End-to-end ML Project

Lecture 3

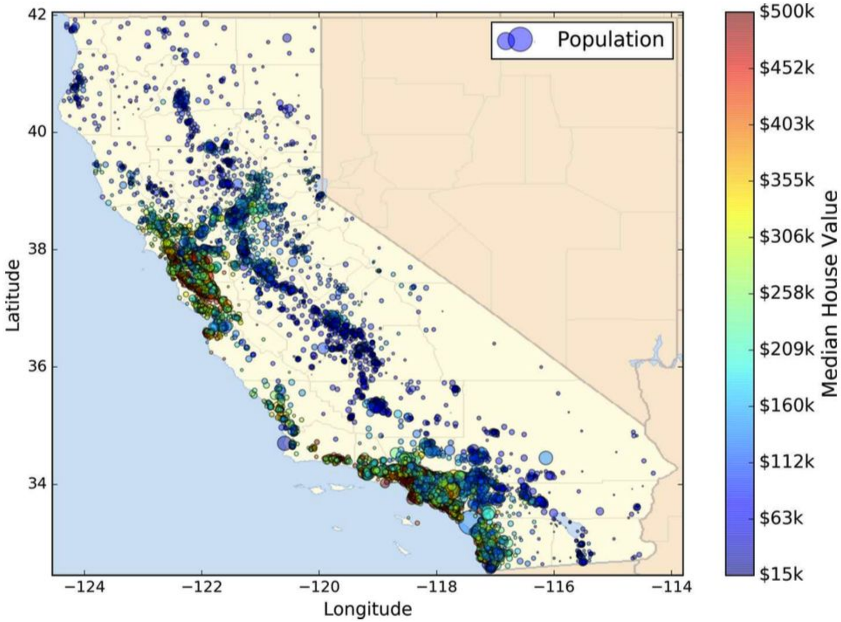
You’re hired as a data scientist for a real estate investment company

To analyze the housing market in California

Working with real data

[The California Housing Dataset]

- Based on US Census Bureau
- Organized by **district** (600-3000 people)



Other popular data repositories:

- UC Irvine Machine Learning Repository
- Amazon AWS Datasets
- Wikipedia's list of Machine Learning datasets
- Kaggle

8 Main Steps of a ML Project

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.

