Demonstrate the steps to build a machine-learning model that predicts the median housing price using the California housing price dataset.

1. Perform the describe and info steps

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
housing = pd.read_csv("/content/housing.csv")

housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Next steps: Generate code with housing | New interactive sheet

```
housing.info()
```

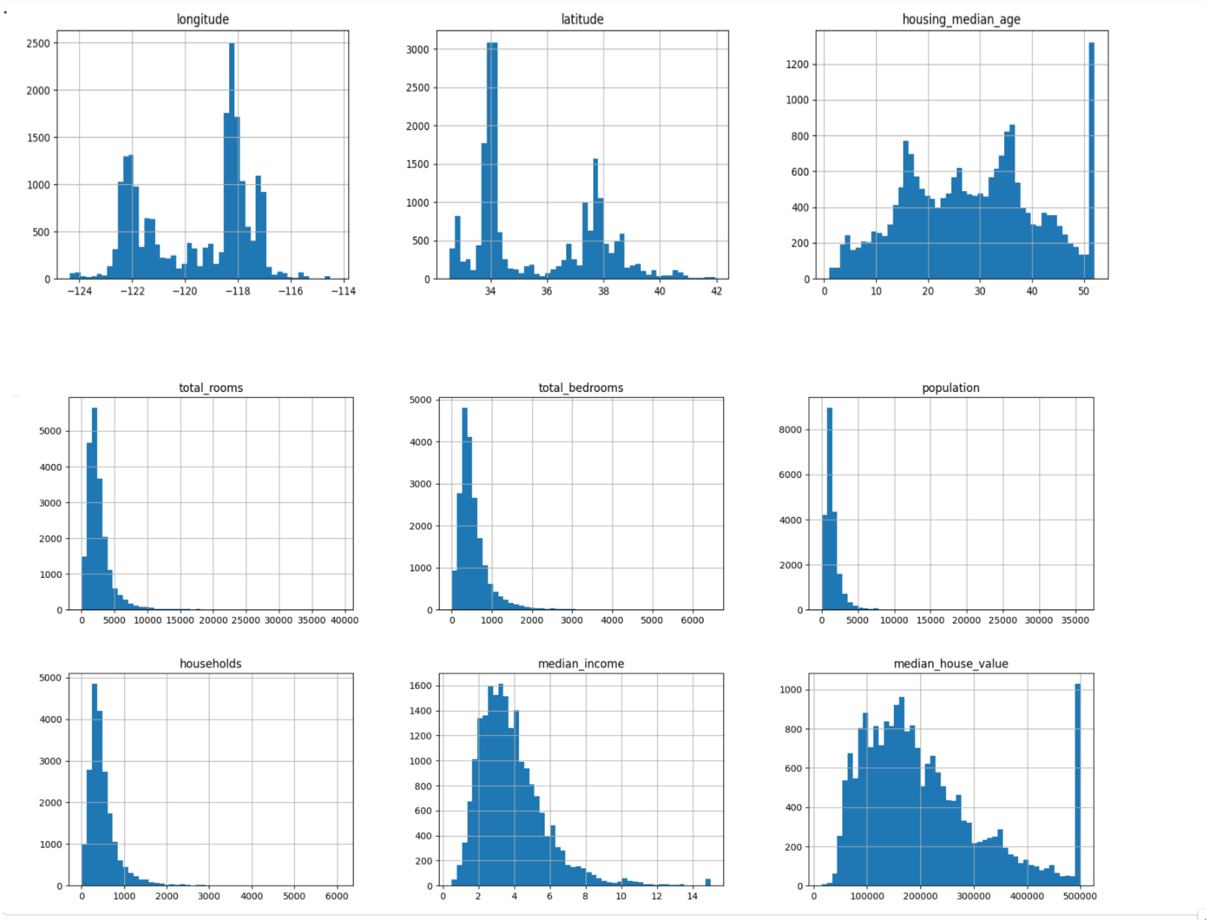
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 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
```

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

2. Plot the histogram of each feature(Indicate what does histogram indicate on median_income and house_median_age)

