

[Get unlimited access](#)[Open in app](#)

Published in Towards Data Science

This is your **last** free member-only story this month. [Upgrade for unlimited access.](#)



Senthil E

[Follow](#)

May 31 · 8 min read · ✨ · 🎧 Listen



Save

Photo by [Lucas Favre](#) on [Unsplash](#)

LineaPy Data Science Workflow In Just Two Lines: MLOps Made Easy

Data engineering, simplified





Get unlimited access

Open in app

LineaPy is a Python package for capturing, analyzing, and automating data science workflows. At a high level, LineaPy traces the sequence of code execution to form a comprehensive understanding of the code and its context. This understanding allows LineaPy to provide a set of tools that help data scientists bring their work to production more quickly and easily, with just two lines of code.

I saw their announcement last week about LineaPy. This is originated from UC Berkeley research like Apache Spark and now open-sourced. I tried LineaPy and it looks very interesting and useful. If you want to know more about LineaPy then please continue reading.

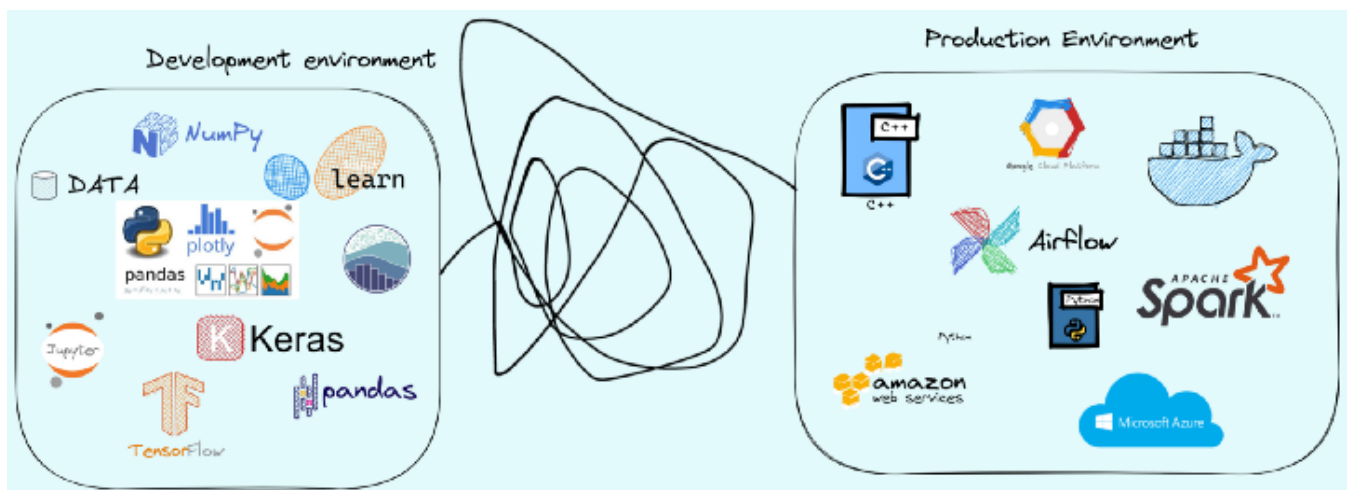


Image by the author (using Excalidraw and Excalidraw stickers)

Table of Contents:

1. Why do we need LineaPy?
2. LineaPy Installation.
3. Concepts.
- 4.2 Lines of Code.
5. Walkthrough LineaPy Example.
6. Conclusion

I have used the following tools to create the diagrams and code snippets.

->Excalidraw

->Gitmind





Get unlimited access

Open in app

Why do we need LineaPy:

The process of data science development to production is a complex engineering process. An article in VentureBeat that about 90% of data science projects don't make it to production. In short only one out of ten projects makes it to production. It is easy to write a messy code in the Jupyter notebook since you will be doing a lot of EDA, statistical analysis, editing the cells, deleting the cells, etc. Also maintaining the notebook clean and in the sequence is time-consuming and needs a lot of effort. The refactoring of the data science development code and building the pipelines are complex, manual, and time-consuming. LineaPy provides just 2 lines of code to take the development code to the production code and also generate the required pipeline.

