 32.77
```

Numeric: Data Ranges

```
In [71]: 1 df_taxi.trip_distance.min()
```

```
Out[71]: 0.0
```

```
In [72]: 1 df_taxi.trip_distance.max()
```

```
Out[72]: 32.77
```

```
In [73]: 1 df_taxi.min(numeric_only=True)
```

```
Out[73]: trip_distance      0.0  
fare_amount      2.5  
tip_amount      0.0  
day_of_week      0  
is_weekend      False  
dtype: object
```

Numeric: Data Ranges

```
In [71]: 1 df_taxi.trip_distance.min()
```

```
Out[71]: 0.0
```

```
In [72]: 1 df_taxi.trip_distance.max()
```

```
Out[72]: 32.77
```

```
In [73]: 1 df_taxi.min(numeric_only=True)
```

```
Out[73]: trip_distance    0.0  
fare_amount             2.5  
tip_amount              0.0  
day_of_week             0  
is_weekend              False  
dtype: object
```

```
In [74]: 1 df_taxi.max(numeric_only=True)
```

```
Out[74]: trip_distance    32.77  
fare_amount             88.0  
tip_amount             22.7  
day_of_week             6  
is_weekend              True  
dtype: object
```

Numeric: Central Tendency with Mean

Numeric: Central Tendency with Mean

- Sample Mean

$$\bar{x} = \frac{1}{n} \sum x_i$$

Numeric: Central Tendency with Mean

- Sample Mean

$$\bar{x} = \frac{1}{n} \sum x_i$$

```
In [75]: 1 df_taxi.fare_amount.mean()
```

```
Out[75]: 12.4426
```

Numeric: Central Tendency with Mean

- Sample Mean

$$\bar{x} = \frac{1}{n} \sum x_i$$

```
In [75]: 1 df_taxi.fare_amount.mean()
```

```
Out[75]: 12.4426
```

```
In [76]: 1 print(f'{df_taxi.fare_amount.mean() = :0.2f}')
```

```
df_taxi.fare_amount.mean() = 12.44
```

Numeric: Central Tendency with Mean

- Sample Mean

$$\bar{x} = \frac{1}{n} \sum x_i$$

```
In [75]: 1 df_taxi.fare_amount.mean()
```

```
Out[75]: 12.4426
```

```
In [76]: 1 print(f'{df_taxi.fare_amount.mean() = :0.2f}')
```

```
df_taxi.fare_amount.mean() = 12.44
```

- Mean is sensitive to *outliers*
- **Outlier:** a data point that differs significantly from other observations
 - data error
 - effect of heavy tailed distribution?

Numeric: Central Tendency with Median

Numeric: Central Tendency with Median

- Median
 - Divides sorted dataset into two equal sizes
 - 50% of the data is less than or equal to the median

Numeric: Central Tendency with Median

- Median
 - Divides sorted dataset into two equal sizes
 - 50% of the data is less than or equal to the median

```
In [77]: 1 df_taxi.fare_amount.median()
```

```
Out[77]: 9.0
```

Numeric: Central Tendency with Median

- Median
 - Divides sorted dataset into two equal sizes
 - 50% of the data is less than or equal to the median

```
In [77]: 1 df_taxi.fare_amount.median()
```

```
Out[77]: 9.0
```

- Median is *robust* to outliers
- **Robust:** Not affected by outliers

Numeric: Quantiles/Percentiles

Numeric: Quantiles/Percentiles

- **Quantile:** cut point for splitting distribution
- **Percentile:** $x\%$ of data is less than or equal to the x th percentile

Numeric: Quantiles/Percentiles

- **Quantile:** cut point for splitting distribution
- **Percentile:** $x\%$ of data is less than or equal to the x th percentile

```
In [78]: 1 df_taxi.fare_amount.quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

```
Out[78]: 33.5
```

Numeric: Quantiles/Percentiles

- **Quantile:** cut point for splitting distribution
- **Percentile:** $x\%$ of data is less than or equal to the x th percentile

```
In [78]: 1 df_taxi.fare_amount.quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

```
Out[78]: 33.5
```

```
In [79]: 1 df_taxi.fare_amount.quantile([.05,.95], interpolation='linear') # 90% of the data is between 4 and 33.5
```

```
Out[79]: 0.05    4.0  
         0.95   33.5  
         Name: fare_amount, dtype: float64
```

Numeric: Quantiles/Percentiles

- **Quantile:** cut point for splitting distribution
- **Percentile:** $x\%$ of data is less than or equal to the x th percentile

```
In [78]: 1 df_taxi.fare_amount.quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

```
Out[78]: 33.5
```

```
In [79]: 1 df_taxi.fare_amount.quantile([.05,.95], interpolation='linear') # 90% of the data is between 4 and 33.5
```

```
Out[79]: 0.05      4.0  
         0.95     33.5  
         Name: fare_amount, dtype: float64
```

```
In [80]: 1 df_taxi.fare_amount.quantile([0,.25,.5,.75,1]) # Quartiles: 25% of data is between each pair
```

```
Out[80]: 0.00      2.5  
         0.25      6.5  
         0.50      9.0  
         0.75     14.0  
         1.00     88.0  
         Name: fare_amount, dtype: float64
```

Numeric: Spread with Variance

Numeric: Spread with Variance

- Sample Variance

$$s^2 = \frac{\sum (x - \bar{x})^2}{n-1}$$

Numeric: Spread with Variance

- Sample Variance

$$s^2 = \frac{\sum (x - \bar{x})^2}{n-1}$$

```
In [81]: 1 round(df_taxi.fare_amount.var(),3)
```

```
Out[81]: 116.809
```


Numeric: Spread with Variance

- Sample Variance

$$s^2 = \frac{\sum (x - \bar{x})^2}{n-1}$$

```
In [81]: 1 round(df_taxi.fare_amount.var(), 3)
```

```
Out[81]: 116.809
```

but this is in dollars²!

Numeric: Spread with Standard Deviation

Numeric: Spread with Standard Deviation

- Sample Standard Deviation

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

Numeric: Spread with Standard Deviation

- Sample Standard Deviation

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

```
In [82]: 1 round(df_taxi.fare_amount.std(), 3)
```

```
Out[82]: 10.808
```

Numeric: Spread with Standard Deviation

- Sample Standard Deviation

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

```
In [82]: 1 round(df_taxi.fare_amount.std(), 3)
```

```
Out[82]: 10.808
```

- Back in original scale of dollars
- Sensitive to outliers

Numeric: Exploring Spread with IQR

Numeric: Exploring Spread with IQR

- Quartiles
 - ~25% of data is \leq first quartile, 25th percentile
 - ~50% of data is \leq second quartile, 50th percentile (Median)
 - ~75% of data is \leq third quartile, 75th percentile

Numeric: Exploring Spread with IQR

- Quartiles
 - ~25% of data is \leq first quartile, 25th percentile
 - ~50% of data is \leq second quartile, 50th percentile (Median)
 - ~75% of data is \leq third quartile, 75th percentile
- Can find quartiles with: pandas quantile or numpy percentile

Numeric: Exploring Spread with IQR

- Quartiles
 - ~25% of data is \leq first quartile, 25th percentile
 - ~50% of data is \leq second quartile, 50th percentile (Median)
 - ~75% of data is \leq third quartile, 75th percentile
- Can find quartiles with: pandas quantile or numpy percentile
- **Interquartile Range (IQR)**
 - (third quartile - first quartile) or (75th percentile - 25th percentile)

Numeric: Exploring Spread with IQR

- Quartiles
 - ~25% of data is \leq first quartile, 25th percentile
 - ~50% of data is \leq second quartile, 50th percentile (Median)
 - ~75% of data is \leq third quartile, 75th percentile
- Can find quartiles with: pandas quantile or numpy percentile
- Interquartile Range (IQR)
 - (third quartile - first quartile) or (75th percentile - 25th percentile)

```
In [83]: 1 df_taxi.fare_amount.quantile(.75) - df_taxi.fare_amount.quantile(.25)
```

```
Out[83]: 7.5
```

Numeric: Exploring Spread with IQR

- Quartiles
 - ~25% of data is \leq first quartile, 25th percentile
 - ~50% of data is \leq second quartile, 50th percentile (Median)
 - ~75% of data is \leq third quartile, 75th percentile
- Can find quartiles with: pandas quantile or numpy percentile
- Interquartile Range (IQR)
 - (third quartile - first quartile) or (75th percentile - 25th percentile)

```
In [83]: 1 df_taxi.fare_amount.quantile(.75) - df_taxi.fare_amount.quantile(.25)
```

```
Out[83]: 7.5
```

- IQR is robust to outliers

Numeric: Exploring Distribution with Skew

Numeric: Exploring Distribution with Skew

- Skewness
 - measures asymmetry of distribution around mean
 - indicates tail to left (neg) or right (pos)
 - skew will lead to difference between median and mean

Numeric: Exploring Distribution with Skew

- Skewness
 - measures asymmetry of distribution around mean
 - indicates tail to left (neg) or right (pos)
 - skew will lead to difference between median and mean

```
In [84]: 1 df_taxi.fare_amount.skew()
```

```
Out[84]: 2.882730031010152
```

Numeric: Exploring Distribution with Skew

- Skewness
 - measures asymmetry of distribution around mean
 - indicates tail to left (neg) or right (pos)
 - skew will lead to difference between median and mean

```
In [84]: 1 df_taxi.fare_amount.skew()
```

```
Out[84]: 2.882730031010152
```

Easier to understand with a plot (histogram/boxplot)...

Numeric Summary Stats with `.describe`

Numeric Summary Stats with `.describe`

In [85]: 1 df_taxi.describe()

Out[85]:

	trip_distance	fare_amount	tip_amount	day_of_week
count	1000.000000	1000.000000	910.000000	1000.000000
mean	2.880010	12.442600	1.766275	2.987000
std	3.678534	10.807802	2.315507	2.043773
min	0.000000	2.500000	0.000000	0.000000
25%	0.950000	6.500000	0.000000	1.000000
50%	1.565000	9.000000	1.350000	3.000000
75%	3.100000	14.000000	2.460000	5.000000
max	32.770000	88.000000	22.700000	6.000000

Numeric Summary Stats with `.describe`

```
In [85]: 1 df_taxi.describe()
```

Out[85]:

	trip_distance	fare_amount	tip_amount	day_of_week
count	1000.000000	1000.000000	910.000000	1000.000000
mean	2.880010	12.442600	1.766275	2.987000
std	3.678534	10.807802	2.315507	2.043773
min	0.000000	2.500000	0.000000	0.000000
25%	0.950000	6.500000	0.000000	1.000000
50%	1.565000	9.000000	1.350000	3.000000
75%	3.100000	14.000000	2.460000	5.000000
max	32.770000	88.000000	22.700000	6.000000

```
In [86]: 1 df_taxi.describe().round(2) # reduce precision with round
```

Out[86]:

	trip_distance	fare_amount	tip_amount	day_of_week
count	1000.00	1000.00	910.00	1000.00
mean	2.88	12.44	1.77	2.99
std	3.68	10.81	2.32	2.04
min	0.00	2.50	0.00	0.00
25%	0.95	6.50	0.00	1.00
50%	1.56	9.00	1.35	3.00
75%	3.10	14.00	2.46	5.00
max	32.77	88.00	22.70	6.00

Bivariate: Evaluating Correlation

Calculating Correlation

Calculating Correlation

```
In [87]: 1 df_taxi.trip_distance.corr(df_taxi.fare_amount).round(2)
```

```
Out[87]: 0.95
```

Calculating Correlation

```
In [87]: 1 df_taxi.trip_distance.corr(df_taxi.fare_amount).round(2)
```

```
Out[87]: 0.95
```

```
In [88]: 1 from scipy.stats import pearsonr
2 r,p = pearsonr(df_taxi.trip_distance, df_taxi.fare_amount)
3 print(f"{r = :.2f}, {p = :.2f}")
```

```
r = 0.95, p = 0.00
```

Counting Categorical Values with `.value_count()`

Counting Categorical Values with `.value_count()`

```
In [89]: 1 df_taxi.payment_type.value_counts()
```

```
Out[89]: Credit card    663  
Cash                335  
No charge             2  
Name: payment_type, dtype: int64
```

Counting Categorical Values with `.value_count()`

```
In [89]: 1 df_taxi.payment_type.value_counts()
```

```
Out[89]: Credit card    663  
Cash                335  
No charge            2  
Name: payment_type, dtype: int64
```

```
In [90]: 1 df_taxi.payment_type.value_counts(normalize=True)
```

```
Out[90]: Credit card    0.663  
Cash                0.335  
No charge            0.002  
Name: payment_type, dtype: float64
```

Counting Categorical Values with `.value_count()`

```
In [89]: 1 df_taxi.payment_type.value_counts()
```

```
Out[89]: Credit card    663  
Cash                335  
No charge           2  
Name: payment_type, dtype: int64
```

```
In [90]: 1 df_taxi.payment_type.value_counts(normalize=True)
```

```
Out[90]: Credit card    0.663  
Cash                0.335  
No charge           0.002  
Name: payment_type, dtype: float64
```

```
In [91]: 1 tmp = pd.DataFrame()  
2 tmp['count'] = df_taxi.payment_type.value_counts()  
3 tmp['prop'] = df_taxi.payment_type.value_counts(normalize=True)  
4 tmp.round(2)
```

```
Out[91]:
```

	count	prop
Credit card	663	0.66
Cash	335	0.34
No charge	2	0.00

Applying Functions to Groups of Data

Applying Functions to Groups of Data

```
In [92]: 1 df_taxi.groupby('payment_type')
```

```
Out[92]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff1a6762670>
```

Applying Functions to Groups of Data

```
In [92]: 1 df_taxi.groupby('payment_type')
```

```
Out[92]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff1a6762670>
```

```
In [93]: 1 df_taxi.groupby('payment_type').mean()
```

```
/var/folders/78/vhnqkq8n45dd4gj4f5qx8yb00000gn/T/ipykernel_46425/2464981742.py:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.  
df_taxi.groupby('payment_type').mean()
```

```
Out[93]:
```

	trip_distance	fare_amount	tip_amount	day_of_week	is_weekend
payment_type					
Cash	2.732209	11.856716	0.000000	2.898507	0.152239
Credit card	2.961870	12.761086	2.683322	3.039216	0.149321
No charge	0.500000	5.000000	0.000000	0.500000	0.000000

Applying Functions to Groups of Data

```
In [92]: 1 df_taxi.groupby('payment_type')
```

```
Out[92]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff1a6762670>
```

```
In [93]: 1 df_taxi.groupby('payment_type').mean()
```

```
/var/folders/78/vhnqkq8n45dd4gj4f5qx8yb00000gn/T/ipykernel_46425/2464981742.py:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.  
df_taxi.groupby('payment_type').mean()
```

```
Out[93]:
```

	trip_distance	fare_amount	tip_amount	day_of_week	is_weekend
payment_type					
Cash	2.732209	11.856716	0.000000	2.898507	0.152239
Credit card	2.961870	12.761086	2.683322	3.039216	0.149321
No charge	0.500000	5.000000	0.000000	0.500000	0.000000

```
In [94]: 1 # applying multiple aggregation functions  
2 df_taxi.groupby('payment_type')['trip_distance'].agg(['count', 'mean', 'median']).round(2)
```

```
Out[94]:
```

	count	mean	median
payment_type			
Cash	335	2.73	1.37
Credit card	663	2.96	1.70
No charge	2	0.50	0.50

Applying Functions to Groups of Data

```
In [92]: 1 df_taxi.groupby('payment_type')
```

Out[92]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff1a6762670>

```
In [93]: 1 df_taxi.groupby('payment_type').mean()
```

/var/folders/78/vhnqkq8n45dd4gj4f5qx8yb00000gn/T/ipykernel_46425/2464981742.py:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
df_taxi.groupby('payment_type').mean()

Out[93]:

	trip_distance	fare_amount	tip_amount	day_of_week	is_weekend
payment_type					
Cash	2.732209	11.856716	0.000000	2.898507	0.152239
Credit card	2.961870	12.761086	2.683322	3.039216	0.149321
No charge	0.500000	5.000000	0.000000	0.500000	0.000000

```
In [94]: 1 # applying multiple aggregation functions
2 df_taxi.groupby('payment_type')['trip_distance'].agg(['count', 'mean', 'median']).round(2)
```

Out[94]:

	count	mean	median
payment_type			
Cash	335	2.73	1.37
Credit card	663	2.96	1.70
No charge	2	0.50	0.50

```
In [95]: 1 df_taxi[df_taxi.payment_type.isin(['Cash', 'Credit card'])].groupby(['payment_type', 'is_weekend']).trip_distance.agg(['mean', 'median'])
```

Out[95]:

		mean	median
payment_type	is_weekend		

Aside: Dealing with long chains

- long chains may not be visible in notebooks

Aside: Dealing with long chains

- long chains may not be visible in notebooks

```
In [96]: 1 # df_taxi[df_taxi.payment_type.isin(['Cash', 'Credit card'])].groupby(['payment_type', 'is_weekend']).trip_distance.agg(['mean', 'median'])
          2
          3 # use backslashes
          4 df_taxi.loc[df_taxi.payment_type.isin(['Cash'])]\
          5     .groupby(['payment_type', 'is_weekend'])\
          6     .trip_distance.agg(['mean', 'median'])
```

Out[96]:

		mean	median
payment_type	is_weekend		
Cash	False	2.593063	1.28
	True	3.507059	2.10

Aside: Dealing with long chains

- long chains may not be visible in notebooks

```
In [96]: 1 # df_taxi[df_taxi.payment_type.isin(['Cash','Credit card'])].groupby(['payment_type','is_weekend']).trip_distance.agg(['mean','median'])
2
3 # use backslashes
4 df_taxi.loc[df_taxi.payment_type.isin(['Cash'])]\
5     .groupby(['payment_type','is_weekend'])\
6     .trip_distance.agg(['mean','median'])
```

Out[96]:

		mean	median
payment_type	is_weekend		
Cash	False	2.593063	1.28
	True	3.507059	2.10

```
In [97]: 1 # wrap in parentheses
2 (
3     df_taxi
4     .loc[df_taxi.payment_type.isin(['Cash'])]
5     .groupby(['payment_type','is_weekend'])
6     .trip_distance.agg(['mean','median'])
7 )
```

Out[97]:

		mean	median
payment_type	is_weekend		
Cash	False	2.593063	1.28
	True	3.507059	2.10

Questions?

Visualizations in Python

- dataframes as tables
- plotting with `matplotlib.pyplot`
- plotting with `pandas`
- plotting with `seaborn`

- need interactive plots? `plotly`

DataFrames as Tables

DataFrames as Tables

```
In [98]: 1 df_taxi[['trip_distance', 'fare_amount']].head(10)
```

Out[98]:

	trip_distance	fare_amount
0	0.89	5.5
1	2.70	14.0
2	1.41	8.0
3	0.40	4.0
4	2.30	11.0
5	0.80	5.5
6	0.20	4.5
7	2.68	11.5
8	0.60	4.5
9	0.90	6.0

Styling dataframes with `style`

Styling dataframes with `style`

```
In [99]: 1 (
          2     df_taxi[['trip_distance', 'fare_amount']]
          3     .head(10)
          4     .style
          5     .format(precision=1)
          6     .background_gradient()
          7 )
```

Out[99]:

	trip_distance	fare_amount
0	0.9	5.5
1	2.7	14.0
2	1.4	8.0
3	0.4	4.0
4	2.3	11.0
5	0.8	5.5
6	0.2	4.5
7	2.7	11.5
8	0.6	4.5
9	0.9	6.0

Styling dataframes with `style`

```
In [99]: 1 (
          2     df_taxi[['trip_distance', 'fare_amount']]
          3     .head(10)
          4     .style
          5     .format(precision=1)
          6     .background_gradient()
          7 )
```

Out[99]:

	trip_distance	fare_amount
0	0.9	5.5
1	2.7	14.0
2	1.4	8.0
3	0.4	4.0
4	2.3	11.0
5	0.8	5.5
6	0.2	4.5
7	2.7	11.5
8	0.6	4.5
9	0.9	6.0

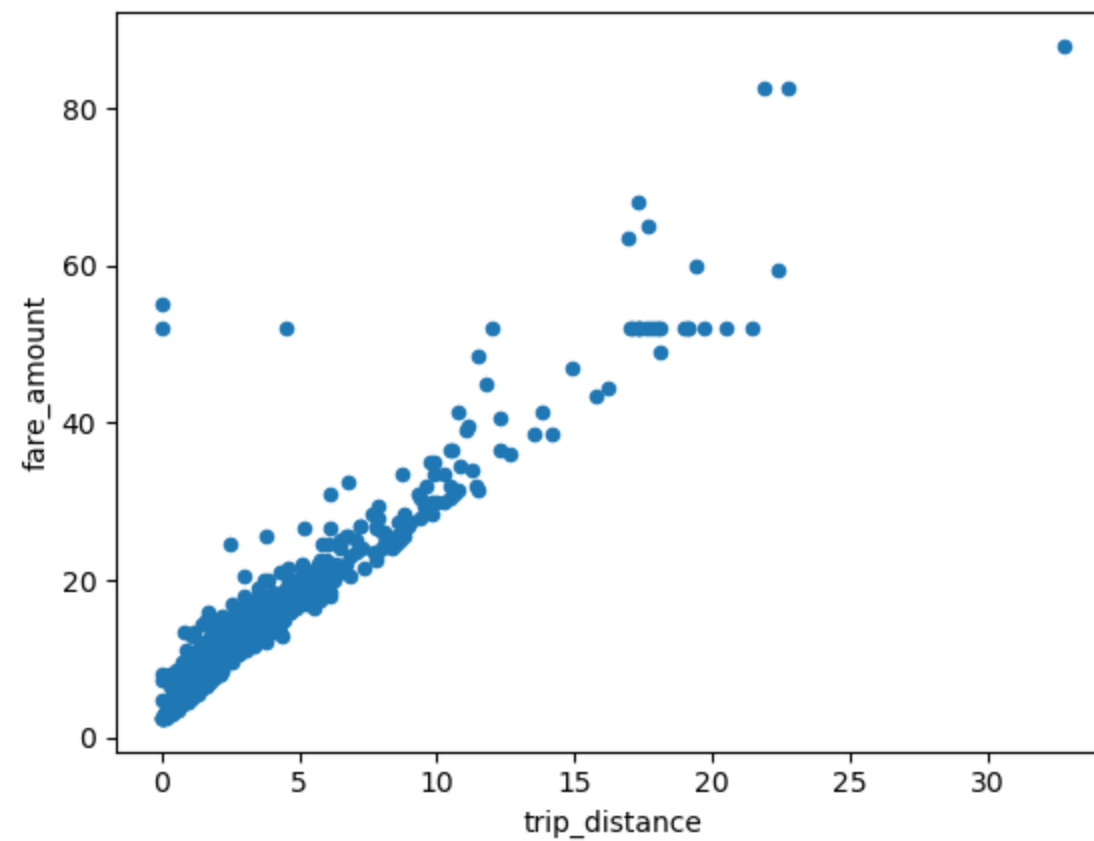
For more info: https://pandas.pydata.org/docs/user_guide/style.html

Plotting via Pandas

Plotting via Pandas

```
In [100]: 1 df_taxi.plot.scatter(x='trip_distance',y='fare_amount')
```

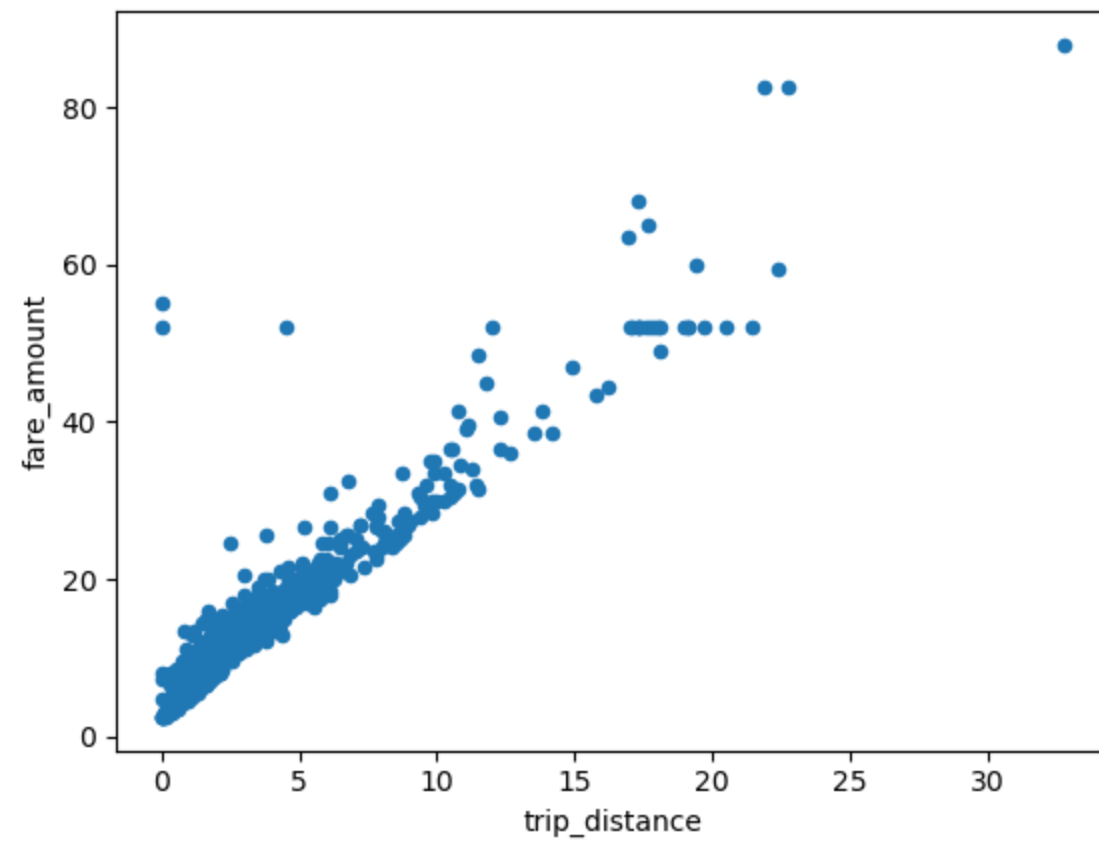
```
Out[100]: <AxesSubplot: xlabel='trip_distance', ylabel='fare_amount'>
```



Using semi-colon to hide supress "end of cell print"

Using semi-colon to hide supress "end of cell print"

```
In [101]: 1 df_taxi.plot.scatter(x='trip_distance',y='fare_amount');
```



Manipulating plots with Matplotlib

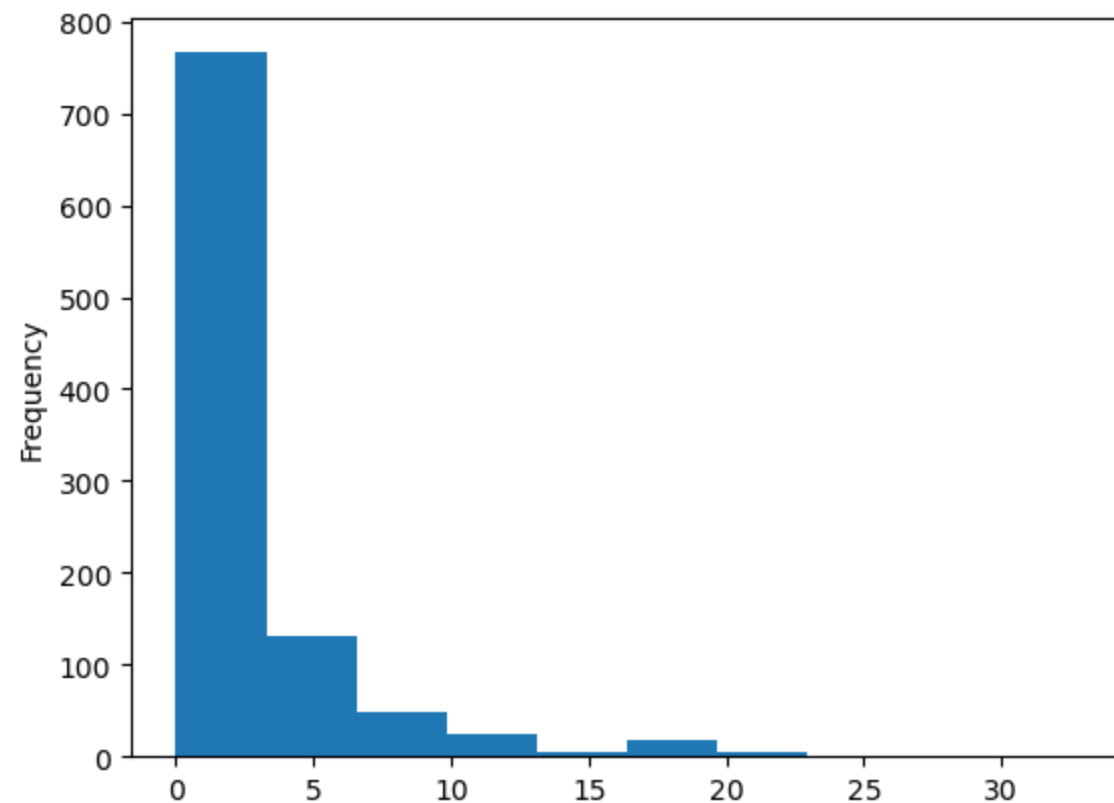
- sizing
- adding titles
- changing axis labels
- changing axis ticks

Manipulating plots with Matplotlib

- sizing
- adding titles
- changing axis labels
- changing axis ticks

```
In [102]: 1 ax = df_taxi.trip_distance.plot.hist()  
          2 type(ax)
```

Out[102]: matplotlib.axes._subplots.AxesSubplot




```
Import matplotlib.pyplot
```

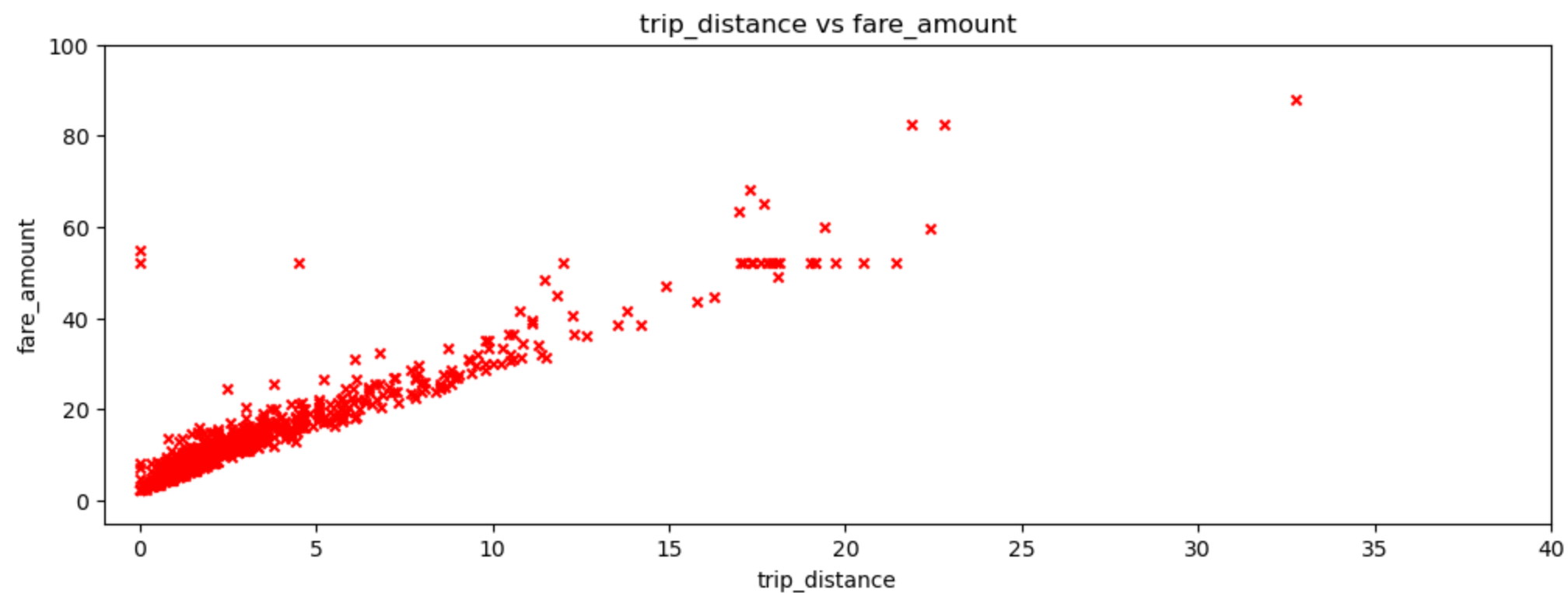
Import matplotlib.pyplot

```
In [103]: 1 import matplotlib.pyplot as plt  
          2  
          3 %matplotlib inline
```

