2 Data wrangling

2.1 Contents

2.2 Introduction

This step focuses on collecting your data, organizing it, and making sure it's well defined. Paying attention to these tasks will pay off greatly later on. Some data cleaning can be done at this stage, but it's important not to be overzealous in your cleaning before you've explored the data to better understand it.

2.2.1 Recap Of Data Science Problem

The purpose of this data science project is to come up with a pricing model for ski resort tickets in our market segment. Big Mountain suspects it may not be maximizing its returns, relative to its position in the market. It also does not have a strong sense of what facilities matter most to visitors, particularly which ones they're most likely to pay more for. This project aims to build a predictive model for ticket price based on a number of facilities, or properties, boasted by resorts (*at the resorts).* This model will be used to provide guidance for Big Mountain's pricing and future facility investment plans.

2.2.2 Introduction To Notebook

Notebooks grow organically as we explore our data. If you used paper notebooks, you could discover a mistake and cross out or revise some earlier work. Later work may give you a reason to revisit earlier work and explore it further. The great thing about Jupyter notebooks is that you can edit, add, and move cells around without needing to cross out figures or scrawl in the margin. However, this means you can lose track of your changes easily. If you worked in a regulated environment, the company may have a a policy of always dating entries and clearly crossing out any mistakes, with your initials and the date.

Best practice here is to commit your changes using a version control system such as Git. Try to get into the habit of adding and committing your files to the Git repository you're working in after you save them. You're are working in a Git repository, right? If you make a significant change, save the notebook and commit it to Git. In fact, if you're about to make a significant change, it's a good idea to commit before as well. Then if the change is a mess, you've got the previous version to go back to.

Another best practice with notebooks is to try to keep them organized with helpful headings and comments. Not only can a good structure, but associated headings help you keep track of what you've done and your current focus. Anyone reading your notebook will have a much easier time following the flow of work. Remember, that 'anyone' will most likely be you. Be kind to future you!

In this notebook, note how we try to use well structured, helpful headings that frequently are self-explanatory, and we make a brief note after any results to highlight key takeaways. This is an immense help to anyone reading your notebook and it will greatly help you when you come to summarise your findings. **Top tip: jot down key findings in a final summary at the end of the notebook as they arise. You can tidy this up later.** This is a great way to ensure important results don't get lost in the middle of your notebooks.

In this, and subsequent notebooks, there are coding tasks marked with `#Code task n#` with code to complete. The `___` will guide you to where you need to insert code.

2.3 Imports

Placing your imports all together at the start of your notebook means you only need to consult one place to check your notebook's dependencies. By all means import something 'in situ' later on when you're experimenting, but if the imported dependency ends up being kept, you should subsequently move the import statement here with the rest.

In [1]:
```
#Code task 1#
#Import pandas, matplotlib.pyplot, and seaborn in the correct lines belo
w
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

In [2]:
```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

2.4 Objectives

There are some fundamental questions to resolve in this notebook before you move on.

- Do you think you may have the data you need to tackle the desired question?
  - Have you identified the required target value?
  - Do you have potentially useful features?
- Do you have any fundamental issues with the data?

2.5 Load The Ski Resort Data

In [3]:
```
# the supplied CSV data file is the raw_data directory
ski_data = pd.read_csv('../raw_data/ski_resort_data.csv')
```

Good first steps in auditing the data are the info method and displaying the first few records with head.

In [4]:
```
#Code task 2#
#Call the info method on ski_data to see a summary of the data
ski_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 330 entries, 0 to 329
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Name               330 non-null    object 
 1   Region             330 non-null    object 
 2   state              330 non-null    object 
 3   summit_elev        330 non-null    int64  
 4   vertical_drop      330 non-null    int64  
 5   base_elev          330 non-null    int64  
 6   trams              330 non-null    int64  
 7   fastEight          164 non-null    float64
 8   fastSixes          330 non-null    int64  
 9   fastQuads          330 non-null    int64  
 10  quad               330 non-null    int64  
 11  triple             330 non-null    int64  
 12  double             330 non-null    int64  
 13  surface            330 non-null    int64  
 14  total_chairs       330 non-null    int64  
 15  Runs               326 non-null    float64
 16  TerrainParks       279 non-null    float64
 17  LongestRun_mi      325 non-null    float64
 18  SkiableTerrain_ac  327 non-null    float64
 19  Snow Making_ac     284 non-null    float64
 20  daysOpenLastYear   279 non-null    float64
 21  yearsOpen          329 non-null    float64
 22  averageSnowfall    316 non-null    float64
 23  AdultWeekday       276 non-null    float64
 24  AdultWeekend       279 non-null    float64
 25  projectedDaysOpen  283 non-null    float64
 26  NightSkiing_ac     187 non-null    float64
dtypes: float64(13), int64(11), object(3)
memory usage: 69.7+ KB
```

Nothing too strange. Pretty straightforward; however, need inspection of 'fastEight' because it has 164 non-null values. Why? also, NightSkiing @ 187 non-nulls. Why is fast8 a float and not int? Check data type for apprpriate feature. Some of them don't make sense lol.

`AdultWeekday` is the price of an adult weekday ticket. `AdultWeekend` is the price of an adult weekend ticket. The other columns are potential features.

This immediately raises the question of what quantity will you want to model? You know you want to model the ticket price, but you realise there are two kinds of ticket price!

In [5]:
```
#Code task 3#
#Call the head method on ski_data to print the first several rows of the data
ski_data.head()
```

Out[5]:

	Name	Region	state	summit_elev	vertical_drop	base_elev	trams	fastEight	fastSixes
0	Alyeska Resort	Alaska	Alaska	3939	2500	250	1	0.0	0
1	Eaglecrest Ski Area	Alaska	Alaska	2600	1540	1200	0	0.0	0
2	Hilltop Ski Area	Alaska	Alaska	2090	294	1796	0	0.0	0
3	Arizona Snowbowl	Arizona	Arizona	11500	2300	9200	0	0.0	1
4	Sunrise Park Resort	Arizona	Arizona	11100	1800	9200	0	NaN	0

Notes: Whats a 'trams' column? find range on the summits, vert drop and height in general of the mountains; prob research vail; look at the regions in the US of the market. What's the deal with fastEight? What do they mean? Lot of good data to explore here! :) So far this is what I need to do:

```
* change features to appropriate dtypes
* investigate NaNs (keep in mind, .sum() ignores these?)
* research ski definitions
* geography of parks in the US
* VAIL as comparison
```

The output above suggests you've made a good start getting the ski resort data organized. You have plausible column headings. You can already see you have a missing value in the `fastEight` column

2.6 Explore The Data

2.6.1 Find Your Resort Of Interest

Your resort of interest is called Big Mountain Resort. Check it's in the data:

In [6]:
```
ski_data.head().T
```

Out[6]:

	0	1	2	3	4
Name	Alyeska Resort	Eaglecrest Ski Area	Hilltop Ski Area	Arizona Snowbowl	Sunrise Park Resort
Region	Alaska	Alaska	Alaska	Arizona	Arizona
state	Alaska	Alaska	Alaska	Arizona	Arizona
summit_elev	3939	2600	2090	11500	11100
vertical_drop	2500	1540	294	2300	1800
base_elev	250	1200	1796	9200	9200
trams	1	0	0	0	0
fastEight	0	0	0	0	NaN
fastSixes	0	0	0	1	0
fastQuads	2	0	0	0	1
quad	2	0	0	2	2
triple	0	0	1	2	3
double	0	4	0	1	1
surface	2	0	2	2	0
total_chairs	7	4	3	8	7
Runs	76	36	13	55	65
TerrainParks	2	1	1	4	2
LongestRun_mi	1	2	1	2	1.2
SkiableTerrain_ac	1610	640	30	777	800
Snow Making_ac	113	60	30	104	80
daysOpenLastYear	150	45	150	122	115
yearsOpen	60	44	36	81	49
averageSnowfall	669	350	69	260	250
AdultWeekday	65	47	30	89	74
AdultWeekend	85	53	34	89	78
projectedDaysOpen	150	90	152	122	104
NightSkiing_ac	550	NaN	30	NaN	80

In [7]:

```
#Code task 4#
#Filter the ski_data dataframe to display just the row for our resort wi
th the name 'Big Mountain Resort'
#Hint: you will find that the transpose of the row will give a nicer out
put. DataFrame's do have a
#transpose method, but you can access this conveniently with the `T` pro
perty.
ski_data[ski_data.Name == 'Big Mountain Resort'].T
```

Out[7]:

	151
Name	Big Mountain Resort
Region	Montana
state	Montana
summit_elev	6817
vertical_drop	2353
base_elev	4464
trams	0
fastEight	0
fastSixes	0
fastQuads	3
quad	2
triple	6
double	0
surface	3
total_chairs	14
Runs	105
TerrainParks	4
LongestRun_mi	3.3
SkiableTerrain_ac	3000
Snow Making_ac	600
daysOpenLastYear	123
yearsOpen	72
averageSnowfall	333
AdultWeekday	81
AdultWeekend	81
projectedDaysOpen	123
NightSkiing_ac	600

Notes: Our col # is 151

It's good that your resort doesn't appear to have any missing values.

2.6.2 Number Of Missing Values By Column

Count the number of missing values in each column and sort them.

In [8]:
```
ski_data.head(3).T
```

Out[8]:

	0	1	2
Name	Alyeska Resort	Eaglecrest Ski Area	Hilltop Ski Area
Region	Alaska	Alaska	Alaska
state	Alaska	Alaska	Alaska
summit_elev	3939	2600	2090
vertical_drop	2500	1540	294
base_elev	250	1200	1796
trams	1	0	0
fastEight	0	0	0
fastSixes	0	0	0
fastQuads	2	0	0
quad	2	0	0
triple	0	0	1
double	0	4	0
surface	2	0	2
total_chairs	7	4	3
Runs	76	36	13
TerrainParks	2	1	1
LongestRun_mi	1	2	1
SkiableTerrain_ac	1610	640	30
Snow Making_ac	113	60	30
daysOpenLastYear	150	45	150
yearsOpen	60	44	36
averageSnowfall	669	350	69
AdultWeekday	65	47	30
AdultWeekend	85	53	34
projectedDaysOpen	150	90	152
NightSkiing_ac	550	NaN	30

In [9]:
```
#Code task 5#
#Count (using `.sum()`) the number of missing values (`.isnull()`) in ea
ch column of
#ski_data as well as the percentages (using `.mean()` instead of `.sum()
`).
#Order them (increasing or decreasing) using sort_values
#Call `pd.concat` to present these in a single table (DataFrame) with th
e helpful column names 'count' and '%'
missing = pd.concat([ski_data.isnull().sum(), 100 * ski_data.isnull().me
an()], axis=1)
missing.columns=['count', '%']
missing.sort_values(by='count',ascending=False)
```

Out[9]:

	count	%
fastEight	166	50.303030
NightSkiing_ac	143	43.333333
AdultWeekday	54	16.363636
AdultWeekend	51	15.454545
daysOpenLastYear	51	15.454545
TerrainParks	51	15.454545
projectedDaysOpen	47	14.242424
Snow Making_ac	46	13.939394
averageSnowfall	14	4.242424
LongestRun_mi	5	1.515152
Runs	4	1.212121
SkiableTerrain_ac	3	0.909091
yearsOpen	1	0.303030
total_chairs	0	0.000000
Name	0	0.000000
Region	0	0.000000
double	0	0.000000
triple	0	0.000000
quad	0	0.000000
fastQuads	0	0.000000
fastSixes	0	0.000000
trams	0	0.000000
base_elev	0	0.000000
vertical_drop	0	0.000000
summit_elev	0	0.000000
state	0	0.000000
surface	0	0.000000

> **Note:** `fast3ight and nightskiing_ac` are the obvious problems. The But there's also significant ones like the 15%. And what's up with the discrepancy between 54 adultweekday v weekend? Maybe something was double counted? Look into this. Ask, "what'd a number of null where it's okay to ignore?"

