Practical Machine Learning 101

An end-to-end machine learning project

Justin Looye – IIJ Research Laboratory

Takeaway from today's talk - 今日の話から学べること

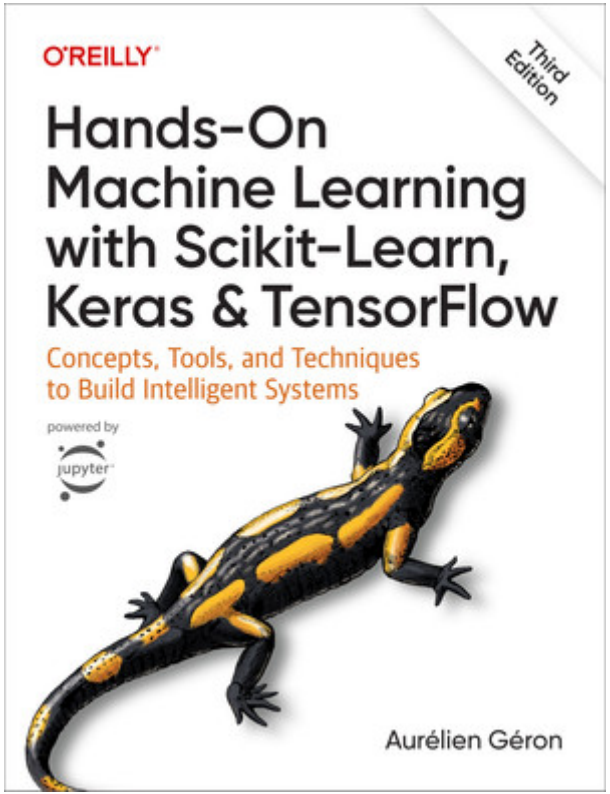
- Overview of machine learning - 機械学習の概要
- Machine learning pipeline in practice with an end-to-end project
  エンドツーエンドプロジェクトによる実践的な機械学習パイプライン

zdataset.com: A Guide to Machine Learning Pipelines and Orchest

- Popular python tools – 人気のある Python ツール
  Pandas, Scikit-Learn, MLflow, FastAPI

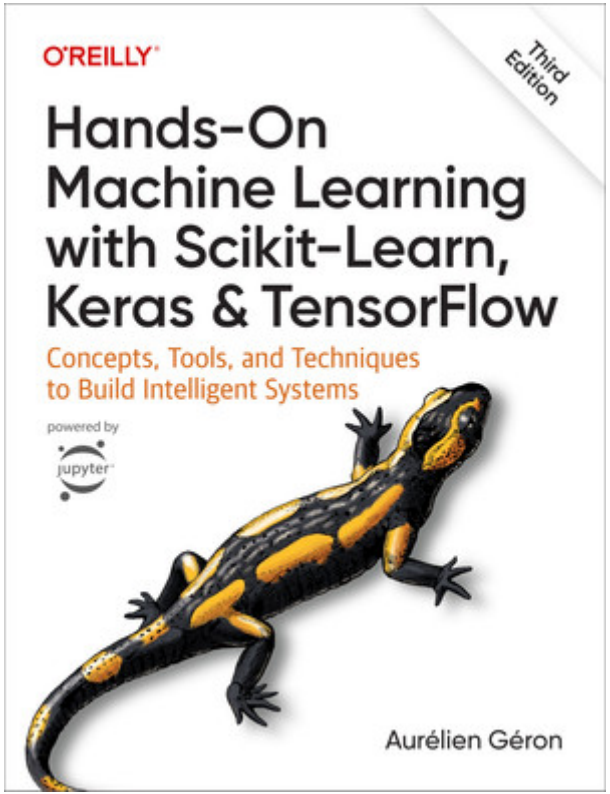
Material for today

- “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 3rd edition” by Aurélien Géron (2022)



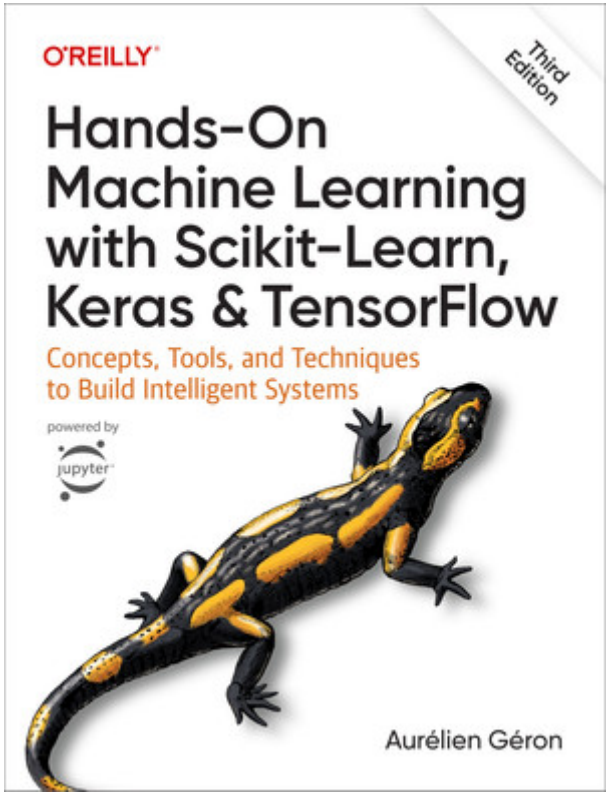
Material for today

- “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 3rd edition” by Aurélien Géron (2022)
- Well written, easy to follow with notebooks



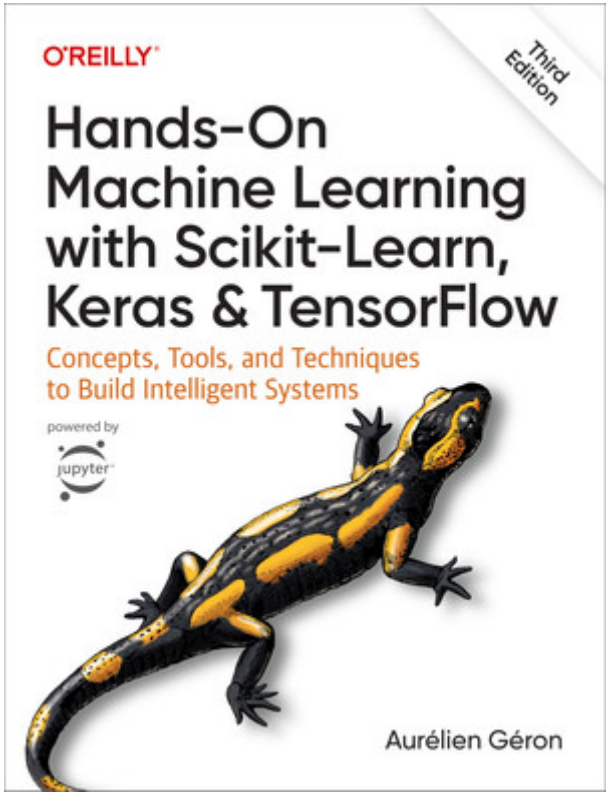
Material for today

- “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 3rd edition” by Aurélien Géron (2022)
- Well written, easy to follow with notebooks
- Low priors



Material for today

- “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 3rd edition” by Aurélien Géron (2022)
- Well written, easy to follow with notebooks
- Low priors
- Permissive apache license

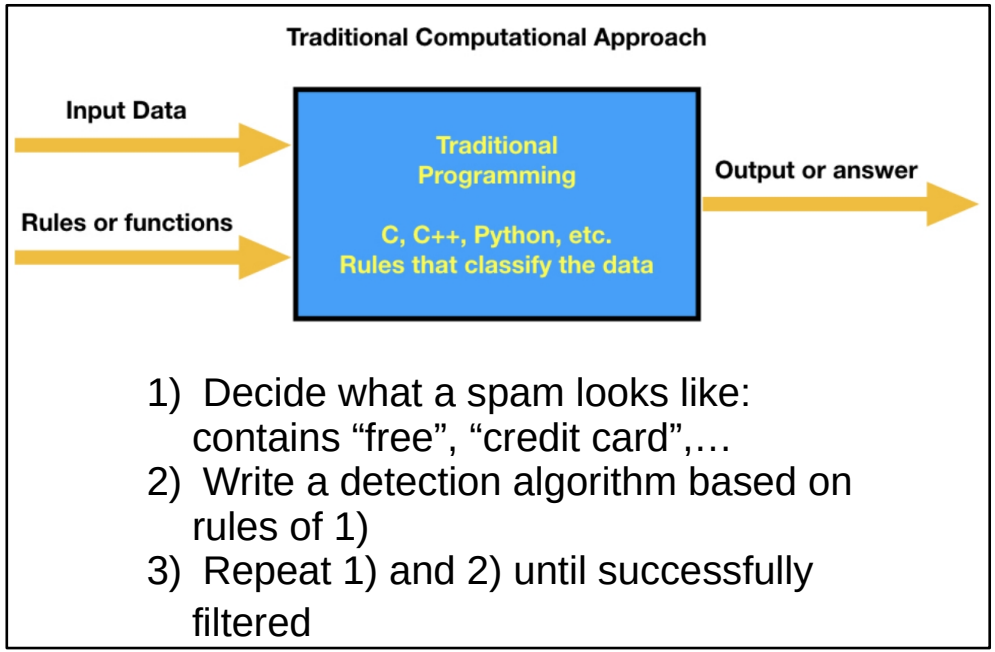


Why using Machine Learning: a paradigm shift

- Let’s consider a spam filter:

Why using Machine Learning: a paradigm shift

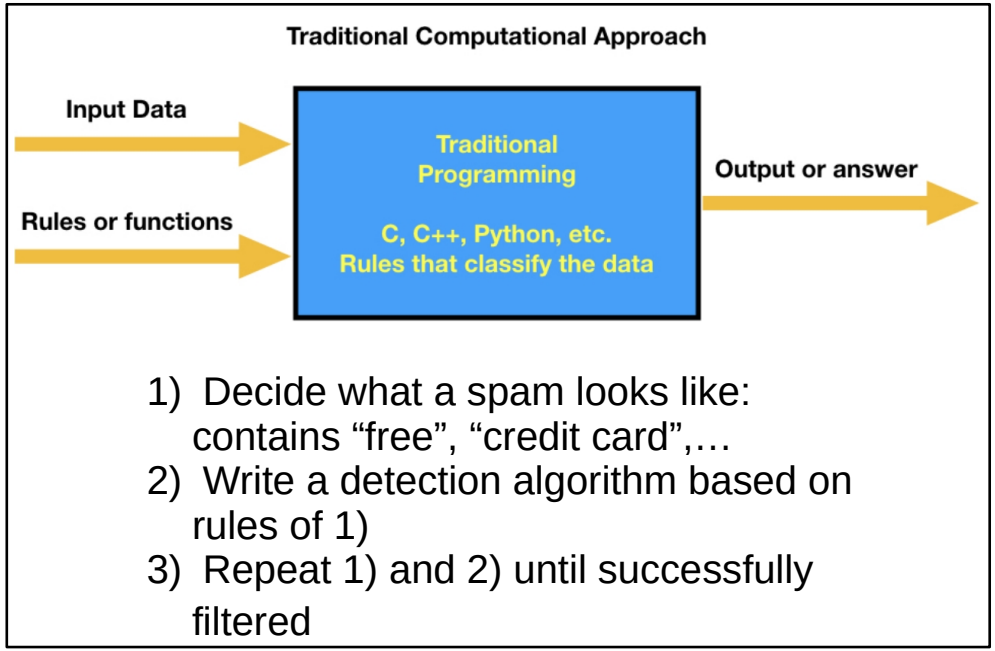
- Let’s consider a spam filter:



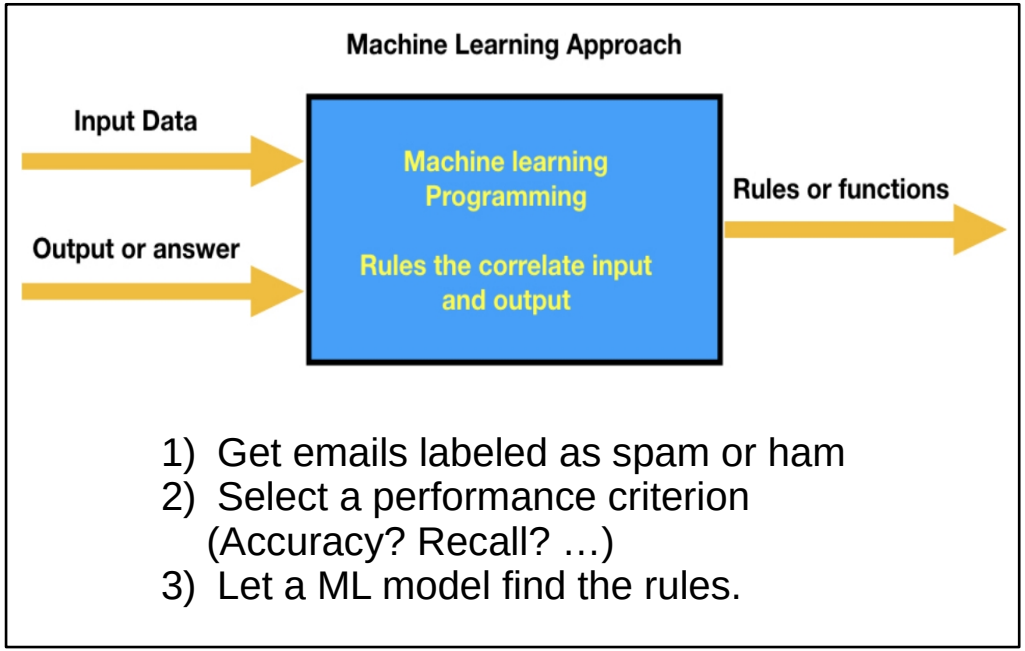
1) Decide what a spam looks like: contains “free”, “credit card”,…
2) Write a detection algorithm based on rules of 1)
3) Repeat 1) and 2) until successfully filtered

Why using Machine Learning: a paradigm shift

- Let’s consider a spam filter:



1) Decide what a spam looks like: contains “free”, “credit card”,…
2) Write a detection algorithm based on rules of 1)
3) Repeat 1) and 2) until successfully filtered

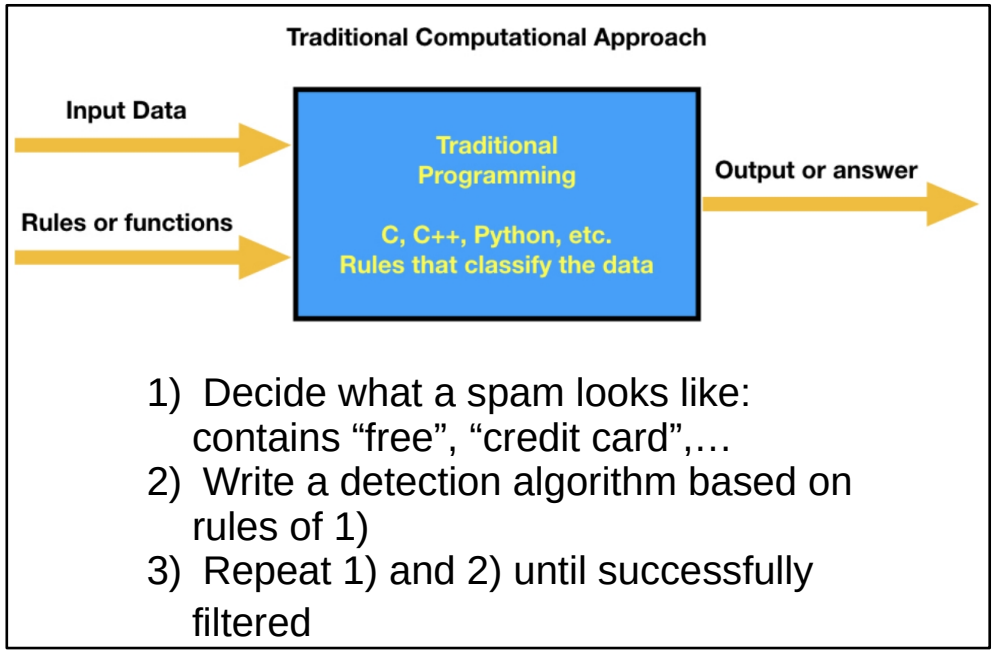


1) Get emails labeled as spam or ham
2) Select a performance criterion (Accuracy? Recall? …)
3) Let a ML model find the rules.