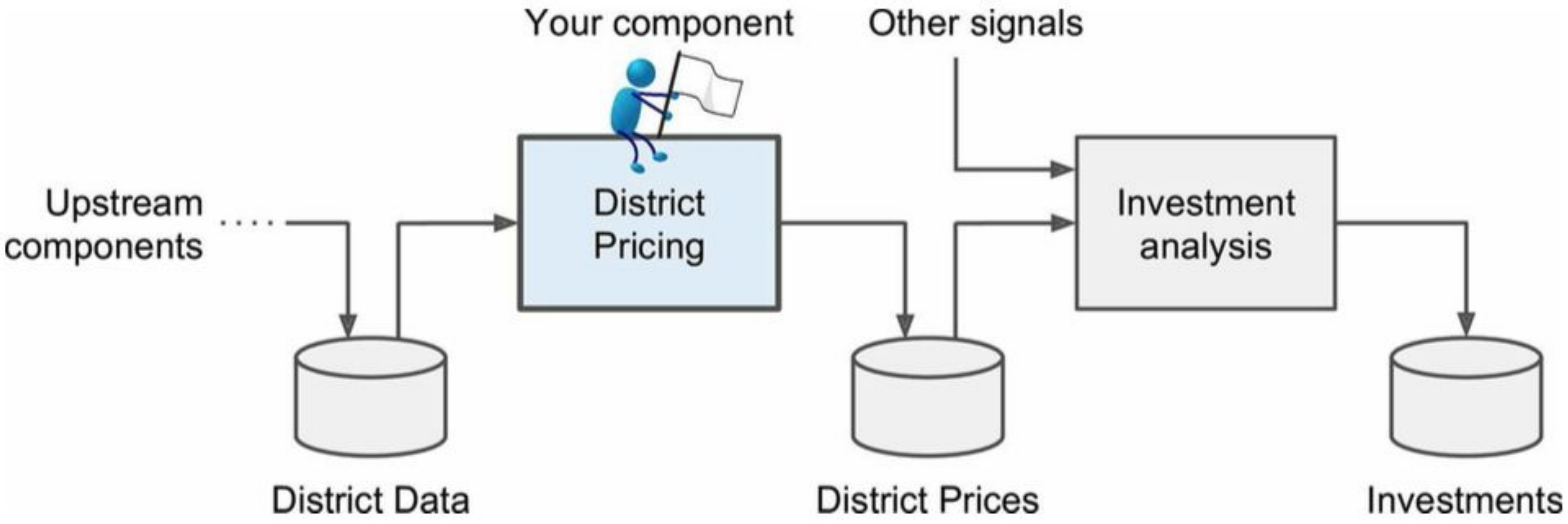
1. Looking at the big picture

The big picture

Ask the purpose of building a model --> **frame** your problems and objectives:

- What to expect, use, and benefit from this model?
- What learning algorithm to use?
- What performance measure to evaluate?
- How much effort to be spent?

ML Pipeline for real-estate investments



Select a performance measure MAE

- One option is **Mean Absolute Error** (MAE)
- Measure the distance between prediction and target values
- Correspond to **L1 norm**, or Manhattan norm (easily understood).

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

All training data

Hypothesis/Model

# of instances

Predicted value of i^{th} instance

feature vector of the i^{th} instance

Value of the i^{th} instance

Select a performance measure RMSE

- A preferred for regression problem is **Root Mean Square Error** (RMSE)
- Correspond to **L2 norm**, or Euclidean norm
- More sensitive to outliers than MAE, but generally perform better (Why?)

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

# of instances

All training data

Value of the i^{th} instance

Hypothesis/Model

Predicted value of i^{th} instance

feature vector of the i^{th} instance

2. Getting the Data

Load the data from a .csv file with read_csv()

```python
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)
```

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

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
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

Take a quick look with info()

```
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
```

Learn some basic statistics with describe()

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrc
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.0000
mean	-119.569704	35.631861	28.639486	2635.763081	537.87055
std	2.003532	2.135952	12.585558	2181.615252	421.38507
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.00000
50%	-118.490000	34.260000	29.000000	2127.000000	435.00000
75%	-118.010000	37.710000	37.000000	3148.000000	647.00000
max	-114.310000	41.950000	52.000000	39320.000000	6445.0000

Create train and test sets with train_test_split()

```
from sklearn.model_selection import train_test_split

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

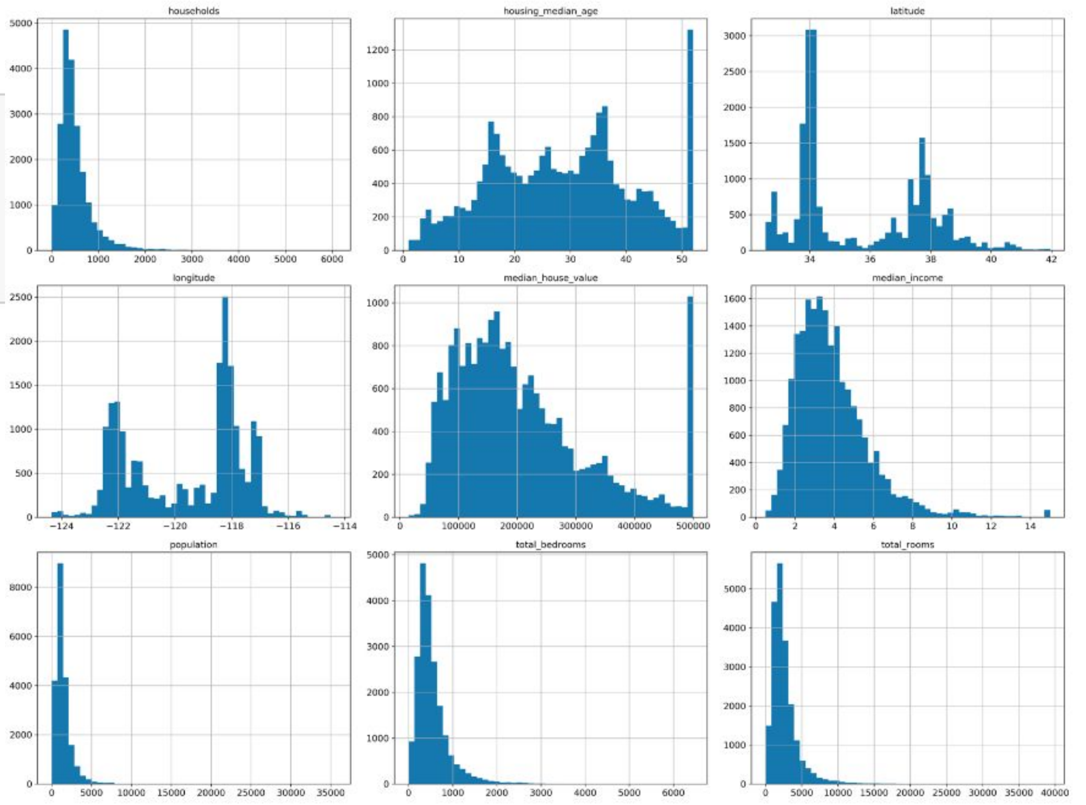
```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	

3. Exploring and Visualizing the Data

Plot a histogram with hist()

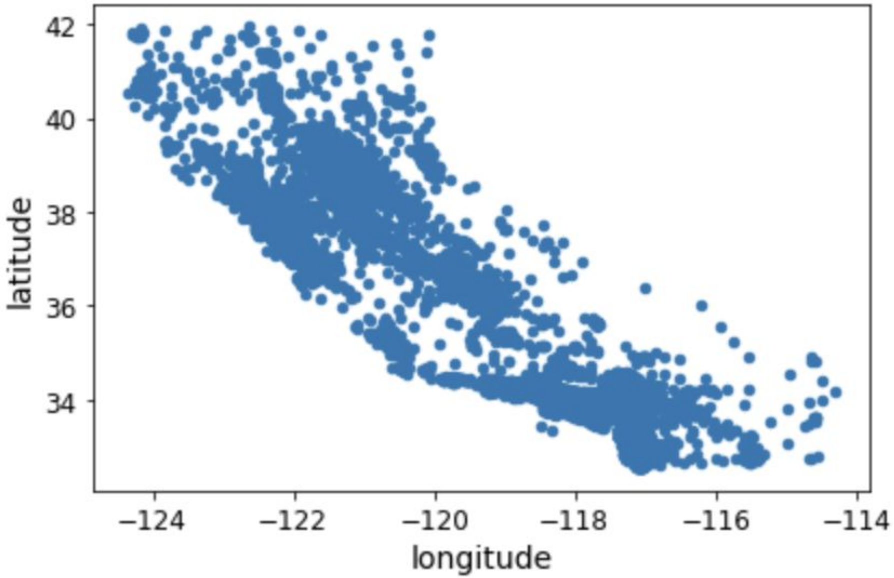
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```



Discover and Visualize the Data with plot()