Matplotlib Axes

Matplotlib Axes

```
In [104]: 1 fig,ax = plt.subplots(1,1,figsize=(12,4)) # set the figure size
2 (
3     df_taxi
4     .plot.scatter(
5         x = 'trip_distance',
6         y = 'fare_amount',
7         marker='x',
8         color='red',
9         ax=ax
10    )
11 )
12 ax.set_xlabel('trip_distance') # set x and y axis labels
13 ax.set_ylabel('fare_amount')
14 ax.set_xlim([-1,40]) # set x and y axis limits
15 ax.set_ylim([-5,100])
16 ax.set_title('trip_distance vs fare_amount'); # set axis title
```



Matplotlib and dpi

Matplotlib and dpi

```
In [105]: 1 def find_dpi(w, h, d):
          2     """
          3     https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
          4     w : width in pixels
          5     h : height in pixels
          6     d : diagonal in inches
          7     """
          8     w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
          9     return round(w / w_inches)
         10
         11 find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[105]: 166

Matplotlib and dpi

```
In [105]: 1 def find_dpi(w, h, d):
          2     """
          3     https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
          4     w : width in pixels
          5     h : height in pixels
          6     d : diagonal in inches
          7     """
          8     w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
          9     return round(w / w_inches)
         10
         11 find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[105]: 166

```
In [106]: 1 fig.dpi # from previous figure
```

Out[106]: 100.0

Matplotlib and dpi

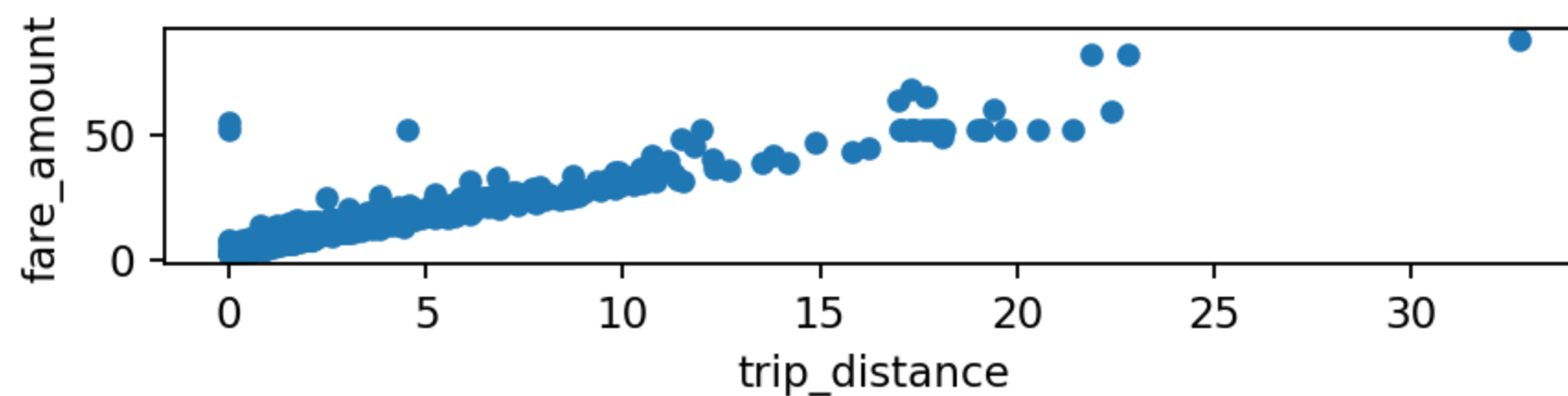
```
In [105]: 1 def find_dpi(w, h, d):
          2     """
          3     https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
          4     w : width in pixels
          5     h : height in pixels
          6     d : diagonal in inches
          7     """
          8     w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
          9     return round(w / w_inches)
         10
         11 find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[105]: 166

```
In [106]: 1 fig.dpi # from previous figure
```

Out[106]: 100.0

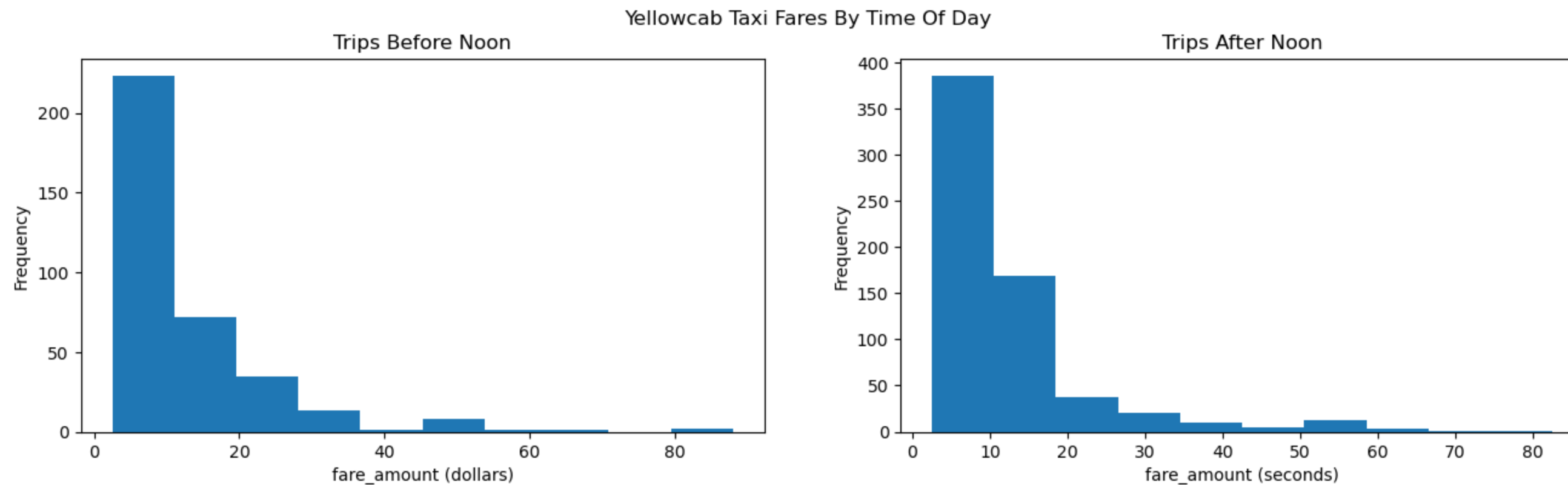
```
In [107]: 1 fig, ax = plt.subplots(figsize=(6, 1), dpi=166)
          2 df_taxi.plot.scatter(x = 'trip_distance', y = 'fare_amount', ax=ax);
```



Matplotlib: Subplots, Figure and Axis

Matplotlib: Subplots, Figure and Axis

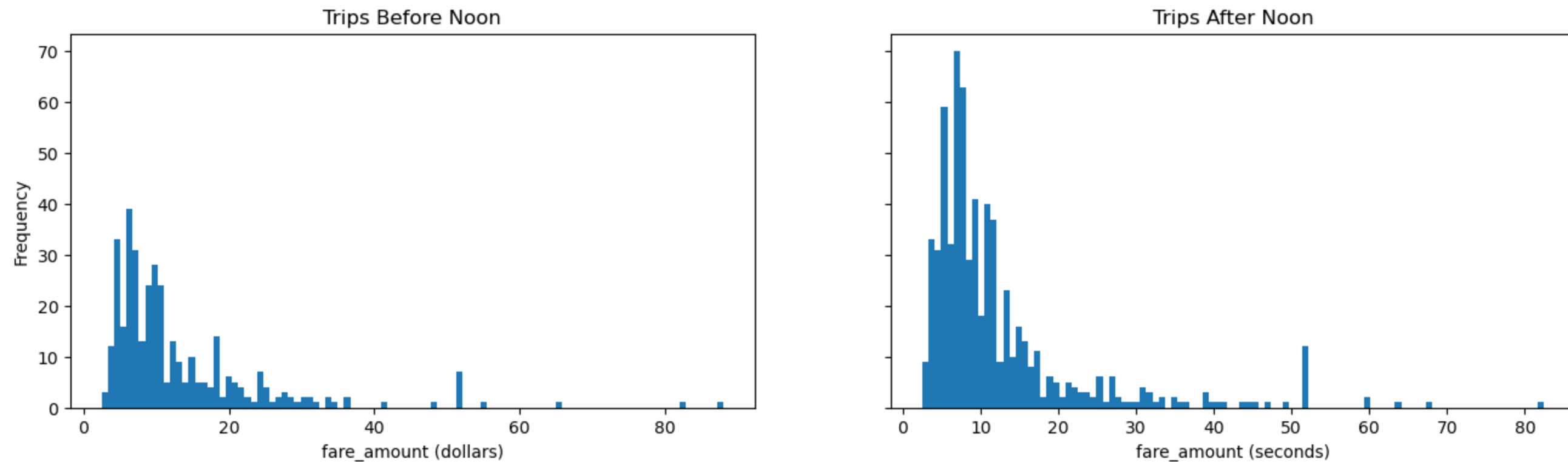
```
In [108]: 1 fig,ax = plt.subplots(1,2,figsize=(16,4))
2
3 df_taxi[df_taxi.pickup_datetime.dt.hour < 12].fare_amount.plot.hist(ax=ax[0]);
4 ax[0].set_xlabel('fare_amount (dollars)');
5 ax[0].set_title('Trips Before Noon');
6
7 df_taxi[df_taxi.pickup_datetime.dt.hour >= 12].fare_amount.plot.hist(ax=ax[1]);
8 ax[1].set_xlabel('fare_amount (seconds)');
9 ax[1].set_title('Trips After Noon');
10
11 fig.suptitle('Yellowcab Taxi Fares By Time Of Day');
```



Matplotlib: Sharing Axes

Matplotlib: Sharing Axes

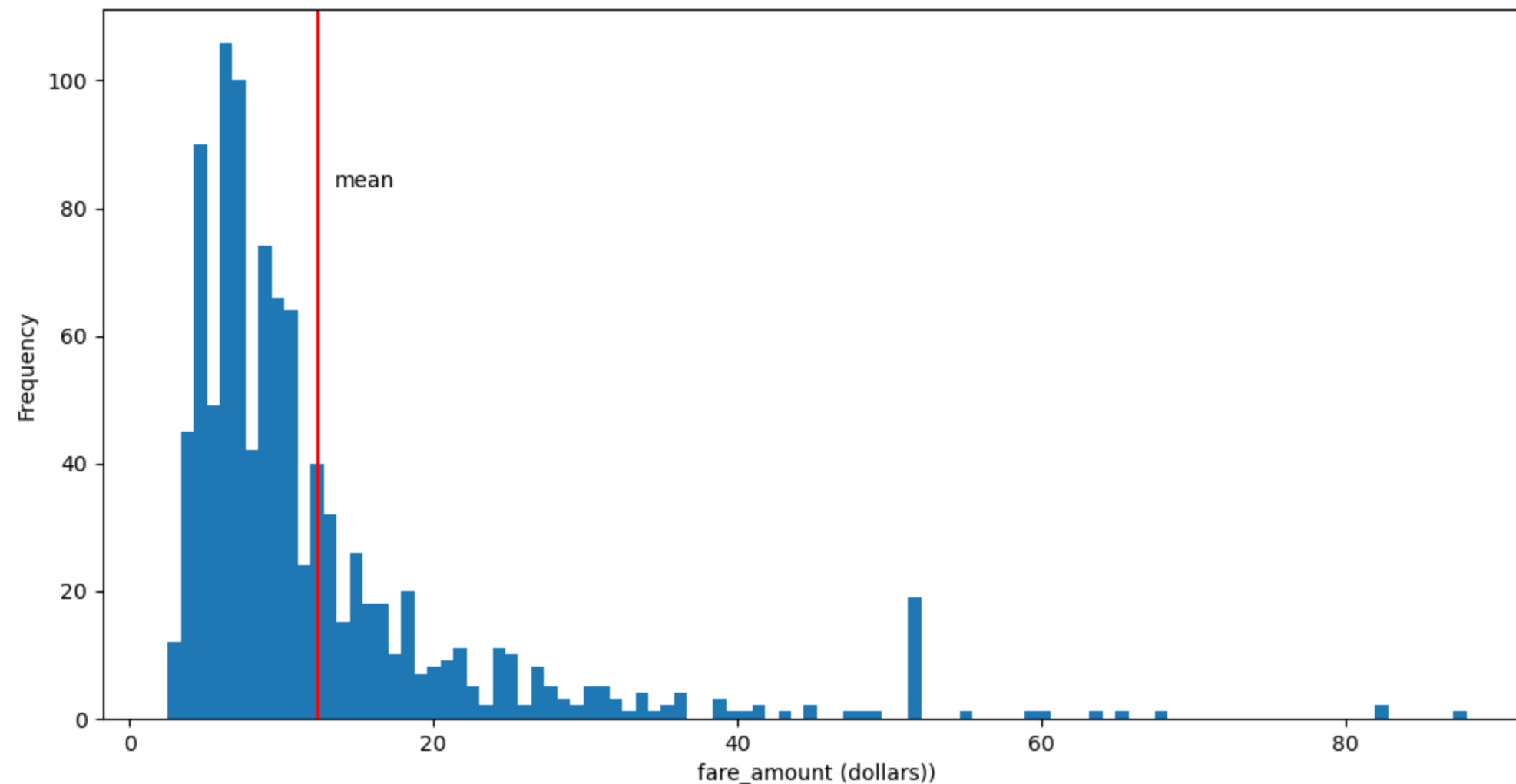
```
In [109]: 1 fig,ax = plt.subplots(1,2,figsize=(16,4), sharey=True)
2
3 df_taxi[df_taxi.pickup_datetime.dt.hour < 12].fare_amount.plot.hist(bins=100,ax=ax[0]);
4 ax[0].set_xlabel('fare_amount (dollars)');
5 ax[0].set_title('Trips Before Noon');
6 df_taxi[df_taxi.pickup_datetime.dt.hour >= 12].fare_amount.plot.hist(bins=100,ax=ax[1]);
7 ax[1].set_xlabel('fare_amount (seconds)');
8 ax[1].set_title('Trips After Noon');
```



Matplotlib: adding lines and annotations

Matplotlib: adding lines and annotations

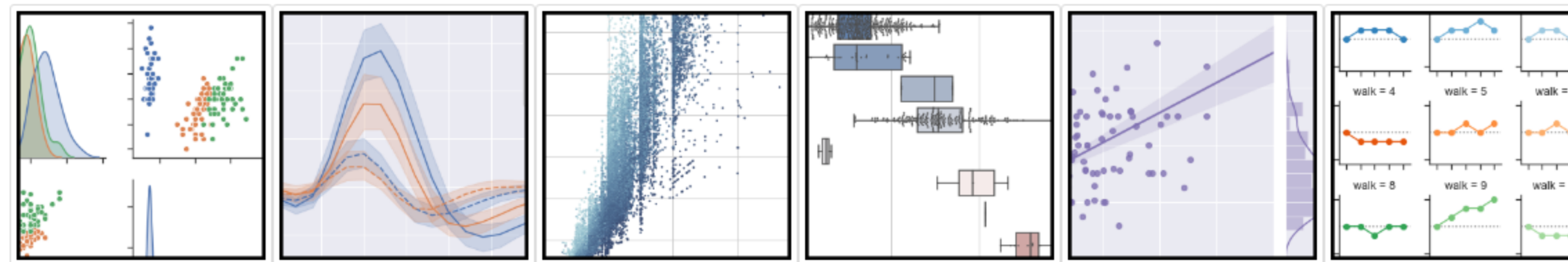
```
In [110]: 1 fig,ax = plt.subplots(1,1,figsize=(12,6));
2
3 df_taxi.fare_amount.plot.hist(bins=100, ax=ax);
4 ax.set_xlabel('fare_amount (dollars)');
5
6 # add a vertical line
7 ax.axvline(df_taxi.fare_amount.mean(),color='r');
8 #ax.vlines(df_taxi.fare_amount.mean(),*ax.get_ylim(),color='r');
9
10 # add some text
11 ax.text(df_taxi.fare_amount.mean()+1,ax.get_ylim()[1]*.75,'mean');
```



Plotting with Seaborn

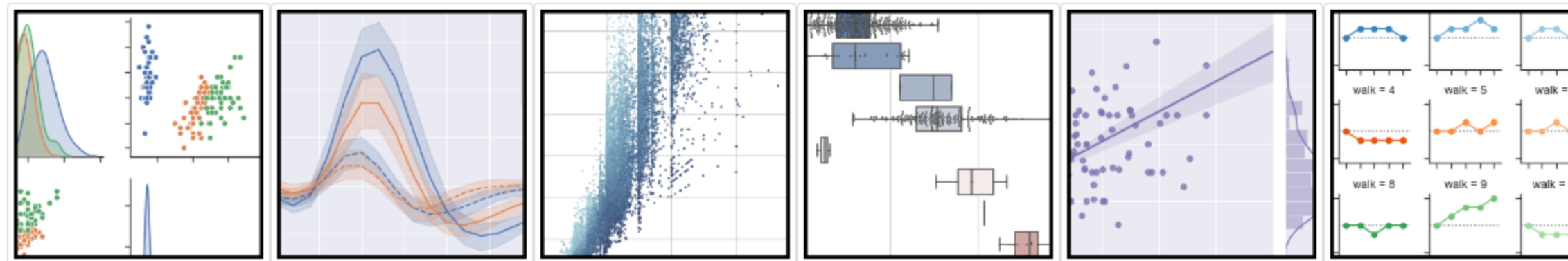
Plotting with Seaborn

- Python data visualization library
- Based on matplotlib.
- It provides a high-level interface for drawing attractive and informative statistical graphics.



Plotting with Seaborn

- Python data visualization library
- Based on matplotlib.
- It provides a high-level interface for drawing attractive and informative statistical graphics.



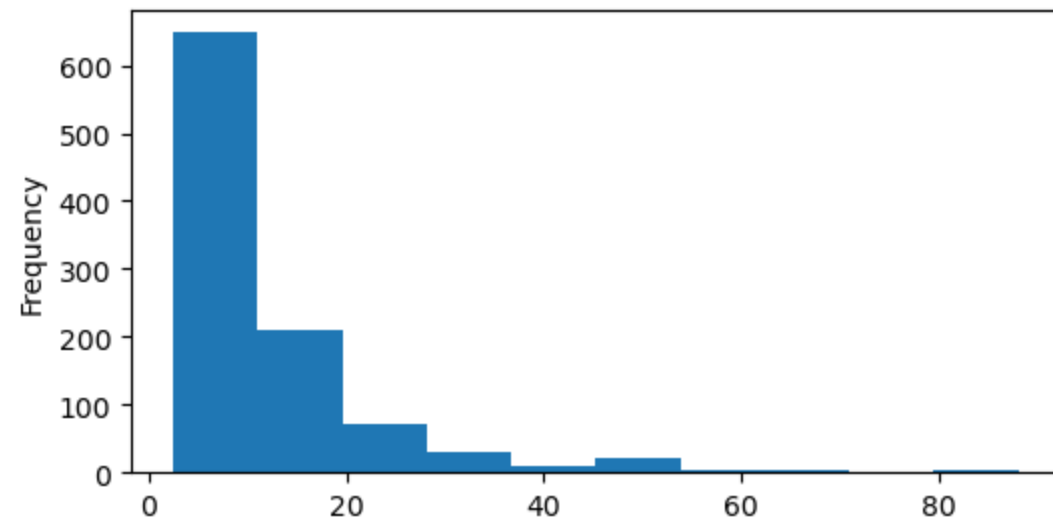
```
In [111]: 1 import seaborn as sns  
          2 sns.__version__
```

