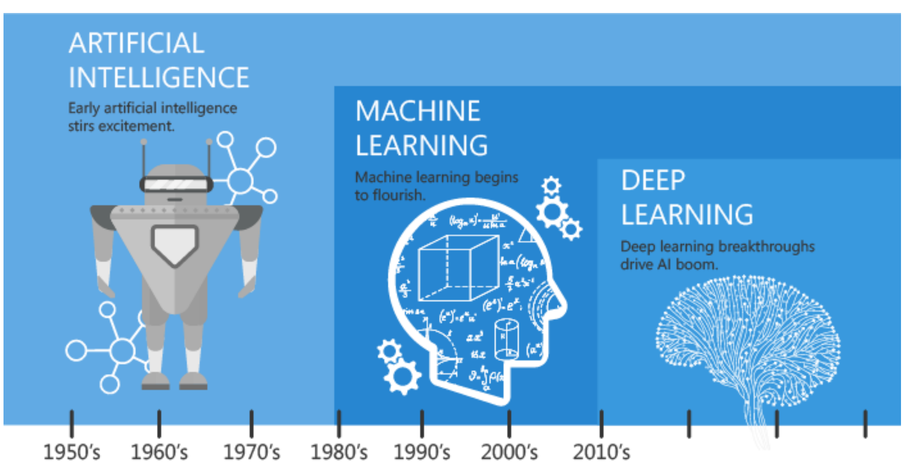
Challenges of Machine Learning

Data Bootcamp
BEST DATA TRAINING

1

History of AI



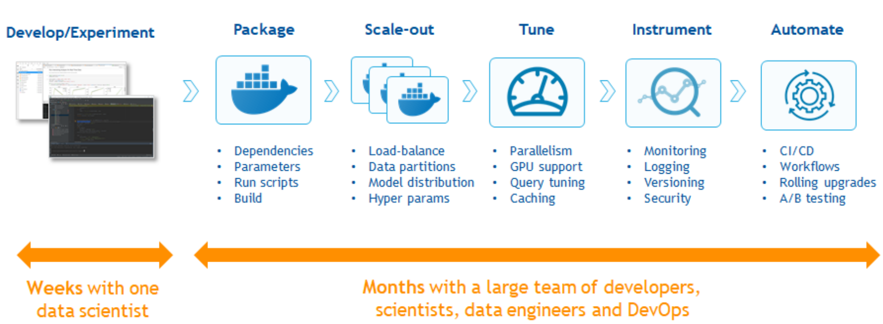
Data Bootcamp
BEST DATA TRAINING

2

3

Challenges

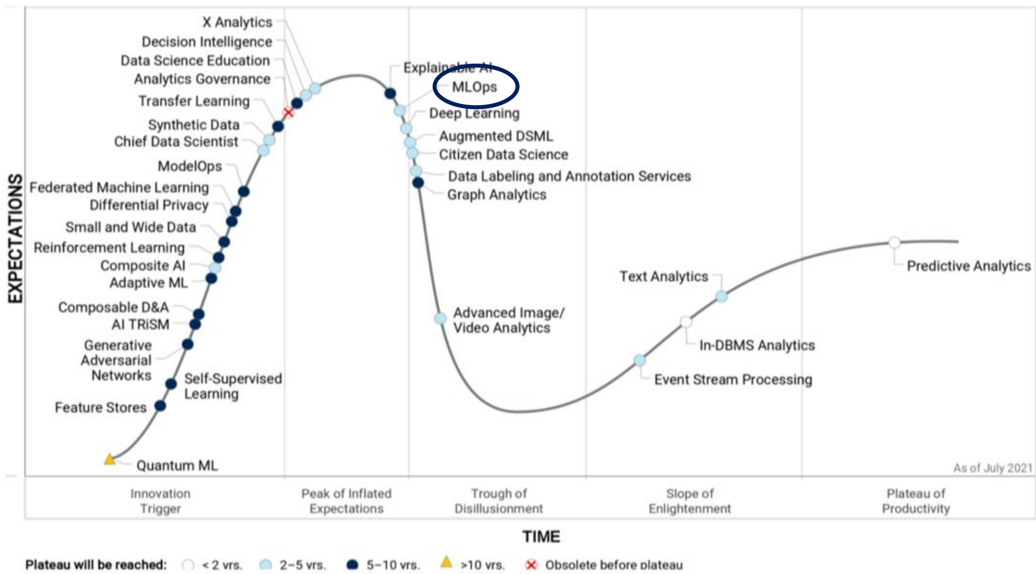
According to a survey, **55% of companies** have never deploy a model. Main **reasons**: lack of talent, lack of processes to manage change and lack of automated systems.



Data Bootcamp
BEST DATA TRAINING

4

Trends

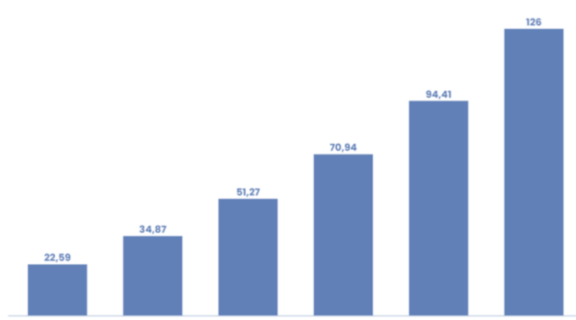


5

Benefits

Those organizations that put **AI into production** saw their profit margin increase from **3% to 15%.**

The MLOps market was estimated at **$23.2 billion** in 2019. It is projected to reach **$126 billion** by 2025 due to rapid adoption



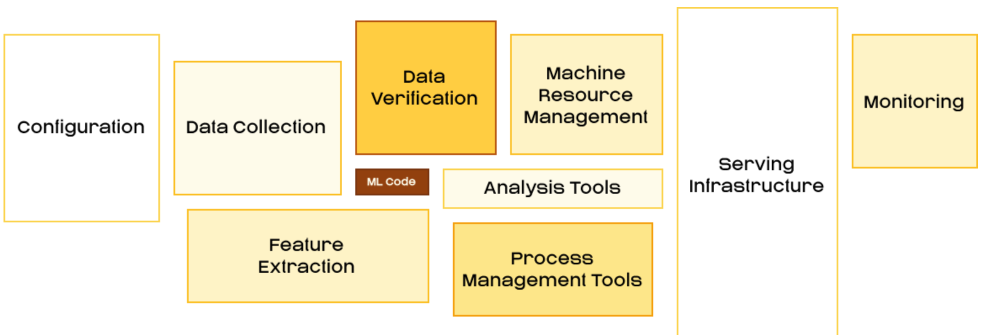
AI software market revenue from 2020 to 2025 [billions of dollars]

Data Bootcamp
BEST DATA TRAINING

6

What is MLOps?

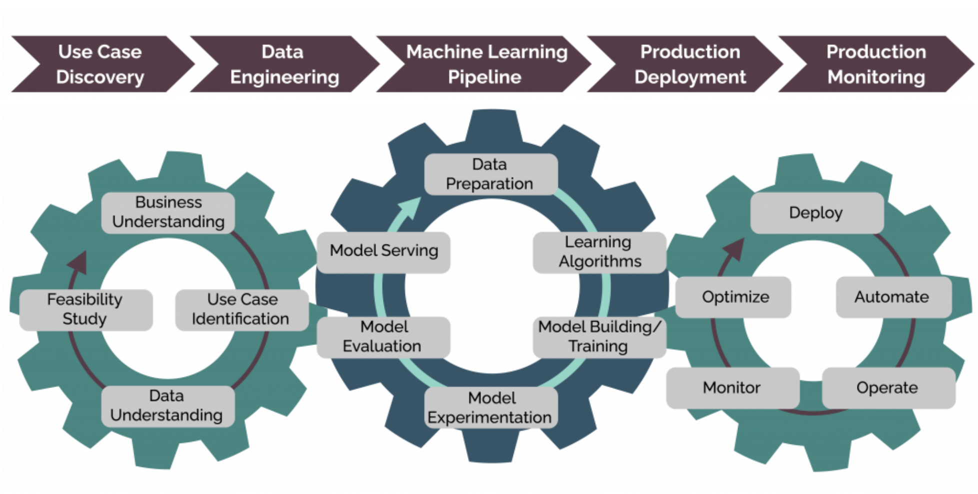
Model creation must be **scalable, collaborative and reproducible**. The principles, tools and techniques that make models scalable, collaborative and reproducible are known as **MLOps.**



Data Bootcamp
BEST DATA TRAINING

7

MLOps Process

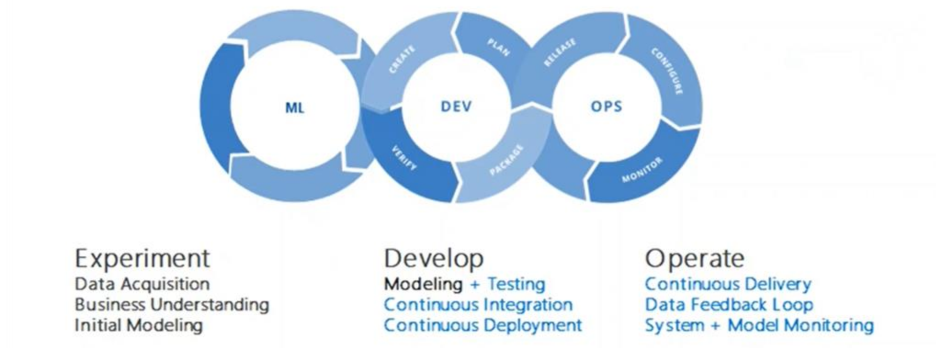


Data Bootcamp
BEST DATA TRAINING

8

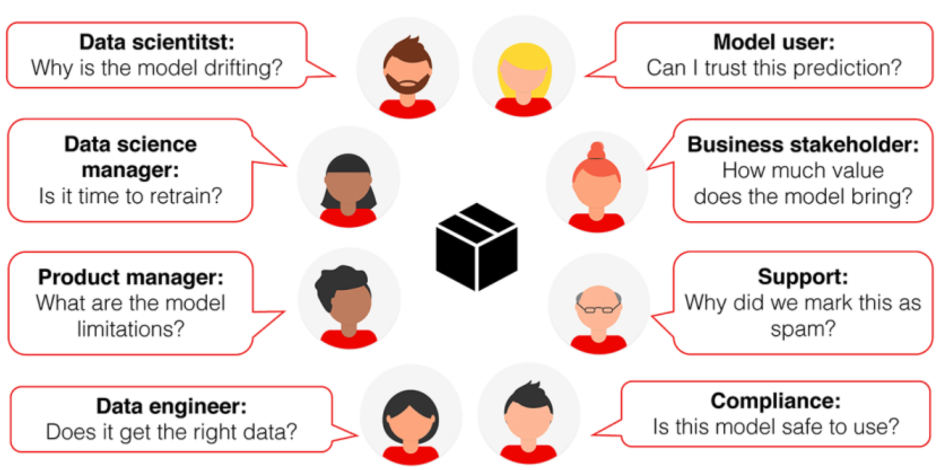
DevOps & DataOps

DevOps applied to Machine Learning is known as MLOps. **DataOps** implies a set of rules that ensure a high quality of data to train models.



9

Roles in MLOps



10

MLOps Fundamentals

Data Bootcamp
BEST DATA TRAINING

11

Chanllenges addressed by MLOps

Versioning
Tools such as Git and GitHub are used in code version control. Also, **data and artifacts** are versioned to ensure **reproducibility..**

Model tracking
Models in production can be **degraded** over time due to **data drift.**

Feature Generation
It requires a lot of resources. MLOps allows to **reuse functions**. So, you can focus on the design/test of the model.

Data Bootcamp
BEST DATA TRAINING

12

Parts of MLOps

Feature store
Stores the functions that has been used in model training

Data Versioning
Data version control ensures reproducibility and facilitates auditing.

Metadata store
It is critical for reproducibility. Everything should be registered, from model´ seed to evaluation metrics …

Model Versioning
Allows you to switch between models in real time or serve different models to monitor

Model Registration
Once a model has been trained, it is stored in a model registry with it´s metadata

Model serving
Serving a model means creating endpoints that can be used to run predictions

Data Bootcamp
BEST DATA TRAINING

13

Parts de MLOps

Model Monitoring
Models should be monitored for deviation and production bias

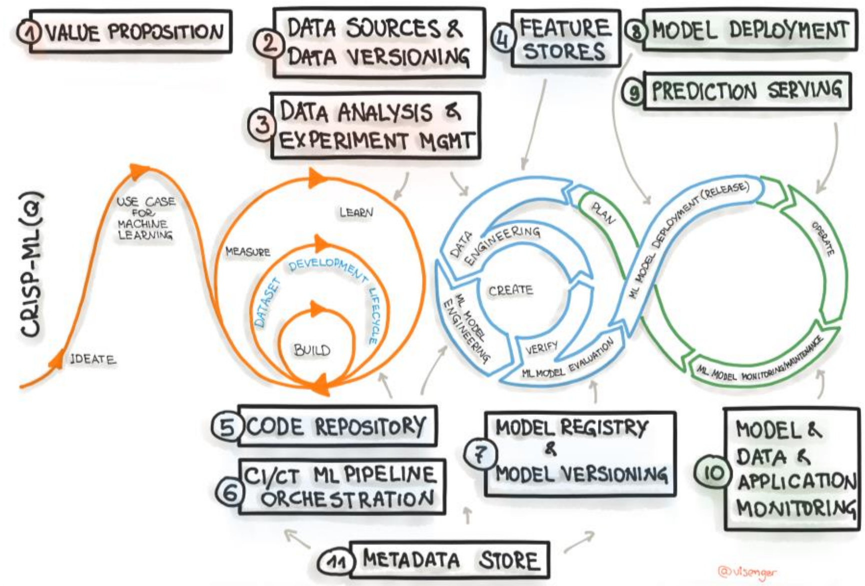
Recycling of models
Models can be retrained to improve performance or when there is new data

CI/CD
This ensures that code is frequently merged with automated process and tests.

