Elements Of Data Science - S2022

Week 3: Pandas, Data Exploration and Visualization

2/1/2022

TODOs

- **Read** Selections from PDSH Chapter 3
- **Read** Selections from PDSH Chapter 4
- (Optional) Seaborn Tutorial **https://seaborn.pydata.org/tutorial.html**

- Complete Week 3 Quiz
- HW1 out this week, includes questions on Hypothesis Testing

TODAY

- Pandas
- Data Exploration
- Visualization in Python

Questions?

Environment Setup

In [1]:

```
import numpy as np
```

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]:

```
# 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

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

In [3]:

```
s = pd.Series(np.random.rand(4))
s
```

Out[3]:

```
0    0.472418
1    0.924422
2    0.033124
3    0.441399
dtype: float64
```

In [4]:

```
# return the values of the series
s.values
```

Out[4]:

```
array([0.47241843, 0.9244225 , 0.03312375, 0.44139901])
```

In [5]:

```
# return the index of the series
s.index
```

Out[5]:

```
RangeIndex(start=0, stop=4, step=1)
```

Pandas Series Cont.

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

In [6]:

```
# create Series from array and set index
s = pd.Series([1,2,3],index=['a','b','c'],name='Example_Series')
s
```

Out[6]:

```
a      1
b      2
c      3
Name: Example_Series, dtype: int64
```

In [7]:

```
s['a']
```

Out[7]:

```
1
```

In [8]:

```
s[['b','c']]
```

Out[8]:

```
b    2
c    3
Name: Example_Series, dtype: int64
```

Pandas Series Cont.

- accessing other Series attributes

In [9]:

```
s
```

Out[9]:

```
a     1
b     2
c     3
Name: Example_Series, dtype: int64
```

In [10]:

```
print(f'{s.index  = :}')
print(f'{s.values = :}')
print(f'{s.name   = :>20s}')
print(f'{s.dtype  = :}')
print(f'{s.shape  = :}')
'{:>20s}'.format(s.name)
```

```
s.index  = Index(['a', 'b', 'c'], dtype='object')
s.values = [1 2 3]
s.name   =       Example_Series
```

```
s.dtype  = int64
s.shape  = (3,)
```

Out[10]:

```
'      Example_Series'
```

Pandas Series Cont.

In [11]:

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

Out[11]:

```
a     1
b     2
c     3
d     4
dtype: int64
```

In [12]:

```
print(f'{s.index  = :}')
print(f'{s.values = :}')
```

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

Pandas DataFrame

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

Pandas DataFrame Cont.

In [13]:

```
df = pd.DataFrame({'Year':[2017,2018,2018,2019],
                   'Class_Name':['A','A','B','A'],
                   'Measure1':[2.1,3.0,2.4,1.9]
                  })
```

In [14]:

```
df
```

Out[14]:

	Year	Class_Name	Measure1
0	2017	A	2.1
1	2018	A	3.0
2	2018	B	2.4
3	2019	A	1.9

In [15]:

```
print(df)
```

```
   Year Class_Name  Measure1
0  2017          A       2.1
```

```
1  2018            A         3.0
2  2018            B         2.4
3  2019            A         1.9
```

In [16]:

```
display(df)
```

	Year	Class_Name	Measure1
0	2017	A	2.1
1	2018	A	3.0
2	2018	B	2.4
3	2019	A	1.9

Pandas DataFrame Cont.

In [17]:

```
data = [[2017,'A',2.1],
        [2018,'A',3.0],
        [2018,'B',2.4],
        [2019,'A',1.9]]
```

In [18]:

```
df = pd.DataFrame(data,
                  columns=['Year','Class_Name','Measure1'],
                  index=['001','002','003','004'])
df.shape
```

Out[18]:

```
(4, 3)
```

In [19]:

```
df
```

Out[19]:

	Year	Class_Name	Measure1
001	2017	A	2.1
002	2018	A	3.0
003	2018	B	2.4

	Year	Class_Name	Measure1
004	2019	A	1.9

Pandas Attributes

- Get shape of DataFrame : `shape`

In [20]:

```
df.shape # rows, columns
```

Out[20]:

```
(4, 3)
```

- Get index values : `index`

In [21]:

```
df.index
```

Out[21]:

```
Index(['001', '002', '003', '004'], dtype='objec
t')
```

- Get column values : `columns`

In [22]:

```
df.columns
```

Out[22]:

```
Index(['Year', 'Class_Name', 'Measure1'], dtype='o
bject')
```

Pandas Indexing/Selection

Select by label:

- `.loc[]`

In [23]:

```
df.loc['001']
```

Out[23]:

```
Year               2017
Class_Name            A
Measure1            2.1
Name: 001, dtype: object
```

In [24]:

```
df.loc['001','Measure1']
```

Out[24]:

```
2.1
```

Pandas Indexing/Selection Cont.

Select by position:

- `.iloc[]`

In [25]:

```
df.iloc[0]
```

Out[25]:

```
Year                2017
Class_Name             A
Measure1             2.1
Name: 001, dtype: object
```

In [26]:

```
df.iloc[0,2]
```

Out[26]:

```
2.1
```

Pandas Indexing/Selection Cont.

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

In [27]:

```
df.loc[['002','004']]
```

Out[27]:

	Year	Class_Name	Measure1
002	2018	A	3.0
004	2019	A	1.9

In [28]:

```
df.loc[['002','004'],['Year','Measure1']]
```

Out[28]:

	Year	Measure1
002	2018	3.0
004	2019	1.9

Pandas Slicing

In [29]:

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

Out[29]:

	Year	Class_Name	Measure1
003	2018	B	2.4
004	2019	A	1.9

In [30]:

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

Out[30]:

	Year	Class_Name
001	2017	A
002	2018	A

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

Pandas Slicing Cont.

Can also slice using labels:

In [31]:

```
df.loc['002':'004']
```

Out[31]:

	Year	Class_Name	Measure1
002	2018	A	3.0
003	2018	B	2.4
004	2019	A	1.9

In [32]:

```
df.loc['002':'004',:'Class_Name']
```

Out[32]:

	Year	Class_Name
002	2018	A
003	2018	B
004	2019	A

NOTE: `.loc` is **inclusive**

Pandas Slicing Cont.

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

In [33]:

```
df.loc[:,'Measure1']
```

Out[33]:

```
001    2.1
002    3.0
003    2.4
004    1.9
Name: Measure1, dtype: float64
```

In [34]:

```
df.iloc[2:,:]
```

Out[34]:

	Year	Class_Name	Measure1
003	2018	B	2.4
004	2019	A	1.9

Pandas Indexing Cont.

Shortcut for indexing:

In [35]:

```
df['Class_Name']
```

Out[35]:

```
001    A
002    A
003    B
004    A
Name: Class_Name, dtype: object
```

In [36]:

```
# can use dot notation if there is no space in label
df.Class_Name
```

Out[36]:

```
001    A
002    A
003    B
```

```
004    A
Name: Class_Name, dtype: object
```

Panda Selection Chaining

Get 'Year' and 'Measure1' for first 3 rows:

In [37]:

```
df.iloc[:3].loc[:,['Year','Measure1']]
```

Out[37]:

	Year	Measure1
001	2017	2.1
002	2018	3.0
003	2018	2.4

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

In [38]:

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

Out[38]:

	Class_Name	Measure1
001	A	2.1
003	B	2.4

Panda Selection Chaining Cont.

For record '001' get last two columns?:

In [39]:

```
# reduce the amount of error information printed
%xmode Minimal
```

```
Exception reporting mode: Minimal
```

In [40]:

```
# Note: add 'raises-exception' tag to cell to continue running after exception

df.loc['001'].iloc[:,-2:] # row with label '001', then all rows, last two columns?
```

```
IndexingError: Too many indexers
```

In [41]:

```
df.loc['001']
```

Out[41]:

```
Year          2017
Class_Name       A
```

```
Measure1        2.1
Name: 001, dtype: object
```

In [42]:

```
df.loc['001'].iloc[-2:] # row with label '001', last two elements of Series
```

Out[42]:

```
Class_Name      A
Measure1      2.1
Name: 001, dtype: object
```

Pandas `head` and `tail`

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

In [43]:

```
df.head() # first 5 rows by default
```

Out[43]:

	Year	Class_Name	Measure1
001	2017	A	2.1
002	2018	A	3.0
003	2018	B	2.4
004	2019	A	1.9

In [44]:

```
df.tail(2) # only print last 2 rows
```

Out[44]:

	Year	Class_Name	Measure1
003	2018	B	2.4
004	2019	A	1.9

Pandas Boolean Mask

In [45]:

```
# Which rows have Class_Name of 'A'?
df.loc[:,'Class_Name'] == 'A'
```

Out[45]:

```
001      True
002      True
003     False
004      True
Name: Class_Name, dtype: bool
```

In [46]:

```
# Get all data for rows with with Class_Name 'A'
df.loc[df.Class_Name == 'A']
```

Out[46]:

	Year	Class_Name	Measure1
001	2017	A	2.1
002	2018	A	3.0
004	2019	A	1.9

In [47]:

```
# Get Measure1 for all records for Class_Name 'A'
df.loc[df.Class_Name == 'A','Measure1']
```

Out[47]:

```
001    2.1
002    3.0
004    1.9
Name: Measure1, dtype: float64
```

Pandas Boolean Mask Cont.

Get all records for class 'A' before 2019

In [48]:

```
df.loc[(df.Class_Name == 'A') & (df.Year < 2019)]
```

Out[48]:

	Year	Class_Name	Measure1
001	2017	A	2.1
002	2018	A	3.0

Get all records in a set of years:

In [49]:

```
df.loc[df.Year.isin([2017,2019])]
```

Out[49]:

	Year	Class_Name	Measure1
001	2017	A	2.1
004	2019	A	1.9

Pandas Selection Review

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

Pandas Sorting

In [50]:

```
df.sort_values(by=['Measure1']).head(3)
```

Out[50]:

	Year	Class_Name	Measure1
004	2019	A	1.9
001	2017	A	2.1
003	2018	B	2.4

In [51]:

```
df.sort_values(by=['Measure1'],ascending=False).head(3)
```

Out[51]:

	Year	Class_Name	Measure1
002	2018	A	3.0
003	2018	B	2.4
001	2017	A	2.1

In [52]:

```
df.sort_values(by=['Year','Measure1']).head(3)
```

Out[52]:

	Year	Class_Name	Measure1
001	2017	A	2.1
003	2018	B	2.4
002	2018	A	3.0

Questions?

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 #

In [53]:

```
!head ../data/yellowcab_demo_withdaycategories.csv
```

```
# A sample of yellocab taxi trip data from Jan 201
7

pickup_datetime,dropoff_datetime,trip_distance,far
e_amount,tip_amount,payment_type,day_of_week,is_we
ekend

2017-01-05 14:49:04,2017-01-05 14:53:53,0.89,5.5,
1.26,Credit card,3,True
```

```
2017-01-15 01:07:22,2017-01-15 01:26:47,2.7,14.0,
0.0,Cash,6,False

2017-01-29 09:55:00,2017-01-29 10:04:43,1.41,8.0,
0.0,Cash,6,False

2017-01-10 05:40:12,2017-01-10 05:42:22,0.4,4.0,0.
0,Cash,1,True

2017-01-06 17:02:48,2017-01-06 17:16:10,2.3,11.0,
0.0,Cash,4,True

2017-01-14 19:03:14,2017-01-14 19:08:41,0.8,5.5,,C
redit card,5,True

2017-01-06 18:51:52,2017-01-06 18:55:45,0.2,4.5,0.
0,Cash,4,True

2017-01-04 20:47:30,2017-01-04 21:01:24,2.68,11.
```

```
5,,Credit card,2,True
```

Loading Datasets with Pandas

In [54]:

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

In [55]:

```
# display first 5 rows
df.head(5)
```

Out[55]:

	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	True
1	2017-01-15 01:07:22	2017-01-15 01:26:47	2.70	14.0	0.00	Cash	6	False
2	2017-01-29 09:55:00	2017-01-29 10:04:43	1.41	8.0	0.00	Cash	6	False
3	2017-01-10 05:40:12	2017-01-10 05:42:22	0.40	4.0	0.00	Cash	1	True

	pickup_datetime	dropoff_datetime	trip_distance	fare_amount	tip_amount	payment_type	day_of_week	is_weekend
4	2017-01-06 17:02:48	2017-01-06 17:16:10	2.30	11.0	0.00	Cash	4	True

Get Size of Dataset

In [56]:

```
df.shape
```

Out[56]:

```
(1000, 8)
```

In [57]:

```
# number of rows
f'{df.shape[0]} rows'
```

Out[57]:

```
'1000 rows'
```

In [58]:

```
# number of columns
f'{df.shape[1]} columns'
```

Out[58]:

```
'8 columns'
```

In [59]:

```
'number of rows: {}, number of columns: {}'.format(*df.shape)
```

Out[59]:

```
'number of rows: 1000, number of columns: 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 [60]:

```
df.shape
```

Out[60]:

```
(1000, 8)
```

In [61]:

```
# call .format( (2,8) )
'number of rows: {}, number of columns: {}'.format(df.shape)
```

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

In [62]:

```
# call .format(2,8)
'number of rows: {}, number of columns: {}'.format(*df.shape)
```

Out[62]:

```
'number of rows: 1000, number of columns: 8'
```

What are the column names?

In [63]:

```
df.columns
```

Out[63]:

```
Index(['pickup_datetime', 'dropoff_datetime', 'tri
p_distance', 'fare_amount',
       'tip_amount', 'payment_type', 'day_of_wee
k', 'is_weekend'],
      dtype='object')
```

In [64]:

```
df.columns.values
```

Out[64]:

```
array(['pickup_datetime', 'dropoff_datetime', 'tri
p_distance',
       'fare_amount', 'tip_amount', 'payment_typ
e', 'day_of_week',
       'is_weekend'], dtype=object)
```

In [65]:

```
df.columns.tolist()
```

Out[65]:

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

What are the column datatypes?

In [66]:

```
df.dtypes
```

Out[66]:

```
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 [67]:

```
type(df.dtypes)
```

Out[67]:

```
pandas.core.series.Series
```

Get Summary Info for DataFrame

In [68]:

```
df.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), in
t64(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

- **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

In [69]:

```
df.trip_distance.min()
```

Out[69]:

```
0.0
```

In [70]:

```
df.trip_distance.max()
```

Out[70]:

```
32.77
```

In [71]:

```
df.min(numeric_only=True)
```

Out[71]:

```
trip_distance     0.0
fare_amount       2.5
tip_amount        0.0
day_of_week         0
```

```
is_weekend        False
dtype: object
```

In [72]:

```
df.max(numeric_only=True)
```

Out[72]:

```
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

- Sample Mean

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

In [73]:

```
df.fare_amount.mean()
```

Out[73]:

12.4426

In [74]:

```
print(f'{df.fare_amount.mean() = :0.2f}')
```

df.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

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

In [75]:

```
df.fare_amount.median()
```

Out[75]:

```
9.0
```

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

Numeric: Quantiles/Percentiles

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

In [76]:

```
df.fare_amount.quantile(.95) # 95% of the data is less than or equal to x?
```

Out[76]:

```
33.5
```

In [77]:

```
df.fare_amount.quantile([.05,.95]) # 90% of the data is between 4 and 33.5
```

Out[77]:

```
0.05     4.0
0.95    33.5
Name: fare_amount, dtype: float64
```

In [78]:

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

Out[78]:

```
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

- Sample Variance

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

In [79]:

```
df.fare_amount.var().round(3)
```

Out[79]:

```
116.809
```

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

Numeric: Spread with Standard Deviation

- Sample Standard Deviation

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

In [80]:

```
df.fare_amount.std().round(3)
```

Out[80]:

```
10.808
```

- Back in original scale of dollars
- Sensitive to outliers

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 [81]:

```
df.fare_amount.quantile(.75) - df.fare_amount.quantile(.25)
```

Out[81]:

```
7.5
```

- IQR is robust to outliers

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 [82]:

```
df.fare_amount.skew()
```

Out[82]:

```
2.882730031010152
```

Easier to understand with a plot...

Numeric Summary Stats with `.describe`

In [83]:

```
df.describe()
```

Out[83]:

	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

Applying Functions to Groups of Data

In [84]:

```
df.groupby('payment_type')
```

Out[84]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f40577e6d00>
```

In [85]:

```
df.groupby('payment_type').mean()
```

Out[85]:

	trip_distance	fare_amount	tip_amount	day_of_week	is_weekend
payment_type					
Cash	2.732209	11.856716	0.000000	2.898507	0.847761
Credit card	2.961870	12.761086	2.683322	3.039216	0.850679
No charge	0.500000	5.000000	0.000000	0.500000	1.000000

In [86]:

```
# applying multiple aggregation functions
df.groupby('payment_type')['trip_distance'].agg(['mean','median'])
```

Out[86]:

	mean	median

payment_type	mean	median
payment_type		
Cash	2.732209	1.37
Credit card	2.961870	1.70
No charge	0.500000	0.50

In [87]:

```
df[df.payment_type.isin(['Cash','Credit card'])].groupby(['payment_type','is_weekend']).trip_distance.agg(['mean','median'])
```

Out[87]:

		mean	median
payment_type	is_weekend		
Cash	False	3.507059	2.10
	True	2.593063	1.28
Credit card	False	3.304646	1.74
	True	2.901702	1.70

Aside: Dealing with long chains

- long chains may not be visible in notebooks

In [88]:

```
# df[df.payment_type.isin(['Cash','Credit card'])].groupby(['payment_type','is_weekend']).trip_distance.agg(['mean','median'])
```

In [89]:

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

Out[89]:

		mean	median
payment_type	**is_weekend**		
Cash	**False**	3.507059	2.10
	True	2.593063	1.28
Credit card	**False**	3.304646	1.74
	True	2.901702	1.70

In [90]:

```
# wrap in parentheses
(df[df.payment_type.isin(['Cash','Credit card'])]
 .groupby(['payment_type','is_weekend'])
```

```
    .trip_distance.agg(['mean','median'])
)
```

Out[90]:

payment_type	is_weekend	mean	median
Cash	False	3.507059	2.10
	True	2.593063	1.28
Credit card	False	3.304646	1.74
	True	2.901702	1.70

Questions?

Visualizations in Python

- plotting with `matplotlib.pyplot`
- plotting with `pandas`
- plotting with `seaborn`
- need interactive plots? `plotly`

Matplotlib.pyplot

In [91]:

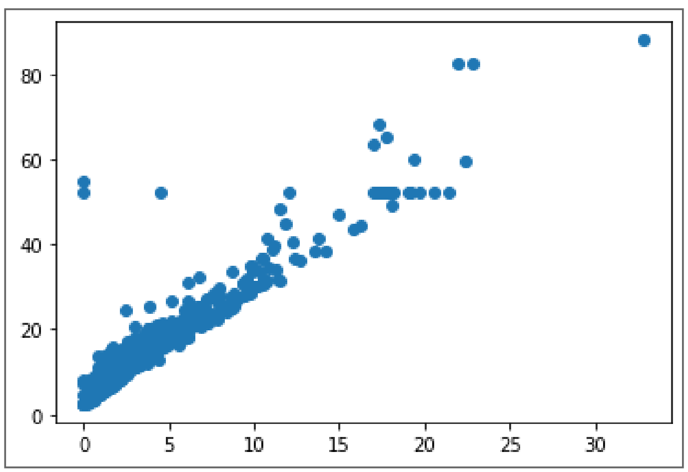
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [92]:

```
plt.scatter(df.trip_distance,df.fare_amount)
```

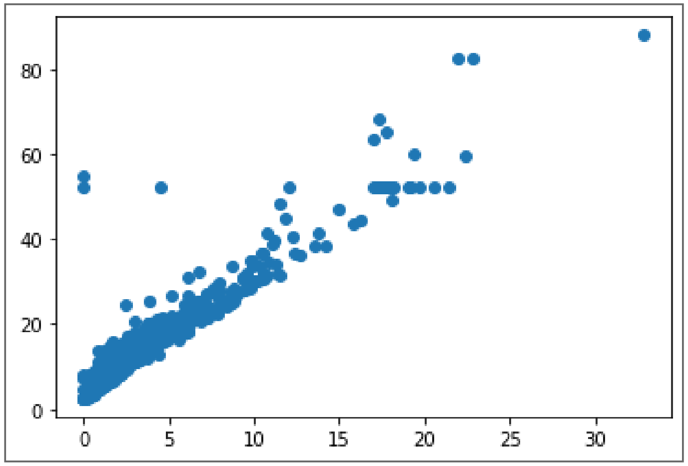
Out[92]:

```
<matplotlib.collections.PathCollection at 0x7f409f77b8e0>
```



In [93]:

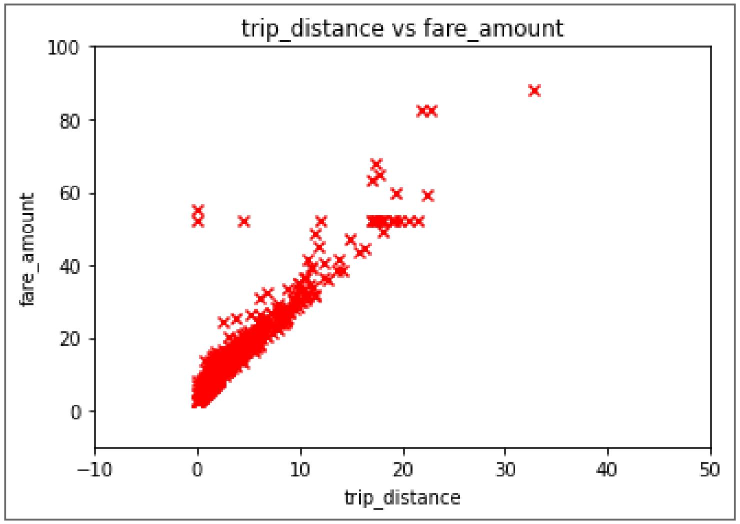
```
plt.scatter(df.trip_distance,df.fare_amount);
```



Matplotlib Axes

In [94]:

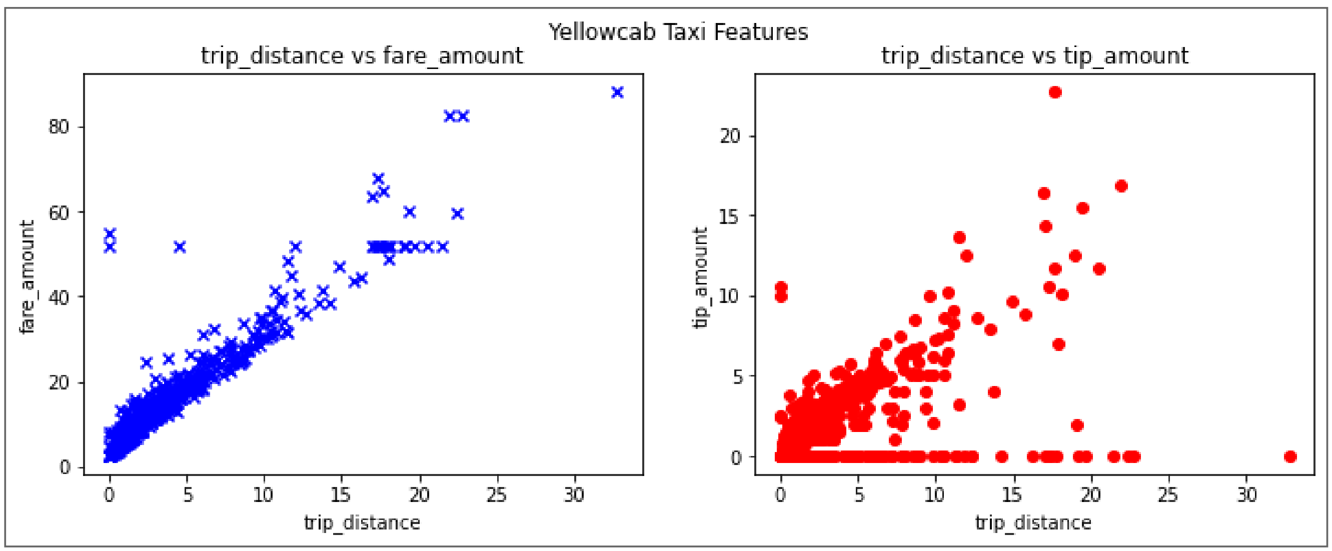
```
fig,ax = plt.subplots(1,1,figsize=(6,4))

ax.scatter(x=df.trip_distance,
           y=df.fare_amount,
           marker='x',
           color='red'
          )

ax.set_xlabel('trip_distance')
ax.set_ylabel('fare_amount')

ax.set_xlim([-10,50])
ax.set_ylim([-10,100])

ax.set_title('trip_distance vs fare_amount');
```



Matplotlib: Subplots, Figure and Axis

In [95]:

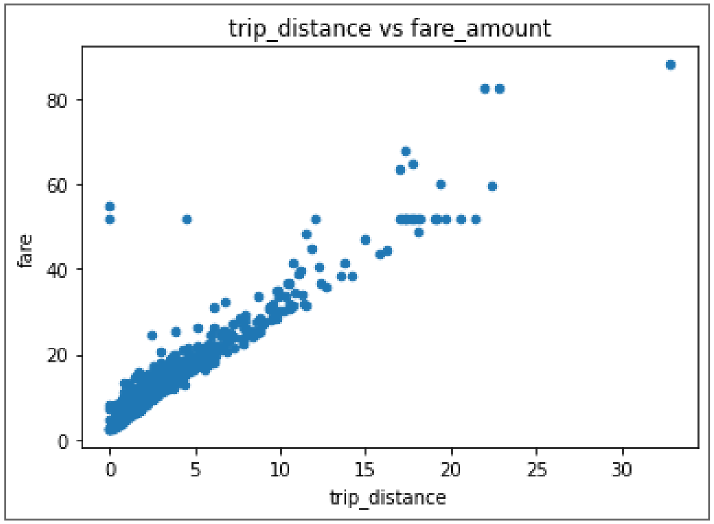
```
fig,ax = plt.subplots(1,2,figsize=(12,4))

ax[0].scatter(df.trip_distance,df.fare_amount,marker='x',color='blue')
ax[1].scatter(df.trip_distance,df.tip_amount,color='red');

ax[0].set_xlabel('trip_distance')
ax[1].set_xlabel('trip_distance')

ax[0].set_ylabel('fare_amount'), ax[1].set_ylabel('tip_amount')

ax[0].set_title('trip_distance vs fare_amount')
ax[1].set_title('trip_distance vs tip_amount')

fig.suptitle('Yellowcab Taxi Features');
```



Plotting via Pandas

In [96]:

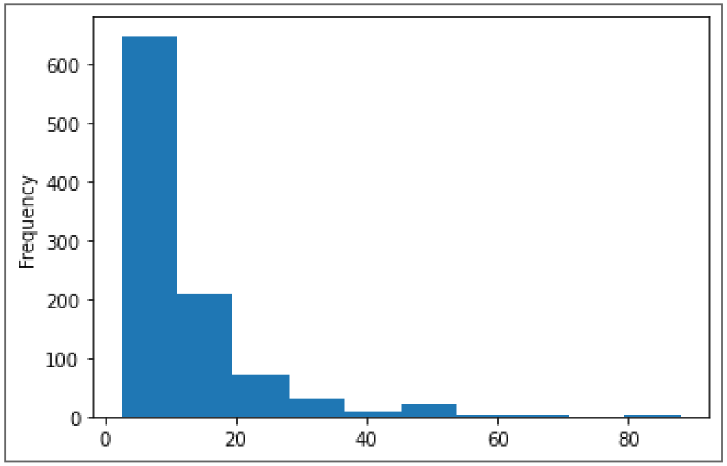
```
ax = df.plot.scatter(x='trip_distance',y='fare_amount');
ax.set_ylabel('fare')
ax.set_title('trip_distance vs fare_amount');
```



Univariate Distribution: Histogram

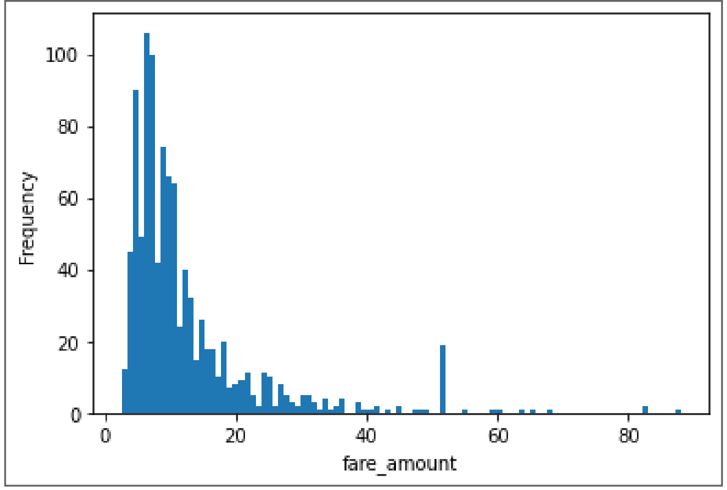
In [97]:

```
df.fare_amount.plot.hist();
```



In [98]:

```
ax = df.fare_amount.plot.hist(bins=100)
ax.set_xlabel('fare_amount');
```

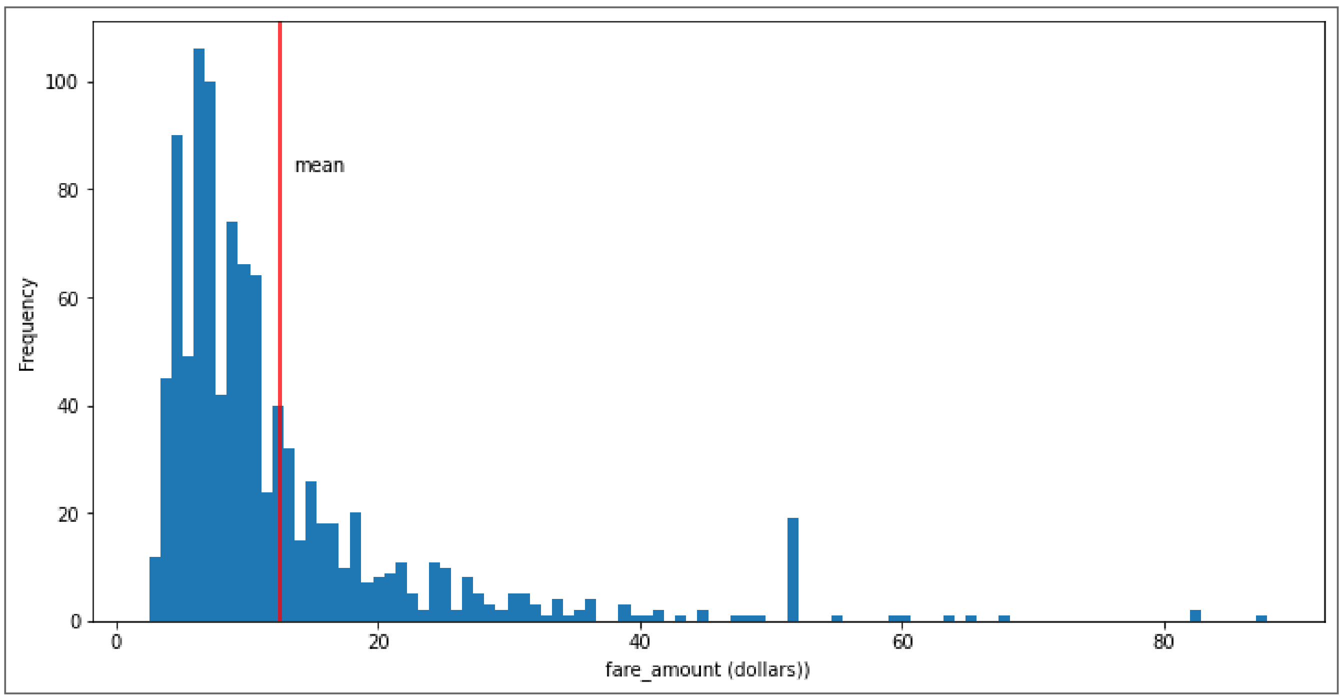


Frequency
fare_amount
0
20
40
60
80
100

Univariate Distribution: Histogram

In [99]:

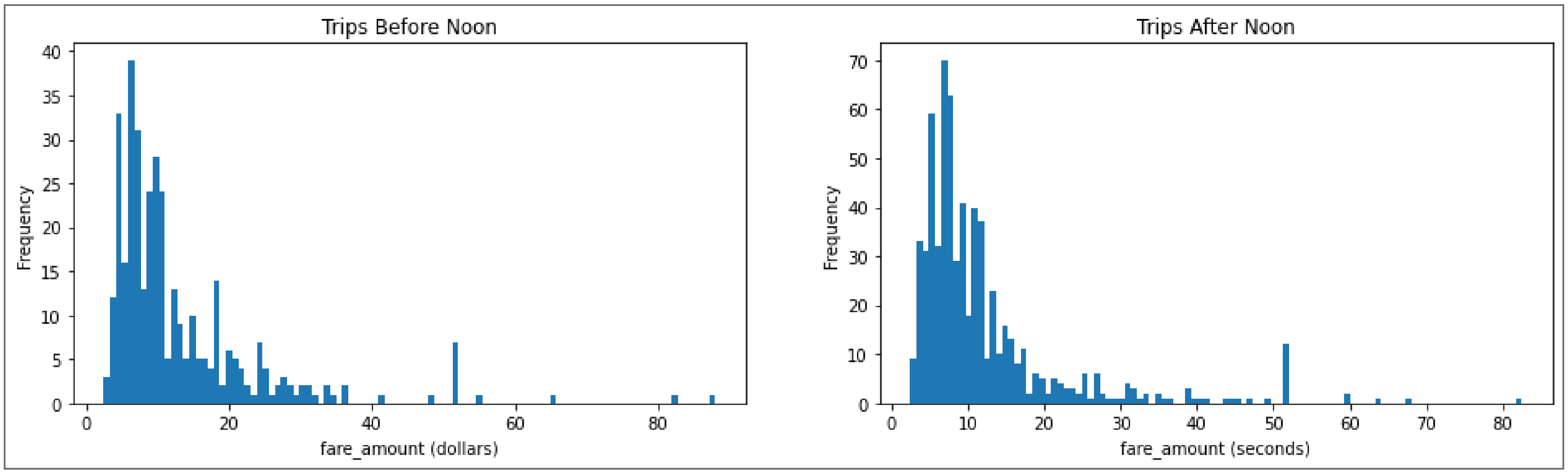
```
fig,ax = plt.subplots(1,1,figsize=(12,6));

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

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

# add some text
ax.text(df.fare_amount.mean()+1,ax.get_ylim()[1]*.75,'mean');
```



Subplots with Pandas

In [100]:

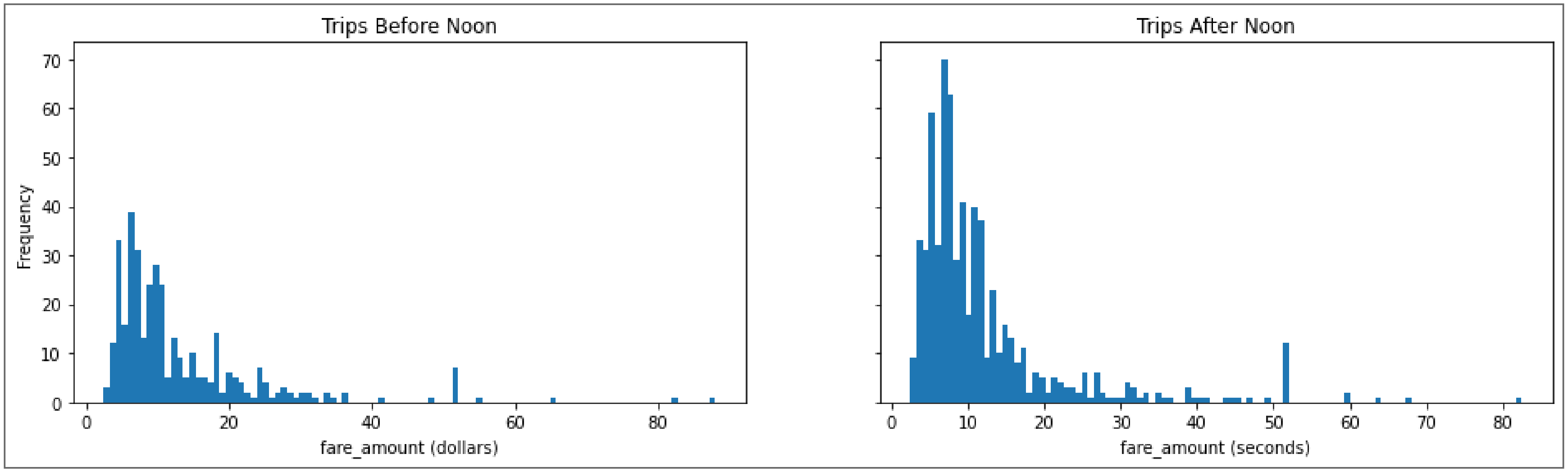
```
fig,ax = plt.subplots(1,2,figsize=(16,4))
df[df.pickup_datetime.dt.hour < 12].fare_amount.plot.hist(bins=100,ax=ax[0]);
ax[0].set_xlabel('fare_amount (dollars)');
ax[0].set_title('Trips Before Noon');
df[df.pickup_datetime.dt.hour >= 12].fare_amount.plot.hist(bins=100,ax=ax[1]);
ax[1].set_xlabel('fare_amount (seconds)');
ax[1].set_title('Trips After Noon');
```



Sharing Axes

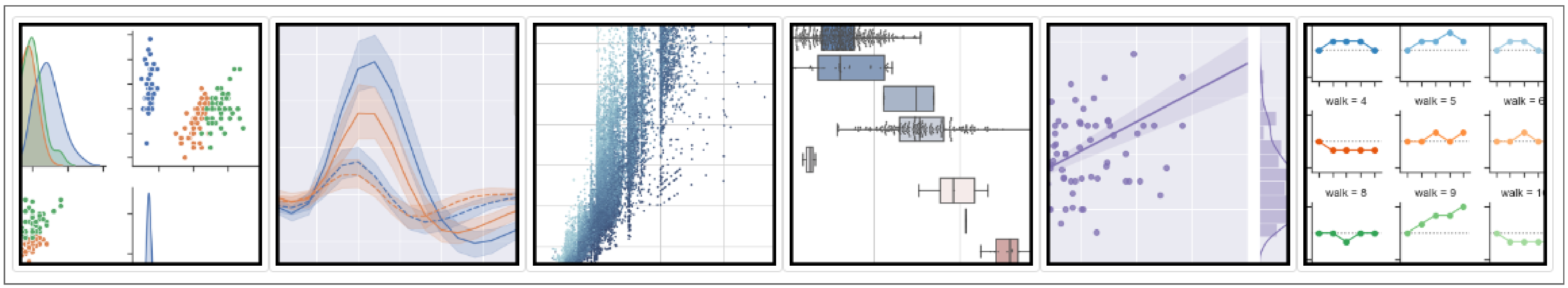
In [101]:

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

df[df.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[df.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');
```



Plotting with Seaborn

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



In [102]:

```
import seaborn as sns
sns.__version__
```

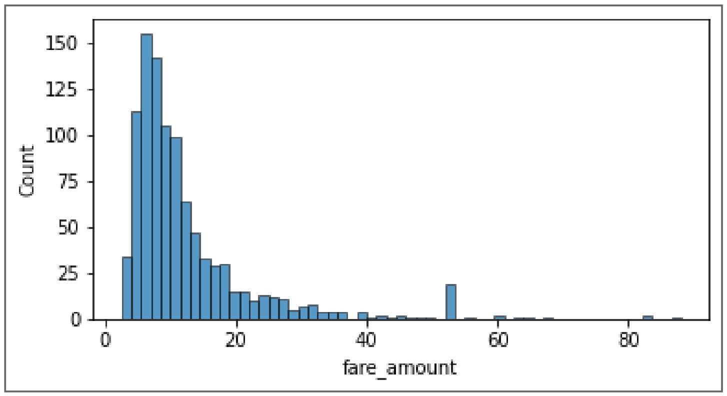
Out[102]:

```
'0.11.2'
```

Univariate Distribution with Seaborn Histplot

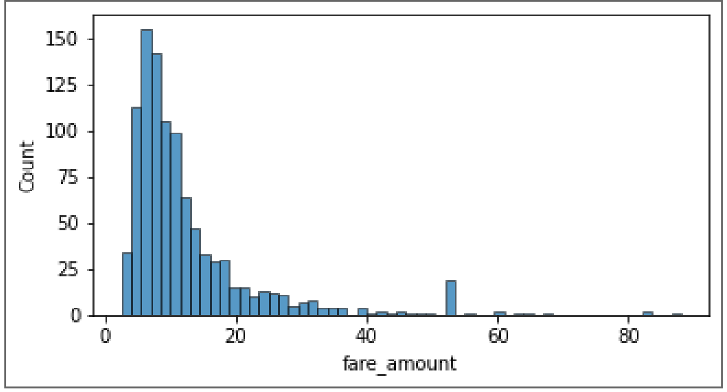
In [103]:

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

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



In [104]:

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

sns.histplot(x=df.fare_amount,ax=ax);
```

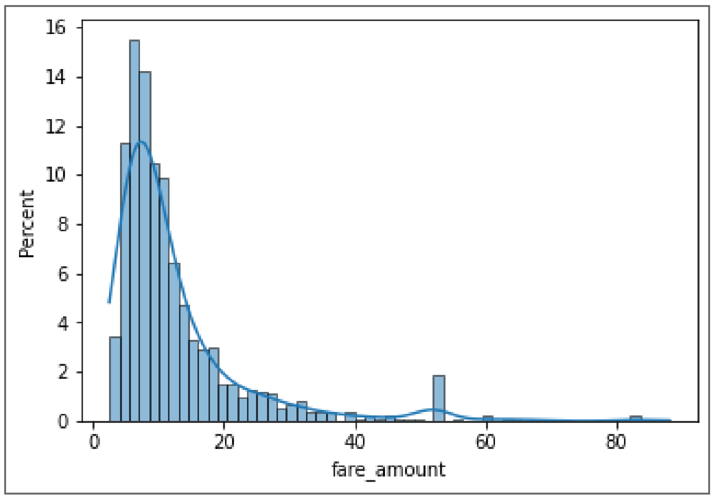


150
125
100
75
50
25
0
Count
0
20
40
60
80
fare_amount

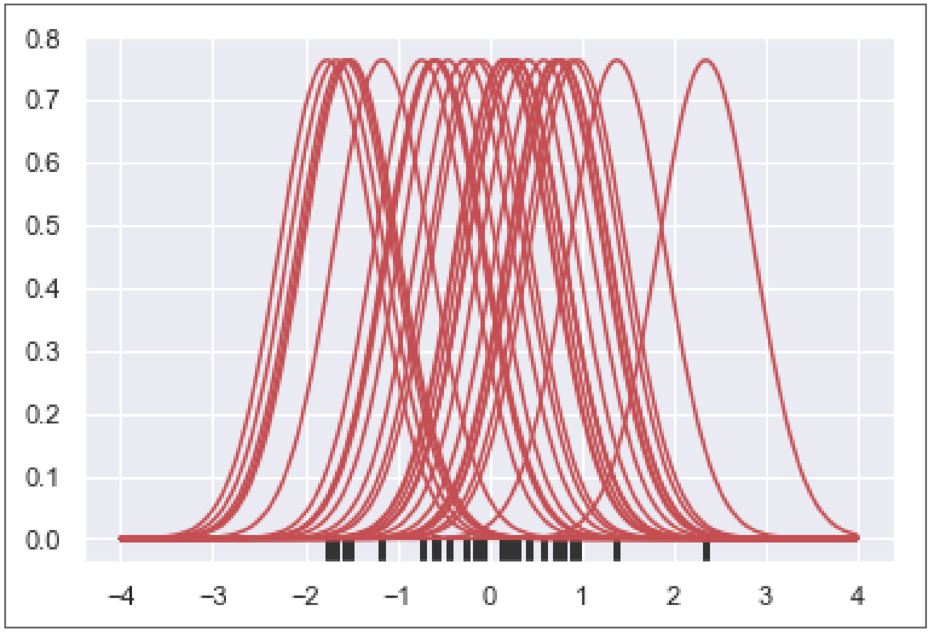
Univariate Distribution with Seaborn Histplot Cont.

In [105]:

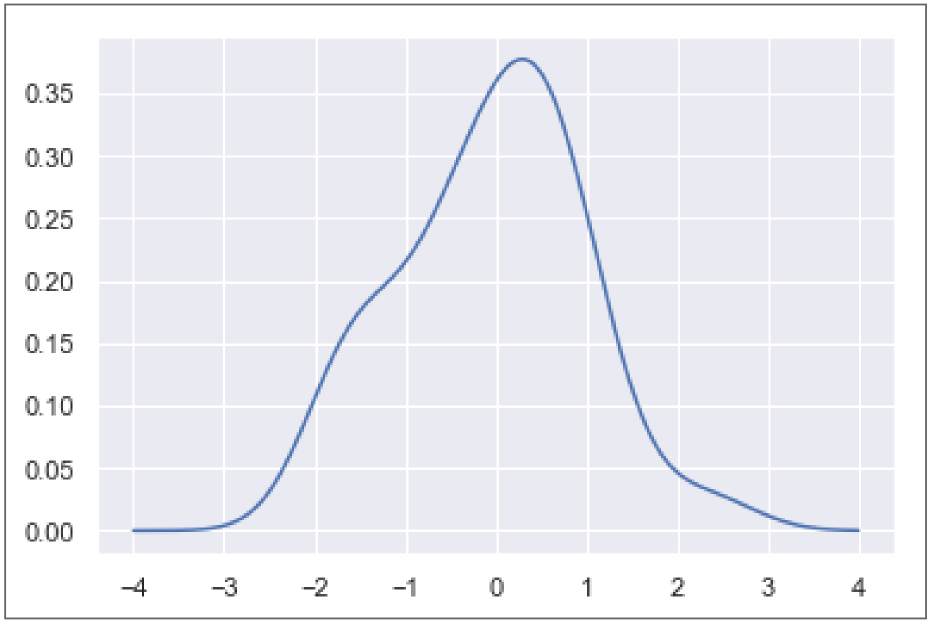
```
fig,ax = plt.subplots(1,1,figsize=(6,4))

# many other parameters to play with
sns.histplot(x='fare_amount',data=df,ax=ax,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

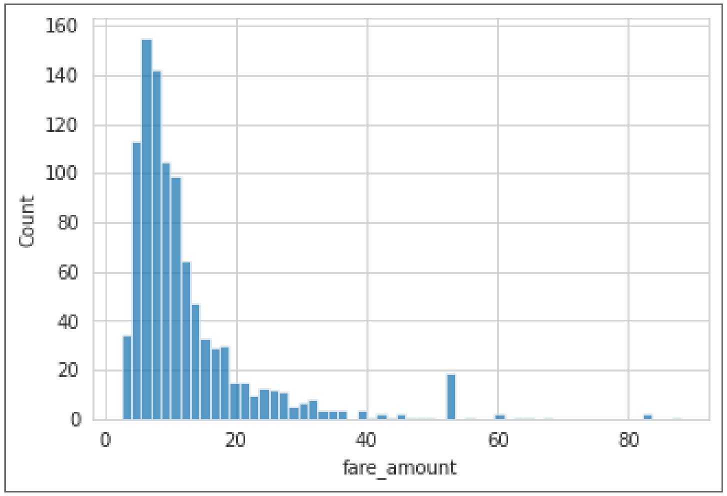


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4
−3
−2
−1
0
1
2
3
4

Seaborn Styles

In [106]:

```
# for a single plot using a context
with sns.axes_style('whitegrid'):
    sns.histplot(x=df.fare_amount);
```

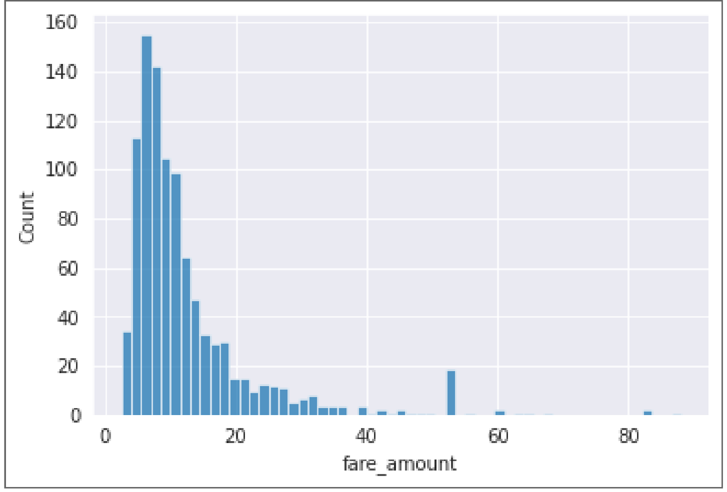


In [107]:

```
# set style globally
sns.set_style('darkgrid')
```

In [108]:

```
sns.histplot(x=df.fare_amount);
```

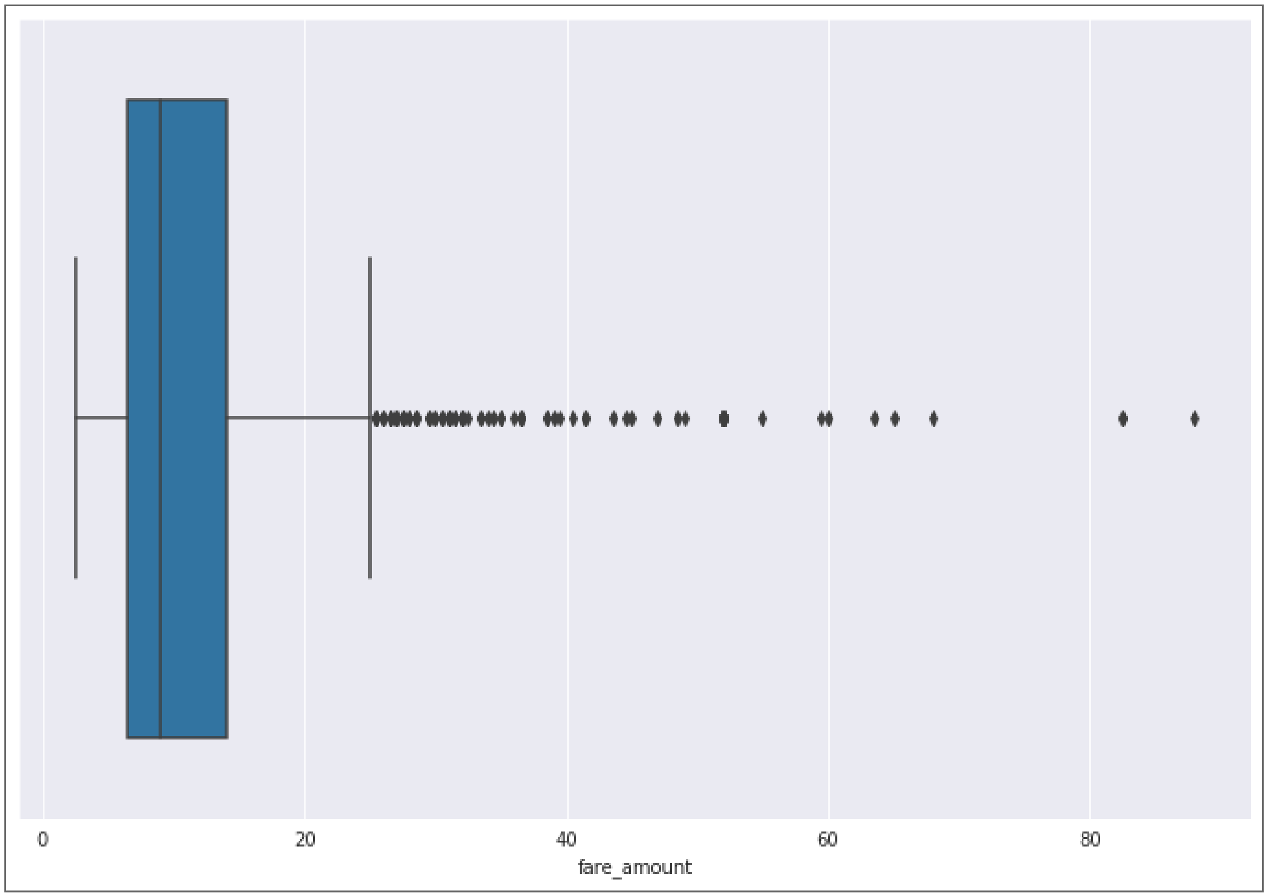


160
140
120
100
80
60
40
20
0
Count
0
20
40
60
80
fare_amount

Univariate Distributions: Boxplot

In [109]:

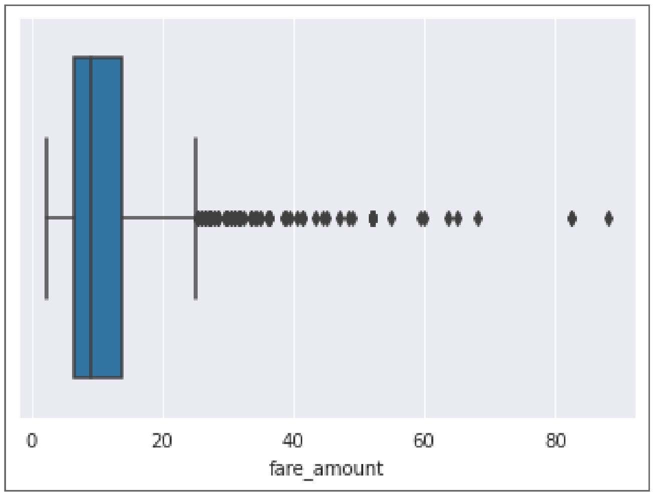
```
fig,ax = plt.subplots(1,1,figsize=(12,8))

sns.boxplot(x=df.fare_amount,ax=ax);
```



Univariate Distributions: Boxplot

In [110]:

```
fig,ax = plt.subplots(1,1,figsize=(6,4))

sns.boxplot(x=df.fare_amount,ax=ax);
```

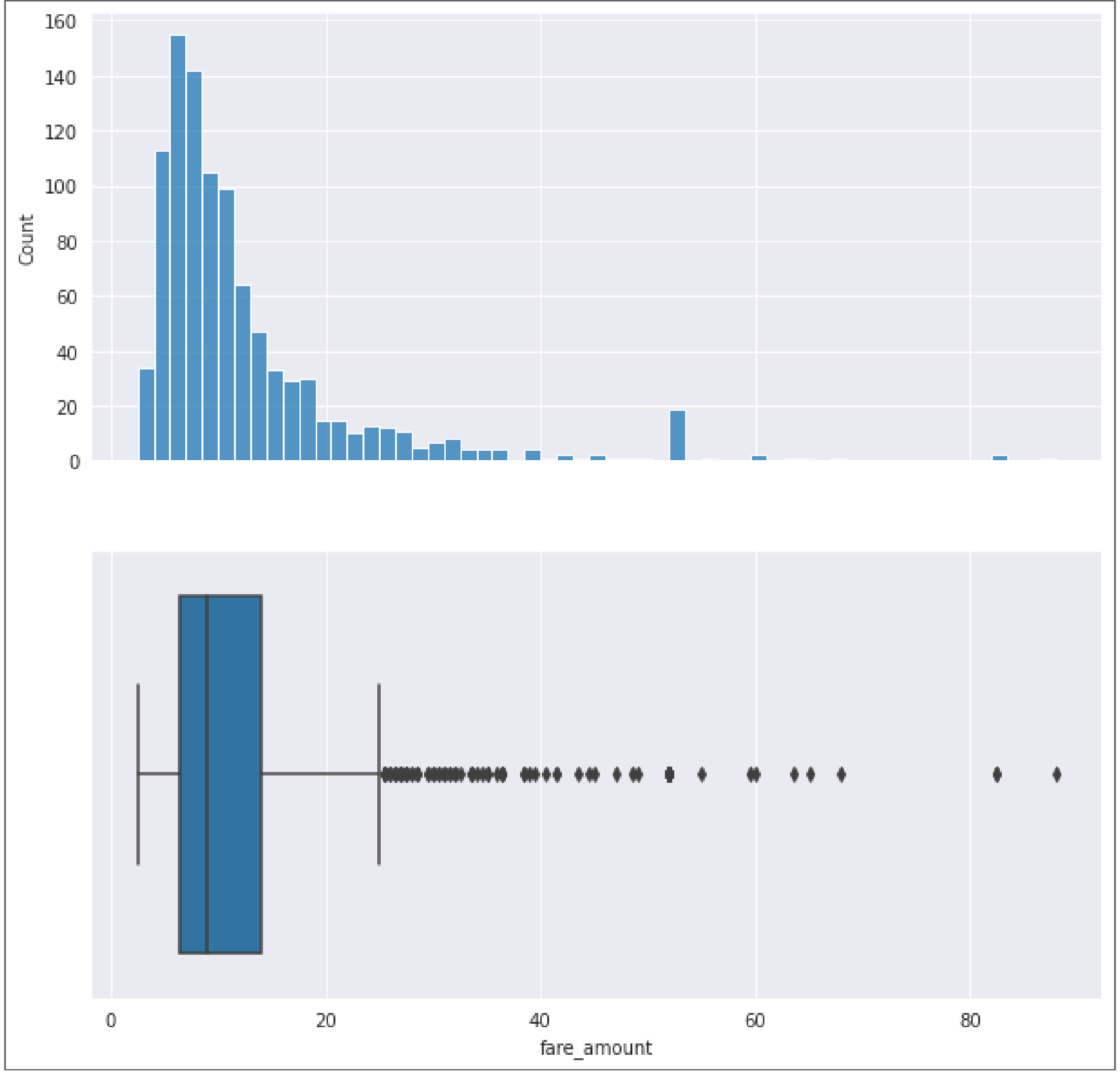


- first quartile
- second quartile (Median)
- third quartile
- whiskers (usually 1.5*IQR)
- outliers