```python
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



3. Demonstrate the process of creating a test set(write the difference between random and stratified test set)

```python
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

print("Total dataset size:", len(housing))
print("Training set size:", len(train_set))
print("Test set size:", len(test_set))
```

```
Total dataset size: 20640
Training set size: 16512
Test set size: 4128
```

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedShuffleSplit

# Create income category
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]
)

# Perform Stratified Split
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]

# ✅ Print Output
print("Total dataset size:", len(housing))
print("\nStratified Sampling:")
print("Training set size:", len(strat_train_set))
print("Test set size:", len(strat_test_set))

print("\nIncome Category Distribution in Full Dataset:")
print(housing["income_cat"].value_counts(normalize=True).sort_index())

print("\nIncome Category Distribution in Stratified Test Set:")
print(strat_test_set["income_cat"].value_counts(normalize=True).sort_index())
```

```python
# Optional: Remove income_cat column after split
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

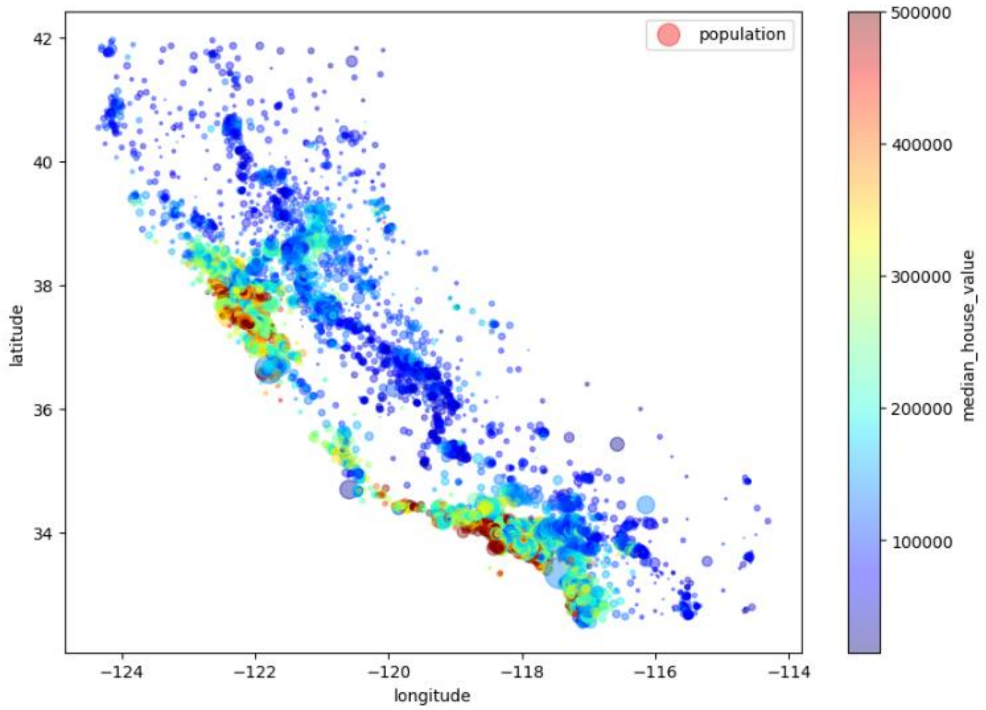
```
Total dataset size: 20640

Stratified Sampling:
Training set size: 16512
Test set size: 4128

Income Category Distribution in Full Dataset:
income_cat
1    0.039826
2    0.318847
3    0.350581
4    0.176308
5    0.114438
Name: proportion, dtype: float64

Income Category Distribution in Stratified Test Set:
income_cat
1    0.039971
2    0.318798
3    0.350533
4    0.176357
5    0.114341
Name: proportion, dtype: float64
```

