Problem Statement :-

Q5. Uber is a taxi service provider as we know, we need to predict the high booking area using an Unsupervised algorithm and price for the location using a supervised algorithm and use some map function to display the data

Dataset link:- https://www.kaggle.com/datasets/brllrb/uber-and-lyft-dataset-boston-ma

1. Importing Library and Dataset

In [1]:
```
# Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

In [3]:
```
df=pd.read_csv("rideshare_kaggle.csv")
df.head()
```

Out[3]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	product_id	name	price
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	1.544953e+09	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square	North Station	Lyft	lyft_line	Shared	5.0
1	4bd23055-6827-41c6-b23b-3c491f24e74d	1.543284e+09	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square	North Station	Lyft	lyft_premier	Lux	11.0
2	981a3613-77af-4620-a42a-0c0866077d1e	1.543367e+09	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square	North Station	Lyft	lyft	Lyft	7.0
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	1.543554e+09	4	30	11	2018-11-30 04:53:02	America/New_York	Haymarket Square	North Station	Lyft	lyft_luxsuv	Lux Black XL	26.0
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	1.543463e+09	3	29	11	2018-11-29 03:49:20	America/New_York	Haymarket Square	North Station	Lyft	lyft_plus	Lyft XL	9.0

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 57 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   id                           693071 non-null  object 
 1   timestamp                    693071 non-null  float64
 2   hour                         693071 non-null  int64  
 3   day                          693071 non-null  int64  
 4   month                        693071 non-null  int64  
 5   datetime                     693071 non-null  object 
 6   timezone                     693071 non-null  object 
 7   source                       693071 non-null  object 
 8   destination                  693071 non-null  object 
 9   cab_type                     693071 non-null  object 
 10  product_id                   693071 non-null  object 
 11  name                         693071 non-null  object 
 12  price                        637976 non-null  float64
 13  distance                     693071 non-null  float64
 14  surge_multiplier             693071 non-null  float64
 15  latitude                     693071 non-null  float64
 16  longitude                    693071 non-null  float64
 17  temperature                  693071 non-null  float64
 18  apparentTemperature          693071 non-null  float64
 19  short_summary                693071 non-null  object 
 20  long_summary                 693071 non-null  object 
 21  precipIntensity              693071 non-null  float64
 22  precipProbability            693071 non-null  float64
 23  humidity                     693071 non-null  float64
 24  windSpeed                    693071 non-null  float64
 25  windGust                     693071 non-null  float64
 26  windGustTime                 693071 non-null  int64  
 27  visibility                   693071 non-null  float64
 28  temperatureHigh              693071 non-null  float64
 29  temperatureHighTime          693071 non-null  int64  
 30  temperatureLow               693071 non-null  float64
 31  temperatureLowTime           693071 non-null  int64  
 32  apparentTemperatureHigh      693071 non-null  float64
 33  apparentTemperatureHighTime  693071 non-null  int64  
 34  apparentTemperatureLow       693071 non-null  float64
 35  apparentTemperatureLowTime   693071 non-null  int64  
 36  icon                         693071 non-null  object 
 37  dewPoint                     693071 non-null  float64
 38  pressure                     693071 non-null  float64
 39  windBearing                  693071 non-null  int64  
 40  cloudCover                   693071 non-null  float64
 41  uvIndex                      693071 non-null  int64  
 42  visibility.1                 693071 non-null  float64
 43  ozone                        693071 non-null  float64
 44  sunriseTime                  693071 non-null  int64  
 45  sunsetTime                   693071 non-null  int64  
 46  moonPhase                    693071 non-null  float64
 47  precipIntensityMax           693071 non-null  float64
 48  uvIndexTime                  693071 non-null  int64  
 49  temperatureMin               693071 non-null  float64
 50  temperatureMinTime           693071 non-null  int64  
 51  temperatureMax               693071 non-null  float64
 52  temperatureMaxTime           693071 non-null  int64  
 53  apparentTemperatureMin       693071 non-null  float64
 54  apparentTemperatureMinTime   693071 non-null  int64  
 55  apparentTemperatureMax       693071 non-null  float64
 56  apparentTemperatureMaxTime   693071 non-null  int64  
dtypes: float64(29), int64(17), object(11)
memory usage: 301.4+ MB
```

In [5]:
```
df['datetime']=pd.to_datetime(df['datetime'])
```

In [10]:
```
df.columns
```

Out[10]:
```
Index(['id', 'timestamp', 'hour', 'day', 'month', 'datetime', 'timezone',
       'source', 'destination', 'cab_type', 'product_id', 'name', 'price',
       'distance', 'surge_multiplier', 'latitude', 'longitude', 'temperature',
       'apparentTemperature', 'short_summary', 'long_summary',
       'precipIntensity', 'precipProbability', 'humidity', 'windSpeed',
       'windGust', 'windGustTime', 'visibility', 'temperatureHigh',
       'temperatureHighTime', 'temperatureLow', 'temperatureLowTime',
       'apparentTemperatureHigh', 'apparentTemperatureHighTime',
       'apparentTemperatureLow', 'apparentTemperatureLowTime', 'icon',
       'dewPoint', 'pressure', 'windBearing', 'cloudCover', 'uvIndex',
       'visibility.1', 'ozone', 'sunriseTime', 'sunsetTime', 'moonPhase',
       'precipIntensityMax', 'uvIndexTime', 'temperatureMin',
       'temperatureMinTime', 'temperatureMax', 'temperatureMaxTime',
       'apparentTemperatureMin', 'apparentTemperatureMinTime',
       'apparentTemperatureMax', 'apparentTemperatureMaxTime'],
      dtype='object')
```

Data Cleaning

In [6]:
```
df.isnull().sum().sum()
```

Out[6]:
```
55095
```

In [7]:
```
df.dropna(axis=0,inplace=True)
```

In [8]:
```
df.isnull().sum().sum()
```

Out[8]:
```
0
```

In [9]:
```
df['visibility'].head()
```

Out[9]:
```
0    10.000
1     4.786
2    10.000
3    10.000
4    10.000
Name: visibility, dtype: float64
```

In [11]:
```
df['visibility.1'].head()
```

Out[11]:
```
0    10.000
1     4.786
2    10.000
3    10.000
4    10.000
Name: visibility.1, dtype: float64
```

In [12]:
```
df = df.drop(['visibility.1'],axis=1)
```

2. EDA and Visualization

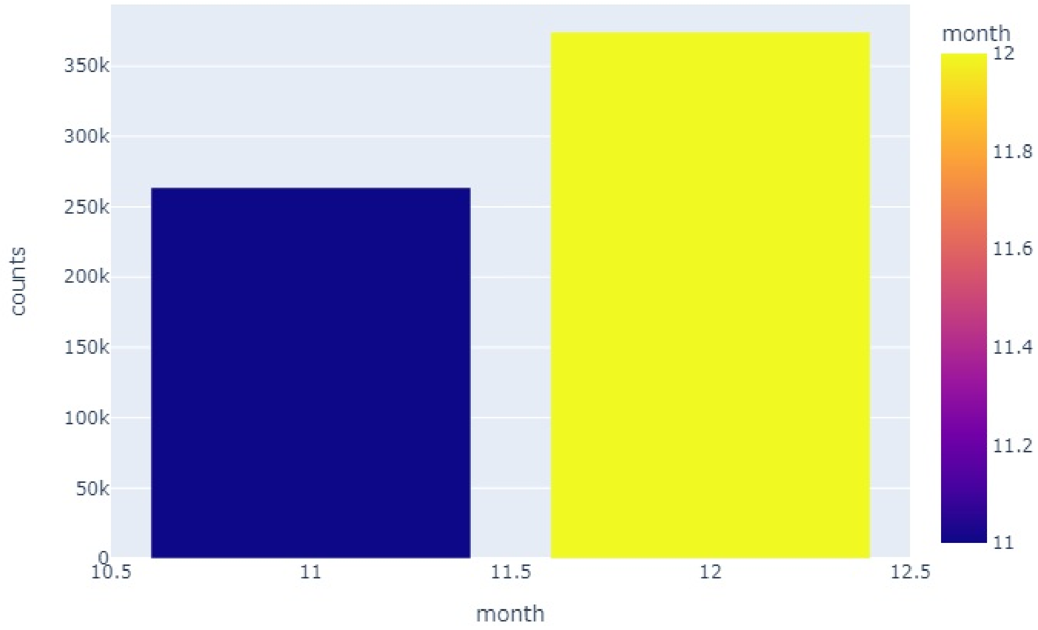
1. Time Analysis

--Month Data--

In [13]:
```
def plot_bar(groupby_column):
    df1 =df.groupby(groupby_column).size().reset_index(name="counts")
    fig1 = px.bar(data_frame=df1, x=groupby_column, y="counts", color=groupby_column, barmode="group")
    print(df1)
    fig1.show(renderer='png')
```

