In [ ]:
```python
import pandas as pd
import requests
```

In [ ]:
```python
# Dataset used to train the model
df = pd.read_csv("Data/bank.csv")
features = ['job', 'duration', 'poutcome']
df = df[features]

df.head()
```

Out[ ]:

	job	duration	poutcome
0	unemployed	79	unknown
1	services	220	failure
2	management	185	failure
3	management	199	unknown
4	blue-collar	226	unknown

Preparing virtual environment with Pipenv

Pipenv permits us to create a virtual environment to isolate your project dependencies, ensuring that packages installed for one project don't interfere with other projects.

Question 1

- Install Pipenv
- What's the version of pipenv you installed?
- Use `--version` to find out

It provides an integrated environment for managing both project dependencies and the Python runtime environment. It combines the functionality of pip (Python's package installer) and virtualenv (a tool for creating isolated Python environments).

`Terminal`

```
marcos@marcos:~$ pip install pipenv
marcos@marcos:~$ pipenv --version
marcos@marcos:~$ pipenv, version 2023.10.3
```

Question 2

- Use Pipenv to install Scikit-Learn version 1.3.1
- What's the first hash for scikit-learn you get in Pipfile.lock?

To install Python packages in an isolated virtual environment I first navigate to the specific project directory. After that, I create an empty folder and execute the package installation command.

`Terminal`

```
marcos@marcos:~$ cd /GitHub/ML_Zoomcamp/05Deploy/Homework
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ mkdir
Homework
marcos@marcos:~$ pipenv install Scikit-Learn==1.3.1

    Output -------------
    Updated Pipfile.lock
    (0e0fec5cb0e411bbb2c1c4f81b061609272a25d0c1f780d06dd30aff281bed02
    To activate this project's virtualenv, run pipenv shell.
    Alternatively, run a command inside the virtualenv with
    pipenv run.
```

- `Pipfile` : list of libraries and version of Python used
- `Pipfile.lock` : Contains the specific versions of the libraries that we used for the project.

The hash value associated with the current `Pipfile.lock` serves a similar function to Git commit hashes.

- Hash `pipfile.lock` : `0e0fec5cb0e411bbb2c1c4f81b061609272a25d0c1f780d06dd30aff281bed02`

Models

We've prepared a dictionary vectorizer and a model.

They were trained (roughly) using this code:

```
features = ['job','duration', 'poutcome']
dicts = df[features].to_dict(orient='records')

dv = DictVectorizer(sparse=False)
X = dv.fit_transform(dicts)

model = LogisticRegression().fit(X, y)
```
And then saved with Pickle. Download them:

- DictVectorizer
- LogisticRegression

Using `wget` , we downloaded the two files: the **pre-trained model** (`model1.bin`) and the **vectorized dictionary** (`dv.bin`).

`Terminal`

```
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$
PREFIX=https://raw.githubusercontent.com/DataTalksClub/machine-
learning-zoomcamp/master/cohorts/2023/05-deployment/homework
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ wget
$PREFIX/model1.bin
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ wget
$PREFIX/dv.bin
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ ls
dv.bin  model1.bin  Pipfile  Pipfile.lock
```

Load Model with Pickle

Pickle allows for the serialization of Python objects, enabling you to save and load machine learning models. With Pickle, we can store a trained model into a binary file and later reload it to make predictions.

