Took me a bit to get this data processed; my IPython script that I used is here:
https://github.com/A-Bauman/Thinkful/blob/master/Unit%201/AirBnB%20Challenge_cleanup%20script.ipynb

```
WITH expensive AS (
	SELECT id, MAX(price)
	FROM listings
	GROUP BY id
	HAVING MAX(price) > 3000
	ORDER BY MAX(price) DESC
)
SELECT l.*
FROM expensive e
JOIN listings l ON e.id = l.id;
```

I used the command above to find the most expensive listing; actually, I found the top several. I had to do some verification, as there were errors (fat-fingering) in the data. One was listed for $9000 and another for $8000, but the weekly price was $3200 for one and the other was a single room that normally went for around $100… So, once I did cleanup, here's what I found:

It's an entire home (condo) in the Western Addition neighborhood of San Francisco. It rents for $8000!! However, it accommodates 14, with 3 bathrooms and 6 bedrooms. Has all the amenities, as you might expect. Being this expensive, it doesn't rent often: only 4 reviews, though they're all very high. There were then several properties in the from the mid-$3k to $4k range.

```
WITH neighborhood_cal AS (
	SELECT
		calendar.listing_id,
		listings.neighb_cl hood,
		calendar.available avail

	FROM calendar
	JOIN listings
	ON listings.id = calendar.listing_id
),
neighborhood_availability AS (
	SELECT
		hood,
		SUM(n.avail)/COUNT(n.avail) percent_available

	FROM neighborhood_cal n
	GROUP BY hood
),
neighborhood_reviews AS (
	SELECT
		neighb_cl AS neighborhood,
		avg(revs_per_mon) revs_per_mon,
		count(*) number_of_listings

	FROM listings
	GROUP BY neighborhood
)
SELECT
	r.neighborhood,
	r.revs_per_mon * r.number_of_listings weighted_reviews,
	r.revs_per_mon avg_monthly_reviews,
	r.number_of_listings,
	a.percent_available

FROM neighborhood_reviews r
JOIN neighborhood_availability a
ON r.neighborhood = a.hood
ORDER BY weighted_reviews DESC
```

Neighborhood popularity, determined by the above; you can see some joins, CTE's, and the averaging / weighting of reviews and listings to determine popularity. Here's what I got for the top 10 using this method:

```
"Mission";	1187.7734920253; 1.75966443263007; 675; 0.38803855910705225774
"Outer Sunset"; 926.432900080272; 3.5632034618472; 260; 0.41447839831401475237
"Western Addition"; 829.823936029566; 1.66966586726271; 497; 0.43858769052672197569
"Castro/
 Upper Market"; 774.068065827406; 1.96463976098327; 394; 0.38881858007092691746
"Bernal Heights"; 723.15459844668; 2.0143582129434; 359; 0.32891975426412790476
"South of Market"; 663.008312566105; 1.52766892296338; 434; 0.54135471245502177893
"Noe Valley";	586.058682057925; 1.89663003902241; 309; 0.41442567717338298533
"Haight Ashbury"; 496.947759861099; 1.65649253287033; 300; 0.47032876712328767123
"Downtown/
 Civic Center"; 495.32999946503; 1.07214285598491; 462; 0.35653798256537982565
"Inner Richmond"; 463.493332156756; 2.22833332767671; 208; 0.43847471022128556375
```

```
WITH monthly_reviews AS (
	SELECT
		extract(month from review_date) AS month,
		COUNT(*) number_of_reviews

	FROM reviews
	GROUP BY month
),
monthly_price AS (
        SELECT
                extract(month from cal_date) AS month,
                SUM(available)/COUNT(available) percent_available,
                avg(price) avg_price

        FROM calendar
        GROUP BY month
)

SELECT
        r.month,
        r.number_of_reviews,
        ROUND(p.avg_price, 2),
        ROUND(p.percent_available, 4)

FROM monthly_reviews r
JOIN monthly_price p
ON r.month = p.month
ORDER BY avg_price ASC
--ORDER BY number_of_reviews DESC
```

It looks like September is the cheapest time of year, averaging $81, followed by April and June (each at $95). In terms of being busy, August is busiest, followed by July.

```
Month Reviews Price           Availability

9       24397       81.00         0.2578
6       27542       95.43         0.4011
4       23091       95.63         0.4192
7       31631       97.36         0.3998

8       35054       98.05         0.4051
7       31631       97.36         0.3998
5       27744       98.25         0.4278
6       27542       95.43         0.4011
```