

Elements Of Data Science - F2023

Week 8: Data Cleaning and Feature Engineering

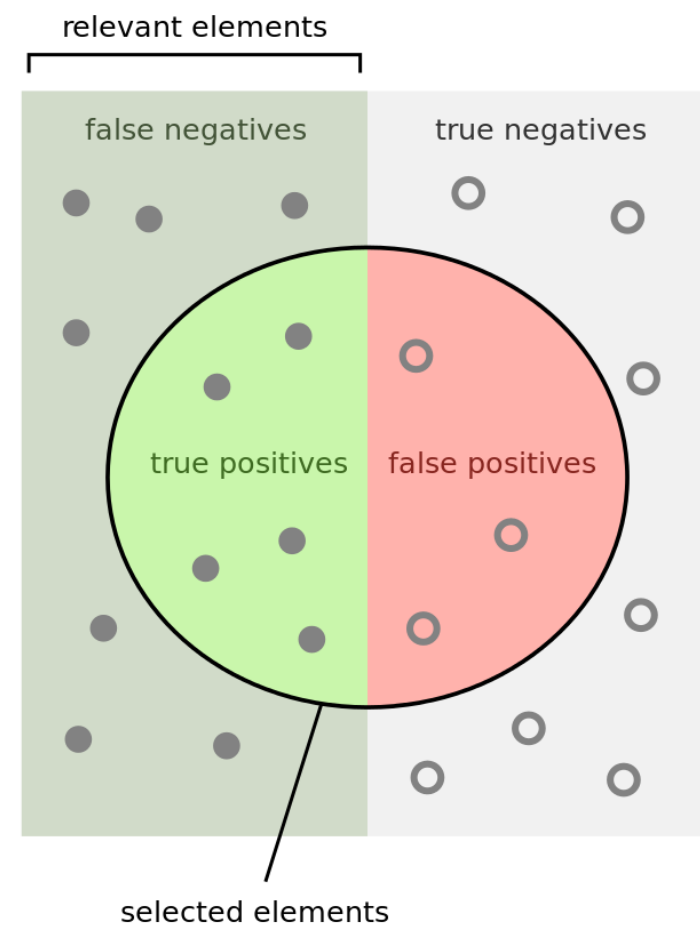
11/13/2023

TODOs

- Readings:
 - PML Ch4.5 : Selecting Meaningful Features
 - PML Ch5.1 : Unsupervised dimensionality reduction via principal component analysis
 - [Recommended] Pandas: Merge, join, concatenate and compare
 - [Additional] PDSH 5.9 : PCA
 - [Optional] : Nice ROC visualization (<http://www.navan.name/roc/>)
- Homework 2, due Monday Nov 13th, 11:59pm ET
- Quiz 7, due Monday Nov 13th, 11:59pm ET
- Quiz 8, due Monday Nov 20th, 11:59pm ET

Precision & Recall and ROC visualizations

Precision & Recall



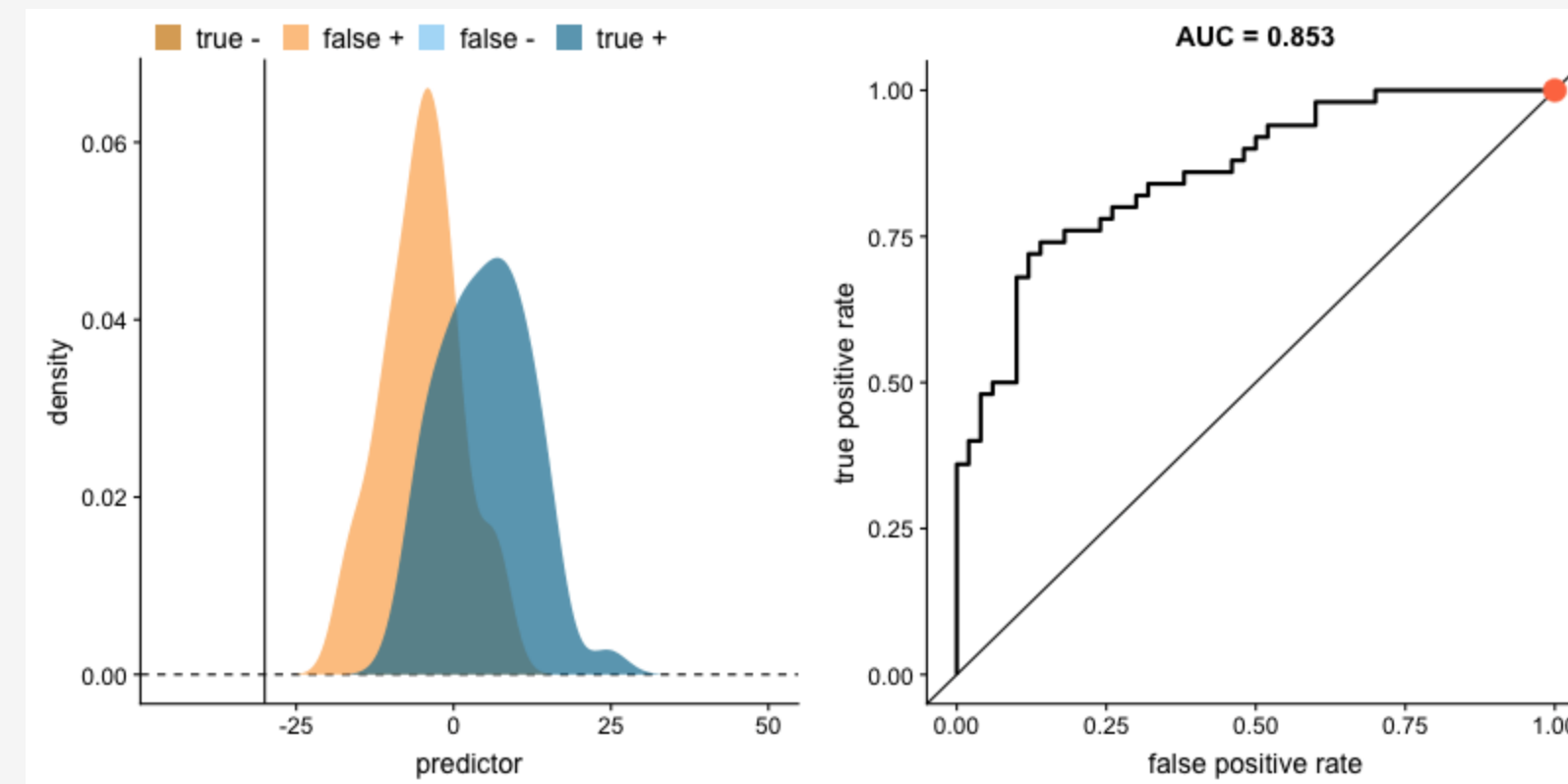
How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

ROC



This and more at https://github.com/dariyasydykova/open_projects/tree/master/ROC_animation

Also see the interactive viz at <http://www.navan.name/roc/>

Notes from Quiz 7

- LinearRegression (regression) vs LogisticRegression (classification)
- using a model "with default settings"

Notes from Quiz 7

- LinearRegression (regression) vs LogisticRegression (classification)
- using a model "with default settings"

```
In [1]: 1 from sklearn.ensemble import GradientBoostingClassifier
        2
        3 gbc = GradientBoostingClassifier()
```

Today

- **Data Cleaning**

- Duplicates
- Missing Data
- Dummy Variables
- Rescaling
- Dealing With Skew
- Removing Outliers

- **Feature Engineering**

- Binning
- One-Hot encoding
- Derived
 - string functions
 - datetime functions

Questions?

Environment Setup

Environment Setup

```
In [2]: 1 import numpy
        2 import numpy as np
        3 import pandas as pd
        4 import matplotlib.pyplot as plt
        5 import seaborn as sns
        6
        7 from mlxtend.plotting import plot_decision_regions
        8
        9 sns.set_style('darkgrid')
       10
       11 %matplotlib inline
```

Data Cleaning

Why do we need clean data?

- **To remove duplicates** Want one row per observation
- **To remove/fill missing** Most models cannot handle missing data
- **To engineer features** Most models require fixed length feature vectors
- Different models require different types of data (transformation)
 - **Linear models:** real valued features with similar scale
 - **Distance based:** real valued features with similar scale
 - **Tree based:** can handle unscaled real and categorical (sklearn requires real)

Example Data

Example Data

```
In [3]: 1 # read in example data
2 df_shop_raw = pd.read_csv('../data/flowershop_data_with_dups_week8.csv',
3                             header=0,
4                             delimiter=',')
5 df_shop_raw['purchase_date'] = pd.to_datetime(df_shop_raw.purchase_date)
6
7 # make a copy for editing
8 df_shop = df_shop_raw.copy()
9
10 df_shop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001 entries, 0 to 1000
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   purchase_id      1001 non-null   int64
1   lastname         1001 non-null   object
2   purchase_date    1001 non-null   datetime64[ns]
3   stars            1001 non-null   int64
4   price            979 non-null    float64
5   favorite_flower  823 non-null    object
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 47.0+ KB
```

Duplicated Data

- Only drop duplicates if you know data should be unique
 - Example: if there is a unique id per row

Duplicated Data

- Only drop duplicates if you know data should be unique
 - Example: if there is a unique id per row

```
In [4]: 1 df_shop.duplicated().iloc[-3:] # are any of the last 3 rows duplicates?
```

```
Out[4]: 998      False
        999      False
        1000     True
        dtype: bool
```

Duplicated Data

- Only drop duplicates if you know data should be unique
 - Example: if there is a unique id per row

```
In [4]: 1 df_shop.duplicated().iloc[-3:] # are any of the last 3 rows duplicates?
```

```
Out[4]: 998      False
        999      False
        1000     True
        dtype: bool
```

```
In [5]: 1 df_shop[df_shop.duplicated(keep='first')] # default: keep 'first' duplicated row
```

```
Out[5]:
```

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil

Duplicated Data

- Only drop duplicates if you know data should be unique
 - Example: if there is a unique id per row

```
In [4]: 1 df_shop.duplicated().iloc[-3:] # are any of the last 3 rows duplicates?
```

```
Out[4]: 998      False
        999      False
        1000     True
        dtype: bool
```

```
In [5]: 1 df_shop[df_shop.duplicated(keep='first')] # default: keep 'first' duplicated row
```

```
Out[5]:
```

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil

```
In [6]: 1 df_shop[df_shop.duplicated(keep=False)] # keep=False to show all duplicated rows
```

```
Out[6]:
```

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
10	1010	FERGUSON	2017-05-04	2	21.02	daffodil
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil

Duplicated Data for Subset of Columns

Duplicated Data for Subset of Columns

```
In [7]: 1 # if it's important that a subset of columns is not duplicated
        2 (
        3     df_shop
        4     .loc[df_shop.duplicated(subset=['purchase_id'], keep=False)]
        5     .sort_values(by='purchase_id')
        6 )
```

Out[7]:

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
10	1010	FERGUSON	2017-05-04	2	21.02	daffodil
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil
100	1101	WEBB	2017-07-13	2	8.00	iris
101	1101	BURKE	2017-08-16	4	18.56	daffodil

Duplicated Data for Subset of Columns

```
In [7]: 1 # if it's important that a subset of columns is not duplicated
        2 (
        3     df_shop
        4     .loc[df_shop.duplicated(subset=['purchase_id'],keep=False)]
        5     .sort_values(by='purchase_id')
        6 )
```

Out[7]:

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
10	1010	FERGUSON	2017-05-04	2	21.02	daffodil
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil
100	1101	WEBB	2017-07-13	2	8.00	iris
101	1101	BURKE	2017-08-16	4	18.56	daffodil

```
In [8]: 1 # could also use the indexing shortcut
        2 df_shop[df_shop.duplicated(subset=['purchase_id'],keep=False)].sort_values(by='purchase_id')
```

Out[8]:

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
10	1010	FERGUSON	2017-05-04	2	21.02	daffodil
1000	1010	FERGUSON	2017-05-04	2	21.02	daffodil
100	1101	WEBB	2017-07-13	2	8.00	iris
101	1101	BURKE	2017-08-16	4	18.56	daffodil

Dropping Duplicates

Dropping Duplicates

```
In [9]: 1 df_shop = df_shop.drop_duplicates(subset=None,      # consider all columns
      2                                     keep='first',      # or 'last' or False)
      3                                     inplace=False      # default, return a modified dataframe
      4                                     )
```

Dropping Duplicates

```
In [9]: 1 df_shop = df_shop.drop_duplicates(subset=None,      # consider all columns
2                                           keep='first',    # or 'last' or False)
3                                           inplace=False    # default, return a modified dataframe
4                                           )
```

```
In [10]: 1 # or can use inplace to change the original dataframe
2 df_shop.drop_duplicates(subset=None,
3                          keep='first',
4                          inplace=True # change the dataframe in place
5                          )
```

Dropping Duplicates

```
In [9]: 1 df_shop = df_shop.drop_duplicates(subset=None,      # consider all columns
2                                         keep='first',    # or 'last' or False)
3                                         inplace=False     # default, return a modified dataframe
4                                         )
```

```
In [10]: 1 # or can use inplace to change the original dataframe
2 df_shop.drop_duplicates(subset=None,
3                          keep='first',
4                          inplace=True # change the dataframe in place
5                          )
```

```
In [11]: 1 # can also drop rows with duplicates considering only a subset of columns
2 df_shop = df_shop.drop_duplicates(subset=[ 'purchase_id' ])
```

Missing Data

- Reasons for missing data
 - Sensor error (random?)
 - Data entry error (random?)
 - Survey-subject decisions (non-random?)
 - etc.