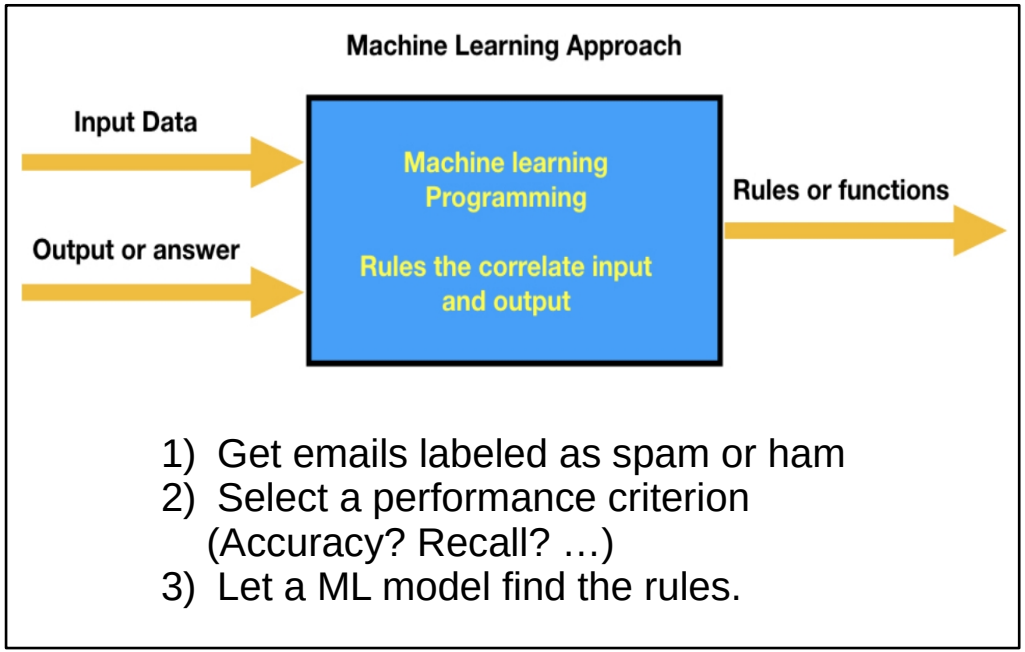
Software carpentry https://wvuhpc.github.io/2019-Machine-Learning/01-introduction/index.html

Why using Machine Learning: a paradigm shift

- Let’s consider a spam filter:



1) Decide what a spam looks like: contains “free”, “credit card”,…
2) Write a detection algorithm based on rules of 1)
3) Repeat 1) and 2) until successfully filtered



1) Get emails labeled as spam or ham
2) Select a performance criterion (Accuracy? Recall? …)
3) Let a ML model find the rules.

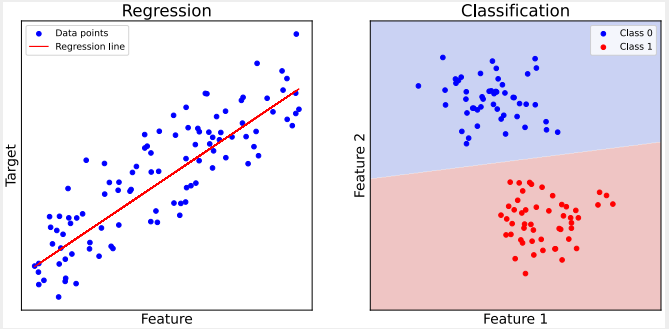
Software carpentry https://wvuhpc.github.io/2019-Machine-Learning/01-introduction/index.html

- Advantages of ML:
  - Automated decision making (easy to update)
  - No human priors (unicode abuse)

Machine learning problems in the real world

Machine learning problems in the real world

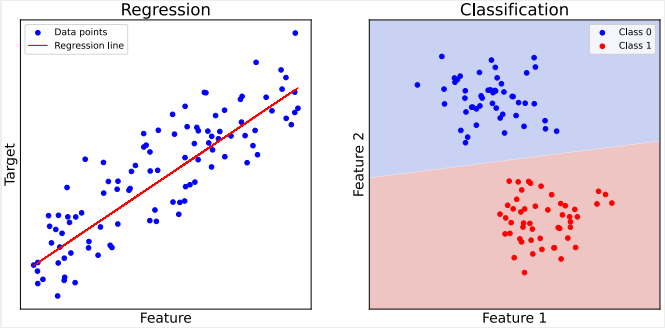
Supervised learning



- Traffic forecasting
- House price prediction
- E-mail spam detection
- Image recognition

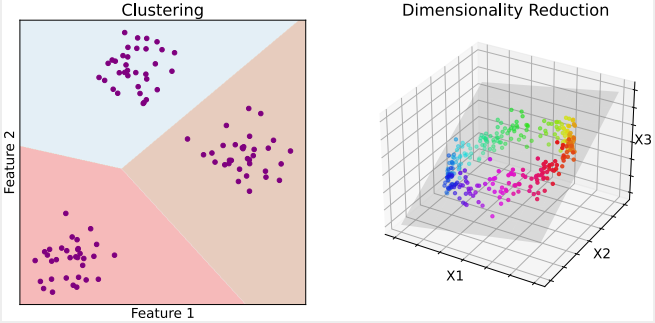
Machine learning problems in the real world

Supervised learning



- Traffic forecasting
- House price prediction
- E-mail spam detection
- Image recognition

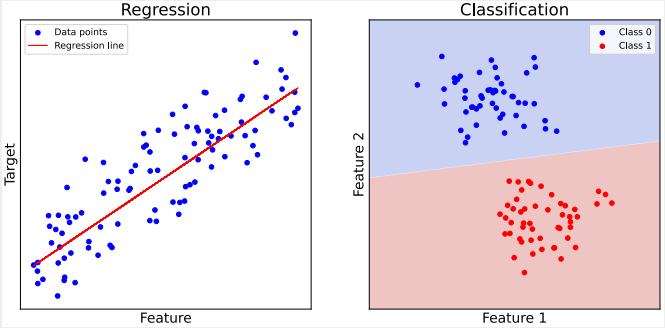
Unsupervised learning



- Credit card fraud detection
- Recommender systems
- Data visualization
- Compression

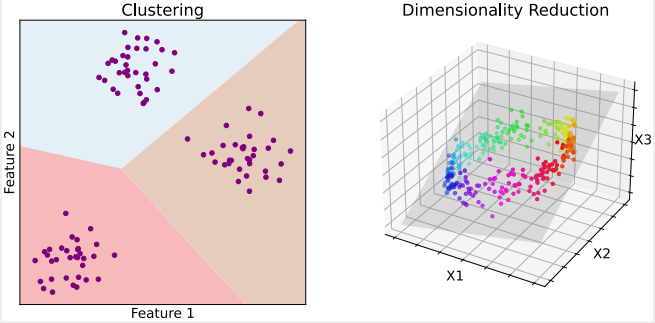
Machine learning problems in the real world

Supervised learning



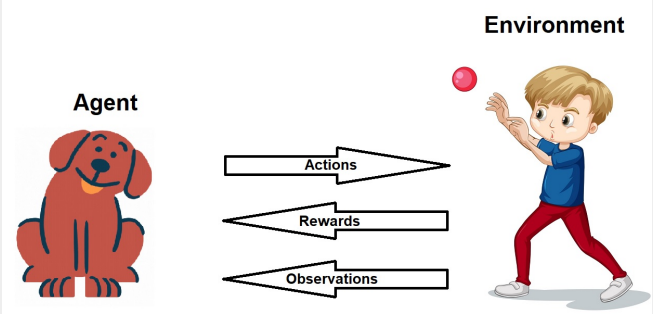
- Traffic forecasting
- House price prediction
- E-mail spam detection
- Image recognition

Unsupervised learning



- Credit card fraud detection
- Recommender systems
- Data visualization
- Compression

