Introduction

Welcome to **CSM148 - Data Science!** We plan on having you go through some grueling training so you can start crunching data out there... in today's day and age "data is the new oil" or perhaps "snake oil" nonetheless, there's a lot of it, each with different purity (so pure that perhaps you could feed off it for a life time) or dirty which then at that point you can either decide to dump it or try to weed out something useful (that's where they need you...)

In this project you will work through an example project end to end.

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 model and train
5. Does it meet the requirements? Fine tune the model

steps

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

In [407...

```
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
import os
%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

# Where to save the figures
ROOT_DIR = "."
IMAGES_PATH = os.path.join(ROOT_DIR, "images")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_name, tight_layout=True, fig_extension="png", resolution=300):
    '''
        plt.savefig wrapper. refer to
        https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html
    '''
    path = os.path.join(IMAGES_PATH, fig_name + "." + fig_extension)
    print("Saving figure", fig_name)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [408…

```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

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

In [409…

```
import pandas as pd

def load_housing_data(housing_path):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [410…

```
housing = load_housing_data(DATASET_PATH) # we load the pandas dataframe
housing.head(5) # show the first five rows of the dataframe
                # typically this is the first thing you do
                # to see how the dataframe looks like
```

Out[410…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mec
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

A dataset may have different types of features

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

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.

In [411…

```
# 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
 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
```

In [412…

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

Out[412…

```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

In [413…

```
# to access a particular row we can use iloc
```

```
housing.iloc[1]
```

Out[413…

```
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
Name: 1, dtype: object
```

In [414…

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

Out[414…

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [415…

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

Out[415…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	(

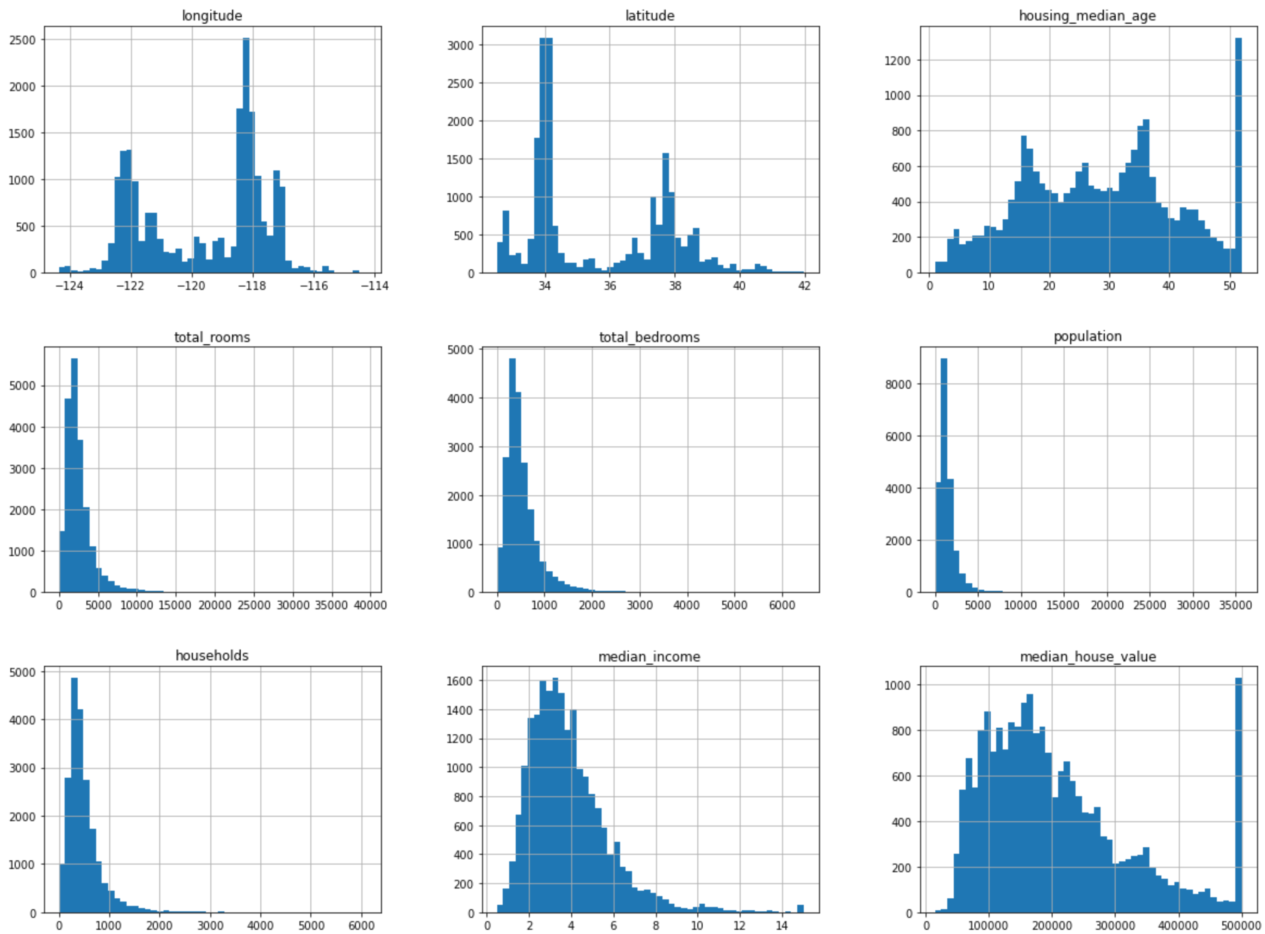
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

Let's start visualizing the dataset

In [416…

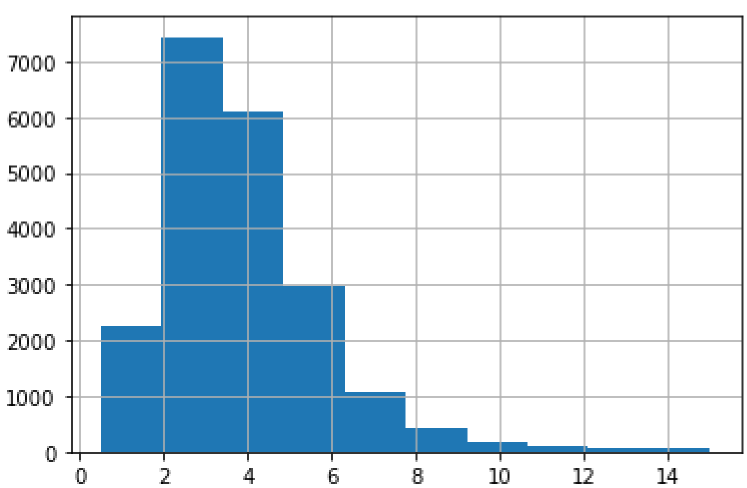
```
# We can draw a histogram for each of the dataframes features
# using the hist function
housing.hist(bins=50, figsize=(20,15))
# save_fig("attribute_histogram_plots")
```

```
plt.show() # pandas internally uses matplotlib, and to display all the figures
           # the show() function must be called
```



In [417…

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



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

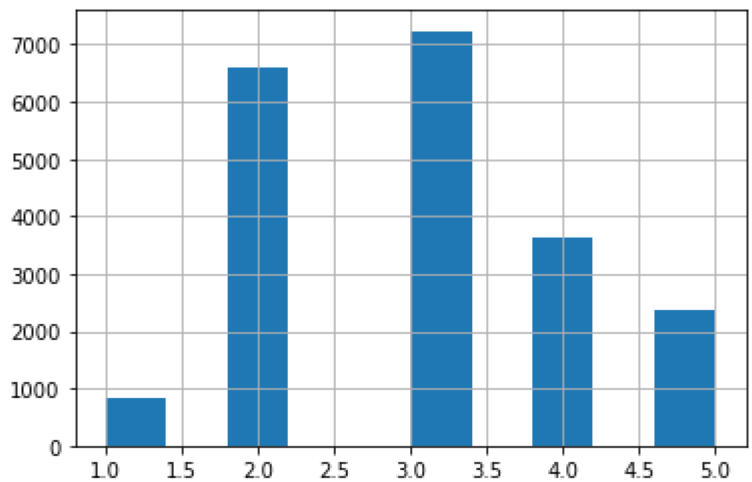
For example, to bin the households based on median_income we can use the pd.cut function

In [418…
```
# 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()
```

Out[418…
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

In [419…
```
housing["income_cat"].hist()
```

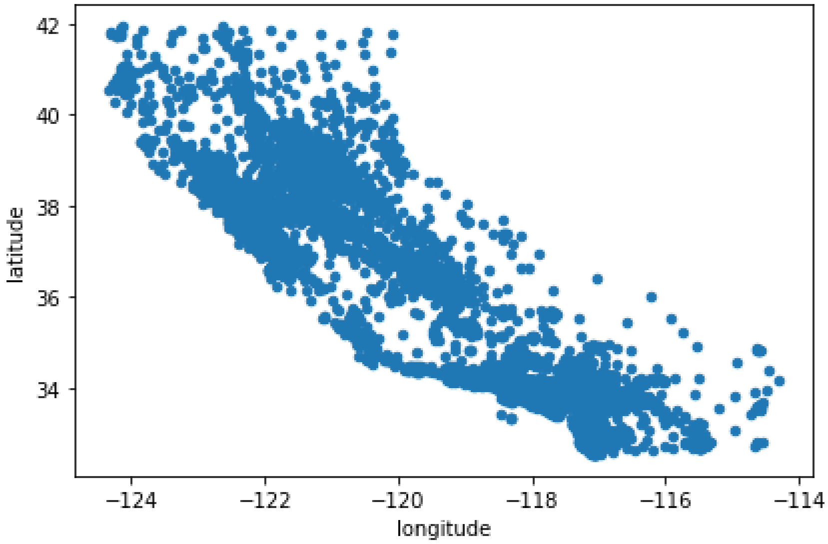
Out[419…
```
<AxesSubplot:>
```



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

In [420…
```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

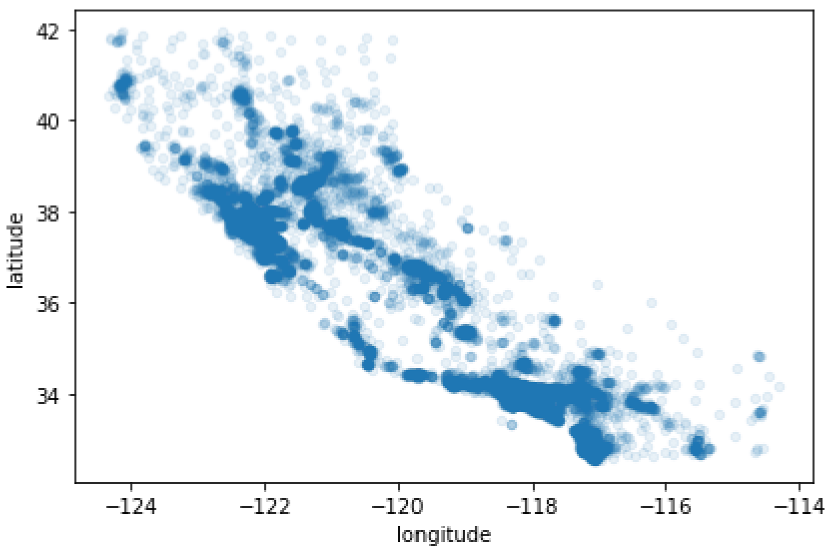
```
Saving figure bad_visualization_plot
```



In [421…

```
# 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)
save_fig("better_visualization_plot")
```

Saving figure better_visualization_plot

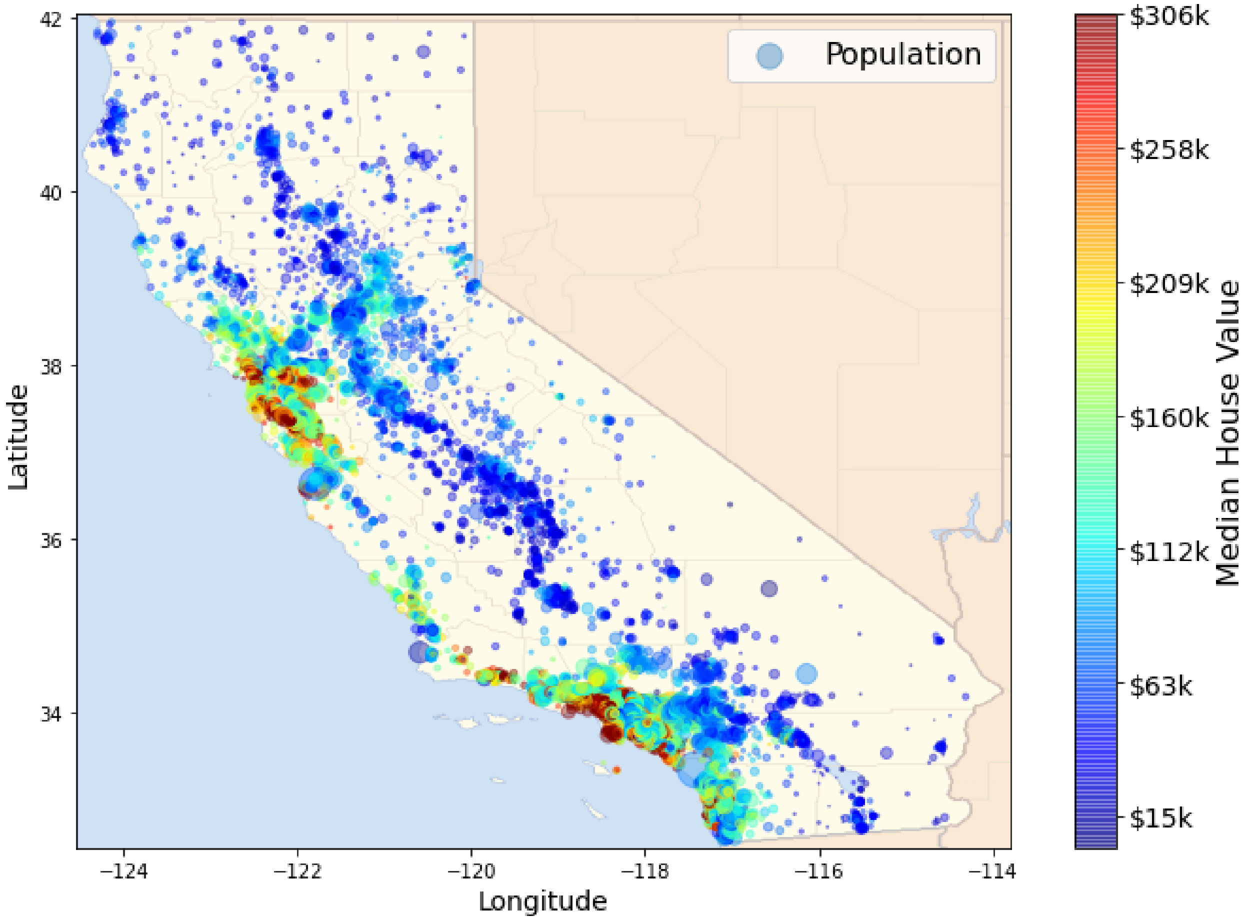


In [422…

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

# load an image of california
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "california.png"

import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
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()
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)
save_fig("california_housing_prices_plot")
plt.show()
```

```
<ipython-input-422-30a6f1a2327a>:28: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
Saving figure california_housing_prices_plot
```



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.

In [423…

```
corr_matrix = housing.corr()
```

In [424…

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

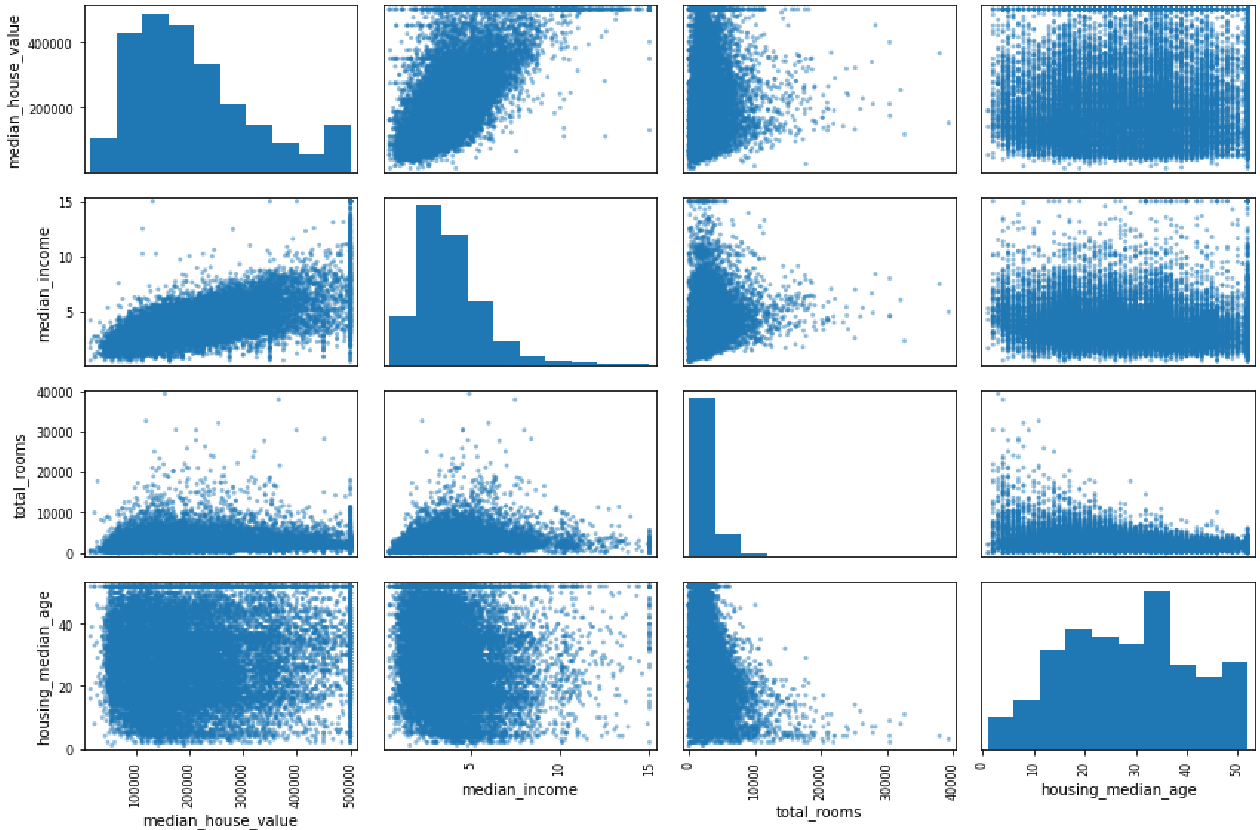
Out[424…

```
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
Name: median_house_value, dtype: float64
```

In [425…

```
# 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))
save_fig("scatter_matrix_plot")
```

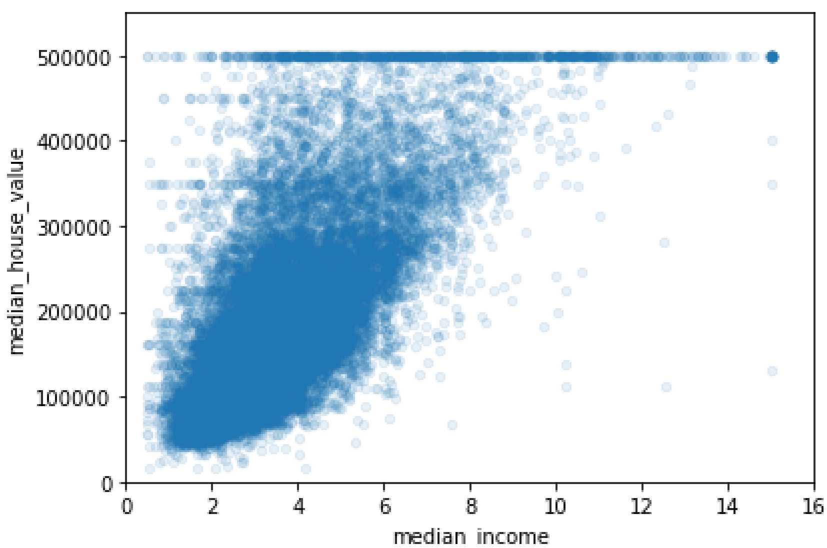
```
Saving figure scatter_matrix_plot
```



In [426…

```
# 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])
save_fig("income_vs_house_value_scatterplot")
```

```
Saving figure income_vs_house_value_scatterplot
```



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.

In [427…

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

In [428…

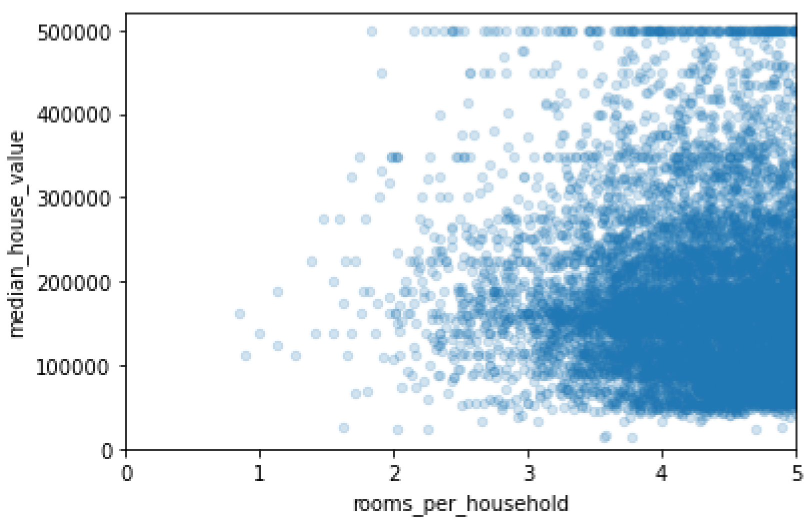
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[428…

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

In [429…

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



In [430…

```
housing.describe()
```

Out[430…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000

Preparing Dastaset for ML

Once we've visualized the data, and have a certain understanding of how the data looks like. It's time to clean!

Most of your time will be spent on this step, although the datasets used in this project are relatively nice and clean... it could get real dirty.

After having cleaned your dataset you're aiming for:

- 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!

In [431…

```
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]
```

In [432…

```
housing = train_set.drop("median_house_value", axis=1) # drop labels for training set f
                                                        # the input to the model should
housing_labels = train_set["median_house_value"].copy()
test_labels = test_set["median_house_value"].copy()
print(housing.shape)
print(housing_labels.shape)
```

```
print(test_set.shape)
print(test_labels.shape)
```

```
(16512, 13)
(16512,)
(4128, 14)
(4128,)
```

Dealing With Incomplete Data

In [433…

```
# 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
```

Out[433…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0

In [434…

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply drop row
```

Out[434…

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	medi

In [435…

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

Out[435…

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662

In [436…

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

Out[436…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0

Could you think of another plausible imputation for this dataset? (Not graded)

Prepare Data

In [437…

```
# This cell implements the complete pipeline for preparing the data
# using sklearns TransformerMixins
# Earlier we mentioned different types of features: categorical, and floats.
# In the case of floats we might want to convert them to categories.
# On the other hand categories in which are not already represented as integers must be
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot vectors or si
# Here we encode them using one hot vectors.

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median") # use median imputation for missing values
housing_num = housing.drop("ocean_proximity", axis=1) # remove the categorical feature
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
    housing["population_per_household"]=housing["population"]/housing["households"]
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
```

```
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
test_set.drop("median_house_value", axis=1, inplace=True)
```

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.

In [438…

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

# let's try the full preprocessing pipeline on a few training instances
data = housing.iloc[:5]
labels = housing_labels.iloc[:5]
data_prepared = full_pipeline.transform(data)

print("Predictions:", lin_reg.predict(data_prepared))
print("Actual labels:", list(labels))
```

```
Predictions: [200860.48973484 325527.93559759 201882.47991703  54956.04539331
 188116.26928254]
Actual labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

We can evaluate our model using certain metrics, one possible metric for regression is the mean absolute error

$$\mathrm{MAE} = \frac{\sum_i^n |\hat{y}_i - y_i|}{n}$$

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

In [439…

```
from sklearn.metrics import mean_absolute_error

preds = lin_reg.predict(housing_prepared)
rmse = mean_absolute_error(housing_labels, preds)
print(rmse)

housing_prepared_test = full_pipeline.transform(test_set)
preds_test = lin_reg.predict(housing_prepared_test)
mae = mean_absolute_error(test_labels, preds_test)
print(mae)
```

```
49145.9385616408
49183.434373798606
```

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

We will apply what we've learnt to another dataset (airbnb dataset). We will predict airbnb price based on other features.

[25 pts] Visualizing Data

[5 pts] Load the data + statistics

- load the dataset
- display the first 10 rows of the data
- drop the following columns: name, host_name, last_review
- display a summary of the statistics of the loaded data
- plot histograms for 3 features of your choice

In [440…

```
import os
import pandas as pd
DATASET_PATH = os.path.join("datasets", "airbnb")

def load_airbnb_data(airbnb_path):
    csv_path = os.path.join(airbnb_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
```

In [441…

```
airbnb_loaded = load_airbnb_data(DATASET_PATH)
airbnb_loaded.head(10)
```

Out[441…

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longituc
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972:
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longituc
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.959
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.943
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.74767	-73.975
6	5121	BlissArtsSpace!	7356	Garon	Brooklyn	Bedford-Stuyvesant	40.68688	-73.955
7	5178	Large Furnished Room Near B'way	8967	Shunichi	Manhattan	Hell's Kitchen	40.76489	-73.984
8	5203	Cozy Clean Guest Room - Family Apt	7490	MaryEllen	Manhattan	Upper West Side	40.80178	-73.967
9	5238	Cute & Cozy Lower East Side 1 bdrm	7549	Ben	Manhattan	Chinatown	40.71344	-73.990

In [442...

```
airbnb = airbnb_loaded.drop(['name','host_name','last_review'], axis=1)
airbnb.info()
airbnb.head()
```

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

```
dtypes: float64(3), int64(7), object(3)
memory usage: 4.8+ MB
```

Out[442…

	id	host_id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimu
0	2539	2787	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	2845	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	3647	4632	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	3831	4869	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	7192	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

In [443…

```
airbnb.describe()
```

Out[443…

	id	host_id	latitude	longitude	price	minimum_nights	numbe
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	4
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	

In [444…

```
# Loaded data: Concise summary of data types, null values, and counts
airbnb_loaded.info()
# Loaded data: typical statistics of columns
airbnb_loaded.describe()
```

```
<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
```

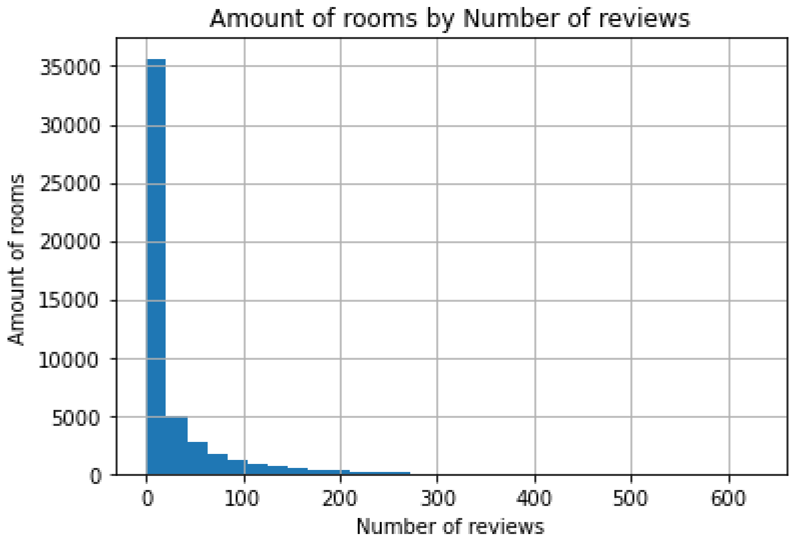
```
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 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
```

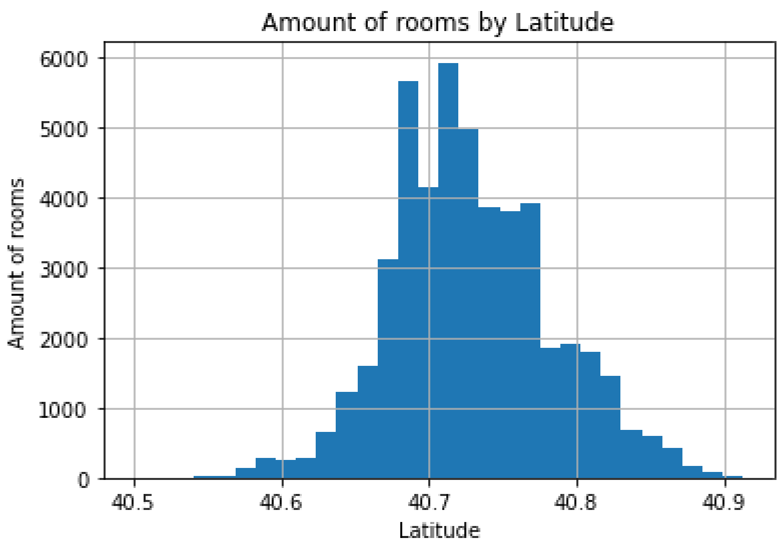
Out[444…

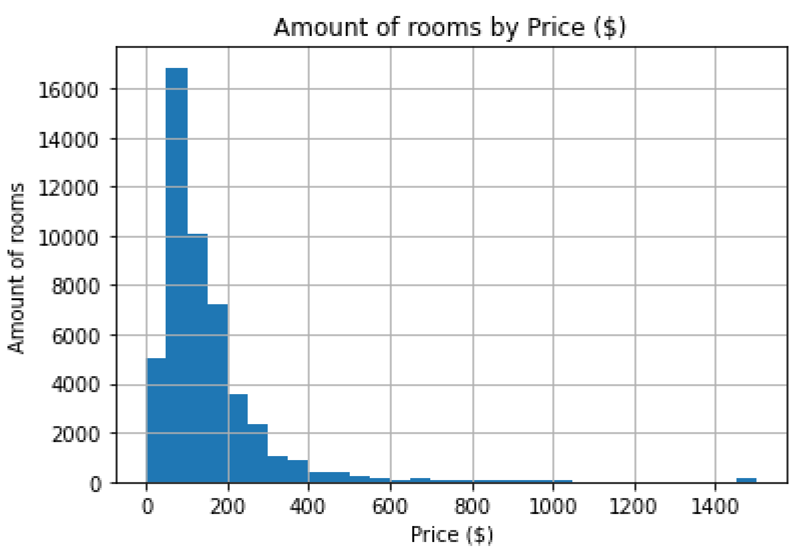
	id	host_id	latitude	longitude	price	minimum_nights	numbe
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	4
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	

In [445…

```
import matplotlib.pyplot as plt

# 3 histograms of choice: number_of_reviews, latitude, host_id
airbnb["number_of_reviews"].hist(bins=30)
plt.title("Amount of rooms by Number of reviews")
plt.xlabel("Number of reviews")
plt.ylabel("Amount of rooms")
plt.show()

airbnb["latitude"].hist(bins=30)
plt.title("Amount of rooms by Latitude")
plt.xlabel("Latitude")
plt.ylabel("Amount of rooms")
plt.show()

# Limit rooms with price > $1500 to one bin
np.clip(airbnb["price"], 0, 1500).hist(bins=30)
plt.title("Amount of rooms by Price ($)")
plt.xlabel("Price ($)")
plt.ylabel("Amount of rooms")
plt.show()
```





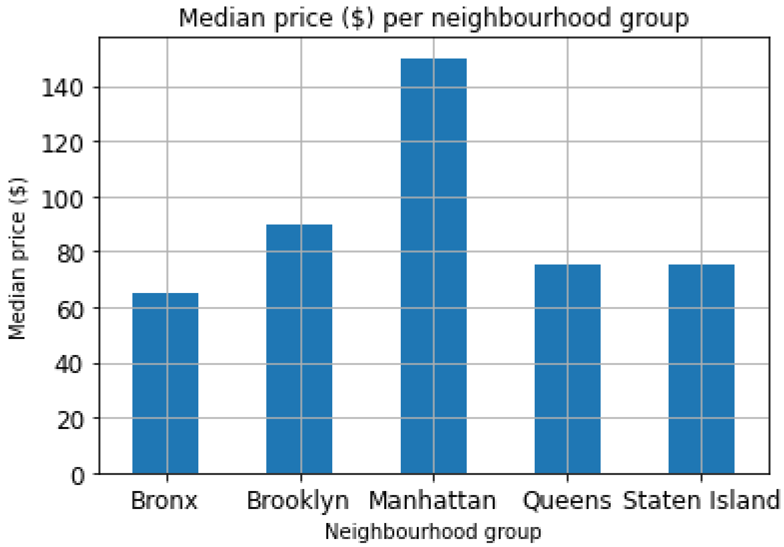


[5 pts] Plot median price per neighbourhood_group

In [446…

```
price_vs_group = airbnb.groupby(['neighbourhood_group'])['price']
price_vs_group.median().plot(kind="bar",
                             title='Median price ($) per neighbourhood group',
                             ylabel= "Median price ($)",
```

```
                                xlabel="Neighbourhood group",
                                fontsize=12, grid=True, rot=0)
plt.show()
```

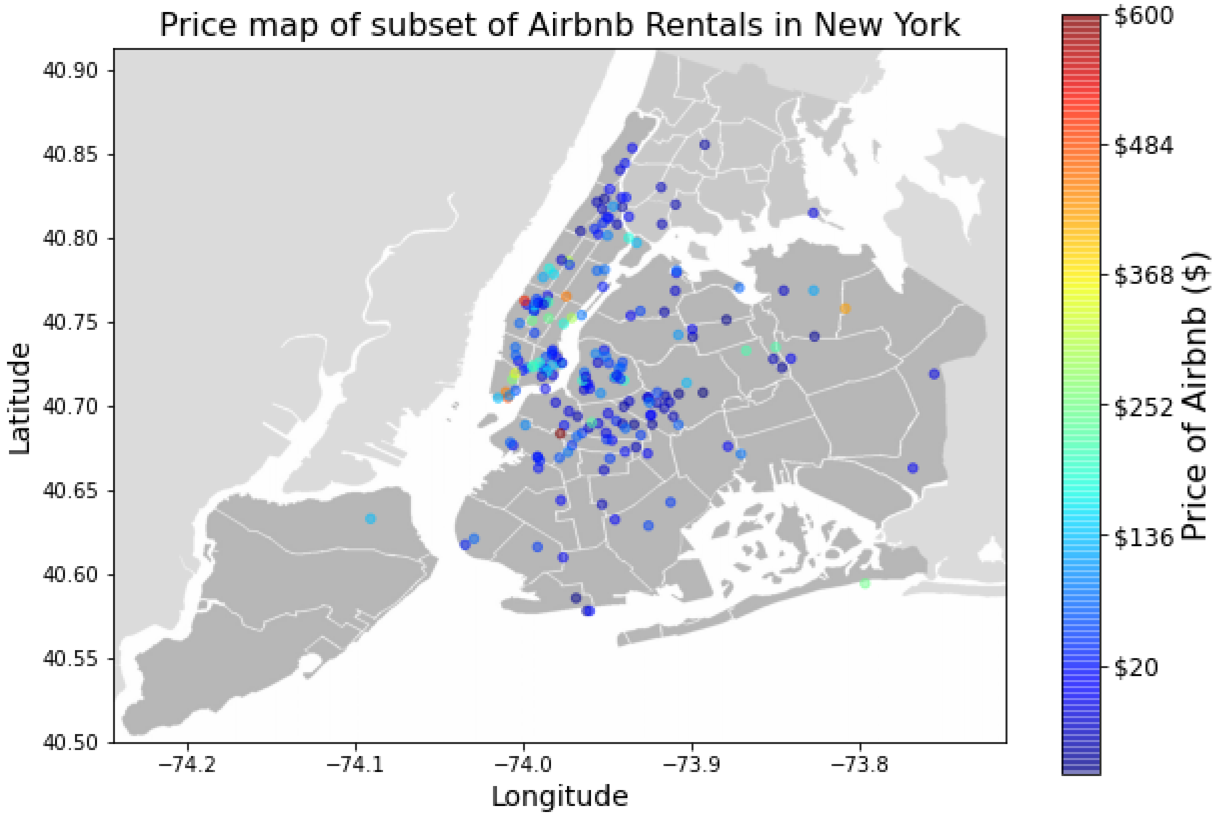


[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

In [447…

```
import matplotlib.image as mpimg
import numpy as np

images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "newyork.png"
ny_img=mpimg.imread(os.path.join(images_path, filename))

subset = airbnb.sample(frac=0.004) # randomly selected
ax = subset.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                      c="price", cmap=plt.get_cmap("jet"),
                      colorbar=False, alpha=0.5
                     )
plt.imshow(ny_img, extent=[airbnb["longitude"].min(),airbnb["longitude"].max(),
            airbnb["latitude"].min(),airbnb["latitude"].max()], alpha=0.5,
            cmap=plt.get_cmap("jet"))

plt.title("Price map of subset of Airbnb Rentals in New York", fontsize=16)
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = subset["price"]
tick_values = np.linspace(prices.min(), prices.max(), 6)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%d"%v for v in tick_values], fontsize=12)
cb.set_label('Price of Airbnb ($)', fontsize=16)

plt.show()
```

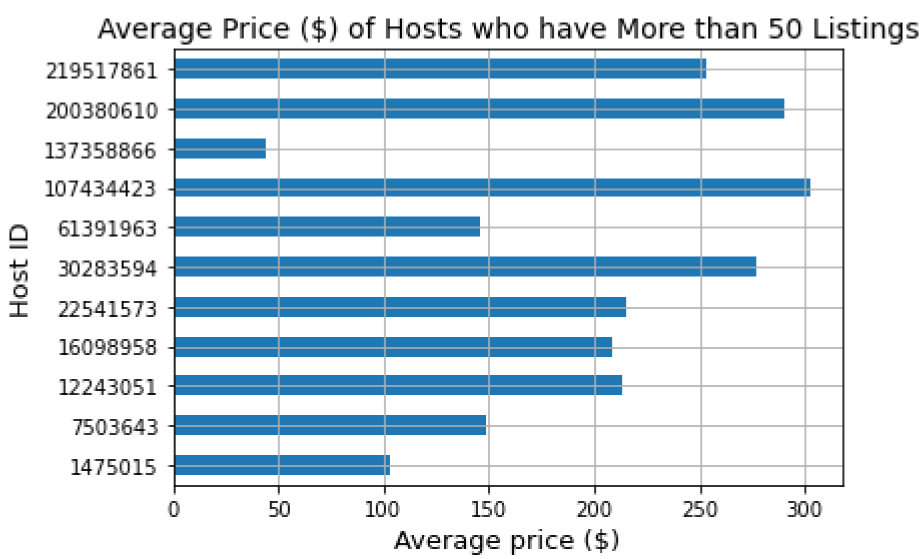
```
<ipython-input-447-f41eed2a86b3>:25: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  cb.ax.set_yticklabels(["$%d"%v for v in tick_values], fontsize=12)
```



[5 pts] Plot average price of hosts (host_id) who have more than 50 listings.

In [448…

```
hosts_more_than_50 = airbnb[airbnb['calculated_host_listings_count'] > 50]
avg_price = hosts_more_than_50.groupby(['host_id']).mean()['price']

avg_price.plot(kind='barh',grid=True)
plt.xlabel('Average price ($)', fontsize=13)
plt.ylabel('Host ID', fontsize=13)
plt.title("Average Price ($) of Hosts who have More than 50 Listings", fontsize=14)
plt.show()
```

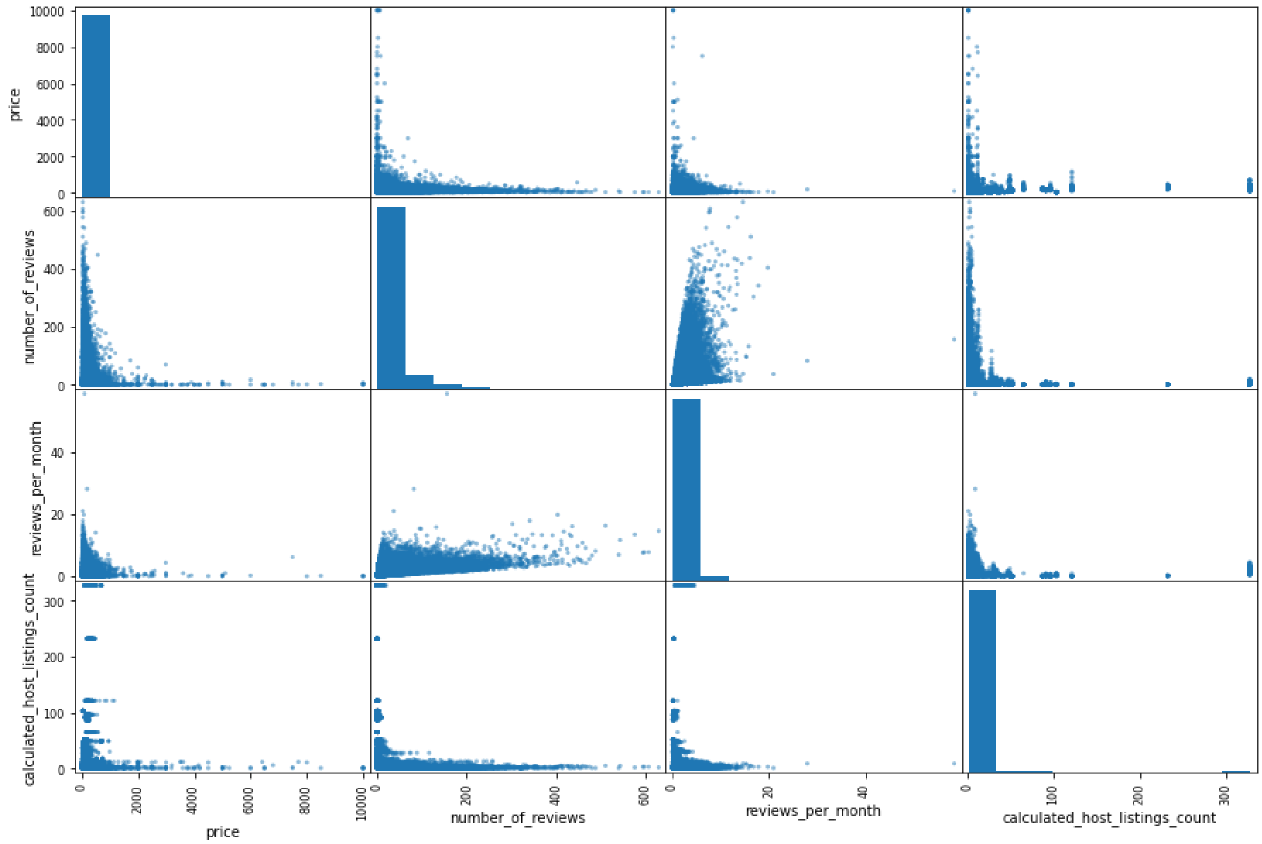


[5 pts] Plot correlation matrix

- which features have positive correlation?
- which features have negative correlation?

In [449…

```
abnb_corr = airbnb.corr()
from pandas.plotting import scatter_matrix
attributes = ["price", "number_of_reviews",
              "reviews_per_month", "calculated_host_listings_count"]
scatter_matrix(airbnb[attributes], figsize=(15, 10))
plt.show()
```



In [450…

```
print(abnb_corr["price"])
print(abnb_corr["number_of_reviews"])
print(abnb_corr["calculated_host_listings_count"])
```

```
id                                0.010619
host_id                           0.015309
latitude                          0.033939
longitude                        -0.150019
price                             1.000000
minimum_nights                    0.042799
number_of_reviews                -0.047954
reviews_per_month                -0.030608
calculated_host_listings_count    0.057472
availability_365                  0.081829
Name: price, dtype: float64
id                               -0.319760
host_id                          -0.140106
latitude                         -0.015389
longitude                         0.059094
price                            -0.047954
minimum_nights                   -0.080116
number_of_reviews                 1.000000
```

```
reviews_per_month                 0.549868
calculated_host_listings_count   -0.072376
availability_365                  0.172028
Name: number_of_reviews, dtype: float64
id                                0.133272
host_id                           0.154950
latitude                          0.019517
longitude                        -0.114713
price                             0.057472
minimum_nights                    0.127960
number_of_reviews                -0.072376
reviews_per_month                -0.009421
calculated_host_listings_count    1.000000
availability_365                  0.225701
Name: calculated_host_listings_count, dtype: float64
```

`price` and `number_of_reviews` have a negative correlation, with coefficient `-0.04795423` . This can be attributed to the fact that when price increases, less people will want to rent this listing, so there would be less reviews.

`price` and `reviews_per_month` have a negative correlation, with coefficient `-0.03060835` . This correlation is similar to that of 'price' and 'number_of_reviews'.

`price` and `calculated_host_listings_count` have a positive correlation, with coefficient `0.05747169` . This positive correlation might be because as a host has more listings, they can increase the price of some of their listings while still having some of their other cheaper listings be rented. A host with only one listing might want to keep the price lower in comparison to other listings so they can ensure it gets rented.

`number_of_reviews` and `reviews_per_month` have a positive correlation, with coefficient `0.54986750` . This is a positive correlation because it is counting the same amount: reviews. As the number of total reviews increase, then so do the reviews per month.

`number_of_reviews` and `calculated_host_listings_count` have a negative correlation, with coefficient `-0.07237606` . This correlation might be because if a host has a a high number of listings, then some listings simply won't be as popular as their other listings, and thus they would get less reviews.

`calculated_host_listings_count` and `reviews_per_month` have a negative correlation, with coefficient `-0.00942116` . This correlation is the weakest of them all, but it has a similar reason to the previous correlation.

Really, the only correlation worth noting is the correlation between number of reviews and reviews per month: `0.54986750` . The other correlations are really too weak to even discuss.

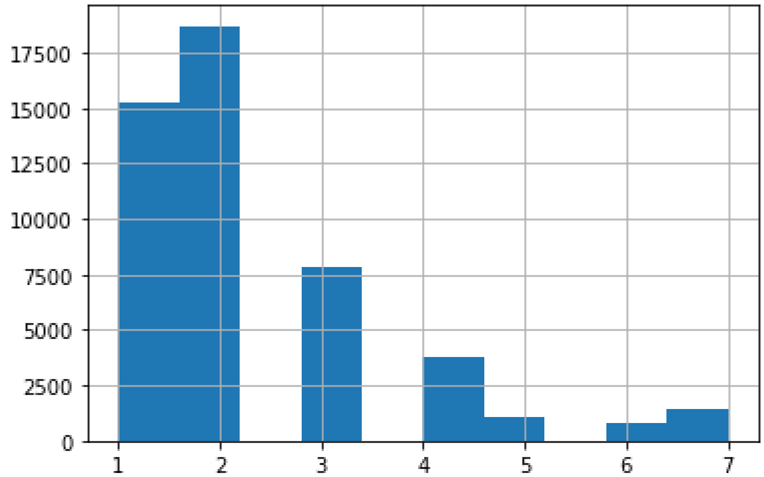
[25 pts] Prepare the Data

[5 pts] Set aside 25% of the data as test set (75% train, 25% test).

In [451…

```
# price_cat like income_cat from example
airbnb["price_cat"] = pd.cut(airbnb["price"],
                                  bins=[-1., 75., 150., 225., 300., 375., 450., np.inf],
```

```
                                  labels=[1,2,3,4,5,6,7])
airbnb['price_cat'].hist()
plt.show()
```



In [452…

```
# Method 1: similar to sample (split by price, drop price columns, use pipeline)
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.25)
for train_index, test_index in split.split(airbnb, airbnb["price_cat"]):
    train_set = airbnb.loc[train_index]
    test_set = airbnb.loc[test_index]

# want to predict price, so drop the true values
airbnb_training = train_set.drop("price", axis=1)

airbnb_training_labels = train_set["price"].copy()
test_labels = test_set["price"].copy()
test_set.drop("price", axis=1, inplace=True)
print(airbnb_training.shape)
print(test_set.shape)
```

```
(36671, 13)
(12224, 13)
```

In [453…

```
# Method 2: different (simply drop the categorical features when fitting model instead
#            (train-test split method from discussion)
abnb_X = airbnb.drop(["price"], axis=1)
abnb_Y = airbnb['price']

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(abnb_X, abnb_Y, test_size=0.25)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(36671, 13)
(12224, 13)
(36671,)
(12224,)
```

[5 pts] Augment the dataframe with two other features which you think would be useful

In [454…

```
# max_num_of_stays: Shows max number of times you can
#   rent listing in year

# reviews_per_min_nights:
# Higher number suggests popular listing (high num of reviews +
#   renter can stay a low minimum num of nights instead of having to
#   stay a long time
# Lower number suggests unpopular listing (either no reviews,
#   or very high number of nights required to stay)

for df in [airbnb, airbnb_training, test_set, X_train, X_test]:
    df["max_num_of_stays"] = df["availability_365"]/df["minimum_nights"]
    df["reviews_per_min_nights"] = df["number_of_reviews"]/df["minimum_nights"]
airbnb[["minimum_nights", 'number_of_reviews','availability_365',
        'max_num_of_stays','reviews_per_min_nights']].head()
```

```
<ipython-input-454-e6571a8661d8>:12: 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-view-versus-a-copy
  df["max_num_of_stays"] = df["availability_365"]/df["minimum_nights"]
<ipython-input-454-e6571a8661d8>:13: 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-view-versus-a-copy
  df["reviews_per_min_nights"] = df["number_of_reviews"]/df["minimum_nights"]
```

Out[454…

	minimum_nights	number_of_reviews	availability_365	max_num_of_stays	reviews_per_min_nights
0	1	9	365	365.000000	9.0
1	1	45	355	355.000000	45.0
2	3	0	365	121.666667	0.0
3	1	270	194	194.000000	270.0
4	10	9	0	0.000000	0.9

[5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

In [455…

```
sample_incomplete_rows = airbnb[airbnb.isnull().any(axis=1)]
sample_incomplete_rows[['number_of_reviews','reviews_per_month']]
```

Out[455…

	number_of_reviews	reviews_per_month
2	0	NaN
19	0	NaN
26	0	NaN

	number_of_reviews	reviews_per_month
36	0	NaN
38	0	NaN
...	...	...
48890	0	NaN
48891	0	NaN
48892	0	NaN
48893	0	NaN
48894	0	NaN

10052 rows × 2 columns

In [456…

```
airbnb["reviews_per_month"].fillna(0, inplace=True)
airbnb_training["reviews_per_month"].fillna(0, inplace=True)
test_set["reviews_per_month"].fillna(0, inplace=True)
X_test["reviews_per_month"].fillna(0, inplace=True)
X_train["reviews_per_month"].fillna(0, inplace=True)
airbnb[['number_of_reviews', 'reviews_per_month']]
```

```
C:\Users\15626\anaconda3\lib\site-packages\pandas\core\series.py:4463: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  return super().fillna(
```

Out[456…

	number_of_reviews	reviews_per_month
0	9	0.21
1	45	0.38
2	0	0.00
3	270	4.64
4	9	0.10
...	...	...
48890	0	0.00
48891	0	0.00
48892	0	0.00
48893	0	0.00
48894	0	0.00

48895 rows × 2 columns

In [457…

```
# Reasoning behind choosing the above imputation method:
```

```
# I chose to fill in the "reviews_per_month" column's null values with 0,
# because the null values in the reivews_per_month column correspond to
# a 0 value in the number_of_reviews column, so if there are no reviews
# at all for the entire listing, then there are not any reviews per month
# for the listing either.
```

[10 pts] Code complete data pipeline using sklearn mixins

In [458…

```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin

airbnb_nums = airbnb_training.drop(["neighbourhood", "neighbourhood_group",
                                    "room_type"], axis=1) # remove cat feature
airbnb_cat = ["neighbourhood", "neighbourhood_group", "room_type"]
min_nights_idx, num_reviews_idx, availability_365_idx = 4, 5, 8

class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    airbnb["max_num_of_stays"] = airbnb["availability_365"]/airbnb["minimum_nights"]
    airbnb["reviews_per_min_nights"] = airbnb["number_of_reviews"]/airbnb["minimum_nigh
    '''
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        max_num_of_stays = X[:, availability_365_idx] / X[:, min_nights_idx]
        reviews_per_min_nights = X[:, num_reviews_idx] / X[:, min_nights_idx]
        return np.c_[X, max_num_of_stays, reviews_per_min_nights]

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

numerical_features = list(airbnb_nums)

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), airbnb_cat),
    ])

airbnb_prepared = full_pipeline.fit_transform(airbnb_training)
```

[15 pts] Fit a model of your choice

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using Mean Absolute Error (MAE). Provide both test and train set MAE values.

In [459…

```
# Method 1: With transformation using the pipeline
```

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(airbnb_prepared, airbnb_training_labels)

# let's try the full preprocessing pipeline on a few training instances
data = airbnb_training.iloc[:5]
labels = airbnb_training_labels.iloc[:5]
data_prepared = full_pipeline.transform(data)

print("Predictions:", lin_reg.predict(data_prepared))
print("Actual labels:", list(labels))

from sklearn.metrics import mean_absolute_error

preds = lin_reg.predict(airbnb_prepared)
mae = mean_absolute_error(airbnb_training_labels, preds)
print("Train MAE: ", mae)

airbnb_prepared_test = full_pipeline.transform(test_set)
preds_test = lin_reg.predict(airbnb_prepared_test)
mae = mean_absolute_error(test_labels, preds_test)
print("Test MAE: ", mae)
```

```
Predictions: [114.21556372  31.99620222 236.58134178  -3.88616147 117.27594854]
Actual labels: [120, 60, 220, 72, 100]
Train MAE:  49.4765658351313
Test MAE:  49.47679202308191
```

In [460…

```
# Method 2: Without transofrmation using the pipeline; instead, just drop the categoric

# Drop the categorical features
X_test_no_cat = X_test.drop(["neighbourhood", "neighbourhood_group", "room_type"], axis
X_train_no_cat = X_train.drop(["neighbourhood", "neighbourhood_group", "room_type"], ax

lm = LinearRegression()
model = lm.fit(X_train_no_cat, Y_train)

predictions = lm.predict(X_train_no_cat)
mae = mean_absolute_error(Y_train, predictions)
print("Train MAE: ", mae)
predictions = lm.predict(X_test_no_cat)
mae = mean_absolute_error(Y_test, predictions)
print("Test MAE: ", mae)
```

```
Train MAE:  47.15080436084287
Test MAE:  49.97318191196053
```

In [ ]: