*Elements Of Data Science - F2025*

# Week 3: Pandas, Data Exploration and Visualization

9/16/2025

# 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**

**Berkeley Source Distribution (BSD)** licenses are used for the distribution of many freeware, shareware and open source software.

In [35]:

```
# usually imported using the alias 'pd'
import pandas as pd
```

In [35]:

```
# usually imported using the alias 'pd'
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]:
```
s = pd.Series(np.random.rand(4), index= ['j','k','m','n'])
s
```

Out[3]:
```
j    0.882688
k    0.795942
m    0.060938
n    0.821973
dtype: float64
```

# Pandas Series

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

In [3]:
```
s = pd.Series(np.random.rand(4), index= ['j','k','m','n'])
s
```

Out[3]:
```
j    0.882688
k    0.795942
m    0.060938
n    0.821973
dtype: float64
```

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

Out[4]:
```
array([0.88268843, 0.79594199, 0.06093834, 0.82197277])
```

# Pandas Series

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

In [3]:
```
s = pd.Series(np.random.rand(4), index= ['j','k','m','n'])
s
```

Out[3]:
```
j    0.882688
k    0.795942
m    0.060938
n    0.821973
dtype: float64
```

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

Out[4]:
```
array([0.88268843, 0.79594199, 0.06093834, 0.82197277])
```

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

Out[5]:
```
Index(['j', 'k', 'm', 'n'], dtype='object')
```

# 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 [36]:
```
# create Series from array and set index
s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
s1
```

Out[36]:
```
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 [36]:
```
# create Series from array and set index
s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
s1
```

Out[36]:
```
house_a    1.0
2          2.0
house c    3.0
Name: NumRooms, dtype: float64
```

In [39]:
```
s1[2] # access a single value via index label
```

Out[39]:
```
2.0
```

# Pandas Series Cont.

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

In [36]:
```
# create Series from array and set index
s1 = pd.Series([1,2,3],index=['house_a',2,'house c'],name='NumRooms',dtype=float)
s1
```

Out[36]:
```
house_a    1.0
2          2.0
house c    3.0
Name: NumRooms, dtype: float64
```

In [39]:
```
s1[2] # access a single value via index label
```

Out[39]:
```
2.0
```

In [8]:
```
s1["house c"] # dot notation (How do we get "house c"?)
```

Out[8]:
```
3.0
```

# Pandas Series Cont.

# Pandas Series Cont.

- accessing other Series attributes

In [9]:
```
1 s1
```

Out[9]:
```
house_a    1.0
2          2.0
house c    3.0
Name: NumRooms, dtype: float64
```

# Pandas Series Cont.

- accessing other Series attributes

In [9]:
```
s1
```

Out[9]:
```
house_a    1.0
2          2.0
house c    3.0
Name: NumRooms, dtype: float64
```

In [10]:
```
#print(f'{s.index  = :}')
#print(f'{s.values = :}')
print(f'{s1.name   = :}')
print(f'{s1.dtype  = :}')
print(f'{s1.shape  = :}')
```

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

# Pandas Series Cont.

# Pandas Series Cont.

In [11]:
```
# Can create series with index from a dictionary
s2 = pd.Series({'a':1,'b':2,'c':3,'d':4})
s2
```

Out[11]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [12]:
```
print(s2.name)
```

```
None
```

# Pandas Series Cont.

In [11]:
```
# Can create series with index from a dictionary
s2 = pd.Series({'a':1,'b':2,'c':3,'d':4})
s2
```

Out[11]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [12]:
```
print(s2.name)
```

```
None
```

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

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

In [40]:
```
s2.values
```

Out[40]:
```
array([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 [42]:

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

# Pandas DataFrame Cont.

In [42]:
```
df = pd.DataFrame({'Year':[2017,2018,2018,2019],
                   'Semester':['Fall','Fall','Spring','Fall'],
                   'Measure_1':[2.1,3.0,2.4,1.9]
                  })
```

In [43]:
```
df
```

Out[43]:

| | 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 [42]:
```
df = pd.DataFrame({'Year':[2017,2018,2018,2019],
                   'Semester':['Fall','Fall','Spring','Fall'],
                   'Measure_1':[2.1,3.0,2.4,1.9]
                  })
```

In [43]:
```
df
```

Out[43]:

| | 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 [44]:
```
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 [42]:
```
df = pd.DataFrame({'Year':[2017,2018,2018,2019],
                   'Semester':['Fall','Fall','Spring','Fall'],
                   'Measure_1':[2.1,3.0,2.4,1.9]
                  })
```

In [43]:
```
df
```

Out[43]:

| | 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 [44]:
```
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 [45]:
```
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 [46]:

```
data = np.array([[2017,'Fall',2.1],
        [2018,'Fall',3.0],
        [2018,'Spring',2.4],
        [2019,'Fall',1.9]])
```

# Pandas DataFrame Cont.

In [46]:
```
data = np.array([[2017,'Fall',2.1],
        [2018,'Fall',3.0],
        [2018,'Spring',2.4],
        [2019,'Fall',1.9]])
```

In [47]:
```
df = pd.DataFrame(data,
                  columns=['Year','Semester','Measure_1'],
                  index=['001','002','003','004'])
df.shape
```

Out[47]:
```
(4, 3)
```

# Pandas DataFrame Cont.

In [46]:
```
data = np.array([[2017,'Fall',2.1],
        [2018,'Fall',3.0],
        [2018,'Spring',2.4],
        [2019,'Fall',1.9]])
```

In [47]:
```
df = pd.DataFrame(data,
                  columns=['Year','Semester','Measure_1'],
                  index=['001','002','003','004'])
df.shape
```

Out[47]: (4, 3)

In [48]:
```
df
```

Out[48]:

| | 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 [49]: 1 df.index
```

```
Out[49]: 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 [49]:
1 df.index
```

```
Out[49]: 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 [49]:
```
1 df.index
```

Out[49]:
```
Index(['001', '002', '003', '004'], dtype='object')
```

- Get column values : `columns`

In [50]:
```
1 df.columns
```

Out[50]:
```
Index(['Year', 'Semester', 'Measure_1'], dtype='object')
```

# Pandas Indexing/Selection

In [24]:
```
1 df
```

Out[24]:

| | 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 Indexing/Selection

In [24]:
```
1 df
```

Out[24]:

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

Select by label:

- `.loc[]`

# Pandas Indexing/Selection

In [24]:
```
1 df
```

Out[24]:

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

Select by label:

- `.loc[]`

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

Out[25]:
```
Year          2017
Semester      Fall
Measure_1      2.1
Name: 001, dtype: object
```

# Pandas Indexing/Selection

In [24]:
```
1 df
```

Out[24]:

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

Select by label:

- `.loc[]`

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

Out[25]:
```
Year          2017
Semester      Fall
Measure_1      2.1
Name: 001, dtype: object
```

In [26]:
```
1 df.loc['001','Measure_1']
```

Out[26]:
```
2.1
```

# Pandas Indexing/Selection Cont.

# Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

In [27]:
```
1 df
```

Out[27]:

| | 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 Indexing/Selection Cont.

Select by position:

- `.iloc[]`

In [27]:
```
1 df
```

Out[27]:

| | 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 [28]:
```
1 df.iloc[0]
```

Out[28]:
```
Year          2017
Semester      Fall
Measure_1      2.1
Name: 001, dtype: object
```

# Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

In [27]:
```
1 df
```

Out[27]:

| | 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 [28]:
```
1 df.iloc[0]
```

Out[28]:
```
Year          2017
Semester      Fall
Measure_1      2.1
Name: 001, dtype: object
```

In [29]:
```
1 df.iloc[0,2]
```

Out[29]:
```
2.1
```

# Pandas Indexing/Selection Cont.

# Pandas Indexing/Selection Cont.

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

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

Out[30]:

| | 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 Indexing/Selection Cont.

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

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

Out[30]:

| | 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 [52]:
```
1 df.loc[['002','004'],:]
```

Out[52]:

| | 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 [30]:
```
1 df
```

Out[30]:

| | 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 [52]:
```
1 df.loc[['002','004'],:]
```

Out[52]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **002** | 2018 | Fall | 3.0 |
| **004** | 2019 | Fall | 1.9 |

In [32]:
```
1 df.loc[['002','004'],['Year','Measure_1']]
```

Out[32]:

| | Year | Measure_1 |
|---|---|---|
| **002** | 2018 | 3.0 |
| **004** | 2019 | 1.9 |

# Pandas Slicing

# Pandas Slicing

In [53]:
```
# Get last two rows
df.iloc[-2:]
```

Out[53]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **003** | 2018 | Spring | 2.4 |
| **004** | 2019 | Fall | 1.9 |

# Pandas Slicing

In [53]:
```
# Get last two rows
df.iloc[-2:]
```

Out[53]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **003** | 2018 | Spring | 2.4 |
| **004** | 2019 | Fall | 1.9 |

In [55]:
```
# Get first two rows and first two columns
df.iloc[:2,:2]
```

Out[55]:

| | Year | Semester |
|---|---|---|
| **001** | 2017 | Fall |
| **002** | 2018 | Fall |

In [56]:
```
df
```

Out[56]:

| | 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 Slicing

In [53]:
```
# Get last two rows
df.iloc[-2:]
```

Out[53]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **003** | 2018 | Spring | 2.4 |
| **004** | 2019 | Fall | 1.9 |

In [55]:
```
# Get first two rows and first two columns
df.iloc[:2,:2]
```

Out[55]:

| | Year | Semester |
|---|---|---|
| **001** | 2017 | Fall |
| **002** | 2018 | Fall |

In [56]:
```
df
```

Out[56]:

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

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

# Pandas Slicing Cont.

# Pandas Slicing Cont.

Can also slice using labels:

In [57]:
```
1 df
```

Out[57]:

| | 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 Slicing Cont.

Can also slice using labels:

In [57]:
```
1 df
```

Out[57]:

| | 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 [58]:
```
1 df.loc['002':'004']
```

Out[58]:

| | 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 [57]:
```
1 df
```

Out[57]:

| | 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 [58]:
```
1 df.loc['002':'004']
```

Out[58]:

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

In [59]:
```
1 df.loc['002':'004','Semester':]
```

Out[59]:

| | Semester | Measure_1 |
|---|---|---|
| **002** | Fall | 3.0 |
| **003** | Spring | 2.4 |
| **004** | Fall | 1.9 |

# Pandas Slicing Cont.

Can also slice using labels:

In [57]:
```
1 df
```

Out[57]:

| | 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 [58]:
```
1 df.loc['002':'004']
```

Out[58]:

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

In [59]:
```
1 df.loc['002':'004','Semester':]
```

Out[59]:

| | Semester | Measure_1 |
|---|---|---|
| **002** | Fall | 3.0 |
| **003** | Spring | 2.4 |
| **004** | 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 [164]:
```
1 df.loc[:,'Measure_1']
```

Out[164]:
```
001    2.1
002    3.0
003    2.4
004    1.9
Name: Measure_1, dtype: object
```

# Pandas Slicing Cont.

How to indicate all rows or all columns? `:`

In [164]:
```
1 df.loc[:,'Measure_1']
```

Out[164]:
```
001    2.1
002    3.0
003    2.4
004    1.9
Name: Measure_1, dtype: object
```

In [61]:
```
1 df.iloc[2:,:]
```

Out[61]:

| | 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 [62]:
```
1 df['Semester']
```

Out[62]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

# Pandas Indexing Cont.

Shortcut for indexing:

In [62]:
```
df['Semester']
```

Out[62]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

In [63]:
```
# can use dot notation if there is no space in label
df.Semester
```

Out[63]:
```
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:

In [165]:
```
1 df
```

Out[165]:

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

# Panda Selection Chaining

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

In [165]:
```
1 df
```

Out[165]:

| | 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 [167]:
```
1 df.iloc[:3].loc[:,['Year','Measure_1']]
```

Out[167]:

| | 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 [165]:
```
1 df
```

Out[165]:

| | 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 [167]:
```
1 df.iloc[:3].loc[:,['Year','Measure_1']]
```

Out[167]:

| | 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 [165]:
```
1 df
```

Out[165]:

| | 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 [167]:
```
1 df.iloc[:3].loc[:,['Year','Measure_1']]
```

Out[167]:

| | 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 [65]:
```
1 df.loc[['001','003']].iloc[:,-2:]
```

Out[65]:

| | 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 [66]:
```
# reduce the amount of error information printed
%xmode Minimal
```

```
Exception reporting mode: Minimal
```

# Pandas Selection Chaining Cont.

For record '002' get last two columns?:

In [66]:
```
# reduce the amount of error information printed
%xmode Minimal
```

```
Exception reporting mode: Minimal
```

In [176]:
```
# Note: add 'raises-exception' tag to cell to continue running after exception

df.loc['002']#.iloc[:,:-1] # row with label '002', then all rows, last two columns?
```

Out[176]:
```
Year          2018
Semester      Fall
Measure_1      3.0
Name: 002, dtype: object
```

# Pandas Selection Chaining Cont.

For record '002' get last two columns?:

In [66]:
```
# reduce the amount of error information printed
%xmode Minimal
```

```
Exception reporting mode: Minimal
```

In [176]:
```
# Note: add 'raises-exception' tag to cell to continue running after exception

df.loc['002']#.iloc[:,:-1] # row with label '002', then all rows, last two columns?
```

Out[176]:
```
Year         2018
Semester     Fall
Measure_1     3.0
Name: 002, dtype: object
```

In [177]:
```
df.loc['002'].iloc[-2:] # row with label '002', last two elements of Series
```

Out[177]:
```
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 [69]:
```
1 df.head() # first 5 rows by default
```

Out[69]:

| | 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 [69]:
```
1 df.head() # first 5 rows by default
```

Out[69]:

| | 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 [70]:
```
1 df.tail(2) # only print last 2 rows
```

Out[70]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **003** | 2018 | Spring | 2.4 |
| **004** | 2019 | Fall | 1.9 |

# Pandas Boolean Mask

In [71]:
```
1 df.loc[:,'Semester']
```

Out[71]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

# Pandas Boolean Mask

In [71]:
```
df.loc[:,'Semester']
```

Out[71]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

In [72]:
```
# Which rows have Semester of 'Fall'?
df.loc[:,'Semester'] == 'Fall'
```

Out[72]:
```
001     True
002     True
003    False
004     True
Name: Semester, dtype: bool
```

# Pandas Boolean Mask

In [71]:
```
df.loc[:,'Semester']
```

Out[71]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

In [72]:
```
# Which rows have Semester of 'Fall'?
df.loc[:,'Semester'] == 'Fall'
```

Out[72]:
```
001     True
002     True
003    False
004     True
Name: Semester, dtype: bool
```

In [73]:
```
# Get all data for rows with with Semester 'Fall'
df[df['Semester'] == 'Fall']
```

Out[73]:

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

# Pandas Boolean Mask

In [71]:
```
df.loc[:,'Semester']
```

Out[71]:
```
001      Fall
002      Fall
003    Spring
004      Fall
Name: Semester, dtype: object
```

In [72]:
```
# Which rows have Semester of 'Fall'?
df.loc[:,'Semester'] == 'Fall'
```

Out[72]:
```
001     True
002     True
003    False
004     True
Name: Semester, dtype: bool
```

In [73]:
```
# Get all data for rows with with Semester 'Fall'
df[df['Semester'] == 'Fall']
```

Out[73]:

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

In [74]:
```
# Get Measure_1 for all records for Semester 'Fall'
df.loc[df.Semester == 'Fall',['Year','Measure_1']]
```

Out[74]:

| | Year | Measure_1 |
|---|---|---|
| **001** | 2017 | 2.1 |
| **002** | 2018 | 3.0 |

# Pandas Boolean Mask Cont.

# Pandas Boolean Mask Cont.

Get all records Fall Semester prior to 2019

In [183]:
```
1 df.loc[df.Semester =='Fall'].loc[df.Year<'2019']
```

Out[183]:

| | 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 [183]:
```
df.loc[df.Semester =='Fall'].loc[df.Year<'2019']
```

Out[183]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **001** | 2017 | Fall | 2.1 |
| **002** | 2018 | Fall | 3.0 |

In [184]:
```
# make sure to use parentheses with comparisons!
df.loc[(df.Semester == 'Fall') & (df.Year < '2019') ]
```

Out[184]:

| | 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 [183]:
```
df.loc[df.Semester =='Fall'].loc[df.Year<'2019']
```

Out[183]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **001** | 2017 | Fall | 2.1 |
| **002** | 2018 | Fall | 3.0 |

In [184]:
```
# make sure to use parentheses with comparisons!
df.loc[(df.Semester == 'Fall') & (df.Year < '2019') ]
```

Out[184]:

| | Year | Semester | Measure_1 |
|---|---|---|---|
| **001** | 2017 | Fall | 2.1 |
| **002** | 2018 | Fall | 3.0 |

In [188]:
```
# or use comparison functions: .eq, .ne, .gt, .ge, .lt, .le
df.loc[df.Semester.eq('Fall') & df.Year.lt('2019')]
```

Out[188]:

| | 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`:

In [198]:
```
1 [str(l) for l in range(2017, 2020)]
```

Out[198]:
```
['2017', '2018', '2019']
```

# Pandas Boolean Mask Cont.

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

In [198]:
```
1 [str(l) for l in range(2017, 2020)]
```

Out[198]:
```
['2017', '2018', '2019']
```

In [201]:
```
1 df.loc[df.Year.isin(np.array([str(l) for l in range(2017, 2019)]))]
```

Out[201]:

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

# Pandas Selection Review

# Pandas Selection Review

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

# Pandas Sorting

# Pandas Sorting

In [204]:
```
1 df.sort_values(by=['Measure_1']).head(3)
```

Out[204]:

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

# Pandas Sorting

In [204]:
```
1 df.sort_values(by=['Measure_1']).head(3)
```

Out[204]:

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

In [80]:
```
1 df.sort_values(by=['Measure_1'],ascending=False).head(3)
```

Out[80]:

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

# Pandas Sorting

In [204]:
```
1 df.sort_values(by=['Measure_1']).head(3)
```

Out[204]:

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

In [80]:
```
1 df.sort_values(by=['Measure_1'],ascending=False).head(3)
```

Out[80]:

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

In [81]:
```
1 df_sorted_top3 = df.sort_values(by=['Semester','Measure_1'], ascending=False).head(3)
```

Out[81]:

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

# 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 [82]:
```
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 [208]:

```
import pandas as pd
df_taxi = pd.read_csv('../data/yellowcab_demo_withdaycategories.csv',
                      sep=',',
                      header=1,
                      parse_dates= ['pickup_datetime','dropoff_datetime'],
                     )

```

# Loading Datasets with Pandas

In [208]:
```
import pandas as pd
df_taxi = pd.read_csv('../data/yellowcab_demo_withdaycategories.csv',
                      sep=',',
                      header=1,
                      parse_dates= ['pickup_datetime','dropoff_datetime'],
                     )

```

In [210]:
```
# display first 5 rows
df_taxi.head(5)
```

Out[210]:

| | 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 [85]:
```
1 df_taxi.shape
```

Out[85]: (1000, 8)

# Get Size of Dataset

In [85]:
```
1 df_taxi.shape
```

Out[85]: (1000, 8)

In [86]:
```
1 # number of rows
2 f'{df_taxi.shape[0]} rows'
```

Out[86]: '1000 rows'

# Get Size of Dataset

In [85]:
```
1 df_taxi.shape
```

Out[85]: (1000, 8)

In [86]:
```
1 # number of rows
2 f'{df_taxi.shape[0]} rows'
```

Out[86]: '1000 rows'

In [87]:
```
1 # number of columns
2 f'{df_taxi.shape[1]} columns'
```

Out[87]: '8 columns'

# Get Size of Dataset

In [85]:
```
df_taxi.shape
```

Out[85]:
```
(1000, 8)
```

In [86]:
```
# number of rows
f'{df_taxi.shape[0]} rows'
```

Out[86]:
```
'1000 rows'
```

In [87]:
```
# number of columns
f'{df_taxi.shape[1]} columns'
```

Out[87]:
```
'8 columns'
```

In [88]:
```
'number of rows: {}, number of columns: {}'.format(*df_taxi.shape)
```

Out[88]:
```
'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))`

In [89]:
```
1 df_taxi.shape
```

Out[89]:
```
(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 [89]:
```
1 df_taxi.shape
```

Out[89]:
```
(1000, 8)
```

In [90]:
```
1 print(*df_taxi.shape)
```

```
1000 8
```

In [212]:
```
1 a = (1,2,3)
```

In [213]:
```
1 print(a)
```

```
(1, 2, 3)
```

In [214]:
```
1 print(*a)
```

```
1 2 3
```

# 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 [89]:
```
1 df_taxi.shape
```

Out[89]:
```
(1000, 8)
```

In [90]:
```
1 print(*df_taxi.shape)
```

```
1000 8
```

In [212]:
```
1 a = (1,2,3)
```

In [213]:
```
1 print(a)
```

```
(1, 2, 3)
```

In [214]:
```
1 print(*a)
```

```
1 2 3
```

In [91]:
```
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 [89]:
```
1 df_taxi.shape
```

Out[89]:
```
(1000, 8)
```

In [90]:
```
1 print(*df_taxi.shape)
```

```
1000 8
```

In [212]:
```
1 a = (1,2,3)
```

In [213]:
```
1 print(a)
```

```
(1, 2, 3)
```

In [214]:
```
1 print(*a)
```

```
1 2 3
```

In [91]:
```
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
```

# What are the column names?

# What are the column names?

In [93]:
```
1 df_taxi.columns
```

Out[93]:
```
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 [93]:
```
1 df_taxi.columns
```

Out[93]:
```
Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',
       'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],
      dtype='object')
```

In [94]:
```
1 # columns as numpy array
2 df_taxi.columns.values
```

Out[94]:
```
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 [93]:
```
df_taxi.columns
```

Out[93]:
```
Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',
       'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],
      dtype='object')
```

In [94]:
```
# columns as numpy array
df_taxi.columns.values
```

Out[94]:
```
array(['pickup_datetime', 'dropoff_datetime', 'trip_distance',
       'fare_amount', 'tip_amount', 'payment_type', 'day_of_week',
       'is_weekend'], dtype=object)
```

In [95]:
```
# columns as list
df_taxi.columns.tolist()
```

Out[95]:
```
['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 [96]:
```
1 df_taxi.dtypes
```

Out[96]:
```
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 [96]:
```
1 df_taxi.dtypes
```

Out[96]:
```
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 [97]:
```
1 type(df_taxi.dtypes)
```

Out[97]:
```
pandas.core.series.Series
```

# Get Summary Info for DataFrame

# Get Summary Info for DataFrame

In [98]:
```
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 [98]:
```
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. Like 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 [99]:
```
1 df_taxi.trip_distance.min()
```

Out[99]:
```
0.0
```

# Numeric: Data Ranges

In [99]:
```
1 df_taxi.trip_distance.min()
```

Out[99]:
```
0.0
```

In [100]:
```
1 df_taxi.trip_distance.max()
```

Out[100]:
```
32.77
```

In [101]:
```
1 df_taxi.columns
```

Out[101]:
```
Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',
       'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],
      dtype='object')
```

# Numeric: Data Ranges

In [99]:
```
1 df_taxi.trip_distance.min()
```

Out[99]:
```
0.0
```

In [100]:
```
1 df_taxi.trip_distance.max()
```

Out[100]:
```
32.77
```

In [101]:
```
1 df_taxi.columns
```

Out[101]:
```
Index(['pickup_datetime', 'dropoff_datetime', 'trip_distance', 'fare_amount',
       'tip_amount', 'payment_type', 'day_of_week', 'is_weekend'],
      dtype='object')
```

In [102]:
```
1 df_taxi.min(numeric_only=True)
```

Out[102]:
```
trip_distance      0.0
fare_amount        2.5
tip_amount         0.0
day_of_week          0
is_weekend       False
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 [103]:
```
1 df_taxi.fare_amount.mean()
```

Out[103]: 12.4426

# Numeric: Central Tendency with Mean

- Sample Mean

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

In [103]:
```
1 df_taxi.fare_amount.mean()
```

Out[103]:
```
12.4426
```

In [104]:
```
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 [103]: 1 df_taxi.fare_amount.mean()
```

```
Out[103]: 12.4426
```

```
In [104]: 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 [105]: 1 df_taxi.fare_amount.median()

Out[105]: 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 [105]: 1 df_taxi.fare_amount.median()

Out[105]: 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 [106]:
```
1 df_taxi['fare_amount'].quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

Out[106]: 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 [106]:
```
1 df_taxi['fare_amount'].quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

Out[106]:
```
33.5
```

In [107]:
```
1 df_taxi.fare_amount.quantile([.05,.95], interpolation='linear') # 90% of the data is between 4 and 33.5
```

Out[107]:
```
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 [106]:
```
1 df_taxi['fare_amount'].quantile(.95, interpolation='linear') # 95% of the data is less than or equal to x
```

Out[106]:
```
33.5
```

In [107]:
```
1 df_taxi.fare_amount.quantile([.05,.95], interpolation='linear') # 90% of the data is between 4 and 33.5
```

Out[107]:
```
0.05     4.0
0.95    33.5
Name: fare_amount, dtype: float64
```

In [108]:
```
1 df_taxi.fare_amount.quantile([0,.25,.5,.75,1]) # Quartiles: 25% of data is between each pair
```

Out[108]:
```
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 [215]: 1 round(df_taxi.fare_amount.var(),1)

Out[215]: 116.8
```

# Numeric: Spread with Variance

- Sample Variance

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

```
In [215]: 1 round(df_taxi.fare_amount.var(),1)

Out[215]: 116.8
```

but this is in $\text{dollars}^2$!