In [14]:
```
plot_bar('month')
```

```
   month  counts
0     11  263771
1     12  374205
```



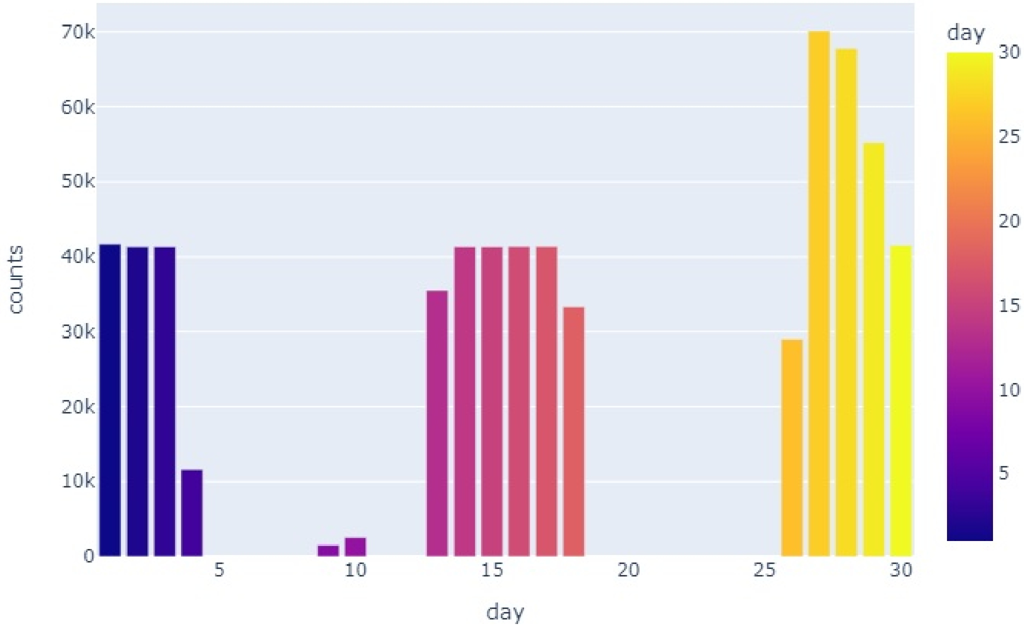
Observation

- It appears that we only have november and december in our month data. It means the data is only recorded or taken in november and december with december data dominating.

--Day Data--

In [15]:
```
plot_bar('day')
```

```
    day  counts
0     1   41680
1     2   41298
2     3   41323
3     4   11627
4     9    1529
5    10    2534
6    13   35496
7    14   41344
8    15   41332
9    16   41359
10   17   41354
11   18   33329
12   26   29028
13   27   70135
14   28   67842
15   29   55222
16   30   41544
```



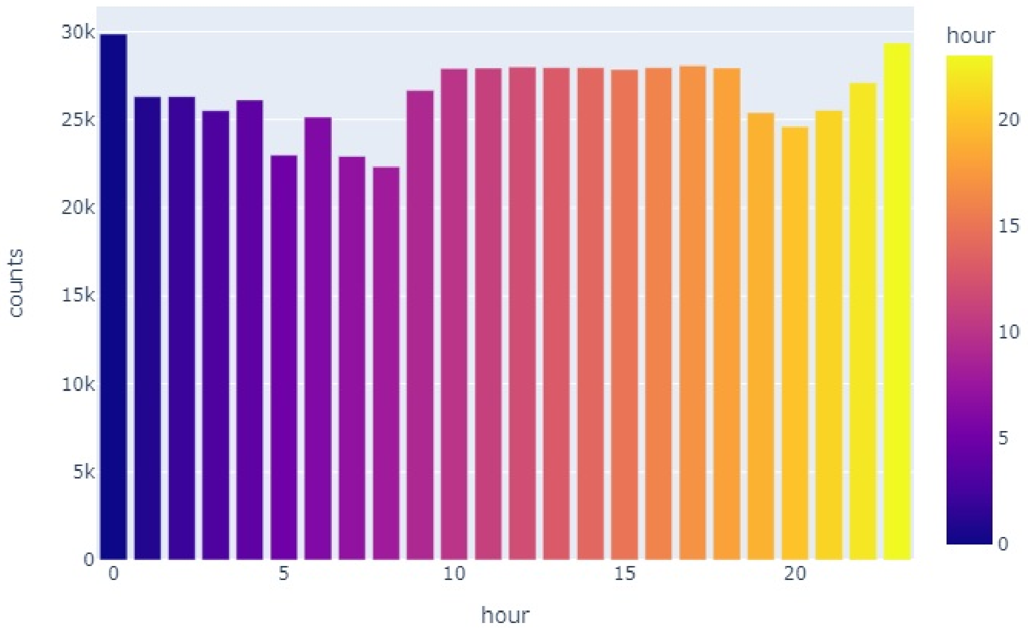
Observation

- It seems we have many gaps in our 'day' data. For example we don't have data from 18th day until 25th day in each month.

--Hour Data--

In [16]:
```
plot_bar('hour')
```

```
    hour  counts
0      0   29872
1      1   26310
2      2   26323
3      3   25530
4      4   26125
5      5   22995
6      6   25147
7      7   22930
8      8   22337
9      9   26673
10    10   27918
11    11   27946
12    12   28017
13    13   27977
14    14   27976
15    15   27868
16    16   27972
17    17   28075
18    18   27958
19    19   25410
20    20   24620
21    21   25549
22    22   27093
23    23   29355
```



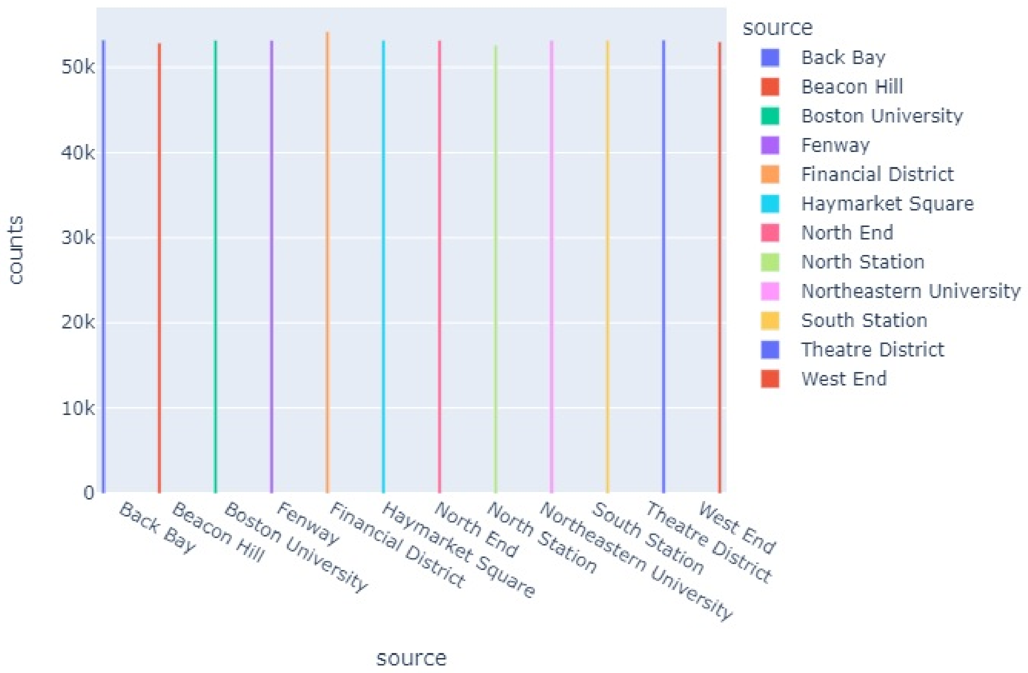
Observation

- It seems we have almost 24 hours recorded data

2. Source and Destination Analysis

In [17]:
```
plot_bar('source')
```

```
                     source  counts
0                  Back Bay   53201
1               Beacon Hill   52841
2         Boston University   53172
3                    Fenway   53166
4        Financial District   54197
5          Haymarket Square   53147
6                 North End   53171
7             North Station   52576
8   Northeastern University   53164
9             South Station   53160
10         Theatre District   53201
11                 West End   52980
```