4. List the geographical features from the dataset and plot a graph to Visualize Geographical Data(what does the graph indicate w.r.t housing prices and location)

```
geo_features = ["longitude", "latitude"]
print("Geographical Features:", geo_features)
```

```
Geographical Features: ['longitude', 'latitude']
```

```
import matplotlib.pyplot as plt

housing.plot(kind="scatter",
        x="longitude",
        y="latitude",
        alpha=0.4,
        s=housing["population"]/100,
        label="population",
        c="median_house_value",
        cmap="jet",
        colorbar=True,
        figsize=(10,7))

plt.legend()
plt.show()
```

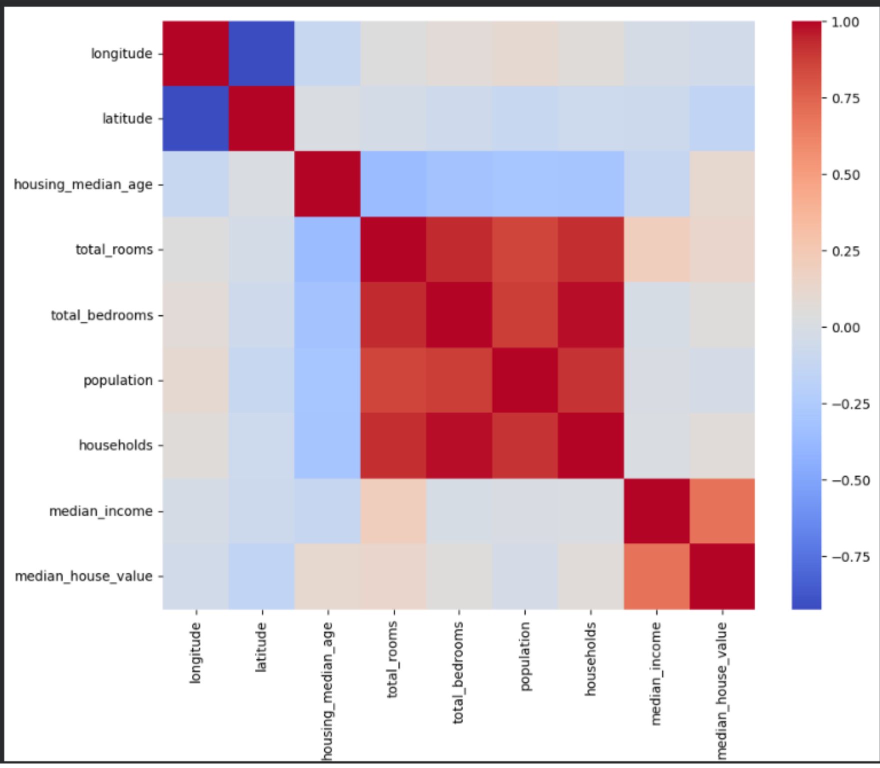


5. Plot a graph to show features correlation with housing price. Which feature corelates to the maximum. Plot the graph for that with housing price and analyze what the graph indicates

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

correlation_with_price = corr_matrix["median_house_value"].sort_values(ascending=False)

print("Correlation with Median House Value:")
print(correlation_with_price)
```

```
Correlation with 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
Name: median_house_value, dtype: float64
```