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

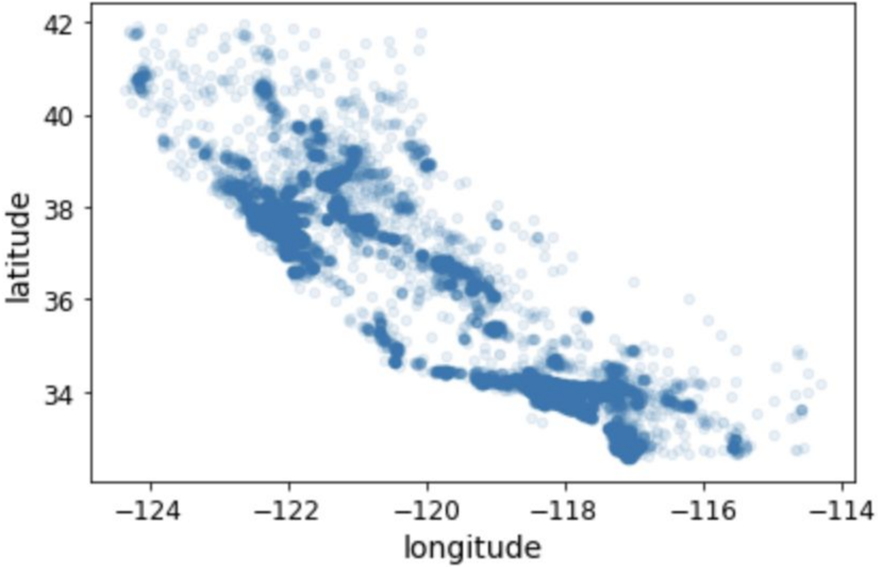
```
Saving figure bad_visualization_plot
```



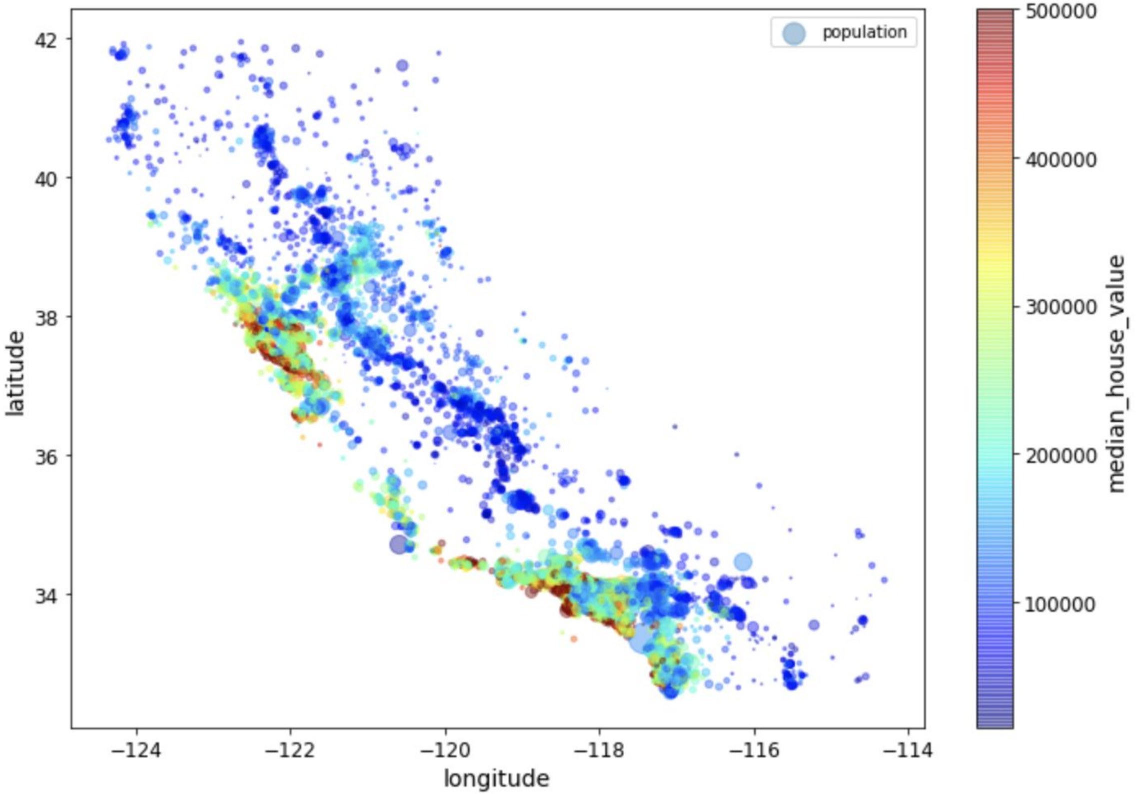
Discover and Visualize the Data with plot()