Missing Data

- Reasons for missing data
 - Sensor error (random?)
 - Data entry error (random?)
 - Survey-subject decisions (non-random?)
 - etc.
- Dealing with missing data
 - Drop rows
 - Impute from data in the same column
 - Infer from other features
 - Fill with adjacent data

Missing Data in Pandas: `np.nan`

- Missing values represented by `np.nan` : Not A Number

Missing Data in Pandas: `np.nan`

- Missing values represented by `np.nan` : Not A Number

```
In [12]: 1 # Earlier, we saw missing values in the dataframe summary
          2 df_shop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  -
 0   purchase_id     999 non-null   int64  
 1   lastname        999 non-null   object  
 2   purchase_date   999 non-null   datetime64[ns]
 3   stars           999 non-null   int64  
 4   price           977 non-null   float64 
 5   favorite_flower 821 non-null   object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 54.6+ KB
```

Missing Data in Pandas: `np.nan`

- Missing values represented by `np.nan` : Not A Number

```
In [12]: 1 # Earlier, we saw missing values in the dataframe summary
          2 df_shop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   purchase_id           999 non-null    int64   
 1   lastname              999 non-null    object  
 2   purchase_date         999 non-null    datetime64[ns]
 3   stars                 999 non-null    int64   
 4   price                 977 non-null    float64  
 5   favorite_flower       821 non-null    object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 54.6+ KB
```

```
In [13]: 1 # can we check for NaN using "x == np.nan"? No!
          2 np.nan == np.nan
          3
```

```
Out[13]: False
```

Missing Data in Pandas: `np.nan`

- Missing values represented by `np.nan` : Not A Number

```
In [12]: 1 # Earlier, we saw missing values in the dataframe summary
          2 df_shop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   purchase_id           999 non-null   int64   
1   lastname               999 non-null   object  
2   purchase_date          999 non-null   datetime64[ns]
3   stars                  999 non-null   int64   
4   price                  977 non-null   float64  
5   favorite_flower        821 non-null   object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 54.6+ KB
```

```
In [13]: 1 # can we check for NaN using "x == np.nan"? No!
          2 np.nan == np.nan
          3
```

```
Out[13]: False
```

`==` operator compares the values of both the operands and checks for value equality. `is` operator, on the other hand, checks whether both the operands refer to the same object or not.

Missing Data in Pandas: `np.nan`

- Missing values represented by `np.nan` : Not A Number

```
In [12]: 1 # Earlier, we saw missing values in the dataframe summary
          2 df_shop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   purchase_id           999 non-null   int64   
1   lastname              999 non-null   object  
2   purchase_date         999 non-null   datetime64[ns]
3   stars                 999 non-null   int64   
4   price                 977 non-null   float64  
5   favorite_flower       821 non-null   object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 54.6+ KB
```

```
In [13]: 1 # can we check for NaN using "x == np.nan"? No!
          2 np.nan == np.nan
          3
```

```
Out[13]: False
```

`==` operator compares the values of both the operands and checks for value equality. `is` operator, on the other hand, checks whether both the operands refer to the same object or not.

How to check for NaN: `.isna()` and `.notna()`

How to check for NaN: `.isna()` and `.notna()`

```
In [18]: 1 # some missing data  
        2 df_shop.loc[20:21, 'price']
```

```
Out[18]: 20      NaN  
        21     10.53  
        Name: price, dtype: float64
```


How to check for NaN: `.isna()` and `.notna()`

```
In [18]: 1 # some missing data
          2 df_shop.loc[20:21, 'price']
```

```
Out[18]: 20      NaN
          21     10.53
          Name: price, dtype: float64
```

```
In [19]: 1 # .isna() returns True where data is missing, False otherwise
          2 df_shop.loc[20:21, 'price'].isna()
```

```
Out[19]: 20      True
          21     False
          Name: price, dtype: bool
```

How to check for NaN: `.isna()` and `.notna()`

```
In [18]: 1 # some missing data
          2 df_shop.loc[20:21, 'price']
```

```
Out[18]: 20      NaN
          21     10.53
          Name: price, dtype: float64
```

```
In [19]: 1 # .isna() returns True where data is missing, False otherwise
          2 df_shop.loc[20:21, 'price'].isna()
```

```
Out[19]: 20      True
          21     False
          Name: price, dtype: bool
```

```
In [20]: 1 # .notna() returns True where data is NOT missing, False otherwise
          2 df_shop.loc[20:21, 'price'].notna()
```

```
Out[20]: 20     False
          21      True
          Name: price, dtype: bool
```

How to check for NaN: `.isna()` and `.notna()`

```
In [18]: 1 # some missing data
          2 df_shop.loc[20:21, 'price']
```

```
Out[18]: 20      NaN
          21     10.53
          Name: price, dtype: float64
```

```
In [19]: 1 # .isna() returns True where data is missing, False otherwise
          2 df_shop.loc[20:21, 'price'].isna()
```

```
Out[19]: 20      True
          21     False
          Name: price, dtype: bool
```

```
In [20]: 1 # .notna() returns True where data is NOT missing, False otherwise
          2 df_shop.loc[20:21, 'price'].notna()
```

```
Out[20]: 20     False
          21      True
          Name: price, dtype: bool
```

```
In [21]: 1 # find rows where price is missing
          2 df_shop[df_shop.price.isna()].head()
```

```
Out[21]:
```

	purchase_id	lastname	purchase_date	stars	price	favorite_flower
20	1020	CLARK	2017-01-05	3	NaN	NaN
41	1041	PETERS	2017-02-01	4	NaN	orchid
54	1054	GREEN	2017-02-13	5	NaN	daffodil
63	1063	BARNETT	2017-08-27	4	NaN	gardenia
145	1145	CARROLL	2017-07-29	3	NaN	tulip

Counting NaNs

Counting NaNs

```
In [22]: 1 # How many nan's in a single column?  
        2 df_shop.price.isna().sum()
```

```
Out[22]: 22
```

Counting NaNs

```
In [22]: 1 # How many nan's in a single column?
          2 df_shop.price.isna().sum()
```

Out[22]: 22

```
In [23]: 1 # How many nan's per column?
          2 df_shop.isna().sum()
```

```
Out[23]: purchase_id      0
          lastname      0
          purchase_date  0
          stars         0
          price         22
          favorite_flower 178
          dtype: int64
```

Counting NaNs

```
In [22]: 1 # How many nan's in a single column?
          2 df_shop.price.isna().sum()
```

Out[22]: 22

```
In [23]: 1 # How many nan's per column?
          2 df_shop.isna().sum()
```

```
Out[23]: purchase_id      0
          lastname      0
          purchase_date  0
          stars         0
          price         22
          favorite_flower 178
          dtype: int64
```

```
In [24]: 1 # How many total nan's?
          2 df_shop.isna().sum().sum()
```

Out[24]: 200

Missing Data: Drop Rows

Missing Data: Drop Rows

```
In [25]: 1 df_shop.shape
```

```
Out[25]: (999, 6)
```

Missing Data: Drop Rows

```
In [25]: 1 df_shop.shape
```

```
Out[25]: (999, 6)
```

```
In [26]: 1 # drop rows with nan in any column  
        2 df_shop.dropna().shape
```

```
Out[26]: (801, 6)
```

Missing Data: Drop Rows

```
In [25]: 1 df_shop.shape
```

```
Out[25]: (999, 6)
```

```
In [26]: 1 # drop rows with nan in any column  
2 df_shop.dropna().shape
```

```
Out[26]: (801, 6)
```

```
In [27]: 1 # drop only rows with nan in price using subset  
2 df_shop.dropna(subset=['price']).shape
```

```
Out[27]: (977, 6)
```

Missing Data: Drop Rows

```
In [25]: 1 df_shop.shape
```

```
Out[25]: (999, 6)
```

```
In [26]: 1 # drop rows with nan in any column  
2 df_shop.dropna().shape
```

```
Out[26]: (801, 6)
```

```
In [27]: 1 # drop only rows with nan in price using subset  
2 df_shop.dropna(subset=['price']).shape
```

```
Out[27]: (977, 6)
```

```
In [28]: 1 # drop only rows with nans in all columns (a row of all nans)  
2 df_shop.dropna(how='all').shape
```

```
Out[28]: (999, 6)
```

Missing Data: Drop Rows Cont.

Missing Data: Drop Rows Cont.

```
In [29]: 1 # save a new dataframe with dropped rows
          2 df_shop = df_shop_raw.dropna().copy()
          3 df_shop.shape
```

```
Out[29]: (803, 6)
```

Missing Data: Drop Rows Cont.

```
In [29]: 1 # save a new dataframe with dropped rows
          2 df_shop = df_shop_raw.dropna().copy()
          3 df_shop.shape
```

Out[29]: (803, 6)

```
In [30]: 1 # drop rows in current dataframe with inplace
          2 df_shop = df_shop_raw.copy()
          3
          4 df_shop.dropna(inplace=True)
          5 df_shop.shape
```

Out[30]: (803, 6)

Missing Data: Drop Rows

- Pros:
 - easy to do
 - simple to understand
- Cons:
 - potentially large data loss

Missing Data: Fill with Constant

- Use `.fillna()`
- Common filler: 0, -1

```
In [31]: 1 df_shop = df_shop_raw.drop_duplicates().copy() # make a new copy of the data
```

Missing Data: Fill with Constant

- Use `.fillna()`
- Common filler: 0, -1

```
In [31]: 1 df_shop = df_shop_raw.drop_duplicates().copy() # make a new copy of the data
```

```
In [32]: 1 df_shop.price[20:22]
```

```
Out[32]: 20      NaN  
        21    10.53  
        Name: price, dtype: float64
```

Missing Data: Fill with Constant

- Use `.fillna()`
- Common filler: 0, -1

```
In [31]: 1 df_shop = df_shop_raw.drop_duplicates().copy() # make a new copy of the data
```

```
In [32]: 1 df_shop.price[20:22]
```

```
Out[32]: 20      NaN  
        21    10.53  
        Name: price, dtype: float64
```

```
In [33]: 1 df_shop.price[20:22].fillna(0)
```

```
Out[33]: 20     0.00  
        21    10.53  
        Name: price, dtype: float64
```

Missing Data: Fill with Constant

- Use `.fillna()`
- Common filler: 0, -1

```
In [31]: 1 df_shop = df_shop_raw.drop_duplicates().copy() # make a new copy of the data
```

```
In [32]: 1 df_shop.price[20:22]
```

```
Out[32]: 20      NaN  
        21     10.53  
        Name: price, dtype: float64
```

```
In [33]: 1 df_shop.price[20:22].fillna(0)
```

```
Out[33]: 20     0.00  
        21     10.53  
        Name: price, dtype: float64
```

```
In [34]: 1 print(df_shop.price.mean().round(2))  
        2 print(df_shop.price.fillna(0).mean().round(2))
```

```
23.4  
22.89
```

