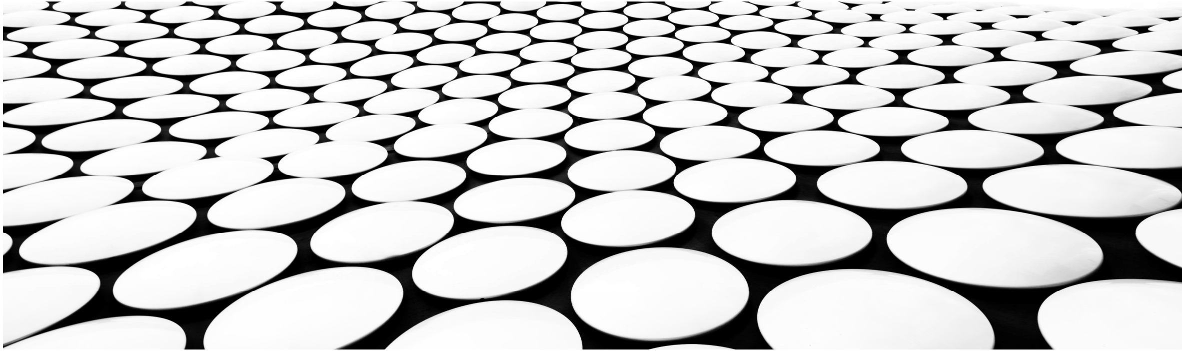
CSE353 – MACHINE LEARNING
END-TO-END ML PROJECT

PRAVIN PAWAR, SUNY KOREA

BASED ON CHAPTER 2 – HANDS-ON ML WITH SCIKIT-LEARN, KERAS AND TENSORFLOW BY AURÉLIEN GÉRON



MAIN STEPS IN THE MACHINE LEARNING PROJECT

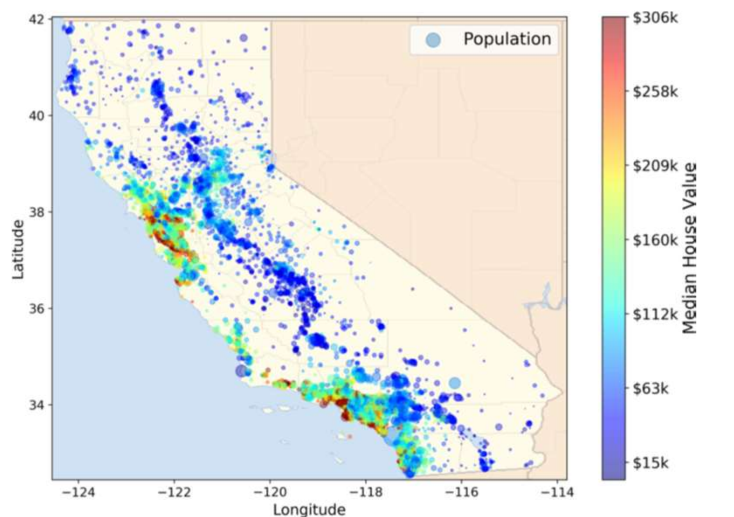
- Formulate the problem
- Data collection
- Data visualization
- Data preprocessing
- Selection of ML models
- Finetune the model
- Present solution
- Launch, monitor and maintain the system

WHERE TO FIND DATASETS FOR EXPERIMENTS?

- Popular open data repositories
  - UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php)
  - CMU Statlib Repository (http://lib.stat.cmu.edu/datasets/)
  - Kaggle datasets (https://www.kaggle.com/datasets/)
  - Amazon's AWS datasets (https://registry.opendata.aws/)
- Meta portals which list open data repositories
  - Data Portals (http://dataportals.org/)
  - OpenDataMonitor (https://www.opendatamonitor.eu/)
  - Quandl (https://www.quandl.com/)
- Other pages listing many popular open data repositories
  - Wikipedia's list of Machine Learning datasets (https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research)
  - Quora.com (https://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public)
  - The datasets subreddit (https://www.reddit.com/r/datasets)

3

CALIFORNIA HOUSING DATASET – STATSLIB REPOSITORY



- The goal is to use California census data to build a model of housing prices in the state
- This data includes metrics such as the population, median income, and median housing price for each block group in California
- Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data
- A block group typically has a population of 600 to 3,000 people
- Also known as "districts"

- The original dataset appeared in R. Kelley Pace and Ronald Barry, "Sparse Spatial Autoregressions," Statistics & Probability Letters 33, no. 3 (1997): 291–297.

4

CALIFORNIA HOUSING DATASET – STATSLIB REPOSITORY

```
housing = load_housing_data()
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

```
housing.info()
```

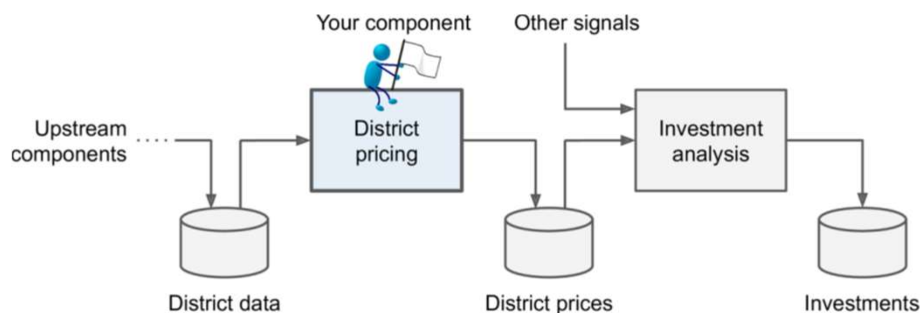
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        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
```

- A measure of how far west a house is
- A measure of how far north a house is
- Median age of a house within a block
- Total number of rooms within a block
- Total number of bedrooms within a block
- Total number of people residing within a block
- Total number of households, a group of people residing within a home unit, for a block
- Median income for households within a block of houses (10000 US$)
- Median house value for households within a block (USD)
- Location of the house w.r.t ocean/sea

5

PROBLEM FORMULATION – REQUIREMENTS ANALYSIS

- As per the broader scope, your solution will be one of the components in a larger picture for investment analysis
- Current solutions use manual estimate using median housing price or using complex rules, and not as accurate
- Manual estimates are more than 20% off



6

PROBLEM FORMULATION – WHICH TYPE OF ML PROBLEM IT IS?

- The problem is to predict the median housing price of houses in California per block
- Which type of ML problem it is?
- Supervised, unsupervised, or Reinforcement Learning?
- Classification task, regression task, or something else?
- Should you use batch learning or online learning technique?

7

PROBLEM FORMULATION – WHICH TYPE OF ML PROBLEM IT IS?

- Supervised learning task as labeled training examples are given
- It is a regression task as it requires predicting a numerical value
- Multiple regression problem as the system uses multiple features to make a prediction
- Univariate regression problem as task involves predicting a single value for each district
- It would be multivariate regression if required to predict multiple values per district
- Since all data is available and small enough to fit in memory, batch learning is a suitable approach

8

PERFORMANCE MEASURE

- For regression problems, a typical performance measure is Root Mean Square Error (RMSE)
- Represents errors in predictions with higher weight for larger errors
- Mathematical formula: $\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$
- *h* is your system's prediction function, also called a hypothesis
- *RMSE(X,h)* is the cost function measured on the set of examples *X* using your hypothesis *h*
- *m:* Number of instances in the dataset
- X^i is a vector of all the feature values (excluding the label) of the ^{i}th instance in the dataset
- Y^i is the label (desired output) of the ^{i}th instance in the dataset

9

FETCH THE DATA USING A FUNCTION

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

10

LOAD THE DATA AND LOOK AT THE STRUCTURE

```
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()
housing.info()
```

11

LOAD THE DATA AND LOOK AT THE STRUCTURE

```
housing = load_housing_data()
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

```
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
```

- A measure of how far west a house is
- A measure of how far north a house is
- Median age of a house within a block
- Total number of rooms within a block
- Total number of bedrooms within a block
- Total number of people residing within a block
- Total number of households, a group of people residing within a home unit, for a block
- Median income for households within a block of houses (10000 US$)
- Median house value for households within a block (USD)
- Location of the house w.r.t ocean/sea

12

USE VALUE COUNTS() FOR CATEGORICAL VARIABLE DESCRIBE() SHOWS A SUMMARY OF NUMERICAL ATTRIBUTES

```
housing["ocean_proximity"].value_counts()
```

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

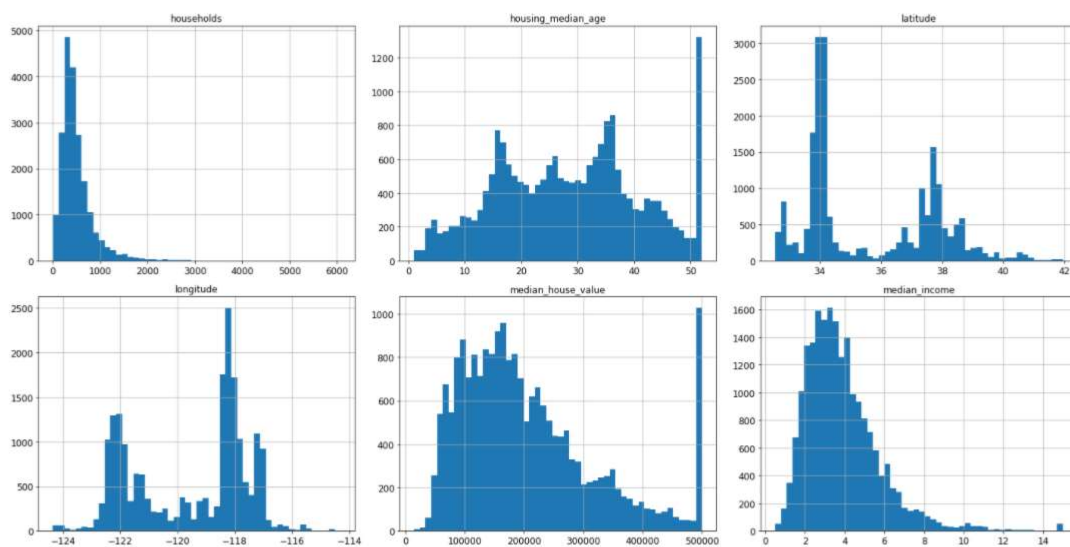
```
housing.describe()
```

	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

13

HISTOGRAM IS ANOTHER WAY TO LOOK AT NUMERICAL DATA

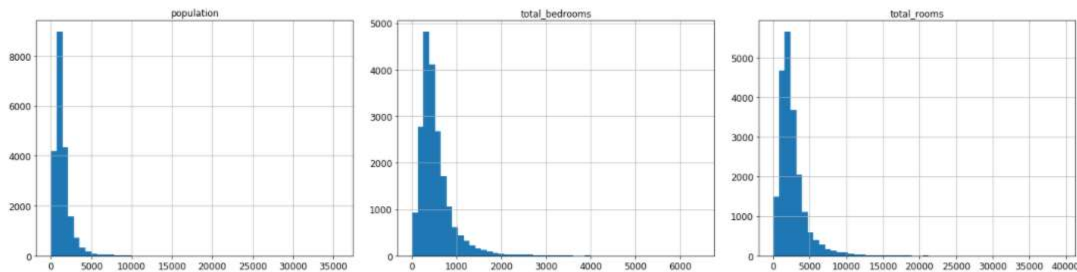
```
%matplotlib inline   # only in a Jupyter notebook
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



14

HISTOGRAM IS ANOTHER WAY TO LOOK AT NUMERICAL DATA

```
%matplotlib inline   # only in a Jupyter notebook
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



15

SPLIT DATA INTO TRAINING AND TESTING SET

```
# to make this notebook's output identical at every run
np.random.seed(42)
```

```
import numpy as np

# For illustration only. Sklearn has train_test_split()
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]
```

```
train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
```

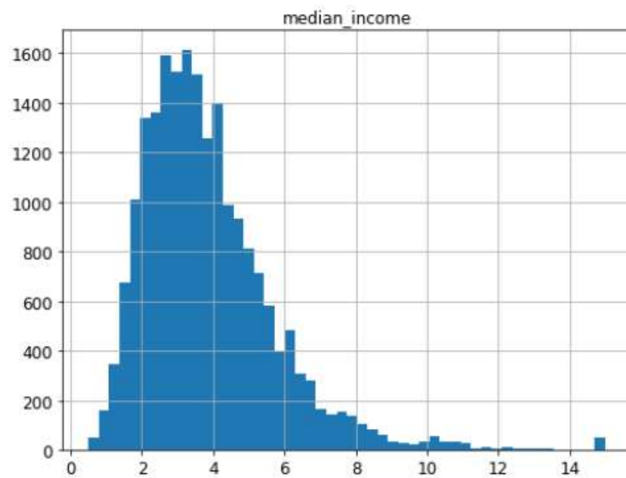
16512

```
len(test_set)
```

4128

16

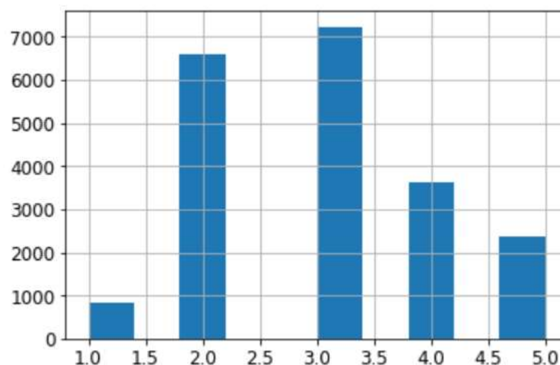
STRATIFIED SAMPLING AS PER MEDIAN INCOME



- What if median income is a very important attribute for predicting median housing prices?
- Then test set should be representative of various categories of incomes
- Most median income values are clustered around 1.5 to 6 (i.e., $15,000–$60,000), but some median incomes go far beyond 6
- It is important to have a sufficient number of instances in the dataset
- There should not have too many strata, and each stratum should be large enough

17

STRATIFIED SAMPLING AS PER MEDIAN INCOME



- Solution: Create income category attribute with suitable categories
- The pd.cut() function creates categories with specified ranges as follows:

```
housing["income_cat"] =
    pd.cut(housing["median_income"],
    bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
    labels=[1, 2, 3, 4, 5])
housing["income_cat"].value_counts()
```

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

18

DOING STRATIFIED SAMPLING WITH SCIKIT-LEARN

```
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]
```

```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

19

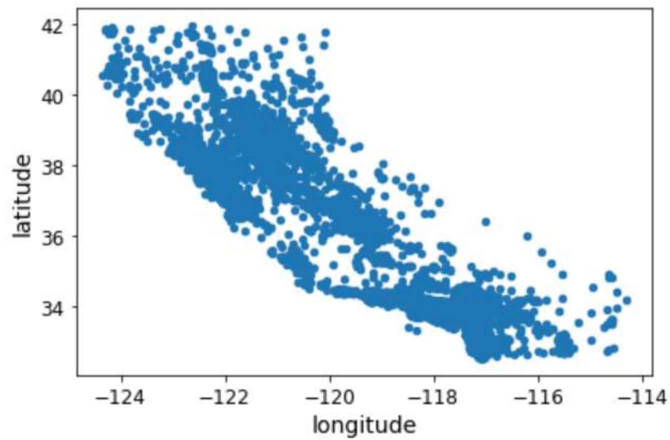
STRATIFIED SAMPLING IS BETTER THAN RANDOM SAMPLING

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

20

DATASET VISUALIZATION

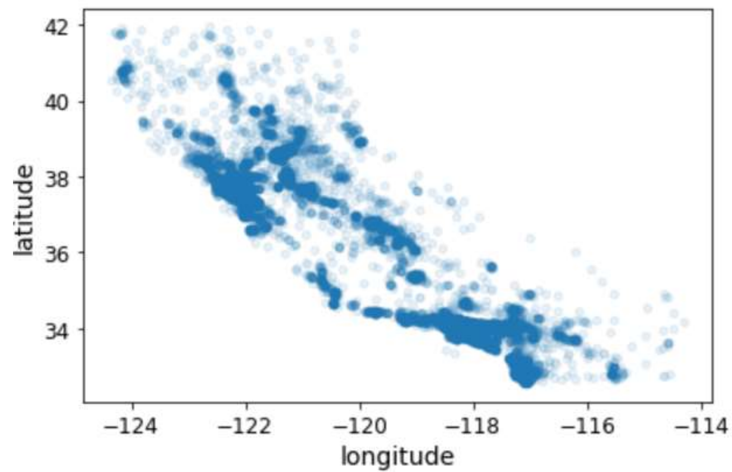
```
housing = strat_train_set.copy()
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```



21

VISUALIZE PLACES WITH HIGH DENSITY

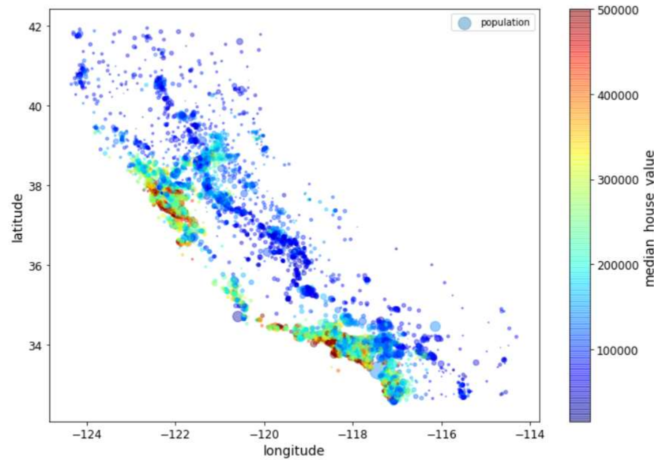
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```



22

SCATTERPLOT OF HOUSING PRICES

```
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.get_cmap("jet"), colorbar=True, sharex=False)
```

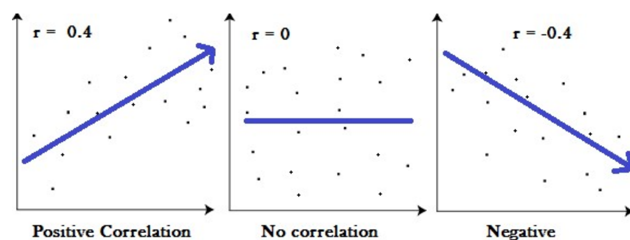


23

CORRELATION OF MEDIAN HOUSE INCOME WITH OTHER ATTRIBUTES

```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

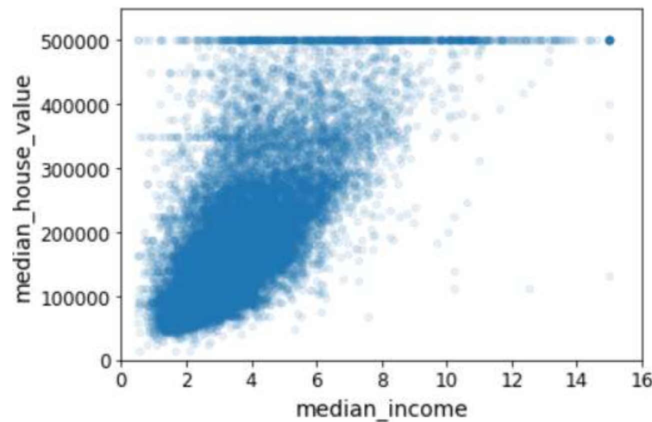
```
median_house_value     1.000000
median_income          0.687160
total_rooms            0.135097
housing_median_age     0.114110
households             0.064506
total_bedrooms         0.047689
population            -0.026920
longitude             -0.047432
latitude              -0.142724
Name: median_house_value, dtype: float64
```



24

INCOME VS. MEDIAN HOUSE PRICE

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
```



25

CREATING NEW ATTRIBUTES (FEATURES) IF NECESSARY

```
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"]
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

26

DATA PREPARATION/PREPROCESSING

- It is a good practice to write functions for data preparation
  - Easy to reproduce transformations
  - Use in a live system
  - Build library of transformations
  - Try various combinations and choose the best
- Prepare a separate copy for experimentation
- Remove target values

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

27

HANDLING MISSING VALUES

- Dataset may have some missing values
- Strategies for handling missing numerical values
  - Delete records with missing values
  - Delete the column
  - Replace missing values with a suitable value such as median or mean
- You can use Pandas na processing functions or Scikit Imputer to achieve required result

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662	<1H OCEAN

28

HANDLING MISSING VALUES

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1
```

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity

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

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662	<1H OCEAN

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

```
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762	<1H OCEAN

29

HANDLING TEXT AND CATEGORICAL ATTRIBUTES

- ocean_proximity is a categorical attribute as it consists of few values
- It is also a text attribute as categories represented using text
- Most ML algorithms prefer to work with numbers
- Hence text attribute need to be converted to numbers
- Strategies to convert to numerical values
  - Convert as ordinal values where numbers representing order (ordinal encoding)
  - Convert to binary attributes (one-hot encoding)

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

30

ORDINAL ENCODING

- Many ML algorithms assume that two nearby values are more similar than distant values
- This may be true for ordered categories such as “bad”, “average”, “good”, “excellent”
- However not suitable for ocean_proximity attribute

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

```
ordinal_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

31

ONE-HOT ENCODING

- Create one binary attribute per category
- E.g. 1 when category “INLAND”, 0 otherwise
- One attribute will be equal to 1 (hot), while others will be 0 (cold)

```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

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

```
cat_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

sparse

	7					6
	7	6	3		4	
	4	3				
4	2					
				3	2	4

DENSE

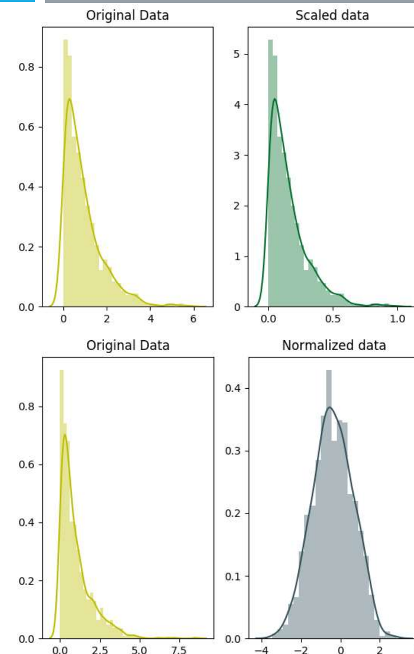
0	7	0	0	0	0	6
0	7	6	3	0	4	0
0	4	3	0	0	0	0
4	2	0	0	0	0	0
0	0	0	0	3	2	4

© Matt Eding

32

FEATURE SCALING

- ML algorithms don't perform well on the numerical data with very different scales
- E.g. total number of rooms range from 6 – 39320, median income ranges from 0 – 15.
- It is required to scale input values, but scaling target values is generally not required
- Approaches for scaling attributes:
  - Min-max scaling: Values are shifted and rescaled so that they end up ranging from 0 to 1 (subtract the min value and divide by the max minus the min)
  - Scikit-Learn has MinMaxScaler transformer $x' = \frac{x - x_{mean}}{x_{max} - x_{min}}$
  - Standardization: Subtracts the mean value (so standardized values always have a zero mean), and then divide by the standard deviation so that the resulting distribution has unit variance
  - Standardization does not bound values to a specific range, but less affected by outliers
  - Scikit-Learn has StandardScaler transformer $x' = \frac{x - x_{mean}}{\sigma}$



Figures taken from:
https://kharshit.github.io/blog/2018/03/23/scaling-vs-normalization

33

TRANSFORMATION PIPELINE USING SCIKIT-LEARN

- Preprocessing numerical attributes

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

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

- Full pipeline for preprocessing numerical and categorical attributes

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

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

housing_prepared = full_pipeline.fit_transform(housing)
```