Combining Plots with Subplots

In [111]:

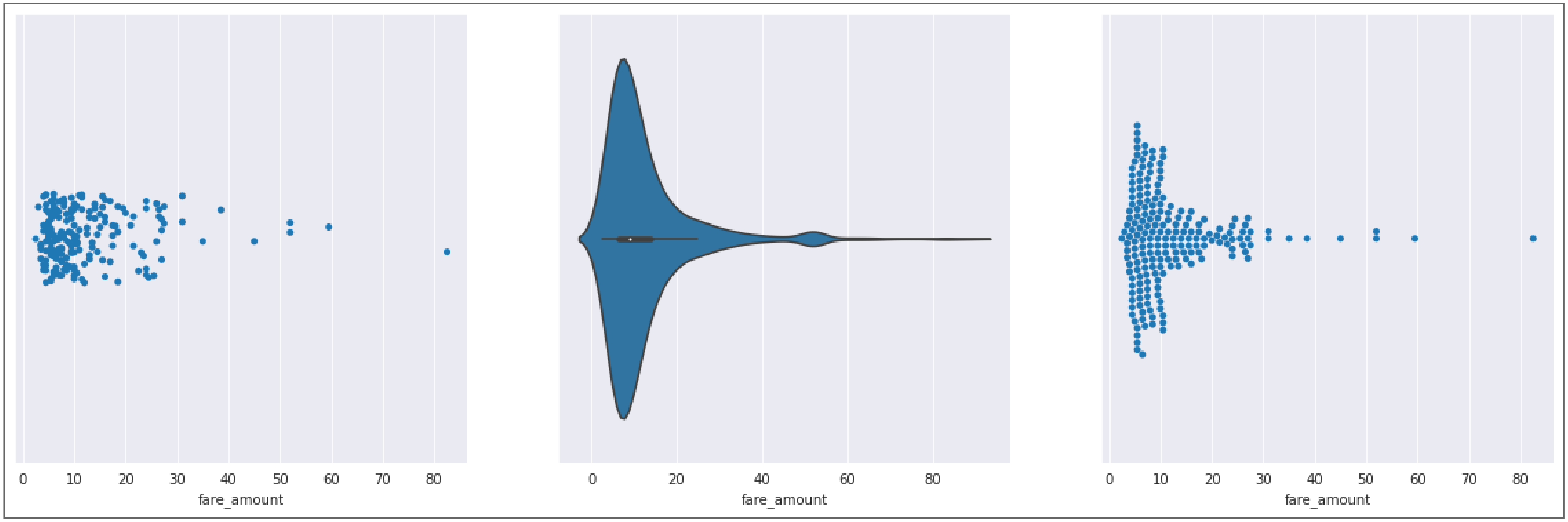
```
fig,ax = plt.subplots(2,1,figsize=(10,10), sharex=True)

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


Count
160
140
120
100
80
60
40
20
0
0
20
40
60
80
fare_amount

Other Univariate Distribution Visualizations

In [112]:

```
fig,ax = plt.subplots(1,3,figsize=(20,6))

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



Bivariate: Evaluating Correlation

- **Correlation:** the degree to which two variables are linearly related
- Pearson Correlation Coefficient: $\rho_{XY} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$
- 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)

In [113]:

```
df.trip_distance.corr(df.fare_amount)
```

Out[113]:

```
0.948701076897808
```

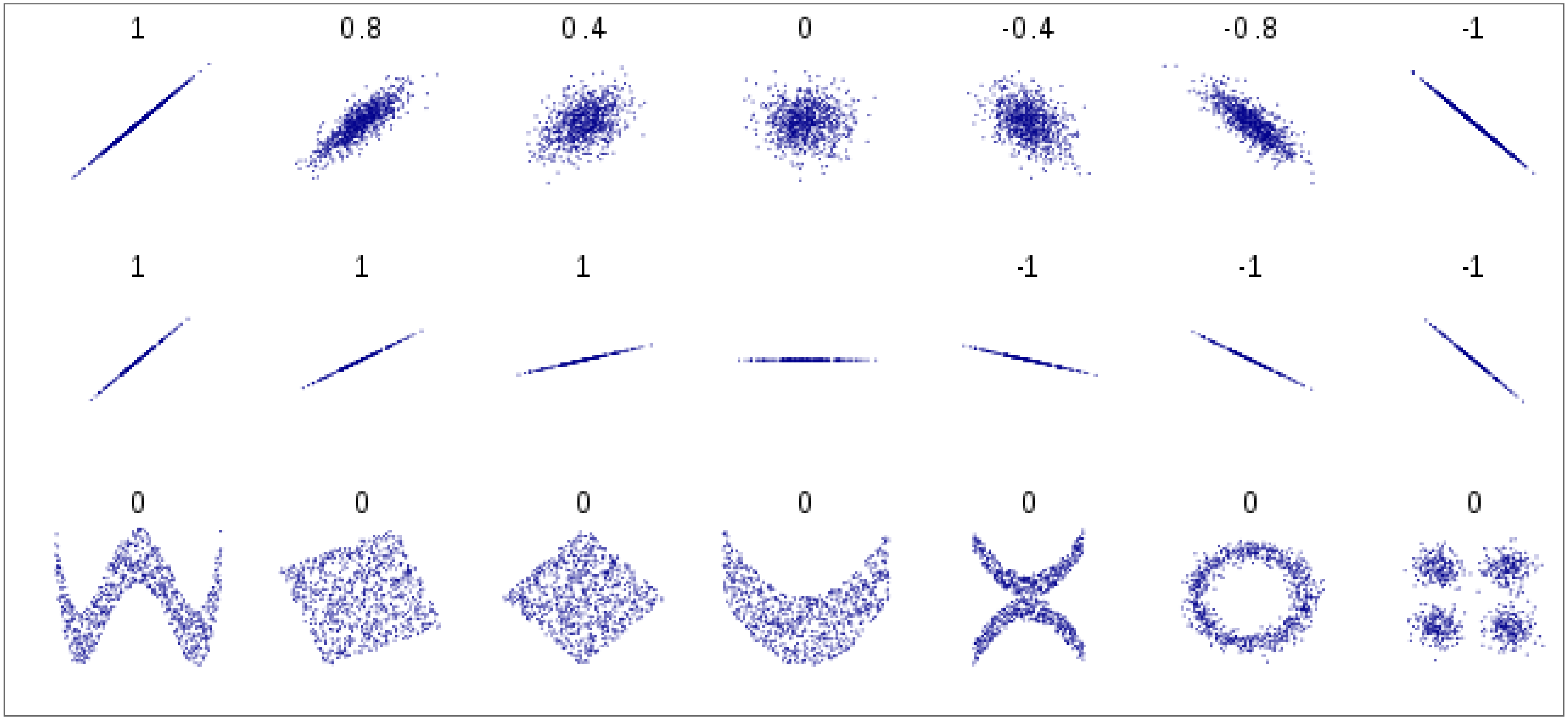
In [114]:

```
from scipy.stats import pearsonr
r,p = pearsonr(df.trip_distance, df.fare_amount)
r,p
```

Out[114]:

```
(0.9487010768978079, 0.0)
```

Pearson Correlation

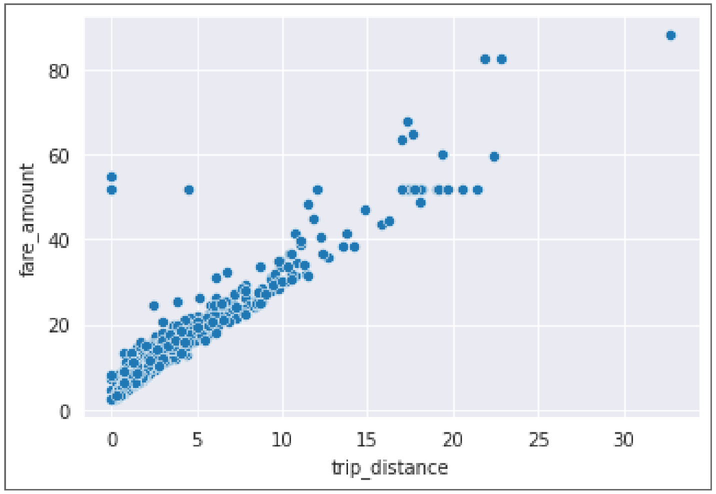


</center>

Bivariate: Scatterplot

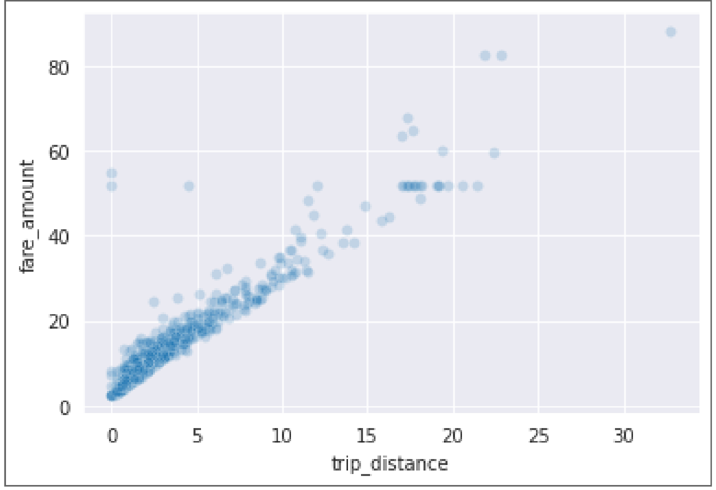
In [115]:

```
sns.scatterplot(x='trip_distance',y='fare_amount',data=df);
```



In [116]:

```
sns.scatterplot(x='trip_distance',y='fare_amount',data=df,alpha=0.2);
```

