Terminology: S3 = cheap, reliable file hosting on AWS; EC2 = remote computers in AWS

1. Save your trained model to a file using the pickle library
2. Upload the model file to S3
3. ssh into your EC2 instance
4. Install the AWS Command Line Interface onto your EC2 instance using `pip3 install awscli`
5. Go to the AWS Management Console and create a new IAM user. Give this new user programmatic access to AWS (but not access to the management console), as well as permissions to access S3 and download the credentials for this user.
6. Back on your EC2 instance, type `aws configure` and enter the downloaded credentials for the new IAM user. Just press enter for the default region name and output format.
7. On your EC2 instance, make a new directory, create a new virtual environment and activate the new venv for your model prediction code:
    a. mkdir predictions
    b. cd predictions
    c. virtualenv prediction_venv
    d. source prediction_venv/bin/activate
8. Put the provided test.py and requirements.txt files into your `predictions` directory. Make sure that the versions of the libraries you use for model predictions are the same as those you used to build the model! The version of Python used should also be the same.
9. Install the libraries in requirements.txt using `pip install -r requirements.txt`
10. Run the model prediction code using `python test.py`. This will copy the model from S3 onto your EC2 instance, load the model, use the model to make predictions and then finally delete the model from the EC2 instance (*not* from S3!). Because the model prediction code always downloads the model from S3 before making predictions, it means we'll always be using the latest version of the model.

Creating a Flask model server

1. Go here: https://github.com/iskandery/flask-ml-server
    a. This contains sample code for creating a Flask model server
    b. ssh into your EC2 instance and run:
    git clone https://github.com/iskandery/flask-ml-server.git
    c. cd flask-ml-server
2. Create a virtual environment: virtualenv flask_venv
3. Activate the virtual environment: source flask_venv/bin/activate
4. Install required libraries: pip install -r requirements.txt
5. Run python main.py to start the Flask web server. Don't worry if there's a message saying No Model -- you'll train the model later!
6. You can stop the server with Ctrl-C, but leave it running for now
7. Create a new terminal window/tab and ssh into your EC2 instance again.
8. This new window
9. Test that the server is running correctly by running
    a. Train with existing Titanic CSV file: curl -X GET -v http://localhost:5000/train
    b. curl -X POST -H "Content-type: application/json" -d '[{"Age": 85, "Sex": "male", "Embarked": "S"}, {"Age": 24, "Sex": "female", "Embarked": "C"}]' -v http://localhost:5000/predict
    c. Train with new data: curl -X POST -H "Content-type: application/json" -d '[{"Age": 85, "Sex": "male", "Embarked": "S", "Survived": 0}, {"Age": 24, "Sex": "female", "Embarked": "C", "Survived": 1}]' -v http://localhost:5000/train_new
    d. At each stage, check both your terminal windows to see both the server logs as well as the response from curl
10. Want to look at a minimal Flask app? Press Ctrl-C to stop your existing Flask server, run python main_barebones.py and take a look at the test endpoint (you can open the main_barebones.py file using rsub). Try to access this test endpoint using the curl command.
11. Stop the barebones server by pressing Ctrl-C and start the full model server by typing python main.py
12. So far you've just been running curl on the EC2 instance itself to talk to the Flask server. If you want to make your server accessible externally, you need to open up the default Flask port 5000 on the EC2 instance:
    a. Log in on aws.amazon.com
    b. Navigate to EC2 -> Instances -> click on the security group for your instance

    c. Add a new inbound rule to your security group: Custom TCP rule, port 5000, source 0.0.0.0/0, description “Flask web server”

13. On your local machine (you can now exit from the EC2 instance you’ve been running the curl command on), download the get_predictions.py file from https://github.com/iskandery/flask-ml-server. You can do this either by using git clone (to get the whole repo) or just grabbing the file itself.
14. Edit the get_predictions.py file to change the API_HOST to point to your own EC2 instance. To get this, go to your EC2 instance description again on aws.amazon.com and use the “IPv4 Public IP” address, keeping the http:// at the beginning and :5000 at the end.
15. Run python get_predictions.py on a terminal on your local machine to train a model and grab predictions from your Flask server. Notice that all the heavy lifting is performed on the server, with the client (your local machine) only responsible for displaying the response from the server.
16. Want to have the server running even after you disconnect?
    a. Look into running the Python script using & (for putting the Python process into the background) and nohup (Google for this!)
    b. Also look into using “supervisor” to control the Python script, so that for example the Flask server will restart automatically when the computer restarts

Deploying a new model to the server

1. After training your model, save it to disk and dump some test data inputs:
    a. from sklearn.externals import joblib
    b. import pickle
    c. joblib.dump(model, ‘model_file.pkl’)
    d. pickle.dump(X_test, open(‘test_data’, ‘wb’)) -- change this to use ‘with’ if you’ve got a moment
2. Upload your model to the server using scp:
    a. scp -i my_private_key.pem model_file.pkl ubuntu@your_ec2_ip_address:~
    b. This puts your model in the home directory on your server. Move it into the flask-ml-server/model directory
3. Edit flask-ml-server/utils/model_utils.py and change MODEL_FILE_NAME to refer to your new model file
4. Change the requirements.txt file to use the version of scikit-learn you used to build the model. You can check your version by running (from a Python interpreter/Notebook):
    a. import sklearn
    b. print(sklearn.__version__)
5. After you’ve updated the requirements.txt file, run pip install -r requirements.txt again to actually make the version change.

6. Start your server using python main.py as before
7. On your local machine, change get_predictions.py:
    a. Remove the train() function completely
    b. Write a load_data() function to load data from the test_data file you made earlier. (Use the pickle documentation to find out how to do this.) You're going to send this test data to the server as a JSON, so convert it to a list first using tolist(). You can't convert a numpy array directly into a JSON.
    c. Make the predict() function use the new data that's been loaded in, not the passenger data.
8. On the server, in main.py change the predict() function to work with the new model:
    a. Tip: Don't try to write all the code in one go! This is very unlikely to work. Instead, use print() liberally to make sure you know what's going on in each step.
    b. Type request.json to get the list of test data inputs for which you need to make predictions
    c. Change the model_utils.py's predict() function to take only a model and the input list as parameters
    d. In model_utils.py, add the following to the predict() function:
        i. input_array = np.array(input) (but first import numpy as np)
        ii. predictions = model.predict(input_array)
        iii. return predictions.tolist()
    e. In main.py's predict(), return the output of model_utils' predict() function using jsonify(predictions)
9. Finally, run python get_predictions.py on your local machine to get predictions from the model server using the test data you're sending it
10. Further tasks:
    a. Try creating a train endpoint to run on the server, assuming that the training data already exists on the server (the current example uses titanic.csv)
    b. Try adjusting the train_new endpoint to create a model based on training data sent from the client
    c. Make an update endpoint that updates the model based on new data as it arrives, without having to rebuild the model from scratch
    d. Think about how you'd make the interface for sending data for training/prediction nicer. Perhaps you could have the data added in a web form or upload a CSV file?