`fastEight` has the most missing values, at just over 50%. Unfortunately, you see you're also missing quite a few of your desired target quantity, the ticket price, which is missing 15-16% of values. `AdultWeekday` is missing in a few more records than `AdultWeekend`. What overlap is there in these missing values? This is a question you'll want to investigate. You should also point out that `isnull()` is not the only indicator of missing data. Sometimes 'missingness' can be encoded, perhaps by a -1 or 999. Such values are typically chosen because they are "obviously" not genuine values. If you were capturing data on people's heights and weights but missing someone's height, you could certainly encode that as a 0 because no one has a height of zero (in any units). Yet such entries would not be revealed by `isnull()`. Here, you need a data dictionary and/or to spot such values as part of looking for outliers. Someone with a height of zero should definitely show up as an outlier!

2.6.3 Categorical Features

So far you've examined only the numeric features. Now you inspect categorical ones such as resort name and state. These are discrete entities. 'Alaska' is a name. Although names can be sorted alphabetically, it makes no sense to take the average of 'Alaska' and 'Arizona'. Similarly, 'Alaska' is before 'Arizona' only lexicographically; it is neither 'less than' nor 'greater than' 'Arizona'. As such, they tend to require different handling than strictly numeric quantities.

> **Note, a feature *can* be numeric but also categorical.** For example, instead of giving the number of `fastEight` lifts, a feature might be `has_fastEights` and have the value 0 or 1 to denote absence or presence of such a lift. In such a case it would not make sense to take an average of this or perform other mathematical calculations on it. Although you digress a little to make a point, month numbers are also, strictly speaking, categorical features. Yes, when a month is represented by its number (1 for January, 2 for Februrary etc.) it provides a convenient way to graph trends over a year. And, arguably, there is some logical interpretation of the average of 1 and 3 (January and March) being 2 (February). However, clearly December of one years precedes January of the next and yet 12 as a number is not less than 1.

The numeric quantities in the section above are truly numeric; they are the number of feet in the drop, or acres or years open or the amount of snowfall etc.

In [10]:

```
#Code task 6#
#Use ski_data's `select_dtypes` method to select columns of dtype 'objec
t'
ski_data.select_dtypes('object')
```

Out[10]:

	Name	Region	state
0	Alyeska Resort	Alaska	Alaska
1	Eaglecrest Ski Area	Alaska	Alaska
2	Hilltop Ski Area	Alaska	Alaska
3	Arizona Snowbowl	Arizona	Arizona
4	Sunrise Park Resort	Arizona	Arizona
5	Yosemite Ski & Snowboard Area	Northern California	California
6	Bear Mountain	Sierra Nevada	California
7	Bear Valley	Sierra Nevada	California
8	Boreal Mountain Resort	Sierra Nevada	California
9	Dodge Ridge	Sierra Nevada	California
10	Donner Ski Ranch	Sierra Nevada	California
11	Heavenly Mountain Resort	Sierra Nevada	California
12	June Mountain	Sierra Nevada	California
13	Kirkwood	Sierra Nevada	California
14	Mammoth Mountain Ski Area	Sierra Nevada	California
15	Mt. Shasta Ski Park	Sierra Nevada	California
16	Mountain High	Sierra Nevada	California
17	Mt. Baldy	Sierra Nevada	California
18	Northstar California	Sierra Nevada	California
19	Sierra-at-Tahoe	Sierra Nevada	California
20	Ski China Peak	Sierra Nevada	California
21	Snow Summit	Sierra Nevada	California
22	Snow Valley	Sierra Nevada	California
23	Soda Springs	Sierra Nevada	California
24	Sugar Bowl Resort	Sierra Nevada	California
25	Tahoe Donner	Sierra Nevada	California
26	Arapahoe Basin Ski Area	Colorado	Colorado
27	Aspen / Snowmass	Colorado	Colorado
28	Beaver Creek	Colorado	Colorado
29	Breckenridge	Colorado	Colorado
30	Copper Mountain Resort	Colorado	Colorado
31	Crested Butte Mountain Resort	Colorado	Colorado
32	Purgatory	Colorado	Colorado
33	Eldora Mountain Resort	Colorado	Colorado

	Name	Region	state
34	Howelsen Hill	Colorado	Colorado
35	Keystone	Colorado	Colorado
36	Loveland	Colorado	Colorado
37	Monarch Mountain	Colorado	Colorado
38	Powderhorn	Colorado	Colorado
39	Silverton Mountain	Colorado	Colorado
40	Cooper	Colorado	Colorado
41	Ski Granby Ranch	Colorado	Colorado
42	Steamboat	Colorado	Colorado
43	Sunlight Mountain Resort	Colorado	Colorado
44	Telluride	Colorado	Colorado
45	Vail	Colorado	Colorado
46	Winter Park Resort	Colorado	Colorado
47	Wolf Creek Ski Area	Colorado	Colorado
48	Mohawk Mountain	Connecticut	Connecticut
49	Mount Southington Ski Area	Connecticut	Connecticut
50	Powder Ridge Park	Connecticut	Connecticut
51	Ski Sundown	Connecticut	Connecticut
52	Woodbury Ski Area	Connecticut	Connecticut
53	Bogus Basin	Idaho	Idaho
54	Brundage Mountain Resort	Idaho	Idaho
55	Kelly Canyon Ski Area	Idaho	Idaho
56	Lookout Pass Ski Area	Idaho	Idaho
57	Magic Mountain Ski Area	Idaho	Idaho
58	Pebble Creek Ski Area	Idaho	Idaho
59	Pomerelle Mountain Resort	Idaho	Idaho
60	Schweitzer	Idaho	Idaho
61	Silver Mountain	Idaho	Idaho
62	Soldier Mountain Ski Area	Idaho	Idaho
63	Sun Valley	Idaho	Idaho
64	Tamarack Resort	Idaho	Idaho
65	Chestnut Mountain Resort	Illinois	Illinois
66	Four Lakes	Illinois	Illinois
67	Ski Snowstar Winter Sports Park	Illinois	Illinois
68	Villa Olivia	Illinois	Illinois

	Name	Region	state
69	Paoli Peaks	Indiana	Indiana
70	Perfect North Slopes	Indiana	Indiana
71	Mt. Crescent Ski Area	Iowa	Iowa
72	Seven Oaks	Iowa	Iowa
73	Sundown Mountain	Iowa	Iowa
74	Big Squaw Mountain Ski Resort	Maine	Maine
75	Camden Snow Bowl	Maine	Maine
76	Lost Valley	Maine	Maine
77	Mt. Abram Ski Resort	Maine	Maine
78	Mt. Jefferson	Maine	Maine
79	New Hermon Mountain	Maine	Maine
80	Shawnee Peak	Maine	Maine
81	Sugarloaf	Maine	Maine
82	Sunday River	Maine	Maine
83	Wisp	Maryland	Maryland
84	Berkshire East	Massachusetts	Massachusetts
85	Blandford Ski Area	Massachusetts	Massachusetts
86	Blue Hills Ski Area	Massachusetts	Massachusetts
87	Bousquet Ski Area	Massachusetts	Massachusetts
88	Bradford Ski Area	Massachusetts	Massachusetts
89	Jiminy Peak	Massachusetts	Massachusetts
90	Nashoba Valley	Massachusetts	Massachusetts
91	Otis Ridge Ski Area	Massachusetts	Massachusetts
92	Ski Butternut	Massachusetts	Massachusetts
93	Ski Ward	Massachusetts	Massachusetts
94	Wachusett Mountain Ski Area	Massachusetts	Massachusetts
95	Alpine Valley Ski Area	Michigan	Michigan
96	Apple Mountain	Michigan	Michigan
97	Big Powderhorn Mountain	Michigan	Michigan
98	Bittersweet Ski Area	Michigan	Michigan
99	Big Snow Resort - Blackjack	Michigan	Michigan
100	Boyne Highlands	Michigan	Michigan
101	Boyne Mountain Resort	Michigan	Michigan
102	Caberfae Peaks	Michigan	Michigan
103	Cannonsburg	Michigan	Michigan

	Name	Region	state
104	Crystal Mountain	Michigan	Michigan
105	Big Snow Resort - Indianhead Mountain	Michigan	Michigan
106	Marquette Mountain	Michigan	Michigan
107	Mont Ripley	Michigan	Michigan
108	Mount Bohemia	Michigan	Michigan
109	Mt. Brighton	Michigan	Michigan
110	Mt. Holiday Ski Area	Michigan	Michigan
111	Mount Holly	Michigan	Michigan
112	Mulligan's Hollow Ski Bowl	Michigan	Michigan
113	Norway Mountain	Michigan	Michigan
114	Nubs Nob Ski Area	Michigan	Michigan
115	Pine Knob Ski Resort	Michigan	Michigan
116	Pine Mountain	Michigan	Michigan
117	Schuss Mountain at Shanty Creek	Michigan	Michigan
118	Ski Brule	Michigan	Michigan
119	Snow Snake Mountain Ski Area	Michigan	Michigan
120	Swiss Valley	Michigan	Michigan
121	The Homestead	Michigan	Michigan
122	Timber Ridge	Michigan	Michigan
123	Treetops Resort	Michigan	Michigan
124	Afton Alps	Minnesota	Minnesota
125	Andes Tower Hills Ski Area	Minnesota	Minnesota
126	Buck Hill	Minnesota	Minnesota
127	Buena Vista Ski Area	Minnesota	Minnesota
128	Coffee Mill Ski & Snowboard Resort	Minnesota	Minnesota
129	Elm Creek Winter Recreation Area	Minnesota	Minnesota
130	Giants Ridge Resort	Minnesota	Minnesota
131	Hyland Ski & Snowboard Area	Minnesota	Minnesota
132	Lutsen Mountains	Minnesota	Minnesota
133	Mount Kato Ski Area	Minnesota	Minnesota
134	Powder Ridge Ski Area	Minnesota	Minnesota
135	Spirit Mountain	Minnesota	Minnesota
136	Welch Village	Minnesota	Minnesota
137	Wild Mountain Ski & Snowboard Area	Minnesota	Minnesota
138	Hidden Valley Ski Area	Missouri	Missouri

