Applying Machine Learning to the Housing Market:

An Analysis of Housing Price Predictions

One potential idea for using machine learning to analyze and build models for business problems on a housing dataset could be to predict housing prices based on various features of the properties. This could be useful for real estate companies, property tax purposes, investors, and home buyers/sellers to make informed decisions about the housing market.

The first step in this process would be to gather and clean the housing dataset. This would involve collecting data on various features of the properties such as location, size, number of bedrooms and bathrooms, age of the property, and any additional amenities. The data would then need to be preprocessed to ensure that it is in a format suitable for analysis.

Once the data is prepared, various machine learning algorithms could be applied to build predictive models. For example, regression analysis could be used to predict the price of a property based on its features. Decision trees or random forests could also be used to identify the most important features that affect housing prices.

The resulting models could then be used by real estate companies and investors to make informed decisions about buying or selling properties. Home buyers and sellers could also use the models to estimate the value of their property and make decisions about whether to buy or sell.

Overall, machine learning has the potential to provide valuable insights and solutions for business problems in the housing market. By using advanced algorithms and techniques, companies and individuals can make more informed decisions and improve their operations.

Obtain Housing Datasets

Data reference 1 : https://github.com/ageron/data/blob/main/housing.tgz

Data reference 2 : https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

Data reference 3 : https://www.kaggle.com/datasets/vikrishnan/boston-house-prices

In [ ]:
```
import os
import opendatasets as od
import shutil

import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

Import data reference 1

In [ ]:
```
folder_path = os.path.join('datasets', 'housing')
file_name = 'housing.csv'
file_path = os.path.join(folder_path, file_name)
```

In [ ]:
```
root_url = 'https://raw.githubusercontent.com/ageron/data/main/housing/'
file_url = os.path.join(root_url, file_name)
od.download(file_url)
```

```
Downloading https://raw.githubusercontent.com/ageron/data/main/housing/housing.csv to
.\housing.csv
1425408it [00:00, 26631066.23it/s]
```

In [ ]:
```
shutil.move(file_name, file_path)
```

Out[ ]:
```
'datasets\\housing\\housing.csv'
```

In [ ]:
```
housing = pd.read_csv(file_path)
```

Exploratory Data Anlysis

In [ ]:
```
housing.head()
```

Out[ ]:

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

Looking at the dataset, all columns are numeric except for ocean_proximity which is categorical. The list of categories and thier counts are shown below:

In [ ]:
```
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 [ ]:
```
housing.ocean_proximity.value_counts()
```

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

In [ ]:
```
housing.describe()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000
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

In [ ]:
```
housing.median()
```