```
import seaborn as sns

plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, cmap="coolwarm")
plt.show()
```

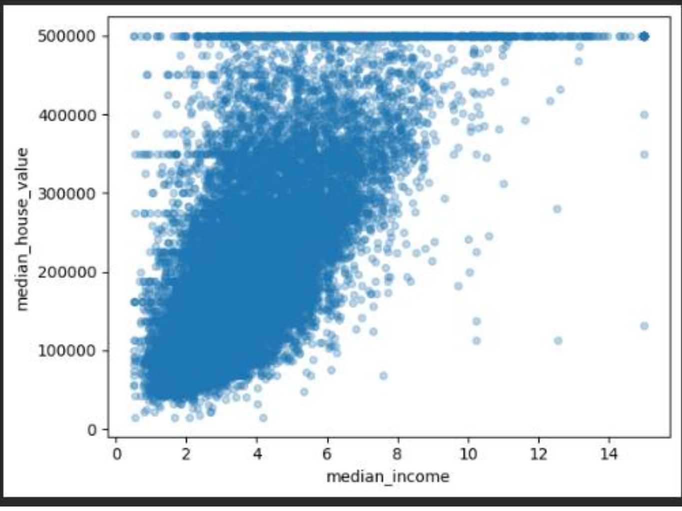


6. List the features that could be combined to improve correlation and plot again to see if correlation has improved

```python
max_corr_feature = correlation_with_price.index[1]  # Skip self correlation
print("Feature with Maximum Correlation:", max_corr_feature)
```

```
Feature with Maximum Correlation: median_income
```

```python
housing.plot(kind="scatter",
        x=max_corr_feature,
        y="median_house_value",
        alpha=0.3)

plt.show()
```




```python
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"]

new_corr_matrix = housing.corr(numeric_only=True)

print("New Correlation with Median House Value:")
print(new_corr_matrix["median_house_value"].sort_values(ascending=False))
```

```
New Correlation with Median House Value:
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
```

7. List the features that needs to be cleaned and demonstrate the process of cleaning

```python
print("Missing Values:")
print(housing.isnull().sum())
```

```
Missing Values:
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
income_cat                   0
rooms_per_household          0
bedrooms_per_room          207
population_per_household     0
dtype: int64
```

```python
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

df_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(df_num)

df_num_imputed = pd.DataFrame(imputer.transform(df_num),
                              columns=df_num.columns)

print("After Imputation:")
print(df_num_imputed.isnull().sum())
```

```
After Imputation:
longitude                   0
latitude                    0
housing_median_age          0
total_rooms                 0
total_bedrooms              0
population                  0
households                  0
median_income               0
median_house_value          0
income_cat                  0
rooms_per_household         0
bedrooms_per_room           0
population_per_household    0
dtype: int64
```

8. Is there any categorical data that needs to be converted to numerical? If so explain the method used to convert and code the same and show the output.

```python
print("Categorical Data:")
print(housing["ocean_proximity"].value_counts())
```

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

9. Discuss the importance of feature scaling
10. Design a pipeline inculcating (Custom transform, feature scaling and encoding). Explain how it works

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_scaled = scaler.fit_transform(df_num_imputed)

print("Scaled Data (First 5 Rows):")
print(df_scaled[:5])
```

```
Scaled Data (First 5 Rows):
[[-1.32783522  1.05254828  0.98214266 -0.8048191  -0.97247648 -0.9744286
  -0.97703285  2.34476576  2.12963148  1.89012782  0.62855945 -1.14993031
  -0.04959654]
 [-1.32284391  1.04318455 -0.60701891  2.0458901   1.35714343  0.86143887
   1.66996103  2.33223796  1.31415614  1.89012782  0.32704136 -0.99038135
  -0.09251223]
 [-1.33282653  1.03850269  1.85618152 -0.53574589 -0.82702426 -0.82077735
  -0.84363692  1.7826994   1.25869341  1.89012782  1.15562047 -1.44586501
  -0.02584253]
 [-1.33781784  1.03850269  1.85618152 -0.62421459 -0.71972345 -0.76602806
  -0.73378144  0.93296751  1.16510007  0.94189394  0.15696608 -0.49362714
  -0.0503293 ]
 [-1.33781784  1.03850269  1.85618152 -0.46240395 -0.61242263 -0.75984669
  -0.62915718 -0.012881    1.17289952 -0.00633994  0.3447108  -0.707889
  -0.08561576]]
```