AGENDA

1. Scikit-Learn API
2. DEPLOYMENT
3. Deploying with Containers (Docker) Introduction

Data Science Academy

I

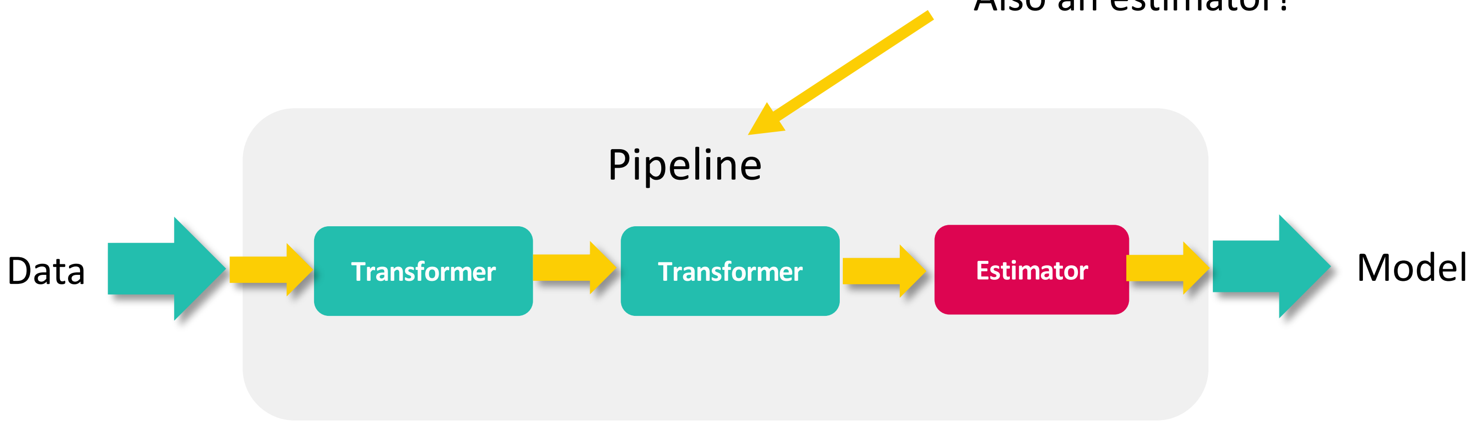
Scikit-Learn API

Data Science Academy

Scikit-Learn Objects

- ○ Estimators
- ○ Transformers
- ○ Pipeline

Pipelines are estimators



Scikit-Learn Estimators

- Estimator – A class with fit() and predict() methods.
- It fits and predicts.
- Any ML algorithm like Lasso, Decision trees, SVMs, are coded as estimators within Scikit-Learn.

```
class Estimator(object):
     def fit(self, X, y = None):
           Fit the estimator to data.
           return self

    def predict(self, X):
           Compute the predictions
           return predictions
```

Scikit-Learn Transformers

- Transformers – class that have fit() and transform() methods.
- It transforms data:
  - Scalers
  - Feature selectors
  - One hot encoders
  - Customizable objective
- This is the core of Feature Engine

```
class Transformer(object):
    def fit(self, X, y = None):
         Learn the parameters to
          engineer the   features

   def transform(X):
         Transforms the input data
         return X_transformed
```

Scikit-Learn Transformers

- Missing Data Imputation
  - SimpleImputer
  - IterativeImputer
- Categorical Variable Encoding
  - OneHotEncoder
  - OrdinalEncoder
- Discretisation
  - KBinsDiscretizer
- Variable Transformation
  - PowerTranformer
  - FunctionTransformer

Scikit-Learn Pipeline

- Pipeline – class that allows to run transformers and estimators in sequence.
- Most steps are Transformers
- Last step can be an Estimator

```
class Pipeline(Transformer):
    @property
     def name_steps(self):
        Sequence of transformers
      return self.steps
    @property
      def _final_estimator(self):
         Estimator
       return self.steps[-1]
```

Scikit-Learn Pipeline

- Scikit-Learn is a Python library that provides a solid implementation of a range of machine learning algorithms.
- Scikit-Learn provides efficient versions of a large number of common algorithms.
- Scikit-Learn is characterized by a clean, uniform, and streamlined API.
- Scikit-Learn is written so that most of its algorithms follow the same functionality
- Once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward
- Scikit-Learn provides useful and complete online documentation that allows you to understand both what the algorithm is about and how to use it from scikit-learn
- Scikit-Learn is so well established in the community, that new packages are typically designed following scikit-learn functionality to be quickly adopted by end users, e.g, Keras, MLXtend.

Scikit-Learn Pipeline

Scikit-Learn Objects

- Transformers - class that have fit and transform method, it transforms data
    - Scalers
    - Feature selectors
    - One hot encoders
- Estimator - class that has fit and predict methods, it fits and predicts.
    - Any ML algorithm like lasso, decision trees, svm, etc
- Pipeline - class that allows you to list and run transformers and predictors in sequence
    - All steps should be transformers except the last one
    - Last step should be a predictor

Scikit-Learn Pipeline

- Here is a good example of Pipeline usage. Pipeline gives you a single interface for all 3 steps of transformation and resulting estimator. It encapsulates transformers and predictors inside, and now you can do something like:

```
vect = CountVectorizer()
tfidf = TfidfTransformer()
clf = SGDClassifier()

vX = vect.fit_transform(Xtrain)
tfidfX = tfidf.fit_transform(vX)
Predicted = clf.fit_predict(tfidfX)

# Now evaluate all steps on test set
vX = vect.transform(Xtest)
tfidfX = tfidf.transform(vX)
predicted = clf.predict(tfidfx)
```

```
pipeline = Pipeline([
      (‘vect’ , CountVectorizer()),
      (‘tfidf’ , TfidfTransformer()),
      (‘clf’ , SGDClassifier()),
   ])
predicted = pipeline.fit(Xtrain).predict(Xtrain)

# Now evaluate all steps on test set
predicted = pipeline.predict(Xtest)
```

Scikit-Learn Pipeline

- Feature Creation and Feature Engineering steps as Scikit-Learn Objects
- Transformers - class that have fit and transform method, it transforms data
- Use of scikit-learn base transformers
  - Inherit class and adjust the fit and transform methods

Scikit-Learn Pipeline

- Advantages
  - Can be tested, versioned, tracked and controlled
  - Can build future models on top
  - Good software developer practice
  - Leverages the power of acknowledged API
  - Data scientists familiar with Pipeline use, reduced over-head
  - Engineering steps can be packaged and re-used in future ML models
- Disadvantages
  - Requires team of software developers to build and maintain
  - Overhead for software developers to familiarize with code for Sk-Learn API ⇒ difficulties debugging

Scikit-Learn API documentation

- https://scikit-learn.org/stable/modules/classes.html
- base.BaseEstimator
- base.TransformerMixin

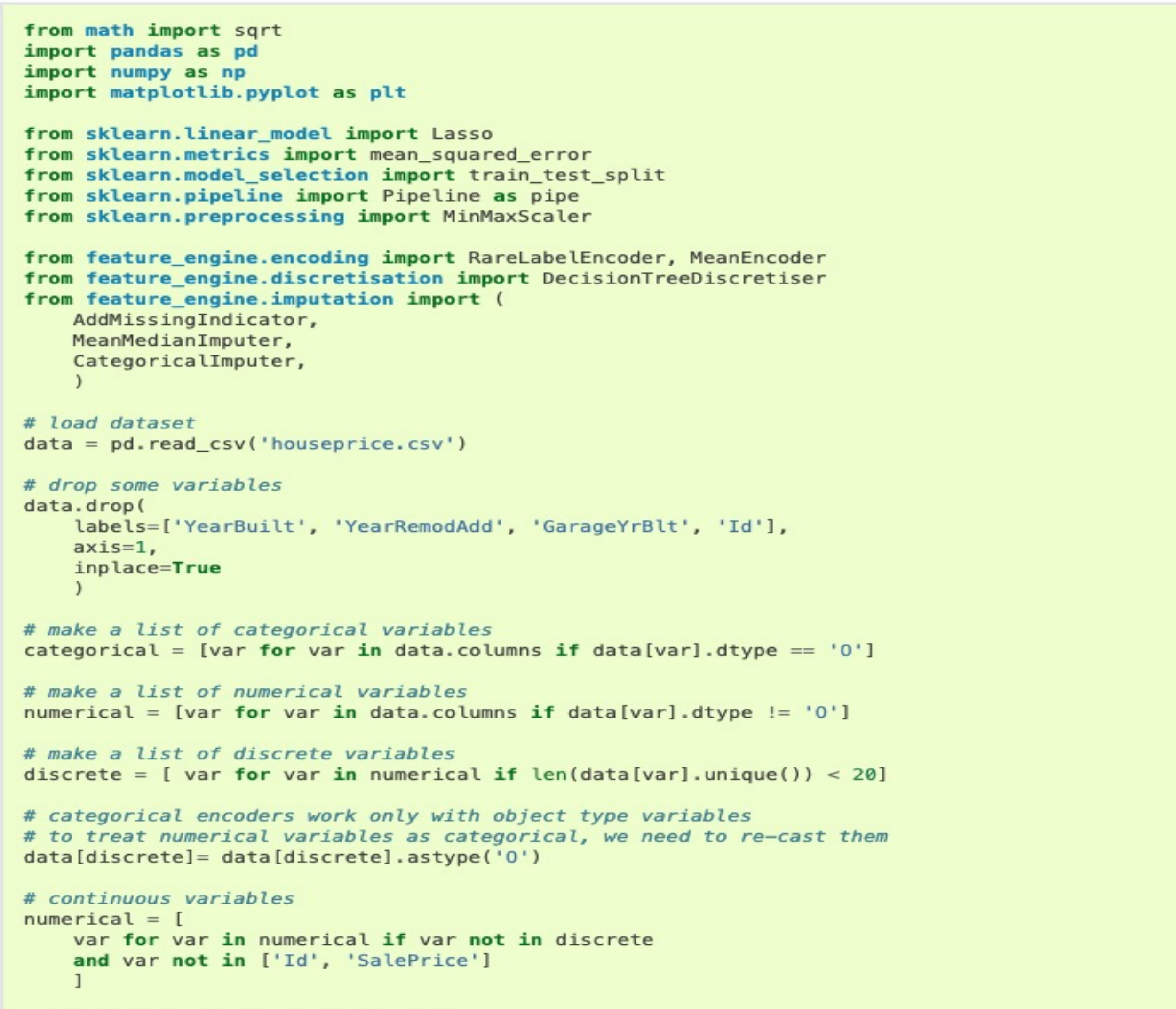
FEATURE-ENGINE

Data Science Academy

Feature Engine Transformers

- Missing Data Imputation
  - MeanMedianImputer
  - RandomSampleImputer
  - EndTailImputer
  - AddNaNBinaryImputer
  - CategoricalVariableImputer
  - FrequentCategoryImputer
- Categorical Variable Encoding
  - CountFrequencyCategoricalEncoder
  - OrdinalCategoricalEncoder
  - MeanCategoricalEncoder
  - WoERatioCategoricalEncoder
  - OneHotCategoricalEncoder
  - RareLabelCategoricalEncoder
- Outlier Removal
  - Windsorizer
  - ArbitaryOutlierCapper
- Discretisation
  - EqualFrequencyDiscretiser
  - EqualWidthDiscretiser
  - DecisionTreeDiscretiser
- Variable Transformation
  - LogTransformer
  - ReciprocalTransformer
  - ExponentialTransformer
  - BoxCoxTransformer

