Lecture 4
The Split-Apply-Combine Paradigm

Flight Delays

Which airline carriers are most likely to be delayed?

Flight Delays

Which airline carriers are most likely to be delayed?

Let's look at a data set of all domestic flights that departed from one of the New York City airports (JFK, LaGuardia, and Newark) on November 16, 2013.

```
import pandas as pd
data_dir = "https://datasci112.stanford.edu/data/"
df = pd.read_csv(data_dir + "flights_nyc_20131116.csv")
df
```

Flight Delays

Which airline carriers are most likely to be delayed?

Let's look at a data set of all domestic flights that departed from one of the New York City airports (JFK, LaGuardia, and Newark) on November 16, 2013.

```
import pandas as pd
data_dir = "https://datasci112.stanford.edu/data/"
df = pd.read_csv(data_dir + "flights_nyc_20131116.csv")
df
```

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

Visualizing Delays

We already know how to visualize and summarize a quantitative variable like **dep_delay**.

Visualizing Delays

We already know how to visualize and summarize a quantitative variable like **dep_delay**.

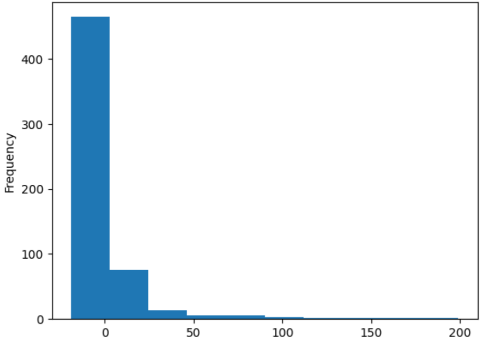
```
df["dep_delay"].plot.hist()
df["dep_delay"].mean()
```

Visualizing Delays

We already know how to visualize and summarize a quantitative variable like **dep_delay**.

```
df["dep_delay"].plot.hist()
df["dep_delay"].mean()
```

2.0469565217391303

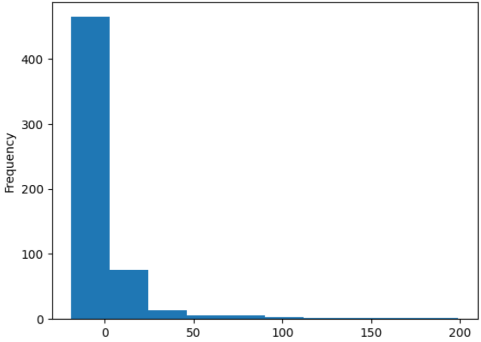


Visualizing Delays

We already know how to visualize and summarize a quantitative variable like **dep_delay**.

```
df["dep_delay"].plot.hist()
df["dep_delay"].mean()
```

2.0469565217391303



But how do we compare delays across different carriers?

1. Boolean Masking
2. The Split-Apply-Combine Paradigm
3. Visualizing Conditional Distributions

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0      False
1      True
2      False
3      True
4      False
       ...
573    False
574    False
575     True
576    False
577    False
Name: carrier, Length: 578, dtype: bool
```

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0       False
1       True
2       False
3       True
4       False
        ...
573     False
574     False
575      True
576     False
577     False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0       False
1       True
2       False
3       True
4       False
        ...
573     False
574     False
575      True
576     False
577     False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

indicates whether each flight was United or not

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0        False
1        True
2        False
3        True
4        False
         ...
573      False
574      False
575       True
576      False
577      False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

indicates whether each flight was United or not

another example of vectorization!

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0        False
1        True
2        False
3        True
4        False
         ...
573      False
574      False
575       True
576      False
577      False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

indicates whether each flight was United or not

another example of vectorization!

What about the following?

```
(df["carrier"] == "UA").sum()
```

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0      False
1      True
2      False
3      True
4      False
       ...
573    False
574    False
575     True
576    False
577    False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

indicates whether each flight was United or not

another example of vectorization!

What about the following?

```
(df["carrier"] == "UA").sum()
```

123

What do you think the following code will produce?

```
df["carrier"] == "UA"
```

```
0        False
1        True
2        False
3        True
4        False
         ...
573      False
574      False
575       True
576      False
577      False
Name: carrier, Length: 578, dtype: bool
```

a Series of booleans

indicates whether each flight was United or not

another example of vectorization!

What about the following?

