Table of contents

Visualization (Exploring variation)

Author

Peter Ganong and Maggie Shi

Published

October 14, 2024

Motivation

Introduction to the next two lectures

Most of our visualization lectures are based on the University of Washington textbook, but the textbook doesn't have enough material on exploratory data analysis. We therefore are supplementing with the Data Visualization and Exploratory Data Analysis material in the R for Data Science textbook (with the code translated to Altair).

- `diamonds` is from "Exploratory Data Analysis"
- `movies` is from the UW textbook
- `penguins` is from "Data Visualization"
- `mpg`

What is exploratory data analysis?

Data visualization has two distinct goals

1. **exploration** for you to learn as much as possible
2. **production** for you to teach someone else what you think the key lessons are

How do the modes differ?

- When you are in exploration mode, you will look at lots of patterns and your brain filters out the noise
- Production mode is like putting a cone on your dog. You are deliberately limiting the reader's field of vision such that they see the key messages from the plot *and avoid too many distractions*

The next two lectures are almost entirely about **exploration**. Then, at the end of lecture 5, we will transition to thinking about graphics for production. Lecture 6 will similarly about graphics for production.

Caveat: these modes make the most sense when thinking about *static* visualization. Later on in the course, when we talk about dashboards, this is closer to making interfaces to help readers who don't code explore the data.

Categorical variables

Categorical variables: roadmap

- introduce `diamonds`
- show table
- show bar graph

introduce dataset `diamonds`

```
from plotnine.data import diamonds, mpg
diamonds
```

	carat	cut	color	clarity	depth	table	price	x	y	z	bins
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	(0.195, 0.681]
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	(0.195, 0.681]
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	(0.195, 0.681]
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	(0.195, 0.681]
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	(0.195, 0.681]
...	...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50	(0.681, 1.162]
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61	(0.681, 1.162]
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56	(0.681, 1.162]
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74	(0.681, 1.162]
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64	(0.681, 1.162]

53940 rows × 11 columns

`diamonds` data dictionary

(Accessed by running `?diamonds` in R) A data frame with 53940 rows and 10 variables:

- `price` - price in US dollars (\$326–\$18,823)
- `carat`- weight of the diamond (0.2–5.01)
- `cut` - quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- `color` - diamond colour, from D (best) to J (worst)
- `clarity` - a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- `x` – length in mm (0–10.74)
- `y` – width in mm (0–58.9)
- `z` – depth in mm (0–31.8)
- `depth` – total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- `table` – width of top of diamond relative to widest point (43–95)

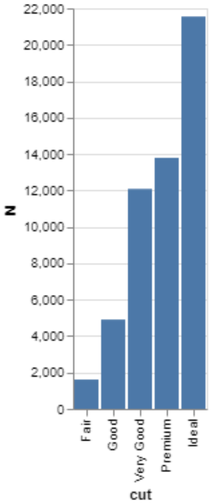
`diamonds`

```
diamonds_cut = diamonds.groupby('cut').size()
diamonds_cut
```

```
cut
Fair          1610
Good          4906
Very Good    12082
Premium      13791
Ideal        21551
dtype: int64
```

Categorical variables

```
diamonds_cut = diamonds_cut.reset_index().rename(columns={0:'N'}) # Prepare to plot

alt.Chart(diamonds_cut).mark_bar().encode(
    alt.X('cut'),
    alt.Y('N')
)
```



Categorical variables – summary

- this section is very brief because there's basically only one good way to plot categorical variables with a small number of categories and this is it.
  - You can use `mark_point()` instead of `mark_bar()`, but overall, there's a clear right answer about how to do this.
- We include this material mainly to foreshadow the fact that we will do a lot on categorical variables in the next lecture when we get to "Exploring Co-variation"

Continuous variables

Roadmap: Continuous variables

- histograms using `movies`
- histograms and density plots using `penguins`
- diamond size (carat)

Remark: The skills are absolutely fundamental and so we will intentionally be a bit repetitive.