Data Bootcamp
BEST DATA TRAINING

14

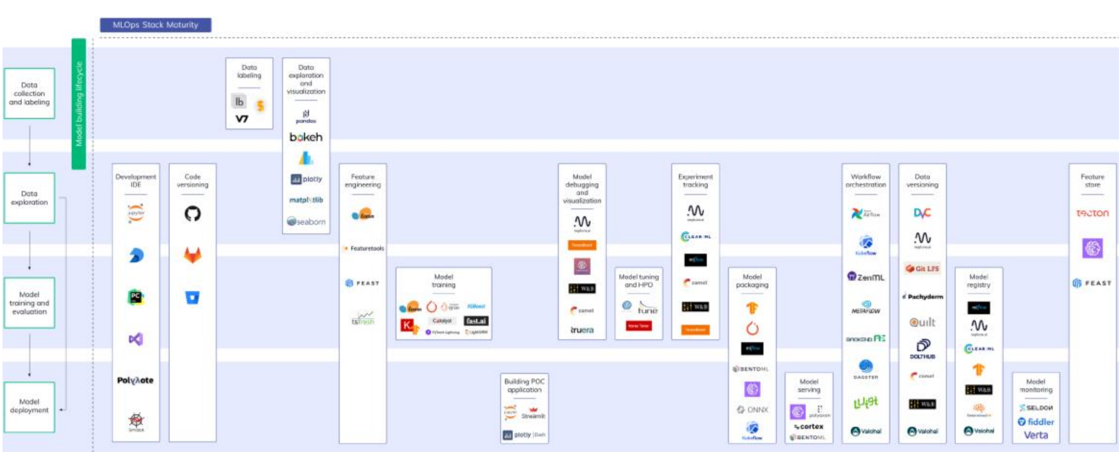
Components of MLOps



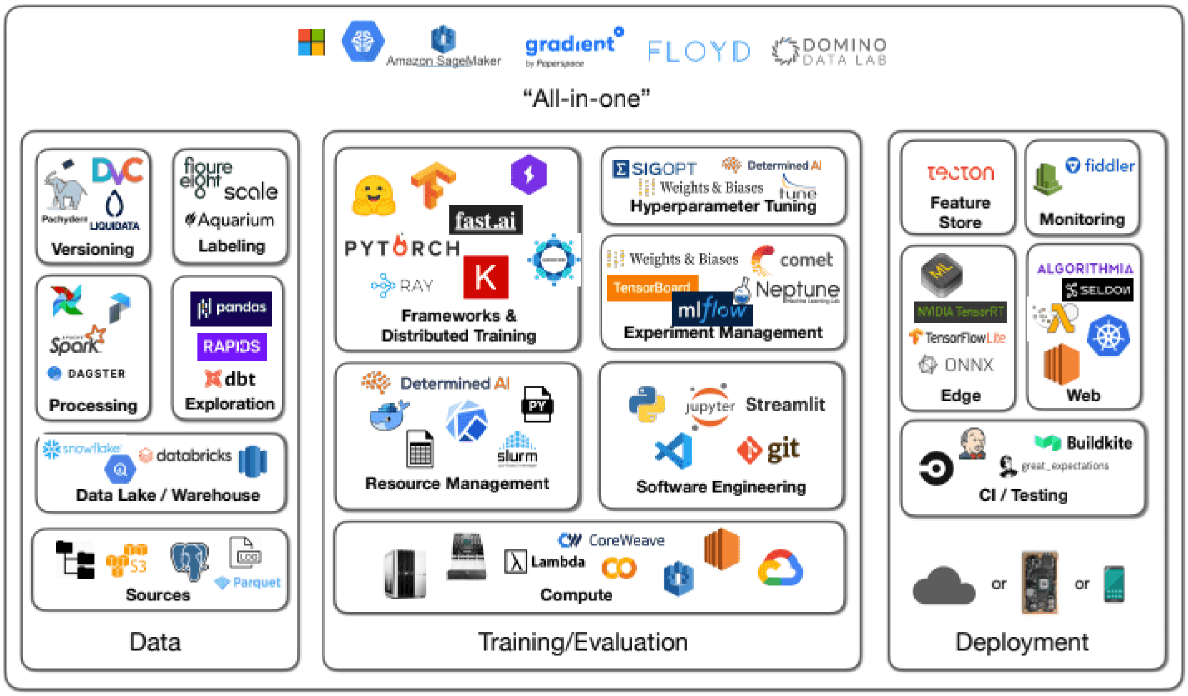
15

MLOps Tools

https://neptune.ai/blog/machine-learning-model-management#&gid=1&pid=1



16



17

MLOps Stages

Data Bootcamp
BEST DATA TRAINING

18

MLOps stages

Stage 1: Model and data **Version Control**

Stage 2: AutoML + Model and Data Version Control

Stage 3: AutoML + Model and Data Version Control + **Model Serving**

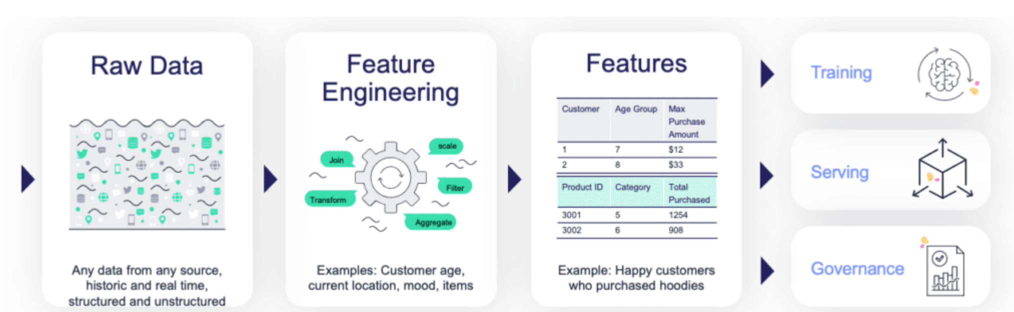
Stage 4: AutoM + Model and Data Version Control + Model Serving **+ Monitoring, Governance and Retraining**

Data Bootcamp
BEST DATA TRAINING

19

Stage 1: Data Collection and Preparation

There is no ML without data. ML teams need access to historical and/or online data from multiple sources. They must catalog and organize this data. Raw data cannot be used, they need to process this data.



Data Bootcamp
BEST DATA TRAINING

20

Data collection and preparation with MLOps

MLOps solutions must incorporate a **Feature Storage** that:

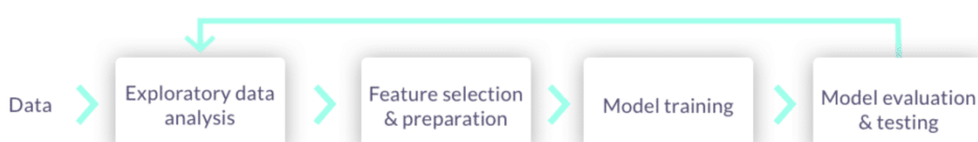
1. Define data collection and transformations **only once** for batch and streaming scenarios
2. Process functions automatically **without manual intervention**
3. Serve functions from a **shared catalog** for training, service and government applications

Data Bootcamp
BEST DATA TRAINING

21

MLOps Stage 2: Automated Development

Model development generally follows the **same process**. Much of it can be **automated** thanks to **AutoML and MLOps**



All runs, along with their data, metadata, code, and results, must be **versioned and logged**

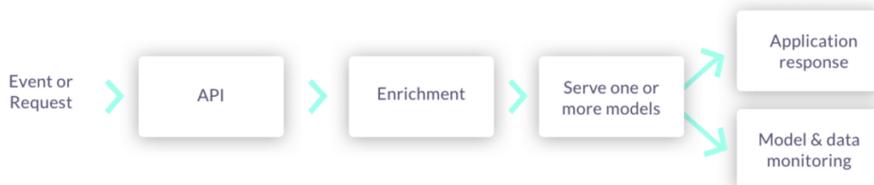
Data Bootcamp
BEST DATA TRAINING

22

MLOps Stage 3: Create ML Services

Once a model has been created, it must **be integrated** with the **business application** or **front-end** services. They must be implemented without interrupting service. Production pipelines implement:

- Real-time data collection, data validation, and feature engineering
- **API services** or application integration
- Data and **model monitoring** services
- **Resource monitoring** and alert services
- Telemetry and event logging services

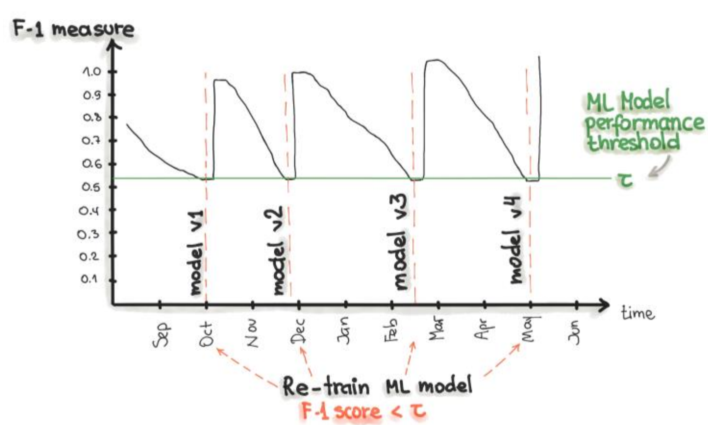


Data Bootcamp
BEST DATA TRAINING

23

MLOps Stage 4: Monitoring, Governance and Retraining

Model monitoring is a core component of MLOps to **keep models up-to-date** and predicting with maximum accuracy. It guarantees the validity of the model in the long term.



Data Bootcamp
BEST DATA TRAINING

24

Installations

Data Bootcamp
BEST DATA TRAINING

25

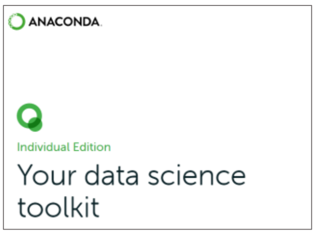
Tools to use

Data Bootcamp
BEST DATA TRAINING

26

Facility

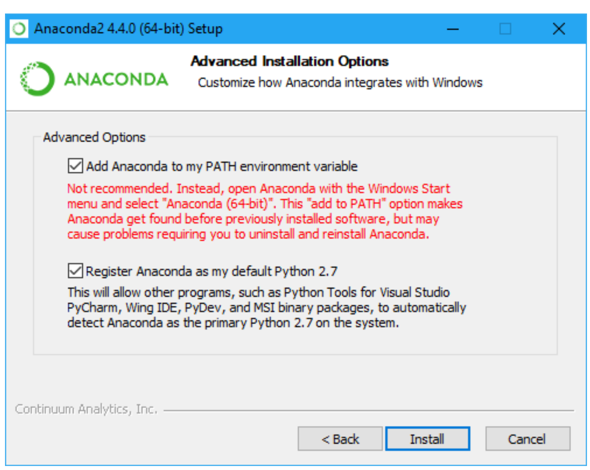
If this is your first-time using Python, you must install **Anaconda Distribution** with **Python 3.7** or higher. Link: https://www.anaconda.com/products/individual



Data Bootcamp
BEST DATA TRAINING

27

Environment variable



Data Bootcamp
BEST DATA TRAINING

28

Set the environment

Step 1. Create a virtual environment
Open the Anaconda Prompt from the start menu and run the following code: *conda create --name mlops python=3.7*

Step 2 Activate the environment
Execute the command: *conda activate mlops*

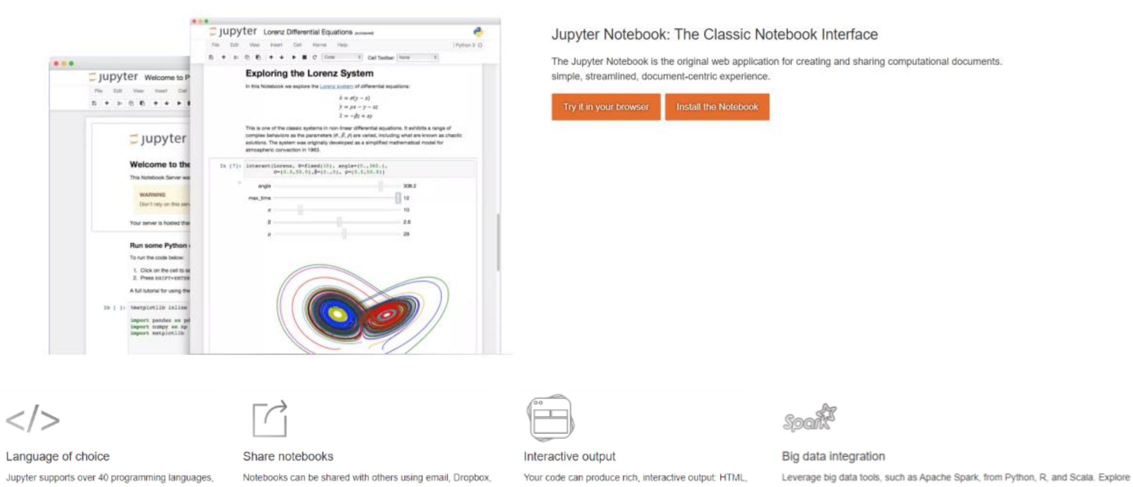
Step 3. Install the necessary libraries
From the Anaconda prompt write the following code, with the name of the library you want to install: *pip install [library—name]*   Example with Pycaret: *pip install pycaret==2.3.5*
if you have requirements file: *pip install -r requirements.txt*

Step 4. Jupyter Notebook
From the correct environment (ml_pycaret) launch jupyter notebook with command *jupyter notebook*

29

Jupyter Notebbok

Jupyter Notebook is a visual IDE for creating and sharing documents with code in different programing languages as: Python, R, Scala, etc. It offers a **simple**, streamlined, document-centric experience.