```
(df["carrier"] == "UA").sum()
```

```
123
```

the number of United flights that day

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

0.21280276816608998

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

0.21280276816608998

the proportion of flights that day that were United

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

0.21280276816608998

the proportion of flights that day that were United

What You Need to Know about Booleans

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

0.21280276816608998

the proportion of flights that day that were United

What You Need to Know about Booleans

- Applying a relational operator like `==`, `<`, `>`, and `!=` on a `Series` produces a `Series` of booleans, by vectorization.

Boolean Series

How would you interpret the following?

```
(df["carrier"] == "UA").mean()
```

0.21280276816608998

the proportion of flights that day that were United

What You Need to Know about Booleans

- Applying a relational operator like `==`, `<`, `>`, and `!=` on a `Series` produces a `Series` of booleans, by vectorization.
- Arithmetic operations can be performed on booleans in `Series`, treating `True` as 1 and `False` as 0.

Boolean Masks

A boolean `Series` can be passed as a key to a `DataFrame` to *mask* the data.

```
df[df["carrier"] == "UA"]
```

Boolean Masks

A boolean `Series` can be passed as a key to a `DataFrame` to *mask* the data.

```
df[df["carrier"] == "UA"]
```

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

A Boolean mask acts as a “filter” to select data.

Boolean Masks

A boolean `Series` can be passed as a key to a `DataFrame` to *mask* the data.

```
df[df["carrier"] == "UA"]
```

The index has changed!

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

Exercise

How would we summarize the United Airlines delays?

Exercise

How would we summarize the United Airlines delays?

```
df[df["carrier"] == "UA"]["dep_delay"].mean()
```

Exercise

How would we summarize the United Airlines delays?

```
df[df["carrier"] == "UA"]["dep_delay"].mean()
```

5.590163934426229

Exercise

How would we summarize the United Airlines delays?

- 5.590163934426229

```
df[df["carrier"] == "UA"]["dep_delay"].mean()
```

- Note that this is a summary of a conditional distribution of
- **dep_delay**:

  - $\text{mean}(\textbf{dep\_delay} | \textbf{carrier} = \text{UA})$.

1. Boolean Masking
2. The Split-Apply-Combine Paradigm
3. Visualizing Conditional Distributions

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

.

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\mathrm{mean}(\mathbf{dep\_delay}|\mathbf{carrier}).$$

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\text{mean}(\textbf{dep\_delay}|\textbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  df[df["carrier"] == carrier]["dep_delay"].mean()
```

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of $\mathbf{dep\_delay}$ given $\mathbf{carrier}$:

$$\text{mean}(\mathbf{dep\_delay} | \mathbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  print(carrier, df[df["carrier"] == carrier]["dep_delay"].mean())
```

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\text{mean}(\textbf{dep\_delay} | \textbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  print(carrier, df[df["carrier"] == carrier]["dep_delay"].mean())
```

```
US -2.324324324324324
UA 5.590163934426229
AA -1.337837837837838
DL 3.295238095238095
B6 1.5378787878787878
EV 1.2476190476190476
```

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\text{mean}(\textbf{dep\_delay} | \textbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  print(carrier, df[df["carrier"] == carrier]["dep_delay"].mean())
```

```
US -2.324324324324324
UA 5.590163934426229
AA -1.3378378378378378
DL 3.295238095238095
B6 1.5378787878787878
EV 1.2476190476190476
```

Problems with this Solution

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\text{mean}(\textbf{dep\_delay}|\textbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  print(carrier, df[df["carrier"] == carrier]["dep_delay"].mean())
```

```
US -2.324324324324324
UA 5.590163934426229
AA -1.3378378378378378
DL 3.295238095238095
B6 1.5378787878787878
EV 1.2476190476190476
```

Problems with this Solution

- It is inconvenient (have to write a `for` loop over the possible values).

The code becomes long and repetitive, especially if there are many carriers.
You have to write a for-loop for each one every time → not practical.

Another Exercise

To compare carriers, we need to summarize all the conditional distributions of **dep_delay** given **carrier**:

$$\text{mean}(\textbf{dep\_delay}|\textbf{carrier}).$$

```
for carrier in df["carrier"].unique():
  print(carrier, df[df["carrier"] == carrier]["dep_delay"].mean())
```

```
US -2.324324324324324
UA 5.590163934426229
AA -1.3378378378378378
DL 3.295238095238095
B6 1.5378787878787878
EV 1.2476190476190476
```

Problems with this Solution

- It is inconvenient (have to write a `for` loop over the possible values).
- The values are not stored in a Pandas object for further analysis (e.g., visualization).