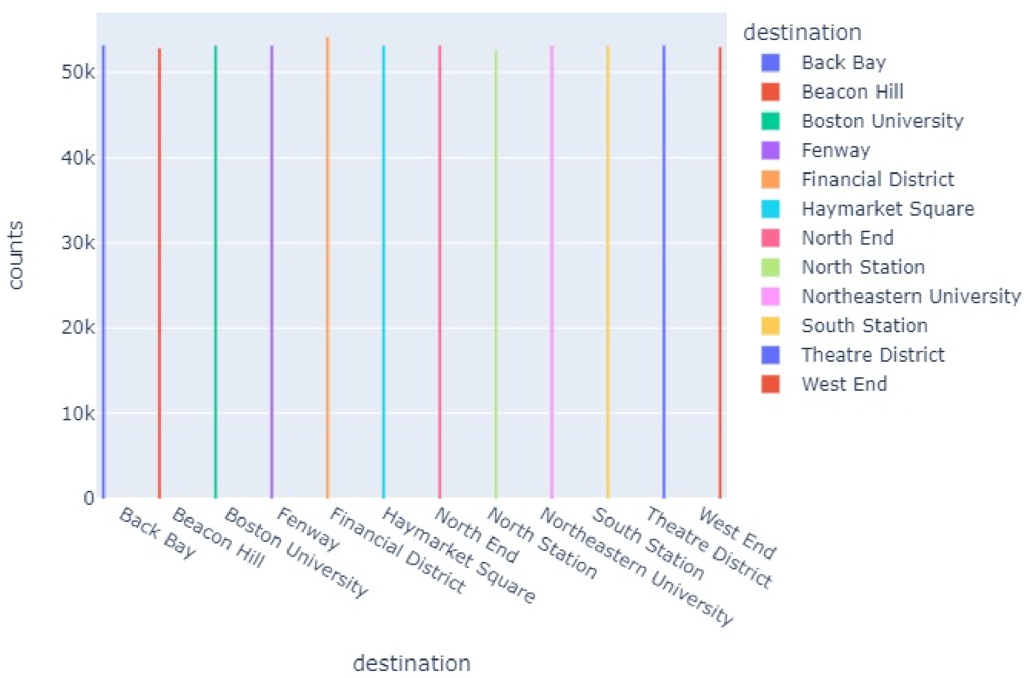


Observation

- It seems that all sources are almost equal in number. There are about 50k data in each source feature (Back Bay, Beacon Hill, Boston University, etc)

In [18]:
```
plot_bar('destination')
```

```
                destination  counts
0                  Back Bay   53190
1               Beacon Hill   52840
2         Boston University   53171
3                    Fenway   53166
4        Financial District   54192
5          Haymarket Square   53171
6                 North End   53164
7             North Station   52577
8   Northeastern University   53165
9             South Station   53159
10         Theatre District   53189
11                 West End   52992
```



Observation

- Same with source feature, there are about 50k data in each destination feature (Back Bay, Beacon Hill, Boston University, etc)

In [19]:
```python
df.groupby(by=["destination","source"]).agg({'latitude':'mean','longitude':'mean'})
```

Out[19]:

destination	source	latitude	longitude
Back Bay	Boston University	42.336960	-71.066178
	Fenway	42.337740	-71.065822
	Haymarket Square	42.337087	-71.065110
	North End	42.338100	-71.066343
	Northeastern University	42.336668	-71.065314
	South Station	42.338897	-71.065908
Beacon Hill	Boston University	42.336917	-71.065885
	Fenway	42.338990	-71.065719
	Haymarket Square	42.337413	-71.066059
	North End	42.338418	-71.065809
	Northeastern University	42.337268	-71.066061
	South Station	42.336316	-71.065699
Boston University	Back Bay	42.337217	-71.065947
	Beacon Hill	42.339364	-71.066517
	Financial District	42.339361	-71.066465
	North Station	42.338372	-71.066191
	Theatre District	42.338152	-71.066276
	West End	42.337556	-71.066265
Fenway	Back Bay	42.340103	-71.065819
	Beacon Hill	42.337595	-71.065471
	Financial District	42.337147	-71.066254
	North Station	42.339660	-71.066504
	Theatre District	42.336378	-71.065388

	West End	42.338521	-71.066339
Financial District	**Boston University**	42.338733	-71.066581
	Fenway	42.337034	-71.066028
	Haymarket Square	42.337781	-71.065863
	North End	42.338338	-71.065965
	Northeastern University	42.338523	-71.065964
	South Station	42.338989	-71.067037
Haymarket Square	**Back Bay**	42.339877	-71.066475
	Beacon Hill	42.337246	-71.065966
	Financial District	42.337398	-71.066237
	North Station	42.338276	-71.066073
	Theatre District	42.338175	-71.065699
	West End	42.339109	-71.066251
North End	**Back Bay**	42.338516	-71.066170
	Beacon Hill	42.336792	-71.066216
	Financial District	42.337654	-71.066158
	North Station	42.339309	-71.066936
	Theatre District	42.338578	-71.066639
	West End	42.338614	-71.065878
North Station	**Boston University**	42.338786	-71.066362
	Fenway	42.338450	-71.066614
	Haymarket Square	42.337260	-71.066279
	North End	42.337672	-71.065832
	Northeastern University	42.337793	-71.066491
	South Station	42.336529	-71.065432
Northeastern University	**Back Bay**	42.338917	-71.066289
	Beacon Hill	42.339002	-71.065600
	Financial District	42.337789	-71.066015
	North Station	42.339770	-71.066493
	Theatre District	42.338356	-71.065319
	West End	42.336812	-71.066274
South Station	**Back Bay**	42.338567	-71.065891
	Beacon Hill	42.338714	-71.066985
	Financial District	42.337748	-71.065971
	North Station	42.338766	-71.066221
	Theatre District	42.338344	-71.066466
	West End	42.338753	-71.066376
Theatre District	**Boston University**	42.338496	-71.066315
	Fenway	42.338203	-71.066281
	Haymarket Square	42.338824	-71.065937
	North End	42.336628	-71.066122
	Northeastern University	42.338128	-71.066390
	South Station	42.338515	-71.066384
West End	**Boston University**	42.337798	-71.065910
	Fenway	42.338291	-71.066356
	Haymarket Square	42.339314	-71.066239
	North End	42.337212	-71.066268
	Northeastern University	42.340015	-71.066646
	South Station	42.338983	-71.066967

Observation

- Here i make a geospatial map to visualize our data which the departure point of the trips is haymarket square. I plot them using clusters instead of marker. The map rendered by folium is interactive, we can slide, drag, and zoom in/out