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

```
Saving figure better_visualization_plot
```

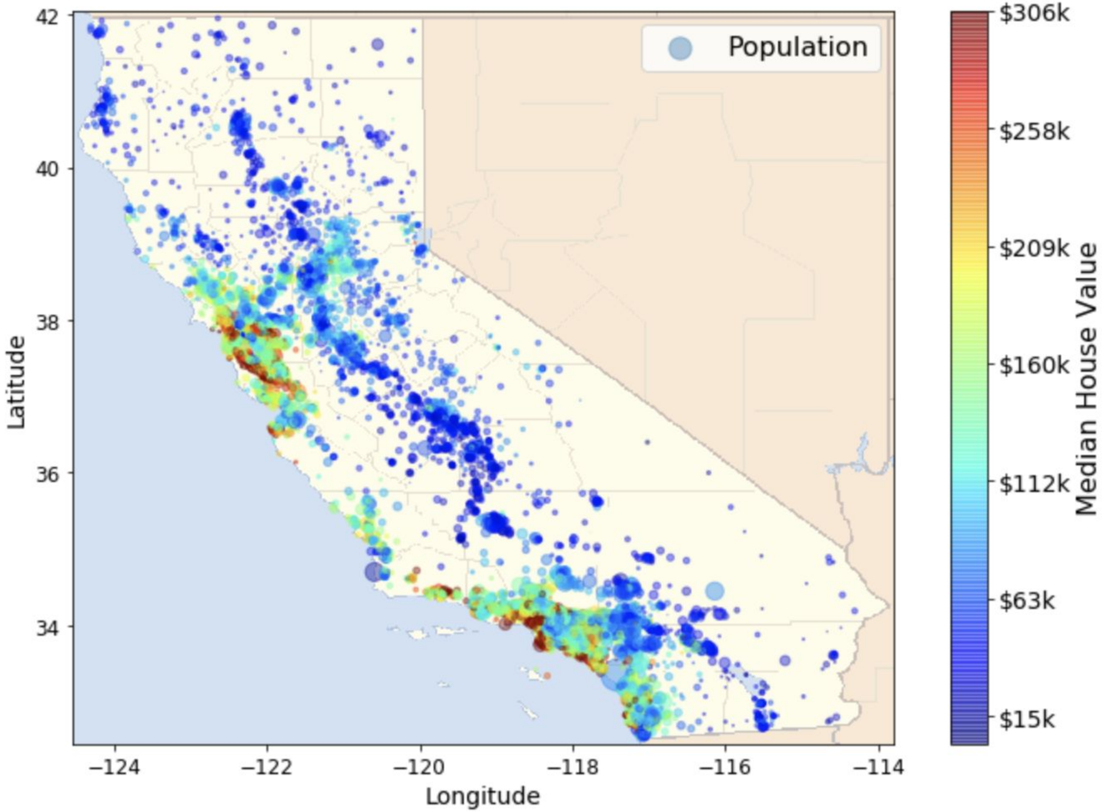


Add more visualization dimension



Colors
Sizes

Add a Geographical Map Image… Nice!



Did you notice some “hot spots”?

