

Chapter 8

How to analyze the data

Objectives

Applied

1. Melt the data in two or more columns.
2. Group and aggregate the data in a DataFrame.
3. Pivot the data in a DataFrame or create a pivot table from the data.
4. Bin the data for a column.
5. Select the rows with the n largest values from a DataFrame, calculate the percent change for the rows in a column of a DataFrame, or rank the rows in a DataFrame by largest or smallest values.

Objectives (continued)

Knowledge

1. Distinguish between the `pivot()` and the `pivot_table()` methods.
2. Explain why binning the data in a column is useful.
3. Describe the way you can find other methods for specific analysis tasks.

The melt() method

Method	Description
<code>melt(params)</code>	Melts the data in two or more columns into two columns.

Parameters of the melt() method

Parameter	Description
<code>id_vars</code>	The column or columns that won't be melted.
<code>value_vars</code>	The columns to melt. If none are specified, all will be melted.
<code>var_name</code>	The name of the column that will contain the melted column names, or “variable” by default.
<code>value_name</code>	The name of the column that will contain the melted column values, or “value” by default.

The cars DataFrame

`cars.head()`

	aspiration	carbody	enginesize	curbweight	price
0	std	convertible	130	2548	13495.0
1	std	convertible	130	2548	16500.0
2	std	hatchback	152	2823	16500.0
3	std	sedan	109	2337	13950.0
4	std	sedan	136	2824	17450.0

How to use the melt() method

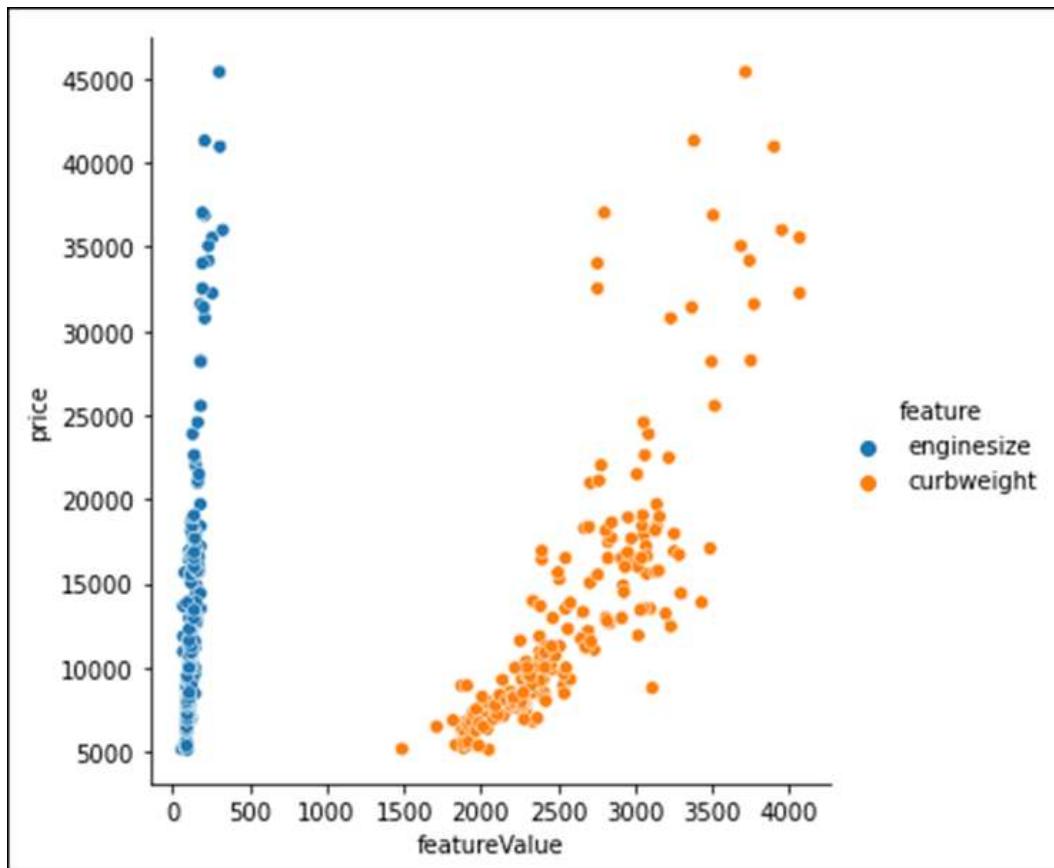
```
cars_melted = pd.melt(cars, id_vars='price',
                      value_vars=['enginesize','curbweight'],
                      var_name='feature',
                      value_name='featureValue')

cars_melted
```