In [21]:
```
!pip install geopandas
```

```
Collecting geopandas
  Downloading geopandas-0.13.0-py3-none-any.whl (1.1 MB)
     ---------------------------------------- 1.1/1.1 MB 3.0 MB/s eta 0:00:00
Collecting fiona>=1.8.19 (from geopandas)
  Downloading Fiona-1.9.4-cp39-cp39-win_amd64.whl (22.8 MB)
     ---------------------------------------- 22.8/22.8 MB 5.5 MB/s eta 0:00:00
Requirement already satisfied: packaging in c:\users\lenovo\anaconda3\lib\site-packages (from geopandas) (22.0)
Requirement already satisfied: pandas>=1.1.0 in c:\users\lenovo\anaconda3\lib\site-packages (from geopandas) (1
.3.4)
Collecting pyproj>=3.0.1 (from geopandas)
  Downloading pyproj-3.5.0-cp39-cp39-win_amd64.whl (5.1 MB)
     ---------------------------------------- 5.1/5.1 MB 5.5 MB/s eta 0:00:00
Collecting shapely>=1.7.1 (from geopandas)
  Downloading shapely-2.0.1-cp39-cp39-win_amd64.whl (1.4 MB)
     ---------------------------------------- 1.4/1.4 MB 5.8 MB/s eta 0:00:00
Requirement already satisfied: attrs>=19.2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from fiona>=1.8.19
->geopandas) (21.2.0)
Requirement already satisfied: certifi in c:\users\lenovo\anaconda3\lib\site-packages (from fiona>=1.8.19->geop
andas) (2021.10.8)
Requirement already satisfied: click~=8.0 in c:\users\lenovo\anaconda3\lib\site-packages (from fiona>=1.8.19->g
eopandas) (8.0.3)
Collecting click-plugins>=1.0 (from fiona>=1.8.19->geopandas)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Collecting cligj>=0.5 (from fiona>=1.8.19->geopandas)
  Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Requirement already satisfied: six in c:\users\lenovo\anaconda3\lib\site-packages (from fiona>=1.8.19->geopanda
s) (1.16.0)
Requirement already satisfied: importlib-metadata in c:\users\lenovo\anaconda3\lib\site-packages (from fiona>=1
.8.19->geopandas) (4.8.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\lenovo\anaconda3\lib\site-packages (from pand
as>=1.1.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\lenovo\anaconda3\lib\site-packages (from pandas>=1.1.0-
>geopandas) (2021.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\anaconda3\lib\site-packages (from pandas>=1.1.0
->geopandas) (1.22.4)
Requirement already satisfied: colorama in c:\users\lenovo\anaconda3\lib\site-packages (from click~=8.0->fiona>
=1.8.19->geopandas) (0.4.4)
Requirement already satisfied: zipp>=0.5 in c:\users\lenovo\anaconda3\lib\site-packages (from importlib-metadat
a->fiona>=1.8.19->geopandas) (3.6.0)
Installing collected packages: shapely, pyproj, cligj, click-plugins, fiona, geopandas
Successfully installed click-plugins-1.1.1 cligj-0.7.2 fiona-1.9.4 geopandas-0.13.0 pyproj-3.5.0 shapely-2.0.1
```

In [22]:

```
import geopandas as gpd
import folium
from folium.plugins import FastMarkerCluster
df1 = df[df['source']=='Haymarket Square']
my_map = folium.Map(location=[df1["latitude"].mean(), df1["longitude"].mean()],zoom_start = 10)
my_map.add_child(FastMarkerCluster(df1[['latitude', 'longitude']].values.tolist(),color='green'))
my_map
```

Out[22]: Make this Notebook Trusted to load map: File -> Trust Notebook

Observation

- We can see that trips which their sources are Haymarket Square have two groups or clusters of destination that contain many places (we can see them if we zoom the map). Many of them are in boston area as we can see that there are 46256 data in that cluster.

3. Cab Type Analysis

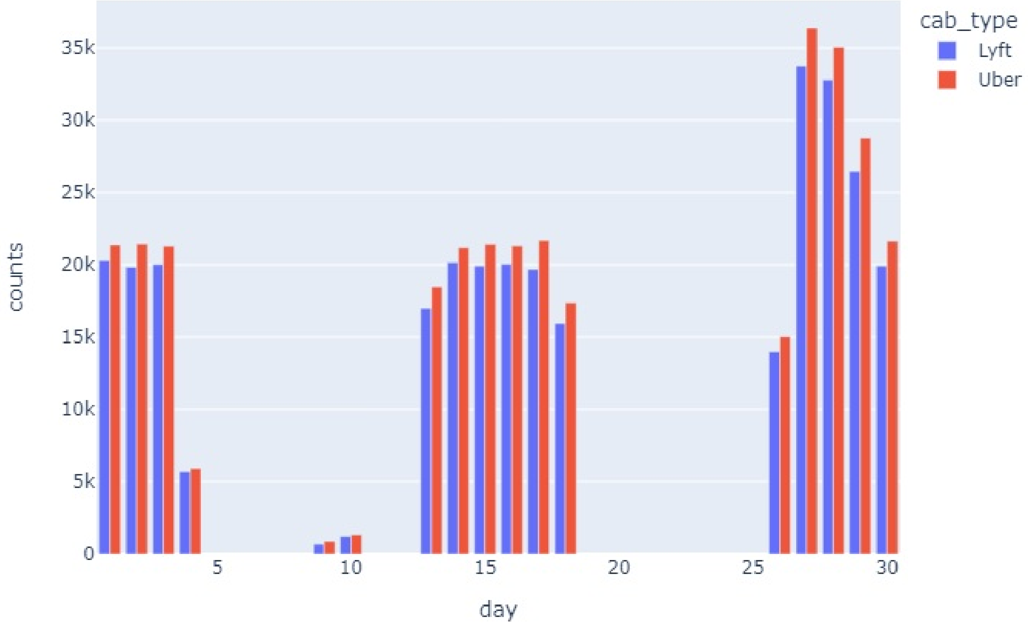
In [23]:
```
plot_bar('cab_type')
```

```
  cab_type  counts
0     Lyft  307408
1     Uber  330568
```

Observation

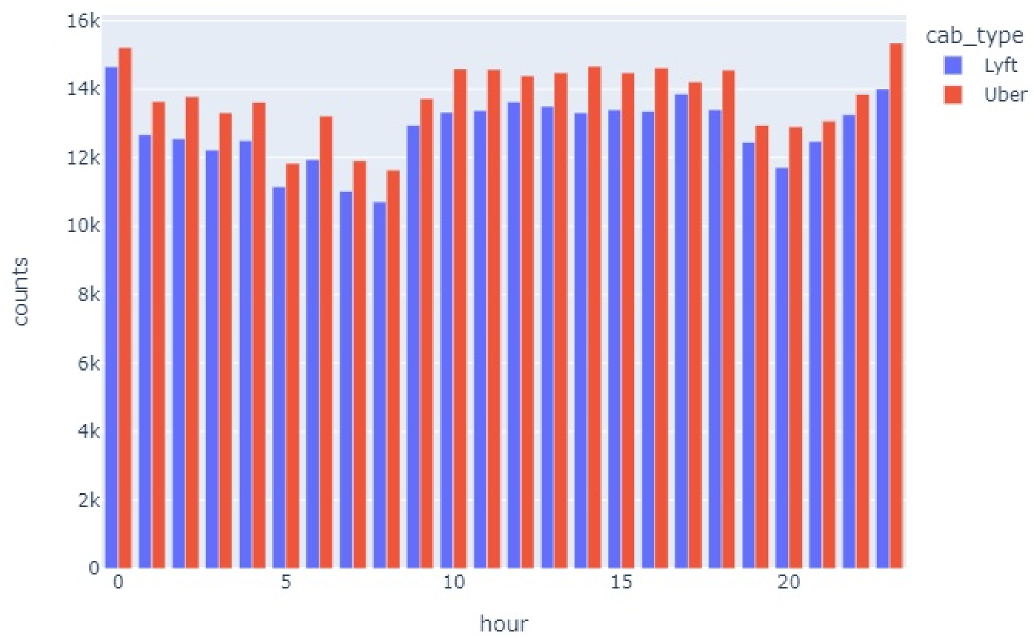
- So for our whole data, we have uber data more than lyft data. The difference is not too big, each cab type has about 300K data.

In [24]:
```
df2 =df.groupby(by=["day","cab_type"]).size().reset_index(name="counts")
fig2 = px.bar(data_frame=df2, x="day", y="counts", color="cab_type", barmode="group")
fig2.show(renderer='png')
```



In [25]:
```
df3 =df.groupby(["hour","cab_type"]).size().reset_index(name="counts")
fig3 = px.bar(data_frame=df3, x="hour", y="counts", color="cab_type", barmode="group")
```

```
fig3.show(renderer='png')
```



Observation

- So in every day and every hour recorded, uber seems dominating booking order in our data

4. Price Analysis

We can see average or mean of our price data in every route (source-destination) through table below

In [27]:
```
pd.set_option('display.max_rows', 72)
df.groupby(by=["source","destination"]).price.agg(["mean"])
```

Out[27]:

source	destination	mean
Back Bay	Boston University	14.039392
	Fenway	13.658752
	Haymarket Square	17.987384
	North End	19.473019
	Northeastern University	13.151040
	South Station	17.700711
Beacon Hill	Boston University	16.376737
	Fenway	16.158840
	Haymarket Square	13.799137
	North End	15.270942
	Northeastern University	16.471792
	South Station	15.950661
Boston University	Back Bay	13.992801
	Beacon Hill	17.315535
	Financial District	24.146085
	North Station	20.185338
	Theatre District	18.689557
	West End	18.611766
Fenway	Back Bay	13.802155
	Beacon Hill	16.796674
	Financial District	23.438818
	North Station	19.701839
	Theatre District	18.232722

	West End	18.161806
Financial District	Boston University	25.498434
	Fenway	23.404850
	Haymarket Square	13.188209
	North End	13.179635
	Northeastern University	21.918584
	South Station	12.349066
Haymarket Square	Back Bay	16.860489
	Beacon Hill	13.338559
	Financial District	12.731618
	North Station	12.332545
	Theatre District	13.677272
	West End	12.529855
North End	Back Bay	19.550935
	Beacon Hill	15.982630
	Financial District	13.417597
	North Station	12.824092
	Theatre District	15.169406
	West End	13.494873
North Station	Boston University	18.931558
	Fenway	18.547603
	Haymarket Square	12.571791
	North End	13.106641
	Northeastern University	19.537848
	South Station	15.374198
Northeastern University	Back Bay	13.698923
	Beacon Hill	16.842433
	Financial District	22.582094
	North Station	19.910939
	Theatre District	16.144805
	West End	18.204155
South Station	Back Bay	19.103822
	Beacon Hill	17.276304
	Financial District	12.436910
	North Station	15.746736
	Theatre District	13.952579
	West End	15.881172
Theatre District	Boston University	20.360662
	Fenway	19.069278
	Haymarket Square	15.204973
	North End	15.159646
	Northeastern University	16.910751
	South Station	12.888926
West End	Boston University	18.157165
	Fenway	17.932692
	Haymarket Square	12.771290
	North End	13.370017
	Northeastern University	18.964969
	South Station	15.018255

And we can see our maximum price data

In [28]:
```
print('Maximum price in our data :',df.price.max())
df[df['price']==df.price.max()]
```

```
Maximum price in our data : 97.5
```

Out[28]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	product_id	name	price
597071	ba1593a1-e4fd-4c7a-a011-e2d4fccbf081	1.543714e+09	1	2	12	2018-12-02 01:28:02	America/New_York	Financial District	Fenway	Lyft	lyft_luxsuv	Lux Black XL	97.5

In [29]:
```
df[df['price']==df.price.max()][['latitude','longitude']]
```

Out[29]:

	latitude	longitude
597071	42.3503	-71.081

I can plot the map of both places using folium to see how far they are from each other (I only inserted the snapshot of the plot)

In [30]:
```
#Using this code:
map1 = folium.Map(location=(42.3503,-71.081),zoom_start = 10)
folium.Marker(location=(42.3503,-71.081)).add_to(map1) # Fenway
folium.Marker(location=(42.3378,-71.066)).add_to(map1) # Financial District
display(map1)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

Observation

- Apparently the 'Financial District - Fenway' route (by lyft) costs 97.5 dollars, which is our maximum price data. But from the map above, the distance between both places is not too far (they are both in boston), so it could be outlier since we don't have information about trip duration or transit. We should check another data with the same route

In [31]:
```
df_group = df.groupby(by=["source","destination"]).price.agg(["mean"]).reset_index()
df_group[(df_group['source']=='Financial District')& (df_group['destination']=='Fenway')]
```

Out[31]:

	source	destination	mean
25	Financial District	Fenway	23.40485

The mean of the price data of that route is 23.4 dollars, which is far from our maximum price data (97.5 dollars). Then it is possible an outlier. We can drop it.

In [32]:
```
df = df.loc[df['price']!=df.price.max()]
```

In [33]:
```
df.head()
```

Out[33]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	product_id	name	price
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	1.544953e+09	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square	North Station	Lyft	lyft_line	Shared	5.0
1	4bd23055-6827-41c6-b23b-3c491f24e74d	1.543284e+09	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square	North Station	Lyft	lyft_premier	Lux	11.0
2	981a3613-77af-4620-a42a-0c0866077d1e	1.543367e+09	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square	North Station	Lyft	lyft	Lyft	7.0
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	1.543554e+09	4	30	11	2018-11-30 04:53:02	America/New_York	Haymarket Square	North Station	Lyft	lyft_luxsuv	Lux Black XL	26.0
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	1.543463e+09	3	29	11	2018-11-29 03:49:20	America/New_York	Haymarket Square	North Station	Lyft	lyft_plus	Lyft XL	9.0

3. Data Preprocessing / Feature Engineering

1. Removing Unnecessary Features

In [34]:
```
# For further modelling i don't think we need date related features. But maybe we need them in the future analy
# so i will make new dataframe