	Name	Region	state
139	Snow Creek	Missouri	Missouri
140	Big Sky Resort	Montana	Montana
141	Blacktail Mountain Ski Area	Montana	Montana
142	Bridger Bowl	Montana	Montana
143	Discovery Ski Area	Montana	Montana
144	Great Divide	Montana	Montana
145	Lost Trail - Powder Mtn	Montana	Montana
146	Maverick Mountain	Montana	Montana
147	Montana Snowbowl	Montana	Montana
148	Red Lodge Mountain	Montana	Montana
149	Showdown Montana	Montana	Montana
150	Teton Pass Ski Resort	Montana	Montana
151	Big Mountain Resort	Montana	Montana
152	Diamond Peak	Sierra Nevada	Nevada
153	Elko SnoBowl	Nevada	Nevada
154	Lee Canyon	Nevada	Nevada
155	Mt. Rose - Ski Tahoe	Sierra Nevada	Nevada
156	Attitash	New Hampshire	New Hampshire
157	Black Mountain	New Hampshire	New Hampshire
158	Bretton Woods	New Hampshire	New Hampshire
159	Cannon Mountain	New Hampshire	New Hampshire
160	Cranmore Mountain Resort	New Hampshire	New Hampshire
161	Crotched Mountain	New Hampshire	New Hampshire
162	Dartmouth Skiway	New Hampshire	New Hampshire
163	Gunstock	New Hampshire	New Hampshire
164	King Pine	New Hampshire	New Hampshire
165	Loon Mountain	New Hampshire	New Hampshire
166	Mount Sunapee	New Hampshire	New Hampshire
167	Pats Peak	New Hampshire	New Hampshire
168	Ragged Mountain Resort	New Hampshire	New Hampshire
169	Waterville Valley	New Hampshire	New Hampshire
170	Whaleback Mountain	New Hampshire	New Hampshire
171	Wildcat Mountain	New Hampshire	New Hampshire
172	Campgaw Mountain	New Jersey	New Jersey
173	Mountain Creek Resort	New Jersey	New Jersey

	Name	Region	state
174	Angel Fire Resort	New Mexico	New Mexico
175	Enchanted Forest Ski Area	New Mexico	New Mexico
176	Pajarito Mountain Ski Area	New Mexico	New Mexico
177	Red River	New Mexico	New Mexico
178	Sandia Peak	New Mexico	New Mexico
179	Sipapu Ski Resort	New Mexico	New Mexico
180	Ski Apache	New Mexico	New Mexico
181	Ski Santa Fe	New Mexico	New Mexico
182	Taos Ski Valley	New Mexico	New Mexico
183	Belleayre	New York	New York
184	Brantling Ski Slopes	New York	New York
185	Bristol Mountain	New York	New York
186	Buffalo Ski Club Ski Area	New York	New York
187	Catamount	New York	New York
188	Dry Hill Ski Area	New York	New York
189	Gore Mountain	New York	New York
190	Greek Peak	New York	New York
191	Holiday Mountain	New York	New York
192	Holiday Valley	New York	New York
193	Holimont Ski Area	New York	New York
194	Hunt Hollow Ski Club	New York	New York
195	Hunter Mountain	New York	New York
196	Kissing Bridge	New York	New York
197	Labrador Mt.	New York	New York
198	Maple Ski Ridge	New York	New York
199	McCauley Mountain Ski Center	New York	New York
200	Mount Peter Ski Area	New York	New York
201	Oak Mountain	New York	New York
202	Peek'n Peak	New York	New York
203	Plattekill Mountain	New York	New York
204	Royal Mountain Ski Area	New York	New York
205	Snow Ridge	New York	New York
206	Song Mountain	New York	New York
207	Swain	New York	New York
208	Thunder Ridge	New York	New York

	Name	Region	state
209	Titus Mountain	New York	New York
210	Toggenburg Mountain	New York	New York
211	West Mountain	New York	New York
212	Whiteface Mountain Resort	New York	New York
213	Willard Mountain	New York	New York
214	Windham Mountain	New York	New York
215	Woods Valley Ski Area	New York	New York
216	Appalachian Ski Mountain	North Carolina	North Carolina
217	Cataloochee Ski Area	North Carolina	North Carolina
218	Sapphire Valley	North Carolina	North Carolina
219	Beech Mountain Resort	North Carolina	North Carolina
220	Sugar Mountain Resort	North Carolina	North Carolina
221	Wolf Ridge Ski Resort	North Carolina	North Carolina
222	Alpine Valley	Ohio	Ohio
223	Boston Mills	Ohio	Ohio
224	Brandywine	Ohio	Ohio
225	Mad River Mountain	Ohio	Ohio
226	Snow Trails	Ohio	Ohio
227	Anthony Lakes Mountain Resort	Oregon	Oregon
228	Cooper Spur	Mt. Hood	Oregon
229	Hoodoo Ski Area	Oregon	Oregon
230	Mt. Ashland	Oregon	Oregon
231	Mt. Bachelor	Oregon	Oregon
232	Mt. Hood Meadows	Mt. Hood	Oregon
233	Mt. Hood Skibowl	Mt. Hood	Oregon
234	Spout Springs	Oregon	Oregon
235	Timberline Lodge	Mt. Hood	Oregon
236	Willamette Pass	Oregon	Oregon
237	Bear Creek Mountain Resort	Pennsylvania	Pennsylvania
238	Ski Big Bear	Pennsylvania	Pennsylvania
239	Big Boulder	Pennsylvania	Pennsylvania
240	Blue Knob	Pennsylvania	Pennsylvania
241	Blue Mountain Resort	Pennsylvania	Pennsylvania
242	Camelback Mountain Resort	Pennsylvania	Pennsylvania
243	Eagle Rock	Pennsylvania	Pennsylvania

	Name	Region	state
244	Elk Mountain Ski Resort	Pennsylvania	Pennsylvania
245	Jack Frost	Pennsylvania	Pennsylvania
246	Liberty	Pennsylvania	Pennsylvania
247	Mount Pleasant of Edinboro	Pennsylvania	Pennsylvania
248	Roundtop Mountain Resort	Pennsylvania	Pennsylvania
249	Seven Springs	Pennsylvania	Pennsylvania
250	Shawnee Mountain Ski Area	Pennsylvania	Pennsylvania
251	Ski Sawmill	Pennsylvania	Pennsylvania
252	Montage Mountain	Pennsylvania	Pennsylvania
253	Spring Mountain Ski Area	Pennsylvania	Pennsylvania
254	Tussey Mountain	Pennsylvania	Pennsylvania
255	Whitetail Resort	Pennsylvania	Pennsylvania
256	Yawgoo Valley	Rhode Island	Rhode Island
257	Deer Mountain Ski Resort	South Dakota	South Dakota
258	Terry Peak Ski Area	South Dakota	South Dakota
259	Ober Gatlinburg Ski Resort	Tennessee	Tennessee
260	Alta Ski Area	Salt Lake City	Utah
261	Beaver Mountain	Utah	Utah
262	Brian Head Resort	Utah	Utah
263	Brighton Resort	Salt Lake City	Utah
264	Deer Valley Resort	Salt Lake City	Utah
265	Eagle Point	Utah	Utah
266	Park City	Salt Lake City	Utah
267	Powder Mountain	Utah	Utah
268	Snowbasin	Utah	Utah
269	Snowbird	Salt Lake City	Utah
270	Solitude Mountain Resort	Salt Lake City	Utah
271	Sundance	Utah	Utah
272	Nordic Valley Resort	Utah	Utah
273	Bolton Valley	Vermont	Vermont
274	Bromley Mountain	Vermont	Vermont
275	Burke Mountain	Vermont	Vermont
276	Jay Peak	Vermont	Vermont
277	Killington Resort	Vermont	Vermont
278	Mad River Glen	Vermont	Vermont

	Name	Region	state
279	Magic Mountain	Vermont	Vermont
280	Mount Snow	Vermont	Vermont
281	Okemo Mountain Resort	Vermont	Vermont
282	Pico Mountain	Vermont	Vermont
283	Smugglers' Notch Resort	Vermont	Vermont
284	Stowe Mountain Resort	Vermont	Vermont
285	Stratton Mountain	Vermont	Vermont
286	Sugarbush	Vermont	Vermont
287	Suicide Six	Vermont	Vermont
288	Bryce Resort	Virginia	Virginia
289	Massanutten	Virginia	Virginia
290	The Homestead Ski Area	Virginia	Virginia
291	Wintergreen Resort	Virginia	Virginia
292	49 Degrees North	Washington	Washington
293	Alpental	Washington	Washington
294	Bluewood	Washington	Washington
295	Crystal Mountain	Washington	Washington
296	Mission Ridge	Washington	Washington
297	Mt. Baker	Washington	Washington
298	Mt. Spokane Ski and Snowboard Park	Washington	Washington
299	Stevens Pass Resort	Washington	Washington
300	The Summit at Snoqualmie	Washington	Washington
301	White Pass	Washington	Washington
302	Canaan Valley Resort	West Virginia	West Virginia
303	Snowshoe Mountain Resort	West Virginia	West Virginia
304	Timberline Four Seasons	West Virginia	West Virginia
305	Winterplace Ski Resort	West Virginia	West Virginia
306	Alpine Valley Resort	Wisconsin	Wisconsin
307	Bruce Mound	Wisconsin	Wisconsin
308	Cascade Mountain	Wisconsin	Wisconsin
309	Christie Mountain	Wisconsin	Wisconsin
310	Christmas Mountain	Wisconsin	Wisconsin
311	Devils Head	Wisconsin	Wisconsin
312	Grand Geneva	Wisconsin	Wisconsin
313	Granite Peak Ski Area	Wisconsin	Wisconsin

	Name	Region	state
314	Little Switzerland	Wisconsin	Wisconsin
315	Mount La Crosse	Wisconsin	Wisconsin
316	Nordic Mountain	Wisconsin	Wisconsin
317	Sunburst	Wisconsin	Wisconsin
318	Trollhaugen	Wisconsin	Wisconsin
319	Tyrol Basin	Wisconsin	Wisconsin
320	Whitecap Mountain	Wisconsin	Wisconsin
321	Wilmot Mountain	Wisconsin	Wisconsin
322	Grand Targhee Resort	Wyoming	Wyoming
323	Hogadon Basin	Wyoming	Wyoming
324	Jackson Hole	Wyoming	Wyoming
325	Meadowlark Ski Lodge	Wyoming	Wyoming
326	Sleeping Giant Ski Resort	Wyoming	Wyoming
327	Snow King Resort	Wyoming	Wyoming
328	Snowy Range Ski & Recreation Area	Wyoming	Wyoming
329	White Pine Ski Area	Wyoming	Wyoming

In [ ]:

You saw earlier on that these three columns had no missing values. But are there any other issues with these columns? Sensible questions to ask here include:

- Is `Name` (or at least a combination of Name/Region/State) unique?
- Is `Region` always the same as `state` ?

In [11]:
```
ski_data.Region.unique()
```

Out[11]:
```
array(['Alaska', 'Arizona', 'Northern California', 'Sierra Nevada',
       'Colorado', 'Connecticut', 'Idaho', 'Illinois', 'Indiana', 'Iow
a',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Missouri', 'Montana', 'Nevada', 'New Hampshire', 'New Jersey',
       'New Mexico', 'New York', 'North Carolina', 'Ohio', 'Oregon',
       'Mt. Hood', 'Pennsylvania', 'Rhode Island', 'South Dakota',
       'Tennessee', 'Salt Lake City', 'Utah', 'Vermont', 'Virginia',
       'Washington', 'West Virginia', 'Wisconsin', 'Wyoming'],
      dtype=object)
```

Region and state values are not all unique. Region is not always the same as state. While state means state. Region according to this data can be a state, part of the state or a city even.

2.6.3.1 Unique Resort Names

In [12]:
```
#Code task 7#
#Use pandas' Series method `value_counts` to find any duplicated resort
 names
ski_data['Name'].value_counts().head()
```

Out[12]:
```
Crystal Mountain             2
Devils Head                  1
Hidden Valley Ski Area       1
Loon Mountain                1
The Summit at Snoqualmie     1
Name: Name, dtype: int64
```

You have a duplicated resort name: Crystal Mountain.