Missing Data: Fill with Constant

Pros:

- easy to do
- simple to understand

Cons:

- values may not be realistic

Missing Data: Impute

- Impute: fill with value inferred from existing values in that column
- Use `.fillna()` or sklearn methods
- Common filler values:
 - mean
 - median
 - "most frequent" aka mode

Missing Data: Impute

Missing Data: Impute

```
In [35]: 1 print(df_shop.price.mean().round(2))  
        2 print(df_shop.price.fillna(df_shop.price.mean()).mean().round(2))
```

23.4

23.4

Missing Data: Impute

```
In [35]: 1 print(df_shop.price.mean().round(2))  
        2 print(df_shop.price.fillna(df_shop.price.mean()).mean().round(2))
```

```
23.4  
23.4
```

```
In [36]: 1 # make a copy to keep our original df  
        2 df_shop_impute = df_shop.copy()
```

Missing Data: Impute

```
In [35]: 1 print(df_shop.price.mean().round(2))  
        2 print(df_shop.price.fillna(df_shop.price.mean()).mean().round(2))
```

```
23.4  
23.4
```

```
In [36]: 1 # make a copy to keep our original df  
        2 df_shop_impute = df_shop.copy()
```

```
In [37]: 1 # fill missing price with mean of price  
        2 df_shop_impute['price'] = df_shop.price.fillna(df_shop.price.mean())
```

Missing Data: Impute

```
In [35]: 1 print(df_shop.price.mean().round(2))  
2 print(df_shop.price.fillna(df_shop.price.mean()).mean().round(2))
```

```
23.4  
23.4
```

```
In [36]: 1 # make a copy to keep our original df  
2 df_shop_impute = df_shop.copy()
```

```
In [37]: 1 # fill missing price with mean of price  
2 df_shop_impute['price'] = df_shop.price.fillna(df_shop.price.mean())
```

```
In [38]: 1 # check to make sure all nulls filled  
2 assert df_shop_impute.price.isna().sum() == 0  
3 assert df_shop_impute.price.notna().all()  
4  
5 # also, that our mean hasn't changed  
6 assert df_shop.price.mean() == df_shop_impute.price.mean()
```

Missing Data: Impute

```
In [35]: 1 print(df_shop.price.mean().round(2))
          2 print(df_shop.price.fillna(df_shop.price.mean()).mean().round(2))
```

```
23.4
23.4
```

```
In [36]: 1 # make a copy to keep our original df
          2 df_shop_impute = df_shop.copy()
```

```
In [37]: 1 # fill missing price with mean of price
          2 df_shop_impute['price'] = df_shop.price.fillna(df_shop.price.mean())
```

```
In [38]: 1 # check to make sure all nulls filled
          2 assert df_shop_impute.price.isna().sum() == 0
          3 assert df_shop_impute.price.notna().all()
          4
          5 # also, that our mean hasn't changed
          6 assert df_shop.price.mean() == df_shop_impute.price.mean()
```

```
In [39]: 1 # inplace works here as well
          2 df_shop_impute.price.fillna(df_shop_impute.price.mean(),inplace=True)
```

Missing Data: Impute Cont.

if data is categorical?

Missing Data: Impute Cont.

if data is categorical?

```
In [40]: 1 df_shop.favorite_flower.mode() # may be more than 1!
```

```
Out[40]: 0    lilac  
         Name: favorite_flower, dtype: object
```

Missing Data: Impute Cont.

if data is categorical?

```
In [40]: 1 df_shop.favorite_flower.mode() # may be more than 1!
```

```
Out[40]: 0    lilac  
         Name: favorite_flower, dtype: object
```

```
In [41]: 1 # Note that we have to index into the DataFrame returned by mode to get a value  
         2 df_shop_impute.favorite_flower.fillna(df_shop_impute.favorite_flower.mode().iloc[0],inplace=True)  
         3  
         4 df_shop_impute.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 1000 entries, 0 to 999  
Data columns (total 6 columns):  
#   Column          Non-Null Count  Dtype  
---  -  
0   purchase_id      1000 non-null   int64  
1   lastname          1000 non-null   object  
2   purchase_date     1000 non-null   datetime64[ns]  
3   stars             1000 non-null   int64  
4   price             1000 non-null   float64  
5   favorite_flower   1000 non-null   object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)  
memory usage: 87.0+ KB
```

Missing Data: Impute Cont. Using SimpleImputer

Missing Data: Impute Cont. Using SimpleImputer

```
In [42]: 1 df_shop[['price', 'stars']].loc[20:22]
```

Out[42]:

	price	stars
20	NaN	3
21	10.53	2
22	19.77	1

Missing Data: Impute Cont. Using SimpleImputer

```
In [42]: 1 df_shop[['price', 'stars']].loc[20:22]
```

Out[42]:

	price	stars
20	NaN	3
21	10.53	2
22	19.77	1

```
In [43]: 1 from sklearn.impute import SimpleImputer
2
3 imp = SimpleImputer(strategy='mean').fit(df_shop[['price', 'stars']])
4 print(f'fill values = {imp.statistics_.round(2)}')
5 imp.transform(df_shop.loc[20:22, ['price', 'stars']]).round(2)
```

```
fill values = [23.4  3.6]
```

Out[43]: array([[23.4 , 3.],
[10.53, 2.],
[19.77, 1.]])

Missing Data: Impute Cont. Using SimpleImputer

```
In [42]: 1 df_shop[['price', 'stars']].loc[20:22]
```

Out[42]:

	price	stars
20	NaN	3
21	10.53	2
22	19.77	1

```
In [43]: 1 from sklearn.impute import SimpleImputer
2
3 imp = SimpleImputer(strategy='mean').fit(df_shop[['price', 'stars']])
4 print(f'fill values = {imp.statistics_.round(2)}')
5 imp.transform(df_shop.loc[20:22, ['price', 'stars']]).round(2)
```

```
fill values = [23.4  3.6]
```

Out[43]: array([[23.4 , 3.],
[10.53, 2.],
[19.77, 1.]])

```
In [44]: 1 df_shop.favorite_flower[:3]
```

Out[44]: 0 iris
1 NaN
2 carnation
Name: favorite_flower, dtype: object

Missing Data: Impute Cont. Using SimpleImputer

```
In [42]: 1 df_shop[['price', 'stars']].loc[20:22]
```

Out[42]:

	price	stars
20	NaN	3
21	10.53	2
22	19.77	1

```
In [43]: 1 from sklearn.impute import SimpleImputer
2
3 imp = SimpleImputer(strategy='mean').fit(df_shop[['price', 'stars']])
4 print(f'fill values = {imp.statistics_.round(2)}')
5 imp.transform(df_shop.loc[20:22, ['price', 'stars']]).round(2)
```

```
fill values = [23.4  3.6]
```

```
Out[43]: array([[23.4 ,  3.  ],
               [10.53,  2.  ],
               [19.77,  1.  ]])
```

```
In [44]: 1 df_shop.favorite_flower[:3]
```

```
Out[44]: 0      iris
1      NaN
2  carnation
Name: favorite_flower, dtype: object
```

```
In [45]: 1 imp = SimpleImputer(strategy='most_frequent').fit(df_shop[['favorite_flower']])
2 imp.transform(df_shop.loc[:2, ['favorite_flower']])
```

```
Out[45]: array(['iris',
               'lilac',
               'carnation'], dtype=object)
```

Missing Data: Impute

- Pros:
 - easy to do
 - simple to understand
- Cons:
 - may miss feature interactions

Missing Data: Infer

- Predict values of missing features using a model
- Ex: Can we predict price based on any of the other features?
- Additional feature engineering may be needed prior to this

Missing Data: Infer

- Predict values of missing features using a model
- Ex: Can we predict price based on any of the other features?
- Additional feature engineering may be needed prior to this

```
In [46]: 1 from sklearn.linear_model import LinearRegression
          2
          3 df_shop_infer = df_shop.copy()
          4
          5 not_missing = df_shop_infer.price.notna()
          6 missing = df_shop_infer.price.isna()
          7
          8 lr = LinearRegression().fit(df_shop_infer.loc[not_missing, ['stars']],
          9                             df_shop_infer.loc[not_missing, 'price'])
          10
          11 df_shop_infer.loc[missing, 'price'] = lr.predict(df_shop_infer.loc[missing, ['stars']])
```

Missing Data: Infer

- Pros:

Missing Data: Adjacent Data

- Use `.fillna()` with method:
 - `ffill`: propagate last valid observation forward to next valid
 - `bfill`: use next valid observation to fill gap backwards
- Use when there is reason to believe data not i.i.d. (eg: **timeseries**)

Missing Data: Adjacent Data

- Use `.fillna()` with method:
 - `ffill`: propagate last valid observation forward to next valid
 - `bfill`: use next valid observation to fill gap backwards
- Use when there is reason to believe data not i.i.d. (eg: **timeseries**)

```
In [47]: 1 df_shop.price.loc[19:21]
```

```
Out[47]: 19    20.45  
        20     NaN  
        21    10.53  
        Name: price, dtype: float64
```

Missing Data: Adjacent Data

- Use `.fillna()` with method:
 - `ffill`: propagate last valid observation forward to next valid
 - `bfill`: use next valid observation to fill gap backwards
- Use when there is reason to believe data not i.i.d. (eg: **timeseries**)

```
In [47]: 1 df_shop.price.loc[19:21]
```

```
Out[47]: 19    20.45  
        20     NaN  
        21    10.53  
        Name: price, dtype: float64
```

```
In [48]: 1 df_shop.price.fillna(method='ffill').loc[19:21]
```

```
Out[48]: 19    20.45  
        20    20.45  
        21    10.53  
        Name: price, dtype: float64
```

Missing Data: Add a Dummy Column First

- Data may be missing for a reason!
- Capture "missing" before filling

Missing Data: Add a Dummy Column First

- Data may be missing for a reason!
- Capture "missing" before filling

```
In [49]: 1 df_shop = df_shop_raw.drop_duplicates().copy()  
2  
3 # storing a column of 1:missing, 0:not-missing  
4 df_shop['price_missing'] = df_shop.price.isna().astype(int)  
5  
6 # can now fill missing values  
7 df_shop['price'] = df_shop.price.fillna(df_shop.price.mean())
```

Missing Data: Add a Dummy Column First

- Data may be missing for a reason!
- Capture "missing" before filling

```
In [49]: 1 df_shop = df_shop_raw.drop_duplicates().copy()  
2  
3 # storing a column of 1:missing, 0:not-missing  
4 df_shop['price_missing'] = df_shop.price.isna().astype(int)  
5  
6 # can now fill missing values  
7 df_shop['price'] = df_shop.price.fillna(df_shop.price.mean())
```

```
In [50]: 1 # finding where data was missing  
2 np.where(df_shop.price_missing == 1)
```

```
Out[50]: (array([ 20,  41,  54,  63, 145, 186, 194, 203, 212, 360, 367, 382, 429,  
                469, 522, 570, 595, 726, 792, 821, 974, 978]),)
```

Missing Data: Add a Dummy Column First

- Data may be missing for a reason!
- Capture "missing" before filling

```
In [49]: 1 df_shop = df_shop_raw.drop_duplicates().copy()  
2  
3 # storing a column of 1:missing, 0:not-missing  
4 df_shop['price_missing'] = df_shop.price.isna().astype(int)  
5  
6 # can now fill missing values  
7 df_shop['price'] = df_shop.price.fillna(df_shop.price.mean())
```

```
In [50]: 1 # finding where data was missing  
2 np.where(df_shop.price_missing == 1)
```

```
Out[50]: (array([ 20,  41,  54,  63, 145, 186, 194, 203, 212, 360, 367, 382, 429,  
                469, 522, 570, 595, 726, 792, 821, 974, 978]),)
```

```
In [51]: 1 df_shop[['price', 'price_missing']].iloc[20:23]
```

```
Out[51]:
```

	price	price_missing
20	23.403384	1
21	10.530000	0
22	19.770000	0

Rescaling

- Often need features to be in the same scale
- Methods of rescaling
 - Standardization (z-score)
 - Min-Max rescaling
 - others...

