CAP 5768: Introduction to Data Science

Data Preprocessing and Transformation in Python

Data Cleaning

- When working with multiple data sources, there are many chances for data to be incorrect, duplicated, or mislabeled. If data is wrong, outcomes and algorithms are unreliable, even though they may look correct.

- Data cleaning is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset.

- There's no such absolute way to describe the precise steps in the data cleaning process because the processes may vary from dataset to dataset.

- Bad data could be:
  - Empty cells
  - Data in wrong format
  - Wrong data
  - Duplicates

Why Is Data Cleaning Essential?

- Data cleaning is the most important task that should be done by a data science professional.
- Having wrong or bad-quality data can be detrimental to processes and analysis.
- Having clean data will ultimately increase overall productivity and permit the very best quality information in your decision-making.

Data Cleaning Cycle

- It is the method of analyzing, distinguishing, and correcting untidy, raw data.
- Data cleaning involves filling in missing values, handling outliers, and distinguishing and fixing errors present in the dataset. Whereas the techniques used for data cleaning might vary in step with different types of datasets.

Basic `pandas` EDA

The **`pandas`** package many functions that are highly useful for data exploration, such as:

`df.head()`	`df.index`
`df.tail()`	`df.columns`
`df.dtypes`	`df.keys()`
`df.shape`	`df.memory_usage( )`
`df.size`	`df.dropna()`
`df.ndim`	`nlargest(n, columns)`
`df.describe()`	`df.isna()`
`df.info()`	`df.duplicated().sum()`
`df.sample()`	`value_counts()`
`df.isnull().sum()`	`df.corr()`
`nunique()`	`df.pivot_table( )`
`unique()`	`df.select_dtypes(include='number')`

Ipython Interactive Shell and Display

- **IPython (Interactive Python) is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history.**

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

- **The IPython kernel that powers Jupyter notebook has a module named "display", which will provide us with a list of classes and methods used for displaying rich media contents of different types in Jupyter notebook and Jupyter lab.**

```
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

Data Types

Get the data types within a data frame:

```
df.dtypes
```

```
Out[90]: year               int64
         month              int64
         day                int64
         dep_time          object
         sched_dep_time     int64
         dep_delay        float64
         arr_time         float64
         sched_arr_time     int64
         arr_delay        float64
         carrier           object
         flight           float64
         tailnum           object
         origin            object
         dest              object
         air_time         float64
         distance         float64
         hour             float64
         minute           float64
         time_hour         object
         dtype: object
```

Select only a certain type:

```
df.select_dtypes(include='number')
df.select_dtypes(include='object')
df.select_dtypes(include='datetime')
df.select_dtypes(include='int64')
df.select_dtypes(include=['float64', 'object])
```

Convert Data Types

String to Numbers

```
df[‘arr_delay'] = df[‘arr_delay'].astype(int)
```

String to Date

```
df['dep_time'] = pd.to_datetime(df['dep_time']).dt.date
```

String to Time

```
df['dep_time'] = pd.to_datetime(df['dep_time']).dt.time
```

```
df['date'] = pd.to_datetime(df['date’],
                           format="%Y-%d-%m %H:%M:%S")
```

Extract day, month, and year from Datetime Datatypes

Extract Day Values

```
df.time_hour.dt.day
```

Extract Month Values

```
df.time_hour.dt.month
```

Extract Year Values

```
df.time_hour.dt.year
```

Change Column Names

```
df.rename(columns ={"A": "a", "B": "b", "C": "c"})
```

In [115]:

```
df[['year','month','day', 'dep_time']].head()
df.rename(columns ={'year':'Year','month':'Month','day':'Day', 'dep_time':'Departure'})
```

Out[115]:

	year	month	day	dep_time
0	2013	1	1	517.0
1	2013	1	1	533.0
2	2013	1	1	542.0
3	2013	1	1	544.0
4	2013	1	1	554.0

Out[115]:

	Year	Month	Day	Departure
0	2013	1	1	517.0
1	2013	1	1	533.0
2	2013	1	1	542.0
3	2013	1	1	544.0
4	2013	1	1	554.0

Merge Datasets with `merge()`

`merge()` performs join operations similar to relational databases like SQL.

```
df.merge(left, right, on =[‘key1’, ‘key2’])
```

- **one-to-one**: joining two DataFrame objects on their indexes which must contain unique values.
- **many-to-one**: joining a unique index to one or more columns in a different DataFrame.
- **many-to-many** : joining columns on columns.

```
df.merge(left, right, how= “outer”, on =[‘key1’, ‘key2’])
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K0	NaN	NaN	C3	D3
5	K2	K1	A3	B3	NaN	NaN

Merge Datasets with `concat()`

The **`concat()`** function concatenates an arbitrary amount of Series or DataFrame objects along an axis while performing optional set logic (union or intersection) of the indexes on the other axes.

```
axis = 1(column)
axis = 0 (row)

frames = [df1, df2, df3]
result = pd.concat(frames) same as pd.concat([df1,df2,df3])
result = pd.concat(frames, ignore_index = True)
```

You can concatenate data frame and series together.

```
pd.concat([df1, s1], axis=1)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Merge Datasets with `join()`

DataFrame.join() combines the columns of multiple, potentially differently-indexed DataFrame into a single result DataFrame . `Inner, Outer, on="key"`

```
result = df1.join(df2)
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

Rebuild Missing Data

Count of null
df.isnull().sum

```
Out[77]: year                 0
         month                0
         day                  0
         dep_time          4129
         sched_dep_time       0
         dep_delay         4129
         arr_time          4268
         sched_arr_time       0
         arr_delay         4549
         carrier              1
         flight               1
         tailnum           1262
         origin               1
         dest                 1
         air_time          4550
         distance             1
         hour                 1
         minute               1
         time_hour            1
         dtype: int64
```

Percentage of null
df.isnull().sum()/len(df)*100

```
Out[78]: year              0.000000
         month             0.000000
         day               0.000000
         dep_time          2.535260
         sched_dep_time    0.000000
         dep_delay         2.535260
         arr_time          2.620608
         sched_arr_time    0.000000
         arr_delay         2.793145
         carrier           0.000614
         flight            0.000614
         tailnum           0.774884
         origin            0.000614
         dest              0.000614
         air_time          2.793759
         distance          0.000614
         hour              0.000614
         minute            0.000614
         time_hour         0.000614
         dtype: float64
```

Missing data

Most datasets contain missing values, which can be present for a number of reasons.

However, regardless of the reason, it is important to be careful when performing analyses on datasets with missing values, and oftentimes we may want to remove observations with such properties.

Python represents missing values as `NAN` (standing for “not available”).

The presence of `NAN` values in your dataset could lead to unexpected results.

