

# 1. Prepare the problem

The first step in a project is to simply define the problem at hand. One need to understand the situation and the problem which needs to be solved.

## 1.1. Load libraries

As the project ML model is sticking to Python, The very first step is to load or Import most probable libraries and the packages required to get the results

Some very primary and almost necessary packages for Machine Learning are as below:

- **NumPy**

It is a well known general-purpose array-processing package. An extensive collection of high complexity mathematical functions make NumPy powerful to process large multi-dimensional arrays and matrices. NumPy is very useful for handling linear algebra, Fourier transforms, and random numbers

- **SciKit**

It has a wide range of supervised and unsupervised learning algorithms that works on a consistent interface in Python. The library can also be used for data-mining and data analysis. The main machine learning functions that the Scikit-learn library can handle are classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

- **Pandas**

It are turning up to be the most popular Python library that is used for data analysis with support for fast, flexible, and expressive data structures designed to work on both "relational" or "labeled" data. Pandas today is an inevitable library for solving practical, real-world data analysis in Python. Pandas is highly stable, providing highly optimized performance.

- **Matplotlib**

Matplotlib is a data visualization library that is used for 2D plotting to produce publication-quality image plots and figures in a variety of formats. The library helps to generate histograms, plots, error charts, scatter plots, bar charts with just a few lines of code.

- **Keras**

Keras works with neural-network building blocks like layers, objectives, activation functions, and optimizers. Keras also have a bunch of features to work on images and text images that comes handy when writing Deep Neural Network code. Apart from the standard neural network, Keras supports convolutional and recurrent neural networks.

- **TensorFlow**

TensorFlow is a popular computational framework for creating machine learning models. TensorFlow supports a variety of different toolkits for constructing models at varying levels of abstraction. TensorFlow exposes a very stable Python and C++ APIs. It can expose, backward compatible APIs for other languages too, but they might be unstable. TensorFlow has a flexible architecture with which it can run on a variety of computational platforms CPUs, GPUs, and TPUs.

- **Pickle**

Python pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects into byte streams is called pickling or serialization or flattening or marshalling. We can convert the byte stream back into python objects by a process called as unpickling.

- **cURL**

cURL is used in command lines or scripts to transfer data. It is also used in cars, television sets, routers, printers, audio equipment, mobile phones, tablets, settop boxes, media players and is the internet transfer backbone for thousands of software applications affecting *billions of humans* daily.

### 1.2.1. Load Dataset

Once the libraries are loaded, you need to get the data loaded. Pandas has a very straightforward function to perform this task – [pandas.read\\_csv](#). The `read.csv` function is not just limited to csv files, but also can read other text based files as well. Other formats can also be read using pandas read functions like html, json, pickled files etc

```
In [1]: import pandas as pd

In [2]: data = pd.read_csv('T1.csv')
      ...: data.head()
Out[2]:
```

	Date/Time	...	Wind Direction (°)
0	01 01 2018 00:00	...	259.994904
1	01 01 2018 00:10	...	268.641113
2	01 01 2018 00:20	...	272.564789
3	01 01 2018 00:30	...	271.258087
4	01 01 2018 00:40	...	265.674286

```
[5 rows x 5 columns]
```

## 2. Preprocessing Data

Pre-processing refers to the transformations applied to our **data** before feeding it to the algorithm. **Data Preprocessing** is a technique that is used to convert the raw **data** into a clean **data** set.

### 2.1. Data Cleaning

Real life data is not arranged and presented nicely and in a dataframe with no abnormalities. Data usually has a lot of abnormalities like missing values, a lot of features with incorrect format, features on different scales, etc. So to overcome these anomalies we use data cleaning process.

So in proposed project we are using following in-built functions:

- **DataFrame.isna()** this function is used to detect missing values.
- **Dataframe.dropna()** this function is used to drop missing value instance. If dataset is large enough one can use this function.
- **Dataframe.fillna()** this function is used to fill missing value instance with statistical answer (e.g. mean, mode, median).

Since dataset is pretty good, we didn't get any anomalies. So project only need isna() function and output is as below :

```
In [3]: data.isna()
Out[3]:
```

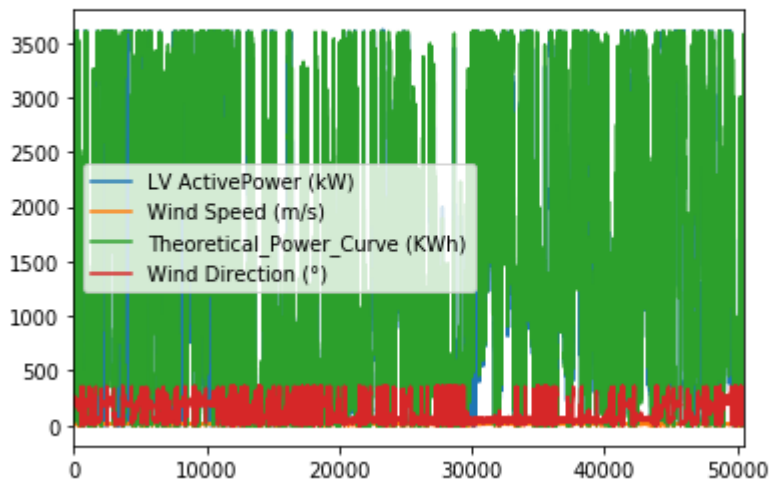
	Date/Time	...	Wind Direction (°)		...	...	...
0	False	...	False	50500	False	...	False
1	False	...	False	50501	False	...	False
2	False	...	False	50502	False	...	False
3	False	...	False	50503	False	...	False
4	False	...	False	50504	False	...	False
5	False	...	False	50505	False	...	False
6	False	...	False	50506	False	...	False
7	False	...	False	50507	False	...	False
8	False	...	False	50508	False	...	False
9	False	...	False	50509	False	...	False
10	False	...	False	50510	False	...	False
11	False	...	False	50511	False	...	False
12	False	...	False	50512	False	...	False
13	False	...	False	50513	False	...	False
14	False	...	False	50514	False	...	False
15	False	...	False	50515	False	...	False
16	False	...	False	50516	False	...	False
17	False	...	False	50517	False	...	False
18	False	...	False	50518	False	...	False
19	False	...	False	50519	False	...	False
20	False	...	False	50520	False	...	False
21	False	...	False	50521	False	...	False
22	False	...	False	50522	False	...	False
23	False	...	False	50523	False	...	False
24	False	...	False	50524	False	...	False
25	False	...	False	50525	False	...	False
26	False	...	False	50526	False	...	False
27	False	...	False	50527	False	...	False
28	False	...	False	50528	False	...	False
29	False	...	False	50529	False	...	False
...	...	...	...	...	...	...	...
50500	False	...	False	...	...	...	...
50501	False	...	False	...	...	...	...

[50530 rows x 5 columns]

## 2.2. Data Visualizations

Data Visualizations are very important as they are the quickest way to know the data and the patterns. Data may have thousands of features and even more instances. It is not possible to analyze the numeric data for all of them. To do so one can rely on functions of Matplotlib library.

```
In [10]: data.plot()  
...: pyplot.show()
```



So we can conclude here dataset is with high variance and low bias .

## 2.3 Feature Engineering

All the features might not be in their best form. Meaning — they can be transformed onto a different scale by using a set of functions. This is in order to increase the correlation with the target and hence the more accuracy score. Some of these transformations are related to scaling, like StandardScaler, Normalizer, MinMaxScaler etc.

As project consist of only 5 columns so without performing any feature engineering related task in external tool like Weka, one with domain knowledge can conclude that all four features are playing important role to predict LV Active Power as a result.

So following features are considered for Model training:

- Date/Time
- Wind Speed (m/s)
- Theoretical\_Power\_Curve (KWh)
- Wind Direction (°)

### 3. Evaluate Algorithms

Once your data is ready, proceed to check the performance of the various regression/classification algorithms (based on the type of problem) You can first make a base model to set a benchmark to compare against.

#### 3.1 Split-out validation dataset

Once the model is trained, it needs to be validated as well to see if it really generalized the data or if it over/under fitted. The data in hand can be split up beforehand as a training set and validation set. This split-out has various techniques – Train Test Split, Shuffle split etc. One can also run Cross Validation on the entire data set for a more robust validation. KFold Cross Validation, Leave-One-Out-CV are the most popular method:

- **sklearn.model\_selection.TimeSeriesSplit():**

Provides train/test indices to split time series data samples that are observed at fixed time intervals, in train/test sets. In each split, test indices must be higher than before, and thus shuffling in cross validator is inappropriate. This cross-validation object is a variation of KFold. In the kth split, it returns first k folds as train set and the (k+1)th fold as test set.

- The data is loaded and stored in dataframe known as "data".
- As the dataset had only one Label data known as "LV ActivePower (kW)" so it is stored in dataframe named as "label" and the column is dropped from the data.
- Both the DataFrames are converted into nparray, as the TimeSeriesSplit() requires nparray as the parameter.
- As TimeSeriesSplit contains parameter `max_train_size` , *number of splits*. Here the number of splits are 5 so the dataset is divided into 80%, for training and 20% is used for testing.
- Following is the code snippet with the output .

```

In [2]: data = pd.read_csv('T1.csv')
...: label = data['LV ActivePower (kW)']
...: data = data.drop(['LV ActivePower (kW)'],axis=1)
...: data=np.array(data)
...: label=np.array(label)

In [3]: tscv = TimeSeriesSplit()
...: print(tscv)
TimeSeriesSplit(max_train_size=None, n_splits=5)

In [4]: X_train=np.array([])
...: X_test=np.array([])
...: y_train=np.array([])
...: y_test=np.array([])

In [5]: for train_index, test_index in tscv.split(data):
...:     print("TRAIN:", train_index, "TEST:", test_index)
...:     X_train, X_test = data[train_index], data[test_index]
...:     y_train, y_test = label[train_index], label[test_index]
TRAIN: [ 0  1  2 ... 8422 8423 8424] TEST: [ 8425 8426 8427 ... 16843 16844 16845]
TRAIN: [ 0  1  2 ... 16843 16844 16845] TEST: [16846 16847 16848 ... 25264 25265 25266]
TRAIN: [ 0  1  2 ... 25264 25265 25266] TEST: [25267 25268 25269 ... 33685 33686 33687]
TRAIN: [ 0  1  2 ... 33685 33686 33687] TEST: [33688 33689 33690 ... 42106 42107 42108]
TRAIN: [ 0  1  2 ... 42106 42107 42108] TEST: [42109 42110 42111 ... 50527 50528 50529]

In [6]:

```

## 3.2 Comapre Algorithms

- Once you have spot run the *test harness*, you can easily see which ones performed the best for your data. The algorithms giving consistently high scores should be your target. You can then take the top ones and tune them further to improve their performance.
- After comparing all the algorithms we encoutered that LSTM is most suitable algorithm for proposed project where experiments performed as below:

### LSTM model configuration:

Input Batch Size : 1

Epochs : 7

No. of Neurons : 10

Look Backs/lag : 24

### Exp 1: 12 hours

12 hours data was predicted with the same model configurations. And a Mean percent error of 8.28% is observed.

### Exp 2: 24 hours

24 hours data was predicted and a Mean percentage error of 11.53% was observed. This is brilliant for a day ahead prediction and from the predicted value plot is seen

that LSTM is able to find the pattern in a day's data and the same pattern which was in expected data is predicted.

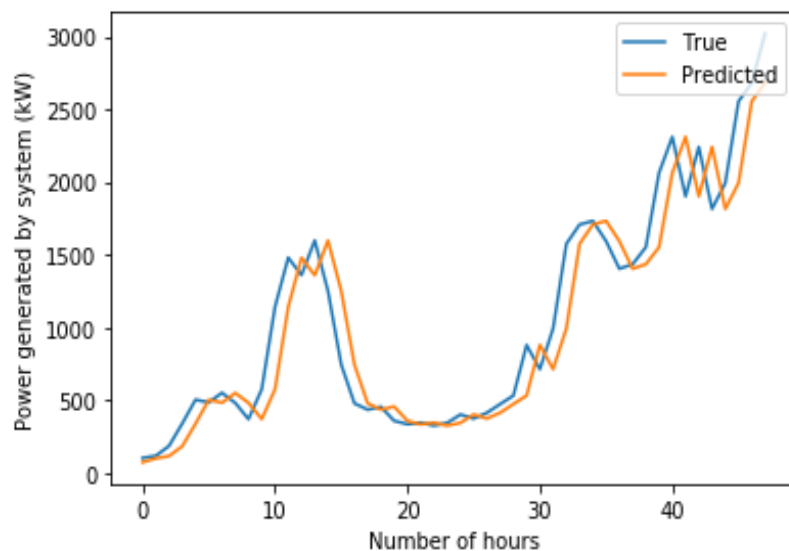
### Exp 3: 48 hours

48 hours of data was predicted and a Mean percentage error of 16.47% was observed. This was great for a prediction for wind power generation which is dependent on highly non-linear parameter as Wind Speed.

### Exp 4: 72 hours

72 hours of data was predicted and a Mean percentage error of 20.70% was observed.

```
In [22]: pyplot.plot(raw_values[-predict_values_exp:], label="True")
...: pyplot.plot(predictions, label="Predicted")
...: pyplot.legend(loc='upper right')
...: pyplot.xlabel("Number of hours")
...: pyplot.ylabel("Power generated by system (kW)")
...: pyplot.show()
```





## 4. Improve Accuracy

After you have the best performing algorithms with you, their parameters and the Hyperparameters can be tuned to give maximum results.

### 4.1 Algorithm Tuning

- Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm.
- This can be achieved by methods like Grid Search and Random Search.

Where for this project we can use following variables as Hyperparameter :

**batch\_size\_exp** = 1

**epoch\_exp** = 7

**neurons\_exp** = 10

**predict\_values\_exp** = 72

**lag\_exp** = 24

### 4.2 Ensembles

- Multiple Machine Learning algorithms can be combined to make a more robust and optimal model that gives better predictions than the single algorithm. This is known as an ensemble.
- There are basically 2 types of ensembles — **Bagging** (Bootstrap-Aggregating) and **Boosting**.
- So without directly relying on Ensembles one can add various models sequentially. In proposed project code snippet is as follow :

```
88|
89 model = Sequential()
90 model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
91 model.add(Dense(1))
92
```

This will help to *increase* accuracy of the model.

## 5.Finalize Model

A Final machine learning model is one trained on all available data and is then used to make predictions on new data.

### 5.1. Predictions on validation dataset

When you have got an optimum performing model with best hyperparameters and ensembles, you can validate it on the unseen test dataset.

- **Create standalone model on entire training dataset**

Once validated, run the model on the entire dataset once to make sure no data points are missed while training/testing. Now, your model is at its optimal position.

- **Save the model for later use**

Once you have the accurate model with you, you still need to save and load it to make it available in future when it is needed. The most common method to get this done is Pickle.

After performing both the step metric of project which is Mean Absolute Percent Error can be checked as follows :

```
In [7]: expectations = np.array(expectations)
...: predictions = np.array(predictions)
...: print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations - predictions) / expectations))*100))
Mean Absolute Percent Error: 20.699783214547328
```

Hour=1, Predicted=-3.303418, Expected=0.000000	Hour=36, Predicted=1139.938230, Expected=1478.978027
Hour=2, Predicted=-2.246128, Expected=0.000000	Hour=37, Predicted=1478.896090, Expected=1359.036987
Hour=3, Predicted=-1.135515, Expected=0.000000	Hour=38, Predicted=1358.955891, Expected=1597.265991
Hour=4, Predicted=-0.487662, Expected=0.000000	Hour=39, Predicted=1597.183409, Expected=1250.109985
Hour=5, Predicted=-0.218322, Expected=0.000000	Hour=40, Predicted=1250.028685, Expected=744.182373
Hour=6, Predicted=-0.098249, Expected=0.000000	Hour=41, Predicted=744.103934, Expected=476.473907
Hour=7, Predicted=0.030016, Expected=0.000000	Hour=42, Predicted=476.396484, Expected=433.640991
Hour=8, Predicted=0.014155, Expected=0.000000	Hour=43, Predicted=433.564903, Expected=453.358185
Hour=9, Predicted=0.102254, Expected=0.000000	Hour=44, Predicted=453.283425, Expected=356.442688
Hour=10, Predicted=0.102234, Expected=0.000000	Hour=45, Predicted=356.370012, Expected=333.740814
Hour=11, Predicted=0.018742, Expected=0.000000	Hour=46, Predicted=333.668969, Expected=343.783997
Hour=12, Predicted=-0.033766, Expected=0.000000	Hour=47, Predicted=343.711577, Expected=323.725311
Hour=13, Predicted=-0.023083, Expected=0.000000	Hour=48, Predicted=323.652789, Expected=341.905487
Hour=14, Predicted=-0.001576, Expected=0.000000	Hour=49, Predicted=341.833064, Expected=399.653809
Hour=15, Predicted=0.004343, Expected=0.000000	Hour=50, Predicted=399.581994, Expected=372.079590
Hour=16, Predicted=0.057448, Expected=0.000000	Hour=51, Predicted=372.008239, Expected=411.729187
Hour=17, Predicted=0.052618, Expected=0.000000	Hour=52, Predicted=411.657986, Expected=471.419708
Hour=18, Predicted=-0.001600, Expected=0.000000	Hour=53, Predicted=471.348291, Expected=529.938904
Hour=19, Predicted=0.014689, Expected=0.000000	Hour=54, Predicted=529.867136, Expected=877.915283
Hour=20, Predicted=0.002170, Expected=0.000000	Hour=55, Predicted=877.843221, Expected=711.683228
Hour=21, Predicted=-0.014443, Expected=0.000000	Hour=56, Predicted=711.610735, Expected=992.034119
Hour=22, Predicted=-0.052332, Expected=0.000000	Hour=57, Predicted=991.961209, Expected=1574.197998
Hour=23, Predicted=-0.055800, Expected=69.737373	Hour=58, Predicted=1574.124651, Expected=1706.860962
Hour=24, Predicted=69.736221, Expected=74.726463	Hour=59, Predicted=1706.787241, Expected=1731.552979
Hour=25, Predicted=74.714347, Expected=100.829201	Hour=60, Predicted=1731.478997, Expected=1592.900024
Hour=26, Predicted=100.793288, Expected=115.665703	Hour=61, Predicted=1592.825912, Expected=1403.121948
Hour=27, Predicted=115.598018, Expected=183.744995	Hour=62, Predicted=1403.047836, Expected=1433.415039
Hour=28, Predicted=183.656641, Expected=338.591187	Hour=63, Predicted=1433.340867, Expected=1550.709961
Hour=29, Predicted=338.493007, Expected=501.613190	Hour=64, Predicted=1550.635702, Expected=2062.549072
Hour=30, Predicted=501.511999, Expected=483.559998	Hour=65, Predicted=2062.474763, Expected=2309.875000
Hour=31, Predicted=483.459862, Expected=548.965698	Hour=66, Predicted=2309.800654, Expected=1901.602051
Hour=32, Predicted=548.863880, Expected=479.424286	Hour=67, Predicted=1901.527685, Expected=2240.629883
Hour=33, Predicted=479.325776, Expected=368.269409	Hour=68, Predicted=2240.555484, Expected=1814.355957
Hour=34, Predicted=368.176285, Expected=573.746521	Hour=69, Predicted=1814.281555, Expected=1992.348999
Hour=35, Predicted=573.658915, Expected=1140.021973	Hour=70, Predicted=1992.274630, Expected=2554.377930
Hour=36, Predicted=1139.938230, Expected=1478.978027	Hour=71, Predicted=2554.303614, Expected=2681.270020
	Hour=72, Predicted=2681.195761, Expected=3019.893066

## 5.2. Converting to Flask API

To maintain consistency, accuracy and reusability of ML model we have to convert the ML model flask API.

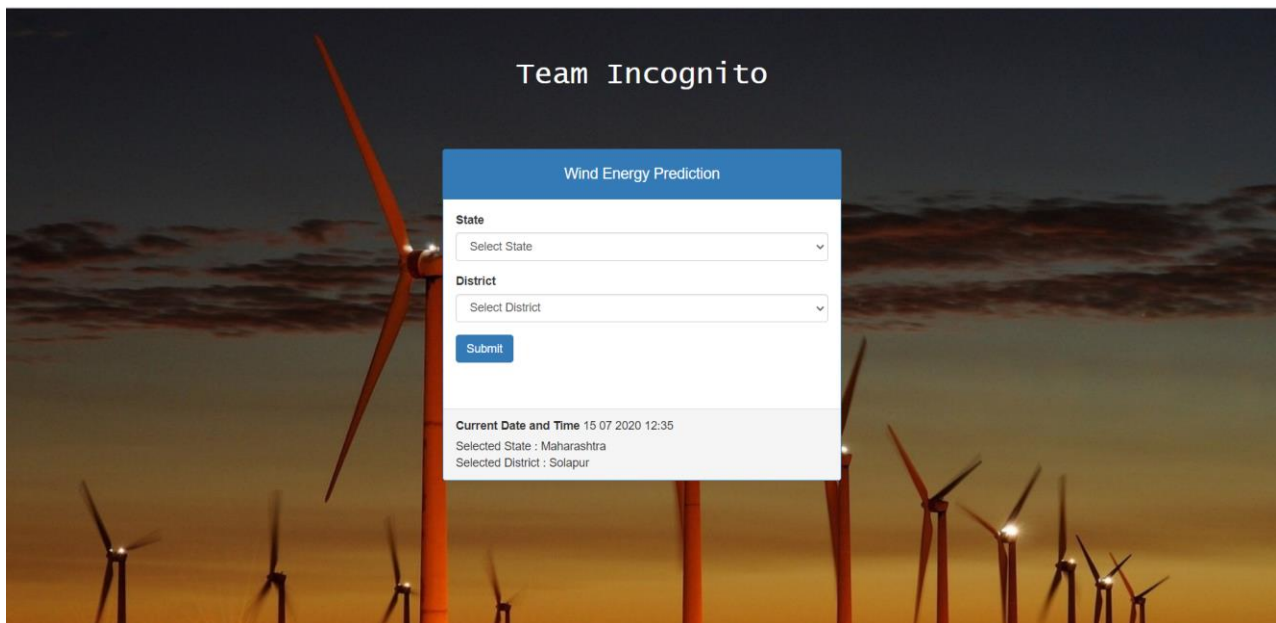
```
In [16]: pickle.dump(stkclf, open('IBM.pkl','wb'))
...: model = pickle.load(open('IBM.pkl','rb'))
```

## 6. Front-End Design and Integration

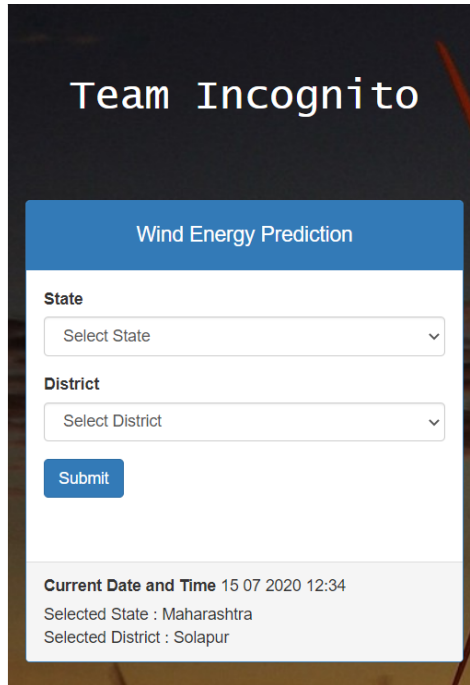
### 6.1 UI Design

- For proposed project one can build responsive webpage which will take a location as a input in state and district format.
- Along with this PHP will fetch current system time and date in backend.

#### ► Dekstop View-



## ► Mobile View-



The image shows a mobile application interface for "Wind Energy Prediction". At the top, the text "Team Incognito" is displayed in white on a dark background. Below this is a blue header bar with the title "Wind Energy Prediction" in white. The main form area is white and contains two dropdown menus: "State" with the placeholder "Select State" and "District" with the placeholder "Select District". A blue "Submit" button is located below the dropdowns. At the bottom of the form, a grey box displays the following information: "Current Date and Time 15 07 2020 12:34", "Selected State : Maharashtra", and "Selected District : Solapur".

## 6.2 Integrating Project

- From any location one can fetch weather information from any weather API(e.g. AccuWeather).
- After parsing received JSON file, one can get wind related information and then by using standard formula can generate theoretical power output. So in this manner input to ML model is ready from frontend.
- By using cURL library of PHP we can provide input to the Flask API and receive output