INFO6250

United States House Rent Prediction

Team 12

Prashanti Salunkhe : 002133330 | Pratik Randad : 002133847

• Introduction

The main goal of the research is to create a prediction model using Machine Learning to predict the rental costs of houses across the United States based on numerous variables defining the houses' attributes. This dataset contains several features that characterize the entire nature of the house and its dynamics, as well as ‘Price’ which is to be predicted.

In [1]:
```
import geopandas as gpd
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set();
%matplotlib inline
```

• Importing Data

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

• Data Description

In [4]:
```
src_data.head()
```

Out[4]:

	id	url	region	region_url
0	7039061606	https://bham.craigslist.org/apa/d/birmingham-h...	birmingham	https://bham.craigslist.org
1	7041970863	https://bham.craigslist.org/apa/d/birmingham-w...	birmingham	https://bham.craigslist.org
2	7041966914	https://bham.craigslist.org/apa/d/birmingham-g...	birmingham	https://bham.craigslist.org
3	7041966936	https://bham.craigslist.org/apa/d/birmingham-f...	birmingham	https://bham.craigslist.org
4	7041966888	https://bham.craigslist.org/apa/d/birmingham-2...	birmingham	https://bham.craigslist.org

5 rows × 22 columns

In [5]:
```
src_data.tail()
```

Out[5]:

	id	url	region	r
265185	7050851033	https://columbus.craigslist.org/apa/d/columbus...	columbus	https://columbus.cra
265186	7050887997	https://columbus.craigslist.org/apa/d/grove-ci...	columbus	https://columbus.cra
265187	7044801015	https://columbus.craigslist.org/apa/d/columbus...	columbus	https://columbus.cra
265188	7050885800	https://columbus.craigslist.org/apa/d/newark-l...	columbus	https://columbus.cra
265189	7050884586	https://columbus.craigslist.org/apa/d/columbus...	columbus	https://columbus.cra

5 rows × 22 columns

In [6]:
```
src_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 265190 entries, 0 to 265189
Data columns (total 22 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   id                       265190 non-null  int64  
 1   url                      265190 non-null  object 
 2   region                   265190 non-null  object 
 3   region_url               265190 non-null  object 
 4   price                    265190 non-null  int64  
 5   type                     265190 non-null  object 
 6   sqfeet                   265190 non-null  int64  
 7   beds                     265190 non-null  int64  
 8   baths                    265190 non-null  float64
 9   cats_allowed             265190 non-null  int64  
 10  dogs_allowed             265190 non-null  int64  
 11  smoking_allowed          265190 non-null  int64  
 12  wheelchair_access        265190 non-null  int64  
 13  electric_vehicle_charge  265190 non-null  int64  
 14  comes_furnished          265190 non-null  int64  
 15  laundry_options          210879 non-null  object 
 16  parking_options          170055 non-null  object 
 17  image_url                265190 non-null  object 
 18  description              265188 non-null  object 
 19  lat                      263771 non-null  float64
 20  long                     263771 non-null  float64
 21  state                    265189 non-null  object 
dtypes: float64(3), int64(10), object(9)
memory usage: 44.5+ MB
```

There are 13 numerical and 9 category features in all.

In [7]:
```
src_data.describe()
```

Out[7]:

	id	price	sqfeet	beds	baths	cats_allov
count	2.651900e+05	2.651900e+05	2.651900e+05	265190.000000	265190.000000	265190.000
mean	7.040888e+09	1.227285e+04	1.093678e+03	1.912414	1.483468	0.716
std	8.778930e+06	5.376352e+06	2.306888e+04	3.691900	0.630208	0.450
min	7.003808e+09	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000
25%	7.035963e+09	8.170000e+02	7.520000e+02	1.000000	1.000000	0.000
50%	7.043109e+09	1.060000e+03	9.500000e+02	2.000000	1.000000	1.000
75%	7.048362e+09	1.450000e+03	1.156000e+03	2.000000	2.000000	1.000
max	7.051263e+09	2.768307e+09	8.388607e+06	1100.000000	75.000000	1.000

In [8]:
```
print(src_data.columns.shape)
```

```
(22,)
```

Our dataset has a total of 265,190 rows and 22 columns.

• Data Cleaning

1.Dropping Unwanted Columns

We can remove the columns id, url, region url, image url, and description from our dataset because we know they aren't needed for our current situation.

In [9]:
```
data = src_data.drop(columns = ['id', 'url', 'region_url', 'image_url', 'descri
```

In [10]:
```
data.head()
```

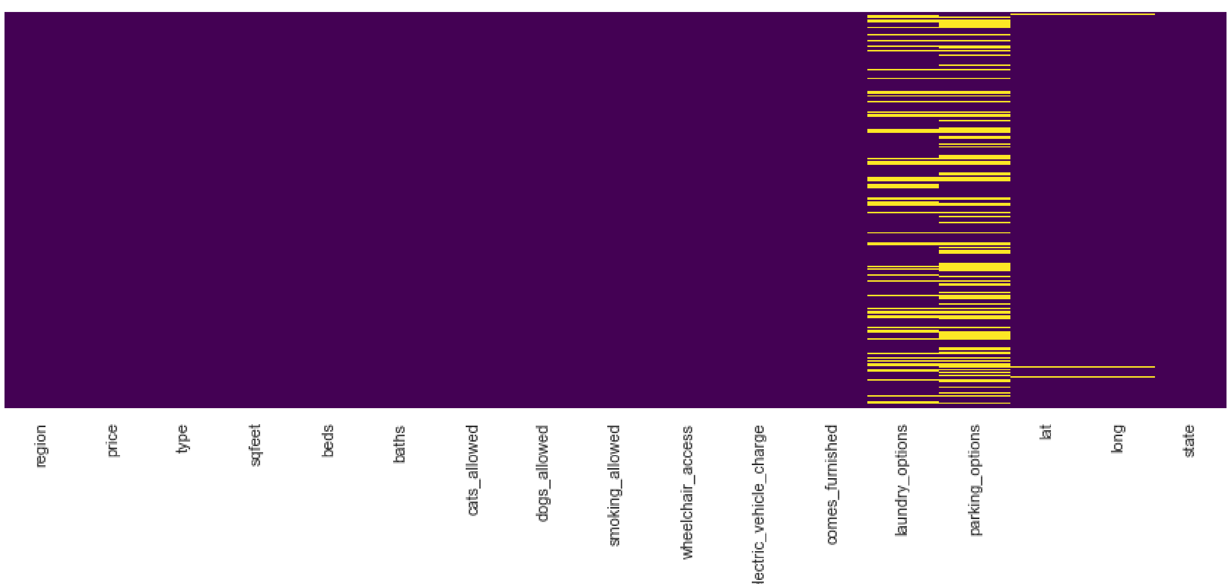
Out[10]:

	region	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allo
0	birmingham	1195	apartment	1908	3	2.0	1	1	
1	birmingham	1120	apartment	1319	3	2.0	1	1	
2	birmingham	825	apartment	1133	1	1.5	1	1	
3	birmingham	800	apartment	927	1	1.0	1	1	
4	birmingham	785	apartment	1047	2	1.0	1	1	

2.Analyzing and Filling Null Values

In [11]:
```
plt.figure(figsize=(15,5))
sns.heatmap(data.isnull(),yticklabels=False,cmap='viridis',cbar=False)
```

Out[11]: <AxesSubplot:>



In [12]:
```
data.isna().sum()
```

Out[12]:
```
region                        0
price                         0
type                          0
sqfeet                        0
beds                          0
baths                         0
cats_allowed                  0
dogs_allowed                  0
smoking_allowed               0
wheelchair_access             0
electric_vehicle_charge       0
comes_furnished               0
laundry_options           54311
parking_options           95135
lat                        1419
long                       1419
state                         1
dtype: int64
```

We see that the 'lat' and 'long' columns have less null values, so we eliminate them.

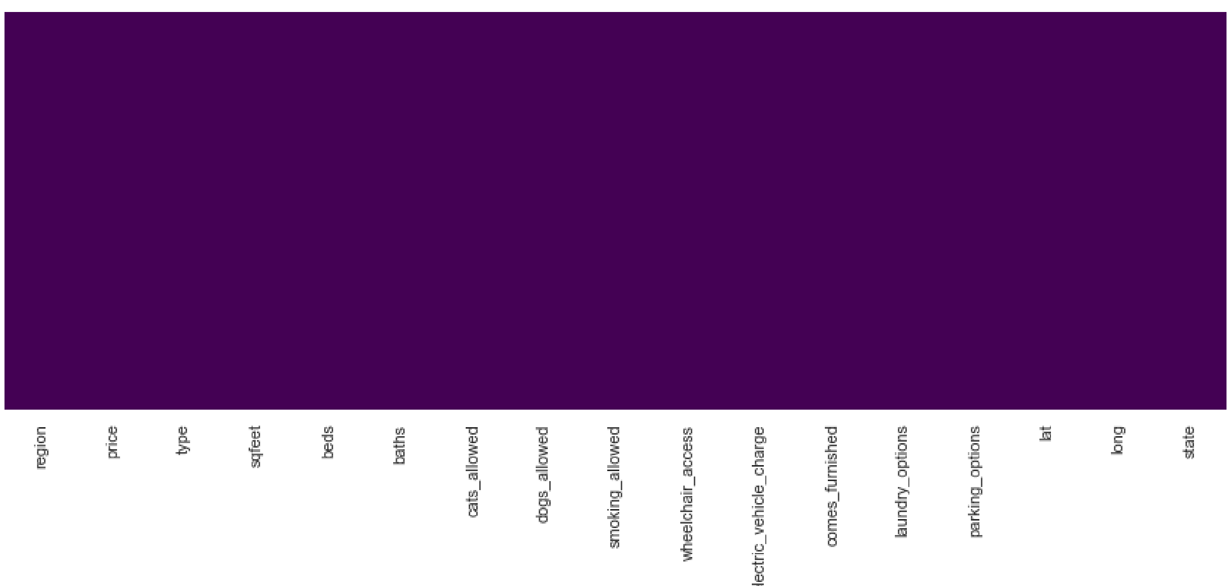
In [13]:
```
data=data[data['lat'].notna()]
```

Used the most frequent values in each column to fill in the missing values in each columns, thereby eliminating null values

In [14]:
```
data=data.fillna(data.mode().iloc[0])
```

In [15]:
```
plt.figure(figsize=(15,5))
sns.heatmap(data.isnull(),yticklabels=False,cmap='viridis',cbar=False)
```

Out[15]: <AxesSubplot:>



- ## Identifying and Handling Outliers

Checking outliers in each column

In [16]:
```
data.columns
```

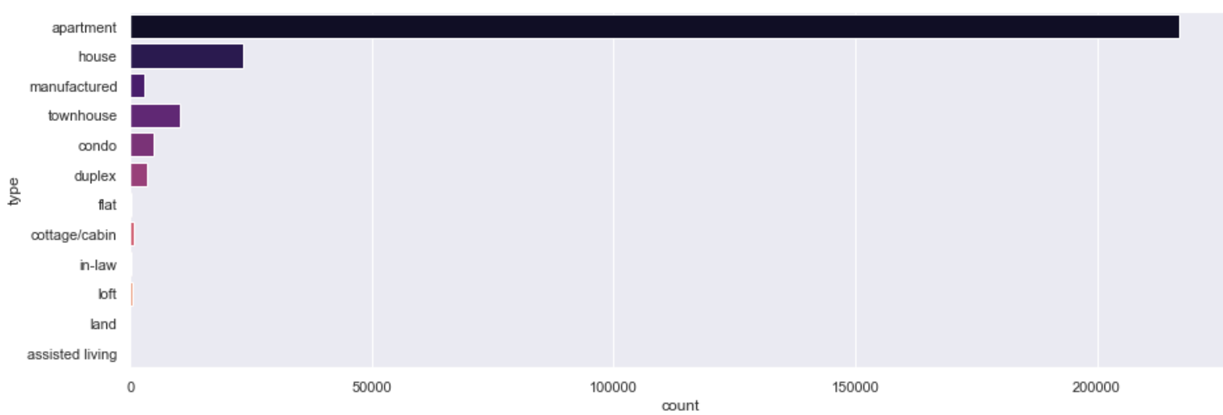
Out[16]:
```
Index(['region', 'price', 'type', 'sqfeet', 'beds', 'baths', 'cats_allowed',
       'dogs_allowed', 'smoking_allowed', 'wheelchair_access',
       'electric_vehicle_charge', 'comes_furnished', 'laundry_options',
       'parking_options', 'lat', 'long', 'state'],
      dtype='object')
```

In [17]:
```
data['type'].value_counts()
```

Out[17]:
```
apartment          217090
house               23400
townhouse           10295
condo                4841
duplex               3436
manufactured         3004
cottage/cabin         697
loft                  510
flat                  349
in-law                144
land                    4
assisted living         1
Name: type, dtype: int64
```

In [18]:
```
plt.figure(figsize=(15,5))
sns.set_theme(style="darkgrid")
sns.countplot(y ='type',data = data,palette="magma")
```

Out[18]:
```
<AxesSubplot:xlabel='count', ylabel='type'>
```



Because the data on land and assisted living is so small in comparison to the rest of the data, deleting them is the greatest choice for improving accuracy.

In [19]:
```
data=data[data['type']!='land']
data=data[data['type']!='assisted living']
```

In [20]:
```
data['sqfeet'].value_counts()
```

Out[20]:
```
1000    7499
900     5781
800     5447
1200    4844
1100    4706
        ...
3097       1
2316       1
2193       1
2854       1
255        1
Name: sqfeet, Length: 3033, dtype: int64
```

As per the data from "statista.com" the average house size in all of the United States is 3,000 square feet, so restricting the value to above 3000

In [21]:
```
#checking houses with squarefeet greater than 3000
data[data.sqfeet > 3000]
```

Out[21]:

	region	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smok
62	birmingham	1260	apartment	5201	3	2.0	1	1	
1237	huntsville / decatur	559	apartment	4980	0	1.0	1	1	
1930	huntsville / decatur	1200	house	3077	5	3.5	1	1	
2021	huntsville / decatur	5	house	5550	5	4.0	0	0	
2035	huntsville / decatur	5	house	5550	5	4.0	0	0	
...	...	...	...	...	...	...	...	...	
263801	cleveland	7995	house	9972	6	7.0	0	0	
263804	cleveland	3999	house	7000	5	5.0	0	0	
263824	cleveland	3000	house	5416	5	5.5	0	0	
264198	cleveland	3500	townhouse	6000	2	2.0	1	1	
264641	cleveland	2800	house	5200	6	3.5	0	0	

976 rows × 17 columns

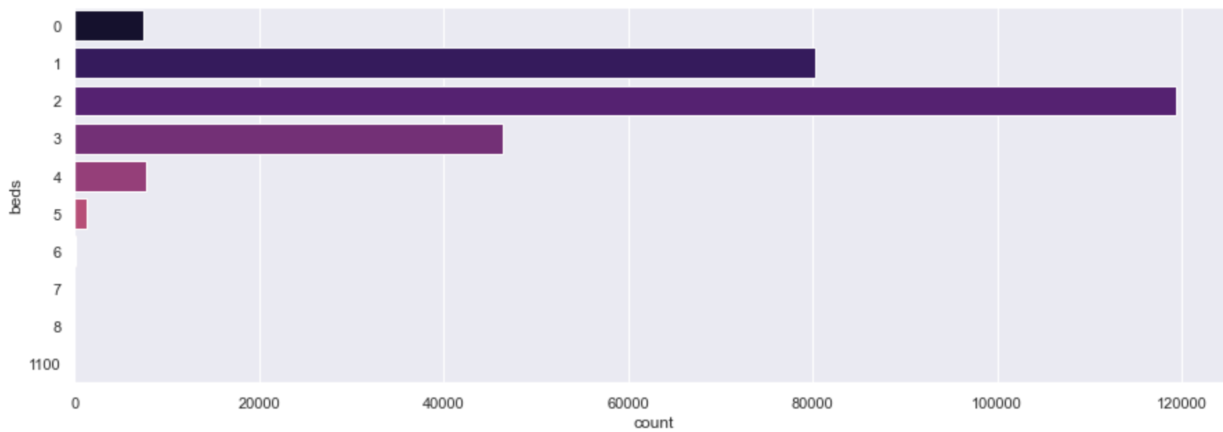
In [22]:
```
data=data[data.sqfeet <= 3000].reset_index(drop = True)
```

In [23]:
```
data['beds'].value_counts()
```

Out[23]:
```
2       119452
1        80256
3        46404
4         7713
0         7461
5         1347
6          117
7           22
8           16
1100         2
Name: beds, dtype: int64
```

In [24]:
```
plt.figure(figsize=(15,5))
sns.countplot(y ='beds',data = data,palette=sns.color_palette('magma',10))
```

Out[24]: <AxesSubplot:xlabel='count', ylabel='beds'>



The number of a beds cannot possibly be 1100, so we dropped it.

In [25]:
```
data=data[data['beds']<1100].reset_index(drop=True)
```

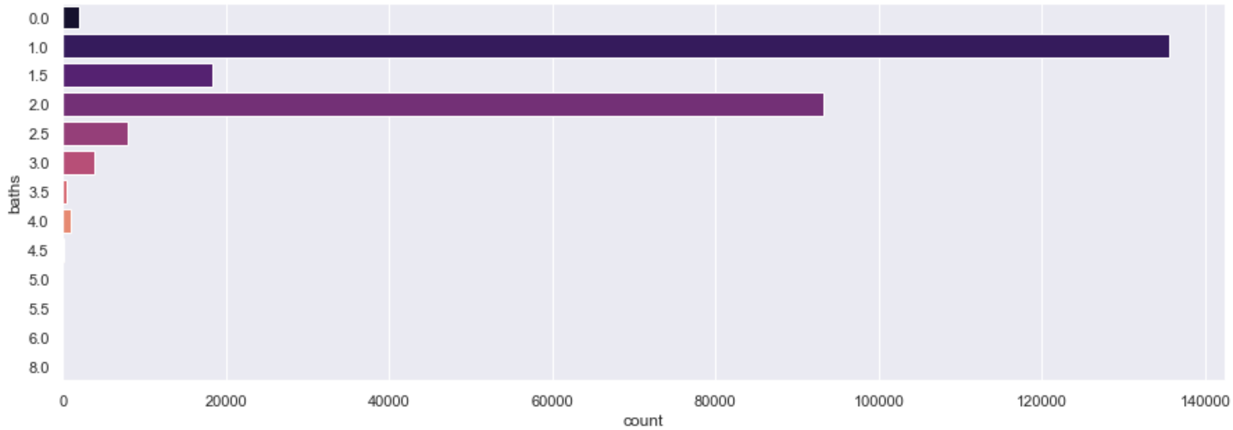
In [26]:
```
data['baths'].value_counts()
```

Out[26]:
```
1.0    135652
2.0     93160
1.5     18377
2.5      7997
3.0      3944
0.0      2024
4.0       988
3.5       475
4.5        77
5.0        65
5.5        21
6.0         7
8.0         1
Name: baths, dtype: int64
```

Number of bathrooms definitely cannot be 0 and more than 6 are outliers so removing them.

In [27]:
```
plt.figure(figsize=(15,5))
sns.countplot(y ='baths',data = data,palette=sns.color_palette('magma',10))
```

Out[27]: <AxesSubplot:xlabel='count', ylabel='baths'>



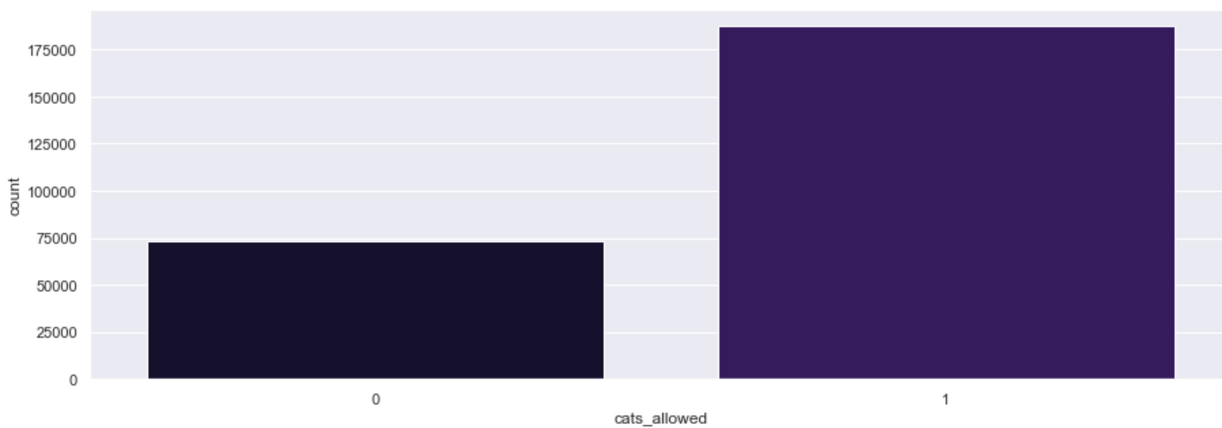
In [28]:
```
data=data.loc[(data['baths']>0) & (data['baths']<7)].reset_index(drop=True)
```

In [29]:
```
data['cats_allowed'].value_counts()
```

Out[29]:
```
1    187246
0     73517
Name: cats_allowed, dtype: int64
```

In [30]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='cats_allowed',data = data,palette=sns.color_palette('magma',1
```

Out[30]: <AxesSubplot:xlabel='cats_allowed', ylabel='count'>

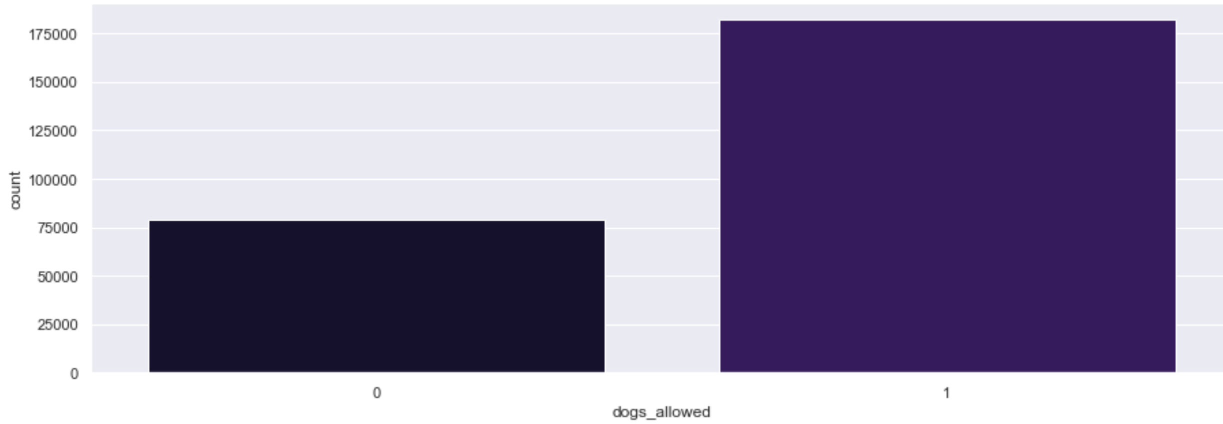


In [31]:
```
data['dogs_allowed'].value_counts()
```

Out[31]:
```
1    181843
0     78920
Name: dogs_allowed, dtype: int64
```

In [32]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='dogs_allowed',data = data,palette=sns.color_palette('magma',1
```

Out[32]: <AxesSubplot:xlabel='dogs_allowed', ylabel='count'>

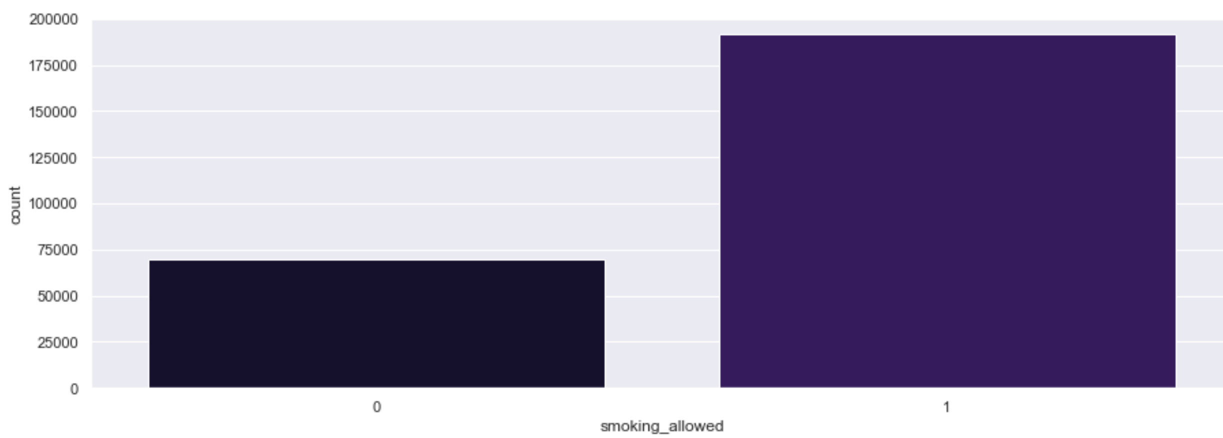


In [33]:
```
data['smoking_allowed'].value_counts()
```

Out[33]:
```
1    191381
0     69382
Name: smoking_allowed, dtype: int64
```

In [34]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='smoking_allowed',data = data,palette=sns.color_palette('magma
```

Out[34]:
```
<AxesSubplot:xlabel='smoking_allowed', ylabel='count'>
```

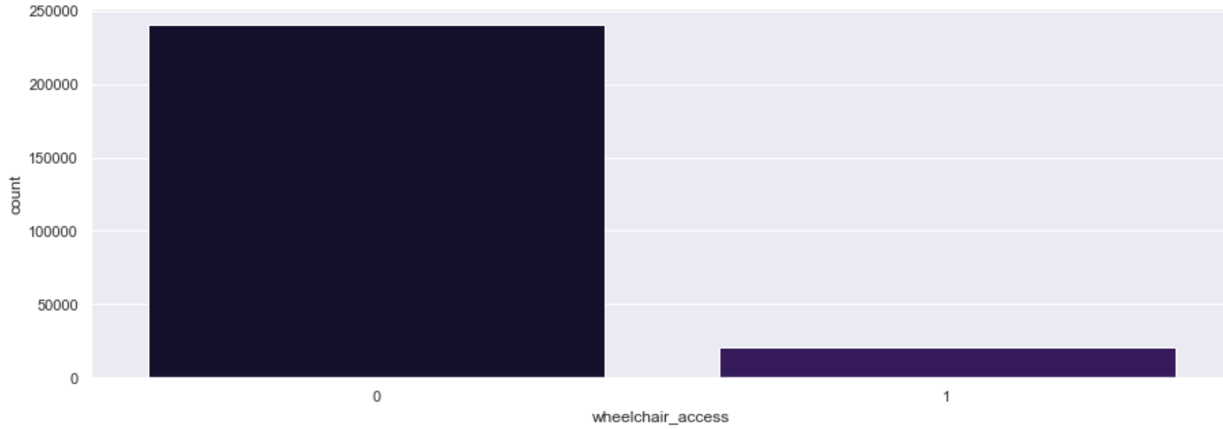


In [35]:
```
data['wheelchair_access'].value_counts()
```

Out[35]:
```
0    240057
1     20706
Name: wheelchair_access, dtype: int64
```

In [36]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='wheelchair_access',data = data,palette=sns.color_palette('mag
```