34

EXPERIMENT READY DATA

```
housing_prepared.shape
```

```
(16512, 16)
```

```
housing_prepared[0]
```

```
array([-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
       -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
        0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
        0.        ])
```

35

TRAINING AND EVALUATING A MODEL – LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
# let's try the full preprocessing pipeline on a few training instances
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

Compare against the actual values:

```
print("Labels:", list(some_labels))
```

```
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

36

FIND OUT TOTAL PREDICTION ERRORS – MSE AND MAE

```python
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
```

```
68628.19819848922
```

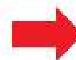
```python
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

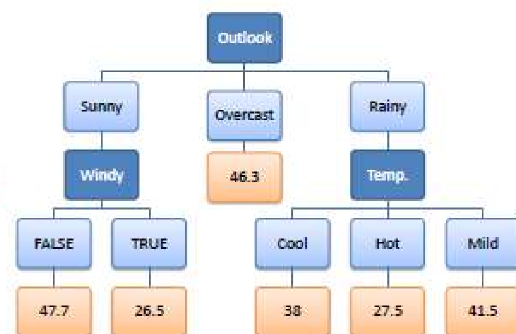
```
49439.89599001897
```

37

APPLY ANOTHER MODEL – DECISION TREE REGRESSOR

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30





38

APPLY ANOTHER MODEL – DECISION TREE REGRESSOR

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=42, splitter='best')
```

```
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
```

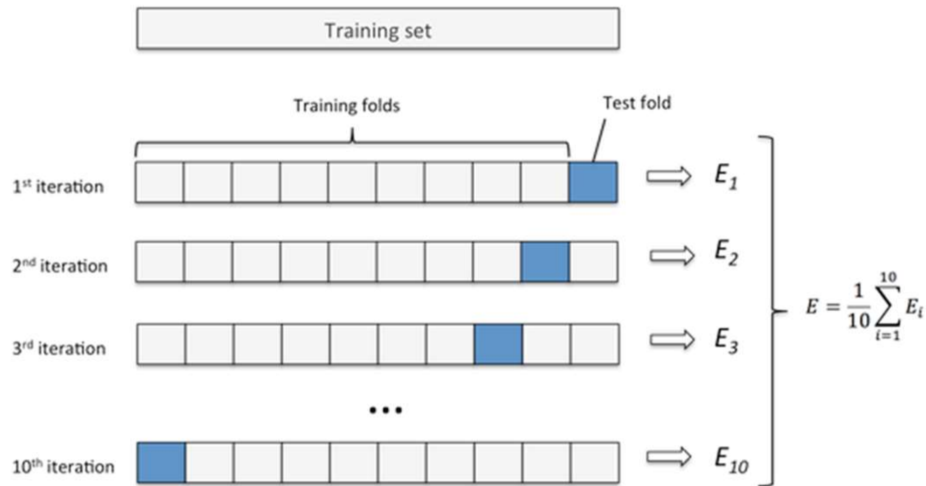
39

DID WE GET A PERFECT MODEL?

- No error at all?
- Could this model really be absolutely perfect?
- Is this a possible case of ....?
- How can you be sure?

40

BETTER EVALUATION USING CROSS VALIDATION



41

BETTER EVALUATION USING CROSS VALIDATION (DECISION TREE REGRESSOR)

```
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)
tree_rmse_scores = np.sqrt(-scores)
```

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

display_scores(tree_rmse_scores)
```

```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

42

BETTER EVALUATION USING CROSS VALIDATION (LINEAR REGRESSOR)

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

43

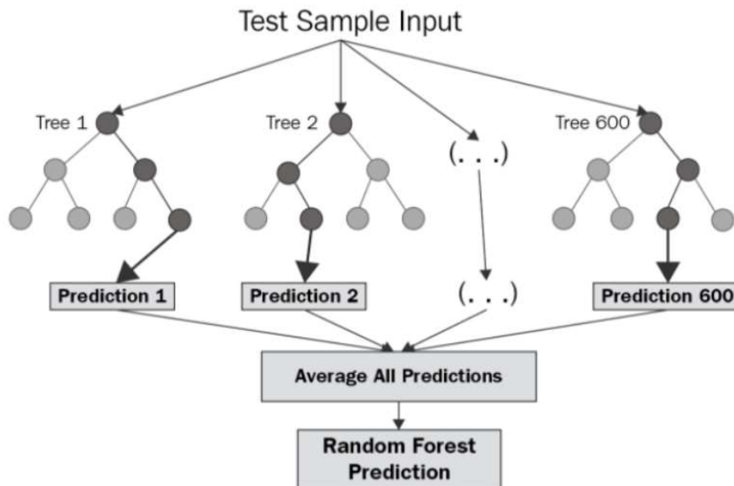
CROSS VALIDATION PERFORMANCE COMPARISON

	Decision Tree Regressor	Linear Regression
RMSE	71407.68	69052.46
Standard Deviation	2439.43	2731.67

- Decision tree performs worse than linear regression
- Cross validation provides performance estimate along with its precision (standard deviation)
- Cross validation comes at the cost of training the model several times
- Training model several times may not be always possible

44

TRY ANOTHER MODEL – RANDOM FOREST REGRESSOR



- Random Forests work by training many Decision Trees on random subsets of the features, then averaging out their predictions.
- Building a model on top of many other models is called *Ensemble Learning*
- It is often a great way to push ML algorithms even further

Figure source: https://levelup.gitconnected.com/random-forest-regression-209c0f354c84

45

TRY ANOTHER MODEL – RANDOM FOREST REGRESSOR

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
18603.515021376355
```

- The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default)
- Otherwise the whole dataset is used to build each tree

46

CROSS VALIDATION USING RANDOM FOREST REGRESSOR

```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

	Decision Tree Regressor	Linear Regression	Random Forest Regressor
RMSE	71407.68	69052.46	50182.30
Standard Deviation	2439.43	2731.67	2097.08

47

FINETUNING THE MODEL

- It is possible to fiddle with the hyperparameters manually a great combination is found (tedious)
- Scikit-Learn's GridSearchCV can be used to search optimal parameters
- Other options: RandomSearchCV, ensemble methods

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

48

WHAT ARE THE BEST HYPERPARAMETERS?

```
grid_search.best_params_
```

```
{'max_features': 8, 'n_estimators': 30}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=30, n_jobs=None, oob_score=False, random_state=42,
           verbose=0, warm_start=False)
```

49

EVALUATE SYSTEM ON THE TEST 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)
```

```
final_rmse
```

```
47730.22690385927
```

50

HOW PRECISE YOUR ESTIMATE IS? – COMPUTE 95% CONFIDENCE INTERVAL

```
from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([45685.10470776, 49691.25001878])
```

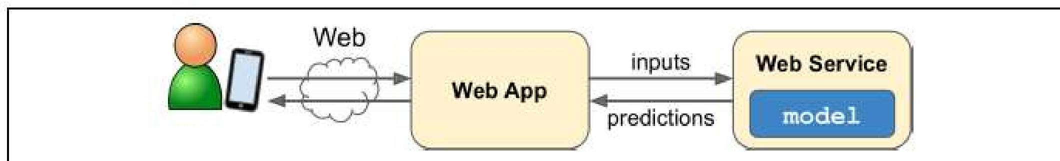
```
#Manually calulating confidence interval
m = len(squared_errors)
mean = squared_errors.mean()
tscore = stats.t.ppf((1 + confidence) / 2, df=m - 1)
tmargin = tscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - tmargin), np.sqrt(mean + tmargin)
```

```
(45685.10470776014, 49691.25001877871)
```

51

LAUNCH, MONITOR AND MAINTAIN YOUR SYSTEM

- Save the trained Scikit-Learn model (e.g., using joblib), include the full preprocessing and prediction pipeline
- Load this trained model within your production environment
- Use the model to make predictions by calling its predict() method
- Alternatively, model can be wrapped as a dedicated web service which can be queried using REST API



- It is also required to check model performance from time to time
- If data keeps evolving, it will be required to update database and retrain model regularly
- These tasks can be automated using custom scripts

52

QUESTIONS?

53