```
Out[111]: '0.12.2'
```

Univariate Distribution: Histograms

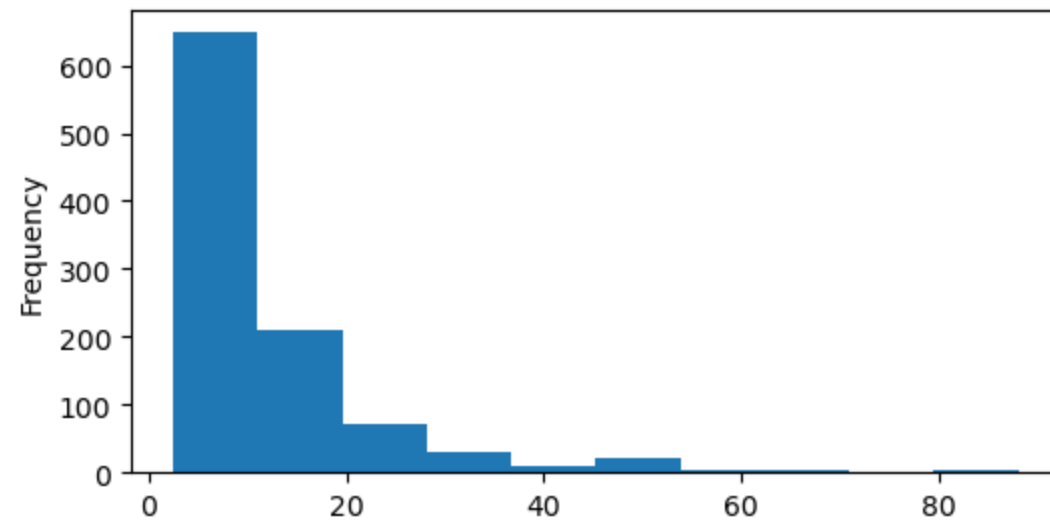
Univariate Distribution: Histograms

```
In [112]: 1 fig,ax = plt.subplots(1,1,figsize=(6,3))  
          2  
          3 df_taxi.fare_amount.plot.hist(ax=ax);
```

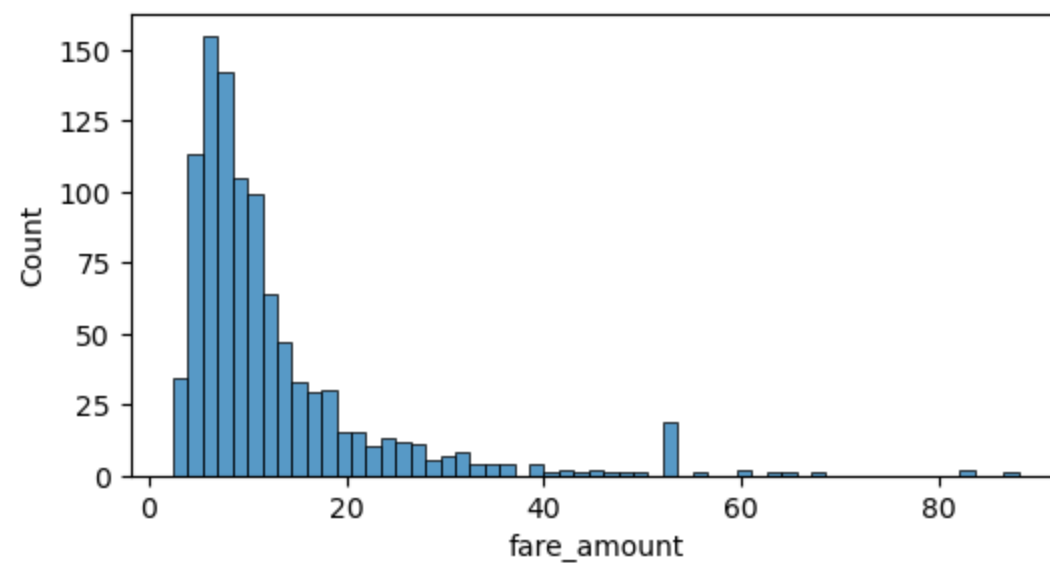


Univariate Distribution: Histograms

```
In [112]: 1 fig,ax = plt.subplots(1,1,figsize=(6,3))
          2
          3 df_taxi.fare_amount.plot.hist(ax=ax);
```



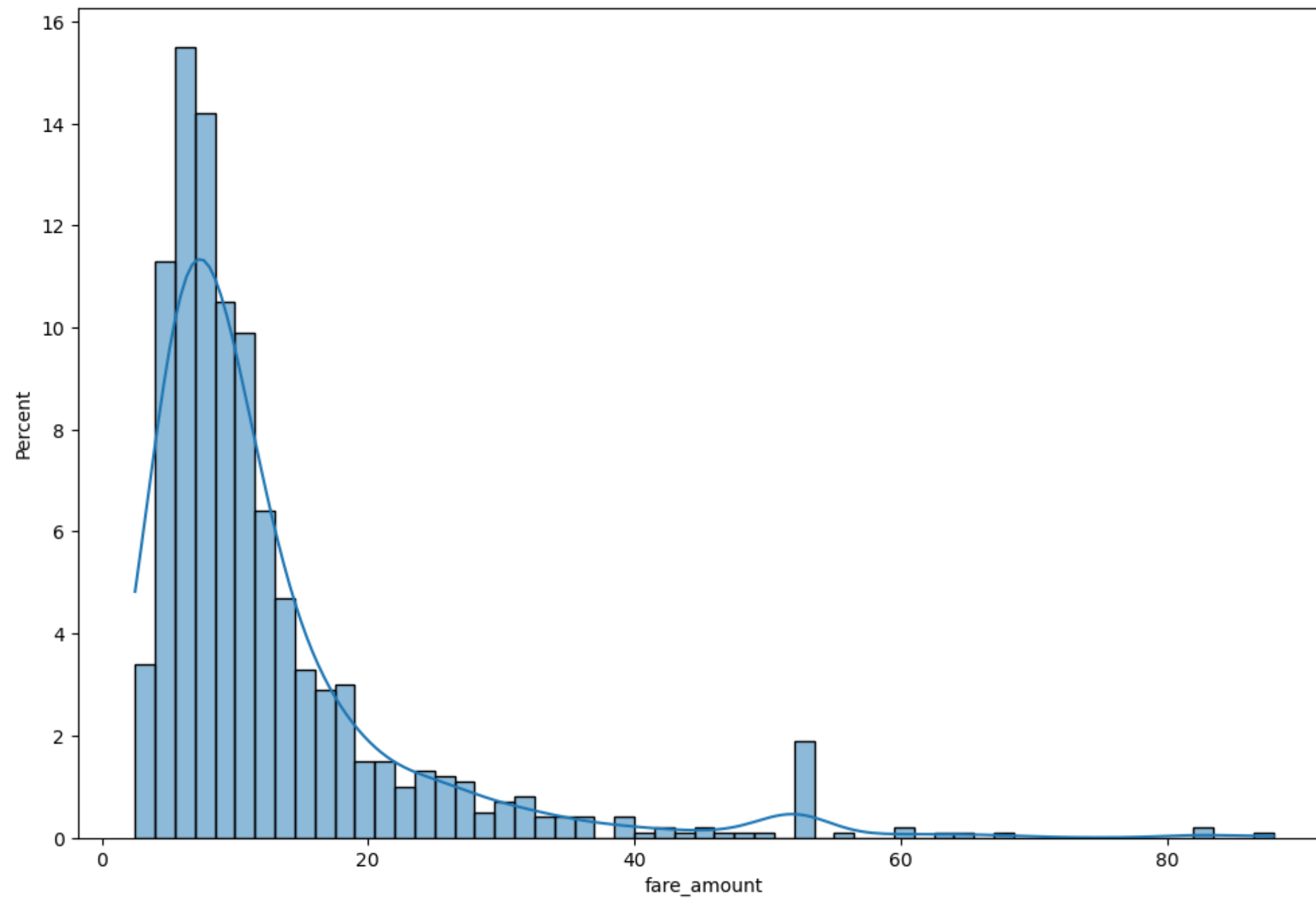
```
In [113]: 1 fig,ax = plt.subplots(1,1,figsize=(6,3))
          2
          3 sns.histplot(x='fare_amount',data=df_taxi,ax=ax); # sns.histplot(x=df_taxi.fare_amount,ax=ax);
```



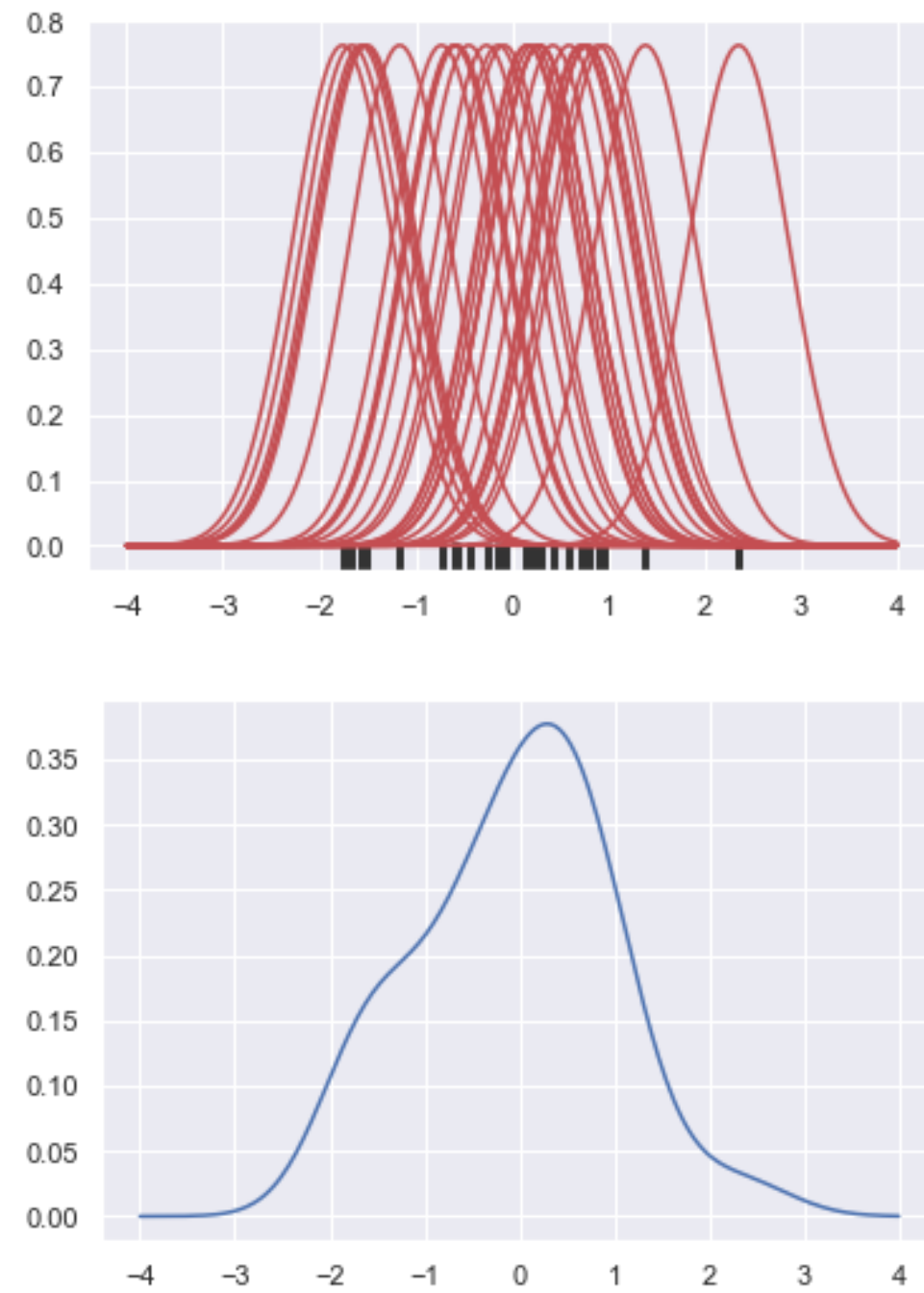
Univariate Distribution: Histograms

Univariate Distribution: Histograms

