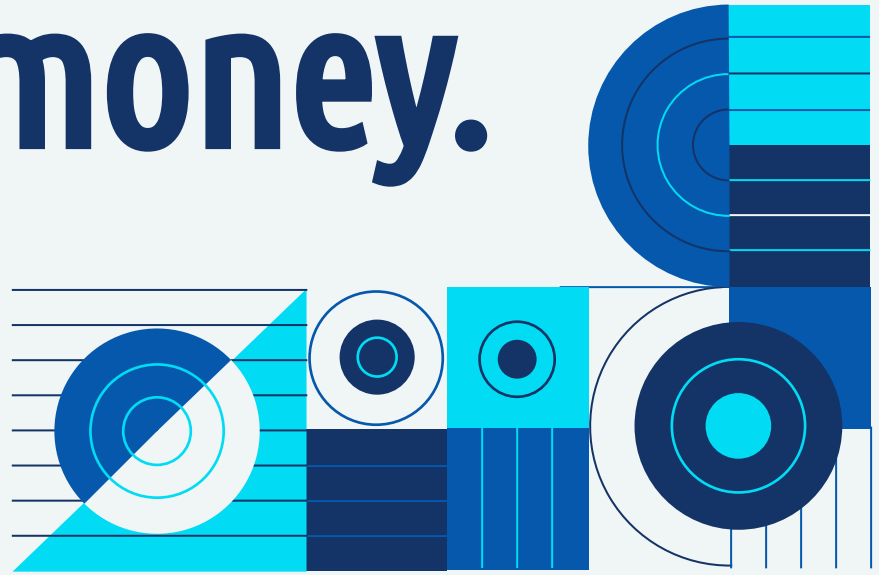
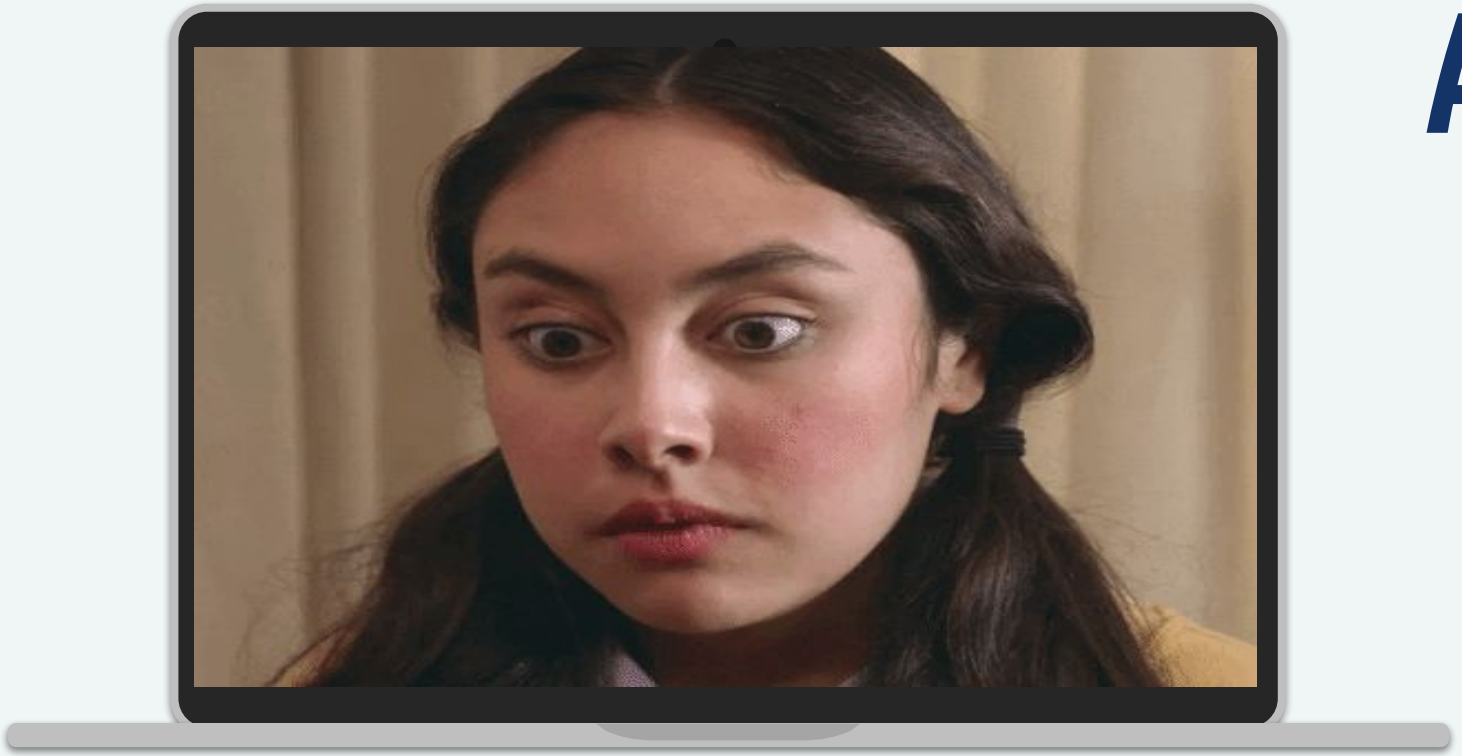
Let’s make data based money.



Sneak peak on the data treasure .

- BIG ASS NUMBERS.
- Show findings between luxury and non-luxury properties.
- Explore data with SQL and python.
- Machine Learning: LinearRegression vs KNN Model.

LOOK AT THE BIG AS NUMBERS...



AND THE BIGGER THE BETTER

BIG ASS NUMBERS

21420

PROPERTIES

BIG ASS NUMBERS

7,700,000

Price for the most expensive ....

BIG ASS NUMBERS

44,928,711

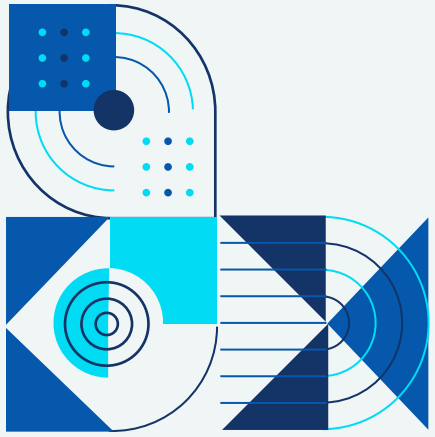
Sqft living

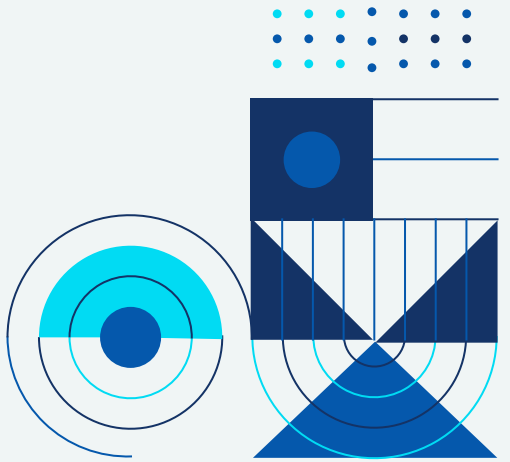
72,851

bedrooms

45,695.5

bathrooms





Very interesting! I know ..

But what is more interesting?

EXACTLY.

HOW CAN WE MAKE MONEY?

We put properties in two categories

NON-Luxury:
properties below 650k

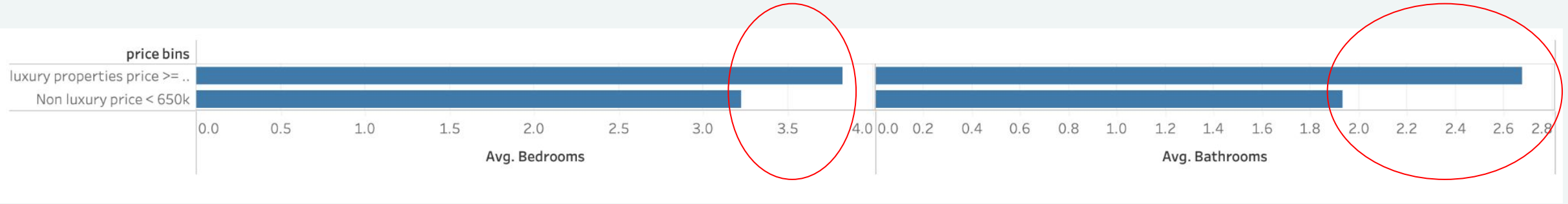
VS

Luxury:
properties starting at 650k

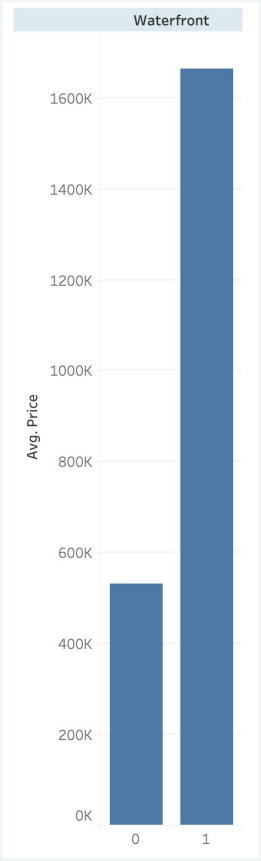
And people looooooove to pay for …

many bedrooms

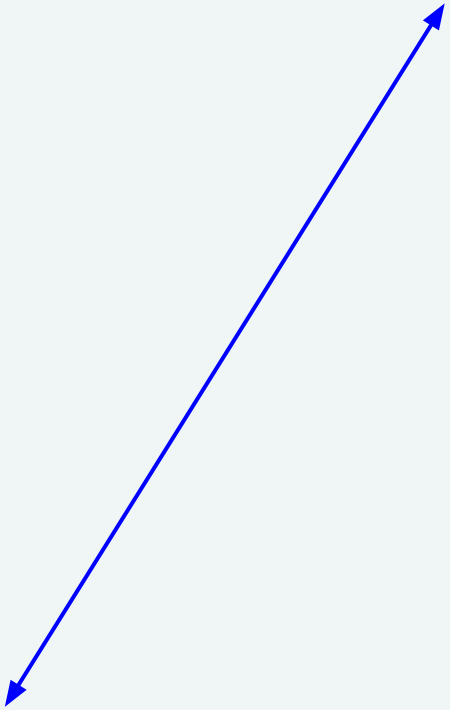
also bathrooms



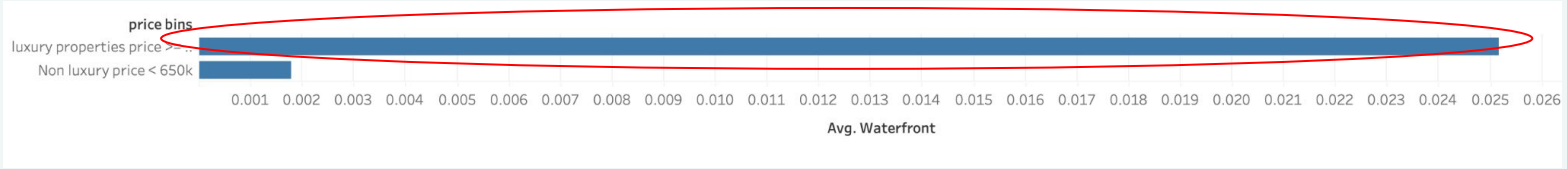
And for sure …



See the average price for properties with waterfront



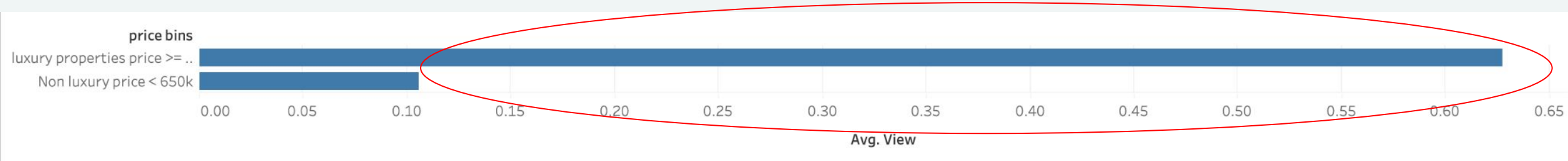
And the share related to luxury vs non-luxury



WATERFRONT

They loving living near the water because it is so refreshing

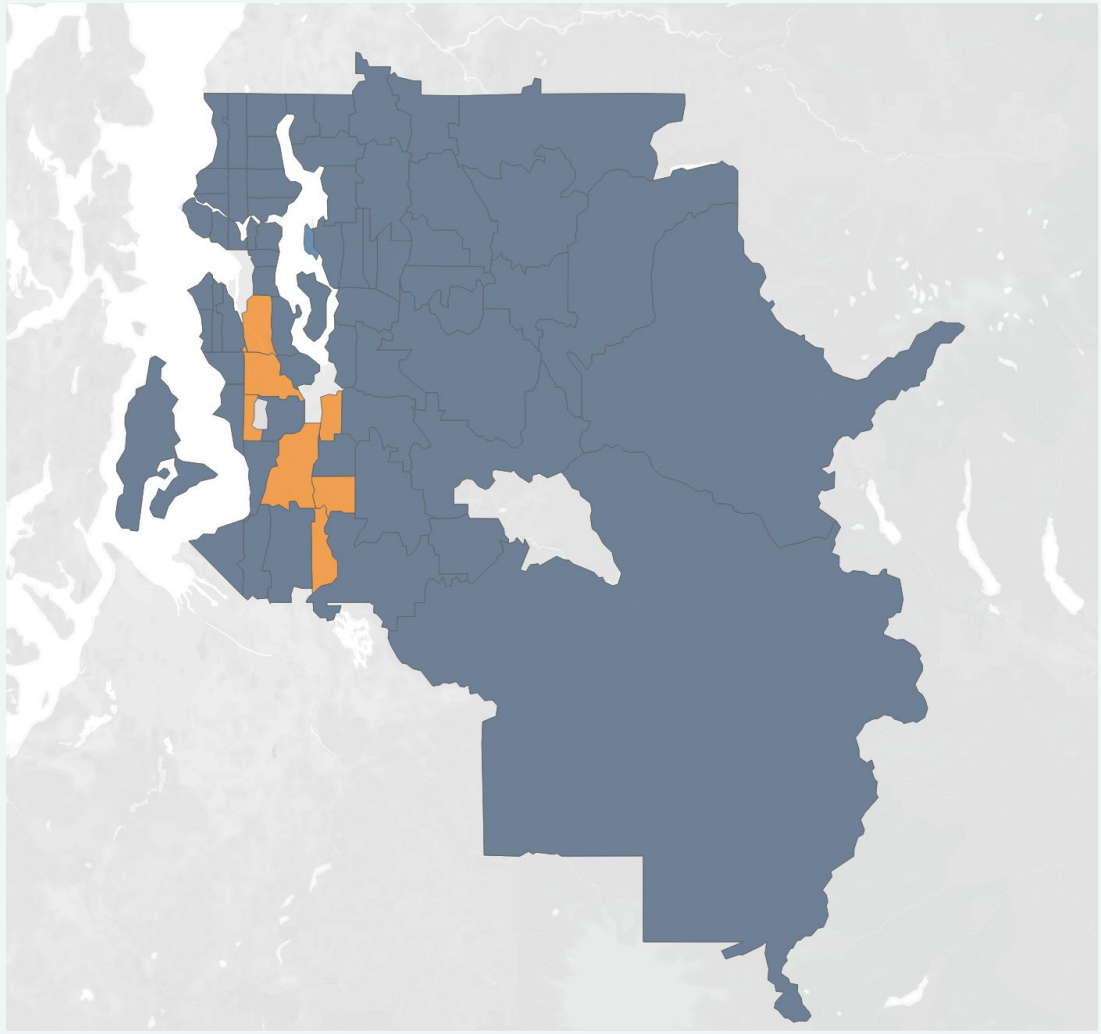
Ah, sorry .. I forgot also a niiiice view …



view

There are more kangaroos than humans in Australia.

Super important: money avoids the orange zip codes



And also with SQL

```sql
#8. Arrange the data in a decreasing order by the price of the house.
# Return only the IDs of the top 10 most expensive houses in your data.
SELECT
    id
FROM
    house_price_data
ORDER BY
    price DESC
LIMIT 10;
```

id
6762700020
9808700762
9208900037
2470100110
8907500070
7558700030
1247600105
1924059029
7738500731
3835500195

```sql
#9. What is the average price of all the properties in your data?
# -> round to to decimals, looks nicer
SELECT
    ROUND(AVG(price), 2)
FROM
    house_price_data;
```

avg_price
540739.30

```sql
#10.1 What is the average price of the houses grouped by bedrooms?
#The returned result should have only two columns, bedrooms and Average of the prices.
#Use an alias to change the name of the second column.
# I also ordered by bedroom to see the difference of average price better
SELECT
    bedrooms,
    ROUND(AVG(price), 2) as avg_price
FROM
    house_price_data
GROUP BY
    bedrooms
ORDER BY
    bedrooms;
```

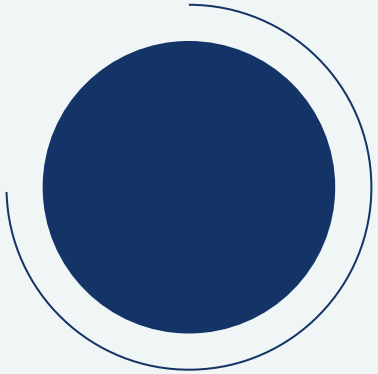
bedrooms	avg_price
1	318293.37
2	400938.86
3	466527.48
4	635794.49
5	788270.16
6	833344.16
7	951447.82
8	1105076.92
9	893999.83
10	820000.00
11	520000.00
33	640000.00

```sql
#10.3 What is the average price of the houses with a waterfront and without a waterfront?
# The returned result should have only two columns, waterfront and Average of the prices.
# Use an alias to change the name of the second column.
SELECT
    waterfront,
    ROUND(AVG(price), 2) as avg_price
FROM
    house_price_data
GROUP BY
    waterfront;
```

waterfront	avg_price
0	532137.39
1	1662524.18

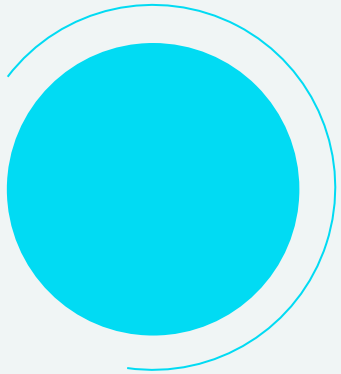
With SQL we see the same results in a more technical way

Sum this up!
publish the money formula ...



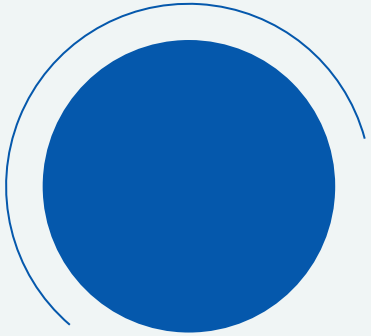
WATERFRONT

Build and sell things near the water ...



VIEW

... with an awesome view ...



SPACE

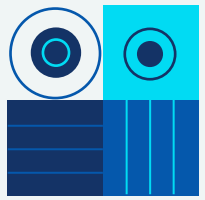
... and 2.5 bathrooms with at least 3 but better 4 bedrooms.

Pssst .. don't forget to avoid some areas:

98108, 98168, 98148, 98032, 98030, 98002, 98005

Talk about the technical part . . .

Step-by-step to machine learning model



Explore

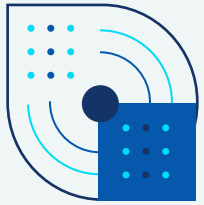
Exploring the data to understand the columns/features with python and sql

Clean

Clean the data to feed our model with nice, fresh data

Prepare

Prepare the data, split out the target for the models



Models

Use Linear Regression Model and KNN Model

Results

Talk about
- R2
- RMSE
- MSE
- MAE

Explore with python ...

Explore

```
In [3]:
1 #have a look at the data - limit to 5
2 df.head()
```

Out[3]:

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	...	sqft_above	sqft_basement	yr_built	yr_renovated	zipc
0	7129300520	2014-10-13	3	1.00	1180	5650	1.0	0	0	3	...	1180	0	1955	0	98
1	6414100192	2014-12-09	3	2.25	2570	7242	2.0	0	0	3	...	2170	400	1951	1991	98
2	5631500400	2015-02-25	2	1.00	770	10000	1.0	0	0	3	...	770	0	1933	0	98
3	2487200875	2014-12-09	4	3.00	1960	5000	1.0	0	0	5	...	1050	910	1965	0	98
4	1954400510	2015-02-18	3	2.00	1680	8080	1.0	0	0	3	...	1680	0	1987	0	98

5 rows × 21 columns

Check dtypes, columns, entries, NAN counts, memory usage

```
2 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  datetime64[ns]
 2   bedrooms       21597 non-null  int64
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64
 5   sqft_lot       21597 non-null  int64
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  int64
 8   view           21597 non-null  int64
 9   condition      21597 non-null  int64
 10  grade          21597 non-null  int64
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21597 non-null  int64
 13  yr_built       21597 non-null  int64
 14  yr_renovated   21597 non-null  int64
 15  zipcode        21597 non-null  int64
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64
 19  sqft_lot15     21597 non-null  int64
 20  price          21597 non-null  int64
dtypes: datetime64[ns](1), float64(4), int64(16)
memory usage: 3.5 MB
```

```
1 #double check nan-values
2 df.isna().sum()
```

```
id               0
date             0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
price            0
dtype: int64
```

Have a look at the data

Explore all columns/features and see if values are corrected formated

0% NAN

You can double click on the desired country and change fill color.

Cleaning, dropping, plotting, splitting and preparing …

cleaning

New: property_age (subtract yr_built from date), deleted duplicates,

plotting

Plot distribution with seaborn

dropping

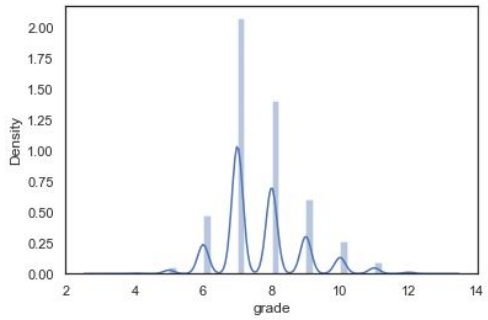
Yr_renovated, lat, long, date, yr_build

preparing

Split in numerical and categorical, “normalize” with minmaxscaler, concat again, split train test data, use the model

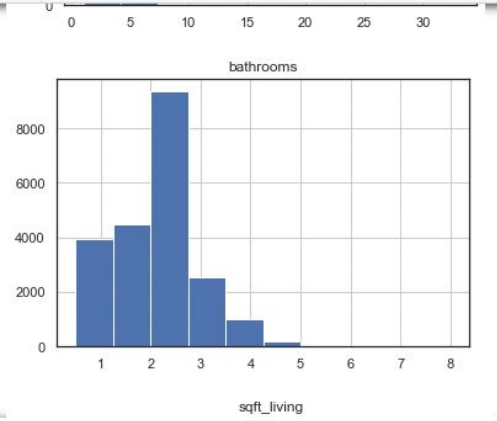
Plotting distribution

In [39]:
```
#see distribution with seaborn
for column in df_latest_sales:
    sns.distplot(df_latest_sales[column])
    plt.show()
```




```
/Users/Xaver/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fi
```

In [40]:
```
#see distribution with matplotlib
for column in df_latest_sales:
    df_latest_sales[column].hist()
    plt.title(column)
    plt.show()
```



Delete double sales … keep only the latest sale per ID

In [27]:
```
#first create a data frame wiht NO duplicates called no_duplicates
no_duplicates = df[~df.duplicated(['id'])]
```

In [28]:
```
no_duplicates.shape
```

Out[28]: (21420, 17)

In [29]:
```
#second create a data frame with duplicates and keep just the latest sales
df_duplicates = df[df.duplicated(['id'], keep='first')].sort_values(['id', 'price'])
```

In [30]:
```
df_duplicates.shape
```

Out[30]: (177, 17)

In [31]:
```
#concat both we should end up with 21,597 rows
df_latest_sales = pd.concat([no_duplicates, df_duplicates], axis=0)
```

In [32]:
```
df_latest_sales.head()
```

Out[32]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	zipcode	sqft_living15	sqft_lot
0	7129300520	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	98178	1340	56
1	6414100192	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	98125	1690	76
2	5631500400	2	1.00	770	10000	1.0	0	0	3	6	770	0	98028	2720	80
3	2487200875	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	98136	1360	50
4	1954400510	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	98074	1800	75

In [33]:
```
df_latest_sales.shape
```

Out[33]: (21597, 17)

Models tested

LinearRegression

KNN

	R2	RMSE	MSE	MAE
LinearRegression	0.81	152304.45	23196646248.88	95901.13
KNN	0.48	251496.74	63250608562.19	150974.79

We take LinearRegression Model

Because we have the highest R2 and the lowest:
RMSE, MSE, MAE

Thank you!

Do you have any questions?

If you liked it:
Follow me on instagram for more :D :D :D