

```
In [1]: # Capstone FLT Project
```

Section 1

```
In [2]: # Import Libraries and Clean Data + Basic EDA
```

```
In [168... import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [4]: df = pd.read_csv("AB_NYC_2019.csv")
```

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

Out[5]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
#   Column              Non-Null Count  Dtype
---  -
0   id                   48895 non-null  int64
1   name                 48879 non-null  object
2   host_id              48895 non-null  int64
3   host_name            48874 non-null  object
4   neighbourhood_group  48895 non-null  object
5   neighbourhood         48895 non-null  object
6   latitude              48895 non-null  float64
7   longitude             48895 non-null  float64
8   room_type            48895 non-null  object
9   price                48895 non-null  int64
10  minimum_nights       48895 non-null  int64
11  number_of_reviews    48895 non-null  int64
12  last_review          38843 non-null  object
13  reviews_per_month    38843 non-null  float64
14  calculated_host_listings_count  48895 non-null  int64
15  availability_365      48895 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

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

Out[7]:

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000

In [8]: df.isna().any()

Out[8]: id False
name True
host_id False
host_name True
neighbourhood_group False
neighbourhood False
latitude False
longitude False
room_type False
price False
minimum_nights False
number_of_reviews False
last_review True
reviews_per_month True
calculated_host_listings_count False
availability_365 False
dtype: bool

In [9]: df = df.dropna()

In [10]: df.isna().any()

Out[10]: id False
name False
host_id False
host_name False
neighbourhood_group False
neighbourhood False
latitude False
longitude False
room_type False
price False
minimum_nights False
number_of_reviews False
last_review False
reviews_per_month False
calculated_host_listings_count False
availability_365 False
dtype: bool

In [11]: df.head()

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_ni
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.74767	-73.97500	Entire home/apt	200	

In [12]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38821 entries, 0 to 48852
Data columns (total 16 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     38821 non-null  int64
1   name                                 38821 non-null  object
2   host_id                             38821 non-null  int64
3   host_name                           38821 non-null  object
4   neighbourhood_group                 38821 non-null  object
5   neighbourhood                       38821 non-null  object
6   latitude                           38821 non-null  float64
7   longitude                           38821 non-null  float64
8   room_type                           38821 non-null  object
9   price                               38821 non-null  int64
10  minimum_nights                      38821 non-null  int64
11  number_of_reviews                   38821 non-null  int64
12  last_review                         38821 non-null  object
13  reviews_per_month                  38821 non-null  float64
14  calculated_host_listings_count      38821 non-null  int64
15  availability_365                    38821 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 5.0+ MB
```

In [13]: df.describe()

Out[13]:

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month
count	3.882100e+04	3.882100e+04	38821.000000	38821.000000	38821.000000	38821.000000	38821.000000	38821.000000
mean	1.810081e+07	6.424582e+07	40.728129	-73.951149	142.332526	5.869220	29.290255	1.373225
std	1.069372e+07	7.589752e+07	0.054991	0.046693	196.994756	17.389026	48.182900	1.680328
min	2.539000e+03	2.438000e+03	40.506410	-74.244420	0.000000	1.000000	1.000000	0.010000
25%	8.721444e+06	7.029525e+06	40.688640	-73.982460	69.000000	1.000000	3.000000	0.190000
50%	1.887286e+07	2.837092e+07	40.721710	-73.954810	101.000000	2.000000	9.000000	0.720000
75%	2.756746e+07	1.018905e+08	40.762990	-73.935020	170.000000	4.000000	33.000000	2.020000
max	3.645581e+07	2.738417e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000

In [14]: df.head()

Out[14]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_ni
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_ni
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.74767	-73.97500	Entire home/apt	200	

```
In [15]: df = df.drop("id", axis =1)
df = df.drop("host_id", axis =1)
df = df.drop("host_name", axis =1)
df = df.drop("name", axis =1)
df = df.drop("latitude", axis =1)
df = df.drop("longitude", axis =1)
df = df.drop("last_review", axis =1)
```

```
In [16]: df.head()
```

```
Out[16]:
```

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host
0	Brooklyn	Kensington	Private room	149	1	9	0.21	
1	Manhattan	Midtown	Entire home/apt	225	1	45	0.38	
3	Brooklyn	Clinton Hill	Entire home/apt	89	1	270	4.64	
4	Manhattan	East Harlem	Entire home/apt	80	10	9	0.10	
5	Manhattan	Murray Hill	Entire home/apt	200	3	74	0.59	

```
In [17]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38821 entries, 0 to 48852
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                ---
0   neighbourhood_group                  38821 non-null  object
1   neighbourhood                        38821 non-null  object
2   room_type                           38821 non-null  object
3   price                               38821 non-null  int64
4   minimum_nights                      38821 non-null  int64
5   number_of_reviews                   38821 non-null  int64
6   reviews_per_month                   38821 non-null  float64
7   calculated_host_listings_count      38821 non-null  int64
8   availability_365                    38821 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 3.0+ MB
```

```
In [18]: df.neighbourhood_group.unique()
```

```
Out[18]: array(['Brooklyn', 'Manhattan', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

```
In [19]: df.loc[df["neighbourhood_group"] == "Brooklyn", "neighbourhood_group"] = "1"
df.loc[df["neighbourhood_group"] == "Manhattan", "neighbourhood_group"] = "2"
df.loc[df["neighbourhood_group"] == "Queens", "neighbourhood_group"] = "3"
df.loc[df["neighbourhood_group"] == "Staten Island", "neighbourhood_group"] = "4"
df.loc[df["neighbourhood_group"] == "Bronx", "neighbourhood_group"] = "5"
```

```
In [20]: df.room_type.unique()
```

```
Out[20]: array(['Private room', 'Entire home/apt', 'Shared room'], dtype=object)
```

```
In [21]: df.loc[df["room_type"] == "Private room", "room_type"] = "1"
df.loc[df["room_type"] == "Entire home/apt", "room_type"] = "2"
df.loc[df["room_type"] == "Shared room", "room_type"] = "3"
```

```
In [22]: df.head()
```

```
Out[22]:
```

