**PROJECT**

**ON**

# UNITED STATES HOUSE RENT PREDICTION


**Team 12**

Prashanti Salunkhe-002133330
Pratik Randad-002133847

# Table of Contents

# Introduction:

### 1.Background:

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.

### 2.Motivation:

The house rent is an important deciding factor for us to manage the finances. With the growing number of real estates, a rent prediction study only serves to assist investors in determining the earning potential of a specific property in each region with specific qualities, thereby boosting the efficiency of real estate investment in the market. The goal of this project is to assist both landlords and tenants in pricing their rental properties appropriately. We're working to improve the rental market's transparency and accessibility. This prediction technique will also assist investors in making informed investment selections to optimize their profits.

### 3.Goal

We want to build a model that can inform people what a fair rent for a specific listing (home) in a specific location with amenities would cost at any given time. Using various types of models, we want to reduce the disparity between the real rent and the rent anticipated. Various sorts of machine learning techniques will be used to assess the performance of our model.

## Methodology:

### 1.Data Summary:

The dataset for housing prices is 380.29 MB in size and has 22 columns. It contains the unique id of every house along with its details like region, type, number of rooms, amenities, pet allowances, description, location, and most important attribute ‘price’.

### 2.Software and Libraries:

Software used:

- Jupyter Notebooks

Libraries:

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Geopandas
- Shapely
- Sklearn

### 3.Data Cleaning:

Because the data was so large, there were numerous missing and NaN values in the dataset. Since many rows contained NaN values, which could result in lower accuracy models. To sort, all the rows with NaN and missing values have their values eliminated. We cleaned the dataset and created our model on 16 columns because the original dataset had 20+ columns, which can cause too much noise or distortion in the final dataset. Many of those columns included NaN or missing values and some of the columns were redundant therefore we developed our model on 16 columns.

## 4.Data Visualization:

We visualized the dataset to understand each column better and answer questions. We did Data Visualization utilizing various graphs and visualizations after cleansing the dataset and dealing with outliers. First, we showed data visualization and removed the price>4000, then we plotted again after encoding the variables of missing values and handling all the missing values, then we visualized outliers and removed all the outliers in the dataset, and finally, we created a correlation heatmap that shows the highest co-relation of certain columns. This was used for feature selection.

## 5.Models Used:

Since we had to predict a dependent variable (price) by using the independent variables we chose regression models.

The following are some of the models that were utilized in the project:

- Linear Regression Regressor
- K-Nearest Neighbor Classification
- Decision Tree Regressor
- Random Forest Regressor
- Gradient Boosting Regressor

### 1.Linear Regressor

Linear regression is a widely used algorithm that predicts the output variable using a single input variable. This method also works with multivariable input. We have factors in our dataset that have a direct relationship with the goal variable, such as the location of a property, which can increase the rent price, and so on. This model, we believe, has the potential to work better with our data and should be evaluated as a possible model.

### 2.K-Nearest Neighbor Regressor

KNN can be utilized for both classification and regression issues, as we know. It operates by predicting values based on feature similarity and assigning a value based on similarity in the training set. By analyzing the closeness of a house in a similar region with similar traits or features, it might work with our data.

3.Decision Tree Regressor

The decision tree divides data into smaller and smaller subsets until it reaches the lowest leaf node. This model is a possible model because it has elements such as dogs allowed, parking options, etc. that could influence the rent of the house.

4.Random Forest Regressor

A decision tree collection known as a Random Forest is a collection of decision trees. It selects K data points at random from the data collection to create a decision tree for these data points. We must select the number of decision trees we require. Thus, we believe this will provide the most efficiency.

**6.Feature Selection**

To lower the calculation time and accuracy of our model, we chose variables that had the greatest impact on our prediction variable.

# Dataset Description

## 1.List of Attributes:

| **Features / Columns** | **Description** |
|---|---|
| **id** | Unique id of every House in the dataset |
| **url** | URL of the house listing |
| **region** | Regional location of house |
| **region_url** | Region URL |
| **price** | Rent of the house |
| **type** | Type of house. E.g., Apartment |
| **sqfeet** | Size of house in Square-feet |
| **beds** | Number of Bedrooms |
| **baths** | Number of Bathrooms |
| **cats_allowed** | Indicates whether cats are allowed |
| **dogs_allowed** | Indicates whether dogs are allowed |
| **smoking_allowed** | Indicates whether smoking is allowed |
| **wheelchair_access** | Indicate whether the house has wheelchair access |
| **electric_vehicle_charge** | Indicates if house have electrical vehicle charging? |
| **comes_furnished** | House is furnished or not |
| **laundary_options** | Type of laundry options available |
| **parking_options** | Type of parking available for the house |
| **image_url** | URL of the house's picture |
| **description** | Description of the house |
| **lat** | Latitude location of the house |
| **long** | Longitude location of the house |

| Features / Columns | Description |
| --- | --- |
| **state** | State location of the house |

**2.Data Source:**

This dataset has been taken from Kaggle
https://www.kaggle.com/datasets/rkb0023/houserentpredictiondataset

# Results and Analysis

## 1.Data Exploration and cleaning

There are 22 columns in the dataset. The project's goal is to create a Machine Learning model that can estimate the rental pricing of a house across the United States based on several criteria that describe the houses' characteristics. This dataset contains several features that characterize the entire nature of the house and its dynamics, as well as a goal feature called 'Price,' which must be forecasted. We were able to remove several columns and rows.

