Chanin Nantasenamat Follow

May 27, 2021 · 11 min read ★ · Listen

Save

Created (with license) using the image by alexdndz from envato elements.

DATA SCIENCE | MACHINE LEARNING

How to Build a Machine Learning App in Python

Step-by-step tutorial from scratch in < 150 lines of Python code

Have you ever wished for a web app that would allow you to build a machine learning model automatically by simply uploading a CSV file? In this article, you will learn how to build your very own machine learning web app in Python in a little over 100 lines of code.

How to Build a Machine Learning App | Streamlit #13 @ Data Professor YouTube Channel

Importance of model deployment

Before diving further let's take a step back to look at the big picture. Data collection, data cleaning, exploratory data analysis, model construction, and model deployment are all part of the data science life cycle. The following is a summary infographic of the life cycle:

Data science lifecycle. (Drawn by Chanin Nantasenamat aka Data Professor)

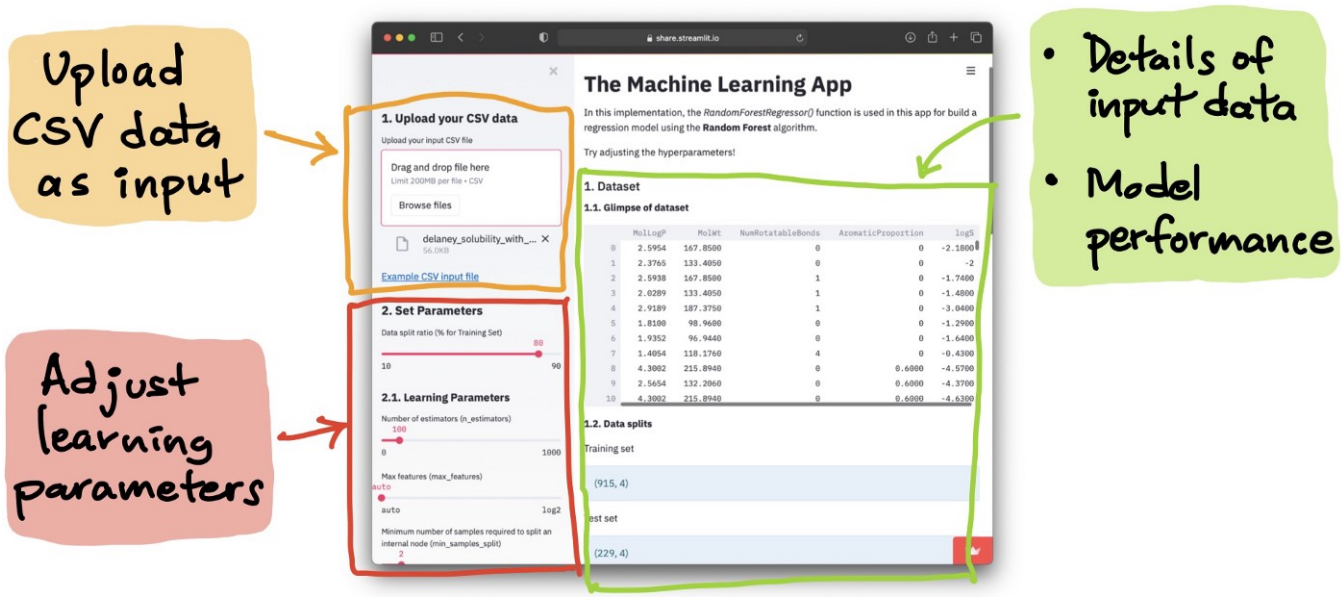
It is critical for us as Data Scientists or Machine Learning Engineers to be able to deploy our data science projects in order to complete the data science life cycle. The deployment of machine learning models using established frameworks such as Django or Flask may be a daunting and/or time-consuming task.

the random forest algorithm.

Now, let take a granular look at the details of what is happening at the front-end and back-end of the web app.

Front-end

Users are able to upload their own dataset as CSV file and they are also able to adjust the learning parameters (in the left panel) and upon adjustment of these parameters, a new machine learning model will be built and its model performance will then be displayed (in the right panel).



Anatomy of the machine learning app. The left panel accepts the input and the right panel displays the results.

Upload CSV data as input

The CSV file should have a header as the first line (containing the column names) followed by the dataset in subsequent lines (line 2 and beyond). We can see that below the upload box in the left panel there is a link to the example CSV file (*Example CSV input file*). Let's have a look at this example CSV file shown below:

Search this file…

	MolLogP	MolWt	NumRotatableBonds	AromaticProportion	logS
1					
2	2.5954000000000006	167.85	0.0	0.0	-2.18

delaney_solubility_with_descriptors.csv hosted with ♥ by GitHub

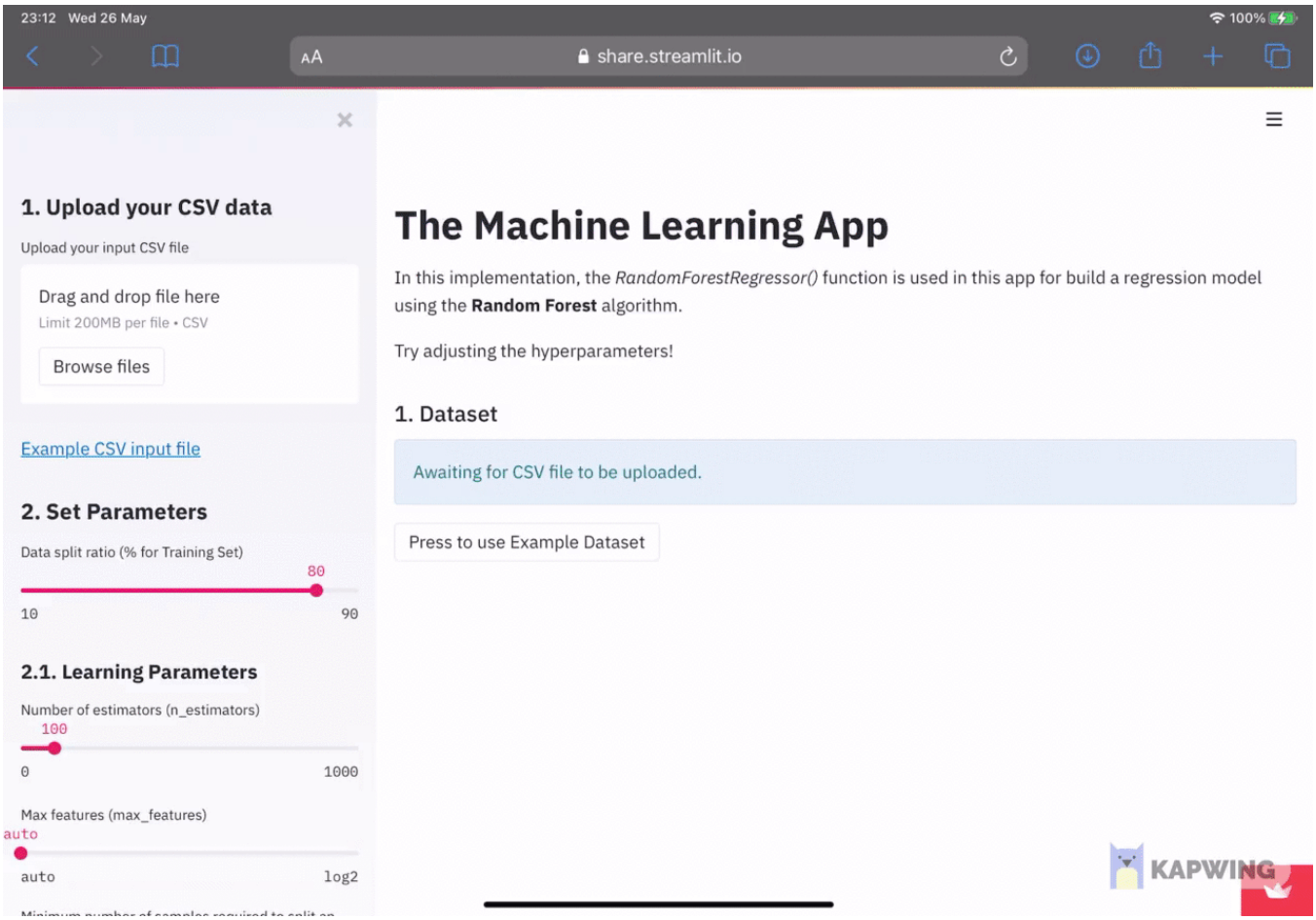
An excerpt of the first few lines from the Example CSV file linked in the app.

Adjust learning parameters

After uploading the CSV file, you should be able to see that a machine learning model has been built and its results are displayed in the right panel. It should be noted that the model is built using default parameters. The user can adjust learning parameters via the slider input and with each adjustment a new model will be built.

Model output (right panel)

In the right panel, we can see that the first data being shown is the input dataframe in the *1. Dataset* section and their results will be displayed in the *2. Model Performance* section. Finally, learning parameters used in the model building is provided in the *3. Model Parameters* section.



Model built upon uploading the input CSV file.

Back-end

Now, let's take a high-level look under the hood of the inner workings of the app.

Upon uploading the input CSV file, the contents of the file will be converted into a Pandas dataframe and assigned to the `df` variable. The dataframe will then be separated into the `X` and `y` variables in order to prepare it as input for Scikit-learn. Next, these 2 variables are used for data splitting using the user specified value in the left panel (by default it is using the 80/20 split ratio). Details on the data split dimension and the column names are printed out in the right panel of the app's front-end. A random forest model is then built using the major subset (80% subset) and the constructed model is applied to make predictions on the major (80%) and minor (20%) subsets. Model performance for this regression model is then reported into the right panel under the *2. Model Performance section*.

Tech Stacks used in this Tutorial

This will be carried out using just 3 Python libraries including Streamlit, Pandas and Scikit-learn.

Streamlit is a simple to use web framework that allows you to quickly implement a data-driven app in no time.

Pandas is a data structure tool that makes it possible to handle, manipulate and transform tabular datasets.

Scikit-learn is a powerful tool that provides users the ability to build machine learning models (i.e. that can perform various learning tasks including classification, regression and clustering) as well as coming equipped with example datasets and feature engineering capabilities.

Line-by-Line Explanation

The full code of this app is shown below. The code spans 131 lines and white spaces are added along with commented lines in order to make the code readable.

```
1  import streamlit as st
2  import pandas as pd
```

```
#---------------------------------#
# Page layout
## Page expands to full width
st.set_page_config(page_title='The Machine Learning App',
    layout='wide')

#---------------------------------#
# Model building
def build_model(df):
    X = df.iloc[:,:-1] # Using all column except for the last column as X
    Y = df.iloc[:,-1] # Selecting the last column as Y

    # Data splitting
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=(100-split_siz

    st.markdown('**1.2. Data splits**')
    st.write('Training set')
    st.info(X_train.shape)
    st.write('Test set')
    st.info(X_test.shape)

    st.markdown('**1.3. Variable details**:')
    st.write('X variable')
    st.info(list(X.columns))
    st.write('Y variable')
    st.info(Y.name)

    rf = RandomForestRegressor(n_estimators=parameter_n_estimators,
        random_state=parameter_random_state,
        max_features=parameter_max_features,
        criterion=parameter_criterion,
        min_samples_split=parameter_min_samples_split,
        min_samples_leaf=parameter_min_samples_leaf,
        bootstrap=parameter_bootstrap,
        oob_score=parameter_oob_score,
        n_jobs=parameter_n_jobs)
    rf.fit(X_train, Y_train)

    st.subheader('2. Model Performance')

    st.markdown('**2.1. Training set**')
    Y_pred_train = rf.predict(X_train)
```

```
    st.markdown('**2.2. Test set**')
    Y_pred_test = rf.predict(X_test)
    st.write('Coefficient of determination ($R^2$):')
    st.info( r2_score(Y_test, Y_pred_test) )

    st.write('Error (MSE or MAE):')
    st.info( mean_squared_error(Y_test, Y_pred_test) )

    st.subheader('3. Model Parameters')
    st.write(rf.get_params())

#---------------------------------#
st.write("""
# The Machine Learning App
In this implementation, the *RandomForestRegressor()* function is used in this app for
Try adjusting the hyperparameters!
""")

#---------------------------------#
# Sidebar - Collects user input features into dataframe
with st.sidebar.header('1. Upload your CSV data'):
    uploaded_file = st.sidebar.file_uploader("Upload your input CSV file", type=["csv"
    st.sidebar.markdown("""
[Example CSV input file](https://raw.githubusercontent.com/dataprofessor/data/master/d
""")

# Sidebar - Specify parameter settings
with st.sidebar.header('2. Set Parameters'):
    split_size = st.sidebar.slider('Data split ratio (% for Training Set)', 10, 90, 80

with st.sidebar.subheader('2.1. Learning Parameters'):
    parameter_n_estimators = st.sidebar.slider('Number of estimators (n_estimators)',
    parameter_max_features = st.sidebar.select_slider('Max features (max_features)', o
    parameter_min_samples_split = st.sidebar.slider('Minimum number of samples require
    parameter_min_samples_leaf = st.sidebar.slider('Minimum number of samples required

with st.sidebar.subheader('2.2. General Parameters'):
    parameter_random_state = st.sidebar.slider('Seed number (random_state)', 0, 1000,
    parameter_criterion = st.sidebar.select_slider('Performance measure (criterion)',
    parameter_bootstrap = st.sidebar.select_slider('Bootstrap samples when building tr
    parameter_oob_score = st.sidebar.select_slider('Whether to use out-of-bag samples
    parameter_n_jobs = st.sidebar.select_slider('Number of jobs to run in parallel (n_
```


```
103  st.subheader('1. Dataset')
104
105  if uploaded_file is not None:
106      df = pd.read_csv(uploaded_file)
107      st.markdown('**1.1. Glimpse of dataset**')
108      st.write(df)
109      build_model(df)
110  else:
111      st.info('Awaiting for CSV file to be uploaded.')
112      if st.button('Press to use Example Dataset'):
113          # Diabetes dataset
114          #diabetes = load_diabetes()
115          #X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
116          #Y = pd.Series(diabetes.target, name='response')
117          #df = pd.concat( [X,Y], axis=1 )
118
119          #st.markdown('The Diabetes dataset is used as the example.')
120          #st.write(df.head(5))
121
122          # Boston housing dataset
123          boston = load_boston()
124          X = pd.DataFrame(boston.data, columns=boston.feature_names)
125          Y = pd.Series(boston.target, name='response')
126          df = pd.concat( [X,Y], axis=1 )
127
128          st.markdown('The Boston housing dataset is used as the example.')
129          st.write(df.head(5))
```

functions from the scikit-learn library.

Lines 8–12

- **Lines 8–10:** Comments explaining about what Lines 11–12 are doing.
- **Lines 11–12:** The page title and its layout is set using the `st.set_page_config()` function. Here we can see that we are setting the `page_title` to `'The Machine Learning App'` while the `layout` is set to `'wide'` which will allow the contents of the app to fit the full width of the browser (i.e. otherwise by default the contents will be confined to a fixed width)


- **Line 16:** Here we are defining a custom function called `build_model()` and the statements below from Lines 17 onwards will dictate what this function will do
- **Lines 17–18:** The contents of the input dataframe stored in the `df` variable will be separated into 2 variables (`X` and `Y`). On line 17, all columns except for the last column will be assigned to the `X` variable while the last column will be assigned to the `Y` variable.
- **Lines 20–21:** Line 20 is a comment saying what Line 21 is doing, which is to use the `train_test_split()` function for performing data splitting of the input data (stored in `X` and `Y` variables). By default the data will split using a ratio of 80/20 whereby the 80% subset will be assigned to `X_train` and `Y_train` while the 20% subset will be assigned to `X_test` and `Y_test`.
- **Lines 23–27:** Line 23 prints `1.2. Data splits` as a bold text using Markdown syntax (i.e. here we can see that we are using the ** symbols before and after the phrase that we want to make the text to be bold as in `**1.2. Data splits**`. Next, we are going to print the data dimensions of the `X` and `Y` variables where Lines 24 and 26 will print out `Training set` and `Testing set` using the `st.write()` function while Lines 25 and 27 will print out the data dimensions using the `st.info()` function by appending `.shape` after `X_train` and `X_test` variables as in `X_train.shape` and `X_test.shape`, respectively. Note that the `st.info()` function will create a colored box around the variable output.
- **Lines 29–33:** In a similar fashion to the code block on Lines 23–27, this block will print out the X and Y variable names that are stored in `X.columns` and `Y.name`, respectively.
- **Lines 35–43:** The `RandomForestRegressor()` function will be used for build a regression model. The various input arguments for building the random forest model will use the user specified value from the left-hand panel of the app's front-end (in the back-end this corresponds to Lines 82–97).
- **Line 44:** The model will now be trained by using the `rf.fit()` function and as input argument we will be using `X_train` and `Y_train`.


- **Lines 48–54:** Line 48 prints the heading for `2. Training set` using the `st.markdown()` function. Line 49 applies the trained model to make a prediction on the training set using the `rd.predict()` function using `X_test` as the input argument. Line 50 prints the text of the performance metric to be printed for the Coefficient of determination (R2). Line 51 uses the `st.info()` function to print the R2 score via the `r2_score()` function by using `Y_train` and `Y_pred_train` (representing the actual Y values and predicted Y values for the training set) as input arguments. Line 53 uses the `st.write()` function to print the text of the next performance metric, which is the Error. Next, Line 54 uses the `st.info()` function to print the mean squared error value via the `mean_squared_error()` function by using `Y_train` and `Y_pred_train` as input arguments.

- **Lines 56–59:** This block of code performs exactly the same procedures but instead of the Training set it will perform it on the Test set. So instead of using the Training set data (`Y_train` and `Y_pred_train`) you would use the Test set data (`Y_test` and `Y_pred_test`).

- **Lines 64–65:** Line 64 prints the header `3. Model Parameters` by using the `st.subheader()` function.

Lines 67–75

The web app's title will be printed here. Lines 68 and 75 initiates and ends the use of the `st.write()` function to write the page's header in Mardown syntax. Line 69 uses the `#` symbol to make the text to be a Heading 1 size (according to the Markdown syntax). Lines 71 and 73 will then print a description about the web app.

Lines 78–100

- Several code blocks for the left sidebar panel is described here. Line 78 comments what the next several code blocks is about which is the Left sidebar panel for collecting user specified input.

- Lines 79–83 defines the CSV upload box. Line 79 prints `1. Upload your CSV data` as the header via the `st.sidebar.header()` function. Note here that we added `.sidebar` in between `st` and `header` in order to specify that this header should go

use to test out the app (here you can feel free to replace this with your own custom dataset in CSV file format).

- Lines 85–87 starts by commenting that the following code blocks will pertain to parameter settings for the random forest model. Line 86 then uses the `st.sidebar.header()` function to print `2. Set Parameters` as the header text. Finally, Line 87 creates a slider bar using the `st.sidebar.slider()` function where its input arguments specify `Data split ratio (% for Training Set)` as the text label for the slider while the 4 sets of numerical values (`10, 90, 80, 5`) represents the minimum value, maximum value, default value and the increment step size value. The minimum and maximum values are used to set the boundaries for the slider bar and we can see that the minimum value is 10 (shown at the far left of the slider bar) and the maximum value is 90 (shown at the far right of the slider bar). The default value of 80 will be used if the user does not adjust the slider bar. The increment step size will allow user to incrementally increase or decrease the slider value by a step size of 5 (e.g. 75, 80, 85, etc.)

- Lines 89–93 defines the various slider bars for the learning parameters in `2.1. Learning Parameters` in a similar fashion to what was described for Line 87. These parameters include `n_estimators` , `max_features` , `min_samples_split` and `min_samples_leaf` .

- Lines 95–100 defines the various slider bars for the general parameters in `2.2. General Parameters` in a similar fashion to what was described for Line 87. These parameters include `random_state` , `criterion` , `bootstrap` , `oob_score` and `n_jobs` .

Lines 102–103

Comments that the forthcoming blocks of code will print the model output into the main or right panel.

Lines 108–134

- Applies the if-else statement to detect whether the CSV file is uploaded or not. Upon loading the web app for the first time it will default to the `else` statement since no CSV file is yet uploaded. Upon loading a CSV file the `if` statement is activated

`to use Example Dataset` (which we will explain in a short moment what this button does).

- If the `if` statement (Lines 108–112) is activated, the uploaded CSV file (whose contents are contained within the `uploaded_file` variable) will be assigned to the `df` variable (Line 109). Next, the heading `1.1. Glimpse of dataset` is printed using the `st.markdown()` function (Line 110) followed by printing the dataframe content of the `df` variable (Line 111). Then, the dataframe contents in the `df` variable will be used as input argument to the `build_model()` custom function (i.e. described earlier in Lines 14–65) where the random forest model will be built and its model results will be displayed to the front-end.

Running the web app

Okay, so now that the web app has been coded. Let's proceed to running the web app.

Create the conda environment

Let's assume that you are starting from scratch, you will have to create a new conda environment (a good idea to ensure reproducibility of your code).

Firstly, create a new conda environment called `ml` as follows in a terminal command line:

```
conda create -n ml python=3.7.9
```

Secondly, we will login to the `ml` environment

```
conda activate ml
```

Install prerequisite libraries

Firstly, download the requirements.txt file

Secondly, install the libraries as shown below

```
pip install -r requirements.txt
```

Download machine learning web app files

Now, download the web app files hosted on the GitHub repo of the Data Professor or use the 134 lines of code found above.

```
wget https://github.com/dataprofessor/ml-app/archive/main.zip
```

Then unzip the contents

```
unzip main.zip
```

Change into the `main` directory

```
cd main
```

Now that you're in the `main` directory you should be able to see the `ml-app.py` file.

Launching the web app

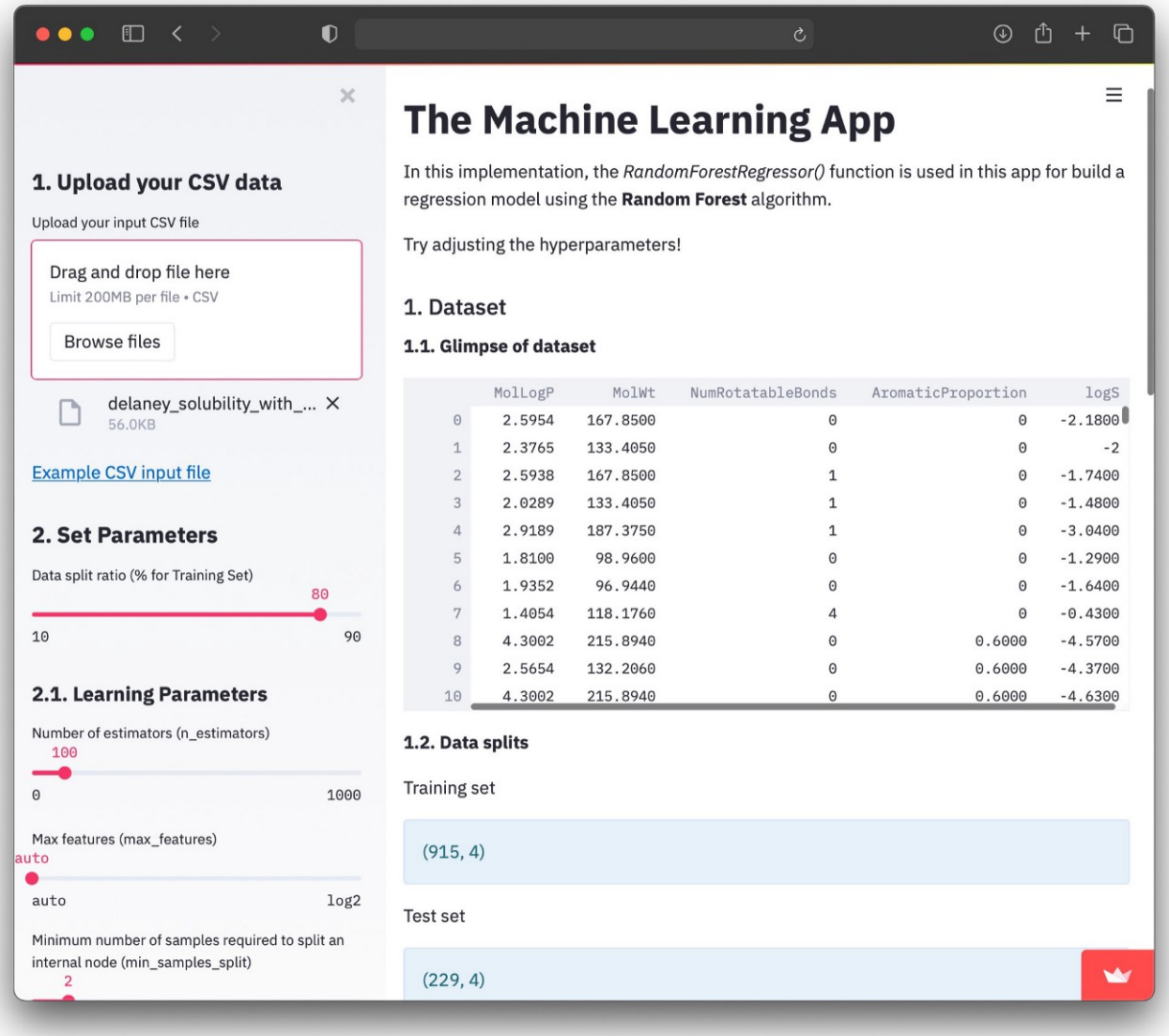
To launch the app, type the following into a terminal command line (i.e. also make sure that the `ml-app.py` file is in the current working directory):

```
streamlit run ml-app.py
```

In a few moments you will see the following message in the terminal prompt.


```
  Local URL: http://localhost:8501
  Network URL: http://10.0.0.11:8501
```

Finally, a browser pops up and you will see the app.



Screenshot of the machine learning web app.

Congratulations, you have now created the machine learning web app!

What Next?

To make your web app public and available to the world, you can deploy it to the internet. I've created a YouTube video showing how you can do that on *Heroku* and

- How to Deploy Data Science Web App to Streamlit Sharing

Subscribe to my Mailing List for my best updates (and occasionally freebies) in Data Science!

About Me

I work full-time as an Associate Professor of Bioinformatics and Head of Data Mining and Biomedical Informatics at a Research University in Thailand. In my after work hours, I'm a YouTuber (AKA the Data Professor) making online videos about data science. In all tutorial videos that I make, I also share Jupyter notebooks on GitHub (Data Professor GitHub page).

Data Professor

Data Science, Machine Learning, Bioinformatics, Research and Teaching are my passion. The Data Professor YouTube...

www.youtube.com

Connect with Me on Social Network

✅ YouTube: http://youtube.com/dataprofessor/
✅ Website: http://dataprofessor.org/ (Under construction)
✅ LinkedIn: https://www.linkedin.com/company/dataprofessor/
✅ Twitter: https://twitter.com/thedataprof
✅ FaceBook: http://facebook.com/dataprofessor/
✅ GitHub: https://github.com/dataprofessor/
✅ Instagram: https://www.instagram.com/data.professor/