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_
0	1	Kensington	1	149	1	9	0.21	
1	2	Midtown	2	225	1	45	0.38	
3	1	Clinton Hill	2	89	1	270	4.64	
4	2	East Harlem	2	80	10	9	0.10	
5	2	Murray Hill	2	200	3	74	0.59	

```
In [23]: df.neighbourhood.unique()
```

```
Out[23]: array(['Kensington', 'Midtown', 'Clinton Hill', 'East Harlem',
'Murray Hill', 'Bedford-Stuyvesant', 'Hell's Kitchen',
'Upper West Side', 'Chinatown', 'South Slope', 'West Village',
'Williamsburg', 'Fort Greene', 'Chelsea', 'Crown Heights',
'Park Slope', 'Windsor Terrace', 'Inwood', 'East Village',
'Harlem', 'Greenpoint', 'Bushwick', 'Lower East Side',
'Prospect-Lefferts Gardens', 'Long Island City', 'Kips Bay',
'SoHo', 'Upper East Side', 'Prospect Heights',
'Washington Heights', 'Woodside', 'Flatbush', 'Brooklyn Heights',
'Carroll Gardens', 'Gowanus', 'Flatlands', 'Cobble Hill',
'Flushing', 'Boerum Hill', 'Sunnyside', 'DUMBO', 'St. George',
'Highbridge', 'Financial District', 'Ridgewood',
'Morningside Heights', 'Jamaica', 'Middle Village', 'NoHo',
'Ditmars Steinway', 'Flatiron District', 'Roosevelt Island',
'Greenwich Village', 'Little Italy', 'East Flatbush',
'Tompkinsville', 'Astoria', 'Eastchester', 'Kingsbridge',
'Two Bridges', 'Rockaway Beach', 'Forest Hills', 'Nolita',
'Woodlawn', 'University Heights', 'Gramercy', 'Allerton',
'East New York', 'Theater District', 'Concourse Village',
'Sheepshead Bay', 'Emerson Hill', 'Fort Hamilton', 'Bensonhurst',
'Tribeca', 'Shore Acres', 'Sunset Park', 'Concourse', 'Elmhurst',
'Brighton Beach', 'Jackson Heights', 'Cypress Hills', 'St. Albans',
'Arrochar', 'Rego Park', 'Wakefield', 'Clifton', 'Bay Ridge',
'Graniteville', 'Spuyten Duyvil', 'Stapleton', 'Briarwood',
'Ozone Park', 'Columbia St', 'Vinegar Hill', 'Mott Haven',
'Longwood', 'Canarsie', 'Battery Park City', 'Civic Center',
'East Elmhurst', 'New Springville', 'Morris Heights', 'Arverne',
'Gravesend', 'Tottenville', 'Mariners Harbor', 'Concord',
'Borough Park', 'Bayside', 'Downtown Brooklyn', 'Port Morris',
'Fieldston', 'Kew Gardens', 'Midwood', 'College Point',
'Mount Eden', 'City Island', 'Glendale', 'Red Hook',
'Richmond Hill', 'Queens Village', 'Maspeth', 'Port Richmond',
'Williamsbridge', 'Soundview', 'Woodhaven', 'Co-op City',
'Stuyvesant Town', 'Parkchester', 'North Riverdale',
'Dyker Heights', 'Bronxdale', 'Sea Gate', 'Riverdale',
'Kew Gardens Hills', 'Bay Terrace', 'Norwood', 'Claremont Village',
'Whitestone', 'Fordham', 'Bayswater', 'Navy Yard', 'Brownsville',
'Eltingville', 'Mount Hope', 'Clason Point', 'Lighthouse Hill',
'Springfield Gardens', 'Howard Beach', 'Belle Harbor',
'Jamaica Estates', 'Van Nest', 'Bellerose', 'Fresh Meadows',
'Morris Park', 'West Brighton', 'Far Rockaway', 'South Ozone Park',
'Tremont', 'Corona', 'Great Kills', 'Manhattan Beach',
'Marble Hill', 'Dongan Hills', 'East Morrisania', 'Hunts Point',
'Neponsit', 'Pelham Bay', 'Randall Manor', 'Throgs Neck',
'Todt Hill', 'West Farms', 'Silver Lake', 'Laurelton',
'Grymes Hill', 'Holliswood', 'Pelham Gardens', 'Rosedale',
'Castleton Corners', 'Edgemere', 'New Brighton', 'Baychester',
'Melrose', 'Bergen Beach', 'Cambria Heights', 'Richmondtown',
'Howland Hook', 'Schuylerville', 'Coney Island', 'Prince's Bay',
'South Beach', 'Bath Beach', 'Midland Beach', 'Jamaica Hills',
'Oakwood', 'Castle Hill', 'Douglaston', 'Huguenot', 'Edenwald',
'Belmont', 'Grant City', 'Westerleigh', 'Morrisania',
'Bay Terrace, Staten Island', 'Westchester Square', 'Little Neck',
'Rosebank', 'Unionport', 'Mill Basin', 'Hollis', 'Arden Heights',
```

```
"Bull's Head", 'Olinville', 'Rossville', 'Breezy Point',
'Willowbrook', 'New Dorp Beach'], dtype=object)
```

```
In [24]: df.loc[df["neighbourhood"] == "Kensington", "neighbourhood"] = "1"
df.loc[df["neighbourhood"] == "Midtown", "neighbourhood"] = "2"
df.loc[df["neighbourhood"] == "Clinton Hill", "neighbourhood"] = "3"
df.loc[df["neighbourhood"] == "Murray Hill", "neighbourhood"] = "4"
df.loc[df["neighbourhood"] == "East Harlem", "neighbourhood"] = "5"
df.loc[df["neighbourhood"] == "Bedford-Stuyvesant", "neighbourhood"] = "6"
df.loc[df["neighbourhood"] == "Hell's Kitchen", "neighbourhood"] = "7"
df.loc[df["neighbourhood"] == "Upper West Side", "neighbourhood"] = "8"
df.loc[df["neighbourhood"] == "Chinatown", "neighbourhood"] = "9"
df.loc[df["neighbourhood"] == "South Slope", "neighbourhood"] = "10"
```

```
In [25]: df.loc[df["neighbourhood"] == "West Village", "neighbourhood"] = "11"
df.loc[df["neighbourhood"] == "Williamsburg", "neighbourhood"] = "12"
df.loc[df["neighbourhood"] == "Fort Greene", "neighbourhood"] = "13"
df.loc[df["neighbourhood"] == "Chelsea", "neighbourhood"] = "14"
df.loc[df["neighbourhood"] == "Crown Heights", "neighbourhood"] = "15"
df.loc[df["neighbourhood"] == "Park Slope", "neighbourhood"] = "16"
df.loc[df["neighbourhood"] == "Windsor Terrace", "neighbourhood"] = "17"
df.loc[df["neighbourhood"] == "Inwood", "neighbourhood"] = "18"
df.loc[df["neighbourhood"] == "East Village", "neighbourhood"] = "19"
df.loc[df["neighbourhood"] == "Harlem", "neighbourhood"] = "20"
```

```
In [26]: df.loc[df["neighbourhood"] == "Greenpoint", "neighbourhood"] = "21"
df.loc[df["neighbourhood"] == "Bushwick", "neighbourhood"] = "22"
df.loc[df["neighbourhood"] == "Lower East Side", "neighbourhood"] = "23"
df.loc[df["neighbourhood"] == "Prospect-Lefferts Gardens", "neighbourhood"] = "24"
df.loc[df["neighbourhood"] == "Long Island City", "neighbourhood"] = "25"
df.loc[df["neighbourhood"] == "Kips Bay", "neighbourhood"] = "26"
df.loc[df["neighbourhood"] == "SoHo", "neighbourhood"] = "27"
df.loc[df["neighbourhood"] == "Upper East Side", "neighbourhood"] = "28"
df.loc[df["neighbourhood"] == "Prospect Heights", "neighbourhood"] = "29"
df.loc[df["neighbourhood"] == "Washington Heights", "neighbourhood"] = "30"
```

```
In [27]: df.loc[df["neighbourhood"] == "Woodside", "neighbourhood"] = "31"
df.loc[df["neighbourhood"] == "Flatbush", "neighbourhood"] = "32"
df.loc[df["neighbourhood"] == "Brooklyn Heights", "neighbourhood"] = "33"
df.loc[df["neighbourhood"] == "Carroll Gardens", "neighbourhood"] = "34"
df.loc[df["neighbourhood"] == "Gowanus", "neighbourhood"] = "35"
df.loc[df["neighbourhood"] == "Flatlands", "neighbourhood"] = "36"
df.loc[df["neighbourhood"] == "Cobble Hill", "neighbourhood"] = "37"
df.loc[df["neighbourhood"] == "Flushing", "neighbourhood"] = "38"
df.loc[df["neighbourhood"] == "Boerum Hill", "neighbourhood"] = "39"
df.loc[df["neighbourhood"] == "Sunnyside", "neighbourhood"] = "40"
```

```
In [28]: df.loc[df["neighbourhood"] == "DUMBO", "neighbourhood"] = "41"
df.loc[df["neighbourhood"] == "St. George", "neighbourhood"] = "42"
df.loc[df["neighbourhood"] == "Highbridge", "neighbourhood"] = "43"
df.loc[df["neighbourhood"] == "Financial District", "neighbourhood"] = "44"
df.loc[df["neighbourhood"] == "Ridgewood", "neighbourhood"] = "45"
df.loc[df["neighbourhood"] == "Morningside Heights", "neighbourhood"] = "46"
df.loc[df["neighbourhood"] == "Jamaica", "neighbourhood"] = "47"
df.loc[df["neighbourhood"] == "Middle Village", "neighbourhood"] = "48"
df.loc[df["neighbourhood"] == "NoHo", "neighbourhood"] = "49"
df.loc[df["neighbourhood"] == "Ditmars Steinway", "neighbourhood"] = "50"
```

```
In [29]: df.loc[df["neighbourhood"] == "Flatiron District", "neighbourhood"] = "51"
df.loc[df["neighbourhood"] == "Roosevelt Island", "neighbourhood"] = "52"
df.loc[df["neighbourhood"] == "Greenwich Village", "neighbourhood"] = "53"
df.loc[df["neighbourhood"] == "Little Italy", "neighbourhood"] = "54"
df.loc[df["neighbourhood"] == "East Flatbush", "neighbourhood"] = "55"
df.loc[df["neighbourhood"] == "Tompkinsville", "neighbourhood"] = "56"
df.loc[df["neighbourhood"] == "Astoria", "neighbourhood"] = "57"
df.loc[df["neighbourhood"] == "Eastchester", "neighbourhood"] = "58"
df.loc[df["neighbourhood"] == "Kingsbridge", "neighbourhood"] = "59"
df.loc[df["neighbourhood"] == "Two Bridges", "neighbourhood"] = "60"
```

```
In [30]: df.loc[df["neighbourhood"] == "Rockaway Beach", "neighbourhood"] = "61"
df.loc[df["neighbourhood"] == "Forest Hills", "neighbourhood"] = "62"
df.loc[df["neighbourhood"] == "Nolita", "neighbourhood"] = "63"
df.loc[df["neighbourhood"] == "Woodlawn", "neighbourhood"] = "64"
```

```
df.loc[df["neighbourhood"] == "University Heights", "neighbourhood"] = "65"
df.loc[df["neighbourhood"] == "Gramercy", "neighbourhood"] = "66"
df.loc[df["neighbourhood"] == "Allerton", "neighbourhood"] = "67"
df.loc[df["neighbourhood"] == "East New York", "neighbourhood"] = "68"
df.loc[df["neighbourhood"] == "Theater District", "neighbourhood"] = "69"
df.loc[df["neighbourhood"] == "Concourse Village", "neighbourhood"] = "70"
```

```
In [31]: df.loc[df["neighbourhood"] == "Sheepshead Bay", "neighbourhood"] = "71"
df.loc[df["neighbourhood"] == "Emerson Hill", "neighbourhood"] = "72"
df.loc[df["neighbourhood"] == "Fort Hamilton", "neighbourhood"] = "73"
df.loc[df["neighbourhood"] == "Bensonhurst", "neighbourhood"] = "74"
df.loc[df["neighbourhood"] == "Tribeca", "neighbourhood"] = "75"
df.loc[df["neighbourhood"] == "Shore Acres", "neighbourhood"] = "76"
df.loc[df["neighbourhood"] == "Sunset Park", "neighbourhood"] = "77"
df.loc[df["neighbourhood"] == "Concourse", "neighbourhood"] = "78"
df.loc[df["neighbourhood"] == "Elmhurst", "neighbourhood"] = "79"
df.loc[df["neighbourhood"] == "Brighton Beach", "neighbourhood"] = "80"
```

```
In [32]: df.loc[df["neighbourhood"] == "Jackson Heights", "neighbourhood"] = "81"
df.loc[df["neighbourhood"] == "Cypress Hills", "neighbourhood"] = "82"
df.loc[df["neighbourhood"] == "St. Albans", "neighbourhood"] = "83"
df.loc[df["neighbourhood"] == "Arrochar", "neighbourhood"] = "84"