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** [(Wickham, 2011).]

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

split

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

:

	carrier	flight	origin	dest	dep_delay
5	DL	731	LGA	DTW	-7.0
16	DL	1377	LGA	ATL	-1.0
25	DL	575	EWR	ATL	-10.0
34	DL	479	JFK	ATL	-4.0
46	DL	315	JFK	SJU	-4.0
...	...	...	...	...	...
533	DL	2454	JFK	DEN	-3.0
535	DL	2065	JFK	FLL	-5.0
539	DL	347	JFK	SJU	38.0
544	DL	427	JFK	LAX	14.0
567	DL	985	JFK	BOS	106.0

105 rows × 5 columns

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

split

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

:

	carrier	flight	origin	dest	dep_delay
5	DL	731	LGA	DTW	-7.0
16	DL	1377	LGA	ATL	-1.0
25	DL	575	EWR	ATL	-10.0
34	DL	479	JFK	ATL	-4.0
46	DL	315	JFK	SJU	-4.0
...	...	...	...	...	...
533	DL	2454	JFK	DEN	-3.0
535	DL	2065	JFK	FLL	-5.0
539	DL	347	JFK	SJU	38.0
544	DL	427	JFK	LAX	14.0
567	DL	985	JFK	BOS	106.0

105 rows × 5 columns

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

split

apply

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

.mean()

:

	carrier	flight	origin	dest	dep_delay
5	DL	731	LGA	DTW	-7.0
16	DL	1377	LGA	ATL	-1.0
25	DL	575	EWR	ATL	-10.0
34	DL	479	JFK	ATL	-4.0
46	DL	315	JFK	SJU	-4.0
...	...	...	...	...	...
533	DL	2454	JFK	DEN	-3.0
535	DL	2065	JFK	FLL	-5.0
539	DL	347	JFK	SJU	38.0
544	DL	427	JFK	LAX	14.0
567	DL	985	JFK	BOS	106.0

105 rows × 5 columns

.mean()

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

split

apply

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

.mean()

5.590164

:

	carrier	flight	origin	dest	dep_delay
5	DL	731	LGA	DTW	-7.0
16	DL	1377	LGA	ATL	-1.0
25	DL	575	EWR	ATL	-10.0
34	DL	479	JFK	ATL	-4.0
46	DL	315	JFK	SJU	-4.0
...	...	...	...	...	...
533	DL	2454	JFK	DEN	-3.0
535	DL	2065	JFK	FLL	-5.0
539	DL	347	JFK	SJU	38.0
544	DL	427	JFK	LAX	14.0
567	DL	985	JFK	BOS	106.0

105 rows × 5 columns

.mean()

3.295238

The Split-Apply-Combine Paradigm

This analysis fits into the **split-apply-combine paradigm** (Wickham, 2011).

split | apply | combine

key

	carrier	flight	origin	dest	dep_delay
0	US	1895	EWR	CLT	-5.0
1	UA	1014	LGA	IAH	-3.0
2	AA	2243	JFK	MIA	2.0
3	UA	303	JFK	SFO	-8.0
4	US	795	LGA	PHL	-8.0
...	...	...	...	...	...
573	B6	745	JFK	PSE	-3.0
574	B6	839	JFK	BQN	0.0
575	UA	360	EWR	PBI	NaN
576	US	1946	EWR	CLT	NaN
577	US	2142	LGA	BOS	NaN

578 rows × 5 columns

	carrier	flight	origin	dest	dep_delay
1	UA	1014	LGA	IAH	-3.0
3	UA	303	JFK	SFO	-8.0
8	UA	1187	LGA	ORD	-5.0
9	UA	258	EWR	MCO	-2.0
15	UA	665	EWR	SFO	-1.0
...	...	...	...	...	...
537	UA	1631	EWR	IAH	-3.0
549	UA	1409	EWR	TPA	-1.0
552	UA	1071	EWR	BQN	5.0
562	UA	1066	EWR	BOS	-5.0
575	UA	360	EWR	PBI	NaN

123 rows × 5 columns

.mean() → 5.590164

:

	carrier	flight	origin	dest	dep_delay
5	DL	731	LGA	DTW	-7.0
16	DL	1377	LGA	ATL	-1.0
25	DL	575	EWR	ATL	-10.0
34	DL	479	JFK	ATL	-4.0
46	DL	315	JFK	SJU	-4.0
...	...	...	...	...	...
533	DL	2454	JFK	DEN	-3.0
535	DL	2065	JFK	FLL	-5.0
539	DL	347	JFK	SJU	38.0
544	DL	427	JFK	LAX	14.0
567	DL	985	JFK	BOS	106.0

105 rows × 5 columns

.mean() → 3.295238

```
carrier
AA    -1.337838
B6     1.537879
DL     3.295238
EV     1.247619
UA     5.590164
US    -2.324324
Name: dep_delay, dtype: float6
```

Split-Apply-Combine in Pandas

Split-Apply-Combine in Pandas

The split-apply-combine paradigm is implemented in Pandas as the `.groupby()` method.