Rescaling

- Often need features to be in the same scale
- Methods of rescaling
 - Standardization (z-score)
 - Min-Max rescaling
 - others...

```
In [52]: 1 # load taxi data
2 df_taxi_raw = pd.read_csv('../data/yellowcab_tripdata_2017-01_subset10000rows.csv',
3                             parse_dates=['tpep_pickup_datetime', 'tpep_dropoff_datetime'])
4 # create trip_duration
5 df_taxi_raw['trip_duration'] = (df_taxi_raw.tpep_dropoff_datetime - df_taxi_raw.tpep_pickup_datetime).dt.seconds
6
7 # select subset
8 df_taxi_raw = df_taxi_raw[df_taxi_raw.trip_duration.lt(3600) &
9                             df_taxi_raw.tip_amount.between(0,10,inclusive='neither')]
10
11 df_taxi = df_taxi_raw.copy()
```


Rescaling

- Often need features to be in the same scale
- Methods of rescaling
 - Standardization (z-score)
 - Min-Max rescaling
 - others...

```
In [52]: 1 # load taxi data
2 df_taxi_raw = pd.read_csv('../data/yellowcab_tripdata_2017-01_subset10000rows.csv',
3                             parse_dates=['tpep_pickup_datetime', 'tpep_dropoff_datetime'])
4 # create trip_duration
5 df_taxi_raw['trip_duration'] = (df_taxi_raw.tpep_dropoff_datetime - df_taxi_raw.tpep_pickup_datetime).dt.seconds
6
7 # select subset
8 df_taxi_raw = df_taxi_raw[df_taxi_raw.trip_duration.lt(3600) &
9                             df_taxi_raw.tip_amount.between(0,10,inclusive='neither')]
10
11 df_taxi = df_taxi_raw.copy()
```

```
In [53]: 1 df_taxi[['trip_duration', 'tip_amount']].agg(['mean', 'std', 'min', 'max'],axis=0).round(2)
```

Out[53]:

	trip_duration	tip_amount
mean	765.03	2.41
std	496.83	1.55
min	2.00	0.01

Rescaling: Standardization

- rescale to 0 mean, standard deviation of 1
 - $X_{\text{scaled}} = (X - X.\text{mean}()) / X.\text{std}()$

Rescaling: Standardization

- rescale to 0 mean, standard deviation of 1
 - $X_{\text{scaled}} = (X - X.\text{mean}()) / X.\text{std}()$

```
In [54]: 1 from sklearn.preprocessing import StandardScaler
          2
          3 # instantiate
          4 ss = StandardScaler(with_mean=True, with_std=True) # default is center and scale
          5
          6 # fit to the data
          7 ss.fit(df_taxi[['trip_duration', 'tip_amount']])
          8
          9 # transform the data
         10 X_ss = ss.transform(df_taxi[['trip_duration', 'tip_amount']])
         11 X_ss[:2].round(2)
```

```
Out[54]: array([[ -0.5 , -0.48],
                [-0.17, -0.91]])
```

Rescaling: Standardization

- rescale to 0 mean, standard deviation of 1
 - $X_{\text{scaled}} = (X - X.\text{mean}()) / X.\text{std}()$

```
In [54]: 1 from sklearn.preprocessing import StandardScaler
          2
          3 # instantiate
          4 ss = StandardScaler(with_mean=True, with_std=True) # default is center and scale
          5
          6 # fit to the data
          7 ss.fit(df_taxi[['trip_duration', 'tip_amount']])
          8
          9 # transform the data
         10 X_ss = ss.transform(df_taxi[['trip_duration', 'tip_amount']])
         11 X_ss[:2].round(2)
```

```
Out[54]: array([[ -0.5 , -0.48],
                [-0.17, -0.91]])
```

```
In [55]: 1 df_taxi_ss = pd.DataFrame(X_ss, columns=['trip_duration_scaled', 'tip_amount_scaled'])
          2 df_taxi_ss.agg(['mean', 'std', 'min', 'max'], axis=0).round(2)
```

```
Out[55]:
```

	trip_duration_scaled	tip_amount_scaled
mean	0.00	-0.00
std	1.00	1.00
min	-1.54	-1.54
max	5.62	4.88

Rescaling: Min-Max

- rescale values between 0 and 1
- $X_{\text{scaled}} = (X - X.\text{min}()) / (X.\text{max}() - X.\text{min}())$
- removes negative values

Rescaling: Min-Max

- rescale values between 0 and 1
- $X_{\text{scaled}} = (X - X.\text{min}()) / (X.\text{max}() - X.\text{min}())$
- removes negative values

```
In [56]: 1 from sklearn.preprocessing import MinMaxScaler
          2
          3 X_mms = MinMaxScaler(feature_range=(0,1) # default is to rescale between 0 and 1
          4                                     ).fit_transform(df_taxi[['trip_duration', 'tip_amount']])
          5
          6 df_taxi_mms = pd.DataFrame(X_mms, columns=['trip_duration_scaled', 'tip_amount_scaled'])
          7 df_taxi_mms.agg(['mean', 'std', 'min', 'max']).round(2)
```

Out[56]:

	trip_duration_scaled	tip_amount_scaled
mean	0.21	0.24
std	0.14	0.16
min	0.00	0.00
max	1.00	1.00

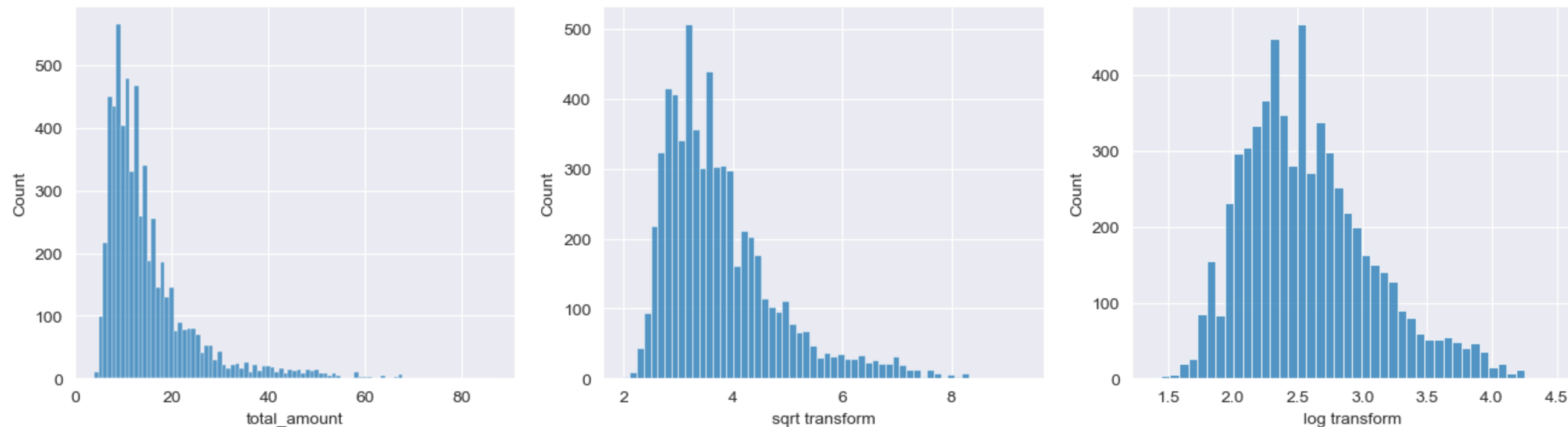
Dealing with Skew

- Many models expect "normal", symmetric data (ex: linear models)
- Highly skewed: tail has larger effect on model (outliers?)
- Transform with `log` or `sqrt`

Dealing with Skew

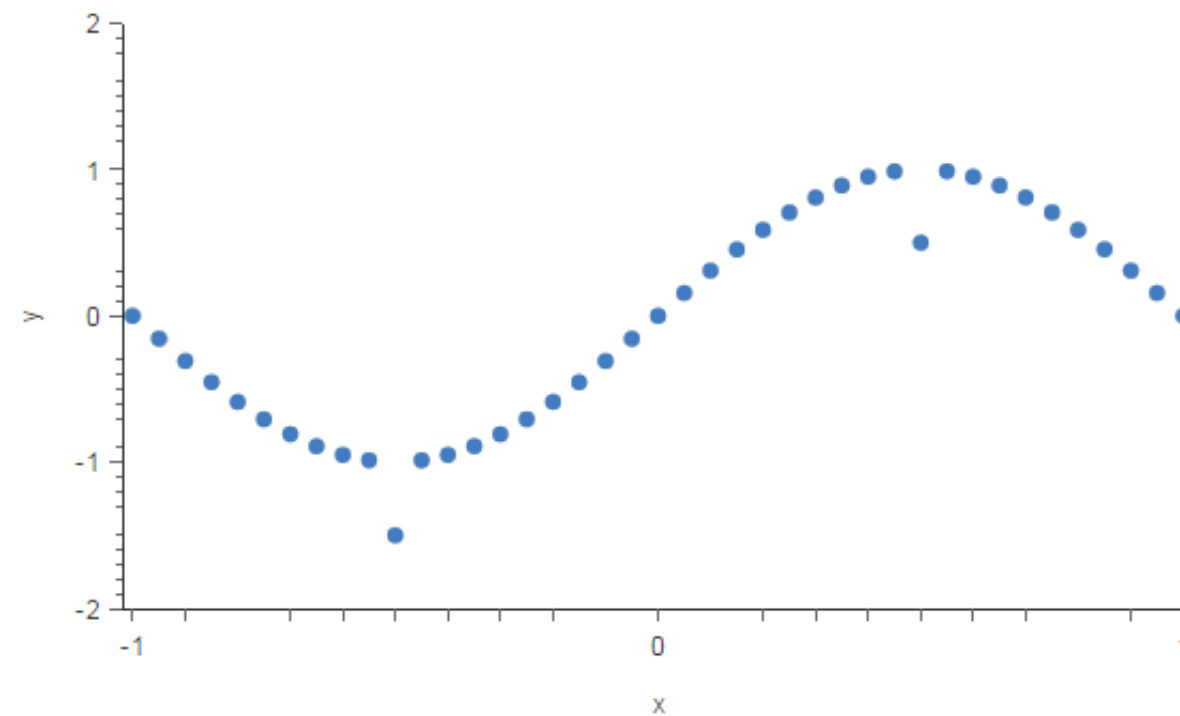
- Many models expect "normal", symmetric data (ex: linear models)
- Highly skewed: tail has larger effect on model (outliers?)
- Transform with `log` or `sqrt`

```
In [57]: 1 fig,ax = plt.subplots(1,3,figsize=(16,4))
2 sns.histplot(x=df_taxi.total_amount, ax=ax[0]);
3 sns.histplot(x=df_taxi.total_amount.apply(np.sqrt), ax=ax[1]); ax[1].set_xlabel('sqrt transform');
4 sns.histplot(x=df_taxi.total_amount.apply(np.log), ax=ax[2]); ax[2].set_xlabel('log transform');
```



Outliers

- Similar to missing data:
 - human data entry error
 - instrument measurement errors
 - data processing errors
 - natural deviations