df.loc[df["neighbourhood"] == "Rego Park", "neighbourhood"] = "85"
df.loc[df["neighbourhood"] == "Wakefield", "neighbourhood"] = "86"
df.loc[df["neighbourhood"] == "Clifton", "neighbourhood"] = "87"
df.loc[df["neighbourhood"] == "Bay Ridge", "neighbourhood"] = "88"
df.loc[df["neighbourhood"] == "Graniteville", "neighbourhood"] = "89"
df.loc[df["neighbourhood"] == "Spuyten Duyvil", "neighbourhood"] = "90"
```

```
In [33]: df.loc[df["neighbourhood"] == "Stapleton", "neighbourhood"] = "91"
df.loc[df["neighbourhood"] == "Briarwood", "neighbourhood"] = "92"
df.loc[df["neighbourhood"] == "Ozone Park", "neighbourhood"] = "93"
df.loc[df["neighbourhood"] == "Columbia St", "neighbourhood"] = "94"
df.loc[df["neighbourhood"] == "Vinegar Hill", "neighbourhood"] = "95"
df.loc[df["neighbourhood"] == "Mott Haven", "neighbourhood"] = "96"
df.loc[df["neighbourhood"] == "Longwood", "neighbourhood"] = "97"
df.loc[df["neighbourhood"] == "Canarsie", "neighbourhood"] = "98"
df.loc[df["neighbourhood"] == "Battery Park City", "neighbourhood"] = "99"
df.loc[df["neighbourhood"] == "Civic Center", "neighbourhood"] = "100"
```

```
In [34]: df.loc[df["neighbourhood"] == "East Elmhurst", "neighbourhood"] = "101"
df.loc[df["neighbourhood"] == "New Springville", "neighbourhood"] = "102"
df.loc[df["neighbourhood"] == "Morris Heights", "neighbourhood"] = "103"
df.loc[df["neighbourhood"] == "Arverne", "neighbourhood"] = "104"
df.loc[df["neighbourhood"] == "Gravesend", "neighbourhood"] = "105"
df.loc[df["neighbourhood"] == "Tottenville", "neighbourhood"] = "106"
df.loc[df["neighbourhood"] == "Mariners Harbor", "neighbourhood"] = "107"
df.loc[df["neighbourhood"] == "Concord", "neighbourhood"] = "108"
df.loc[df["neighbourhood"] == "Borough Park", "neighbourhood"] = "109"
df.loc[df["neighbourhood"] == "Bayside", "neighbourhood"] = "110"
```

```
In [35]: df.loc[df["neighbourhood"] == "Downtown Brooklyn", "neighbourhood"] = "111"
df.loc[df["neighbourhood"] == "Port Morris", "neighbourhood"] = "112"
df.loc[df["neighbourhood"] == "Fieldston", "neighbourhood"] = "113"
df.loc[df["neighbourhood"] == "Kew Gardens", "neighbourhood"] = "114"
df.loc[df["neighbourhood"] == "Midwood", "neighbourhood"] = "115"
df.loc[df["neighbourhood"] == "College Point", "neighbourhood"] = "116"
df.loc[df["neighbourhood"] == "Mount Eden", "neighbourhood"] = "117"
df.loc[df["neighbourhood"] == "City Island", "neighbourhood"] = "118"
df.loc[df["neighbourhood"] == "Glendale", "neighbourhood"] = "119"
df.loc[df["neighbourhood"] == "Red Hook", "neighbourhood"] = "120"
```

```
In [36]: df.loc[df["neighbourhood"] == "Richmond Hill", "neighbourhood"] = "121"
df.loc[df["neighbourhood"] == "Queens Village", "neighbourhood"] = "122"
df.loc[df["neighbourhood"] == "Maspeth", "neighbourhood"] = "123"
df.loc[df["neighbourhood"] == "Port Richmond", "neighbourhood"] = "124"
df.loc[df["neighbourhood"] == "Williamsbridge", "neighbourhood"] = "125"
df.loc[df["neighbourhood"] == "Soundview", "neighbourhood"] = "126"
df.loc[df["neighbourhood"] == "Woodhaven", "neighbourhood"] = "127"
df.loc[df["neighbourhood"] == "Co-op City", "neighbourhood"] = "128"
df.loc[df["neighbourhood"] == "Stuyvesant Town", "neighbourhood"] = "129"
df.loc[df["neighbourhood"] == "Parkchester", "neighbourhood"] = "130"
```



```
In [37]: df.loc[df["neighbourhood"] == "North Riverdale", "neighbourhood"] = "131"
df.loc[df["neighbourhood"] == "Dyker Heights", "neighbourhood"] = "132"
df.loc[df["neighbourhood"] == "Bronxdale", "neighbourhood"] = "133"
df.loc[df["neighbourhood"] == "Sea Gate", "neighbourhood"] = "134"
df.loc[df["neighbourhood"] == "Riverdale", "neighbourhood"] = "135"
df.loc[df["neighbourhood"] == "Kew Gardens Hills", "neighbourhood"] = "136"
df.loc[df["neighbourhood"] == "Bay Terrace", "neighbourhood"] = "137"
df.loc[df["neighbourhood"] == "Norwood", "neighbourhood"] = "138"
df.loc[df["neighbourhood"] == "Claremont Village", "neighbourhood"] = "139"
df.loc[df["neighbourhood"] == "Whitestone", "neighbourhood"] = "140"
```

```
In [38]: df.loc[df["neighbourhood"] == "Fordham", "neighbourhood"] = "141"
df.loc[df["neighbourhood"] == "Bayswater", "neighbourhood"] = "142"
df.loc[df["neighbourhood"] == "Navy Yard", "neighbourhood"] = "143"
df.loc[df["neighbourhood"] == "Brownsville", "neighbourhood"] = "144"
df.loc[df["neighbourhood"] == "Eltingville", "neighbourhood"] = "145"
df.loc[df["neighbourhood"] == "Mount Hope", "neighbourhood"] = "146"
df.loc[df["neighbourhood"] == "Clason Point", "neighbourhood"] = "147"
df.loc[df["neighbourhood"] == "Lighthouse Hill", "neighbourhood"] = "148"
df.loc[df["neighbourhood"] == "Springfield Gardens", "neighbourhood"] = "149"
df.loc[df["neighbourhood"] == "Howard Beach", "neighbourhood"] = "150"
```

```
In [39]: df.loc[df["neighbourhood"] == "Belle Harbor", "neighbourhood"] = "151"
df.loc[df["neighbourhood"] == "Jamaica Estates", "neighbourhood"] = "152"
df.loc[df["neighbourhood"] == "Van Nest", "neighbourhood"] = "153"
df.loc[df["neighbourhood"] == "Bellerose", "neighbourhood"] = "154"
df.loc[df["neighbourhood"] == "Fresh Meadows", "neighbourhood"] = "155"
df.loc[df["neighbourhood"] == "Morris Park", "neighbourhood"] = "156"
df.loc[df["neighbourhood"] == "West Brighton", "neighbourhood"] = "157"
df.loc[df["neighbourhood"] == "Far Rockaway", "neighbourhood"] = "158"
df.loc[df["neighbourhood"] == "South Ozone Park", "neighbourhood"] = "159"
df.loc[df["neighbourhood"] == "Tremont", "neighbourhood"] = "160"
```

```
In [40]: df.loc[df["neighbourhood"] == "Corona", "neighbourhood"] = "161"
df.loc[df["neighbourhood"] == "Great Kills", "neighbourhood"] = "162"
df.loc[df["neighbourhood"] == "Manhattan Beach", "neighbourhood"] = "163"
df.loc[df["neighbourhood"] == "Marble Hill", "neighbourhood"] = "164"
df.loc[df["neighbourhood"] == "Dongan Hills", "neighbourhood"] = "165"
df.loc[df["neighbourhood"] == "East Morrisania", "neighbourhood"] = "166"
df.loc[df["neighbourhood"] == "Hunts Point", "neighbourhood"] = "167"
df.loc[df["neighbourhood"] == "Neponsit", "neighbourhood"] = "168"
df.loc[df["neighbourhood"] == "Pelham Bay", "neighbourhood"] = "169"
df.loc[df["neighbourhood"] == "Randall Manor", "neighbourhood"] = "170"
```

```
In [41]: df.loc[df["neighbourhood"] == "Throgs Neck", "neighbourhood"] = "171"
df.loc[df["neighbourhood"] == "Todt Hill", "neighbourhood"] = "172"
df.loc[df["neighbourhood"] == "West Farms", "neighbourhood"] = "173"
df.loc[df["neighbourhood"] == "Silver Lake", "neighbourhood"] = "174"
df.loc[df["neighbourhood"] == "Laurelton", "neighbourhood"] = "175"
df.loc[df["neighbourhood"] == "Grymes Hill", "neighbourhood"] = "176"
df.loc[df["neighbourhood"] == "Holliswood", "neighbourhood"] = "177"
df.loc[df["neighbourhood"] == "Pelham Gardens", "neighbourhood"] = "178"
df.loc[df["neighbourhood"] == "Rosedale", "neighbourhood"] = "179"
df.loc[df["neighbourhood"] == "Melrose", "neighbourhood"] = "180"
```

```
In [42]: df.loc[df["neighbourhood"] == "Bergen Beach", "neighbourhood"] = "181"
df.loc[df["neighbourhood"] == "Cambria Heights", "neighbourhood"] = "182"
df.loc[df["neighbourhood"] == "Richmondton", "neighbourhood"] = "183"
df.loc[df["neighbourhood"] == "Howland Hook", "neighbourhood"] = "184"
df.loc[df["neighbourhood"] == "Schuylerville", "neighbourhood"] = "185"
df.loc[df["neighbourhood"] == "Coney Island", "neighbourhood"] = "186"
df.loc[df["neighbourhood"] == "Prince's Bay", "neighbourhood"] = "187"
df.loc[df["neighbourhood"] == "South Beach", "neighbourhood"] = "188"
df.loc[df["neighbourhood"] == "Bath Beach", "neighbourhood"] = "189"
df.loc[df["neighbourhood"] == "Midland Beach", "neighbourhood"] = "190"
```

```
In [43]: df.loc[df["neighbourhood"] == "Jamaica Hills", "neighbourhood"] = "191"
df.loc[df["neighbourhood"] == "Oakwood", "neighbourhood"] = "192"
df.loc[df["neighbourhood"] == "Castle Hill", "neighbourhood"] = "193"
df.loc[df["neighbourhood"] == "Douglaston", "neighbourhood"] = "194"
df.loc[df["neighbourhood"] == "Huguenot", "neighbourhood"] = "195"
df.loc[df["neighbourhood"] == "Edenwald", "neighbourhood"] = "196"
df.loc[df["neighbourhood"] == "Belmont", "neighbourhood"] = "197"
```



```
df.loc[df["neighbourhood"] == "Grant City", "neighbourhood"] = "198"
df.loc[df["neighbourhood"] == "Westerleigh", "neighbourhood"] = "199"
df.loc[df["neighbourhood"] == "Morrisania", "neighbourhood"] = "200"
```

```
In [44]: df.loc[df["neighbourhood"] == "Bay Terrace, Staten Island", "neighbourhood"] = "201"
df.loc[df["neighbourhood"] == "Westchester Square", "neighbourhood"] = "202"
df.loc[df["neighbourhood"] == "Little Neck", "neighbourhood"] = "203"
df.loc[df["neighbourhood"] == "Rosebank", "neighbourhood"] = "204"
df.loc[df["neighbourhood"] == "Unionport", "neighbourhood"] = "205"
df.loc[df["neighbourhood"] == "Mill Basin", "neighbourhood"] = "206"
df.loc[df["neighbourhood"] == "Hollis", "neighbourhood"] = "207"
df.loc[df["neighbourhood"] == "Arden Heights", "neighbourhood"] = "208"
df.loc[df["neighbourhood"] == "Bull's Head", "neighbourhood"] = "209"
df.loc[df["neighbourhood"] == "Olinville", "neighbourhood"] = "210"
```

```
In [45]: df.loc[df["neighbourhood"] == "Rossville", "neighbourhood"] = "211"
df.loc[df["neighbourhood"] == "Breezy Point", "neighbourhood"] = "212"
df.loc[df["neighbourhood"] == "Willowbrook", "neighbourhood"] = "213"
df.loc[df["neighbourhood"] == "New Dorp Beach", "neighbourhood"] = "214"
df.loc[df["neighbourhood"] == "Castleton Corners", "neighbourhood"] = "215"
df.loc[df["neighbourhood"] == "Edgemere", "neighbourhood"] = "216"
df.loc[df["neighbourhood"] == "New Brighton", "neighbourhood"] = "217"
df.loc[df["neighbourhood"] == "Baychester", "neighbourhood"] = "218"
```

```
In [46]: df.head(20)
```

```
Out[46]:
```

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host
0	1	1	1	149	1	9	0.21	
1	2	2	2	225	1	45	0.38	
3	1	3	2	89	1	270	4.64	
4	2	5	2	80	10	9	0.10	
5	2	4	2	200	3	74	0.59	
6	1	6	1	60	45	49	0.40	
7	2	7	1	79	2	430	3.47	
8	2	8	1	79	2	118	0.99	
9	2	9	2	150	1	160	1.33	
10	2	8	2	135	5	53	0.43	
11	2	7	1	85	2	188	1.50	
12	1	10	1	89	4	167	1.34	
13	2	8	1	85	2	113	0.91	
14	2	11	2	120	90	27	0.22	
15	1	12	2	140	2	148	1.20	
16	1	13	2	215	2	198	1.72	
17	2	14	1	140	1	260	2.12	
18	1	15	2	99	3	53	4.44	
20	1	12	2	299	3	9	0.07	
21	1	16	1	130	2	130	1.09	

```
In [47]: df.neighbourhood_group.unique()
```

```
Out[47]: array(['1', '2', '3', '4', '5'], dtype=object)
```

```
In [48]: df.neighbourhood.unique()
```

```
Out[48]: array(['1', '2', '3', '5', '4', '6', '7', '8', '9', '10', '11', '12',
                '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23',
                '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34',
```

```
'35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45',
'46', '47', '48', '49', '50', '51', '52', '53', '54', '55', '56',
'57', '58', '59', '60', '61', '62', '63', '64', '65', '66', '67',
'68', '69', '70', '71', '72', '73', '74', '75', '76', '77', '78',
'79', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89',
'90', '91', '92', '93', '94', '95', '96', '97', '98', '99', '100',
'101', '102', '103', '104', '105', '106', '107', '108', '109',
'110', '111', '112', '113', '114', '115', '116', '117', '118',
'119', '120', '121', '122', '123', '124', '125', '126', '127',
'128', '129', '130', '131', '132', '133', '134', '135', '136',
'137', '138', '139', '140', '141', '142', '143', '144', '145',
'146', '147', '148', '149', '150', '151', '152', '153', '154',
'155', '156', '157', '158', '159', '160', '161', '162', '163',
'164', '165', '166', '167', '168', '169', '170', '171', '172',
'173', '174', '175', '176', '177', '178', '179', '215', '216',
'217', '218', '180', '181', '182', '183', '184', '185', '186',
'187', '188', '189', '190', '191', '192', '193', '194', '195',
'196', '197', '198', '199', '200', '201', '202', '203', '204',
'205', '206', '207', '208', '209', '210', '211', '212', '213',
'214'], dtype=object)
```

```
In [49]: df = df.astype({'neighbourhood_group': int, 'neighbourhood': float, 'room_type': float, 'price': float, 'minimum_nights': float, 'number_of_reviews': float, 'reviews_per_month': float, 'calculated_host_listings_count': float, 'availability_365': float})
```

```
In [50]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38821 entries, 0 to 48852
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   neighbourhood_group                   38821 non-null  int32
1   neighbourhood                        38821 non-null  float64
2   room_type                           38821 non-null  float64
3   price                               38821 non-null  float64
4   minimum_nights                      38821 non-null  float64
5   number_of_reviews                   38821 non-null  float64
6   reviews_per_month                  38821 non-null  float64
7   calculated_host_listings_count      38821 non-null  float64
8   availability_365                    38821 non-null  float64
dtypes: float64(8), int32(1)
memory usage: 2.8 MB
```

```
In [51]: df = df.drop(['calculated_host_listings_count', 'availability_365', 'minimum_nights'], axis=1)
```

```
In [52]: df.head()
```

```
Out[52]:
```