	price	feature	featureValue
0	13495.0	enginesize	130
1	16500.0	enginesize	130
2	16500.0	enginesize	152
3	13950.0	enginesize	109
4	17450.0	enginesize	136
...
405	16845.0	curbweight	2952
406	19045.0	curbweight	3049
407	21485.0	curbweight	3012
408	22470.0	curbweight	3217
409	22625.0	curbweight	3062

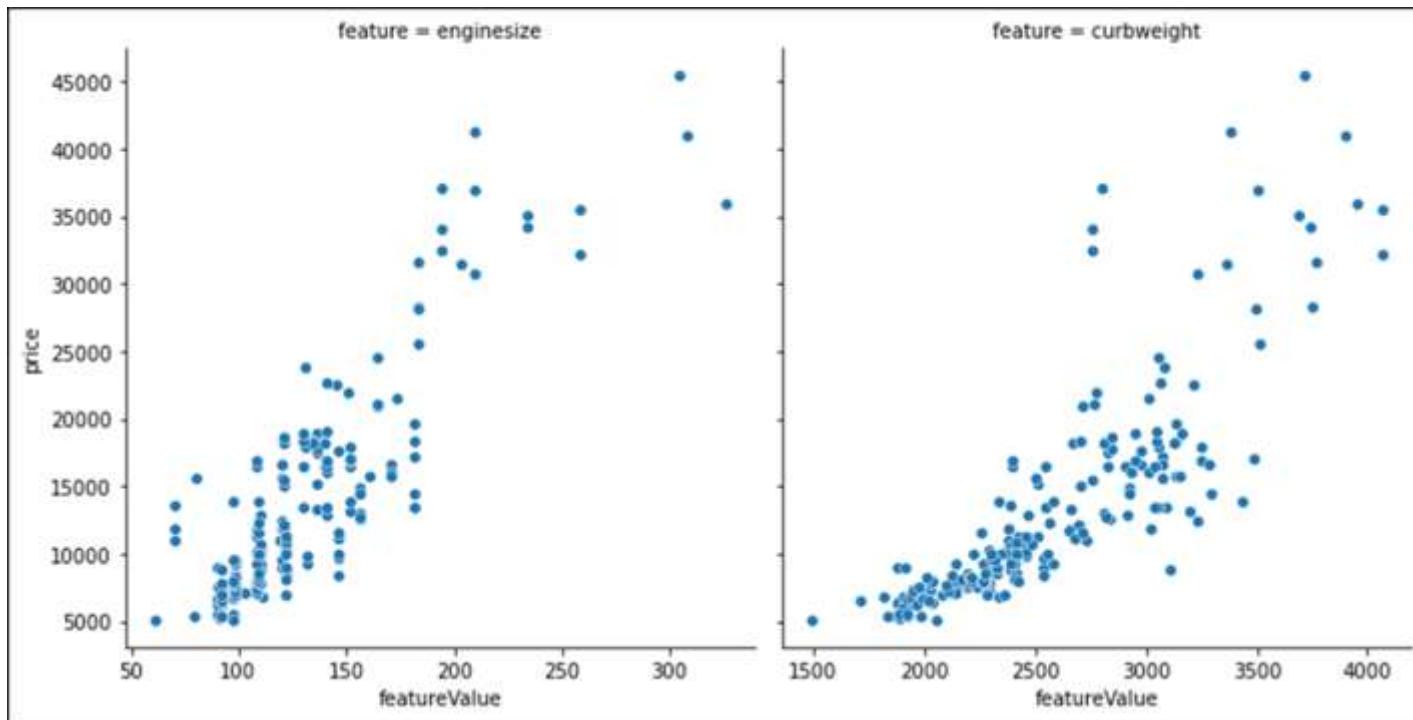
How to plot melted data with the hue parameter

```
sns.relplot(data=cars_melted, x='featureValue', y='price',  
            hue='feature')
```



How to plot melted data with the col parameter

```
sns.relplot(data=cars_melted, x='featureValue', y='price',  
            col='feature', facet_kws={'sharex':False})
```



Some of the aggregate methods that are optimized for grouping

- `sum()`
- `mean()`
- `median()`
- `count()`
- `std()`
- `min()`
- `max()`

The fires DataFrame before the data is grouped

```
fires.head(3)
```

	fire_name	acres_burned	state	fire_year	discovery_date	fire_month	days_burning
0	Power	16823.0	CA	2004	2004-10-06	10	15.0
1	Freds	7700.0	CA	2004	2004-10-13	10	4.0
2	Bachelor	10.0	NM	2004	2004-07-20	7	0.0

How to get the average for each numeric column in each state

```
fires.groupby('state').mean().head(3)
```

	acres_burned	fire_year	fire_month	days_burning
state				
AK	11367.199362	2004.742504	6.264198	32.081535
AL	42.348169	2003.885422	5.022529	0.272676
AR	50.281673	2005.850793	5.581081	0.400992

How to get the maximum value for each month in each state

```
fires.groupby(['state','fire_year','fire_month']) \  
    .max().dropna().head(3)
```

			acres_burned	discovery_date	days_burning
state	fire_year	fire_month			
AK	1992	5	1410.0	1992-05-31	50.0
		6	48087.0	1992-06-29	82.0
		7	35090.0	1992-07-30	77.0

The `groupby()` method

Method	Description
<code>groupby(params)</code>	Returns a GroupBy object that supports aggregate methods such as <code>sum()</code> .

Parameters of the `groupby()` method

Parameter	Description
<code>by</code>	The column or list of columns to group by.
<code>as_index</code>	If <code>False</code> , doesn't create an index based on the groupby columns. If <code>True</code> (the default), it does.

The fires DataFrame

`fires.head(3)`

	fire_name	acres_burned	state	fire_year	discovery_date	fire_month	days_burning
0	Power	16823.0	CA	2004	2004-10-06	10	15.0
1	Freds	7700.0	CA	2004	2004-10-13	10	4.0
2	Bachelor	10.0	NM	2004	2004-07-20	7	0.0

A GroupBy object with the fire_year column as the index

```
yearly_group = fires.groupby('fire_year')
yearly_sums = yearly_group.sum()
yearly_sums.head(3)
```

fire_year	acres_burned	fire_month	days_burning
1992	2123889.91	45643	6230.0
1993	2118394.10	52880	7283.0
1994	4033880.06	57669	20158.0

A GroupBy object without indexes

```
yearly_group = fires.groupby(  
    'fire_year', as_index=False).sum()  
yearly_sums = yearly_group.sum()  
yearly_sums.head(3)
```

	fire_year	acres_burned	fire_month	days_burning
0	1992	2123889.91	45643	6230.0
1	1993	2118394.10	52880	7283.0
2	1994	4033880.06	57669	20158.0

The agg() method

Method	Description
agg()	Applies an aggregate method or list of methods to a Series or DataFrame object.

The GroupBy object

```
monthly_group = fires.groupby(  
    ['state','fire_year','fire_month'])
```

How to apply aggregate methods to all numeric columns

```
monthly_group.agg(['sum','count','mean']).dropna().head(3)
```

state	fire_year	fire_month	acres_burned			days_burning		
			sum	count	mean	sum	count	mean
AK	1992	5	4202.0	15	280.133333	135.0	14	9.642857
		6	86401.0	26	3323.115385	417.0	25	16.680000
		7	48516.7	26	1866.026923	500.0	22	22.727273

How to apply aggregate methods to a single column

```
monthly_group.days_burning.agg(  
    ['sum', 'count', 'mean']).dropna().head(3)
```

			sum	count	mean
state	fire_year	fire_month			
AK	1992		5	135.0	14
			6	417.0	25
			7	500.0	22
					16.680000
					22.727273

How to apply varied aggregate methods to numeric columns

```
df = monthly_group.agg({'acres_burned': ['sum', 'max', 'min'],
                       'days_burning': ['sum', 'mean'],
                       'fire_name': 'count'}).dropna()
df.head(3)
```

state	fire_year	fire_month	acres_burned			days_burning		fire_name
			sum	max	min	sum	mean	count
AK	1992	5	4202.0	1410.0	10.0	135.0	9.642857	14
		6	86401.0	48087.0	10.0	417.0	16.680000	23
		7	48516.7	35090.0	10.0	500.0	22.727273	26

The `pivot()` method

Method	Description
<code>pivot(params)</code>	Pivots the data based on the index, columns, and values parameters.