Reinforcement learning



Nisha Arya @ EJable.com: Three Types of Machine Learning

- Autonomous agents
- Robots
- Optimization

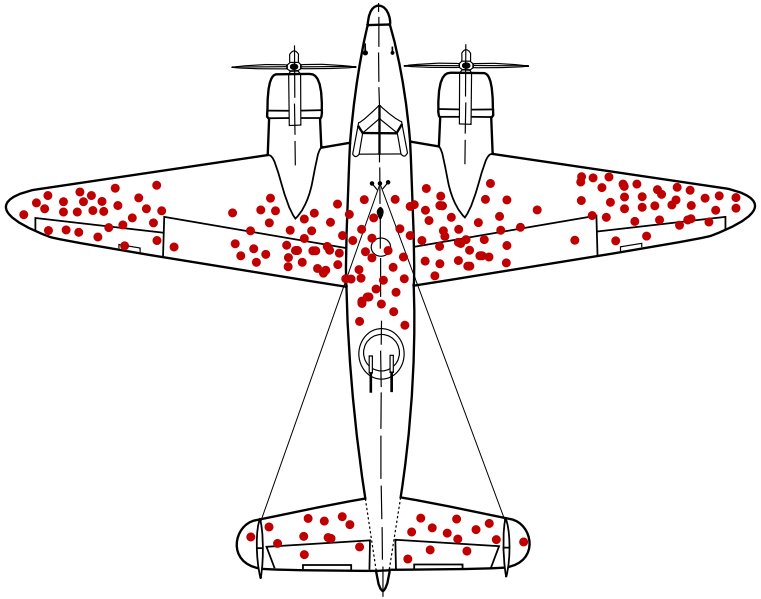
Basic challenges of ML

Basic challenges of ML

- Insufficient quantity of training data

Basic challenges of ML

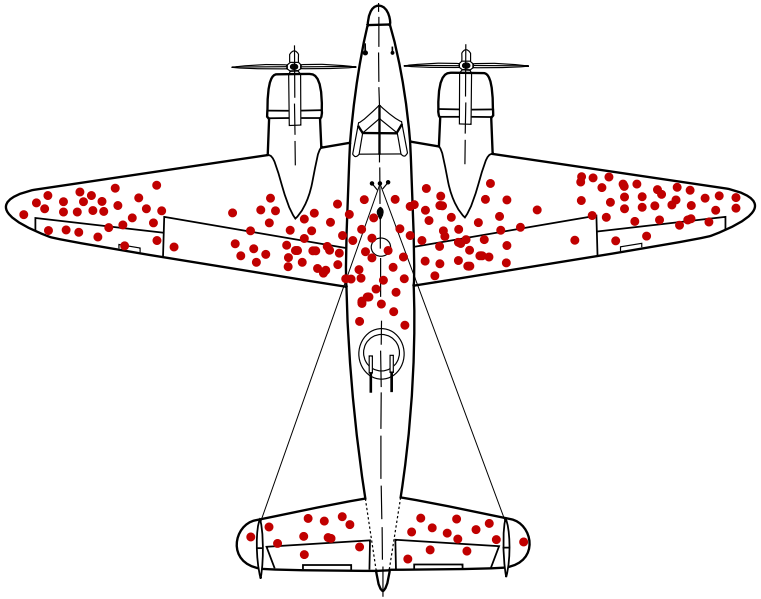
- Insufficient quantity of training data
- Nonrepresentative training data



Wikipedia: Martin Grandjean (vector), McGeddon (picture), Cameron Moll (concept)

Basic challenges of ML

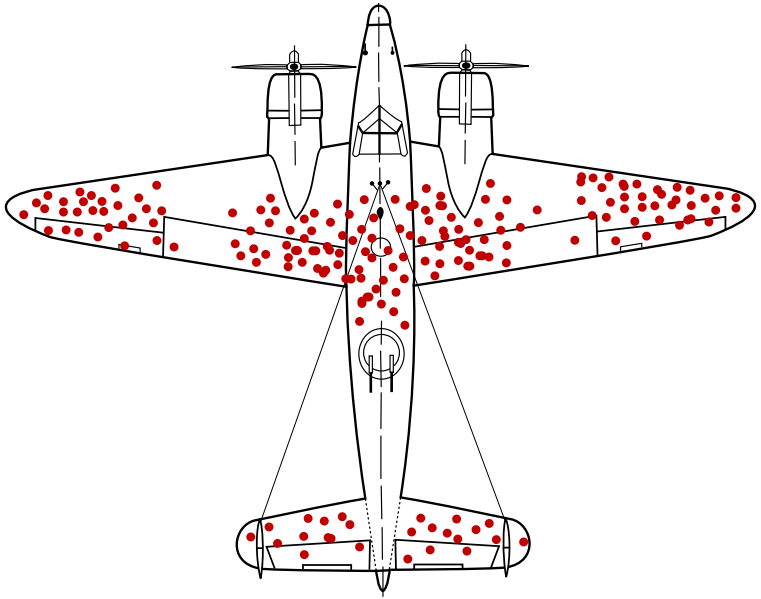
- Insufficient quantity of training data
- Nonrepresentative training data
- Poor-Quality data



Wikipedia: Martin Grandjean (vector), McGeddon (picture), Cameron Moll (concept)

Basic challenges of ML

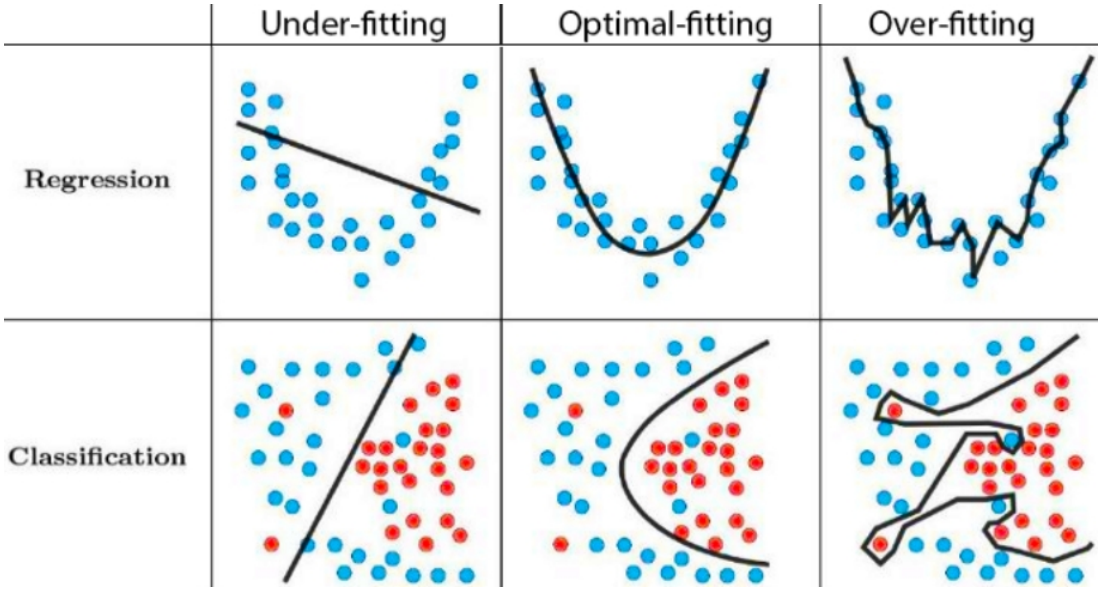
- Insufficient quantity of training data
- Nonrepresentative training data
- Poor-Quality data
- Irrelevant features



Wikipedia: Martin Grandjean (vector), McGeddon (picture), Cameron Moll (concept)

Basic challenges of ML

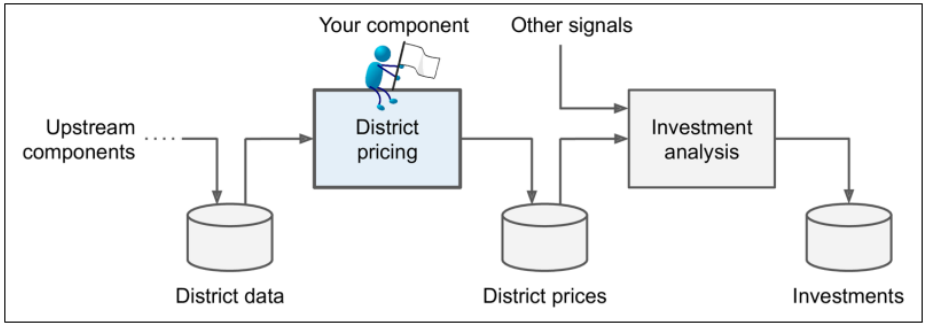
- Insufficient quantity of training data
- Nonrepresentative training data
- Poor-Quality data
- Irrelevant features
- Overfitting the training data
- Underfitting the training data



Jeeva Saravanan @ Medium: How to Evaluate your Machine Learning model

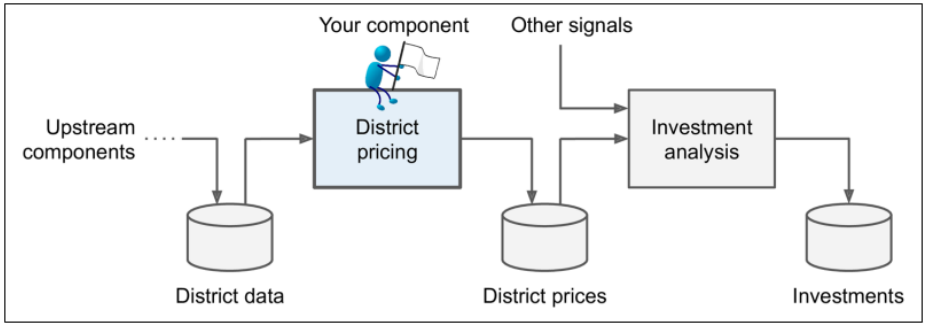
End-to-end ML project

- We are a recently hired data scientist in a real estate company
- The company want to be able to predict district median house values



End-to-end ML project

- We are a recently hired data scientist in a real estate company
- The company want to be able to predict district median house values



- Company gives us a dataset of district median house values

ID | Features (X) | Target (Y)

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

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

Frame the problem

Frame the problem

1) Supervised, unsupervised, reinforcement learning?