# 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 [110]: 1 round(df_taxi.fare_amount.std(),3)

Out[110]: 10.808
```

# Numeric: Spread with Standard Deviation

- Sample Standard Deviation

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

```
In [110]: 1 round(df_taxi.fare_amount.std(),3)

Out[110]: 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 ≤ first quartile, 25th percentile
    - ~50% of data is ≤ second quartile, 50th percentile (Median)
    - ~75% of data is ≤ third quartile, 75th percentile

# Numeric: Exploring Spread with IQR

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

- Can find quartiles with: pandas quantile or numpy percentile

# Numeric: Exploring Spread with IQR

- Quartiles
    - ~25% of data is ≤ first quartile, 25th percentile
    - ~50% of data is ≤ second quartile, 50th percentile (Median)
    - ~75% of data is ≤ 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 ≤ first quartile, 25th percentile
    - ~50% of data is ≤ second quartile, 50th percentile (Median)
    - ~75% of data is ≤ 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 [111]: 1 df_taxi.fare_amount.quantile(.75) - df_taxi.fare_amount.quantile(.25)

Out[111]: 7.5
```

# Numeric: Exploring Spread with IQR

- Quartiles
    - ~25% of data is ≤ first quartile, 25th percentile
    - ~50% of data is ≤ second quartile, 50th percentile (Median)
    - ~75% of data is ≤ 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 [111]: 1 df_taxi.fare_amount.quantile(.75) - df_taxi.fare_amount.quantile(.25)

Out[111]: 7.5
```

- IQR is robust to outliers

# Numeric: Exploring Distribution with Skew

# Numeric: Exploring Distribution with Skew

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

```
In [112]:
1 df_taxi.fare_amount.skew()
```

```
Out[112]: 2.882730031010152
```

# Numeric: Exploring Distribution with Skew

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

```
In [112]: 1 df_taxi.fare_amount.skew()

Out[112]: 2.882730031010152
```

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

# Numeric Summary Stats with `.describe`

# Numeric Summary Stats with `.describe`

In [113]:
```
1 df_taxi.describe()
```

Out[113]:

| | 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 [113]:
```
1 df_taxi.describe()
```

Out[113]:

| | 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 [114]:
```
1 df_taxi.describe().round(2) # reduce precision with round
```

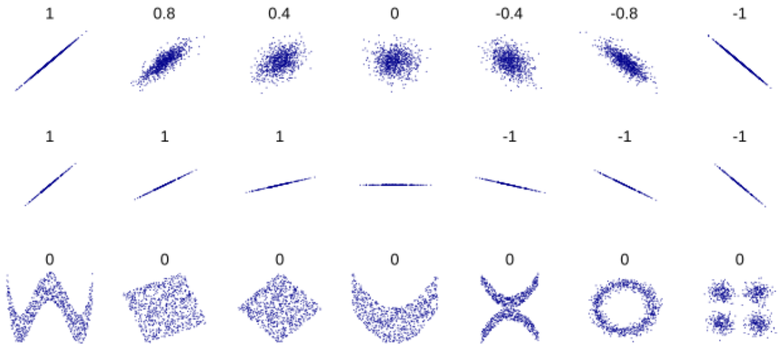
Out[114]:

| | 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

# Bivariate: Evaluating Correlation

- **Correlation:** the degree to which two variables are linearly related
- Pearson Correlation Coefficient: $\rho_{XY} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$
- Sample Correlation: $r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}$
- Takes values between:
    - -1 (highly negatively correlated)
    - 0 (not correlated)
    - 1 (highly positively correlated)



# Calculating Correlation

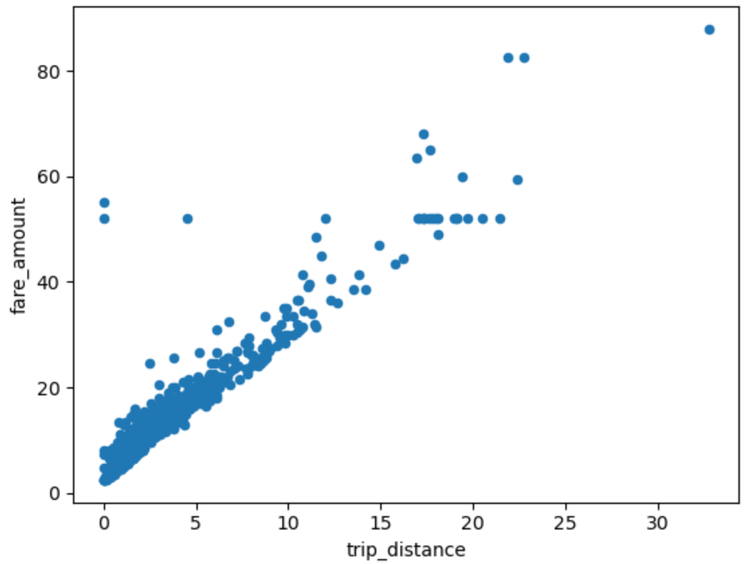
# Calculating Correlation

In [115]:
```
1 df_taxi.trip_distance.corr(df_taxi.fare_amount).round(2)
```

Out[115]: 0.95

In [116]:
```
1 df_taxi[['trip_distance','fare_amount' ]].plot.scatter('trip_distance', 'fare_amount' )
```

Out[116]: <AxesSubplot: xlabel='trip_distance', ylabel='fare_amount'>



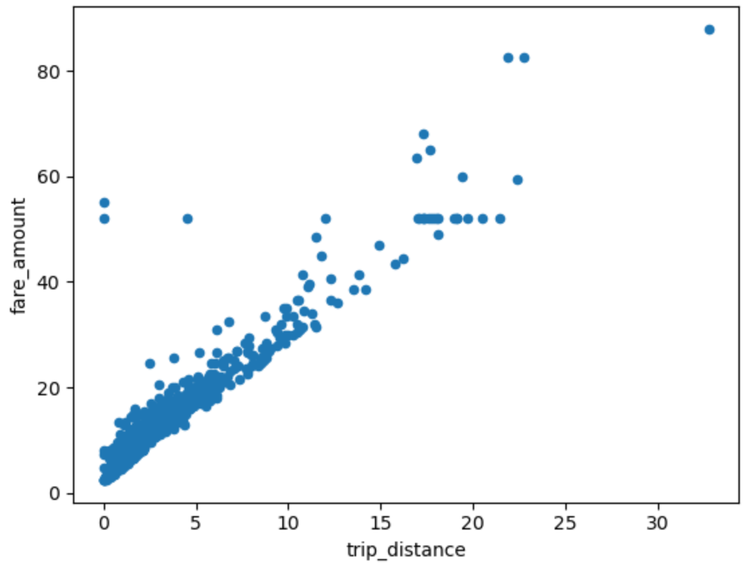
# Calculating Correlation

In [115]:
```
1 df_taxi.trip_distance.corr(df_taxi.fare_amount).round(2)
```

Out[115]: 0.95

In [116]:
```
1 df_taxi[['trip_distance','fare_amount' ]].plot.scatter('trip_distance', 'fare_amount' )
```

Out[116]: <AxesSubplot: xlabel='trip_distance', ylabel='fare_amount'>




In [117]:
```
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_counts()`

# Counting Categorical Values with `.value_counts()`

In [118]:
```
1 df_taxi.payment_type.value_counts()
```

Out[118]:
```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

# Counting Categorical Values with `.value_counts()`

In [118]:
```
1 df_taxi.payment_type.value_counts()
```

Out[118]:
```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

In [119]:
```
1 df_taxi.payment_type.value_counts(normalize=True)
```

Out[119]:
```
Credit card    0.663
Cash           0.335
No charge      0.002
Name: payment_type, dtype: float64
```

# Counting Categorical Values with `.value_counts()`

In [118]:
```
1 df_taxi.payment_type.value_counts()
```

Out[118]:
```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

In [119]:
```
1 df_taxi.payment_type.value_counts(normalize=True)
```

Out[119]:
```
Credit card    0.663
Cash           0.335
No charge      0.002
Name: payment_type, dtype: float64
```

In [120]:
```
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[120]:

| | 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 [121]:
```
1 df_taxi.groupby('payment_type')
```

Out[121]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fa9fb8c2b80>
```

In [122]:
```
1 df_taxi.trip_distance
```

Out[122]:
```
0       0.89
1       2.70
2       1.41
3       0.40
4       2.30
        ...
995     6.50
996     0.36
997     2.80
998     0.79
999     5.90
Name: trip_distance, Length: 1000, dtype: float64
```

In [216]:
```
1 df_taxi
```

Out[216]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 2017-01-26 19:58:14 | 2017-01-26 20:30:30 | 6.50 | 25.0 | 5.25 | Credit card | 3 | False |
| **996** | 2017-01-06 14:51:41 | 2017-01-06 14:54:12 | 0.36 | 3.5 | 1.29 | Credit card | 4 | False |
| **997** | 2017-01-26 02:54:52 | 2017-01-26 03:04:26 | 2.80 | 10.5 | 2.35 | Credit card | 3 | False |
| **998** | 2017-01-31 11:25:23 | 2017-01-31 11:33:17 | 0.79 | 6.5 | 0.00 | Cash | 1 | False |

# Applying Functions to Groups of Data

In [121]:
```
1 df_taxi.groupby('payment_type')
```

Out[121]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fa9fb8c2b80>
```

In [122]:
```
1 df_taxi.trip_distance
```

Out[122]:
```
0       0.89
1       2.70
2       1.41
3       0.40
4       2.30
        ...
995     6.50
996     0.36
997     2.80
998     0.79
999     5.90
Name: trip_distance, Length: 1000, dtype: float64
```

In [216]:
```
1 df_taxi
```

Out[216]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 2017-01-26 19:58:14 | 2017-01-26 20:30:30 | 6.50 | 25.0 | 5.25 | Credit card | 3 | False |
| **996** | 2017-01-06 14:51:41 | 2017-01-06 14:54:12 | 0.36 | 3.5 | 1.29 | Credit card | 4 | False |
| **997** | 2017-01-26 02:54:52 | 2017-01-26 03:04:26 | 2.80 | 10.5 | 2.35 | Credit card | 3 | False |
| **998** | 2017-01-31 11:25:23 | 2017-01-31 11:33:17 | 0.79 | 6.5 | 0.00 | Cash | 1 | False |

# Applying Functions to Groups of Data

In [121]:
```
1 df_taxi.groupby('payment_type')
```

Out[121]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fa9fb8c2b80>
```

In [122]:
```
1 df_taxi.trip_distance
```

Out[122]:
```
0       0.89
1       2.70
2       1.41
3       0.40
4       2.30
        ...
995     6.50
996     0.36
997     2.80
998     0.79
999     5.90
Name: trip_distance, Length: 1000, dtype: float64
```

In [216]:
```
1 df_taxi
```

Out[216]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 2017-01-26 19:58:14 | 2017-01-26 20:30:30 | 6.50 | 25.0 | 5.25 | Credit card | 3 | False |
| **996** | 2017-01-06 14:51:41 | 2017-01-06 14:54:12 | 0.36 | 3.5 | 1.29 | Credit card | 4 | False |
| **997** | 2017-01-26 02:54:52 | 2017-01-26 03:04:26 | 2.80 | 10.5 | 2.35 | Credit card | 3 | False |
| **998** | 2017-01-31 11:25:23 | 2017-01-31 11:33:17 | 0.79 | 6.5 | 0.00 | Cash | 1 | False |

# 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 [125]:
```
# df_taxi[df_taxi.payment_type.isin(['Cash','Credit card'])].groupby(['payment_type','is_weekend']).trip_distance.agg(['mean','
# use backslashes
df_taxi.loc[df_taxi.payment_type.isin(['Cash'])]\
    .groupby(['payment_type','is_weekend'])\
    .trip_distance.agg(['mean','median'])
```

Out[125]:

| | | 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 [125]:
```
# df_taxi[df_taxi.payment_type.isin(['Cash','Credit card'])].groupby(['payment_type','is_weekend']).trip_distance.agg(['mean','

# use backslashes
df_taxi.loc[df_taxi.payment_type.isin(['Cash'])]\
    .groupby(['payment_type','is_weekend'])\
    .trip_distance.agg(['mean','median'])
```

Out[125]:

| | | mean | median |
|---|---|---|---|
| payment_type | is_weekend | | |
| Cash | False | 2.593063 | 1.28 |
| | True | 3.507059 | 2.10 |

In [126]:
```
# wrap in parentheses
(
    df_taxi
    .loc[df_taxi.payment_type.isin(['Cash'])]
    .groupby(['payment_type','is_weekend'])
    .trip_distance.agg(['mean','median'])
)
```

Out[126]:

| | | 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 [127]:
```
1 df_taxi[['trip_distance','fare_amount']].head(10)
```

Out[127]:

| | 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 [128]:
```
(
    df_taxi[['trip_distance','fare_amount']]
    .head(10)
    .style
    .format(precision=1)
    .background_gradient()
)
```

Out[128]:

| | 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 [128]:
```
(
    df_taxi[['trip_distance','fare_amount']]
    .head(10)
    .style
    .format(precision=1)
    .background_gradient()
)
```

Out[128]:

| | 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

In [224]:
```
1 df_taxi.head(5)
```

Out[224]:

| | 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 |

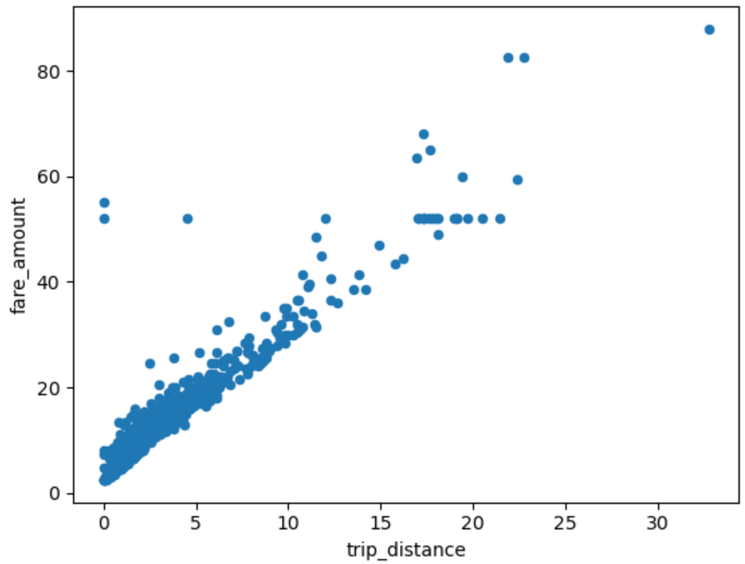
# Plotting via Pandas

In [224]:
```
1 df_taxi.head(5)
```

Out[224]:

| | 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 |

In [226]:
```
1 df_taxi.plot.scatter(x='trip_distance',y='fare_amount');
```



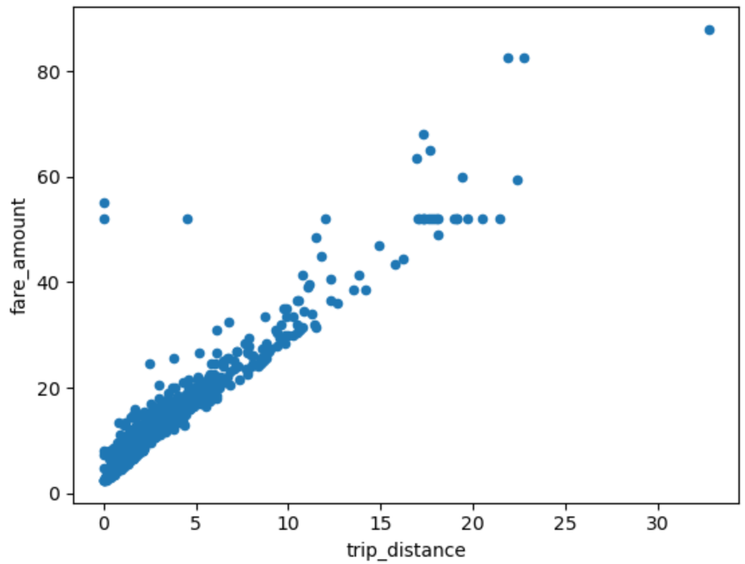
# Using semi-colon to hide supress ”end of cell print”