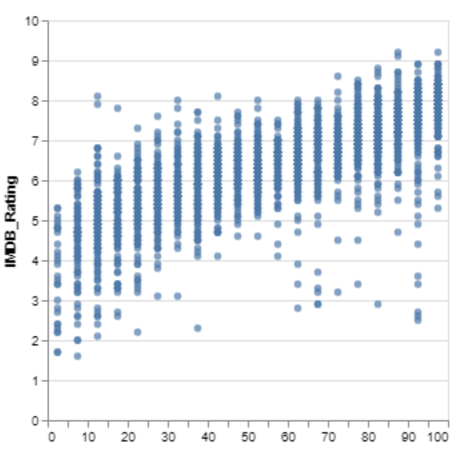
movies dataset

```
movies_url = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/movies.json'
```

```
movies = pd.read_json(movies_url)
```

recap scatter plot from lecture 3

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('IMDB_Rating:Q')
)
```

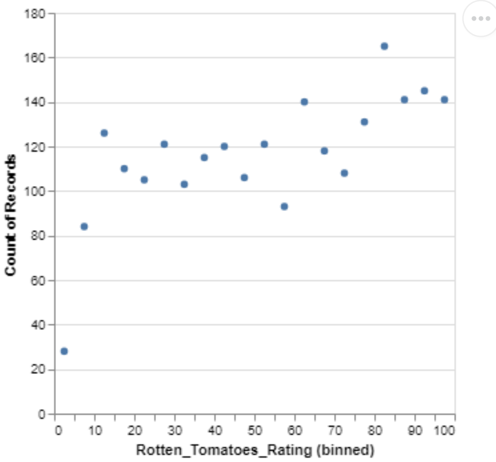


One question which came up (which is hard to tell from this scatter plot, even with bins) is how many observations are there in each bin

scatter plot – N movies per bin

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=alt.BinParams(maxbins=20)),
```

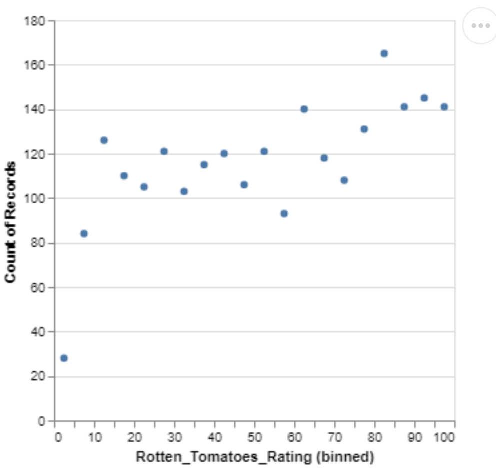
```
    alt.Y('count(IMDB_Rating):Q')
)
```



scatter plot – syntax trick

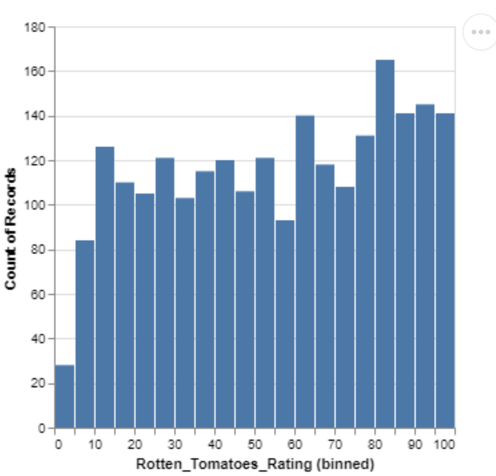
Replace `count(IMDB_Rating)` with `count()` because we aren't using IMDB rating any more.

```
alt.Chart(movies_url).mark_circle().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('count():Q')
)
```



