## Today's Lecture

- Course Development Environment Options
  - install new jupyter on 605 computers
- Load, plot, pre-process some example data
- Try three methods:
  - Linear Regression,
  - Decision Tree,
  - Random Forest
- begin discussion of how these work

## Course Development Environment Options

- ssh remote access to xunil-05 (last lecture) from anywhere
- follow last lecture's method on your personal laptop/desktop
- access an online jupyter server
- we will update a local jupyter installation on 605 computers today
- later in quarter: GCP credits ($50/student)

## new install on 605 computers

- anaconda installation in 605 is ancient and can't upgrade
- 605: `C:\Users\abc123>` is on local HDD
  - what you save will not be available on another computer
  - oneDrive sync too slow to serve as ubiquitous home, & worse, non-portable registry keys set w/ installation
  - solution: stay on same 605 computer, or repeat these install steps if you switch. backup notebooks in oneDrive.
- Process we will follow
  1. get a python3 executable via new Anaconda2 environment
  2. set up a virtual environment with that python3
  3. upgrade that virtual environment & install what we need

## new install on 605 computers, continued

1. get a python3 executable via new Anaconda2 environment
   (a) run **jupyter**
   (b) click "conda" tab
   (c) create and name a new python3 environment (e.g. getp3).
   (d) close the jupyter server
2. set up a virtual environment with that python3
   (a) run **cmd**
   (b) `C:\Anaconda2\envs\getp3\python3` **-m venv mlCourse**
   (c) `C:\Users\abc123\mlCourse\Scripts\activate`
3. upgrade that virtual environment & install what we need
   (a) **python3 -m pip install -U pip**
   (b) **python3 -m pip install jupyter matplotlib numpy pandas scipy scikit-learn**
4. run our environments (updated) jupyter server
   (a) **jupyter notebook**
   (b) navigate to `http://localhost:8888/` in chrome

## Today's Lecture

- Course Development Environment Options
  - install new jupyter on 605 computers
- **Load, plot, pre-process some example data**
- Linear Regression
  - block form solution
  - local optimization via gradient descent
  - stochastic gradient descent
- validation & early stopping (review)
- Decision Trees
- SVMs (start)

# Load, plot, pre-process some example data

1. create a new python3 notebook

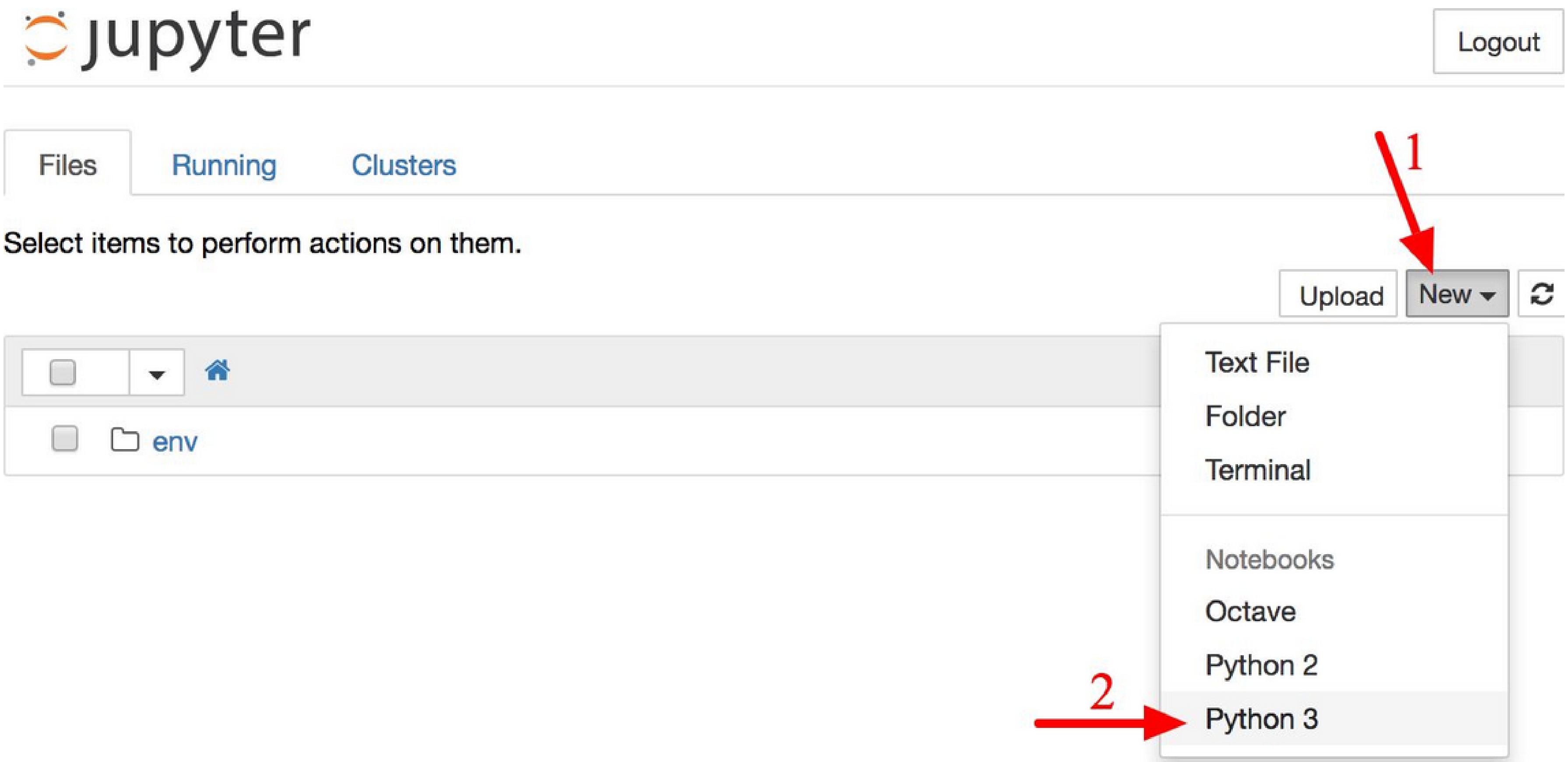


Figure 2-3. Your workspace in Jupyter

## Load, plot, pre-process some example data

1. create a new python3 notebook
2. name it housingExample
3. download example data (california housing prices)

```
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()
```