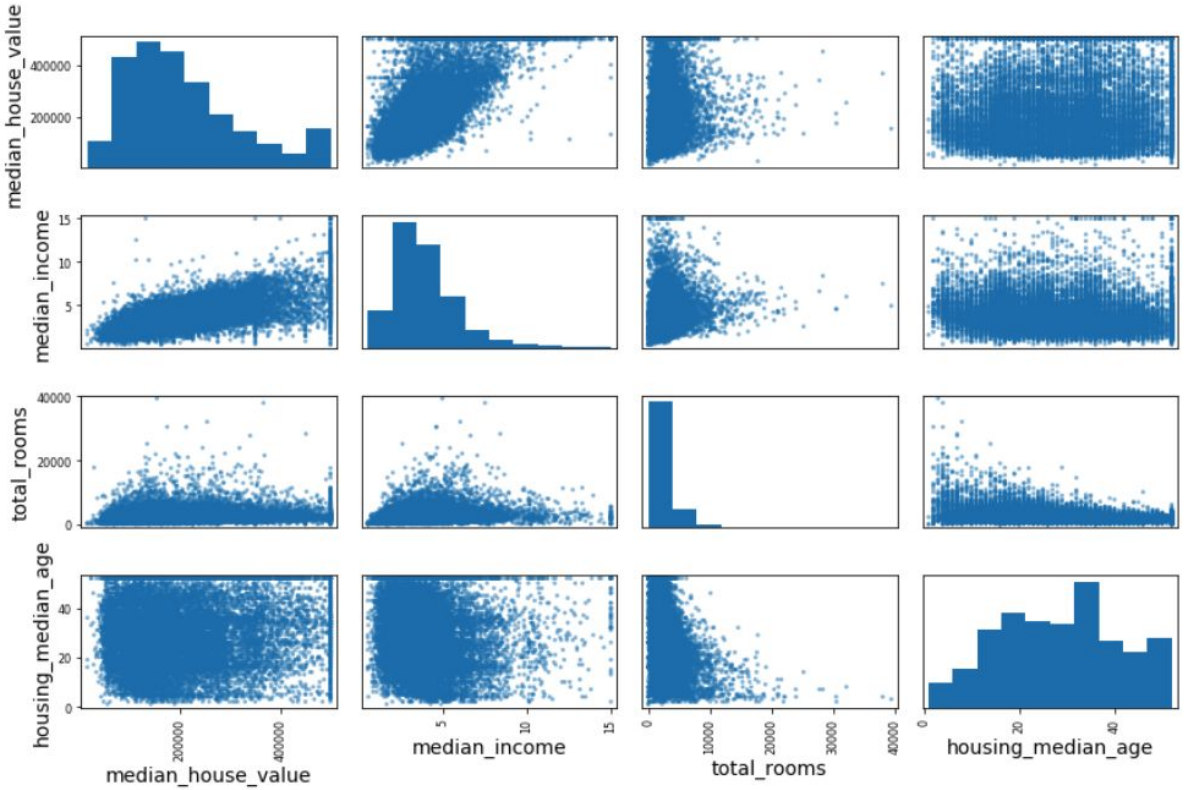
Look at correlations with corr()

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

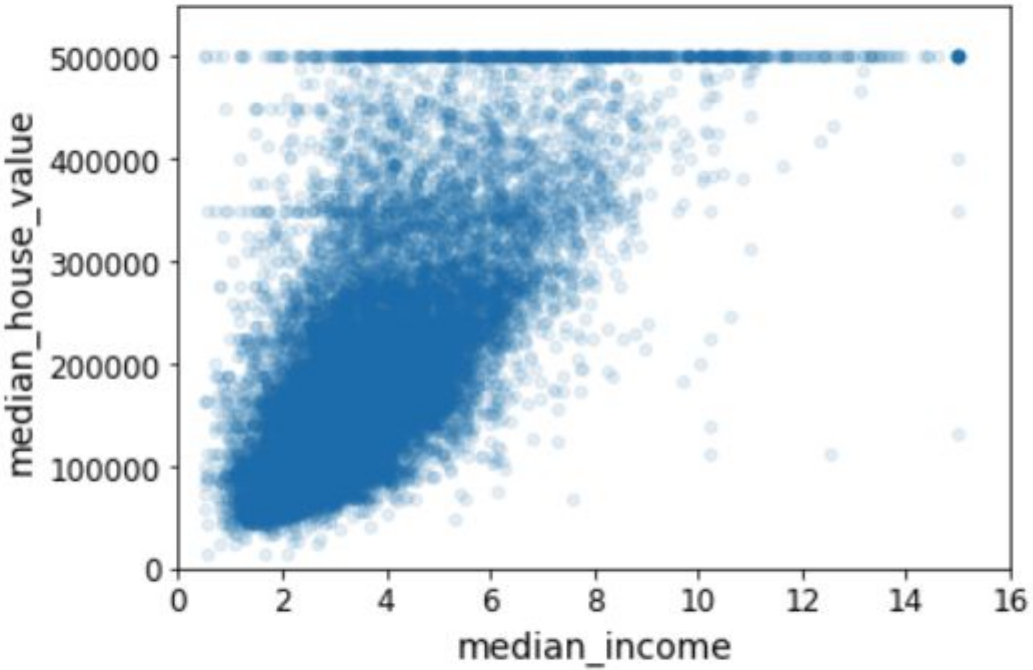
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
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
```

Check all correlations with scatter_matrix()



Isolate the most “interesting” one



Why is this plot “**interesting**”?

Experiment with Feature Extraction

