Frame the Problem and Look at the Big Picture

Define the objective in business terms.

Predict the median housing in any disctrict, given all the other metrics.

How will your solution be used?

What are the current solutions/workarounds (if any)?

How should you frame this problem (supervised/unsupervised, online/offline, etc.)?

How should performance be measured?

Is the performance measure aligned with the business objective?

What would be the minimum performance needed to reach the business objective?

What are comparable problems? Can you reuse experience or tools?

Is human expertise available?

How would you solve the problem manually?

List the assumptions you (or others) have made so far.

Very assumptions if possible.

Get the Data

List the data you need and how much you need.

In [ ]:

Find and document where you can get that data.

https://raw.githubusercontent.com/ageron/handson-ml/master/

Check how much space it will take.

In [ ]:

Check legal obligations, and get authorization if necessary.

Get access authorizations.

Create a workspace (with enough storage space).

Get the data

In [1]:
```
import os
import tarfile
import urllib

# URL del file da scaricare
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
```

```
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

# Percorso dove i dati saranno salvati
HOUSING_PATH = os.path.join("../Data", "housing")

# Funzione per scaricare il file
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.exists(housing_path):  # Se la cartella non esiste, creala
        os.makedirs(housing_path)

    # Scarica il file
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    print(f"File scaricato in: {tgz_path}")

    # Estrai il file .tgz senza filtro o con un filtro che non influisce sull'es
    with tarfile.open(tgz_path) as housing_tgz:
        # Rimuoviamo il filtro, estraiamo tutti i file
        housing_tgz.extractall(path=housing_path)
    print(f"File estratti in: {housing_path}")

# Esegui la funzione per scaricare e estrarre il dataset
fetch_housing_data()
```

```
File scaricato in: ../Data\housing\housing.tgz
File estratti in: ../Data\housing
```

Convert the data to a format you can easily manipulate (without changing the data itself).

In [2]:

```
import pandas as pd

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

housing = load_housing_data()
housing.head()
```

Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Ensure sensitive information is deleted or protected (eg. anonymized).

In [ ]:

Check the size and type of data (time series, sample, geographical, etc.).