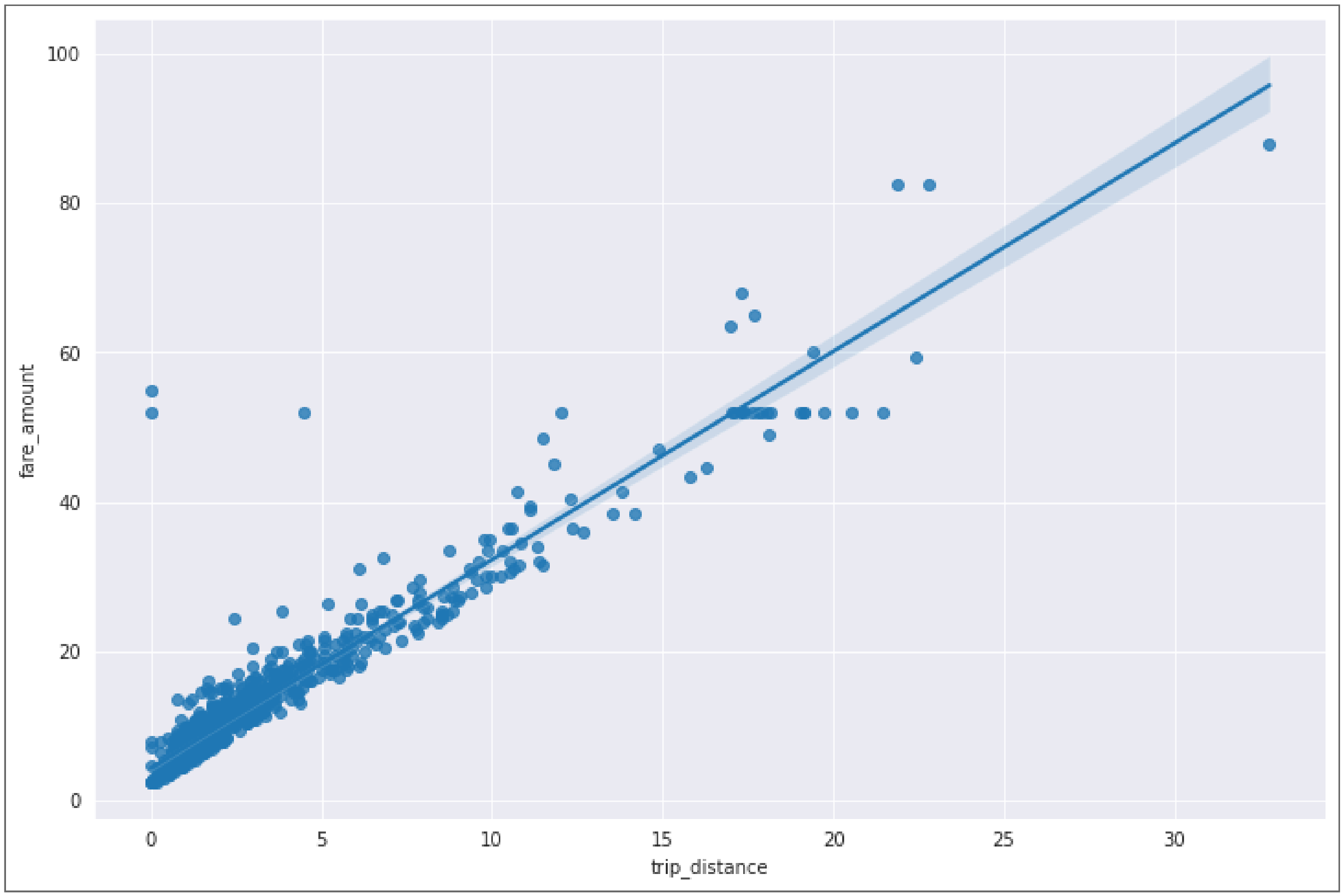


fare_amount
trip_distance
0
20
40
60
80
0
5
10
15
20
25
30

Bivariate: Add Regression Line

In [117]:

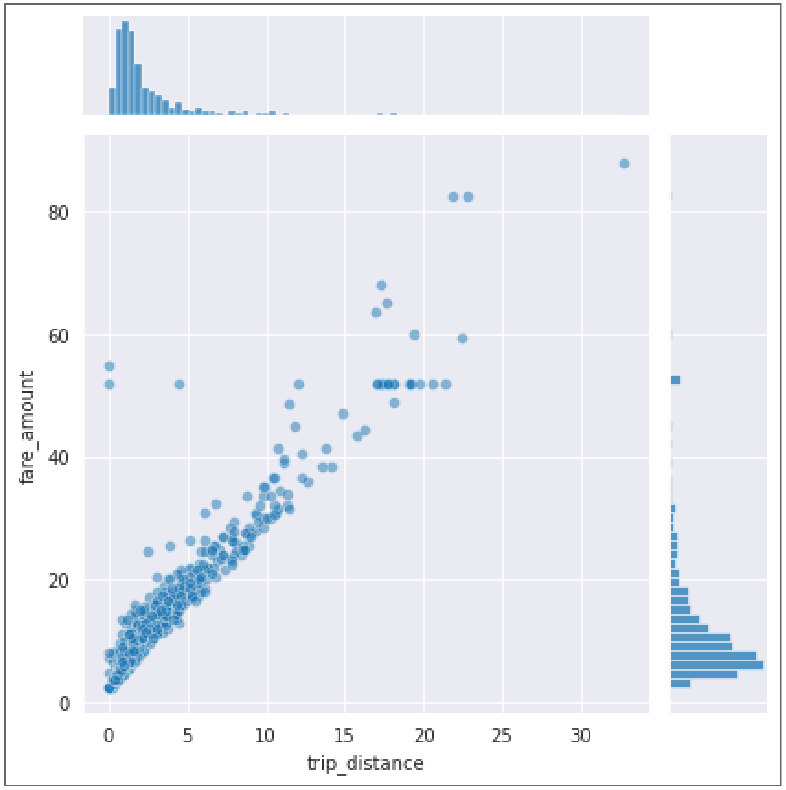
```
fig,ax = plt.subplots(1,1,figsize=(12,8))

sns.regplot(x='trip_distance',y='fare_amount',data=df,ax=ax);
```



Bivariate: Joint Plot

In [118]:

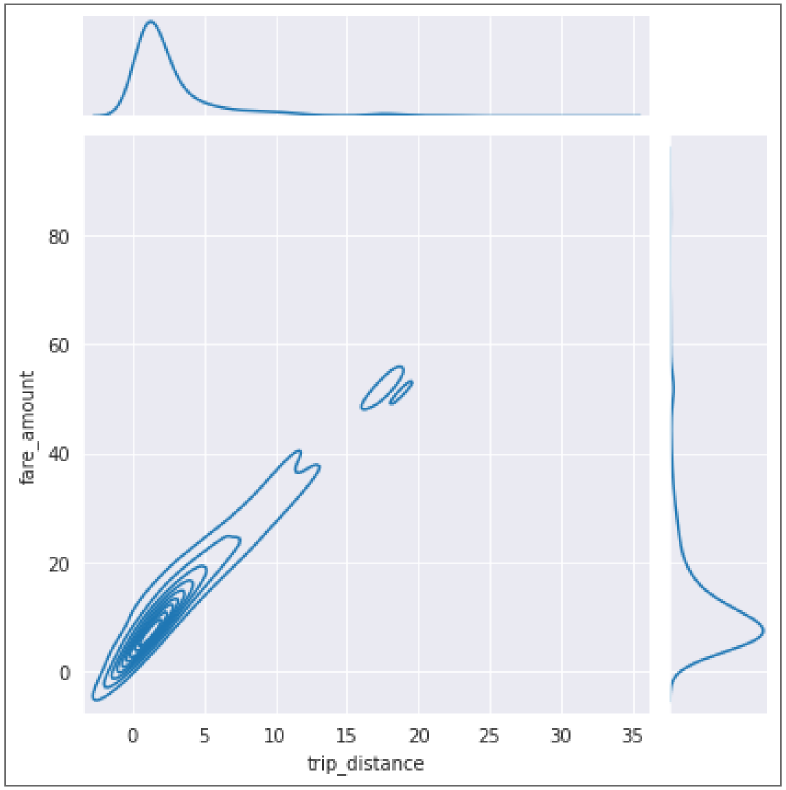
```
sns.jointplot(x='trip_distance',y='fare_amount',data=df,alpha=0.5);
```



Bivariate: Joint Plot with KDE

In [119]:

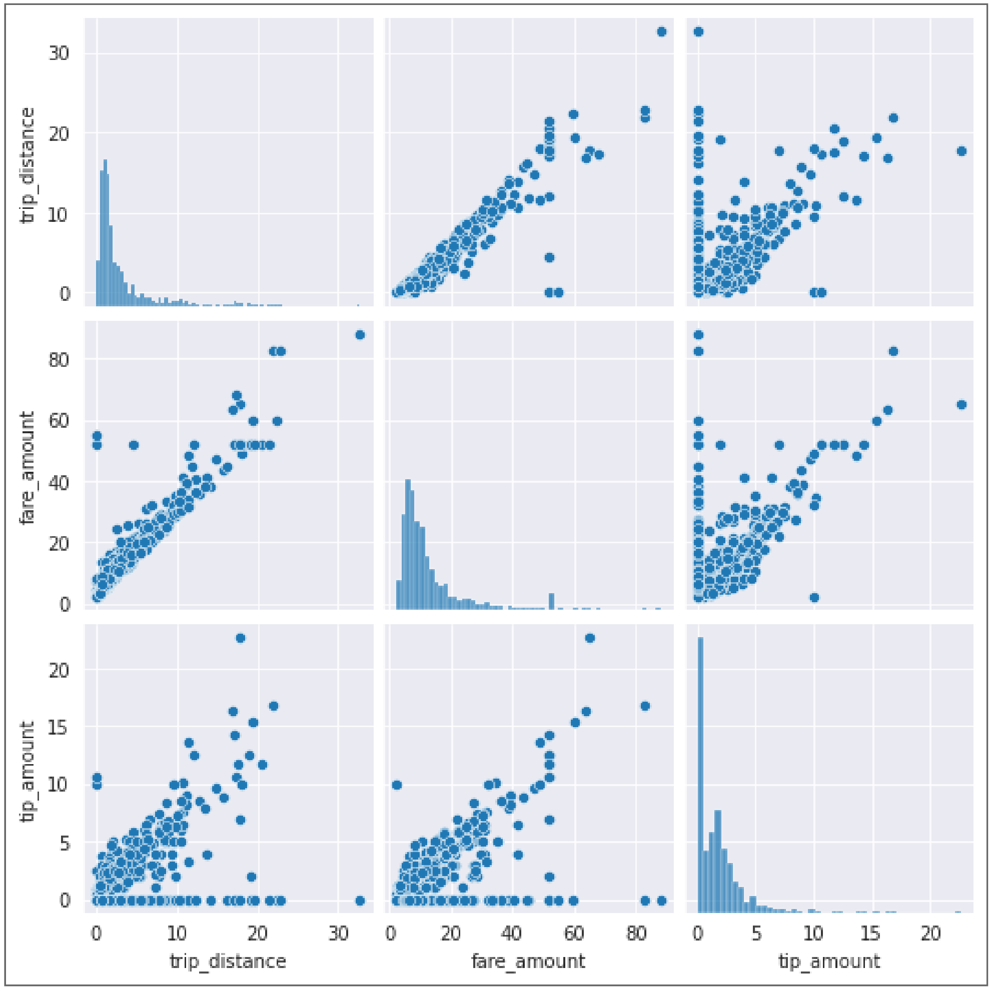
```
sns.jointplot(x='trip_distance', y='fare_amount',
              data=df,
              kind='kde');
```



Comparing Multiple Variables with `pairplot`

In [120]:

```
sns.pairplot(df[['trip_distance','fare_amount','tip_amount']]);
```



Categorical Variables: Frequency

In [121]:

```
df.payment_type.value_counts()
```

Out[121]:

```
Credit card    663
Cash           335
No charge        2
Name: payment_type, dtype: int64
```

In [122]:

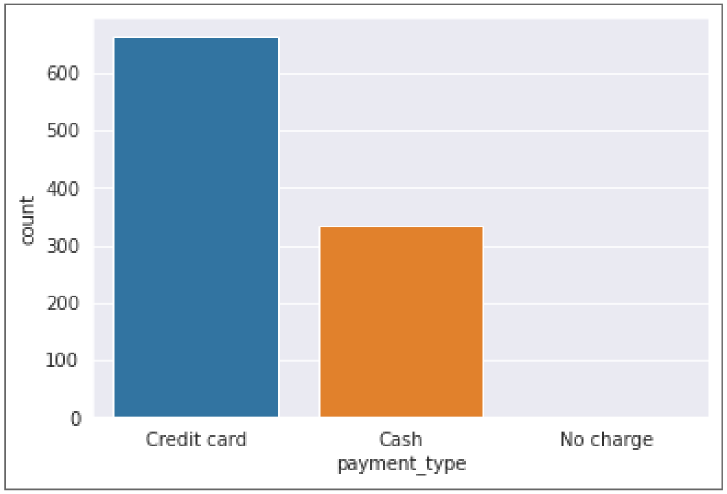
```
df.payment_type.value_counts(normalize=True)
```

Out[122]:

```
Credit card    0.663
Cash           0.335
No charge      0.002
Name: payment_type, dtype: float64
```

In [123]:

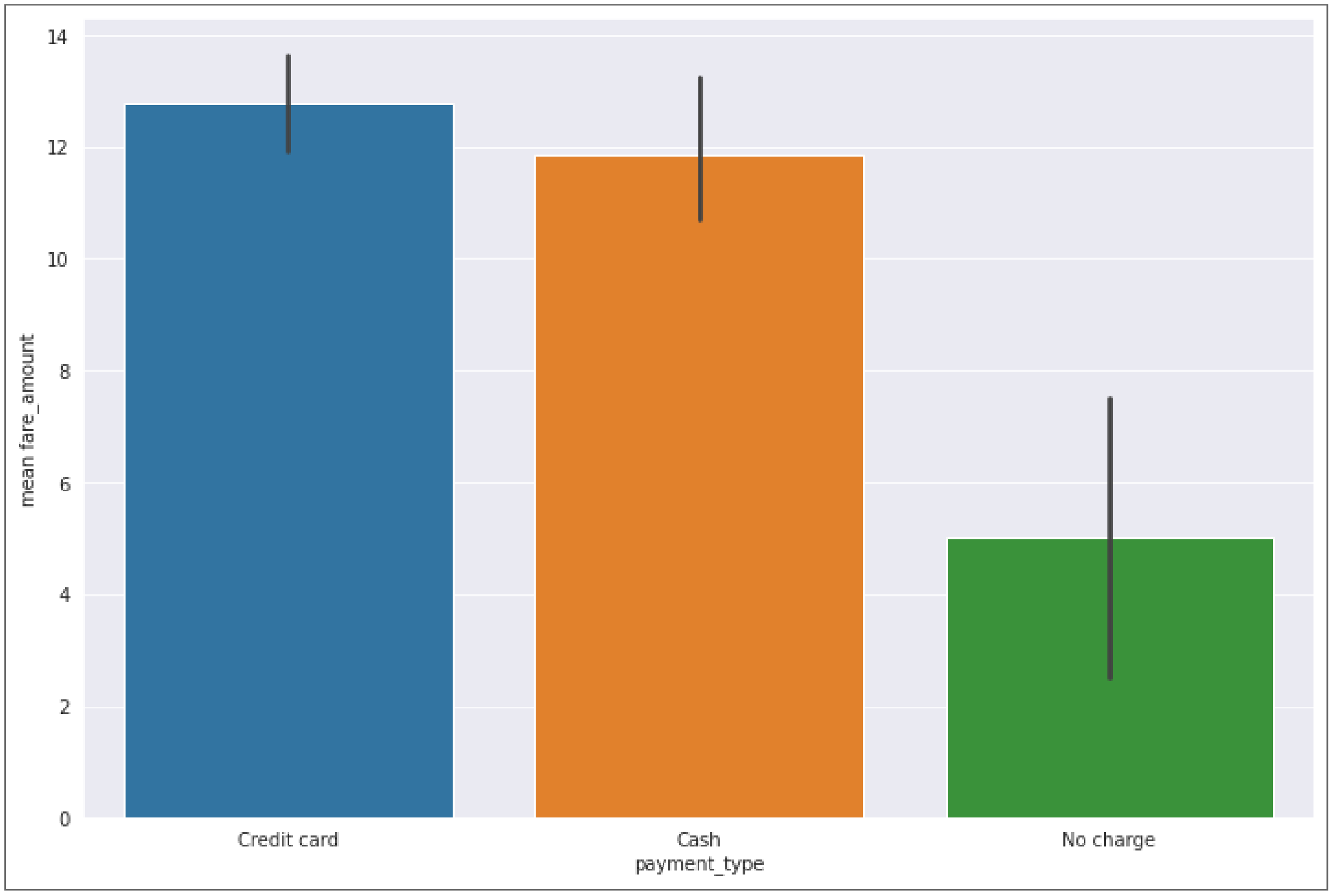
```
sns.countplot(x=df.payment_type);
```


count
600
500
400
300
200
100
0
Credit card
Cash
No charge
payment_type

Plotting Numeric and Categorical

In [124]:

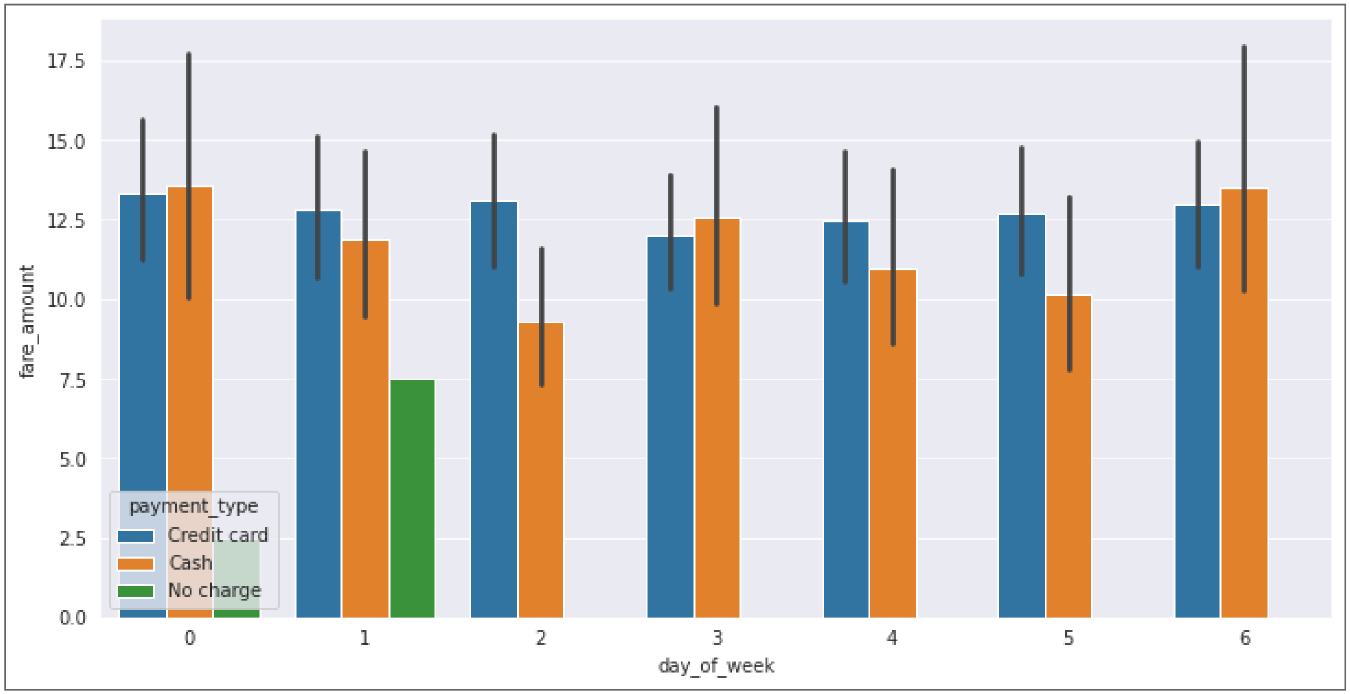
```
fig,ax = plt.subplots(1,1,figsize=(12,8))

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



Plotting with Hue

In [125]:

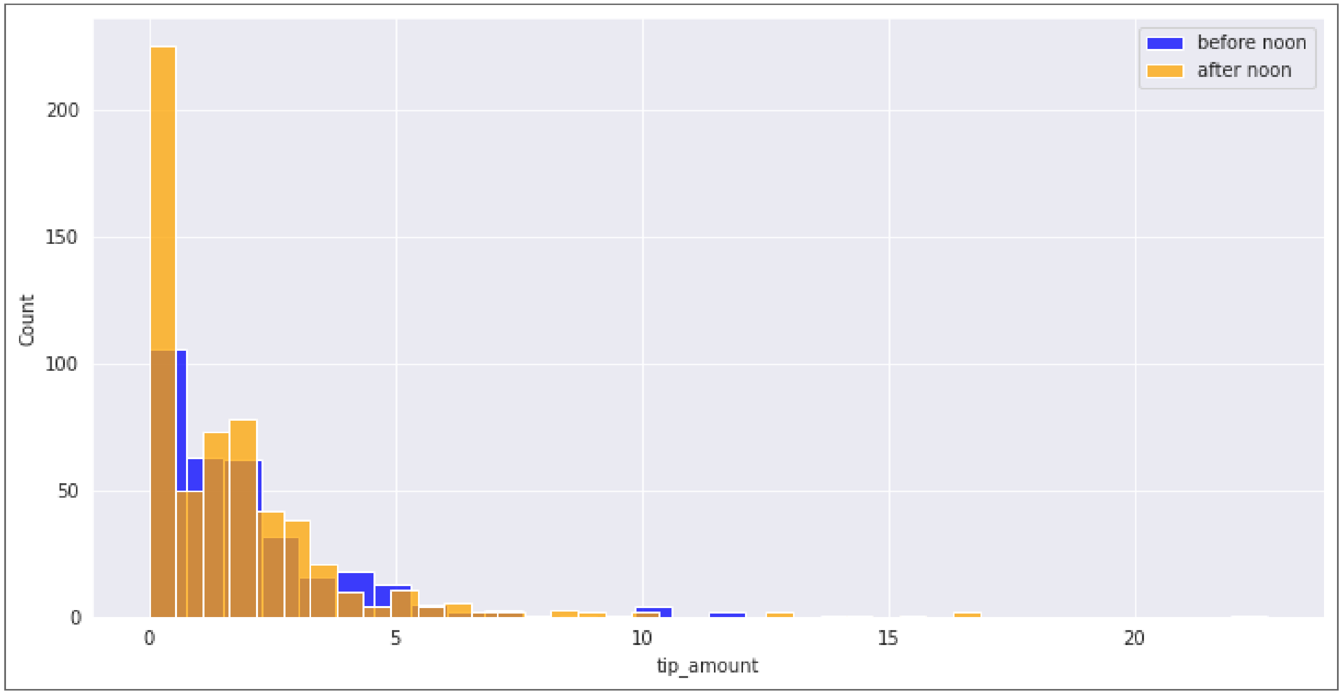
```
fig,ax = plt.subplots(1,1,figsize=(12,6))

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



Same Axis, Multiple Plots with Seaborn

In [126]:

```
fig,ax = plt.subplots(1,1,figsize=(12,6))
sns.histplot(x=df[df.pickup_datetime.dt.hour < 12].tip_amount, label='before noon',color='blue',ax=ax);
sns.histplot(x=df[df.pickup_datetime.dt.hour >= 12].tip_amount, 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 real valued variables: histogram, scatter, regplot
- plotting categorical variables: count, bar
- plotting interactions: jointplot, pairplot

Questions?