```
In [114]: 1 fig,ax = plt.subplots(1,1,figsize=(12,8))
          2
          3 # many other parameters to play with
          4 sns.histplot(x='fare_amount',data=df_taxi,ax=ax,kde=True,stat='percent');
```



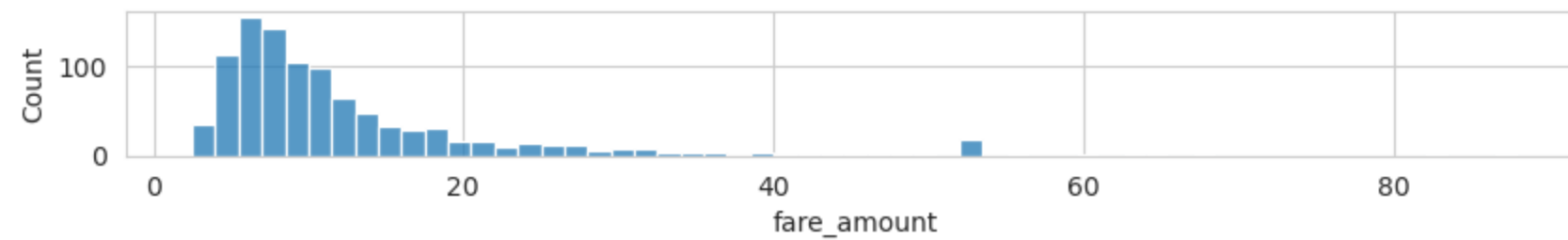
Aside: KDE



Seaborn Styles

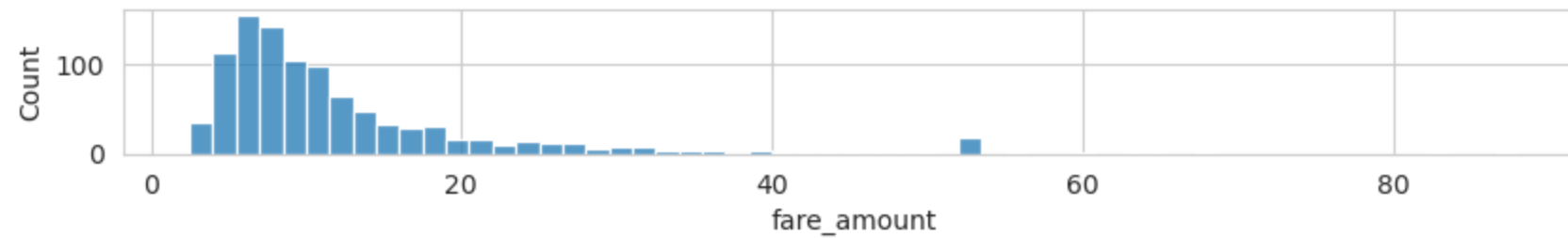
Seaborn Styles

```
In [115]: 1 # for a single plot using a context
          2 with sns.axes_style('whitegrid'):
          3     fig, ax = plt.subplots(1, 1, figsize=(10, 1))
          4     sns.histplot(x='fare_amount', data=df_taxi);
```



Seaborn Styles

```
In [115]: 1 # for a single plot using a context
          2 with sns.axes_style('whitegrid'):
          3     fig, ax = plt.subplots(1, 1, figsize=(10, 1))
          4     sns.histplot(x='fare_amount', data=df_taxi);
```



```
In [116]: 1 # set style globally: darkgrid, whitegrid, dark, white, ticks
          2 sns.set_style('darkgrid')
```

Seaborn Styles

```
In [115]: 1 # for a single plot using a context
          2 with sns.axes_style('whitegrid'):
          3     fig,ax = plt.subplots(1,1,figsize=(10,1))
          4     sns.histplot(x='fare_amount',data=df_taxi);
```



```
In [116]: 1 # set style globally: darkgrid, whitegrid, dark, white, ticks
          2 sns.set_style('darkgrid')
```

```
In [117]: 1 fig,ax = plt.subplots(1,1,figsize=(10,1))
          2 sns.histplot(x='fare_amount',data=df_taxi);
```



Seaborn Styles

```
In [115]: 1 # for a single plot using a context
          2 with sns.axes_style('whitegrid'):
          3     fig,ax = plt.subplots(1,1,figsize=(10,1))
          4     sns.histplot(x='fare_amount',data=df_taxi);
```



```
In [116]: 1 # set style globally: darkgrid, whitegrid, dark, white, ticks
          2 sns.set_style('darkgrid')
```

```
In [117]: 1 fig,ax = plt.subplots(1,1,figsize=(10,1))
          2 sns.histplot(x='fare_amount',data=df_taxi);
```

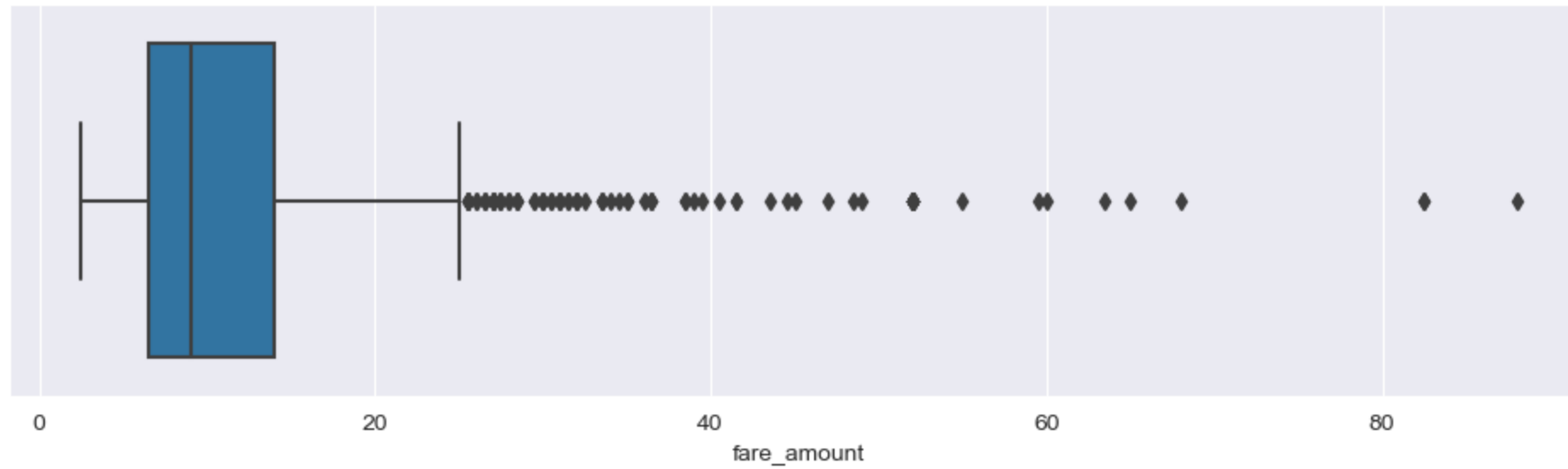


```
In [118]: 1 # to reset to matplotlib defaults
          2 #import matplotlib
          3 #matplotlib.rc_file_defaults()
```

Univariate Distributions: Boxplot

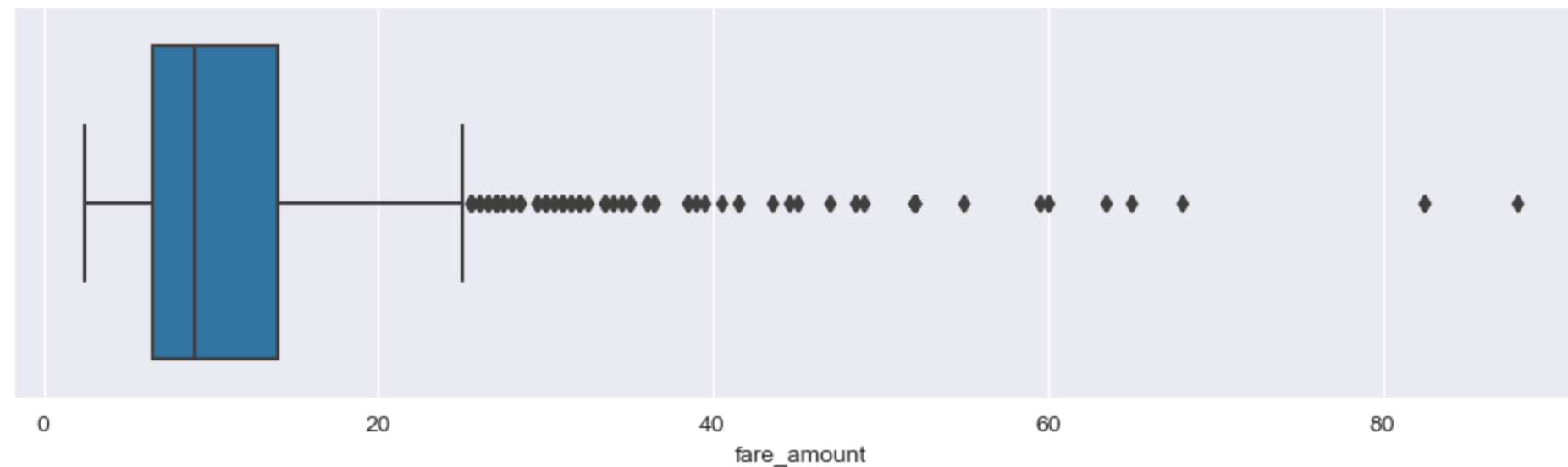
Univariate Distributions: Boxplot

```
In [119]: 1 fig,ax = plt.subplots(1,1,figsize=(12,3))  
          2  
          3 sns.boxplot(x='fare_amount',data=df_taxi,ax=ax);
```



Univariate Distributions: Boxplot