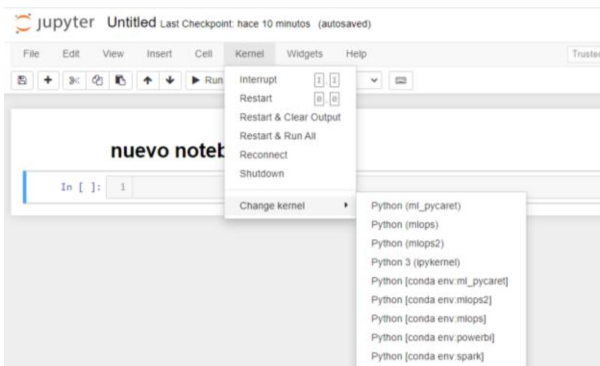
30

Jupyter Notebok Kernel

In some versions, to change enviroment in Jupyter Notebook we must install some aditional libreries. The commands are:

```
conda install -n python_env ipykernel
python -m ipykernel install --user --name mlops--display-name "Python (mlops)”
```

More information at: https://stackoverflow.com/questions/39604271/conda-environments-not-showing-up-in-jupyter-notebook



Data Bootcamp
BEST DATA TRAINING

31

Docker

In order to install Docker, we must follow those following steps:

1. Download **Docker Desktop** and install https://docs.docker.com/desktop/windows/install/
2. Install **Windows Subsystem for Linux** (Step 4) https://docs.microsoft.com/en-us/windows/wsl/install-manual
3. Step 5 from Powershell
4. Install **Ubuntu**

docker

Data Bootcamp
BEST DATA TRAINING

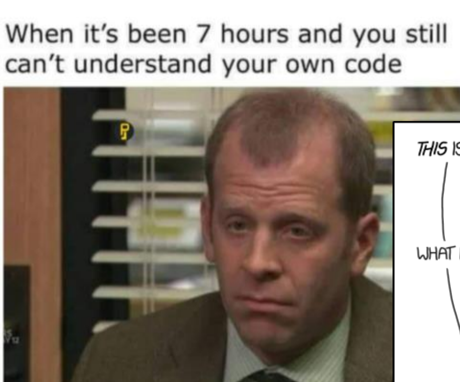
32

Structuring ML projects

Data Bootcamp
BEST DATA TRAINING

33

The importance of organizing the project

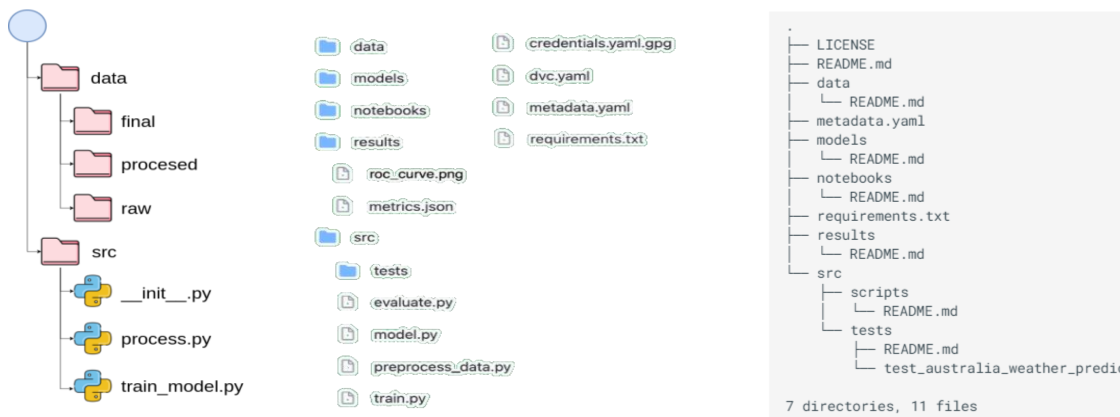


Data Bootcamp
BEST DATA TRAINING

34

Structure ML projects

It is important to **structure** the project according to a **standard**. But what kind of standard should you follow?



35

Cookiecutter

Cookiecutter is a tool for creating **projects folder structure** automatically **using templates**. You can create static file and folder structures based on input information.

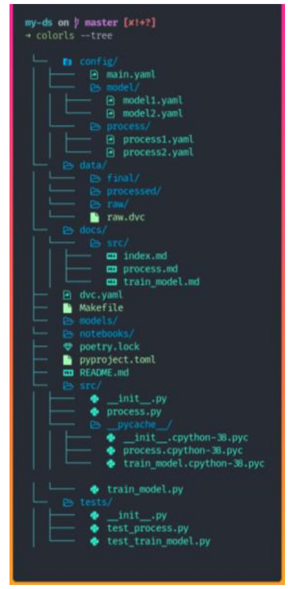
pip install cookiecutter

cookiecutter https://github.com/khuyentran1401/data-science-template

36

ML Tools

- **Poetry:** Dependency Management
- **Hydra:** To manage configuration files
- **Pre-commit plugins**: Automate code review and formatting
- **DVC**: Data Version Control
- **pdoc**: automatically create documentation for your project



37

Poetry

An **alternative** to installing libraries with **pip** is using Poetry. **Poetry** allows to:

- Separate main dependencies and sub dependencies into two separate files (vs requirements.txt)
- Create readable dependency files
- Remove all unused sub-dependencies when removing a library
- Avoid installing new libraries in conflict with existing libraries
- Package the project with few lines of code

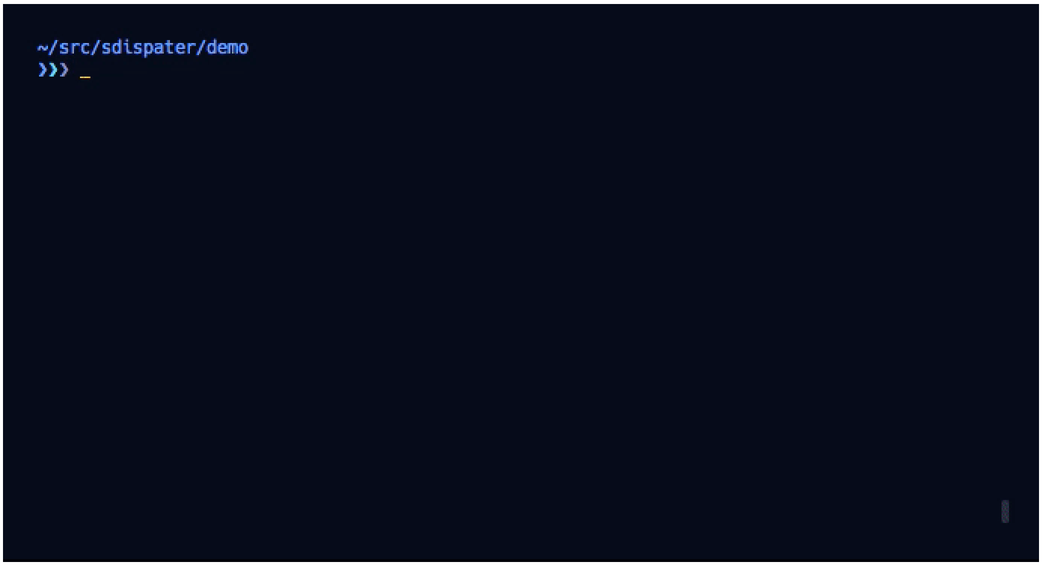
All the dependencies of the project are specified in **pyproject.toml.**

Generate project	Install dependencies	To add a new PyPI library	To delete a library
poetry new <proyect-name>	poetry install	poetry add <library-name>	poetry remove <library-name>

Data Bootcamp
BEST DATA TRAINING

38

Poetry



39

Makefile

Makefile creates short and readable commands for configuration tasks. You can use Makefile to **automate tasks** such as setting up the environment.

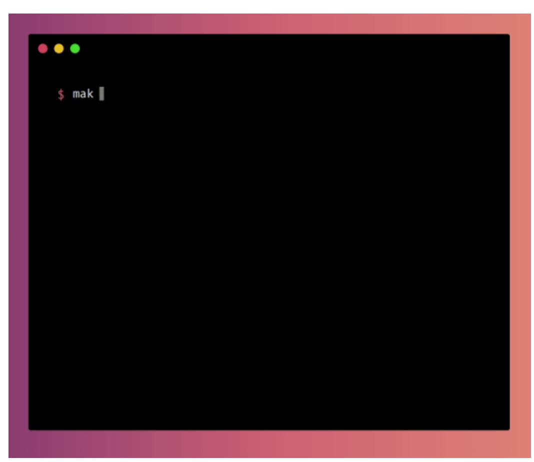
1

```
install:
	@echo "Installing..."
	poetry install
	poetry run pre-commit install

activate:
	@echo "Activating virtual environment"
	poetry shell

initialize_git:
  @echo "Initialize git"
	git init

setup: initialize_git install
```

2

```
make activate
make setup
```

3



40

Hard-coding

In data science is common to **execute different configurations and models**, so configuration **should not be hardcoded**. For example, if we want to modify the input variables of model, it will take time to change them.

```
columns = ['iid', 'id', 'idg', 'wave', 'career']
df.drop(columns, axis=1, inplace=True)
```

Wouldn't it be better to set the columns in **a config file**?

```
variables:
  drop_features: ['iid','id','idg','wave','position','positin1', 'pid',  'field', 'from', 'career'

  # categorical variables to transform to numerical variables
  numerical_vars_from_numerical: ['income','mn_sat', 'tuition']

  # categorical variables to encode
  categorical_vars: ['undergra', 'zipcode']
  categorical_label_extraction: ['zipcode']
  categorical_onehot: ['undergra']
```


41

Configuration file

A **configuration file** contains **parameters** that define the configuration of the program. It is good practice to avoid hard coding in Python scripts. **YAML** is a common language for a configuration file.

```
# get current path
current_path = utils.get_original_cwd() + "/"

# read training data
data = pd.read_csv(current_path + config.dataset.data,  encoding=config.dataset.encoding)

# divide train and test
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(config.target.target, axis=1),
    data[config.target.target],
    test_size=0.1,
    random_state=0)
```

Config.yaml

```
# data
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

pipeline:
  pipeline01: decisiontree

target: match
```

config.yaml hosted with ❤ by GitHub

42

Hydra

There are some tools to manage a configuration file such as PyYaml or **Hydra**. Why Hydra?