```
C:\Users\jyuba\AppData\Local\Temp\ipykernel_17456\2435994035.py:1: FutureWarning: Dro
pping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is depre
cated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  housing.median()
```

Out[ ]:
```
longitude               -118.4900
latitude                  34.2600
housing_median_age        29.0000
total_rooms             2127.0000
total_bedrooms           435.0000
population              1166.0000
households               409.0000
median_income              3.5348
median_house_value    179700.0000
dtype: float64
```

There are some missing values in total_bedrooms columns

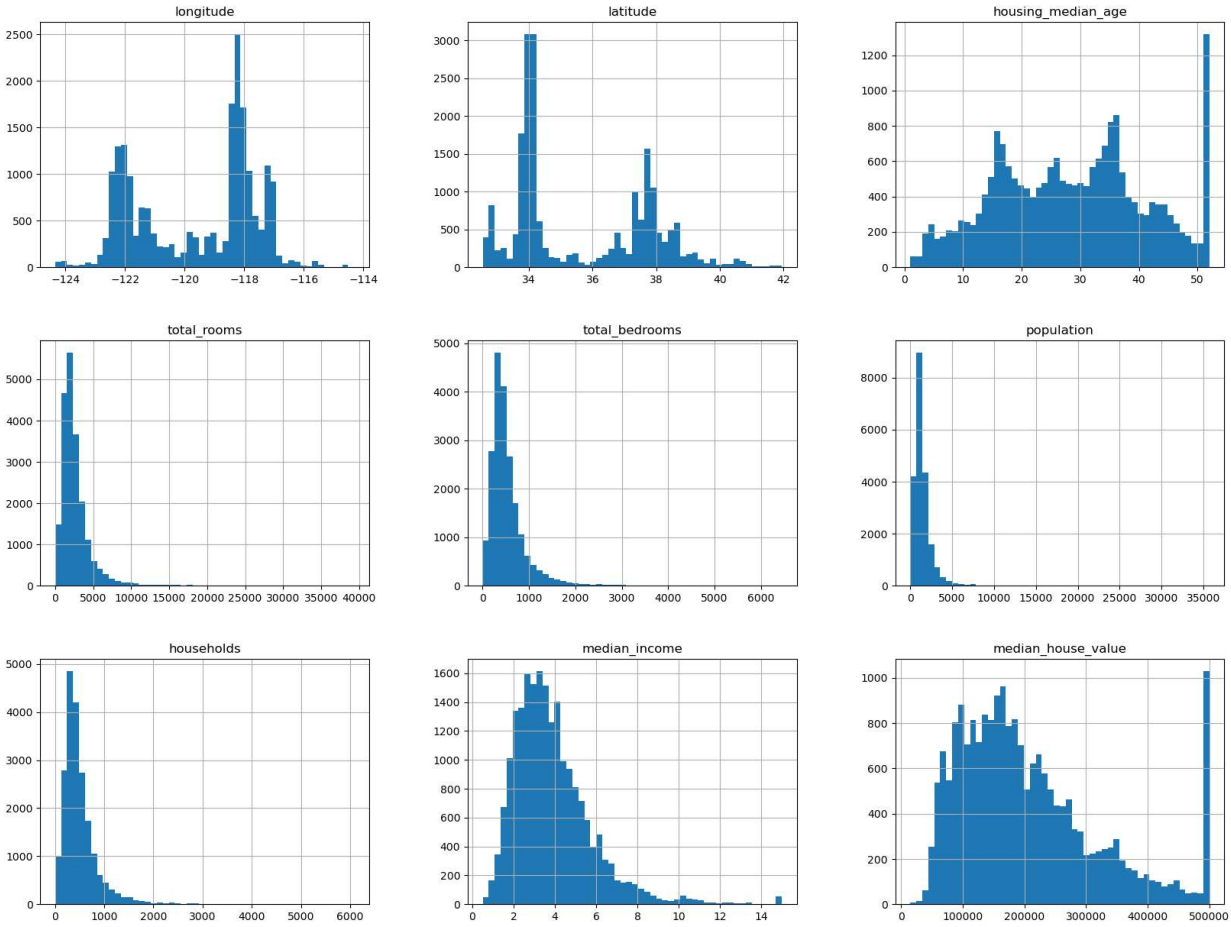
Median_income is in tens of thousands dollars. The median income is thus $35,348.

In [ ]:
```
housing.count()
```

Out[ ]:
```
longitude             20640
latitude              20640
housing_median_age    20640
total_rooms           20640
total_bedrooms        20433
population            20640
households            20640
median_income         20640
median_house_value    20640
ocean_proximity       20640
dtype: int64
```

In [ ]:
```
housing.hist(bins=50, figsize=(20,15))
```

Out[ ]:
```
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
```



Looking at the histogram, at glance the data is skewed to the right, meaning mean > mode. Outliers can cause an issue when training ML algorithm.

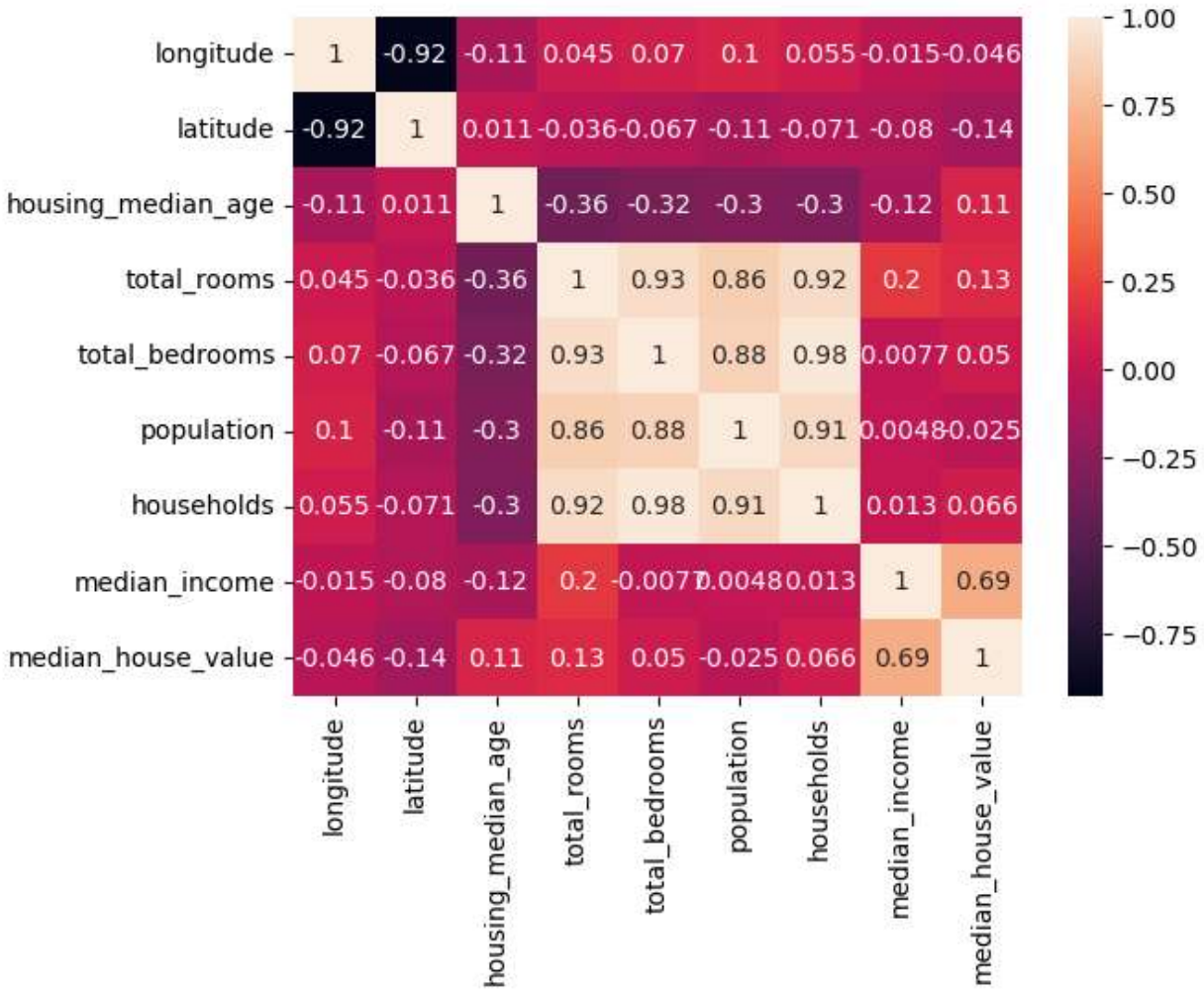
In [ ]:
```
corr_matrix = housing.corr()
```

In [ ]:
```
print(corr_matrix)
```

```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069608 -0.066983           -0.320451     0.930380
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153

                    total_bedrooms  population  households  median_income  \
longitude                 0.069608    0.099773    0.055310      -0.015176
latitude                 -0.066983   -0.108785   -0.071035      -0.079809
housing_median_age       -0.320451   -0.296244   -0.302916      -0.119034
total_rooms               0.930380    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.877747    0.979728      -0.007723
population                0.877747    1.000000    0.907222       0.004834
households                0.979728    0.907222    1.000000       0.013033
median_income            -0.007723    0.004834    0.013033       1.000000
median_house_value        0.049686   -0.024650    0.065843       0.688075