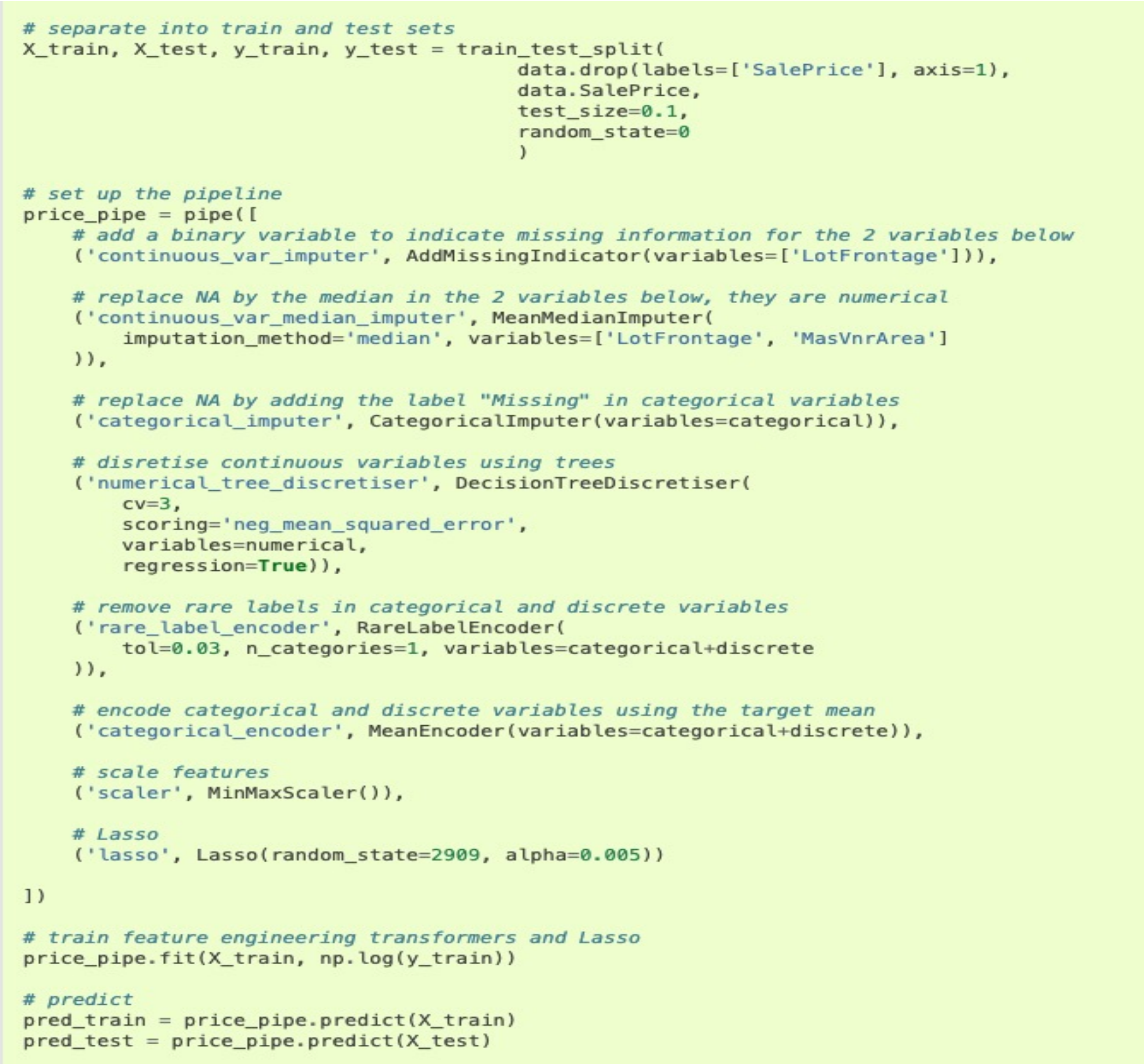
Pipeline with Feature-engine



```
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline as pipe
from sklearn.preprocessing import MinMaxScaler

from feature_engine.encoding import RareLabelEncoder, MeanEncoder
from feature_engine.discretisation import DecisionTreeDiscretiser
from feature_engine.imputation import (
    AddMissingIndicator,
    MeanMedianImputer,
    CategoricalImputer,
    )

# load dataset
data = pd.read_csv('houseprice.csv')

# drop some variables
data.drop(
    labels=['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'Id'],
    axis=1,
    inplace=True
    )

# make a list of categorical variables
categorical = [var for var in data.columns if data[var].dtype == 'O']

# make a list of numerical variables
numerical = [var for var in data.columns if data[var].dtype != 'O']

# make a list of discrete variables
discrete = [ var for var in numerical if len(data[var].unique()) < 20]

# categorical encoders work only with object type variables
# to treat numerical variables as categorical, we need to re-cast them
data[discrete]= data[discrete].astype('O')

# continuous variables
numerical = [
    var for var in numerical if var not in discrete
    and var not in ['Id', 'SalePrice']
    ]
```

Import predefined transformers

Pipeline with Feature-engine



```
# separate into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
                                        data.drop(labels=['SalePrice'], axis=1),
                                        data.SalePrice,
                                        test_size=0.1,
                                        random_state=0
                                        )

# set up the pipeline
price_pipe = pipe([
    # add a binary variable to indicate missing information for the 2 variables below
    ('continuous_var_imputer', AddMissingIndicator(variables=['LotFrontage'])),

    # replace NA by the median in the 2 variables below, they are numerical
    ('continuous_var_median_imputer', MeanMedianImputer(
        imputation_method='median', variables=['LotFrontage', 'MasVnrArea']
    )),

    # replace NA by adding the label "Missing" in categorical variables
    ('categorical_imputer', CategoricalImputer(variables=categorical)),

    # disretise continuous variables using trees
    ('numerical_tree_discretiser', DecisionTreeDiscretiser(
        cv=3,
        scoring='neg_mean_squared_error',
        variables=numerical,
        regression=True)),

    # remove rare labels in categorical and discrete variables
    ('rare_label_encoder', RareLabelEncoder(
        tol=0.03, n_categories=1, variables=categorical+discrete
    )),

    # encode categorical and discrete variables using the target mean
    ('categorical_encoder', MeanEncoder(variables=categorical+discrete)),

    # scale features
    ('scaler', MinMaxScaler()),

    # Lasso
    ('lasso', Lasso(random_state=2909, alpha=0.005))

])

# train feature engineering transformers and Lasso
price_pipe.fit(X_train, np.log(y_train))

# predict
pred_train = price_pipe.predict(X_train)
pred_test = price_pipe.predict(X_test)
```

Accommodate transformers in pipeline

Fit the pipeline and make predictions

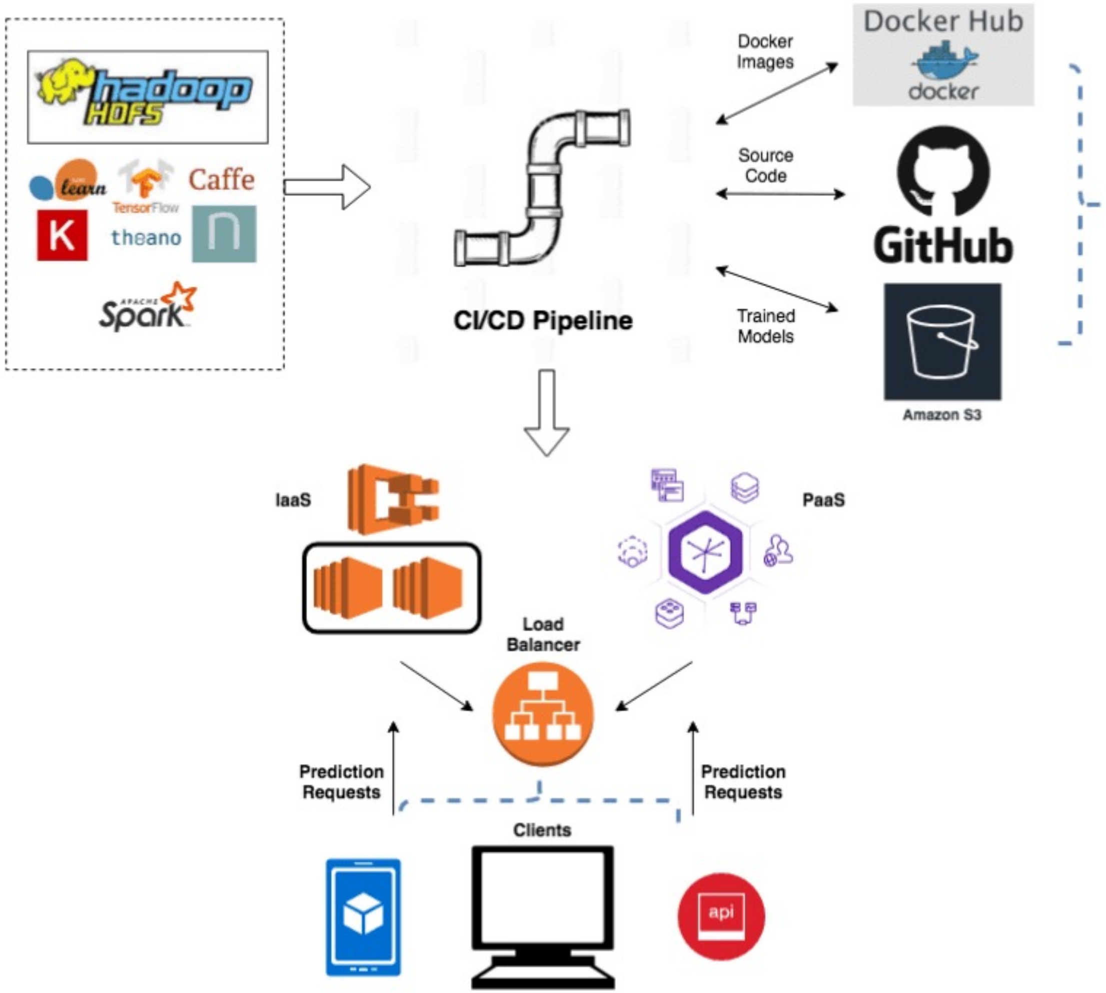
II

DEPLOYMENT

Data Science Academy

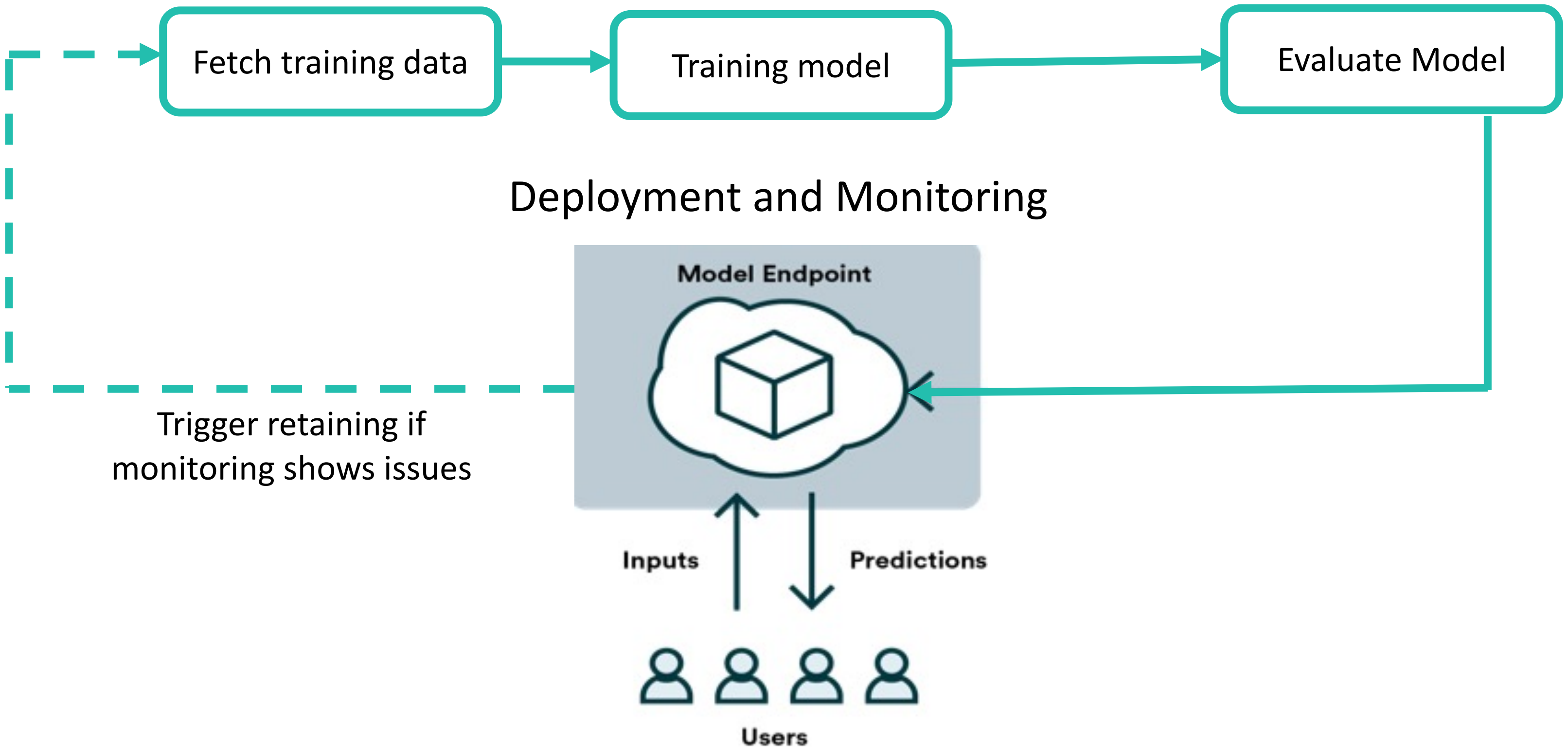
Machine Learning Deployment Outline

- What is Model Deployment?
  - Preparation with Sk-Learn API: Transformer, Estimator, Pipeline
  - Feature Selection
- Deployment with Docker



Machine Learning Deployment

- Deployment is the application of a model for prediction using a new data.



Fetch training data

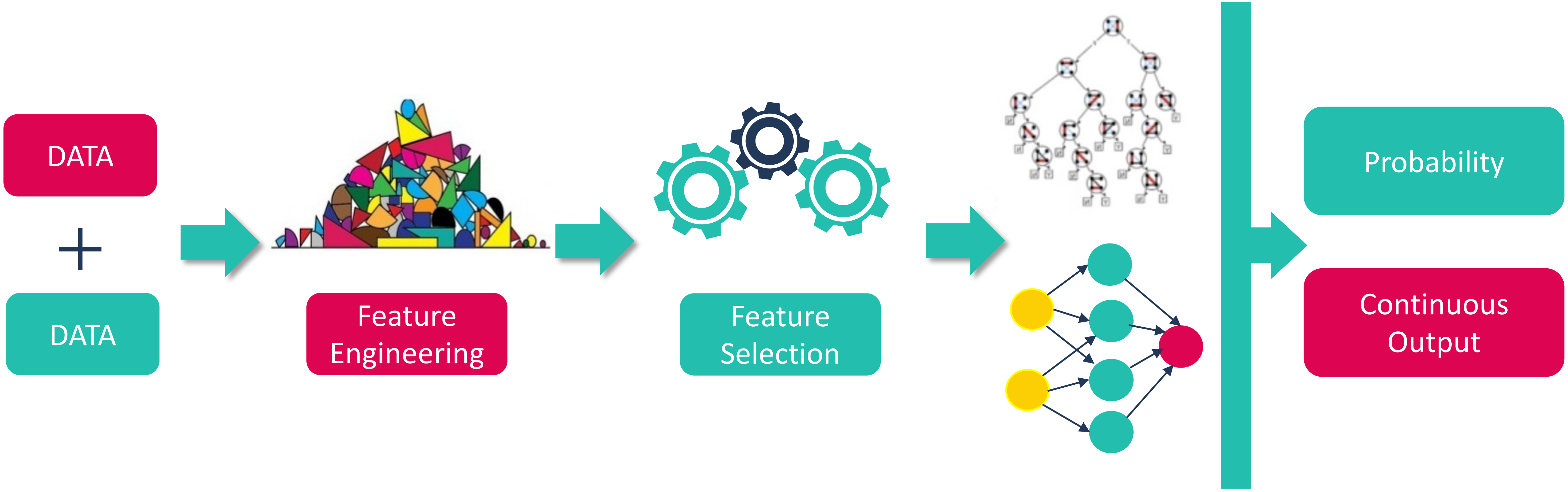
Training model

Evaluate Model

Deployment and Monitoring

Trigger retaining if monitoring shows issues

Machine Learning Pipeline: Production



DATA

+

DATA

Feature Engineering

Feature Selection

Probability

Continuous Output

Gathering Data Sources

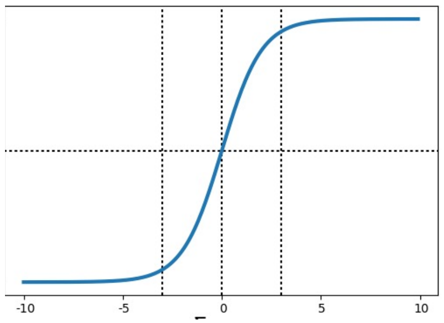
Data Pre-processing

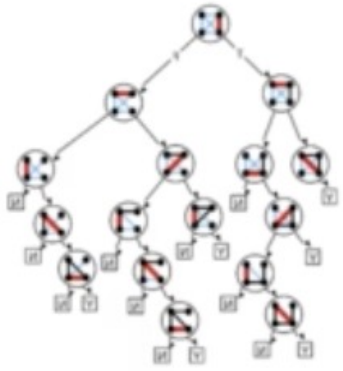
Variable selection

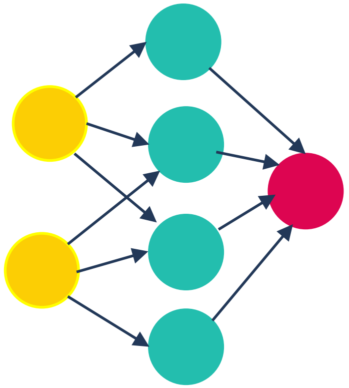
Machine Learning Model building

Machine Learning Model Building

DATA
Prediction







Linear Models

Logistic Regression

MARS

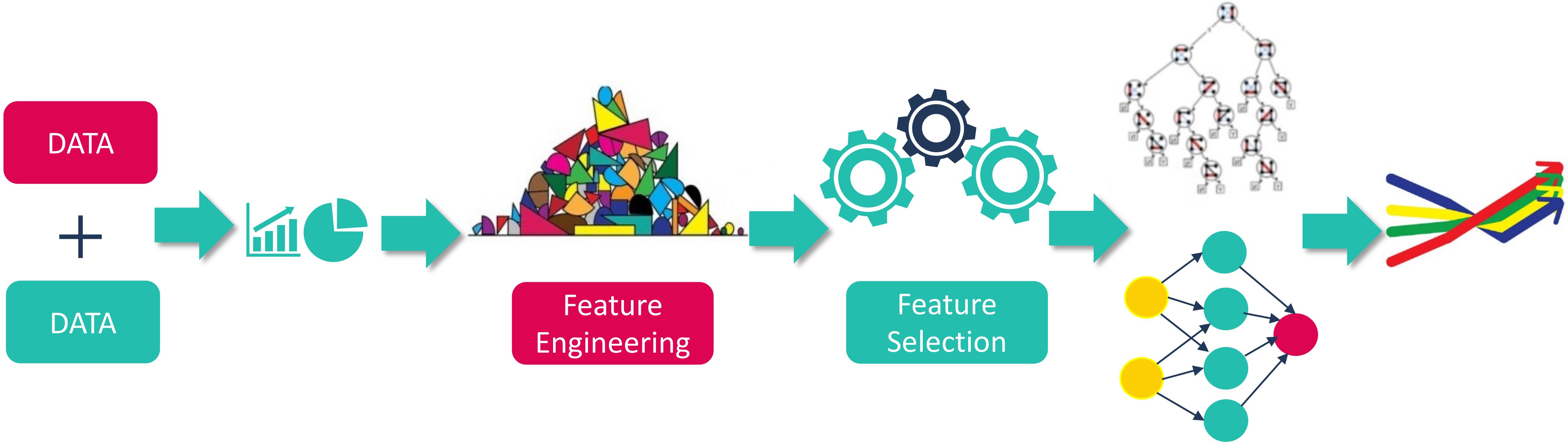
Tree Models

Random Forests

Gradient Boosted Trees

Neural Networks

Machine Learning Pipeline: Overview



DATA

+

DATA

Feature Engineering

Feature Selection

Gathering Data Sources

Data Pre-processing

Variable selection

Machine Learning Model building

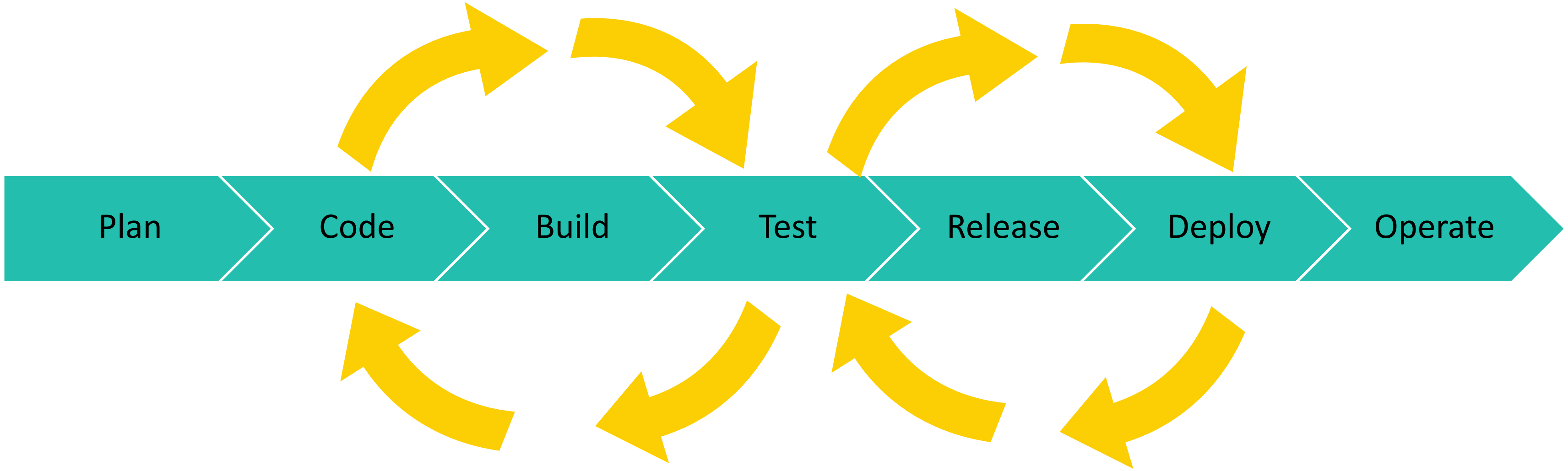
Model Deployment

Towards deployment code

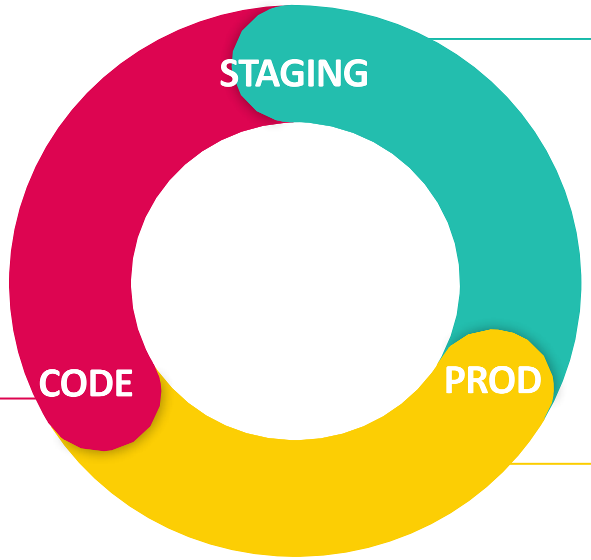
- Code to:
  - Create and Transform features
  - Incorporate the feature selection
  - Build Machine Learning Models
  - Score new data

What is CI/CD

- **C**ontinuous **I**ntegration is an automated process for continually integrating software development changes. CI processes automate the building, testing, and validation of source code.
- The goal of **C**ontinuous **D**elivery is to deliver a packaged artifact into a production environment. CD automates the entire delivery process, including the deployment process



Feature selection in CI/CD



- ○ Tests

- ○ New model
- ○ Model refresh

- ○ Model deployed
- ○ Automatically available to other systems

Feature selection in CI/CD

- Advantages
  - Reduced overhead in the implementation of the new model
  - The new model is almost immediately available to the business systems

- Disadvantages
  - Lack of data versatility
  - No additional data can be fed through the pipeline, as the entire processes are based on the first dataset on which it was built

Feature selection in CI/CD

- Including a feature selection algorithm as part of the pipeline, ensures that the most useful ones are selected from all the available features to train the model.
  - Potentially avoids overfitting
  - Enhances model interpretability
- However, we would need to deploy code to engineer all available features in the dataset, regardless of whether they will be finally used by the model. For example : Error handling and Unit testing

Feature selection in CI/CD

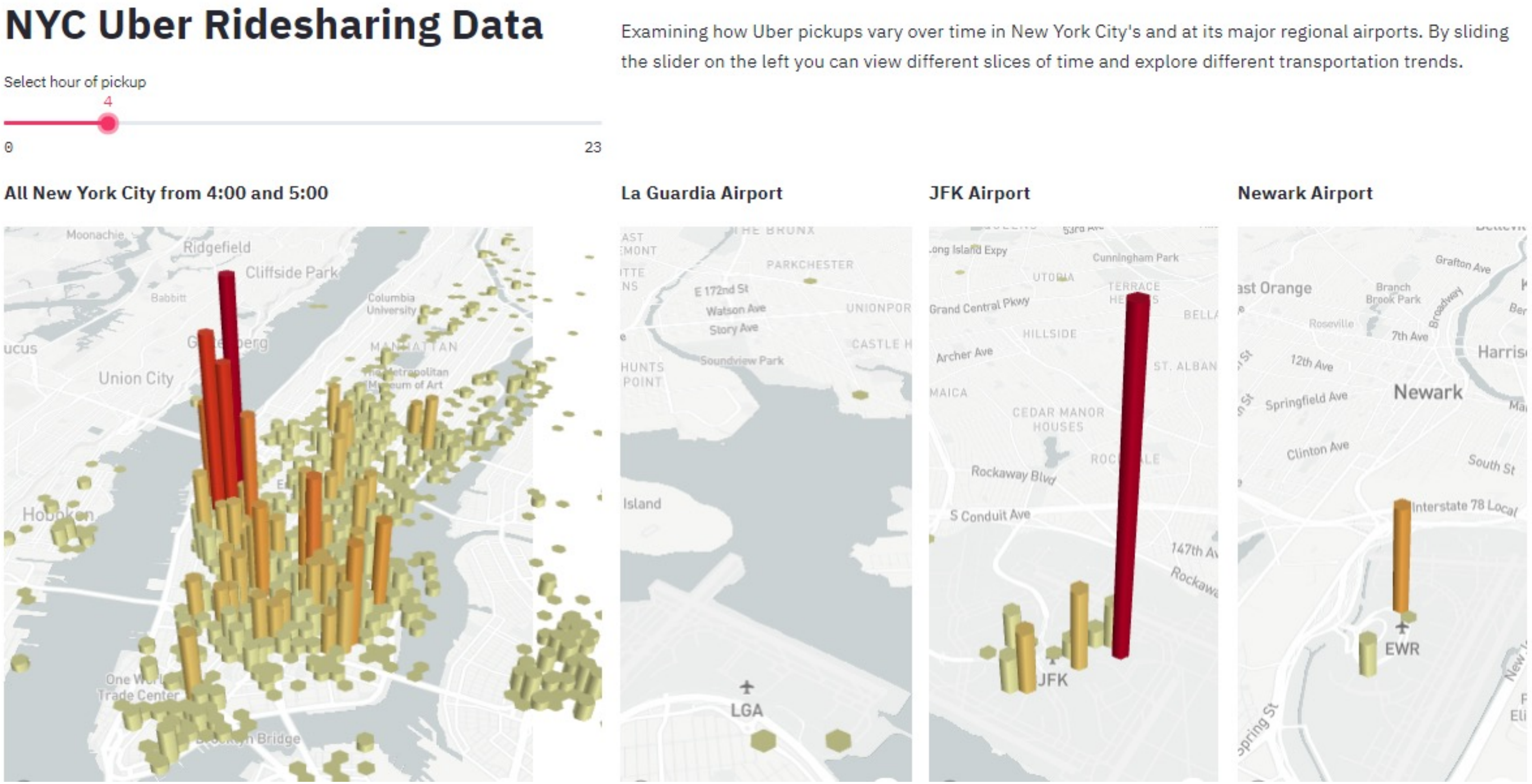
- Suitable if:
  - Model build and refresh on same data
  - Model build and refresh on smaller datasets
- Not suitable if:
  - Model is built using datasets with a high feature space
  - Model is constantly enriched with new data sources

Streamlit.io

The fastest way to build and share data apps. Streamlit turns data scripts into shareable web apps in minutes.

- $ pip install streamlit
- $ streamlit hello
  - Open a new Python file, import streamlit, and write some code
  - Run the file with: streamlit run [filename]
  - When you’re ready, click ‘Deploy’ from the streamlit menu

Streamlit.io



NYC Uber Ridesharing Data

Examining how Uber pickups vary over time in New York City's and at its major regional airports. By sliding the slider on the left you can view different slices of time and explore different transportation trends.

Select hour of pickup

4

0 23

All New York City from 4:00 and 5:00

La Guardia Airport

JFK Airport

Newark Airport

Streamlit.io – Display Texts

- Streamlit apps usually start with a call to **st.title** to set the app's title. After that, there are 2 heading levels you can use: **st.header** and **st.subheader.** Pure text is entered with **st.text**, and Markdown with **st.markdown**.
  - Example:

In [ ]:

```
>>> st.text('This is some text.')
```

This is some text.

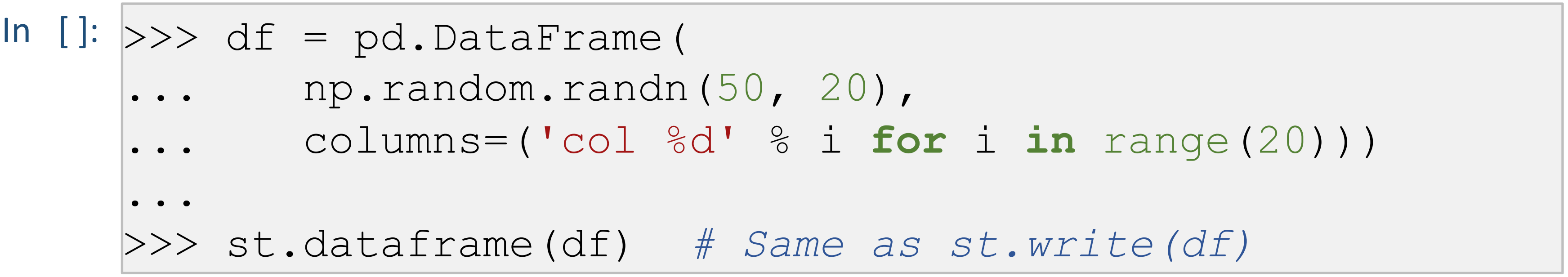
In [ ]:

```
>>> st.markdown('Streamlit is **_really_ cool**.')
```

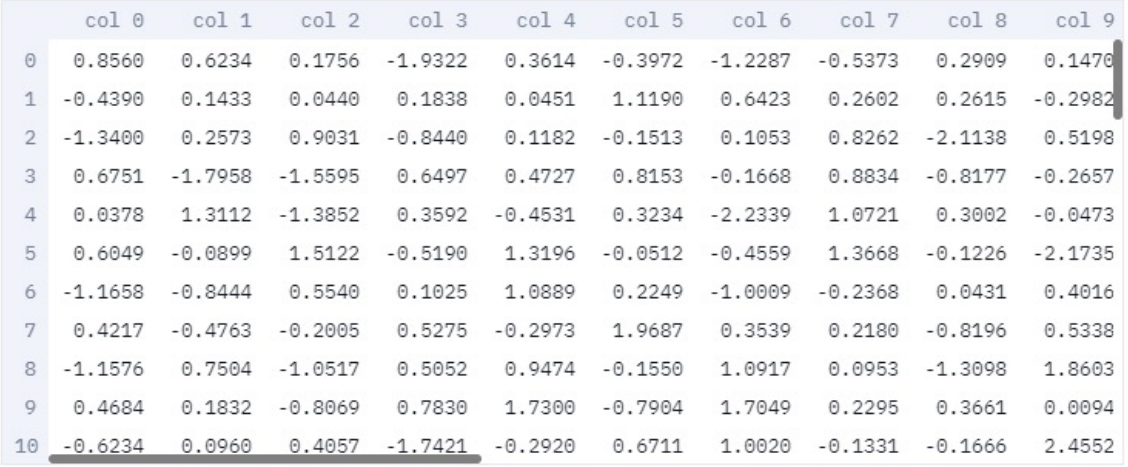
Streamlit is ***really* cool**.

Streamlit.io – Display Data

- You can display data via charts, and you can display it in raw form. These are the Streamlit commands you can use to display raw data:



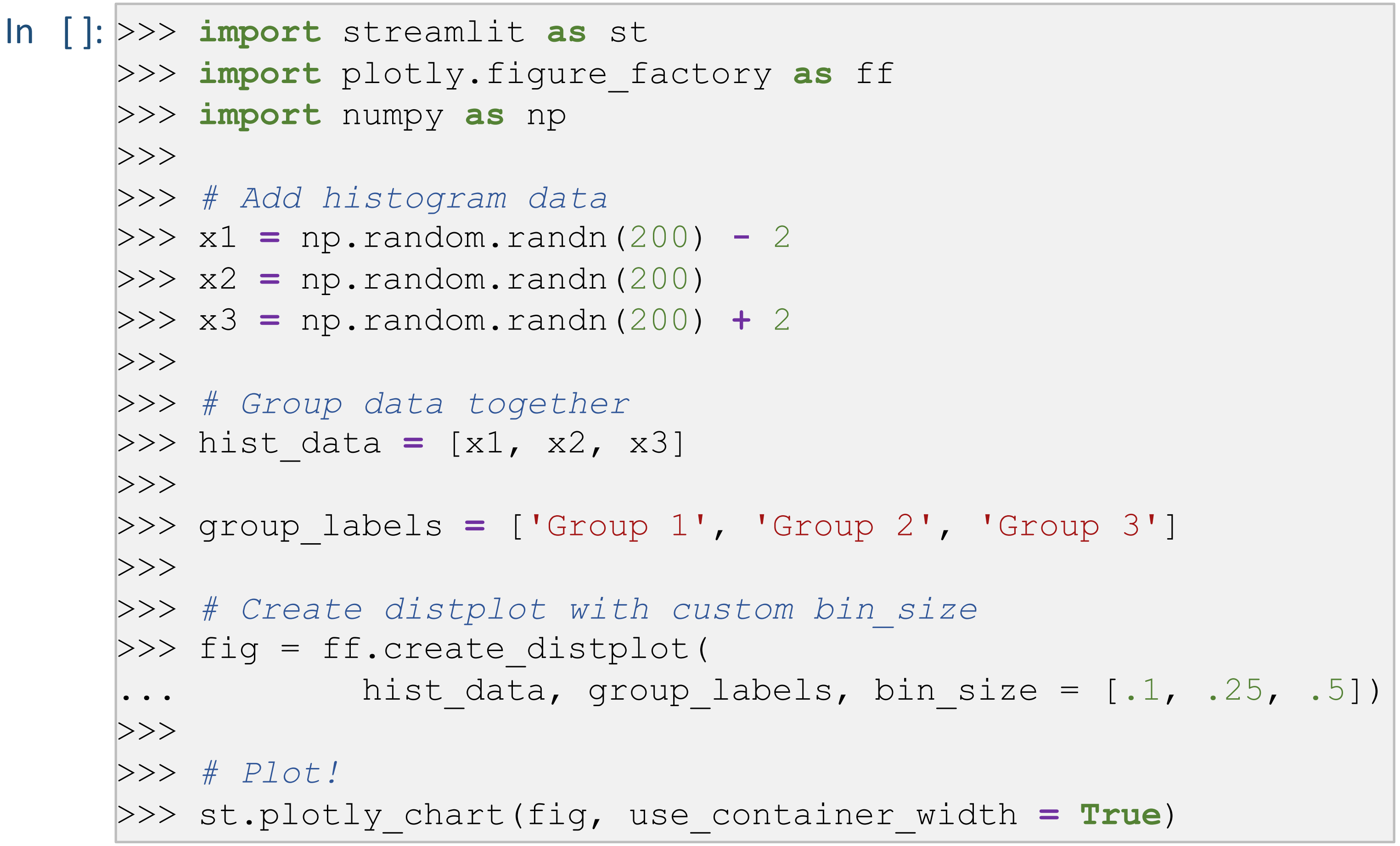
```
In [ ]: >>> df = pd.DataFrame(
        ...     np.random.randn(50, 20),
        ...     columns=('col %d' % i for i in range(20)))
        ...
        >>> st.dataframe(df)   # Same as st.write(df)
```



	col 0	col 1	col 2	col 3	col 4	col 5	col 6	col 7	col 8	col 9
0	0.8560	0.6234	0.1756	-1.9322	0.3614	-0.3972	-1.2287	-0.5373	0.2909	0.1470
1	-0.4390	0.1433	0.0440	0.1838	0.0451	1.1190	0.6423	0.2602	0.2615	-0.2982
2	-1.3400	0.2573	0.9031	-0.8440	0.1182	-0.1513	0.1053	0.8262	-2.1138	0.5198
3	0.6751	-1.7958	-1.5595	0.6497	0.4727	0.8153	-0.1668	0.8834	-0.8177	-0.2657
4	0.0378	1.3112	-1.3852	0.3592	-0.4531	0.3234	-2.2339	1.0721	0.3002	-0.0473
5	0.6049	-0.0899	1.5122	-0.5190	1.3196	-0.0512	-0.4559	1.3668	-0.1226	-2.1735
6	-1.1658	-0.8444	0.5540	0.1025	1.0889	0.2249	-1.0009	-0.2368	0.0431	0.4016
7	0.4217	-0.4763	-0.2005	0.5275	-0.2973	1.9687	0.3539	0.2180	-0.8196	0.5338
8	-1.1576	0.7504	-1.0517	0.5052	0.9474	-0.1550	1.0917	0.0953	-1.3098	1.8603
9	0.4684	0.1832	-0.8069	0.7830	1.7300	-0.7904	1.7049	0.2295	0.3661	0.0094
10	-0.6234	0.0960	0.4057	-1.7421	-0.2920	0.6711	1.0020	-0.1331	-0.1666	2.4552

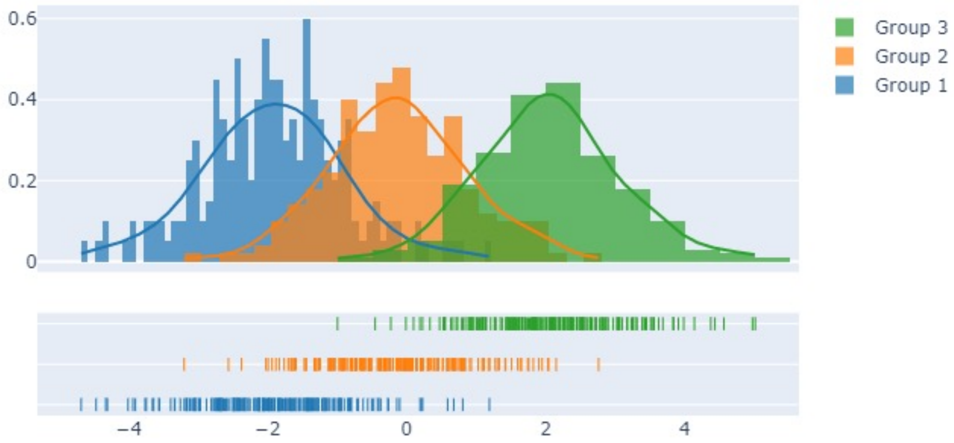
Streamlit.io – Display Charts

○ Plotly is an interactive charting library for Python. The arguments to this function closely follow the ones for Plotly’s plot() function.



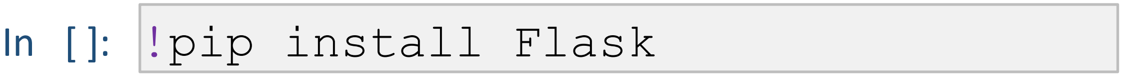
In [ ]:

```
>>> import streamlit as st
>>> import plotly.figure_factory as ff
>>> import numpy as np
>>>
>>> # Add histogram data
>>> x1 = np.random.randn(200) - 2
>>> x2 = np.random.randn(200)
>>> x3 = np.random.randn(200) + 2
>>>
>>> # Group data together
>>> hist_data = [x1, x2, x3]
>>>
>>> group_labels = ['Group 1', 'Group 2', 'Group 3']
>>>
>>> # Create distplot with custom bin_size
>>> fig = ff.create_distplot(
...         hist_data, group_labels, bin_size = [.1, .25, .5])
>>>
>>> # Plot!
>>> st.plotly_chart(fig, use_container_width = True)
```

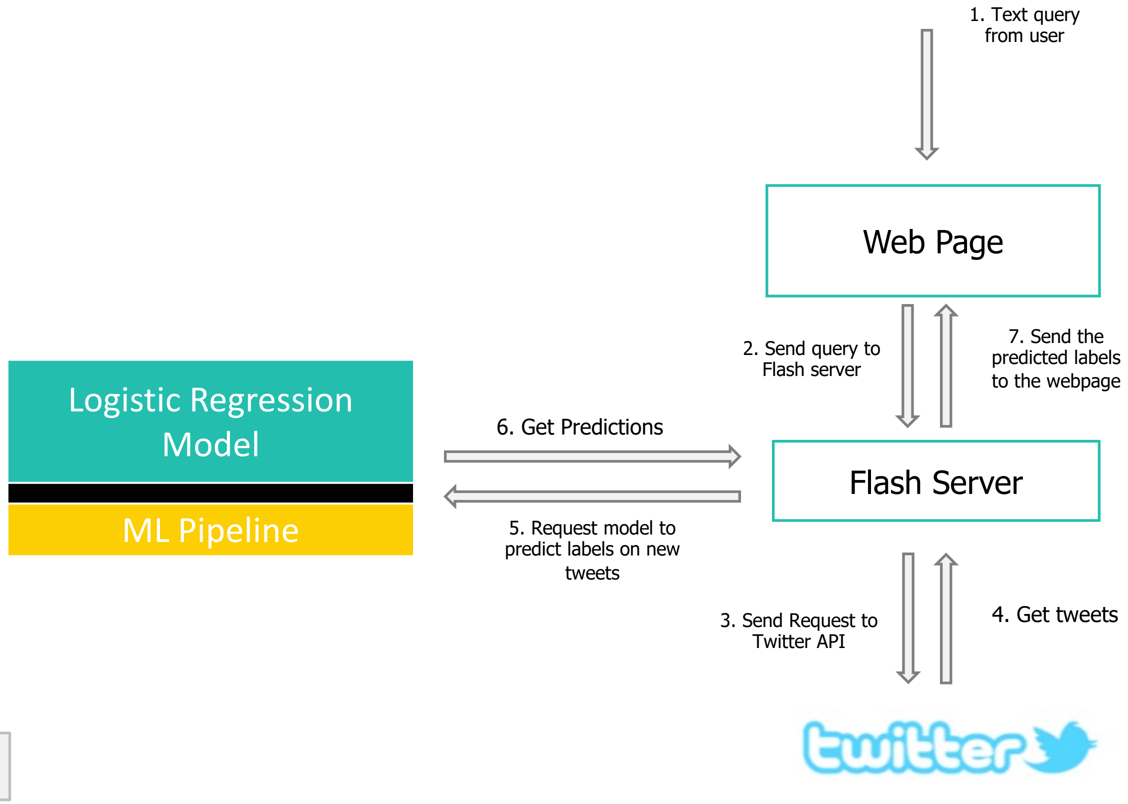


Flask

- Flask is a web application framework written in Python. It has multiple modules that make it easier for a web developer to write applications without having to worry about the details like protocol management, thread management, etc.



```
In [ ]: !pip install Flask
```



1. Text query from user

Web Page

2. Send query to Flash server

7. Send the predicted labels to the webpage

Logistic Regression Model

ML Pipeline

6. Get Predictions

5. Request model to predict labels on new tweets

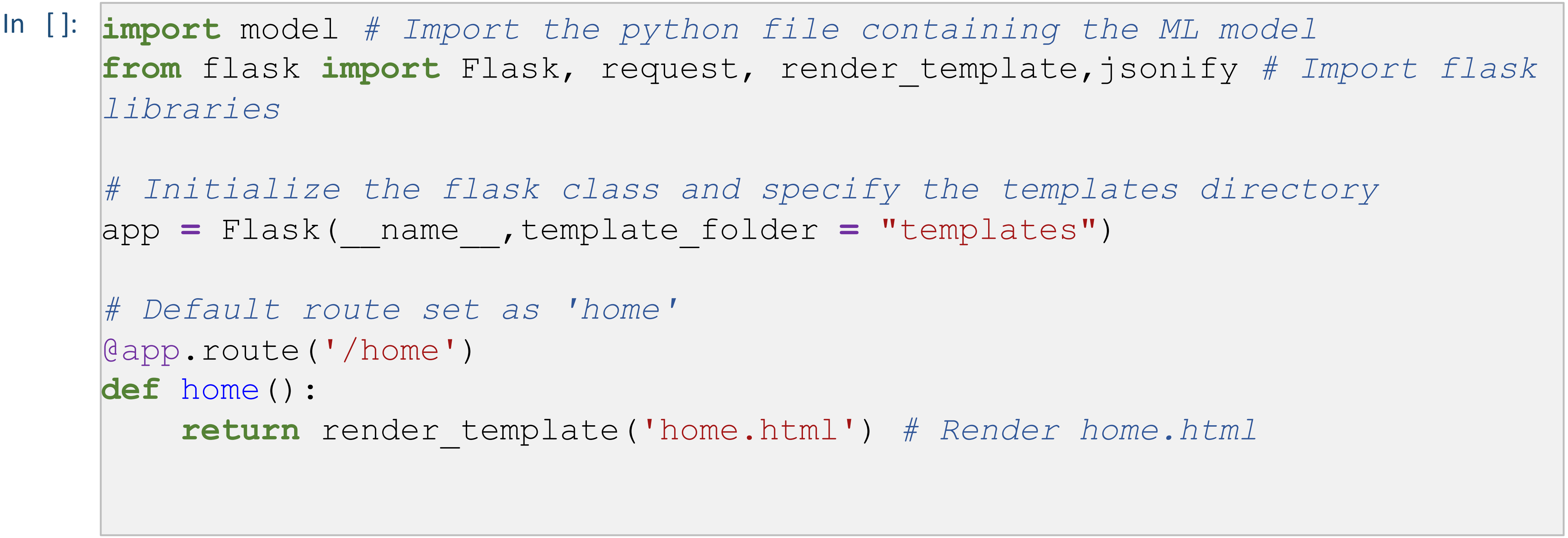
Flash Server

3. Send Request to Twitter API

4. Get tweets

twitter

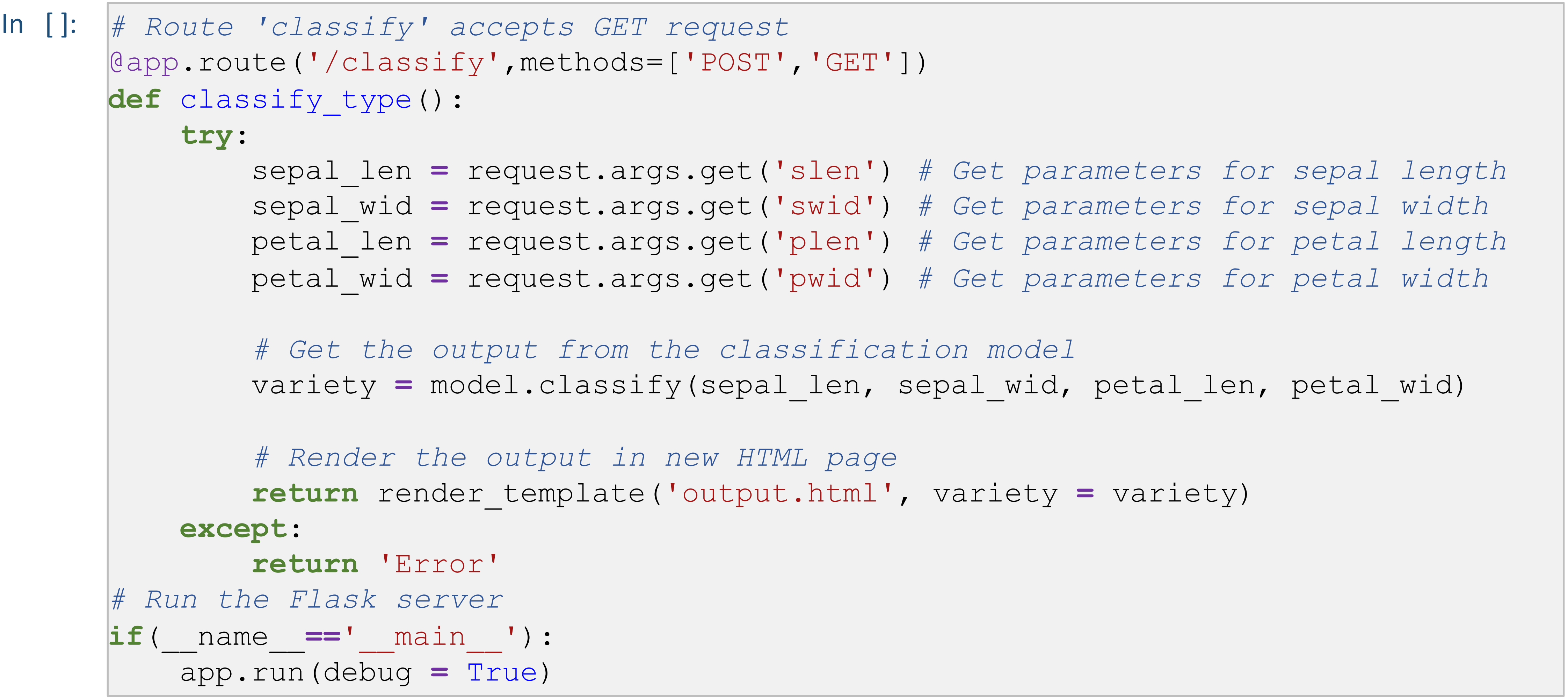
Flask



In [ ]:

```python
import model # Import the python file containing the ML model
from flask import Flask, request, render_template,jsonify # Import flask libraries

# Initialize the flask class and specify the templates directory
app = Flask(__name__,template_folder = "templates")

# Default route set as 'home'
@app.route('/home')
def home():
    return render_template('home.html') # Render home.html
```

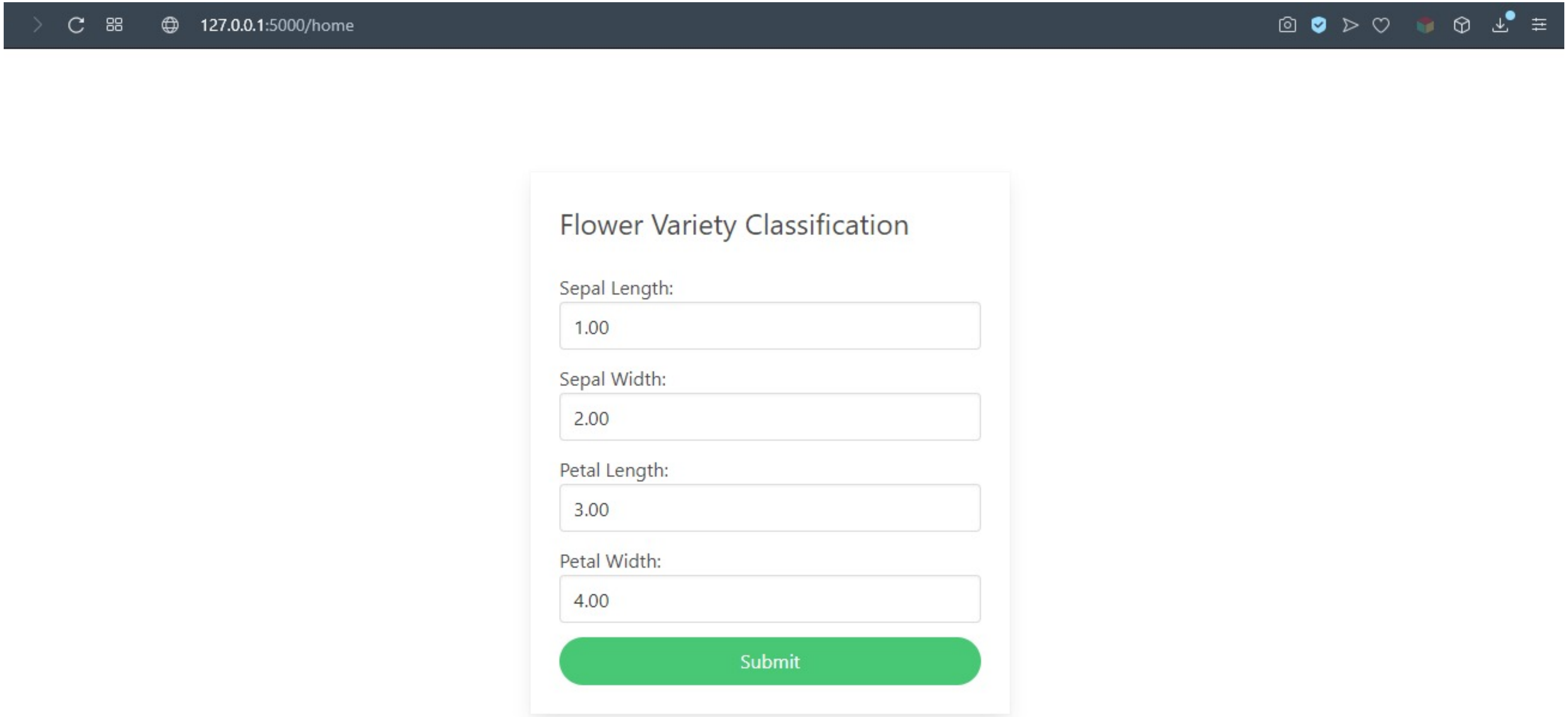
Flask



In [ ]:

```
# Route 'classify' accepts GET request
@app.route('/classify',methods=['POST','GET'])
def classify_type():
    try:
        sepal_len = request.args.get('slen') # Get parameters for sepal length
        sepal_wid = request.args.get('swid') # Get parameters for sepal width
        petal_len = request.args.get('plen') # Get parameters for petal length
        petal_wid = request.args.get('pwid') # Get parameters for petal width

        # Get the output from the classification model
        variety = model.classify(sepal_len, sepal_wid, petal_len, petal_wid)

        # Render the output in new HTML page
        return render_template('output.html', variety = variety)
    except:
        return 'Error'
# Run the Flask server
if(__name__=='__main__'):
    app.run(debug = True)
```

Flask



Flower Variety Classification

Sepal Length: 1.00

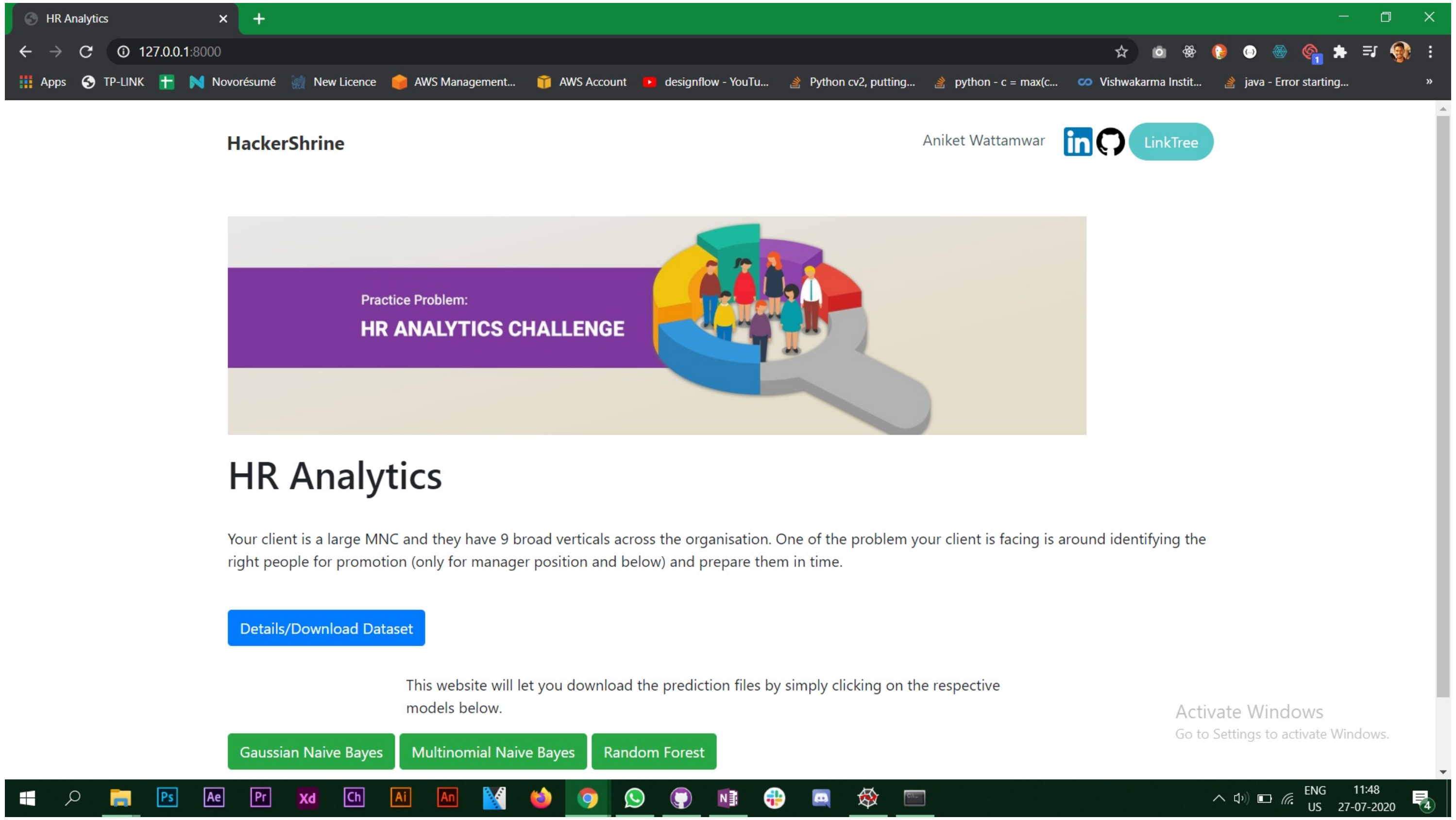
Sepal Width: 2.00

Petal Length: 3.00

Petal Width: 4.00

Django

- The Django web framework is the most advanced way of deploying a machine learning model and is capable of building large and complex scale web applications. Websites like Instagram, Washington Post, and Pinterest all use Django to weave machine learning models into their application.



HR Analytics

Your client is a large MNC and they have 9 broad verticals across the organisation. One of the problem your client is facing is around identifying the right people for promotion (only for manager position and below) and prepare them in time.

This website will let you download the prediction files by simply clicking on the respective models below.

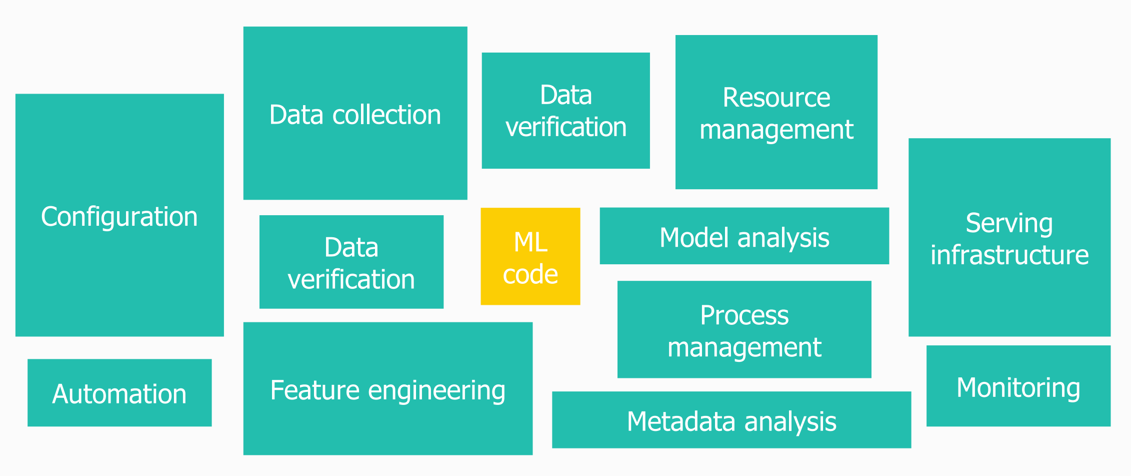
III

Deploying with Containers (Docker) Introduction

Data Science Academy

ML Deployment Methods

- Docker containers
- AWS Sagemaker – (Train Jobs, Inference Jobs, Endpoints, Processing Jobs)
- REST API – Python/Flask
- Serverless APIs - AWS Lambda Functions, Function as a service, FaaS
- Metaflow by Netflix



What We Will Be Covering

- What are containers? What is Docker? (this section)
- Why use Containers and Docker? (this section)
- Deploying Docker Images

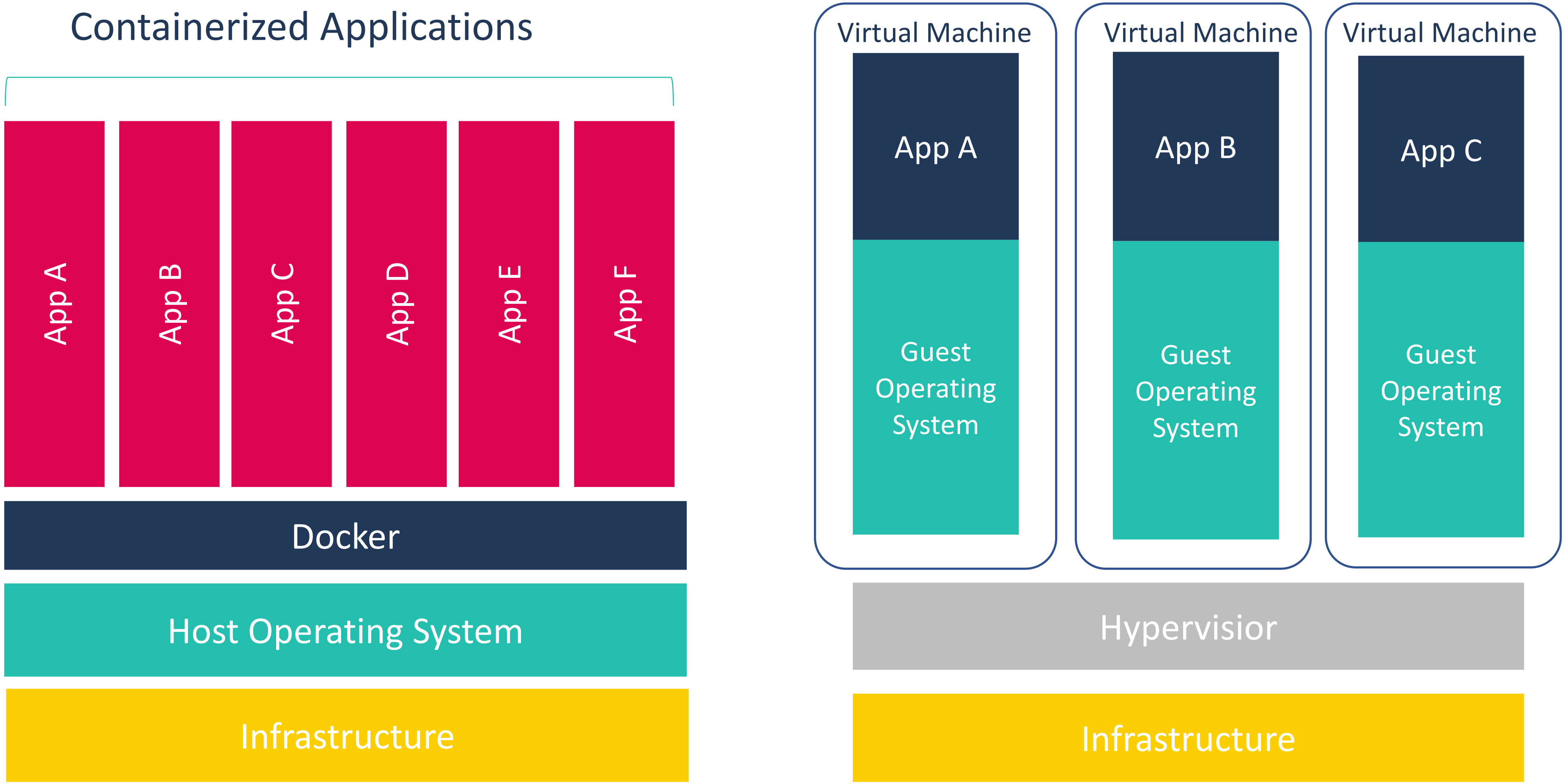
What is a Container?

- “A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.”

What is Docker ?

- Put simply, Docker is a tool to make creating, deploying and running containers easy.
- Docker is open source
- Released in 2013
- A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc.
- Containers are created from a read-only template called an image

Containers vs. Virtual Machines



Why Use Containers?

- Reproducibility
- Isolation
- Simplicity of environment management (Great for making staging/UAT match production)
- Ease of continuous integration
- Much faster and more lightweight than a VM
- Container orchestration options (e.g. Kubernetes)
- Docker is the most popular tool for creating and running containers

Steps to deploy with Docker

- Building the machine learning model
  - you can export the model to reuse it with **pickle**

```
#Serializing our model to a file called model.pkl
import pickle
Pickle.dump(model , open (“model.pkl” , wb))
```

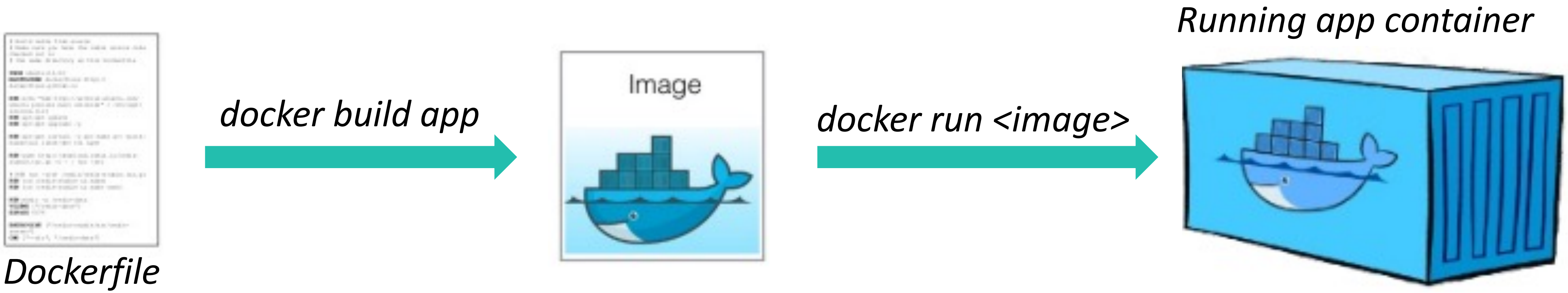
- Building the REST API using **Flask**
  - it will accept parameters that are required to predict targets using our trained model stored in pickle file

- Building Docker image
  - Dockerfile is a text file that defines a Docker image

- Building and run Docker container

Docker File

- The way to get our Python code running in a container is to pack it as a Docker image and then run a container based on it. The steps are sketched below.



Dockerfile

docker build app

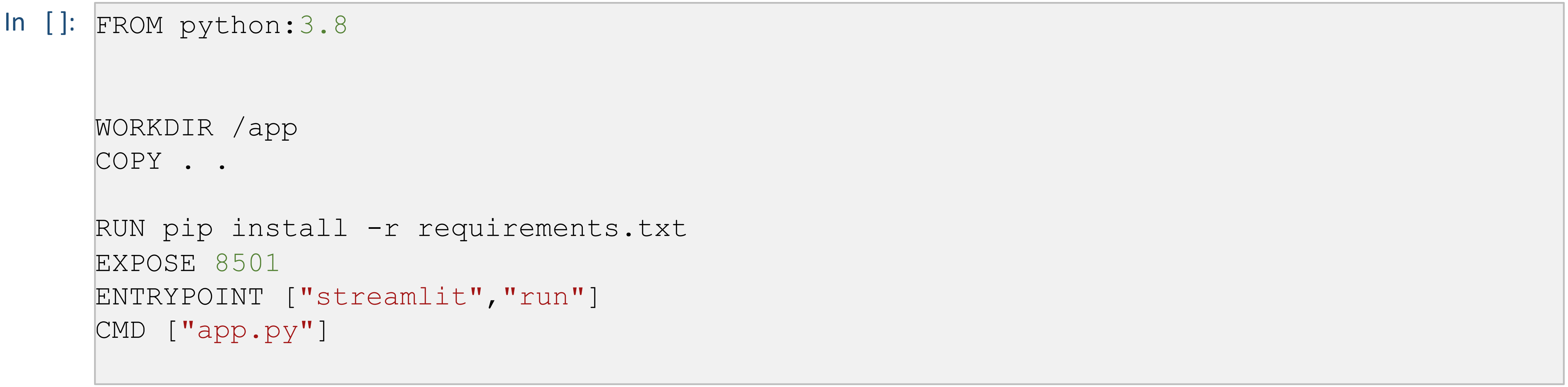
docker run <image>

Running app container

- To generate a Docker image, we need to create a Dockerfile which contains instructions needed to build the image. The Dockerfile is then processed by the Docker builder which generates the Docker image. Then, with a simple docker run command, we create and run a container with the Python service.

Creating Image

- An example of a Dockerfile containing instructions for assembling a Docker image:

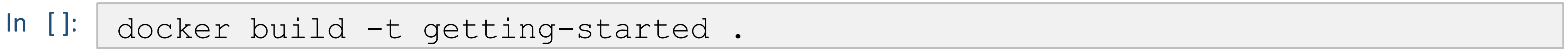


```
FROM python:3.8


WORKDIR /app
COPY . .

RUN pip install -r requirements.txt
EXPOSE 8501
ENTRYPOINT ["streamlit","run"]
CMD ["app.py"]
```

- For each instruction or command from the Dockerfile, the Docker builder generates an image layer and stacks it upon the previous ones. Therefore, the Docker image resulting from the process is simply a read-only stack of different layers.

Creating Image - CMD

- Open a terminal and go to the app directory with the Dockerfile. Now build the container image using the **docker build** command.

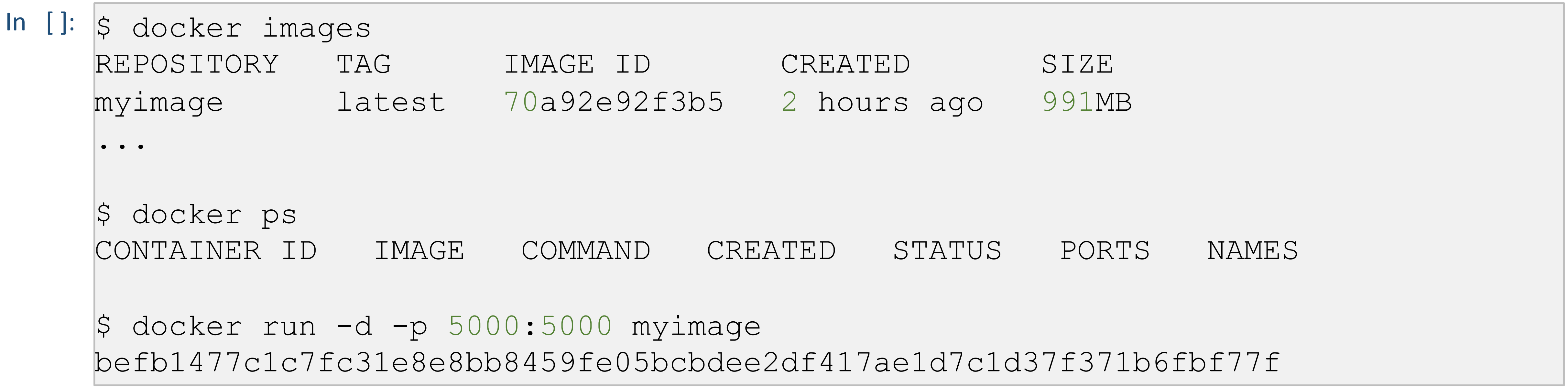


In [ ]:
```
docker build -t getting-started .
```

- The **-t** flag tags our image. Think of this simply as a human-readable name for the final image. Since we named the image *getting-started*, we can refer to that image when we run a container.
- The "." at the end of the docker build command tells that Docker should look for the Dockerfile in the current directory.

Run the Container - CMD

- After writing the Dockerfile and building the image from it, we can run the container with our Python service.



```
$ docker images
REPOSITORY   TAG      IMAGE ID       CREATED        SIZE
myimage      latest   70a92e92f3b5   2 hours ago    991MB
...

$ docker ps
CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES

$ docker run -d -p 5000:5000 myimage
befb1477c1c7fc31e8e8bb8459fe05bcbdee2df417ae1d7c1d37f371b6fbf77f
```

Save the Image- CMD

- Save one or more images to a tar archive

```
In [ ]: $ docker save [OPTIONS] IMAGE [IMAGE...]
```

- Load an image from a tar archive or STDIN

```
In [ ]: $ docker load [OPTIONS]
```

- Export a container’s filesystem as a tar archive

```
In [ ]: $ docker export [OPTIONS] CONTAINER
```

- Push the Image (Docker Hub)

```
In [ ]:  docker push YOUR-USER-NAME/getting-started
```

Thank you!

Data Science Academy