Q1: Is this resort duplicated if you take into account Region and/or state as well?

In [13]:
```
#Code task 8#
#Concatenate the string columns 'Name' and 'Region' and count the values
again (as above)

(ski_data['Name'] + ', ' + ski_data['Region']).value_counts().head()
```

Out[13]:
```
Alta Ski Area, Salt Lake City        1
Hilltop Ski Area, Alaska             1
Devils Head, Wisconsin               1
Royal Mountain Ski Area, New York    1
Ski Butternut, Massachusetts         1
dtype: int64
```

In [14]:
```
#Code task 9#
#Concatenate 'Name' and 'state' and count the values again (as above)

(ski_data['Name'] + ', ' + ski_data['state']).value_counts().head()
```

Out[14]:
```
Mount Pleasant of Edinboro, Pennsylvania    1
49 Degrees North, Washington                1
Cranmore Mountain Resort, New Hampshire     1
Hilltop Ski Area, Alaska                    1
Dodge Ridge, California                     1
dtype: int64
```

NB because you know `value_counts()` sorts descending, you can use the `head()` method and know the rest of the counts must be 1.

> **A1:** No, Crystal Mountain resort is not duplicated once you search for the (Name + Region) or (Name + state), which means there are two resorts in two different states.

In [15]:
```
ski_data[ski_data['Name'] == 'Crystal Mountain']
```

Out[15]:

	Name	Region	state	summit_elev	vertical_drop	base_elev	trams	fastEight	f
104	Crystal Mountain	Michigan	Michigan	1132	375	757	0	0.0	
295	Crystal Mountain	Washington	Washington	7012	3100	4400	1	NaN	

So there are two Crystal Mountain resorts, but they are clearly two different resorts in two different states. This is a powerful signal that you have unique records on each row.

2.6.3.2 Region And State

What's the relationship between region and state?

You know they are the same in many cases (e.g. both the Region and the state are given as 'Michigan'). In how many cases do they differ?

In [16]:
```
#Code task 10#
#Calculate the number of times Region does not equal state
(ski_data.Region != ski_data.state).value_counts()
```

Out[16]:
```
False    297
True      33
dtype: int64
```

> **Note:** Of our 330 resorts, 33 (or 10%) of the data have 'Region' and 'state' as the same value.

You know what a state is. What is a region? You can tabulate the distinct values along with their respective frequencies using `value_counts()` .

In [17]:
```
ski_data['Region'].value_counts()
```

Out[17]:
```
New York               33
Michigan               29
Sierra Nevada          22
Colorado               22
Pennsylvania           19
New Hampshire          16
Wisconsin              16
Vermont                15
Minnesota              14
Montana                12
Idaho                  12
Massachusetts          11
Washington             10
Maine                   9
New Mexico              9
Wyoming                 8
Utah                    7
Salt Lake City          6
Oregon                  6
North Carolina          6
Ohio                    5
Connecticut             5
West Virginia           4
Illinois                4
Mt. Hood                4
Virginia                4
Iowa                    3
Alaska                  3
Arizona                 2
Missouri                2
Nevada                  2
New Jersey              2
South Dakota            2
Indiana                 2
Maryland                1
Northern California     1
Tennessee               1
Rhode Island            1
Name: Region, dtype: int64
```

A casual inspection by eye reveals some non-state names such as Sierra Nevada, Salt Lake City, and Northern California. Tabulate the differences between Region and state. On a note regarding scaling to larger data sets, you might wonder how you could spot such cases when presented with millions of rows. This is an interesting point. Imagine you have access to a database with a Region and state column in a table and there are millions of rows. You wouldn't eyeball all the rows looking for differences!

Bear in mind that our first interest lies in establishing the answer to the question "Are they always the same?"

- One approach might be to ask the database to return records where they differ, but limit the output to 10 rows. If there were differences, you'd only get up to 10 results, and so you wouldn't know whether you'd located all differences, but you'd know that there were 'a nonzero number' of differences.
- If you got an empty result set back, then you would know that the two columns always had the same value. At the risk of digressing, some values in one column only might be NULL (missing) and different databases treat NULL differently, so be aware that on many an occasion a seamingly 'simple' question gets very interesting to answer very quickly!

In [18]:
```
len(ski_data.columns)
```

Out[18]:
```
27
```

In [19]:
```
#Code task 11#
#Filter the ski_data dataframe for rows where 'Region' and 'state' are d
ifferent,
#group that by 'state' and perform `value_counts` on the 'Region'
(ski_data[ski_data['Region'] != ski_data['state']]
           .groupby('state')
            ['Region'].value_counts())
```

Out[19]:
```
state       Region
California  Sierra Nevada          20
            Northern California     1
Nevada      Sierra Nevada           2
Oregon      Mt. Hood                4
Utah        Salt Lake City          6
Name: Region, dtype: int64
```

In [ ]:

In [ ]:

The vast majority of the differences are in California, with most Regions being called Sierra Nevada and just one referred to as Northern California.

2.6.3.3 Number of distinct regions and states

In [20]:
```
#Code task 12#
#Select the 'Region' and 'state' columns from ski_data and use the `nuni
que` method to calculate
#the number of unique values in each
ski_data[['Region', 'state']].nunique()
```

Out[20]:
```
Region    38
state     35
dtype: int64
```

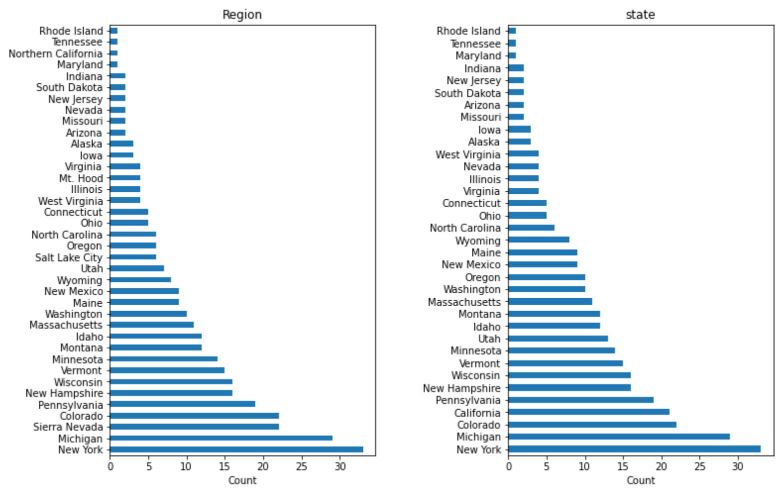
> **Note** .nunique() is for dataframes!

Because a few states are split across multiple named regions, there are slightly more unique regions than states.

2.6.3.4 Distribution Of Resorts By Region And State

If this is your first time using [matplotlib (https://matplotlib.org/3.2.2/index.html)](https://matplotlib.org/3.2.2/index.html)'s [subplots (https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.subplots.html)](https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.subplots.html), you may find the online documentation useful.

In [21]:
```
#Code task 13#
#Create two subplots on 1 row and 2 columns with a figsize of (12, 8)
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12,8))


#Specify a horizontal barplot ('barh') as kind of plot (kind=)
ski_data.Region.value_counts().plot(kind='barh', ax=ax[0])



#Give the plot a helpful title of 'Region'
#Label the xaxis 'Count'
ax[0].set_title('Region')
ax[0].set_xlabel('Count')

#Specify a horizontal barplot ('barh') as kind of plot (kind=)
ski_data.state.value_counts().plot(kind='barh', ax=ax[1])

#Give the plot a helpful title of 'state'
#Label the xaxis 'Count'
ax[1].set_title('state')
ax[1].set_xlabel('Count')


#Give the subplots a little "breathing room" with a wspace of 0.5
plt.subplots_adjust(wspace=0.5);



#You're encouraged to explore a few different figure sizes, orientation
s, and spacing here
# as the importance of easy-to-read and informative figures is frequentl
y understated
# and you will find the ability to tweak figures invaluable later on
```



How's your geography? Looking at the distribution of States, you see New York accounting for the majority of resorts. Our target resort is in Montana, which comes in at 13th place.

You should think carefully about how, or whether, you use this information.

- Does New York command a premium because of its proximity to population?
- Even if a resort's State were a useful predictor of ticket price, your main interest lies in Montana. Would you want a model that is skewed for accuracy by New York?
- Should you just filter for Montana and create a Montana-specific model? This would slash your available data volume.

> **Your problem task includes the contextual insight that the data are for resorts all belonging to the same market share.**

This suggests one might expect prices to be similar amongst them. You can look into this. A boxplot grouped by State is an ideal way to quickly compare prices. Another side note worth bringing up here is that, **in reality, the best approach here definitely would include consulting with the client or other domain expert.** They might know of good reasons for treating states equivalently or differently. The data scientist is rarely the final arbiter of such a decision. But here, you'll see if we can find any supporting evidence for treating states the same or differently.

2.6.3.5 Distribution Of Ticket Price By State

Our primary focus is our Big Mountain resort, in Montana. Does the state give you any clues to help decide what your primary target response feature should be (weekend or weekday ticket prices)?

2.6.3.5.1 Average weekend and weekday price by state

In [22]:
```
ski_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 330 entries, 0 to 329
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Name               330 non-null    object 
 1   Region             330 non-null    object 
 2   state              330 non-null    object 
 3   summit_elev        330 non-null    int64  
 4   vertical_drop      330 non-null    int64  
 5   base_elev          330 non-null    int64  
 6   trams              330 non-null    int64  
 7   fastEight          164 non-null    float64
 8   fastSixes          330 non-null    int64  
 9   fastQuads          330 non-null    int64  
 10  quad               330 non-null    int64  
 11  triple             330 non-null    int64  
 12  double             330 non-null    int64  
 13  surface            330 non-null    int64  
 14  total_chairs       330 non-null    int64  
 15  Runs               326 non-null    float64
 16  TerrainParks       279 non-null    float64
 17  LongestRun_mi      325 non-null    float64
 18  SkiableTerrain_ac  327 non-null    float64
 19  Snow Making_ac     284 non-null    float64
 20  daysOpenLastYear   279 non-null    float64
 21  yearsOpen          329 non-null    float64
 22  averageSnowfall    316 non-null    float64
 23  AdultWeekday       276 non-null    float64
 24  AdultWeekend       279 non-null    float64
 25  projectedDaysOpen  283 non-null    float64
 26  NightSkiing_ac     187 non-null    float64
dtypes: float64(13), int64(11), object(3)
memory usage: 69.7+ KB
```

In [23]:

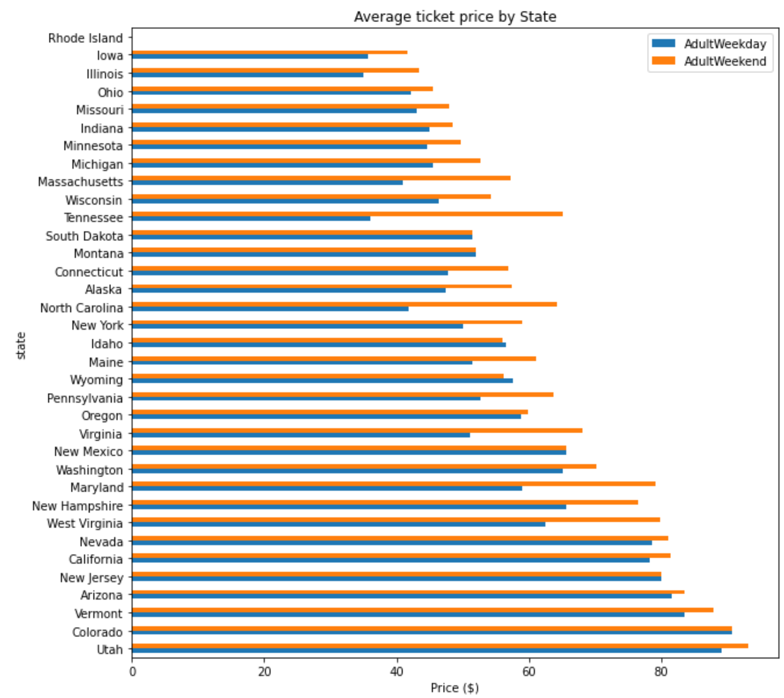
```
#Code task 14#
# Calculate average weekday and weekend price by state and sort by the a
verage of the two
# Hint: use the pattern dataframe.groupby(<grouping variable>)[<list of
 columns>].mean()
state_price_means = ski_data.groupby(by='state')[['AdultWeekday', 'Adult
Weekend']].mean()
state_price_means.head()
```

Out[23]:

	AdultWeekday	AdultWeekend
state		
Alaska	47.333333	57.333333
Arizona	81.500000	83.500000
California	78.214286	81.416667
Colorado	90.714286	90.714286
Connecticut	47.800000	56.800000

In [24]:

```
# The next bit simply reorders the index by increasing average of weekda
y and weekend prices
# Compare the index order you get from

# state_price_means.index v.

# state_price_means.mean(axis=1).sort_values(ascending=False).index

# See how this expression simply sits within the reindex()
(state_price_means.reindex(index=state_price_means.mean(axis=1)
    .sort_values(ascending=False)
    .index)
    .plot(kind='barh', figsize=(10, 10), title='Average ticket price by
 State'))
plt.xlabel('Price ($)');
```



The figure above represents a dataframe with two columns, one for the average prices of each kind of ticket. This tells you how the average ticket price varies from state to state. But can you get more insight into the difference in the distributions between states?

In [25]:
```
ski_data.loc[ski_data['state']=='Rhode Island',:]
```

Out[25]:

	Name	Region	state	summit_elev	vertical_drop	base_elev	trams	fastEight	fastSixes
256	Yawgoo Valley	Rhode Island	Rhode Island	315	245	70	0	NaN	0

2.6.3.5.2 Distribution of weekday and weekend price by state

Next, you can transform the data into a single column for price with a new categorical column that represents the ticket type.

In [26]:
```
#Code task 15#
#Use the pd.melt function, pass in the ski_data columns 'state', 'AdultW
eekday', and 'Adultweekend' only,
#specify 'state' for `id_vars`
#gather the ticket prices from the 'Adultweekday' and 'AdultWeekend' col
umns using the `value_vars` argument,
#call the resultant price column 'Price' via the `value_name` argument,
#name the weekday/weekend indicator column 'Ticket' via the `var_name` a
rgument
ticket_prices = pd.melt(ski_data[['state', 'AdultWeekday', 'AdultWeeken
d']],
                        id_vars='state',
                        value_vars=['AdultWeekday', 'AdultWeekend'],
                        var_name='Ticket',
                        value_name='Price')
```

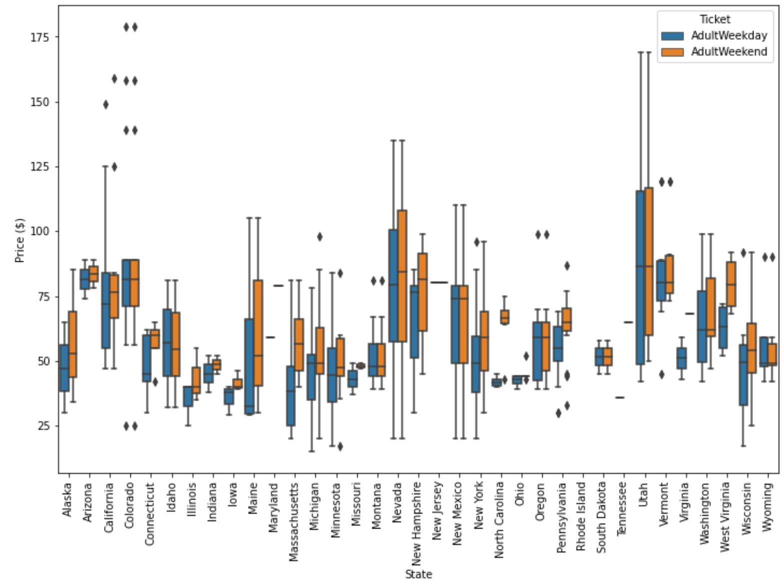
In [27]:
```
ticket_prices.head()
```

Out[27]:

	state	Ticket	Price
0	Alaska	AdultWeekday	65.0
1	Alaska	AdultWeekday	47.0
2	Alaska	AdultWeekday	30.0
3	Arizona	AdultWeekday	89.0
4	Arizona	AdultWeekday	74.0

This is now in a format we can pass to seaborn (https://seaborn.pydata.org/)'s boxplot (https://seaborn.pydata.org/generated/seaborn.boxplot.html) function to create boxplots of the ticket price distributions for each ticket type for each state.

In [28]:
```
#Code task 16#
#Create a seaborn boxplot of the ticket price dataframe we created abov
e,
#with 'state' on the x-axis, 'Price' as the y-value, and a hue that indi
cates 'Ticket'
#This will use boxplot's x, y, hue, and data arguments.

plt.subplots(figsize=(12, 8))
sns.boxplot(x='state', y='Price', hue='Ticket', data=ticket_prices)
plt.xticks(rotation='vertical')
plt.ylabel('Price ($)')
plt.xlabel('State');
```



> **Notes:** Things to look for:
>
> - outliers
> - range
> - top 3-5 states
> - your state of interest (Montana)
> - max/min
> - weird patterns
> - variability
>
> Of all the states, Oregon, South Dakota, and Wyoming are the most comparable states to Montana based on price ranges and mean. This makes sense because these areas are in the same geography in the US.

Aside from some relatively expensive ticket prices in California, Colorado, and Utah, most prices appear to lie in a broad band from around 25 to over 100 dollars. Some States show more **variability** than others. Montana and South Dakota, for example, both show fairly small variability as well as matching weekend and weekday ticket prices. Nevada and Utah, on the other hand, show the most range in prices. Some States, notably North Carolina and Virginia, have weekend prices far higher than weekday prices.

> You could be inspired from this exploration to **consider a few potential groupings of resorts**, those with low spread, those with lower averages, and those that charge a premium for weekend tickets.

However, you're told that you are taking all resorts to be part of the same market share, you could argue against further segment the resorts. Nevertheless, ways to consider using the State information in your modelling include:

1. disregard State completely
2. retain all State information
3. retain State in the form of Montana vs not Montana, as our target resort is in Montana

> You've also noted another effect above: some States show a marked difference between weekday and weekend ticket prices. It may **make sense to allow a model to take into account not just State but also weekend vs weekday.**

Thus we currently have two main questions you want to resolve:

- What do you do about the two types of ticket price?
- What do you do about the state information?

2.6.4 Numeric Features

Having decided to reserve judgement on how exactly you utilize the State, turn your attention to cleaning the numeric features.

2.6.4.1 Numeric data summary

In [29]:
```
#Code task 17#
#Call ski_data's `describe` method for a statistical summary of the numerical columns
#Hint: there are fewer summary stat columns than features, so displaying the transpose
#will be useful again
ski_data.select_dtypes(include=['float64', 'int64']).describe().T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	max
summit_elev	330.0	4591.818182	3735.535934	315.0	1403.75	3127.5	7806.00	13487.0
vertical_drop	330.0	1215.427273	947.864557	60.0	461.25	964.5	1800.00	4425.0
base_elev	330.0	3374.000000	3117.121621	70.0	869.00	1561.5	6325.25	10800.0
trams	330.0	0.172727	0.559946	0.0	0.00	0.0	0.00	4.0
fastEight	164.0	0.006098	0.078087	0.0	0.00	0.0	0.00	1.0
fastSixes	330.0	0.184848	0.651685	0.0	0.00	0.0	0.00	6.0
fastQuads	330.0	1.018182	2.198294	0.0	0.00	0.0	1.00	15.0
quad	330.0	0.933333	1.312245	0.0	0.00	0.0	1.00	8.0
triple	330.0	1.500000	1.619130	0.0	0.00	1.0	2.00	8.0
double	330.0	1.833333	1.815028	0.0	1.00	1.0	3.00	14.0
surface	330.0	2.621212	2.059636	0.0	1.00	2.0	3.00	15.0
total_chairs	330.0	8.266667	5.798683	0.0	5.00	7.0	10.00	41.0
Runs	326.0	48.214724	46.364077	3.0	19.00	33.0	60.00	341.0
TerrainParks	279.0	2.820789	2.008113	1.0	1.00	2.0	4.00	14.0
LongestRun_mi	325.0	1.433231	1.156171	0.0	0.50	1.0	2.00	6.0
SkiableTerrain_ac	327.0	739.801223	1816.167441	8.0	85.00	200.0	690.00	26819.0
Snow Making_ac	284.0	174.873239	261.336125	2.0	50.00	100.0	200.50	3379.0
daysOpenLastYear	279.0	115.103943	35.063251	3.0	97.00	114.0	135.00	305.0
yearsOpen	329.0	63.656535	109.429928	6.0	50.00	58.0	69.00	2019.0
averageSnowfall	316.0	185.316456	136.356842	18.0	69.00	150.0	300.00	669.0
AdultWeekday	276.0	57.916957	26.140126	15.0	40.00	50.0	71.00	179.0
AdultWeekend	279.0	64.166810	24.554584	17.0	47.00	60.0	77.50	179.0
projectedDaysOpen	283.0	120.053004	31.045963	30.0	100.00	120.0	139.50	305.0
NightSkiing_ac	187.0	100.395722	105.169620	2.0	40.00	72.0	114.00	650.0

> **Recall you're missing the ticket prices for some 16% of resorts.**

This is a fundamental problem that means you simply lack the required data for those resorts and will have to drop those records. But you may have a weekend price and not a weekday price, or vice versa. You want to keep any price you have.

In [30]:
```
missing_price = ski_data[['AdultWeekend', 'AdultWeekday']].isnull().sum(axis=1)
missing_price.value_counts()/len(missing_price) * 100
```

Out[30]:
```
0    82.424242
2    14.242424
1     3.333333
dtype: float64
```

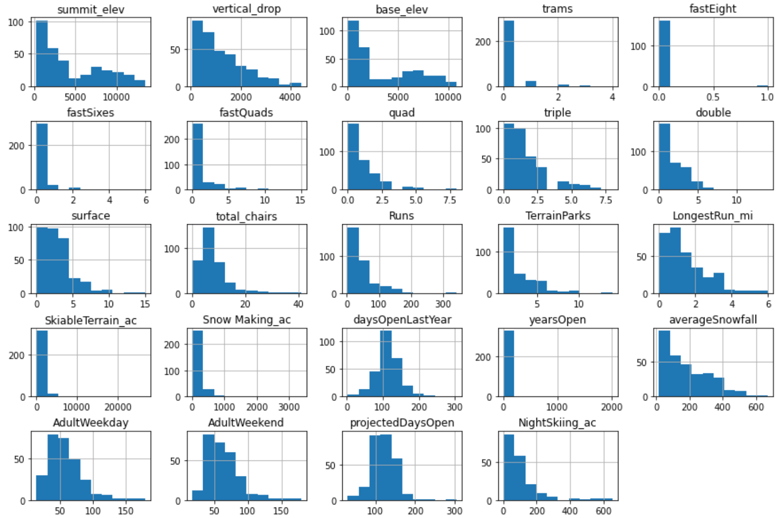
- Just over 82% of resorts have no missing ticket price, 3% are missing one value, and 14% are missing both.
- You will **definitely want to drop the records for which you have no price information, however you will not do so just yet.**
- There may still be **useful information about the distributions of other features in that 14% of the data.**

2.6.4.2 Distributions Of Feature Values

Note that, although we are still in the 'data wrangling and cleaning' phase rather than exploratory data analysis, looking at distributions of features is immensely useful in getting a feel for whether the values look sensible and whether there are any obvious outliers to investigate. Some exploratory data analysis belongs here, and data wrangling will inevitably occur later on. It's more a matter of emphasis.

> **Here, we're interesting in focusing on whether distributions look plausible or wrong. Later on, we're more interested in relationships and patterns.**

In [31]:
```
#Code task 18#

#Call ski_data's `hist` method to plot histograms of each of the numeric features
#Try passing it an argument figsize=(15,10)
#Try calling plt.subplots_adjust() with an argument hspace=0.5 to adjust the spacing
#It's important you create legible and easy-to-read plots
ski_data.hist(figsize=(15,10))
plt.subplots_adjust(hspace=0.5);
#Hint: notice how the terminating ';' "swallows" some messy output and leads to a tidier notebook
```



What features do we have possible cause for concern about and why?

- **clustering**
  - **SkiableTerrain_ac** - because values are clustered down the low end,
  - **Snow Making_ac** for the same reason,
- **fastEight**
  - because all but one value is 0 so it has very little variance, and half the values are missing,
- **variability**
  - **fastSixes** raises an amber flag; it has more variability, but still mostly 0,
  - **trams** also may get an amber flag for the same reason,
- **yearsOpen**
  - because most values are low but it has a maximum of 2019, which strongly suggests someone recorded calendar year rather than number of years.

2.6.4.2.1 SkiableTerrain_ac

In [32]:
```
#Code task 19#
#Filter the 'SkiableTerrain_ac' column to print the values greater than 10000
ski_data.loc[ski_data.SkiableTerrain_ac > 10000]
```

Out[32]:

	Name	Region	state	summit_elev	vertical_drop	base_elev	trams	fastEight	fastSix
39	Silverton Mountain	Colorado	Colorado	13487	3087	10400	0	0.0	

Q: 2 One resort has an incredibly large skiable terrain area! Which is it?

In [33]:

```
#Code task 20#
#Now you know there's only one, print the whole row to investigate all v
alues, including seeing the resort name
#Hint: don't forget the transpose will be helpful here
ski_data[ski_data.SkiableTerrain_ac > 10000].T
```

Out[33]:

	39
Name	Silverton Mountain
Region	Colorado
state	Colorado
summit_elev	13487
vertical_drop	3087
base_elev	10400
trams	0
fastEight	0
fastSixes	0
fastQuads	0
quad	0
triple	0
double	1
surface	0
total_chairs	1
Runs	NaN
TerrainParks	NaN
LongestRun_mi	1.5
SkiableTerrain_ac	26819
Snow Making_ac	NaN
daysOpenLastYear	175
yearsOpen	17
averageSnowfall	400
AdultWeekday	79
AdultWeekend	79
projectedDaysOpen	181
NightSkiing_ac	NaN

A: 2 Silverton Mountain

But what can you do when you have one record that seems highly suspicious?

You can see if your data are correct. Search for "silverton mountain skiable area". If you do this, you get some useful information (https://www.google.com/search?q=silverton+mountain+skiable+area).

Silverton Mountain information

Investigating Silverton Mountain: You can spot check data. You see your top and base elevation values agree, but the skiable area is very different. Your suspect value is 26819, but the value you've just looked up is 1819. The last three digits agree. This sort of error could have occured in transmission or some editing or transcription stage. You could plausibly replace the suspect value with the one you've just obtained. Another cautionary note to make here is that although you're doing this in order to progress with your analysis, this is most definitely an issue that should have been raised and fed back to the client or data originator as a query. You should view this "data correction" step as a means to continue (documenting it carefully as you do in this notebook) rather than an ultimate decision as to what is correct.

In [34]:

```
#Code task 21#
#Use the .loc accessor to print the 'SkiableTerrain_ac' value only for t
his resort
ski_data.loc[39, 'SkiableTerrain_ac']
```

Out[34]: 26819.0

In [35]:

```
#Code task 22#
#Use the .loc accessor again to modify this value with the correct value
of 1819
ski_data.loc[39, 'SkiableTerrain_ac'] = 1819
```

In [36]:

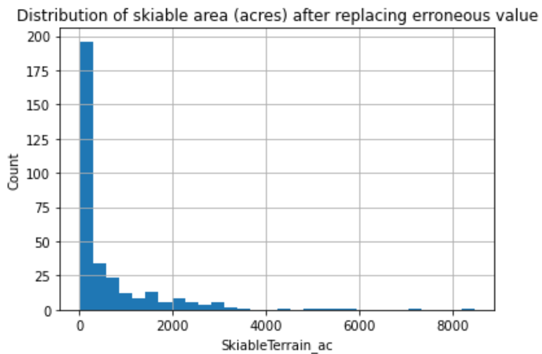
```
#Code task 23#
#Use the .loc accessor a final time to verify that the value has been mo
dified
ski_data.loc[39, 'SkiableTerrain_ac']
```

Out[36]: 1819.0

NB whilst you may become suspicious about your data quality, and you know you have missing values, you will not here dive down the rabbit hole of checking all values or web scraping to replace missing values.

What does the distribution of skiable area look like now?

In [37]:
```
ski_data.SkiableTerrain_ac.hist(bins=30)
plt.xlabel('SkiableTerrain_ac')
plt.ylabel('Count')
plt.title('Distribution of skiable area (acres) after replacing erroneou
s value');
```



You now see a rather long tailed distribution. You may wonder about the now most extreme value that is above 8000, but similarly you may also wonder about the value around 7000. If you wanted to spend more time manually checking values you could, but leave this for now. **The above distribution is plausible even with the outliers.**

2.6.4.2.2 Snow Making_ac

In [38]:
```
ski_data['Snow Making_ac'][ski_data['Snow Making_ac'] > 1000]
```

Out[38]:
```
11    3379.0
18    1500.0
Name: Snow Making_ac, dtype: float64
```

In [39]:
```
ski_data[ski_data['Snow Making_ac'] > 3000].T
```

Out[39]:

	11
Name	Heavenly Mountain Resort
Region	Sierra Nevada
state	California
summit_elev	10067
vertical_drop	3500
base_elev	7170
trams	2
fastEight	0
fastSixes	2
fastQuads	7
quad	1
triple	5
double	3
surface	8
total_chairs	28
Runs	97
TerrainParks	3
LongestRun_mi	5.5
SkiableTerrain_ac	4800
Snow Making_ac	3379
daysOpenLastYear	155
yearsOpen	64
averageSnowfall	360
AdultWeekday	NaN
AdultWeekend	NaN
projectedDaysOpen	157
NightSkiing_ac	NaN

You can adopt a similar approach as for the suspect skiable area value and do some spot checking. To save time, here is a link to the website for [Heavenly Mountain Resort (https://www.skiheavenly.com/the-mountain/about-the-mountain/mountain-info.aspx)](https://www.skiheavenly.com/the-mountain/about-the-mountain/mountain-info.aspx). **From this you can glean that you have values for skiable terrain that agree. Furthermore, you can read that snowmaking covers 60% of the trails.**

What, then, is your rough guess for the area covered by snowmaking?

In [40]:
```
# where 4800 is the Skiable Terrain
.6 * 4800
```

Out[40]:
```
2880.0
```

This is less than the value of 3379 in your data so you may have a judgement call to make. However, notice something else. You have no ticket pricing information at all for this resort. Any further effort spent worrying about values for this resort will be wasted. **You'll simply be dropping the entire row!**

2.6.4.2.3 fastEight

Look at the different fastEight values more closely:

In [41]:
```
ski_data.fastEight.value_counts()
```

Out[41]:
```
0.0    163
1.0      1
Name: fastEight, dtype: int64
```

Drop the fastEight column in its entirety; half the values are missing and all but the others are the value zero. There is essentially no information in this column.

In [42]:
```
#Code task 24#
#Drop the 'fastEight' column from ski_data. Use inplace=True
ski_data.drop(columns='fastEight', inplace=True)
```

What about yearsOpen? How many resorts have purportedly been open for more than 100 years?

In [43]:
```
#Code task 25#
#Filter the 'yearsOpen' column for values greater than 100
ski_data.loc[ski_data['yearsOpen'] > 100]
```

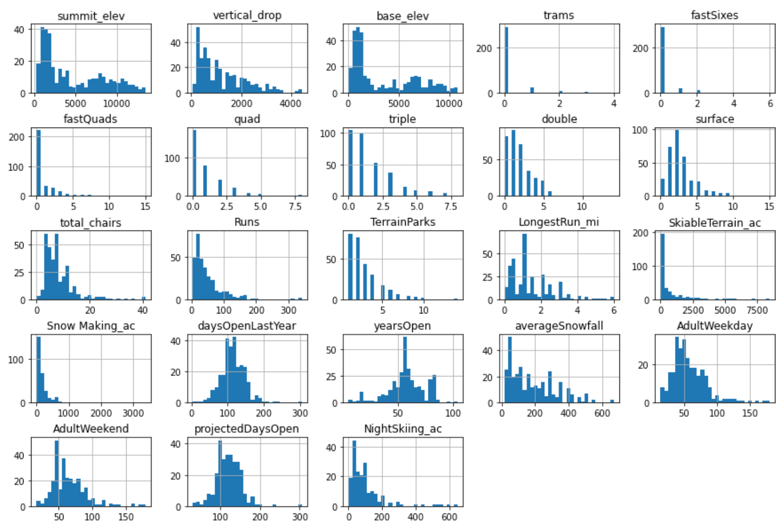
Out[43]:

	Name	Region	state	summit_elev	vertical_drop	base_elev	trams	fastSixes	fastC
34	Howelsen Hill	Colorado	Colorado	7136	440	6696	0	0	
115	Pine Knob Ski Resort	Michigan	Michigan	1308	300	1009	0	0	

Okay, one seems to have been open for 104 years. But beyond that, one is down as having been open for 2019 years. This is wrong! What shall you do about this?

What does the distribution of yearsOpen look like if you exclude just the obviously wrong one?

In [44]:
```python
#Code task 26#
#Call the hist method on 'yearsOpen' after filtering for values under 10
00
#Pass the argument bins=30 to hist(), but feel free to explore other val
ues
# ski_data.hist(figsize=(15,10))


ski_data.loc[ski_data.yearsOpen < 1000].hist(figsize=(15,10),bins=30)
plt.subplots_adjust(hspace=0.5)

plt.xlabel('Years open')
plt.ylabel('Count')
plt.title('Distribution of years open excluding 2019');
```



The above distribution of years seems entirely plausible, including the 104 year value. You can certainly state that no resort will have been open for 2019 years! It likely means the resort opened in 2019. It could also mean the resort is due to open in 2019. You don't know when these data were gathered!

Let's review the summary statistics for the years under 1000.

In [45]:
```python
ski_data.yearsOpen[ski_data.yearsOpen < 1000].describe()
```

Out[45]:
```
count    328.000000
mean      57.695122
std       16.841182
min        6.000000
25%       50.000000
50%       58.000000
75%       68.250000
max      104.000000
Name: yearsOpen, dtype: float64
```

The smallest number of years open otherwise is 6. You can't be sure whether this resort in question has been open zero years or one year and even whether the numbers are projections or actual. In any case, you would be adding a new youngest resort so it feels best to simply drop this row.

In [46]:
```python
# overrides the dataset
ski_data = ski_data[ski_data.yearsOpen < 1000]
```

2.6.4.2.4 fastSixes and Trams

The other features you had mild concern over, you will not investigate further. Perhaps take some care when using these features.

2.7 Derive State-wide Summary Statistics For Our Market Segment

You have, by this point removed one row, but it was for a resort that may not have opened yet, or perhaps in its first season. Using your business knowledge, you know that **state-wide supply and demand of certain skiing resources may well factor into pricing strategies.**

- Does a resort dominate the available night skiing in a state?
- Or does it account for a large proportion of the total skiable terrain or days open?

> **If you want to add any features to your data that captures the state-wide market size, you should do this now, before dropping any more rows.**

In the next section, you'll drop rows with missing price information. Although you don't know what those resorts charge for their tickets, you do know the resorts exists and have been open for at least six years. Thus, you'll now calculate some state-wide summary statistics for later use.

Many features in your data pertain to chairlifts, that is for getting people around each resort. These aren't relevant, nor are the features relating to altitudes.

Features that you may be interested in are:

- TerrainParks
- SkiableTerrain_ac
- daysOpenLastYear
- NightSkiing_ac

When you think about it, these are features it makes sense to sum: the total number of terrain parks, the total skiable area, the total number of days open, and the total area available for night skiing. **You might consider the total number of ski runs, but understand that the skiable area is more informative than just a number of runs.**

A fairly new groupby behaviour is [named aggregation (https://pandas-docs.github.io/pandas-docs-travis/whatsnew/v0.25.0.html)](https://pandas-docs.github.io/pandas-docs-travis/whatsnew/v0.25.0.html). This allows us to clearly perform the aggregations you want whilst also creating informative output column names.

In [47]:

```
#Code task 27#
#Add named aggregations for the sum of 'daysOpenLastYear', 'TerrainPark
s', and 'NightSkiing_ac'
#call them 'state_total_days_open', 'state_total_terrain_parks', and 'st
ate_total_nightskiing_ac',
#respectively
#Finally, add a call to the reset_index() method (we recommend you exper
iment with and without this to see
#what it does)

state_summary = ski_data.groupby('state').agg(
    resorts_per_state=pd.NamedAgg(column='Name', aggfunc='size'), #could
pick any column here
    state_total_skiable_area_ac=pd.NamedAgg(column='SkiableTerrain_ac',
aggfunc='sum'),
    state_total_days_open=pd.NamedAgg(column='daysOpenLastYear', aggfunc
='sum'),
    state_total_terrain_parks=pd.NamedAgg(column='TerrainParks', aggfunc
='sum'),
    state_total_nightskiing_ac=pd.NamedAgg(column='NightSkiing_ac', aggf
unc='sum')
).reset_index()
state_summary.head()
```

Out[47]:

	state	resorts_per_state	state_total_skiable_area_ac	state_total_days_open	state_total_ter
0	Alaska	3	2280.0	345.0	
1	Arizona	2	1577.0	237.0	
2	California	21	25948.0	2738.0	
3	Colorado	22	43682.0	3258.0	
4	Connecticut	5	358.0	353.0	

2.8 Drop Rows With No Price Data

You know there are two columns that refer to price: 'AdultWeekend' and 'AdultWeekday'. You can calculate the number of price values missing per row. This will obviously have to be either 0, 1, or 2, where 0 denotes no price values are missing and 2 denotes that both are missing.

In [48]:

```
missing_price = ski_data[['AdultWeekend', 'AdultWeekday']].isnull().sum(
axis=1)
missing_price.value_counts()/len(missing_price) * 100
```

Out[48]:

```
0    82.317073
2    14.329268
1     3.353659
dtype: float64
```

About 14% of the rows have no price data. As the price is your target, these rows are of no use. Time to lose them.

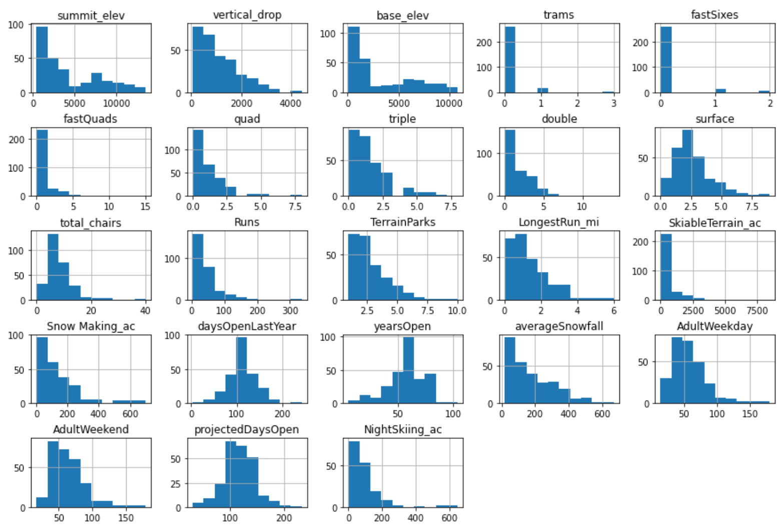
```
In [49]: #Code task 28#
         #Use `missing_price` to remove rows from ski_data where both price value
         s are missing
         ski_data = ski_data[missing_price != 2]
```

```
In [50]: # check that rows are dropped
         len(ski_data)
```

```
Out[50]: 281
```

2.9 Review distributions

```
In [51]: ski_data.hist(figsize=(15, 10))
         plt.subplots_adjust(hspace=0.5);
```



These distributions are much better. There are clearly **some skewed distributions**, so keep an eye on `fastQuads` , `fastSixes` , and perhaps `trams` . These lack much variance away from 0 and may have a small number of relatively extreme values.

> **Models failing to rate a feature as important when domain knowledge tells you it should be is an issue to look out for, as is a model being overly influenced by some extreme values.**

If you build a **good machine learning pipeline**, hopefully it will be **robust to such issues**, but you may also wish to **consider nonlinear transformations of features.**

2.10 Population data

Population and area data for the US states can be obtained from [wikipedia (https://simple.wikipedia.org/wiki/List_of_U.S._states)](https://simple.wikipedia.org/wiki/List_of_U.S._states). Listen, you should have a healthy concern about using data you "found on the Internet". Make sure it comes from a reputable source. This table of data is useful because it allows you to easily pull and incorporate an external data set. It also allows you to proceed with an analysis that includes state sizes and populations for your 'first cut' model. Be explicit about your source (we documented it here in this workflow) and ensure it is open to inspection. All steps are subject to review, and it may be that a client has a specific source of data they trust that you should use to rerun the analysis.

```
In [52]: #Code task 29#
         #Use pandas' `read_html` method to read the table from the URL below
         states_url = 'https://simple.wikipedia.org/wiki/List_of_U.S._states'
         usa_states = pd.read_html(states_url)
```