```
In [119]: 1 fig,ax = plt.subplots(1,1,figsize=(12,3))  
2  
3 sns.boxplot(x='fare_amount',data=df_taxi,ax=ax);
```

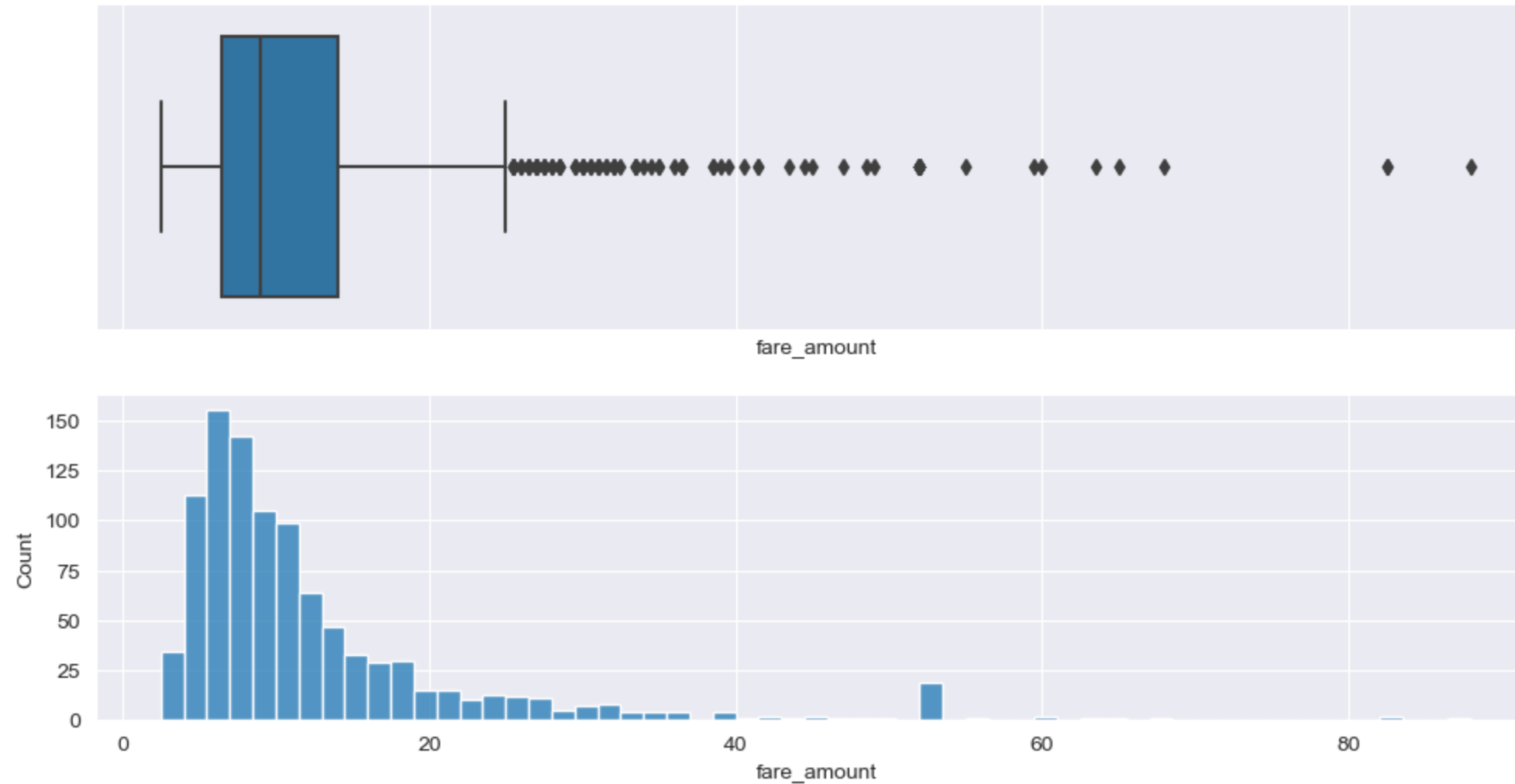


- first quartile
- second quartile (Median)
- third quartile
- whiskers (usually $1.5 \times \text{IQR}$)
- outliers

Seaborn: Combining Plots with Subplots

Seaborn: Combining Plots with Subplots

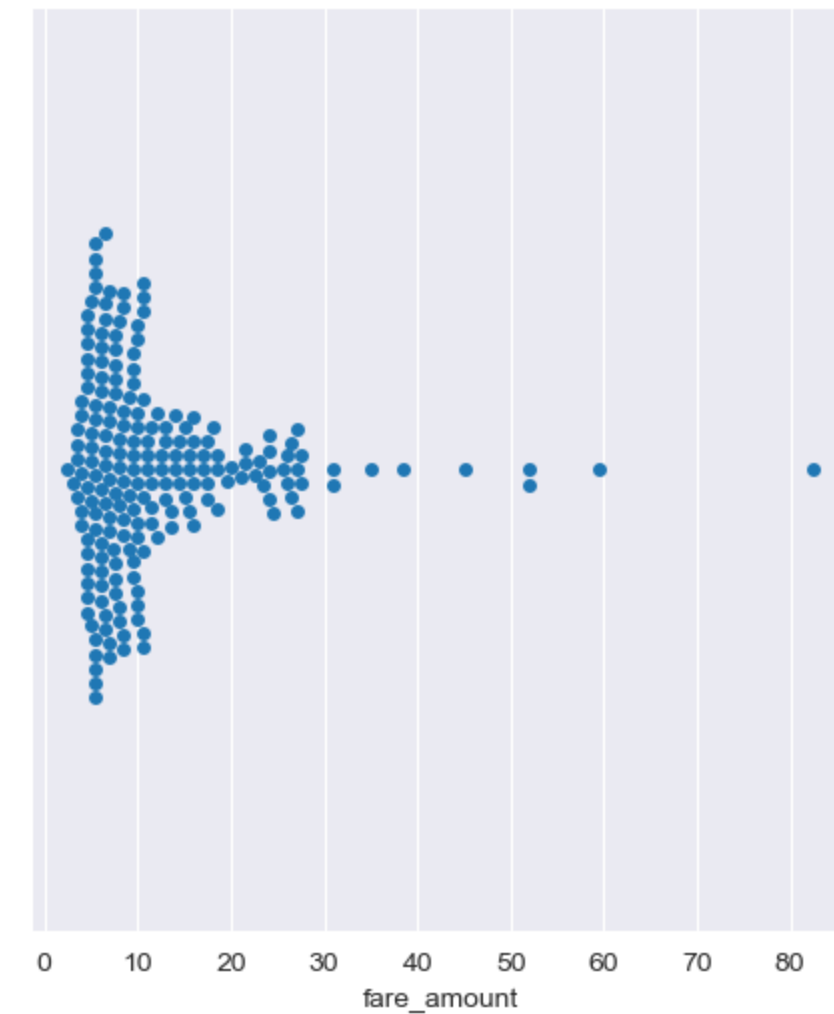
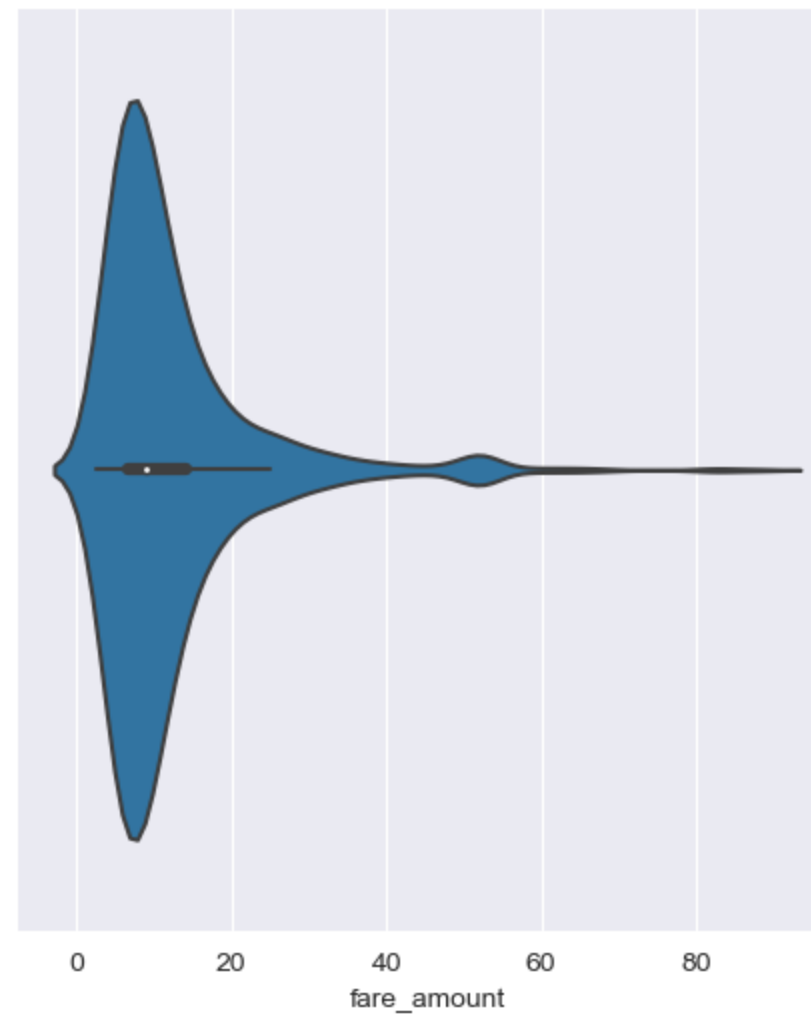
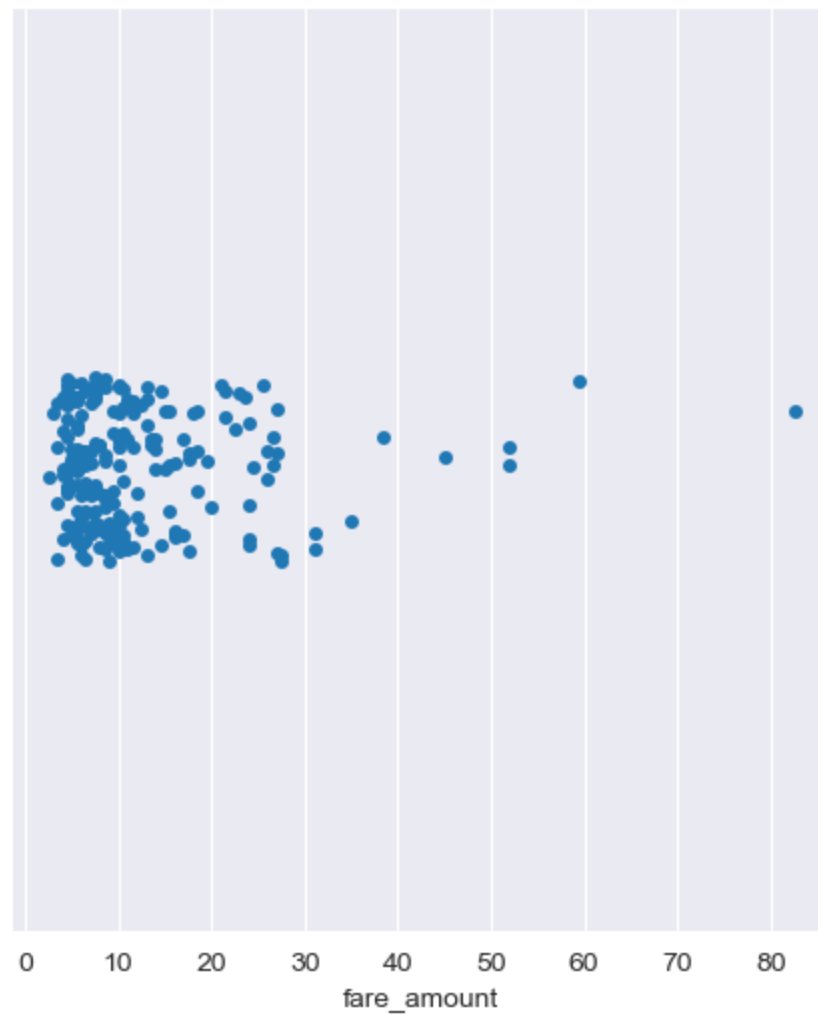
```
In [120]: 1 fig,ax = plt.subplots(2,1,figsize=(12,6), sharex=True)
          2
          3 sns.boxplot(x='fare_amount', data=df_taxi, ax=ax[0]);
          4 sns.histplot(x='fare_amount', data=df_taxi, ax=ax[1]);
```



Other Univariate Distribution Visualizations

Other Univariate Distribution Visualizations

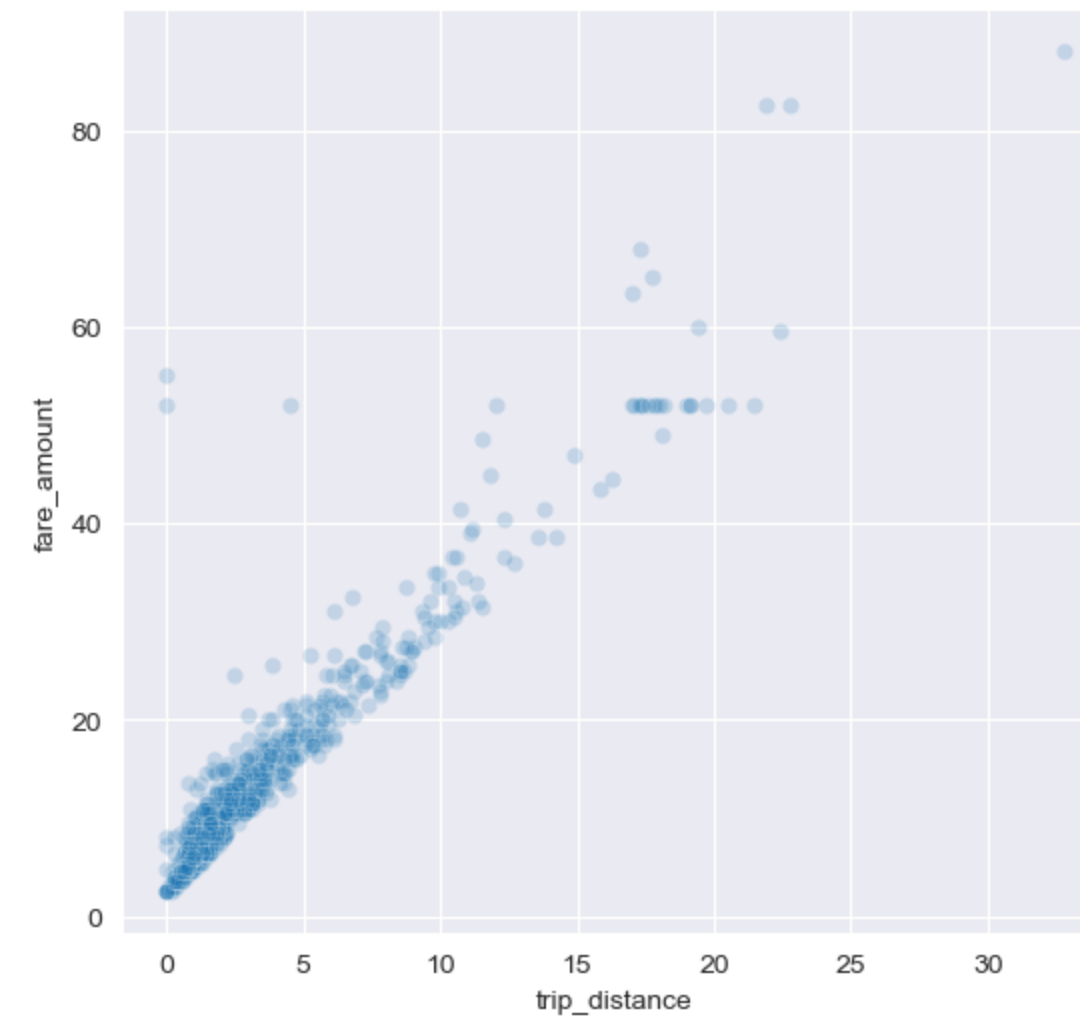
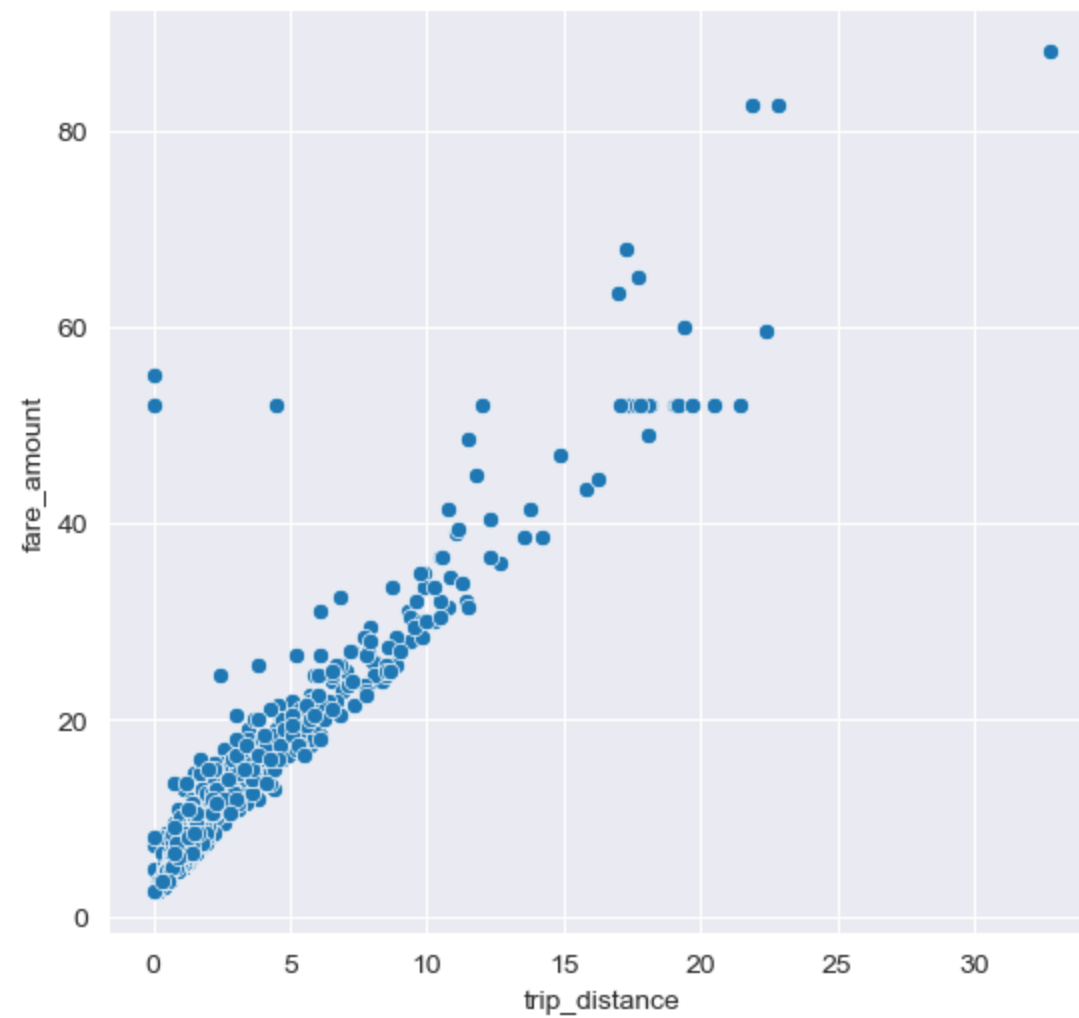
```
In [121]: 1 fig,ax = plt.subplots(1,3,figsize=(18,6))  
2  
3 sns.stripplot(x='fare_amount',data=df_taxi[:200],ax=ax[0])  
4 sns.violinplot(x='fare_amount',data=df_taxi,ax=ax[1])  
5 sns.swarmplot(x='fare_amount',data=df_taxi[:200],ax=ax[2]);
```



