End-to-End Machine Learning Project

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9.

The 7 Steps of Machine Learning

- YouTube Video: *The 7 Steps of Machine Learning* from Google Cloud Platform

https://youtu.be/nKW8Ndu7Mjw

Caution: Alcohol is forbidden in the Islamic religion and causes addiction and has negative effects on health.

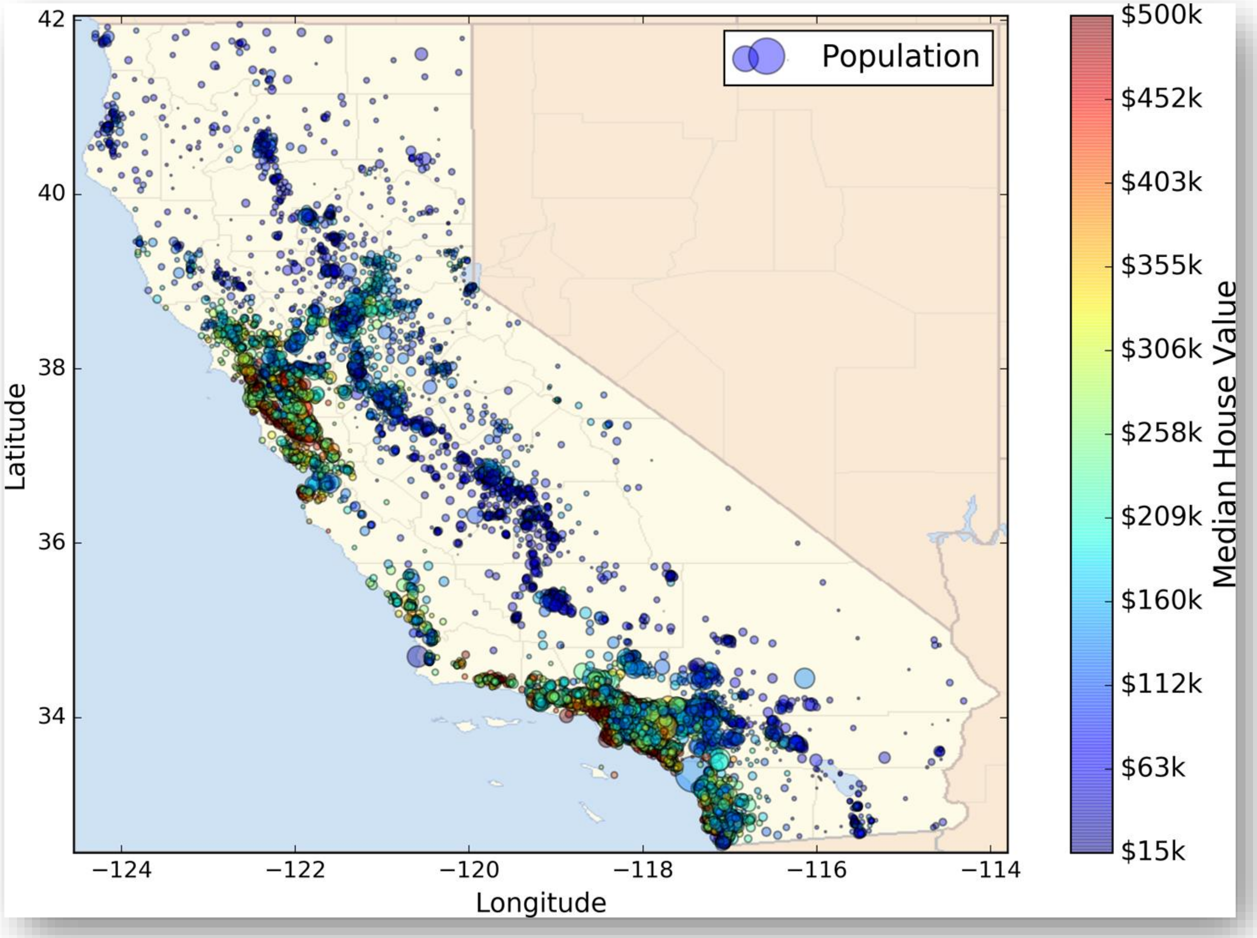
Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

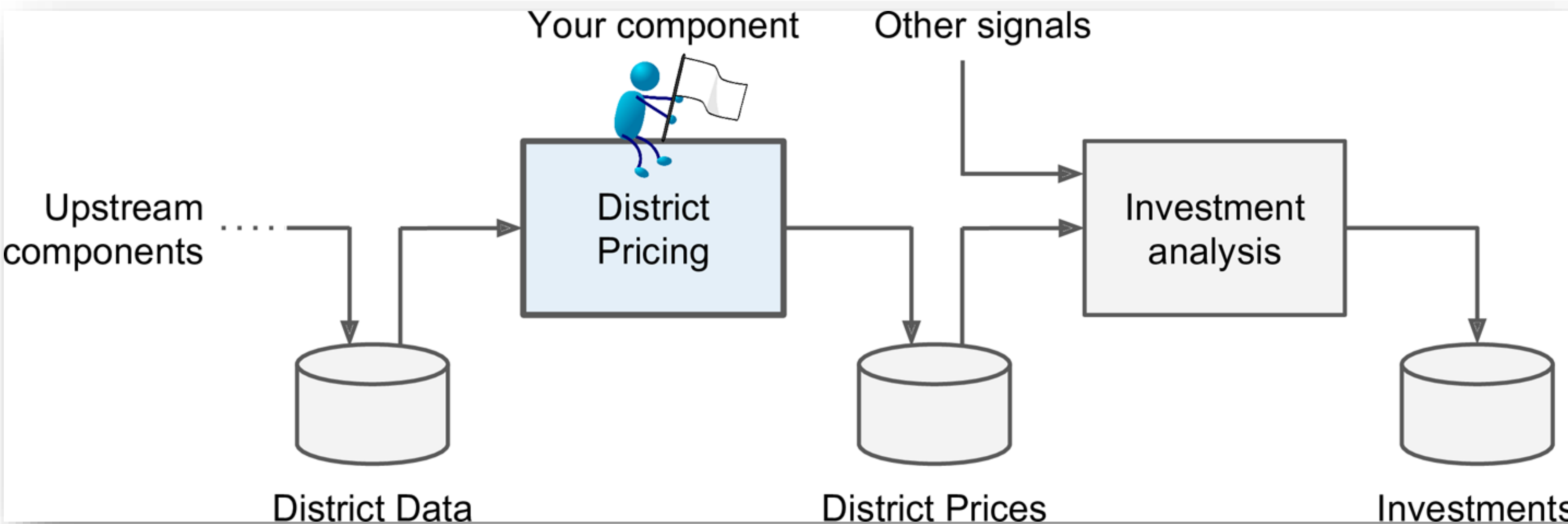
Working with Real Data

- Popular open data repositories:
  - UC Irvine Machine Learning Repository
  - Kaggle datasets
  - Amazon's AWS datasets
- Meta portals (they list open data repositories):
  - http://dataportals.org/
  - http://opendatamonitor.eu/
  - http://quandl.com/
- Other pages listing many popular open data repositories:
  - Wikipedia's list of Machine Learning datasets
  - Quora.com question
  - Datasets subreddit
- Abandah has handwritten Arabic samples and diacritized Arabic text.

1. Look at the Big Picture: CA Housing Data

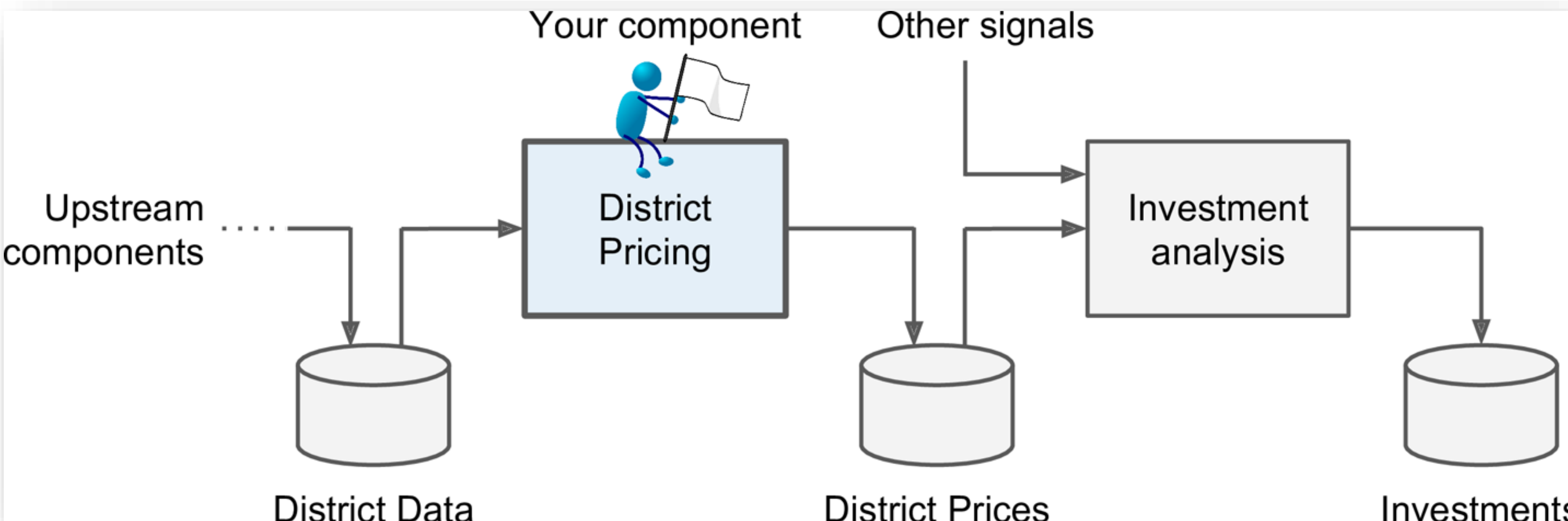


1.1. Frame the Problem



Is it supervised, unsupervised, or Reinforcement Learning?
Is it a classification task, a regression task, or something else?
Should you use batch learning or online learning techniques?
Instance-based or Model-based learning?

1.1. Frame the Problem



Is it **supervised**, unsupervised, or Reinforcement Learning?
Is it a classification task, a **regression** task, or something else?
Should you use **batch** learning or online learning techniques?
Instance-based or **Model-based** learning?

1.2. Select a Performance Measure

- **Root Mean Square Error (RMSE)**

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

- m is the number of samples
- $\mathbf{x}^{(i)}$ is the feature vector of Sample i
- $y^{(i)}$ is the label or desired output
- $\mathbf{X}$ is a matrix containing all the feature values

$$\mathbf{X} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^T \\ \left(\mathbf{x}^{(2)}\right)^T \\ \vdots \\ \left(\mathbf{x}^{(1999)}\right)^T \\ \left(\mathbf{x}^{(2000)}\right)^T \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

1.2. Select a Performance Measure

- **Mean Absolute Error**

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

- MAE is better than RMSE when there are outlier samples.

Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

2. Get the Data

- If you didn't do it before, it is time now to download the Jupyter notebooks of the textbook from

  https://github.com/ageron/handson-ml

- Start Jupyter notebook and open Chapter 2 notebook.
- Hint: If you get kernel connection problem, try

  ```
  C:\>jupyter notebook –port 8889
  ```

- The following slides summarize the code used in this notebook.

2. Get the Data

1. Download the *housing.tgz* file from *Github* using `urllib.request.urlretrieve()` from the *six.moves* package
2. Extract the data from this compressed tar file using `tarfile.open()` and `extractall()`. The data will be in the CSV file *housing.csv*
3. Read the CSV file into a Pandas DataFrame called *housing* using `pandas.read_csv()`

2.1. Take a Quick Look at the Data Structure

- Display the top five rows using the DataFrame's `head()` method
- The `info()` method is useful to get a quick description of the data
- To find categories and repetitions of some column use `housing.['key'].value_counts()`
- The `describe()` method shows a summary of the numerical attributes.
- Show histogram using the `hist()` method and `matplotlib.pyplot.show()`

In [6]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

207 missing features

```
>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

2.2. Create a Test Set

- Split the available data randomly to:
  - Training set (80%)
  - Test set (20%)
- The example defines a function called `split_train_test()` for illustration.
- Scikit-Learn has `train_test_split()`.
- Scikit-Learn also has `StratifiedShuffleSplit()` that does stratified sampling.
- **Stratification** ensures that the test samples are representative of the target categories.

2.2.1. Create a Test Set: User-defined function

```
import numpy as np

def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * 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]
```

You can then use this function like this:

```
>>> train_set, test_set = split_train_test(housing, 0.2)
>>> print(len(train_set), "train +", len(test_set), "test")
16512 train + 4128 test
```

2.2.2. Create a Test Set: Using *Scikit-Learn* functions

```
from sklearn.model_selection import train_test_split

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

Stratification is usually done on the target class.

```
from sklearn.model_selection import StratifiedShuffleSplit

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]
```

Outline

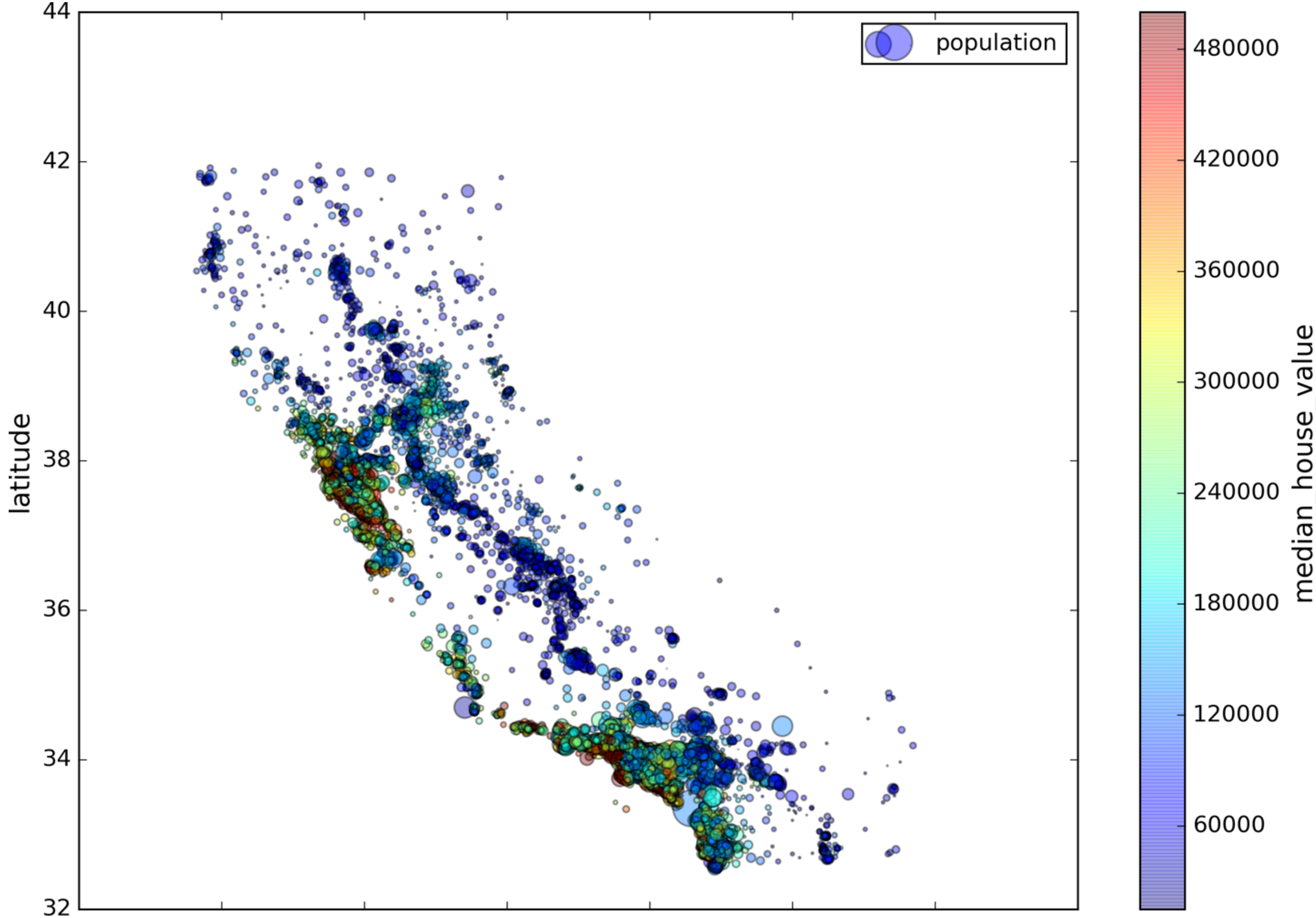
1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

3. Discover and Visualize the Data to Gain Insights

- Visualize geographical data using

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

`alpha`: Transparency, **`s`**: size, **`c`**: color, **`cmap`**: blue to red


population
latitude
median_house_value
44
42
40
38
36
34
32
480000
420000
360000
300000
240000
180000
120000
60000

3.1. Looking for Correlations

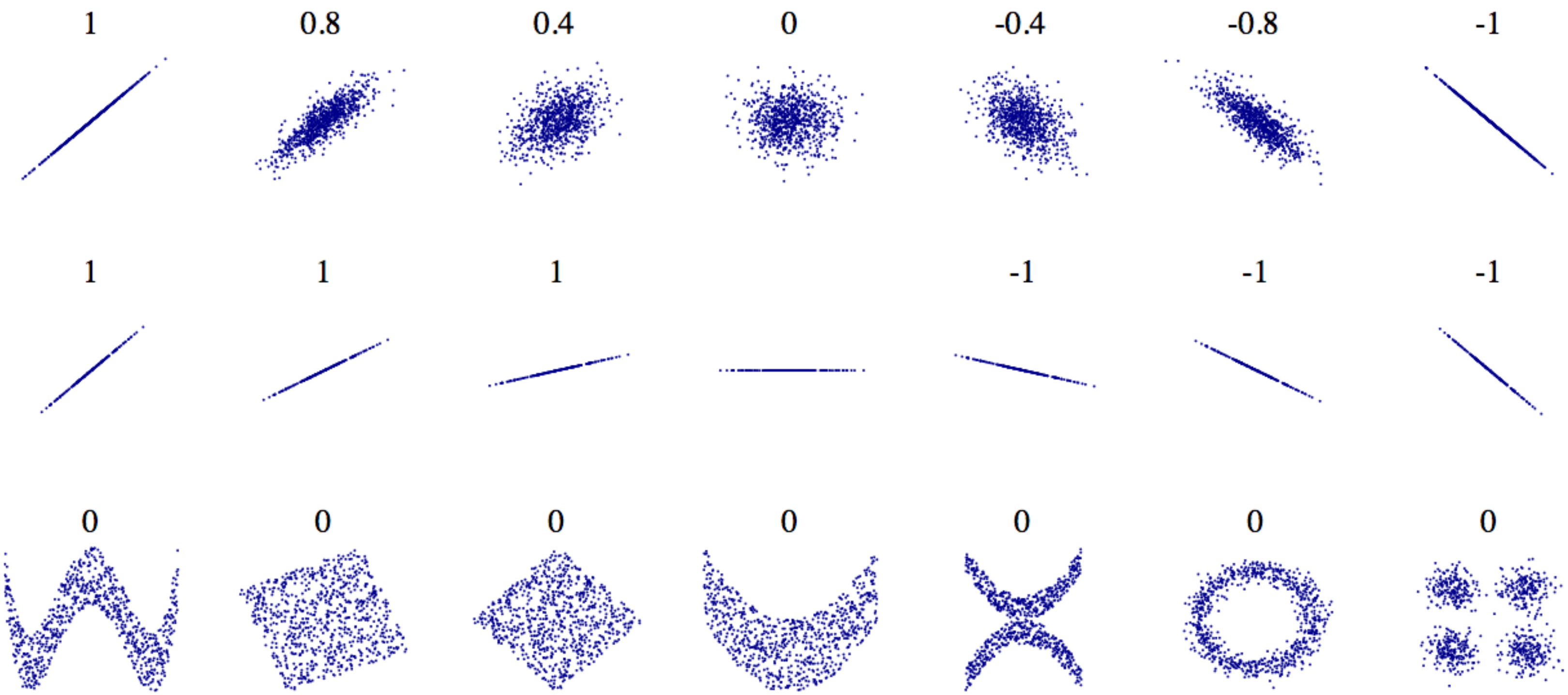
- Compute the *standard correlation coefficient* (also called *Pearson's r*) between every pair of attributes using `corr_matrix = housing.corr()`

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.687170
total_rooms           0.135231
housing_median_age    0.114220
households            0.064702
total_bedrooms        0.047865
population           -0.026699
longitude            -0.047279
latitude             -0.142826
```

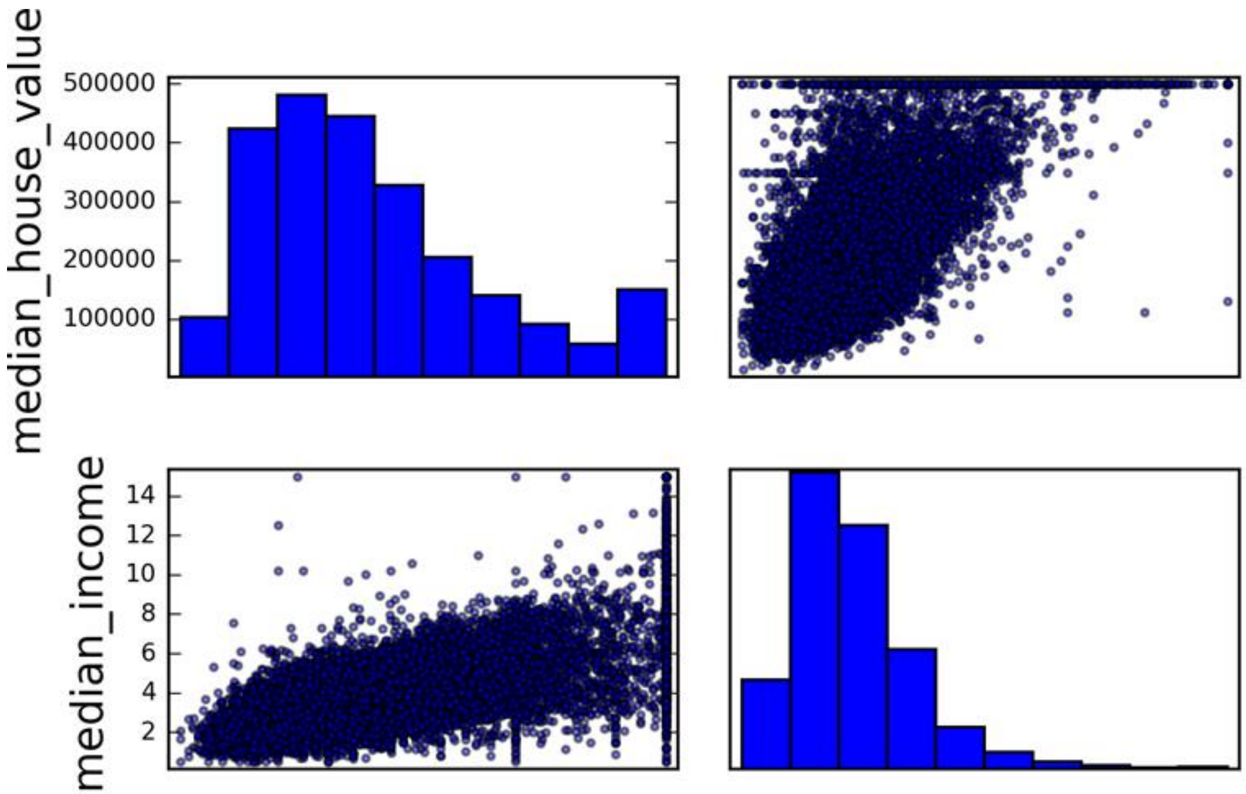
3.1. Looking for Correlations

- Zero linear correlation ($r = 0$) does not guarantee independence.



3.2. Pandas Scatter Matrix

```python
from pandas.tools.plotting import scatter_matrix
attributes = ["median_house_value", "median_income"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```



3.3. Experimenting with Attribute Combinations

- Rooms per household is better than total rooms:

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
```

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value          1.000000
median_income               0.687170
rooms_per_household         0.199343
total_rooms                 0.135231
```

- Similarly, BMI is better than weight or height for medical purposes.

Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

4. Prepare the Data for Machine Learning Algorithms

- Separate the features from the response.

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

- Options of handling missing features:
  1. Get rid of the corresponding districts
  2. Get rid of the whole attribute
  3. Set the values to some value (0, mean, median, etc.)

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median)     # option 3
```

4.1. Handling Missing Features Using *Scikit-Learn*

- Use `Imputer` on the numerical features. Need to remove categorical variables before doing the fit. The attribute `statistics_` has the means.

```
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
>>> imputer.statistics_
array([ -118.51 , 34.26 , 29. , 2119. , 433. , 1164. , 408. , 3.5414])
>>> housing_num.median().values
array([ -118.51 , 34.26 , 29. , 2119. , 433. , 1164. , 408. , 3.5414])
X = imputer.transform(housing_num)
```

4.2. Handling Text and Categorical Attributes

- Most machine learning algorithms prefer to work with numbers.
- Need to convert the feature *ocean_proximity* from text to numbers.

```
>>> from sklearn.preprocessing import LabelEncoder
>>> encoder = LabelEncoder()
>>> housing_cat = housing["ocean_proximity"]
>>> housing_cat_encoded = encoder.fit_transform(housing_cat)
>>> housing_cat_encoded
array([1, 1, 4, ..., 1, 0, 3])

>>> print(encoder.classes_)
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

Numerical values imply distances

4.2. Handling Text and Categorical Attributes

- To ensure encoding neutrality, we can use the one-hot encoding.

```
>>> from sklearn.preprocessing import LabelBinarizer
>>> encoder = LabelBinarizer()
>>> housing_cat_1hot = encoder.fit_transform(housing_cat)
>>> housing_cat_1hot
array([[0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1],
       ...,
       [0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0]])
```

4.3. Custom Transformers

- Scikit-Learn allows you to create your own transformers.
- You can create a transformer to create derived features.
- Create a class and implement three methods: fit() (returning self), transform(), and fit_transform(). Include base classes:
  - *TransformerMixin* to get fit_transform()
  - *BaseEstimator* to get get_params() and set_params()

4.3. Custom Transformers

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, household_ix = 3, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
    return np.c_[X, rooms_per_household]

attr_adder = CombinedAttributesAdder()
housing_extra_attribs = attr_adder.transform(housing.values)
```

4.4. Feature Scaling

- ML algorithms generally don't perform well when the input numerical attributes have very different scales.

- Scaling techniques:
  - *Min-max scaling*

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  - *Standardization*.

$$x' = \frac{x - \bar{x}}{\sigma}$$

4.5. Transformation Pipelines


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

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

4.6. Pipeline Unions

```
from sklearn.pipeline import FeatureUnion
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', Imputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),])
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('label_binarizer', LabelBinarizer()),])
full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),])
>>> housing_prepared = full_pipeline.fit_transform(housing)
```

Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

5. Select and Train a Model

- Let us start by training a simple **linear regressor**.

```
from sklearn.linear_model import LinearRegression

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

- Try it out on five instances from the training set.

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:\t", lin_reg.predict(some_data_prepared))
Predictions:     [ 303104.   44800.   308928.   294208.   368704.]
>>> print("Labels:\t\t", list(some_labels))
Labels:          [359400.0, 69700.0, 302100.0, 301300.0, 351900.0]
```

50% off

5.1. Evaluate the Model on the Entire Training Set

- Use RMSE

```
>>> from sklearn.metrics import mean_squared_error
>>> housing_predictions = lin_reg.predict(housing_prepared)
>>> lin_mse = mean_squared_error(housing_labels, housing_predictions)
>>> lin_rmse = np.sqrt(lin_mse)
>>> lin_rmse
68628.413493824875
```

This is not a satisfactory result as the *median_housing_values* range between $120,000 and $265,000.

5.2. Try the Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

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

>>> housing_predictions = tree_reg.predict(housing_prepared)
>>> tree_mse = mean_squared_error(housing_labels, housing_predictions)
>>> tree_rmse = np.sqrt(tree_mse)
>>> tree_rmse
0.0
```

Overfitting: It has memorized the entire training set!

5.3. Better Evaluation Using Cross-Validation

- Segment the training data into 10 sets and repeat training and evaluation 10 times.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
```

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(rmse_scores)
Scores: [ 74678.4916885   64766.2398337  …  ]
Mean: 71199.4280043
Standard deviation: 3202.70522793
```

Worse than Linear Regressor

5.4. Try the Random Forests Regressor

- Repeating training and evaluation:

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> forest_reg = RandomForestRegressor()
>>> forest_reg.fit(housing_prepared, housing_labels)
>>> [...]
>>> forest_rmse
22542.396440343684
>>> display_scores(forest_rmse_scores)
Scores: [ 53789.2879722   50256.19806622   …   ]
Mean: 52634.1919593
Standard deviation: 1576.20472269
```

Best Accuracy

Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

6. Fine-Tune Your Model

- Fine-tune your system by fiddling with:
  - The hyperparameters
  - Removing and adding features
  - Changing feature preprocessing techniques
- Can experiment manually. But it is best to automate this process using Scikit-Learn:
  - *GridSearchCV*
  - or *RandomizedSearchCV*

6.1. Grid Search

- Can automate exploring a search space of $3 \times 4 + 2 \times 3 = 12 + 6 = 18$

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error')

grid_search.fit(housing_prepared, housing_labels)
```

6.2 Examine the Results of Your Grid Search

- Can examine the best hyperparameters using:

```
>>> grid_search.best_params_
{'max_features': 6, 'n_estimators': 30}
```

- Can examine all search results using:

```
>>> cvres = grid_search.cv_results_
... for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
...     print(np.sqrt(-mean_score), params)
...
64912.0351358 {'max_features': 2, 'n_estimators': 3}
55535.2786524 {'max_features': 2, 'n_estimators': 10}
      …
49958.9555932 {'max_features': 6, 'n_estimators': 30}
```

Best Tuned Accuracy

6.2 Evaluate Your System on the Test Set

- The final model is the best estimator found by the grid search.
- To evaluate it on the test set, transform the test features, predict using transformed features, and evaluate accuracy.

Better than train set!

```
final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)    # => evaluates to 48,209.6
```

6.3 Save Your Best Model for the Production System

```
from sklearn.externals import joblib

joblib.dump(my_model, "my_model.pkl")
# and later...
my_model_loaded = joblib.load("my_model.pkl")
```

Outline

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

7. Present Your Solution

- Present your solution highlighting:
  - What you have learned
  - What worked and what did not
  - What assumptions were made
  - What your system's limitations are
- Document everything, and create nice presentations with:
  - Clear visualizations
  - Easy-to-remember statements, e.g., "the median income is the number one predictor of housing prices".

8. Launch, Monitor, and Maintain Your System

- Prepare your production program that uses your best trained model and launch it.
- Monitor the accuracy of your system. Also monitor the input data.
- Retrain your system periodically using fresh data.

Summary

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system
9. Exercises

Exercise

- Try a Support Vector Machine regressor (sklearn.svm.SVR), with various hyperparameters such as kernel="linear" (with various values for the C hyperparameter) or kernel="rbf" (with various values for the C and gamma hyperparameters). Don't worry about what these hyperparameters mean for now. How does the best SVR predictor perform?