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

In [ ]:

```
import seaborn as sns
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



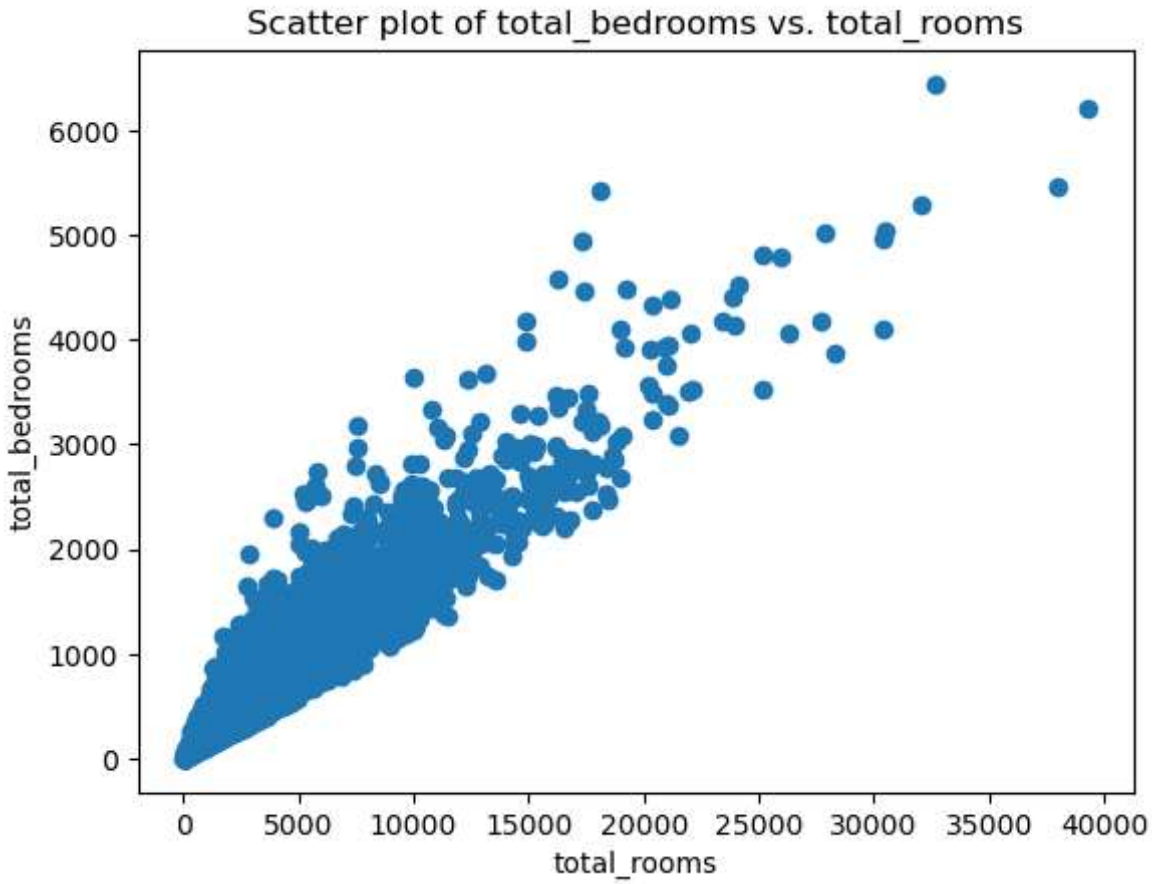
There are some variables that are highly correlated. For example, the correlation between total_bedrooms and households is 0.98. That means when there is additioanl household member, it is likely for the total number of bedroom increase by 1. In this case we might want to consider removing one of those two variables.

total_rooms and total_berooms are also highly correlated, thus that leads to total_room and households are highly correlated.

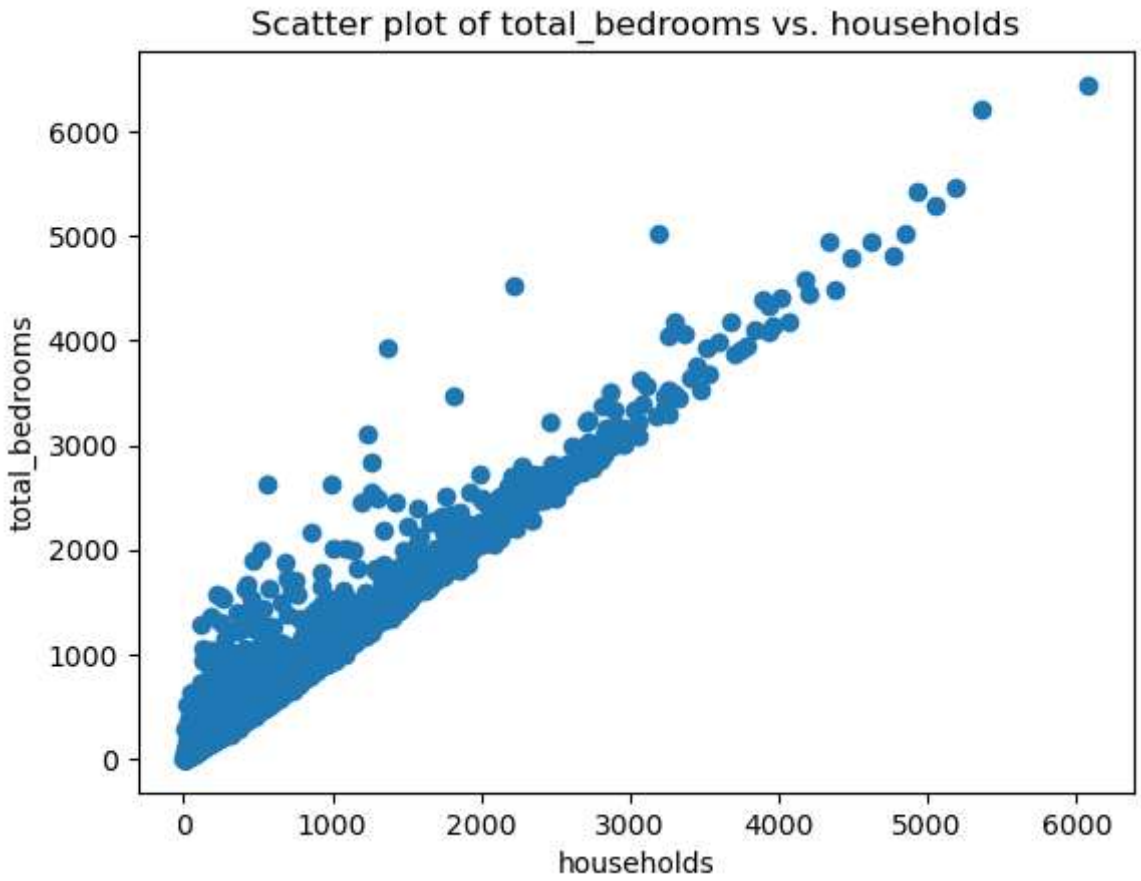
In [ ]:
```
longitude = housing['longitude']
latitude = housing['latitude']
housing_median_age = housing['housing_median_age']
total_rooms = housing['total_rooms']
total_bedrooms = housing['total_bedrooms']
population = housing['population']
households = housing['households']
median_income = housing['median_income']
median_house_value = housing['median_house_value']
```

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd

plt.scatter(total_rooms, total_bedrooms)

plt.xlabel('total_rooms')
plt.ylabel('total_bedrooms')
```

```
plt.title('Scatter plot of total_bedrooms vs. total_rooms')
plt.show()
```