Bivariate: Scatterplot (with alpha)

Bivariate: Scatterplot (with alpha)

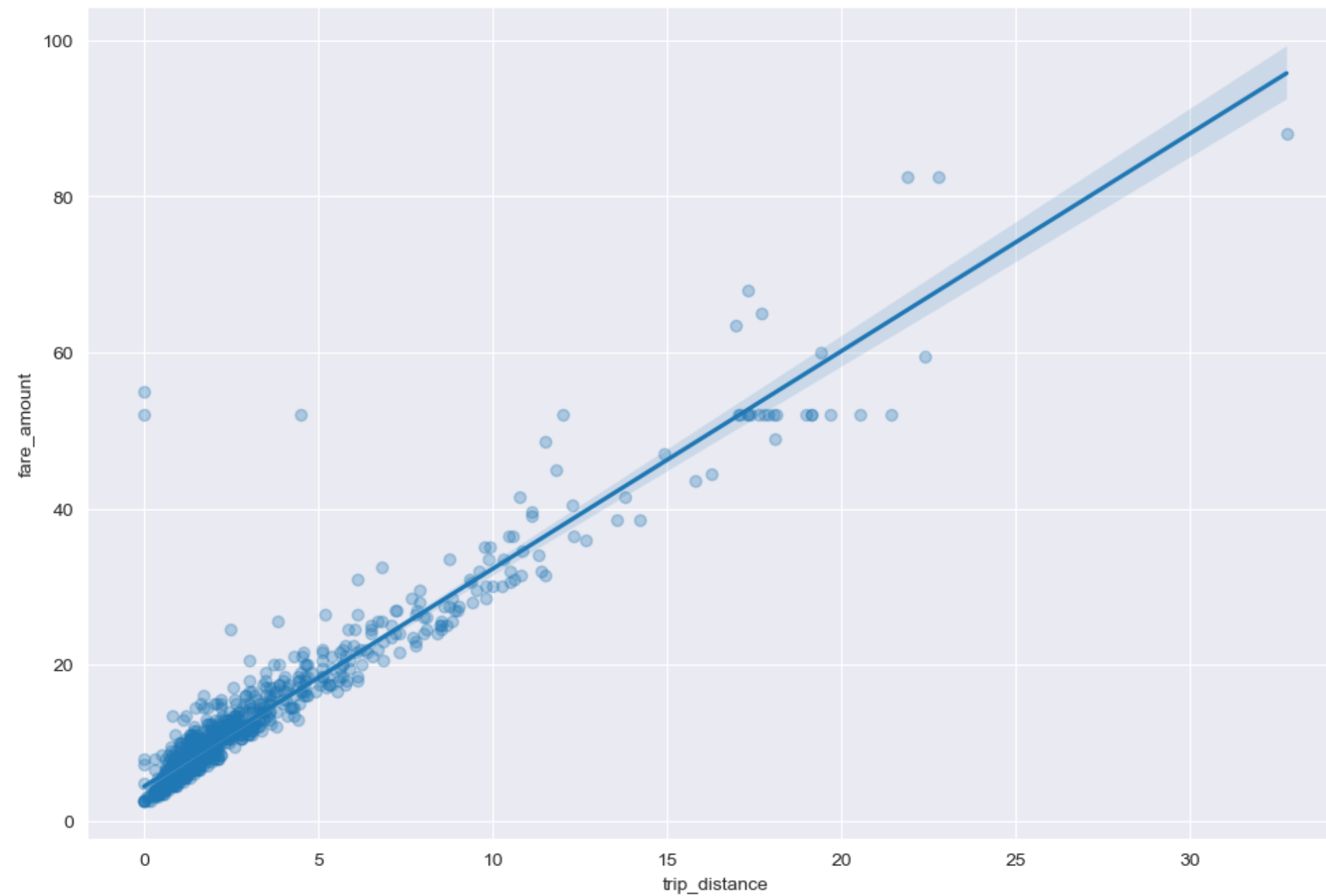
```
In [122]: 1 fig,ax = plt.subplots(1,2,figsize=(14,6))
          2 sns.scatterplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax[0]);
          3 sns.scatterplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax[1], alpha=0.2);
```



Bivariate: Add Regression Line

Bivariate: Add Regression Line

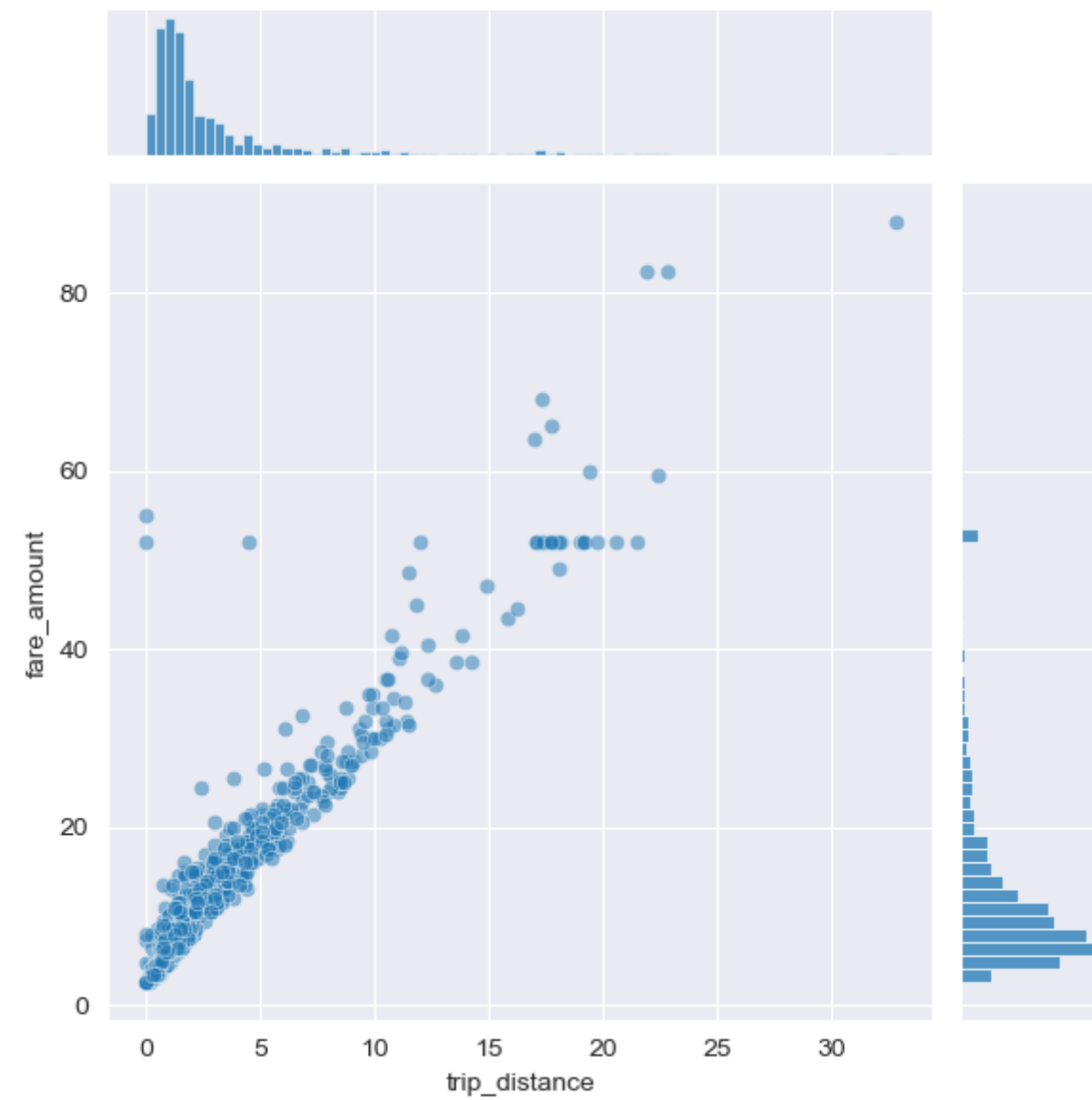
```
In [123]: 1 fig,ax = plt.subplots(1,1,figsize=(12,8))  
2  
3 sns.regplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax, scatter_kws={'alpha':0.3});
```



Bivariate: Joint Plot

Bivariate: Joint Plot

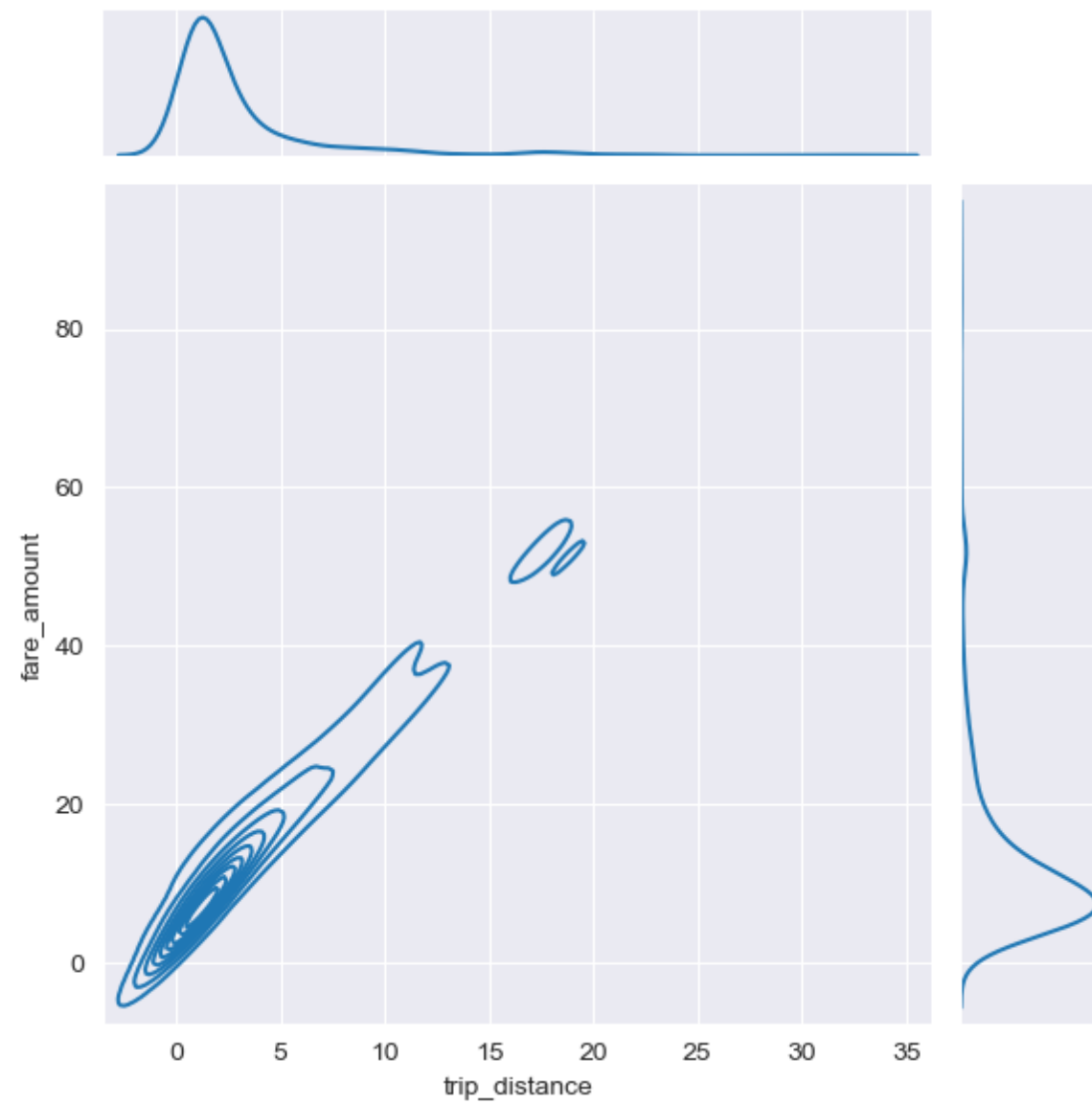
```
In [124]: 1 sns.jointplot(x='trip_distance',y='fare_amount',data=df_taxi,alpha=0.5);
```



Bivariate: Joint Plot with KDE

Bivariate: Joint Plot with KDE

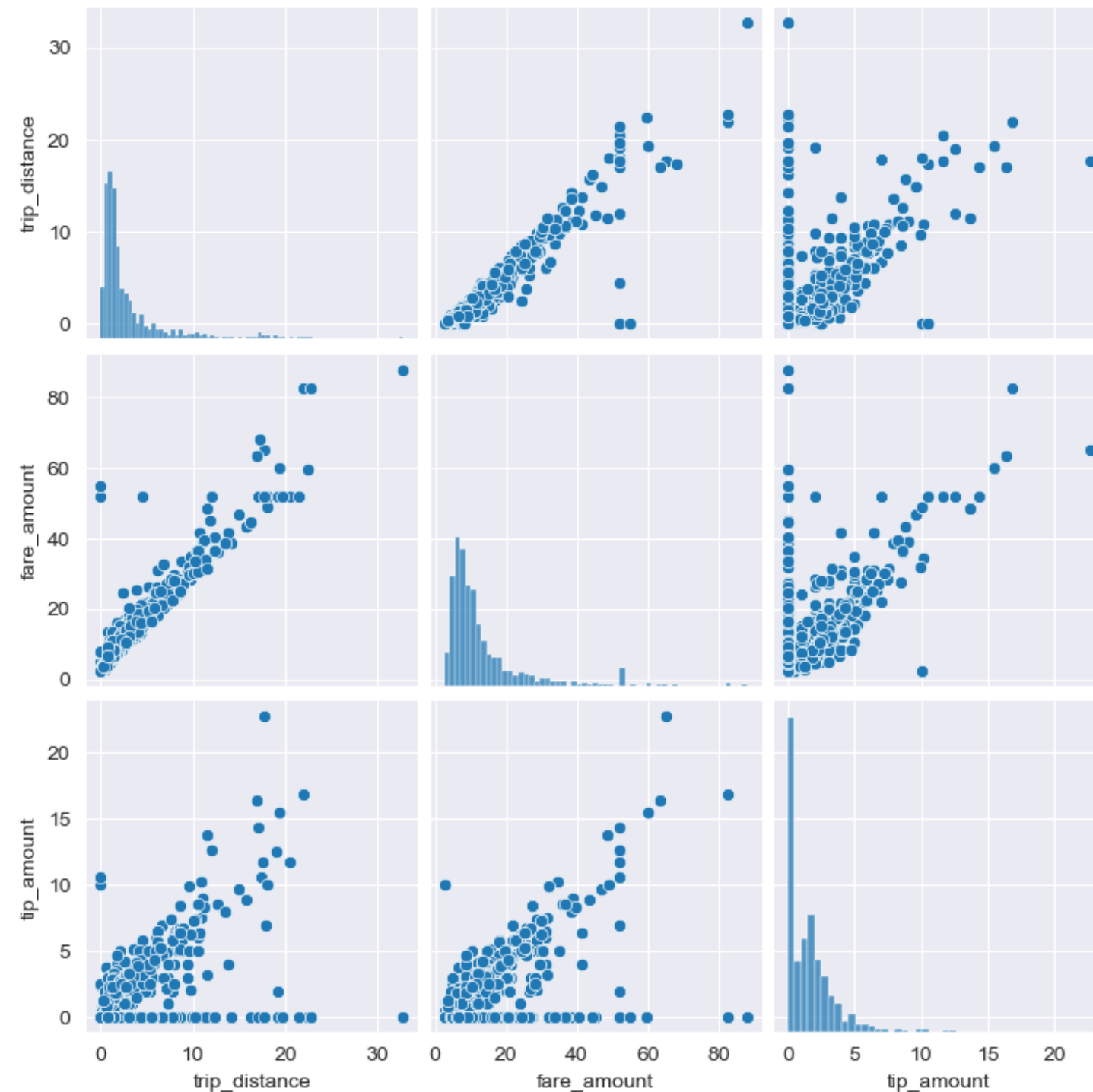
```
In [125]: 1 sns.jointplot(x='trip_distance', y='fare_amount',  
2                   data=df_taxi,  
3                   kind='kde');
```



Comparing Multiple Variables with `pairplot`

Comparing Multiple Variables with `pairplot`