Frame the problem

1) Supervised, unsupervised, reinforcement learning?
2) Task? Regression, classification, clustering,...?

Frame the problem

1) Supervised, unsupervised, reinforcement learning?
2) Task? Regression, classification, clustering,...?
3) Performance criterion?

Frame the problem

1) Supervised, unsupervised, reinforcement learning?
2) Task? Regression, classification, clustering,...?
3) Performance criterion?
4) Baseline? Human-level performance? Experts?

Tool #1: Pandas for handling data

- Open-source Python library providing high-performance, easy-to-use data structures and analysis tools

Tool #1: Pandas for handling data

- Open-source Python library providing high-performance, easy-to-use data structures and analysis tools
- Key feature: DataFrame object for tabular data manipulation with integrated indexing

Tool #1: Pandas for handling data

- Open-source Python library providing high-performance, easy-to-use data structures and analysis tools
- Key feature: DataFrame object for tabular data manipulation with integrated indexing
- Core features:
  - Data manipulation
  - Data cleaning
  - File handling
  - Time series

Tool #1: Pandas for handling data

- Open-source Python library providing high-performance, easy-to-use data structures and analysis tools
- Key feature: DataFrame object for tabular data manipulation with integrated indexing
- Core features:
  - Data manipulation
  - Data cleaning
  - File handling
  - Time series
- Use case in ML:
  - Data preparation
  - Exploratory data analysis and visualization

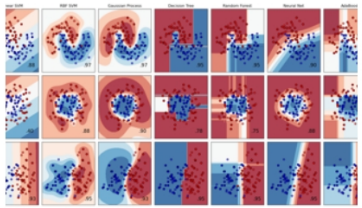
Tool #2 : scikit-learn for Machine learning

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on Numpy, SciPy and matplotlib
- Open source, commercially usable – BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

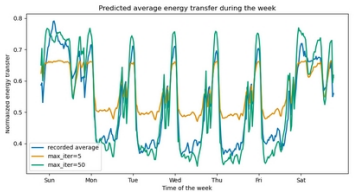


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

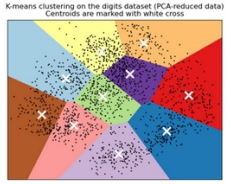


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

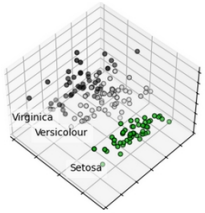


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, increased efficiency.
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

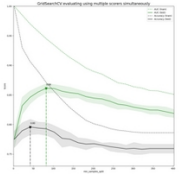


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning.
Algorithms: Grid search, cross validation, metrics, and more...

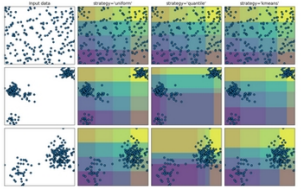


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: Preprocessing, feature extraction, and more...



Examples

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

Get the data

Get the data

- Parse the data

```
housing = pd.read_csv(csv_path)
housing.head()
```

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

Get the data

- Parse the data
- Take a quick look
  - Type
  - Missing Values

```
[9]: 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
```

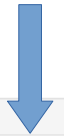
Get the data

- Parse the data
- Take a quick look
  - Type
  - Missing Values
  - Basic stats

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

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

Get the data

- Parse the data
- Take a quick look
  - Type
  - Missing Values
  - Basic stats



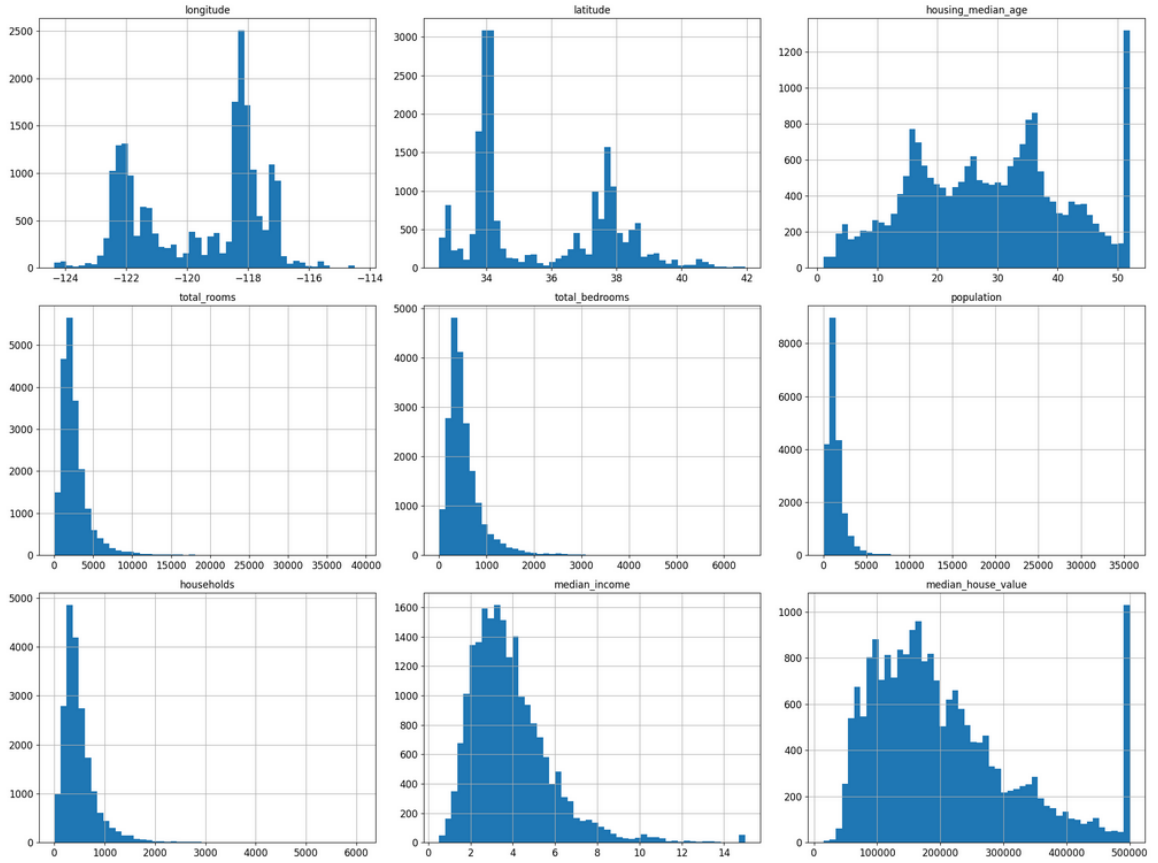
```
housing.describe()
```

	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

Get the data

- Parse the data
- Take a quick look
  - Type
  - Missing Values
  - Basic stats

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



Get the data

- Parse the data
- Take a quick look
  - Type
  - Missing Values
  - Basic stats
- Train-test split

```python
from sklearn.model_selection import train_test_split

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

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

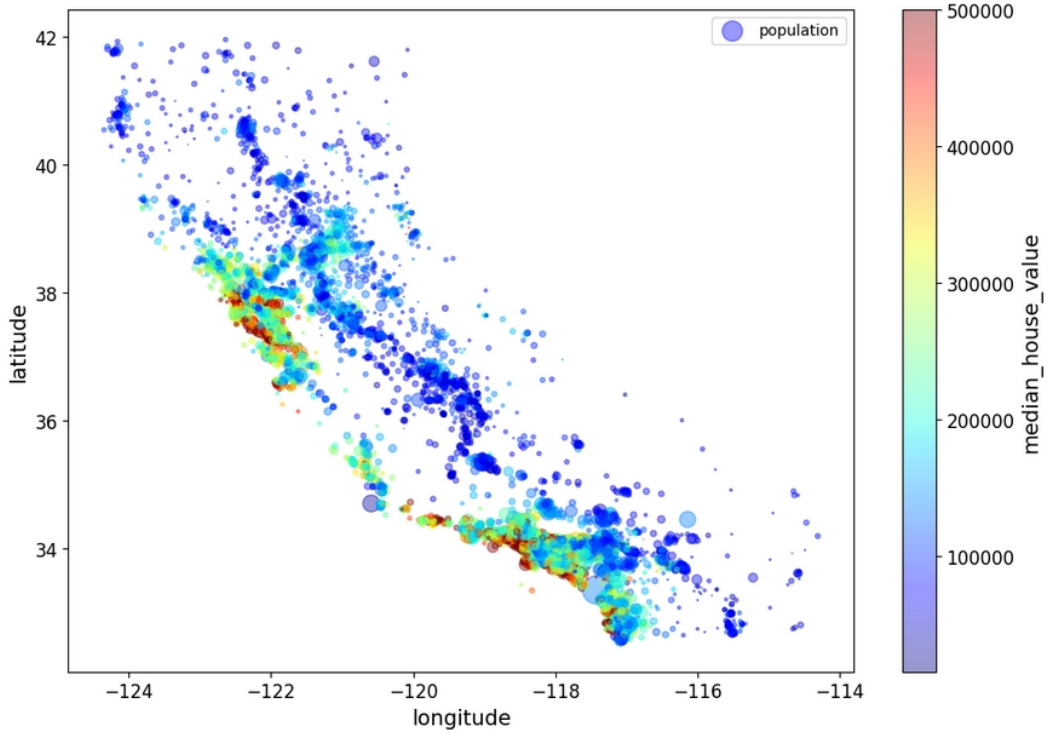
Discover and visualize data to gain insights

Discover and visualize data to gain insights

- Visualize geographical data

```
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)
plt.legend()
save_fig("housing_prices_scatterplot")
```

```
Saving figure housing_prices_scatterplot
```



Discover and visualize data to gain insights

- Visualize geographical data
- Look for correlation

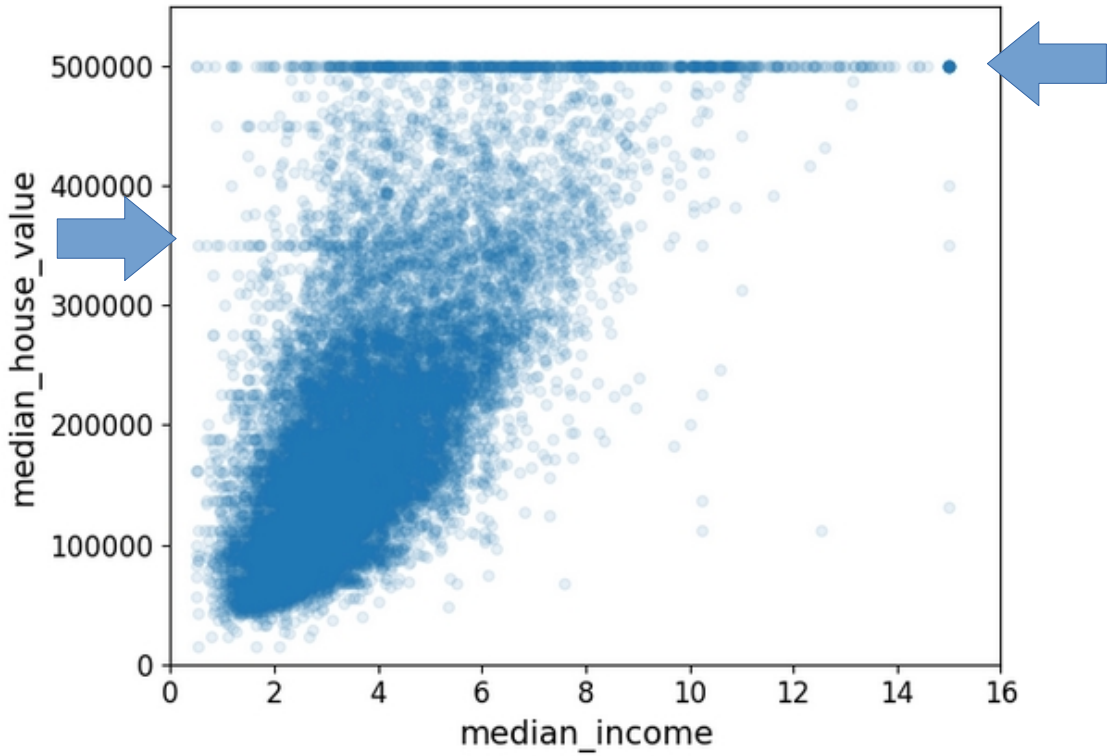
```
[42]: corr_matrix = housing.corr(numeric_only=True)

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

[43]: median_house_value    1.000000
      median_income         0.687151
      total_rooms           0.135140
      housing_median_age    0.114146
      households            0.064590
      total_bedrooms        0.047781
      population           -0.026882
      longitude            -0.047466
      latitude             -0.142673
      Name: median_house_value, dtype: float64
```

Discover and visualize data to gain insights

- Visualize geographical data
- Look for correlation

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

```
Saving figure income_vs_house_value_scatterplot
```



Discover and visualize data to gain insights

- Visualize geographical data
- Look for correlation
- Experiment with attribute combinations

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

```
median_house_value          1.000000
median_income               0.687151
rooms_per_household         0.146255
total_rooms                 0.135140
housing_median_age          0.114146
households                  0.064590
total_bedrooms              0.047781
population_per_household   -0.021991
population                 -0.026882
longitude                  -0.047466
latitude                   -0.142673
bedrooms_per_room          -0.259952
Name: median_house_value, dtype: float64
```

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data

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

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data
- Handling categorical features

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

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data
- Handling categorical features

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

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

Ordinal encoding

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
pd.DataFrame(housing_cat_encoded[:10, 0],
             index=housing_cat.head(10).index,
             columns=["ocean_proximity"])
```

	ocean_proximity
12655	1.0
15502	4.0
2908	1.0
14053	4.0
20496	0.0
1481	3.0
18125	0.0
5830	0.0
17989	0.0
4861	0.0

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data
- Handling categorical features

One-hot encoding

```
from sklearn.preprocessing import OneHotEncoder

# One-hot encode the categorical data
onehot_encoder = OneHotEncoder()
housing_cat_1hot = onehot_encoder.fit_transform(housing_cat)

# Pretty formatting for the presentation
housing_cat_1hot_df = pd.DataFrame(housing_cat_1hot.toarray(),
                                   columns=onehot_encoder.categories_[0],
                                   index=housing_cat.index)
housing_cat_1hot_df.columns = [f"is_{col.replace(' ', '_').lower()}"
                               for col in housing_cat_1hot_df.columns]
housing_cat_1hot_df.head(10)
```

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

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

	is_<1h_ocean	is_inland	is_island	is_near_bay	is_near_ocean
12655	0.0	1.0	0.0	0.0	0.0
15502	0.0	0.0	0.0	0.0	1.0
2908	0.0	1.0	0.0	0.0	0.0
14053	0.0	0.0	0.0	0.0	1.0
20496	1.0	0.0	0.0	0.0	0.0
1481	0.0	0.0	0.0	1.0	0.0
18125	1.0	0.0	0.0	0.0	0.0
5830	1.0	0.0	0.0	0.0	0.0
17989	1.0	0.0	0.0	0.0	0.0
4861	1.0	0.0	0.0	0.0	0.0

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data
- Handling categorical features
- Feature scaling
- Pipeline

```
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)
```

Prepare data for ML algorithms

Most ML algorithms work best with scaled numerical features

- Handling missing data
- Handling categorical features
- Feature scaling
- Pipeline

```
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)
```

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

Select a model and train it

Select a model and train it

- Train a model
  Predict

```python
from sklearn.linear_model import LinearRegression

# Choose the model
lin_reg = LinearRegression()

# Fit to training data
lin_reg.fit(housing_prepared, housing_labels)

# 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: [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

Select a model and train it

- Train a model
  Predict
- Evaluate
  performance

```
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
```

```
68627.87390018745
```

Select a model and train it

- Train a model
  Predict
- Evaluate
  performance

```
from sklearn.tree import DecisionTreeRegressor

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

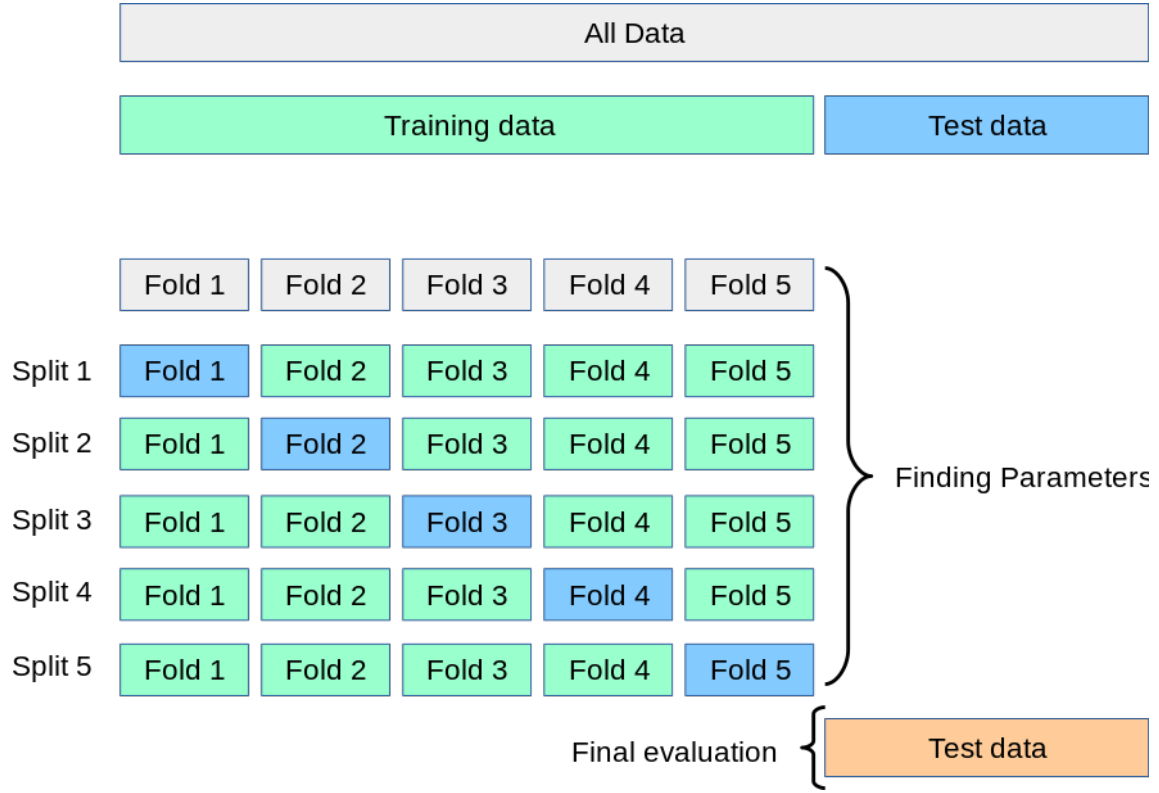
```
▾        DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

```
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
```

Select a model and train it

- Train a model
  Predict
- Evaluate
  performance
- Avoid overfit with
  cross-validation



Select a model and train it

- Train a model
  Predict
- Evaluate performance
- Avoid overfit with cross-validation

```
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)
```

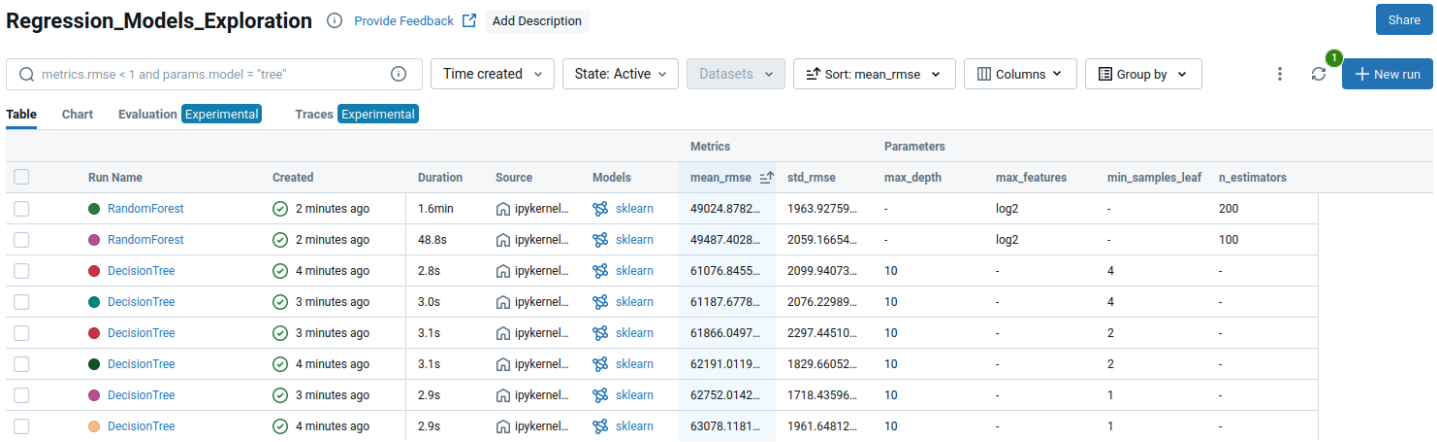
Select a model and train it

- Train a model Predict
- Evaluate performance
- Avoid overfit with cross-validation
- Try out different models/parameters

```python
# Define model and parameters
models = {
    'DecisionTree': (DecisionTreeRegressor(), {'max_depth': [None, 10, 20], 'min_samples_leaf': [1, 2, 4]}),
    'RandomForest': (RandomForestRegressor(), {'n_estimators': [100, 200], 'max_features': ['log2', 'sqrt']}),
    'GradientBoosting': (GradientBoostingRegressor(), {'n_estimators': [100, 200], 'learning_rate': [0.1, 0.05, 0.01]})
}

# Initialize MLflow
mlflow.set_experiment('Regression_Models_Exploration')

for model_name, (model, params) in models.items():
    for param_set in ParameterGrid(params):
        with mlflow.start_run(run_name=f"{model_name}"):
            # Set the parameters and train the model
            model.set_params(**param_set)

            # Perform cross-validation
            scores = cross_val_score(model, housing_prepared, housing_labels,
                                     scoring="neg_mean_squared_error", cv=10)
            rmse_scores = np.sqrt(-scores)

            # Log parameters, metrics, and model
            mlflow.log_params(param_set)
            mlflow.log_metric("mean_rmse", np.mean(rmse_scores))
            mlflow.log_metric("std_rmse", np.std(rmse_scores))

            # Log the model
            mlflow.sklearn.log_model(model, f"model_{model_name}")

            print(f"Logged {model_name} with params {param_set} with mean RMSE: {np.mean(rmse_scores)}")
```

Select a model and train it

- Train a model
  Predict
- Evaluate performance
- Avoid overfit with cross-validation
- Try out different models/parameters



Select a model and train it

- Train a model
  Predict
- Evaluate performance
- Avoid overfit with cross-validation
- Try out different models/parameters
- Save with joblib

```python
# Define model and parameters
models = {
    'DecisionTree': (DecisionTreeRegressor(), {'max_depth': [None, 10, 20], 'min_samples_leaf': [1, 2, 4]}),
    'RandomForest': (RandomForestRegressor(), {'n_estimators': [100, 200], 'max_features': ['log2', 'sqrt']}),
    'GradientBoosting': (GradientBoostingRegressor(), {'n_estimators': [100, 200], 'learning_rate': [0.1, 0.05, 0.01]})
}

best_rmse = float('inf')
best_model = None

# Initialize MLflow
mlflow.set_experiment('Regression_Models_Exploration')

for model_name, (model, params) in models.items():
    for param_set in ParameterGrid(params):
        with mlflow.start_run(run_name=f"{model_name}"):
            # Set the parameters and train the model
            model.set_params(**param_set)

            # Perform cross-validation
            scores = cross_val_score(model, housing_prepared, housing_labels,
                                     scoring="neg_mean_squared_error", cv=10)
            rmse_scores = np.sqrt(-scores)
            mean_rmse = np.mean(rmse_scores)

            # Log parameters, metrics, and model
            mlflow.log_params(param_set)
            mlflow.log_metric("mean_rmse", mean_rmse)
            mlflow.log_metric("std_rmse", np.std(rmse_scores))
            mlflow.sklearn.log_model(model, f"model_{model_name}")