Split-Apply-Combine in Pandas

The split-apply-combine paradigm is implemented in Pandas as the `.groupby()` method.

```
df.groupby("carrier")["dep_delay"].mean()
```

groupby("carrier") → groups the DataFrame by each unique carrier.

["dep_delay"] → selects only the dep_delay column for each group.

The result is a GroupBy object, not actual calculations yet.

.mean()
average departure delay for each carrier.

Split-Apply-Combine in Pandas

The split-apply-combine paradigm is implemented in Pandas as the `.groupby()` method.

```
df.groupby("carrier")["dep_delay"].mean()
```

```
carrier
AA    -1.337838
B6     1.537879
DL     3.295238
EV     1.247619
UA     5.590164
US    -2.324324
Name: dep_delay, dtype: float64
```

The values are in a Series for further analysis!

A Python Series is a one-dimensional labeled array capable of holding data of any type (integers, strings, floats, etc.). Each element has an index label, which allows easy access and alignment.
Key points:
One-dimensional (like a single column in a table).
Has values and index.Can be created from lists, dictionaries, or NumPy arrays.

Split-Apply-Combine in Pandas

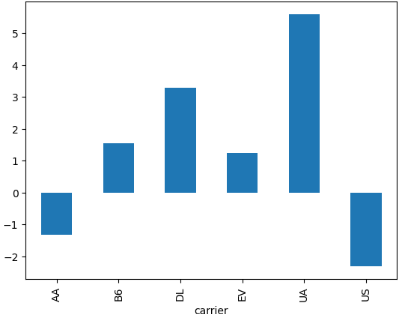
For example, we could plot the mean delay for each carrier.

```
df.groupby("carrier")["dep_delay"].mean().plot.bar()
```

Split-Apply-Combine in Pandas

For example, we could plot the mean delay for each carrier.

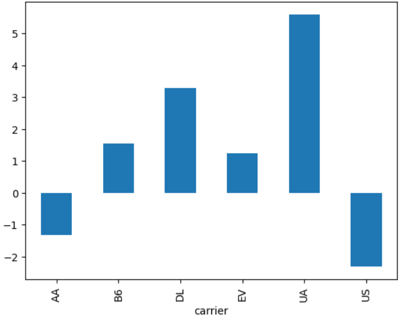
```
df.groupby("carrier")["dep_delay"].mean().plot.bar()
```



Split-Apply-Combine in Pandas

For example, we could plot the mean delay for each carrier.

```
df.groupby("carrier")["dep_delay"].mean().plot.bar()
```



Notice that United Airlines had the longest average delay.

Splitting on Multiple Keys

What if we wanted to also split by the origin airport?

Splitting on Multiple Keys

What if we wanted to also split by the origin airport?

```
df.groupby(["carrier", "origin"])["dep_delay"].mean()
```

Splitting on Multiple Keys

What if we wanted to also split by the origin airport?

```
df.groupby(["carrier", "origin"])["dep_delay"].mean()
```

```
carrier  origin
AA       EWR       -3.375000
         JFK        1.771429
         LGA       -4.322581
B6       EWR       -0.823529
         JFK       -0.836735
         LGA       17.588235
DL       EWR       19.222222
         JFK        4.980000
         LGA       -1.652174
EV       EWR        1.483146
         JFK        0.000000
         LGA       -0.083333
UA       EWR        7.525773
         JFK        1.909091
         LGA       -4.928571
US       EWR       -5.000000
         JFK        5.400000
         LGA       -5.312500
Name: dep_delay, dtype: float64
```

Splitting on Multiple Keys

What if we wanted to also split by the origin airport?

```
df.groupby(["carrier", "origin"])["dep_delay"].mean()
```

```
carrier  origin
AA       EWR       -3.375000
         JFK        1.771429
         LGA       -4.322581
B6       EWR       -0.823529
         JFK       -0.836735
         LGA       17.588235
DL       EWR       19.222222
         JFK        4.980000
         LGA       -1.652174
EV       EWR        1.483146
         JFK        0.000000
         LGA       -0.083333
UA       EWR        7.525773
         JFK        1.909091
         LGA       -4.928571
US       EWR       -5.000000
         JFK        5.400000
         LGA       -5.312500
Name: dep_delay, dtype: float64
```

⟶

```
.unstack("origin")
```

origin	EWR	JFK	LGA
carrier			
AA	-3.375000	1.771429	-4.322581
B6	-0.823529	-0.836735	17.588235
DL	19.222222	4.980000	-1.652174
EV	1.483146	0.000000	-0.083333
UA	7.525773	1.909091	-4.928571
US	-5.000000	5.400000	-5.312500

Crosstab

Splitting on Multiple Keys

What if we wanted to also split by the origin airport?

```
df.groupby(["carrier", "origin"])["dep_delay"].mean()
```

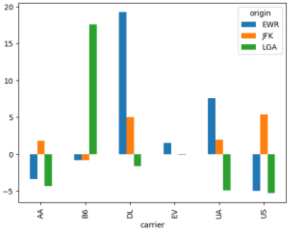
```
carrier  origin
AA       EWR        -3.375000
         JFK         1.771429
         LGA        -4.322581
B6       EWR        -0.823529
         JFK        -0.836735
         LGA        17.588235
DL       EWR        19.222222
         JFK         4.980000
         LGA        -1.652174
EV       EWR         1.483146
         JFK         0.000000
         LGA        -0.083333
UA       EWR         7.525773
         JFK         1.909091
         LGA        -4.928571
US       EWR        -5.000000
         JFK         5.400000
         LGA        -5.312500
Name: dep_delay, dtype: float64
```

→

```
.unstack("origin")
```

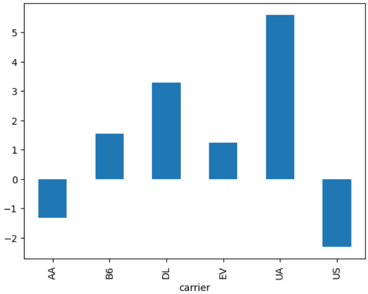
origin	EWR	JFK	LGA
carrier			
AA	-3.375000	1.771429	-4.322581
B6	-0.823529	-0.836735	17.588235
DL	19.222222	4.980000	-1.652174
EV	1.483146	0.000000	-0.083333
UA	7.525773	1.909091	-4.928571
US	-5.000000	5.400000	-5.312500

```
.plot.bar()
```

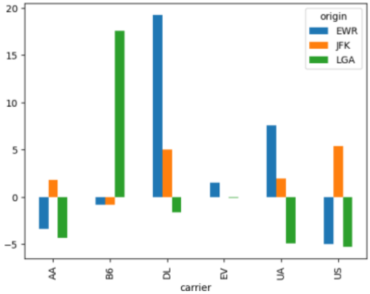


ird

```
ax = (df.
      groupby("carrier")
      ["dep_delay"].mean().
      plot.bar())
```



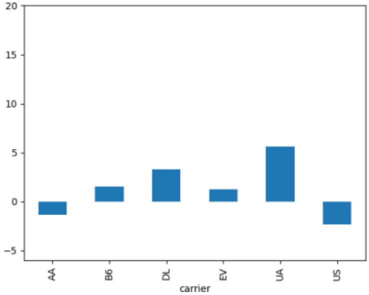

```
ax = (df.
      groupby(["carrier", "origin"])
      ["dep_delay"].mean().
      unstack("origin").
      plot.bar())
```

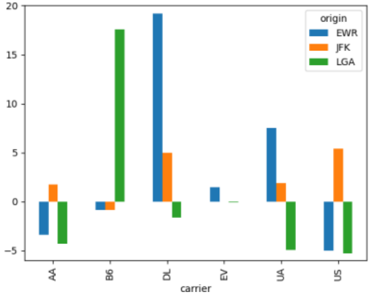


Notice Anything Weird?

```
ax = (df.
      groupby("carrier")
      ["dep_delay"].mean().
      plot.bar())
ax.set_ylim(-6, 20)
```

```
ax = (df.
      groupby(["carrier", "origin"])
      ["dep_delay"].mean().
      unstack("origin").
      plot.bar())
ax.set_ylim(-6, 20)
```

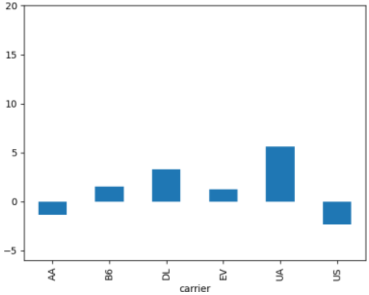


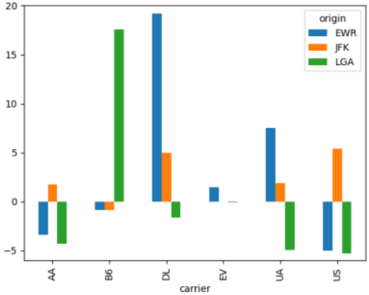


Summary / Weird Points
Negative delays → some flights departed earlier than scheduled.
Extremely high delays → certain carrier–origin combinations have averages far above the norm.
Distribution imbalance → some carriers show values that are very different compared to others.

```
ax = (df.
      groupby("carrier")
      ["dep_delay"].mean().
      plot.bar())
ax.set_ylim(-6, 20)
```

```
ax = (df.
      groupby(["carrier", "origin"])
      ["dep_delay"].mean().
      unstack("origin").
      plot.bar())
ax.set_ylim(-6, 20)
```





Let's investigate in a Colab!

1. Boolean Masking

2. The Split-Apply-Combine Paradigm

3. Visualizing Conditional Distributions

Comparing Distributions

It is possible to use `.groupby()` with all kinds of operations.

Comparing Distributions

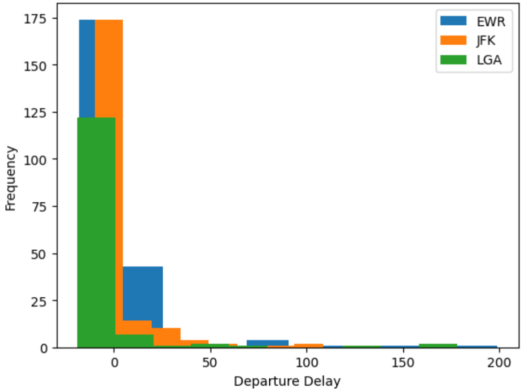
It is possible to use `.groupby()` with all kinds of operations.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True)
axes[0].set_xlabel("Departure Delay")
```

Comparing Distributions

It is possible to use `.groupby()` with all kinds of operations.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True)
axes[0].set_xlabel("Departure Delay")
```



Comparing Distributions

To prevent *overplotting*, we set the opacity parameter `alpha`.

Comparing Distributions

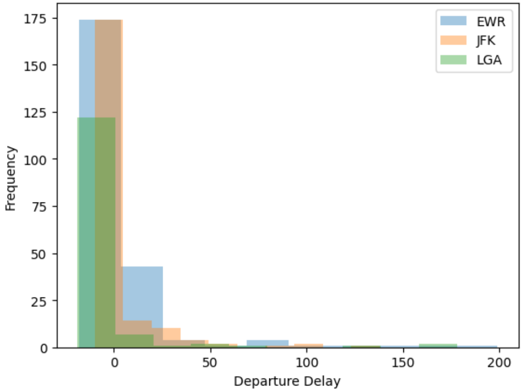
To prevent *overplotting*, we set the opacity parameter `alpha`.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True,
                                                    alpha=0.4)
axes[0].set_xlabel("Departure Delay")
```

Comparing Distributions

To prevent *overplotting*, we set the opacity parameter `alpha`.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True,
                                                   alpha=0.4)
axes[0].set_xlabel("Departure Delay")
```



Comparing Distributions

Density histograms visualize the conditional distribution $\mathbf{dep\_delay} \mid \mathbf{carrier}$ directly, allowing for easy comparison.

Comparing Distributions

Density histograms visualize the conditional distribution **dep_delay | carrier** directly, allowing for easy comparison.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True,
                                                   alpha=0.4,
                                                   density=True)
axes[0].set_xlabel("Departure Delay")
```

Comparing Distributions

Density histograms visualize the conditional distribution **dep_delay | carrier** directly, allowing for easy comparison.

```
axes = df.groupby("origin")["dep_delay"].plot.hist(legend=True,
                                                    alpha=0.4,
                                                    density=True)
axes[0].set_xlabel("Departure Delay")
```