## Load, plot, pre-process some example data

1. create a new python3 notebook
2. name it housingExample
3. download example data (california housing prices)
4. import with pandas

```
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)
```

## Load, plot, pre-process some example data

1. create a new python3 notebook
2. name it housingExample
3. download example data (california housing prices)
4. import with pandas
5. quick look at the data

```
fetch_housing_data()
housing=load_housing_data()
housing.head()
housing.info()
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50,figsize=(20,15))
plt.show()
```

**observations** – there are missing (null values) in one feature, some attributes capped, widely varying scales, last attribute is categorical. should do some pre-processing

## Load, plot, pre-process some example data

First, set aside some testing data.

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing,test_size=0.2,random_state=42)
housing = train_set.copy()
```

Also may be interested in proportionate representation? see stratified sampling discussion & sklearn method in text. Will revisit this later.

## Load, plot, pre-process some example data

Play a little more with pandas. Scatter plot, radius = population, color = housing price

```
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)
plt.legend()
```

Let's play with inferring median housing price

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

## Load, plot, pre-process some example data

Recall issues we must address:

- missing values in total_bedrooms
- widely varying scales
- ocean_proximity is categorical

## Load, plot, pre-process some example data

Recall issues we must address:

- **missing values in total_bedrooms**
  - *to get rid of instances with missing values, could:*

    ```
    housing.dropna(subset=["total_bedrooms"])
    ```
  - *to drop the attribute entirely, could:*

    ```
    housing.drop("total_bedrooms",axis=1)
    ```
  - *to fill in median whenever missing:*

    ```
    housing_num=housing.drop("ocean_proximity",axis=1)
    from sklearn.impute import SimpleImputer
    imputer=SimpleImputer(strategy="median")
    imputer.fit(housing_num)
    X=imputer.transform(housing_num)
    housing_tr=pd.Dataframe(X,columns=housing_num.columns,
            index=housing_num.index)
    ```
- widely varying scales
- ocean_proximity is categorical

## Load, plot, pre-process some example data

Recall issues we must address:

- missing values in total_bedrooms
- **widely varying scales**
  - MinMaxScaler (set range over training data to 0 to 1 by sub min, divide by max-min)
  - StandardScaler (subtract mean, divide by std.dev.)
- ocean_proximity is categorical

## Load, plot, pre-process some example data

Recall issues we must address:

- missing values in total_bedrooms
- widely varying scales
- **ocean_proximity is categorical**

```
housing_cat = housing[["ocean_proximity"]]
from sklearn.preprocessing import OneHotEncoder
cat_encoder=OneHotEncoder()
housing_cat_1hot=cat_encoder.fit_transform(housing_cat)
```

# Load, plot, pre-process some example data

We can build our own transformer. Let's consider adding some intuitive features

```
from sklearn.base import BaseEstimator, TransformerMixin
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
        def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
                self.add_bedrooms_per_room = add_bedrooms_per_room
        def fit(self, X, y=None):
                return self # nothing else to do
        def transform(self, X, y=None):
                rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
                population_per_household = X[:, population_ix] / X[:, households_ix]
                if self.add_bedrooms_per_room:
                        bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
                        return np.c_[X, rooms_per_household, population_per_household,
bedrooms_per_room]

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

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

## Load, plot, pre-process some example data

For repeatability, organization, and clear code, combine these into a 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)



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)
```

## Try Out Three Different Models

```
from sklearn.linear_model import LinearRegression

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

from sklearn.tree import DecisionTreeRegressor
tree_reg=DecisionTreeRegressor()
tree_reg.fit(housing_prepared,housing_labels)

from sklearn.ensemble import RandomForestRegressor
forest_reg=RandomForestRegressor()
forest_reg.fit(housing_prepared,housing_labels)
```

## Naive Evaluation on Training Data

```
linPreds=lin_reg.predict(housing_prepared)
treePreds=tree_reg.predict(housing_prepared)
forestPreds=forest_reg.predict(housing_prepared)

from sklearn.metrics import mean_squared_error

lin_rmse=np.sqrt(mean_squared_error(housing_labels,linPreds))
tree_rmse=np.sqrt(mean_squared_error(housing_labels,treePreds))
forest_rmse=np.sqrt(mean_squared_error(housing_labels,forestPreds))
print(lin_rmse)
print(tree_rmse)
print(forest_rmse)
```

What did you get? Should you believe this is likely to be an accurate evaluation of their performance?

# Importance of a Validation Set

divide data into k folds, use 1 to validate and rest to train, save scores

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

scores=cross_val_score(forest_reg,housing_prepared,housing_labels,
        scoring="neg_mean_squared_error",cv=10)
forest_rms_scores=np.sqrt(-scores)

print(tree_rms_scores)
print(forest_rms_scores)
```