In [217]:
```
1 import matplotlib.pyplot as plt
```

# Using semi-colon to hide supress ”end of cell print”

In [217]:
```
1 import matplotlib.pyplot as plt
```

In [218]:
```
1 df_taxi.plot.scatter(x='trip_distance',y='fare_amount')
2 plt.show()
```



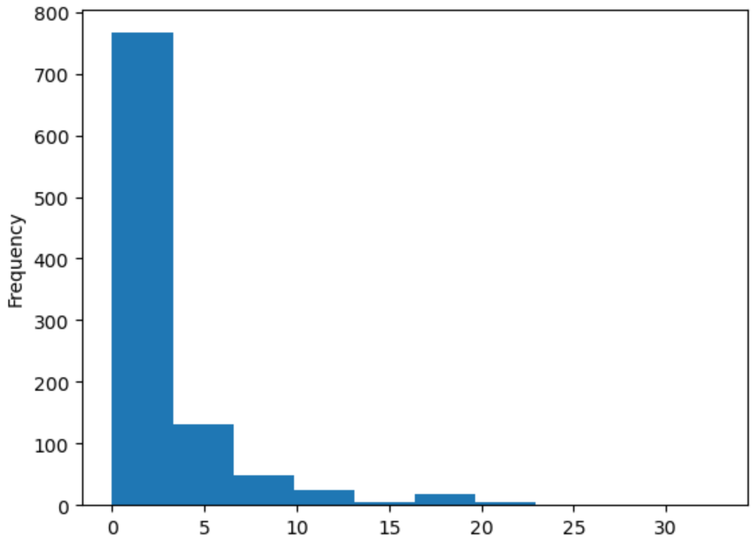
# Manipulating plots with Matplotlib

- sizing
- adding titles
- changing axis labels
- changing axis tics

# Manipulating plots with Matplotlib

- sizing
- adding titles
- changing axis labels
- changing axis tics

In [228]:
```
1 ax = df_taxi.trip_distance.plot.hist();
2
```



# Import matplotlib.pyplot

# Import matplotlib.pyplot

In [132]:

```
1 import matplotlib.pyplot as plt
2
3 %matplotlib inline
```

# Matplotlib Axes

# Matplotlib Axes

In [251]:

```
andy,(ax1,ax2,ax3) = plt.subplots(1,3,figsize=(12,4)) # set the figure size
(
    df_taxi
    .plot.scatter(
        x = 'trip_distance',
        y = 'fare_amount',
        marker='.',
        color='blue',
        ax=ax1
    )
);
ax.set_xlabel('Trip distance') # set x and y axis labels
ax.set_ylabel('Fare amount')
ax.set_xlim([10,40]) # set x and y axis limits
ax.set_ylim([-5,100])
ax.set_title('trip_distance vs fare_amount for NY taxi trips in Jan 2017'); # set axis title



(
    df_taxi
    .plot.scatter(
        x = 'tip_amount',
        y = 'fare_amount',
        marker='.',
        color='blue',
        ax=ax2
    )
);

(
    df_taxi
    .plot.scatter(
```

# Matplotlib and dpi

# Matplotlib and dpi

In [134]:

```python
def find_dpi(w, h, d):
    """
    https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
    w : width in pixels
    h : height in pixels
    d : diagonal in inches
    """
    w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
    return round(w / w_inches)

find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[134]: 166

# Matplotlib and dpi

In [134]:
```python
def find_dpi(w, h, d):
    """
    https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
    w : width in pixels
    h : height in pixels
    d : diagonal in inches
    """
    w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
    return round(w / w_inches)

find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[134]: 166

In [135]:
```python
fig.dpi # from previous figure
```

Out[135]: 100.0

# Matplotlib and dpi

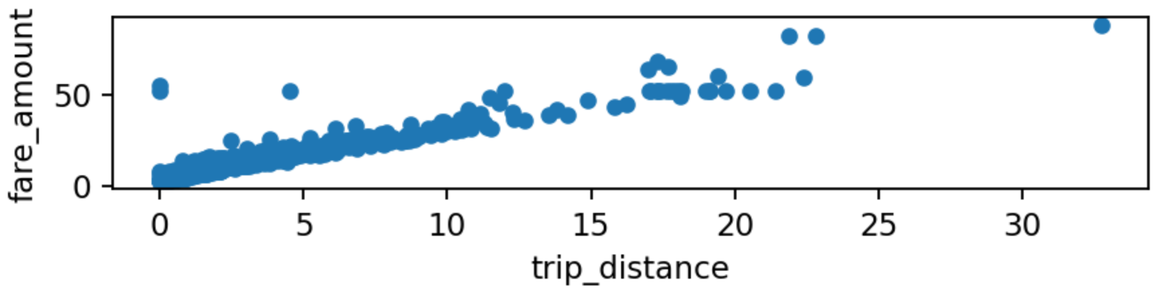
In [134]:
```
def find_dpi(w, h, d):
    """
    https://medium.com/dunder-data/why-matplotlib-figure-inches-dont-match-your-screen-inches-and-how-to-fix-it-993fa0417dba
    w : width in pixels
    h : height in pixels
    d : diagonal in inches
    """
    w_inches = (d ** 2 / (1 + h ** 2 / w ** 2)) ** 0.5
    return round(w / w_inches)

find_dpi(1920, 1080, 13.25) # approx what my native dpi is
```

Out[134]: 166

In [135]:
```
fig.dpi # from previous figure
```

Out[135]: 100.0

In [136]:
```
fig, ax = plt.subplots(figsize=(6, 1), dpi=300)
df_taxi.plot.scatter(x = 'trip_distance',y = 'fare_amount',ax=ax);
```

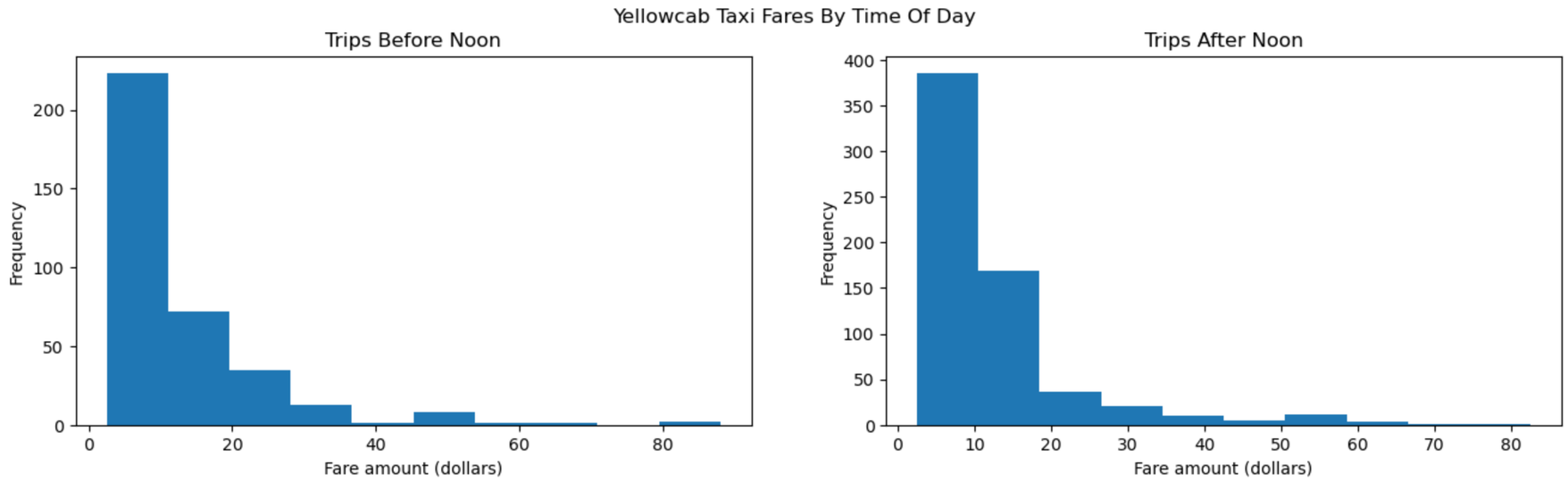


# Matplotlib: Subplots, Figure and Axis

# Matplotlib: Subplots, Figure and Axis

In [254]:

```
fig,ax = plt.subplots(1,2,figsize=(16,4))

df_taxi[df_taxi.pickup_datetime.dt.hour < 12].fare_amount.plot.hist(ax=ax[0]);
ax[0].set_xlabel('Fare amount (dollars)');
ax[0].set_title('Trips Before Noon');

df_taxi[df_taxi.pickup_datetime.dt.hour >= 12].fare_amount.plot.hist(ax=ax[1]);
ax[1].set_xlabel('Fare amount (dollars)');
ax[1].set_title('Trips After Noon');
# Matplotlib: Subplots, Figure and Axis
fig.suptitle('Yellowcab Taxi Fares By Time Of Day');
```

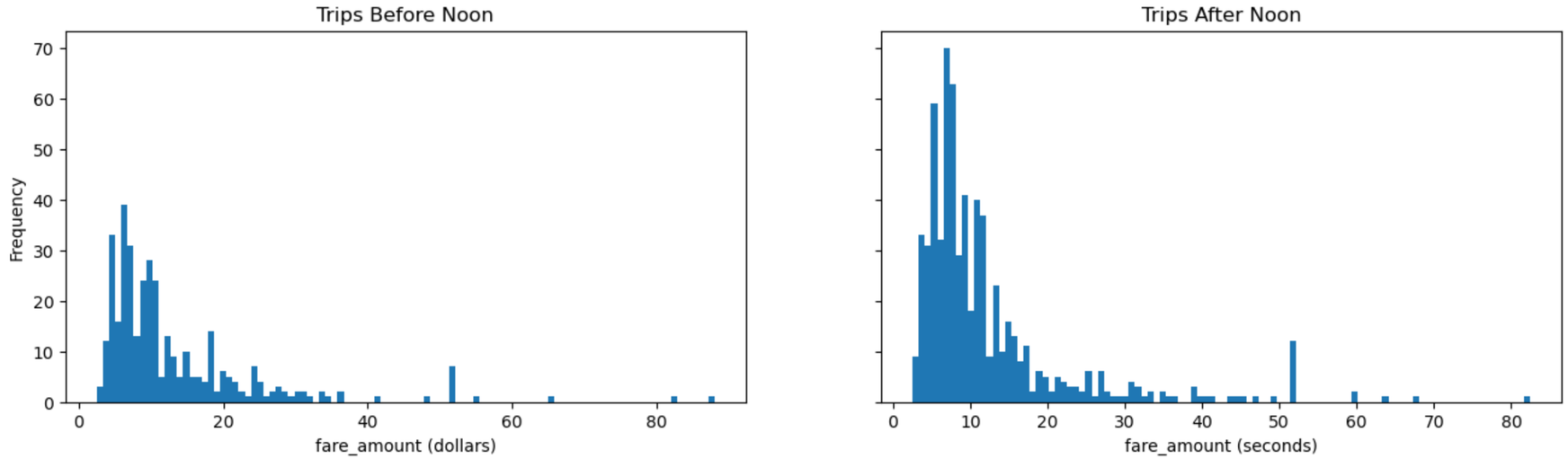


# Matplotlib: Sharing Axes

# Matplotlib: Sharing Axes

In [138]:

```
fig,ax = plt.subplots(1,2,figsize=(16,4), sharey=True)

df_taxi[df_taxi.pickup_datetime.dt.hour < 12].fare_amount.plot.hist(bins=100,ax=ax[0]);
ax[0].set_xlabel('fare_amount (dollars)');
ax[0].set_title('Trips Before Noon');
df_taxi[df_taxi.pickup_datetime.dt.hour >= 12].fare_amount.plot.hist(bins=100,ax=ax[1]);
ax[1].set_xlabel('fare_amount (seconds)');
ax[1].set_title('Trips After Noon');
```