new_df = df.drop(['id','timestamp','datetime','long_summary','apparentTemperatureHighTime','apparentTemperature
                  'apparentTemperatureLowTime','windGustTime','sunriseTime','sunsetTime','uvIndexTime','tempera
                 'temperatureMaxTime','apparentTemperatureMinTime','temperatureLowTime','apparentTemperatureMax
```

In [35]:
```
new_df.shape
```

Out[35]: (637975, 41)

Our goal is to make linear regression model. First we check correlation between our features and target feature (price)

First, i want to check the correlation of our temperature related features with our target feature (Price)

In [37]:
```
temp_cols= ['temperature','apparentTemperature','temperatureHigh','temperatureLow','apparentTemperatureHigh',
            'apparentTemperatureLow','temperatureMin','temperatureHighTime','temperatureMax',
            'apparentTemperatureMin','apparentTemperatureMax','price']
```

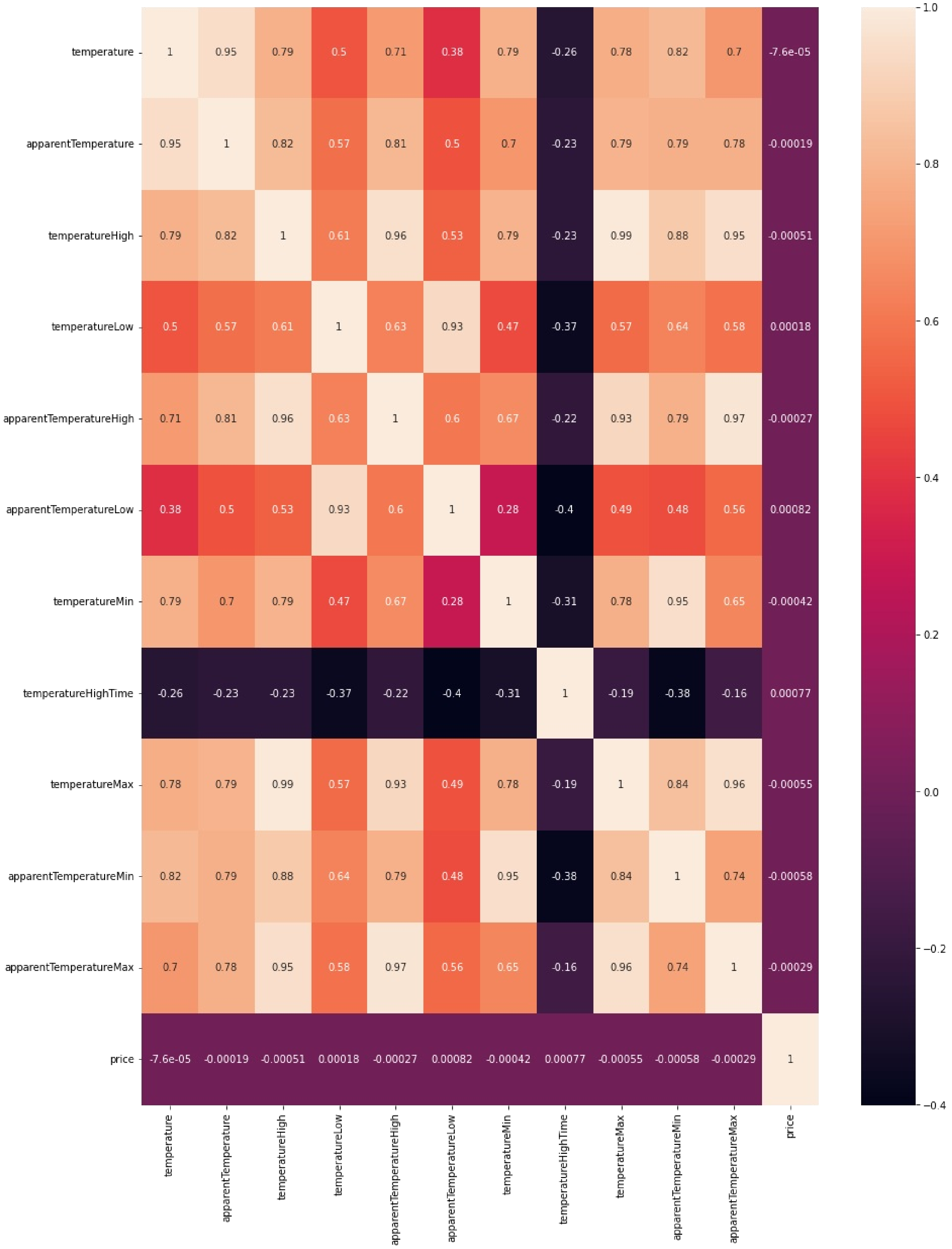
In [38]:
```
df_temp = new_df[temp_cols]
df_temp.head()
```

Out[38]:

	temperature	apparentTemperature	temperatureHigh	temperatureLow	apparentTemperatureHigh	apparentTemperatureLow	temperatureMin
0	42.34	37.12	43.68	34.19	37.95	27.39	39.89
1	43.58	37.35	47.30	42.10	43.92	36.20	40.49
2	38.33	32.93	47.55	33.10	44.12	29.11	35.36
3	34.38	29.63	45.03	28.90	38.53	26.20	34.67
4	37.44	30.88	42.18	36.71	35.75	30.29	33.10

In [39]:
```
plt.figure(figsize=(15,20))
sns.heatmap(df_temp.corr(),annot=True)
```

Out[39]: <AxesSubplot:>



Observation

- We see that all temperature related features have weak correlation with our target feature which is price

Removing all of them will not make any impact to our regression model

In [40]:
```
new_df = new_df.drop(['temperature','apparentTemperature','temperatureHigh','temperatureLow','apparentTemperatu
                'apparentTemperatureLow','temperatureMin','temperatureHighTime','temperatureMax',
                      'apparentTemperatureMin','apparentTemperatureMax'],axis=1)
new_df.shape
```

Out[40]: (637975, 30)

Second, i want to check the correlation of our cilmate related features with our target feature (Price)¶

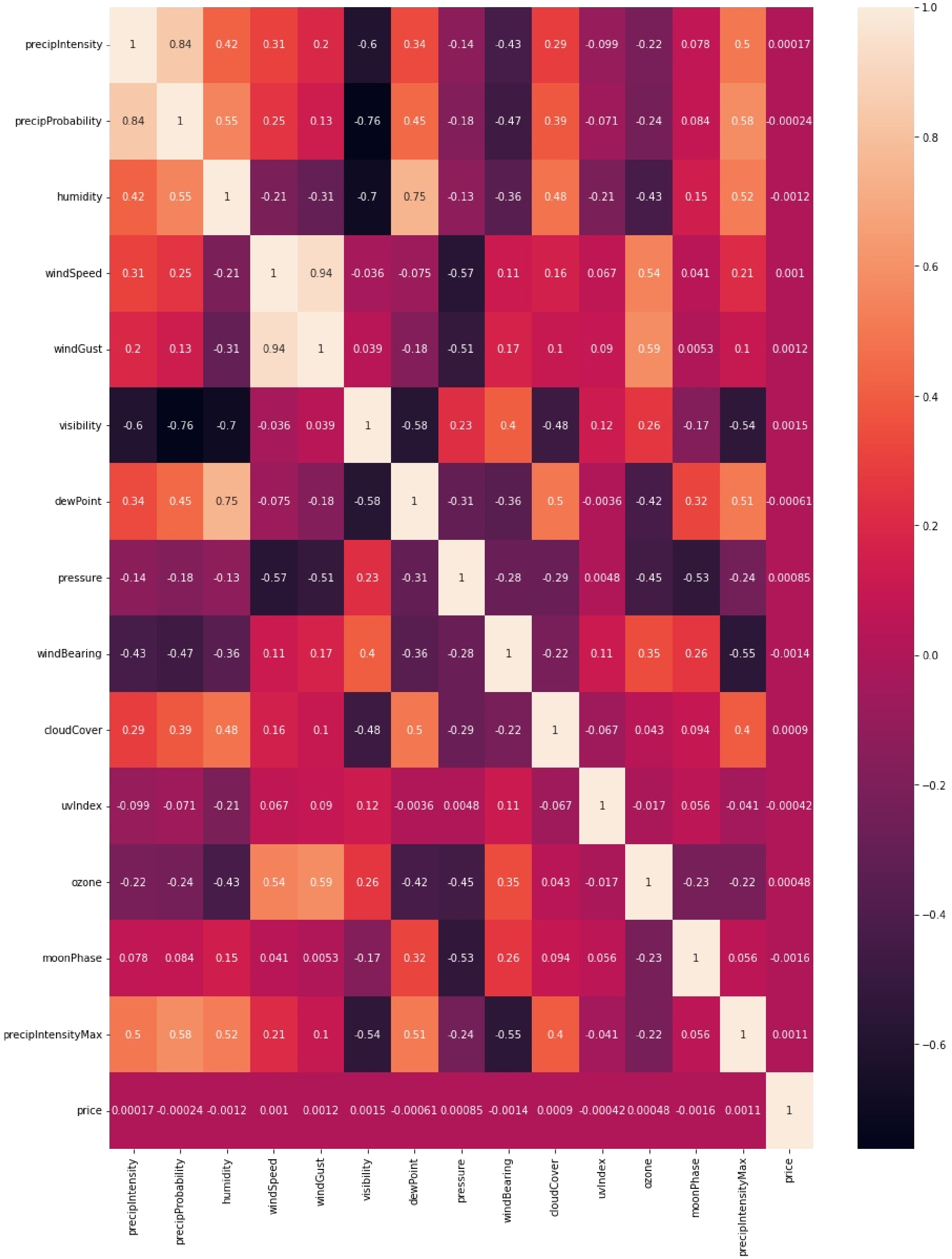
In [41]:
```
climate_column = ['precipIntensity', 'precipProbability', 'humidity', 'windSpeed',
       'windGust', 'visibility', 'dewPoint', 'pressure', 'windBearing',
       'cloudCover', 'uvIndex', 'ozone', 'moonPhase',
       'precipIntensityMax','price']
df_clim = new_df[climate_column]
df_clim.head()
```

Out[41]:

	precipIntensity	precipProbability	humidity	windSpeed	windGust	visibility	dewPoint	pressure	windBearing	cloudCover	uvIndex	ozone
0	0.0000	0.0	0.68	8.66	9.17	10.000	32.70	1021.98	57	0.72	0	303.8
1	0.1299	1.0	0.94	11.98	11.98	4.786	41.83	1003.97	90	1.00	0	291.1
2	0.0000	0.0	0.75	7.33	7.33	10.000	31.10	992.28	240	0.03	0	315.7
3	0.0000	0.0	0.73	5.28	5.28	10.000	26.64	1013.73	310	0.00	0	291.1
4	0.0000	0.0	0.70	9.14	9.14	10.000	28.61	998.36	303	0.44	0	347.7

In [42]:
```
plt.figure(figsize=(15,20))
sns.heatmap(df_clim.corr(),annot=True)
```

Out[42]: <AxesSubplot:>



Observation

- Apparently all climate related features also have weak correlation with our target feature which is price.

Once again, removing all of them will not make any impact to our regression model

In [44]:
```
new_df = new_df.drop(['precipIntensity', 'precipProbability', 'humidity', 'windSpeed',
        'windGust', 'visibility', 'dewPoint', 'pressure', 'windBearing',
        'cloudCover', 'uvIndex', 'ozone', 'moonPhase',
        'precipIntensityMax'],axis=1)
new_df.shape
```

Out[44]:
```
(637975, 16)
```

Third, i want to check our categorical value in our dataset features

In [45]:
```
category_col = new_df.select_dtypes(include=['object','category']).columns.tolist()
for column in new_df[category_col]:
    print(f'{column} : {new_df[column].unique()}')
    print()
```

