Task 1. Set up your environment

Enable the Vertex AI API

- Navigate to the Vertex AI section of your Cloud Console and click **Enable All Recommended AI API**.

Create dataset

1. To create a BigQuery dataset, navigate to BigQuery on Google Cloud Console.

2. Make sure that you select the right project from the top of the console page.

3. In the **Explorer** panel, click on **View actions** (⋮) next to your project ID and select **Create dataset**.

A pop-up will appear.

4. Enter the *Dataset ID*: **titanic**, *Data location*: **eu (multiple regions in European Union)** and then click **Create dataset**.

Create table

You need a table to load your data.

1. First download the [Titanic dataset] locally.

> **Note:** In case of any difficulty with downloading the dataset in Incognito mode, use the normal window to download the `Titanic dataset`.

2. Rename your downloded dataset as **titanic_toy.csv**.

Then, from the UI:

1. Open the **titanic** dataset that you created in the previous step. (Click on **View actions** (⋮) next to your dataset and select **Open**).

2. Click **Create table** and specify the following:

   - Create table from: **Upload**
   - Select file: *Use the downloaded Titanic dataset*
   - File format: **CSV**
   - Table name: **survivors**

- Auto-detect: Select auto-detect checkbox - **Schema**

3. Click **Create table**.

4. Click **View actions** icon next to `survivors` and select **Copy ID**. Save the copied table ID to use later in the lab.

You have now created and populated the table with the Titanic dataset! You can explore its contents, run queries, and analyze your data.

Task 2. Create a dataset

Datasets in Vertex AI allow you to create datasets for your machine learning workloads. You can create datasets for structured data (CSV files or BigQuery tables) or unstructured data such as images and text. It is important to notice that Vertex AI datasets just reference your original data and there is no duplication.

Create ML dataset

Create ML dataset

1. In the Google Cloud Console, on the **Navigation Menu**, select **Vertex AI > Datasets**.

Once you select Vertex AI, you can select a region you want your resources to use. This lab is using `europe-west4` as a region. If you need to use a different region, you can do so; just replace `europe-west4` with the region of your choice for the rest of this lab.

2. Select **europe-west4** and click **Create dataset**.

3. Give your dataset a name, like **titanic**.

You can create datasets for images, text, or videos, as well as tabular data.

4. The Titanic dataset is tabular, so you should click the **Tabular** tab.

5. For region selection, select **europe-west4** and click **Create**.

At this stage, you have just created a placeholder. You have not yet connected to the datasource; you will do so on the following step.

Select datasource

As you have already loaded the Titanic dataset in BigQuery, you can connect your ML

As you have already loaded the Titanic dataset in BigQuery, you can connect your ML dataset to your BigQuery table.

1. Choose **Select a table or view from BigQuery**.

2. Paste the already copied table ID in the BROWSE field.

3. Once you select the dataset, click **Continue**.

Generate statistics

Under the **Analyze** tab you can generate statistics regarding your data. This gives you the ability to quickly peek at the data and check for distributions, missing values, etc.

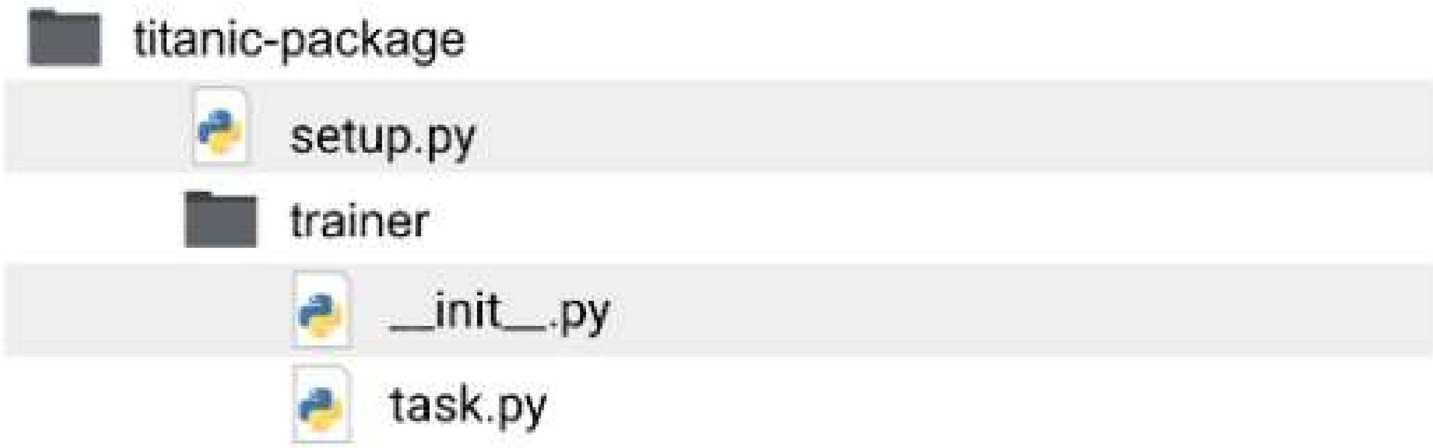
- In order to run the statistical analysis, click **Generate statistics**. It can take a couple of minutes to execute, so if you'd like you can continue with the lab and come back later to see the results.

Task 3. Custom training package using Workbench

It is a good practice to package and parameterize your code so that it becomes a portable asset.

In this section, you will create a training package with custom code using [Vertex AI Workbench](). A fundamental step in using the service is to be able to create a Python source distribution, AKA a distribution package. This is not much more than creating folders and files within the distribution package. The next section will explain how a package is structured.

Application structure



Let's see what those folders and files are for:

- **titanic-package**: This is your working directory. Inside this folder you will have your package and code related to the Titanic survivor classifier.

- **setup.py**: The setup file specifies how to build your distribution package. It includes information such as the package name, version, and any other packages that you might need for your training job and which are not included by default in GCP's pre-built training containers.

- **trainer**: The folder that contains the training code. This is also a Python package. What makes it a package is the empty `__init__.py` file that is inside the folder.

- `__init__.py`: Empty file called `__init__.py`. It signifies that the folder that it belongs to is a package.

- **task.py**: The task.py is a package module. Here is the entry point of your code and it also accepts CLI parameters for model training. You can include your training code in

task.py: The task.py is a package module. Here is the entry point of your code and it also accepts CLI parameters for model training. You can include your training code in this module as well or you can create additional modules inside your package. This is entirely up to you and how you want to structure your code. Now that you have an understanding of the structure, we can clarify that the names used for the package and module do not have to be "trainer" and "task.py". We are using this naming convention in this lab so that it aligns with our online documentation, but you can in fact pick the names that suit you.

Create your notebook instance

Now let's create a notebook instance and try training a custom model.

1. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**.

2. On the Notebook instances page, click **New Notebook** and start an instance with **Python 3**, which includes Scikit-learn. You will use a Scikit-learn model for your classifier.

A pop-up will appear. Here you can change settings like the region in which your notebook instance will be created and the compute power you require.

3. As you are not dealing with a lot of data and you only need the instance for development purposes, please do not change any of the settings; simply click **Create**.

The instance will be up and running in no more than a couple of minutes.

4. Once the instance is ready, go ahead and **Open Jupyterlab**.

5. You will see "Build recommended" pop up, click Build. If you see the build failed, ignore it.

Create your package

Now that the notebook is up and running, you can start building your training assets.

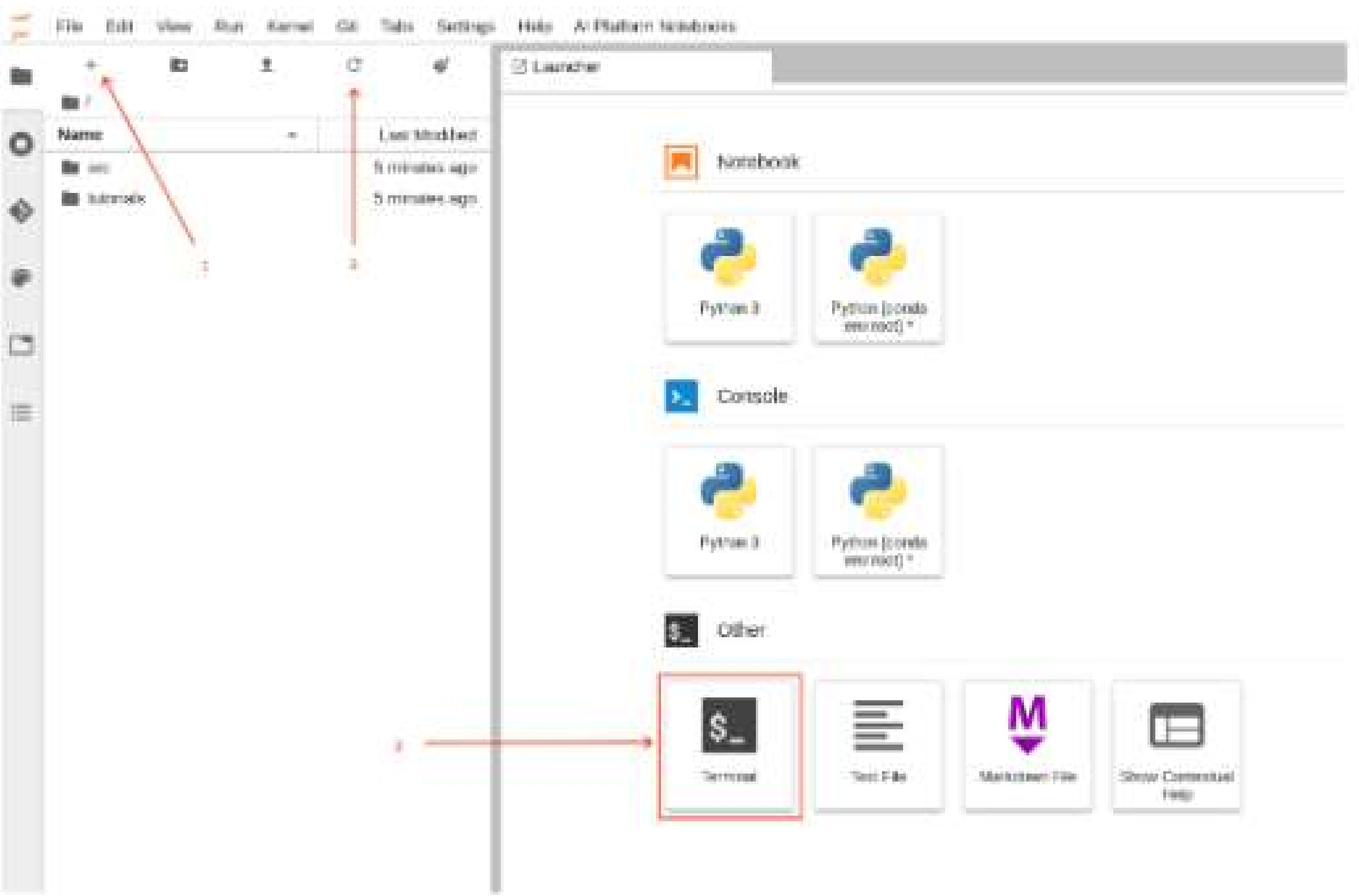
For this task it is easier to use the terminal.

1. From the Launcher, click on **Terminal** to create a new terminal session.

2. Now, in the terminal, execute the following commands to create the folder structure with the required files:

```
mkdir -p /home/jupyter/titanic/trainer
touch /home/jupyter/titanic/setup.py
/home/jupyter/titanic/trainer/__init__.py
/home/jupyter/titanic/trainer/task.py
```

3. Once you run the commands, click the refresh button to see the newly created folder and files.



4. Copy-paste the following code in **titanic/trainer/task.py**. The code contains comments, so it will help to spend a few minutes going through the file to better understand it:

Build your package

Now it is time to build your package so that you can use it with the training service.

1. Copy-paste the following code in **titanic/setup.py**:

```
from setuptools import find_packages
from setuptools import setup
REQUIRED_PACKAGES = [
    'gcsfs==0.7.1',
    'dask[dataframe]==2021.2.0',
    'google-cloud-bigquery-storage==1.0.0',
    'six==1.15.0'
]
setup(
    name='trainer',
    version='0.1',
    install_requires=REQUIRED_PACKAGES,
    packages=find_packages(), # Automatically find packages
within this directory or below.
    include_package_data=True, # if packages include any data
files, those will be packed together.
    description='Classification training titanic survivors
prediction model'
)
```

2. Press **Ctrl+S** to save the file.

3. Return to your terminal and test whether you can train a model using **task.py**.

4. First, create the following environment variables, but remember to ensure that you have selected the right GCP project from the console:

   - **PROJECT_ID** Will be set to the selected project ID

   - **BUCKET_NAME** Will be the PROJECT_ID and "-bucket" attached to it

```
export REGION="europe-west4"
export PROJECT_ID=$(gcloud config list --format
'value(core.project)')
export BUCKET_NAME=$PROJECT_ID"-bucket"
```

5. Create a bucket where you want to export your trained model:

```
gsutil mb -l $REGION "gs://"$BUCKET_NAME
```

Now run the following commands. You are using all of your training data to test. The same dataset is used for testing, validation, and training. Here you want to ensure that the code executes and that it is free of bugs. In reality you will want to use different test and

Now run the following commands. You are using all of your training data to test. The same dataset is used for testing, validation, and training. Here you want to ensure that the code executes and that it is free of bugs. In reality you will want to use different test and validation data. You will leave that for Vertex AI training service to handle.

6. First, install the required libraries.

```
cd /home/jupyter/titanic
pip install setuptools
python setup.py install
```

Note: You can ignore the `error: google-auth 2.3.3 is installed but google-auth<2.0dev,>=1.25.0 is required by {'google-api-core'}`, as it does not affect the lab functionality.

7. Now run your training code to verify that it executes without issues:

```
python -m trainer.task -v \
    --model_param_kernel=linear \
    --model_dir="gs://"$BUCKET_NAME"/titanic/trial" \
    --data_format=bigquery \
    --training_data_uri="bq://"$PROJECT_ID".titanic.survivors"
\
    --test_data_uri="bq://"$PROJECT_ID".titanic.survivors" \
    --
```

```
    --training_data_uri="bq://"$PROJECT_ID".titanic.survivors"
\
    --test_data_uri="bq://"$PROJECT_ID".titanic.survivors" \
    --
validation_data_uri="bq://"$PROJECT_ID".titanic.survivors"
```

If the code executed successfully, you will be able to see INFO logs printed. The two lines indicate the f1 score, which should be around 0.85, and the last line indicating that the training job completed successfully:

```
INFO:root:f1score: 0.85
INFO:root:Training job completed. Exiting...
```

Congratulations! You are ready to create your training Python package!

8. The following command does exactly that:

```
cd /home/jupyter/titanic
python setup.py sdist
```

After the command executes, you will see a new folder called **dist** that contains a tar.gz file. This is your Python package.

9. You should copy the package to GCS so that the training service can use it to train a new model when you need to:

9. You should copy the package to GCS so that the training service can use it to train a new model when you need to:

```
gsutil cp dist/trainer-0.1.tar.gz
"gs://"$BUCKET_NAME"/titanic/dist/trainer-0.1.tar.gz"
```

Task 4. Model training

In this section you will train a model on Vertex AI. You are going to use the GUI for that. There is also a programmatic way to do this using a Python SDK; however, using the GUI will help you to better understand the process.

1. From the Google Cloud Console, navigate to **Vertex AI** > **Training**.

2. Select the region as **europe-west4**.

3. Click **Create**.

Training method

Training method

In this step, select the dataset and define the objective for the training job.

1. **Dataset**: The dataset you created a few steps back. The name should be **titanic**.
2. **Objective**: The model predicts whether an individual was likely to survive the Titanic tragedy. This is a **Classification** problem.
3. **Custom Training**: You want to use your custom training package.
4. Click **Continue**.

Model details

Now define the model name.

The default name should be the name of the dataset and a timestamp. You can leave it as is.

1. If you click **Advanced Options**, you will see the option to define the split of data into training, testing, and validation sets. Random assignment will randomly split the data into training, testing, and validation. This seems like a good option.

into training, testing, and validation. This seems like a good option.

2. Click **Continue**.

Training container

Define your training environment.

1. **Pre-built container**: Google Cloud offers a set of pre-built containers that make it easy to train your models. Those containers support frameworks such as Scikit-learn, TensorFlow and XGBoost. If your training job is using something exotic you will need to prepare and provide a container for training(custom container). Your model is based on Scikit-learn and a pre-built container already exists.

2. **Model framework**: `Scikit-learn`. This is the library you used for model training.

3. **Model framework version**: Your code is compatible with `0.23`.

4. **Package location**: You can browse to the location of your training package. This is the location where you uploaded training-0.1.tar.gz. If you followed the previous steps correctly, the location should be `gs://YOUR-BUCKET-NAME/titanic/dist/trainer-0.1.tar.gz` and YOUR-BUCKET-NAME is the name of the bucket you used under the *Build your package* section.

5. **Python module**: The Python module you created in Notebooks. It will correspond to the

5. **Python module**: The Python module you created in Notebooks. It will correspond to the folder that has your training code/module and the name of the entry file. This should be `trainer.task`

6. **BigQuery project for exporting data**: In Step 1 you selected the dataset and defined an automatic split. A new dataset and tables for train/test/validate sets will be created under the selected project.

    - **Enter the same project ID you are running for the lab**. Additionally, training/test/validation datasets URIs will be set as environment variables in the training container, so you can automatically use those variables to load your data. The environment variable names for the datasets will be `AIP_TRAINING_DATA_URI`, `AIP_TEST_DATA_URI`, `AIP_VALIDATION_DATA_URI`. An additional variable will be `AIP_DATA_FORMAT` which will be either `csv` or `bigquery`, depending on the type of the selected dataset in Step 1. You have already built this logic in task.py. Observe this example code (taken from task.py):

```
...
parser.add_argument( '--training_data_uri ',
   help = 'Directory to output model and artifacts',
   type = str,
   default = os.environ['AIP_TRAINING_DATA_URI'] if
'AIP_TRAINING_DATA_URI' in os.environ else ""  )
...
```

```
'AIP_TRAINING_DATA_URI' in os.environ else ""  )
...
```

7. **Model output directory**: The location the model will be exported to. This is going to be an environment variable in the training container called `AIP_MODEL_DIR`. In our task.py there is an input parameters to capture this:

```
...
parser.add_argument( '--model_dir',
        help = 'Directory to output model and artifacts',
       type = str,
       default = os.environ['AIP_MODEL_DIR'] if 'AIP_MODEL_DIR' in
os.environ else ""  )
...
```

8. You can use the environment variable to know where to export the training job artifacts. Let's select: `gs://YOUR-BUCKET-NAME/titanic/`

9. Click **Continue**.

Hyperparameter tuning

The hyperparameter tuning section allows you to define a set of model parameters that you would like to tune your model with. Different values will be explored in order to

The hyperparameter tuning section allows you to define a set of model parameters that you would like to tune your model with. Different values will be explored in order to produce the model with the best parameters. In your code, you did not implement the hyperparameter tuner functionality. It's only a few lines of code (about five lines) but you did not want to add this complexity now.

- Let's skip this step by selecting **Continue**.

Compute and pricing

Where do you want your training job to run and what type of server do you want to use? Your model training process is not hungry for resources. You were able to run the training job inside a relatively small notebook instance and the execution finishes quite fast.

1. With that in mind, you choose:

- **Region**: `europe-west4`

- **Machine type**: `n1-standard-4`

2. Click **Continue**.

Prediction container

Prediction container

In this step you can decide if you want to just train the model, or also add settings for the prediction service used to productionize your model.

You will be using a **pre-built container** in this lab. However, keep in mind that Vertex AI gives you a few options for model serving:

- **No prediction container**: Just train the model, and worry about productionizing the model later.
- **Pre-built container**: Train the model and define the pre-built container to be used for deployment.
- **Custom container**: Train the model and define a custom container to be used for deployment.

1. You should choose a **pre-built container**, since Google Cloud already offers a Scikit-Learn container. You will deploy the model after the training job is completed.

- **Model framework**: `scikit-learn`
- **Model framework version**: `0.23`
- **Model directory**: `gs://YOUR-BUCKET-NAME/titanic/`. This should be the same as the model output directory you defined in Step 3.

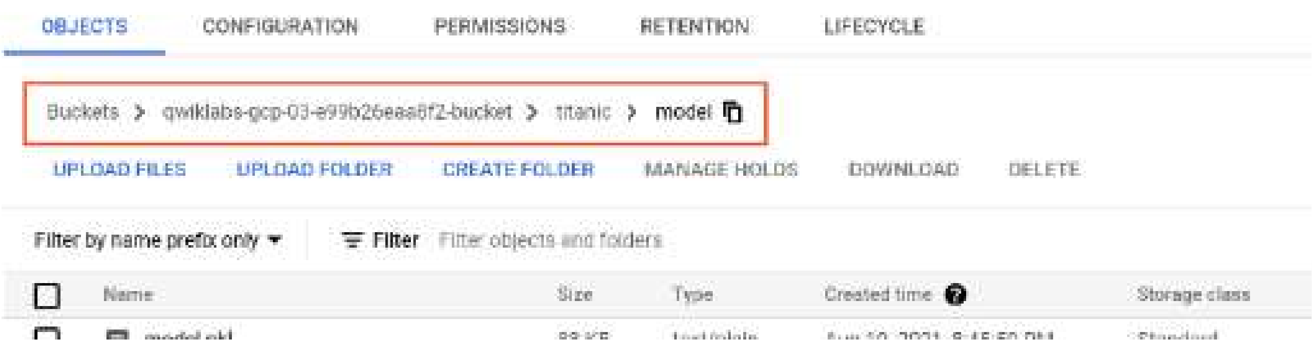
the model output directory you defined in Step 3.

2. Click **Start training**.

The new training job will show under the **Training pipeline** tab. The training will take around 15 minutes to complete.

Task 5. Model evaluation

After the training job completes, artifacts will be exported under `gs://YOUR-BUCKET-NAME/titanic/model/`. You can inspect the `report.txt` file which contains evaluation metrics and classification report of the model.



Task 6. Model deployment

1. In Cloud Console, on the **Navigation menu**, click **Vertex AI** > **Training**.

2. After the model training job is completed, select the trained model and deploy it to an endpoint.

3. Navigate to **DEPLOY & TEST** tab and then click **DEPLOY TO ENDPOINT**.

On the pop-up, you can define the required resources for model deployment:

- **Endpoint name**: Endpoint URL where the model is served. A reasonable name for that would be `titanic-endpoint`. Click **Continue**.

- **Traffic split**: Defines the percentage of traffic that you want to direct to this model. An endpoint can have multiple models and you can decide how to split the traffic among endpoint can have multiple models and you can decide how to split the traffic among them. In this case you are deploying a single model so the traffic has to be `100` percent.

- **Minimum number of compute nodes**: The minimum number of nodes required to serve model predictions. Start with `1`. Additionally the prediction service will autoscale in case there is traffic

- **Maximum number of compute nodes**: In case of autoscaling, this variable defines the upper limit of nodes. It helps protect against unwanted costs that autoscaling might result in. Set this variable to `2`.

- **Machine type**: Google Cloud offers a set of machine types you can deploy your model to. Each machine has its own memory and vCPU specs. Your model is simple, so serving on an `n1-standard-4` instance will do the job.

4. Click **Done** and then click **Deploy**.

Task 7. Model prediction

Task 7. Model prediction

1. Under **Deploy your model**, test the model prediction endpoint. The GUI provides a form to send a JSON request payload and responds back with the predictions as well as the model ID used for the prediction. That is because you can deploy more than one model to an endpoint and split the traffic.

2. Try the following payload and perhaps change some of the values to see how the predictions change: The sequence of the input features is ['sex', 'age', 'fare', 'pclass', 'embarked', 'home_dest', 'parch', 'sibsp'].

```
{
```

2. Try the following payload and perhaps change some of the values to see how the predictions change: The sequence of the input features is ['sex', 'age', 'fare', 'pclass', 'embarked', 'home_dest', 'parch', 'sibsp'].


```
{
    "instances": [
      ["male", 29.8811345124283, 26.0, 1, "S", "New York, NY",
0, 0],
      ["female", 48.0, 39.6, 1, "C", "London / Paris", 0, 1]]
}
```


3. Click **Predict**.

The endpoint responds with a list of zeros or ones in the same order as your input. 0 means it is more likely that the individual will not survive the Titanic accident and 1 means the individual is likely to survive it.

Task 8. Cleaning up

Task 8. Cleaning up

Congratulations! You have created a dataset, packaged your training code, and run a custom training job using Vertex AI. Furthermore, you deployed the trained model and sen some data for predictions.

Given that you do not need the created resources, it is a good idea to delete them in order to avoid unwanted charges.

1. Navigate to the Datasets page in the console, click the three dots on the dataset you want to delete, and click **Delete dataset**. Then click **Delete** to confirm the deletion.

2. Navigate to the Workbench page in the console, select **only** the notebook you want to delete, and click **Delete** from the top menu. Then click **Delete** to confirm the deletion.

3. To delete the endpoint you deployed, in the **Endpoints** section of your Vertex AI console, click on the endpoint, then click the overflow menu (⋮) and select **Undeploy model from endpoint**, and then click **Undeploy**.

4. To remove the endpoint, click the overflow menu (⋮), and then click **Delete endpoint**. Then click **Confirm**.

5. Navigate to Models console page, click the three dots (⋮) on the model you want to delete, and click **Delete model**. Then click **Delete**.

6. To delete the Cloud Storage bucket, on the **Cloud Storage** page, select your bucket, anc then click **Delete**. Confirm deletion by typing `DELETE` and then click **Delete**.

6. To delete the Cloud Storage bucket, on the **Cloud Storage** page, select your bucket, and then click **Delete**. Confirm deletion by typing `DELETE` and then click **Delete**.

7. To delete the BigQuery dataset, perform the following steps:

- Navigate to the BigQuery console.
- In the **Explorer** panel, click on the **View actions** icon next to your dataset. Click **Delete**.
- In the Delete dataset dialog box, confirm the delete command by typing `delete` and then click **Delete**.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.