Project 1

Graded

Student

TIYA CHOKHANI

Total Points

92 / 100 pts

Question 1

Visualizing the Data **35** / 35 pts

1.1 **Load the data + statistics** **10** / 10 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Did not call describe method or info for summary output
- **– 2.5 pts** No code for removing columns is provided.
- **– 2.5 pts** No code provided for the head method

1.2 **Some basic visualizations** **10** / 10 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Wrong plot/pie chart
- **– 10 pts** Missing/wrong plots
- **– 1 pt** Sum not used (Total Reviews per Room Type)
- **– 2 pts** partial credit

1.3 **Plot a map with matplotlib** **5** / 5 pts

- ✔ **– 0 pts** Correct. [As long as the type and layout of the plots are correct, it's acceptable. For instance, with the map plots, as long as they have scatter points overlaid on a map and the positioning is well aligned, they will be credited.]
- **– 5 pts** Missing plot
- **– 3 pts** Unrecognizable plot, but correct functions are used.
- **– 3 pts** Missing Colour
- **– 5 pts** Incorrect Plot Features

1.4 **Plot a map with Plotly** **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Missing plot
- **– 3 pts** Unrecognizable plot, but correct functions are used.
- **– 2 pts** Missing color

1.5 **Plot the average price of room types** **5** / 5 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Missing plot
- **– 3 pts** Unrecognizable plot, but correct functions are used.
- **– 1 pt** Minor mistake (e.g. missing one filtering condition)

Question 2

Prepare the Data **45** / 50 pts

2.1 **Feature engineering** **10** / 10 pts

✔ **– 0 pts** Correct

– 10 pts No binning of Data

– 2 pts Distribution of values not checked

2.2 **Data Inputation** **5** / 10 pts

– 0 pts Correct

– 0 pts Filled null values with 0. fillna() should be applied to individual columns with mean, median, or mode as the default value (unless sufficient data exploration occurred / reasoning is provided).

– 3 pts Didn't impute price_cat feature (when necessary)

✔ **– 5 pts** Dropped features/rows instead of filling null values.

– 10 pts No imputation.

2.3 **Numeric Conversions** **10** / 10 pts

✔ **– 0 pts** Correct

– 10 pts Not answered or wrong answer. [At least two features should be processed: `neighbourhood_group` and `room_type`.]

– 0.5 pts Encoded price_cat

– 5 pts Partial credit

2.4 **Prepare data for machine learning** **20** / 20 pts

✔ **– 0 pts** Correct

– 20 pts Stratified Sampling is not performed on "price_cat"

– 10 pts Incorrect train/test split

– 10 pts No stratified sampling

– 10 pts Price_cat or price are not removed from train features

Question 3

Fit a Linear Regression Model

12 / 15 pts

- **– 0 pts** Correct. [The value does not have to match exactly that in the provided answer.]
- **– 15 pts** No answer
- **– 5 pts** Didn't fit a model
- **– 5 pts** Didn't report MSE
- ✔ **– 3 pts** Missing train OR test MSE
  - 1 missing train MSE
- **– 0 pts** Prediction should have been performed on price, not price_cat

No questions assigned to the following page.

24W-COM SCI-M148 Project 1

Name: Tiya Chokhani UID: 305933966

Submission Guidelines (Due: Jan 27 before the class)

1. Please fill in your name and UID above.
2. Please submit a **PDF printout** of your Jupyter Notebook to **Gradescope**. If you have any trouble accessing Gradescope, please let a TA know ASAP.
3. When submitting to Gradescope, you will be taken to a page that asks you to assign questions and pages. As the PDF can get long, please make sure to assign pages to corresponding questions to ensure the readers know where to look.

Introduction

Welcome to **CS148 - Introduction to Data Science!** As we're planning to move through topics aggressively in this course, to start out, we'll look to do an end-to-end walkthrough of a datascience project, and then ask you to replicate the code yourself for a new dataset.

Please note: We don't expect you to fully grasp everything happening here in either code or theory. This content will be reviewed throughout the quarter. Rather we hope that by giving you the full perspective on a data science project it will better help to contextualize the pieces as they're covered in class

In that spirit, we will first work through an example project from end to end to give you a feel for the steps involved.

Here are the main steps:

1. Get the data
2. Visualize the data for insights
3. Preprocess the data for your machine learning algorithm
4. Select a machine learning model and train it
5. Evaluate its performance

Working with Real Data

It is best to experiment with real-data as opposed to aritifical datasets.

There are many different open datasets depending on the type of problems you might be interested in!

Here are a few data repositories you could check out:

- UCI Datasets
- Kaggle Datasets
- AWS Datasets

Below we will run through an California Housing example collected from the 1990's.

Setup

We'll start by importing a series of libraries we'll be using throughout the project.

```
import sys
assert sys.version_info >= (3, 5) # python>=3.5
import sklearn
#assert sklearn.__version__ >= "0.20" # sklearn >= 0.20

import numpy as np #numerical package in python
%matplotlib inline
import matplotlib.pyplot as plt #plotting package

# to make this notebook's output identical at every run
np.random.seed(42)

#matplotlib magic for inline figures
%matplotlib inline
import matplotlib # plotting library
import matplotlib.pyplot as plt
```

No questions assigned to the following page.

Intro to Data Exploration Using Pandas

In this section we will load the dataset, and visualize different features using different types of plots.

Packages we will use:

- **Pandas**: is a fast, flexibile and expressive data structure widely used for tabular and multidimensional datasets.
- **Matplotlib**: is a 2d python plotting library which you can use to create quality figures (you can plot almost anything if you're willing to code it out!)
    - other plotting libraries:seaborn, ggplot2

Note: If you're working in CoLab for this project, the CSV file first has to be loaded into the environment. This can be done manually using the sidebar menu option, or using the following code here.

If you're running this notebook locally on your device, simply proceed to the next step.

```
# from google.colab import files
# files.upload()
```

We'll now begin working with Pandas. Pandas is the principle library for data management in python. It's primary mechanism of data storage is the dataframe, a two dimensional table, where each column represents a datatype, and each row a specific data element in the set.

To work with dataframes, we have to first read in the csv file and convert it to a dataframe using the code below.

```
# We'll now import the holy grail of python datascience: Pandas!
import pandas as pd
housing = pd.read_csv('housing.csv')
```

```
housing.head() # show the first few elements of the dataframe
               # typically this is the first thing you do
               # to see how the dataframe looks like
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

A dataset may have different types of features

- real valued
- Discrete (integers)
- categorical (strings)
- Boolean

The two categorical features are essentialy the same as you can always map a categorical string/character to an integer.

In the dataset example, all our features are real valued floats, except ocean proximity which is categorical.

```
# to see a concise summary of data types, null values, and counts
# use the info() method on the dataframe
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
```

No questions assigned to the following page.

```
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
# you can access individual columns similarly
# to accessing elements in a python dict
housing["ocean_proximity"].head() # added head() to avoid printing many columns..
```

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY

dtype: object

```
# to access a particular row we can use iloc
housing.iloc[1]
```

	1
longitude	-122.22
latitude	37.86
housing_median_age	21.0
total_rooms	7099.0
total_bedrooms	1106.0
population	2401.0
households	1138.0
median_income	8.3014
median_house_value	358500.0
ocean_proximity	NEAR BAY

dtype: object

```
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

ocean_proximity	count
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

dtype: int64

```
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

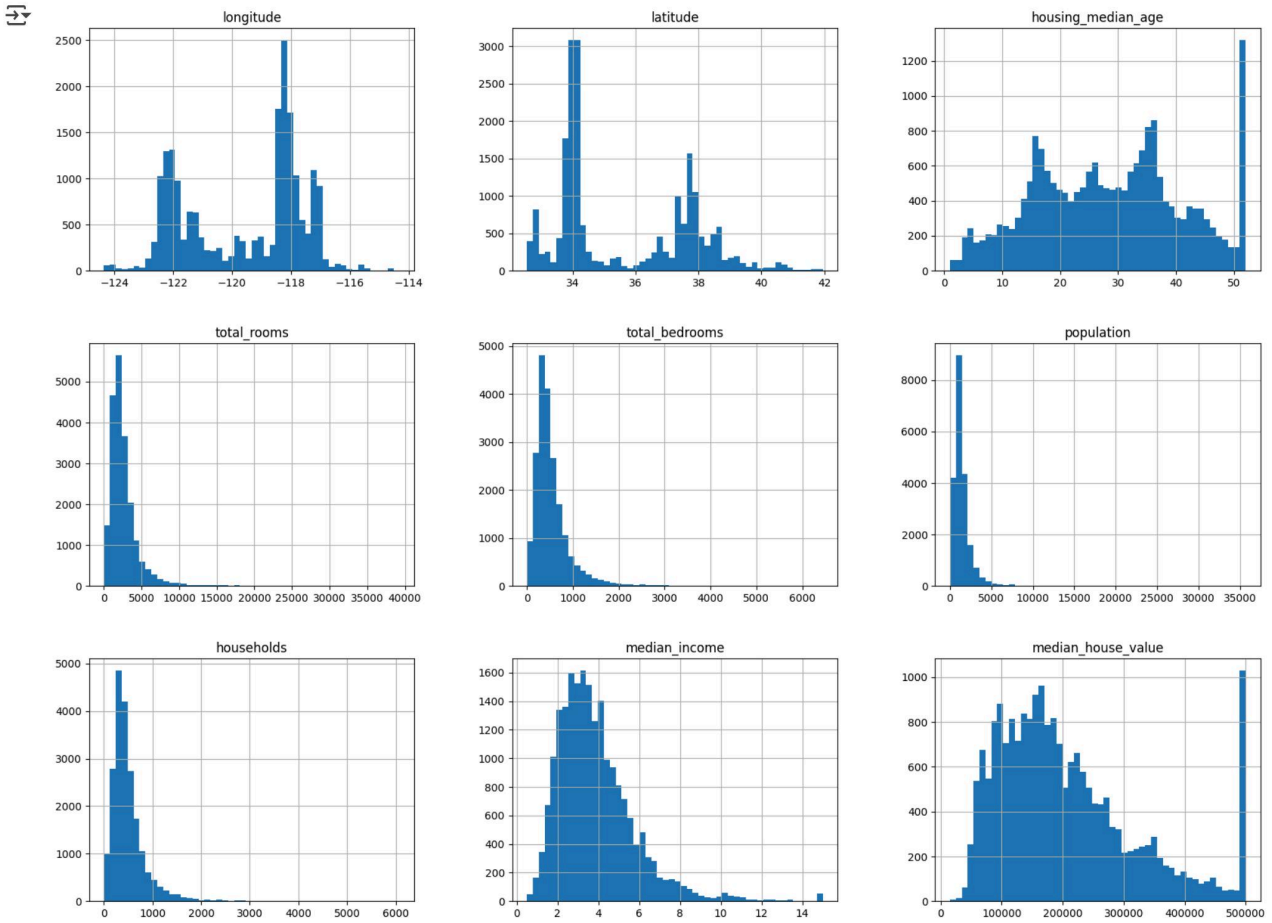
No questions assigned to the following page.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	m
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	

If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

Let's start visualizing the dataset

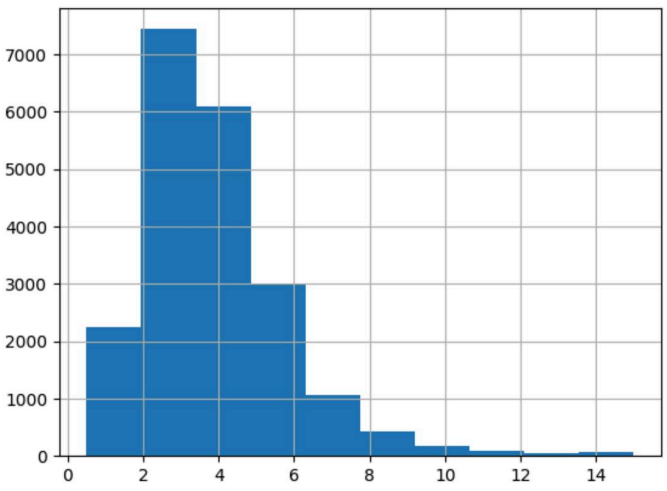
```
# We can draw a histogram for each of the dataframes features
# using the hist function
housing.hist(bins=50, figsize=(20,15))
# save_fig("attribute_histogram_plots")
plt.show() # pandas internally uses matplotlib, and to display all the figures
           # the show() function must be called
```




```
# if you want to have a histogram on an individual feature:
housing["median_income"].hist()
```

No questions assigned to the following page.

```
plt.show()
```



We can convert a floating point feature to a categorical feature by binning or by defining a set of intervals.

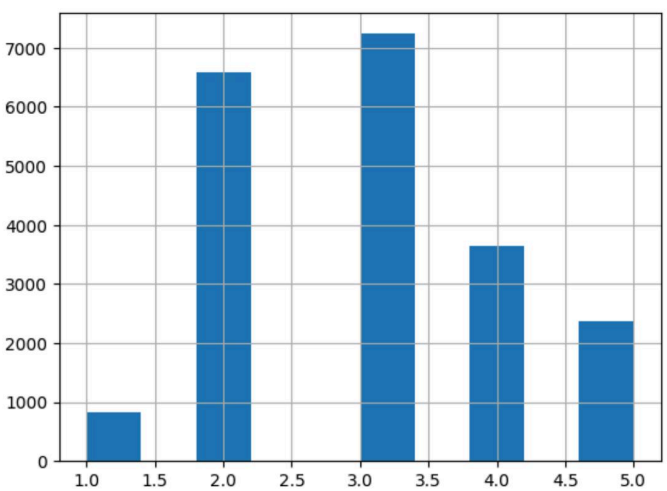
For example, to bin the households based on median_income we can use the pd.cut function

```
# assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

housing["income_cat"].value_counts()
```

income_cat	count
3	7236
2	6581
4	3639
5	2362
1	822

dtype: int64

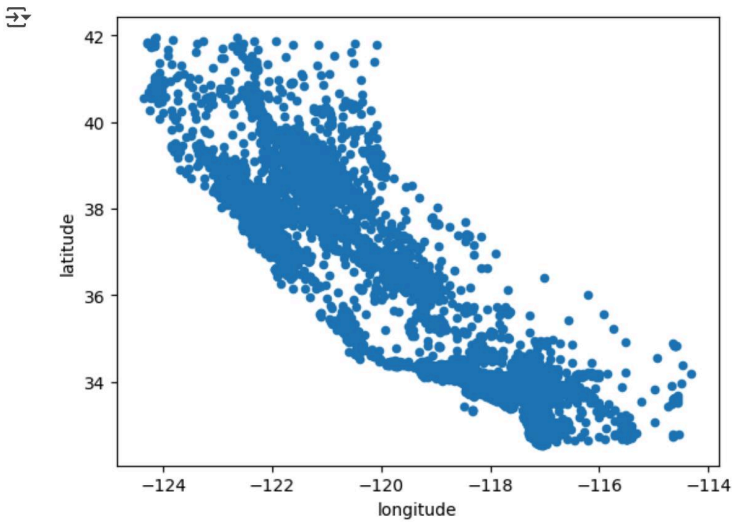
```
housing["income_cat"].hist()
plt.show()
```



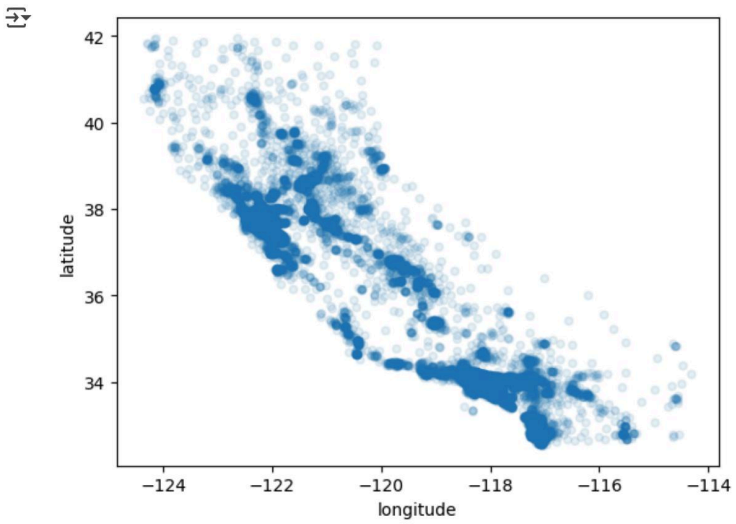
Next let's visualize the household incomes based on latitude & longitude coordinates

No questions assigned to the following page.

```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
plt.show()
```




```
# we can make it look a bit nicer by using the alpha parameter,
# it simply plots less dense areas lighter.
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
plt.show()
```

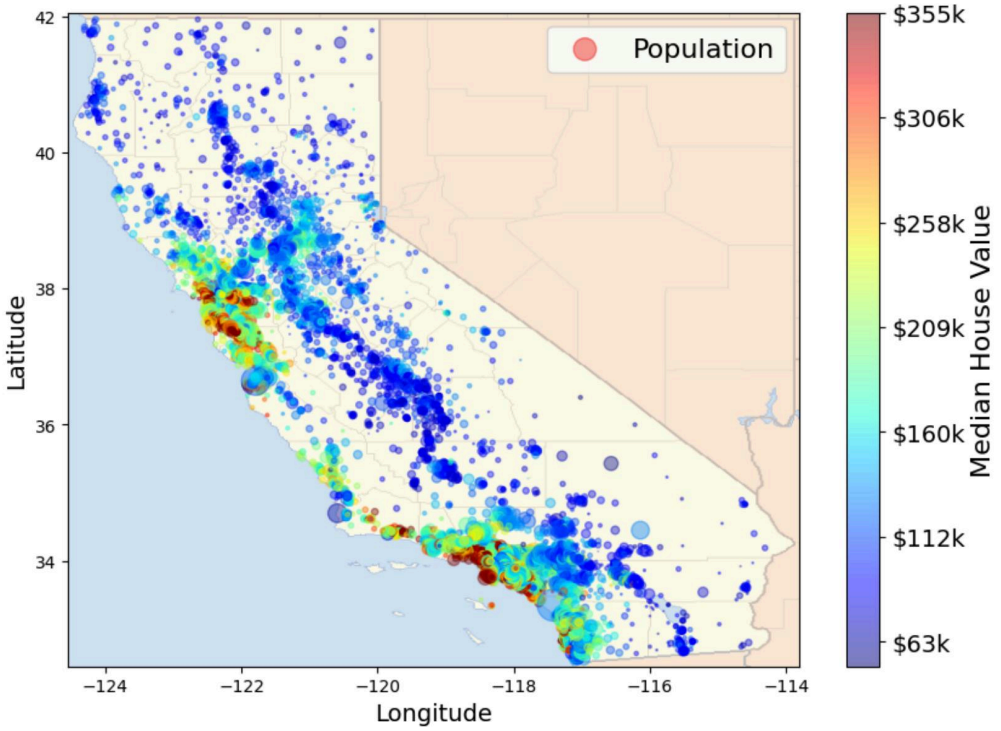



```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# Please note: In order for this to work, ensure that you've loaded an image
# of california (california.png) into this directory prior to running this

import matplotlib.image as mpimg
california_img=mpimg.imread('california.png')
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                      s=housing['population']/100, label="Population",
                      c="median_house_value", cmap=plt.get_cmap("jet"),
                      colorbar=False, alpha=0.4,
                     )
# overlay the califronia map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
```

No questions assigned to the following page.

```
cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cb.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```

```
<ipython-input-19-f4f6d29f5992>:26: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e.
  cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
```



Not suprisingly, the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

Up until now we have only visualized feature histograms and basic statistics.

When developing machine learning models the predictiveness of a feature for a particular target of intrest is what's important.

It may be that only a few features are useful for the target at hand, or features may need to be augmented by applying certain transfomrations.

None the less we can explore this using correlation matrices.

```
# Select only numeric columns
numeric_housing = housing.select_dtypes(include=[float, int])

# Compute the correlation matrix
corr_matrix = numeric_housing.corr()
corr_matrix
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_in
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-0.0
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-0.0
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-0.1
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	0.1
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-0.0
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	0.0
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	0.0
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	1.0
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	0.6

Next steps: Generate code with corr_matrix | View recommended plots | New interactive sheet

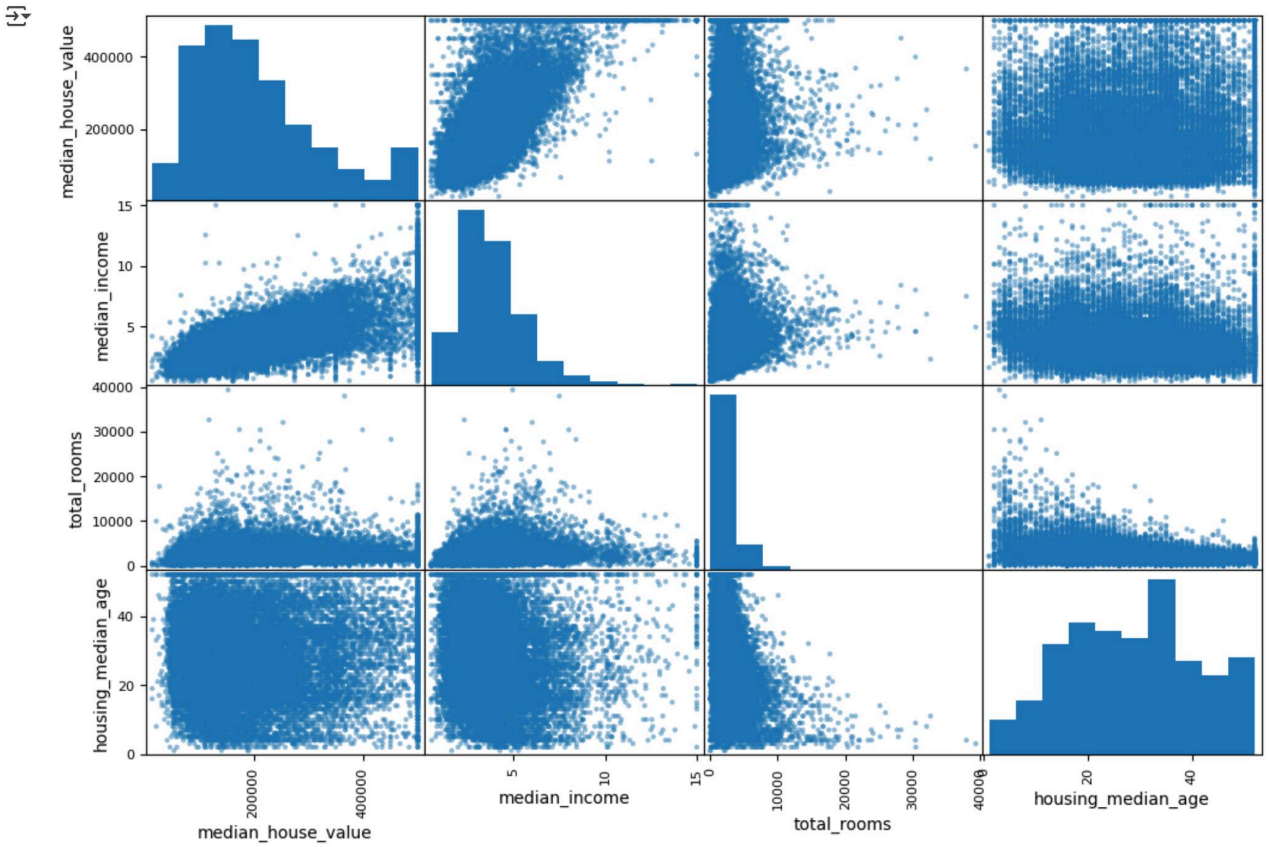
No questions assigned to the following page.

```
# for example if the target is "median_house_value", most correlated features can be sorted
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688075
total_rooms	0.134153
housing_median_age	0.105623
households	0.065843
total_bedrooms	0.049686
population	-0.024650
longitude	-0.045967
latitude	-0.144160

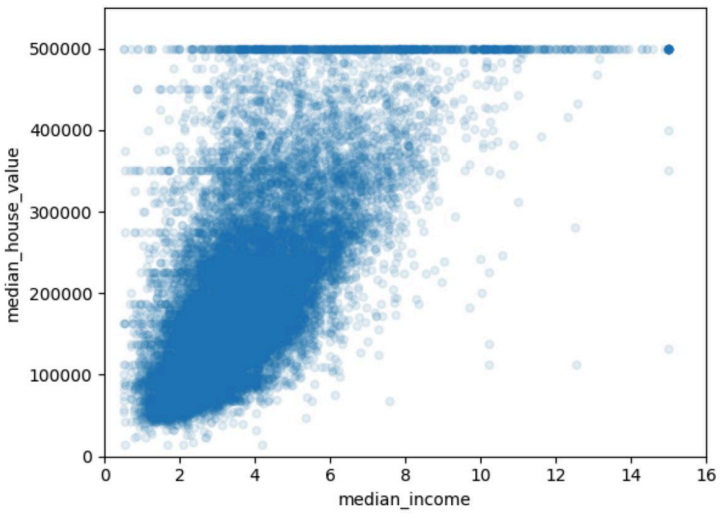
dtype: float64

```
# the correlation matrix for different attributes/features can also be plotted
# some features may show a positive correlation/negative correlation or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))

plt.show()
```




```
# median income vs median house vlue plot plot 2 in the first row of top figure
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
plt.show()
```

No questions assigned to the following page.



Preparing Dastaset for ML

Dealing With Incomplete Data

```
# have you noticed when looking at the dataframe summary certain rows
# contained null values? we can't just leave them as nulls and expect our
# model to handle them for us...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	4.3750	
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	1.6196	
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	2.5762	
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	4.9489	
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	3.9063	

Next steps: Generate code with `sample_incomplete_rows` View recommended plots New interactive sheet

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply drop rows that have null values
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_hous

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the complete feature
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	median_house_value	ocea
290	-122.16	37.77	47.0	1256.0	570.0	218.0	4.3750	161900.0	
341	-122.17	37.75	38.0	992.0	732.0	259.0	1.6196	85100.0	
538	-122.28	37.78	29.0	5154.0	3741.0	1273.0	2.5762	173400.0	
563	-122.24	37.75	45.0	891.0	384.0	146.0	4.9489	247100.0	
696	-122.10	37.69	41.0	746.0	387.0	161.0	3.9063	178400.0	

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3: replace na values with median values
sample_incomplete_rows
```

No questions assigned to the following page.

```
<ipython-input-27-855601105a83>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c

  sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3: replace na values with median values
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	4.3750	
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	1.6196	
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	2.5762	
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	4.9489	
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	3.9063	

Next steps: Generate code with `sample_incomplete_rows` | View recommended plots | New interactive sheet

Now that we've played around with this, lets finalize this approach by replacing the nulls in our final dataset

```
housing["total_bedrooms"].fillna(median, inplace=True)
```

```
<ipython-input-28-3087f14a5ecd>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c

  housing["total_bedrooms"].fillna(median, inplace=True)
```

Could you think of another plausible imputation for this dataset?

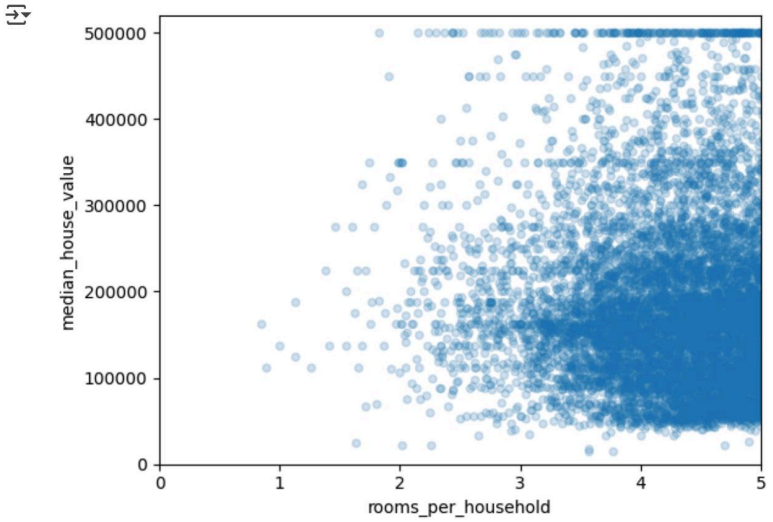
Augmenting Features

New features can be created by combining different columns from our data set.

- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

```
housing["rooms_per_household"] = housing["total_rooms"]/(housing["households"] + 1e-6)
housing["bedrooms_per_room"] = housing["total_bedrooms"]/(housing["total_rooms"] + 1e-6)
housing["population_per_household"]=housing["population"]/(housing["households"] + 1e-6)
```

```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



No questions assigned to the following page.

Dealing with Non-Numeric Data

So we're almost ready to feed our dataset into a machine learning model, but we're not quite there yet!

Generally speaking all models can only work with numeric data, which means that if you have Categorical data you want included in your model, you'll need to do a numeric conversion. We'll explore this more later, but for now we'll take one approach to converting our `ocean_proximity` field into a numeric one.

```
from sklearn.preprocessing import LabelEncoder

# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
housing['ocean_proximity'] = labelencoder.fit_transform(housing['ocean_proximity'])
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

Next steps: Generate code with `housing` | View recommended plots | New interactive sheet

Divide up the Dataset for Machine Learning

After having cleaned your dataset you're ready to train your machine learning model.

To do so you'll aim to divide your data into:

- train set
- test set

In some cases you might also have a validation set as well for tuning hyperparameters (don't worry if you're not familiar with this term yet..)

In supervised learning setting your train set and test set should contain (**feature**, **target**) tuples.

- **feature**: is the input to your model
- **target**: is the ground truth label
  - when target is categorical the task is a classification task
  - when target is floating point the task is a regression task

We will make use of **scikit-learn** python package for preprocessing.

Scikit learn is pretty well documented and if you get confused at any point simply look up the function/object!

```
from sklearn.model_selection import StratifiedShuffleSplit
# let's first start by creating our train and test sets
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    train_set = housing.loc[train_index]
    test_set = housing.loc[test_index]


housing_training = train_set.drop("median_house_value", axis=1) # drop labels for training set features
                                                                # the input to the model should not contain the true label
housing_labels = train_set["median_house_value"].copy()


housing_testing = test_set.drop("median_house_value", axis=1) # drop labels for training set features
                                                              # the input to the model should not contain the true label
housing__test_labels = test_set["median_house_value"].copy()
```

Select a model and train

Once we have prepared the dataset it's time to choose a model.

As our task is to predict the median_house_value (a floating value), regression is well suited for this.

Question assigned to the following page: 1.1

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_training, housing_labels)
```

```
▾ LinearRegression
LinearRegression()
```

```
# let's try our model on a few testing instances
data = housing_testing.iloc[:5]
labels = housing__test_labels.iloc[:5]

print("Predictions:", np.round(lin_reg.predict(data), 1))
print("Actual labels:", list(labels))
```

```
Predictions: [418197.2 305620.5 232253.  188754.6 251166.4]
Actual labels: [500001.0, 162500.0, 204600.0, 159700.0, 184000.0]
```

We can evaluate our model using certain metrics, a fitting metric for regresison is the mean-squared-loss

$$L(\hat{Y}, Y) = \frac{1}{N}\sum_{i}^{N}(\hat{y_i} - y_i)^2$$

where $\hat{y}$ is the predicted value, and y is the ground truth label.

```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(housing_testing)
mse = mean_squared_error(housing__test_labels, preds)
rmse = np.sqrt(mse)
rmse
```

```
67694.08184344426
```

Is this a good result? What do you think an acceptable error rate is for this sort of problem?

TODO: Applying the end-end ML steps to a different dataset.

Ok now it's time to get to work! We will apply what we've learnt to another dataset (airbnb dataset). For this project we will attempt to **predict the airbnb rental price based on other features in our given dataset.**

Visualizing Data

Load the data + statistics

Let's do the following set of tasks to get us warmed up:

- load the dataset
- display the first few rows of the data
- drop the following columns: name, host_id, host_name, last_review, neighbourhood
- display a summary of the statistics of the loaded data

```
import pandas as pd
airbnb =  pd.read_csv('AB_NYC_2019.csv') # we load the pandas dataframe
airbnb.head()
```

Question assigned to the following page: 1.1

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	mini
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	

Next steps: Generate code with airbnb | View recommended plots | New interactive sheet

```
airbnb_drop = airbnb.drop(["name", "host_id", "host_name", "last_review", "neighbourhood"], axis=1)
airbnb_drop
```

	id	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per
0	2539	Brooklyn	40.64749	-73.97237	Private room	149	1	9	
1	2595	Manhattan	40.75362	-73.98377	Entire home/apt	225	1	45	
2	3647	Manhattan	40.80902	-73.94190	Private room	150	3	0	
3	3831	Brooklyn	40.68514	-73.95976	Entire home/apt	89	1	270	
4	5022	Manhattan	40.79851	-73.94399	Entire home/apt	80	10	9	
...	...	...	...	...	...	...	...	...	...
48890	36484665	Brooklyn	40.67853	-73.94995	Private room	70	2	0	
48891	36485057	Brooklyn	40.70184	-73.93317	Private room	40	4	0	
48892	36485431	Manhattan	40.81475	-73.94867	Entire home/apt	115	10	0	
48893	36485609	Manhattan	40.75751	-73.99112	Shared room	55	1	0	
48894	36487245	Manhattan	40.76404	-73.98933	Private room	90	7	0	

48895 rows × 11 columns

Next steps: Generate code with airbnb_drop | View recommended plots | New interactive sheet

```
airbnb_drop.describe()
```

	id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	

Questions assigned to the following page: 1.1 and 1.2

```
airbnb_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   neighbourhood_group             48895 non-null  object 
 2   latitude                        48895 non-null  float64
 3   longitude                       48895 non-null  float64
 4   room_type                       48895 non-null  object 
 5   price                           48895 non-null  int64  
 6   minimum_nights                  48895 non-null  int64  
 7   number_of_reviews               48895 non-null  int64  
 8   reviews_per_month               38843 non-null  float64
 9   calculated_host_listings_count  48895 non-null  int64  
 10  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(6), object(2)
memory usage: 4.1+ MB
```

Some Basic Visualizations

Let's try another popular python graphics library: Plotly.

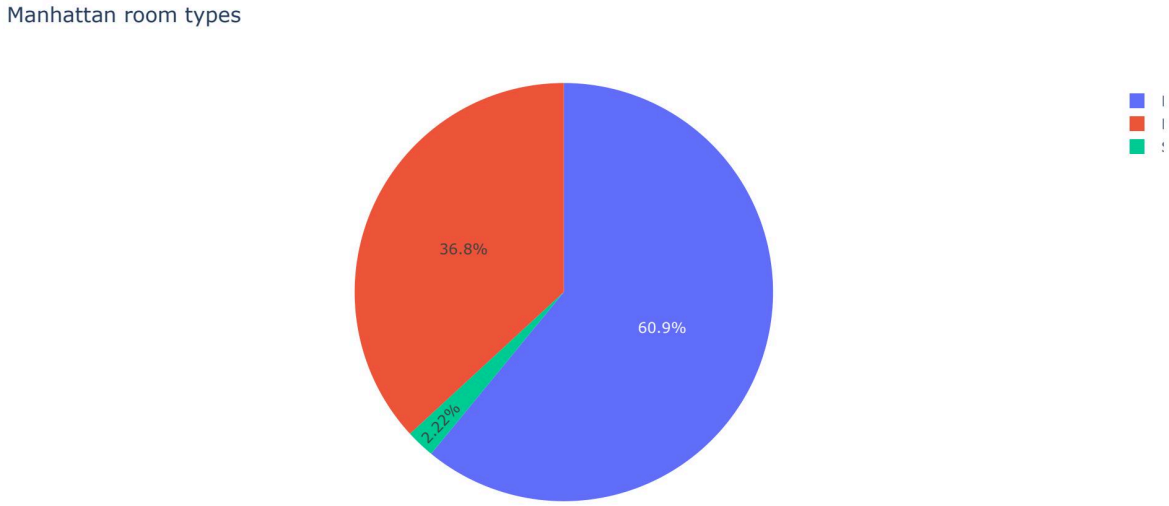
You can find documentation and all the examples you'll need here: Plotly Documentation

Let's start out by getting a better feel for the distribution of rentals in the market.

Generate a pie chart showing the distribution of room type (`room_type` in the dataset) across NYC's 'Manhattan' Buroughs (fitlered by `neighbourhood_group` in the dataset)

```
#manually checking to see that my pie chart is correct
#airbnb_drop.loc[airbnb_drop['neighbourhood_group'] == "Manhattan"]['room_type'].value_counts()
```

```
import plotly.express as px
manhattan_only = airbnb_drop.loc[airbnb_drop['neighbourhood_group'] == "Manhattan"]
fig = px.pie(manhattan_only, names='room_type', title='Manhattan room types')
fig.show()
```



Plot the total number_of_reviews per room_type

We now want to see the total number of reviews left for each room type group in the form of a Bar Chart (where the X-axis is the room type group and the Y-axis is a count of review.

This is a two step process:

1. You'll have to sum up the reviews per room type group **(hint! try using the groupby function)**

Questions assigned to the following page: 1.2 and 1.3

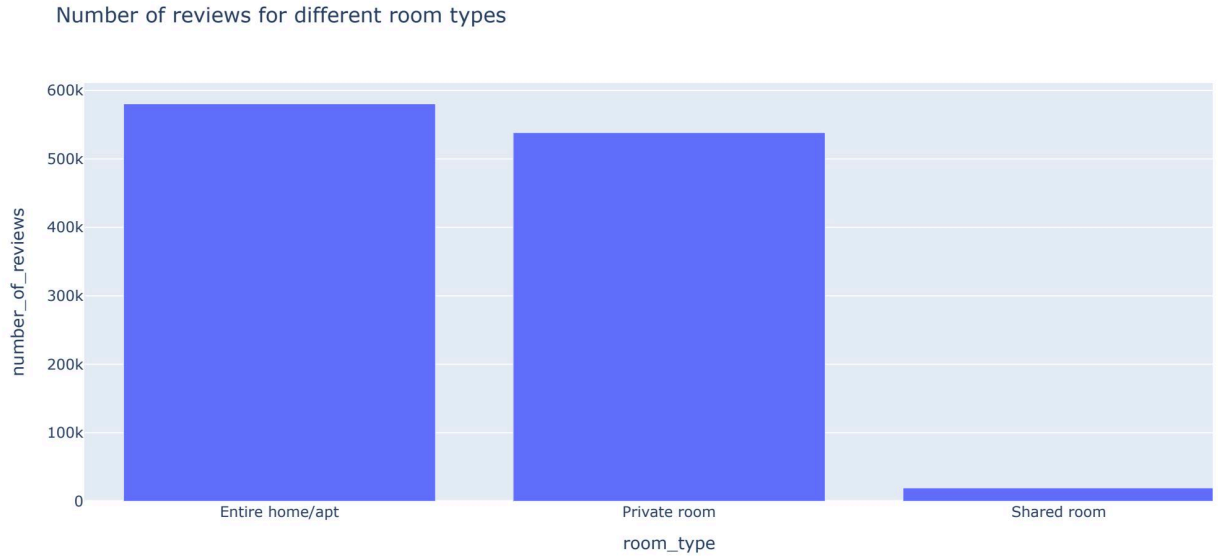
2. Then use Plotly to generate the graph

```
room = airbnb_drop.groupby('room_type')['number_of_reviews'].sum().reset_index() # WRITE YOUR CODE HERE #
room.head()
```

	room_type	number_of_reviews
0	Entire home/apt	580403
1	Private room	538346
2	Shared room	19256

Next steps: Generate code with room | View recommended plots | New interactive sheet

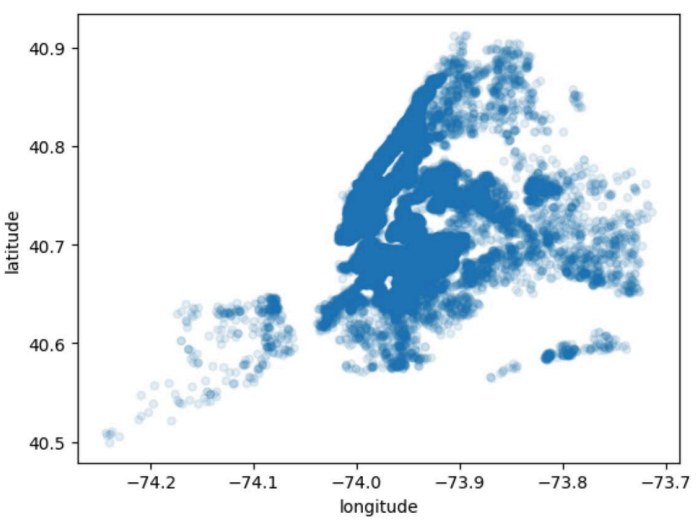
```
fig = px.bar(room, x='room_type', y='number_of_reviews', title = "Number of reviews for different room types") # WRITE YOUR
fig.show()
```



Plot a map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

For reference you can use the Matplotlib code above to replicate this graph here.

```
airbnb.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
plt.show()
```



Questions assigned to the following page: 1.2 and 1.3

```
miniairbnb = airbnb[airbnb['price'] <= 500].reset_index(drop=True)
miniairbnb = miniairbnb.sample(n = 1000, random_state = 42) # you can change n here

miniairbnb
```

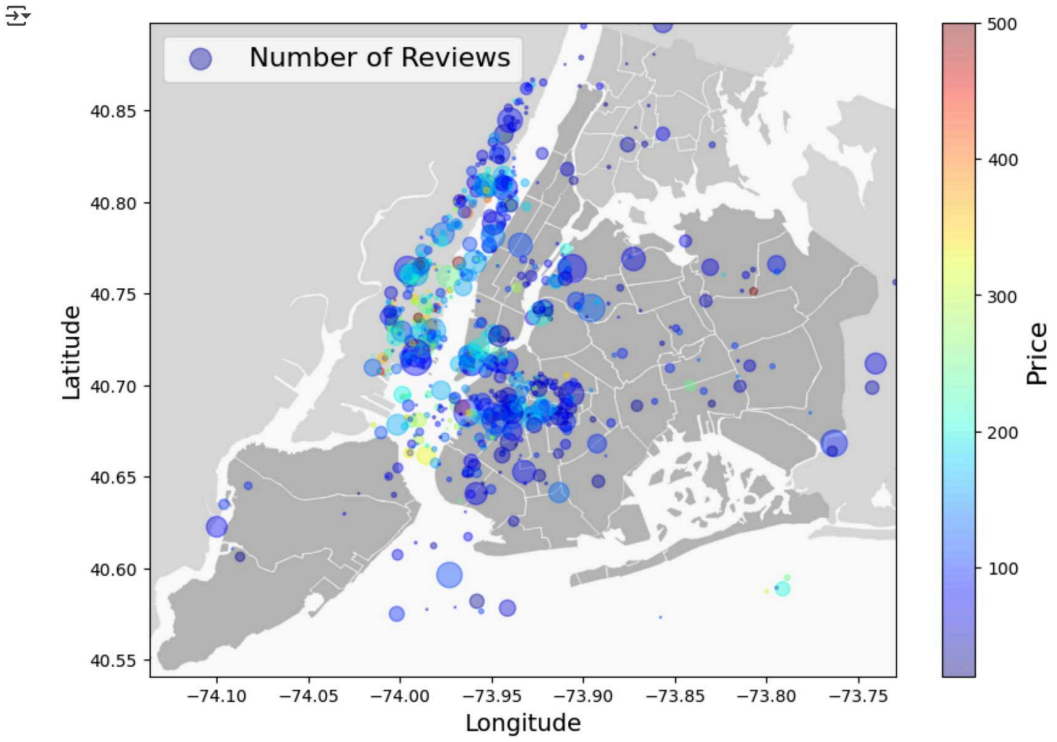
	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
14851	12088134	Spacious, bright on Upper East Side	64791940	Charles	Manhattan	Upper East Side	40.77433	-73.94854	Private room	35
17163	13762046	Private Micro Room 1	59529529	Han	Manhattan	Hell's Kitchen	40.76296	-73.99565	Private room	70
11247	8925082	Chic apt on cutest corner in Soho	33834627	Charlotte	Manhattan	Nolita	40.72217	-73.99375	Entire home/apt	200
13033	10000943	1 Bedroom in UWS Manhattan	6891770	Mido	Manhattan	Upper West Side	40.78239	-73.97581	Entire home/apt	199
29620	23277230	2 bedrooms flat near Central Park	128579948	Giampiero	Manhattan	Upper East Side	40.76646	-73.96760	Entire home/apt	500
...	...	...	...	...	...	...	...	...	...	...
24999	20387176	Professor Trelawney's Celestial Tower in Brooklyn	20540369	Aviva	Brooklyn	Bushwick	40.70031	-73.91718	Private room	55
30423	24034429	Oasis in East Village	180897611	Daniel	Manhattan	East Village	40.72642	-73.98402	Entire home/apt	175
42655	33708600	Sonder | The Biltmore | Chic 1BR + City View	219517861	Sonder (NYC)	Manhattan	Theater District	40.76008	-73.98721	Entire home/apt	227
322	79067	Lovely 3 bedroom in Italianate Brownstone w/ga...	425506	Bliss	Brooklyn	Clinton Hill	40.68480	-73.96219	Entire home/apt	350
28364	22277233	Sunny & Dreamy Bedstuy Room	105018962	Karly	Brooklyn	Bedford-Stuyvesant	40.68379	-73.92932	Private room	45

1000 rows × 16 columns

```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# load an image of New York
import matplotlib.image as mpimg
nyc_img=mpimg.imread('nyc.png', -1)


ax = miniairbnb.plot(kind="scatter",  x="longitude", y="latitude", figsize=(10,7),
                c="price", cmap=plt.get_cmap("jet"), colorbar=True, alpha=0.4,
                 s=miniairbnb['number_of_reviews'], label="Number of Reviews")

cbar = ax.collections[0].colorbar
cbar.set_label('Price', fontsize=16)

# overlay the NYC map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.

# WRITE YOUR CODE HERE #
min_longitude = miniairbnb["longitude"].min()
max_longitude = miniairbnb["longitude"].max()
min_latitude = miniairbnb["latitude"].min()
max_latitude = miniairbnb["latitude"].max()
```

Questions assigned to the following page: 1.3 and 1.4

```
plt.imshow(nyc_img, extent=[min_longitude, max_longitude, min_latitude - 0.01, max_latitude], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
# prices = miniairbnb["price"]
# current_cb = plt.gcf().axes[-1]  # Get the colorbar axis

# tick_locations = np.linspace(
#     miniairbnb['price'].min(),
#     miniairbnb['price'].max(),
#     6
# )
# tick_labels = [f"${round(v)}" for v in tick_locations]

# current_cb.set_yticklabels(tick_labels, fontsize=14)
# current_cb.set_ylabel('Price', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```

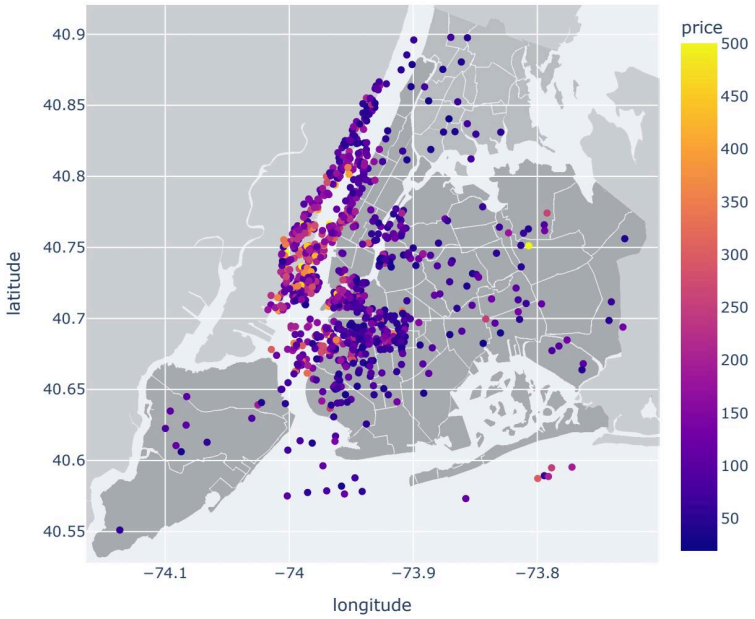


Now try to recreate this plot using Plotly's Scatterplot functionality. Note that the increased interactivity of the plot allows for some very cool functionality

```
fig = px.scatter(miniairbnb, x="longitude", y="latitude", color="price", )

import base64
#set a local image as a background
image_filename = 'nyc.png'
plotly_logo = base64.b64encode(open(image_filename, 'rb').read()).decode()

# WRITE YOUR CODE HERE #
fig.update_layout(
    width=645,  # Set a smaller width for the scatter plot to fit the map
    height=600,
    images=[dict(
        source="" + plotly_logo,  # Embed the image as base64
        xref="paper",
        yref="paper",
        x=0,
        y=1,
        sizex=4,
        sizey = 1,
        opacity=0.5,
        layer="below"
    )]
```

Questions assigned to the following page: 1.4 and 1.5

```
)

fig.show()
```



Use Plotly to plot the average price of room types in Brooklyn who have at least 10 Reviews.

Like with the previous example you'll have to do a little bit of data engineering before you actually generate the plot.

Generally I'd recommend the following series of steps:

1. Filter the data by neighborhood group and number of reviews to arrive at the subset of data relevant to this graph.
2. Groupby the room type
3. Take the mean of the price for each roomtype group
4. FINALLY (seriously!?!?) plot the result

```
# WRITE YOUR CODE HERE #
subgroup = airbnb[airbnb['neighbourhood_group'] == "Brooklyn"]
subgroup = subgroup[subgroup['number_of_reviews'] >= 10]
subgroup = subgroup.groupby('room_type')['price'].mean().reset_index()
```

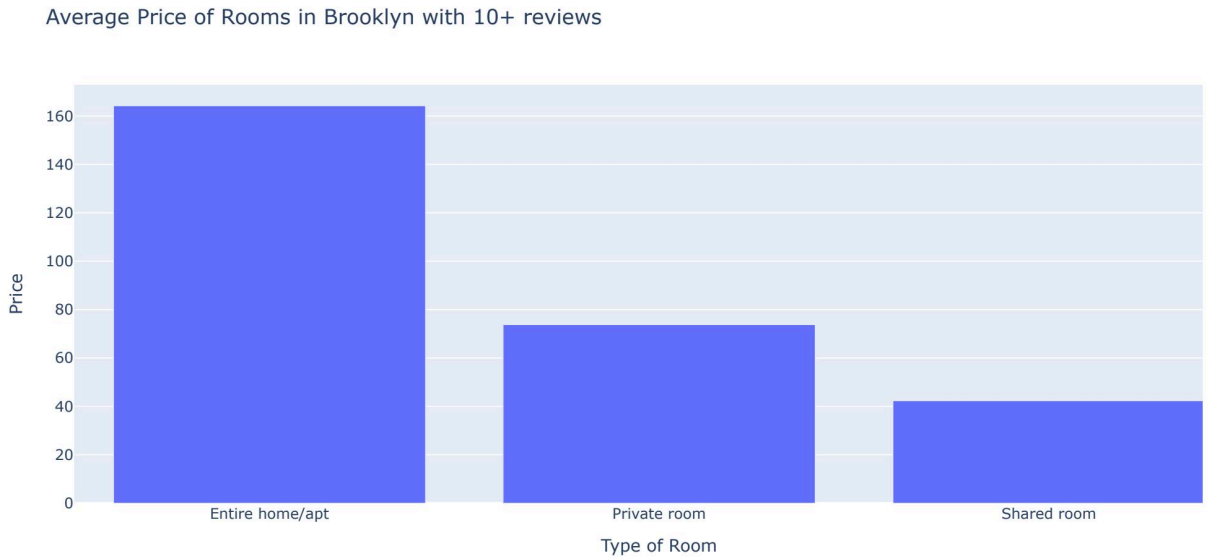
```
subgroup.head()
```

	room_type	price
0	Entire home/apt	164.191258
1	Private room	73.743515
2	Shared room	42.291667

Next steps: Generate code with subgroup | View recommended plots | New interactive sheet

```
# WRITE YOUR CODE HERE #
fig = px.bar(subgroup, x='room_type', y='price', title="Average Price of Rooms in Brooklyn with 10+ reviews", labels={'room_
fig.show()
```

Questions assigned to the following page: 2.1 and 1.5



Prepare the Data

```
airbnb_drop.head()
```

	id	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month
0	2539	Brooklyn	40.64749	-73.97237	Private room	149	1	9	0.21
1	2595	Manhattan	40.75362	-73.98377	Entire home/apt	225	1	45	0.38
2	3647	Manhattan	40.80902	-73.94190	Private room	150	3	0	NaN
3	3831	Brooklyn	40.68514	-73.95976	Entire home/apt	89	1	270	4.64
4	5022	Manhattan	40.79851	-73.94399	Entire home/apt	80	10	9	0.10

Next steps: Generate code with airbnb_drop | View recommended plots | New interactive sheet

Feature Engineering

Let's create a new binned feature, `price_cat` that will divide our dataset into quintiles (1-5) in terms of price level (you can choose the levels to assign)

Do a value count to check the distribution of values

```
# assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
airbnb_drop["price_cat"] = pd.qcut(airbnb_drop["price"], 5, labels=[1, 2, 3, 4, 5])

airbnb_drop["price_cat"].value_counts()
```

Questions assigned to the following page: 2.1, 2.2, and 2.3

price_cat	count
4	10809
1	10063
2	9835
3	9804
5	8384

dtype: int64

Data Imputation

Determine if there are any null-values and impute them.

```
airbnb_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype   
---  ------                          --------------  -----   
 0   id                              48895 non-null  int64   
 1   neighbourhood_group             48895 non-null  object  
 2   latitude                        48895 non-null  float64 
 3   longitude                       48895 non-null  float64 
 4   room_type                       48895 non-null  object  
 5   price                           48895 non-null  int64   
 6   minimum_nights                  48895 non-null  int64   
 7   number_of_reviews               48895 non-null  int64   
 8   reviews_per_month               38843 non-null  float64 
 9   calculated_host_listings_count  48895 non-null  int64   
 10  availability_365                48895 non-null  int64   
 11  price_cat                       48895 non-null  category
dtypes: category(1), float64(3), int64(6), object(2)
memory usage: 4.2+ MB
```

```
airbnb_drop = airbnb_drop.dropna()
airbnb_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38843 entries, 0 to 48852
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype   
---  ------                          --------------  -----   
 0   id                              38843 non-null  int64   
 1   neighbourhood_group             38843 non-null  object  
 2   latitude                        38843 non-null  float64 
 3   longitude                       38843 non-null  float64 
 4   room_type                       38843 non-null  object  
 5   price                           38843 non-null  int64   
 6   minimum_nights                  38843 non-null  int64   
 7   number_of_reviews               38843 non-null  int64   
 8   reviews_per_month               38843 non-null  float64 
 9   calculated_host_listings_count  38843 non-null  int64   
 10  availability_365                38843 non-null  int64   
 11  price_cat                       38843 non-null  category
dtypes: category(1), float64(3), int64(6), object(2)
memory usage: 3.6+ MB
```

Numeric Conversions

Finally, review what features in your dataset are non-numeric and convert them.

Questions assigned to the following page: 2.4 and 2.3

```
# WRITE YOUR CODE HERE #
from sklearn.preprocessing import LabelEncoder

#airbnb_drop.info()
le = LabelEncoder()

airbnb_drop['neighbourhood_group'] = le.fit_transform(airbnb_drop['neighbourhood_group'])
airbnb_drop['room_type'] = le.fit_transform(airbnb_drop['room_type'])
airbnb_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38843 entries, 0 to 48852
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              38843 non-null  int64
 1   neighbourhood_group             38843 non-null  int64
 2   latitude                        38843 non-null  float64
 3   longitude                       38843 non-null  float64
 4   room_type                       38843 non-null  int64
 5   price                           38843 non-null  int64
 6   minimum_nights                  38843 non-null  int64
 7   number_of_reviews               38843 non-null  int64
 8   reviews_per_month               38843 non-null  float64
 9   calculated_host_listings_count  38843 non-null  int64
 10  availability_365                38843 non-null  int64
 11  price_cat                       38843 non-null  category
dtypes: category(1), float64(3), int64(8)
memory usage: 3.6 MB
<ipython-input-57-d014996aacd4>:7: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-

<ipython-input-57-d014996aacd4>:8: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
```

Prepare Data for Machine Learning

Using our `StratifiedShuffleSplit` function example from above, let's split our data into a 80/20 Training/Testing split using `price_cat` to partition the dataset

```
airbnb_drop = airbnb_drop.reset_index(drop=True)

from sklearn.model_selection import StratifiedShuffleSplit
# let's first start by creating our train and test sets
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(airbnb_drop, airbnb_drop["price_cat"]):
    train_set = airbnb_drop.loc[train_index]
    test_set = airbnb_drop.loc[test_index]


test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7769 entries, 19003 to 4113
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              7769 non-null   int64
 1   neighbourhood_group             7769 non-null   int64
 2   latitude                        7769 non-null   float64
 3   longitude                       7769 non-null   float64
 4   room_type                       7769 non-null   int64
 5   price                           7769 non-null   int64
 6   minimum_nights                  7769 non-null   int64
 7   number_of_reviews               7769 non-null   int64
 8   reviews_per_month               7769 non-null   float64
 9   calculated_host_listings_count  7769 non-null   int64
 10  availability_365                7769 non-null   int64
 11  price_cat                       7769 non-null   category
dtypes: category(1), float64(3), int64(8)
memory usage: 736.1 KB
```

Questions assigned to the following page: 2.4 and 3

Finally, remove your labels `price` and `price_cat` from your testing and training cohorts, and create separate label features.

```
airbnb_training = train_set.drop(["price_cat", "price"], axis=1) # drop labels for training set features
                                                                 # the input to the model should not contain the true label
airbnb_labels_cat = train_set["price_cat"].copy()
airbnb_labels = train_set["price"].copy()
```

```
airbnb_testing = test_set.drop(["price_cat", "price"], axis=1) # drop labels for test set features
                                                               # the input to the model should not contain the true label
airbnb_test_labels_cat = test_set["price_cat"].copy()
airbnb_test_labels = test_set["price"].copy()
```

```
airbnb_training.head()
```

	id	neighbourhood_group	latitude	longitude	room_type	minimum_nights	number_of_reviews	reviews_per_month
22913	21610946	1	40.67411	-73.96532	1	1	55	2.76
20696	19913070	1	40.69723	-73.93769	0	3	29	1.22
10021	9049582	2	40.77370	-73.95616	2	1	17	0.38
38479	35681857	1	40.68884	-73.97330	0	3	1	1.00
4308	3322922	1	40.65320	-73.96216	0	18	11	0.20

Next steps: Generate code with `airbnb_training` View recommended plots New interactive sheet

Fit a linear regression model

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using **MSE**. Provide both **test and train set MSE values**.

```
# WRITE YOUR CODE HERE #
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(airbnb_training, airbnb_labels)
```

```
▾ LinearRegression
LinearRegression()
```

```
# let's try our model on a few testing instances
data = airbnb_testing.iloc[:5]
labels = airbnb_test_labels.iloc[:5]

print("Predictions:", np.round(lin_reg.predict(data), 1))
print("Actual labels:", list(labels))
```

```
Predictions: [107.3  66.2  51.2 199.4  48.4]
Actual labels: [100, 43, 49, 120, 60]
```

```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(airbnb_testing)
mse = mean_squared_error(airbnb_test_labels, preds)
rmse = np.sqrt(mse)
rmse
```

```
164.05379721066774
```

Also doing it for price category

```
from sklearn.linear_model import LinearRegression

lin_reg_cat = LinearRegression()
lin_reg_cat.fit(airbnb_training, airbnb_labels_cat)

# let's try our model on a few testing instances
data = airbnb_testing.iloc[:5]
```

Question assigned to the following page: 3

```
labels = airbnb_test_labels_cat.iloc[:5]

print("Predictions:", np.round(lin_reg_cat.predict(data), 1)
```

```
Predictions: [2.3 1.8 1.4 3.9 1.5]
Actual labels: [3, 1, 1, 3, 1]
```

```
from sklearn.metrics import mean_squared_error

preds = lin_reg_cat.predict(airbnb_testing)
mse = mean_squared_error(airbnb_test_labels_cat, preds)
rmse = np.sqrt(mse)
rmse
```

```
0.9419261720105154
```

```
# save as pdf
%%capture
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
from google.colab import drive
drive.mount('/content/drive')
!cp drive/My Drive/Colab Notebooks/CSM148_Project_1_W24_TODO.ipynb ./
!jupyter nbconvert --to PDF "pdffff"
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('CSM148_Project_1_W24_TODO.ipynb')
```

Start coding or generate with AI.

Start coding or generate with AI.