Parameters of the `pivot()` method

Parameter	Description
<code>index</code>	The column or list of columns to use as the row index (no duplicates).
<code>columns</code>	The column or list of columns to use as the column index.
<code>values</code>	The column or list of columns to use to populate the new DataFrame. By default, all remaining columns are used.

The top_states DataFrame

```
states = ['AK', 'CA', 'ID', 'TX']
top_states = fires.groupby(['state', 'fire_year'],
                           as_index=False).sum()
top_states = top_states.query('state in @states')
top_states.head(2)
```

	state	fire_year	acres_burned	fire_month	days_burning
0	AK	1992	142444.7	454	1145.0
1	AK	1993	686630.5	961	3373.0

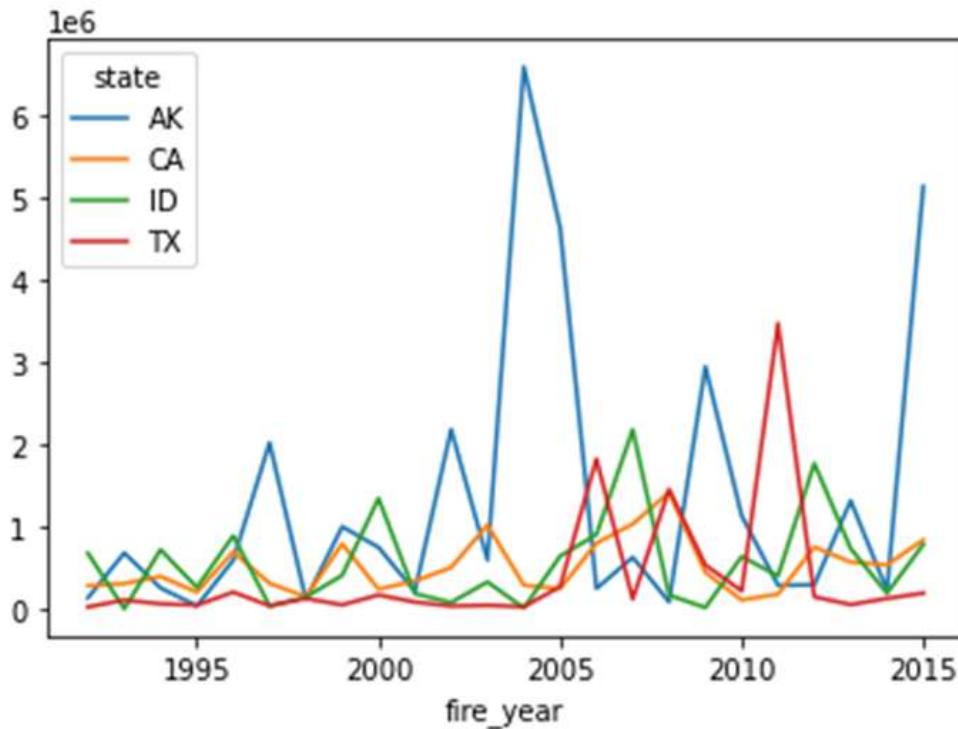
How to pivot the data

```
top_states.pivot(index='fire_year', columns='state',  
                  values='acres_burned').head(2)
```

state	AK	CA	ID	TX
fire_year				
1992	142444.7	289254.9	683495.2	31500.3
1993	686630.5	315011.1	7658.5	114265.5

How to plot the data with the Pandas plot() method

```
top_states.pivot(index='fire_year', columns='state',  
                  values='acres_burned').plot()
```



The `pivot_table()` method

Method	Description
<code>pivot_table(params)</code>	Produces a pivot table with an applied aggregate method.

Parameters of the `pivot_table()` method

Parameter	Description
<code>index</code>	The column or list of columns to use as the row index (allows duplicates).
<code>columns</code>	The column or list of columns to use as the column index.
<code>values</code>	The column or list of columns that contain the values to be aggregated. By default, all non-nuisance columns are aggregated.

Parameters of the `pivot_table()` method (cont.)

Parameter	Description
<code>aggfunc</code>	The aggregate method or list of methods to be applied to each column in the <code>values</code> parameter.
<code>fill_value</code>	The value to replace any missing values with in the resulting pivot table.

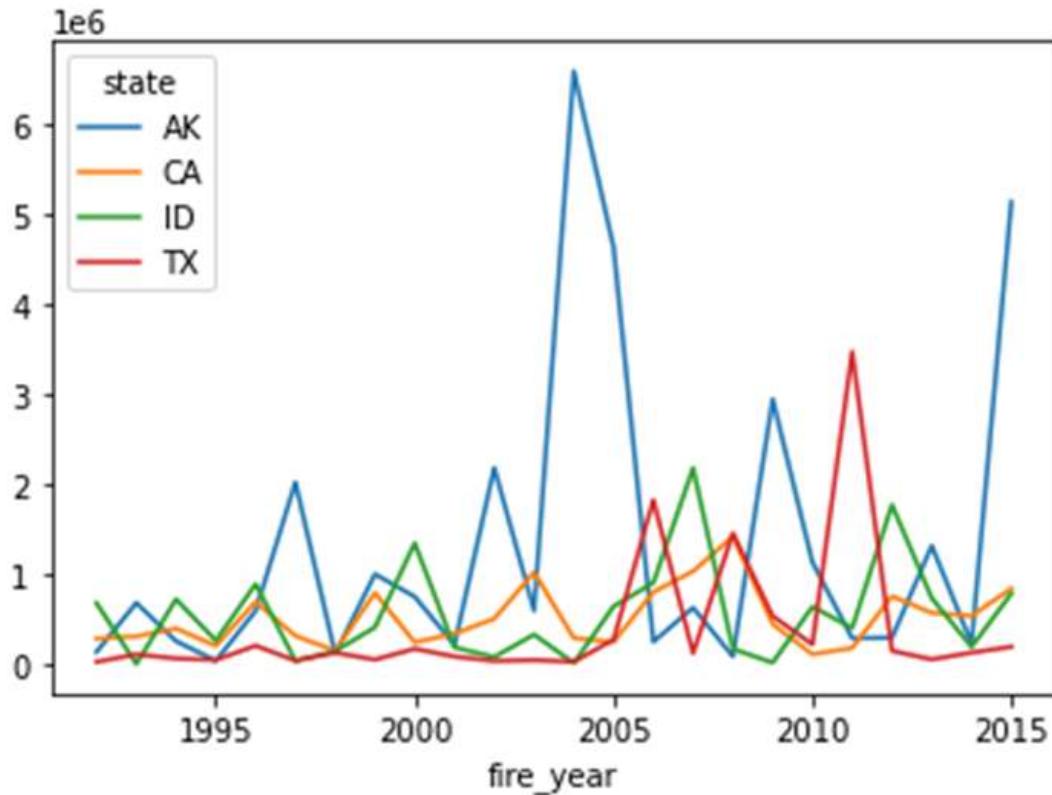
How to use the `pivot_table()` method to create a DataFrame

```
states = ['AK', 'CA', 'ID', 'TX']
fires_top_4 = fires.query('state in @states')
fires_top_4 = fires_top_4.pivot_table(index='fire_year',
                                       columns='state', values='acres_burned', aggfunc='sum')
fires_top_4.head(2)
```

state	AK	CA	ID	TX
fire_year				
1992	142444.7	289254.9	683495.2	31500.3
1993	686630.5	315011.1	7658.5	114265.5

How to plot the data with the Pandas plot() method

`fires_top_4.plot()`



The `cut()` method

Method	Description
<code>cut(params)</code>	Bins the data into equal-sized bins.

Parameters of the `cut()` method

Parameter	Description
<code>x</code>	The column that contains the data to be binned.
<code>bins</code>	The number of bins to create, or a list of values for the bin edges.
<code>labels</code>	The labels to use for the bins.
<code>right</code>	If set to False, the right edges are not included in the bins.

The fires_filtered DataFrame

```
fires_filtered = fires.query(  
    'fire_year == 2010 and days_burning > 0').dropna()
```

How to create four bins for the data

```
pd.cut(fires_filtered.acres_burned, bins=4)  
=====  
1145154    (-296.103, 76535.75]  
1145175    (-296.103, 76535.75]  
...  
1879725    (-296.103, 76535.75]  
1880370    (-296.103, 76535.75]  
Name: acres_burned, Length: 1858, dtype: category  
Categories (4, interval[float64]): [(-296.103, 76535.75] < (76535.75,  
153061.5] < (153061.5, 229587.25] < (229587.25, 306113.0]]
```

How to add labels to the bins

```
pd.cut(fires_filtered.acres_burned,  
       bins=[0,100000,200000,300000,400000],  
       labels=['small','medium','large','very large'])  
=====  
1145154    small  
1145175    small  
...  
1879725    small  
1880370    small  
Name: acres_burned, Length: 1858, dtype: category  
Categories (4, object): ['small' < 'medium' < 'large' < 'very large']
```

The distribution of values in the bins

```
pd.cut(fires_filtered.acres_burned,  
       bins=[0,100000,200000,300000,400000],  
       labels=['small','medium','large','very large']).value_counts()  
=====  
small          1855  
medium           2  
very large       1  
large            0  
Name: acres_burned, dtype: int64
```

The qcut() method

Method	Description
<code>qcut(params)</code>	Bins the data into quantiles with the same number of unique values in each bin. The number of rows in each bin will be skewed if there are duplicates.

Parameters of the qcut() method

Parameter	Description
<code>x</code>	The column that contains the data to be binned.
<code>q</code>	The number of quantiles to create.
<code>labels</code>	The labels to use for the bins.
<code>duplicates</code>	What to do with bins that have the same edges. The default is raise, which raises a ValueError. If set to drop, the non-unique bins are dropped.

How to use four quantiles to bin the data

```
pd.qcut(fires_filtered.acres_burned, q=4,  
        labels=['small','medium','large','very large'])  
=====  
1145154      medium  
1145175      very large  
...  
1880209      small  
1880370      large  
Name: acres_burned, Length: 4882, dtype: category  
Categories (4, object): ['small' < 'medium' < 'large' < 'very large']
```

The distribution of the values in each bin

```
pd.qcut(fires_filtered.acres_burned, q=4,  
        labels=['small','medium','large','very large']).value_counts()  
=====  
medium      1227  
small       1221  
very large   1220  
large        1214  
Name: acres_burned, dtype: int64
```

How to assign bin labels to a new column

```
fires_filtered['fire_size'] = pd.qcut(  
    fires_filtered.acres_burned, q=4,  
    labels=['small', 'medium', 'large', 'very large'])
```

A qcut() method that drops duplicate bins

```
pd.qcut(fires_filtered.days_burning, q=4,  
        labels=['short','medium','long'],  
        duplicates='drop').value_counts()  
=====  
short      1018  
long       433  
medium     407  
Name: days_burning, dtype: int64
```

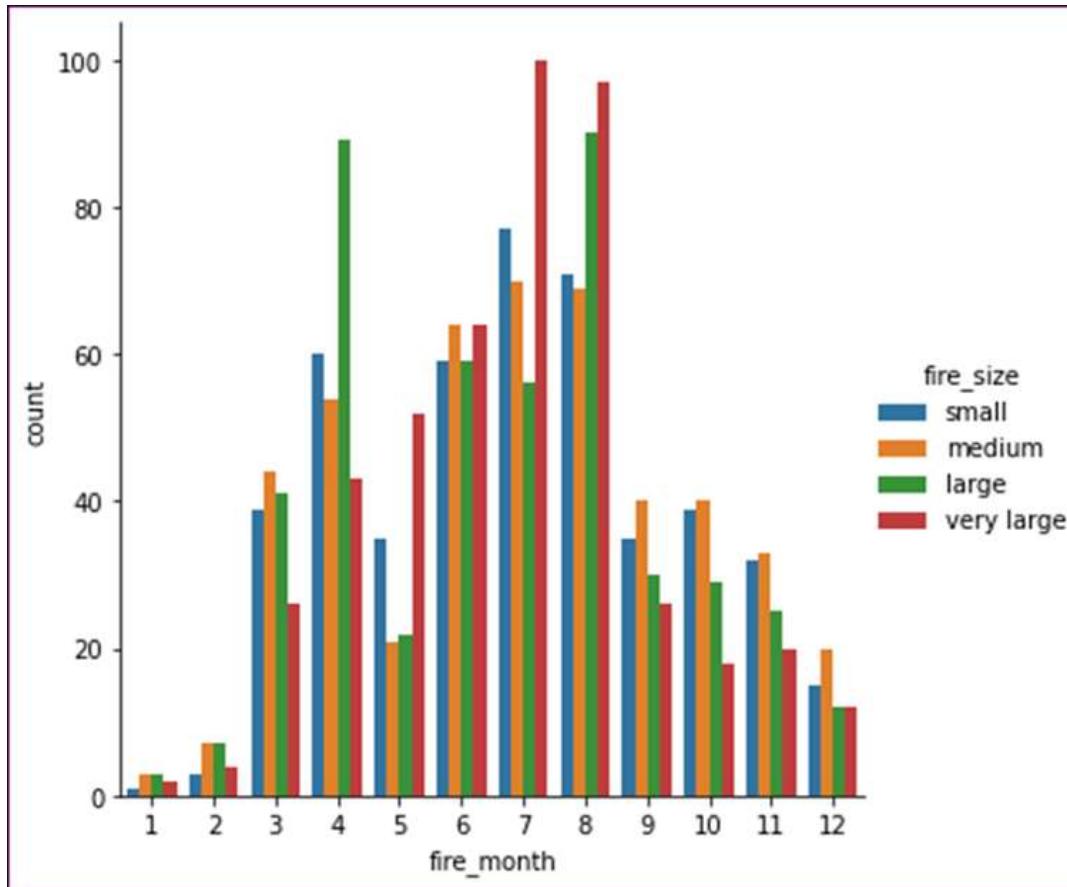
The DataFrame with a `fire_size` column that bins the data

`fires_filtered.head()`

	fire_name	fire_year	state	discovery_date	contain_date	acres_burned	fire_month	days_burning	fire_size
1145154	Fourmile Trail	2010	AK	2010-04-28	2010-05-05	16.8	4	7.0	small
1145175	Granite Tors	2010	AK	2010-05-27	2010-08-11	7880.0	5	76.0	very large
1145187	Goldbug Creek	2010	AK	2010-06-23	2010-06-28	2777.0	6	5.0	very large
1145198	Broken Tree	2010	AK	2010-05-24	2010-07-08	112.0	5	45.0	large
1145208	Folger Creek	2010	AK	2010-07-01	2010-07-16	90.0	7	15.0	medium

How to plot the binned data

```
sns.catplot(data=fires_filtered, kind='count',  
            x='fire_month', hue='fire_size')
```



The `nlargest()` method

Method	Description
<code>nlargest(params)</code>	Returns the first n rows with the largest values in the specified columns.

Parameters of the `nlargest()` method

Parameter	Description
<code>n</code>	The number of rows to return.
<code>columns</code>	The columns that determine which rows to keep.
<code>keep</code>	The rows to keep in the event of a tie. Possible values: first, last, and all. The default is first.

How to use the nlargest() method

```
cars.nlargest(n=6, columns='enginesize')
```

	aspiration	carbody	enginesize	curbweight	price
49	std	sedan	326	3950	36000.0
73	std	sedan	308	3900	40960.0
74	std	hardtop	304	3715	45400.0
47	std	sedan	258	4066	32250.0
48	std	sedan	258	4066	35550.0
71	std	sedan	234	3740	34184.0

Another nlargest() example

```
cars.nlargest(n=6, columns=['enginesize','price'])
```

	aspiration	carbody	enginesize	curbweight	price
49	std	sedan	326	3950	36000.0
73	std	sedan	308	3900	40960.0
74	std	hardtop	304	3715	45400.0
48	std	sedan	258	4066	35550.0
47	std	sedan	258	4066	32250.0
72	std	convertible	234	3685	35056.0

The `pct_change()` method

Method	Description
<code>pct_change()</code>	Calculates the percent change from the previous row to the current row for a DataFrame or Series object.

The fires data

```
df = fires[['state','fire_year','acres_burned']] \
    .groupby(['state','fire_year']).sum()
df.head()
```

acres_burned		
state	fire_year	
AK	1992	142444.7
	1993	686630.5
	1994	261604.7
	1995	43762.6
	1996	598407.2

How to use the pct_change() method

`df.pct_change()`

acres_burned		
state	fire_year	
AK	1992	NaN
	1993	3.820330
	1994	-0.619002
	1995	-0.832715
	1996	12.673941

WY	2011	0.552941
	2012	2.582104
	2013	-0.888021
	2014	-0.866764
	2015	4.046849

The rank() method

Method	Description
<code>rank(params)</code>	Computes numerical data ranks (1 through n) along an axis.

Parameters of the rank() method

Parameter	Description
<code>ascending</code>	If False, ranks in descending order. If True (the default), ranks in ascending order.
<code>method</code>	How to rank the group of records that have ties. Possible values include average (the default), min, max, first, and dense.
<code>pct</code>	If True, displays the rankings in percentile form. False is the default.

The state totals

```
df = fires.groupby('state').sum() \
      [['acres_burned', 'fire_year', 'days_burning']]
df.head(3)
```

	acres_burned	fire_year	days_burning
state			
AK	3.222601e+07	5683445	80268.0
AL	8.101628e+05	38336332	2886.0
AR	4.502221e+05	17960388	1132.0

How to add an acres_rank column based on acres burned

```
df['acres_rank'] = df.acres_burned.rank(ascending=False)  
df.head(3)
```

	acres_burned	fire_year	days_burning	acres_rank
state				
AK	3.222601e+07	5683445	80268.0	1.0
AL	8.101628e+05	38336332	2886.0	23.0
AR	4.502221e+05	17960388	1132.0	27.0

How to add a days_rank column based on days burning

```
df['days_rank'] = df.days_burning.rank(method='max')  
df.sort_values('days_burning').head(4)
```

state	acres_burned	fire_year	days_burning	acres_rank	days_rank
RI	147.45	22092	0.0	51.0	1.0
VT	985.70	46240	6.0	50.0	3.0
CT	7358.20	364159	6.0	46.0	3.0
NH	1232.23	82240	9.0	49.0	4.0

A Google search for a way to count the unique values in a column

The screenshot shows a Google search results page with the query "pandas dataframe count of unique values". The results are filtered under the "All" tab. The first result is a link to a GeeksforGeeks article titled "How to Count Distinct Values of a Pandas Dataframe Column ...". Below it is a link to a Stack Overflow question titled "Count unique values with pandas per groups - Stack Overflow". The Stack Overflow page lists several related questions and their answers, such as "Counting unique values in a column in pandas ..." and "Pandas count(distinct) equivalent - Stack Overflow". Further down the page is another link to a w3resource article titled "Pandas Series: value_counts() function - w3resource".

Google search results for "pandas dataframe count of unique values":

- [www.geeksforgeeks.org › how-to-count-distinct-values...](#) :: How to Count Distinct Values of a Pandas Dataframe Column ...
Aug 19, 2020 — The Dataframe has been created and one can hard coded using for loop and count the number of unique values in a specific column.
- [stackoverflow.com › questions › count-unique-values-...](#) :: Count unique values with pandas per groups - Stack Overflow
Jul 11, 2016 — The difference is that nunique() returns a Series and agg() returns a DataFrame.
4 answers · Top answer: You need nunique: df = df.groupby('domain')[ID].nunique() print (df) d...
 - [Counting unique values in a column in pandas ...](#) 6 answers Aug 18, 2017
 - [Pandas count\(distinct\) equivalent - Stack Overflow](#) 8 answers Jul 2, 2017
 - [Count of unique value in column pandas - Stack ...](#) 2 answers Jan 15, 2017
 - [How to find out number of unique values in a column ...](#) 3 answers Mar 28, 2017
- [www.w3resource.com › pandas › series-value_counts](#) :: Pandas Series: value_counts() function - w3resource
Aug 15, 2020 — Pandas Series: value_counts() function The value_counts() function is used to get a Series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.