Out[36]:
```
<AxesSubplot:xlabel='wheelchair_access', ylabel='count'>
```

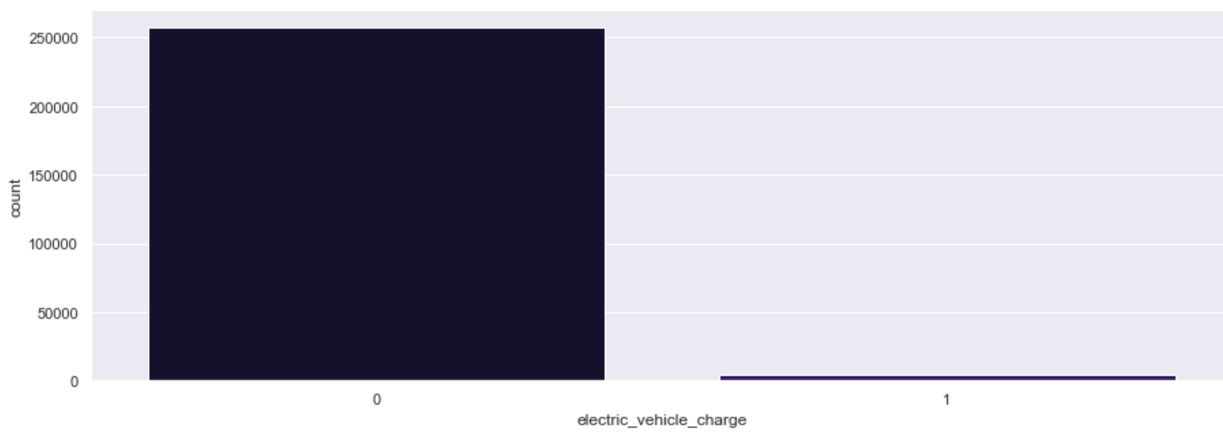


In [37]:
```
data['electric_vehicle_charge'].value_counts()
```

Out[37]:
```
0    256995
1      3768
Name: electric_vehicle_charge, dtype: int64
```

In [38]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='electric_vehicle_charge',data = data,palette=sns.color_palett
```

Out[38]:
```
<AxesSubplot:xlabel='electric_vehicle_charge', ylabel='count'>
```

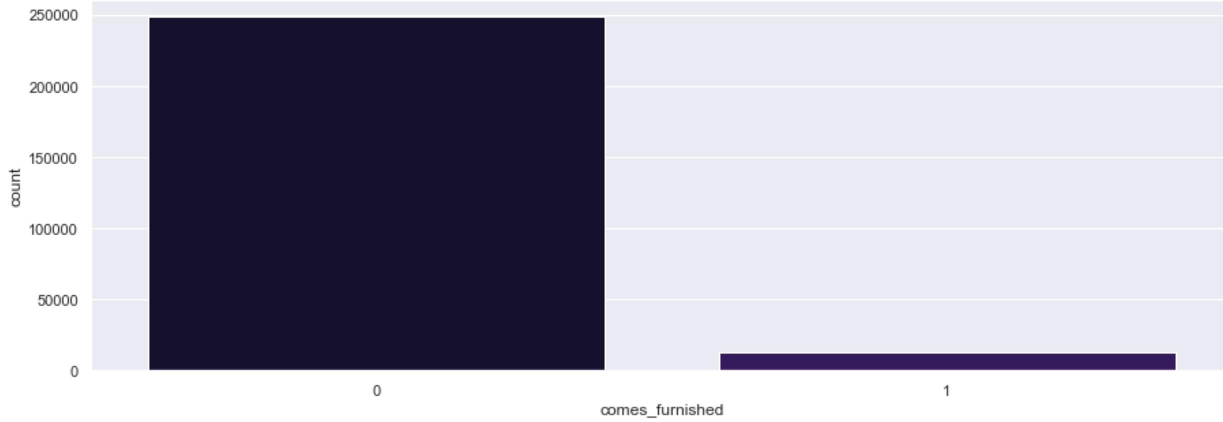


In [39]:
```
data['comes_furnished'].value_counts()
```

Out[39]:
```
0    248364
1     12399
Name: comes_furnished, dtype: int64
```

In [40]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='comes_furnished',data = data,palette=sns.color_palette('magma
```

Out[40]:
```
<AxesSubplot:xlabel='comes_furnished', ylabel='count'>
```

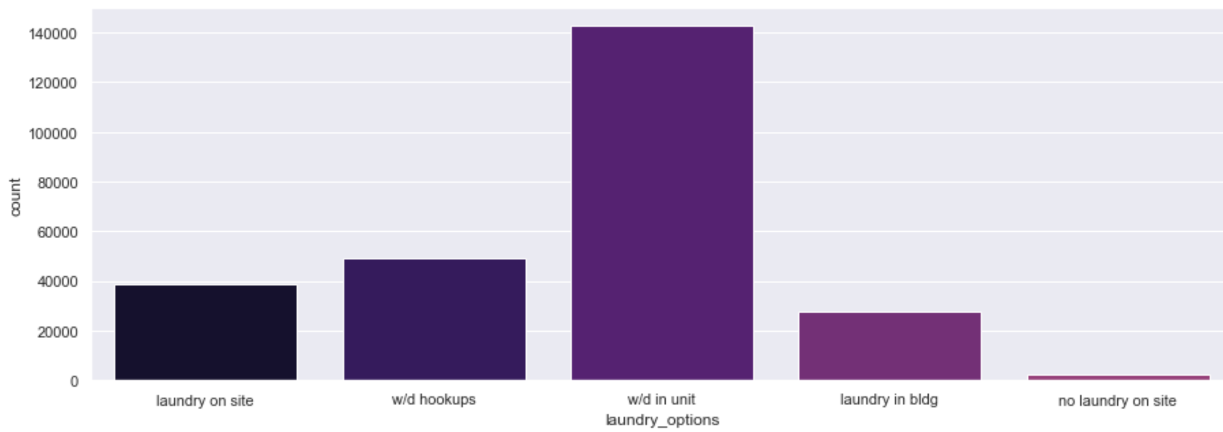


In [41]:
```
data['laundry_options'].value_counts()
```

Out[41]:
```
w/d in unit          142655
w/d hookups           49366
laundry on site       38819
laundry in bldg       27422
no laundry on site     2501
Name: laundry_options, dtype: int64
```

In [42]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='laundry_options',data = data,palette=sns.color_palette('magma
```

Out[42]:
```
<AxesSubplot:xlabel='laundry_options', ylabel='count'>
```

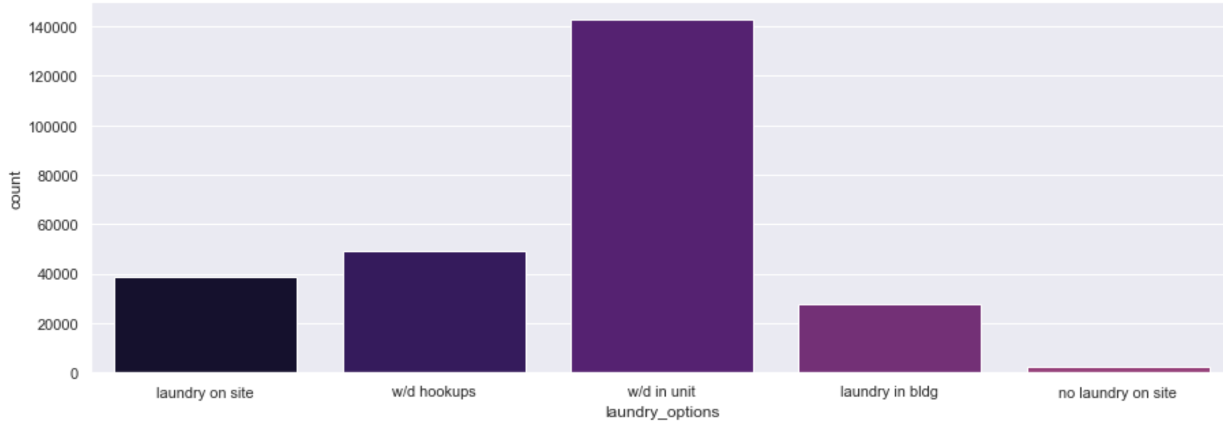


In [43]:
```
data['parking_options'].value_counts()
```

Out[43]:
```
off-street parking    180532
carport                28432
attached garage        26594
detached garage        12678
street parking         10456
no parking              1952
valet parking            119
Name: parking_options, dtype: int64
```

In [44]:
```
plt.figure(figsize=(15,5))
sns.countplot(x ='laundry_options',data = data,palette=sns.color_palette('magma
```

Out[44]:
```
<AxesSubplot:xlabel='laundry_options', ylabel='count'>
```

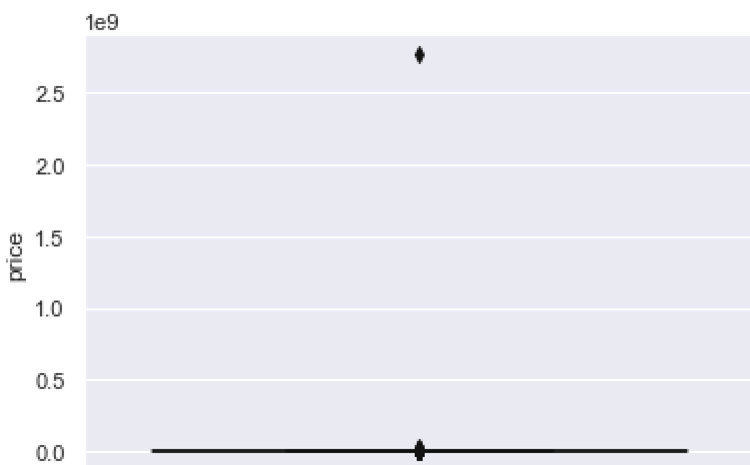


Identifying ouliers in 'Price' column

In [45]:
```
data.price
```

Out[45]:
```
0         1195
1         1120
2          825
3          800
4          785
          ... 
260758     929
260759       0
260760    1069
260761    1507
260762    1001
Name: price, Length: 260763, dtype: int64
```

In [46]:
```
sns.boxplot( y=data["price"],palette=sns.color_palette("magma",10));
plt.show()
```



In [47]:
```
data.price.describe()
```

Out[47]:
```
count    2.607630e+05
mean     1.235957e+04
std      5.421717e+06
min      0.000000e+00
25%      8.170000e+02
50%      1.059000e+03
75%      1.449000e+03
max      2.768307e+09
Name: price, dtype: float64
```

We can't see the plot since the prices are so disparate. Limiting the rent to a maximum of $5000

In [48]:
```
data=data[data['price']<=5000].reset_index(drop=True)
```

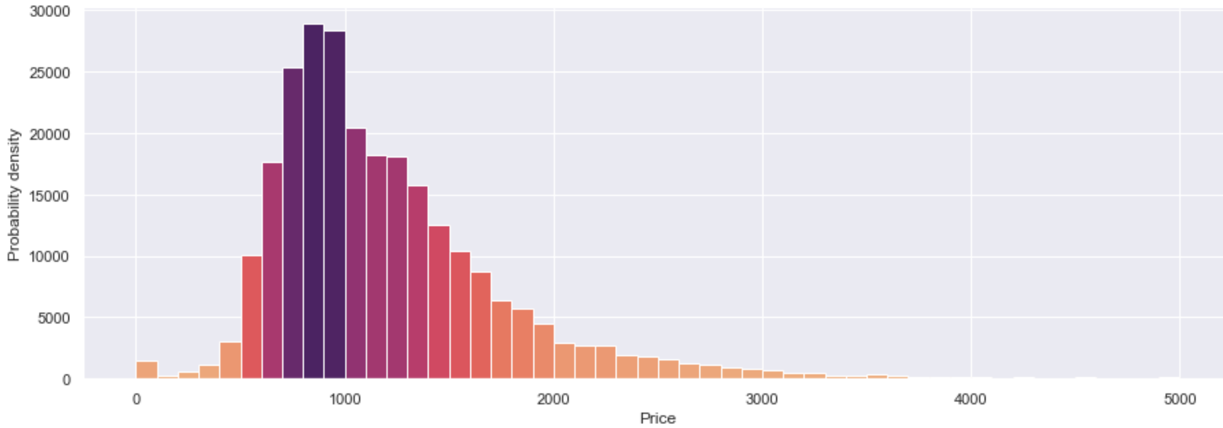
In [49]:
```
sns.boxplot( y=data["price"],palette=sns.color_palette("magma",10));
plt.show()
```

In [50]:
```
plt.figure(figsize=(15,5))
bins = 50

cm = plt.cm.get_cmap('flare')
n, bins, patches = plt.hist(data.price, bins)

col = (n-n.min())/(n.max()-n.min())
for c, p in zip(col, patches):
    plt.setp(p, 'facecolor', cm(c))

plt.xlabel('Price')
plt.ylabel('Probability density')
plt.show()
```

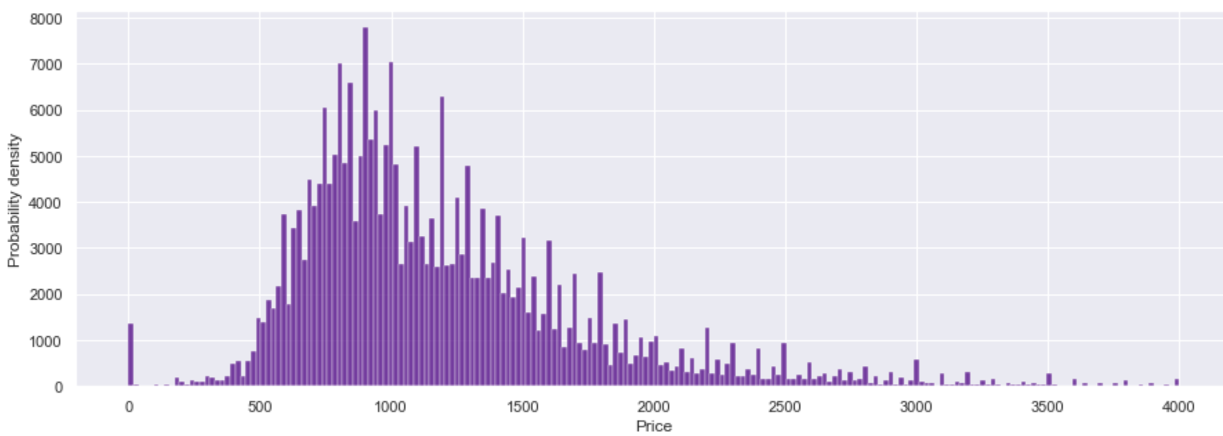


Removing price greater than $4000

In [51]:
```
data=data[data['price']<=4000].reset_index(drop=True)
```

In [52]:
```
plt.figure(figsize=(15,5))
sns.histplot(data, x="price", color='indigo')
plt.xlabel('Price')
plt.ylabel('Probability density')
```

Out[52]:
```
Text(0, 0.5, 'Probability density')
```



• Data Visualization

Visualizing type of apartments on a map

In [53]:
```
import geopandas as gpd
from shapely.geometry import Point, Polygon
from geopandas import GeoDataFrame as gdf
import warnings
```

In [54]:
```
viz=gpd.read_file('s_22mr22.shp')
geometry=[Point(xy) for xy in zip(data['long'], data['lat'])]
crs={'init':'epsg:4326'}
```

In [55]:
```
geo_df=gpd.GeoDataFrame(data,crs=crs,geometry=geometry)
geo_df.head()
```

```
warnings.simplefilter('ignore')
```

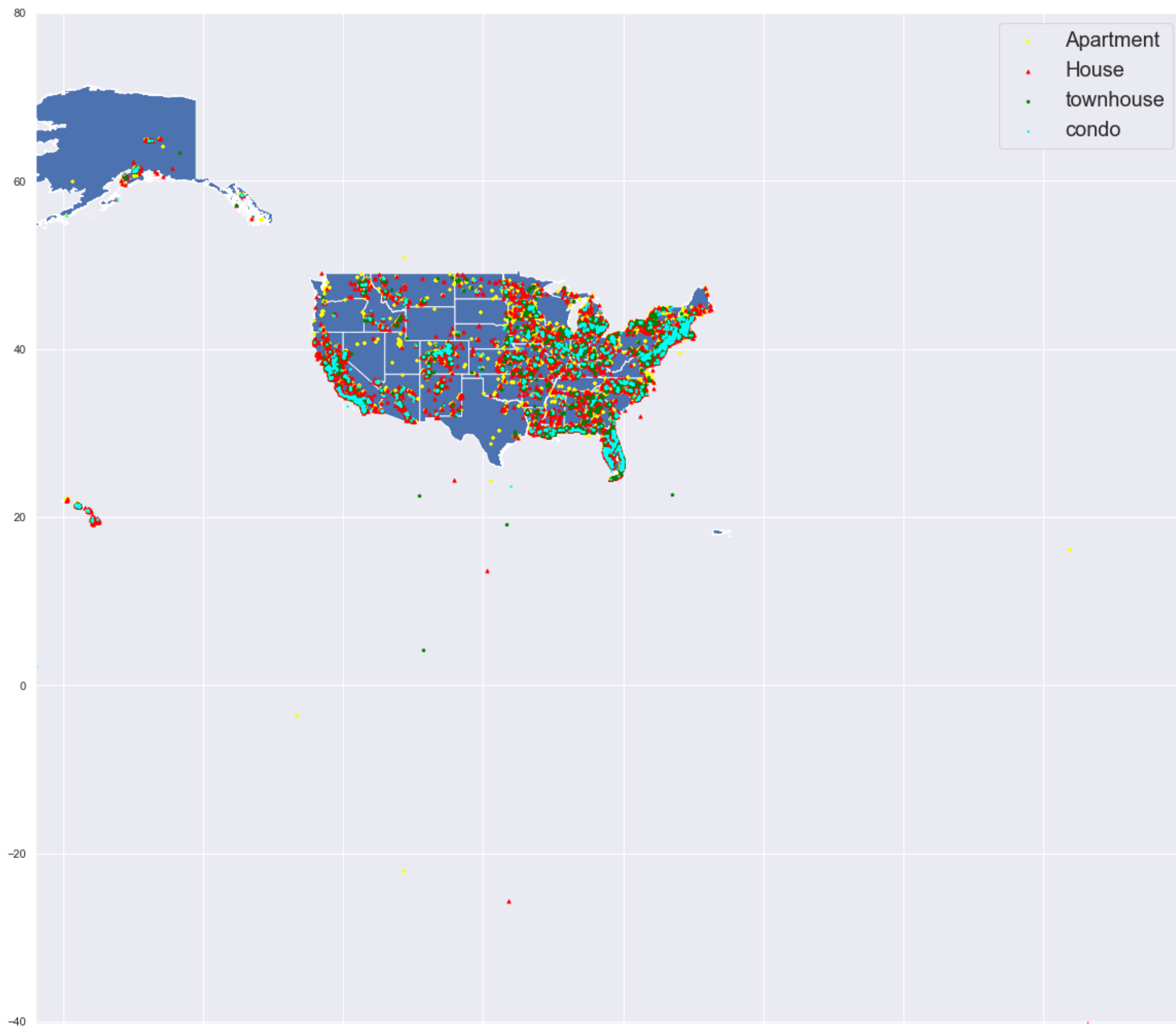
```
C:\Users\spras\anaconda3\envs\geo_env\lib\site-packages\pyproj\crs\crs.py:131:
FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<
code>' is the preferred initialization method. When making the change, be mind
ful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.htm
l#axis-order-changes-in-proj-6
  in_crs_string = _prepare_from_proj_string(in_crs_string)
```

In [56]:

```
fig,ax=plt.subplots(figsize=(20,20))
viz.plot(ax=ax)
geo_df[geo_df['type']=='apartment'].plot(ax=ax,markersize=10,color="yellow",mar
geo_df[geo_df['type']=='house'].plot(ax=ax,markersize=10,color="red",marker="^'
geo_df[geo_df['type']=='townhouse'].plot(ax=ax,markersize=10,color="green",mark
geo_df[geo_df['type']=='condo'].plot(ax=ax,markersize=10,color="cyan",marker=".
minx, miny, maxx, maxy = geo_df.total_bounds
ax.set_xlim(minx, 0)
ax.set_ylim(miny, 80)
plt.legend(prop={'size':20})
```

Out[56]:

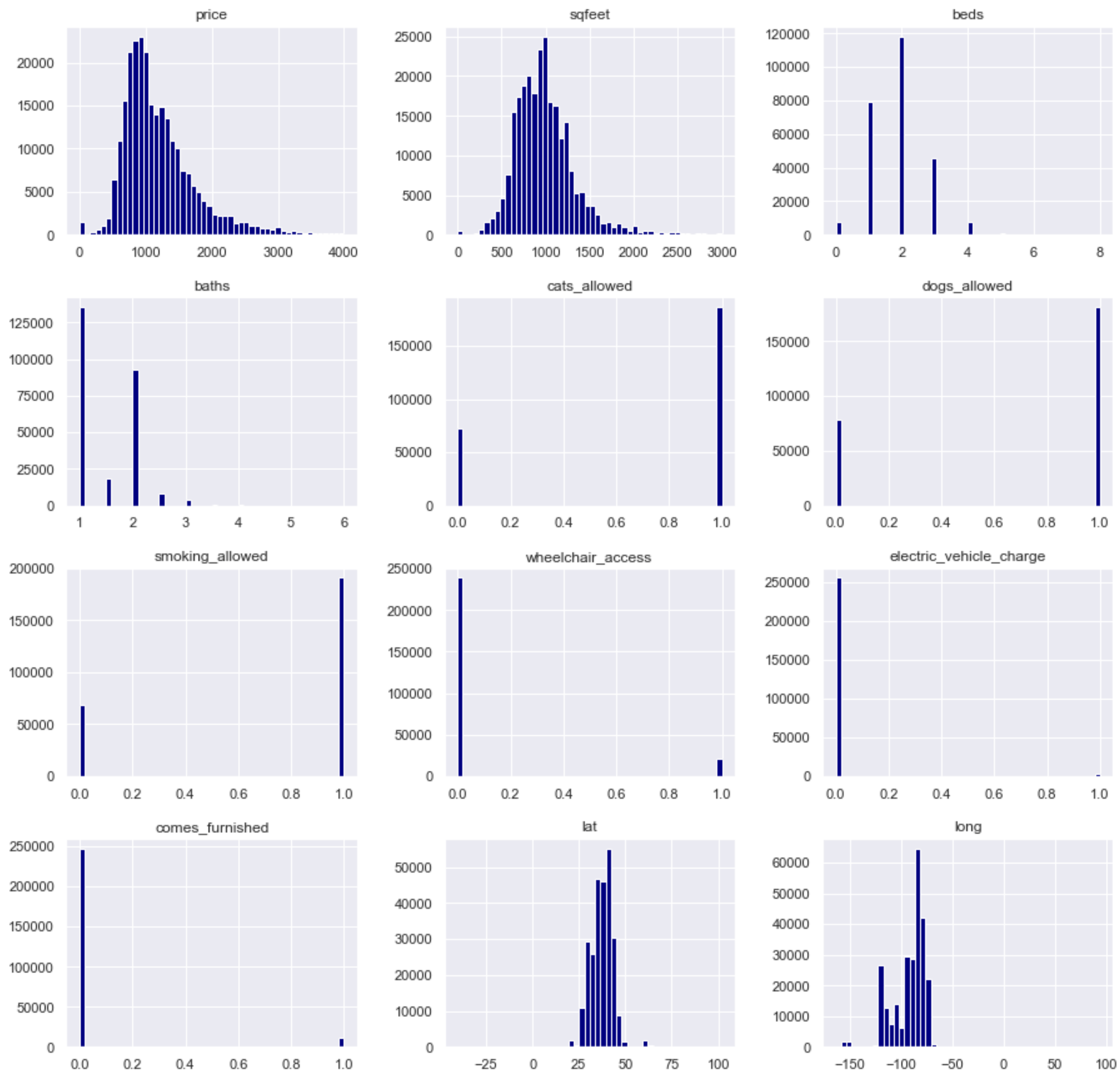
```
<matplotlib.legend.Legend at 0x1ffccefb4c0>
```



In [57]:

```
data.hist(bins = 50, figsize = (15,15),color="navy")
```

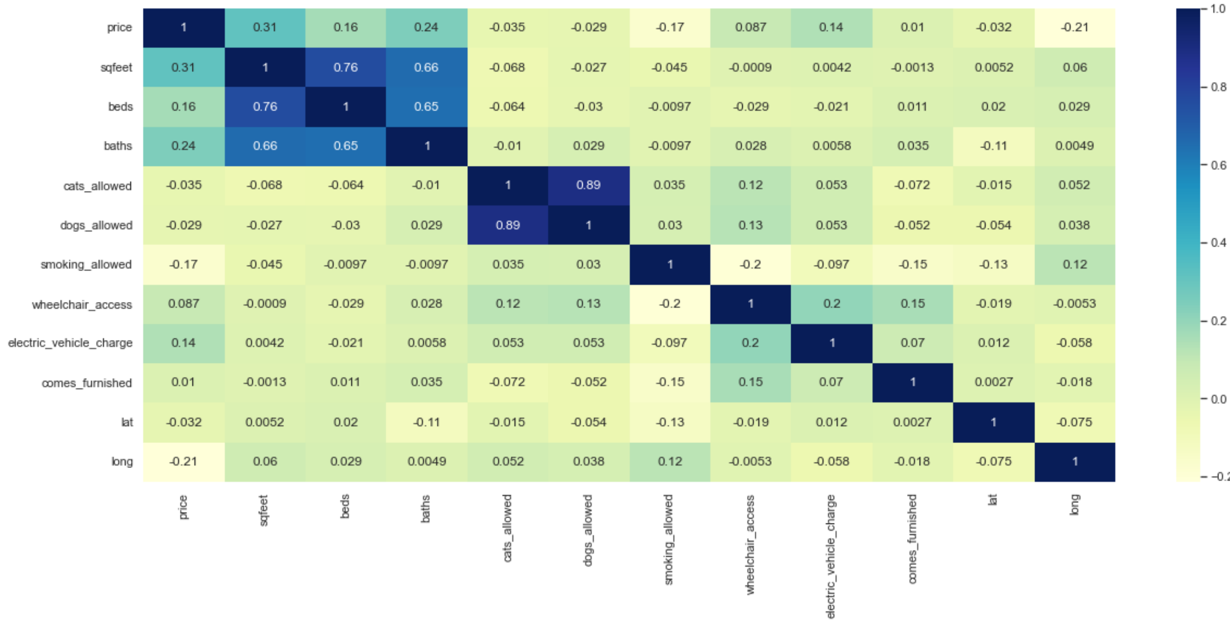
Out[57]:
```
array([[<AxesSubplot:title={'center':'price'}>,
        <AxesSubplot:title={'center':'sqfeet'}>,
        <AxesSubplot:title={'center':'beds'}>],
       [<AxesSubplot:title={'center':'baths'}>,
        <AxesSubplot:title={'center':'cats_allowed'}>,
        <AxesSubplot:title={'center':'dogs_allowed'}>],
       [<AxesSubplot:title={'center':'smoking_allowed'}>,
        <AxesSubplot:title={'center':'wheelchair_access'}>,
        <AxesSubplot:title={'center':'electric_vehicle_charge'}>],
       [<AxesSubplot:title={'center':'comes_furnished'}>,
        <AxesSubplot:title={'center':'lat'}>,
        <AxesSubplot:title={'center':'long'}>]], dtype=object)
```



Heatmap to visualize co-relation between columns

In [58]:
```
corr=data.corr(method='pearson')
plt.figure(figsize=(20, 8))
sns.heatmap(corr,cmap="YlGnBu",annot=True)
```

Out[58]:
```
<AxesSubplot:>
```



In [59]:
```python
plt.figure(figsize=(15,10))
plt.title('How does sqfeet vary with type of houses and number of beds')
sns.scatterplot(data=data, x="sqfeet", y="beds",hue="type",palette="tab20b")
```

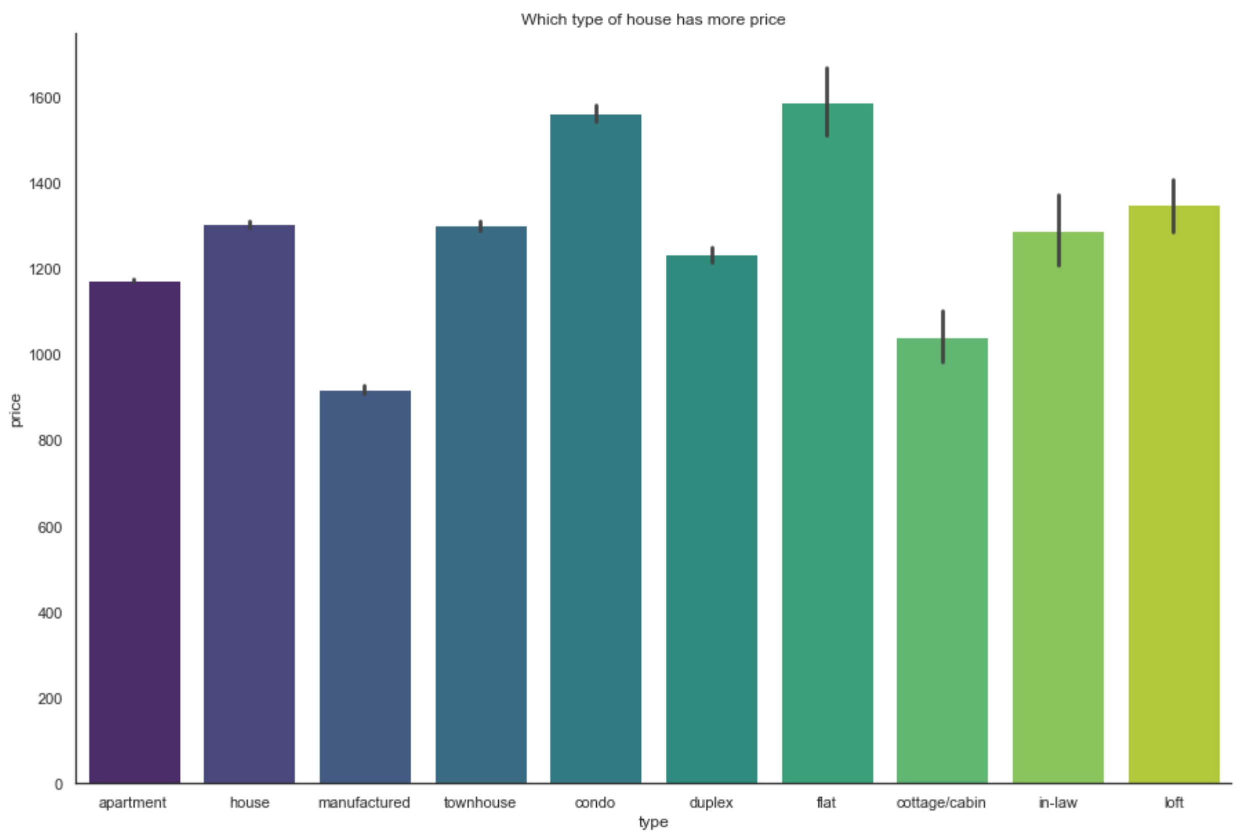
Out[59]:
```
<AxesSubplot:title={'center':'How does sqfeet vary with type of houses and num
ber of beds'}, xlabel='sqfeet', ylabel='beds'>
```

In [60]:
```python
plt.figure(figsize=(15,5))
plt.scatter(data.price, data.sqfeet,color="teal")
```

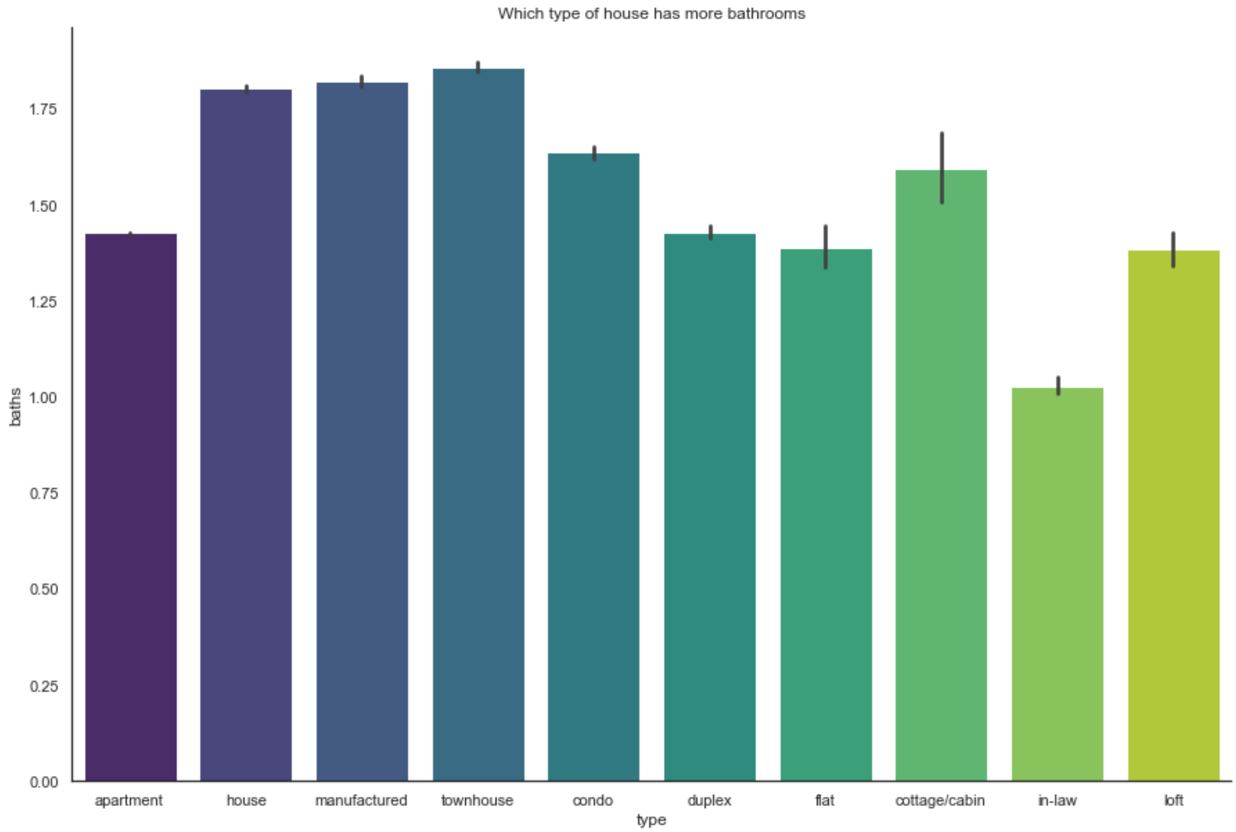
Out[60]:
```
<matplotlib.collections.PathCollection at 0x1ff8aeb59c0>
```

In [61]:
```
sns.set_style("white")

fig, ax = plt.subplots(figsize=(15, 10))
sns.barplot(data["type"], data["price"],ax=ax,palette="viridis")
plt.title('Which type of house has more price')
sns.despine()
```

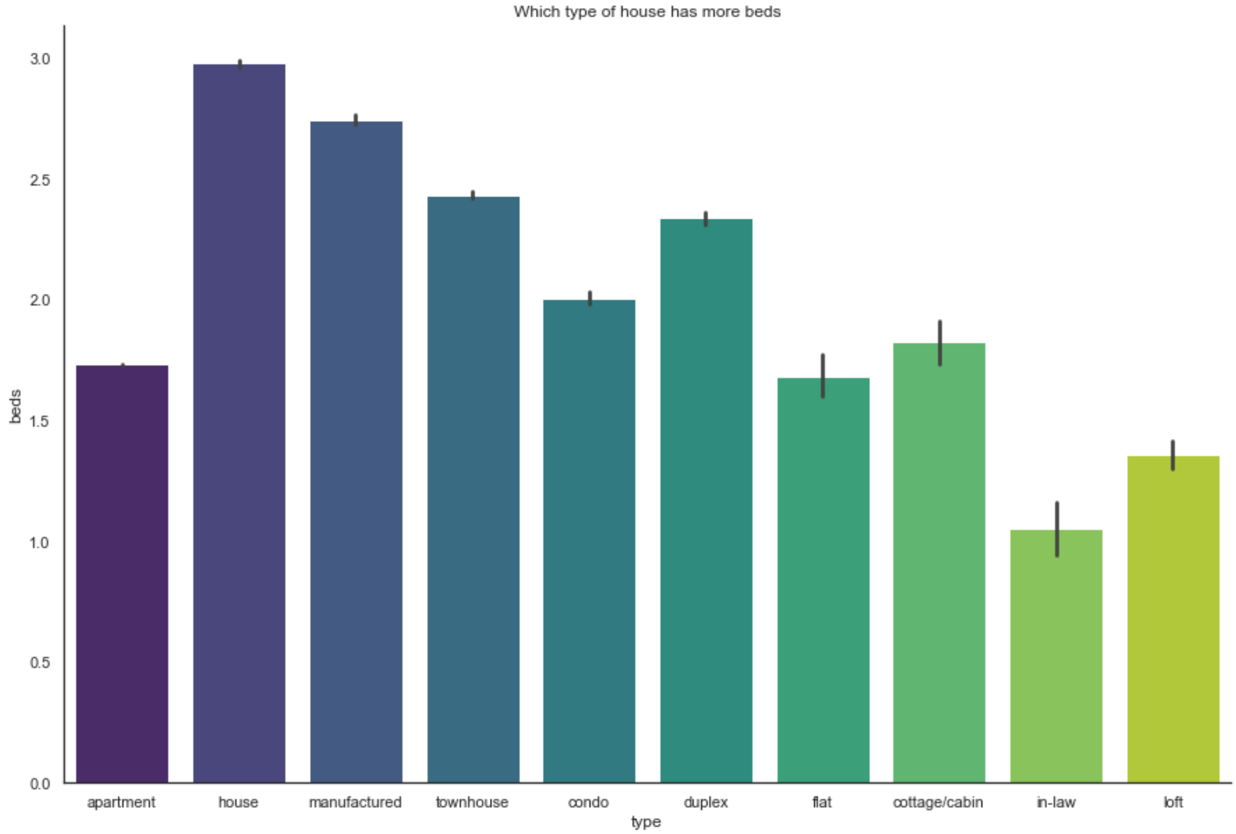


In [62]:
```
sns.set_style("white")
fig, ax = plt.subplots(figsize=(15, 10))
sns.barplot(data["type"], data["baths"],ax=ax,palette="viridis")
plt.title('Which type of house has more bathrooms')
sns.despine()
```



In [63]:
```
sns.set_style("white")
fig, ax = plt.subplots(figsize=(15, 10))
sns.barplot(data["type"], data["beds"],ax=ax, palette="viridis")
sns.despine()
plt.title('Which type of house has more beds')
```

Out[63]:
```
Text(0.5, 1.0, 'Which type of house has more beds')
```



- # Data Processing

In [64]:
```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
```

In [65]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 259319 entries, 0 to 259318
Data columns (total 18 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   region                   259319 non-null  object
 1   price                    259319 non-null  int64
 2   type                     259319 non-null  object
 3   sqfeet                   259319 non-null  int64
 4   beds                     259319 non-null  int64
 5   baths                    259319 non-null  float64
 6   cats_allowed             259319 non-null  int64
 7   dogs_allowed             259319 non-null  int64
 8   smoking_allowed          259319 non-null  int64
 9   wheelchair_access        259319 non-null  int64
 10  electric_vehicle_charge  259319 non-null  int64
 11  comes_furnished          259319 non-null  int64
 12  laundry_options          259319 non-null  object
 13  parking_options          259319 non-null  object
 14  lat                      259319 non-null  float64
 15  long                     259319 non-null  float64
 16  state                    259319 non-null  object
 17  geometry                 259319 non-null  geometry
dtypes: float64(3), geometry(1), int64(9), object(5)
memory usage: 35.6+ MB
```

In [66]:
```
data["region"]=label.fit_transform(data["region"])
data["type"]=label.fit_transform(data["type"])
data["laundry_options"]=label.fit_transform(data["laundry_options"])
data["parking_options"]=label.fit_transform(data["parking_options"])
data["state"]=label.fit_transform(data["state"])
```

In [67]:
```
data.head()
```

Out[67]:

	region	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wh
0	21	1195	0	1908	3	2.0	1	1	1	
1	21	1120	0	1319	3	2.0	1	1	1	
2	21	825	0	1133	1	1.5	1	1	1	
3	21	800	0	927	1	1.0	1	1	1	
4	21	785	0	1047	2	1.0	1	1	1	

All of our data is transformed to a numerical format.

In [68]:
```
data.drop(columns = ['geometry'], inplace = True)
```

Because dogs allowed and cats allowed have such a strong co-relation, so can remove one and rename the field to pets allowed.

In [69]:
```
data.drop(columns = ['cats_allowed'], inplace = True)
data.rename(columns = {'dogs_allowed' : 'pets_allowed'}, inplace = True)
data.head()
```

Out[69]:

	region	price	type	sqfeet	beds	baths	pets_allowed	smoking_allowed	wheelchair_access
0	21	1195	0	1908	3	2.0	1	1	0
1	21	1120	0	1319	3	2.0	1	1	0
2	21	825	0	1133	1	1.5	1	1	0
3	21	800	0	927	1	1.0	1	1	0
4	21	785	0	1047	2	1.0	1	1	0

In [70]:
```
data.describe().T
```

Out[70]:

	count	mean	std	min	25%	50%
region	259319.0	139.562651	88.461359	0.0000	59.0000	139.0000
price	259319.0	1194.645518	557.423583	0.0000	815.0000	1053.0000
type	259319.0	0.954770	2.324722	0.0000	0.0000	0.0000
sqfeet	259319.0	985.955264	351.330701	0.0000	750.0000	950.0000
beds	259319.0	1.887582	0.868298	0.0000	1.0000	2.0000
baths	259319.0	1.483763	0.565016	1.0000	1.0000	1.0000
pets_allowed	259319.0	0.698059	0.459101	0.0000	0.0000	1.0000
smoking_allowed	259319.0	0.734940	0.441366	0.0000	0.0000	1.0000
wheelchair_access	259319.0	0.078872	0.269539	0.0000	0.0000	0.0000
electric_vehicle_charge	259319.0	0.014025	0.117595	0.0000	0.0000	0.0000
comes_furnished	259319.0	0.046599	0.210779	0.0000	0.0000	0.0000
laundry_options	259319.0	2.920804	1.447557	0.0000	1.0000	4.0000
parking_options	259319.0	3.206846	1.484986	0.0000	2.0000	4.0000
lat	259319.0	37.219989	5.659076	-40.2666	33.5059	37.9863
long	259319.0	-92.298855	17.300744	-163.8940	-103.6240	-86.4231
state	259319.0	15.402901	10.001982	0.0000	7.0000	14.0000

- # Data Prediction and Modeling

In [71]:
```
X = data.drop(columns = ['price'])
y = data.price
```

In [72]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.20,random_
```

In [73]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn import neighbors
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.ensemble import GradientBoostingRegressor
```

1.Random Forest Regressor

In [74]:
```
acc = np.array([])
randomForest = RandomForestRegressor()
randomForest.fit(X_train,y_train)
randomForest.predict(X_test)
x=randomForest.score(X_test, y_test)
print(x)
acc = np.append(acc, x)
```

```
0.8966836560555288
```

2.Decision Tree Regressor

In [75]:
```
decisionTree = DecisionTreeRegressor()
decisionTree.fit(X_train,y_train)
decisionTree.predict(X_test)
x=decisionTree.score(X_test, y_test)
print(x)
acc = np.append(acc, x)
```

```
0.816347966733835
```

3.Linear Regression

In [76]:
```
linearRegressor = LinearRegression()
linearRegressor.fit(X_train,y_train)
linearRegressor.predict(X_test)
x=linearRegressor.score(X_test, y_test)
print(x)
acc = np.append(acc, x)
```

```
0.24567602753126283
```

4.KNN Regressor

with n_neighbors=1

In [77]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 1)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
```

```
0.7362307181733981
```

with n_neighbors=2

In [78]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 2)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
acc = np.append(acc, x)
```

```
0.765593306793707
```

with n_neighbors=4

In [79]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 4)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
```

```
0.7644116263364554
```

with n_neighbors=5

In [80]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 5)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
```

```
0.7585553796697284
```

with n_neighbors=6

In [81]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 6)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
```

```
0.7527925099926892
```

with n_neighbors=8

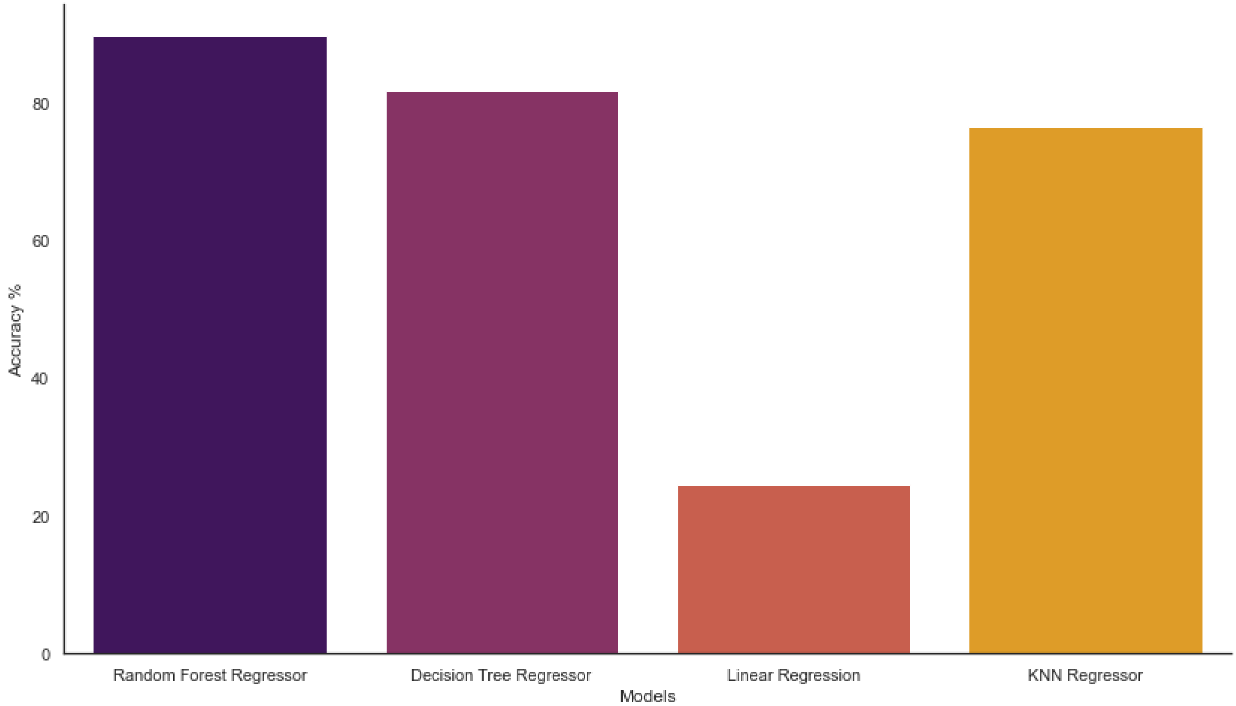
In [82]:
```
KNN = neighbors.KNeighborsRegressor(n_neighbors = 8)
KNN.fit(X_train,y_train)
KNN.predict(X_test)
x=KNN.score(X_test, y_test)
print(x)
```

```
0.7410984855693326
```

Model Accuracies

In [83]:
```
models = ['Random Forest Regressor', 'Decision Tree Regressor', 'Linear Regress
accuracy = acc
fig, ax = plt.subplots(figsize=(14, 8))
sns.barplot(models, accuracy*100,ax=ax, palette="inferno")
plt.xlabel('Models')
```

```
plt.ylabel('Accuracy %')
sns.despine()
```



- # Conclusion

We've completed all the essential data processing and calculations, and Random Forest Regressor has the greatest accuracy score for Rent Prediction of all the algorithms employed.