Following is the detailed explanation and screenshots of how the idea worked:

- **Import required libraries and data:**

```
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
```

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

- **Data Information and Description:**

Out[7]:

| | id | price | sqfeet | beds | baths | cats_allowed | dog |
|---|---|---|---|---|---|---|---|
| count | 2.651900e+05 | 2.651900e+05 | 2.651900e+05 | 265190.000000 | 265190.000000 | 265190.000000 | 2651 |
| mean | 7.040888e+09 | 1.227285e+04 | 1.093678e+03 | 1.912414 | 1.483468 | 0.716822 | |
| std | 8.778930e+06 | 5.376352e+06 | 2.306888e+04 | 3.691900 | 0.630208 | 0.450543 | |
| min | 7.003808e+09 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 7.035963e+09 | 8.170000e+02 | 7.520000e+02 | 1.000000 | 1.000000 | 0.000000 | |
| 50% | 7.043109e+09 | 1.060000e+03 | 9.500000e+02 | 2.000000 | 1.000000 | 1.000000 | |
| 75% | 7.048362e+09 | 1.450000e+03 | 1.156000e+03 | 2.000000 | 2.000000 | 1.000000 | |
| max | 7.051263e+09 | 2.768307e+09 | 8.388607e+06 | 1100.000000 | 75.000000 | 1.000000 | |

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
```

- **Data Cleaning and Feature Selection:**
  Dropping unwanted columns

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

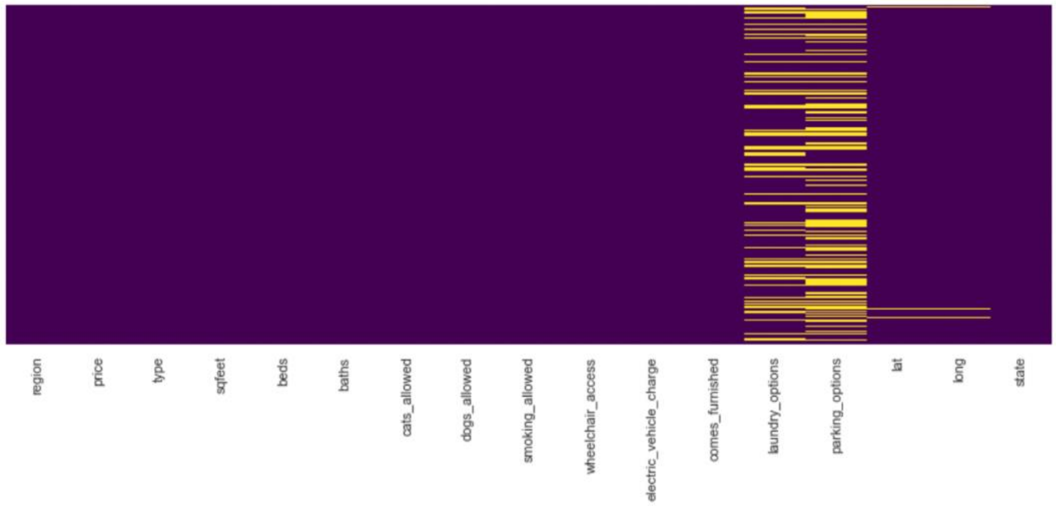
In [10]:
```
data.head()
```

Out[10]:

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

**2.Analyzing and Filling Null Values:**

Null Values Visualization




Total Missing values in each column

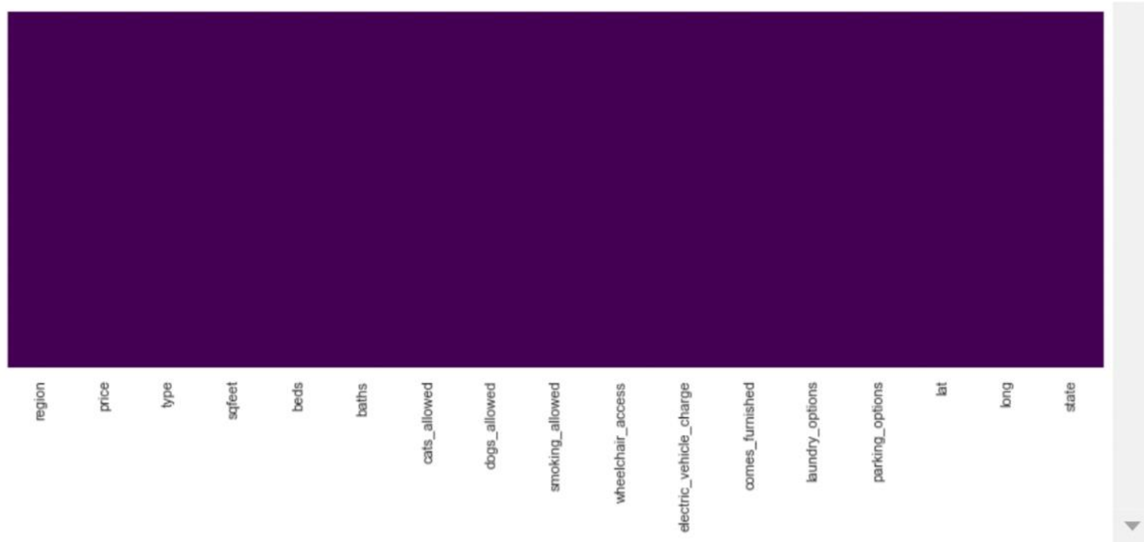
```
In [12]: data.isna().sum()

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
```

Filling Null values by frequently used values in each column

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

Verifying Null Values after filling them



**3.Handling Outliers**

Detecting and managing outliers is one of the most critical tasks in data preparation since they can significantly affect statistical analysis and the training process of a machine learning system, resulting in reduced accuracy.

- **Checking Outliers in each column**

  Example 1 - 'Type' Column

```
data['type'].value_counts()
```

```
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
```

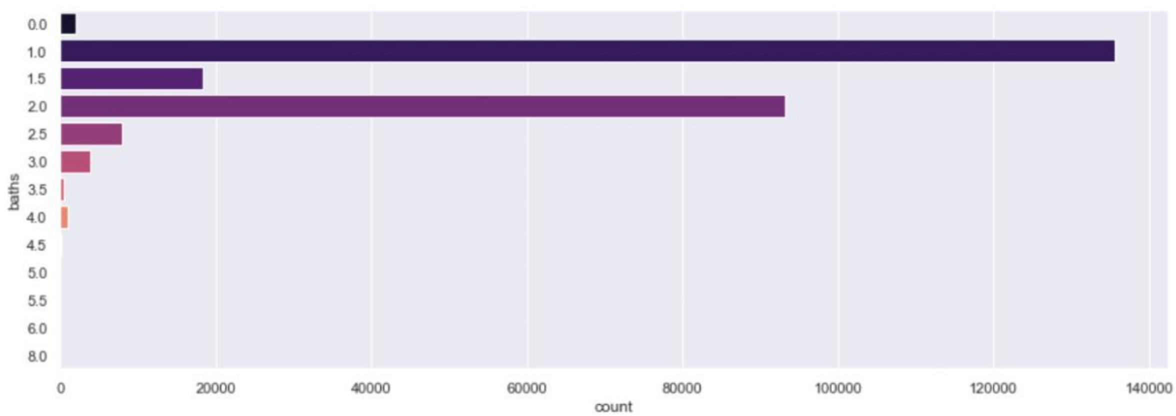

Removing data of land and assisted living as it is so small compared to other values

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

Example 2- 'Bath' Column