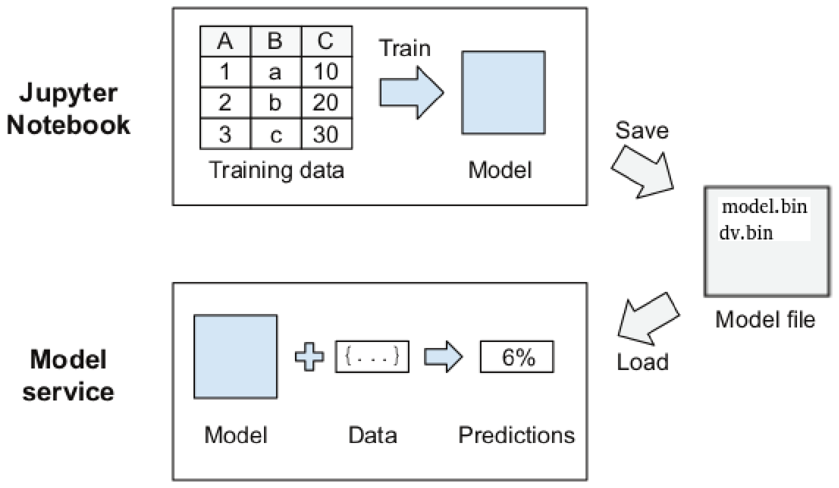
Question 3

Let's use these models!

- Write a script for loading these models with pickle
- Score this client:

```
{"job": "retired", "duration": 445, "poutcome": "success"}
```
What's the probability that this client will get a credit?

The process of loading a pre-trained model and utilizing it to predict the probability of get credit for a single customer is described in the following diagram:



First, let's create a simple Python script that loads the existing model and utilizes it to make a prediction for a single customer. This script is saved in the

Homework/question3.py folder. Inside this Python file, we have:

`question3.py`

```python
import pickle
import numpy as np
import os

# Preditiction for a single customer
def single_pred(customer, dv, model):
    X = dv.transform([customer])
    y_pred = model.predict_proba(X)[:, 1]
    return y_pred[0]

# Paths
filepath_dv = 'dv.bin'
filepath_model = 'model1.bin'

# Load dictionary vectorizer
if os.path.exists(filepath_dv):
    with open(filepath_dv, 'rb') as f_dv:
        dv = pickle.load(f_dv)
else:
    print(f"File {filepath_dv} not found.")

# Load model
if os.path.exists(filepath_model):
    with open(filepath_model, 'rb') as f_model:
        model = pickle.load(f_model)
else:
    print(f"File {filepath_model} not found.")

customer = {"job": "retired", "duration": 445, "poutcome":
"success"}
pred = single_pred(customer, dv, model)

print('prediction: %.3f' % pred)
if pred >= 0.5:
    print('Get Credit: High')
else:
    print('Get Credit: Low')
```

With the script ready, let's activate the virtual environment and execute the script in the terminal, therefore isolating the required dependencies to run the script.

`Terminal`

```
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ pipenv
shell
Launching subshell in virtual environment...
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$  .
/home/marcos/.local/share/virtualenvs/Homework-
zL5dlJ2M/bin/activate
```

```
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$  python
question3.py


$ prediction: 0.902
$ Get Credit: High
```

---

then we can exit the virtual environment with `$ exit`

Load Model in Web service with Flask

Question 4

Now let's serve this model as a web service

- Install Flask and gunicorn (or waitress, if you're on Windows)
- Write Flask code for serving the model
- Now score this client using `requests`:

```
url = "YOUR_URL"
customer = {"job": "unknown", "duration": 270, "poutcome":
"failure"}
requests.post(url, json=customer).json()
```

Again, let's create the Python script. This script is saved in the Homework/question4.py folder. Inside this Python file, we have:

`question4.py`

```
import pickle
import os
from flask import Flask, request, jsonify

# request: To get the content of a POST request
# jsonify: To respond with JSON (dictionary)

# Preditition for a single customer
def single_pred(customer, dv, model):
    X = dv.transform([customer])
    y_pred = model.predict_proba(X)[:, 1]
    return y_pred[0]


# Paths
filepath_dv = 'dv.bin'
filepath_model = 'model1.bin'

# Load dictionary vectorizer
if os.path.exists(filepath_dv):
    with open(filepath_dv, 'rb') as f_dv:
        dv = pickle.load(f_dv)
```

```
else:
    print(f"File {filepath_dv} not found.")

# Load model
if os.path.exists(filepath_model):
    with open(filepath_model, 'rb') as f_model:
        model = pickle.load(f_model)
else:
    print(f"File {filepath_model} not found.")

app = Flask('churn')
# Registers the /predict route, and assigns it to the predict
function
@app.route('/predict', methods=['POST'])
def predict():
    customer =  request.get_json()
    pred = single_pred(customer, dv, model)
    result = {'Credit probability': float(pred)}
    return jsonify(result)

if __name__ == '__main__':
    #To test it, open the browser and type
'localhost:9696/predict'
    app.run(debug=True, host='0.0.0.0', port=9696)
```

To run the web service, we run the Python file. This starts the Flask web service, making it accessible at `http://127.0.0.1:9696.`

`Terminal`

```
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ python3
question4.py
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9696
 * Running on http://192.168.1.64:9696
```

Once the web service is running, is possible to use HTTP POST requests to predict the credit probability for a given customer.

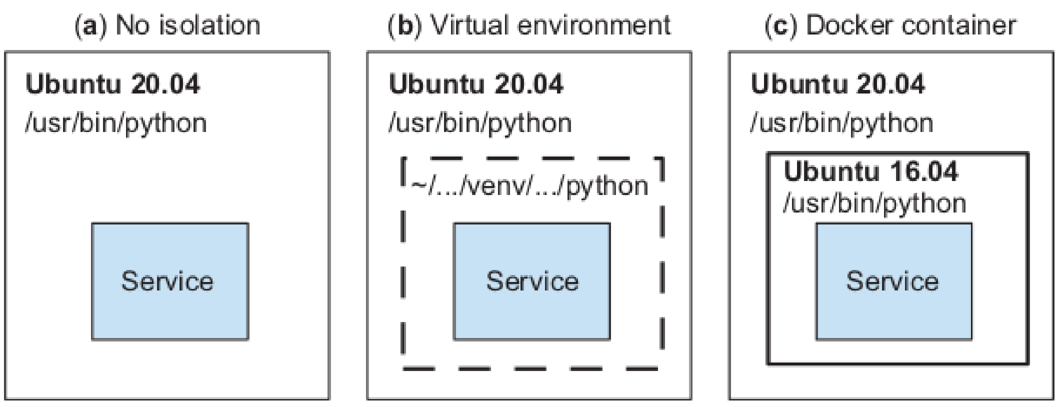
In [ ]:
```
url = "http://127.0.0.1:9696/predict"
customer = {"job": "unknown", "duration": 270, "poutcome": "failure"}
requests.post(url, json = customer).json()

# Output: {'Credit probability': 0.13968947052356817}
```

Out[ ]:
```
{'Credit probability': 0.13968947052356817}
```

Docker

Docker solves the problem with environment inconsistencies, by creating an isolated application called container, with Python dependencies, the operating system, and system libraries. This ensures uniform behavior across diverse operation systems.



Question 5

- Base image: `svizor/zoomcamp-model:3.10.12-slim`.

This image is based on `python:3.10.12-slim` and has a logistic regression model (a different one) as well a dictionary vectorizer inside.

This is how the Dockerfile for this image looks like:

`Dockerfile`

```
FROM python:3.10.12-slim
WORKDIR /app
COPY ["model2.bin", "dv.bin", "./"]
```

The Dockerfile is a script containing commands that creates a snapshot of our application along with its dependencies, environment settings. To create a Docker image from the Dockerfile we run the command line ``sudo docker build -t custom-image-name .`.

The image here was already built it and then pushed to `svizor/zoomcamp-model:3.10.12-slim`.

Download the base image `svizor/zoomcamp-model:3.10.12-slim` by using docker pull command.

So what's the size of this base image?

`Terminal`

```
marcos@marcos:~$ sudo docker pull svizor/zoomcamp-model:3.10.12-
slim
marcos@marcos:~$ sudo docker images
$ REPOSITORY              TAG            IMAGE ID        CREATED
SIZE
```

```
$ svizor/zoomcamp-model   3.10.12-slim   08266c8f0c4b   6 days
ago   147MB
```

Question 6

Now create your own Dockerfile based on the image we prepared.

It should start like that:

`Dockerfile`

```
FROM svizor/zoomcamp-model:3.10.12-slim
# add your stuff here
```

Now complete it:

- Install all the dependencies form the Pipenv file
- Copy your Flask script
- Run it with Gunicorn

After that, you can build your docker image. Now Let's run your docker container!

After running it, score this client once again:

```
url = "YOUR_URL"
client = {"job": "retired", "duration": 445, "poutcome":
"success"}
requests.post(url, json=client).json()
```

What's the probability that this client will get a credit now?

The first step is to include all the necessary Python packages that your Docker container will use. Let's update our Pipfile to includes Flask, Requests, Scikit-Learn, and Gunicorn.

`Pipfile`

```
    [[source]]
    url = "https://pypi.org/simple"
    verify_ssl = true
    name = "pypi"

    [packages]
    flask = "*"
    requests = "*"
    scikit-learn = "==1.3.1"
    gunicorn = "*"

    [dev-packages]

    [requires]
    python_version = "3.10"
```

The Dockerfile serves as a blueprint for building your Docker image. Setting up the Dockerfile:

`Dockerfile`

```dockerfile
# Start with the existing image as a base
FROM svizor/zoomcamp-model:3.10.12-slim

# Set environment variables
ENV PYTHONUNBUFFERED=TRUE

# Install pipenv
RUN pip --no-cache-dir install pipenv

# Set the working directory inside the container
WORKDIR /app

# Copy Pipenv files into the container
COPY ["Pipfile", "Pipfile.lock", "./"]

# Install Python dependencies and clean cache
RUN pipenv install --deploy --system && \
rm -rf /root/.cache

# Copy the Flask script and the .bin files into the container
COPY ["question4.py", "model1.bin", "dv.bin", "./"]

# Port the app runs on
EXPOSE 9696

# Run Gunicorn
ENTRYPOINT ["gunicorn", "--bind", "0.0.0.0:9696",
"question4:app"]
```

Once the Dockerfile is completed, we can build the base image. This is achieved using the command `sudo docker build -t custom-image-name .` . Here, the -t flag allows us to assign a custom tag name to the image. The final argument — represented by the dot — indicates that the Dockerfile resides in the current directory.

After successfully building the image, it can be executed using the command `sudo docker run -p 9696:9696 custom-image-name`. In the terminal, we first update the `Pipfile.lock` to ensure it is synced with our Pipfile. Following that, we prepare the Docker image for deployment.

`Terminal`

```
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ pipenv lock
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ sudo docker
build -t homework5 .
marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ sudo docker
images
$ REPOSITORY              TAG              IMAGE ID        CREATED
```

```
SIZE
$ homework5               latest         8c766943de0c   4 minutes
ago   431MB
$ svizor/zoomcamp-model   3.10.12-slim   08266c8f0c4b   7 days
ago      147MB

marcos@marcos:~/GitHub/ML_Zoomcamp/05Deploy/Homework$ sudo docker
run -p 9696:9696 homework5
[2023-10-16 12:14:51 +0000] [1] [INFO] Starting gunicorn 21.2.0
[2023-10-16 12:14:51 +0000] [1] [INFO] Listening at:
http://0.0.0.0:9696 (1)
[2023-10-16 12:14:51 +0000] [1] [INFO] Using worker: sync
[2023-10-16 12:14:51 +0000] [7] [INFO] Booting worker with pid: 7
```

The Docker container will have the following directory structure:

- `/` (Root directory)
  - `usr/`
    - `local/`
      - `lib/`
        - `python3.10/`
          - `site-packages/` (Python packages)
  - `app/` (Set as WORKDIR)
    - `question4.py`
    - `model1.bin`
    - `dv.bin`
    - `Pipfile`
    - `Pipfile.lock`

Once Flask is operational inside the Docker container, we can interact with it via HTTP POST requests to predict a customer's credit probability.

In [ ]:
```python
url = "http://127.0.0.1:9696/predict"
customer = {"job": "retired", "duration": 445, "poutcome": "success"}
requests.post(url, json = customer).json()

# Output: {'Credit probability': 0.9019309332297606}
```

Out[ ]:
```
{'Credit probability': 0.9019309332297606}
```