```
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
```

4. Preparing the data
(aka. Data Cleaning!)

Detect “missing values”

```
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_p
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708	<1H
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762	<1H
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328	<1H
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675	
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662	<1H

Fill in “missing values” with Imputer

```
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy="median")
```

```
housing_num = housing.drop('ocean_proximity', axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
```

```
imputer.fit(housing_num)
```

```
Imputer(axis=0, copy=True, missing_values='NaN', strategy='median', verbo
se=0)
```

Transform the training set

```
X = imputer.transform(housing_num)
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index = list(housing.index.values))
```

```
housing_tr.loc[sample_incomplete_rows.index.values]
```

latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708
34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762
37.35	30.0	1955.0	433.0	999.0	386.0	4.6328
34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675
38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662

Process “categorical inputs”

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

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

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

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND

Encode the categories with OneHotEncoder

```
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)]
```

Use a Pipeline for a sequence of transformations

```
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)
```

```
housing_num_tr
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
```

Combine columns with ColumnTransformer

```
from future_encoders 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)
```

```
housing_prepared
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,
         0.        ,  1.        ],
       ...,
```

5. Selecting a Model to train

Select a model

```
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=1, normalize=False)
```

Evaluate on some test data

```
# let's try the full 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]
```

Calculate the Errors

```
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
```

```
68628.198198489219
```

```
from sklearn.metrics import mean_absolute_error

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

```
49439.895990018973
```

Try another model: DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
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
```

← **Zero** error!? What’s happened?

Better Evaluation using cross_val_score()

```
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)
```

Display the scores

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

display_scores(tree_rmse_scores)
```

```
Scores: [70232.0136482  66828.46839892 72444.08721003 70761.50186201
 71125.52697653 75581.29319857 70169.59286164 70055.37863456
 75370.49116773 71222.39081244]
Mean: 71379.07447706361
Standard deviation: 2458.3188204349362
```

← Off by $71,379

Let’s do the same with the Linear Regression

```
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   70361.18285107 74742.02420674
 68022.09224176 71193.07033936 64969.63056405 68276.69992785
 71543.69797334 67665.10082067]
Mean: 69051.63554357362
Standard deviation: 2732.3913087537303
```

← Off by $69,051

Try one more model: RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor

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

```
from sklearn.model_selection import cross_val_score

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

```
Scores: [51650.94405471 48920.80645498 52979.16096752 54412.74042021
 50861.29381163 56488.55699727 51866.90120786 49752.24599537
 55399.50713191 53309.74548294]
Mean: 52564.19025244012
Standard deviation: 2301.873803919754
```

Our best model so far is off by $52,564!
Can we do better?

6. Fine-tuning your model

Fine tune your model

- Grid Search
- Randomize Search
- Ensemble Methods

Do a Grid Search

```python
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)
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)
```

How many times does this search have to call training on RandomForestRegressor?

What are the cases?