```
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
```

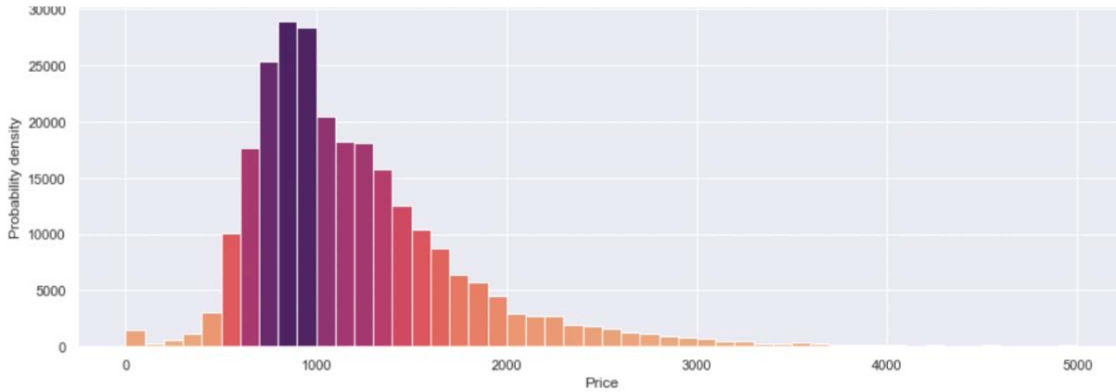



Number of baths cannot be 0 and values more than 6 are outliers, hence removed them

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