histogram using `mark_bar()`

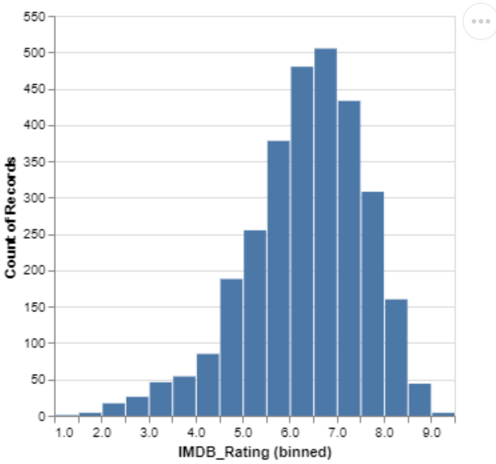
```
hist_rt = alt.Chart(movies_url).mark_bar().encode(
    alt.X('Rotten_Tomatoes_Rating:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('count():Q')
)
hist_rt
```



Discussion question: how would you describe the distribution of rotten tomatoes ratings?

histogram of IMDB ratings

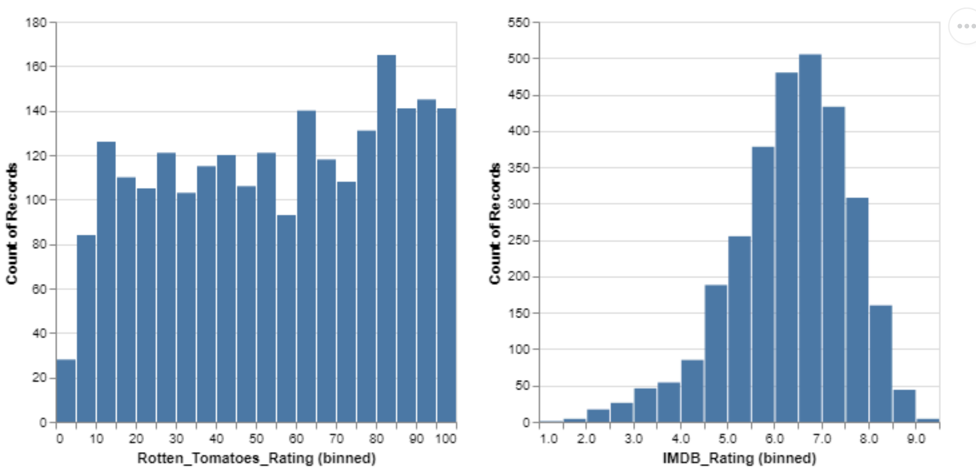
```
hist_imdb = alt.Chart(movies_url).mark_bar().encode(
    alt.X('IMDB_Rating:Q', bin=alt.BinParams(maxbins=20)),
    alt.Y('count():Q')
)
hist_imdb
```



Side-by-side

Discussion question – compare the two ratings distributions. If *your goal is to differentiate between good and bad movies*, which is more informative?

```
hist_rt | hist_imdb
```



