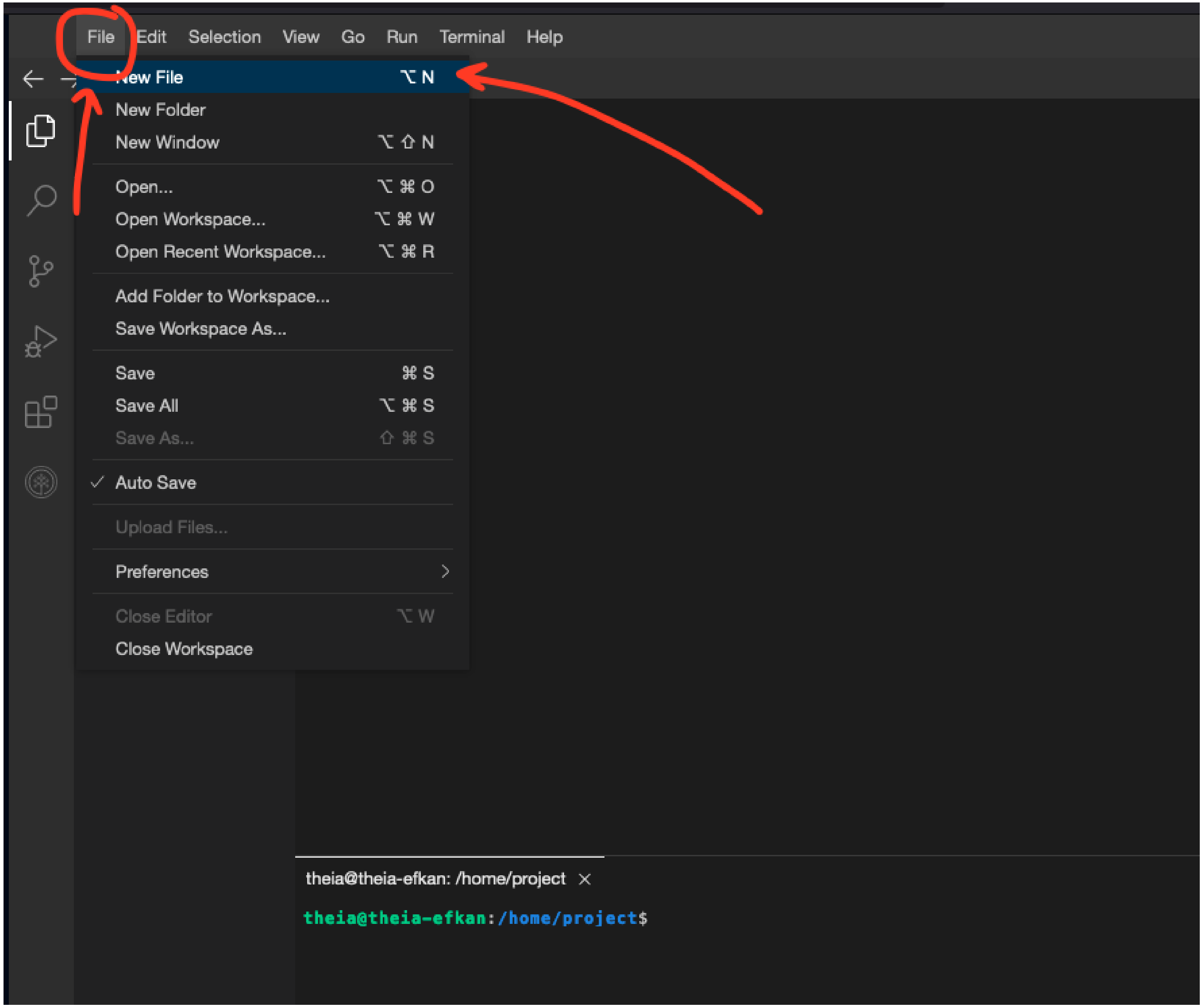
Vectorizing Sample Data

Now that we've had our introduction, let's vectorize some data.

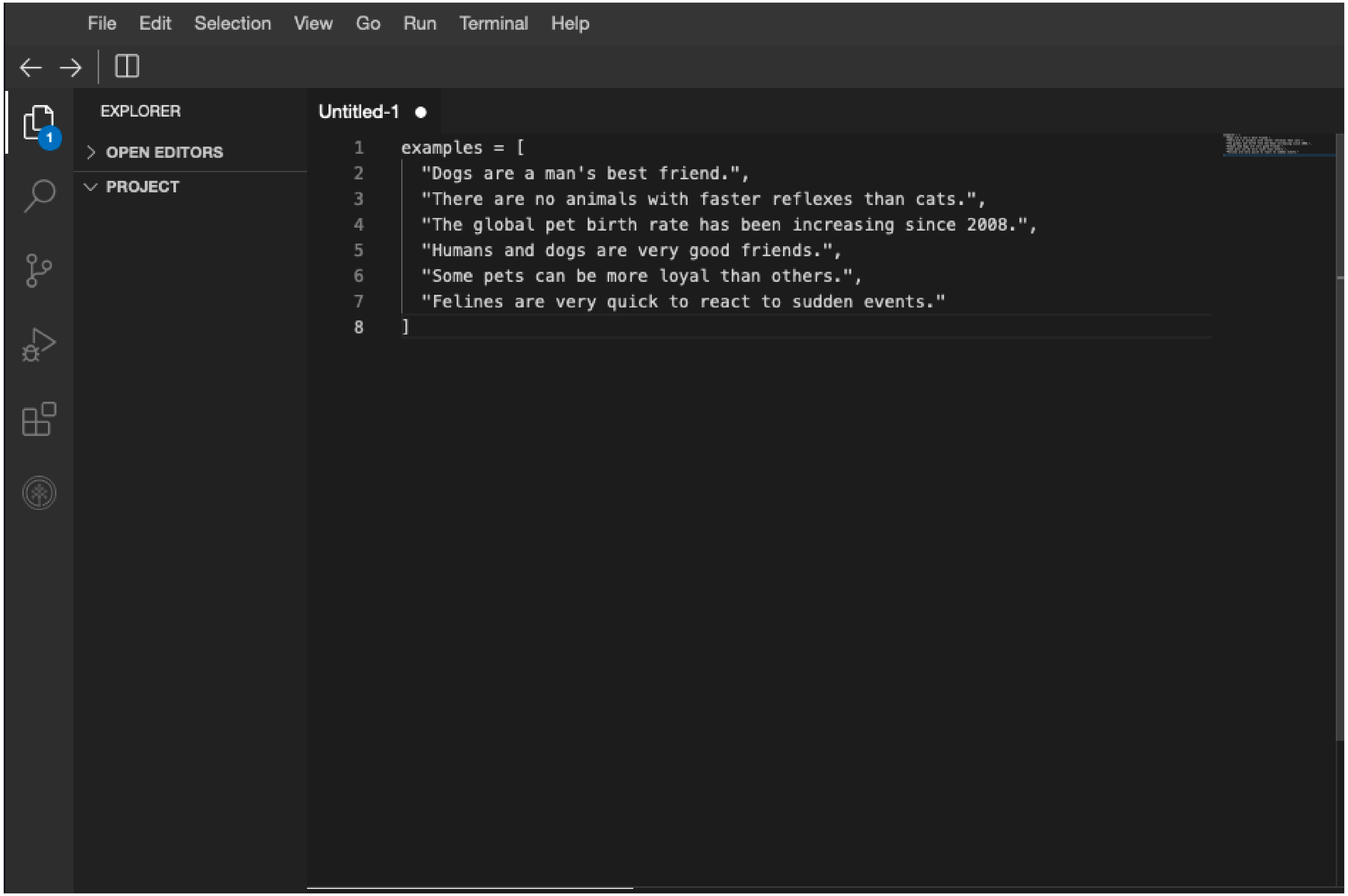
First off, let's create a file called `demo.py` in our IDE. We can do this by clicking "File -> New File" from the horizontal menu at the top.



```
theia@theia-efkan:/home/project$
```

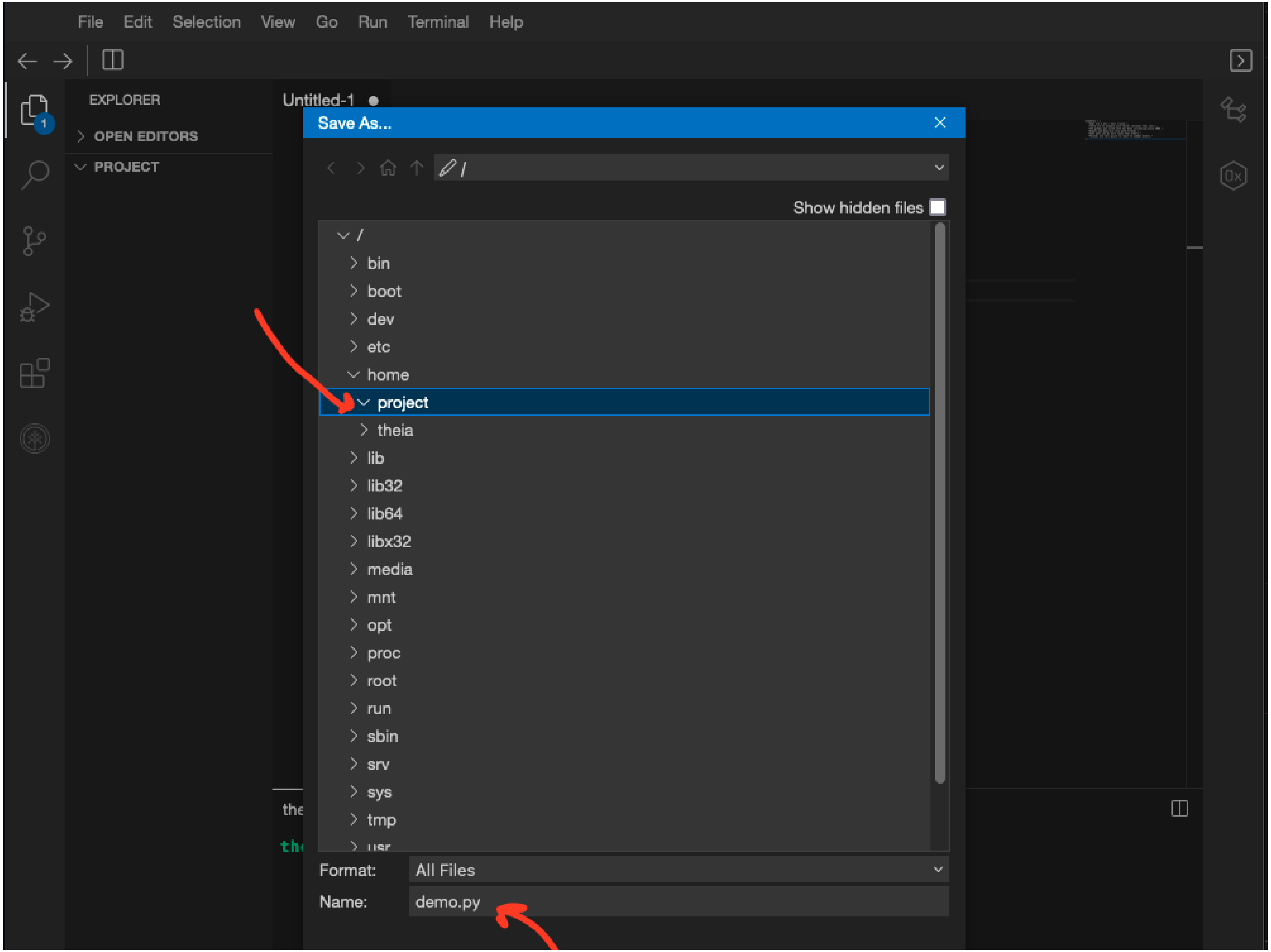
An empty document labelled Untitled-1 should appear in our editor. Inside, let’s add our example sentences:

```
examples = [
  "Dogs are a man's best friend.",
  "There are no animals with faster reflexes than cats.",
  "The global pet birth rate has been increasing since 2008.",
  "Humans and dogs are very good friends.",
  "Some pets can be more loyal than others.",
  "Felines are very quick to react to sudden events."
]
```



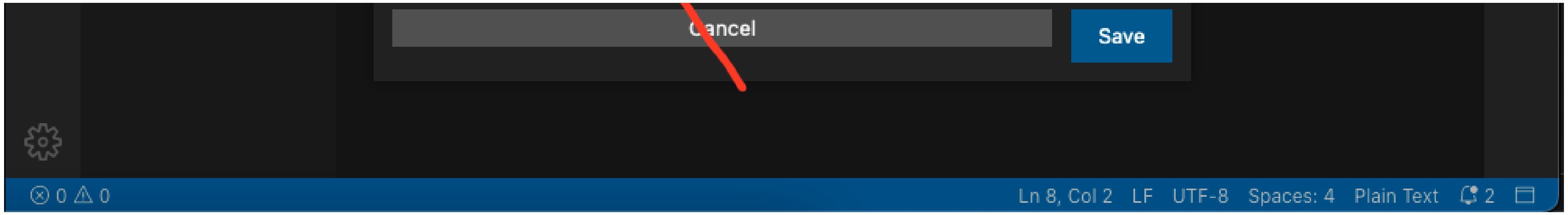
```
examples = [
  "Dogs are a man's best friend.",
  "There are no animals with faster reflexes than cats.",
  "The global pet birth rate has been increasing since 2008.",
  "Humans and dogs are very good friends.",
  "Some pets can be more loyal than others.",
  "Felines are very quick to react to sudden events."
]
```

Then, let's save our document by clicking "File -> Save" from the top menu. When the Save dialogue appears, select the `/home/project` folder by clicking on `home`, and then `project`. Name the file `demo.py` and the bottom, and hit save.

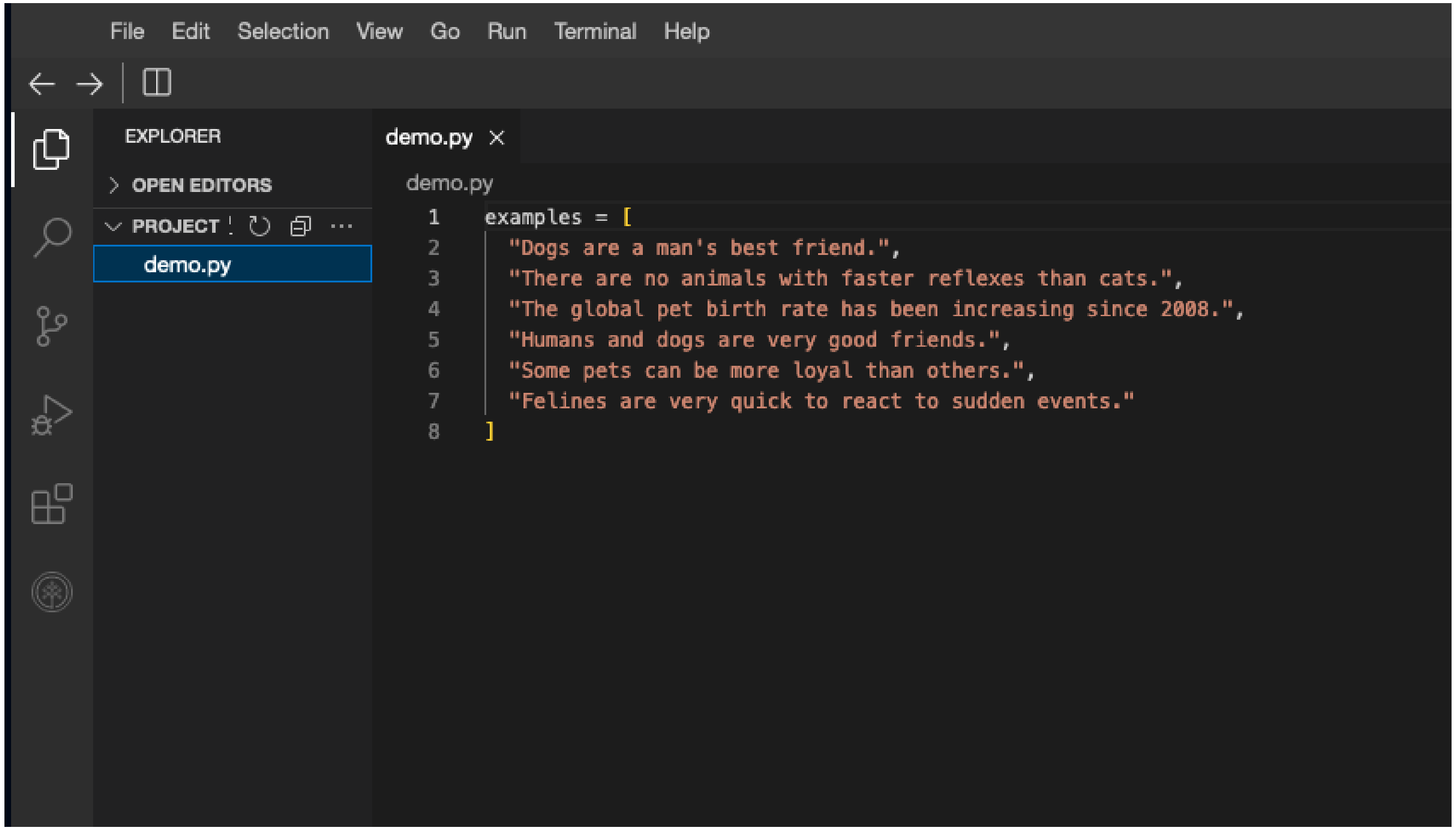


Format: All Files

Name: demo.py



After doing so, you should see `demo.py` in your explorer on the left.



```python
examples = [
    "Dogs are a man's best friend.",
    "There are no animals with faster reflexes than cats.",
    "The global pet birth rate has been increasing since 2008.",
    "Humans and dogs are very good friends.",
    "Some pets can be more loyal than others.",
    "Felines are very quick to react to sudden events."
]
```

Great! Now let’s see what these sentences look like when converted to vector embeddings.

To use the Hugging Face inference API, you must create an account and obtain an access token.
After doing so, we specify the endpoint and model.

```
import requests

model_id = "sentence-transformers/all-MiniLM-L6-v2"
hf_token = "get your token in http://hf.co/settings/tokens"

api_url = f"https://api-inference.huggingface.co/pipeline/feature-extraction/{model_id}"
headers = {"Authorization": f"Bearer {hf_token}"}
```

Next, let's make a request to the inference API with our sample data!

```
response = requests.post(api_url, headers=headers, json={"inputs": examples, "options":{"wait_for_model":True}})
embeddings = response.json()

print(embeddings)

print("Number of lists returned:", len(embeddings))
print("Length of each list:", len(embeddings))
```

To run the code, type `python3.11 demo.py` in the IDE terminal.

Now, the output will likely be an enormous spew of numbers. Structurally, however, it's very simple: the output is a list of lists. There are 5 lists returned, and the length of each should be 384 (or something similar).

Each list represents the **vector** corresponding to the sentence.

And there we have it! We've just vectorized our sample sentences.

Now, the idea here is that sentences that are "similar", as determined by the machine learning model, will have close integers in many cells in there vector embeddings.

Consequently, sentences that are not similar to each other are expected to have much different values across their respective vectors.

This allows for a very easy way of determining “how similar” two sentences are: just take the dot product of their vectors! The greater the dot product, the more similar the sentences are!

Dot Product

$$\vec{a} = (a_1, a_2, a_3)$$

$$\vec{b} = (b_1, b_2, b_3)$$

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Here’s a very simple way to explain the dot product of two vectors:

IF you have two vectors `a` and `b` such that `a = [a1, a2, a3]` and `b = [b1, b2, b3]`

THEN the dot product of `a` and `b` is equal to `[a1*b1, a2*b2, a3*b3]`

Since we’ll be using the dot product operation in vector databases, let’s add a dot product implementation to our Python script:

```python
def dot_product(A, B):
    return sum(i[0] * i[1] for i in zip(A, B))
```

Previous