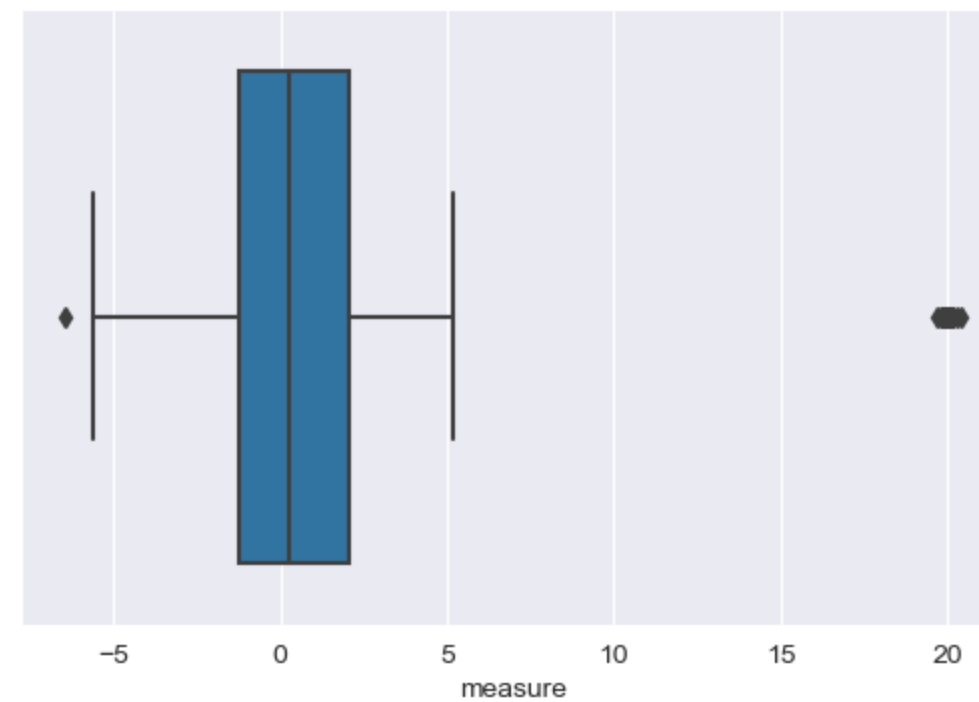
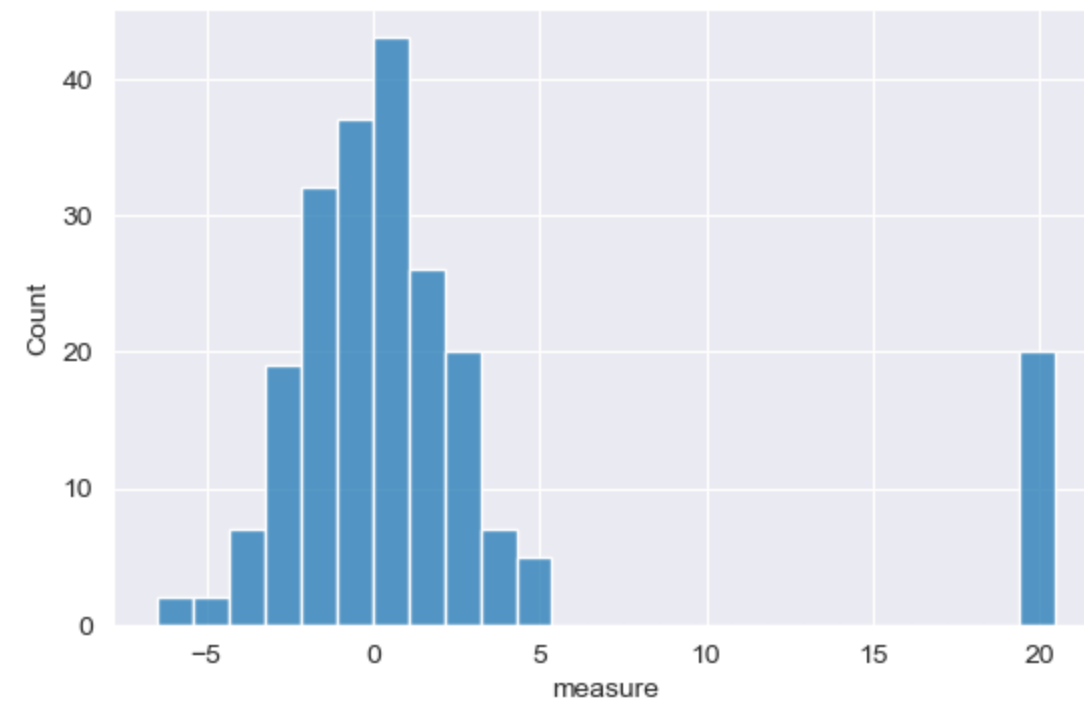
Outliers

- Why worry about them?
 - can give misleading results
 - can indicate issues in data/measurement
- Detecting Outliers
 - understand your data!
 - visualizations
 - $1.5 \times \text{IQR}$
 - z-scores
 - etc..

Detecting Outliers

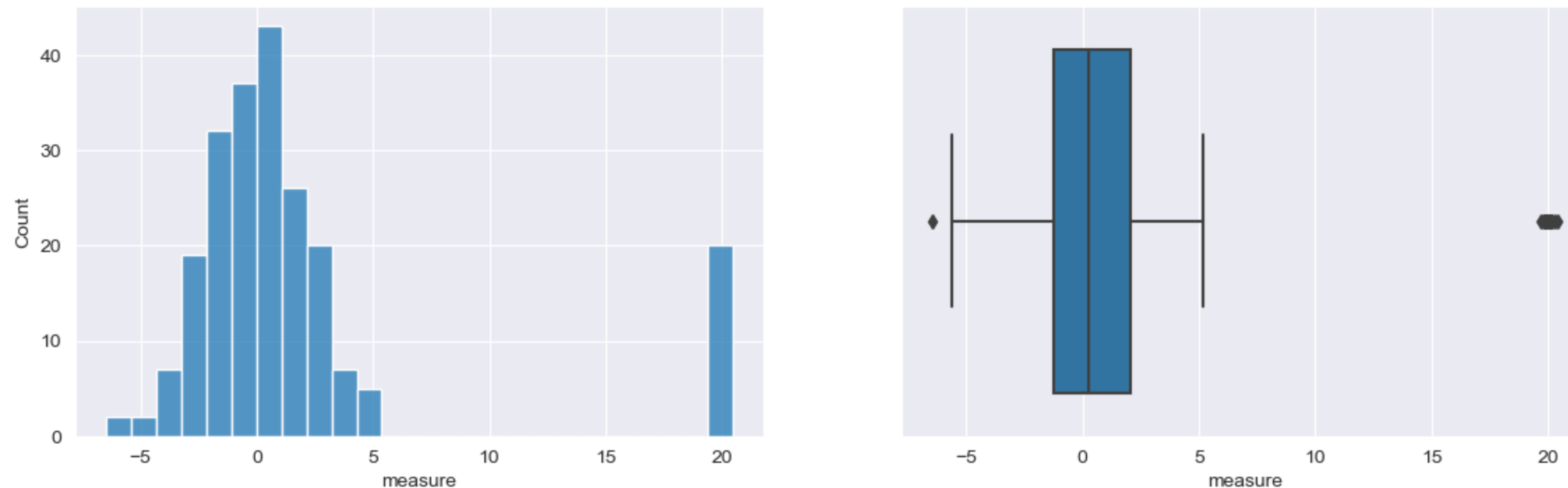
Detecting Outliers

```
In [58]: 1 np.random.seed(123)
2 data_rand = np.concatenate([np.random.normal(0,2,200),np.random.normal(20,.2,20)])
3 df_rand = pd.DataFrame({'measure':data_rand})
4
5 fig,ax = plt.subplots(1,2, figsize=(14,4))
6 sns.histplot(x=df_rand.measure,ax=ax[0]);sns.boxplot(x=df_rand.measure,ax=ax[1]);
```



Detecting Outliers

```
In [58]: 1 np.random.seed(123)
2 data_rand = np.concatenate([np.random.normal(0,2,200),np.random.normal(20,.2,20)])
3 df_rand = pd.DataFrame({'measure':data_rand})
4
5 fig,ax = plt.subplots(1,2, figsize=(14,4))
6 sns.histplot(x=df_rand.measure,ax=ax[0]);sns.boxplot(x=df_rand.measure,ax=ax[1]);
```

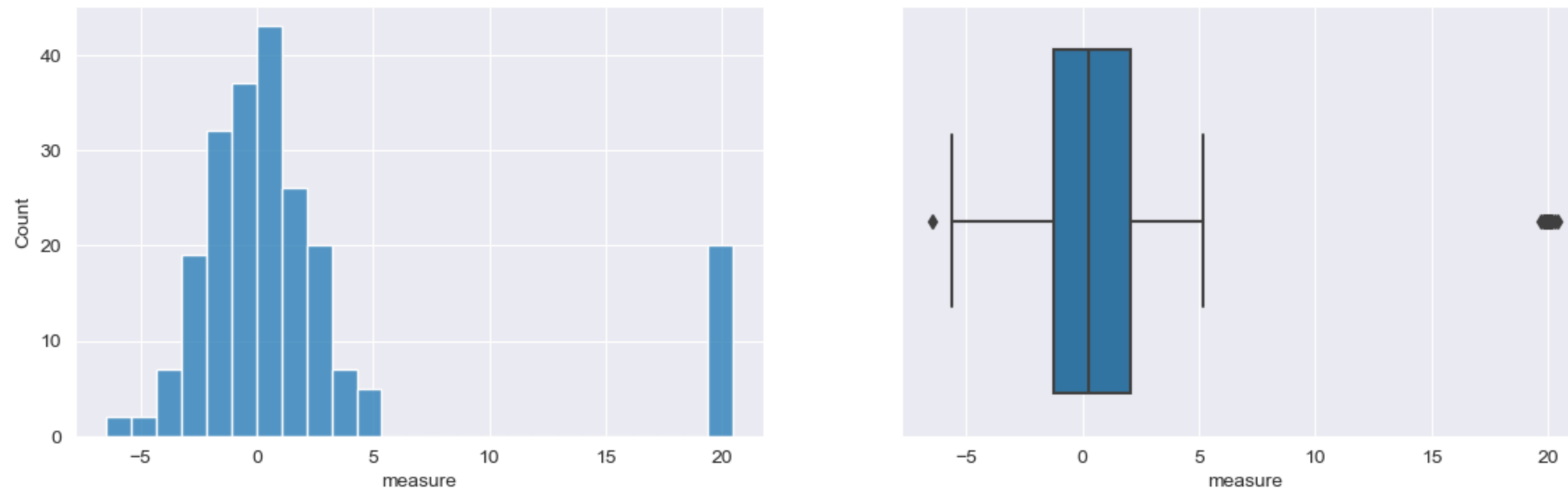


```
In [59]: 1 # Calculating IQR
2 p25,p75 = df_rand.measure.quantile([.25,.75])
3 iqr = p75 - p25
4 round(iqr,2)
```

Out[59]: 3.3

Detecting Outliers

```
In [58]: 1 np.random.seed(123)
2 data_rand = np.concatenate([np.random.normal(0,2,200),np.random.normal(20,.2,20)])
3 df_rand = pd.DataFrame({'measure':data_rand})
4
5 fig,ax = plt.subplots(1,2, figsize=(14,4))
6 sns.histplot(x=df_rand.measure,ax=ax[0]);sns.boxplot(x=df_rand.measure,ax=ax[1]);
```



```
In [59]: 1 # Calculating IQR
2 p25,p75 = df_rand.measure.quantile([.25,.75])
3 iqr = p75 - p25
4 round(iqr,2)
```

Out[59]: 3.3

```
In [60]: 1 # Finding outliers with IQR (first two examples found)
2 df_rand.measure[(df_rand.measure > p75+(1.5*iqr)) | (df_rand.measure < p25-(1.5*iqr))].sort_values().head(2).round(2)
```