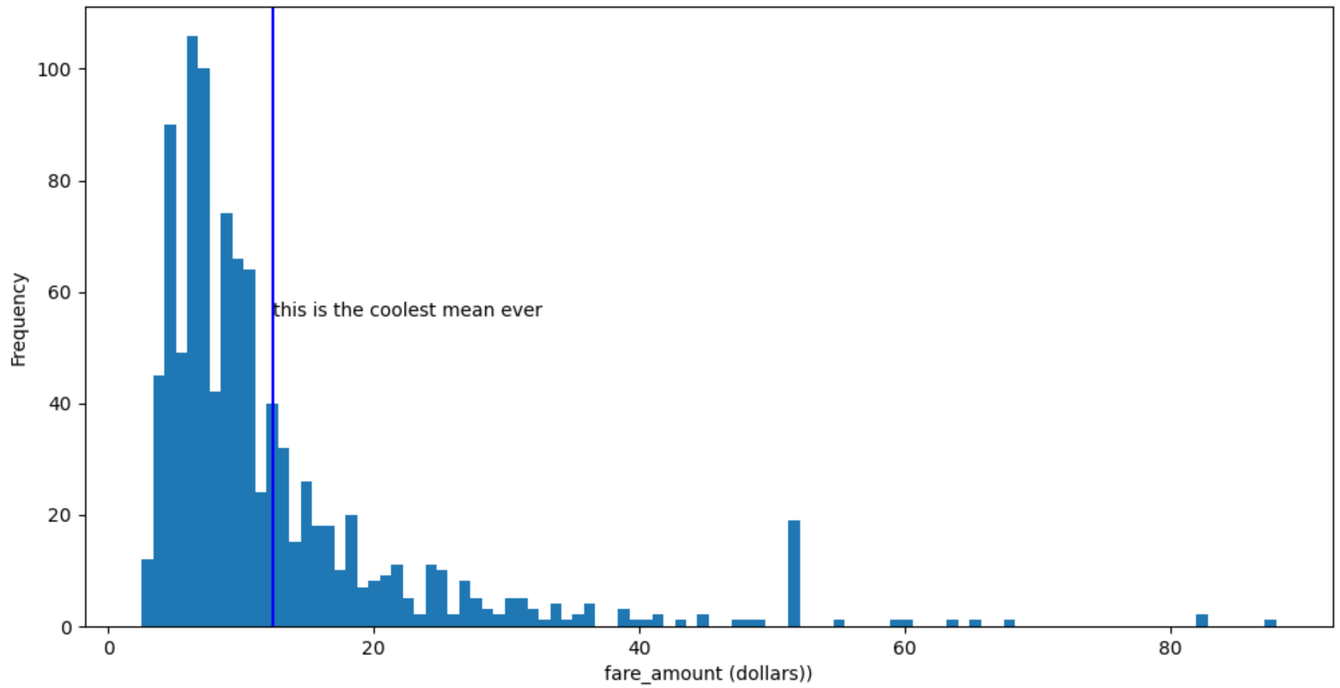


# Matplotlib: adding lines and annotations

# Matplotlib: adding lines and annotations

In [260]:

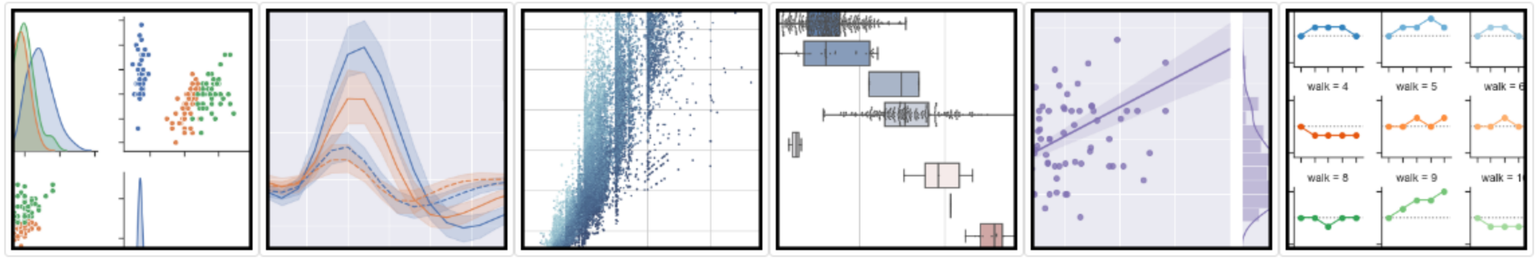
```
fig,ax = plt.subplots(1,1,figsize=(12,6));

df_taxi.fare_amount.plot.hist(bins=100, ax=ax);
ax.set_xlabel('fare_amount (dollars))');

# add a vertical line
ax.axvline(df_taxi.fare_amount.mean(),color='b');
#ax.vlines(df_taxi.fare_amount.mean(),*ax.get_ylim(),color='r');

# add some text
ax.text(df_taxi.fare_amount.mean(),ax.get_ylim()[1]*.5,'this is the coolest mean ever');
```



# Plotting with Seaborn

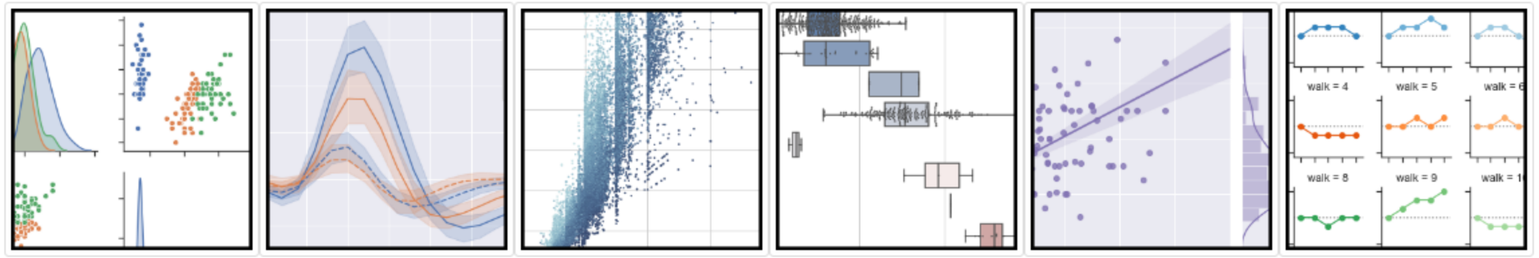
# Plotting with Seaborn

- 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 [140]:
```
1 import seaborn as sns
2 sns.__version__
```

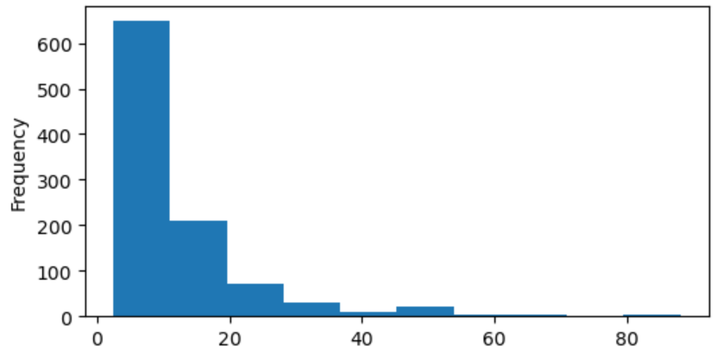
Out[140]: '0.12.2'

# Univariate Distribution: Histograms

# Univariate Distribution: Histograms

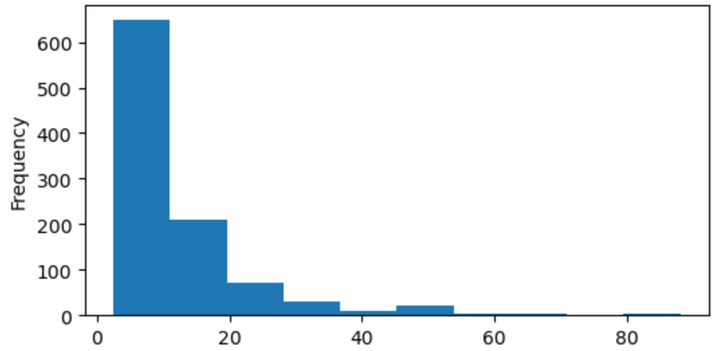
In [141]:

```
fig,ax = plt.subplots(1,1,figsize=(6,3))

df_taxi.fare_amount.plot.hist(ax=ax);
```

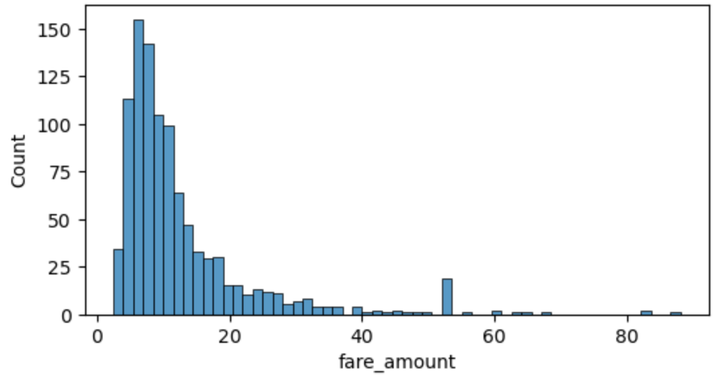


# Univariate Distribution: Histograms

In [141]:
```
fig,ax = plt.subplots(1,1,figsize=(6,3))

df_taxi.fare_amount.plot.hist(ax=ax);
```



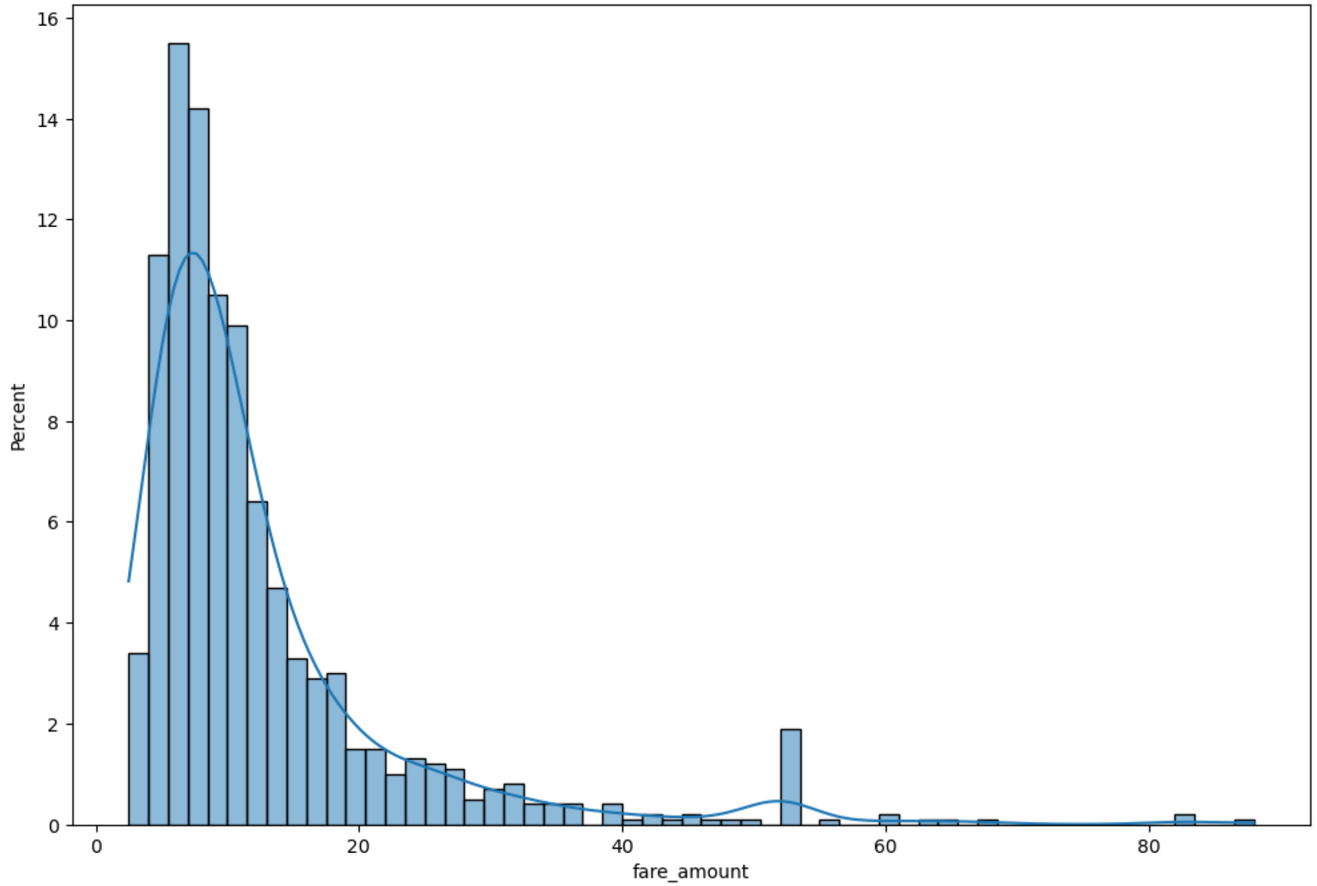

In [262]:
```
fig,ax = plt.subplots(1,1,figsize=(6,3))

sns.histplot(x='fare_amount',data=df_taxi,ax=ax); # sns.histplot(x=df_taxi.fare_amount,ax=ax);
```