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63647.854446 {'max_features': 2, 'n_estimators': 3}
55611.5015988 {'max_features': 2, 'n_estimators': 10}
53370.0640736 {'max_features': 2, 'n_estimators': 30}
60959.1388585 {'max_features': 4, 'n_estimators': 3}
52740.5841667 {'max_features': 4, 'n_estimators': 10}
50374.1421461 {'max_features': 4, 'n_estimators': 30}
58661.2866462 {'max_features': 6, 'n_estimators': 3}
52009.9739798 {'max_features': 6, 'n_estimators': 10}
50154.1177737 {'max_features': 6, 'n_estimators': 30}
57865.3616801 {'max_features': 8, 'n_estimators': 3}
51730.0755087 {'max_features': 8, 'n_estimators': 10}
49694.8514333 {'max_features': 8, 'n_estimators': 30}
62874.4073931 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54643.4998083 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59437.8922859 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52735.3582936 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.0168279 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51008.2615672 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Identify 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=1, oob_score=False, random_state=42,
           verbose=0, warm_start=False)
```

7. Presenting your solution

Evaluate 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
```

```
47766.003966433083
```

Our error is getting smaller after each step:
$71K → $69K → $52K → $49K → $47K

As a data scientist at an investment firm, you **analyzed the housing market in California**

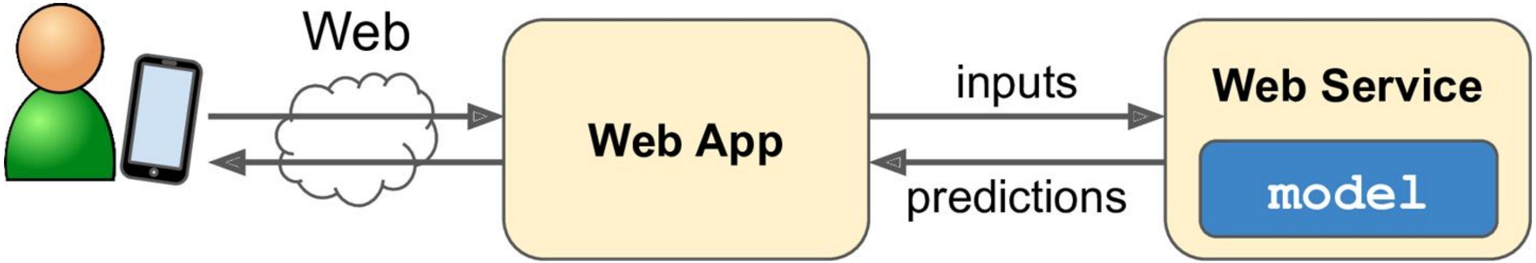
- Create a nice presentation with clear visual aids and easy to remember statements
- Present your solution to your stakeholders highlighting what you've learned:
  - What worked and what did not
  - What assumptions were made
  - What the system's limitations are

8. Launching and Maintaining

Launch model on the cloud

Google Cloud AI Platform: just save your model using joblib and upload it to Google Cloud Storage (GCS), then head over to Google Cloud AI Platform and create a new model version, pointing it to the GCS file. This gives you a simple web service that takes care of load balancing and scaling for you.

It take JSON requests containing the input data (e.g., of a district) and returns JSON responses containing the predictions. You can then use this web service in your website.



Monitor and Maintain

You’ve got the approval to launch! Now what?

- Update the model over time with new training data
- Evaluate based on the field experts (or a crowd-sourcing platforms)
- Ensure the quality of incoming training data
- Refresh the model at a regular interval
- Backup the previous model

Learning Outcomes

- ❑ Have a good idea of the 8 steps of a ML project
- ❑ Learn a whole host of tools for data exploration and visualization in Colab using Python
- ❑ Try out a few ML learning algorithms
- ❑ Fine-tune some hyperparameters of a learning method

By now, you know **a lot** about ML. Keep practicing!

Up next: It’s your turn to get hands-on experience with the Codeathon 1