```
In [53]: type(usa_states)
```

```
Out[53]: list
```

```
In [54]: len(usa_states)
```

```
Out[54]: 1
```

In [55]:
```
usa_states = usa_states[0]
usa_states.head()
```

Out[55]:

	Name &postal abbs. [1]		Cities		Established[upper-alpha 1]	Population[upper-alpha 2][3]	Total area[4]	
	Name &postal abbs. [1]	Name &postal abbs. [1].1	Capital	Largest[5]	Established[upper-alpha 1]	Population[upper-alpha 2][3]	mi2	km
0	Alabama	AL	Montgomery	Birmingham	Dec 14, 1819	4903185	52420	1:
1	Alaska	AK	Juneau	Anchorage	Jan 3, 1959	731545	665384	17:
2	Arizona	AZ	Phoenix	Phoenix	Feb 14, 1912	7278717	113990	2!
3	Arkansas	AR	Little Rock	Little Rock	Jun 15, 1836	3017804	53179	1:
4	California	CA	Sacramento	Los Angeles	Sep 9, 1850	39512223	163695	4:

Note, in even the last year, the capability of `pd.read_html()` has improved. The merged cells you see in the web table are now handled much more conveniently, with 'Phoenix' now being duplicated so the subsequent columns remain aligned. But check this anyway. If you extract the established date column, you should just get dates. Recall previously you used the `.loc` accessor, because you were using labels. Now you want to refer to a column by its index position and so use `.iloc` . For a discussion on the difference use cases of `.loc` and `.iloc` refer to the [pandas documentation (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html)](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html).

In [56]:
```
#Code task 30#
#Use the iloc accessor to get the pandas Series for column number 4 from
`usa_states`
#It should be a column of dates
established = usa_states.iloc[:, 4]
```

In [57]:
```
established.head()
```

Out[57]:
```
0    Dec 14, 1819
1     Jan 3, 1959
2    Feb 14, 1912
3    Jun 15, 1836
4     Sep 9, 1850
Name: (Established[upper-alpha 1], Established[upper-alpha 1]), dtype:
object
```

Extract the state name, population, and total area (square miles) columns.

In [58]:
```
#Code task 31#
#Now use the iloc accessor again to extract columns 0, 5, and 6 and the
 dataframe's `copy()` method
#Set the names of these extracted columns to 'state', 'state_populatio
n', and 'state_area_sq_miles',
#respectively.
usa_states_sub = usa_states.iloc[:, [0,4,5]].copy()
usa_states_sub.columns = ['state','state_population','state_area_sq_mile
s']
usa_states_sub.head()
```

Out[58]:

	state	state_population	state_area_sq_miles
0	Alabama	Dec 14, 1819	4903185
1	Alaska	Jan 3, 1959	731545
2	Arizona	Feb 14, 1912	7278717
3	Arkansas	Jun 15, 1836	3017804
4	California	Sep 9, 1850	39512223

Do you have all the ski data states accounted for?

In [59]:
```
state_summary.head()
```

Out[59]:

	state	resorts_per_state	state_total_skiable_area_ac	state_total_days_open	state_total_ter
0	Alaska	3	2280.0	345.0	
1	Arizona	2	1577.0	237.0	
2	California	21	25948.0	2738.0	
3	Colorado	22	43682.0	3258.0	
4	Connecticut	5	358.0	353.0	

In [60]:
```
#Code task 32#
#Find the states in `state_summary` that are not in `usa_states_sub`
#Hint: set(list1) - set(list2) is an easy way to get items in list1 that
are not in list2
missing_states = set(state_summary.state) - set(usa_states_sub.state)
missing_states
```

Out[60]:
```
{'Massachusetts', 'Pennsylvania', 'Rhode Island', 'Virginia'}
```

No??

If you look at the table on the web, you can perhaps start to guess what the problem is. You can confirm your suspicion by pulling out state names that *contain* 'Massachusetts', 'Pennsylvania', or 'Virginia' from usa_states_sub:

In [61]:
```
usa_states_sub.state[usa_states_sub.state.str.contains('Massachusetts|Pe
nnsylvania|Rhode Island|Virginia')]
```

Out[61]:
```
20    Massachusetts[upper-alpha 3]
37     Pennsylvania[upper-alpha 3]
38     Rhode Island[upper-alpha 4]
45         Virginia[upper-alpha 3]
47                   West Virginia
Name: state, dtype: object
```

Delete square brackets and their contents and try again:

In [62]:
```
#Code task 33#
#Use pandas' Series' `replace()` method to replace anything within squar
e brackets (including the brackets)
#with the empty string. Do this inplace, so you need to specify the argu
ments:
#to_replace='\[.*\]' #literal square bracket followed by anything or not
hing followed by literal closing bracket
#value='' #empty string as replacement
#regex=True #we used a regex in our `to_replace` argument
#inplace=True #Do this "in place"
usa_states_sub.state.replace(to_replace='\[.*\]', value="", regex=True,
inplace=True)
usa_states_sub.state[usa_states_sub.state.str.contains('Massachusetts|Pe
nnsylvania|Rhode Island|Virginia')]
```

Out[62]:
```
20    Massachusetts
37     Pennsylvania
38     Rhode Island
45         Virginia
47    West Virginia
Name: state, dtype: object
```

In [63]:
```
#Code task 34#
#And now verify none of our states are missing by checking that there ar
e no states in
#state_summary that are not in usa_states_sub (as earlier using `set()`)
missing_states = set(state_summary.state) - set(usa_states_sub.state)
missing_states
```

Out[63]:
```
set()
```

Better! You have an empty set for missing states now. You can confidently add the population and state area columns to the ski resort data.

In [64]:
```
state_summary.head()
```

Out[64]:

	state	resorts_per_state	state_total_skiable_area_ac	state_total_days_open	state_total_ter
0	Alaska	3	2280.0	345.0	
1	Arizona	2	1577.0	237.0	
2	California	21	25948.0	2738.0	
3	Colorado	22	43682.0	3258.0	
4	Connecticut	5	358.0	353.0	

In [65]:
```
usa_states_sub.head()
```

Out[65]:

	state	state_population	state_area_sq_miles
0	Alabama	Dec 14, 1819	4903185
1	Alaska	Jan 3, 1959	731545
2	Arizona	Feb 14, 1912	7278717
3	Arkansas	Jun 15, 1836	3017804
4	California	Sep 9, 1850	39512223

In [66]:
```
#Code task 35#
#Use 'state_summary's `merge()` method to combine our new data in 'usa_s
tates_sub'
#specify the arguments how='left' and on='state'
state_summary = state_summary.merge(usa_states_sub, how='left', on='stat
e')
state_summary.head().T
```

Out[66]:

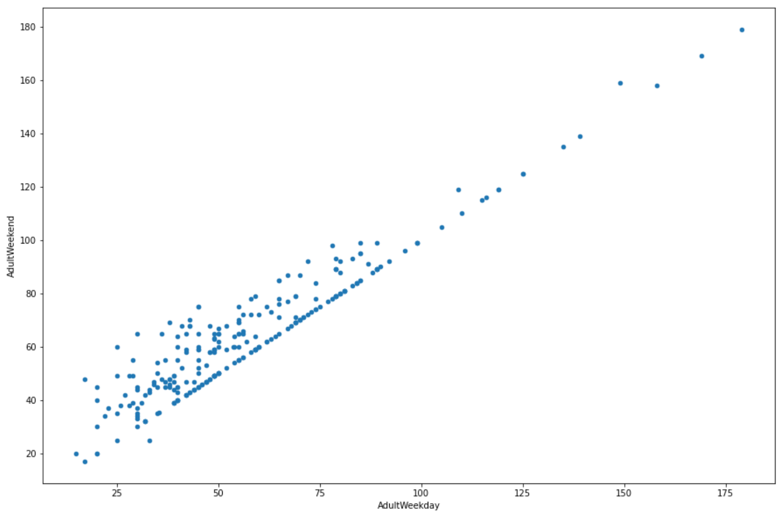
	0	1	2	3	4
state	Alaska	Arizona	California	Colorado	Connecticut
resorts_per_state	3	2	21	22	5
state_total_skiable_area_ac	2280	1577	25948	43682	358
state_total_days_open	345	237	2738	3258	353
state_total_terrain_parks	4	6	81	74	10
state_total_nightskiing_ac	580	80	587	428	256
state_population	Jan 3, 1959	Feb 14, 1912	Sep 9, 1850	Aug 1, 1876	Jan 9, 1788
state_area_sq_miles	731545	7278717	39512223	5758736	3565278

Having created this data frame of summary statistics for various states, it would seem obvious to join this with the ski resort data to augment it with this additional data. You will do this, but not now. In the next notebook you will be exploring the data, including the relationships between the states. For that you want a separate row for each state, as you have here, and joining the data this soon means you'd need to separate and eliminate redundances in the state data when you wanted it.

2.11 Target Feature

Finally, what will your target be when modelling ticket price? What relationship is there between weekday and weekend prices?

In [67]:
```
#Code task 36#
#Use ski_data's `plot()` method to create a scatterplot (kind='scatter') with 'AdultWeekday' on the x-axis and
#'AdultWeekend' on the y-axis

ski_data.plot(x='AdultWeekday', y='AdultWeekend', kind='scatter', figsize=(15,10));
```



A couple of observations can be made. Firstly, there is a clear line where weekend and weekday prices are equal. Weekend prices being higher than weekday prices seem restricted to sub $100 resorts. Recall from the boxplot earlier that the distribution for weekday and weekend prices in Montana seemed equal. Is this confirmed in the actual data for each resort? Big Mountain resort is in Montana, so the relationship between these quantities in this state are particularly relevant.

In [68]:
```
#Code task 37#
#Use the loc accessor on ski_data to print the 'AdultWeekend' and 'AdultWeekday' columns for Montana only
ski_data.loc[ski_data.state == 'Montana', ['AdultWeekend', 'AdultWeekday']]
```

Out[68]:

	AdultWeekend	AdultWeekday
141	42.0	42.0
142	63.0	63.0
143	49.0	49.0
144	48.0	48.0
145	46.0	46.0
146	39.0	39.0
147	50.0	50.0
148	67.0	67.0
149	47.0	47.0
150	39.0	39.0
151	81.0	81.0

Is there any reason to prefer weekend or weekday prices? Which is of the two tickets prices offered are missing the least?

In [69]:
```
ski_data[['AdultWeekend', 'AdultWeekday']].isnull().sum()
```

Out[69]:
```
AdultWeekend    4
AdultWeekday    7
dtype: int64
```

Weekend prices have the least missing values of the two, so drop the weekday prices and then keep just the rows that have weekend price.

In [70]:
```
ski_data.drop(columns='AdultWeekday', inplace=True)
ski_data.dropna(subset=['AdultWeekend'], inplace=True)
```

In [71]:
```
ski_data.shape
```

Out[71]:
```
(277, 25)
```

Perform a final quick check on the data.

2.11.1 Number Of Missing Values By Row - Resort

Having dropped rows missing the desired target ticket price, what degree of missingness do you have for the remaining rows?

In [72]:
```
missing = pd.concat([ski_data.isnull().sum(axis=1), 100 * ski_data.isnull().mean(axis=1)], axis=1)
missing.columns=['count', '%']
missing.sort_values(by='count', ascending=False).head(10)
```

Out[72]:

	count	%
329	5	20.0
62	5	20.0
141	5	20.0
86	5	20.0
74	5	20.0
146	5	20.0
184	4	16.0
108	4	16.0
198	4	16.0
39	4	16.0

These seem possibly curiously quantized...

In [73]:
```
missing['%'].unique()
```

Out[73]:
```
array([ 0.,  4.,  8., 12., 16., 20.])
```

Yes, the percentage of missing values per row appear in multiples of 4.

In [74]:
```
missing['%'].value_counts()
```

Out[74]:
```
0.0     107
4.0      94
8.0      45
12.0     15
16.0     10
20.0      6
Name: %, dtype: int64
```

This is almost as if values have been removed artificially... Nevertheless, what you don't know is how useful the missing features are in predicting ticket price.

> **You shouldn't just drop rows that are missing several useless features.**

In [75]:
```
ski_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 277 entries, 0 to 329
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Name               277 non-null    object 
 1   Region             277 non-null    object 
 2   state              277 non-null    object 
 3   summit_elev        277 non-null    int64  
 4   vertical_drop      277 non-null    int64  
 5   base_elev          277 non-null    int64  
 6   trams              277 non-null    int64  
 7   fastSixes          277 non-null    int64  
 8   fastQuads          277 non-null    int64  
 9   quad               277 non-null    int64  
 10  triple             277 non-null    int64  
 11  double             277 non-null    int64  
 12  surface            277 non-null    int64  
 13  total_chairs       277 non-null    int64  
 14  Runs               274 non-null    float64
 15  TerrainParks       233 non-null    float64
 16  LongestRun_mi      272 non-null    float64
 17  SkiableTerrain_ac  275 non-null    float64
 18  Snow Making_ac     240 non-null    float64
 19  daysOpenLastYear   233 non-null    float64
 20  yearsOpen          277 non-null    float64
 21  averageSnowfall    268 non-null    float64
 22  AdultWeekend       277 non-null    float64
 23  projectedDaysOpen  236 non-null    float64
 24  NightSkiing_ac     163 non-null    float64
dtypes: float64(11), int64(11), object(3)
memory usage: 56.3+ KB
```

There are still some missing values, and it's good to be aware of this, but leave them as is for now.

2.12 Save data

In [76]:
```
ski_data.shape
```

Out[76]:
```
(277, 25)
```

Save this to your data directory, separately. Note that you were provided with the data in `raw_data` and you should saving derived data in a separate location. This guards against overwriting our original data.

In [77]:
```
datapath = '../data'
# renaming the output data directory and re-running this notebook, for e
xample,
# will recreate this (empty) directory and resave the data files.
# NB this is not a substitute for a modern data pipeline, for which ther
e are
# various tools. However, for our purposes here, and often in a "one of
f" analysis,
# this is useful because we have to deliberately move/delete our data in
order
# to overwrite it.
if not os.path.exists(datapath):
    os.mkdir(datapath)
```

In [78]:
```
datapath_skidata = os.path.join(datapath, 'ski_data_cleaned.csv')
if not os.path.exists(datapath_skidata):
    ski_data.to_csv(datapath_skidata, index=False)
```

In [79]:
```
datapath_states = os.path.join(datapath, 'state_summary.csv')
if not os.path.exists(datapath_states):
    state_summary.to_csv(datapath_states, index=False)
```

2.13 Summary

Q: 3 Write a summary statement that highlights the key processes and findings from this notebook.

This should include information such as:

- the original number of rows in the data,
- whether our own resort was actually present etc.
- What columns, if any, have been removed? Any rows? Summarise the reasons why.
- Were any other issues found? What remedial actions did you take?
- State where you are in the project.
- Can you confirm what the target feature is for your desire to predict ticket price?
- How many rows were left in the data?

Hint: this is a great opportunity to reread your notebook, check all cells have been executed in order and from a "blank slate" (restarting the kernel will do this), and that your workflow makes sense and follows a logical pattern. As you do this you can pull out salient information for inclusion in this summary. Thus, this section will provide an important overview of "what" and "why" without having to dive into the "how" or any unproductive or inconclusive steps along the way.

A: 3 Your answer here