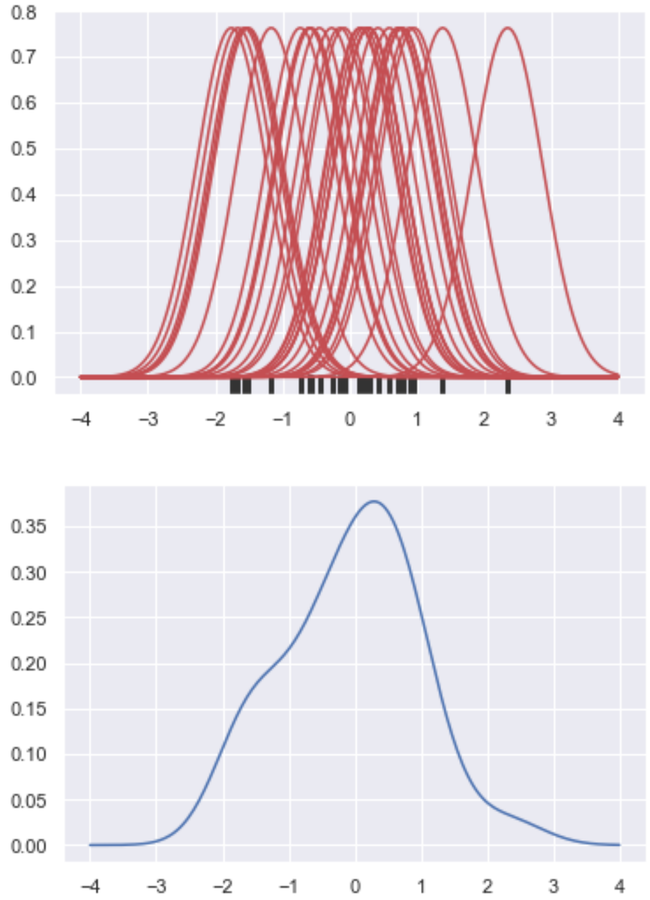
# Univariate Distribution: Histograms

# Univariate Distribution: Histograms

In [143]:
```
fig,nd = plt.subplots(1,1,figsize=(12,8))

# many other parameters to play with
sns.histplot(x='fare_amount',data=df_taxi,ax=nd,kde=True,stat='percent');
```



# Aside: KDE


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−3
−2
−1
0
1
2
3
4
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4
−3
−2
−1
0
1
2
3
4

# Seaborn Styles

# Seaborn Styles

In [144]:

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

# Seaborn Styles

In [144]:
```
# for a single plot using a context
with sns.axes_style('whitegrid'):
    fig,ax = plt.subplots(1,1,figsize=(10,1))
    sns.histplot(x='fare_amount',data=df_taxi);
```


In [145]:
```
# set style globally: darkgrid, whitegrid, dark, white, ticks
sns.set_style('darkgrid')
```

# Seaborn Styles

In [144]:
```
# for a single plot using a context
with sns.axes_style('whitegrid'):
    fig,ax = plt.subplots(1,1,figsize=(10,1))
    sns.histplot(x='fare_amount',data=df_taxi);
```


In [145]:
```
# set style globally: darkgrid, whitegrid, dark, white, ticks
sns.set_style('darkgrid')
```

In [146]:
```
fig,ax = plt.subplots(1,1,figsize=(10,1))
sns.histplot(x='fare_amount',data=df_taxi);
```

# Seaborn Styles

In [144]:
```
# for a single plot using a context
with sns.axes_style('whitegrid'):
    fig,ax = plt.subplots(1,1,figsize=(10,1))
    sns.histplot(x='fare_amount',data=df_taxi);
```


In [145]:
```
# set style globally: darkgrid, whitegrid, dark, white, ticks
sns.set_style('darkgrid')
```

In [146]:
```
fig,ax = plt.subplots(1,1,figsize=(10,1))
sns.histplot(x='fare_amount',data=df_taxi);
```

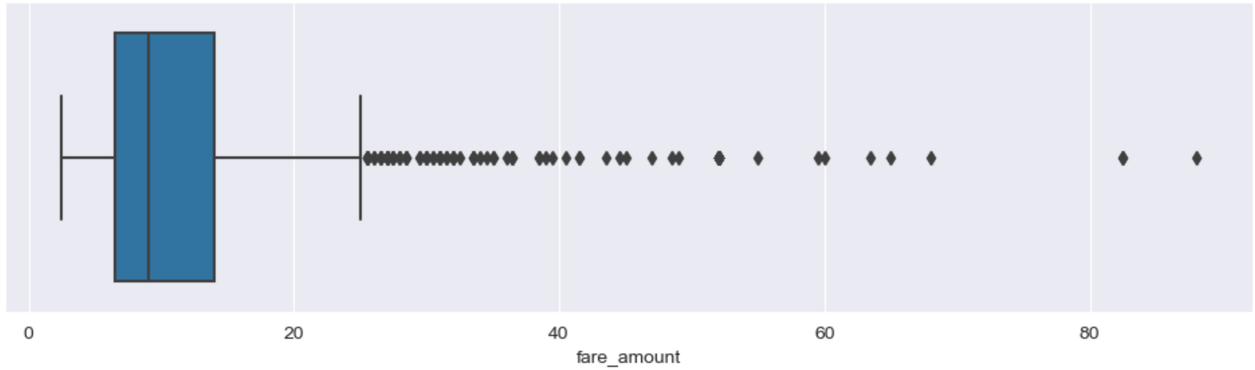

In [147]:
```
# to reset to matplotlib defaults
#import matplotlib
#matplotlib.rc_file_defaults()
```

# Univariate Distributions: Boxplot

# Univariate Distributions: Boxplot

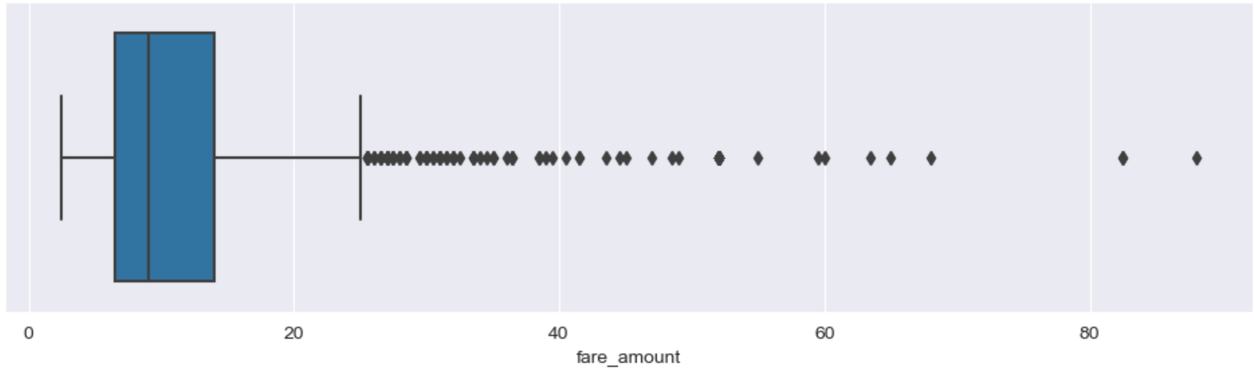
In [148]:

```
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 [148]:
```
1 fig,ax = plt.subplots(1,1,figsize=(12,3))
2
3 sns.boxplot(x='fare_amount',data=df_taxi,ax=ax);
```



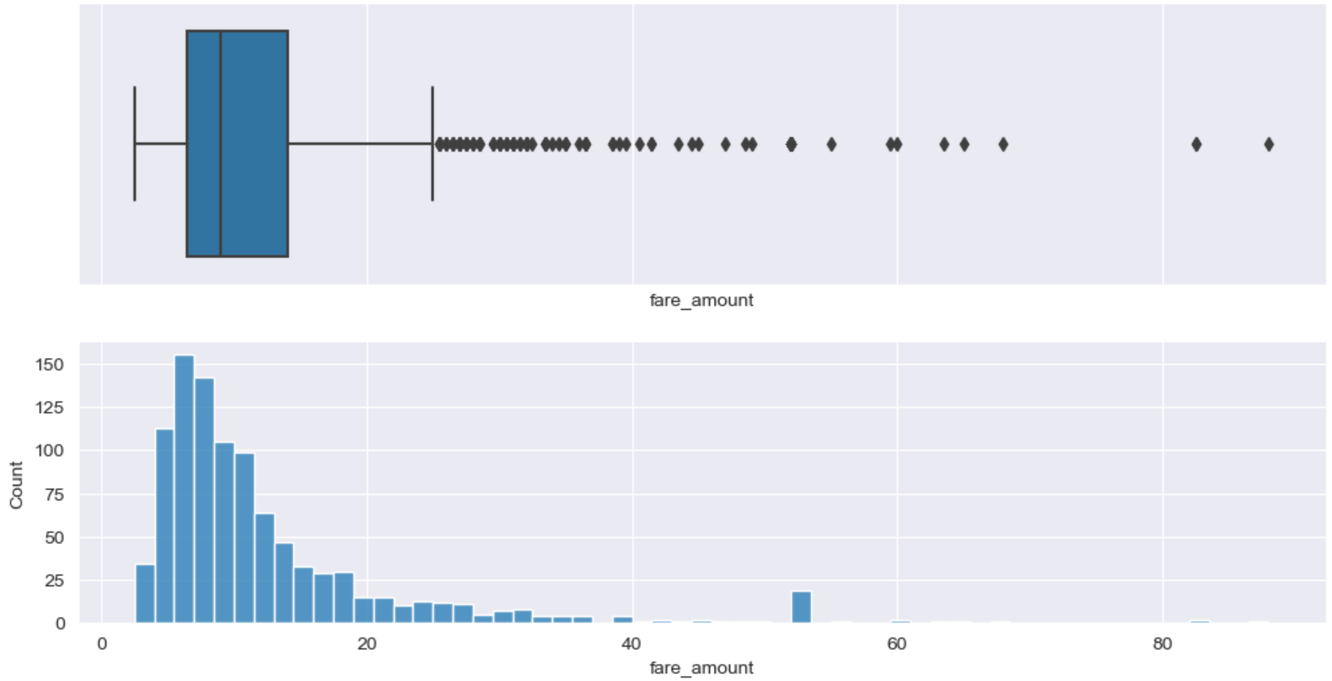

- first quartile
- second quartile (Median)
- third quartile
- whiskers (usually 1.5*IQR)
- outliers

# Seaborn: Combining Plots with Subplots

# Seaborn: Combining Plots with Subplots

In [149]:

```
fig,ax = plt.subplots(2,1,figsize=(12,6), sharex=True)

sns.boxplot(x='fare_amount', data=df_taxi, ax=ax[0]);
sns.histplot(x='fare_amount', data=df_taxi, ax=ax[1]);
```

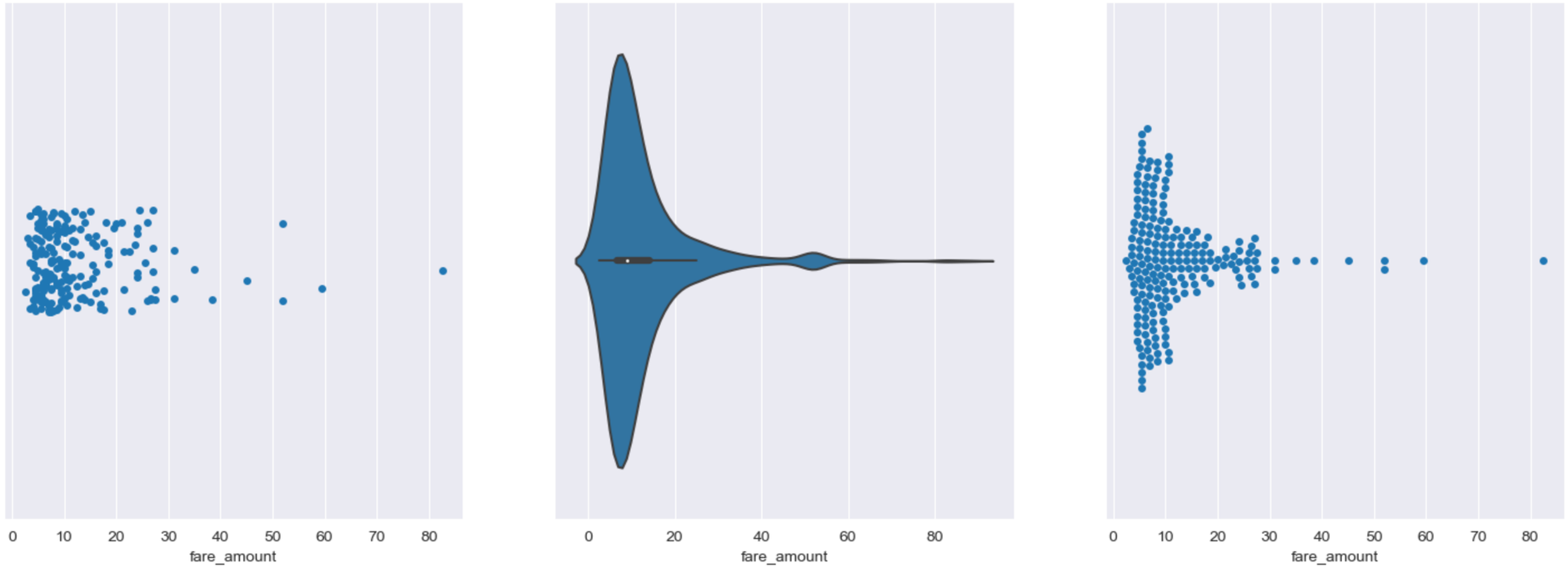


# Other Univariate Distribution Visualizations

# Other Univariate Distribution Visualizations

In [150]:

```
fig,ax = plt.subplots(1,3,figsize=(18,6))

sns.stripplot(x='fare_amount',data=df_taxi[:200],ax=ax[0])
sns.violinplot(x='fare_amount',data=df_taxi,ax=ax[1])
sns.swarmplot(x='fare_amount',data=df_taxi[:200],ax=ax[2]);
```

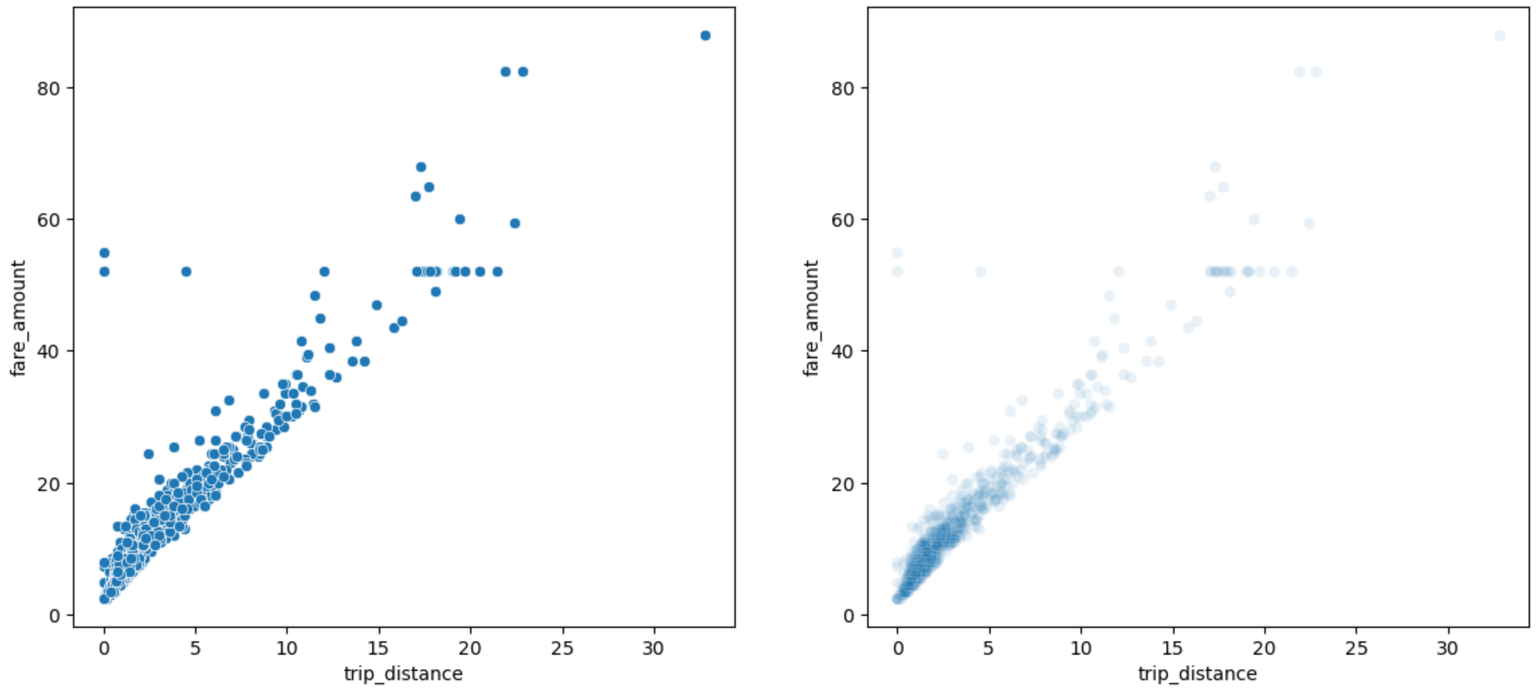


# Bivariate: Scatterplot (with alpha)

# Bivariate: Scatterplot (with alpha)

In [263]:

```
fig,ax = plt.subplots(1,2,figsize=(14,6))
sns.scatterplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax[0]);
sns.scatterplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax[1], alpha=0.1);
```

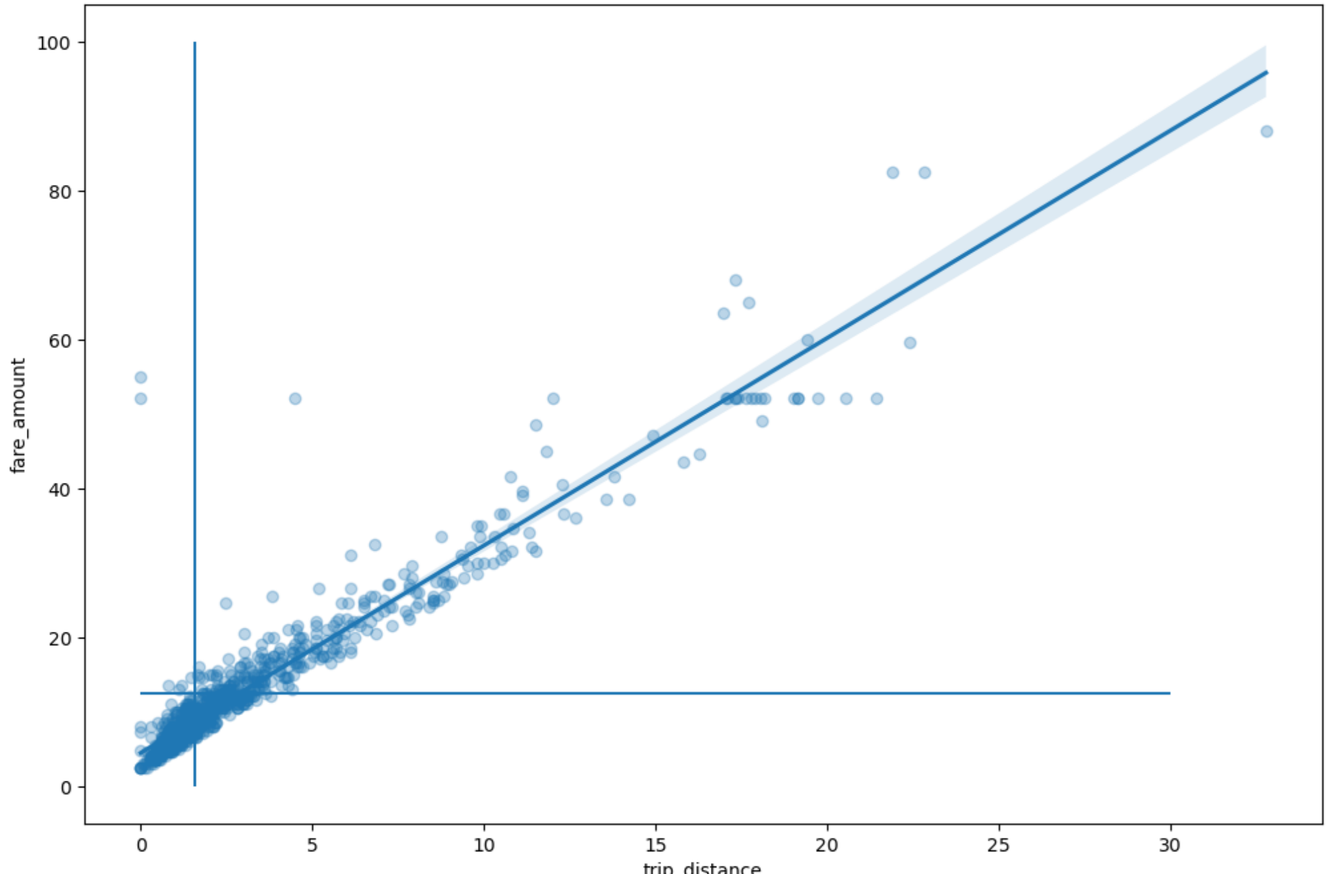


# Bivariate: Add Regression Line

# Bivariate: Add Regression Line

In [266]:

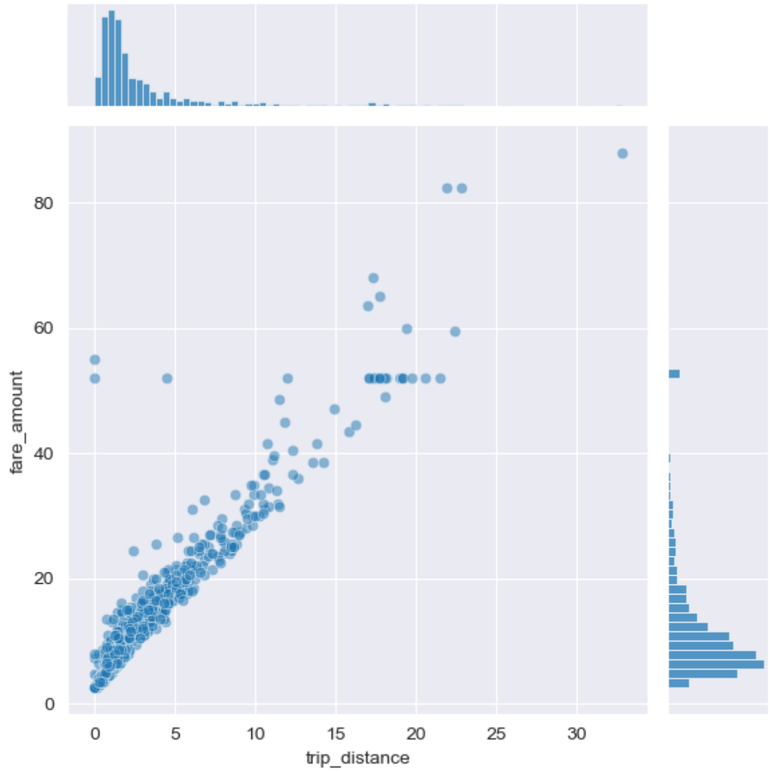
```
fig,ax = plt.subplots(1,1,figsize=(12,8))

sns.regplot(x='trip_distance', y='fare_amount', data=df_taxi, ax=ax, scatter_kws={'alpha':0.3});
plt.vlines(x = df_taxi['trip_distance'].median(), ymin=0, ymax=100)
plt.hlines(y = df_taxi['fare_amount'].mean(), xmin=0, xmax=30)

plt.show()
```



# Bivariate: Joint Plot

# Bivariate: Joint Plot

In [153]:
```
1 sns.jointplot(x='trip_distance',y='fare_amount',data=df_taxi,alpha=0.5);
```

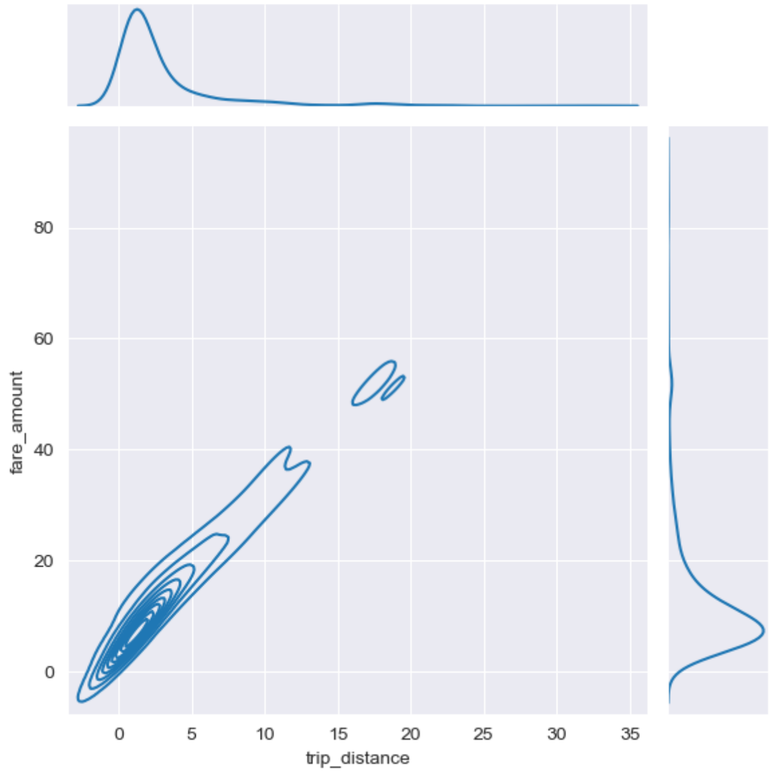


# Bivariate: Joint Plot with KDE

# Bivariate: Joint Plot with KDE

In [154]:

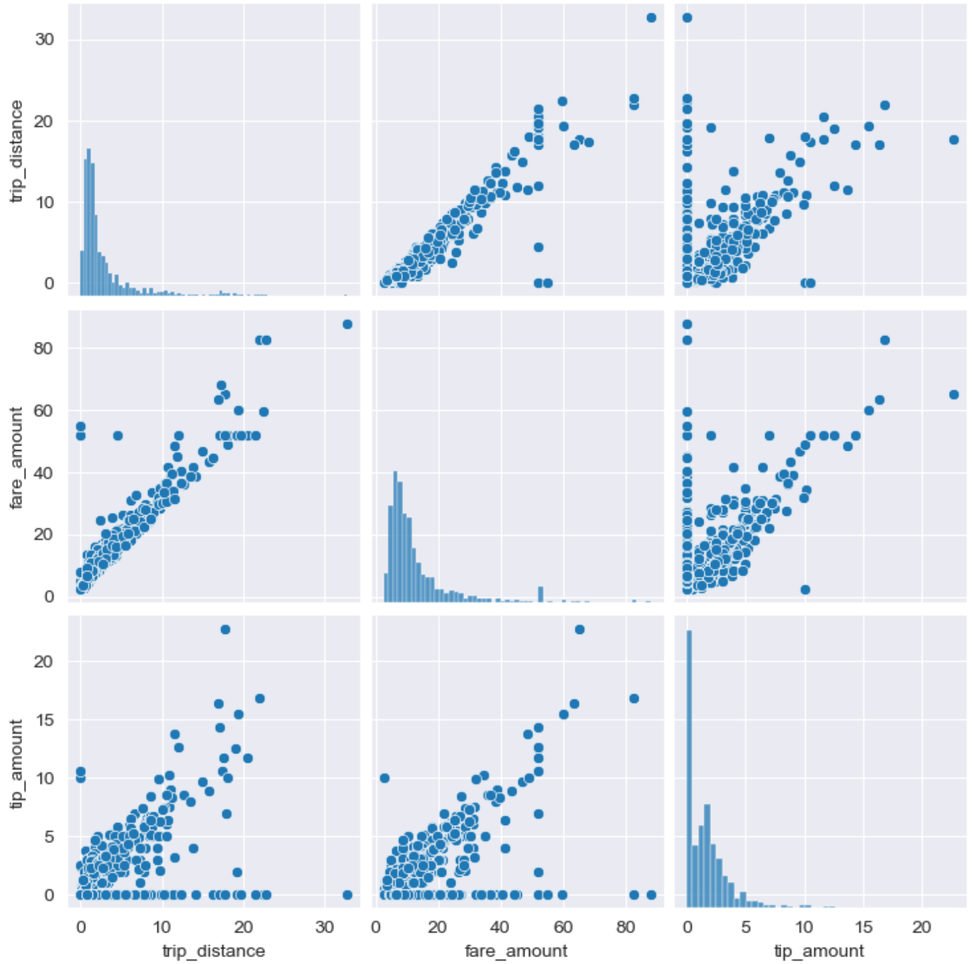
```
sns.jointplot(x='trip_distance', y='fare_amount',
              data=df_taxi,
              kind='kde');
```



# Comparing Multiple Variables with `pairplot`

# Comparing Multiple Variables with `pairplot`

In [155]:
```
1 sns.pairplot(data=df_taxi[['trip_distance','fare_amount','tip_amount']]);
```



# Categorical Variables: Frequency

# Categorical Variables: Frequency

In [156]:
```
1 df_taxi.payment_type.value_counts()
```

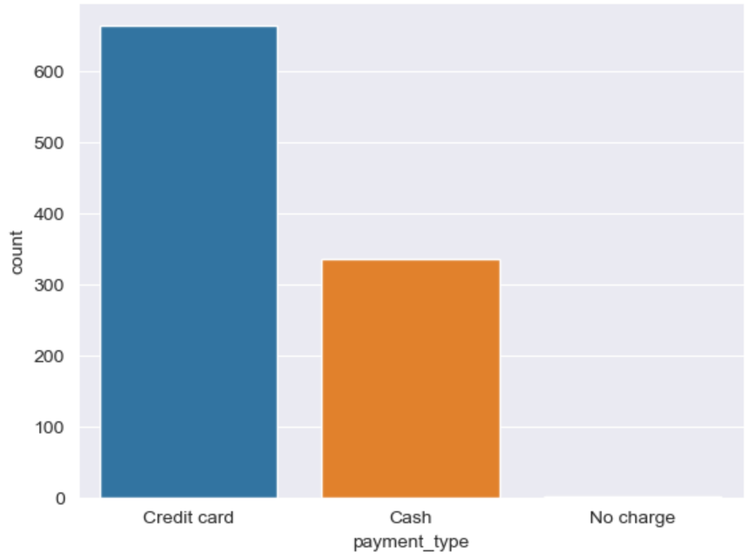
Out[156]:
```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