Handling Missing Data

Replace NULL Values:

- ***To replace all NaN values with a scalar***
```
df.fillna(value=10)
```

- ***To replace NaN values with the values in the previous row.***
```
df.fillna(axis=0, method='ffill’)
```

- ***To replace NaN values with the values in the previous column.***
```
df.fillna(axis=1, method='ffill’)
```

- ***Replace with the values in the next row***
```
df.fillna(axis=0, method='bfill’)
```

- ***Replace with the values in the next column***
```
df.fillna(axis=1, method='bfill’)
```

- ***Replace NaN values with the mean***
```
df['Age'].fillna(value=df['Age'].mean(), inplace=True)
```

Handling Missing Data

Drop NULL Values:

- ***To drop columns if any NaN values are present***

```
df.dropna(axis = 1)
```

- ***To drop rows if any NaN values are present***

```
df.dropna(axis = 0)
```

- ***To drop columns in which more than 10% of values are missing***

```
df.dropna(thresh=len(df)*0.9, axis=1)
```

Standardization and Normalization

Standardization centers data around a mean of zero and a standard deviation of one, while normalization scales data to a set range, often [0, 1], by using the minimum and maximum values.

One-hot encoding

One-Hot Encoding

- `pd.get_dummies(df.carrier, prefix='Carrier')`

In [106]:

```
df.head()
pd.get_dummies(df.carrier, prefix='Carrier').head()
```

Out[106]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	NaN	515	2.0	1970-01-01	819	11.0	UA	1545.0	N14228	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00
1	2013	1	1	NaN	529	4.0	1970-01-01	830	20.0	UA	1714.0	N24211	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00
2	2013	1	1	NaN	540	2.0	1970-01-01	850	33.0	AA	1141.0	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00
3	2013	1	1	NaN	545	-1.0	1970-01-01	1022	-18.0	B6	725.0	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00
4	2013	1	1	NaN	600	-6.0	1970-01-01	837	-25.0	DL	461.0	N668DN	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00

Out[106]:

	Carrier_9E	Carrier_AA	Carrier_AS	Carrier_B6	Carrier_DL	Carrier_EV	Carrier_F9	Carrier_FL	Carrier_HA	Carrier_MQ	Carrier_OO	Carrier_UA	Carrier_US	Carrier_VX	Carrier_WN	Carrier_YV
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

One-hot encoding

Attach dummies to the data frame

```
carrier_dummies = pd.get_dummies(df.carrier,prefix='Carrier')
df = pd.concat([df, carrier_dummies], axis=1)
```

In [112]:

```
carrier_dummies = pd.get_dummies(df.carrier, prefix='Carrier')
df = pd.concat([df, carrier_dummies], axis=1)
df.head()
```

Out[112]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	Carrier_F9	Carrier_FL	Carrier_HA	Carrier_MQ	Carrier_OO	Carrier_UA	Carrier_US	Carrier_VX	Carrier_WN	Carrier_YV
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	...	0	0	0	0	0	1	0	0	0	0
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	...	0	0	0	0	0	1	0	0	0	0
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	...	0	0	0	0	0	0	0	0	0	0
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	...	0	0	0	0	0	0	0	0	0	0
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	...	0	0	0	0	0	0	0	0	0	0

5 rows × 35 columns

One-hot encoding

Attach dummies to the data frame

```
carrier_dummies = pd.get_dummies(df.carrier,prefix='Carrier')
df = pd.concat([df, carrier_dummies], axis=1)
```

In [112]:

```
carrier_dummies = pd.get_dummies(df.carrier, prefix='Carrier')
df = pd.concat([df, carrier_dummies], axis=1)
df.head()
```

Out[112]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	Carrier_F9	Carrier_FL	Carrier_HA	Carrier_MQ	Carrier_OO	Carrier_UA	Carrier_US	Carrier_VX	Carrier_WN	Carrier_YV
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	...	0	0	0	0	0	1	0	0	0	0
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	...	0	0	0	0	0	1	0	0	0	0
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	...	0	0	0	0	0	0	0	0	0	0
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	...	0	0	0	0	0	0	0	0	0	0
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	...	0	0	0	0	0	0	0	0	0	0

5 rows × 35 columns

Pandas `transform()`

- Pandas is an amazing library that contains extensive built-in functions for manipulating data. Among them, transform() is super useful when you are looking to manipulate rows or columns.

```
def plus_10(x):
    return x+10

df.transform(plus_10)


df.transform(lambda x: x+10)

df.apply(lambda x: x+10)
```

For a Single column

```
df[‘dep_time'] = df[‘dep_time'].apply(plus_10)
```

The **`flights`** data frame

In addition, we will employ the **`flights`** dataset, to create a **`pandas`** data frame to introduce data transformation.

This dataset is available in Canvas, under Datasets in Modules.

The **`flights`** data frame contains information for 19 features (columns) on all 336,776 flights (observations in rows) that departed from New York City in 2013, and was compiled by the US Bureau of Transportation and Statistics.

A snapshot of the `flights` data frame

In [3]:

```python
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

import pandas as pd

```

In [2]:

```python
flights = pd.read_csv('flights.csv')
flights.head()
```

Out[2]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545.0	N14228	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714.0	N24211	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141.0	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725.0	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461.0	N668DN	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00

Learning about the `flights` data frame

`year, months, day`	Date of departure
`dep_time, arr_time`	Actual departure and arrival times (format HHMM or HMM), local tz
`sched_dep_time, sched_arr_time`	Scheduled departure and arrival times (format HHMM or HMM), local tz
`dep_delay, arr_delay`	Departure and arrival delays, in minutes. Negative times represent early departures/arrivals.
`carrier`	Two letter carrier abbreviation.
`flight`	Flight number
`tailnum`	Plane tail number
`origin, dest`	Origin and destination
`air_time`	Amount of time spent in air (minutes)
`distance`	Distance between airports (miles)
`hour, minute`	Time of schedule departure
`time_hour`	Scheduled flight date & hour (**`POSIXct`**)

Aggregate with pivot_table( )

df.pivot_table( )

```
df.pivot_table(index=['month'], aggfunc='mean')
```

month	air_time	arr_delay	arr_time	day	dep_delay	dep_time	distance	flight	hour	minute	sched_arr_time	sched_dep_time	year
1	154.187401	6.129972	1523.154526	15.991261	10.036665	1347.209531	1006.843616	1958.625426	13.157125	25.197045	1547.597874	1340.909532	2013
2	151.346364	5.613019	1522.206593	14.743617	10.816843	1347.574462	1000.982285	1956.268406	13.172458	25.232456	1547.108853	1342.478257	2013
3	148.990776	6.913722	1502.096992	14.815675	13.997943	1351.137639	1009.878677	2006.679391	13.204847	25.361788	1539.285543	1345.844755	2013
10	148.886086	-0.167063	1519.898785	15.975008	6.243988	1340.106376	1038.875904	2008.254595	13.099173	26.420991	1539.308595	1336.338260	2013
11	155.468614	0.461347	1522.722191	15.290047	5.435362	1344.431552	1050.305046	1961.823676	13.158537	25.743656	1544.705846	1341.597404	2013
12	162.591414	14.870355	1505.251662	15.724436	16.576688	1357.193877	1064.655554	1926.505634	13.187773	26.269451	1543.297174	1345.046774	2013

```
df.pivot_table(index=['month'], values=['arr_delay', 'distance'], aggfunc='mean')
```

month	arr_delay	distance
1	6.129972	1006.843616
2	5.613019	1000.982285
3	6.913722	1009.878677
10	-0.167063	1038.875904
11	0.461347	1050.305046
12	14.870355	1064.655554

Obtaining only flights from January 1st

```
df[(df['month']==1) & (df['day']==1)]
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545.0	N14228	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714.0	N24211	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141.0	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725.0	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461.0	N668DN	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
837	2013	1	1	2356.0	2359	-3.0	425.0	437	-12.0	B6	727.0	N588JB	JFK	BQN	186.0	1576.0	23.0	59.0	1/1/2013 23:00
838	2013	1	1	NaN	1630	NaN	NaN	1815	NaN	EV	4308.0	N18120	EWR	RDU	NaN	416.0	16.0	30.0	1/1/2013 16:00
839	2013	1	1	NaN	1935	NaN	NaN	2240	NaN	AA	791.0	N3EHAA	LGA	DFW	NaN	1389.0	19.0	35.0	1/1/2013 19:00
840	2013	1	1	NaN	1500	NaN	NaN	1825	NaN	AA	1925.0	N3EVAA	LGA	MIA	NaN	1096.0	15.0	0.0	1/1/2013 15:00
841	2013	1	1	NaN	600	NaN	NaN	901	NaN	B6	125.0	N618JB	JFK	FLL	NaN	1069.0	6.0	0.0	1/1/2013 6:00

842 rows × 19 columns

Obtaining on the first of the month except January

```
df[(df['month']!=1) & (df['day']==1)]
```

- **Or other conditions based on your target or requirements**

```
df[df[‘tail_num’]== ‘N538UV’]
```

```
df[df[‘arr_delay’]>50]
```

```
df[df[‘carrier’]== ‘AA']
```

Obtaining only flights from Sep to Dec.

```
df[(df['month']>=9) & (df['month']<=12)]

df[df['month'].between(9, 12)]
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
27004	2013	10	1	447.0	500	-13.0	614.0	648	-34.0	US	1877.0	N538UW	EWR	CLT	69.0	529.0	5.0	0.0	10/1/2013 5:00
27005	2013	10	1	522.0	517	5.0	735.0	757	-22.0	UA	252.0	N556UA	EWR	IAH	174.0	1400.0	5.0	17.0	10/1/2013 5:00
27006	2013	10	1	536.0	545	-9.0	809.0	855	-46.0	AA	2243.0	N630AA	JFK	MIA	132.0	1089.0	5.0	45.0	10/1/2013 5:00
27007	2013	10	1	539.0	545	-6.0	801.0	827	-26.0	UA	1714.0	N37252	LGA	IAH	172.0	1416.0	5.0	45.0	10/1/2013 5:00
27008	2013	10	1	539.0	545	-6.0	917.0	933	-16.0	B6	1403.0	N789JB	JFK	SJU	186.0	1598.0	5.0	45.0	10/1/2013 5:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
111291	2013	12	31	NaN	705	NaN	NaN	931	NaN	UA	1729.0	NaN	EWR	DEN	NaN	1605.0	7.0	5.0	12/31/2013 7:00
111292	2013	12	31	NaN	825	NaN	NaN	1029	NaN	US	1831.0	NaN	JFK	CLT	NaN	541.0	8.0	25.0	12/31/2013 8:00
111293	2013	12	31	NaN	1615	NaN	NaN	1800	NaN	MQ	3301.0	N844MQ	LGA	RDU	NaN	431.0	16.0	15.0	12/31/2013 16:00
111294	2013	12	31	NaN	600	NaN	NaN	735	NaN	UA	219.0	NaN	EWR	ORD	NaN	719.0	6.0	0.0	12/31/2013 6:00
111295	2013	12	31	NaN	830	NaN	NaN	1154	NaN	UA	443.0	NaN	JFK	LAX	NaN	2475.0	8.0	30.0	12/31/2013 8:00

84292 rows × 19 columns

Flights in Sep or Dec

```
df[(df['month’]==9) | (df['month']==12)]
```

Creating a new data frame from subset of flights

In the previous example, we obtained all flights for January 1st.

However, we may wish to actually store this new dataset, so that we can further manipulate it downstream.

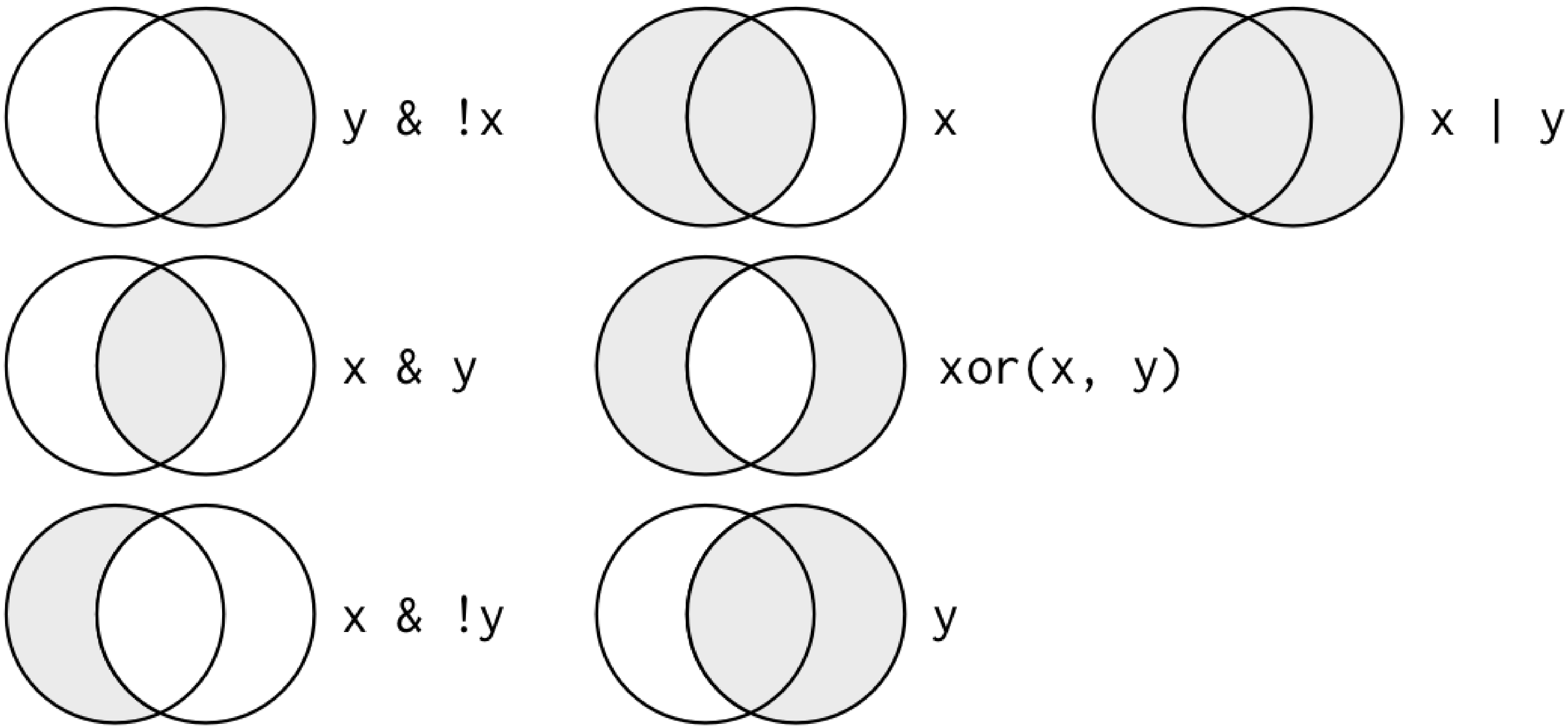
To save the data frame, we can simply assign it to a new variable, called for example **`Jan1`**, as

```
Jan1 =  df['month']==1) & (df['day']==1)]
```

The set of logical operators

Suppose that **x** and **y** are two expressions (or sets).

Logical operators to consider are **&** (and), **|** (or), **^** (exclusive or), and **!** (not)



Select subset of variables

In [130]:

```
df_numeric = df.select_dtypes(include=['float64', 'int64'])
df_categories = df.select_dtypes(include='object')
df_numeric
df_categories
```

Out[130]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	flight	air_time	distance	hour	minute
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	1545.0	227.0	1400.0	5.0	15.0
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	1714.0	227.0	1416.0	5.0	29.0
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	1141.0	160.0	1089.0	5.0	40.0
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	725.0	183.0	1576.0	5.0	45.0
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	461.0	116.0	762.0	6.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
162858	2013	3	29	1251.0	1300	-9.0	1448.0	1505	-17.0	3867.0	94.0	569.0	13.0	0.0
162859	2013	3	29	1251.0	1300	-9.0	1358.0	1407	-9.0	2175.0	45.0	214.0	13.0	0.0
162860	2013	3	29	1251.0	1300	-9.0	1438.0	1450	-12.0	4426.0	74.0	479.0	13.0	0.0
162861	2013	3	29	1254.0	1259	-5.0	1511.0	1534	-23.0	781.0	106.0	762.0	12.0	59.0
162862	2013	3	29	1254.0	1300	-6.0	1352.0	1406	-140.0	NaN	NaN	NaN	NaN	NaN

162863 rows × 14 columns

Out[130]:

	carrier	tailnum	origin	dest	time_hour
0	UA	N14228	EWR	IAH	1/1/2013 5:00
1	UA	N24211	LGA	IAH	1/1/2013 5:00
2	AA	N619AA	JFK	MIA	1/1/2013 5:00
3	B6	N804JB	JFK	BQN	1/1/2013 5:00
4	DL	N668DN	LGA	ATL	1/1/2013 6:00
...	...	...	...	...	...
162858	9E	N8968E	EWR	CVG	3/29/2013 13:00
162859	US	N703UW	LGA	DCA	3/29/2013 13:00
162860	MQ	N720MQ	LGA	CMH	3/29/2013 13:00
162861	DL	N678DL	LGA	ATL	3/29/2013 12:00
162862	NaN	NaN	NaN	NaN	NaN

162863 rows × 5 columns

Select a subset of variables

```
df[['month', 'day', 'year', 'carrier']]
```

	month	day	year	carrier
0	1	1	2013	UA
1	1	1	2013	UA
2	1	1	2013	AA
3	1	1	2013	B6
4	1	1	2013	DL
...	...	...	...	...
162858	3	29	2013	9E
162859	3	29	2013	US
162860	3	29	2013	MQ
162861	3	29	2013	DL
162862	3	29	2013	NaN

162863 rows × 4 columns

Select a subset of variables using `filter()`

```
df.filter(['month', 'day', 'year', 'carrier'])
```

	month	day	year	carrier
0	1	1	2013	UA
1	1	1	2013	UA
2	1	1	2013	AA
3	1	1	2013	B6
4	1	1	2013	DL
...	...	...	...	...
162858	3	29	2013	9E
162859	3	29	2013	US
162860	3	29	2013	MQ
162861	3	29	2013	DL
162862	3	29	2013	NaN

162863 rows × 4 columns

Select a subset of variables using `iloc[]`

```
df.iloc[:, 0:4]
```

	year	month	day	dep_time
0	2013	1	1	517.0
1	2013	1	1	533.0
2	2013	1	1	542.0
3	2013	1	1	544.0
4	2013	1	1	554.0
...	...	...	...	...
162858	2013	3	29	1251.0
162859	2013	3	29	1251.0
162860	2013	3	29	1251.0
162861	2013	3	29	1254.0
162862	2013	3	29	1254.0

162863 rows × 4 columns

Select a subset of variables using `iloc[]`

```
df.iloc[0:10, 0:4]
```

	year	month	day	dep_time
0	2013	1	1	517.0
1	2013	1	1	533.0
2	2013	1	1	542.0
3	2013	1	1	544.0
4	2013	1	1	554.0
5	2013	1	1	554.0
6	2013	1	1	555.0
7	2013	1	1	557.0
8	2013	1	1	557.0
9	2013	1	1	558.0

Select a subset of variables using `iloc[]`

```
df.iloc[0:10, 9:-1]
```

	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute
0	UA	1545.0	N14228	EWR	IAH	227.0	1400.0	5.0	15.0
1	UA	1714.0	N24211	LGA	IAH	227.0	1416.0	5.0	29.0
2	AA	1141.0	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0
3	B6	725.0	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0
4	DL	461.0	N668DN	LGA	ATL	116.0	762.0	6.0	0.0
5	UA	1696.0	N39463	EWR	ORD	150.0	719.0	5.0	58.0
6	B6	507.0	N516JB	EWR	FLL	158.0	1065.0	6.0	0.0
7	EV	5708.0	N829AS	LGA	IAD	53.0	229.0	6.0	0.0
8	B6	79.0	N593JB	JFK	MCO	140.0	944.0	6.0	0.0
9	AA	301.0	N3ALAA	LGA	ORD	138.0	733.0	6.0	0.0

All columns except one

```
df.iloc[0:10, df.columns != 'month']
```

	year	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	517.0	515	2.0	830.0	819	11.0	UA	1545.0	N14228	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00
1	2013	1	533.0	529	4.0	850.0	830	20.0	UA	1714.0	N24211	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00
2	2013	1	542.0	540	2.0	923.0	850	33.0	AA	1141.0	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00
3	2013	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725.0	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00
4	2013	1	554.0	600	-6.0	812.0	837	-25.0	DL	461.0	N668DN	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00
5	2013	1	554.0	558	-4.0	740.0	728	12.0	UA	1696.0	N39463	EWR	ORD	150.0	719.0	5.0	58.0	1/1/2013 5:00
6	2013	1	555.0	600	-5.0	913.0	854	19.0	B6	507.0	N516JB	EWR	FLL	158.0	1065.0	6.0	0.0	1/1/2013 6:00
7	2013	1	557.0	600	-3.0	709.0	723	-14.0	EV	5708.0	N829AS	LGA	IAD	53.0	229.0	6.0	0.0	1/1/2013 6:00
8	2013	1	557.0	600	-3.0	838.0	846	-8.0	B6	79.0	N593JB	JFK	MCO	140.0	944.0	6.0	0.0	1/1/2013 6:00
9	2013	1	558.0	600	-2.0	753.0	745	8.0	AA	301.0	N3ALAA	LGA	ORD	138.0	733.0	6.0	0.0	1/1/2013 6:00

Arranging rows with `pd.sort_values()`

The `sort_values()` function allows one to alter the order of rows.

Function usage is similar to `filter()`, except that instead of choosing rows to retain, it simply changes their order.

Default is Ascending

```
df.sort_values(by='carrier’)
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
18180	2013	1	21	2103.0	2045	18.0	2257.0	2216	41.0	9E	3395.0	N932XJ	JFK	DCA	51.0	213.0	20.0	45.0	1/21/2013 20:00
97606	2013	12	16	1646.0	1700	-14.0	1843.0	1913	-30.0	9E	3335.0	N8877A	EWR	CVG	101.0	569.0	17.0	0.0	12/16/2013 17:00
14165	2013	1	17	812.0	815	-3.0	1007.0	958	9.0	9E	3521.0	N908XJ	JFK	ORD	140.0	740.0	8.0	15.0	1/17/2013 8:00
69517	2013	11	15	1557.0	1600	-3.0	1729.0	1751	-22.0	9E	3357.0	N904XJ	LGA	BNA	125.0	764.0	16.0	0.0	11/15/2013 16:00
136499	2013	3	1	926.0	930	-4.0	1044.0	1103	-19.0	9E	3913.0	N8891A	JFK	ROC	50.0	264.0	9.0	30.0	3/1/2013 9:00

Descending multi-level sort

```
df.sort_values(by=['carrier’, dest], ascending = False)
```

Adding new variables

New Empty Variable:

```
df['new_column'] = np.nan
```

	air_time	distance	hour	minute	time_hour	new_column
	227.0	1400.0	5.0	15.0	1/1/2013 5:00	NaN
	227.0	1416.0	5.0	29.0	1/1/2013 5:00	NaN
	160.0	1089.0	5.0	40.0	1/1/2013 5:00	NaN
	183.0	1576.0	5.0	45.0	1/1/2013 5:00	NaN
	116.0	762.0	6.0	0.0	1/1/2013 6:00	NaN
	...	...	...	...	...	...
	94.0	569.0	13.0	0.0	3/29/2013 13:00	NaN
	45.0	214.0	13.0	0.0	3/29/2013 13:00	NaN
	74.0	479.0	13.0	0.0	3/29/2013 13:00	NaN
	106.0	762.0	12.0	59.0	3/29/2013 12:00	NaN
	NaN	NaN	NaN	NaN	NaN	NaN

Sometimes the features available in a dataset are not the features of interest.

For example, a dataset may contain a feature for distance and a feature for time, but the relevant feature may instead be speed, which is distance divided by time.

Adding new variables

Add derived variable:

```
df['total_delay_time'] =  df.dep_delay + df.arr_delay
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	tailnum	origin	dest	air_time	distance	hour	minute	time_hour	new_column	total_delay_time
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	...	N14228	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00	NaN	13.0
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	...	N24211	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00	NaN	24.0
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	...	N619AA	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00	NaN	35.0
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	...	N804JB	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00	NaN	-19.0
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	...	N668DN	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00	NaN	-31.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
162858	2013	3	29	1251.0	1300	-9.0	1448.0	1505	-17.0	9E	...	N8968E	EWR	CVG	94.0	569.0	13.0	0.0	3/29/2013 13:00	NaN	-26.0
162859	2013	3	29	1251.0	1300	-9.0	1358.0	1407	-9.0	US	...	N703UW	LGA	DCA	45.0	214.0	13.0	0.0	3/29/2013 13:00	NaN	-18.0
162860	2013	3	29	1251.0	1300	-9.0	1438.0	1450	-12.0	MQ	...	N720MQ	LGA	CMH	74.0	479.0	13.0	0.0	3/29/2013 13:00	NaN	-21.0
162861	2013	3	29	1254.0	1259	-5.0	1511.0	1534	-23.0	DL	...	N678DL	LGA	ATL	106.0	762.0	12.0	59.0	3/29/2013 12:00	NaN	-28.0
162862	2013	3	29	1254.0	1300	-6.0	1352.0	1406	-140.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-146.0

162863 rows × 21 columns

Summarizing dataf rame with `groupby()`

`groupby()` collapses a data frame into groups row, by summarizing the dataset based on particular features and operation.

```
df.groupby(['carrier']).mean()
```

carrier	year	month	day	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	flight	air_time	distance	hour	minute	new_column	total_delay_time	speed
9E	2013.0	6.667551	15.295310	1448.823536	13.941268	1627.400045	1644.496180	5.631379	3492.207237	89.395815	529.225594	14.217954	27.028120	NaN	19.366159	0.415614
AA	2013.0	6.517031	15.431279	1289.384625	6.999228	1531.417729	1551.227980	0.284746	913.239753	195.042715	1347.759554	12.628723	26.512372	NaN	7.283391	0.216402
AS	2013.0	6.370262	15.623907	1294.810496	6.492669	1594.507331	1607.842566	-6.093842	12.469388	331.222874	2402.000000	12.740525	20.758017	NaN	0.398827	0.120541
B6	2013.0	6.595915	15.552636	1391.289990	10.366709	1416.964260	1489.573287	7.602378	659.467825	155.263119	1071.253974	13.619571	29.332890	NaN	17.945406	0.002282
DL	2013.0	6.710799	15.517999	1347.801381	6.210496	1594.234956	1609.353381	-1.459821	1397.029267	178.028882	1233.931608	13.257197	22.081636	NaN	4.714612	0.243843
EV	2013.0	6.711074	15.372918	1361.961668	20.331161	1506.759695	1540.627312	17.550134	4592.914819	92.205249	553.003760	13.342414	27.720244	NaN	37.833631	0.273580
F9	2013.0	6.732353	15.458824	1319.250000	15.044379	1579.467456	1568.647059	19.340237	706.155882	236.044379	1620.000000	12.891176	30.132353	NaN	34.384615	0.148495
FL	2013.0	5.699617	15.417092	1358.154337	12.612272	1583.864794	1575.796556	13.550327	530.500000	106.250327	676.830357	13.354592	22.695153	NaN	26.027451	0.308049
HA	2013.0	6.141975	15.654321	945.925926	11.654321	1506.938272	1525.339506	-5.808642	51.000000	633.271605	4983.000000	9.407407	5.185185	NaN	5.845679	0.115994
MQ	2013.0	6.485538	15.419896	1392.473454	7.067210	1567.293863	1571.564045	7.163167	3893.668590	94.381240	570.256646	13.660014	26.472051	NaN	14.137838	0.331228
OO	2013.0	9.333333	19.166667	1468.500000	11.833333	1694.500000	1657.500000	17.000000	5262.500000	139.000000	832.333333	14.333333	35.166667	NaN	28.833333	0.303773
UA	2013.0	6.695488	15.407200	1300.847304	9.812574	1521.624759	1543.631898	2.800622	931.843081	213.273527	1494.713858	12.748874	25.959917	NaN	12.563151	0.177033
US	2013.0	6.652810	15.367874	1247.072344	1.881328	1412.462841	1419.979114	0.189042	1630.216224	91.796529	556.184542	12.316113	15.461003	NaN	2.081660	0.486999
VX	2013.0	7.544371	15.496247	1257.899779	5.760393	1557.341834	1601.111258	-4.735070	224.989404	346.310283	2497.875055	12.320971	25.802649	NaN	1.035025	0.118126
WN	2013.0	6.778933	15.504045	1265.880799	14.377639	1461.984623	1462.063728	7.897952	1491.674096	152.604502	984.691927	12.394420	26.438831	NaN	22.227619	0.191671
YV	2013.0	7.232727	15.589091	1529.240000	14.612648	1693.773810	1691.730909	8.904762	3467.687273	61.019841	316.767273	14.992727	29.967273	NaN	23.289683	0.509817

Summarizing dataf rame with groupby()

```
df.groupby(['carrier'])['arr_delay', 'dep_delay'].mean()
```

	arr_delay	dep_delay
carrier		
9E	5.631379	13.941268
AA	0.284746	6.999228
AS	-6.093842	6.492669
B6	7.602378	10.366709
DL	-1.459821	6.210496
EV	17.550134	20.331161
F9	19.340237	15.044379
FL	13.550327	12.612272
HA	-5.808642	11.654321
MQ	7.163167	7.067210
OO	17.000000	11.833333
UA	2.800622	9.812574
US	0.189042	1.881328
VX	-4.735070	5.760393
WN	7.897952	14.377639
YV	8.904762	14.612648

What if missing values relate to events of interest?

In the flights dataset, missing values represent cancelled flights.

That is, for cancelled flights, features like **`dep_delay`** and **`arr_delay`** will be missing, because the flight never departed or arrived, and therefore would not have a valid value for either of these features.

Therefore, we could create a new dataset based only on flights that are not cancelled (by filtering flights with **`NAN`** delay), and then learn about the mean departure delay without needing to worry about filtering **`NAN`** values.

The following slide conveys this idea.

Summarizing set of cancelled flights

```
df[ (df['dep_delay'].isna()) & (df['arr_delay'].isna())]
```

Cancelled Flights

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	origin	dest	air_time	distance	hour	minute	time_hour	new_column	total_delay_time	speed
838	2013	1	1	NaT	1630	NaN	NaN	1815	NaN	EV	...	EWR	RDU	NaN	416.0	16.0	30.0	1/1/2013 16:00	NaN	NaN	NaN
839	2013	1	1	NaT	1935	NaN	NaN	2240	NaN	AA	...	LGA	DFW	NaN	1389.0	19.0	35.0	1/1/2013 19:00	NaN	NaN	NaN
840	2013	1	1	NaT	1500	NaN	NaN	1825	NaN	AA	...	LGA	MIA	NaN	1096.0	15.0	0.0	1/1/2013 15:00	NaN	NaN	NaN
841	2013	1	1	NaT	600	NaN	NaN	901	NaN	B6	...	JFK	FLL	NaN	1069.0	6.0	0.0	1/1/2013 6:00	NaN	NaN	NaN
1777	2013	1	2	NaT	1540	NaN	NaN	1747	NaN	EV	...	EWR	CVG	NaN	569.0	15.0	40.0	1/2/2013 15:00	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
161458	2013	3	27	NaT	905	NaN	NaN	1115	NaN	MQ	...	LGA	DTW	NaN	502.0	9.0	5.0	3/27/2013 9:00	NaN	NaN	NaN
162437	2013	3	28	NaT	1350	NaN	NaN	1557	NaN	EV	...	EWR	CVG	NaN	569.0	13.0	50.0	3/28/2013 13:00	NaN	NaN	NaN
162438	2013	3	28	NaT	1000	NaN	NaN	1121	NaN	US	...	LGA	DCA	NaN	214.0	10.0	0.0	3/28/2013 10:00	NaN	NaN	NaN
162439	2013	3	28	NaT	1400	NaN	NaN	1509	NaN	US	...	LGA	DCA	NaN	214.0	14.0	0.0	3/28/2013 14:00	NaN	NaN	NaN
162440	2013	3	28	NaT	1700	NaN	NaN	1817	NaN	US	...	LGA	DCA	NaN	214.0	17.0	0.0	3/28/2013 17:00	NaN	NaN	NaN

4129 rows × 22 columns

Summarizing set of non-cancelled flights

```
not_cancelled = df[-(df['dep_delay'].isna()) & (-df['arr_delay'].isna())]
```

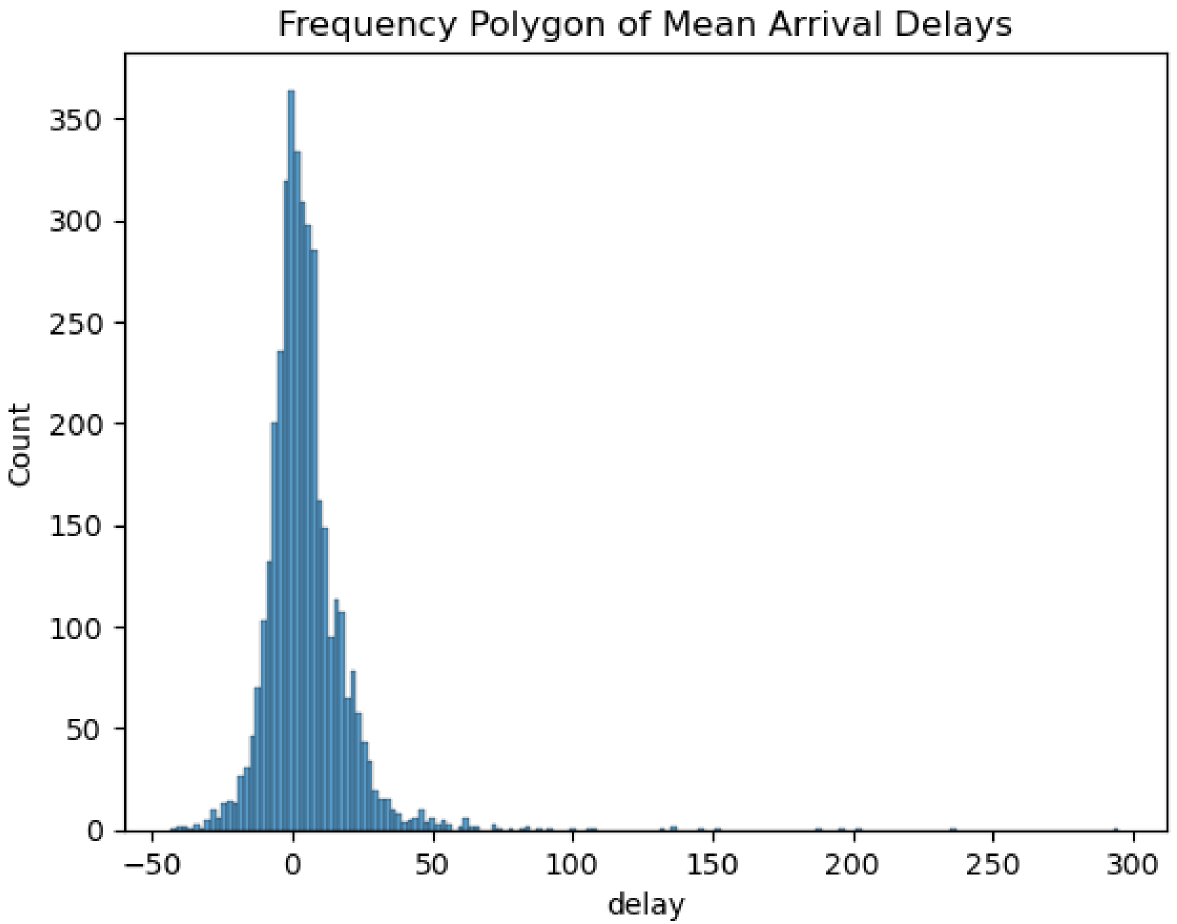
Not Cancelled Flights

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	origin	dest	air_time	distance	hour	minute	time_hour	new_column	total_delay_time	speed
0	2013	1	1	00:00:00	515	2.0	830.0	819	11.0	UA	...	EWR	IAH	227.0	1400.0	5.0	15.0	1/1/2013 5:00	NaN	13.0	0.223571
1	2013	1	1	00:00:00	529	4.0	850.0	830	20.0	UA	...	LGA	IAH	227.0	1416.0	5.0	29.0	1/1/2013 5:00	NaN	24.0	0.223870
2	2013	1	1	00:00:00	540	2.0	923.0	850	33.0	AA	...	JFK	MIA	160.0	1089.0	5.0	40.0	1/1/2013 5:00	NaN	35.0	0.349862
3	2013	1	1	00:00:00	545	-1.0	1004.0	1022	-18.0	B6	...	JFK	BQN	183.0	1576.0	5.0	45.0	1/1/2013 5:00	NaN	-19.0	0.291878
4	2013	1	1	00:00:00	600	-6.0	812.0	837	-25.0	DL	...	LGA	ATL	116.0	762.0	6.0	0.0	1/1/2013 6:00	NaN	-31.0	0.338583
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
162858	2013	3	29	00:00:00.000001	1300	-9.0	1448.0	1505	-17.0	9E	...	EWR	CVG	94.0	569.0	13.0	0.0	3/29/2013 13:00	NaN	-26.0	0.346221
162859	2013	3	29	00:00:00.000001	1300	-9.0	1358.0	1407	-9.0	US	...	LGA	DCA	45.0	214.0	13.0	0.0	3/29/2013 13:00	NaN	-18.0	0.500000
162860	2013	3	29	00:00:00.000001	1300	-9.0	1438.0	1450	-12.0	MQ	...	LGA	CMH	74.0	479.0	13.0	0.0	3/29/2013 13:00	NaN	-21.0	0.390397
162861	2013	3	29	00:00:00.000001	1259	-5.0	1511.0	1534	-23.0	DL	...	LGA	ATL	106.0	762.0	12.0	59.0	3/29/2013 12:00	NaN	-28.0	0.337270
162862	2013	3	29	00:00:00.000001	1300	-6.0	1352.0	1406	-140.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-146.0	NaN

158314 rows × 22 columns

Some flights have a mean delay of five hours

```
delays = not_cancelled.groupby('tailnum')['arr_delay'].mean().reset_index(name='delay')
sns.histplot(data=delays, x='delay', binwidth=2)
plt.title('Frequency Polygon of Mean Arrival Delays')
```



Story is a little more nuanced here

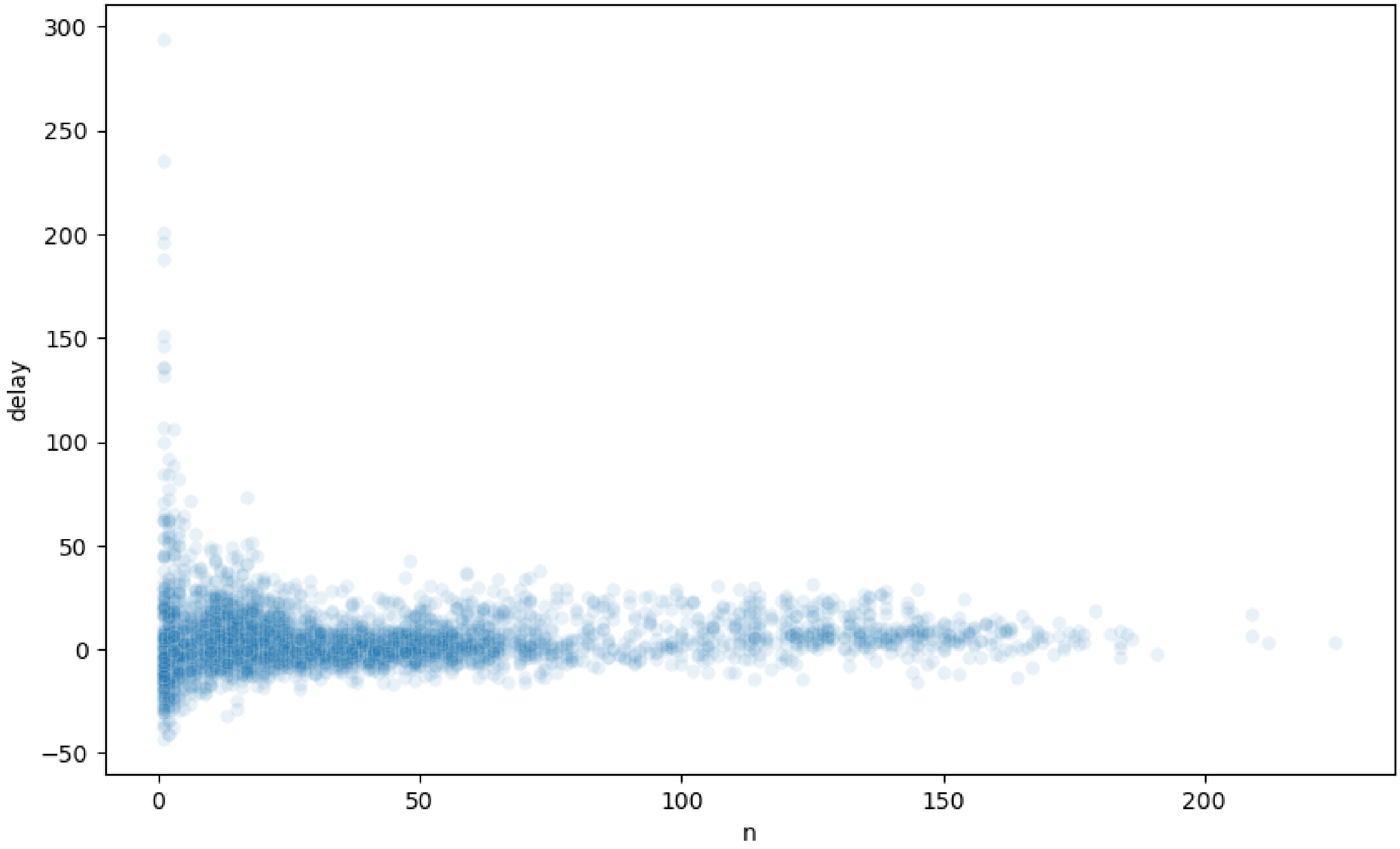
This example showed that there were some flights with mean delay time of over five hours, but that this delay amount was relatively rare (few flights).

However, this last graph provide us with only a summary of the data.

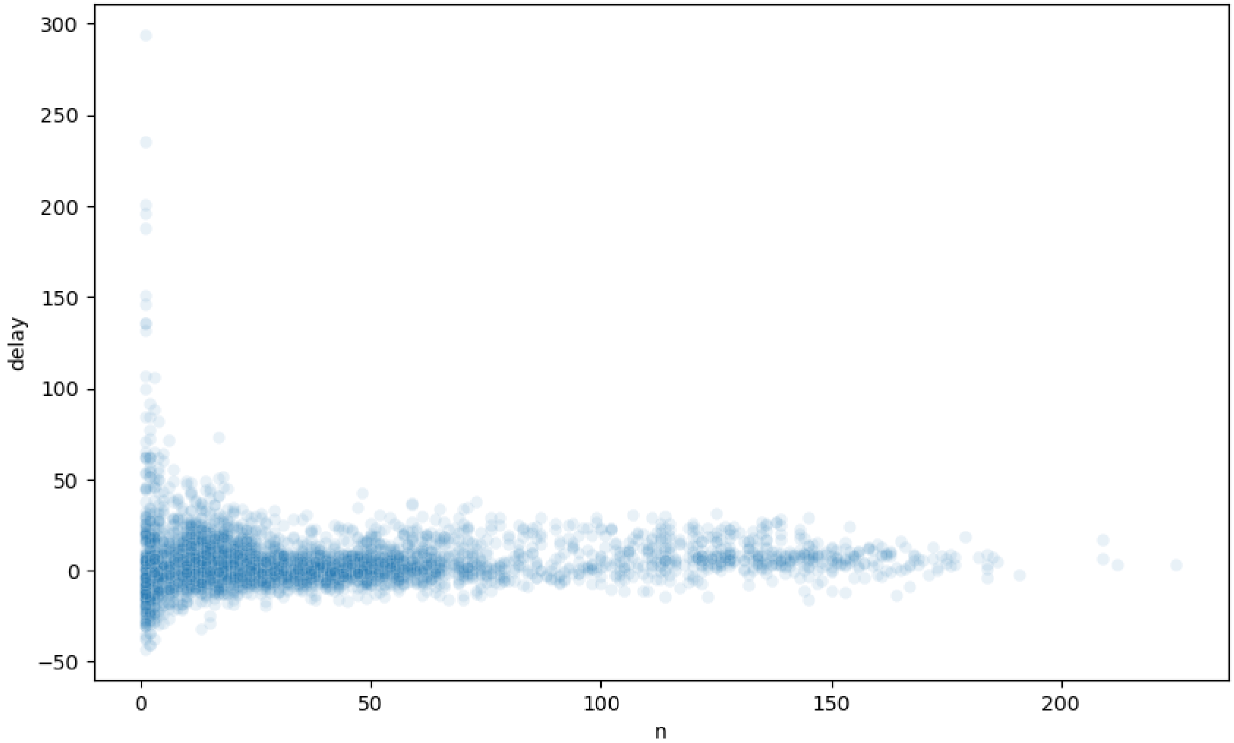
We can potentially learn more if we understood the variation in mean delay time as a function of the number of flights with that delay.

Greater variation in mean delay with fewer flights

```
not_cancelled = flights[-flights['dep_delay'].isna()]
delays =
not_cancelled.groupby('tailnum').agg(delay=('arr_delay',
                        lambda x: x.mean(skipna=True)),
                        n=('arr_delay', 'size')).reset_index()
```



Characteristic trend of less variation with increasing n.



The above graph displays a plot that is commonly observed, whenever the mean is plotted as a function of group size n.

Here, variation in the summary (mean) decreases as group size n increases.

That is, variation in mean is decreasing with increased sample size.

Removing noisy points (small group or sample sizes)

```
filtered_delays = delays[delays['n'] > 25]
sns.scatterplot(data=filtered_delays, x='n', y='delay', alpha=0.1)
```