In [3]:
```
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
```

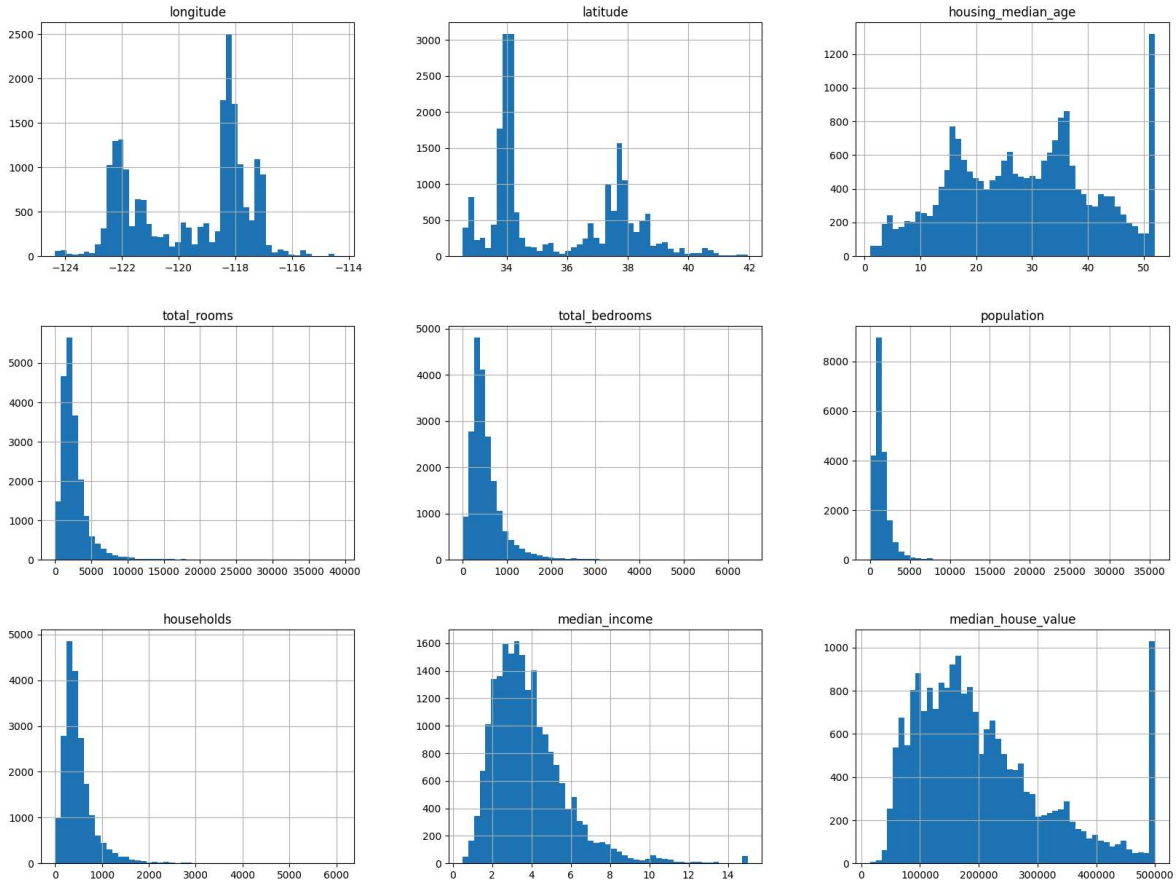
In [4]:
```
housing.describe()
```

Out[4]:

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

In [5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show
```

Out[5]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Sample a test set, put it aside, and never look at it (no data snooping!).

Random Sampling

In [10]:
```
from sklearn.model_selection import train_test_split

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

Stratfield sampling

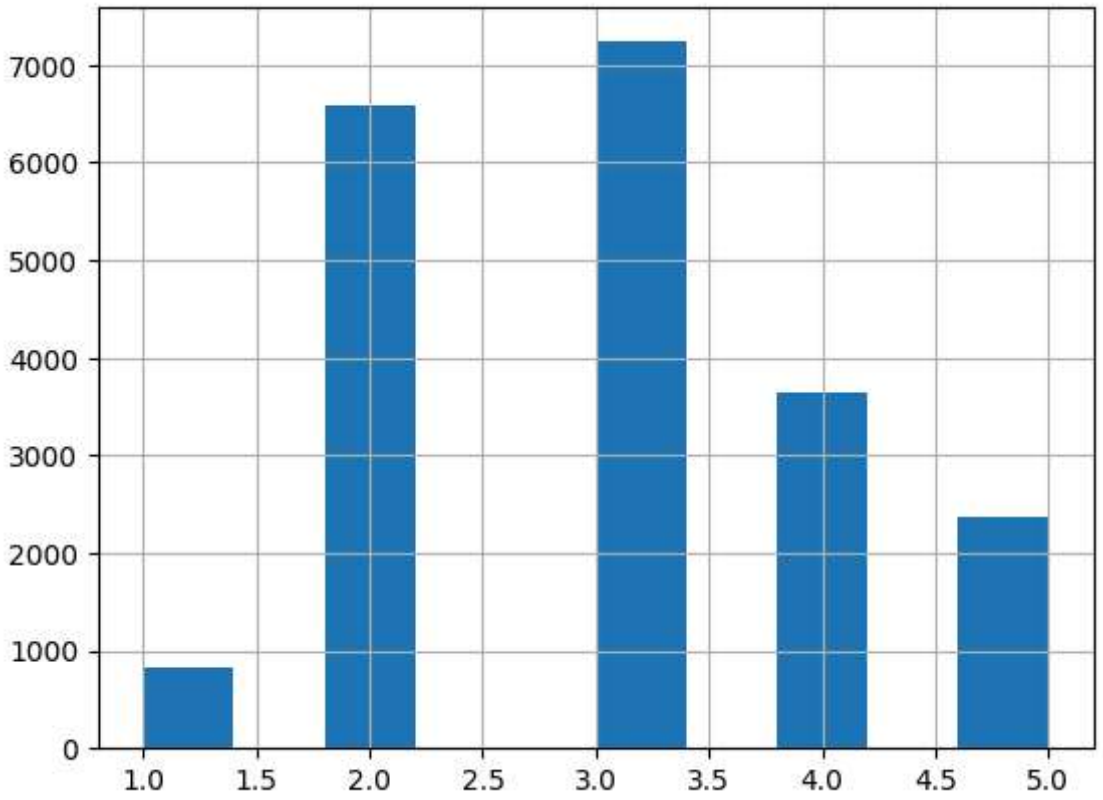
In [11]:
```
import numpy as np
housing['income_cat'] = pd.cut(housing['median_income'],
                               bins = [0, 1.5, 3, 4.5, 6, np.inf],
                               labels = [1, 2, 3, 4, 5])
housing[['median_income','income_cat']]
```

Out[11]:

	median_income	income_cat
0	8.3252	5
1	8.3014	5
2	7.2574	5
3	5.6431	4
4	3.8462	3
...	...	...
20635	1.5603	2
20636	2.5568	2
20637	1.7000	2
20638	1.8672	2
20639	2.3886	2

20640 rows × 2 columns

In [12]:
```
import numpy as np
housing['income_cat'] = pd.cut(housing['median_income'],
                               bins = [0, 1.5, 3, 4.5, 6, np.inf],
                               labels = [1, 2, 3, 4, 5])
housing['income_cat'].hist()
plt.show()
```



In [13]:
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.20, 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("Category proportion for overall dataset:\n", housing['income_cat'].value_
print("Category proportion for stratified sampling:\n", strat_train_set['income_
```

```
Category proportion for overall dataset:
 income_cat
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: count, dtype: float64
Category proportion for stratified sampling:
 income_cat
3    0.350594
2    0.318859
4    0.176296
5    0.114462
1    0.039789
Name: count, dtype: float64
```

In [80]:
```
#for set_ in (strat_train_set, strat_test_set):
#    set_.drop("income_cat", axis=1, inplace=True)
```

Explore the Data

Create a copy of the data for exploration (sampling it down to a manageable size if necessary).

In [ ]:
```
housing = strat_train_set.copy()
```

Study each attribute and its characteristics.

- Name
- Type (categorical, int/float, bounded/unbounded, text, structured, etc.)
- % of missing values
- Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)
- Usefulness for the task
- Type of distribution (Gaussian, uniform, logarithmic, etc.)

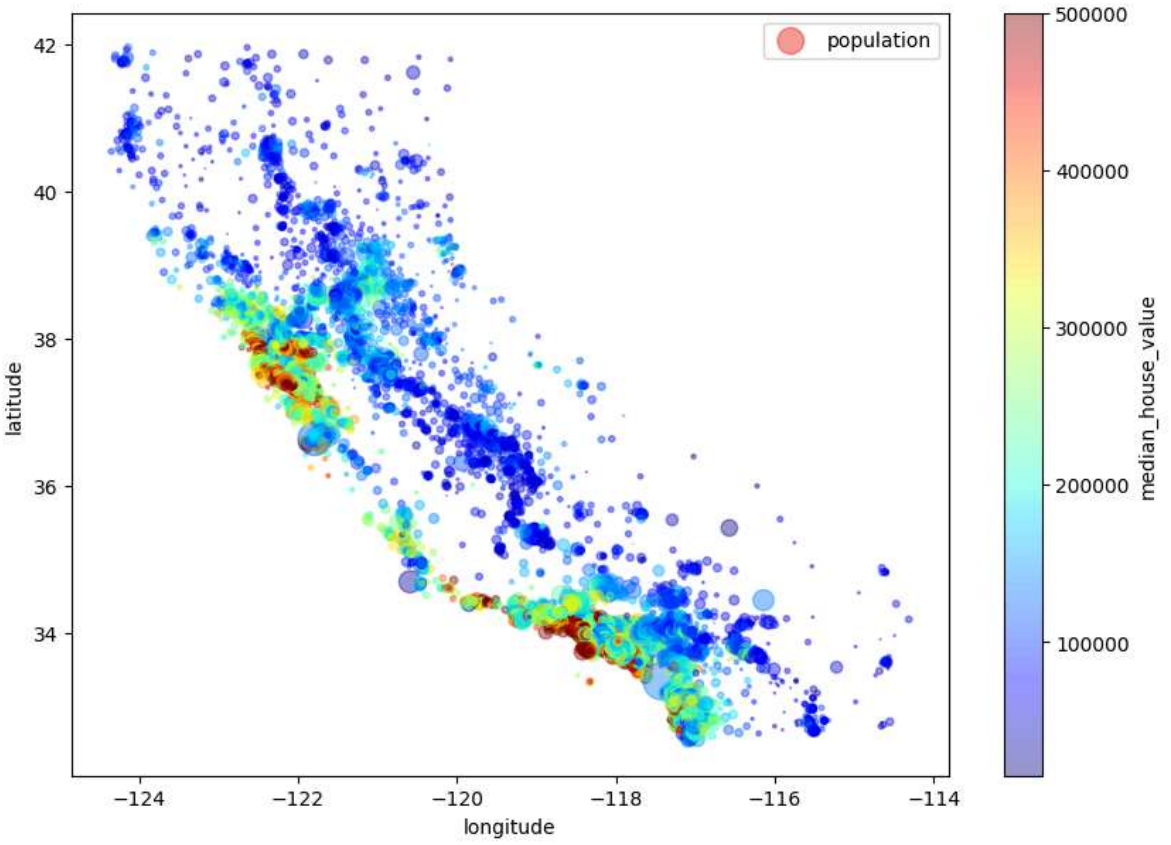
In [16]:
```
#READ ME > Dependences problem with Profile Report and pandas_profiling
#from pandas_profiling import ProfileReport
#profile = ProfileReport(housing, title="Pandas Profiling Report")
#profile.to_notebook_iframe()
```

For supervised learning tasks, identify the target attribute(s).

The target varable is housing_median_value column

Visualize the data

In [17]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.cm.jet, colorbar=True)
plt.legend()
plt.show()
```



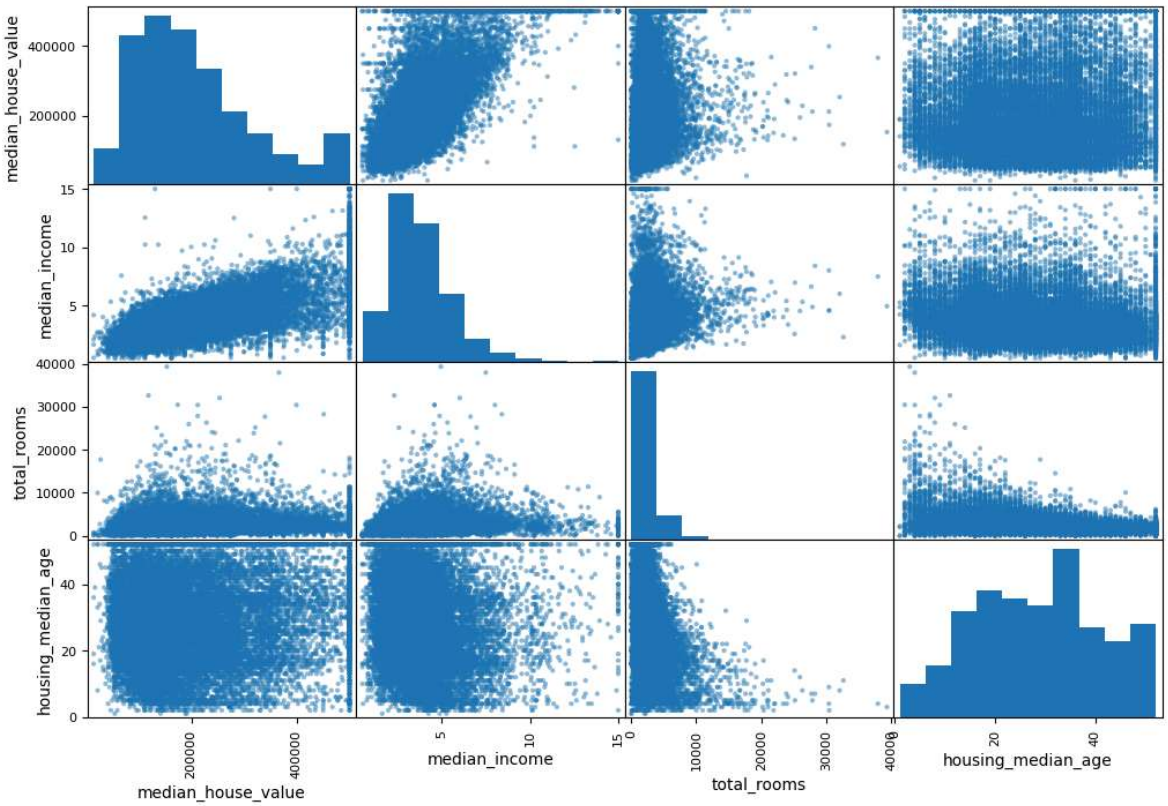
Study the correlations between attributes.

In [18]:
```
#corr column
corr_matrix = housing.select_dtypes(include=[float, int]).corr()
median_house_value_corr = corr_matrix["median_house_value"].sort_values(ascendin
print(median_house_value_corr)
```

```
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 [19]:
```
from pandas.plotting import scatter_matrix

attributes = median_house_value_corr.index[0:4]
```

```
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```



Study how you would solve the problem manually.

In [ ]:

Identify the promising transformations you may want to apply.

In [20]:
```
housing["rooms_per_household"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_per_household"] = housing["total_bedrooms"] / housing["total_r
housing["populations_per_household"] = housing["population"] / housing["househol

corr_matrix = housing.select_dtypes(include=['float64', 'int64']).corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[20]:
```
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
populations_per_household   -0.023737
population                  -0.024650
longitude                   -0.045967
latitude                    -0.144160
bedrooms_per_household      -0.255880
Name: median_house_value, dtype: float64
```

Identify extra data that would be useful.

In [ ]:

Document what you have learned.

Prepare the Data

- Work on copies of the data (keep the original dataset intact).
- Write functions for all data transformations you apply.
  - So you can easily prepare the data the next time you get a fresh dataset
  - So you can apply these transformations in future projects
  - To clean and prepare the test set
  - To clean and prepare new data instances once your solution is live
  - To make it easy to treat your preparation choices as hyperparameters

Data cleaning.

- Fix or remove outliers (optional)
- Fill in missing values (e.g., with zeros, mean, median...) or drop their rows (or columns)

In [21]:
```
housing = strat_train_set.drop ("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

In [22]:
```
#prepare data for missing data with the median of the column
#calculate median of each column
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy = "median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
print(imputer.statistics_)
```

```
[-118.51      34.26      29.      2119.       433.      1164.
  408.         3.54155]
```

Feature selection (optional).

- Drop the attributes that provide no useful information for the task
- Use some dimensionality reduction technique if necessary (PCA, KernelPCA, LLE...)

In [ ]:

Feature engineering, where appropriate.

- Discretize continuous features
- Decompose features (e.g., categorical, date/time, etc.)
- Add promising transformations of features (e.g., log(x), sqrt(x), x^2, etc.)
- Aggregate features into promising new features

Handling Text and Categorical Attributes (added after processing 4.3)

In [23]:
```
#manipulate columns for manage categorical var (like string)
housing_cat = housing[['ocean_proximity']].reset_index()
housing_cat.head()
```

Out[23]:

	index	ocean_proximity
0	12655	INLAND
1	15502	NEAR OCEAN
2	2908	INLAND
3	14053	NEAR OCEAN
4	20496	<1H OCEAN

In [24]:
```
#convert column in a float
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
print(housing_cat_1hot.toarray())
print(cat_encoder.categories_)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[array([    0,     1,     2, ..., 20636, 20637, 20638], shape=(16512,)), array
(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

Aggregate features into promising new features (after 4.3)

In [25]:
```
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class CombinedAttributeAdder(BaseEstimator, TransformerMixin):
    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

    def transform(self, X):
        # Supponendo che X sia un array numpy o DataFrame
        rooms_ix = housing.columns.get_loc("total_rooms")
```

```
        bedrooms_ix = housing.columns.get_loc("total_bedrooms")
        population_ix = housing.columns.get_loc("population")
        households_ix = housing.columns.get_loc("households")

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

# Uso corretto
att_adder = CombinedAttributeAdder(add_bedrooms_per_room=False)
housing_extra_attribs = att_adder.transform(housing_num.values)
```

In [26]:
```
housing_extra_attribs.shape
```

Out[26]:
```
(16512, 10)
```

Feauture scaling

- Standardize or normalize features

In [27]:
```
#scale features > set the same order of magnitude of the values
from sklearn.preprocessing import StandardScaler #we take the min e max of each
#keep attention to the ouliers
scaler = StandardScaler()
housing_extra_attribs_scaled = scaler.fit_transform(housing_extra_attribs)
housing_extra_attribs_scaled
```

Out[27]:
```
array([[-0.94135046,  1.34743822,  0.02756357, ..., -0.8936472 ,
         0.01739526,  0.00622264],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  1.292168  ,
         0.56925554, -0.04081077],
       [ 0.26758118, -0.1259716 ,  1.22045984, ..., -0.52543365,
        -0.01802432, -0.07537122],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ..., -0.36547546,
        -0.5092404 , -0.03743619],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.16826095,
         0.32814891, -0.05915604],
       [-1.28105026,  2.02567448, -0.13148926, ..., -0.390569  ,
         0.01407228,  0.00657083]], shape=(16512, 10))
```

Reconstruct data

- use pipelines to automate all steps if possible

In [28]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

#merge vars num and cat

num_attribs = housing_num.columns.tolist()
cat_attribs = ["ocean_proximity"]
```

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

full_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs)
])

housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

Out[28]:
```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]], shape=(16512, 16))
```

Shortlisting Promising Models

- If the data is huge, you may want to sample smaller training sets so you can train many different models in a reasonable time (be aware that this penalizes complex models such as large neural nets or Random Forests).
- Once again, try to automate these steps as much as possible.

Train many quick-and-dirty models from different categories (e.g., linear, naive Bayes, SVM, Random Forest, neural net, etc.) using standard parameters.

In [29]:
```
#test linear regression
from sklearn.linear_model import LinearRegression

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

Out[29]:
```
▾ LinearRegression
LinearRegression()
```

In [30]:
```
from sklearn.metrics import mean_squared_error #evaluate our model trough MSE
#MSE = index of how much the predicted values differ from those actually measure
```

```
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse

#result 68376.51$ > it is a high value, it indicates that the typical price of a
```

Out[30]: np.float64(68627.87390018745)

Decision Tree

In [31]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

Out[31]:
```
DecisionTreeRegressor()
```

In [32]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[32]: np.float64(0.0)

Random Forest

In [52]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

Out[52]:
```
RandomForestRegressor()
```

In [55]:
```
#the result performe better the the other 2 as the error result 1860$
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[55]: np.float64(18630.68352534902)

Measure and compare their performance.

- For each model, use N-fold cross-validation and compute the mean and standard deviation of the performance measure on the N folds.

use cross validation, train the model severl time in the same training set

In [ ]:

In [43]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

Liner Regression

In [44]:
```
from sklearn.model_selection import cross_val_score

lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels, scoring=
lin_rmse = np.sqrt(-lin_scores)
display_scores(lin_rmse)

#we can understand which is the avr error if we use a linear model
```

```
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.3282098180666
```

Decision Tree

In [45]:
```
tree_scores = cross_val_score(tree_reg, housing_prepared, housing_labels, scorin
tree_rmse = np.sqrt(-tree_scores)
display_scores(lin_rmse)

#values are highe than LR. Worst than the LR, we discover it trought the cross v
```

```
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.3282098180666
```

Random Forest

In [62]:
```
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels, sc
forest_rmse = np.sqrt(-forest_scores)
display_scores(forest_rmse)
```

```
Scores: [50102.0741734  49869.05173681 49607.3275937  51474.04373317
 51474.28850872]
Mean: 50505.35714915836
Standard deviation: 806.3697038431519
```

Analyze the most significant variables for each algorithm.

In [64]:
```
extra_attribs = ["rooms_per_household", "population_per_household", "bedrooms_pe
cat_one_hot_attribs = list(full_pipeline.named_transformers_["cat"].categories_[
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
def feature_importance(features, attributes):
    for f, a in sorted(zip(features, attributes), reverse=True):
        print(f"{a}\t{f}")
# This function will take the feature importance values
#(the model coefficients, e.g. from a regression or classification model)
#and their attribute names, sort them and print them.
```

Linear regression

In [66]:
```
feature_importance(lin_reg.coef_, attributes)
```

```
ISLAND  110357.84610619607
median_income   74714.15226132619
households      45453.262776622236
housing_median_age      13734.720841922466
bedrooms_per_room       9248.31607776975
total_bedrooms  7343.229797309003
rooms_per_household     6604.583966284028
population_per_household        1043.0545298050245
total_rooms     -1943.0558635496077
NEAR OCEAN      -14642.48658971256
<1H OCEAN       -18015.98870783896
NEAR BAY        -22484.659973912247
population      -45709.28253579062
INLAND  -55214.71083473229
longitude       -55649.63398452769
latitude        -56711.597428916226
```

Decision Tree

In [68]:
```
feature_importance(tree_reg.feature_importances_, attributes)
```

```
median_income   0.4711388741315356
INLAND  0.14028383402268668
population_per_household        0.12306688900978814
longitude       0.06690612250975332
latitude        0.05549807920961478
housing_median_age      0.040750777668199754
rooms_per_household     0.027418904822986378
bedrooms_per_room       0.02244549043647493
total_rooms     0.014020281372469084
total_bedrooms  0.01261647369219833
population      0.011714992835345158
households      0.01132360497759157
NEAR OCEAN      0.001525931342696223
<1H OCEAN       0.0007416404400118311
NEAR BAY        0.0005481035286482879
ISLAND  0.0
```

Random Forest

In [69]:
```
feature_importance(forest_reg.feature_importances_, attributes)
```

```
median_income   0.47272972583789374
INLAND  0.14095321915497508
population_per_household        0.12253267806413917
longitude       0.05855025662995198
latitude        0.05501566229149033
housing_median_age      0.04535718270810775
rooms_per_household     0.02731760454270308
bedrooms_per_room       0.023485283386364762
total_rooms     0.013033645268098384
households      0.012751413324092128
total_bedrooms  0.011903443308400903
population      0.01188851020069625
NEAR OCEAN      0.002273475756591268
<1H OCEAN       0.0012537988930384704
NEAR BAY        0.0009204796736629117
ISLAND  3.3620959793872865e-05
```

Analyze the types of errors the models make.

- What data would a human have used to avoid these errors?

In [ ]:

Perform a quick round of feature selection and engineering.

In [ ]:

Perform one or two more quick iterations of the five previous steps.

In [ ]:

Shortlist the top three to five most promising models, preferring models that make different types of errors.

In [ ]:

Fine-Tune the System

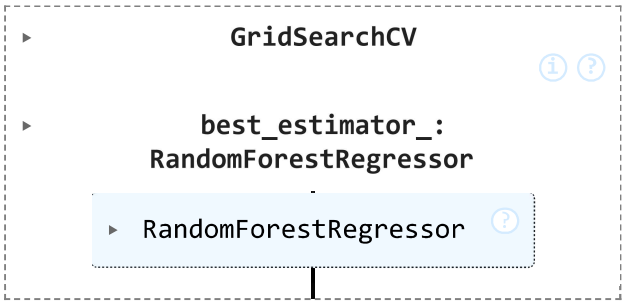
- You will want to use as much data as possible for this step, especially as you move towards the end of fine-tuning.
- As always automate what you can.

Fine-tune the hyperparameters using cross-validation.

- Treat your data transformation choices as hyperparameters, especially when you are not sure about them (e.g., if you are not sure whether to replace missing values with zeros or with the median value, or to just drop the rows).
- Unless there are very few hyperparameter values to explore, prefer random search over grid search. If training is very long, you may prefer a Bayesian optimization approach (e.g., using Gaussian process priors).

In [71]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 5, 8]},  # Search combi
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}
]

forest_reg = RandomForestRegressor()  # Initialize the RandomForestRegressor mod

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,  # Perform grid search
                           scoring="neg_mean_squared_error",  # Use negative mean
                           return_train_score=True)  # Return training scores in

grid_search.fit(housing_prepared, housing_labels)  # Fit the grid search model t
```

Out[71]:



In [ ]:
```
```

Try Ensemble methods. Combining your best models will often produce better performance than running them individually.

In [74]:
```
#see the best value where the model performed better
print(grid_search.best_params_)
print(grid_search.best_estimator_)
```

```
{'max_features': 8, 'n_estimators': 30}
RandomForestRegressor(max_features=8, n_estimators=30)
```

In [75]:
```
cv_result = grid_search.cv_results_  # Get the results of the grid search
for mean_score, param in zip(cv_result["mean_test_score"], cv_result["params"]):
    print(np.sqrt(-mean_score), param)  # Print the root mean square error and t
```

```
64089.457203853715 {'max_features': 2, 'n_estimators': 3}
55580.43729729614 {'max_features': 2, 'n_estimators': 10}
52761.99507495128 {'max_features': 2, 'n_estimators': 30}
60819.42735906754 {'max_features': 4, 'n_estimators': 3}
52388.478649513825 {'max_features': 4, 'n_estimators': 10}
50503.03535969599 {'max_features': 4, 'n_estimators': 30}
60315.25918493064 {'max_features': 5, 'n_estimators': 3}
52661.41687040495 {'max_features': 5, 'n_estimators': 10}
50066.194146796704 {'max_features': 5, 'n_estimators': 30}
58862.71732565496 {'max_features': 8, 'n_estimators': 3}
52207.69333754847 {'max_features': 8, 'n_estimators': 10}
50024.850289382215 {'max_features': 8, 'n_estimators': 30}
62849.232277177434 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54038.73138132223 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59469.71202260795 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52705.331536278405 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58778.28602101022 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51214.33859822695 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Once you are confident about your final model, measure its performance on the test set to estimate the generalization error.

- Don't tweak your model after measuring the generalization error: you would just start overfitting the test set.

In [76]:
```
final_model = grid_search.best_estimator_  # Get the best model from the grid se

# Fix typo in variable names
X_test = strat_test_set.drop("median_house_value", axis=1)  # Drop the target co
y_test = strat_test_set["median_house_value"].copy()  # Corrected typo here: 'st

# Prepare the test data using the same pipeline
X_test_prepared = full_pipeline.transform(X_test)

# Make predictions using the final model
final_predictions = final_model.predict(X_test_prepared)

# Calculate the Mean Squared Error and then the Root Mean Squared Error
final_mse = mean_squared_error(y_test, final_predictions)
final_mse = np.sqrt(final_mse)

# Print the RMSE
print(final_mse)
```

```
47572.05614017531
```

CONCLUSION

Our odel perform with an error of 47572 $. It perform not so good, but we can try other model and combine or add another features or paraeters.

Present Your Solution

Document what you have down.

Create a nice presentation.

- Make sure you highlight the big picture first.

Explain why your solution achieves the business objective.

Don't forget to present interesting points you noticed along the way.

- Describe what worked and what did not.
- List your assumptions and your system's limitations.

Ensure your key findings are communicated through beautiful visalizations or easy-to-remember statements.

Launch!

Get your solution ready for production (plug into production data inputs, write unit tests, etc.).

Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.

- Beware of slow degradation: models tend to "rot" as data evolves.
- Measuring performance may require a human pipeline (e.g., via a crowdsourcing service).

- Also monitor your inputs' quality (e.g., a malfunctioning sensor sending random values, or another team's output becoming stale). This is particularly important for online learning systems.

Retrain your models on regular basis on fresh data (automate as much as possible).