	neighbourhood_group	neighbourhood	room_type	price	number_of_reviews	reviews_per_month
0	1	1.0	1.0	149.0	9.0	0.21
1	2	2.0	2.0	225.0	45.0	0.38
3	1	3.0	2.0	89.0	270.0	4.64
4	2	5.0	2.0	80.0	9.0	0.10
5	2	4.0	2.0	200.0	74.0	0.59

```
In [53]: df.describe()
```

```
Out[53]:
```

	neighbourhood_group	neighbourhood	room_type	price	number_of_reviews	reviews_per_month
count	38821.000000	38821.000000	38821.000000	38821.000000	38821.000000	38821.000000
mean	1.778110	31.344633	1.567038	142.332526	29.290255	1.373229
std	0.852771	34.231016	0.537678	196.994756	48.182900	1.680328
min	1.000000	1.000000	1.000000	0.000000	1.000000	0.010000
25%	1.000000	11.000000	1.000000	69.000000	3.000000	0.190000
50%	2.000000	20.000000	2.000000	101.000000	9.000000	0.720000
75%	2.000000	38.000000	2.000000	170.000000	33.000000	2.020000
max	5.000000	218.000000	3.000000	10000.000000	629.000000	58.500000

In [54]: `# Data Cleaning Complete`

Section 2

In [55]: `# Creation of ALL Models`

MLR

In [56]: `from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm`

In [57]: `X = df.drop(columns = ["price"], axis =1)
y = df["price"]`

In [58]: `X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)`

In [59]: `LR = LinearRegression()`

In [60]: `LR.fit(X_train, y_train)`

Out[60]: `LinearRegression()`

In [61]: `y_pred_LR= LR.predict(X_test)`

In [62]: `LR_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_LR})
LR_diff.head()`

Out[62]:

	Actual value	Predicted value
5792	225.0	198.247324
43334	150.0	131.433455
30638	105.0	101.399811
26976	67.0	138.673313
24171	150.0	189.743681

In [63]: `meanAbErr = metrics.mean_absolute_error(y_test, y_pred_LR)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_LR)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_LR))`

In [64]: `print('R squared: {:.2f}'.format(LR.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)`

R squared: 6.14
Mean Absolute Error: 65.76500421880998
Mean Square Error: 36915.32169292407
Root Mean Square Error: 192.13360375770833

In [65]: `X_train_LR = sm.add_constant(X_train)
LR_1 = sm.OLS(y_train, X_train_LR.astype(float)).fit()
LR_1.summary()`

Out[65]:

OLS Regression Results			
Dep. Variable:	price	R-squared:	0.063
Model:	OLS	Adj. R-squared:	0.062
Method:	Least Squares	F-statistic:	388.3
Date:	Sun, 06 Feb 2022	Prob (F-statistic):	0.00
Time:	11:36:36	Log-Likelihood:	-1.9414e+05

No. Observations:		29115		AIC:		3.883e+05	
Df Residuals:		29109		BIC:		3.884e+05	
Df Model:		5					
Covariance Type:		nonrobust					
		coef	std err	t	P> t	[0.025	0.975]
	const	12.8702	4.273	3.012	0.003	4.495	21.246
neighbourhood_group		8.3634	1.527	5.478	0.000	5.371	11.356
neighbourhood		-0.4903	0.038	-12.918	0.000	-0.565	-0.416
room_type		85.3737	2.081	41.031	0.000	81.295	89.452
number_of_reviews		-0.1127	0.028	-4.043	0.000	-0.167	-0.058
reviews_per_month		0.0594	0.806	0.074	0.941	-1.520	1.639
Omnibus:	69236.897	Durbin-Watson:				2.008	
Prob(Omnibus):	0.000	Jarque-Bera (JB):		1092976520.131			
Skew:	24.120	Prob(JB):				0.00	
Kurtosis:	950.962	Cond. No.				253.	

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [66]: vif = pd.DataFrame()

In [67]: vif['Features'] = X_train.columns

In [68]: vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]

In [69]: vif['VIF'] = round(vif['VIF'], 2)

In [70]: vif = vif.sort_values(by = "VIF", ascending = False)

In [71]: vif
```

Out[71]:

	Features	VIF
0	neighbourhood_group	5.58
2	room_type	3.79
1	neighbourhood	2.49
4	reviews_per_month	2.40
3	number_of_reviews	1.95

KNN

```
In [72]: from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import MinMaxScaler

In [73]: X = df.drop(columns = ["price"], axis =1)
y = df["price"]

In [74]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
In [75]: Reg = KNeighborsRegressor()
```

```
In [76]: Reg.fit(X_train, y_train)
```

```
Out[76]: KNeighborsRegressor()
```

```
In [77]: rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(X_train, y_train) #fit the model
    pred=model.predict(X_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ', K , 'is:', error)
```

```
RMSE value for k= 1 is: 246.77138848400793
RMSE value for k= 2 is: 218.95824267150468
RMSE value for k= 3 is: 207.95939967322516
RMSE value for k= 4 is: 203.3985736301778
RMSE value for k= 5 is: 201.7147394047799
RMSE value for k= 6 is: 201.431439039874
RMSE value for k= 7 is: 200.5757502938718
RMSE value for k= 8 is: 198.93962015824027
RMSE value for k= 9 is: 198.62885414879366
RMSE value for k= 10 is: 197.3234811173989
RMSE value for k= 11 is: 197.49115250325906
RMSE value for k= 12 is: 196.81821984047053
RMSE value for k= 13 is: 196.41501255252342
RMSE value for k= 14 is: 195.9261568134352
RMSE value for k= 15 is: 195.29151546599397
RMSE value for k= 16 is: 194.96287820141777
RMSE value for k= 17 is: 194.75649056654424
RMSE value for k= 18 is: 194.52999610970775
RMSE value for k= 19 is: 194.69160301489276
RMSE value for k= 20 is: 194.65713251873999
```

```
In [78]: from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]}

knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

```
Out[78]: {'n_neighbors': 20}
```

```
In [79]: test_preds = model.predict(X_test)
```

```
In [80]: print(model.score(X_test, y_test))
```

```
0.032959885194633554
```

```
In [81]: meanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
In [82]: print('R squared: {:.2f}'.format(model.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 11.61
Mean Absolute Error: 69.0433906861735
Mean Square Error: 37891.3992404183
Root Mean Square Error: 194.65713251873999
```

DTR

```
In [170]: from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
```

```
In [84]: regressor = DecisionTreeRegressor(random_state = 0)
```

```
In [85]: X = df.drop(columns = ["price"], axis =1)
y = df["price"]
```

```
In [86]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
In [87]: regressor.fit(X_train, y_train)
```

```
Out[87]: DecisionTreeRegressor(random_state=0)
```

```
In [88]: test_preds = regressor.predict(X_test)
```

```
In [89]: print(regressor.score(X_test, y_test))
```

```
-0.8054396867414602
```

```
In [90]: meanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
In [91]: print('R squared: {:.2f}'.format(regressor.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 44.72
Mean Absolute Error: 77.85441288863075
Mean Square Error: 70742.29385880775
Root Mean Square Error: 265.9742353289276
```

```
In [92]: parameters={"splitter":["best","random"],
                    "max_depth" : [1,3,5,7],
                    "min_samples_leaf":[1,2,3,4],
                    "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4],
                    "max_features":["auto","log2","sqrt",None],
                    "max_leaf_nodes":[None,10,20,30,40] }
```

```
In [93]: tuning_model=GridSearchCV(regressor,param_grid=parameters,scoring='neg_mean_squared_error',cv=3,verbose=3)
```

```
In [94]: tuning_model.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 2560 candidates, totalling 7680 fits
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=best, score=-13616.488, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=best, score=-43230.951, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=best, score=-50581.540, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=random, score=-13616.488, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=random, score=-43230.951, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, split
ter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1, spli
tter=random, score=-50581.540, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, split
ter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, spli
tter=best, score=-13616.488, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, split
ter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, spli
tter=best, score=-43230.951, total= 0.0s
```

[illegible]

16/401

17/401

18/401

19/401

[illegible]

[illegible]

22/401

[illegible]

[illegible]

25/401

26/401

27/401

[illegible]

[illegible]

30/401

[illegible]

32/401

[illegible]

34/401

35/401

36/401

37/401

38/401

39/401

[illegible]

41/401

42/401

[illegible]

44/401

45/401

46/401

47/401

48/401

49/401

50/401

51/401

52/401

[illegible]

54/401

55/401

[illegible]

57/401

58/401

59/401

[illegible]

61/401

62/401

63/401

64/401

65/401

66/401

67/401

[illegible]

[illegible]

70/401

71/401

72/401

73/401

74/401

[illegible]

[illegible]

77/401

78/401

[illegible]

[illegible]

81/401

82/401

83/401

[illegible]

85/401

86/401

[illegible]

88/401

89/401

90/401

91/401

92/401

93/401

94/401

95/401

96/401

97/401

98/401

99/401

100/401

[illegible]

102/401

103/401

[illegible]

[illegible]

106/401

[illegible]