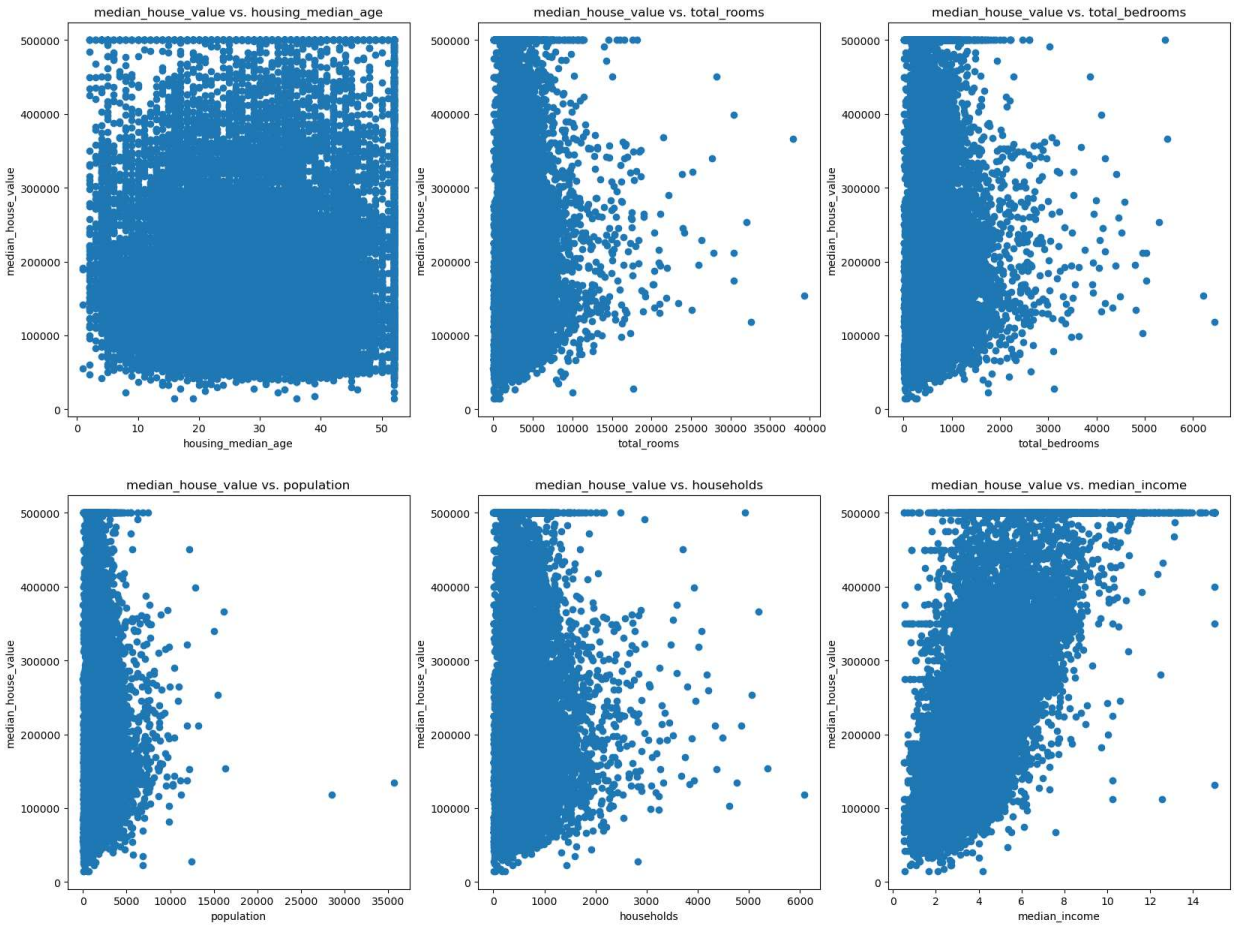
In [ ]:
```
plt.scatter(households, total_bedrooms)

plt.xlabel('households')
plt.ylabel('total_bedrooms')
plt.title('Scatter plot of total_bedrooms vs. households')
plt.show()
```



Which feature will better explain the price of house?

In [ ]:
```
fig, axs = plt.subplots(2, 3, figsize=(20,15))

axs[0,0].scatter(housing_median_age, median_house_value)
axs[0,1].scatter(total_rooms, median_house_value)
axs[0,2].scatter(total_bedrooms , median_house_value)
axs[1,0].scatter(population, median_house_value)
axs[1,1].scatter(households, median_house_value)
axs[1,2].scatter(median_income, median_house_value)

axs[0,0].set_title('median_house_value vs. housing_median_age')
axs[0,0].set_xlabel('housing_median_age')
axs[0,0].set_ylabel('median_house_value')

axs[0,1].set_title('median_house_value vs. total_rooms')
axs[0,1].set_xlabel('total_rooms')
axs[0,1].set_ylabel('median_house_value')

axs[0,2].set_title('median_house_value vs. total_bedrooms')
axs[0,2].set_xlabel('total_bedrooms')
axs[0,2].set_ylabel('median_house_value')

axs[1,0].set_title('median_house_value vs. population')
axs[1,0].set_xlabel('population')
axs[1,0].set_ylabel('median_house_value')

axs[1,1].set_title('median_house_value vs. households')
axs[1,1].set_xlabel('households')
axs[1,1].set_ylabel('median_house_value')
```

```
axs[1,2].set_title('median_house_value vs. median_income')
axs[1,2].set_xlabel('median_income')
axs[1,2].set_ylabel('median_house_value')

plt.show()
```



Looking at the scatter plots between the dependent variable (median_house_value) and other independent variables, there seems no clear patterns and relationships that can be detected by human eyes except for median_income which has positive relationship with hourse value. We are going to provide the dataset to various ML models such as linear regression and categorization tree, and find some patterns.

Term Project: Term Project Milestone 2: Data Preparation

Data Processing

Missing values

In [ ]:
```
housing.isnull().sum()
```

Out[ ]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [ ]:
```
def missing_summary(data):
    for col in data.columns:
        miss_num = data[col].isnull().sum()
        if miss_num > 0:
            print("{} has {} missing value(s)".format(col, miss_num))
        else:
            print("{} has No missing values!".format(col))
```

In [ ]:
```
missing_summary(housing)
```

```
longitude has No missing values!
latitude has No missing values!
housing_median_age has No missing values!
total_rooms has No missing values!
total_bedrooms has 207 missing value(s)
population has No missing values!
households has No missing values!
median_income has No missing values!
median_house_value has No missing values!
ocean_proximity has No missing values!
```

In [ ]:
```
# Set missing values with median value
median_total_bedroom = housing.median()['total_bedrooms']
housing['total_bedrooms'].fillna(median_total_bedroom, inplace=True)
```

```
C:\Users\jyuba\AppData\Local\Temp\ipykernel_17456\1801044438.py:2: FutureWarning: Dro
pping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is depre
cated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  median_total_bedroom = housing.median()['total_bedrooms']
```

In [ ]:
```
missing_summary(housing)
```

```
longitude has No missing values!
latitude has No missing values!
housing_median_age has No missing values!
total_rooms has No missing values!
total_bedrooms has No missing values!
population has No missing values!
households has No missing values!
median_income has No missing values!
median_house_value has No missing values!
ocean_proximity has No missing values!
```

In [ ]:
```
housing.head()
```

Out[ ]:

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

Attribute Combinations

In order to make some variable more useful in ML processing and increase correlation, we need to test out attribute combinations.