- Change parameters in the **terminal**
- Switch between **setting groups**
- Automatically **record** results

```yaml
# data
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

pipeline:
  pipeline01: decisiontree

target: match
```

```python
import hydra
from hydra import utils
import pandas as pd

@hydra.main(config_path='preprocessing.yaml')
def run_training(config):
    """Train the model."""

    # Get current path
    current_path = utils.get_original_cwd() + "/"

    # read training data
    data = pd.read_csv(current_path + config.dataset.data,  encoding=config.dataset.encoding)

    # divide train and test
    X_train, X_test, y_train, y_test = train_test_split(
        data.drop(config.target, axis=1),
        data[config.target],
        test_size=0.1,
        random_state=0)

if __name__=='__main__':
    run_training()
```

43

Hydra uses

1. **Modify the parameters from the terminal** without the need to modify the config file
2. Switch between different **configuration groups**
3. **Logging**

```yaml
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

model: decisiontree

```

model.yaml hosted with ❤ by GitHub

```
python file.py model=logisticregression
```

Run a logistic regression

```
├── configs
│   └── model
│       ├── decisiontree.yaml
│       └── logistic.yaml
├── data
│   └── raw.csv
```

```yaml
hyperparamters:
  penalty: l1
  dual: False
  C: 1
```

logistic.yaml hosted with

```
python file.py model=logistic
```

```
outputs
> 2020-04-26
> 2020-04-27
```

```
outputs
  2020-04-26
    10-56-35
      .hydra
        config.yaml
        hydra.yaml
        overrides.yaml
      pipeline.log
      train_pipeline.log
```

Data Bootcamp
BEST DATA TRAINING

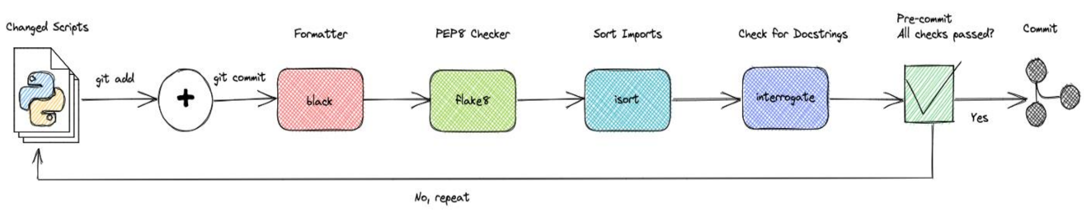
44

Check code before commit

When committing Python code to Git, you must ensure that your code:

- It is correct
- It is organized
- Conforms to PEP 8 style guide
- Includes documentation (docstrings)

Different **plugins** can be added to **pre-commit** for automagical code **review.** Those plugins will review the code and will **correct it.**



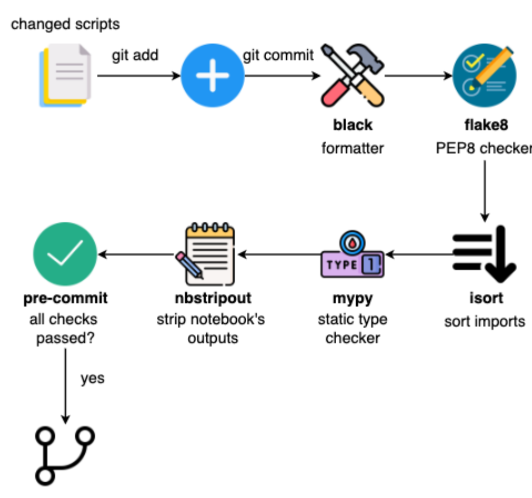
Data Bootcamp
BEST DATA TRAINING

45

Commit plugins

In this example we will use the following plugins. Those plugins are specified in .pre-commit-config.yaml.

- **Black:** Format Python code
- **Flake8**: Check the style and quality of Python code
- **Isort:** Sort alphabetically imported libraries and separate them into types
- **Interrogate:** checks the code for missing docstrings



Data Bootcamp
BEST DATA TRAINING

46

Commit plugins in action

black

```
def very_long_function(long_variable_name, long_variable_name2, long_variable_name3, long_variable_
    pass
```

```
def very_long_function(
    long_variable_name,
    long_variable_name2,
    long_variable_name3,
    long_variable_name4,
    long_variable_name5,
):
    pass
```

flake8

```
def very_long_function_name(var1, var2, var3,
var4, var5):
    print(var1, var2, var3, var4, var5)

very_long_function_name(1, 2, 3, 4, 5)
```

```
flake8_example.py:2:1: E128 continuation line under-indented for visual indent
flake8_example.py:5:1: E305 expected 2 blank lines after class or function definition, found 1
flake8_example.py:5:39: W292 no newline at end of file
```

```
def very_long_function_name(var1, var2, var3, var4, var5):
    print(var1, var2, var3, var4, var5)


very_long_function_name(1, 2, 3, 4, 5)
```

isort

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from flake8_example import very_long_function_name
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, OrderedLogisticRegression, \
    LinearRegression, LogisticRegressionCV, LinearRegressionCV
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from flake8_example import very_long_function_name
from sklearn.linear_model import (
    LinearRegression,
    LinearRegressionCV,
    LogisticRegression,
    LogisticRegressionCV,
    OrderedLogisticRegression,
)
from sklearn.model_selection import train_test_split
```

47

Add documentation

As a data scientist, we will be **collaborating** a lot with other team members. Therefore, it is important to create a **good documentation** for the project. To create API documentation based on the **dockstrings** we can use Makefile.

```
make docs_view
```

```
Save the output to docs...
pdoc src --http localhost:8080
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
```

```
make docs_save
```

All packages

Package **src**

Source code of your project

▸ EXPAND SOURCE CODE

Sub-modules

`src.process`

This is the demo code that uses hydra to access the parameters in under the directory config …

`src.train_model`

This is the demo code that uses hy …

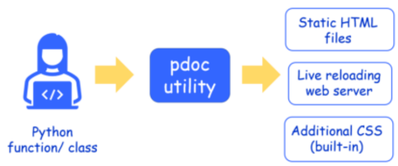
Source of : https://github.com/khuyentran1401/data-science-template/

Data Bootcamp
BEST DATA TRAINING

48

Pdoc3 to create Python documentation

It would be great to **generate beautiful web documentation** directly from functions docstrings without writing a single line of HTML/CSS code.



dockstrings

```
def addition(num1, num2=0):
    """
    Adds two numbers

    Args:

    num1: The first number

    num2: The second number, default 0

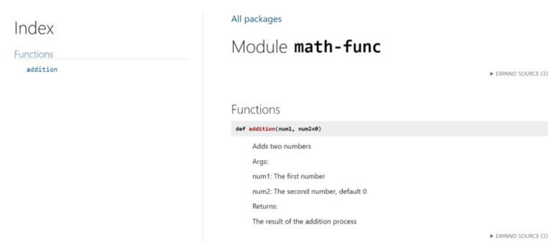
    Returns:

    The result of the addition process
    """

    return (num1+num2)
```

```
pip install pdoc3
pdoc --http localhost:8080 math-func.py
```

```
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
```



49

Markdown: the key to usability

The great thing about pdoc is that it allows seamless integration of **Markdown** text within the docstring.

```
"""

...