108/401

109/401

110/401

111/401

112/401

113/401

114/401

115/401

[illegible]

117/401

118/401

119/401

120/401

121/401

122/401

123/401

[illegible]

125/401

126/401

[illegible]

[illegible]

[illegible]

130/401

131/401

[illegible]

133/401

[illegible]

[illegible]

136/401

137/401

138/401

139/401

140/401

[illegible]

142/401

144/401

145/401

146/401

[illegible]

[illegible]

149/401

150/401

151/401

152/401

153/401

154/401

[illegible]

156/401

[illegible]

[illegible]

159/401

160/401

[illegible]

162/401

163/401

164/401

165/401

166/401

[illegible]

[illegible]

[illegible]

170/401

171/401

[illegible]

173/401

174/401

175/401

176/401

[illegible]

178/401

179/401

180/401

181/401

182/401

183/401

184/401

[illegible]

186/401

187/401

188/401

189/401

190/401

191/401

[illegible]

193/401

194/401

195/401

196/401

[illegible]

198/401

[illegible]

[illegible]

201/401

202/401

203/401

[illegible]

205/401

206/401

[illegible]

208/401

209/401

210/401

[illegible]

212/401

213/401

214/401

215/401

216/401

[illegible]

218/401

219/401

220/401

221/401

222/401

[illegible]

[illegible]

225/401

226/401

227/401

228/401

229/401

230/401

[illegible]

[illegible]

233/401

234/401

235/401

236/401

[illegible]

238/401

239/401

[illegible]

241/401

242/401

[illegible]

244/401

245/401

246/401

247/401

[illegible]

[illegible]

250/401

[illegible]

252/401

253/401

254/401

255/401

256/401

257/401

258/401

[illegible]

260/401

[illegible]

262/401

[illegible]

[illegible]

[illegible]

266/401

[illegible]

268/401

[illegible]

[illegible]

271/401

272/401

273/401

274/401

275/401

276/401

277/401

278/401

279/401

[illegible]

281/401

282/401

283/401

[illegible]

[illegible]

286/401

287/401

[illegible]

289/401

290/401

[illegible]

292/401

[illegible]

294/401

[illegible]

296/401

297/401

298/401

299/401

300/401

301/401

302/401

303/401

[illegible]

305/401

306/401

[illegible]

308/401

[illegible]

310/401

311/401

[illegible]

313/401

314/401

315/401

[illegible]

317/401

318/401

[illegible]

[illegible]

[illegible]

322/401

323/401

324/401

[illegible]

326/401

327/401

[illegible]

329/401

330/401

331/401

[illegible]

333/401

334/401

335/401

336/401

337/401

338/401

339/401

[illegible]

341/401

342/401

343/401

[illegible]

345/401

346/401

347/401

348/401

[illegible]

350/401

351/401

[illegible]

353/401

354/401

355/401

356/401

[illegible]

358/401

359/401

[illegible]

361/401

362/401

[illegible]

364/401

[illegible]

366/401

367/401

368/401

369/401

370/401

371/401

372/401

[illegible]

374/401

375/401

376/401

[illegible]

378/401

379/401

380/401

381/401

382/401

383/401

384/401

[illegible]

386/401

387/401

388/401

[illegible]

390/401

[illegible]

[illegible]

```

er=random, score=-50424.585, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best, score=-13482.996, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best, score=-43022.765, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=best, score=-50239.234, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random, score=-13597.159, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random, score=-43221.280, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.2, splitter=random, score=-50547.114, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best, score=-13616.488, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best, score=-43230.951, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=best, score=-50581.540, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random, score=-13616.488, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random, score=-43230.951, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.3, splitter=random, score=-50581.540, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best, score=-13616.488, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best, score=-43230.951, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=best, score=-50581.540, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random, score=-13616.488, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random, score=-43230.951, total= 0.0s
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random
[CV] max_depth=7, max_features=None, max_leaf_nodes=40, min_samples_leaf=4, min_weight_fraction_leaf=0.4, splitter=random, score=-50581.540, total= 0.0s
[Parallel(n_jobs=1)]: Done 7680 out of 7680 | elapsed: 47.5s finished

```

```

Out[94]: GridSearchCV(cv=3, estimator=DecisionTreeRegressor(random_state=0),
                  param_grid={'max_depth': [1, 3, 5, 7],
                              'max_features': ['auto', 'log2', 'sqrt', None],
                              'max_leaf_nodes': [None, 10, 20, 30, 40],
                              'min_samples_leaf': [1, 2, 3, 4],
                              'min_weight_fraction_leaf': [0.1, 0.2, 0.3, 0.4]},

```

```
        'splitter': ['best', 'random']},  
        scoring='neg_mean_squared_error', verbose=3)
```

```
In [95]: tuning_model.best_params_
```

```
Out[95]: {'max_depth': 3,  
         'max_features': 'auto',  
         'max_leaf_nodes': None,  
         'min_samples_leaf': 1,  
         'min_weight_fraction_leaf': 0.1,  
         'splitter': 'best'}
```

```
In [96]: tuned_model = DecisionTreeRegressor(max_depth=5,max_features='auto',max_leaf_nodes=None,min_samples_leaf=1,min_we
```

```
In [97]: tuned_model.fit(X_train, y_train)
```

```
Out[97]: DecisionTreeRegressor(max_depth=5, max_features='auto',  
                               min_weight_fraction_leaf=0.1)
```

```
In [98]: tuned_pred=tuned_model.predict(X_test)
```

```
In [99]: print(tuned_model.score(X_test, y_test))
```

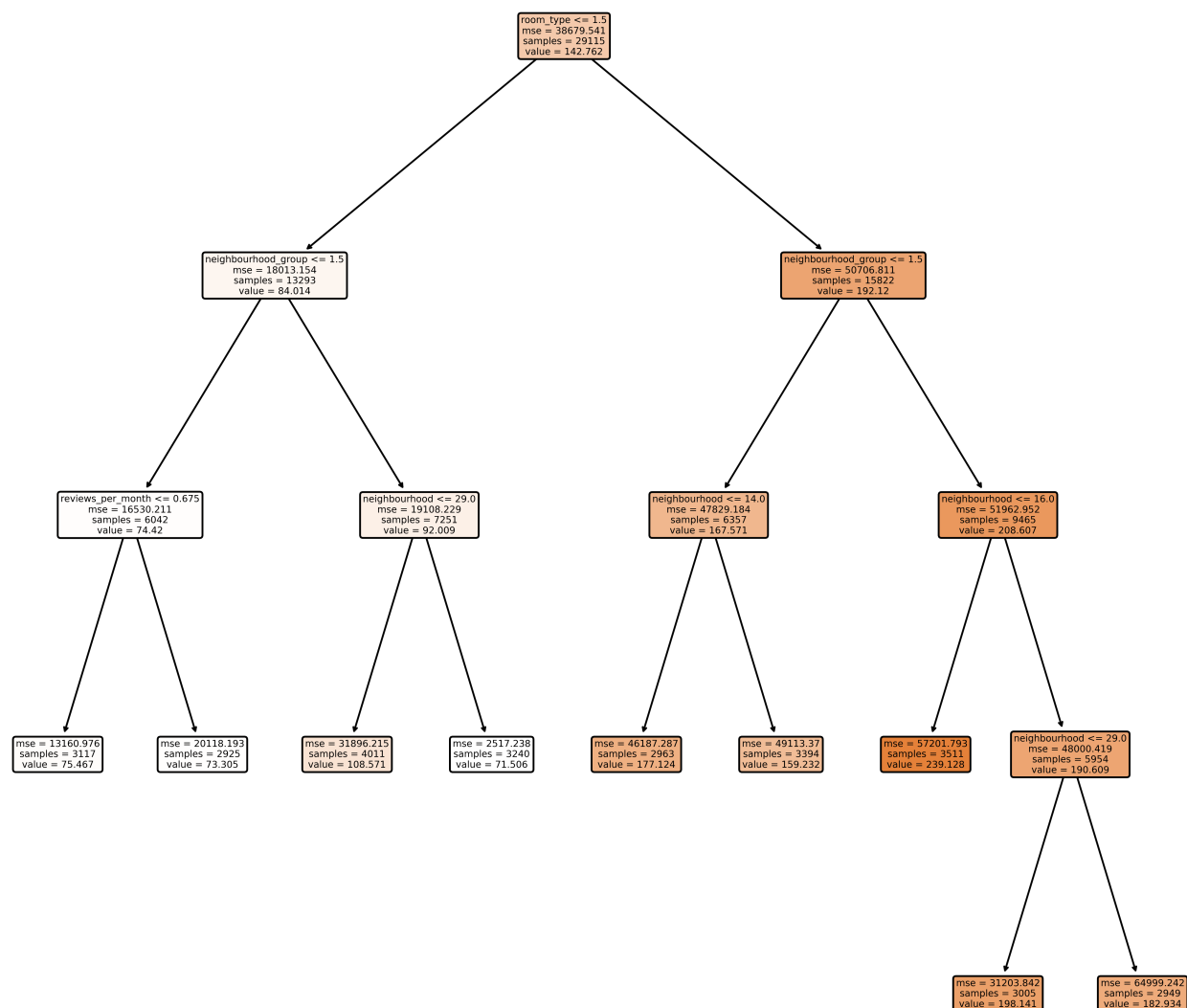
```
0.0809398109436571
```

```
In [100... meanAbErr = metrics.mean_absolute_error(y_test, tuned_pred)  
meanSqErr = metrics.mean_squared_error(y_test, tuned_pred)  
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, tuned_pred))
```

```
In [101... print('R squared: {:.2f}'.format(tuned_model.score(X,y)*100))  
print('Mean Absolute Error:', meanAbErr)  
print('Mean Square Error:', meanSqErr)  
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 8.71  
Mean Absolute Error: 62.65934621355715  
Mean Square Error: 36011.40843729863  
Root Mean Square Error: 189.76672110066778
```

```
In [171... plt.figure(figsize=(12,12), dpi=500)  
tree.plot_tree(tuned_model,  
               feature_names=X.columns,  
               class_names=np.unique(y).astype('str'),  
               filled=True, rounded=True)  
  
plt.show()
```