introducing the penguins

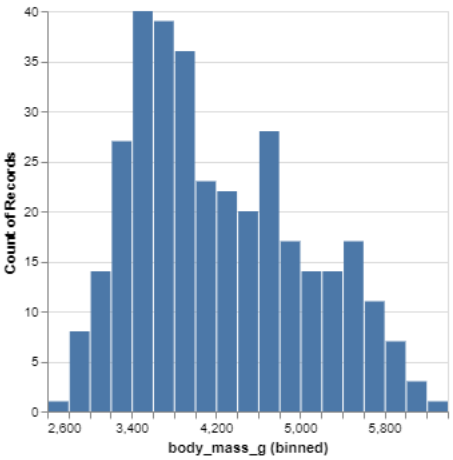
```
from palmerpenguins import load_penguins
penguins = load_penguins()
display(penguins)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	male	2007
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	female	2007
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	female	2007
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	female	2007
...	...	...	...	...	...	...	...	...
339	Chinstrap	Dream	55.8	19.8	207.0	4000.0	male	2009
340	Chinstrap	Dream	43.5	18.1	202.0	3400.0	female	2009
341	Chinstrap	Dream	49.6	18.2	193.0	3775.0	male	2009
342	Chinstrap	Dream	50.8	19.0	210.0	4100.0	male	2009
343	Chinstrap	Dream	50.2	18.7	198.0	3775.0	female	2009

344 rows × 8 columns

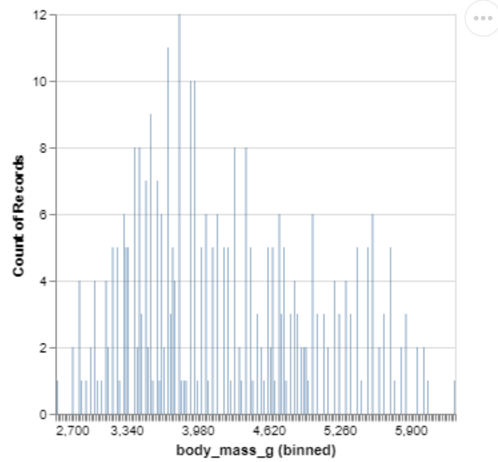
histogram with steps of 200

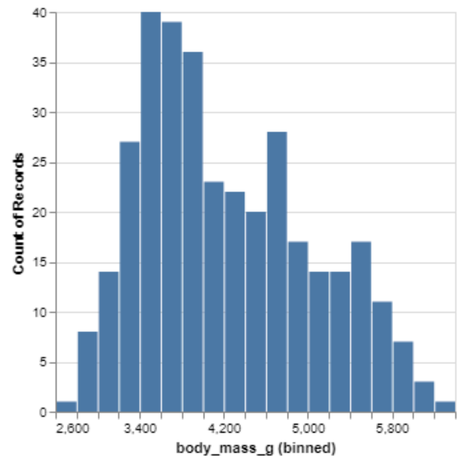
```
alt.Chart(penguins).mark_bar().encode(
    alt.X('body_mass_g', bin=alt.BinParams(step=200)),
    alt.Y('count()')
)
```

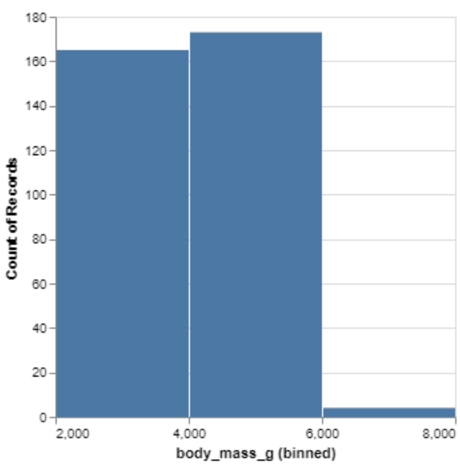


histogram `step` parameter

20 vs 200 vs 2000



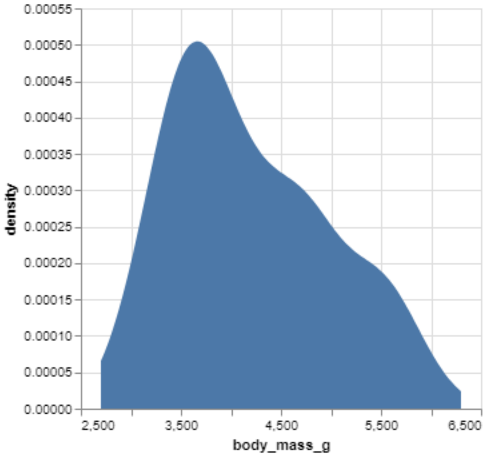




Discussion q – what message comes from each `binwidth` choice? Which do you prefer?

numeric variable: `transform_density()`

```
alt.Chart(penguins).transform_density(
    'body_mass_g',
    as_=['body_mass_g', 'density']
).mark_area().encode(
    x='body_mass_g:Q',
    y='density:Q'
)
```



Back to diamonds, focus on `carat`