**4.Discover Data Patterns**

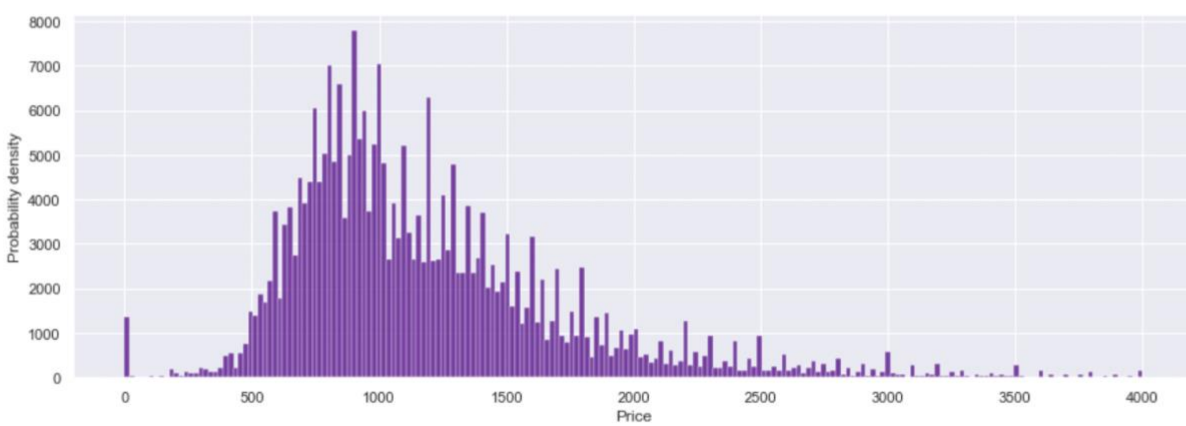
- **Identifying outliers of the key attribute 'Price'**



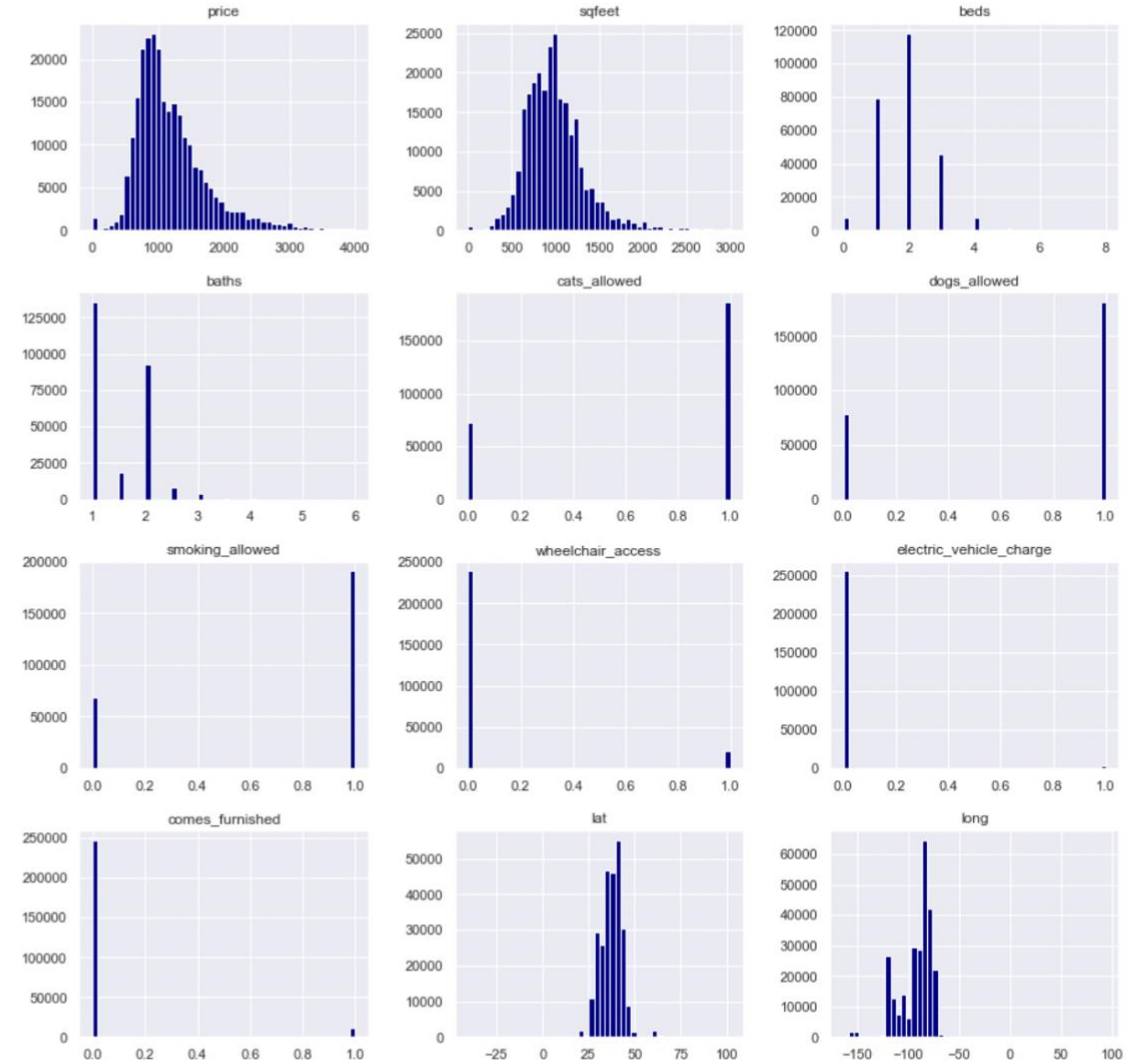

Removing price greater than 4000

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

After removal



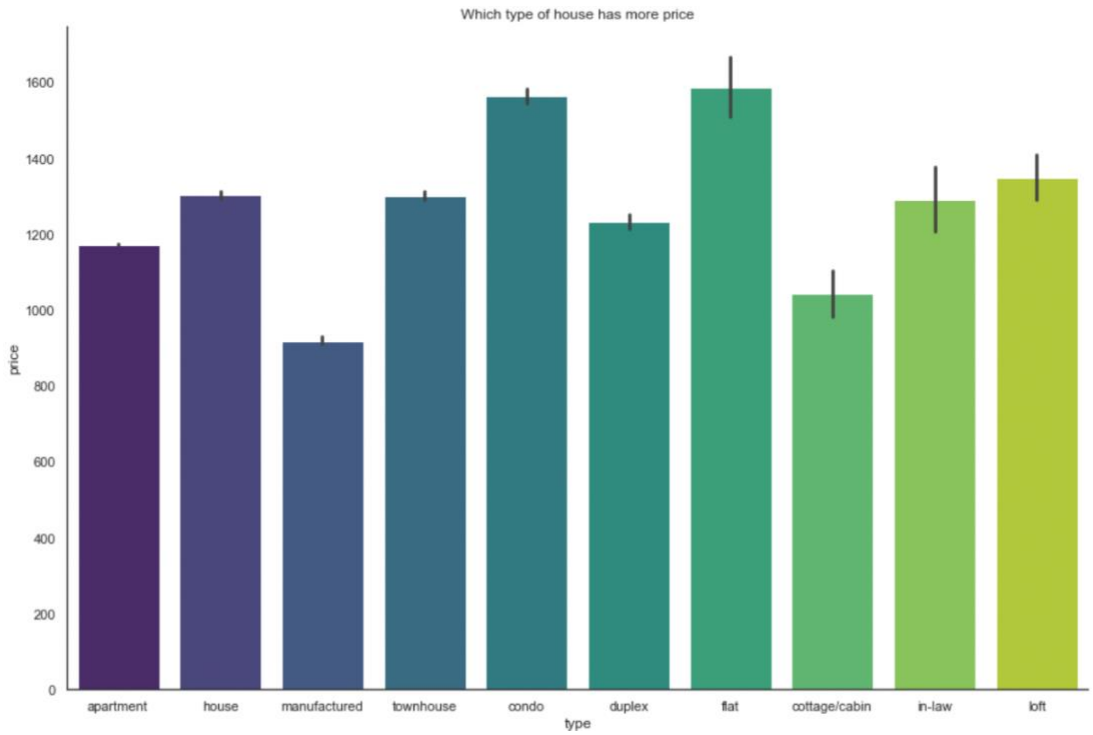
- **Visualizing target attribute vs other attributes:**



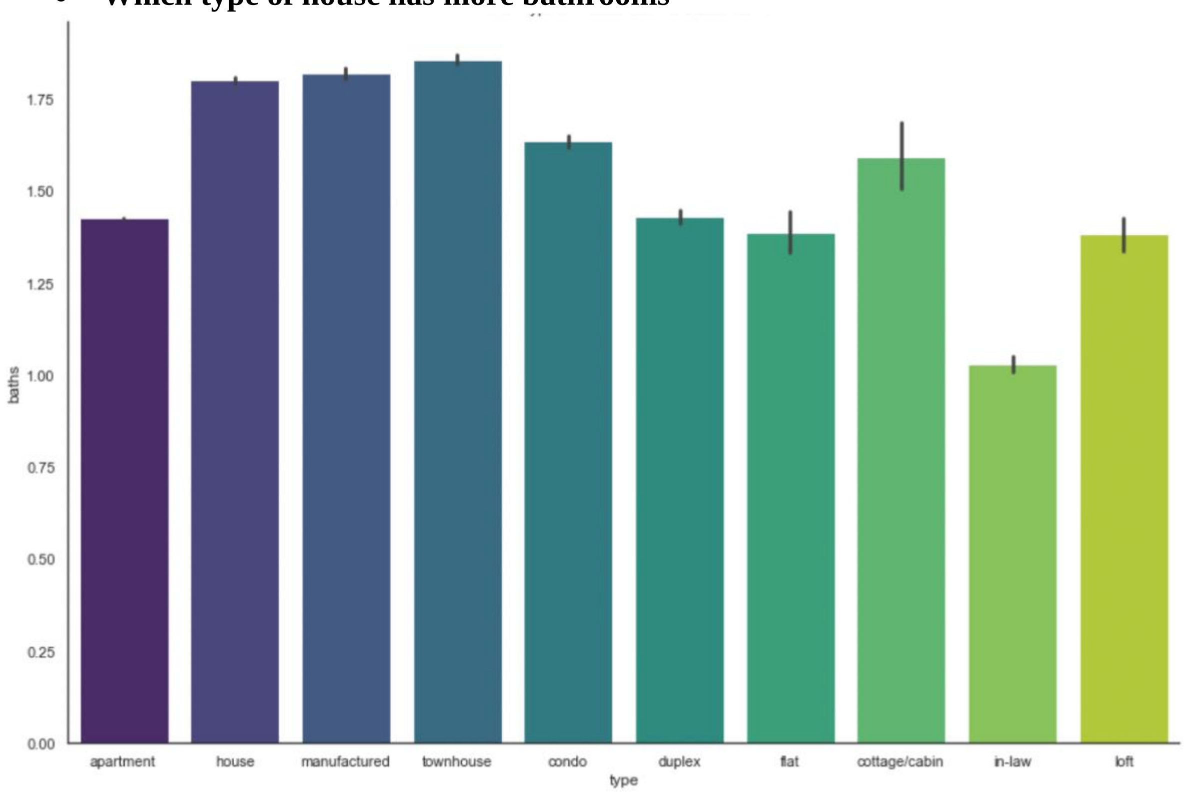
- **Correlation between columns:**


- **Sqfeet vs type vs beds:**

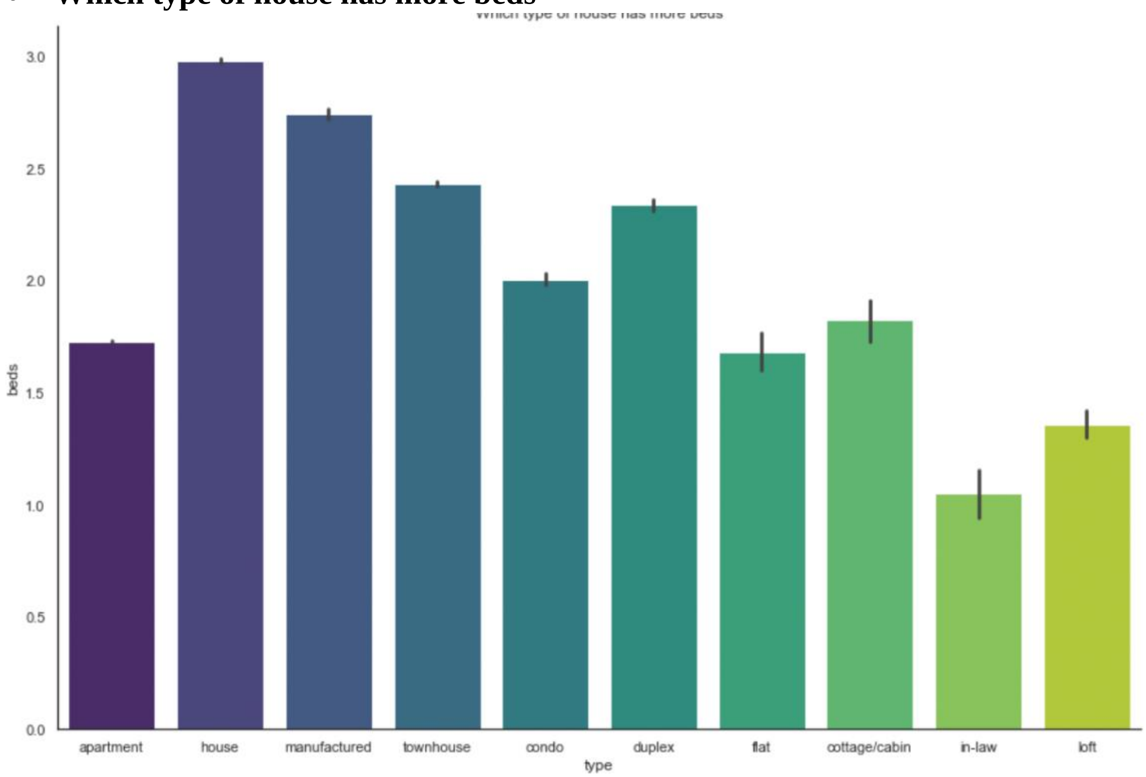
- **Which type of house has more price**



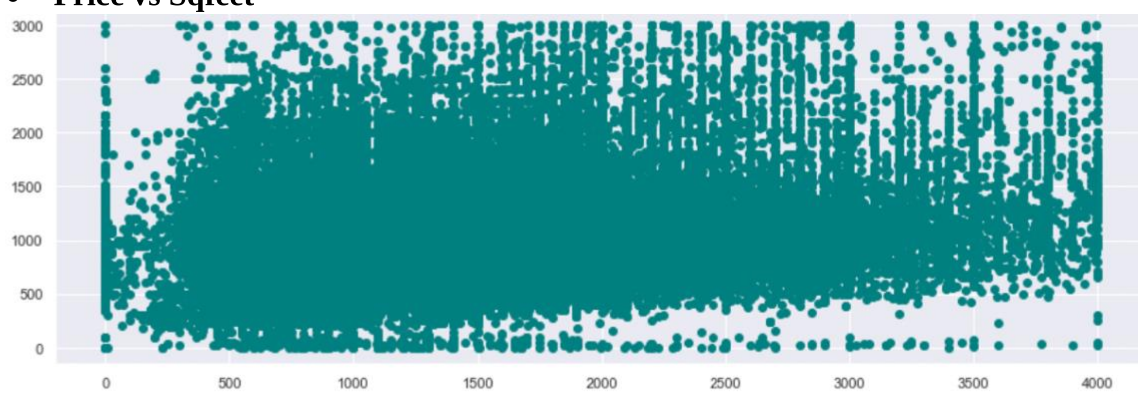

- **Which type of house has more bathrooms**



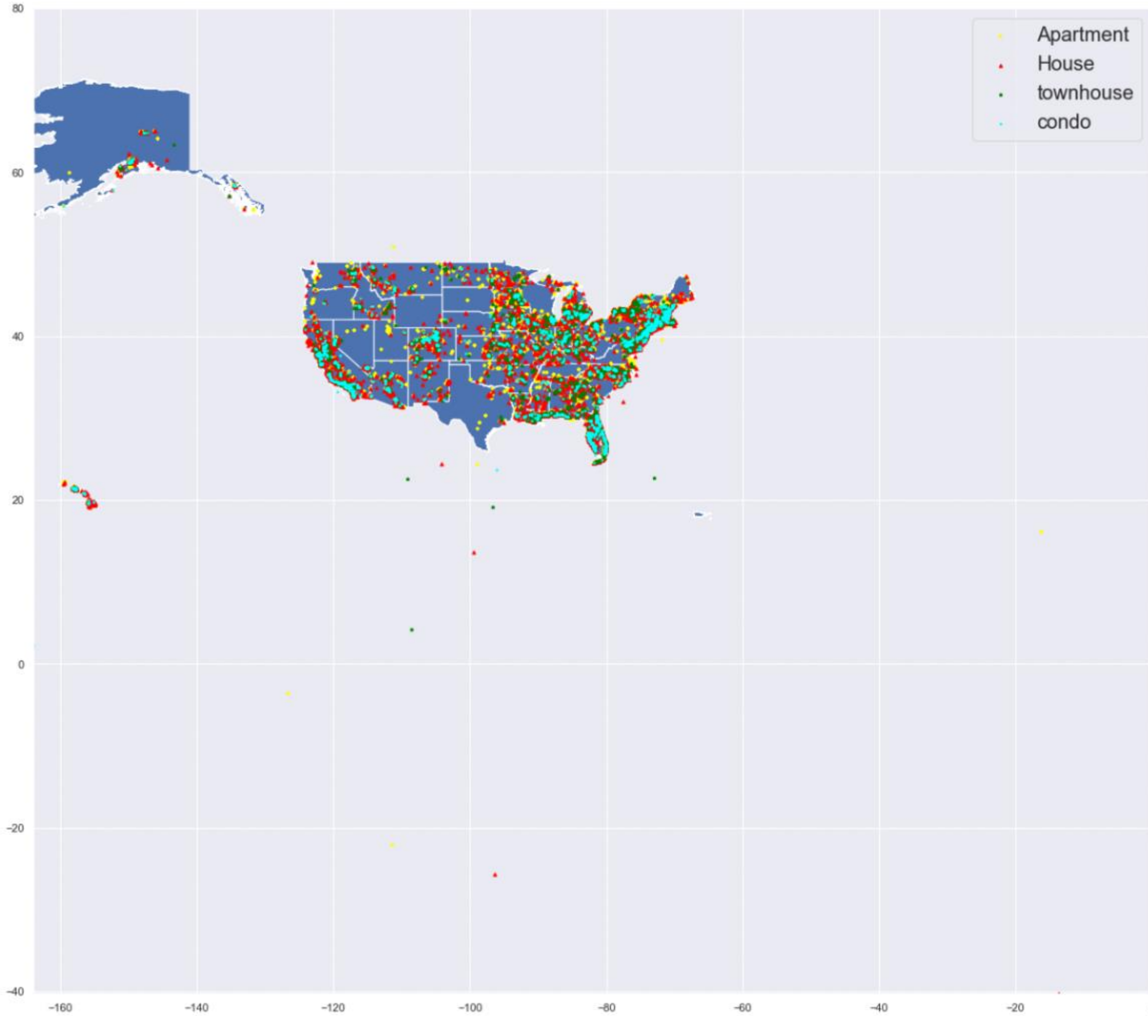
- **Which type of house has more beds**




- **Price vs Sqfeet**



- **Visualizing Data with Geo Map:**



## 5.Data Pre-Processing:

- **Verifying null values again:**

```
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
```

- **Transforming all data into numerical format:**

```
[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"])
```

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

[67]:

| | region | price | type | sqfeet | beds | baths | cats_allowed | dogs_allowed | smoking_allowed | wheelchai |
|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |

---

- **Removing strongly correlated columns:**

```
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 | elect |
|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |

---

- **Final data Description:**

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

Out[70]:

| | count | mean | std | min | 25% | 50% | 75% |
|---|---|---|---|---|---|---|---|
| **region** | 259319.0 | 139.562651 | 88.461359 | 0.0000 | 59.0000 | 139.0000 | 217.0000 |
| **price** | 259319.0 | 1194.645518 | 557.423583 | 0.0000 | 815.0000 | 1053.0000 | 1435.0000 |
| **type** | 259319.0 | 0.954770 | 2.324722 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **sqfeet** | 259319.0 | 985.955264 | 351.330701 | 0.0000 | 750.0000 | 950.0000 | 1150.0000 |
| **beds** | 259319.0 | 1.887582 | 0.868298 | 0.0000 | 1.0000 | 2.0000 | 2.0000 |
| **baths** | 259319.0 | 1.483763 | 0.565016 | 1.0000 | 1.0000 | 1.0000 | 2.0000 |
| **pets_allowed** | 259319.0 | 0.698059 | 0.459101 | 0.0000 | 0.0000 | 1.0000 | 1.0000 |
| **smoking_allowed** | 259319.0 | 0.734940 | 0.441366 | 0.0000 | 0.0000 | 1.0000 | 1.0000 |
| **wheelchair_access** | 259319.0 | 0.078872 | 0.269539 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **electric_vehicle_charge** | 259319.0 | 0.014025 | 0.117595 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **comes_furnished** | 259319.0 | 0.046599 | 0.210779 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **laundry_options** | 259319.0 | 2.920804 | 1.447557 | 0.0000 | 1.0000 | 4.0000 | 4.0000 |
| **parking_options** | 259319.0 | 3.206846 | 1.484986 | 0.0000 | 2.0000 | 4.0000 | 4.0000 |
| **lat** | 259319.0 | 37.219989 | 5.659076 | -40.2666 | 33.5059 | 37.9863 | 41.1901 |
| **long** | 259319.0 | -92.298855 | 17.300744 | -163.8940 | -103.6240 | -86.4231 | -81.2846 |
| **state** | 259319.0 | 15.402901 | 10.001982 | 0.0000 | 7.0000 | 14.0000 | 23.0000 |

## 6.Data Modelling and Prediction:

- **Data Models:**

Random Forest Regressor:

```
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)
```

```
0.8961643213123793
```

Decision Tree Regressor:

```
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)
```

```
0.8167479247706172
```

Linear Regression:

```
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)
```

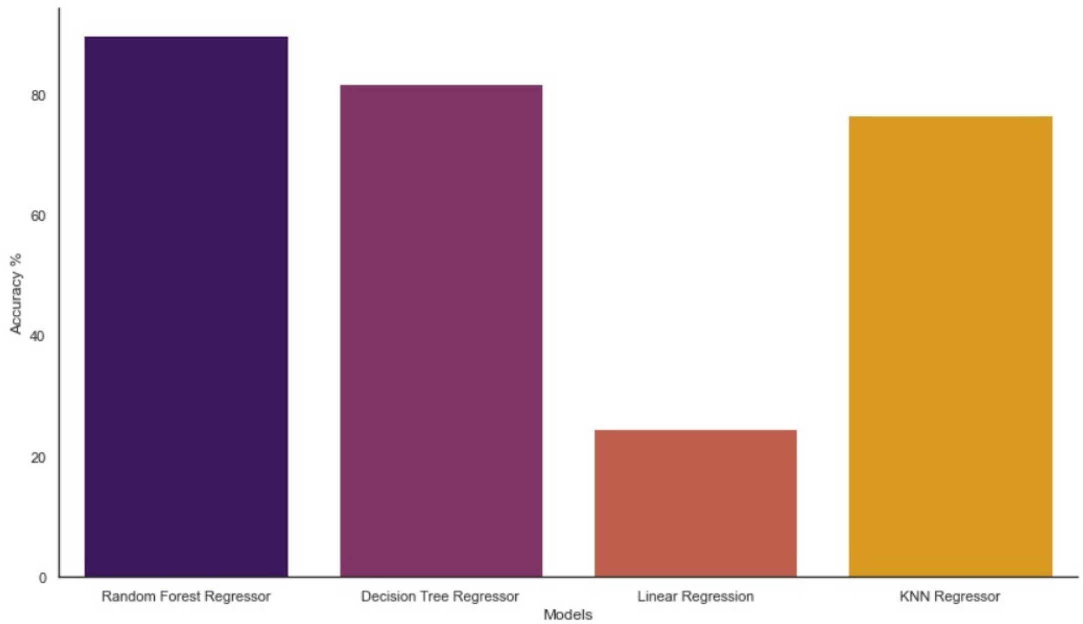
```
0.24567602753126283
```

KNN Regression:

```
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
```

- **Model Accuracies comparison:**



## Conclusion:

- We've completed all the essential data processing and calculations, and Random Forest has the greatest accuracy score for Rent Prediction of all the algorithms employed.
- For achieving higher accuracy with Linear regression model, we need data that has linear relationship between dependent and independent variables, since in our model only some of the features followed it Linear regression model had lowest accuracy.
- As Decision tree algorithm works for both the classification and regression problems, here also we got good accuracy for decision tree regression algorithm.
  If we do further feature selection that may increase the efficiency even more.
- We tried multiple values for neighbors in KNN regression model and we got the highest accuracy for neighbors=2.
- Random forest uses ensemble learning techniques for regression and classification tasks. Since it uses average value predicted from multiple random trees it gives the highest accuracy.

## References:


https://medium.com/analytics-vidhya/fastest-way-to-install-geopandas-in-jupyter-notebook-on-windows-8f734e11fa2b

https://www.statista.com/statistics/456925/median-size-of-single-family-home-usa/

https://hersanyagci.medium.com/detecting-and-handling-outliers-with-pandas-7adbfcd5cad8

https://catalog.data.gov/dataset/tiger-line-shapefile-2017-nation-u-s-current-state-and-equivalent-national

https://seaborn.pydata.org/tutorial/color_palettes.html

https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/

https://stackoverflow.com/questions/49780491/plotting-histogram-for-all-columns-in-a-data-frame