Chapter 7

How to prepare the data

Objectives

Applied

1. Add datetime, string, and numeric columns that are derived from other columns to a DataFrame.
2. Add summary columns to a DataFrame.
3. Apply functions, user-defined functions, and lambda expressions to a DataFrame.
4. Set an index, unstack the data based on an index, and reset the index.
5. Join, merge, and concatenate DataFrames.
6. Fix any problems that are identified by the SettingWithCopyWarning.

Objectives (continued)

Knowledge

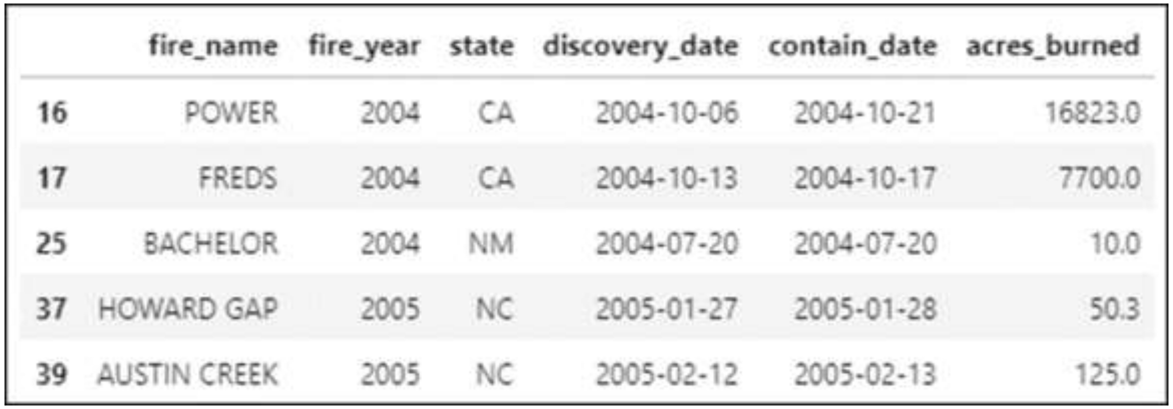
1. Describe the appropriate uses for the join(), merge(), and concat() methods.
2. Describe an inner join.
3. Describe the SettingWithCopyWarning message and the type of problem that it may identify.

Some of the properties that are available from the dt accessor

```
year
month
day
quarter
days
seconds
```

The fires DataFrame

```
fires.head()
```



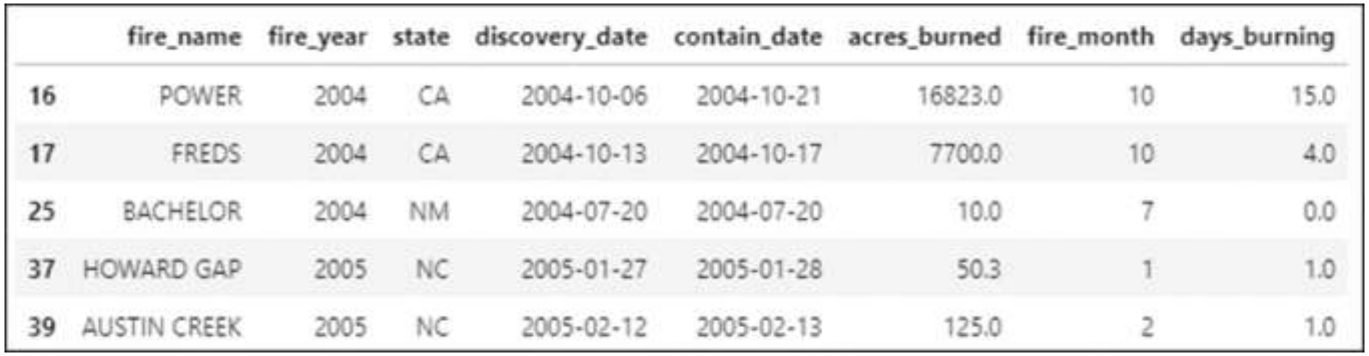
	fire_name	fire_year	state	discovery_date	contain_date	acres_burned
16	POWER	2004	CA	2004-10-06	2004-10-21	16823.0
17	FREDS	2004	CA	2004-10-13	2004-10-17	7700.0
25	BACHELOR	2004	NM	2004-07-20	2004-07-20	10.0
37	HOWARD GAP	2005	NC	2005-01-27	2005-01-28	50.3
39	AUSTIN CREEK	2005	NC	2005-02-12	2005-02-13	125.0

How to add a numeric column derived from a datetime column

```
fires['fire_month'] = fires.discovery_date.dt.month
```

How to add a numeric column derived from a datetime calculation

```
fires['days_burning'] = \
    (fires.contain_date – fires.discovery_date).dt.days
```



	fire_name	fire_year	state	discovery_date	contain_date	acres_burned	fire_month	days_burning
16	POWER	2004	CA	2004-10-06	2004-10-21	16823.0	10	15.0
17	FREDS	2004	CA	2004-10-13	2004-10-17	7700.0	10	4.0
25	BACHELOR	2004	NM	2004-07-20	2004-07-20	10.0	7	0.0
37	HOWARD GAP	2005	NC	2005-01-27	2005-01-28	50.3	1	1.0
39	AUSTIN CREEK	2005	NC	2005-02-12	2005-02-13	125.0	2	1.0

Some of the methods that are available from the str accessor

```
count(str)
lower(str)
upper(str)
title(str)
lstrip(str)
rstrip(str)
strip(str)
startswith(str)
endswith(str)
find(str)
replace(old,new)
join(sequence)
```

How to modify a column derived from string data

```
fires['fire_name'] = fires.fire_name.str.title()
```

How to add a column derived from string data

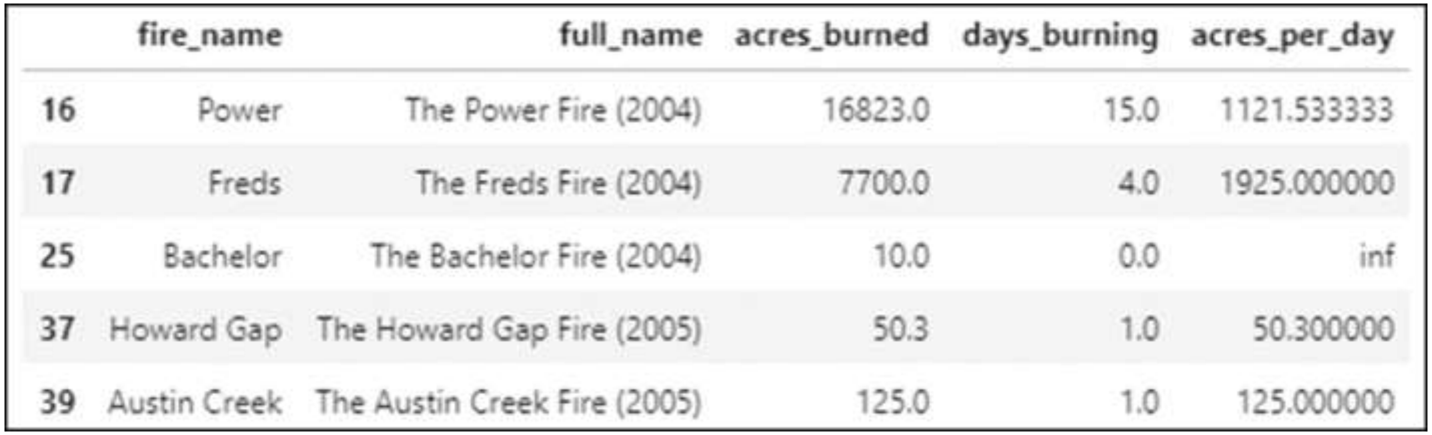
```
fires['full_name'] = 'The ' + fires.fire_name + ' Fire ' \
                     + '(' + fires.fire_year.astype(str) + ')'
```

How to add a column derived from numeric computations

```
fires['acres_per_day'] = \
    fires.dropna().acres_burned / fires.dropna().days_burning
```

The new and modified columns

```
fires[['fire_name','full_name','acres_burned','days_burning',
    'acres_per_day']].head()
```



	fire_name	full_name	acres_burned	days_burning	acres_per_day
16	Power	The Power Fire (2004)	16823.0	15.0	1121.533333
17	Freds	The Freds Fire (2004)	7700.0	4.0	1925.000000
25	Bachelor	The Bachelor Fire (2004)	10.0	0.0	inf
37	Howard Gap	The Howard Gap Fire (2005)	50.3	1.0	50.300000
39	Austin Creek	The Austin Creek Fire (2005)	125.0	1.0	125.000000

The transform() method

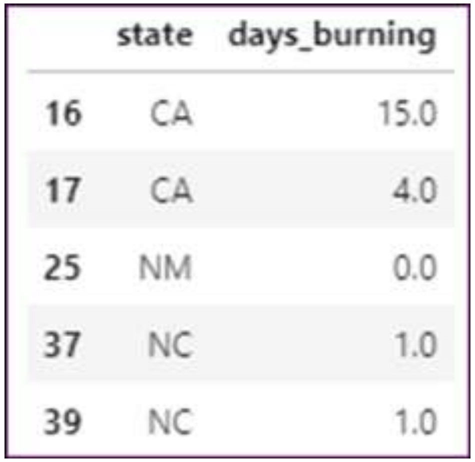
Method	Description
`transform(params)`	Adds summary values to each row.

Parameters of the transform() method

Parameter	Description
`func`	The function to apply.
`axis`	0 (the default) for rows; 1 for columns.

The fires data

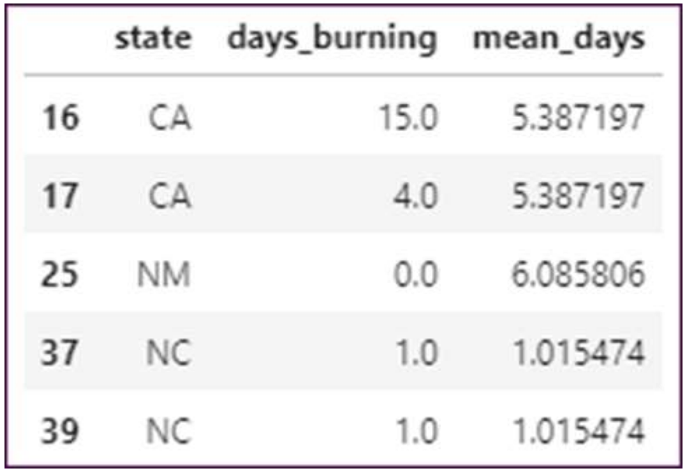
```
fires[['state','days_burning']].head()
```



```
   state  days_burning
16    CA          15.0
17    CA           4.0
25    NM           0.0
37    NC           1.0
39    NC           1.0
```

How to add a summary column to a DataFrame

```
fires['mean_days'] = fires.groupby('state')['days_burning']. \
    transform(func='mean')
fires[['state','days_burning','mean_days']].head()
```



	state	days_burning	mean_days
16	CA	15.0	5.387197
17	CA	4.0	5.387197
25	NM	0.0	6.085806
37	NC	1.0	1.015474
39	NC	1.0	1.015474

The apply() method

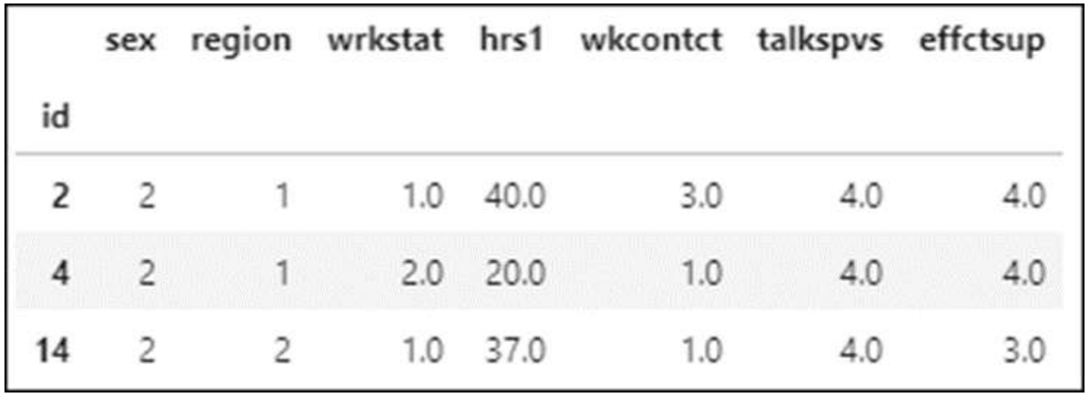
Method	Description
`apply(params)`	Applies a function to the data in a row or a column and returns a Series.

Parameters of the apply() method

Parameter	Description
`function`	The function that's applied to each row or column. It can be a built-in function, a NumPy function, a user-defined function, or a lambda expression.
`axis`	The axis that the function is applied to: axis=0 (the default) for columns and axis=1 for rows.

The workData DataFrame

```
workData.head(3)
```



	sex	region	wrkstat	hrs1	wkcontct	talkspvs	effctsup
id							
2	2	1	1.0	40.0	3.0	4.0	4.0
4	2	1	2.0	20.0	1.0	4.0	4.0
14	2	2	1.0	37.0	1.0	4.0	3.0

How to apply a built-in Pandas function to all numeric columns

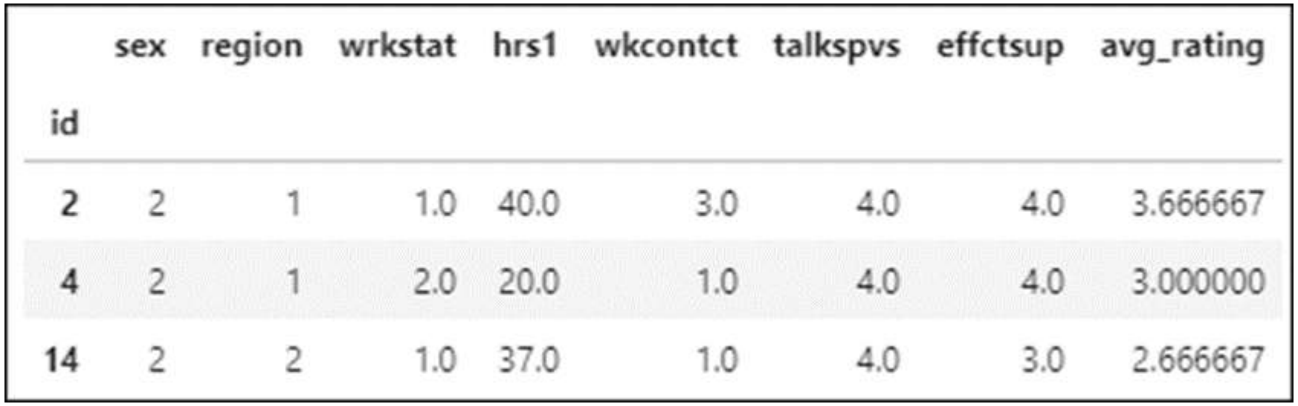
```
workData.apply('mean')
```

How to apply a NumPy function to two columns

```
import numpy as np
workData[['sex','hrs1']].apply(np.mean)
==============================================
sex       1.529897
hrs1     42.083505
```

How to apply a function to row data

```
workData['avg_rating'] = workData[
    ['wkcontct','talkspvs','effctsup']].apply(np.mean, axis=1)
workData.head(3)
```

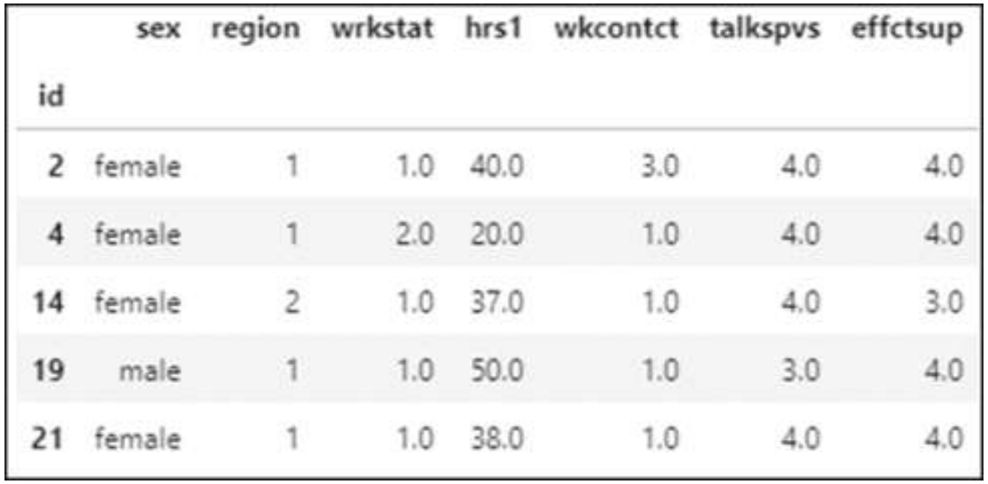


	sex	region	wrkstat	hrs1	wkcontct	talkspvs	effctsup	avg_rating
id								
2	2	1	1.0	40.0	3.0	4.0	4.0	3.666667
4	2	1	2.0	20.0	1.0	4.0	4.0	3.000000
14	2	2	1.0	37.0	1.0	4.0	3.0	2.666667

How to apply a user-defined function to a column

```
def convert_sex(row):
    if row.sex == 1:
        return 'male'
    elif row.sex == 2:
        return 'female'
    else:
        return 'non-binary'

workData['sex'] = workData.apply(convert_sex, axis=1)
workData.head()
```

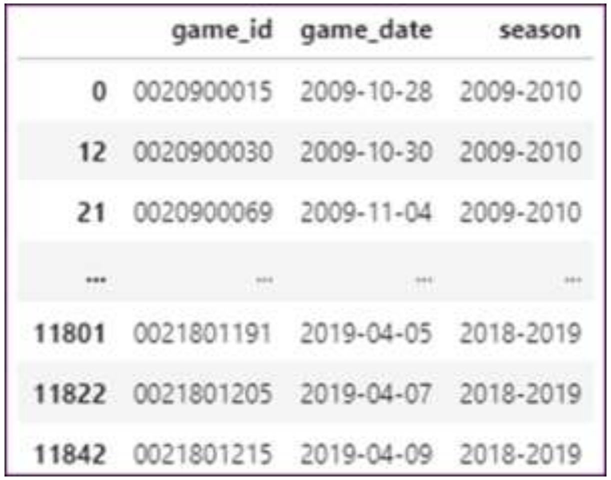


id	sex	region	wrkstat	hrs1	wkcontct	talkspvs	effctsup
2	female	1	1.0	40.0	3.0	4.0	4.0
4	female	1	2.0	20.0	1.0	4.0	4.0
14	female	2	1.0	37.0	1.0	4.0	3.0
19	male	1	1.0	50.0	1.0	3.0	4.0
21	female	1	1.0	38.0	1.0	4.0	4.0

How to apply another user-defined function to a column

```
def get_season(row):
    if row.game_date.month > 6:
        season = f'{row.game_date.year}-{row.game_date.year + 1}'
    else:
        season = f'{row.game_date.year - 1}-{row.game_date.year}'
    return season

gameData['season'] = gameData.apply(get_season, axis=1)
with pd.option_context('display.max_rows', 6,
                       'display.max_columns', None):
    display(gameData)
```



	game_id	game_date	season
0	0020900015	2009-10-28	2009-2010
12	0020900030	2009-10-30	2009-2010
21	0020900069	2009-11-04	2009-2010
...	...	...	...
11801	0021801191	2019-04-05	2018-2019
11822	0021801205	2019-04-07	2018-2019
11842	0021801215	2019-04-09	2018-2019

The syntax of a lambda expression

The if syntax

```
lambda arguments: return-value-if-true if condition
```

The if-else syntax

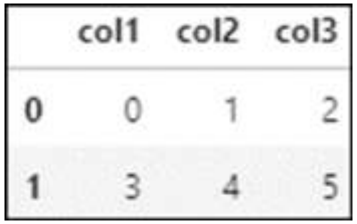
```
lambda arguments: return-value-if-true if condition else
    return_value-if-false
```

The if-elif-else syntax

```
lambda arguments: return_value-if-condition1-true if
condition1 else
    (return-value-if-condition-2-true if condition-2 else
     return_value-if-condition-2-false)
```

The example data

```
df
```



	col1	col2	col3
0	0	1	2
1	3	4	5

A lambda expression that sums and then doubles the value in each column

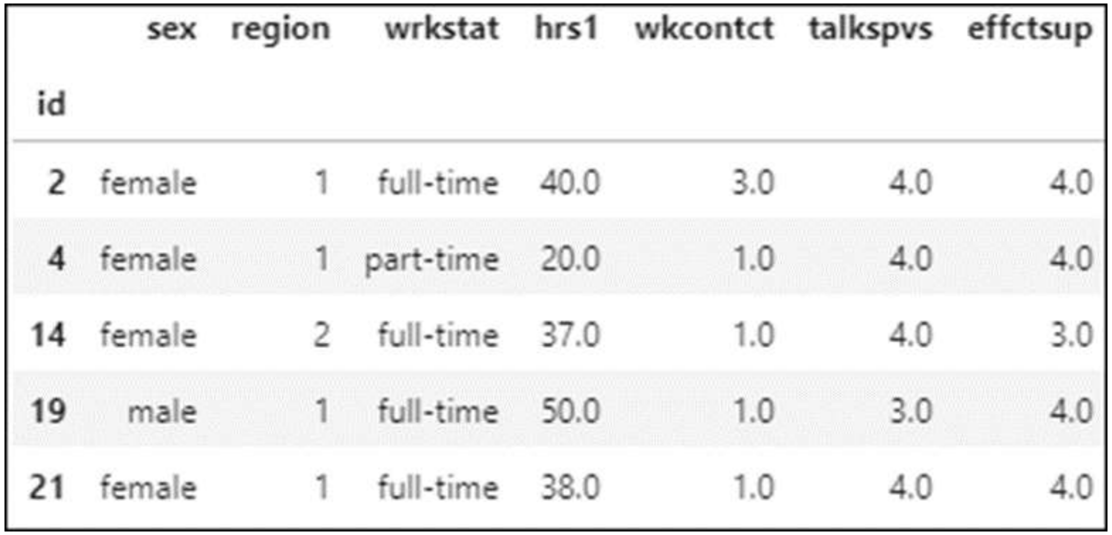
```
df.apply(lambda x: x.sum() * 2, axis=0)
=======================================
col1     6
col2    10
col3    14
dtype: int64
```

A lambda expression that sums and then doubles the value in each row

```
df.apply(lambda x: x.sum() * 2, axis=1)
=======================================
0     6
1    24
dtype: int64
```

How to apply a lambda expression to a column

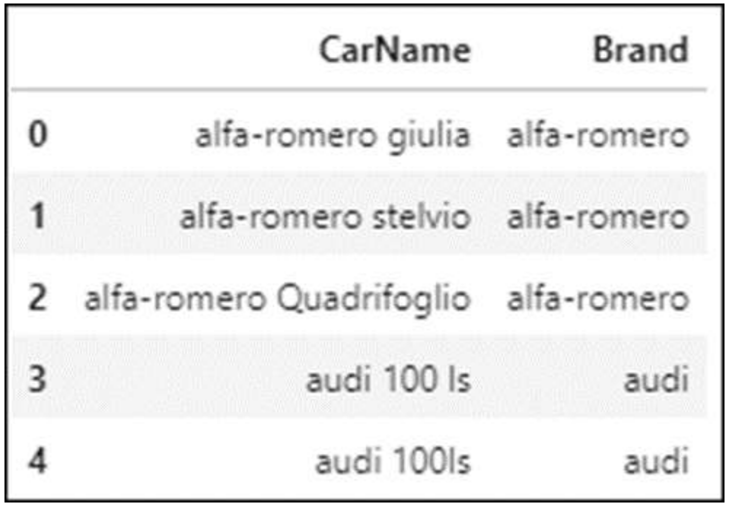
```
workData['wrkstat'] = workData.apply(
    lambda row: 'full-time' if row.wrkstat == 1.0 else 'part-time',
        axis=1)
workData.head()
```



	sex	region	wrkstat	hrs1	wkcontct	talkspvs	effctsup
id							
2	female	1	full-time	40.0	3.0	4.0	4.0
4	female	1	part-time	20.0	1.0	4.0	4.0
14	female	2	full-time	37.0	1.0	4.0	3.0
19	male	1	full-time	50.0	1.0	3.0	4.0
21	female	1	full-time	38.0	1.0	4.0	4.0

How to use a lambda expression to add a new column

```
carsData['Brand'] = carsData.apply(
    lambda x: x.CarName.split()[0], axis=1)
carsData[['CarName','Brand']].head()
```



	CarName	Brand
0	alfa-romero giulia	alfa-romero
1	alfa-romero stelvio	alfa-romero
2	alfa-romero Quadrifoglio	alfa-romero
3	audi 100 ls	audi
4	audi 100ls	audi

The set_index() method

Method	Description
`set_index(params)`	Sets an index for the specified column or list of columns.

Parameters of the set_index() method

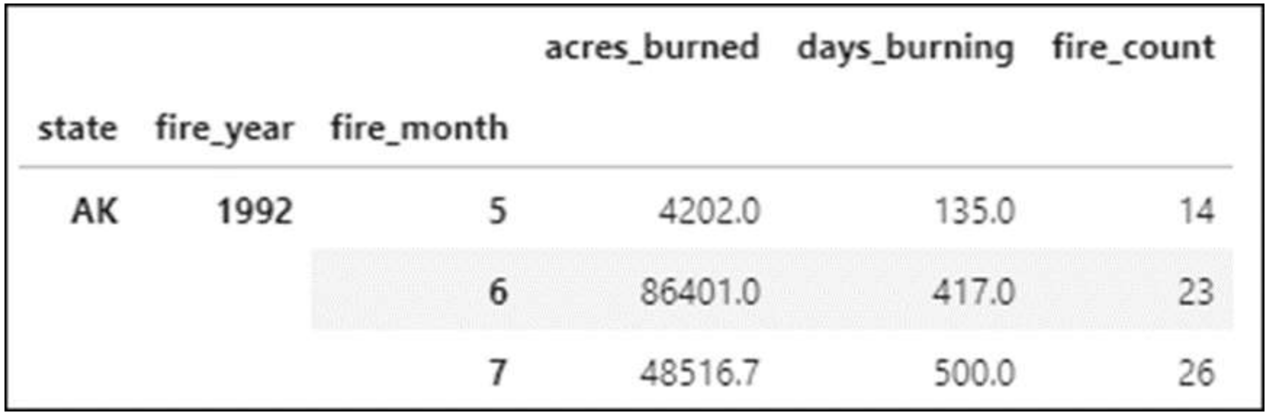
Parameter	Description
`keys`	The column or list of columns to be used for the index.
`inplace`	If set to True, modifies the DataFrame in place.
`verify_integrity`	If set to True, throws an error if the specified index contains any duplicates.

How to set an index

```
fires_by_month.set_index('state', inplace=True)
```

How to set an index on multiple columns

```
fires_by_month.set_index(['state','fire_year','fire_month'],
inplace=True)
fires_by_month.head(3)
```



			acres_burned	days_burning	fire_count
state	fire_year	fire_month			
AK	1992	5	4202.0	135.0	14
		6	86401.0	417.0	23
		7	48516.7	500.0	26

The reset_index() method

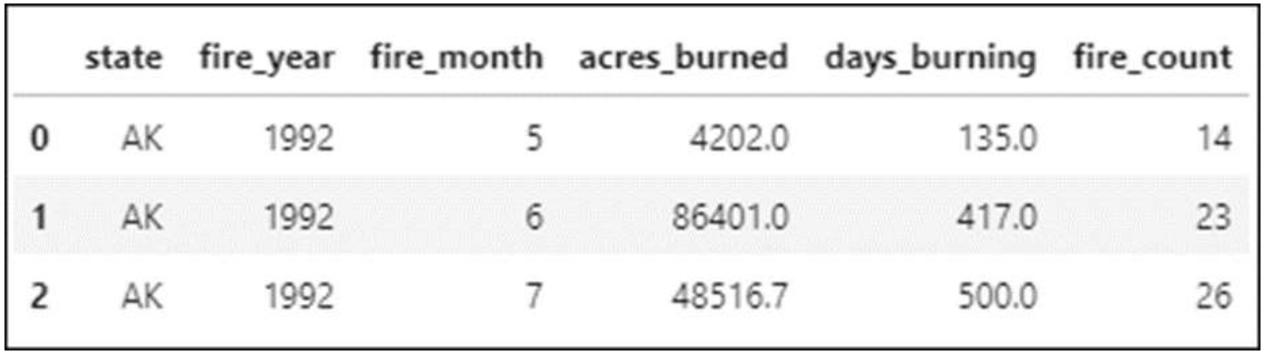
Method	Description
`reset_index(params)`	Resets the index to an auto-generated list of integers.

Parameters of the reset_index() method

Parameter	Description
`inplace`	If set to True, modifies the DataFrame in place.
`level`	The level of index to reset. By default, all indexes are reset.
`drop`	If set to True, drops the index columns instead of converting them to DataFrame columns.

How to remove an index

```
fires_by_month.reset_index(inplace=True)
fires_by_month.head(3)
```



	state	fire_year	fire_month	acres_burned	days_burning	fire_count
0	AK	1992	5	4202.0	135.0	14
1	AK	1992	6	86401.0	417.0	23
2	AK	1992	7	48516.7	500.0	26

The unstack() method

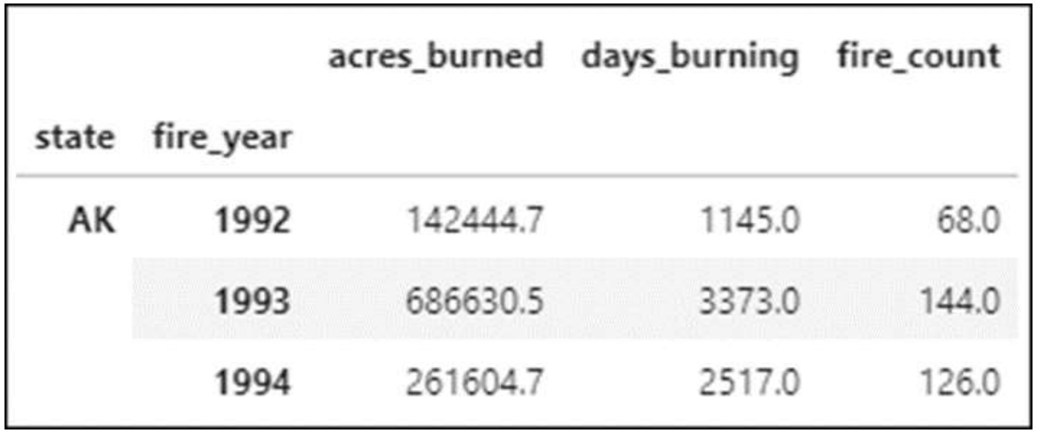
Method	Description
`unstack(params)`	Unstacks the data in an indexed DataFrame.

Parameters of the unstack() method

Parameter	Description
`level`	The index column or list of index columns to unstack.
`fill_value`	A value that replaces any NA values.

The top_states DataFrame in long form

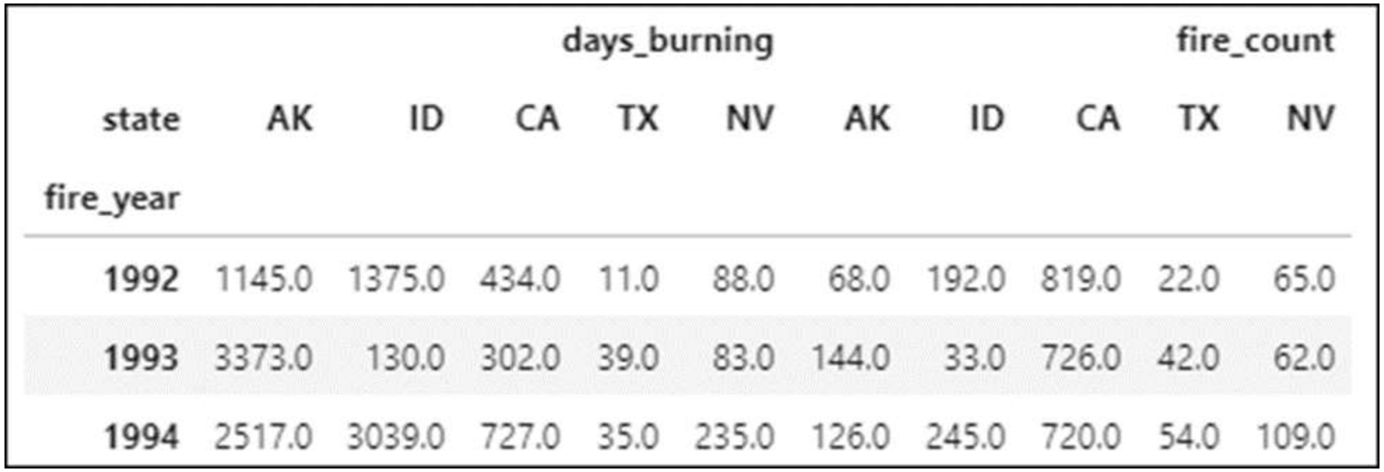
```
top5_states.set_index(['state','fire_year'], inplace=True)
top5_states.head(3)
```



		acres_burned	days_burning	fire_count
state	fire_year			
AK	1992	142444.7	1145.0	68.0
	1993	686630.5	3373.0	144.0
	1994	261604.7	2517.0	126.0

How to unstack two of the columns at the state level

```
top_wide = \
    top5_states[['days_burning','fire_count']].unstack(level='state')
top_wide.head(3)
```



	days_burning					fire_count				
state	AK	ID	CA	TX	NV	AK	ID	CA	TX	NV
fire_year										
1992	1145.0	1375.0	434.0	11.0	88.0	68.0	192.0	819.0	22.0	65.0
1993	3373.0	130.0	302.0	39.0	83.0	144.0	33.0	726.0	42.0	62.0
1994	2517.0	3039.0	727.0	35.0	235.0	126.0	245.0	720.0	54.0	109.0

How to unstack all columns at the state level

```
top_wide = top5_states.unstack(level='state')
```

How to unstack a single column at the state level

```
top_wide = top5_states.fire_count.unstack(level='state')
```

How to use an integer to specify the state level

```
top_wide = top5_states[['days_burning','fire_count']].unstack(level=0')
```

How to unstack the rightmost index column

```
top_wide = top5_states.unstack()
```

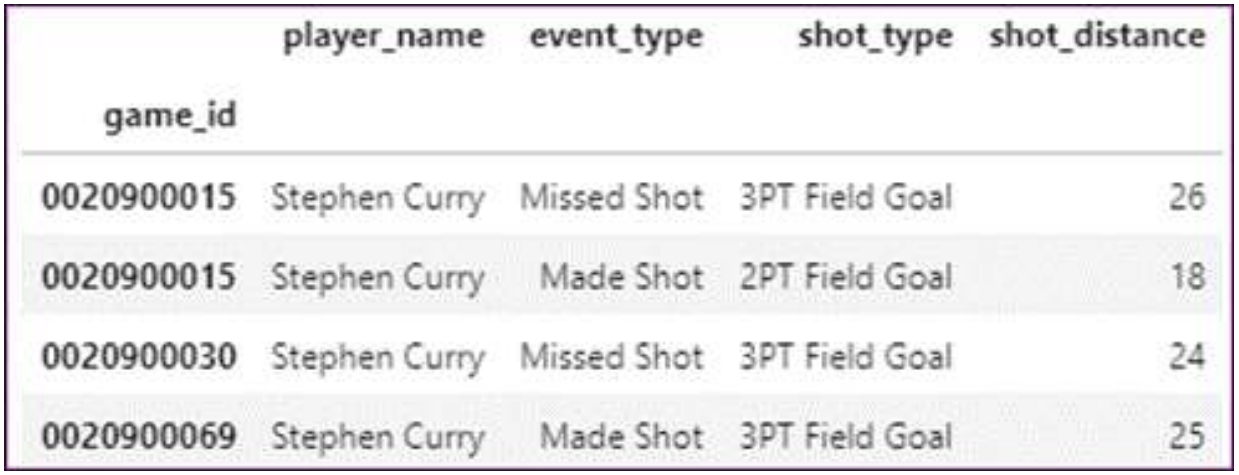
The join() method

Method	Description
`join(params)`	Joins the columns in the left DataFrame with the columns in the right DataFrame based on an index.

Parameters of the join() method

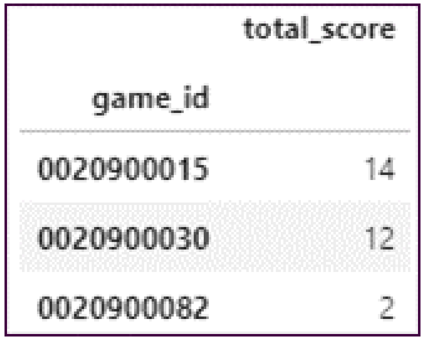
Parameter	Description
`df`	The DataFrame or list of DataFrames to join with the calling DataFrame.
`on`	The column to be used for the index in the left DataFrame.
`how`	The type of join to use: left (the default), right, inner, or outer.
`lsuffix`	The suffix to append to any overlapping columns in the left DataFrame.
`rsuffix`	The suffix to append to any overlapping columns in the right DataFrame.

The shots DataFrame (just 4 rows)



	player_name	event_type	shot_type	shot_distance
game_id				
0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26
0020900015	Stephen Curry	Made Shot	2PT Field Goal	18
0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24
0020900069	Stephen Curry	Made Shot	3PT Field Goal	25

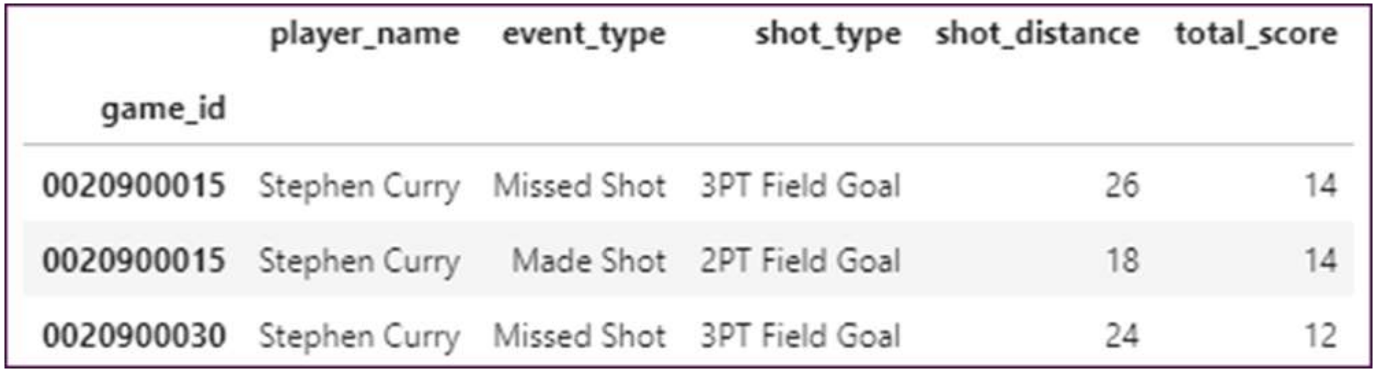
The points_by_game DataFrame (just 3 rows)



	total_score
game_id	
0020900015	14
0020900030	12
0020900082	2

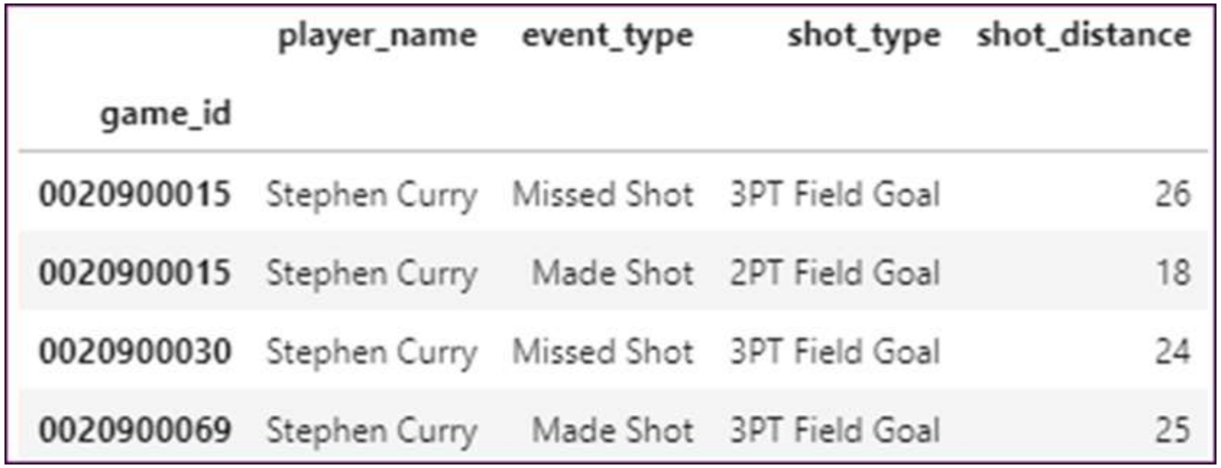
How to inner join the two DataFrames

```
shots_joined = shots.join(points_by_game, how='inner')
shots_joined
```



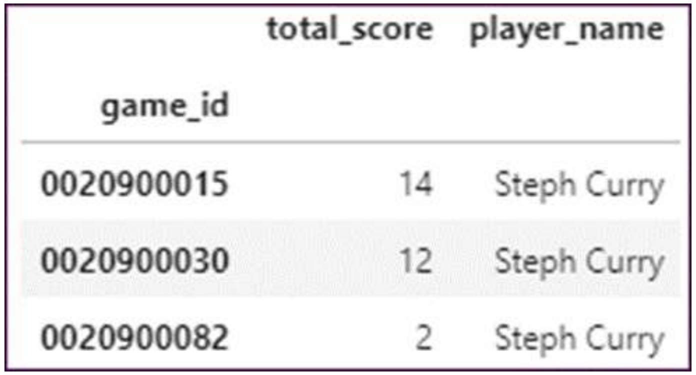
	player_name	event_type	shot_type	shot_distance	total_score
game_id					
0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26	14
0020900015	Stephen Curry	Made Shot	2PT Field Goal	18	14
0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24	12

The shots DataFrame (just 4 rows)



	player_name	event_type	shot_type	shot_distance
game_id				
0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26
0020900015	Stephen Curry	Made Shot	2PT Field Goal	18
0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24
0020900069	Stephen Curry	Made Shot	3PT Field Goal	25

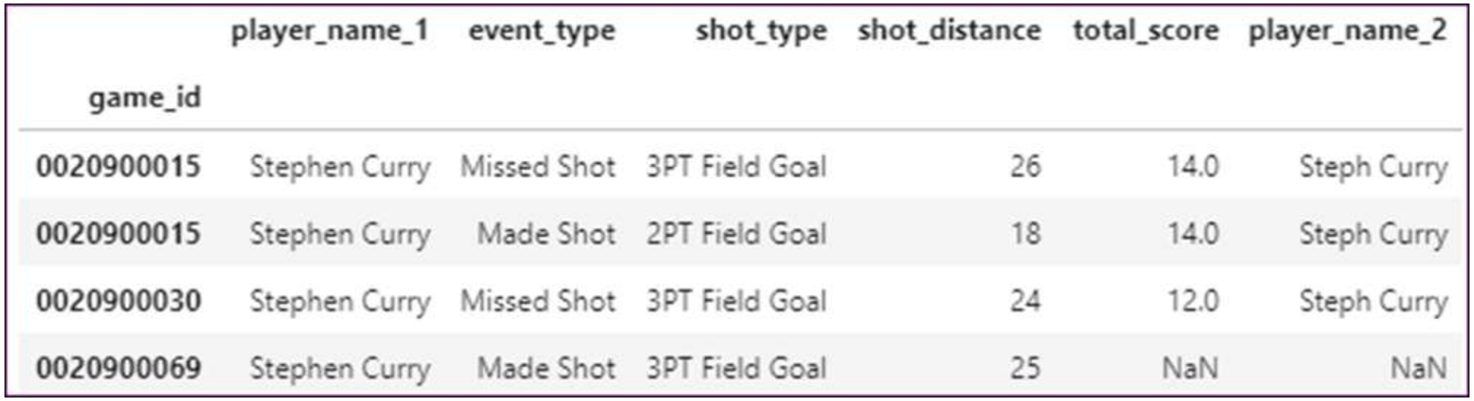
The points_by_game DataFrame (just 3 rows)



	total_score	player_name
game_id		
0020900015	14	Steph Curry
0020900030	12	Steph Curry
0020900082	2	Steph Curry

How to left join the two DataFrames

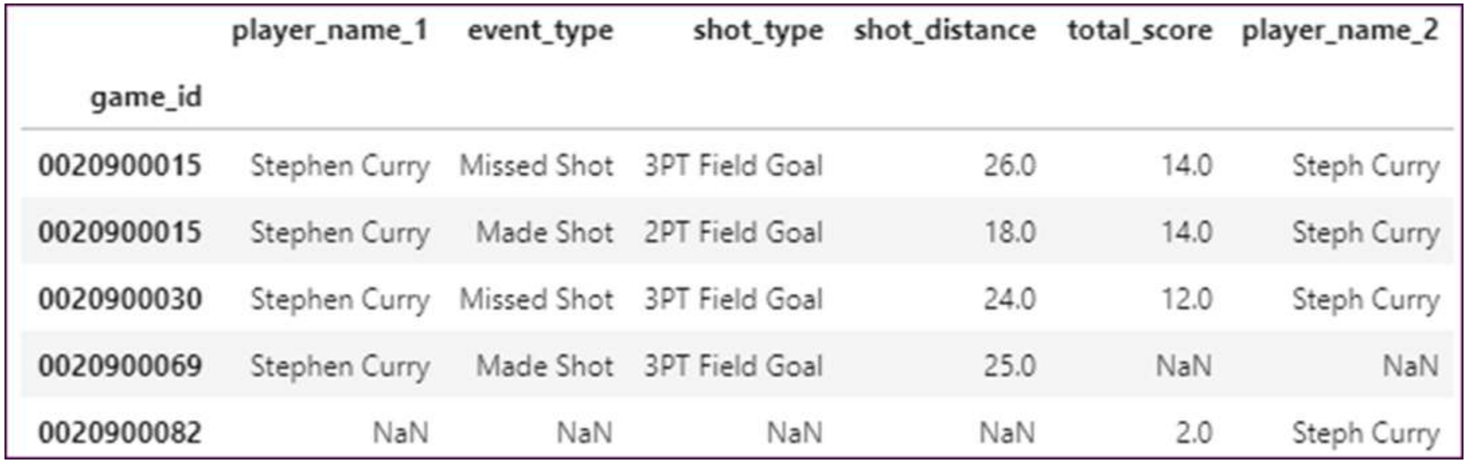
```
shots_joined = shots.join(points_by_game, lsuffix='_1',
                           rsuffix='_2', how='left')
shots_joined
```



```
            player_name_1  event_type      shot_type  shot_distance  total_score  player_name_2
game_id
0020900015  Stephen Curry  Missed Shot  3PT Field Goal            26         14.0    Steph Curry
0020900015  Stephen Curry    Made Shot  2PT Field Goal            18         14.0    Steph Curry
0020900030  Stephen Curry  Missed Shot  3PT Field Goal            24         12.0    Steph Curry
0020900069  Stephen Curry    Made Shot  3PT Field Goal            25          NaN            NaN
```

How to outer join the two DataFrames

```
shots_joined_outer = shots.join(points_by_game, lsuffix='_1',
                                          rsuffix='_2', how='outer')
shots_joined_outer
```



game_id	player_name_1	event_type	shot_type	shot_distance	total_score	player_name_2
0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26.0	14.0	Steph Curry
0020900015	Stephen Curry	Made Shot	2PT Field Goal	18.0	14.0	Steph Curry
0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24.0	12.0	Steph Curry
0020900069	Stephen Curry	Made Shot	3PT Field Goal	25.0	NaN	NaN
0020900082	NaN	NaN	NaN	NaN	2.0	Steph Curry

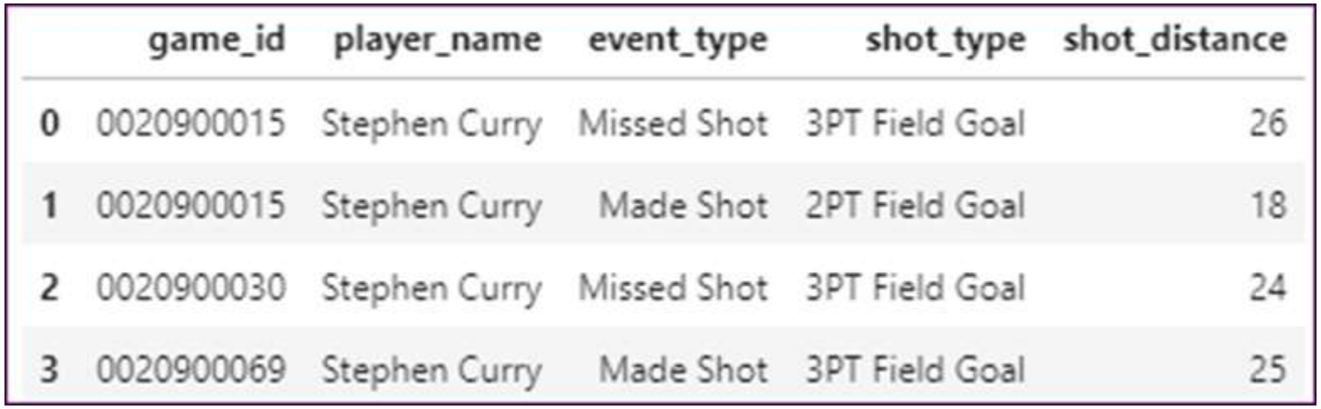
The merge() method

Method	Description
`merge(params)`	Merges the columns in the left DataFrame with the columns in the right DataFrame based on the data in one or more columns that aren’t indexed.

Parameters of the merge() method

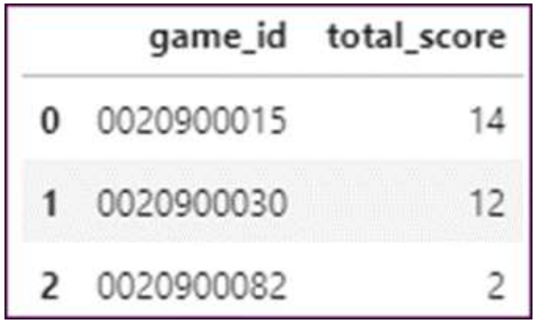
Parameter	Description
`right`	The DataFrame to merge with the current DataFrame.
`on`	The column or list of columns to merge on. By default, it uses the column names that are the same in both DataFrames.
`how`	Works like the join() method but uses 'inner' by default.
`suffixes`	A tuple that provides the values to append to columns with the same name in the left and right DataFrames.

The shots DataFrame (just 4 rows)



	game_id	player_name	event_type	shot_type	shot_distance
0	0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26
1	0020900015	Stephen Curry	Made Shot	2PT Field Goal	18
2	0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24
3	0020900069	Stephen Curry	Made Shot	3PT Field Goal	25

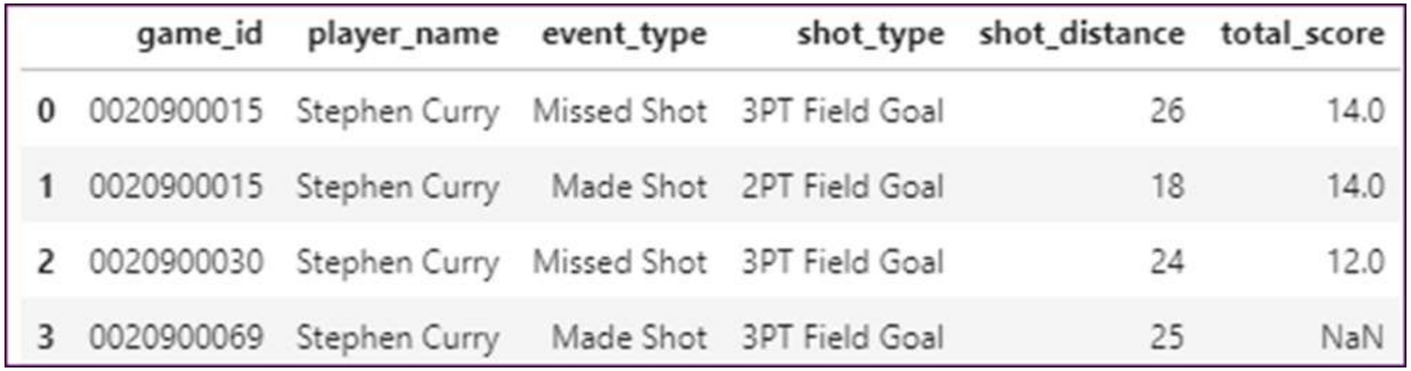
The points_by_game DataFrame (just 3 rows)



	game_id	total_score
0	0020900015	14
1	0020900030	12
2	0020900082	2

How to merge the two DataFrames

```
shots_merged = shots.merge(points_by_game, on='game_id',
                           how='left')
shots_merged
```



	game_id	player_name	event_type	shot_type	shot_distance	total_score
0	0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26	14.0
1	0020900015	Stephen Curry	Made Shot	2PT Field Goal	18	14.0
2	0020900030	Stephen Curry	Missed Shot	3PT Field Goal	24	12.0
3	0020900069	Stephen Curry	Made Shot	3PT Field Goal	25	NaN

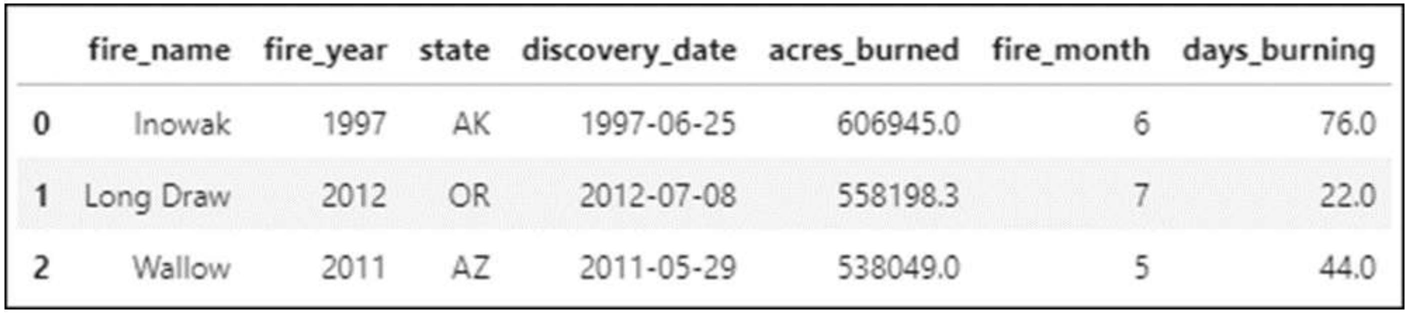
The concat() method

Method	Description
`concat(params)`	Concatenates (adds) the data in one DataFrame to another DataFrame.

Parameters of the concat() method

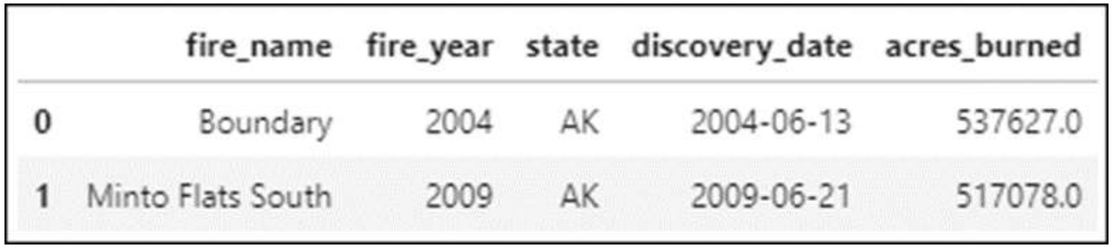
Parameter	Description
`objs`	A list of the DataFrames that you want to concatenate.
`axis`	The default of 0 adds rows to the bottom of the first DataFrame. Setting it to 1 adds columns to the right side of the first DataFrame.
`ignore_index`	If True, don't keep the index values along the concatenation axis. Instead, reset the index on that axis.
`join`	If 'inner', use an inner join. Otherwise, use an outer join.

The fires_1 DataFrame (just 3 rows)



	fire_name	fire_year	state	discovery_date	acres_burned	fire_month	days_burning
0	Inowak	1997	AK	1997-06-25	606945.0	6	76.0
1	Long Draw	2012	OR	2012-07-08	558198.3	7	22.0
2	Wallow	2011	AZ	2011-05-29	538049.0	5	44.0

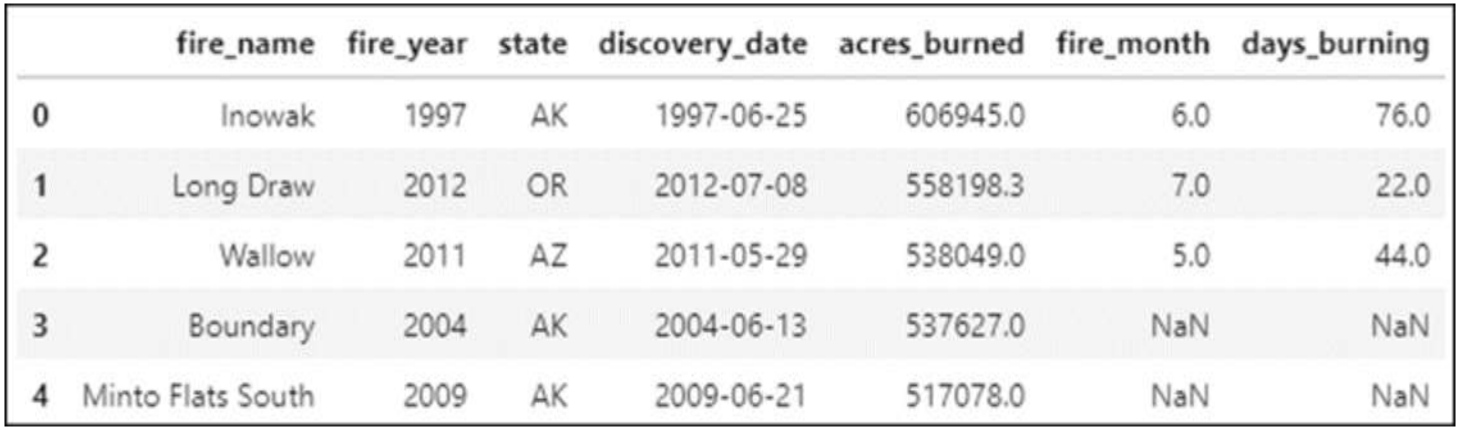
The fires_2 DataFrame (just 2 rows)



	fire_name	fire_year	state	discovery_date	acres_burned
0	Boundary	2004	AK	2004-06-13	537627.0
1	Minto Flats South	2009	AK	2009-06-21	517078.0

The concatenated DataFrame

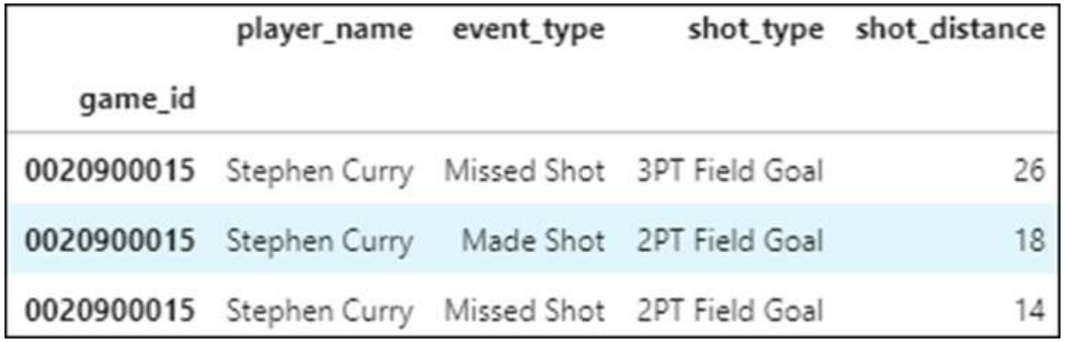
```
fires_concat = pd.concat([fires_1,fires_2], ignore_index=True)
fires_concat
```



	fire_name	fire_year	state	discovery_date	acres_burned	fire_month	days_burning
0	Inowak	1997	AK	1997-06-25	606945.0	6.0	76.0
1	Long Draw	2012	OR	2012-07-08	558198.3	7.0	22.0
2	Wallow	2011	AZ	2011-05-29	538049.0	5.0	44.0
3	Boundary	2004	AK	2004-06-13	537627.0	NaN	NaN
4	Minto Flats South	2009	AK	2009-06-21	517078.0	NaN	NaN

The SettingWithCopyWarning

```
df.head()
```



	player_name	event_type	shot_type	shot_distance
game_id				
0020900015	Stephen Curry	Missed Shot	3PT Field Goal	26
0020900015	Stephen Curry	Made Shot	2PT Field Goal	18
0020900015	Stephen Curry	Missed Shot	2PT Field Goal	14

Code that generates the warning and corrupts the data

```
dfSlice = df.loc['0020900015',:]            # creates the slice (view)
dfSlice.loc[:,'player_name'] = 'Curry'      # modifies the slice
```

The warning

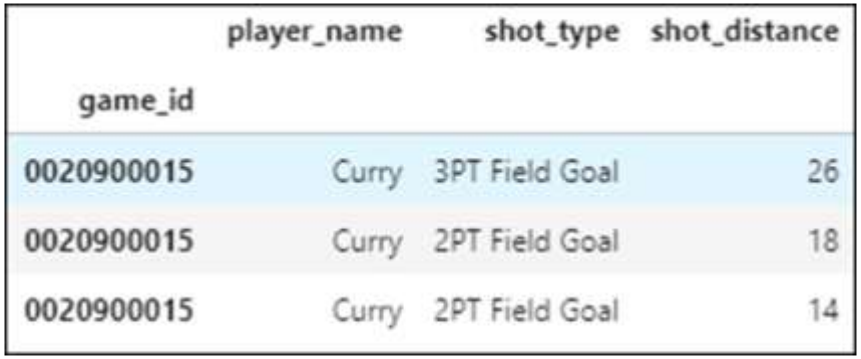
```
<ipython-input-93-57fca793a825>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

The SettingWithCopyWarning (continued)

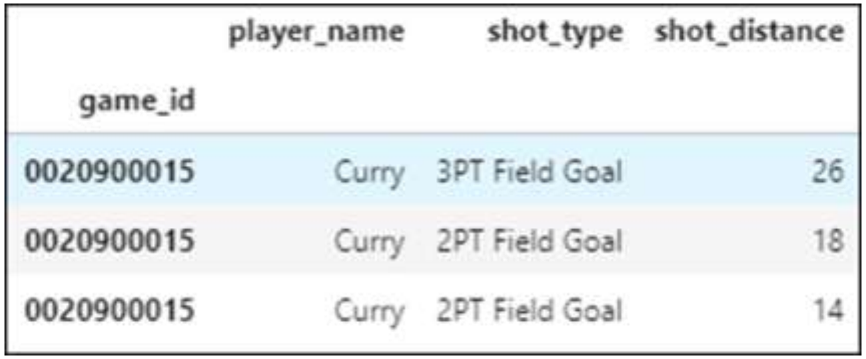
The data after modification...but both DataFrames have been modified!

```
df.head(3)
```



	player_name	shot_type	shot_distance
game_id			
0020900015	Curry	3PT Field Goal	26
0020900015	Curry	2PT Field Goal	18
0020900015	Curry	2PT Field Goal	14

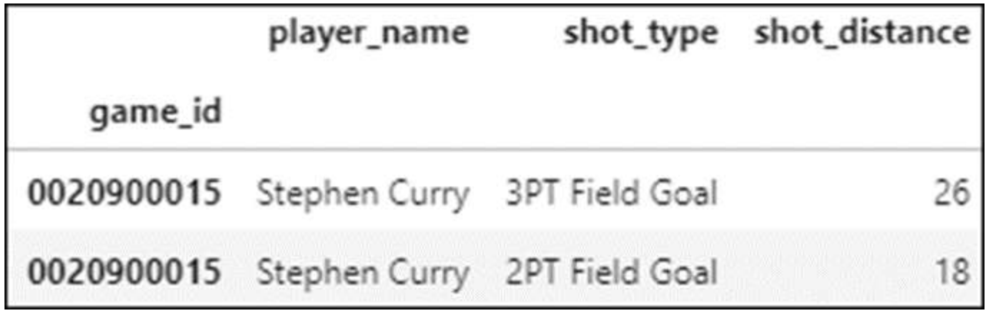
```
dfSlice.head(3)
```



	player_name	shot_type	shot_distance
game_id			
0020900015	Curry	3PT Field Goal	26
0020900015	Curry	2PT Field Goal	18
0020900015	Curry	2PT Field Goal	14

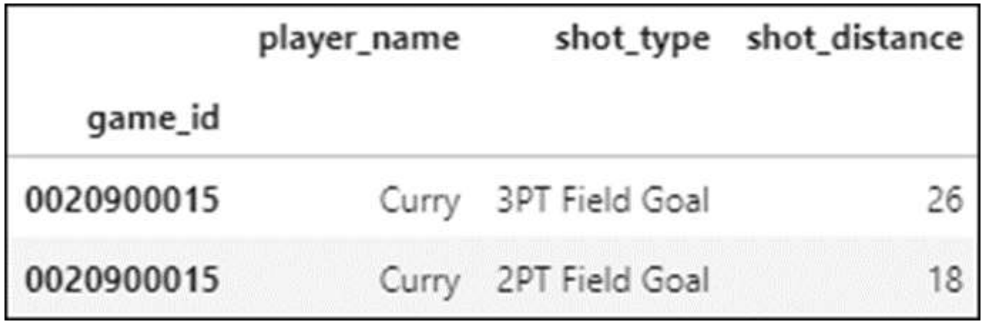
Code that generates the warning and doesn’t corrupt the data

```
dfSlice = df.query('game_id == "0020900015"')
dfSlice.loc[:,'player_name'] = 'Curry'
df.head(2)
```



	player_name	shot_type	shot_distance
game_id			
0020900015	Stephen Curry	3PT Field Goal	26
0020900015	Stephen Curry	2PT Field Goal	18

```
dfSlice.head(2)
```



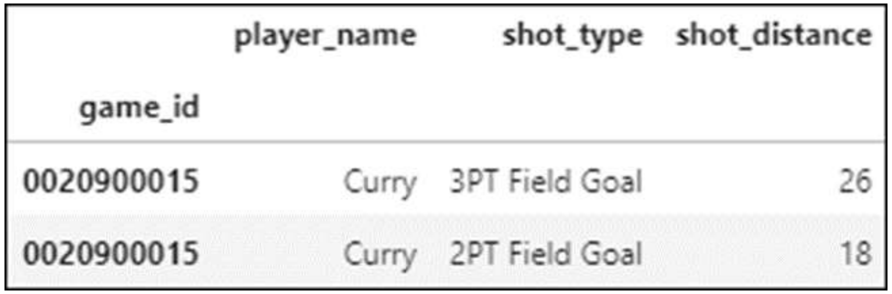
	player_name	shot_type	shot_distance
game_id			
0020900015	Curry	3PT Field Goal	26
0020900015	Curry	2PT Field Goal	18

How to use the copy() method to stop the warning message

```
dfFixed = df.query('game_id == "0020900015"').copy()
dfFixed.loc[:,'player_name'] = 'Curry'
```

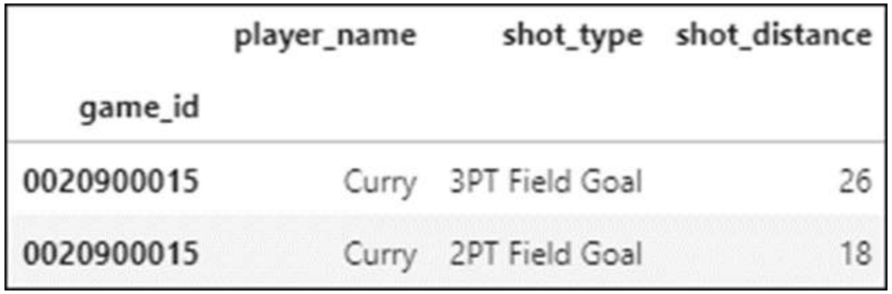
Code that generates the warning and does corrupt the data

```
dfSlice = df.loc['0020900015',:]
dfSlice.loc[:,'player_name'] = 'Curry'
df.head(2)
```



	player_name	shot_type	shot_distance
game_id			
0020900015	Curry	3PT Field Goal	26
0020900015	Curry	2PT Field Goal	18

```
dfSlice.head(2)
```



	player_name	shot_type	shot_distance
game_id			
0020900015	Curry	3PT Field Goal	26
0020900015	Curry	2PT Field Goal	18

How to use the copy() method to fix this code

```
dfFixed = df.loc['0020900015',:].copy()
dfFixed.loc[:,'player_name'] = 'Curry'
```