```
In [126]: 1 sns.pairplot(data=df_taxi[['trip_distance', 'fare_amount', 'tip_amount']]);
```



Categorical Variables: Frequency

Categorical Variables: Frequency

```
In [127]: 1 df_taxi.payment_type.value_counts()
```

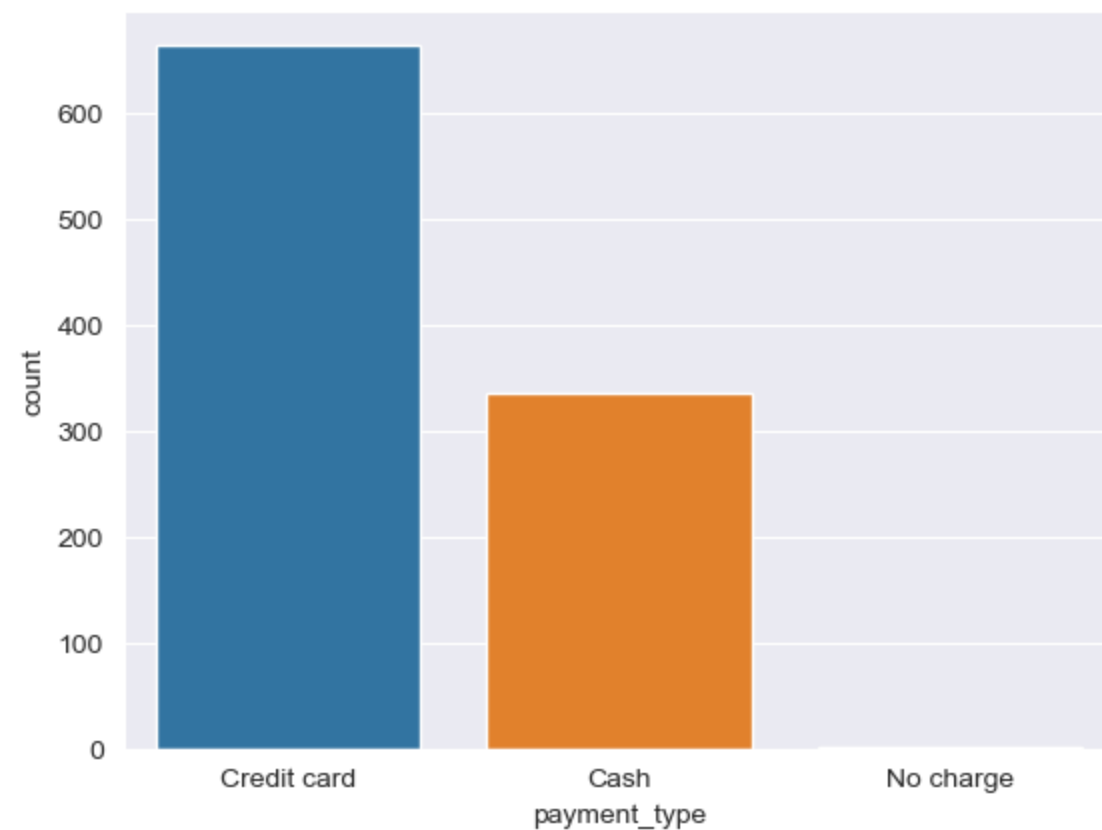
```
Out[127]: Credit card    663  
Cash                335  
No charge            2  
Name: payment_type, dtype: int64
```

Categorical Variables: Frequency

```
In [127]: 1 df_taxi.payment_type.value_counts()
```

```
Out[127]: Credit card    663  
Cash                335  
No charge            2  
Name: payment_type, dtype: int64
```

```
In [128]: 1 sns.countplot(x='payment_type', data=df_taxi);
```



Plotting Numeric and Categorical

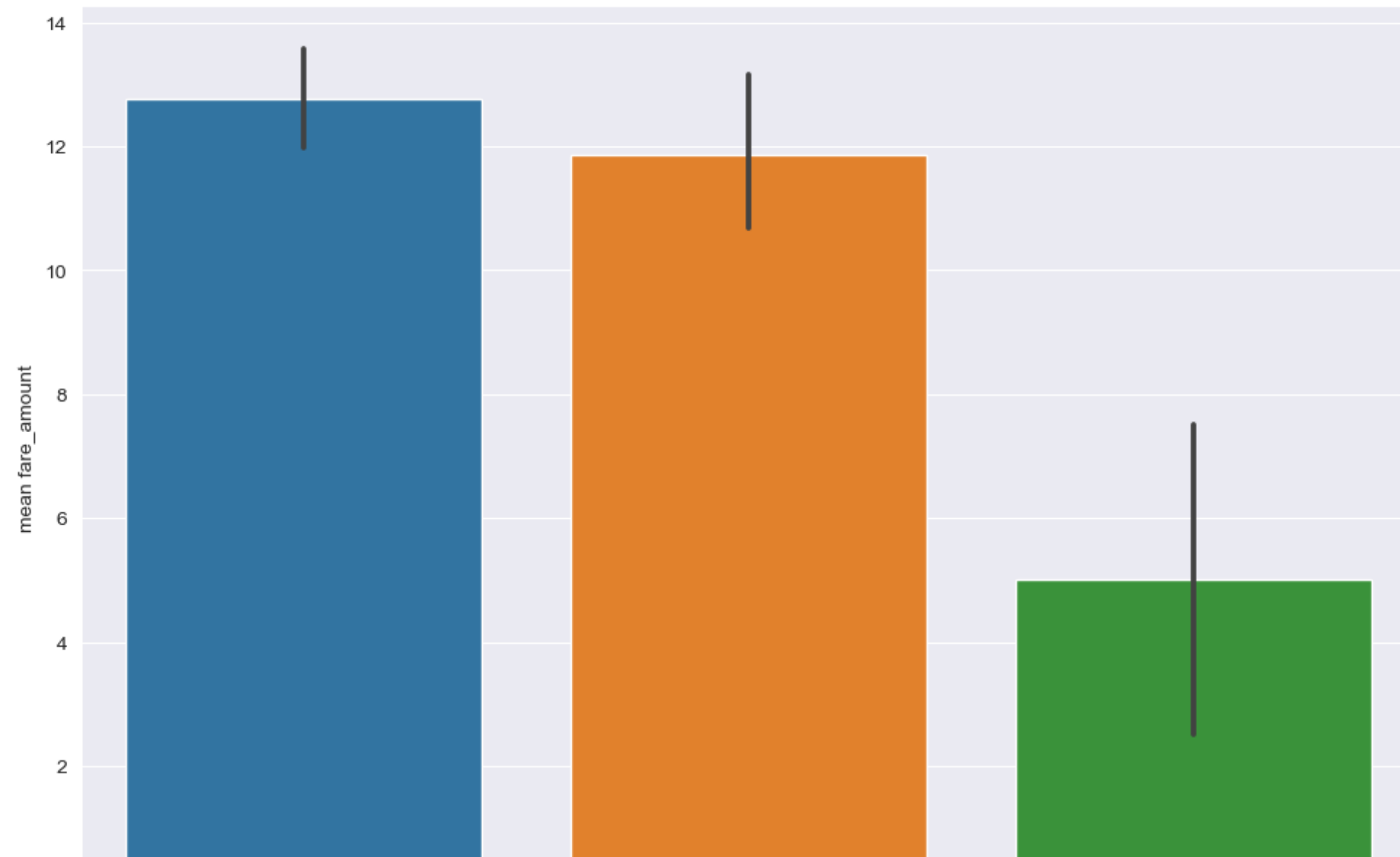
Plotting Numeric and Categorical

```
In [129]: 1 fig,ax = plt.subplots(1,1,figsize=(12,8))
          2
          3 sns.barplot(x='payment_type',y='fare_amount',data=df_taxi,estimator=np.mean,ci=95);
          4 ax.set_ylabel('mean fare_amount');
```

/var/folders/78/vhnqkq8n45dd4gj4f5qx8yb00000gn/T/ipykernel_46425/1040376770.py:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.

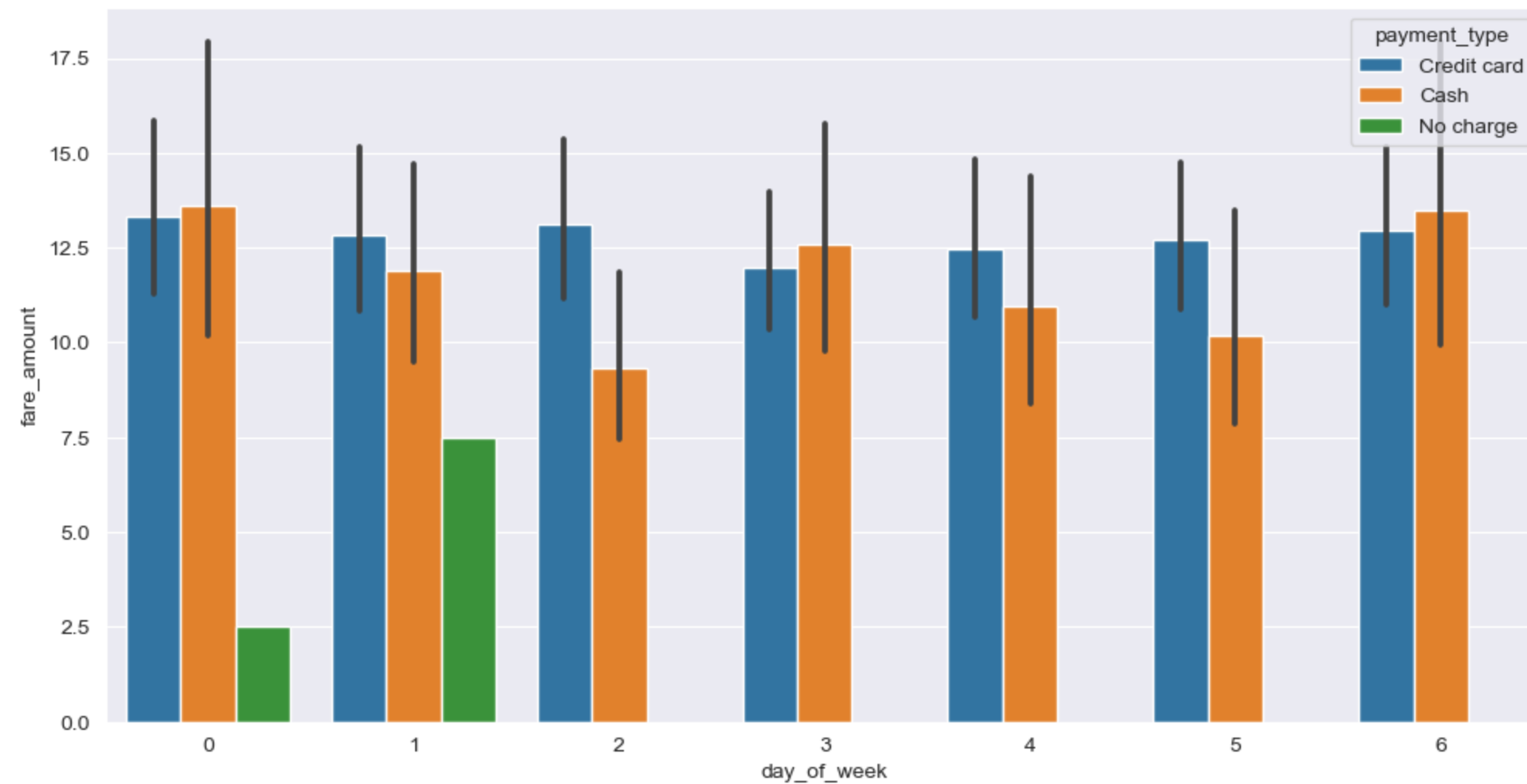
```
sns.barplot(x='payment_type',y='fare_amount',data=df_taxi,estimator=np.mean,ci=95);
```



Plotting with Hue

Plotting with Hue

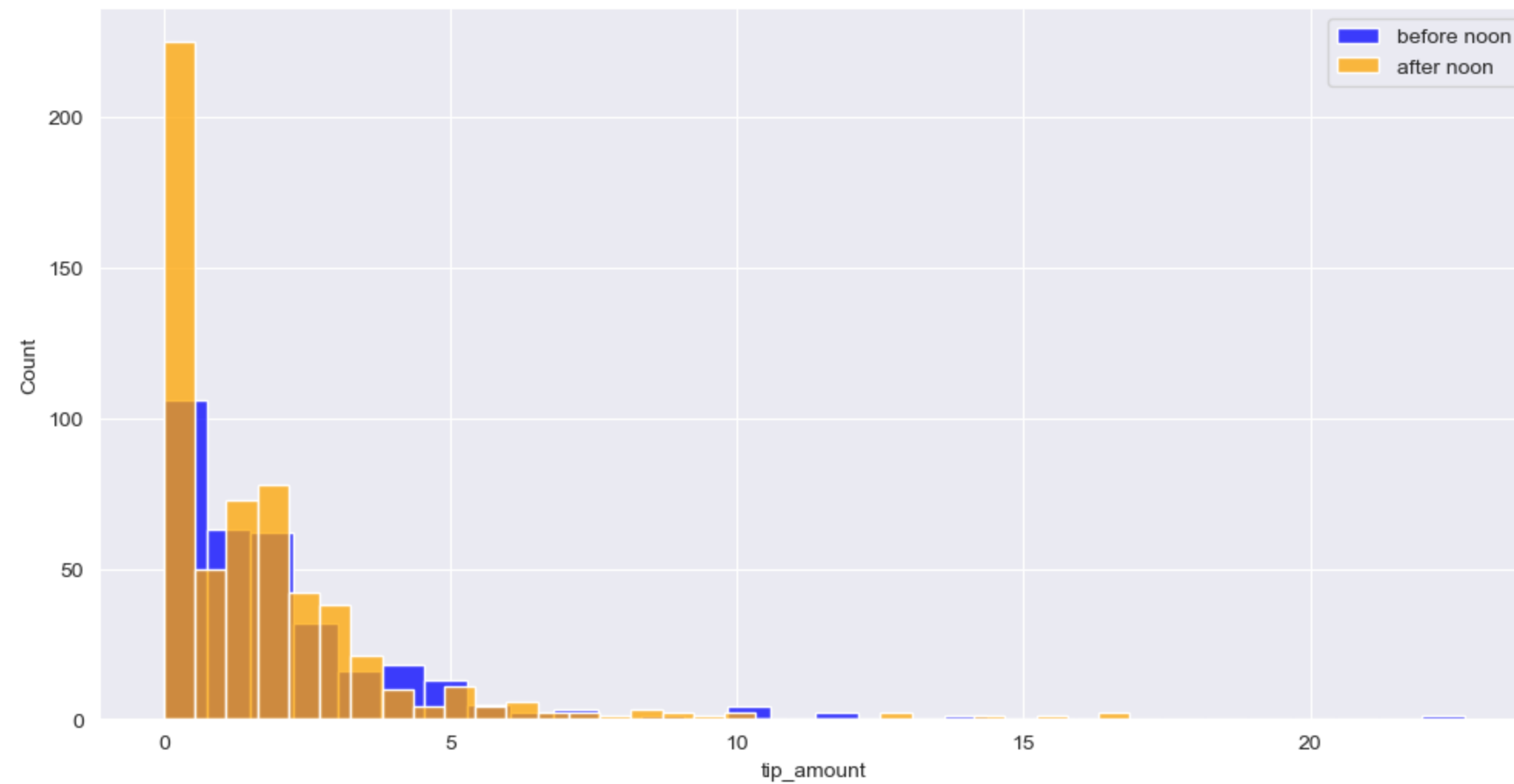
```
In [130]: 1 fig,ax = plt.subplots(1,1,figsize=(12,6))
          2
          3 # add a second categorical variable day_of_week
          4 sns.barplot(x='day_of_week',
          5               y='fare_amount',
          6               hue='payment_type',
          7               data=df_taxi,
          8               ax=ax,
          9               );
```



Same Axis, Multiple Plots with Seaborn (with legend)

Same Axis, Multiple Plots with Seaborn (with legend)

```
In [131]: 1 fig,ax = plt.subplots(1,1,figsize=(12,6))
2 sns.histplot(x='tip_amount',data=df_taxi[df_taxi.pickup_datetime.dt.hour < 12], label='before noon',color='blue', ax=ax);
3 sns.histplot(x='tip_amount',data=df_taxi[df_taxi.pickup_datetime.dt.hour >= 12], label='after noon', color='orange',ax=ax);
4 plt.legend(loc='best');
```



Data Exploration and Viz Review

- central tendencies: mean, median
- spread: variance, std deviation, IQR
- correlation: pearson correlation coefficient
- plotting with Matplotlib and Seaborn
- plotting real valued variables: histogram, scatter, regplot
- plotting categorical variables: count, bar
- plotting interactions: jointplot, pairplot

Where to go from here

- Additional Dataframe styling with `.style()` (https://pandas.pydata.org/docs/user_guide/style.html)
- Seaborn Figure-level plots: `relplot`, `displot`, `catplot`
(https://seaborn.pydata.org/tutorial/function_overview.html)
- Interactive visuals with plotly (<https://plotly.com/python/plotly-fundamentals/>)

Questions?