# Categorical Variables: Frequency

In [156]:
```
1 df_taxi.payment_type.value_counts()
```

Out[156]:
```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

In [157]:
```
1 sns.countplot(x='payment_type',data=df_taxi);
```



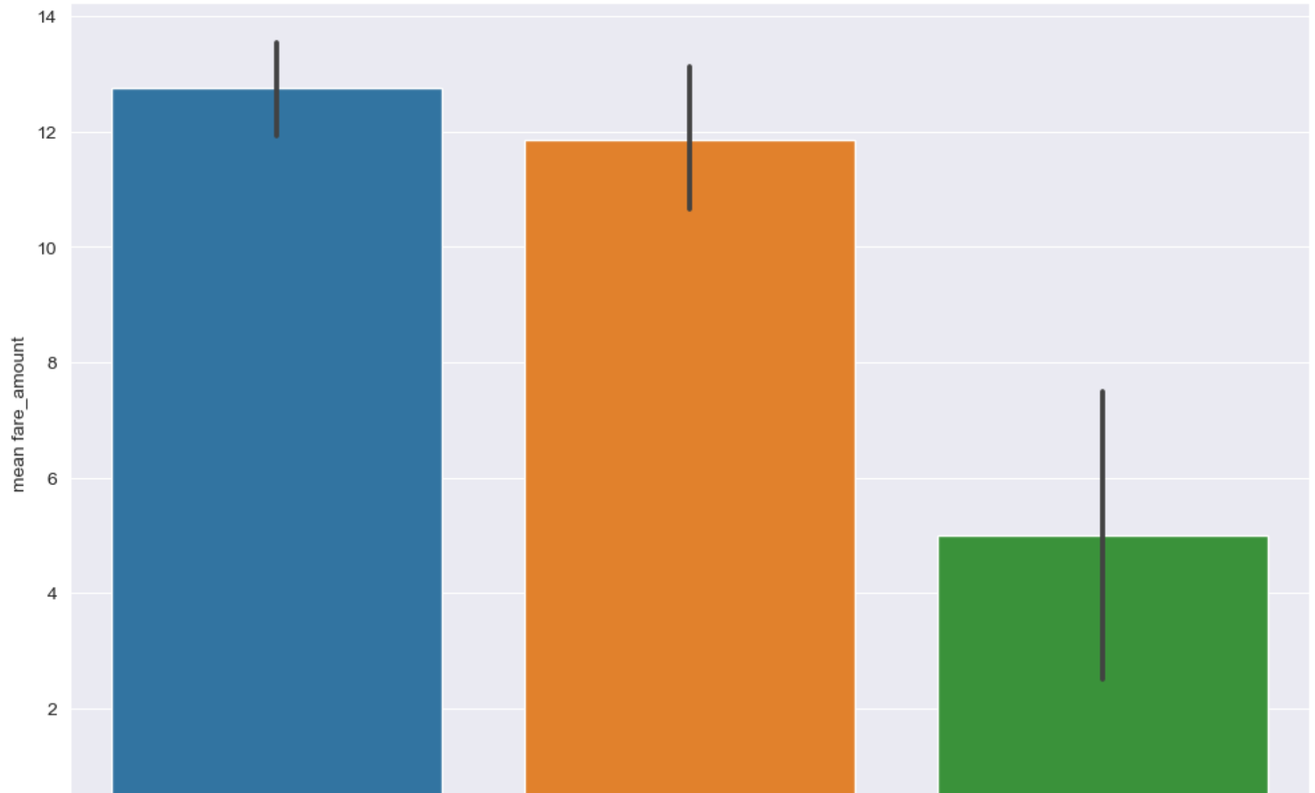
# Plotting Numeric and Categorical

# Plotting Numeric and Categorical

In [158]:

```
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_18658/1040376770.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);
```

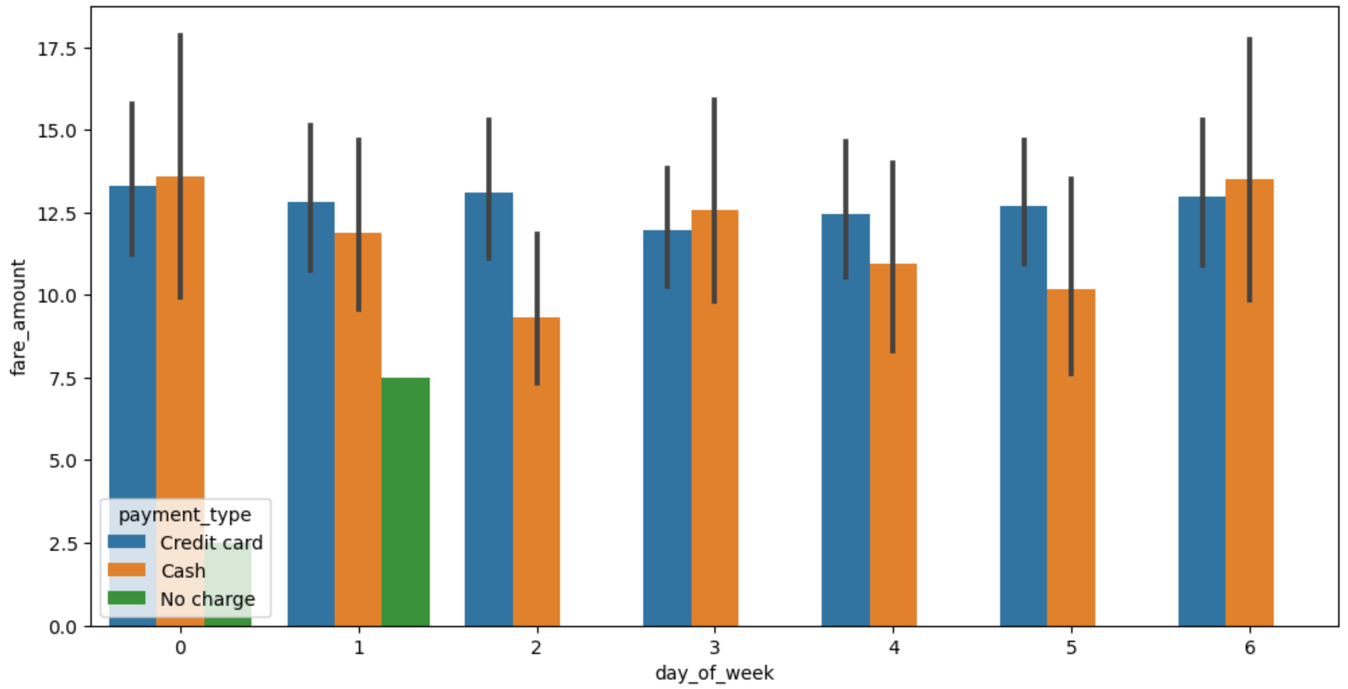


# Plotting with Hue

# Plotting with Hue

In [268]:

```
fig,ax = plt.subplots(1,1,figsize=(12,6))

# add a second categorical variable day_of_week
sns.barplot(x='day_of_week',
            y='fare_amount',
            hue='payment_type',
            data=df_taxi,
            ax=ax,
           );
```

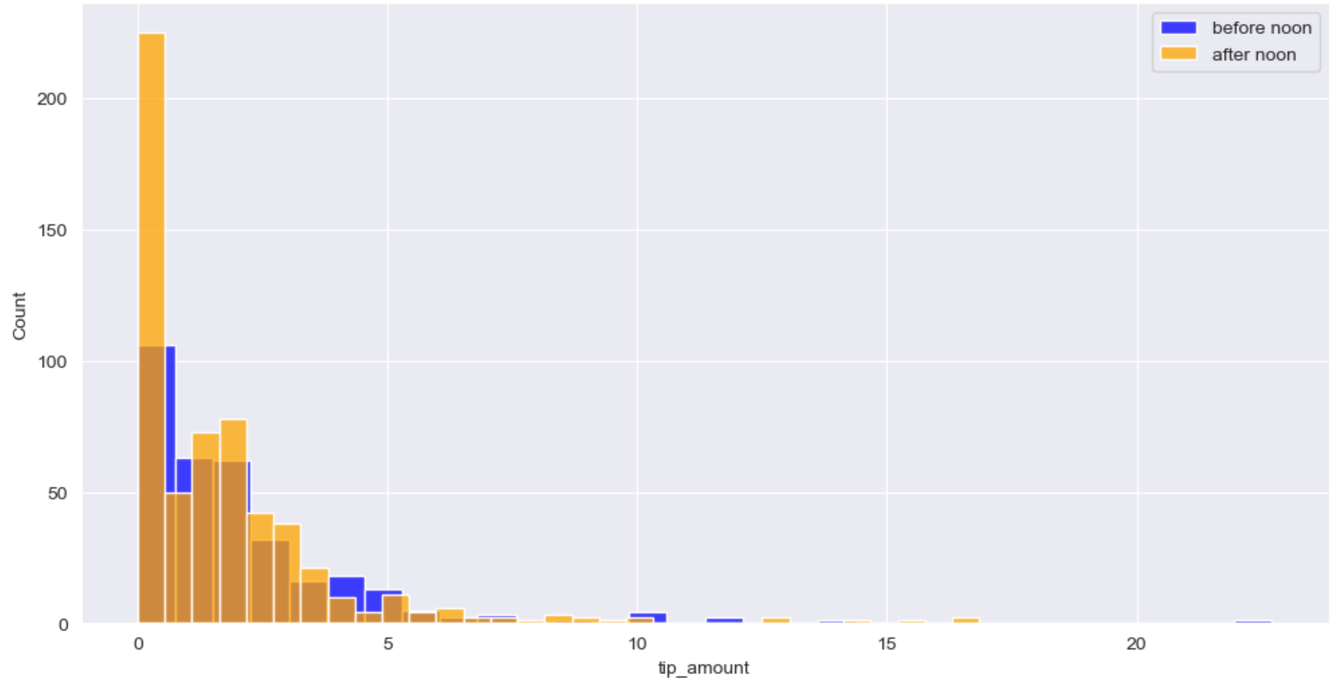


# Same Axis, Multiple Plots with Seaborn (with legend)

# Same Axis, Multiple Plots with Seaborn (with legend)

In [160]:

```
fig,ax = plt.subplots(1,1,figsize=(12,6))
sns.histplot(x='tip_amount',data=df_taxi[df_taxi.pickup_datetime.dt.hour  < 12], label='before noon',color='blue', ax=ax);
sns.histplot(x='tip_amount',data=df_taxi[df_taxi.pickup_datetime.dt.hour >= 12], label='after noon', color='orange',ax=ax);
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?