            # Update the best model if the current model is better
            if mean_rmse < best_rmse:
                best_rmse = mean_rmse
                best_model = model

            print(f"Logged {model_name} with params {param_set} with mean RMSE: {mean_rmse}")

# Save the best model
joblib.dump(best_model, 'best_model.joblib')
```

Project overview

1) Look at the big picture – frame the problem
2) Get the data
3) Discover and visualize data to gain insights
4) Prepare data for ML algorithms
5) Select a model and train it
6) Launch, monitor and maintain

Launch, monitor and maintain

Let’s deploy with FastAPI

Launch, monitor and maintain

Let’s deploy with FastAPI

```python
from fastapi import FastAPI
import joblib
import pandas as pd
from pydantic import BaseModel

# Load the data processing pipeline
full_pipeline = joblib.load('full_pipeline.joblib')

# Load the ML model
model = joblib.load('best_model.joblib')

# Launch API
app = FastAPI()

class HousingData(BaseModel):
    longitude: float = 119.0
    latitude: float = 35.0
    housing_median_age: float = 30.0
    total_rooms: float = 2635.0
    total_bedrooms: float = 500.0
    population: float = 1425.0
    households: float = 500.0
    median_income: float = 3.87
    ocean_proximity: str = "INLAND"

@app.post("/predict/")
def predict(housing_data: HousingData):
    # Convert the input data to a DataFrame
    data_dict = housing_data.dict()
    df = pd.DataFrame([data_dict])

    # Use the loaded pipeline to transform the data
    prepared_data = full_pipeline.transform(df)

    # Predict using the loaded model
    prediction = model.predict(prepared_data)
    return {"prediction": prediction.tolist()}
```

Launch, monitor and maintain

Let’s deploy with FastAPI

```python
from fastapi import FastAPI
import joblib
import pandas as pd
from pydantic import BaseModel

# Load the data processing pipeline
full_pipeline = joblib.load('full_pipeline.joblib')

# Load the ML model
model = joblib.load('best_model.joblib')

# Launch API
app = FastAPI()

class HousingData(BaseModel):
    longitude: float = 119.0
    latitude: float = 35.0
    housing_median_age: float = 30.0
    total_rooms: float = 2635.0
    total_bedrooms: float = 500.0
    population: float = 1425.0
    households: float = 500.0
    median_income: float = 3.87
    ocean_proximity: str = "INLAND"

@app.post("/predict/")
def predict(housing_data: HousingData):
    # Convert the input data to a DataFrame
    data_dict = housing_data.dict()
    df = pd.DataFrame([data_dict])

    # Use the loaded pipeline to transform the data
    prepared_data = full_pipeline.transform(df)

    # Predict using the loaded model
    prediction = model.predict(prepared_data)
    return {"prediction": prediction.tolist()}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "longitude": 119,
  "latitude": 35,
  "housing_median_age": 30,
  "total_rooms": 2635,
  "total_bedrooms": 500,
  "population": 1425,
  "households": 500,
  "median_income": 3.87,
  "ocean_proximity": "INLAND"
}'
```

Request URL

```
http://127.0.0.1:8000/predict/
```

Server response

Code	Details
200	**Response body**

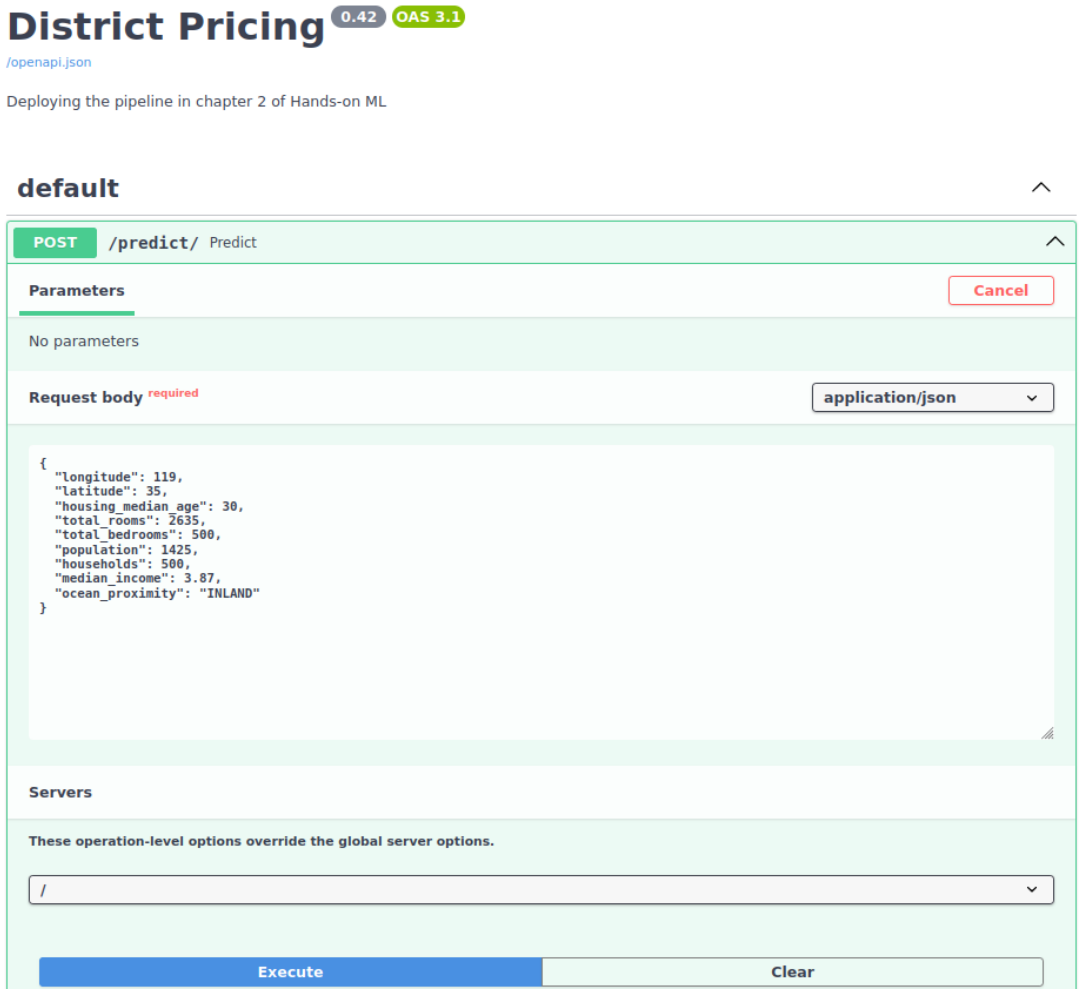
```
{
  "prediction": [
    83991
  ]
}
```

Download

Launch, monitor and maintain

Let’s deploy with FastAPI

```python
from fastapi import FastAPI
import joblib
import pandas as pd
from pydantic import BaseModel

# Load the data processing pipeline
full_pipeline = joblib.load('full_pipeline.joblib')

# Load the ML model
model = joblib.load('best_model.joblib')

# Launch API
app = FastAPI()

class HousingData(BaseModel):
    longitude: float = 119.0
    latitude: float = 35.0
    housing_median_age: float = 30.0
    total_rooms: float = 2635.0
    total_bedrooms: float = 500.0
    population: float = 1425.0
    households: float = 500.0
    median_income: float = 3.87
    ocean_proximity: str = "INLAND"

@app.post("/predict/")
def predict(housing_data: HousingData):
    # Convert the input data to a DataFrame
    data_dict = housing_data.dict()
    df = pd.DataFrame([data_dict])

    # Use the loaded pipeline to transform the data
    prepared_data = full_pipeline.transform(df)

    # Predict using the loaded model
    prediction = model.predict(prepared_data)
    return {"prediction": prediction.tolist()}
```



Current challenges of ML

Current challenges of ML

- Explainability

Current challenges of ML

- Explainability
- Ethical and Social Implications

Current challenges of ML

- Explainability
- Ethical and Social Implications
- Resource Efficiency

Current challenges of ML

- Explainability
- Ethical and Social Implications
- Resource Efficiency
- Security and Privacy

Current challenges of ML

- Explainability
- Ethical and Social Implications
- Resource Efficiency
- Security and Privacy
- Lack of “world model”: machines are stupid

Current challenges of ML

- Explainability
- Ethical and Social Implications
- Resource Efficiency
- Security and Privacy
- Lack of “world model”: machines are stupid
- Multimodal learning
  - Type of data: graphs, images, text, ...
  - Joint representation and decision making