In [ ]:
```
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 [ ]:
```
corr_matrix = housing.corr()
```

In [ ]:
```
corr_matrix
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	p
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069120	
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066484	
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.319026	
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.927058	
total_bedrooms	0.069120	-0.066484	-0.319026	0.927058	1.000000	
population	0.099773	-0.108785	-0.296244	0.857126	0.873535	
households	0.055310	-0.071035	-0.302916	0.918484	0.974366	
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007617	
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049457	
rooms_per_household	-0.027540	0.106389	-0.153277	0.133798	0.001765	
bedrooms_per_room	0.081205	-0.098619	0.135622	-0.187381	0.071649	
population_per_household	0.002476	0.002366	0.013191	-0.024581	-0.028325	

It appeared that rooms_per_household and bedrooms_per_room have better correlation with median_house_value

In [ ]:
```
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[ ]:
```
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              0.049457
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.233303
Name: median_house_value, dtype: float64
```

In [ ]:
```
housing.head()
```

Out[ ]:

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

Text Handling (one-hot encoding)

In [ ]:
```
housing['ocean_proximity'].value_counts()
```

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

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

one_hot_encoder = OneHotEncoder()
housing_cat = housing[['ocean_proximity']]
```

In [ ]:
```
housing_cat
```

Out[ ]:

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY
...	...
20635	INLAND
20636	INLAND
20637	INLAND
20638	INLAND
20639	INLAND

20640 rows × 1 columns

In [ ]:
```
housing_cat_1hot = one_hot_encoder.fit_transform(housing_cat)
```

In [ ]:
```
housing_cat_1hot.toarray()
```

Out[ ]:
```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

Feature Scaling

In [ ]:
```
housing.head()
```

Out[ ]:

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

In [ ]:
```
# Select the features to scale
features_to_scale = ['housing_median_age', 'total_rooms', 'total_bedrooms', 'populatio
```

In [ ]:
```
# Create an instance of the StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [ ]:
```
scaler.fit(housing[features_to_scale])
```

Out[ ]:
```
StandardScaler()
```

In [ ]:
```
# Transform the data
scaled_data = scaler.transform(housing[features_to_scale])
```

In [ ]:
```
print(scaled_data)
```

```
[[ 0.98214266 -0.8048191  -0.97247648 ...  0.62855945 -1.02998783
  -0.04959654]
 [-0.60701891  2.0458901   1.35714343 ...  0.32704136 -0.8888972
  -0.09251223]
 [ 1.85618152 -0.53574589 -0.82702426 ...  1.15562047 -1.29168566
  -0.02584253]
 ...
 [-0.92485123 -0.17499526 -0.12360781 ... -0.09031802  0.02113407
  -0.0717345 ]
 [-0.84539315 -0.35559977 -0.30482697 ... -0.04021111  0.09346655
  -0.09122515]
 [-1.00430931  0.06840827  0.18875678 ... -0.07044252  0.11327519
  -0.04368215]]
```

In [ ]:
```
# Update the original dataset with the scaled data
housing[features_to_scale] = scaled_data
```

In [ ]:
```
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
0	-122.23	37.88	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	
1	-122.22	37.86	-0.607019	2.045890	1.357143	0.861439	1.669961	
2	-122.24	37.85	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	
3	-122.25	37.85	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	
4	-122.25	37.85	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	

In [ ]:
```
housing_cat_1hot_df = pd.DataFrame(housing_cat_1hot.toarray(),
                                   columns = one_hot_encoder.get_feature_names_out(['
```

In [ ]:
```
housing_final = pd.concat([housing, housing_cat_1hot_df], axis=1)
```

In [ ]:
```
housing_final.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
0	-122.23	37.88	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	
1	-122.22	37.86	-0.607019	2.045890	1.357143	0.861439	1.669961	
2	-122.24	37.85	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	
3	-122.25	37.85	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	
4	-122.25	37.85	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	

Term Project: Term Project Milestone 3: Model Building and Evaluation

Prepare Train and Test Dataset

In [ ]:
```
test_ratio = 0.2
seed_num = 42
```

In [ ]:
```
def split_train_test(seed_num, data, test_ratio):
    np.random.seed(seed_num)
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(housing)*test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [ ]:
```
train_set, test_set = split_train_test(seed_num, housing_final, test_ratio)
```

In [ ]:
```
[len(train_set), len(test_set)]
```

Out[ ]:
```
[16512, 4128]
```

In [ ]:
```
train_set
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
14196	-117.22	32.75	0.425936	1.542581	1.369066	1.084851	1.49733
8267	-117.03	32.69	-1.481058	-0.795193	-0.891405	-0.642401	-0.86979
17445	-122.27	37.74	-0.050812	1.958797	2.425383	1.367428	2.57496
14265	-121.82	37.25	-0.289187	0.634975	0.231677	0.664518	0.39354
2271	-115.98	33.32	-1.639974	-1.098187	-1.170387	-1.203139	-1.24382
...	...	...	...	...	...	...	.
11284	-122.37	37.94	1.617807	-0.764023	-0.734030	-0.729823	-0.79655
11964	-118.38	33.89	0.505394	-0.393188	-0.493200	-0.612377	-0.49053
5390	-119.33	36.28	-1.004309	-0.005392	-0.023460	-0.307723	0.05351
860	-117.19	34.08	-0.527561	-0.077359	0.043305	0.124973	-0.01449
15795	-118.86	34.22	-0.527561	-0.644384	-0.803180	-0.664477	-0.79655

16512 rows × 18 columns

In [ ]:
```
test_set
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
20046	-122.38	40.67	-1.481058	-0.162619	-0.221371	-0.133762	-0.16096
3024	-118.37	33.83	0.505394	-0.654927	-0.786488	-0.728057	-0.74947
15663	-117.24	32.72	0.823227	0.207758	-0.252369	-0.221184	-0.17665
20484	-118.44	34.05	-0.845393	0.982890	1.562207	0.406667	1.40316
9814	-118.44	34.18	0.346478	-0.233210	-0.292905	-0.326268	-0.28389
...	...	...	...	...	...	...	.
15362	-121.92	40.52	-1.242684	0.891671	0.820639	0.329841	0.61325
16623	-122.08	37.68	-0.209729	-0.013185	0.346131	-0.021614	0.28107
18086	-119.00	35.39	1.061601	0.093161	-0.049689	-0.196459	-0.03279
2144	-117.92	33.63	0.823227	-0.534829	-0.741184	-0.770443	-0.71808
3665	-118.39	34.02	0.743768	-0.086527	0.236446	-0.100206	0.19475

4128 rows × 18 columns

In [ ]:
```
train_set.columns
```

Out[ ]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'rooms_per_household',
       'bedrooms_per_room', 'population_per_household',
       'ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN'],
      dtype='object')
```

In [ ]:
```
features = ['housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'rooms_per_household',
       'bedrooms_per_room', 'population_per_household',
       'ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN']
label = 'median_house_value'
```

In [ ]:
```
train_feature = train_set[features]
train_label = train_set[label]
```

In [ ]:
```
train_feature.head()
```

Out[ ]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	r
14196	0.425936	1.542581	1.369066	1.084851	1.497331	0.377481	
8267	-1.481058	-0.795193	-0.891405	-0.642401	-0.869793	0.418012	
17445	-0.050812	1.958797	2.425383	1.367428	2.574961	-0.096417	
14265	-0.289187	0.634975	0.231677	0.664518	0.393545	0.681991	
2271	-1.639974	-1.098187	-1.170387	-1.203139	-1.243825	-1.264292	

In [ ]:
```
train_label
```

Out[ ]:
```
14196    291000.0
8267     156100.0
17445    353900.0
14265    241200.0
2271      53800.0
           ...
11284     71600.0
11964    379300.0
5390     104200.0
860       84700.0
15795    251400.0
Name: median_house_value, Length: 16512, dtype: float64
```

Select and Train a ML Model

Linear Regression

In [ ]:
```
# Select linear regression model
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
```

In [ ]:
```
# Fit training data
lin_reg.fit(train_feature, train_label)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
# Make prediction
lin_reg_pred = lin_reg.predict(train_feature)
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
lin_mse = mean_squared_error(train_label, lin_reg_pred)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[ ]:
```
69201.68545572885
```

In [ ]:
```
np.mean(train_label)
```

Out[ ]:
```
206111.15297965117
```

I trained the linear regression model using the training housing dataset. The calculated RMSE is 69K. At this point, I can barely say if this model is good or not but when calculating the mean amount of the training label, it appeared that the amount is 206K. It seems the error of 68K is not really accurate. So I will try another ML model as well as K-fold cross validation.

In [ ]:
```
from sklearn.model_selection import cross_val_score

lin_reg_score = cross_val_score(lin_reg, train_feature, train_label,
                                scoring='neg_mean_squared_error', cv=10)
```

In [ ]:
```
lin_rmse_score = np.sqrt(-lin_reg_score)
lin_rmse_score
```

Out[ ]:
```
array([71628.11736904, 72536.31495598, 67538.65287286, 68562.01475162,
       71733.3907633 , 68827.57918948, 68288.3593127 , 68414.36852853,
       70618.38764183, 67703.47683603])
```

In [ ]:
```
lin_rmse_score.mean()
```

Out[ ]:
```
69585.06622213697
```

It seems K-fold cross validation shows 69K RMSE is the average of 10 training results for linear regression.

Decision Tree Regressor

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

```
tree_reg = DecisionTreeRegressor()
```

In [ ]:
```
# Fit train dataset
tree_reg.fit(train_feature, train_label)
```

Out[ ]:
```
DecisionTreeRegressor()
```

In [ ]:
```
# Make preiction
tree_reg_pred = tree_reg.predict(train_feature)
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
tree_mse = mean_squared_error(train_label, tree_reg_pred)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[ ]:
```
0.0
```

It appeared that RMSE is 0 which doesn't make sense. We will use K-fold corss validation to see what's going on.

In [ ]:
```
from sklearn.model_selection import cross_val_score

tree_reg_score = cross_val_score(tree_reg, train_feature, train_label,
                                 scoring='neg_mean_squared_error', cv=10)
```

In [ ]:
```
tree_rmse_score = np.sqrt(-tree_reg_score)
tree_rmse_score
```

Out[ ]:
```
array([83237.04173106, 83396.68050819, 82942.36673398, 82227.73154069,
       82154.76580487, 82253.84538787, 81218.01950075, 82264.31079569,
       84845.22197146, 79919.30838651])
```

In [ ]:
```
tree_rmse_score.mean()
```

Out[ ]:
```
82445.9292361068
```

The cross validation revealed that the first training was overfitted and the average of RMSE from the cross validation shows that decision tree regressor didn't perform better than linear regression model.

Random Forest Regressor

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
```

In [ ]:
```
# Train model
forest_reg.fit(train_feature, train_label)
```

Out[ ]:
```
RandomForestRegressor()
```

In [ ]:
```
# Make preiction
forest_reg_pred = forest_reg.predict(train_feature)
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
forest_mse = mean_squared_error(train_label, forest_reg_pred)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[ ]:
```
21868.732491691295
```

This 22K is much better than 69K from the linear regression model. Let's perform K-fold cross validation.

In [ ]:
```
from sklearn.model_selection import cross_val_score

forest_reg_score = cross_val_score(forest_reg, train_feature, train_label,
                                   scoring='neg_mean_squared_error', cv=10)
```

In [ ]:
```
forest_rmse_score = np.sqrt(-forest_reg_score)
forest_rmse_score
```

Out[ ]:
```
array([59012.20517966, 59288.95888669, 59937.69295443, 58124.63510073,
       61325.12438997, 57060.35203055, 58660.98742165, 56615.68949957,
       59513.53458082, 58934.49761721])
```

In [ ]:
```
forest_rmse_score.mean()
```

Out[ ]:
```
58847.36776612686
```

The average RMSE from the cross validation is increaed to 59K but this amount is still better than the amount from linear regression. There could be more ways to improve the performance of the model and better models outperforming the random forest model. However, the trained model is quite accurate to predict the housing values and satisfying the purpose of the project.

In [ ]:
```
# Test on test dataset
forest_reg_test_pred = forest_reg.predict(test_set[features])
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
forest_test_mse = mean_squared_error(forest_reg_test_pred, test_set[label])
forest_test_rmse = np.sqrt(forest_test_mse)
forest_test_rmse
```

Out[ ]:
```
23415.261154116517
```

12.2 Term Project: Final Project Submission

One potential idea for using machine learning to analyze and build models for business problems on a housing dataset could be to predict housing prices based on various features of the properties. This could be useful for real estate companies, property tax purposes, investors, and home buyers/sellers to make informed decisions about the housing market.

The first step in this process would be to gather and clean the housing dataset. This would involve collecting data on various features of the properties such as location, size, number of bedrooms and bathrooms, age of the property, and any additional amenities. The data would then need to be preprocessed to ensure that it is in a format suitable for analysis.

Thanks to Dr. Jason, one of the the housing prices dataset is available in Kaggle. The dataset name is Boston House Prices. Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970.

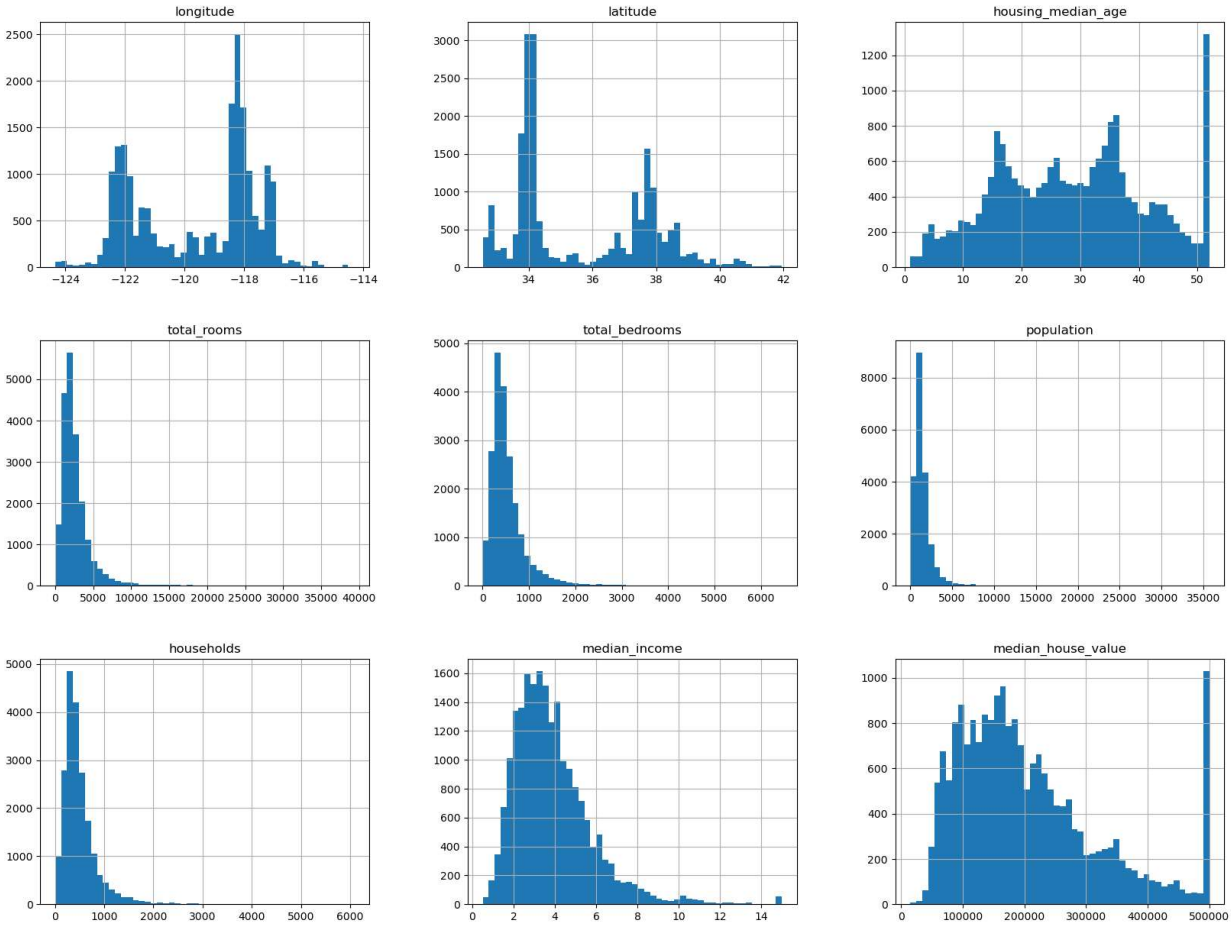
Once the data is obtained and prepared, various machine learning algorithms could be applied to build predictive models. For example, regression analysis could be used to predict the price of a property based on its features. Decision trees or random forests could also be used to identify the most important features that affect housing prices.

The resulting models could then be used by real estate companies and investors to make informed decisions about buying or selling properties. Home buyers and sellers could also use the models to estimate the value of their property and make decisions about whether to buy or sell.

Overall, machine learning has the potential to provide valuable insights and solutions for business problems in the housing market. By using advanced algorithms and techniques, companies and individuals can make more informed decisions and improve their operations.

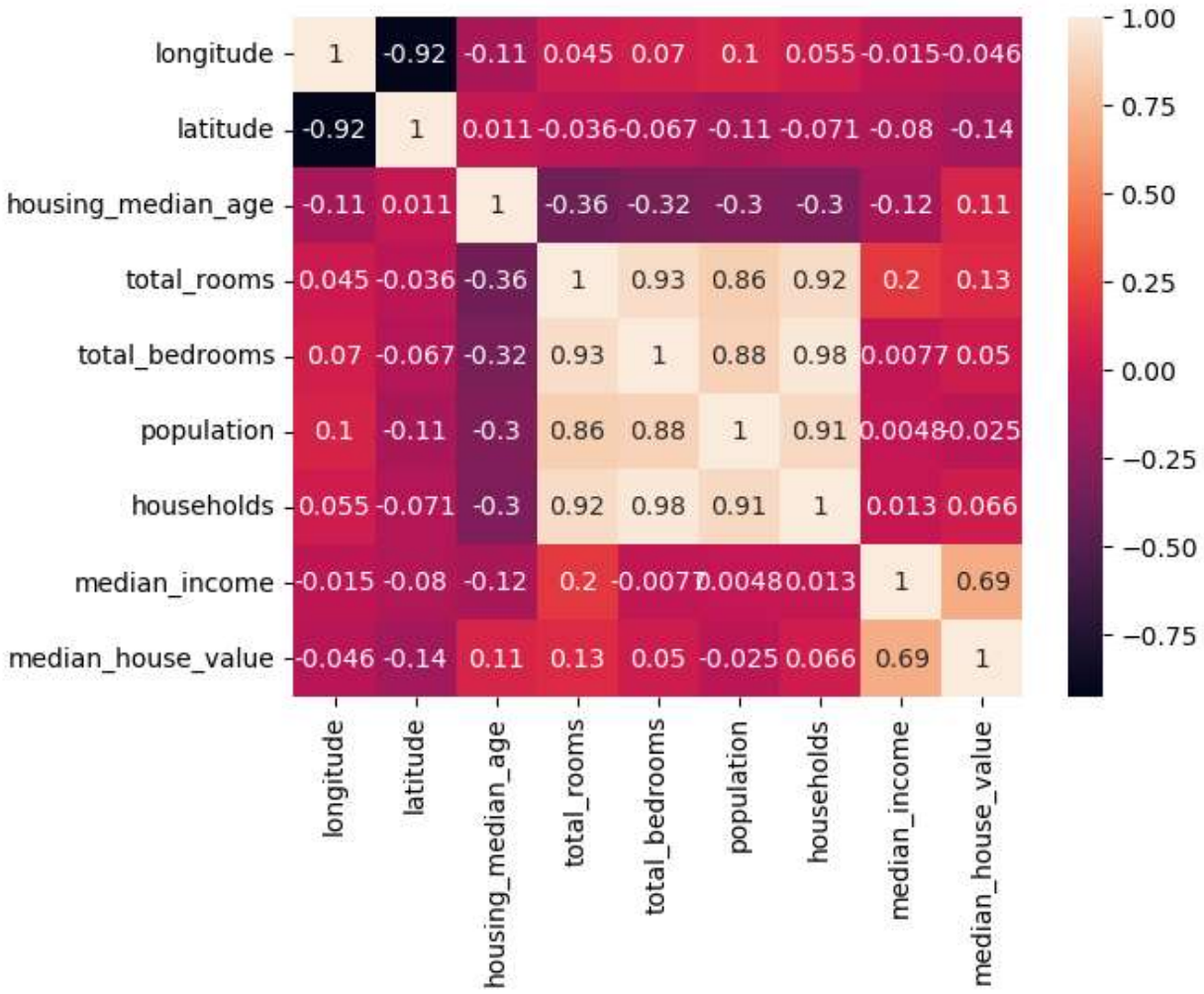
In [ ]:
```
housing.hist(bins=50, figsize=(20,15))
```

```
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
```



Looking at the histogram, at glance the data is skewed to the right, meaning mean > mode. Outliers can cause an issue when training ML algorithm.

In [ ]:
```
import seaborn as sns
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



There are some variables that are highly correlated. For example, the correlation between total_bedrooms and households is 0.98. That means when there is additioanl household member, it is likely for the total number of bedroom increase by 1. In this case we might want to consider removing one of those two variables.

total_rooms and total_berooms are also highly correlated, thus that leads to total_room and households are highly correlated.

Attribute Combinations

In order to make some variable more useful in ML processing and increase correlation, we need to test out attribute combinations.

In [ ]:
```
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 [ ]:
```
corr_matrix = housing.corr()
```

In [ ]:
```
corr_matrix
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	p
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069120	
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066484	
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.319026	
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.927058	
total_bedrooms	0.069120	-0.066484	-0.319026	0.927058	1.000000	
population	0.099773	-0.108785	-0.296244	0.857126	0.873535	
households	0.055310	-0.071035	-0.302916	0.918484	0.974366	
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007617	
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049457	
rooms_per_household	-0.027540	0.106389	-0.153277	0.133798	0.001765	
bedrooms_per_room	0.081205	-0.098619	0.135622	-0.187381	0.071649	
population_per_household	0.002476	0.002366	0.013191	-0.024581	-0.028325	

It appeared that rooms_per_household and bedrooms_per_room have better correlation with median_house_value

In [ ]:
```
corr_matrix['median_house_value'].sort_values(ascending=False)
```

```
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              0.049457
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.233303
Name: median_house_value, dtype: float64
```

In order to make some variable more useful in ML processing and increase correlation, we need to test out attribute combinations.

In [ ]:
```
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 [ ]:
```
corr_matrix = housing.corr()
```

In [ ]:
```
corr_matrix
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	p
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069120	
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066484	
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.319026	
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.927058	
total_bedrooms	0.069120	-0.066484	-0.319026	0.927058	1.000000	
population	0.099773	-0.108785	-0.296244	0.857126	0.873535	
households	0.055310	-0.071035	-0.302916	0.918484	0.974366	
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007617	
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049457	
rooms_per_household	-0.027540	0.106389	-0.153277	0.133798	0.001765	
bedrooms_per_room	0.081205	-0.098619	0.135622	-0.187381	0.071649	
population_per_household	0.002476	0.002366	0.013191	-0.024581	-0.028325	

It appeared that rooms_per_household and bedrooms_per_room have better correlation with median_house_value

In [ ]:
```
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[ ]:
```
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              0.049457
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.233303
Name: median_house_value, dtype: float64
```

Numeric variables are scaled/standardized, and dummy varariables are created for categorical variable. Dataset is divided into training and test dateset.

Finally, training dataset is used to train 3 different ML algorithm: linear regression, decision tree regressor, and random forest regressor. After testing their performance using 10-folds cross validation technique, it appeared that random forest produce the most accurate prediction.

Then, testing on test dataset shows that the model has RMSE error of only 23K.

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
```

In [ ]:
```
# Train model
forest_reg.fit(train_feature, train_label)
```

```
RandomForestRegressor()
```

In [ ]:
```
# Make preiction
forest_reg_pred = forest_reg.predict(train_feature)
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
forest_mse = mean_squared_error(train_label, forest_reg_pred)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
21868.732491691295
```

In [ ]:
```
# Test on test dataset
forest_reg_test_pred = forest_reg.predict(test_set[features])
```

In [ ]:
```
# Calculate RMSE
from sklearn.metrics import mean_squared_error
forest_test_mse = mean_squared_error(forest_reg_test_pred, test_set[label])
forest_test_rmse = np.sqrt(forest_test_mse)
forest_test_rmse
```

```
23415.261154116517
```

The model suggests that on average, the model's predictions are $23K away from the actual values. This number is much better than ones from linear regression and decision tree regression.

One challenge is optimizing their performance for use with small datasets resulting from two-phase sampling designs. We can increase the data size by conducting data gathering and generate a large dataset. Also, we can improve the performance by fine tuning hyperparameter of random forest regression model. Another opportunity for improving the performance of random forest models is by combining them with other machine learning methods. For example, stacking random forests and simple linear models can offer improvements over using random forests alone.

In conclusion, it appears that a random forest regression model was trained on the Boston housing dataset and achieved an RMSE of 23,000. This performance was better than that of linear regression or decision tree regression models. This suggests that the random forest model was able to capture more complex relationships in the data and make more accurate predictions than the other models