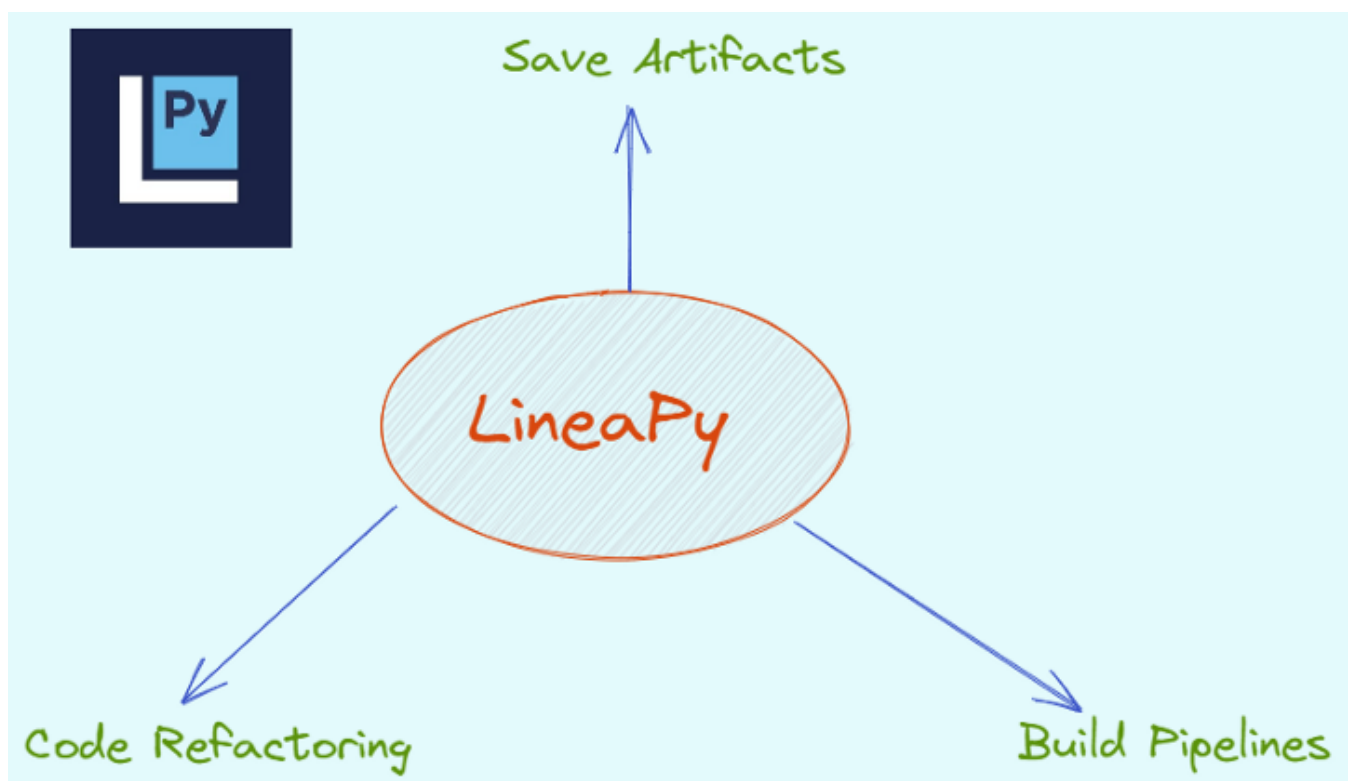


Image by the author

LineaPy Installation:

```
pip install lineapy[minimal]
```

```
#or
```



[Get unlimited access](#)[Open in app](#)

#or

```
pip install lineapy[postgres]
```

Image by the author

Concepts:

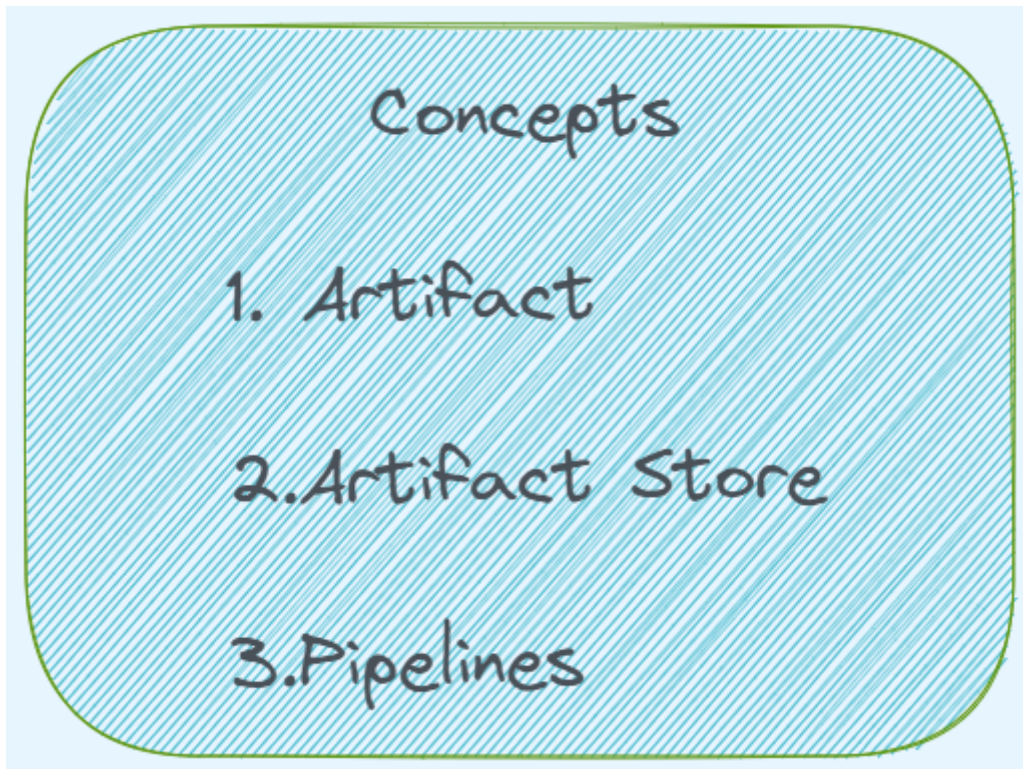


Image by the author

Artifact:

- Artifact refers to an intermediate result in the data science development process.
- In the data science workflow, an artifact can be a model, a chart, a statistic, a dataframe, or a feature function.
- LineaPy treats the artifact as code and a value. It stores the value of the artifact and also the necessary code to derive the artifact.





Get unlimited access

Open in app

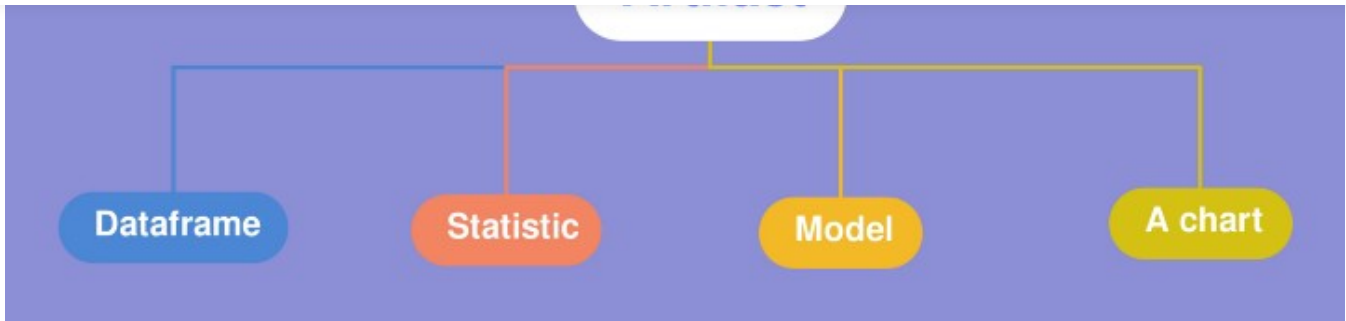


Image by the author

Artifact Store:

- Artifacts are stored in the artifact store.
- The artifact store also saves the metadata of the artifact like creation time and version, etc.
- The artifact store is globally accessible by everyone. The user can view, load, and build on artifacts across different development sessions and even different projects.

Pipeline:

- A pipeline refers to a series of steps that transform data into useful information/product.

For example below is a pipeline

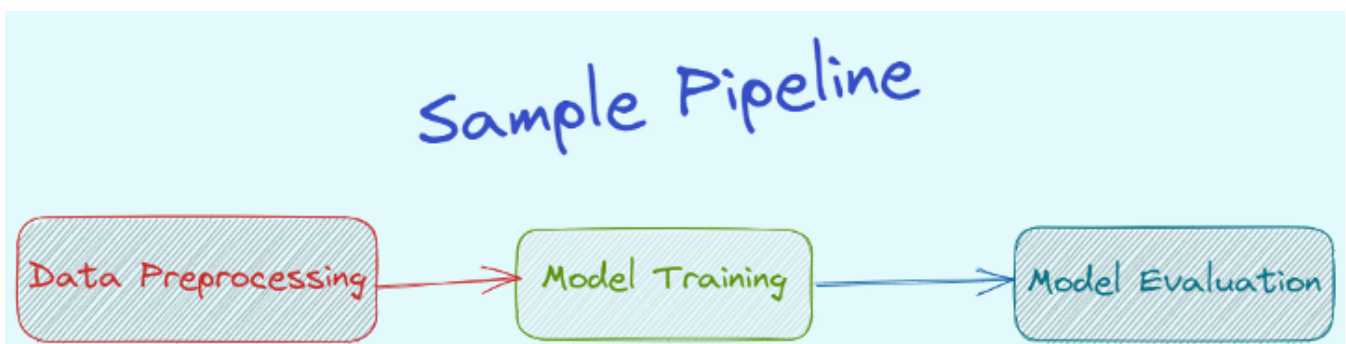


Image by the author

- These pipelines are developed each component at a time and then later all the



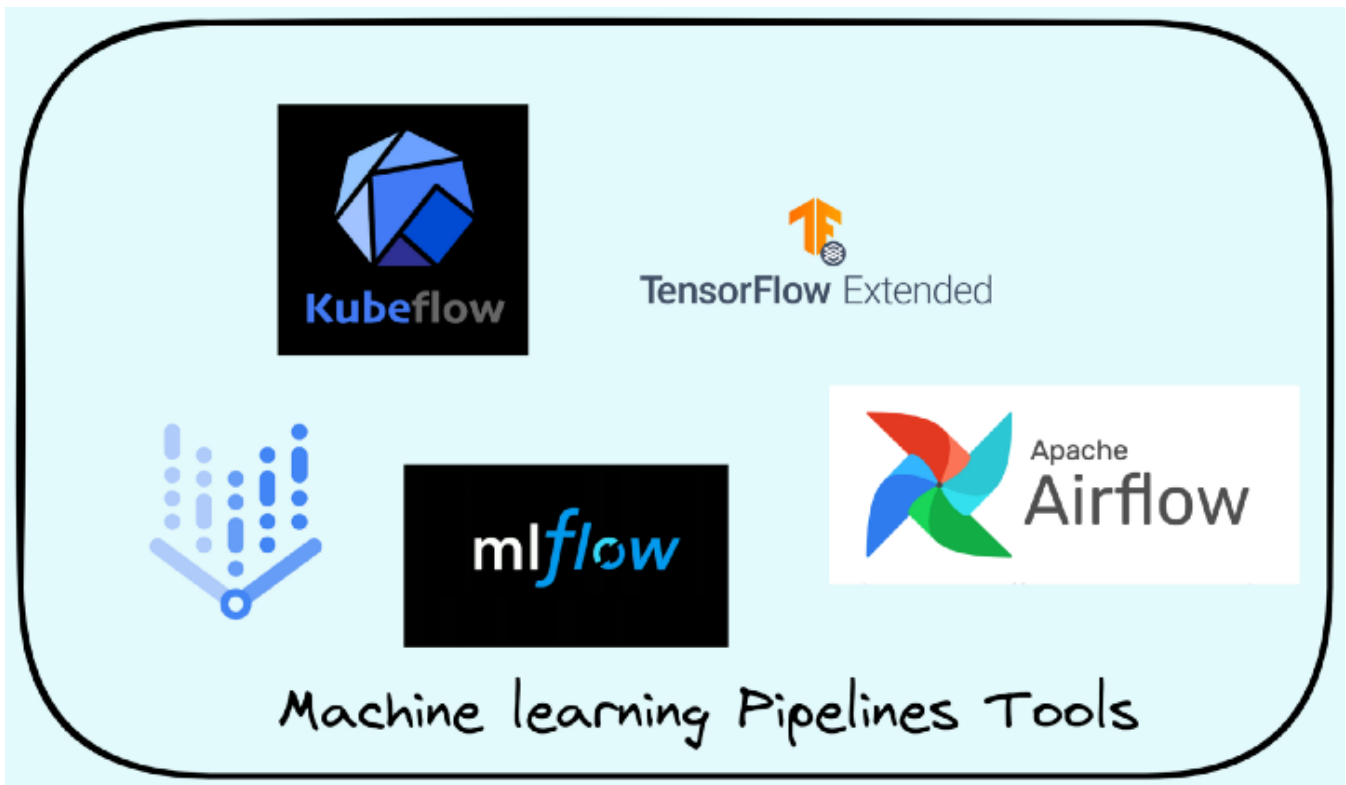
[Get unlimited access](#)[Open in app](#)

Image by the author

2 Lines of Code:

```
#save() & get()  
  
Import LineaPy  
  
lineapy.save  
lineapy.get  
lineapy.to_pipeline
```





Get unlimited access

Open in app

lineapy.save API creates LineaArtifacts and saves them to the database. The code to save the artifact is

```
# Step 1
#Store the variable as an artifact
saved_artifact = lineapy.save(model, "my_model_name")
```

- The method requires two arguments: the variable to save and the string name to save it as. It returns the saved artifact.
- **LineaArtifact** the object has two key APIs:
- **.get_value()** returns value of the artifact, e.g., an integer or a dataframe
- **.get_code()** returns minimal essential code to create the value

The code is

```
# Step2:Check the value of the artifact
print(saved_artifact.get_value())

# Check minimal essential code to generate the artifact
print(saved_artifact.get_code())

#Check the session code
saved_artifact.get_session_code()

#To check the saved artifacts
lineapy.catalog()

#Get version info of the retrieved artifact
desired_version = saved_artifact.version

# Check the version info
print(desired_version)
print(type(desired_version))
```





Get unlimited access

Open in app

```
# Step 1
#Store the variable as an artifact
saved_artifact = lineapy.save(model, "my_model_name")

# Step2:Check the value of the artifact
saved_artifact.get_value()
```

Image by the author

Walkthrough LineaPy Example:

About the dataset: The dataset is the Auto MPG dataset available at

UCI Machine learning Repository

Center for Machine Learning and Intelligent Systems

Source: This dataset was taken from the StatLib library maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.

Dataset: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: the University of California, School of Information and Computer Science.

Dataset features:

1. *mpg: continuous*
2. *cylinders: multi-valued discrete*
3. *displacement: continuous*
4. *horsepower: continuous*
5. *weight: continuous*
6. *acceleration: continuous*
7. *model year: multi-valued discrete*
8. *origin: multi-valued discrete*
9. *car name: string (unique for each instance)*





Get unlimited access

Open in app

attributes like cylinders, displacement, horsepower, and weight.

The workflow followed in my example is below. The main goal is how to save the artifact, get the value/code and generate the pipeline.

1. Load the train and test data into a pandas dataframe
2. EDA & Statistical Analysis.
3. Save the final training and test data as an artifact-Use save()
4. Check the artifact-get()
5. Build the model using different methods.
6. Choose the best model.
7. Save the best model as an artifact-save()
8. Display the artifact catalog-catalog()
9. Build the pipeline using the saved artifacts.

To practice locally please use Google Colab. I used **Google Colab**. Again the main objective of this article is to demo the usage of the LineaPy package and again not worried much about the model building or performance of the model. Please check the Tensorflow tutorial in the reference for the more details. Here the goal is to look at lineapy functionality.

Declare all the necessary packages.

```
1 %%capture
2 import IPython
3
4 if (IPython.version_info[0] < 7):
5     !pip -q install ipython --upgrade
6     # To load the updated ipython that we have just installed,
7     # we need to restart the runtime. The exit() command allows
8     # us to stop the current runtime, and executing the cell after
9     # it would restart the runtime.
```





Get unlimited access

Open in app

```
15     # Use seaborn for pairplot.
16     !pip install -q seaborn
17
18
19     %%capture
20     !pip -q install lineapy
21
22     %load_ext lineapy
23
24     import os
25     import lineapy
26
27
28     import matplotlib.pyplot as plt
29     import numpy as np
30     import pandas as pd
31     import seaborn as sns
32
33     # Make NumPy printouts easier to read.
34     np.set_printoptions(precision=3, suppress=True)
35
36     import tensorflow as tf
37
38     from tensorflow import keras
39     from tensorflow.keras import layers
40
41     print(tf.__version__)
```

Image by the author

Download the dataset and upload the data into a pandas dataframe.

```
1 url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'
2 column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
3                 'Acceleration', 'Model Year', 'Origin']
4
5 raw_dataset = pd.read_csv(url, names=column_names,
6                           na_values='?', comment='\t',
7                           sep=' ', skipinitialspace=True)
```

lp2.py hosted with ❤ by GitHub

[view raw](#)

Image by the author





Get unlimited access

Open in app

```
1 dataset = raw_dataset.copy()
2 dataset.tail()
3
4 dataset.isna().sum()
5 #drop na
6 dataset = dataset.dropna()
7
8 dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
9 #one hot encoding
10 dataset = pd.get_dummies(dataset, columns=['Origin'], prefix='', prefix_sep='')
11 dataset.tail()
12
13 train_dataset = dataset.sample(frac=0.8, random_state=0)
14 test_dataset = dataset.drop(train_dataset.index)
15
16 sns.pairplot(train_dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']], diag_kind='kde')
17
18 train_dataset.describe().transpose()
19
20 #normalization
21 train_dataset.describe().transpose()[['mean', 'std']]
22
23 normalizer = tf.keras.layers.Normalization(axis=-1)
24 normalizer.adapt(np.array(train_features))
25
26 print(normalizer.mean.numpy())
27
28 first = np.array(train_features[:1])
29
30 with np.printoptions(precision=2, suppress=True):
31     print('First example:', first)
32     print()
33     print('Normalized:', normalizer(first).numpy())
34
35
```

Image by the author

Now save the train and test data dataframes as artifacts-save()

```
1 # Store the variable as an artifact
```





Get unlimited access

Open in app


```
7 # Store the variable as an artifact
8 train_labels = lineapy.save(train_labels, "train_labels")
9
10 # Check object type
11 print(type(train_labels))
12
13 # Store the variable as an artifact
14 test_artifact = lineapy.save(test_features, "test_data")
15
16 # Check object type
17 print(type(test_artifact))
18
19 # Store the variable as an artifact
20 test_labels = lineapy.save(test_labels, "test_labels")
21
22 # Check object type
23 print(type(test_labels))
```

In4py hosted with ❤️ by GitHub

[view raw](#)

Image by the author

For example



```
# Store the variable as an artifact
train_artifact = lineapy.save(train_features,
                              "train_data")

# Check object type
print(type(train_artifact))
```

Image by the author

The type of the artifact is





Get unlimited access

Open in app



```
# Check the value of the artifact
print(train_artifact.get_value())

# Check minimal essential code to generate the
artifact
print(train_artifact.get_code())
```

Image by the author

The output of the saved artifact-train dataframe and the code.

	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	\
146	4	90.0	75.0	2125.0	14.5	74	
282	4	140.0	88.0	2890.0	17.3	79	
69	8	350.0	160.0	4456.0	13.5	72	
378	4	105.0	63.0	2125.0	14.7	82	
331	4	97.0	67.0	2145.0	18.0	80	
..	
281	6	200.0	85.0	2990.0	18.2	79	
229	8	400.0	180.0	4220.0	11.1	77	
150	4	108.0	93.0	2391.0	15.5	74	
145	4	83.0	61.0	2003.0	19.0	74	
182	4	107.0	86.0	2464.0	15.5	76	
	Europe	Japan	USA				
146	0	0	1				
282	0	0	1				

Image by the author

All the EDA code and unnecessary code are dropped.





Get unlimited access

Open in app

import pandas as pd

```
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
column_names = [
    "MPG",
    "Cylinders",
    "Displacement",
    "Horsepower",
    "Weight",
    "Acceleration",
    "Model Year",
    "Origin",
]
raw_dataset = pd.read_csv(
    url, names=column_names, na_values="?", comment="\t", sep=" ", skipinitialspace=True
)
dataset = raw_dataset.copy()
dataset = dataset.dropna()
dataset["Origin"] = dataset["Origin"].map({1: "USA", 2: "Europe", 3: "Japan"})
dataset = pd.get_dummies(dataset, columns=["Origin"], prefix="", prefix_sep="")
train_dataset = dataset.sample(frac=0.8, random_state=0)
train_features = train_dataset.copy()
train_labels = train_features.pop("MPG")
```

Image by the author

To display the original session code use the `get_session`



```
from lineapy.utils.utils import prettify
print(test_artifact.get_session_code())
```

Image by the author

The above `get_session_code` will show the entire code which has all the eda code, etc.

Now build the models and choose the best model. You can use the artifacts stored





Get unlimited access

Open in app

```
1  #Linear regression Model
2  linear_model = tf.keras.Sequential([
3      normalizer,
4      layers.Dense(units=1)
5  ])
6
7  linear_model.predict(train_features1[:10])
8
9  linear_model.layers[1].kernel
10
11 linear_model.compile(
12     optimizer=tf.optimizers.Adam(learning_rate=0.1),
13     loss='mean_absolute_error')
14
15 history = linear_model.fit(
16     train_features1,
17     train_labels1,
18     epochs=100,
19     # Suppress logging.
20     verbose=0,
21     # Calculate validation results on 20% of the training data.
22     validation_split = 0.2)
23
24 def plot_loss(history):
25     plt.plot(history.history['loss'], label='loss')
26     plt.plot(history.history['val_loss'], label='val_loss')
27     plt.ylim([0, 10])
28     plt.xlabel('Epoch')
29     plt.ylabel('Error [MPG]')
30     plt.legend()
31     plt.grid(True)
32
33 plot_loss(history)
34
35 test_results = {}
36 test_results['linear_model'] = linear_model.evaluate(
37     test_features1, test_labels1, verbose=0)
```

Image by the author

2nd Model-DNN Model:





Get unlimited access

Open in app

```
5     layers.Dense(64, activation='relu'),
6     layers.Dense(64, activation='relu'),
7     layers.Dense(1)
8 ])
9
10 model.compile(loss='mean_absolute_error',
11               optimizer=tf.keras.optimizers.Adam(0.001))
12 return model
13
14 dnn_model = build_and_compile_model(normalizer)
15 dnn_model.summary()
16
17
18 history = dnn_model.fit(
19     train_features1,
20     train_labels1,
21     validation_split=0.2,
22     verbose=0, epochs=100)
23
24 def plot_loss(history):
25     plt.plot(history.history['loss'], label='loss')
26     plt.plot(history.history['val_loss'], label='val_loss')
27     plt.ylim([0, 10])
28     plt.xlabel('Epoch')
29     plt.ylabel('Error [MPG]')
30     plt.legend()
31     plt.grid(True)
32
33 plot_loss(history)
34
35 test_results['dnn_model'] = dnn_model.evaluate(test_features1, test_labels1, verbose=0)
36
37
38
39
```

Image by the author

Results:

```
1 pd.DataFrame(test_results, index=['Mean absolute error [MPG]']).T
```

lp12.py hosted with ❤ by GitHub

[view raw](#)



Get unlimited access

Open in app



Mean absolute error [MPG]



linear_model	2.512473
dnn_model	1.689494

Image by the author

DNN model is the better model than the linear regression model. I will save the DNN model.

Now save the DNN model.

```
1 model_artifact = lineapy.save(dnn_model, 'dnn_model')
2
3 #Assets written to: ram://044e58dc-eeb9-42b8-ab3a-b99c11c66927/assets
```

ip8.py hosted with ❤ by GitHub

[view raw](#)

Image by the author

To display the code of the model, use the **get()** method.



```
from lineapy.utils.utils import prettify
print(prettify(model_artifact.get_code(use_lineapy_serialization=False)))
```

Image by the author

The below is the code generated when you use the **get_code** for the **model_artifact**.





Get unlimited access

Open in app

```
5 import tensorflow as tf
6 from lineapy.utils.utils import prettify
7 from tensorflow import keras
8 from tensorflow.keras import layers
9
10 normalizer = tf.keras.layers.Normalization(axis=-1)
11 train_features1 = pickle.load(open("/root/.lineapy/linea_pickles/yzU2NL1", "rb"))
12 train_labels1 = pickle.load(open("/root/.lineapy/linea_pickles/GPSupKi", "rb"))
13
14
15 def build_and_compile_model(norm):
16     model = keras.Sequential(
17         [
18             norm,
19             layers.Dense(64, activation="relu"),
20             layers.Dense(64, activation="relu"),
21             layers.Dense(1),
22         ]
23     )
24
25     model.compile(loss="mean_absolute_error", optimizer=tf.keras.optimizers.Adam(0.001))
26     return model
27
28
29 dnn_model = build_and_compile_model(normalizer)
30 history = dnn_model.fit(
31     train_features1, train_labels1, validation_split=0.2, verbose=0, epochs=100
32 )
```

Image by the author

List all the saved artifacts.

```
1 # List all saved artifacts
2 lineapy.catalog()
```

lp13.py hosted with ❤ by GitHub

[view raw](#)

Image by the author

The output-displays all the artifacts stored



[Get unlimited access](#)[Open in app](#)

```
test_data:0 created on 2022-05-28 02:07:14.520631
test_labels:0 created on 2022-05-28 02:07:20.777316
dnn_model:0 created on 2022-05-28 02:15:23.632722
```

Now you can also build a data pipeline with the artifacts you have saved. The code to generate the pipeline are

- Get the artifact and assign it to a variable.

```
1 train_art = lineapy.get("train_data")
2 train_art
3 #LineaArtifact(name='train_data', _version=0)
4 y_art = lineapy.get("train_labels")
5 y_art
6 #LineaArtifact(name='train_labels', _version=0)
7 model_art = lineapy.get("dnn_model")
8 model_art
9 #LineaArtifact(name='dnn_model', _version=0)
```

lp6.py hosted with ❤ by GitHub

[view raw](#)

Image by the author

Now build the data pipeline.

- Pre-processed data.
- Model building

```
1 import os
2 directory = lineapy.to_pipeline(
3     [train_art.name, y_art.name, model_art.name],
4     framework = 'AIRFLOW',
5     pipeline_name = "dnn_pipeline",
6     dependencies = {'dnn_pipeline_dnn_model': {'dnn_pipeline_train_data', 'dnn_pipeline_y'}},
7     output_dir = os.environ.get("AIRFLOW_HOME", "~/airflow") + "/dags")
```

lp5.py hosted with ❤ by GitHub

[view raw](#)

Image by the author

... is the list of artifact names to be used for the pipeline. Here we are





Get unlimited access

Open in app

- **dependencies** is the dependency graph among artifacts
- If artifact A depends on artifacts B and C, then the graph is specified as `{ A: { B, C } }`
- If A depends on B and B depends on C, then the graph is specified as `{ A: { B }, B: { C } }`
- **output_dir** is the location to put the files for running the pipeline
- **framework** is the name of the orchestration framework to use
- LineaPy currently supports **"AIRFLOW"** and **"SCRIPT"**
- If **"AIRFLOW"**, it will generate files that can run Airflow DAGs. You can execute the file in airflow CLI.
- If **"SCRIPT"**, it will generate files that can run the pipeline as a Python script

Running `lineapy.to_pipeline()` generates several files that can be used to execute the pipeline from the UI or CLI. The following files are generated

The following files are generated

```
<pipeline_name>.py  
<pipeline_name>_dag.py  
<pipeline_name>_requirements.txt  
<pipeline_name>_Dockerfile
```

Image by the author

In this case, the pipeline_name is `titanic_pipeline`.



[Get unlimited access](#)[Open in app](#)

- dnn_pipeline.py
- dnn_pipeline_Dockerfile
- dnn_pipeline_dag.py
- dnn_pipeline_requirements.txt

Image by the author

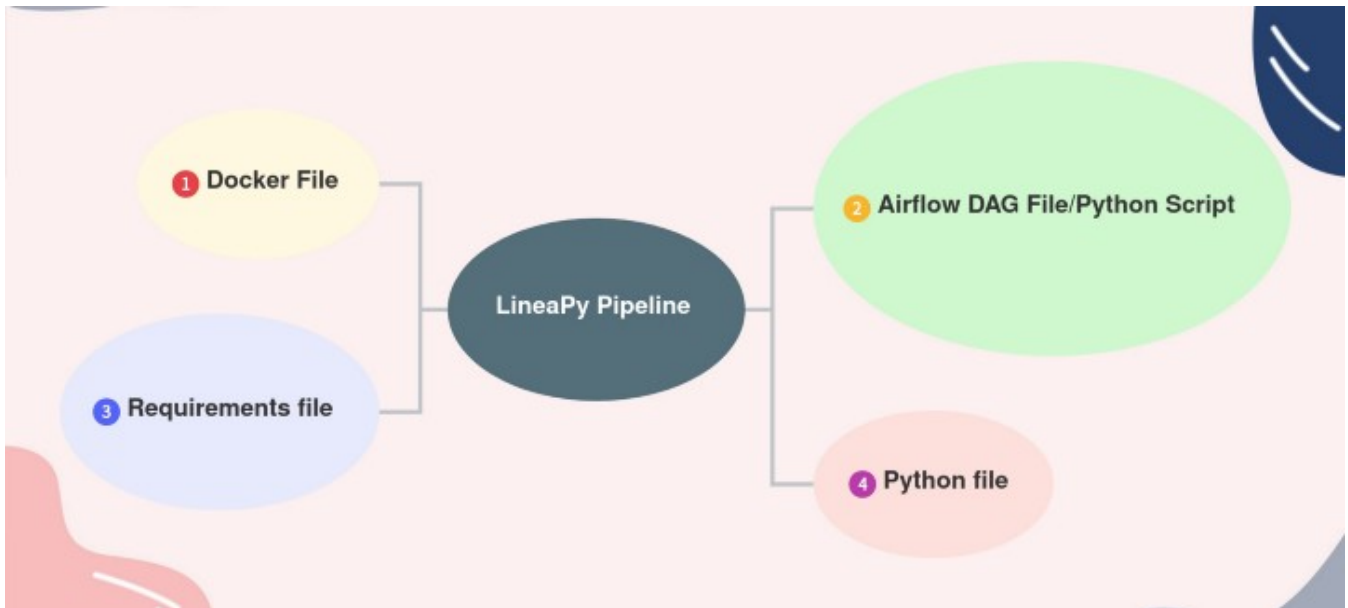


Image by the author

The requirements file is automatically generated

```
lineapy
matplotlib==3.2.2
numpy==1.21.6
pandas==1.3.5
seaborn==0.11.2
tensorflow==2.8.0
tensorflow.keras==2.8.0
```

The dockerfile is generated automatically when you build the pipeline.





Get unlimited access

Open in app

```
WORKDIR /tmp/containers

# copy all the requirements to run the current dag
COPY ./dnn_pipeline_requirements.txt ./
# install the required libs
RUN pip install -r ./dnn_pipeline_requirements.txt

WORKDIR /opt/airflow/dags
COPY
```

Image by the author

The airflow dag file is generated

```
train_data = PythonOperator(
    dag=dag,
    task_id="train_data",
    python_callable=dnn_pipeline.train_data,
)

train_labels = PythonOperator(
    dag=dag,
    task_id="train_labels_task",
    python_callable=dnn_pipeline.train_labels,
)

dnn_model = PythonOperator(
    dag=dag,
    task_id="dnn_model_task",
```

Image by the author

The python file which consists of all the data

```
learning-databases/auto-mpg/auto-mpg.data"
    column_names = [
        "MPG",
        "Cylinders",
        "Displacement",
        "Horsepower",
        "Weight",
        "Acceleration",
        "Model Year",
        "Origin",
    ]
    raw_dataset = pd.read_csv(
        url,
```



[Get unlimited access](#)[Open in app](#)

For more information on building the pipelines, [please check the documentation](#).

To know more about the artifact store, [please check the documentation](#).

Please check out the following for detailed information. Also, there are a few examples using the iris dataset and housing price prediction dataset from Kaggle on their Github. I tried it locally using Google Colab.

- [LineaPy Github](#)
- [LineaPy Documentation](#)

Conclusion:

LineaPy package will definitely help the MLOps teams in automating the workflow by using just 2 lines of code which are `save()`, `get()` and then `to_pipelinemethods()`. Maybe it can be used as a first cut and then further modifications can be done. I checked and the refactoring code looks good. Also, the docker and the airflow dag file look good. LineaPy is an open-source and for more information check out their GitHub repo.

Please free to get connected on [Linkedin](#).

References:

1. **Dataset Source:** Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: the University of California, School of Information and Computer Science.
2. About LineaPy -<https://lineapy.org/why-lineapy/>
3. LineaPy Github Repo: <https://github.com/LineaLabs/lineapy>
4. Lineapy ai-<https://linea.ai/>
5. Keras Tutorial:<https://www.tensorflow.org/tutorials/keras/regression>



[Get unlimited access](#)[Open in app](#)

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Emails will be sent to nk0962058@gmail.com. [Not you?](#)



Get this newsletter