RTR

```
In [102... from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
```

```
In [103... n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

max_features = ['auto', 'sqrt']

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

min_samples_split = [2, 5, 10]

min_samples_leaf = [1, 2, 4]

bootstrap = [True, False]
```

```
In [104... random_grid = {'n_estimators': n_estimators,
                  'max_features': max_features,
                  'max_depth': max_depth,
                  'min_samples_split': min_samples_split,
```

```
'min_samples_leaf': min_samples_leaf,
'bootstrap': bootstrap}
```

```
In [105... rf = RandomForestRegressor()
```

```
In [106... X = df.drop(columns = ["price"], axis =1)
y = df["price"]
```

```
In [107... X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
In [108... rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2
```

```
In [109... rf_random.fit(X_train, y_train)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done 9 tasks | elapsed: 23.2s
[Parallel(n_jobs=-1)]: Done 130 tasks | elapsed: 5.3min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 12.4min finished
```

```
Out[109... RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                    n_jobs=-1,
                    param_distributions={'bootstrap': [True, False],
                                         'max_depth': [10, 20, 30, 40, 50, 60,
                                                         70, 80, 90, 100, 110,
                                                         None],
                                         'max_features': ['auto', 'sqrt'],
                                         'min_samples_leaf': [1, 2, 4],
                                         'min_samples_split': [2, 5, 10],
                                         'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                    random_state=42, verbose=2)
```

```
In [110... rf_random.best_params_
```

```
Out[110... {'n_estimators': 1600,
'min_samples_split': 2,
'min_samples_leaf': 4,
'max_features': 'sqrt',
'max_depth': 10,
'bootstrap': True}
```

```
In [111... test_preds = rf_random.predict(X_test)
```

```
In [112... print(rf_random.score(X_test, y_test))
```

0.10302530723004966

```
In [113... meanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
In [114... print('R squared: {:.2f}'.format(rf_random.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

R squared: 19.57
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35146.03548699563
Root Mean Square Error: 187.4727593198426

```
In [115... param_grid = {
    'bootstrap': [True],
    'max_depth': [10, 20, 30, 40],
    'max_features': ["sqrt"],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [1, 2, 3, 4],
    'n_estimators': [100, 200, 300, 400, 500]
}
```

```
In [116... rf = RandomForestRegressor()
```

```
In [176... grid_search_rf = GridSearchCV(estimator = rf, param_grid = param_grid,
```



```
cv = 3, n_jobs = -1, verbose = 2)
```

```
In [177... grid_search_rf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks    | elapsed:   2.0s
[Parallel(n_jobs=-1)]: Done  130 tasks    | elapsed:  23.8s
[Parallel(n_jobs=-1)]: Done  333 tasks    | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  616 tasks    | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  720 out of 720 | elapsed:  3.2min finished
```

```
Out[177... GridSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
              param_grid={'bootstrap': [True], 'max_depth': [10, 20, 30, 40],
                           'max_features': ['sqrt'],
                           'min_samples_leaf': [3, 4, 5],
                           'min_samples_split': [1, 2, 3, 4],
                           'n_estimators': [100, 200, 300, 400, 500]},
              verbose=2)
```

```
In [178... grid_search_rf.best_params_
```

```
Out[178... {'bootstrap': True,
           'max_depth': 10,
           'max_features': 'sqrt',
           'min_samples_leaf': 5,
           'min_samples_split': 4,
           'n_estimators': 200}
```

```
In [179... test_preds = grid_search_rf.predict(X_test)
```

```
In [180... print(grid_search_rf.score(X_test, y_test))
```

```
0.10534836988331808
```

```
In [181... MeanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
In [182... print('R squared: {:.2f}'.format(grid_search_rf.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 18.28
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35055.01124393906
Root Mean Square Error: 187.2298353466644
```

```
In [ ]:
```

XG

```
In [193... from sklearn import preprocessing
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
from xgboost import plot_tree
```

```
In [196... pip install graphviz
```

```
Collecting graphviz
  Downloading graphviz-0.19.1-py3-none-any.whl (46 kB)
Installing collected packages: graphviz
Successfully installed graphviz-0.19.1
Note: you may need to restart the kernel to use updated packages.
```

```
In [125... xgb1 = XGBRegressor()

parameters = {'nthread': [4],
              'objective': ['reg:squarederror'],
              'learning_rate': [.03, 0.05, .07,],
              'max_depth': [4, 5, 6],
              'min_child_weight': [4],
              'silent': [1],
              'subsample': [0.5, 0.7, 0.9],
              'colsample_bytree': [0.5, 0.7, 0.9],
```

```

        'n_estimators': [400]}

xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)

```

Fitting 2 folds for each of 81 candidates, totalling 162 fits

[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.

[Parallel(n_jobs=5)]: Done 40 tasks | elapsed: 11.7s

[Parallel(n_jobs=5)]: Done 162 out of 162 | elapsed: 46.8s finished

[11:54:30] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.

This may not be accurate due to some parameters are only used in language bindings but passed down to XGBoost core. Or some parameters are not used but slip through this verification. Please open an issue if you find above cases.

0.10437773543551332

{'colsample_bytree': 0.5, 'learning_rate': 0.03, 'max_depth': 4, 'min_child_weight': 4, 'n_estimators': 400, 'nth_read': 4, 'objective': 'reg:squarederror', 'silent': 1, 'subsample': 0.7}

In [126... test_preds = xgb_grid.predict(X_test)

In [127... MeanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))

In [128... print('R squared: {:.2f}'.format(xgb_grid.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)

R squared: 15.26
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35147.01263402663
Root Mean Square Error: 187.47536540576905

In []:

In []:

NN

In [129... from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
import talos

In [130... X = df.drop(columns = ["price"], axis =1)
y = df["price"]

In [131... X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

In [132... NN_model = Sequential()

The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = X_train.shape[1], activation='relu'))

The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

```
# Compile the network :
NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.summary()

history = NN_model.fit(X_train,
                        y_train,
                        epochs=40,
                        batch_size=32,
                        validation_data=(X_test, y_test))
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	768
dense_1 (Dense)	(None, 256)	33024
dense_2 (Dense)	(None, 256)	65792
dense_3 (Dense)	(None, 256)	65792
dense_4 (Dense)	(None, 1)	257
Total params: 165,633		
Trainable params: 165,633		
Non-trainable params: 0		

Epoch 1/40
910/910 [=====] - 1s 982us/step - loss: 38821.4375 - mean_squared_error: 38821.4375 - va
l_loss: 37505.3281 - val_mean_squared_error: 37505.3281
Epoch 2/40
910/910 [=====] - 1s 852us/step - loss: 36684.6016 - mean_squared_error: 36684.6016 - va
l_loss: 37302.7812 - val_mean_squared_error: 37302.7812
Epoch 3/40
910/910 [=====] - 1s 879us/step - loss: 36660.6211 - mean_squared_error: 36660.6211 - va
l_loss: 36761.3828 - val_mean_squared_error: 36761.3828
Epoch 4/40
910/910 [=====] - 1s 869us/step - loss: 36359.3477 - mean_squared_error: 36359.3477 - va
l_loss: 36570.2188 - val_mean_squared_error: 36570.2188
Epoch 5/40
910/910 [=====] - 1s 854us/step - loss: 36181.5039 - mean_squared_error: 36181.5039 - va
l_loss: 36641.8359 - val_mean_squared_error: 36641.8359
Epoch 6/40
910/910 [=====] - 1s 868us/step - loss: 36124.6797 - mean_squared_error: 36124.6797 - va
l_loss: 36775.9844 - val_mean_squared_error: 36775.9844
Epoch 7/40
910/910 [=====] - 1s 853us/step - loss: 35968.9883 - mean_squared_error: 35968.9883 - va
l_loss: 36429.4219 - val_mean_squared_error: 36429.4219
Epoch 8/40
910/910 [=====] - 1s 846us/step - loss: 35971.6055 - mean_squared_error: 35971.6055 - va
l_loss: 36687.5781 - val_mean_squared_error: 36687.5781
Epoch 9/40
910/910 [=====] - 1s 878us/step - loss: 35856.8867 - mean_squared_error: 35856.8867 - va
l_loss: 37139.4258 - val_mean_squared_error: 37139.4258
Epoch 10/40
910/910 [=====] - 1s 855us/step - loss: 35872.0508 - mean_squared_error: 35872.0508 - va
l_loss: 36335.6484 - val_mean_squared_error: 36335.6484
Epoch 11/40
910/910 [=====] - 1s 862us/step - loss: 35731.0586 - mean_squared_error: 35731.0586 - va
l_loss: 36346.7891 - val_mean_squared_error: 36346.7891
Epoch 12/40
910/910 [=====] - 1s 862us/step - loss: 35745.1250 - mean_squared_error: 35745.1250 - va
l_loss: 36447.9453 - val_mean_squared_error: 36447.9453
Epoch 13/40
910/910 [=====] - 1s 857us/step - loss: 35680.0703 - mean_squared_error: 35680.0703 - va
l_loss: 37874.3477 - val_mean_squared_error: 37874.3477
Epoch 14/40
910/910 [=====] - 1s 861us/step - loss: 35611.9102 - mean_squared_error: 35611.9102 - va
l_loss: 36282.8750 - val_mean_squared_error: 36282.8750
Epoch 15/40
910/910 [=====] - 1s 861us/step - loss: 35634.2109 - mean_squared_error: 35634.2109 - va
l_loss: 36535.4062 - val_mean_squared_error: 36535.4062
Epoch 16/40
910/910 [=====] - 1s 862us/step - loss: 35633.1797 - mean_squared_error: 35633.1797 - va
l_loss: 36820.5625 - val_mean_squared_error: 36820.5625
Epoch 17/40
910/910 [=====] - 1s 853us/step - loss: 35534.8320 - mean_squared_error: 35534.8320 - va
l_loss: 36258.6328 - val_mean_squared_error: 36258.6328
Epoch 18/40
910/910 [=====] - 1s 853us/step - loss: 35414.6211 - mean_squared_error: 35414.6211 - va

```

l_loss: 36277.9336 - val_mean_squared_error: 36277.9336
Epoch 19/40
910/910 [=====] - 1s 866us/step - loss: 35397.5508 - mean_squared_error: 35397.5508 - va
l_loss: 36047.1758 - val_mean_squared_error: 36047.1758
Epoch 20/40
910/910 [=====] - 1s 851us/step - loss: 35331.3750 - mean_squared_error: 35331.3750 - va
l_loss: 36172.6953 - val_mean_squared_error: 36172.6953
Epoch 21/40
910/910 [=====] - 1s 850us/step - loss: 35368.3438 - mean_squared_error: 35368.3438 - va
l_loss: 37746.6719 - val_mean_squared_error: 37746.6719
Epoch 22/40
910/910 [=====] - 1s 878us/step - loss: 35268.4414 - mean_squared_error: 35268.4414 - va
l_loss: 35998.4570 - val_mean_squared_error: 35998.4570
Epoch 23/40
910/910 [=====] - 1s 851us/step - loss: 35178.7617 - mean_squared_error: 35178.7617 - va
l_loss: 36435.6641 - val_mean_squared_error: 36435.6641
Epoch 24/40
910/910 [=====] - 1s 857us/step - loss: 35299.0352 - mean_squared_error: 35299.0352 - va
l_loss: 35935.2930 - val_mean_squared_error: 35935.2930
Epoch 25/40
910/910 [=====] - 1s 879us/step - loss: 35176.1836 - mean_squared_error: 35176.1836 - va
l_loss: 35931.9453 - val_mean_squared_error: 35931.9453
Epoch 26/40
910/910 [=====] - 1s 843us/step - loss: 35082.7891 - mean_squared_error: 35082.7891 - va
l_loss: 35959.6719 - val_mean_squared_error: 35959.6719
Epoch 27/40
910/910 [=====] - 1s 855us/step - loss: 35079.4375 - mean_squared_error: 35079.4375 - va
l_loss: 35869.4102 - val_mean_squared_error: 35869.4102
Epoch 28/40
910/910 [=====] - 1s 870us/step - loss: 35020.8516 - mean_squared_error: 35020.8516 - va
l_loss: 35845.1523 - val_mean_squared_error: 35845.1523
Epoch 29/40
910/910 [=====] - 1s 867us/step - loss: 34979.4023 - mean_squared_error: 34979.4023 - va
l_loss: 36008.2656 - val_mean_squared_error: 36008.2656
Epoch 30/40
910/910 [=====] - 1s 868us/step - loss: 35021.2422 - mean_squared_error: 35021.2422 - va
l_loss: 35883.0664 - val_mean_squared_error: 35883.0664
Epoch 31/40
910/910 [=====] - 1s 930us/step - loss: 34939.9453 - mean_squared_error: 34939.9453 - va
l_loss: 35750.3516 - val_mean_squared_error: 35750.3516
Epoch 32/40
910/910 [=====] - 1s 960us/step - loss: 34857.5117 - mean_squared_error: 34857.5117 - va
l_loss: 35851.1914 - val_mean_squared_error: 35851.1914
Epoch 33/40
910/910 [=====] - 1s 926us/step - loss: 34921.9180 - mean_squared_error: 34921.9180 - va
l_loss: 36079.6953 - val_mean_squared_error: 36079.6953
Epoch 34/40
910/910 [=====] - 1s 926us/step - loss: 34908.3945 - mean_squared_error: 34908.3945 - va
l_loss: 35815.1992 - val_mean_squared_error: 35815.1992
Epoch 35/40
910/910 [=====] - 1s 932us/step - loss: 34886.3477 - mean_squared_error: 34886.3477 - va
l_loss: 35861.3086 - val_mean_squared_error: 35861.3086
Epoch 36/40
910/910 [=====] - 1s 885us/step - loss: 34727.7109 - mean_squared_error: 34727.7109 - va
l_loss: 35769.8516 - val_mean_squared_error: 35769.8516
Epoch 37/40
910/910 [=====] - 1s 866us/step - loss: 34709.2070 - mean_squared_error: 34709.2070 - va
l_loss: 35939.3320 - val_mean_squared_error: 35939.3320
Epoch 38/40
910/910 [=====] - 1s 862us/step - loss: 34817.3281 - mean_squared_error: 34817.3281 - va
l_loss: 35465.8164 - val_mean_squared_error: 35465.8164
Epoch 39/40
910/910 [=====] - 1s 862us/step - loss: 34685.1523 - mean_squared_error: 34685.1523 - va
l_loss: 35695.0742 - val_mean_squared_error: 35695.0742
Epoch 40/40
910/910 [=====] - 1s 849us/step - loss: 34662.6016 - mean_squared_error: 34662.6016 - va
l_loss: 35872.1602 - val_mean_squared_error: 35872.1602

```

```
In [133... results_test = NN_model.evaluate(X_test, y_test)
```

```
304/304 [=====] - 0s 408us/step - loss: 35872.1602 - mean_squared_error: 35872.1602
```

```
In [134... test_preds = NN_model.predict(X_test)
```

```
In [135... MeanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
In [136... print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
```

```
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Mean Absolute Error: 58.07161729550634  
Mean Square Error: 35872.17842360018  
Root Mean Square Error: 189.39952065303697
```

In []:

Section 3

Results

In [157...

```
# MLR Results:  
  
# R squared: 6.14  
# Mean Absolute Error: 65.76500421880998  
# Mean Square Error: 36915.32169292407  
# Root Mean Square Error: 192.13360375770833
```

In [158...

```
# KNN Results:  
  
# R squared: 11.61  
# Mean Absolute Error: 69.0433906861735  
# Mean Square Error: 37891.3992404183  
# Root Mean Square Error: 194.65713251873999
```

In [159...

```
# Tuned Decision Tree Results:  
  
# R squared: 8.71  
# Mean Absolute Error: 62.65934621355715  
# Mean Square Error: 36011.40843729863  
# Root Mean Square Error: 189.76672110066778
```

In [161...

```
# Tuned Random Forest Results:  
  
# R squared: 18.48  
# Mean Absolute Error: 58.07161729550634  
# Mean Square Error: 35048.80209410964  
# Root Mean Square Error: 187.21325298736102
```

In [163...

```
# Tuned XGBoost Results:  
  
# R squared: 15.26  
# Mean Absolute Error: 58.07161729550634  
# Mean Square Error: 35147.01263402663  
# Root Mean Square Error: 187.47536540576905
```

In [164...

```
# Neural Network Results:  
  
# Mean Absolute Error: 58.07161729550634  
# Mean Square Error: 35872.17842360018  
# Root Mean Square Error: 189.39952065303697
```

In []: