Homework 1 Data Exploration

Kevin Gao, Djounia Saint-fleurant

Finished on: 20230208

Assigned on: 20230201

- Exercise 1. House Sales in Kings County - This data set contains house sale prices for King County, which includes Seattle, Washington. It includes homes sold between May 2014 and May 2015.
- Exercise 2. Diamonds - This data set contains prices and other attributes for nearly 54,000 diamonds. Figures 1, 2, and 3 show the dimensions, color, and clarity used in the diamonds data set, respectively.

For each of the data sets, do the following:

1. Create a Jupyter notebook to read the data set.
2. Conduct an exploratory analysis, so as to best understand the data set's features.
3. If data is missing, make assumptions about the missing data and impute values as you deem appropriate.

As we can see below, there is no missing value in the dataset. However, if there is missing value, I would use the average of similar items(houses or diamonds to fill the gap)

4. After looking at the data set, make assumptions about the drivers of prices (houses in Exercise 1 and diamonds in Exercise 2) and see if your assumptions are valid or invalid.
5. Use charts, tables, or whatever you deem appropriate.

Solutions to problem 1, 2, 4, 5 are below.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
House = pd.read_csv('kc_house_data.csv')
```

In [3]:
```
diamonds = pd.read_csv('diamonds.csv')
```

Data exploration of house price dataset.

In [4]:
```
House.dtypes
```

Out[4]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

Data description:

id - Unique ID for each home sold

date - Date of the home sale

price - Price of each home sold

bedrooms - Number of bedrooms

bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower

sqft_living - Square footage of the apartments interior living space

sqft_lot - Square footage of the land space

floors - Number of floors

waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not

view - An index from 0 to 4 of how good the view of the property was

condition - An index from 1 to 5 on the condition of the apartment,

grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of

construction and design, and 11-13 have a high quality level of construction and design.

sqft_above - The square footage of the interior housing space that is above ground level

sqft_basement - The square footage of the interior housing space that is below ground level

yr_built - The year the house was initially built

yr_renovated - The year of the house's last renovation

zipcode - What zipcode area the house is in

lat - Lattitude

long - Longitude

sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors

sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

[Here (https://www.slideshare.net/PawanShivhare1/predicting-king-county-house-prices)](https://www.slideshare.net/PawanShivhare1/predicting-king-county-house-prices) is the source of the description.

In [5]:
```
House.head()
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_ab
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1

5 rows × 21 columns

As we can see, the date is in the format of 'YYYYMMDDT000000', then we can extract the data from the original data.

In [6]:
```
def dateyear(x):
    return x[0:4]
def datemonth(x):
    return x[4:6]
```

In [7]:
```
House['year'] = House['date'].apply(dateyear)
House['month'] = House['date'].apply(datemonth)
```

In [8]:
```
House.head()
```

Out[8]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	sqft_basement
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	400
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	910
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	0

5 rows × 23 columns

In [9]:
```
House.isnull().sum()
```
Out[9]:
```
id               0
date             0
price            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
year             0
month            0
dtype: int64
```

There is no missing value in the datasset.

Assumption 1 The house prices v.s. time

First assumption is that the prices are different among years, and has a seasonality inside. So I draw a bar graph

Assumption 1.1 The overall house prices are different across seasons and months

In [10]:
```
price_year_month = House.groupby(['year', 'month'])['price'].mean().unstack()
```
In [11]:
```
price_year_month
```
Out[11]:

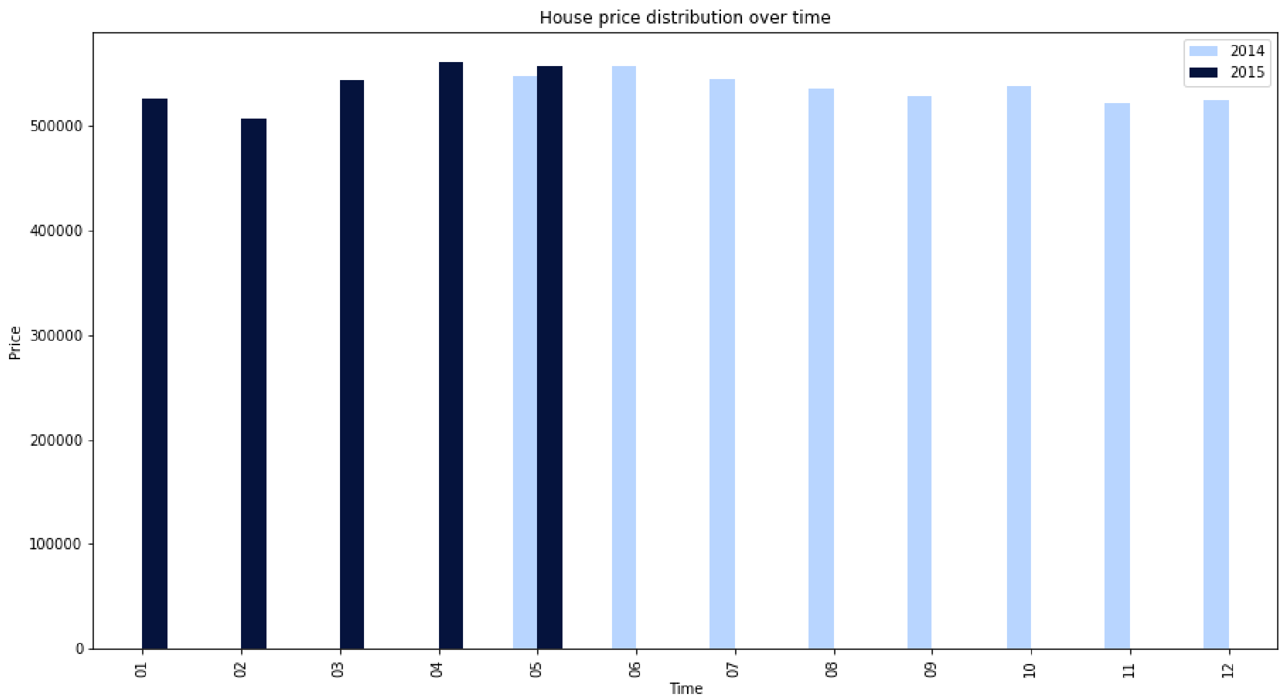
month year	01	02	03	04	05	06	07	08	
2014	NaN	NaN	NaN	NaN	548080.276584	558002.199541	544788.76436	536445.276804	5292
2015	525870.889571	507851.3712	543977.1872	561837.774989	558126.811146	NaN	NaN	NaN	

In [12]:
```
House['date'].max()
```
Out[12]:
```
'20150527T000000'
```

As we can see, the data is ranged from *201405* to *201505*, therefore we cannot figure out anything about seasonality, but we can try to figure out whethter the prices are different among the months.

In [13]:
```python
fig, axes = plt.subplots()
house_month = House.groupby(['month', 'year'])['price'].mean().unstack()
house_month.fillna(0, inplace = True)
house_month.plot(kind = 'bar', figsize = (15, 8), ax = axes, color = ['#B8D5ff', '#05133d'])
axes.legend(["2014", "2015"])
axes.set_ylabel("Price")
axes.set_xlabel("Time")
axes.set(title="House price distribution over time")
```

Out[13]: [Text(0.5, 1.0, 'House price distribution over time')]



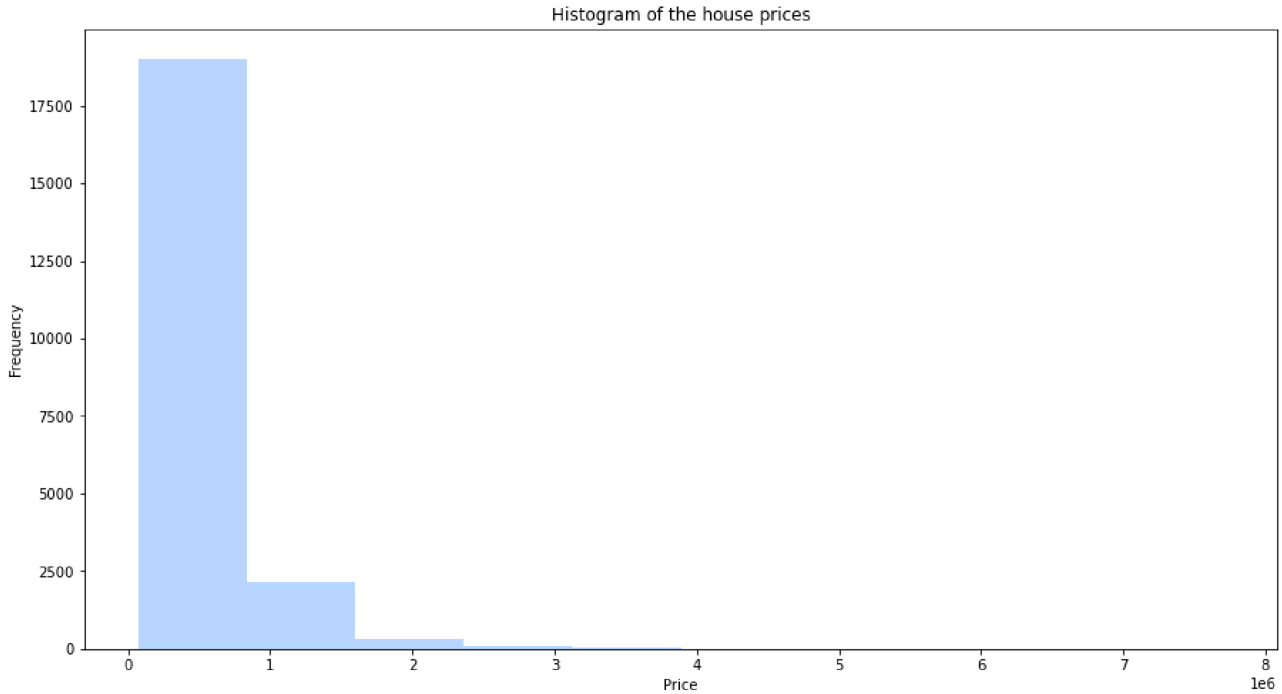
Assumption 1.1 disproved

Assumption 1.2: for some groups of population, the price is time-sensitive.

In total, the price doesn't change with time. However, people of different income have different demoandings for houses, so I would break the dataset into several subsets based on prices.

In [14]:
```
plt.figure(figsize = (15, 8))
plt.hist(House['price'], color = '#b8d5ff')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Histogram of the house prices')
```

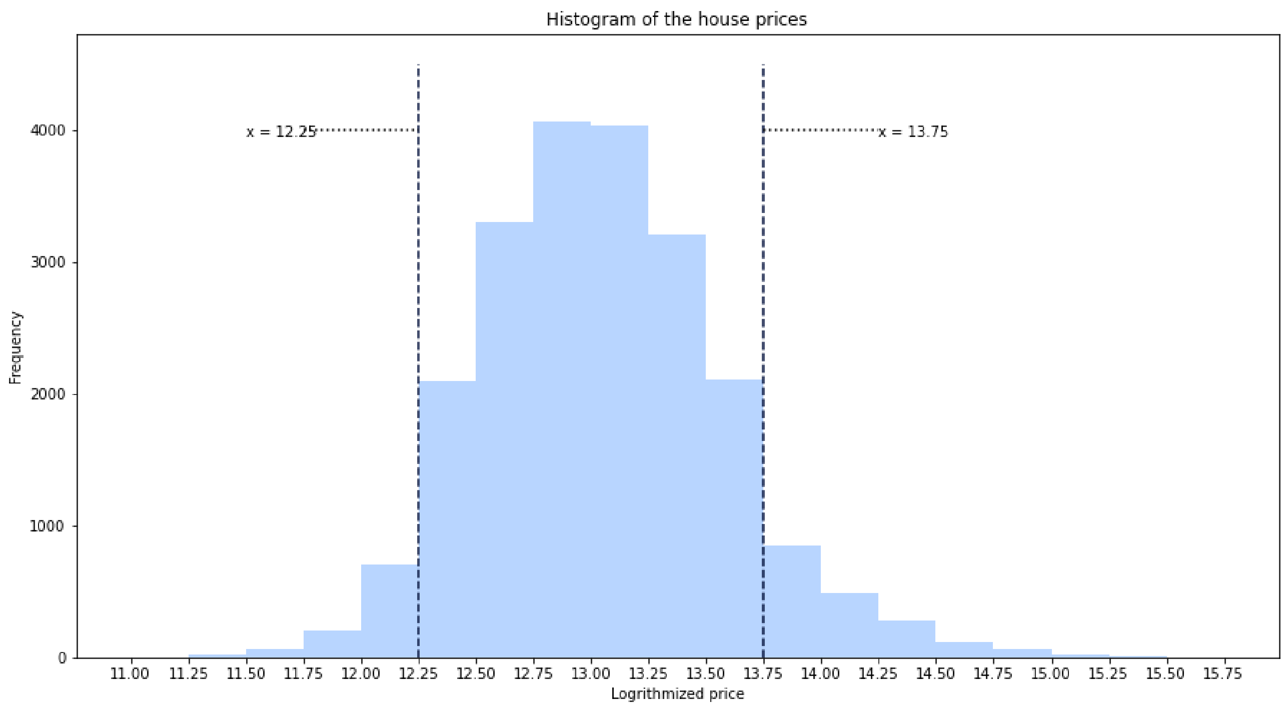
Out[14]: `Text(0.5, 1.0, 'Histogram of the house prices')`



Clearly, there are extreme values in the dataset. So I would like to conduct a logrithm transformation on the price dataset and to figure out the threshold.

In [15]:
```
plt.figure(figsize = (15, 8))
plt.hist(np.log(House['price']), color = '#b8d5ff', bins = np.arange(11, 16, .25))
plt.xlabel('Logrithmized price')
plt.xticks(np.arange(11, 16, .25))
plt.vlines(x = np.array([12.25, 13.75]), ymin = 0, ymax = 4500, linestyle = 'dashed', color = '#0!
plt.hlines(y = 4000, xmin = 11.75, xmax = 12.25, linestyle = 'dotted', color = 'k')
plt.hlines(y = 4000, xmin = 13.75, xmax = 14.25, linestyle = 'dotted', color = 'k')
plt.annotate(text = 'x = 12.25', xy = (12.25, 4000), xytext = (11.5, 3950))
plt.annotate(text = 'x = 13.75', xy = (13.75, 4000), xytext = (14.25, 3950))
plt.ylabel('Frequency')
plt.title('Histogram of the house prices')
```

Out[15]:
```
Text(0.5, 1.0, 'Histogram of the house prices')
```



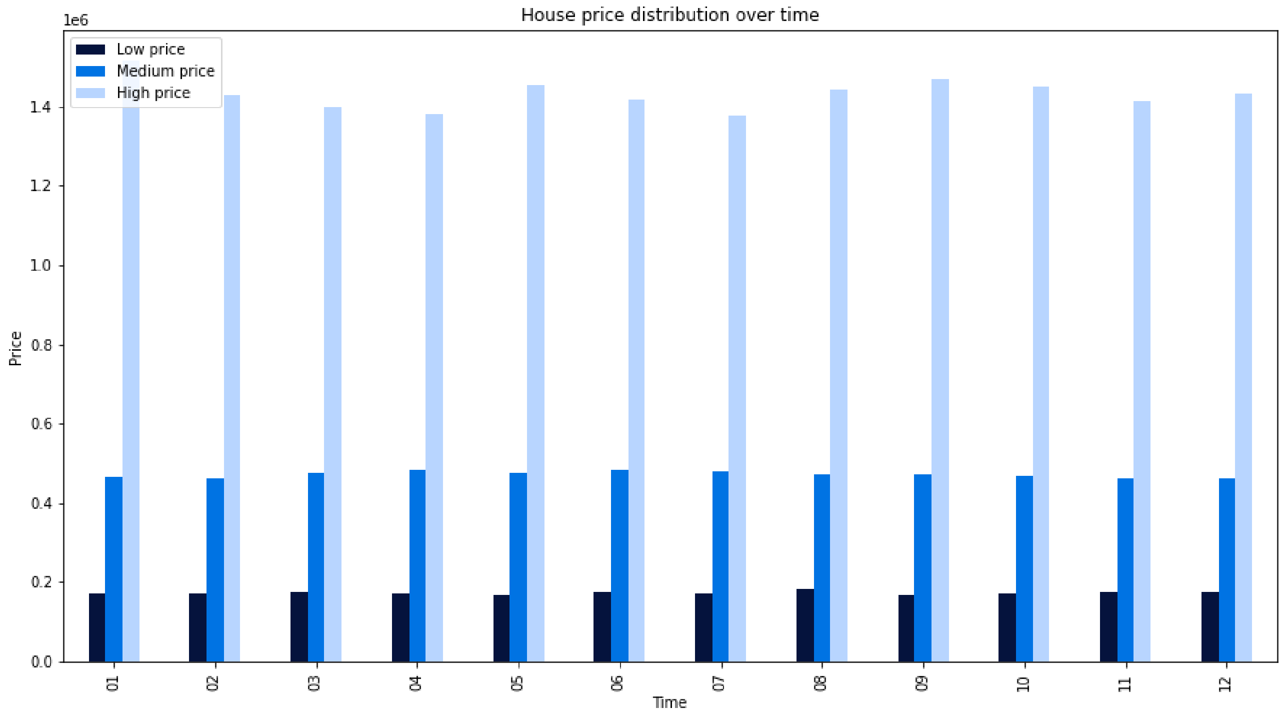
The histogram shows clear gaps at $x = 12.25$ and $x = 13.75$, So I would choose $e^{12.25} \approx 208980$ and $e^{13.75} \approx 936589$ as the thesholds

In [16]:
```
House['category'] = (House['price']> 208980).astype(int) + (House['price'] > 936589).astype(int)
```

Because the house price in May 2014 and May 2015 does not change much, I will gather the two month's data into one part

In [17]:
```
fig, axes = plt.subplots()
house_month = House.groupby(['month', 'category'])['price'].mean().unstack()
house_month.fillna(0, inplace = True)
house_month.plot(kind = 'bar', figsize = (15, 8), ax = axes, color = ['#05133d', '#0073e3', '#B8D
axes.legend(['Low price', 'Medium price', 'High price'])
axes.set_ylabel("Price")
axes.set_xlabel("Time")
axes.set(title="House price distribution over time")
```

Out[17]: [Text(0.5, 1.0, 'House price distribution over time')]



Assumption 1.2 disproved

Therefore based on the dataset, we found that the house prices do not vary across time, no matter whether we consider the price level or not.

Assumption 2: Location and the price.

First I tried to classify the house based on the zipcode.

In [18]:
```
House['zipcode'].unique().shape
```

Out[18]: (70,)

There are 70 zipcodes in the King County, which is a huge number to classify, then I decide to do some clustering based on the longitudes and lattitudes.

In [19]:
```
House1 = House[House['category'] == 1]
House2 = House[House['category'] == 2]
House3 = House[House['category'] == 3]
```

In [20]:
```
House['price_normalized'] = np.log(House['price'] + 1)
```

In [21]:
```
def hex_to_RGB(hex_str):
    """ #FFFFFF -> [255,255,255]"""
    #Pass 16 to the integer function for change of base
    return [int(hex_str[i:i+2], 16) for i in range(1,6,2)]

def get_color_gradient(c1, c2, n):
    """
    Given two hex colors, returns a color gradient
    with n colors.
    """
    assert n > 1
    c1_rgb = np.array(hex_to_RGB(c1))/255
    c2_rgb = np.array(hex_to_RGB(c2))/255
    mix_pcts = [x/(n-1) for x in range(n)]
    rgb_colors = [((1-mix)*c1_rgb + (mix*c2_rgb)) for mix in mix_pcts]
    return ["#" + "".join([format(int(round(val*255)), "02x") for val in item]) for item in rgb_co
```

In [22]:
```
House.shape
```

Out[22]:
```
(21613, 25)
```

In [23]:
```
-122.519-.05
```

Out[23]:
```
-122.569
```

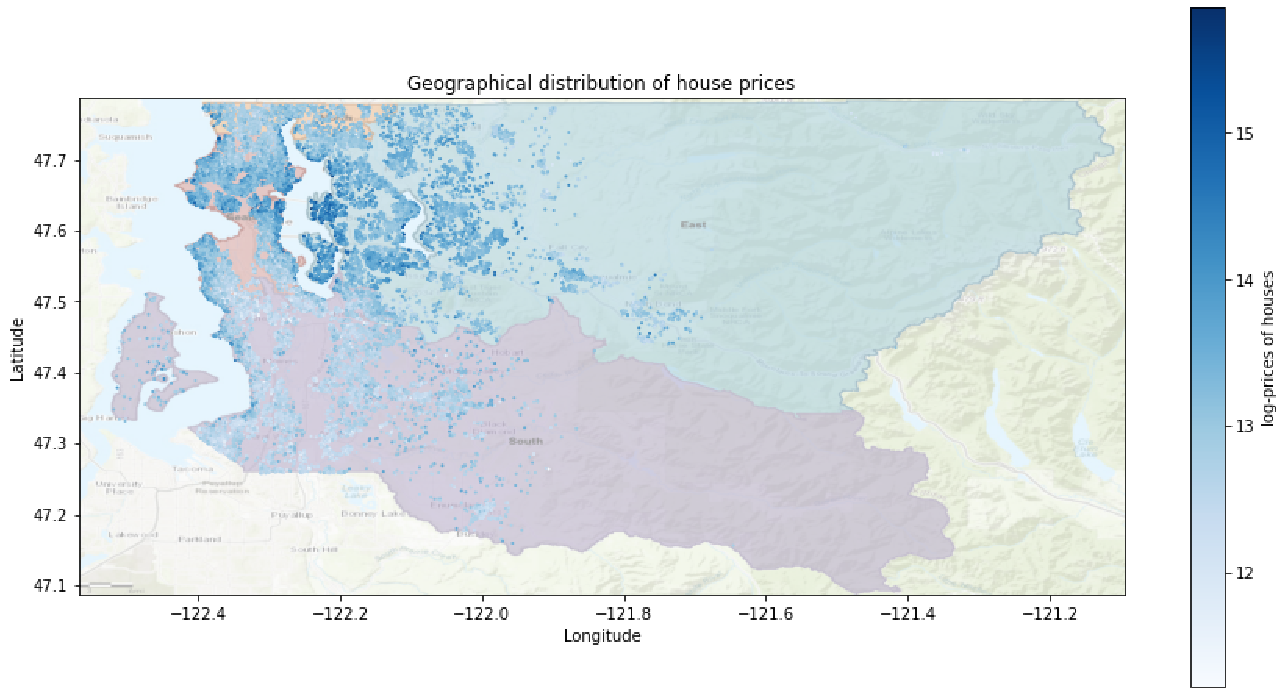
In [24]:
```
(House['price_normalized'] - House['price_normalized'].min())
```

Out[24]:
```
0        1.084730
1        1.970359
2        0.875461
3        2.086074
4        1.916911
           ...
21608    1.568605
21609    1.673966
21610    1.679204
21611    1.673966
21612    1.466327
Name: price_normalized, Length: 21613, dtype: float64
```

In [25]:
```
def minmax(series):
    return (series - series.min()) / (series.max() - series.min())
```

```
In [26]: img = plt.imread('King_County_map.png')
fig, ax = plt.subplots(figsize = (15, 8))
ax.imshow(img, extent= [-122.569, -121.094, 47.0859, 47.78619], alpha = .5)
cmap = get_color_gradient('#B8D5FF', '#05133D', 21613)
plt.set_cmap('Blues')
# plt.figure(figsize = (15, 8))
plt.scatter(x = House['long'], y = House['lat'], c = House['price_normalized'], s= .9)
plt.title('Geographical distribution of house prices')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar(label = 'log-prices of houses')
```

```
Out[26]: <matplotlib.colorbar.Colorbar at 0x7fc48249a790>
```



As we can see, houses of higher prices are clustered in th e northwestern part in King County.

Therefore, we proved the **assumption 2: House prices is related to the grographical location, houses of higher prices are clustered in th e northwestern part of King County.**

In a further investigation, we found that the northwestern part of King County is next to the open water (continuous light blue in the map), it might be the reason why the price is higher. On the other hand, the prices are higher may also be a result of its location in Seatle (crimson part in the western part of map).

Assumption 3: The utilities in the house is correlated with prices, with better utilities, the prices of the houses would be higher.

Because the utilities contain a lot of factors, I decide to use regression model to figure whether this is true. Here I use `bedrooms` , `bathrooms` , `sqft_living` , `sqft_lot` , `floors` , `waterfront` , `sqft_above` and `sqft_basement` to measure the utilities quantitatively.

```
In [27]: from statsmodels.formula.api import ols
```

Because the price is exponentially distributed, if we regress price directly on other variables, the model would not be able to estimate the prices properly. So I used logrithm term of price.

In [28]:
```
model = ols('price_normalized ~ bedrooms + bathrooms + sqft_living + sqft_lot + \
floors + waterfront + sqft_above + sqft_basement', data = House).fit()
model.summary()
```

Out[28]:

OLS Regression Results

Dep. Variable:	price_normalized	**R-squared:**	0.506
Model:	OLS	**Adj. R-squared:**	0.506
Method:	Least Squares	**F-statistic:**	3165.
Date:	Tue, 21 Feb 2023	**Prob (F-statistic):**	0.00
Time:	18:18:02	**Log-Likelihood:**	-9181.8
No. Observations:	21613	**AIC:**	1.838e+04
Df Residuals:	21605	**BIC:**	1.844e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	12.2249	0.011	1095.545	0.000	12.203	12.247
bedrooms	-0.0472	0.003	-13.897	0.000	-0.054	-0.041
bathrooms	0.0239	0.006	4.333	0.000	0.013	0.035
sqft_living	0.0003	3.28e-06	83.423	0.000	0.000	0.000
sqft_lot	-1.982e-07	6.25e-08	-3.170	0.002	-3.21e-07	-7.56e-08
floors	0.0892	0.006	14.374	0.000	0.077	0.101
waterfront	0.5928	0.029	20.171	0.000	0.535	0.650
sqft_above	9.834e-05	3.36e-06	29.253	0.000	9.17e-05	0.000
sqft_basement	0.0002	4.55e-06	38.583	0.000	0.000	0.000

Omnibus:	7.933	**Durbin-Watson:**	1.979
Prob(Omnibus):	0.019	**Jarque-Bera (JB):**	8.344
Skew:	0.021	**Prob(JB):**	0.0154
Kurtosis:	3.086	**Cond. No.**	1.42e+17

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.1e-21. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

As mentioned above, there are some multicollinearity problem in the model, so I plot a correlation matrix below.

In [29]:
```
utility = House[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'sqf
```

In [30]:
```
utility['normalized_price'] = House['price_normalized']
```

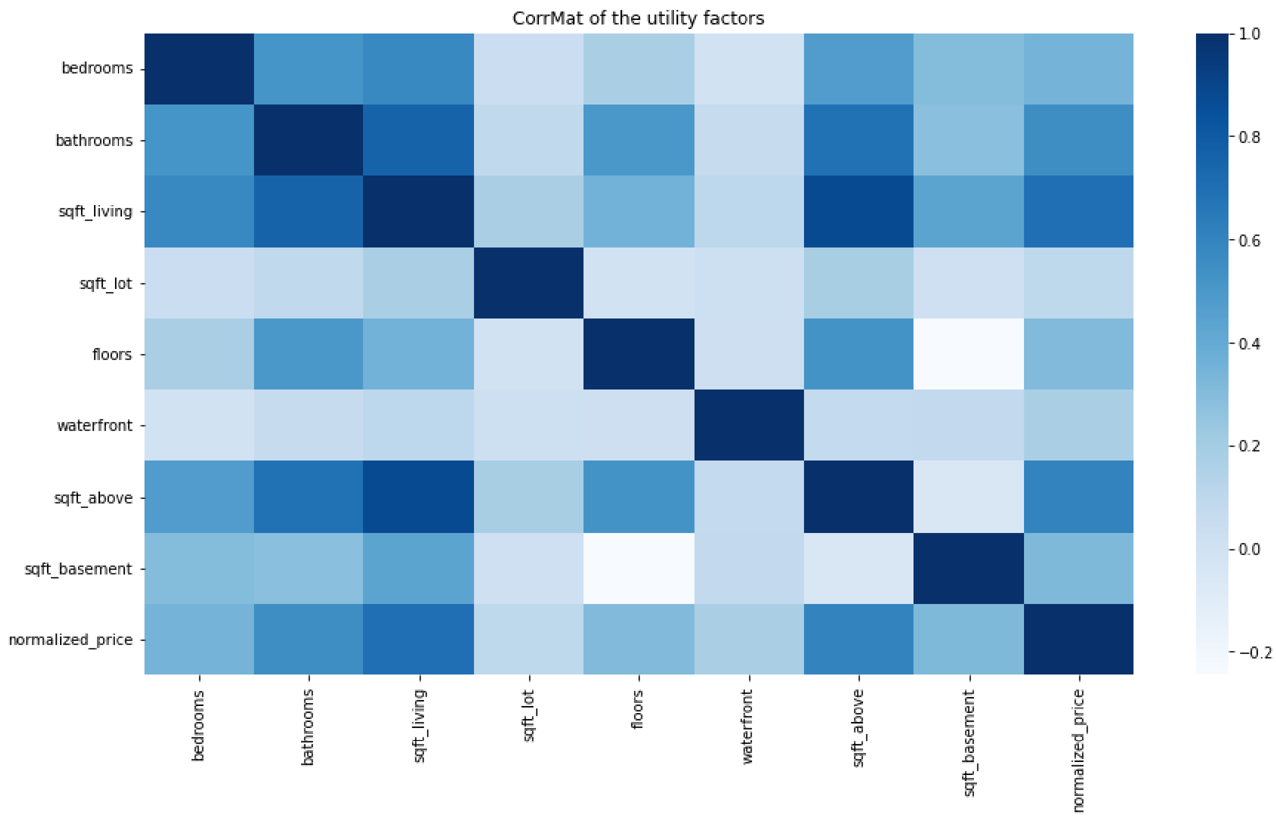
```
/var/folders/1j/d9ymnnfx2j1249hwwzs12qw80000gn/T/ipykernel_70678/3768307260.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
  utility['normalized_price'] = House['price_normalized']
```

In [31]:
```
cor_mat1 = utility.corr()
```

In [32]:
```
import seaborn as sns
```

In [33]:
```
plt.figure(figsize = (15, 8))
plt.title('CorrMat of the utility factors')
sns.heatmap(cor_mat1, cmap = 'Blues')
```

Out[33]:
```
<AxesSubplot:title={'center':'CorrMat of the utility factors'}>
```



- The size of living area is highly correlated with the number fo the bedrooms, it makes sense logically because bedrooms make up a huge proportion for the living area.
- The size of living area is highly correlated with the size of the square above ground, for people mainly near outside the basement.

Obviously we can prove **Assumption 3: the house price is positively correlated with the utility in the house**

Assumption 4: the house price is positively correlated with the outlook of the building.

To measure the outlooking of the building, we can choose the `View` , `Condition` , `Grade` , `Sqft_living15` , `Sqft_lot15` . Similarly, I want to a regression model to investigate my assumption quantitatively.

In [34]:
```
model = ols('price_normalized ~ view + condition + grade + sqft_living15 + sqft_lot15', data = Hou
model.summary()
```

Out[34]:

OLS Regression Results

Dep. Variable:	price_normalized	**R-squared:**	0.563
Model:	OLS	**Adj. R-squared:**	0.563
Method:	Least Squares	**F-statistic:**	5570.
Date:	Tue, 21 Feb 2023	**Prob (F-statistic):**	0.00
Time:	18:18:03	**Log-Likelihood:**	-7861.1
No. Observations:	21613	**AIC:**	1.573e+04
Df Residuals:	21607	**BIC:**	1.578e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	10.5042	0.023	463.516	0.000	10.460	10.549
view	0.1017	0.003	31.372	0.000	0.095	0.108
condition	0.1052	0.004	28.501	0.000	0.098	0.112
grade	0.2410	0.003	83.005	0.000	0.235	0.247
sqft_living15	0.0002	5.03e-06	32.091	0.000	0.000	0.000
sqft_lot15	-4.118e-07	8.83e-08	-4.665	0.000	-5.85e-07	-2.39e-07

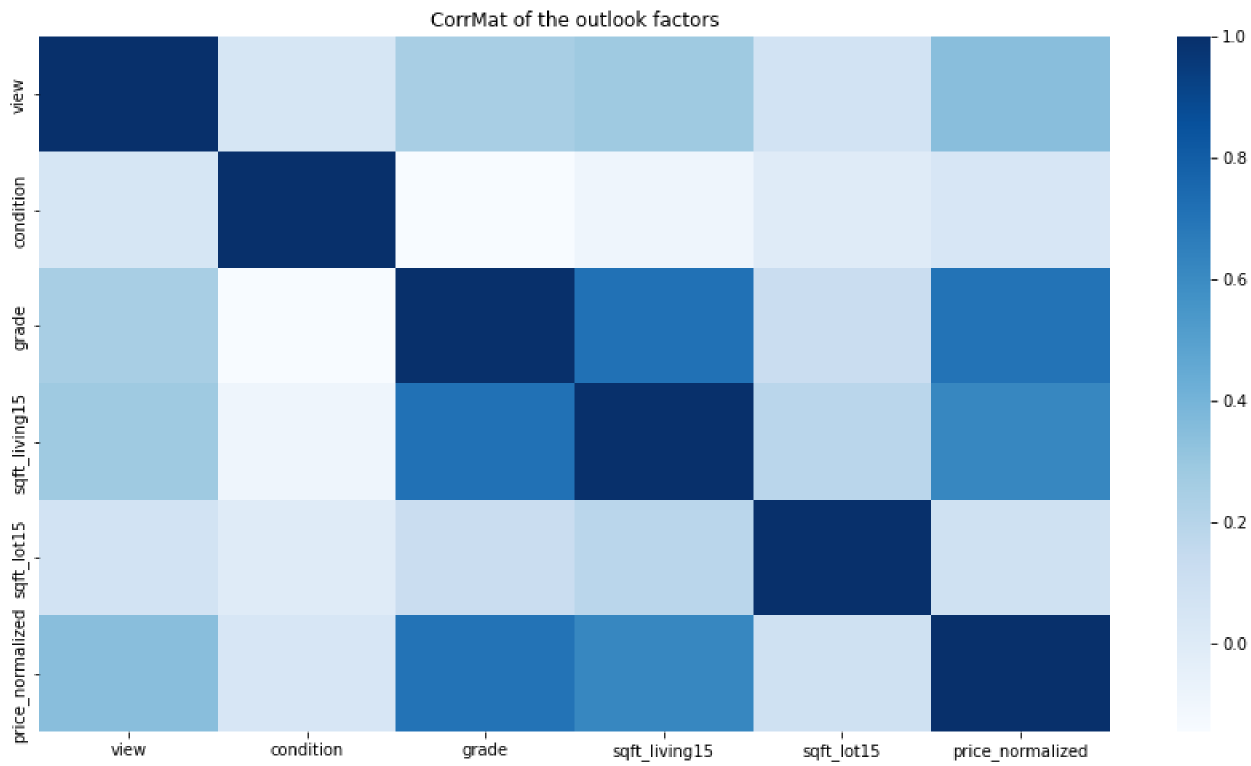
Omnibus:	13.153	**Durbin-Watson:**	1.970
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	13.188
Skew:	0.060	**Prob(JB):**	0.00137
Kurtosis:	2.982	**Cond. No.**	2.92e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.92e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [35]:
```
Outlook = House[['view', 'condition', 'grade', 'sqft_living15', 'sqft_lot15', 'price_normalized']
corr2 = Outlook.corr()
```

In [36]:
```
plt.figure(figsize = (15, 8))
plt.title('CorrMat of the outlook factors')
sns.heatmap(corr2, cmap = 'Blues')
```

Out[36]: <AxesSubplot:title={'center':'CorrMat of the outlook factors'}>



Clearly, the grade contributes to the price most, and the grade is highly correlated with the size of the living area of neighbors.

Therefore we prove **Assumption 4: the house price is possitively correlated with its outlook**

Assumption 5: Building Aging is negatively correlated with the price.

We can draw a scatter plot to investigate this assumption.

In [37]:
```
House['yr_renovated'].describe()
```

Out[37]:
```
count    21613.000000
mean        84.402258
std        401.679240
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       2015.000000
Name: yr_renovated, dtype: float64
```

In the original dataset, the houses never renovated is marked as 'yr_renovated' = 0, I can set the year renovated to be the year built.

Here I use the time the house was last renovated (if not renovated then the time built) to indicate the age of the house.

In [38]:
```
House.loc[House['yr_renovated'] == 0, 'yr_renovated'] = House.loc[House['yr_renovated'] == 0, 'yr_
```

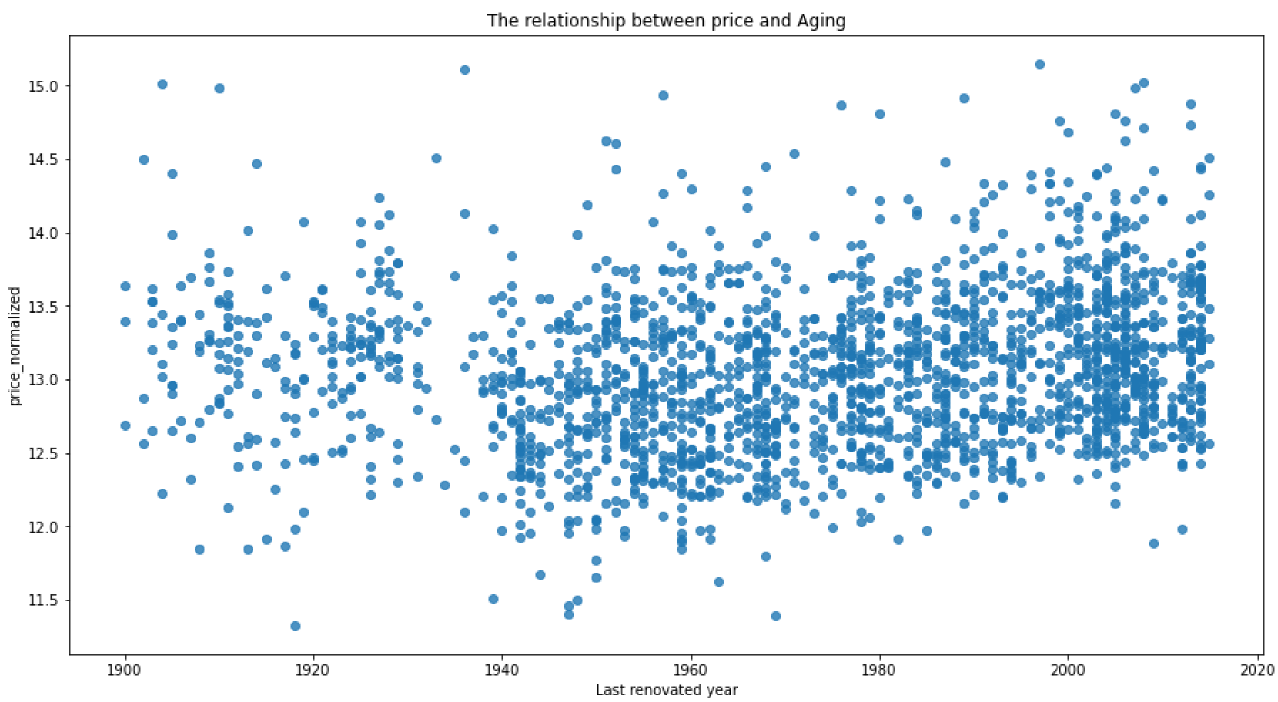
In [39]:
```
House.shape
```

Out[39]:
```
(21613, 25)
```

In [40]:
```
cmap = get_color_gradient('#B8D5FF', '#05133D', 21613)
House['mask'] = (np.random.uniform(size = House.shape[0]) > .9)
House_sample = House.loc[House['mask']]
# Because if I draw all the data points on the graph it would be too dense, I took random samples
plt.set_cmap('Blues')
plt.figure(figsize = (15, 8))
plt.scatter(data = House_sample, x = 'yr_renovated', y = 'price_normalized', alpha = .8)
plt.xlabel('Last renovated year')
plt.ylabel('price_normalized')
plt.title('The relationship between price and Aging')
```

Out[40]:
```
Text(0.5, 1.0, 'The relationship between price and Aging')
```

```
<Figure size 432x288 with 0 Axes>
```



There is not a clear pattern between aging and price, which might be a result for some people love vintage buildings. Therefore, we **disproved Assumption 5**

Wrap up:

The house price is affected by its location, utility and the outlook.

Data Exploration on the Diamond Dataset

Data Description

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 J (worst) to D (best)

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)

[Here (https://www.kaggle.com/datasets/shivam2503/diamonds)](https://www.kaggle.com/datasets/shivam2503/diamonds) is the source of the description.

In [41]:
```
Diamonds = pd.read_csv('diamonds.csv')
```

In [42]:
```
Diamonds.dtypes
```

Out[42]:
```
Unnamed: 0      int64
carat         float64
cut            object
color          object
clarity        object
depth         float64
table         float64
price           int64
x             float64
y             float64
z             float64
dtype: object
```

In [43]:
```
Diamonds.head()
```

Out[43]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Because of some problems happened when saving the dataset, the default index was saved, then we drop this column.

In [44]:
```
Diamonds.drop('Unnamed: 0', inplace = True, axis = 1)
```

In [45]:
```
Diamonds.isna().sum()
```

Out[45]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

Therefore there is no missing value in the data set.

In [46]:
```
Diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

There are several text data, and we can use groupby function to deal with it.

In [47]:
```
Diamonds.describe()
```

Out[47]:

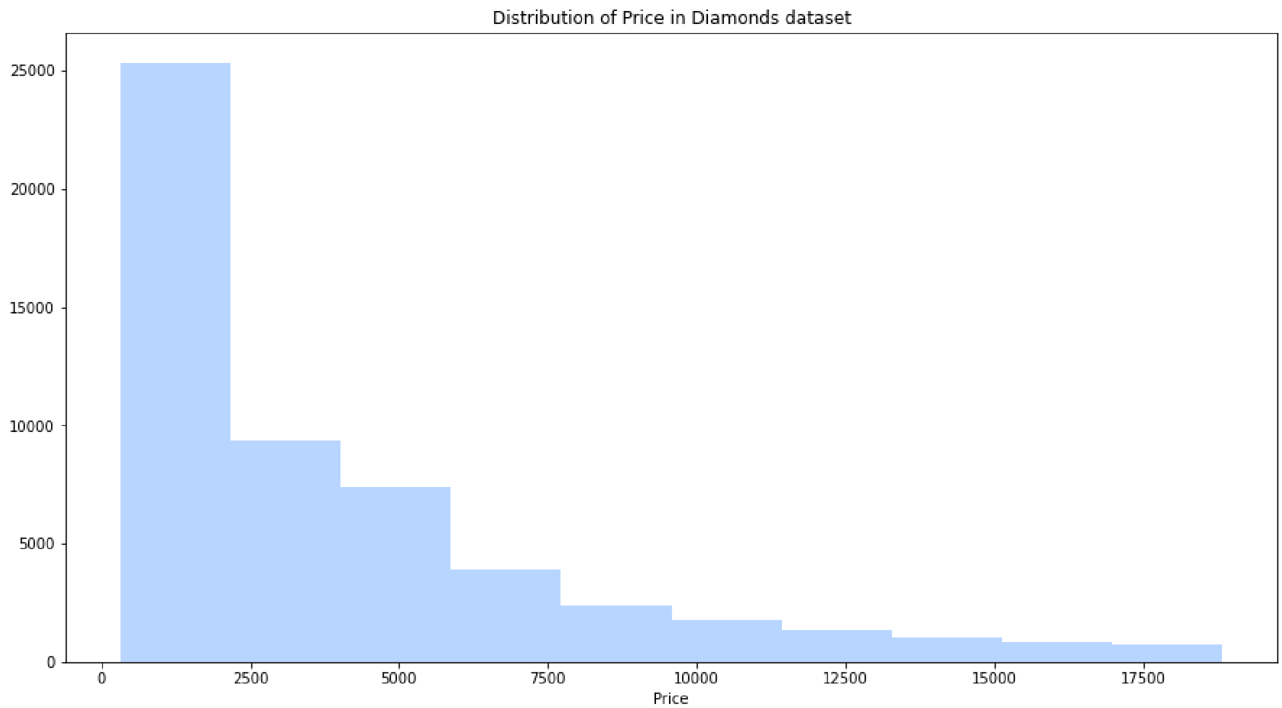
	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

In [48]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sb
import sklearn.linear_model as sl
import matplotlib.pyplot as plt
%matplotlib inline
```

Assumption 1: The more the carat weight, the higher the price.

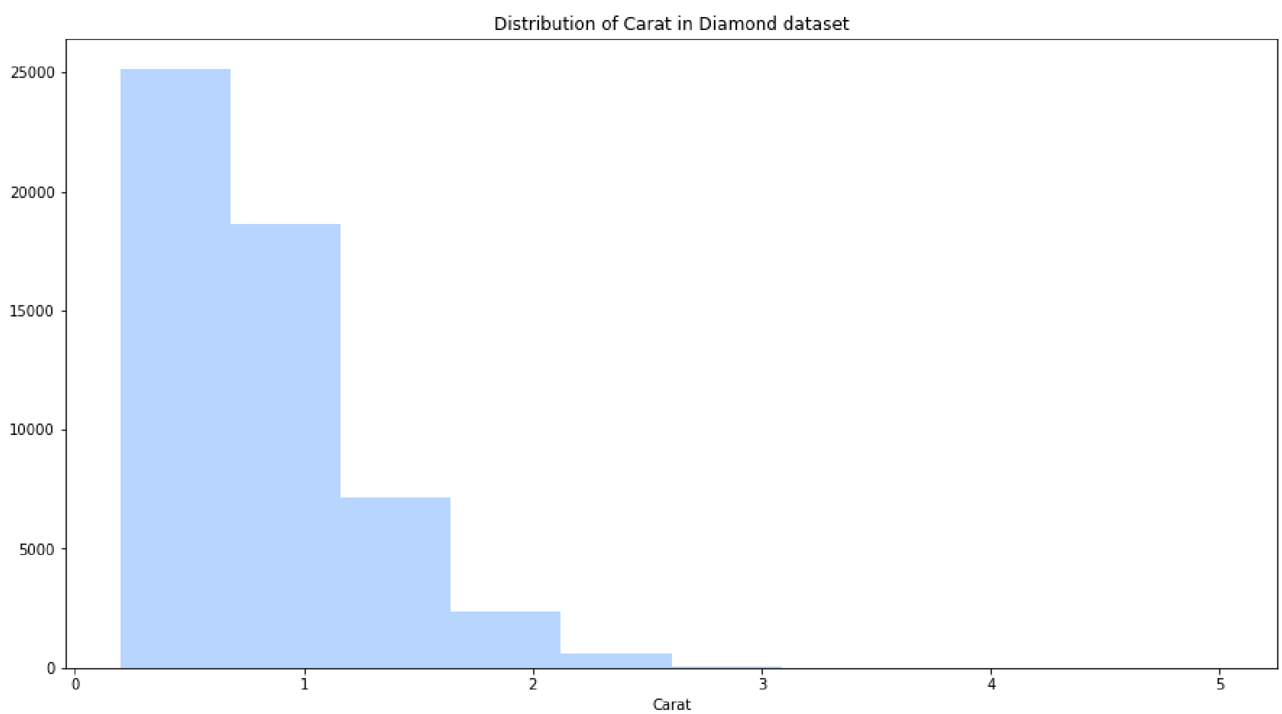
In [49]:
```
plt.figure(figsize=(15, 8))
plt.hist(Diamonds["price"], color = '#b8d5ff')
plt.xlabel("Price")
plt.title("Distribution of Price in Diamonds dataset")
```

Out[49]: Text(0.5, 1.0, 'Distribution of Price in Diamonds dataset')



In [50]:
```
plt.figure(figsize=(15, 8))
plt.hist(Diamonds["carat"], color = '#b8d5ff')
plt.xlabel("Carat")
plt.title("Distribution of Carat in Diamond dataset")
```

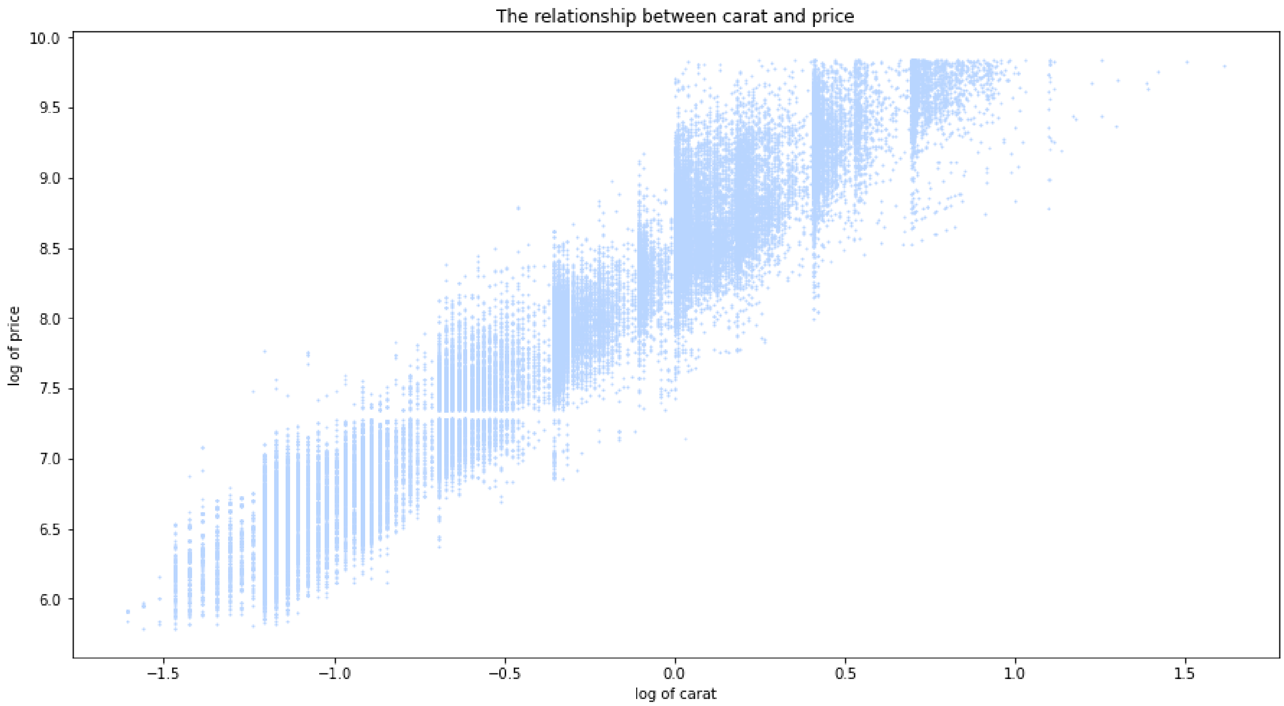
Out[50]: Text(0.5, 1.0, 'Distribution of Carat in Diamond dataset')



Because both price and carat are exponentially distributed, we can work on the log form of both of them.

```
In [51]: plt.figure(figsize = (15, 8))
         plt.scatter(x=np.log(Diamonds.carat) , y=np.log(Diamonds.price), color = '#b8d5ff', s = 1)
         plt.xlabel('log of carat')
         plt.ylabel('log of price')
         plt.title('The relationship between carat and price')
```

```
Out[51]: Text(0.5, 1.0, 'The relationship between carat and price')
```



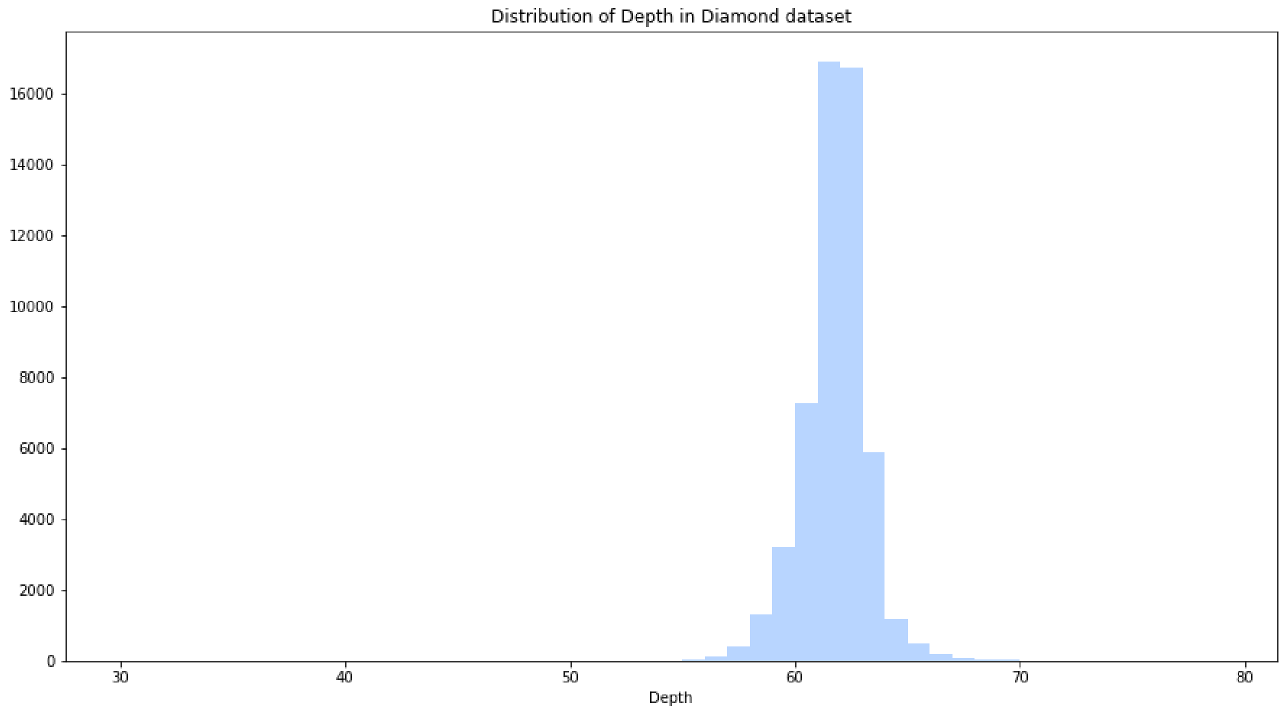
There is a strong linear relationship in the graph, so we proved **Assumption 1: The more the carat weight, the higher the price.**

Assumption 2: The depth is positively correlated with the price.

First we check the distribution of the depth.

In [52]:
```
plt.figure(figsize=(15, 8))
plt.hist(Diamonds["depth"], color = '#b8d5ff', bins = np.arange(30, 80, 1))
plt.xlabel("Depth")
plt.title("Distribution of Depth in Diamond dataset")
```

Out[52]:
```
Text(0.5, 1.0, 'Distribution of Depth in Diamond dataset')
```

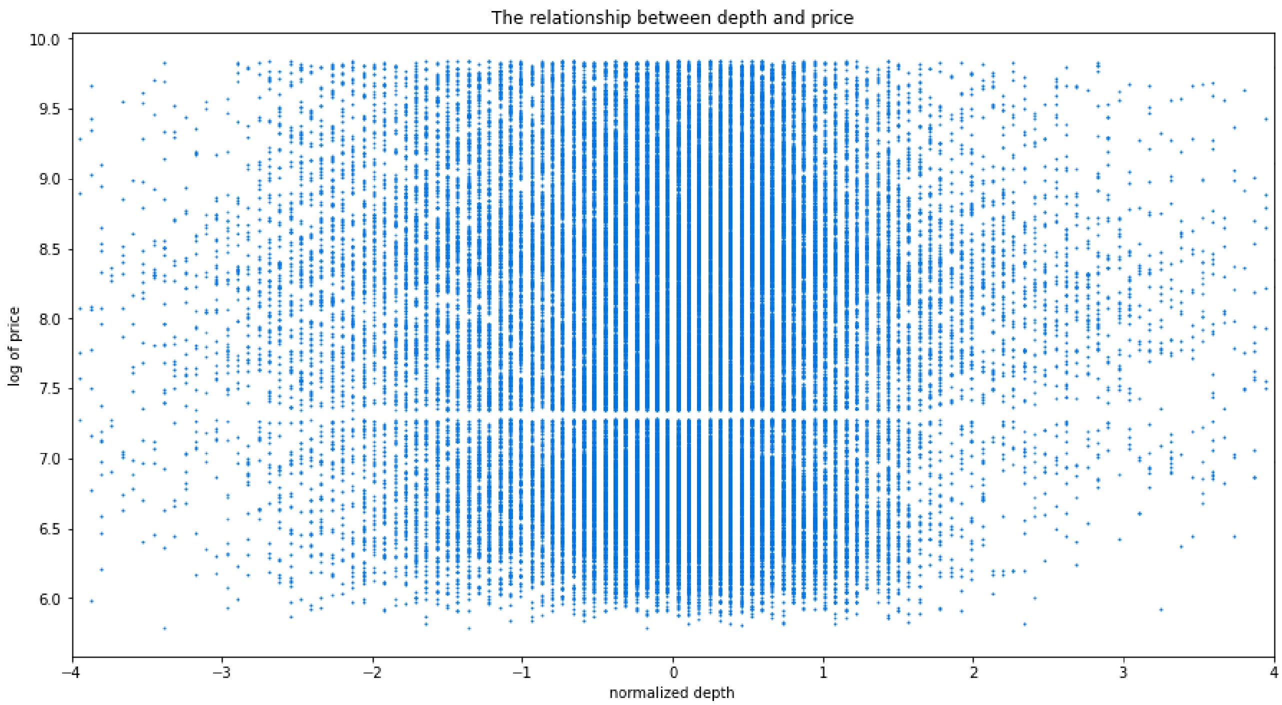


The Depth is symmetrically distributed, but highly centered. I would run a normalization on it.

In [53]:
```
def normalize(x):
    x = x.values
    return (x - np.mean(x))/np.std(x)
```

In [54]:
```
plt.figure(figsize = (15, 8))
plt.xlim(-4, 4)
plt.scatter(x=normalize(Diamonds['depth']) , y=np.log(Diamonds.price), color = '#0073e3', s = 1)
plt.xlabel('normalized depth')
plt.ylabel('log of price')
plt.title('The relationship between depth and price')
```

Out[54]:
```
Text(0.5, 1.0, 'The relationship between depth and price')
```



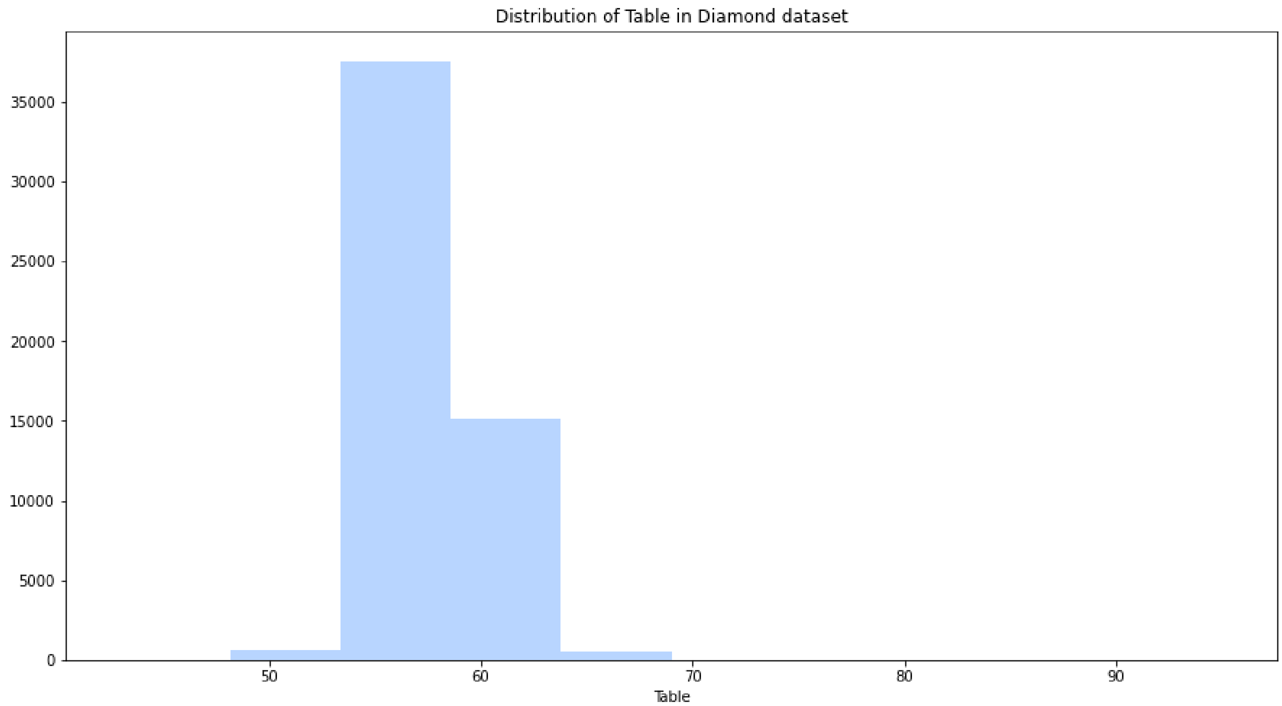
Therefore, the depth would not determine the price, we disprove **The depth is positively correlated with the price**

Assumption 3: table would affect the price of the diamond.

First we check the distribution of the table.

In [55]:
```
plt.figure(figsize=(15, 8))
plt.hist(Diamonds["table"], color = '#b8d5ff')
plt.xlabel("Table")
plt.title("Distribution of Table in Diamond dataset")
```

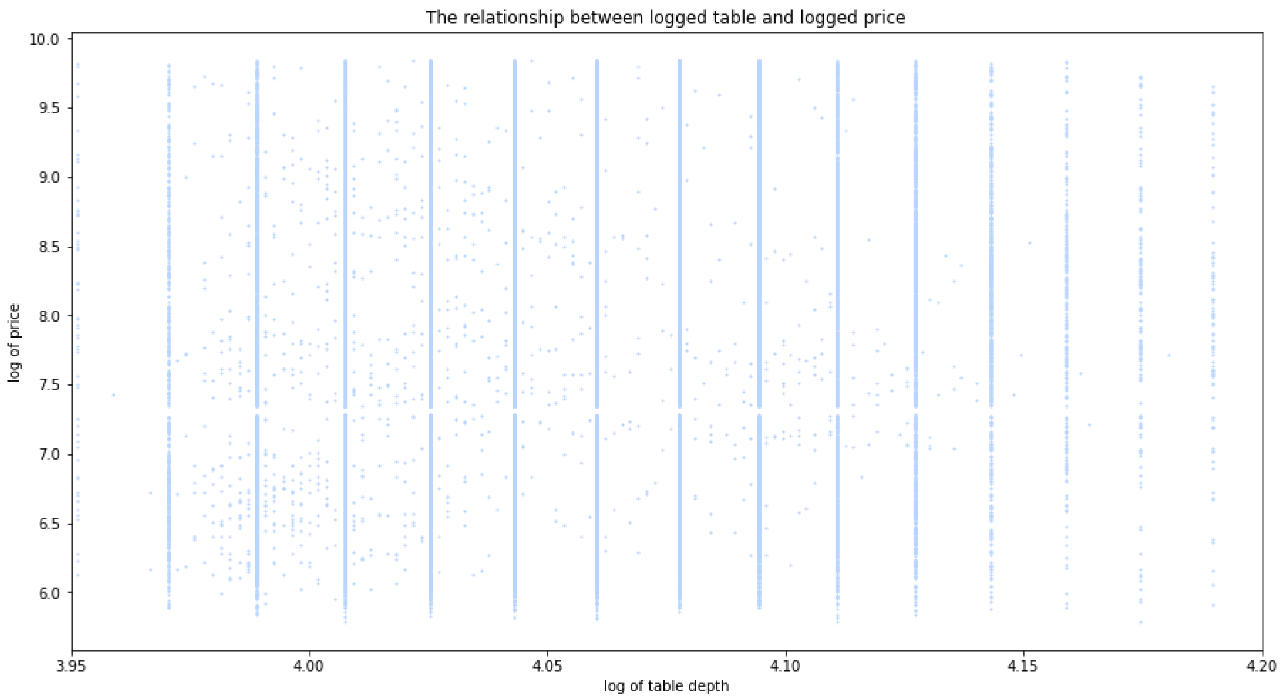
Out[55]: Text(0.5, 1.0, 'Distribution of Table in Diamond dataset')



Looks like it is exponentially distributed. So we draw scatter plots of the logs.

In [56]:
```
plt.figure(figsize = (15, 8))
plt.scatter(x=np.log(Diamonds['table']) , y=np.log(Diamonds.price), color = '#b8d5ff', s = 1)
plt.xlim(3.95, 4.2)
plt.xlabel('log of table depth')
plt.ylabel('log of price')
plt.title('The relationship between logged table and logged price')
```

Out[56]:
```
Text(0.5, 1.0, 'The relationship between logged table and logged price')
```

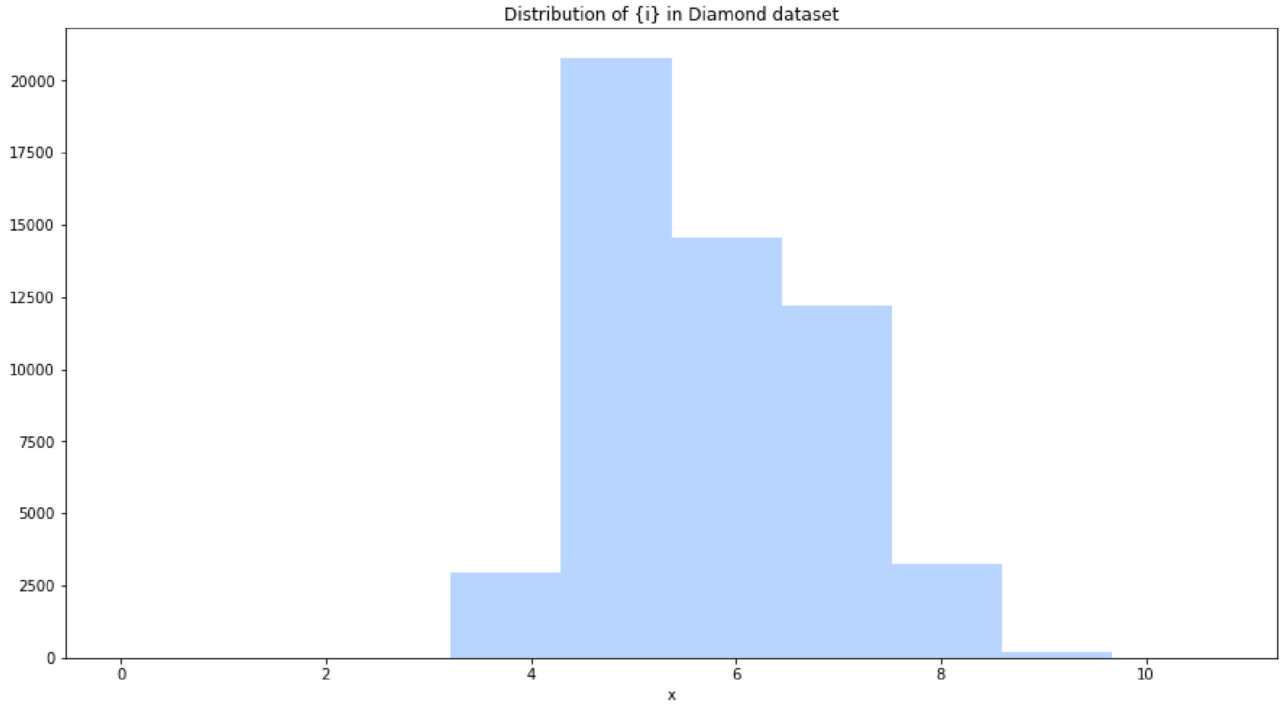


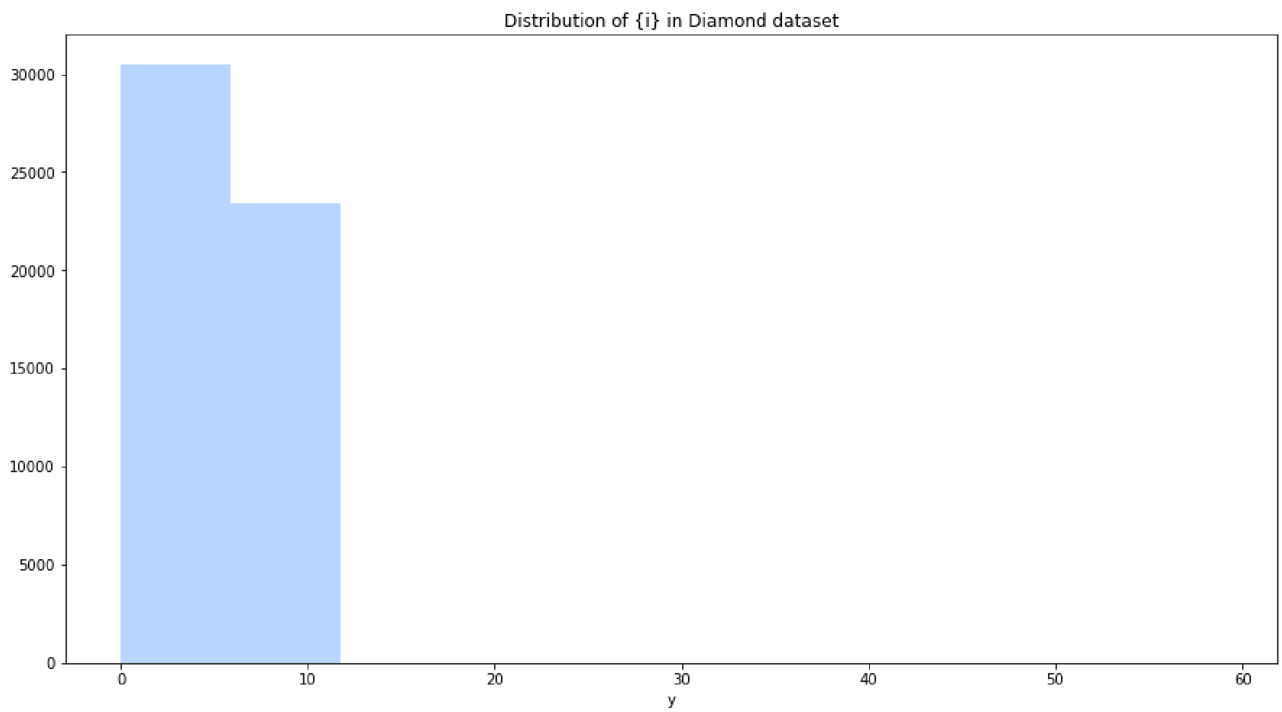
From the plot we disproved **Assumption 3: the table is correlated with the price**

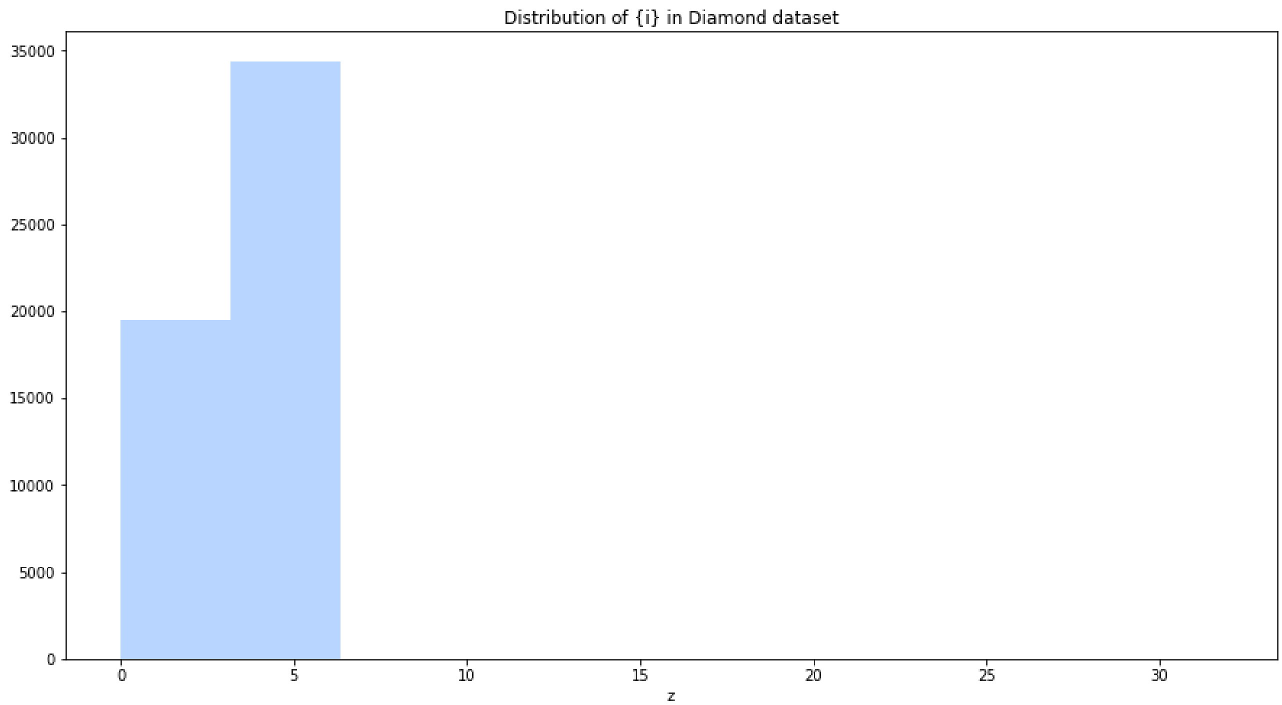
Assumption 4: x, y or z would affect the price independently.

First, we check the distribution of x, y and z.

In [57]:
```
for i in ['x', 'y', 'z']:
    plt.figure(figsize=(15, 8))
    plt.hist(Diamonds[i], color = '#b8d5ff')
    plt.xlabel(i)
    plt.title("Distribution of {i} in Diamond dataset")
    plt.show()
```







From the plots above, we can see that the x is symmetriccally but not normally distibuted, while y and z are exponentially distributed.

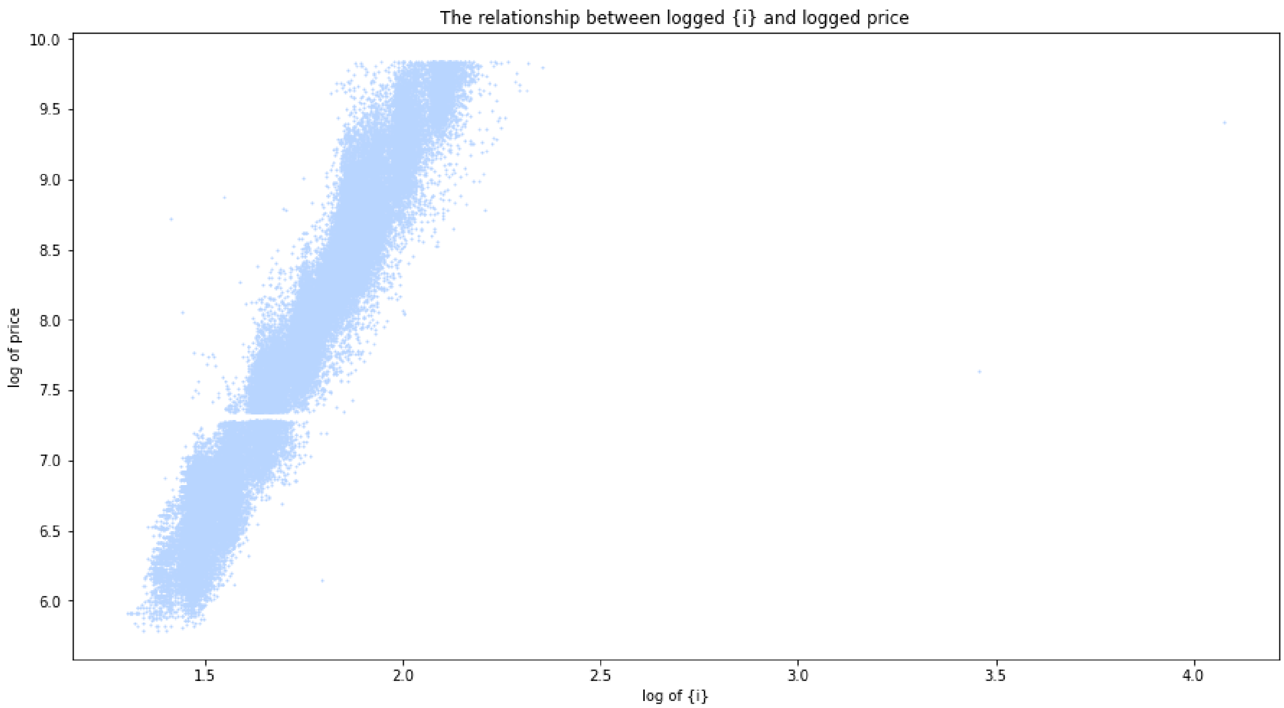
In [58]:
```
plt.figure(figsize = (15, 8))
plt.scatter(x=normalize(Diamonds['x']) , y=np.log(Diamonds.price), color = '#b8d5ff', s = 1)
plt.xlabel('normalized x')
plt.ylabel('log of price')
plt.title('The relationship between x and price')
```

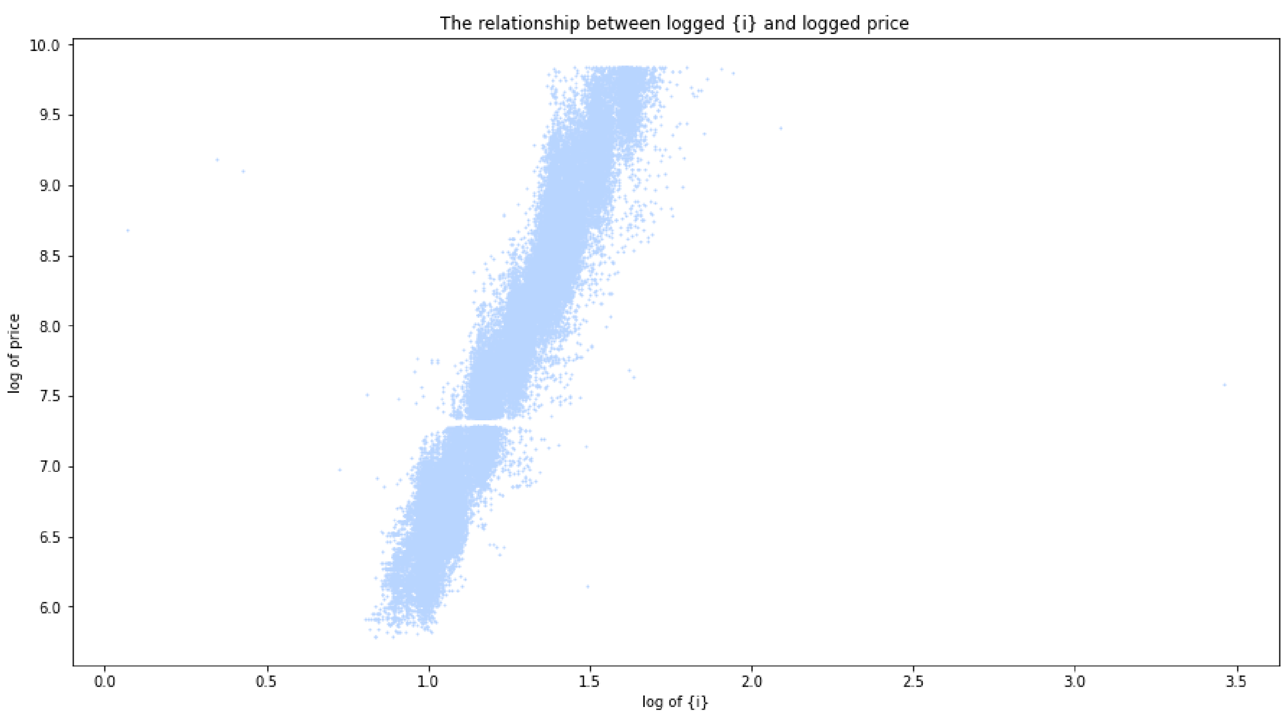
Out[58]:
```
Text(0.5, 1.0, 'The relationship between x and price')
```

It looks like normalized x is positively correlated with the logarithm of the price.

In [59]:
```
for i in ['y', 'z']:
    plt.figure(figsize = (15, 8))
    plt.scatter(x=np.log(Diamonds[i]) , y=np.log(Diamonds.price), color = '#b8d5ff', s = 1)
    plt.xlabel('log of {i}')
    plt.ylabel('log of price')
    plt.title('The relationship between logged {i} and logged price')
    plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/pandas/core/arraylike.py:397: RuntimeWarning: divide
by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```





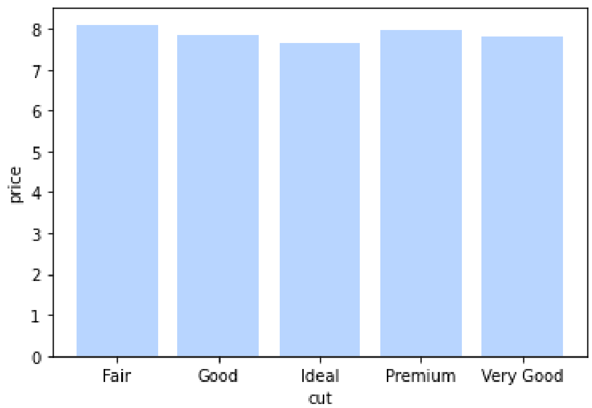
The logarithm of price is positively correlated with the logarithm of x and y.

Assumption 5: the price would be affected by the price.

In [60]:
```
Diamonds['log_price'] = np.log(Diamonds['price'])
data_for_plot = Diamonds.groupby('cut').log_price.mean()
plt.figure(figsize = (15, 8))
fig, ax = plt.subplots()
ax.bar(data_for_plot.index, data_for_plot, color='#b8d5ff')
ax.set_xticks([0,1,2,3,4])
ax.set_xlabel('cut')
ax.set_ylabel('price')
```

Out[60]:
```
Text(0, 0.5, 'price')
```

```
<Figure size 1080x576 with 0 Axes>
```



The price is distributed uniformly across the cut

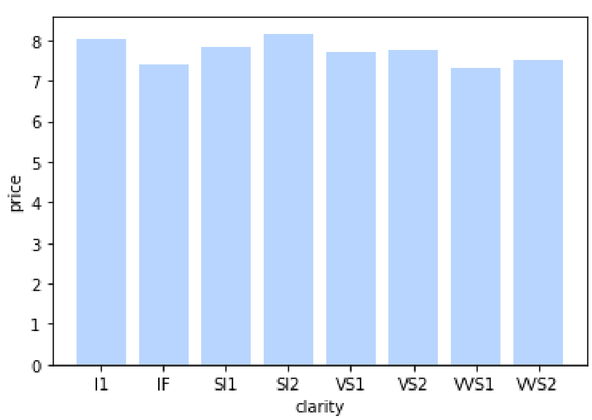
Assumption 5 is disproved, the cut of the diamond doesn't affect the price.

Assumption 6: The price would be affected by clarity.

In [61]:
```
Diamonds['log_price'] = np.log(Diamonds['price'])
data_for_plot = Diamonds.groupby('clarity').log_price.mean()
plt.figure(figsize = (15, 8))
fig, ax = plt.subplots()
ax.bar(data_for_plot.index, data_for_plot, color='#b8d5ff')
ax.set_xticks([0,1,2,3,4,5,6,7])
ax.set_xlabel('clarity')
ax.set_ylabel('price')
```

Out[61]:
```
Text(0, 0.5, 'price')
```

```
<Figure size 1080x576 with 0 Axes>
```

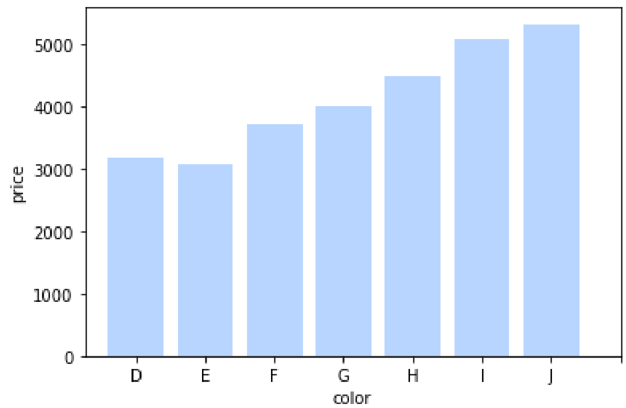


The price is distributed uniformly across the different clarities

Therefore we disproved **Assumption 6:The price would be affected by clarity.**

In [62]:
```
data_for_plot = Diamonds.groupby('color').price.mean()
fig, ax = plt.subplots()
ax.bar(data_for_plot.index, data_for_plot, color=['#B8D5FF'])
ax.set_xticks([0,1,2,3,4,5,6,7])
ax.set_xlabel('color')
ax.set_ylabel('price')
```

Out[62]: `Text(0, 0.5, 'price')`



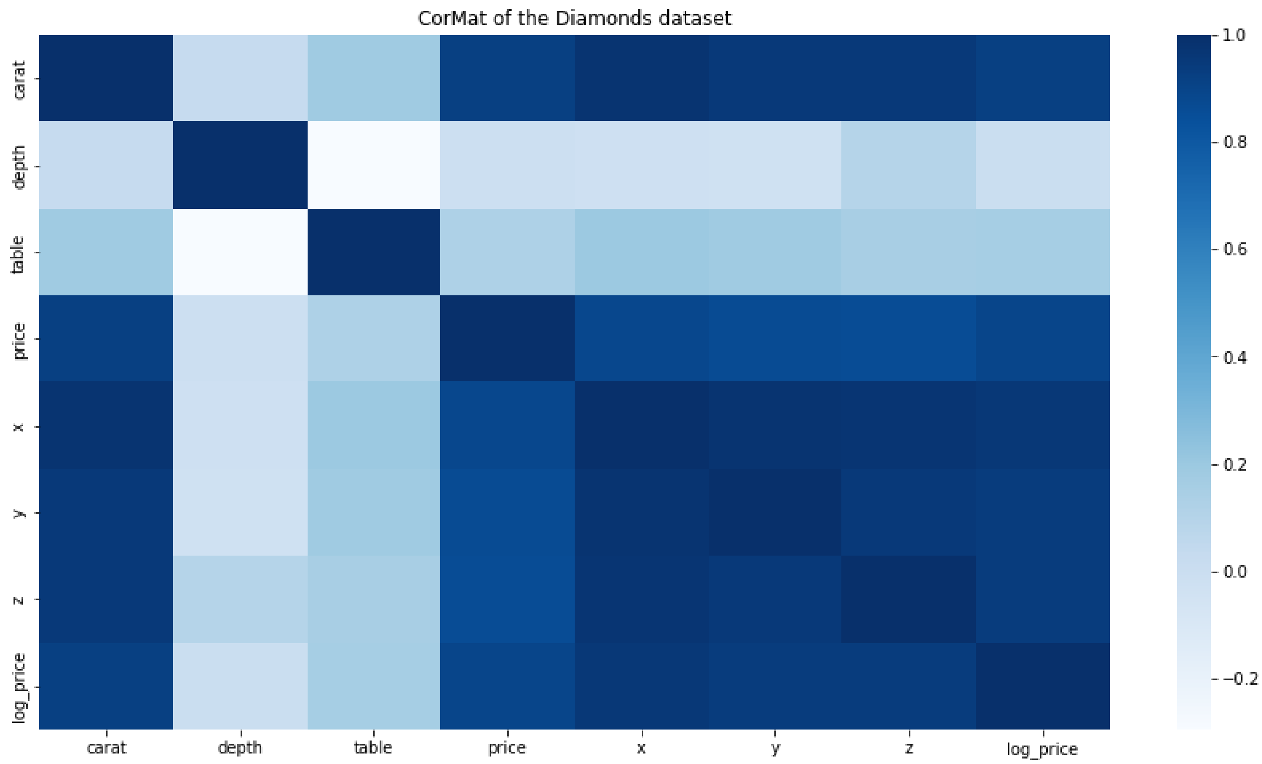
For different colors, the prices are significantly different. Price of color D is lowest and price of color J is highest.

So we proved **Assumption 7: Color determines the price of the diamonds**

Below are the correlations of different variables, and log_price is highly correlated with carat, x, y, z.

In [63]:
```
plt.figure(figsize = (15, 8))
plt.title('CorMat of the Diamonds dataset')
corr3 = Diamonds.corr()
sns.heatmap(corr3, cmap = 'Blues')
```

Out[63]: <AxesSubplot:title={'center':'CorMat of the Diamonds dataset'}>



Wrap up:

The price of the diamonds are determined by weight, color, x, y, and z.

Reference:

[1] Pawan Shivhare. (2017, September 12). Predicting King County House Prices. https://www.slideshare.net/PawanShivhare1/predicting-king-county-house-prices (https://www.slideshare.net/PawanShivhare1/predicting-king-county-house-prices)

[2] (2023). Arcgis.com. https://www.arcgis.com/apps/Embed/index.html?webmap=fc48611b42a5425a8d93c36004afa377&extent=-122.6079 (https://www.arcgis.com/apps/Embed/index.html?webmap=fc48611b42a5425a8d93c36004afa377&extent=-122.6079)

[3] Diamonds. (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/shivam2503/diamonds (https://www.kaggle.com/datasets/shivam2503/diamonds)