Handles exception by a check,

    ```python
    if num2 != 0:
        return (num1/num2)
    else:
        raise ValueError('The second argument cannot be zero')
    ```
 """
```

`def divide(num1, num2=1)`

Divides the first number by the second

Args:

`num1` : The first number

`num2` : The second number, default `1`

Returns:

The result of the **division** process

Exception:

Handles exception by a check,

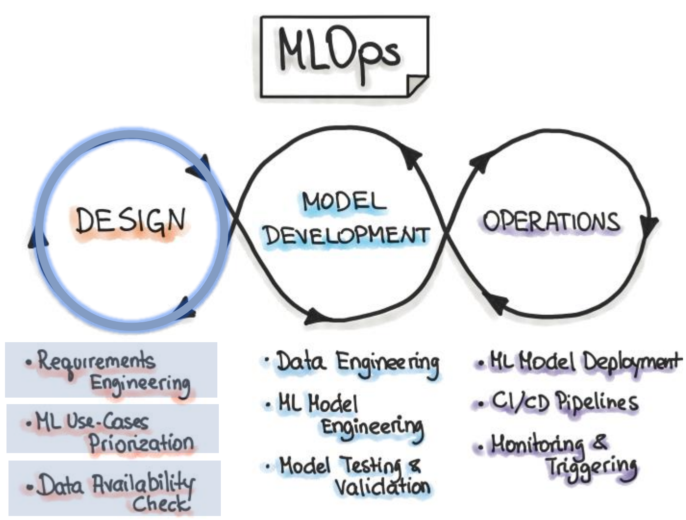
```python
if num2 != 0:
    return (num1/num2)
else:
    raise ValueError('The second argument cannot be zero')
```

Data Bootcamp
BEST DATA TRAINING

50

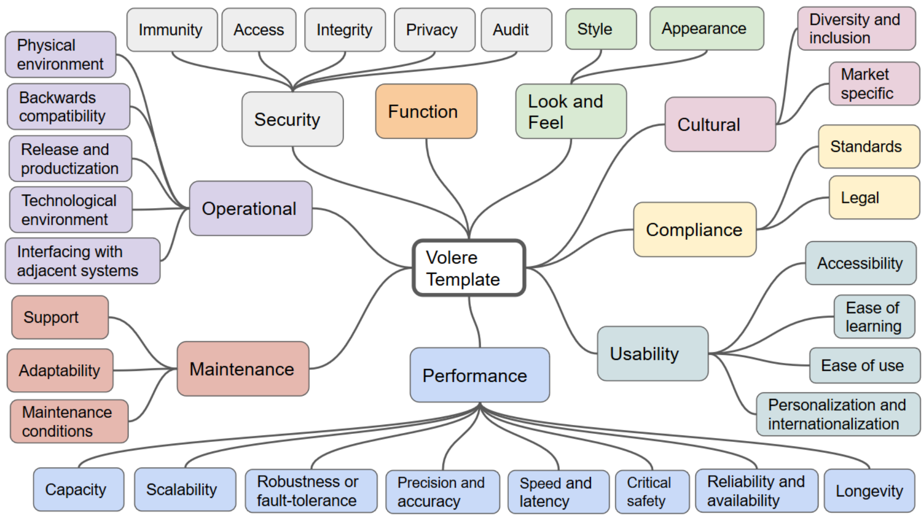
ML product design

MLOps stages



Tools

Tool to complete the design phase of the model https://github.com/ttzt/catalog_of_requirements_for_ai_products



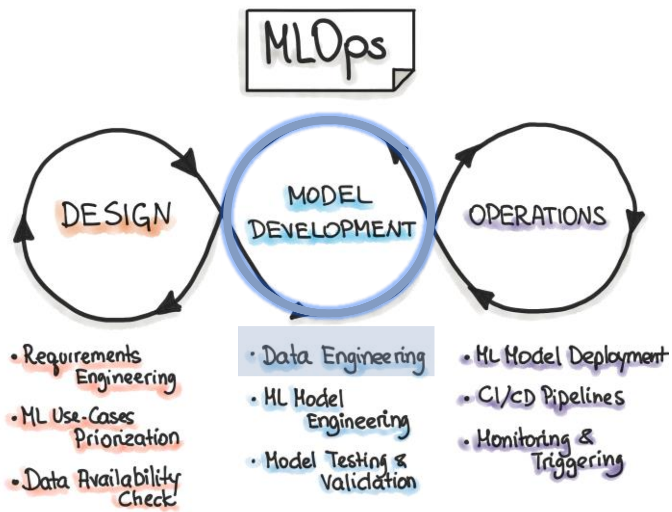
53

Feature Store

Data Bootcamp
BEST DATA TRAINING

54

MLOps stages



Data Bootcamp BEST DATA TRAINING

55

Reference tools

Feature Store is a data management layer for machine learning that allows you to share features and build more efficient **machine learning** pipelines.

Platform	Open Source	Offline	Online	Real Time Ingestion	Feature Ingestion API	Write Amplification	Supported Platforms	Training API	Training Data
Hopsworks	AGPL-V3	Hudi/Hive and pluggable	RonDB	Flink, Spark Streaming	(Py)Spark, Python, SQL, Flink	No	AWS, GCP, On-Prem	Spark	DataFrame (Spark or Pandas), files (.csv, .tfrecord, etc)
Michelangelo	No	Hive	Cassandra	Flink, Spark Streaming	Spark, DSL	None	Proprietary	Spark	DataFrame (Pandas)
Zipline	No	Hive	Unknown KV Store	Flink	DSL	None	Proprietary	Spark	Streamed to models?
Twitter	No	GCS	Manhatten, Cockroach	Unknown	Python, BigQuery	Yes. Ingestion Jobs	Proprietary	BigQuery	DataFrame (Pandas)
Iguazio	No	Parquet	V3IO, proprietary DB	Nuclio	Spark, Python, Nuclio	Unknown	AWS, Azure, GCP, on-prem	No details	DataFrame (Pandas)
Databricks	No	Delta Lake	Mysql or Aurora	None	Spark, SparkSQL	Unknown	Unknown	Spark	Spark Dataframes

Reference : https://www.featurestore.org/

Data Bootcamp BEST DATA TRAINING

56

DVC Studio

The **DVC Studio** interface allows us **to work with** data and also perform **experiments** from the web application:

- It helps us **manage data** and models,
- Allows you to run and track experiments
- Allows you to view and share results.
- It allows us to track our code, experiments, and data all the time.

Data Bootcamp
BEST DATA TRAINING

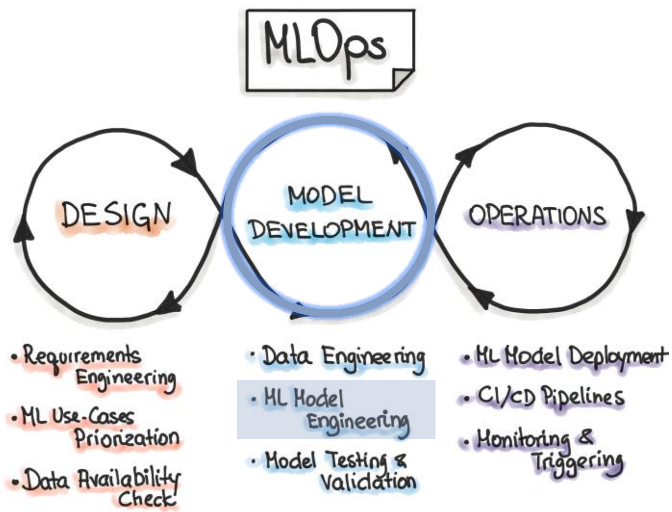
57

Automated model development

Data Bootcamp
BEST DATA TRAINING

58

MLOps stages



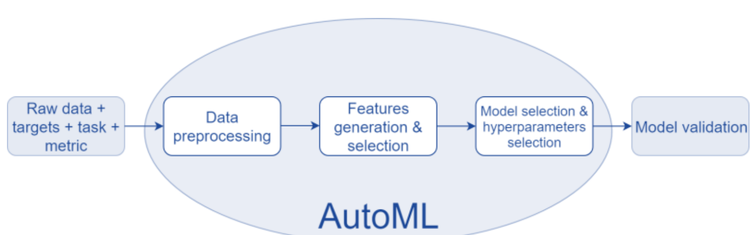
Data Bootcamp
BEST DATA TRAINING

59

AutoML

AutoML automates much of the model training process:

- AutoML helps to **Preprocess** the data
- AutoML generates new **variables** and selects the most significant ones
- AutoML trains and selects best **model**
- AutoML adjusts the **hyperparameters** of the chosen model
- AutoML makes model **evaluation** easy
- AutoML helps in model **deployment**

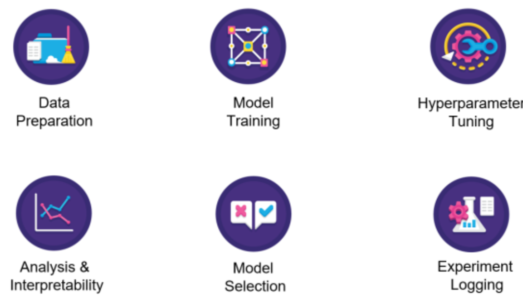


Data Bootcamp
BEST DATA TRAINING

60

Pycaret

PyCaret is an open source, low-code **machine learning** library. I has been developed in **Python** and reduce the time needed to create a model to minutes



Data Bootcamp
BEST DATA TRAINING

61

Built-in Pycaret modules

scikit learn
RAPIDS
RAY
mlflow
XGBoost
LightGBM
CatBoost
OPTUNA
Yellowbrick
plotly
GENSIM
spaCy

Data Bootcamp
BEST DATA TRAINING

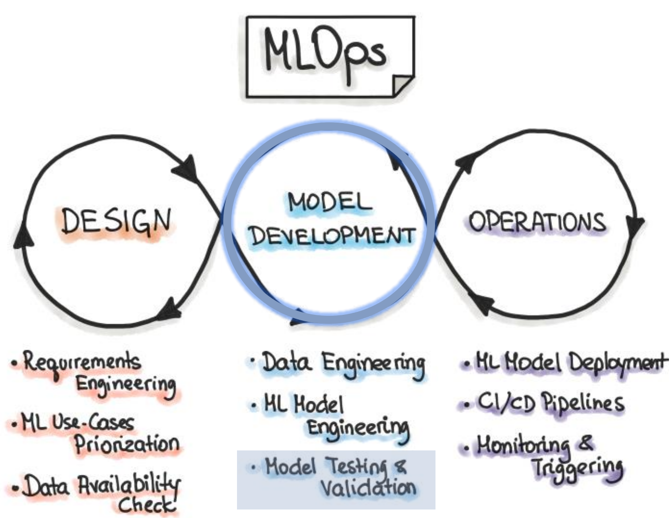
62

Model interpretability

Data Bootcamp
BEST DATA TRAINING

63

MLOps stages



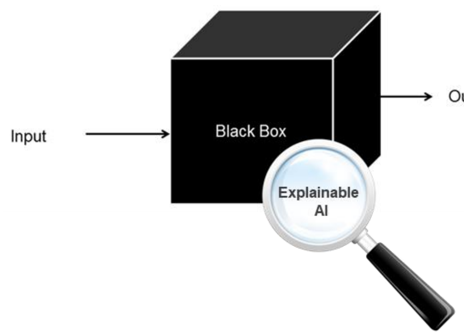
Data Bootcamp
BEST DATA TRAINING

64

Black box

ML models are commonly known as "**Black Box**", due to their **difficult interpretability**. This can be a **problem** in many sectors.

Usage example: One model predicts that a bank should not lend someone money, and the bank is legally required to explain the basis for each loan refusal.

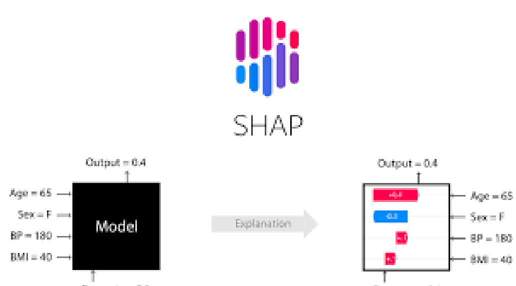


Data Bootcamp
BEST DATA TRAINING

65

SHAP

A widely used technique to understand the **impact** of a variable on the prediction of a model is **SHAP (SHapley Additive exPlanations).**

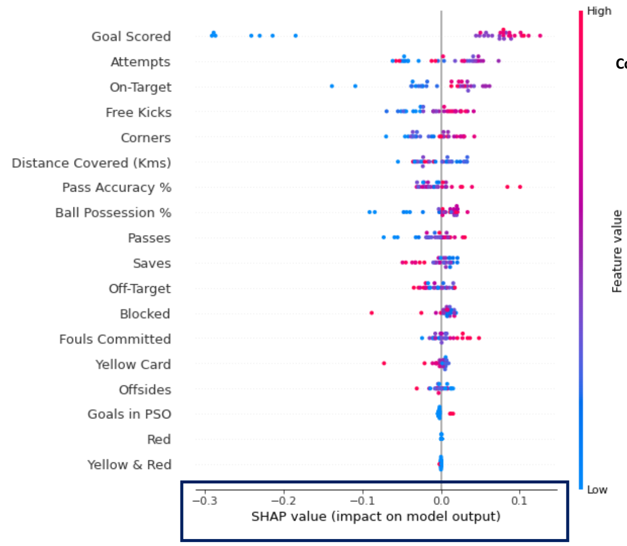


Data Bootcamp
BEST DATA TRAINING

66

SHAP Summary Plots

Verticality: Variables ordered by relevance



Color: Value of the variable. Pink for high values, blue for low

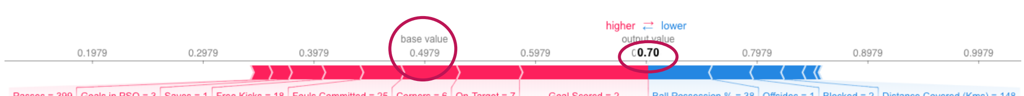
Horizontality: Effect in prediction. Left: less chance of winning, Right: more chance of winning

Data Bootcamp
BEST DATA TRAINING

67

SHAP Values

SHAP values interpret the **impact** of having a **certain value for a feature** compared to the prediction we would make if that feature took some reference value. Example: How much would a prediction change if the team scored 2 goals?



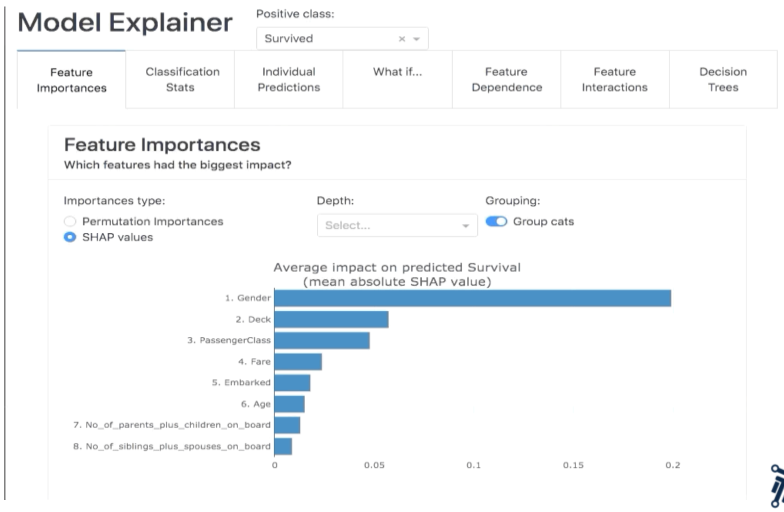
0.7 vs base value: 0.4979. Features that cause an increase in predictions in pink and features that decrease prediction in blue. The biggest impact of the goal scored is 2. Possession of the ball has a significant effect that decreases the prediction.

Data Bootcamp
BEST DATA TRAINING

68

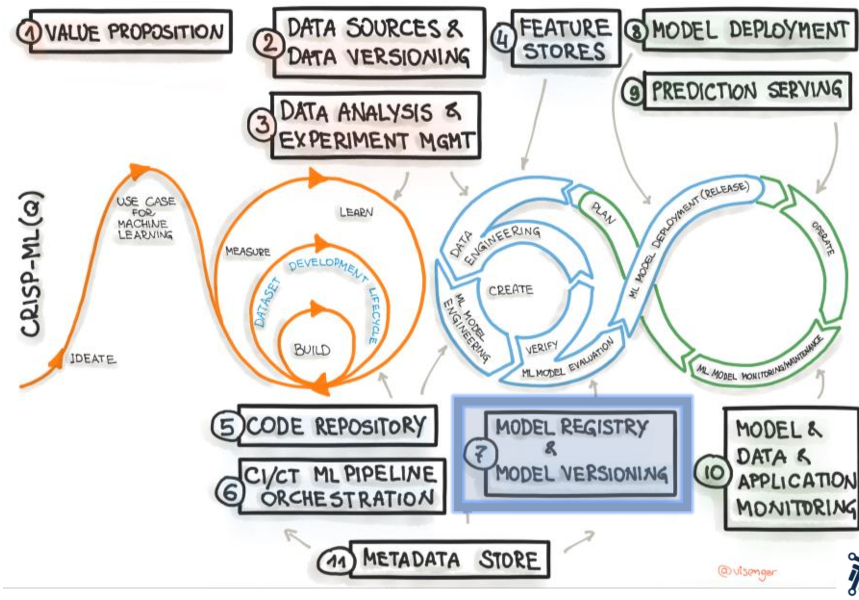
Explainer Dashboard

The **Explainer dashboard** is a library for quickly creating interactive dashboards to analyze and explain the predictions and performance of machine learning models.



Model registration and versioning

MLOps stages

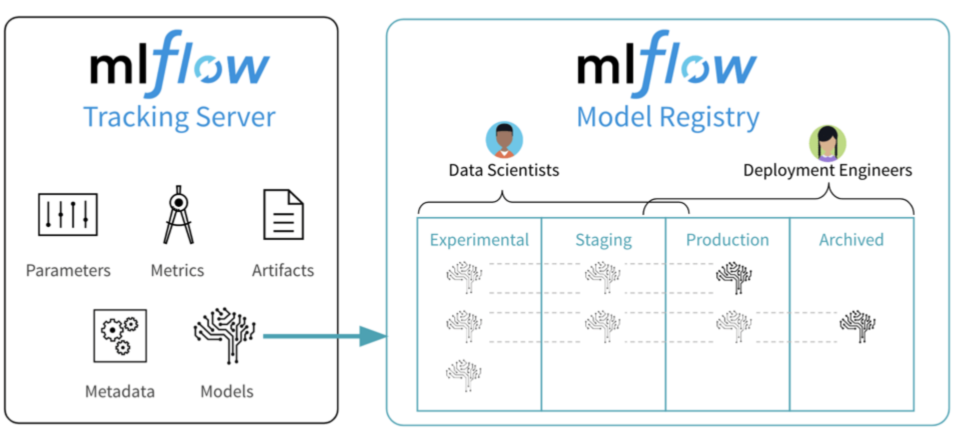


Data Bootcamp
BEST DATA TRAINING

71

Tools for MLOps

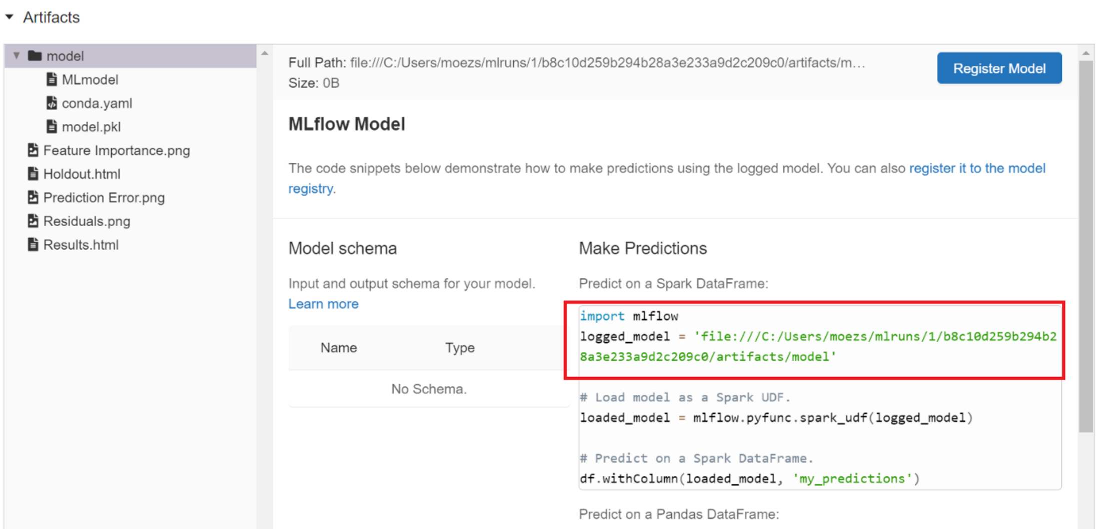
MLflow is an open source platform for managing the ML lifecycle, including model experimentation, reproducibility, deployment, and registration.



Data Bootcamp
BEST DATA TRAINING

72

MLFow platform

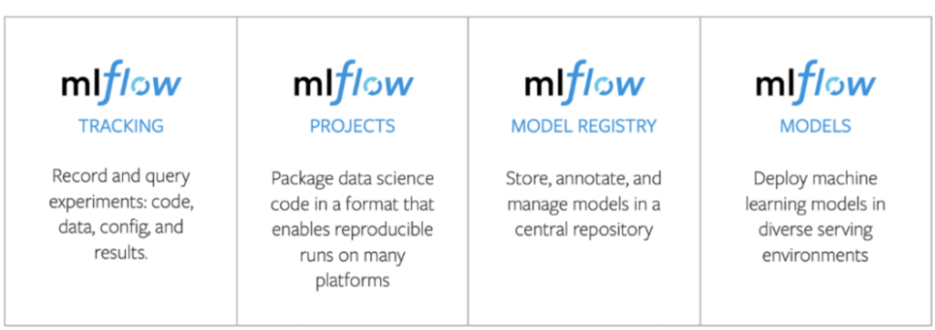


Data Bootcamp
BEST DATA TRAINING

Registration and versioning with MLFlow

To be able to enter a code from scratch in **MLFlow**, follow these steps:

https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html



Data Bootcamp
BEST DATA TRAINING

Pycaret & MLFlow

To be able to register models in MLFlow from Pycaret is very simple.

```
# initialize configuration
s = setup(data, target = 'Precio', transform_target = True, log_experiment = True, experiment_name
= 'diamond')

# within notebook (notice ! sign infront)
!mlflow ui
# on command line in the same folder
mlflow ui

# acceder a MLFlow
"localhost:5000"
```

Data Bootcamp
BEST DATA TRAINING

75

Load the MLFlow model

In order to load the model and use it, we only have to access the MLFlow path that contains the model

```
# load model
from pycaret.regression import load_model
pipeline =
load_model('C:/Users/moezs/mlruns/1/b8c10d259b294b28a3e233a9d2c209c0/artifacts/model/model')

# print pipeline
print(pipeline)
```

Data Bootcamp
BEST DATA TRAINING

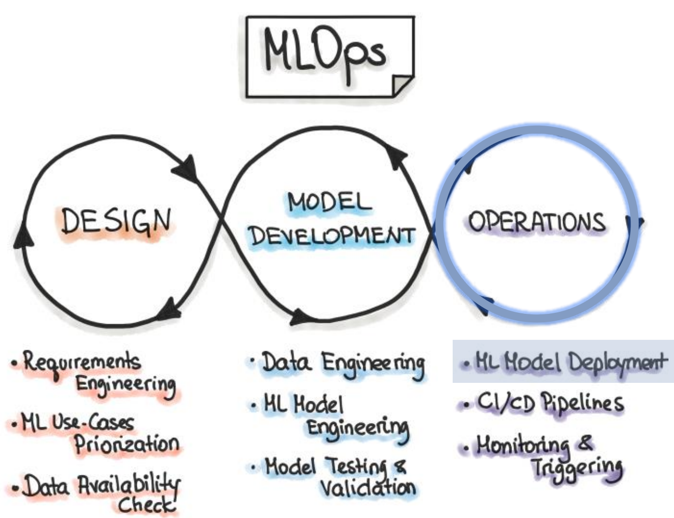
76

Deploying Models in Production

Data Bootcamp
BEST DATA TRAINING

77

MLOps stages



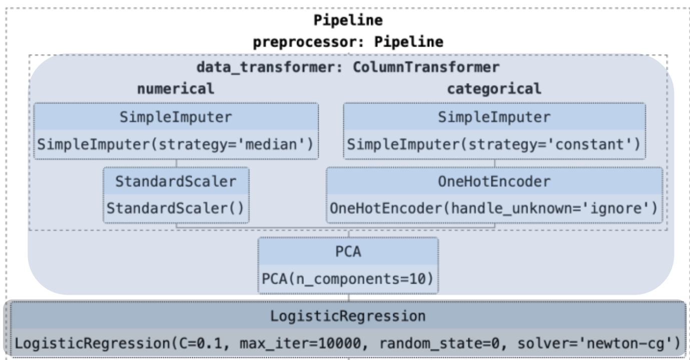
Data Bootcamp
BEST DATA TRAINING

78

Model Deployment

Model Deployment consists of integrating a model into **production environment** to make data-driven business decisions.

Models must be available for **web applications**, enterprise **software** (ERP), and **APIs**, to provide predictions for new data.



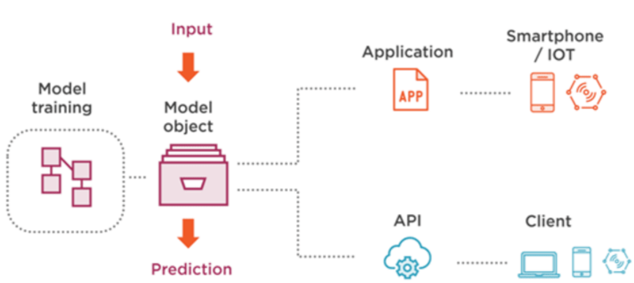
Data Bootcamp
BEST DATA TRAINING

79

Model Serving

There are different **alternatives to deploy a model** in a production environment:

1. Through **API**
2. Through **applications** (mobile/web)



Data Bootcamp
BEST DATA TRAINING

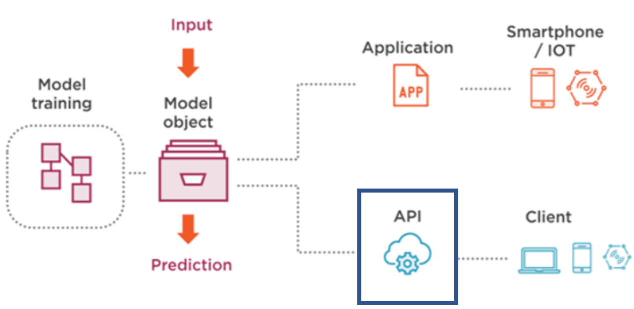
80

API creation

Model Serving

There are different **alternatives to deploy a model** in a production environment:

1. Through **API**
2. Through **applications** (mobile/web)



What is an API?

API (Application Programming Interface) creates an entry point for an application, through HTTP requests.

API: Application Abstraction + Simplification of Third Party Integration

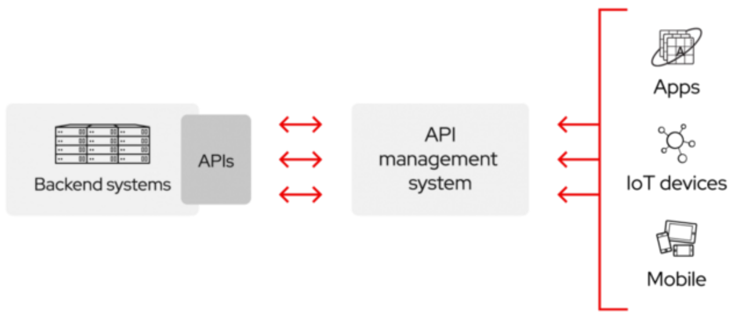


Image source: https://kunal3836

Data Bootcamp
BEST DATA TRAINING

83

Status codes and HTTP methods

HTTP methods

- **GET**: retrieve an existing resource (read only)
- **POST**: Create a new resource/send information
- **PUT**: Update an existing resource
- **PATCH**: partially update an existing resource
- **DELETE**: Delete a resource

HTTP status code

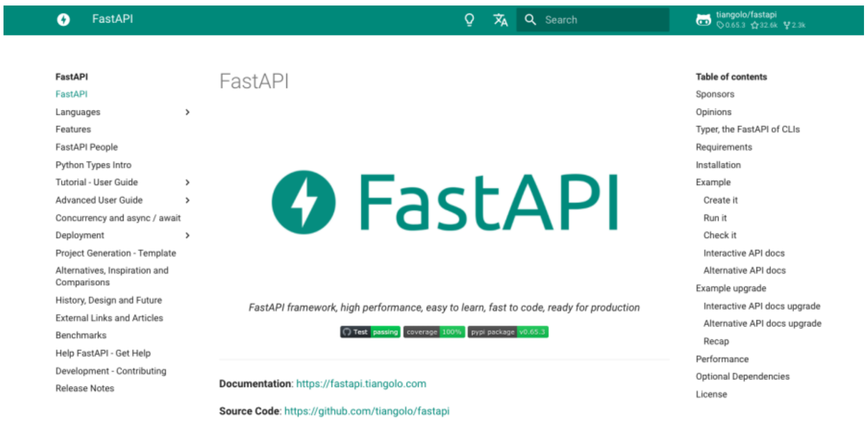
- **2xx**: Successful operation
- **3xx**: Redirect
- **4xx**: client error
- **5xx**: Server Error

Data Bootcamp
BEST DATA TRAINING

84

FastAPI

It is the reference framework for **creating robust and high-performance APIs** for production environments.

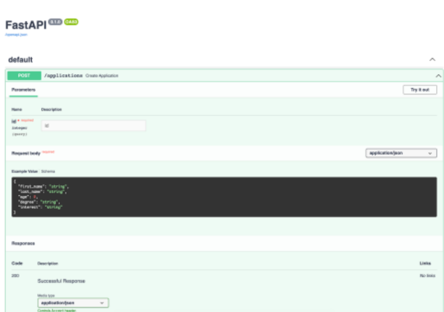


85

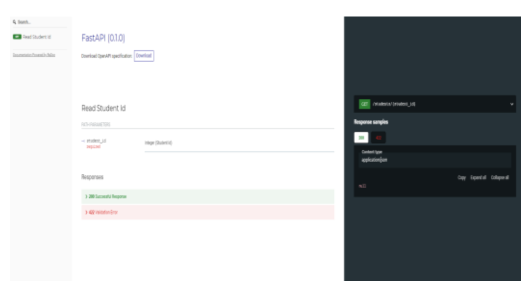
API Documentation

Interactive API **exploration** and **documentation** are automatically generated

http://localhost:8000/docs



http://localhost:8000/redoc



86

Fast API Basics

Basic function

```
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
   return {"Hello": "World"}

if __name__ == "__main__":
   uvicorn.run("hello_world_fastapi:app")
```

Methods

```
@app.get("/")
def home():
return {"Hello": "GET"}

@app.post("/")
def home_post():
return {"Hello": "POST"}
```

Data Bootcamp
BEST DATA TRAINING

87

Fast API Basics

Query Parameters

```
@app.get("/employee")
def home(department: str):
   return {"department": department}
```

Path parameters

```
@app.get("/employee/{id}")
def home(id: int):
   return {"id": id}
```

Data Type Validation: Pydantic

```
from fastapi import FastAPI, Query
from pydantic import BaseModel
from typing import Optional


class Application(BaseModel):
    first_name: str
    last_name: str
    age: int
    degree: str
    interest: Optional[str] = None

class Decision(BaseModel):
    first_name: str
    last_name: str
    probability: float
    acceptance: bool

app = FastAPI()

@app.post("/applications", response_model=Decision)
async def create_application(id: int, application: Application):

    first_name = application.first_name
    last_name = application.last_name
    proba = random.random()
    acceptance = proba > 0.5
```

Data Bootcamp
BEST DATA TRAINING

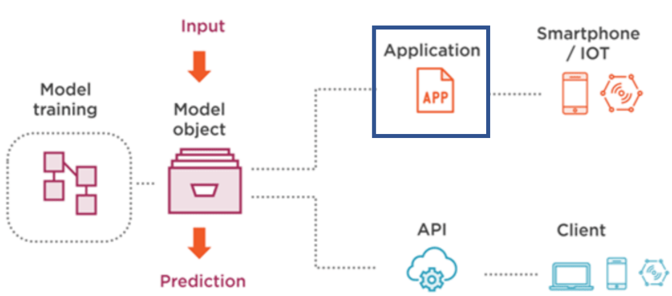
88

Developing web applications

Model Serving

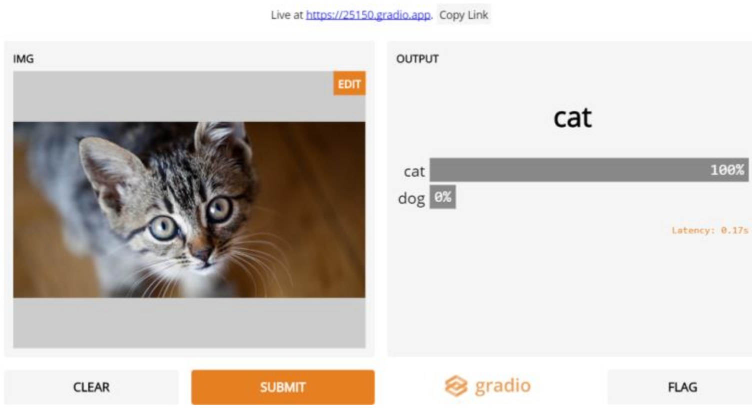
There are different **alternatives to deploy** a model in a **production** environment:

1. Through **API**
2. Through **Applications** (mobile/web)



Create web application

A **business user** will not execute a code or notebook to get a prediction. For this reason, it is advisable to facilitate the process through **a front-end** (web application, mobile, etc.).



Data Bootcamp
BEST DATA TRAINING

91

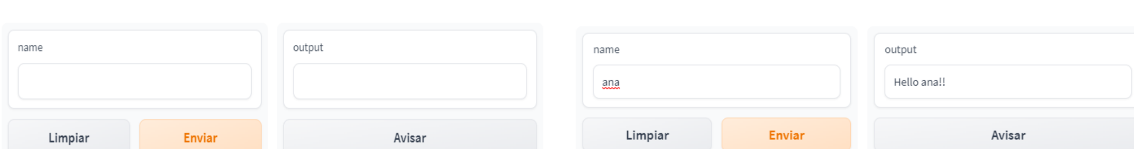
Gradio Basics

Here you can access Gradio documentation https://gradio.app/getting_started/

```
import gradio as gr

def greet(name):
    return "Hello " + name + "!!"

demo = gr.Interface(fn=greet, inputs="text", outputs="text")

demo.launch()
```

The Interface has three mandatory parameters:

- **fn:** the function to use in the user interface
- **inputs**: which component(s) to use for input, example: "text“, "image“ or "audio“
- **outputs**: which component(s) to use for the output, example: "text“, "image“, "label"



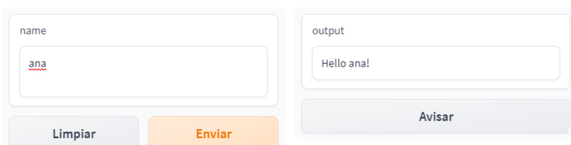
Data Bootcamp
BEST DATA TRAINING

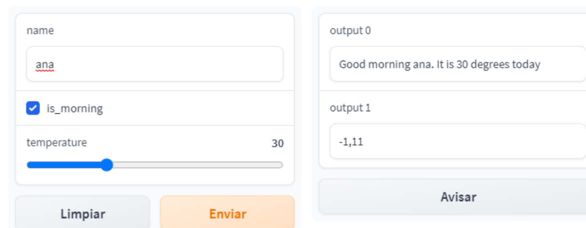
92

Gradio Basics

```
import gradio as gr

def greet(name):
    return "Hello " + name + "!"

demo = gr.Interface(
    fn=greet,
    inputs=gr.Textbox(lines=2, placeholder="Name Here..."),
    outputs="text",
)

demo.launch()
```

```
import gradio as gr

def greet(name, is_morning, temperature):
    salutation = "Good morning" if is_morning else "Good evening"
    greeting = "%s %s. It is %s degrees today" % (salutation, name, temperatu
    celsius = (temperature - 32) * 5 / 9
    return greeting, round(celsius, 2)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "checkbox", gr.Slider(0, 100)],
    outputs=["text", "number"],
)
demo.launch()
```



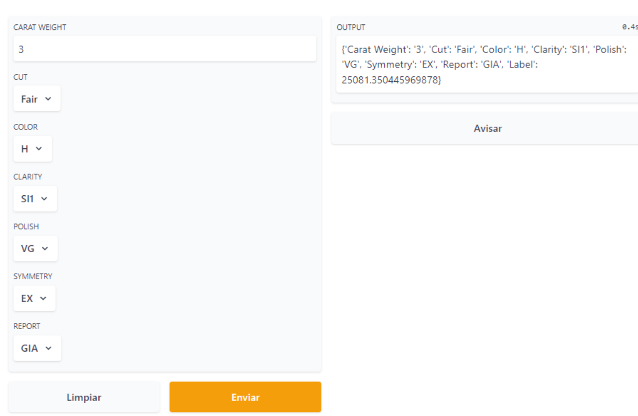


93

Create web application with Gradio

There are **libraries** that facilitate the development of applications such as Streamlit or Gradio. **Pycaret** implements a function to generate a basic web application with **Gradio**.

```
# create gradio app
create_app(lr)
```



94

Application development with Flask

Data Bootcamp
BEST DATA TRAINING

95

Flask

Flask is a web development framework written in **Python**. Supports multiple **extensions**. Flask is very easy to learn and quick to implement.

Benefits:

- Easy to use
- Flexible
- Allows testing

Flask
web development,
one drop at a time

Data Bootcamp
BEST DATA TRAINING

96

Flask Features: Simple Syntax

FLASK

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
   app.run()
```

FASTAPI

```python
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
   return {"Hello": "World"}

if __name__ == "__main__":
   uvicorn.run("hello_world_fastapi:app")
```


Flask Features: Define Routes

Next we see how we can **define the routes**

FLASK

```python
from flask import request

@app.route("/", methods=["POST", "GET"])
def home():
   if request.method == "POST":
      return {"Hello": "POST"}

   return {"Hello": "GET"}
```

FASTAPI

```python
@app.get("/")
def home():
    return {"Hello": "GET"}

@app.post("/")
def home_post():
    return {"Hello": "POST"}
```

Features of Flask

Next we have **the route and query parameters**

FLASK

```python
@app.route("/employee/<int:id>/")
def home():
    return {"id": id}
```

FASTAPI

```python
@app.get("/employee/{id}")
def home(id: int):
    return {"id": id}
```

FLASK:

```python
@app.route("/employee")
def home():
    department = request.args.get("department")
    return {"department": department}
```

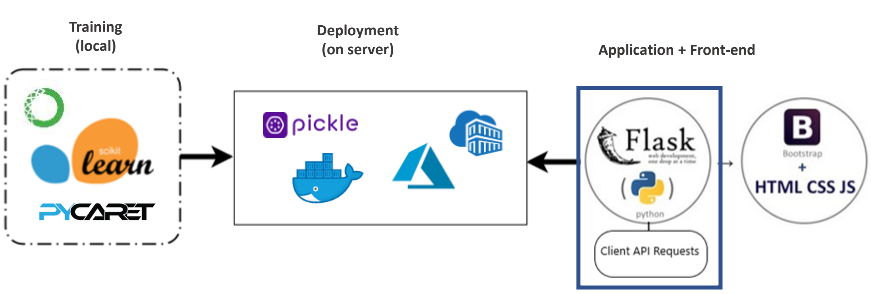
FASTAPI:

```python
@app.get("/employee")
def home(department: str):
    return {"department": department}
```

99

Create an app with Flask and HTML

We are going to develop a web application based on Flask. You will need a Flask **back-end** and **a front-end** for example in **HTML**. Then that application must be deployed on a **server** (cloud or on-promise) such as Azure, Heroku, etc.



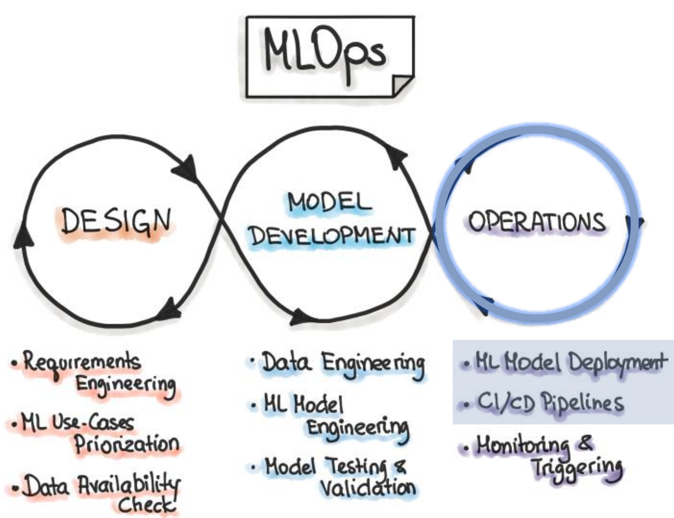
100

Containers

Data Bootcamp
BEST DATA TRAINING

101

MLOps stages



Data Bootcamp
BEST DATA TRAINING

102

Problematica de los entornos

How many times has it happened to you that your code **works fine on your computer**, **but not on others**? The reason: your computer and the rest have different Python environments.

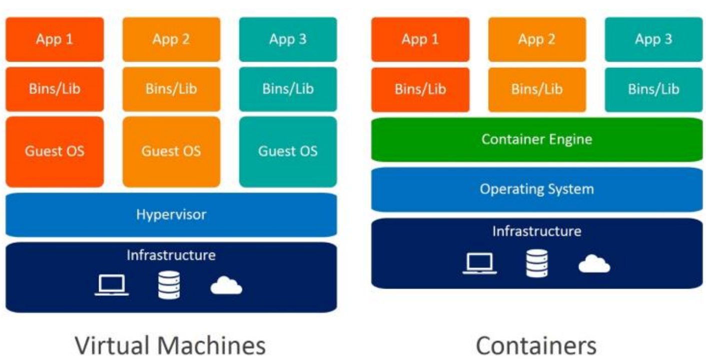
An **environment** includes all the libraries and dependencies used to create an application. If we can transfer that environment into a **container**, the model can be used elsewhere.

103

Solutions for environments

Alternatives to create an isolated environment for our application:

- Have a separate **machine**
- Use **virtual machines**
- **Container**

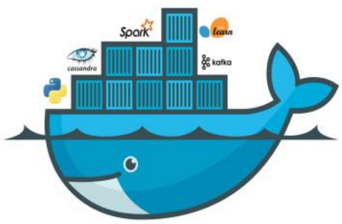


Data Bootcamp
BEST DATA TRAINING

104

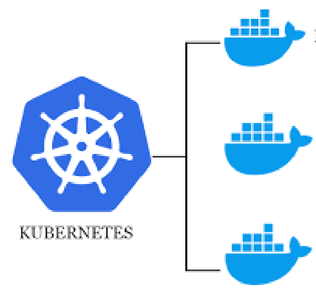
Docker

Docker is a tool to facilitate the creation, deployment and execution of applications using **containers**. These allow you to bundle an application with all of its components and ship it as a single package.



Dockerfile

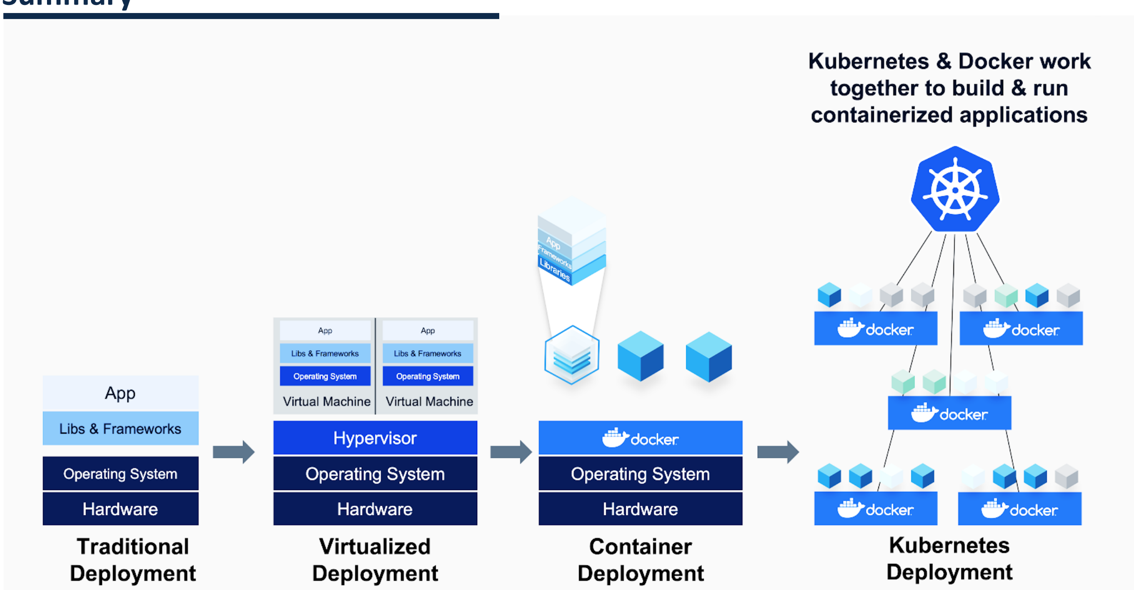
```
FROM python:3.7

RUN pip install virtualenv
ENV VIRTUAL_ENV=/venv
RUN virtualenv venv -p python3
ENV PATH="VIRTUAL_ENV/bin:$PATH"

WORKDIR /app
ADD . /app

# install dependencies
RUN pip install -r requirements.txt

# expose port
EXPOSE 5000

# run application
CMD ["python", "app.py"]
```



Data Bootcamp
BEST DATA TRAINING

105

Summary

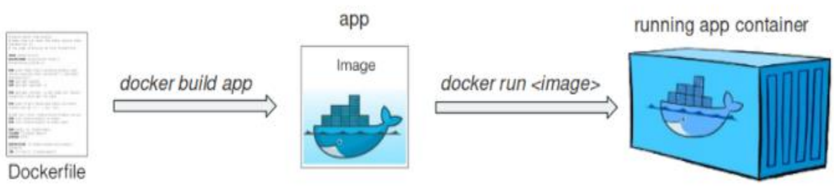


Data Bootcamp
BEST DATA TRAINING

106

Creating a container

In order to create a container we will have to generate a **Dockerfile** with the necessary libraries for the model. Then we generate the **Docker Image**, which contains all the information necessary for the execution of the model.



Container for an API

Pycaret allows you to easily **create a container for an API** to consume a Machine Learning model.

1

```
create_docker('my_first_api')
```

```
Writing requirements.txt
Writing Dockerfile
Dockerfile and requirements.txt successfully created.
To build image you have to run --> !docker image build -f "Dockerfile" -t IMAGE_NAME:IMAGE_TAG .
```

2

```
# %load requirements.txt

pycaret
fastapi
uvicorn

```

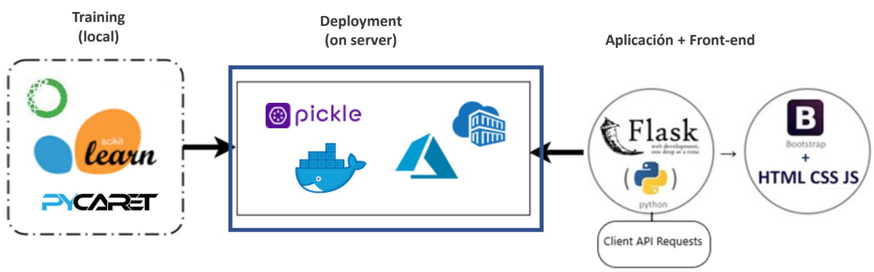
```
# %load Dockerfile


FROM python:3.8-slim

WORKDIR /app

ADD . /app

RUN apt-get update && apt-get install -y libgomp1

RUN pip install -r requirements.txt

EXPOSE 8000

CMD ["python", "my_first_api.py"]

```

3

```
!docker image build -f "Dockerfile" -t my_first_image:latest .
```

Flask Application Container

We will generate a **Docker container** for our web application. This application is developed with **Flask and HTML**, and will have a Machine Learning **model** embebed.



Data Bootcamp
BEST DATA TRAINING

109

Flask Application Container

We are going to generate a Docker container for our application.

For this we will follow the following steps:

1. Prepare the **environment** cd/path and pip install -r requirements.txt
2. Create the requirementx.txt and the Dockerfile
3. Build the Docker image with **docker build -t pycaret.azurecr.io/pycaret-insurance:latest** .
4. Test running **docker run -d -p 5000:5000 pycaret.azurecr.io/pycaret-insurance**

Data Bootcamp
BEST DATA TRAINING

110

Machine Learning in Cloud

Data Bootcamp
BEST DATA TRAINING

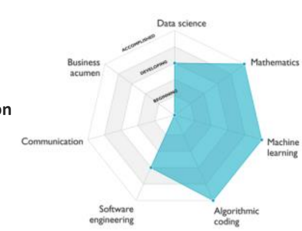
111

Importance of the Cloud

ML has been **out of reach** for most **companies** due to the high level of specialization it requires, high implementation costs and difficulties to scale. **ML in the Cloud** has a lot of benefits, such as:

Cost efficiency

Less specialization



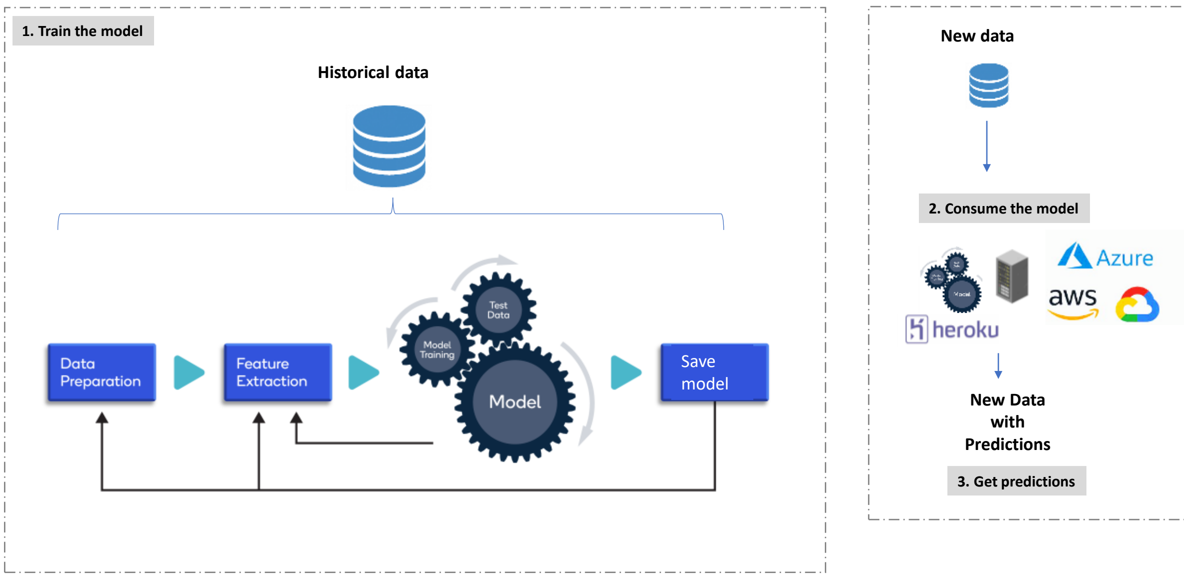
Scalability

Experimentation

Data Bootcamp
BEST DATA TRAINING

112

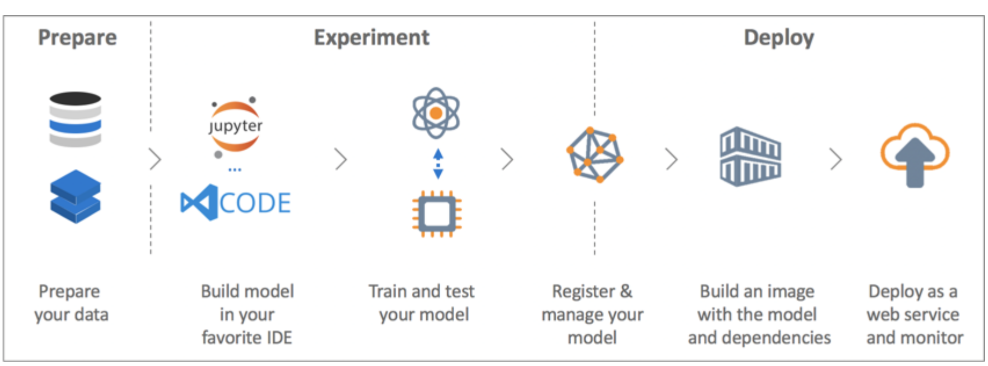
Architecture



113

Deploy to Azure Container

We can deploy our model in the cloud using **Docker Containers.** In this example, we will test deploying the model to Azure using the **Azure Container Registry.**



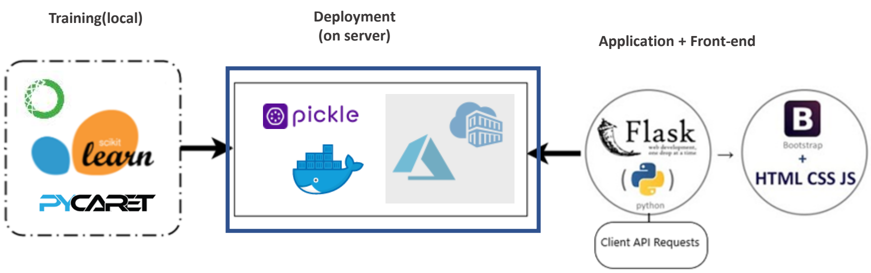
Data Bootcamp
BEST DATA TRAINING

114

Container deployment in Azure

We are going to **deploy the container** of the application that we had developed from Flask and HTML in Azure.

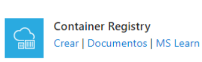
To do this, we will use the **Azure Container Registry** to upload the Docker image.



115

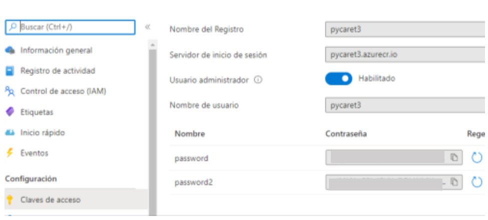
Steps for deployment to Azure Container Registry

1



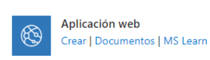
Create an Azure Container Registry service. You will create the service with pycaret3 subscription name

2



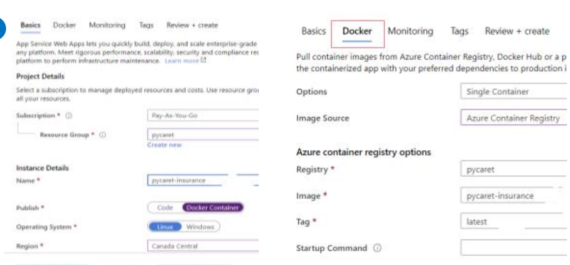
Login. Log in Azure Container registry from Anaconda prompt with command “*docker login pycaret.azurecr.io*” . Enter name and credentials from"access codes".

3



Push to Azure container. To do this, you will enter the command “*docker push pycaret.azurecr.io/pycaret-insurance:latest*”

4



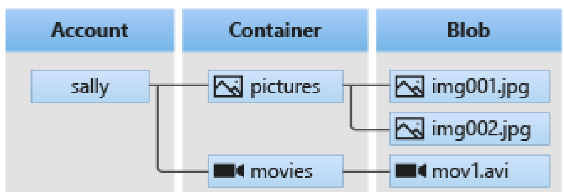
Web application service. Create a web application service with this configuration:Publish = Docker Container, in Docker->Image Source = Azure Container Registry, Image = pycaret, Tag = latest

116

Deploying models to a Blob storage

Another way to **deploy models** in the Azure Cloud is by saving them as **binary file** in **Azure Blob Storage.** To **consume this model**, we can download the model from the Blob Storage and use it to obtain predictions with new data.

Microsoft Azure
Blob Storage



Data Bootcamp
BEST DATA TRAINING

117

Azure SDKs

Azure SDKs are **collections and functions** created to facilitate the use of Azure services with **different languages**. They are designed to be consistent, accessible and reliable.

.NET	Java	JavaScript/TypeScript	Python
Obtención del SDK	Obtención del SDK	Obtención del SDK	Obtención del SDK
Documentación	Documentación	Documentación	Documentación
GitHub	GitHub	GitHub	GitHub
Ir	**C++**	**C**	**Android**
Obtención del SDK	GitHub	GitHub	GitHub
Documentación			
GitHub			
iOS			
GitHub			

https://azure.microsoft.com/es-es/downloads/

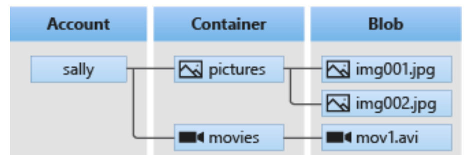
Data Bootcamp
BEST DATA TRAINING

118

Blob management with Python SDKs

Blobs are objects that can contain large amounts of unstructured data (text or binary data, images, documents, or files). There are the following Python classes to interact with these resources:

- *BlobServiceClient*:
- allows you to manipulate Azure Storage resources and containers.
- *ContainerClien*t: allows you to manipulate Azure Storage containers and their blobs.
- *BlobClient* : allows you to manipulate blobs from Azure Storage.



Link: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?toc=%2Fpython%2Fazure%2FTOC.json&tabs=environment-variable-windows

Data Bootcamp
BEST DATA TRAINING

119