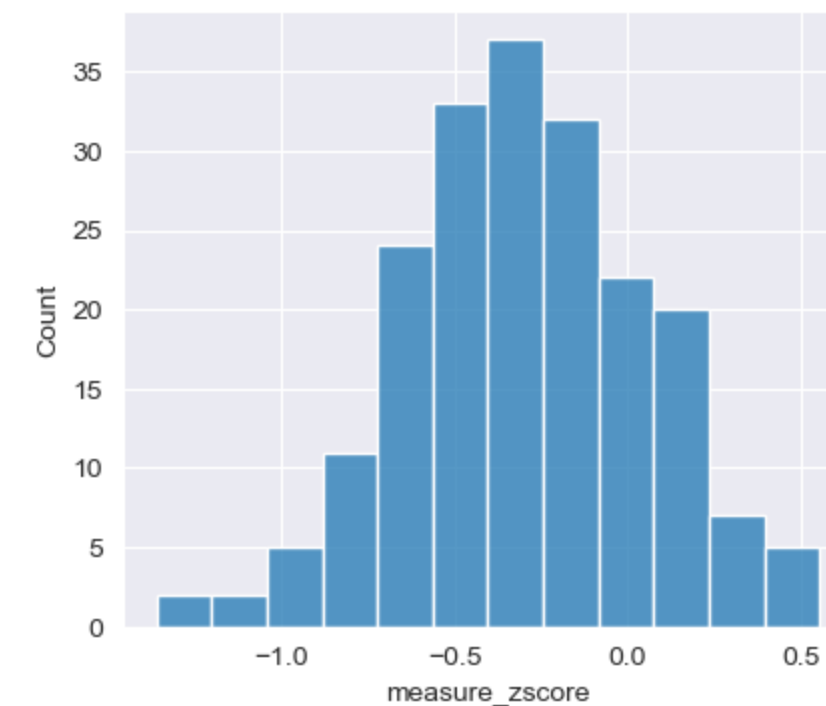
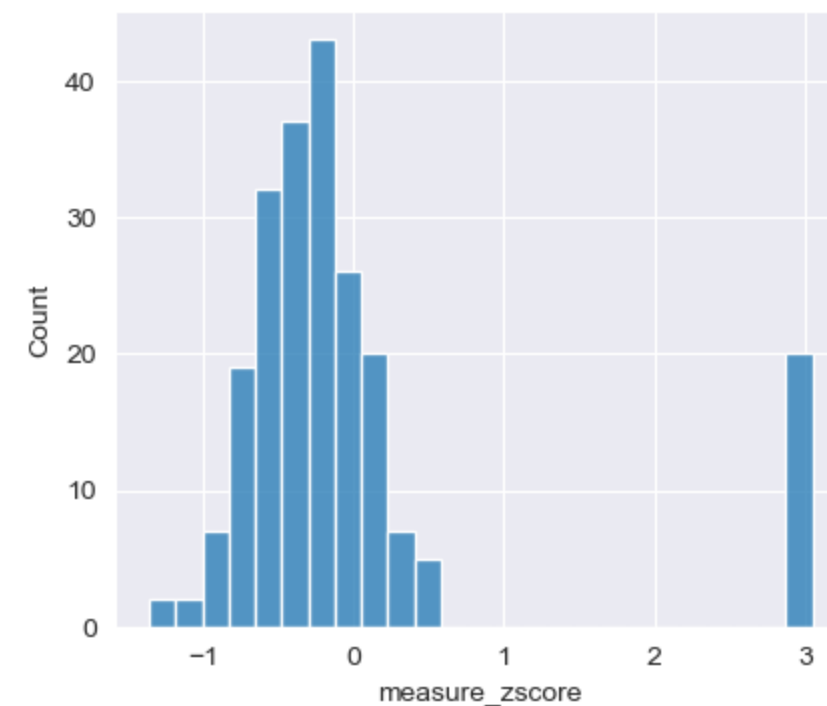
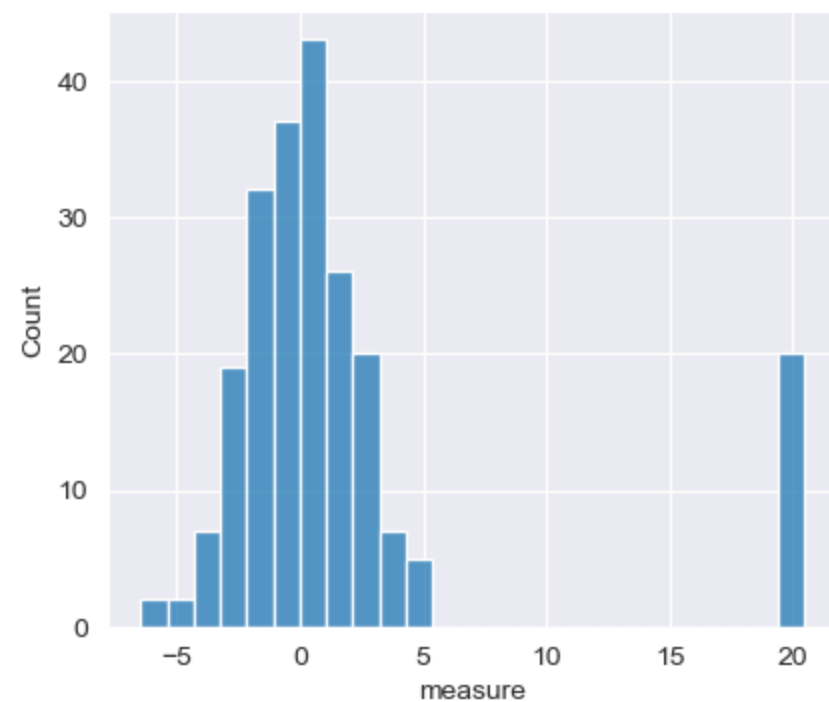
Detecting Outliers with z-score

Detecting Outliers with z-score

```
In [61]: 1 # zscore
2 df_rand['measure_zscore'] = (df_rand.measure - df_rand.measure.mean()) / df_rand.measure.std()
3
4 fig, ax = plt.subplots(1,3,figsize=(16,4))
5 sns.histplot(x=df_rand.measure,ax=ax[0]);
6 sns.histplot(x=df_rand.measure_zscore, ax=ax[1]);
7
8 keep_idx = np.abs(df_rand.measure_zscore) < 2
9 sns.histplot(x=df_rand[keep_idx].measure_zscore, ax=ax[2]);
10
11 # sample of points getting dropped
12 df_rand[np.abs(df_rand.measure_zscore) >= 2].sort_values(by='measure').head(3).round(2)
```

Out[61]:

	measure	measure_zscore
213	19.72	2.93
207	19.82	2.94
218	19.85	2.95



Other Outlier Detection Methods

- Many more parametric and non-parametric methods
 - Standardized Residuals
 - DBScan
 - EllipticEnvelope
 - IsolationForest
 - other Anomaly Detection techniques
 - See [sklearn docs on Outlier Detection](#) for more details

Dealing with Outliers

- How to deal with outliers?
 - drop data
 - treat as missing
 - encode with dummy variable first

Putting It All Together: Different Styles

Putting It All Together: Different Styles

```
In [62]: 1 df_shop1 = pd.read_csv('../data/flowershop_data_with_dups_week8.csv')
          2 df_shop1 = df_shop1.drop_duplicates()
          3 df_shop1['purchase_date'] = pd.to_datetime(df_shop1.purchase_date)
          4 df_shop1['price_missing'] = df_shop1.price.isna().astype(int)
          5 df_shop1['price'] = df_shop1.price.fillna(df_shop1.price.mean())
          6 df_shop1['price_scaled'] = StandardScaler().fit_transform(df_shop1[['price']])
          7 df_shop1['favorite_flower_missing'] = df_shop1.favorite_flower.isna().astype(int)
          8 df_shop1['favorite_flower'] = SimpleImputer(strategy='most_frequent').fit(df_shop1[['favorite_flower']])
```

Putting It All Together: Different Styles

```
In [62]: 1 df_shop1 = pd.read_csv('../data/flowershop_data_with_dups_week8.csv')
2 df_shop1 = df_shop1.drop_duplicates()
3 df_shop1['purchase_date'] = pd.to_datetime(df_shop1.purchase_date)
4 df_shop1['price_missing'] = df_shop1.price.isna().astype(int)
5 df_shop1['price'] = df_shop1.price.fillna(df_shop1.price.mean())
6 df_shop1['price_scaled'] = StandardScaler().fit_transform(df_shop1[['price']])
7 df_shop1['favorite_flower_missing'] = df_shop1.favorite_flower.isna().astype(int)
8 df_shop1['favorite_flower'] = SimpleImputer(strategy='most_frequent').fit(df_shop1[['favorite_flower']])
```

```
In [63]: 1 df_shop2 = (  
2     pd.read_csv('../data/flowershop_data_with_dups_week8.csv')  
3     .drop_duplicates()  
4     .assign(  
5         purchase_date      = lambda df_ : pd.to_datetime(df_.purchase_date),  
6         price_missing      = lambda df_ : df_.price.isna().astype(int),  
7         price              = lambda df_ : df_.price.fillna(df_.price.mean()),  
8         price_scaled       = lambda df_ : StandardScaler().fit_transform(df_[['price']]),  
9         favorite_flower_missing = lambda df_ : df_.favorite_flower.isna().astype(int),  
10        favorite_flower    = lambda df_ : (SimpleImputer(strategy='most_frequent')  
11                                           .fit_transform(df_shop1[['favorite_flower']] )  
12    )  
13 )  
14 )
```

Putting It All Together: Different Styles

```
In [62]: 1 df_shop1 = pd.read_csv('../data/flowershop_data_with_dups_week8.csv')
2 df_shop1 = df_shop1.drop_duplicates()
3 df_shop1['purchase_date'] = pd.to_datetime(df_shop1.purchase_date)
4 df_shop1['price_missing'] = df_shop1.price.isna().astype(int)
5 df_shop1['price'] = df_shop1.price.fillna(df_shop1.price.mean())
6 df_shop1['price_scaled'] = StandardScaler().fit_transform(df_shop1[['price']])
7 df_shop1['favorite_flower_missing'] = df_shop1.favorite_flower.isna().astype(int)
8 df_shop1['favorite_flower'] = SimpleImputer(strategy='most_frequent').fit(df_shop1[['favorite_flower']])
```

```
In [63]: 1 df_shop2 = (
2     pd.read_csv('../data/flowershop_data_with_dups_week8.csv')
3     .drop_duplicates()
4     .assign(
5         purchase_date = lambda df_ : pd.to_datetime(df_.purchase_date),
6         price_missing = lambda df_ : df_.price.isna().astype(int),
7         price = lambda df_ : df_.price.fillna(df_.price.mean()),
8         price_scaled = lambda df_ : StandardScaler().fit_transform(df_[['price']]),
9         favorite_flower_missing = lambda df_ : df_.favorite_flower.isna().astype(int),
10        favorite_flower = lambda df_ : (SimpleImputer(strategy='most_frequent')
11                                         .fit_transform(df_shop1[['favorite_flower']])
12                                         )
13    )
14 )
```

```
In [64]: 1 pd.testing.assert_frame_equal(df_shop1,df_shop2) # throws an exeption when data frames are not the same
```

Data Cleaning Review

- duplicate data
- missing data
- rescaling
- dealing with skew
- outlier detection

Feature Engineering

- Binning
- One-Hot encoding
- Derived Features

Binning

- Transform continuous features to categorical
- Use:
 - `pd.cut`
 - `sklearn.preprocessing.KBinsDiscretizer` (combined binning and one-hot-encoding)

Binning

- Transform continuous features to categorical
- Use:
 - `pd.cut`
 - `sklearn.preprocessing.KBinsDiscretizer` (combined binning and one-hot-encoding)

```
In [65]: 1 trip_duration_bins = [df_taxi.trip_duration.min(),
2                               df_taxi.trip_duration.median(),
3                               df_taxi.trip_duration.quantile(0.75),
4                               df_taxi.trip_duration.max(),]
```

```
In [66]: 1 df_taxi_bin = df_taxi_raw.copy()
2 df_taxi_bin['trip_duration_binned'] = pd.cut(df_taxi_bin.trip_duration,
3                                              bins=trip_duration_bins,           # can pass bin edges or number of bins
4                                              labels=['short', 'medium', 'long'],
5                                              right=True,                       # all bins right-inclusive
6                                              include_lowest=True              # first interval left-inclusive
7                                              )
8 df_taxi_bin[['trip_duration', 'trip_duration_binned']].iloc[:3]
9
```

Out[66]:

	trip_duration	trip_duration_binned
1	516	short
2	683	medium
7	834	medium

One-Hot Encoding

- Encode categorical features for models that can't handle categorical (eg. Linear)
- One column per category, '1' in only one column per row
- Use `pd.get_dummies()` or `sklearn.preprocessing.OneHotEncoder`