```
alt.data_transformers.disable_max_rows() # Needed because len(df) > 5000

alt.Chart(diamonds).mark_bar().encode(
    alt.X('carat', bin=alt.Bin(maxbins=10)),
    alt.Y('count()')
)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~/miniconda3/lib/python3.12/site-packages/IPython/core/formatters.py:974, in MimeBundleFormatter.__call__(self, obj, include, exclude)
    971     method = get_real_method(obj, self.print_method)
    973     if method is not None:
--> 974         return method(include=include, exclude=exclude)
    975     return None
    976 else:

File ~/miniconda3/lib/python3.12/site-packages/altair/vegalite/v5/api.py:3417, in TopLevelMixin._repr_mimebundle_(self, *args, **kwds)
   3415 else:
   3416     if renderer := renderers.get():
-> 3417         return renderer(dct)

File ~/miniconda3/lib/python3.12/site-packages/altair/utils/display.py:225, in HTMLRenderer.__call__(self, spec, **metadata)
    223 kwargs = self.kwargs.copy()
    224 kwargs.update(**metadata, output_div=self.output_div)
--> 225 return spec_to_mimebundle(spec, format="html", **kwargs)

File ~/miniconda3/lib/python3.12/site-packages/altair/utils/mimebundle.py:144, in spec_to_mimebundle(spec, format, mode, vega_version, vegaembed_version, v
    134     return _spec_to_mimebundle_with_engine(
    135         spec,
    136         cast(Literal["png", "svg", "pdf", "vega"], format),
   (...)
    141         **kwargs,
    142     )
    143 elif format == "html":
--> 144     html = spec_to_html(
    145         spec,
    146         mode=internal_mode,
    147         vega_version=vega_version,
    148         vegaembed_version=vegaembed_version,
    149         vegalite_version=vegalite_version,
    150         embed_options=embed_options,
    151         **kwargs,
    152     )
    153     return {"text/html": html}
    154 elif format == "vega-lite":

File ~/miniconda3/lib/python3.12/site-packages/altair/utils/html.py:303, in spec_to_html(spec, mode, vega_version, vegaembed_version, vegalite_version, bas
    299     msg = f"Invalid template: {jinja_template}"
    300     raise ValueError(msg)
    302 return jinja_template.render(
--> 303     spec=json.dumps(spec, **json_kwds),
    304     embed_options=json.dumps(embed_options),
    305     mode=mode,
    306     vega_version=vega_version,
    307     vegalite_version=vegalite_version,
    308     vegaembed_version=vegaembed_version,
    309     base_url=base_url,
    310     output_div=output_div,
    311     fullhtml=fullhtml,
    312     requirejs=requirejs,
    313     **render_kwargs,
    314 )

File ~/miniconda3/lib/python3.12/json/__init__.py:231, in dumps(obj, skipkeys, ensure_ascii, check_circular, allow_nan, cls, indent, separators, default, s
    226 # cached encoder
    227 if (not skipkeys and ensure_ascii and
    228     check_circular and allow_nan and
    229     cls is None and indent is None and separators is None and
    230     default is None and not sort_keys and not kw):
```

```
--> 231     return _default_encoder.encode(obj)
    232 if cls is None:
    233     cls = JSONEncoder

File ~/miniconda3/lib/python3.12/json/encoder.py:200, in JSONEncoder.encode(self, o)
    196         return encode_basestring(o)
    197 # This doesn't pass the iterator directly to ''.join() because the
    198 # exceptions aren't as detailed.  The list call should be roughly
    199 # equivalent to the PySequence_Fast that ''.join() would do.
--> 200 chunks = self.iterencode(o, _one_shot=True)
    201 if not isinstance(chunks, (list, tuple)):
    202     chunks = list(chunks)

File ~/miniconda3/lib/python3.12/json/encoder.py:258, in JSONEncoder.iterencode(self, o, _one_shot)
    253 else:
    254     _iterencode = _make_iterencode(
    255         markers, self.default, _encoder, self.indent, floatstr,
    256         self.key_separator, self.item_separator, self.sort_keys,
    257         self.skipkeys, _one_shot)
--> 258 return _iterencode(o, 0)

File ~/miniconda3/lib/python3.12/json/encoder.py:180, in JSONEncoder.default(self, o)
    161 def default(self, o):
    162     """Implement this method in a subclass such that it returns
    163     a serializable object for ``o``, or calls the base implementation
    164     (to raise a ``TypeError``).
   (...)
    178
    179     """
--> 180     raise TypeError(f'Object of type {o.__class__.__name__} '
    181                     f'is not JSON serializable')

TypeError: Object of type Interval is not JSON serializable
```

```
alt.Chart(...)
```

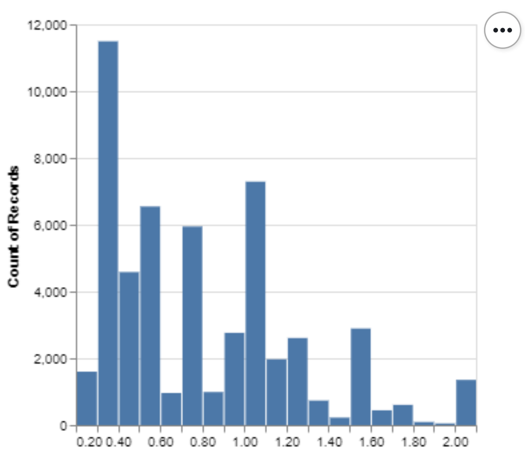
Continuous Variables

```
diamonds['bins'] = pd.cut(diamonds['carat'], bins=10)
diamonds.groupby('bins').size()
```

```
bins
(0.195, 0.681]    25155
(0.681, 1.162]    18626
(1.162, 1.643]     7129
(1.643, 2.124]     2349
(2.124, 2.605]      614
(2.605, 3.086]       53
(3.086, 3.567]        6
(3.567, 4.048]        5
(4.048, 4.529]        2
(4.529, 5.01]         1
dtype: int64
```

Continuous Variables: Typical Values

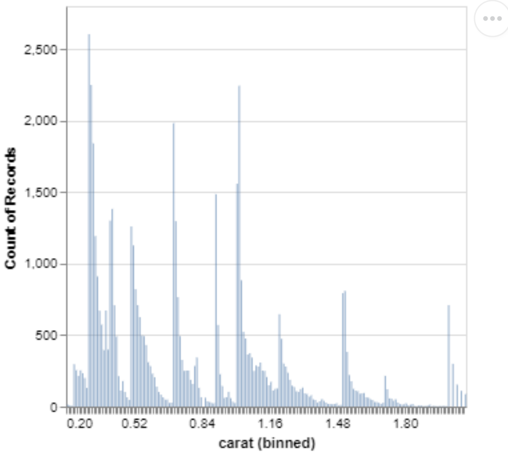
```
diamonds = diamonds.drop('bins', axis=1) # 'Interval' type causes plotting issues
diamonds_small = diamonds.loc[diamonds['carat'] < 2.1] # Subset to small diamonds

alt.Chart(diamonds_small).mark_bar().encode(
    alt.X('carat', bin=alt.BinParams(step=0.1)),
    alt.Y('count()')
)
```



Continuous Variables: Typical Values

```
alt.Chart(diamonds_small).mark_bar().encode(
    alt.X('carat', bin=alt.BinParams(step=0.01)),
```