```
timezone : ['America/New_York']

source : ['Haymarket Square' 'Back Bay' 'North End' 'North Station' 'Beacon Hill'
 'Boston University' 'Fenway' 'South Station' 'Theatre District'
 'West End' 'Financial District' 'Northeastern University']

destination : ['North Station' 'Northeastern University' 'West End' 'Haymarket Square'
 'South Station' 'Fenway' 'Theatre District' 'Beacon Hill' 'Back Bay'
 'North End' 'Financial District' 'Boston University']

cab_type : ['Lyft' 'Uber']

product_id : ['lyft_line' 'lyft_premier' 'lyft' 'lyft_luxsuv' 'lyft_plus' 'lyft_lux'
 '6f72dfc5-27f1-42e8-84db-ccc7a75f6969'
 '6c84fd89-3f11-4782-9b50-97c468b19529'
 '55c66225-fbe7-4fd5-9072-eab1ece5e23e'
 '9a0e7b09-b92b-4c41-9779-2ad22b4d779d'
 '6d318bcc-22a3-4af6-bddd-b409bfce1546'
 '997acbb5-e102-41e1-b155-9df7de0a73f2']

name : ['Shared' 'Lux' 'Lyft' 'Lux Black XL' 'Lyft XL' 'Lux Black' 'UberXL'
 'Black' 'UberX' 'WAV' 'Black SUV' 'UberPool']

short_summary : [' Mostly Cloudy ' ' Rain ' ' Clear ' ' Partly Cloudy ' ' Overcast '
 ' Light Rain ' ' Foggy ' ' Possible Drizzle ' ' Drizzle ']

icon : [' partly-cloudy-night ' ' rain ' ' clear-night ' ' cloudy ' ' fog '
 ' clear-day ' ' partly-cloudy-day ']
```

Observation

- We can see that 'timezone' feature has only 1 value and 'product_id' feature contains many unidentified values. So we can remove or drop them.

In [46]:
```
new_df = new_df.drop(['timezone','product_id'],axis=1)
```

In [47]:
```
new_df.shape
```

Out[47]: (637975, 14)

Fourth, i want to check the correlation of our categorical features with our target feature (price)

In [48]:
```
new_cat = ['source',
 'destination',
 'cab_type',
 'name',
 'short_summary',
 'icon','price']

df_cat = new_df[new_cat]
df_cat.head()
```

Out[48]:

	source	destination	cab_type	name	short_summary	icon	price
0	Haymarket Square	North Station	Lyft	Shared	Mostly Cloudy	partly-cloudy-night	5.0
1	Haymarket Square	North Station	Lyft	Lux	Rain	rain	11.0
2	Haymarket Square	North Station	Lyft	Lyft	Clear	clear-night	7.0
3	Haymarket Square	North Station	Lyft	Lux Black XL	Clear	clear-night	26.0
4	Haymarket Square	North Station	Lyft	Lyft XL	Partly Cloudy	partly-cloudy-night	9.0

In [49]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

df_cat_encode= df_cat.copy()
for col in df_cat_encode.select_dtypes(include='O').columns:
    df_cat_encode[col]=le.fit_transform(df_cat_encode[col])
```

In [50]:
```
df_cat_encode
```

Out[50]:

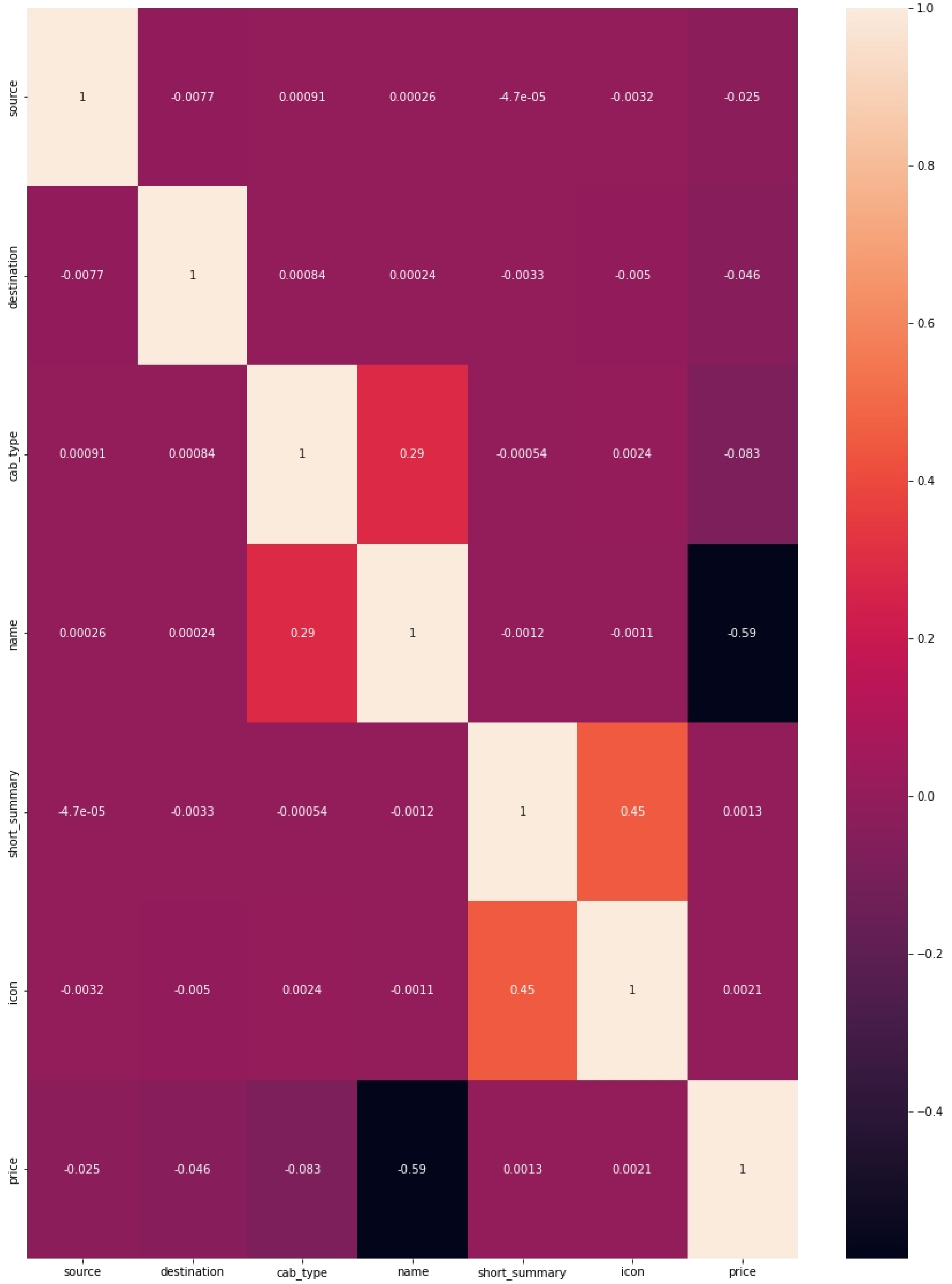
	source	destination	cab_type	name	short_summary	icon	price
0	5	7	0	7	4	5	5.0
1	5	7	0	2	8	6	11.0
2	5	7	0	5	0	1	7.0
3	5	7	0	4	0	1	26.0
4	5	7	0	6	6	5	9.0
...	...	...	...	...	...	...	...
693065	11	6	1	11	6	5	9.5
693066	11	6	1	10	6	5	13.0
693067	11	6	1	9	6	5	9.5
693069	11	6	1	1	6	5	27.0
693070	11	6	1	8	6	5	10.0