One-Hot Encoding

- Encode categorical features for models that can't handle categorical (eg. Linear)
- One column per category, '1' in only one column per row
- Use `pd.get_dummies()` or `sklearn.preprocessing.OneHotEncoder`

```
In [67]: 1 pd.get_dummies(df_taxi_bin.trip_duration_binned, prefix='trip_duration').iloc[:2]
```

```
Out[67]:
```

		trip_duration_short	trip_duration_medium	trip_duration_long
1	1	0	0	
2	0	1	0	

One-Hot Encoding

- Encode categorical features for models that can't handle categorical (eg. Linear)
- One column per category, '1' in only one column per row
- Use `pd.get_dummies()` or `sklearn.preprocessing.OneHotEncoder`

```
In [67]: 1 pd.get_dummies(df_taxi_bin.trip_duration_binned, prefix='trip_duration').iloc[:2]
```

Out[67]:

		trip_duration_short	trip_duration_medium	trip_duration_long
1	1	0	0	
2	0	1	0	

```
In [68]: 1 # to add back to dataframe, use join (will discuss .join() next time)
2 df_taxi_bin.join(pd.get_dummies(df_taxi_bin.trip_duration_binned, prefix='trip_duration')).iloc[:2,-6:] # not saved
```

Out[68]:

	total_amount	trip_duration	trip_duration_binned	trip_duration_short	trip_duration_medium	trip_duration_long
1	9.96	516	short	1	0	0
2	10.30	683	medium	0	1	0

One-Hot Encoding

- Encode categorical features for models that can't handle categorical (eg. Linear)
- One column per category, '1' in only one column per row
- Use `pd.get_dummies()` or `sklearn.preprocessing.OneHotEncoder`

```
In [67]: 1 pd.get_dummies(df_taxi_bin.trip_duration_binned, prefix='trip_duration').iloc[:2]
```

Out[67]:

		trip_duration_short	trip_duration_medium	trip_duration_long
1	1	0	0	
2	0	1	0	

```
In [68]: 1 # to add back to dataframe, use join (will discuss .join() next time)
2 df_taxi_bin.join(pd.get_dummies(df_taxi_bin.trip_duration_binned, prefix='trip_duration')).iloc[:2,-6:] # not saved
```

Out[68]:

	total_amount	trip_duration	trip_duration_binned	trip_duration_short	trip_duration_medium	trip_duration_long
1	9.96	516	short	1	0	0
2	10.30	683	medium	0	1	0

```
In [69]: 1 # or let pandas determine which columns to one-hot
2 pd.get_dummies(df_taxi_bin).iloc[:2,-6:] # not being saved
```

Out[69]:

	trip_duration	store_and_fwd_flag_N	store_and_fwd_flag_Y	trip_duration_binned_short	trip_duration_binned_medium	trip_duration_binned_long
1	516	1	0	1	0	0
2	683	1	0	0	1	0

One-Hot Encoding with sklearn

One-Hot Encoding with sklearn

```
In [70]: 1 from sklearn.preprocessing import OneHotEncoder
          2
          3 ohe = OneHotEncoder(categories=[['short', 'medium', 'long']], # or leave as 'auto'
          4                           sparse=True,
          5                           handle_unknown='ignore')           # will raise error otherwise
          6
          7 ohe.fit(df_taxi_bin[['trip_duration_binned']])
          8 ohe.categories_
```

```
Out[70]: [array(['short', 'medium', 'long'], dtype=object)]
```

One-Hot Encoding with sklearn

```
In [70]: 1 from sklearn.preprocessing import OneHotEncoder
2
3 ohe = OneHotEncoder(categories=[['short', 'medium', 'long']], # or leave as 'auto'
4                       sparse=True,
5                       handle_unknown='ignore') # will raise error otherwise
6
7 ohe.fit(df_taxi_bin[['trip_duration_binned']])
8 ohe.categories_
```

```
Out[70]: [array(['short', 'medium', 'long'], dtype=object)]
```

```
In [71]: 1 ohe.transform(df_taxi_bin[['trip_duration_binned']])[:3] # returns a sparse matrix!
```

```
Out[71]: <3x3 sparse matrix of type '<class 'numpy.float64'>'
         with 3 stored elements in Compressed Sparse Row format>
```

One-Hot Encoding with sklearn

```
In [70]: 1 from sklearn.preprocessing import OneHotEncoder
          2
          3 ohe = OneHotEncoder(categories=[['short', 'medium', 'long']], # or leave as 'auto'
          4                           sparse=True,
          5                           handle_unknown='ignore')           # will raise error otherwise
          6
          7 ohe.fit(df_taxi_bin[['trip_duration_binned']])
          8 ohe.categories_
```

```
Out[70]: [array(['short', 'medium', 'long'], dtype=object)]
```

```
In [71]: 1 ohe.transform(df_taxi_bin[['trip_duration_binned']])[:3] # returns a sparse matrix!
```

```
Out[71]: <3x3 sparse matrix of type '<class 'numpy.float64'>'
          with 3 stored elements in Compressed Sparse Row format>
```

```
In [72]: 1 ohe.transform(df_taxi_bin[['trip_duration_binned']])[:3].todense() # use .todense() to convert sparse to dense
```

```
Out[72]: matrix([[1., 0., 0.],
                  [0., 1., 0.],
                  [0., 1., 0.]])
```

Bin and One-Hot Encode with sklearn

Bin and One-Hot Encode with sklearn

```
In [73]: 1 from sklearn.preprocessing import KBinsDiscretizer
2
3 # NOTE: We're not setting the bin edges explicitly
4 #       For control over bin edges, use Binarizer
5 kbd = KBinsDiscretizer(n_bins=3,
6                        encode="onehot",      # or onehot (sparse), ordinal
7                        strategy="quantile",  # or uniform or kmeans (clustering)
8                        ).fit(df_taxi[['trip_duration']])
9 print(kbd.bin_edges_)
10 print(kbd.bin_edges_[0].astype(int))
```

```
[array([2.000e+00, 4.780e+02, 8.700e+02, 3.556e+03])]
[  2  478  870 3556]
```

Bin and One-Hot Encode with sklearn

```
In [73]: 1 from sklearn.preprocessing import KBinsDiscretizer
2
3 # NOTE: We're not setting the bin edges explicitly
4 #       For control over bin edges, use Binarizer
5 kbd = KBinsDiscretizer(n_bins=3,
6                         encode="onehot",      # or onehot (sparse), ordinal
7                         strategy="quantile", # or uniform or kmeans (clustering)
8                         ).fit(df_taxi[['trip_duration']])
9 print(kbd.bin_edges_)
10 print(kbd.bin_edges_[0].astype(int))
```

```
[array([2.000e+00, 4.780e+02, 8.700e+02, 3.556e+03])]
[  2  478  870 3556]
```

```
In [74]: 1 df_taxi[['trip_duration']].tail(3)
```

Out[74]:

	trip_duration
9994	905
9995	296
9997	2089

Bin and One-Hot Encode with sklearn

```
In [73]: 1 from sklearn.preprocessing import KBinsDiscretizer
2
3 # NOTE: We're not setting the bin edges explicitly
4 #       For control over bin edges, use Binarizer
5 kbd = KBinsDiscretizer(n_bins=3,
6                        encode="onehot",      # or onehot (sparse), ordinal
7                        strategy="quantile",  # or uniform or kmeans (clustering)
8                        ).fit(df_taxi[['trip_duration']])
9 print(kbd.bin_edges_)
10 print(kbd.bin_edges_[0].astype(int))
```

```
[array([2.000e+00, 4.780e+02, 8.700e+02, 3.556e+03])]
[  2  478  870 3556]
```

```
In [74]: 1 df_taxi[['trip_duration']].tail(3)
```

Out[74]:

	trip_duration
9994	905
9995	296
9997	2089

```
In [75]: 1 kbd.transform(df_taxi[['trip_duration']])[-3:]
```

Out[75]: <3x3 sparse matrix of type '<class 'numpy.float64'>'
with 3 stored elements in Compressed Sparse Row format>

Bin and One-Hot Encode with sklearn

```
In [73]: 1 from sklearn.preprocessing import KBinsDiscretizer
2
3 # NOTE: We're not setting the bin edges explicitly
4 #       For control over bin edges, use Binarizer
5 kbd = KBinsDiscretizer(n_bins=3,
6                         encode="onehot",      # or onehot (sparse), ordinal
7                         strategy="quantile",  # or uniform or kmeans (clustering)
8                         ).fit(df_taxi[['trip_duration']])
9 print(kbd.bin_edges_)
10 print(kbd.bin_edges_[0].astype(int))
```

```
[array([2.000e+00, 4.780e+02, 8.700e+02, 3.556e+03])]
[  2  478  870 3556]
```

```
In [74]: 1 df_taxi[['trip_duration']].tail(3)
```

Out[74]:

	trip_duration
9994	905
9995	296
9997	2089

```
In [75]: 1 kbd.transform(df_taxi[['trip_duration']])[-3:]
```

Out[75]: <3x3 sparse matrix of type '<class 'numpy.float64'>'
with 3 stored elements in Compressed Sparse Row format>

```
In [76]: 1 kbd.transform(df_taxi[['trip_duration']])[-3:].todense()
```

Out[76]: matrix([[0., 0., 1.],
[1., 0., 0.],
[0., 0., 1.]])

Dealing with Ordinal Variables

Dealing with Ordinal Variables

```
In [77]: 1 df_pml = pd.DataFrame([[ 'green', 'M', 10.1, 'class2' ],
2                               [ 'red', 'L', 13.5, 'class1' ],
3                               [ 'blue', 'XL', 15.3, 'class2' ]],
4                               columns=[ 'color', 'size', 'price', 'classlabel' ])
5 df_pml
```

Out[77]:

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

Dealing with Ordinal Variables

```
In [77]: 1 df_pml = pd.DataFrame([[ 'green', 'M', 10.1, 'class2' ],
2                               [ 'red', 'L', 13.5, 'class1' ],
3                               [ 'blue', 'XL', 15.3, 'class2' ]],
4                               columns=[ 'color', 'size', 'price', 'classlabel' ])
5 df_pml
```

Out[77]:

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

```
In [78]: 1 # if we know the numerical difference between ordinal values
2 # eg XL = L+1 = M+2
3
4 size_mapping = { 'XL':3,
5                  'L':2,
6                  'M':1}
7
8 df_pml_features = pd.DataFrame()
9
10 df_pml_features[ 'size' ] = df_pml[ 'size' ].map(size_mapping)
11 df_pml_features
```

Out[78]:

	size
0	1
1	2
2	3

Dealing with Ordinal Variables Cont.

Dealing with Ordinal Variables Cont.

In [79]:

```
1 df_pml
```

Out[79]:

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

Dealing with Ordinal Variables Cont.