```
    alt.Y('count()')
)
```

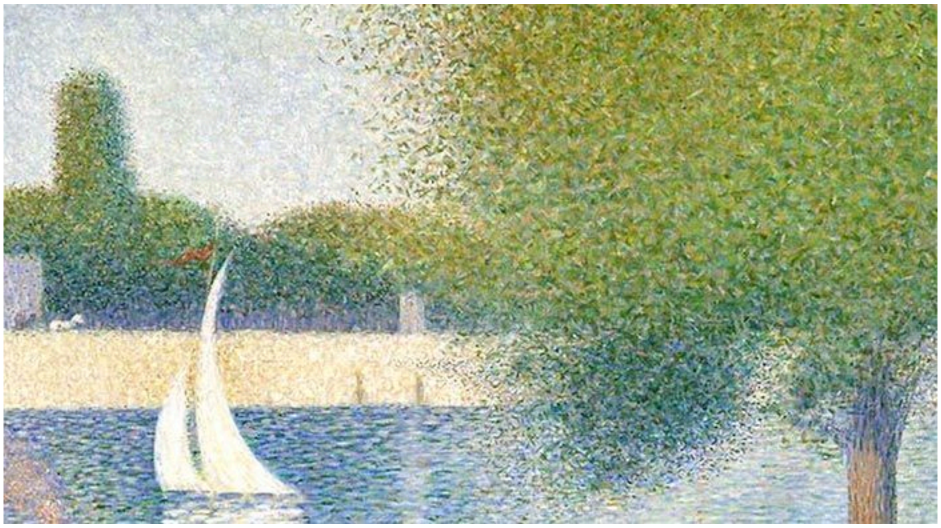


Discussion questions

1. What lessons does this plot teach?
2. What questions does it raise?

Aside: “A Sunday on La Grande Jatte” by Seurat

Aside: “A Sunday on La Grande Jatte” by Seurat



Unusual numeric values (`diamonds`)

roadmap

- case study: `y` dimension in diamonds
  - explore some unusual values
  - three options for handling unusual values

Diamonds: examine unusual values

```
diamonds['y'].describe()
```

```
count    53940.000000
mean         5.734526
std          1.142135
min          0.000000
25%          4.720000
50%          5.710000
75%          6.540000
max         58.900000
Name: y, dtype: float64
```

Diamonds: examine unusual values

```
diamonds.loc[(diamonds['y'] < 3) | (diamonds['y'] > 20)]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
11963	1.00	Very Good	H	VS2	63.3	53.0	5139	0.00	0.0	0.00
15951	1.14	Fair	G	VS1	57.5	67.0	6381	0.00	0.0	0.00
24067	2.00	Premium	H	SI2	58.9	57.0	12210	8.09	58.9	8.06
24520	1.56	Ideal	G	VS2	62.2	54.0	12800	0.00	0.0	0.00
26243	1.20	Premium	D	VVS1	62.1	59.0	15686	0.00	0.0	0.00
27429	2.25	Premium	H	SI2	62.8	59.0	18034	0.00	0.0	0.00
49189	0.51	Ideal	E	VS1	61.8	55.0	2075	5.15	31.8	5.12
49556	0.71	Good	F	SI2	64.1	60.0	2130	0.00	0.0	0.00
49557	0.71	Good	F	SI2	64.1	60.0	2130	0.00	0.0	0.00

Diamonds: sanity check by comparing to 10 random diamonds

```
diamonds.sample(n=10)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
19430	1.00	Good	F	VVS2	60.7	62.0	8079	6.36	6.40	3.87
34770	0.41	Ideal	D	SI2	61.0	56.0	876	4.83	4.78	2.93
15647	1.00	Premium	E	VS2	59.9	59.0	6272	6.45	6.38	3.84
1220	0.80	Premium	E	SI2	59.9	58.0	2939	6.03	5.96	3.59
38013	0.50	Good	G	SI2	63.8	57.0	1009	4.98	5.02	3.19
17430	1.10	Very Good	F	VS2	61.1	57.0	6987	6.65	6.71	4.08
48469	0.74	Very Good	J	VS2	62.3	55.0	1978	5.79	5.83	3.62
24724	2.03	Very Good	I	SI2	62.8	60.0	13063	7.99	8.05	5.04
50110	0.70	Very Good	G	SI1	63.6	58.0	2209	5.61	5.65	3.58
20009	1.55	Ideal	J	VS1	60.4	57.0	8548	7.52	7.54	4.55

What to do with unusual values?

1. Drop row
2. Code value to `NA`
3. Winsorize value

Diamonds: option 1 for unusual values: drop

```
diamonds_clean = diamonds.loc[(diamonds['y'] >= 3) | (diamonds['y'] <= 20)]
diamonds_clean
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

Diamonds: option 2 for unusual values: missing

```
diamonds['y'] = np.where((diamonds['y'] < 3) | (diamonds['y'] > 20), np.nan, diamonds['y'])
rows_with_na_y = diamonds[diamonds['y'].isna()]
print(rows_with_na_y)
```

```
       carat        cut color clarity  depth  table  price     x   y     z
11963   1.00  Very Good     H     VS2   63.3   53.0   5139  0.00 NaN  0.00
15951   1.14       Fair     G     VS1   57.5   67.0   6381  0.00 NaN  0.00
24067   2.00    Premium     H     SI2   58.9   57.0  12210  8.09 NaN  8.06
24520   1.56      Ideal     G     VS2   62.2   54.0  12800  0.00 NaN  0.00
26243   1.20    Premium     D    VVS1   62.1   59.0  15686  0.00 NaN  0.00
27429   2.25    Premium     H     SI2   62.8   59.0  18034  0.00 NaN  0.00
49189   0.51      Ideal     E     VS1   61.8   55.0   2075  5.15 NaN  5.12
49556   0.71       Good     F     SI2   64.1   60.0   2130  0.00 NaN  0.00
49557   0.71       Good     F     SI2   64.1   60.0   2130  0.00 NaN  0.00
```

Diamonds: option 3 for unusual values: winsorize

Winsorizing re-codes outliers, keeping them in the data. To winsorize at 1 percent: * Replace anything less than the 1st percentile with the 1st percentile * Replace anything more than the 99th percentile with the 99th percentile

```
pctile01 = diamonds['y'].quantile(0.01)
pctile99 = diamonds['y'].quantile(0.99)

print(f"1st Percentile: {pctile01}")
print(f"99th Percentile: {pctile99}")
```

```
1st Percentile: 4.04
99th Percentile: 8.34
```

Diamonds: option 3 for unusual values: winsorize

```
diamonds['y_winsor'] = np.where(diamonds['y'] < pctile01, pctile01,
                                np.where(diamonds['y'] > pctile99, pctile99, diamonds['y']))
diamonds
```

	carat	cut	color	clarity	depth	table	price	x	y	z	y_winsor
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	4.04
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	4.04
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	4.07
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	4.23
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	4.35
...	...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50	5.76
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61	5.75
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56	5.68
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74	6.12
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64	5.87

53940 rows × 11 columns

When is this useful? Income data, test scores, stock returns. Important when you are using procedures where the estimates are sensitive to outliers like computing a mean or running a regression

how do I know which option to choose?

- make an educated guess by looking at the data as many ways as possible
- you often can ask your data provider… but they will quickly grow impatient so try to answer as many questions as possible yourself

Diamonds: what would you do?

- What would you do where `x`, `y`, and `z`?
- What would you do where `y > 20`?

Diamonds: what should we actually do?

My take (there is often not a ``right'' answer or you won't know the answer without talking to a data provider)

- Rows where `x`, `y`, and `z` are all zero: set to `NA`
- Rows where `y > 20`: winsorize? (hard to know for sure…)

Summary: handling unusual numeric values

Problem	Action
Erroneous row	drop row
Erroneous cell	set to NA or winsorize

How do I decide which problem I have? Examine unusual values in context of other columns (same row) and other rows (same columns). We will see this again in a future lecture.

How do I decide whether to set to NA or winsorize? Ideally, ask your data provider what's going on with these values.