637975 rows × 7 columns

In [51]:

```
plt.figure(figsize=(15,20))
sns.heatmap(df_cat_encode.corr(),annot=True)
```

Out[51]: <AxesSubplot:>



Observation

- We can see only name feature that has a relatively strong correlation. Source,destination, and cab_type features have relatively weak correlation, but i will pick cab_type feature because it has stronger correlation than other two features. I will drop or remove the rest of the columns

In [52]:
```
new_df = new_df.drop(['source','destination','short_summary','icon'],axis=1)
new_df.head()
```

Out[52]:

	hour	day	month	cab_type	name	price	distance	surge_multiplier	latitude	longitude
0	9	16	12	Lyft	Shared	5.0	0.44	1.0	42.2148	-71.033
1	2	27	11	Lyft	Lux	11.0	0.44	1.0	42.2148	-71.033
2	1	28	11	Lyft	Lyft	7.0	0.44	1.0	42.2148	-71.033
3	4	30	11	Lyft	Lux Black XL	26.0	0.44	1.0	42.2148	-71.033
4	3	29	11	Lyft	Lyft XL	9.0	0.44	1.0	42.2148	-71.033

Also i will remove hour, day, month, latitude, longitude, because we won't need them for now

In [53]:

```
new_df = new_df.drop(['hour','day','month','latitude','longitude'],axis=1)
new_df.head()
```

Out[53]:

	cab_type	name	price	distance	surge_multiplier
0	Lyft	Shared	5.0	0.44	1.0
1	Lyft	Lux	11.0	0.44	1.0
2	Lyft	Lyft	7.0	0.44	1.0
3	Lyft	Lux Black XL	26.0	0.44	1.0
4	Lyft	Lyft XL	9.0	0.44	1.0

In [54]:

```
new_df.columns
```

Out[54]:

```
Index(['cab_type', 'name', 'price', 'distance', 'surge_multiplier'], dtype='object')
```

2. Removing Outliers

We've already done this before but only to one instance which has maximum price value. We want to check another possible outlier.

We're using IQR method for checking top and bottom outliers

In [55]:

```
Qp12 = new_df['price'].quantile(0.25)
Qp32 = new_df['price'].quantile(0.75)
IQRp = Qp32-Qp12
```

In [56]:

```
new_df[new_df['price']>(Qp32+(1.5*IQRp))]
```

Out[56]:

	cab_type	name	price	distance	surge_multiplier
706	Lyft	Lux Black	52.5	3.25	2.00
707	Lyft	Lux Black XL	67.5	3.25	2.00
769	Lyft	Lux Black XL	45.5	4.76	1.00
1094	Lyft	Lux Black XL	45.5	4.31	1.00
1318	Lyft	Lux Black XL	45.5	5.33	1.00
...	...	...	...	...	...
692439	Uber	Black SUV	47.0	5.56	1.00
692698	Lyft	Lux Black XL	52.5	4.58	1.25
692891	Lyft	Lux Black XL	47.5	5.42	1.00
692962	Uber	Black SUV	51.0	7.36	1.00
693007	Uber	Black SUV	49.5	7.36	1.00

5588 rows × 5 columns

In [57]:

```
new_df[new_df['price']<(Qp12-(1.5*IQRp))]
```

Out[57]:

cab_type	name	price	distance	surge_multiplier

We can see that we have 5588 data outliers. We can remove or drop them.

In [58]:

```
print('Size before removing :',new_df.shape)
new_df= new_df[~((new_df['price']>(Qp32+(1.5*IQRp))))]
print('Size after removing :',new_df.shape)
```

```
Size before removing : (637975, 5)
Size after removing : (632387, 5)
```

4. Regression Model

1. Encoding Data (One Hot Encoding)

In [59]:
```
def one_hot_encoder(data,feature,keep_first=True):

    one_hot_cols = pd.get_dummies(data[feature])

    for col in one_hot_cols.columns:
        one_hot_cols.rename({col:f'{feature}_'+col},axis=1,inplace=True)

    new_data = pd.concat([data,one_hot_cols],axis=1)
    new_data.drop(feature,axis=1,inplace=True)

    if keep_first == False:
        new_data=new_data.iloc[:,1:]

    return new_data
```

In [60]:
```
new_df_onehot=new_df.copy()
for col in new_df_onehot.select_dtypes(include='O').columns:
    new_df_onehot=one_hot_encoder(new_df_onehot,col)

new_df_onehot.head()
```

Out[60]:

	price	distance	surge_multiplier	cab_type_Lyft	cab_type_Uber	name_Black	name_Black SUV	name_Lux	name_Lux Black	name_Lux Black XL	name_Lyft
0	5.0	0.44	1.0	1	0	0	0	0	0	0	0
1	11.0	0.44	1.0	1	0	0	0	1	0	0	0
2	7.0	0.44	1.0	1	0	0	0	0	0	0	1
3	26.0	0.44	1.0	1	0	0	0	0	0	1	0
4	9.0	0.44	1.0	1	0	0	0	0	0	0	0

2. Dataset Split

In [61]:
```
from sklearn.model_selection import train_test_split
X = new_df_onehot.drop(columns=['price'],axis=1).values
y = new_df_onehot['price'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

3. Modeling

3.1. Base Model

In [62]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
```

In [63]:
```
# Fit to data training
model = reg.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

In [64]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[64]:
```
0.9337792706482413
```

In [65]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
print(mse)
print(rmse)
```

```
5.108354908943204
2.2601670090821173
```

Observation

- Then for the long journey we have done, we got our regression model with accuracy or score 93.37% and RMSE value 2.26. It's not the best score though, we still can improve it with other regression models which could give better results.¶

3.2. Finding Best Models with best configuration with GridSearch CV

In [68]:
```
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV,ShuffleSplit

def find_best_model_using_gridsearchcv(X,y):
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })


    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

import warnings
warnings.filterwarnings('ignore')

find_best_model_using_gridsearchcv(X,y)
```

Out[68]:

	model	best_score	best_params
0	linear_regression	0.933465	{'normalize': False}
1	lasso	0.211560	{'alpha': 1, 'selection': 'cyclic'}
2	decision_tree	0.964469	{'criterion': 'mse', 'splitter': 'best'}

Observation

Here we got our best model is decision tree regressor with r-squared 0.964, higher than our linear regression before.¶