In [79]:

```
1 df_pml
```

Out[79]:

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

In [80]:

```
1 # if we don't know the numerical difference between ordinal values
2 # generate threshold features
3 df_pml_features = pd.DataFrame()
4 df_pml_features['x > M'] = df_pml['size'].apply(lambda x: 1 if x in ['L','XL'] else 0)
5 df_pml_features['x > L'] = df_pml['size'].apply(lambda x: 1 if x == 'XL' else 0)
6 df_pml_features
```

Out[80]:

	x > M	x > L
0	0	0
1	1	0
2	1	1

Derived Features

- Anything that is a transformation of our data
- This is where the money is!
- Examples:
 - "is a high demand pickup location"
 - "is a problem house sale"
 - "high-performing job candidate"

Polynomial Features

Polynomial Features

```
In [81]: 1 from sklearn.preprocessing import PolynomialFeatures
2
3 pf = PolynomialFeatures(degree=2,
4                          include_bias=False)
5 X_new = pf.fit_transform(df_taxi[['passenger_count', 'trip_duration']])
6
7 new_columns = ['passenger_count', 'trip_duration', 'passenger_count^2', 'passenger_count*trip_duration', 'trip_duration^2']
8 pd.DataFrame(X_new[3:5], columns=new_columns)
```

Out[81]:

	passenger_count	trip_duration	passenger_count^2	passenger_count*trip_duration	trip_duration^2
0	3.0	298.0	9.0	894.0	88804.0
1	1.0	396.0	1.0	396.0	156816.0

Python String Functions

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

```
In [84]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[84]: (['D.S.', 'is', 'good!'], ['D', 'S', ' is good!'])
```

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

```
In [84]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[84]: (['D.S.', 'is', 'good!'], ['D', 'S', ' is good!'])
```

```
In [85]: 1 '|'.join(['ab','c','d'])      # join items in a list together
```

```
Out[85]: 'ab|c|d'
```

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

```
In [84]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[84]: (['D.S.', 'is', 'good!'], ['D', 'S', ' is good!'])
```

```
In [85]: 1 '|'.join(['ab','c','d'])      # join items in a list together
```

```
Out[85]: 'ab|c|d'
```

```
In [86]: 1 '|'.join(doc[:5])             # a string itself is treated like a list of characters
```

```
Out[86]: 'D|. |S|. | '
```

Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

```
In [84]: 1 doc.split() , doc.split('.') # split a string into parts (default is whitespace)
```

```
Out[84]: (['D.S.', 'is', 'good!'], ['D', 'S', ' is good!'])
```

```
In [85]: 1 '|'.join(['ab','c','d'])      # join items in a list together
```

```
Out[85]: 'ab|c|d'
```

```
In [86]: 1 '|'.join(doc[:5])             # a string itself is treated like a list of characters
```

```
Out[86]: 'D|. |S|. | '
```

```
In [87]: 1 ' test '.strip()              # remove whitespace from the beginning and end of a string
```

```
Out[87]: 'test'
```


Python String Functions

```
In [82]: 1 doc = "D.S. is good!"  
        2 doc
```

```
Out[82]: 'D.S. is good!'
```

```
In [83]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[83]: ('d.s. is good!', 'D.S. IS GOOD!')
```

```
In [84]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[84]: (['D.S.', 'is', 'good!'], ['D', 'S', ' ' is good!'])
```

```
In [85]: 1 '|'.join(['ab','c','d'])      # join items in a list together
```

```
Out[85]: 'ab|c|d'
```

```
In [86]: 1 '|'.join(doc[:5])             # a string itself is treated like a list of characters
```

```
Out[86]: 'D|. |S|. | '
```

```
In [87]: 1 ' test '.strip()              # remove whitespace from the beginning and end of a string
```

```
Out[87]: 'test'
```

and more, see <https://docs.python.org/3.8/library/string.html>

String Functions in Pandas

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

```
In [89]: 1 df_shop.loc[:, 'lastname'].iloc[:2].str.lower()
```

```
Out[89]: 0    perkins  
         1    robinson  
         Name: lastname, dtype: object
```

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

```
In [89]: 1 df_shop.loc[:, 'lastname'].iloc[:2].str.lower()
```

```
Out[89]: 0    perkins  
         1    robinson  
         Name: lastname, dtype: object
```

```
In [90]: 1 df_shop.lastname[:2].str.capitalize()
```

```
Out[90]: 0    Perkins  
         1    Robinson  
         Name: lastname, dtype: object
```

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

```
In [89]: 1 df_shop.loc[:, 'lastname'].iloc[:2].str.lower()
```

```
Out[89]: 0    perkins  
         1    robinson  
         Name: lastname, dtype: object
```

```
In [90]: 1 df_shop.lastname[:2].str.capitalize()
```

```
Out[90]: 0    Perkins  
         1    Robinson  
         Name: lastname, dtype: object
```

```
In [91]: 1 df_shop.lastname[:2].str.startswith('ROB') # .endswith() , .contains()
```

```
Out[91]: 0    False  
         1     True  
         Name: lastname, dtype: bool
```

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

```
In [89]: 1 df_shop.loc[:, 'lastname'].iloc[:2].str.lower()
```

```
Out[89]: 0    perkins  
         1    robinson  
         Name: lastname, dtype: object
```

```
In [90]: 1 df_shop.lastname[:2].str.capitalize()
```

```
Out[90]: 0    Perkins  
         1    Robinson  
         Name: lastname, dtype: object
```

```
In [91]: 1 df_shop.lastname[:2].str.startswith('ROB') # .endswith() , .contains()
```

```
Out[91]: 0    False  
         1     True  
         Name: lastname, dtype: bool
```

```
In [92]: 1 df_shop.lastname[:2].str.replace('R', '^')
```

```
Out[92]: 0    PE^KINS  
         1    ^OBINSON  
         Name: lastname, dtype: object
```

String Functions in Pandas

```
In [88]: 1 df_shop.iloc[:2].loc[:, 'lastname']
```

```
Out[88]: 0    PERKINS  
         1    ROBINSON  
         Name: lastname, dtype: object
```

```
In [89]: 1 df_shop.loc[:, 'lastname'].iloc[:2].str.lower()
```

```
Out[89]: 0    perkins  
         1    robinson  
         Name: lastname, dtype: object
```

```
In [90]: 1 df_shop.lastname[:2].str.capitalize()
```

```
Out[90]: 0    Perkins  
         1    Robinson  
         Name: lastname, dtype: object
```

```
In [91]: 1 df_shop.lastname[:2].str.startswith('ROB') # .endswith() , .contains()
```

```
Out[91]: 0    False  
         1     True  
         Name: lastname, dtype: bool
```

```
In [92]: 1 df_shop.lastname[:2].str.replace('R', '^')
```

```
Out[92]: 0    PE^KINS  
         1    ^OBINSON  
         Name: lastname, dtype: object
```

and more: https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html#method-summary

Pandas datetime functions

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
        2    2017-01-11 14:47:52  
        Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
        2    2017-01-11 14:47:52  
        Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [94]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day
```

```
Out[94]: 1      5  
        2     11  
        Name: tpep_pickup_datetime, dtype: int64
```

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
        2    2017-01-11 14:47:52  
        Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [94]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day
```

```
Out[94]: 1      5  
        2     11  
        Name: tpep_pickup_datetime, dtype: int64
```

```
In [95]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day_of_week
```

```
Out[95]: 1      3  
        2      2  
        Name: tpep_pickup_datetime, dtype: int64
```

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
        2    2017-01-11 14:47:52  
        Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [94]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day
```

```
Out[94]: 1      5  
        2     11  
        Name: tpep_pickup_datetime, dtype: int64
```

```
In [95]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day_of_week
```

```
Out[95]: 1      3  
        2      2  
        Name: tpep_pickup_datetime, dtype: int64
```

```
In [96]: 1 (df_taxi.tpep_dropoff_datetime - df_taxi.tpep_pickup_datetime).dt.seconds.iloc[:2]
```

```
Out[96]: 1      516  
        2      683  
        dtype: int64
```

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
2    2017-01-11 14:47:52  
Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [94]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day
```

```
Out[94]: 1      5  
2     11  
Name: tpep_pickup_datetime, dtype: int64
```

```
In [95]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day_of_week
```

```
Out[95]: 1      3  
2      2  
Name: tpep_pickup_datetime, dtype: int64
```

```
In [96]: 1 (df_taxi.tpep_dropoff_datetime - df_taxi.tpep_pickup_datetime).dt.seconds.iloc[:2]
```

```
Out[96]: 1      516  
2      683  
dtype: int64
```

```
In [97]: 1 (pd.to_datetime('today') - df_taxi.tpep_dropoff_datetime).dt.days.div(365).iloc[:2].round(2)
```

```
Out[97]: 1      6.86  
2      6.84  
Name: tpep_dropoff_datetime, dtype: float64
```

Pandas datetime functions

```
In [93]: 1 df_taxi.iloc[:2].tpep_pickup_datetime
```

```
Out[93]: 1    2017-01-05 15:14:52  
        2    2017-01-11 14:47:52  
        Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [94]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day
```

```
Out[94]: 1      5  
        2     11  
        Name: tpep_pickup_datetime, dtype: int64
```

```
In [95]: 1 df_taxi.iloc[:2].tpep_pickup_datetime.dt.day_of_week
```

```
Out[95]: 1      3  
        2      2  
        Name: tpep_pickup_datetime, dtype: int64
```

```
In [96]: 1 (df_taxi.tpep_dropoff_datetime - df_taxi.tpep_pickup_datetime).dt.seconds.iloc[:2]
```

```
Out[96]: 1      516  
        2      683  
        dtype: int64
```

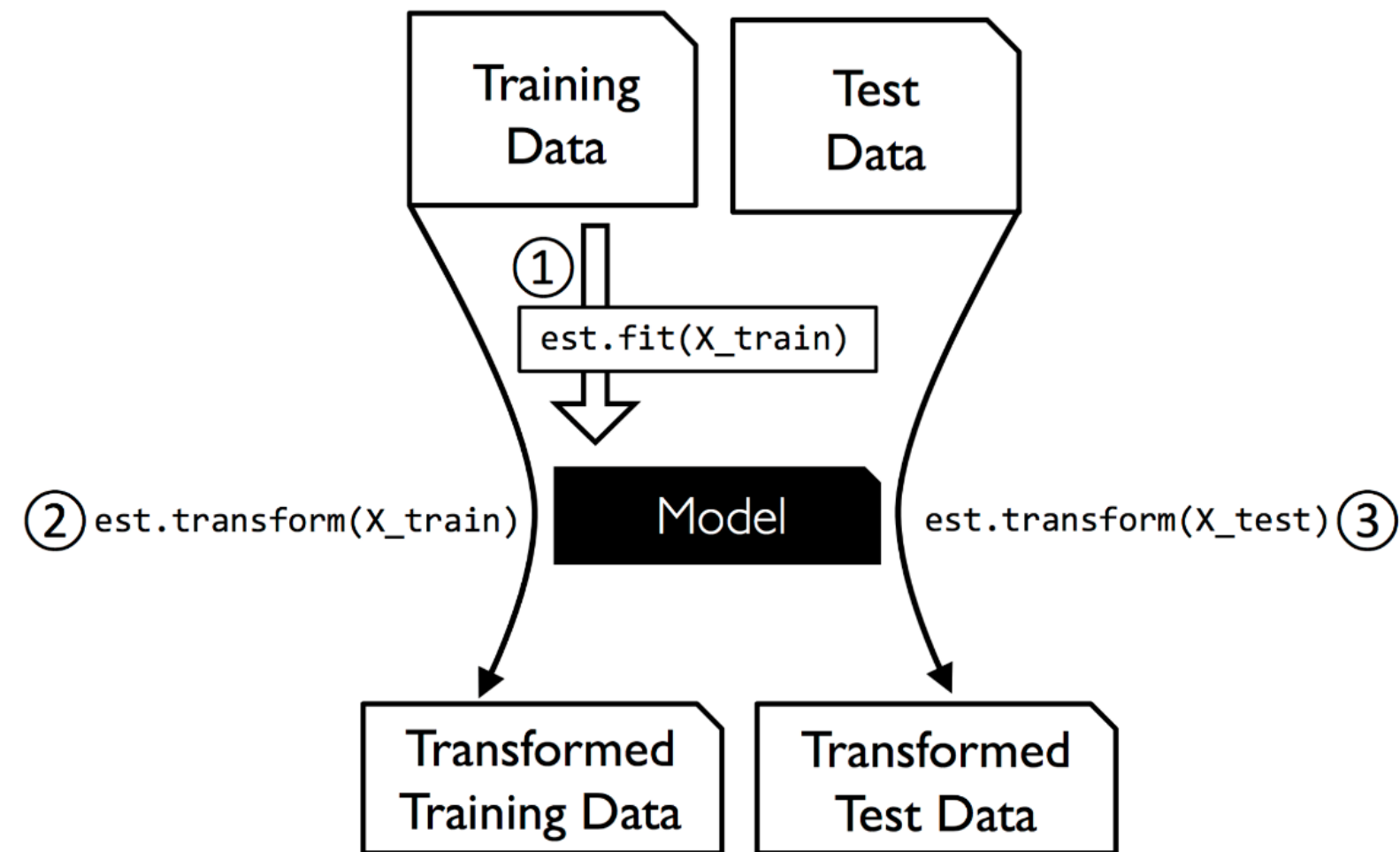
```
In [97]: 1 (pd.to_datetime('today') - df_taxi.tpep_dropoff_datetime).dt.days.div(365).iloc[:2].round(2)
```

```
Out[97]: 1      6.86  
        2      6.84  
        Name: tpep_dropoff_datetime, dtype: float64
```

and more: https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#time-date-

Transforming with Train/Test Split

- When performing data transformation



Next Time

- Dimensionality Reduction
 - Feature Selection
 - Feature Extraction

Questions?