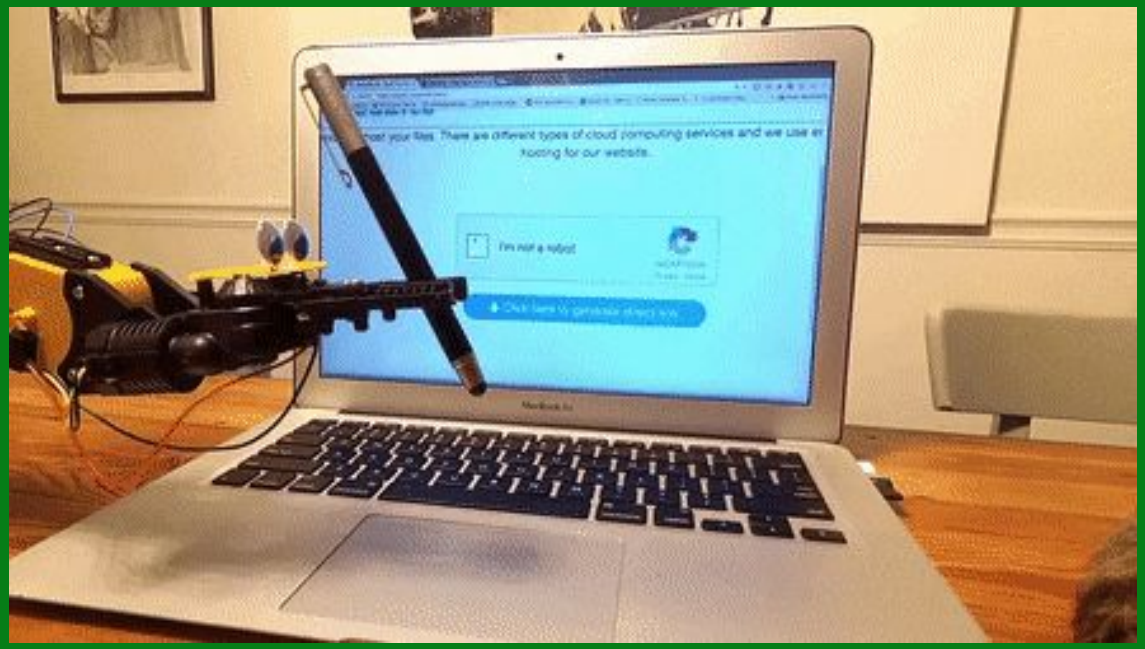
Software Development for Machine Learning in Python

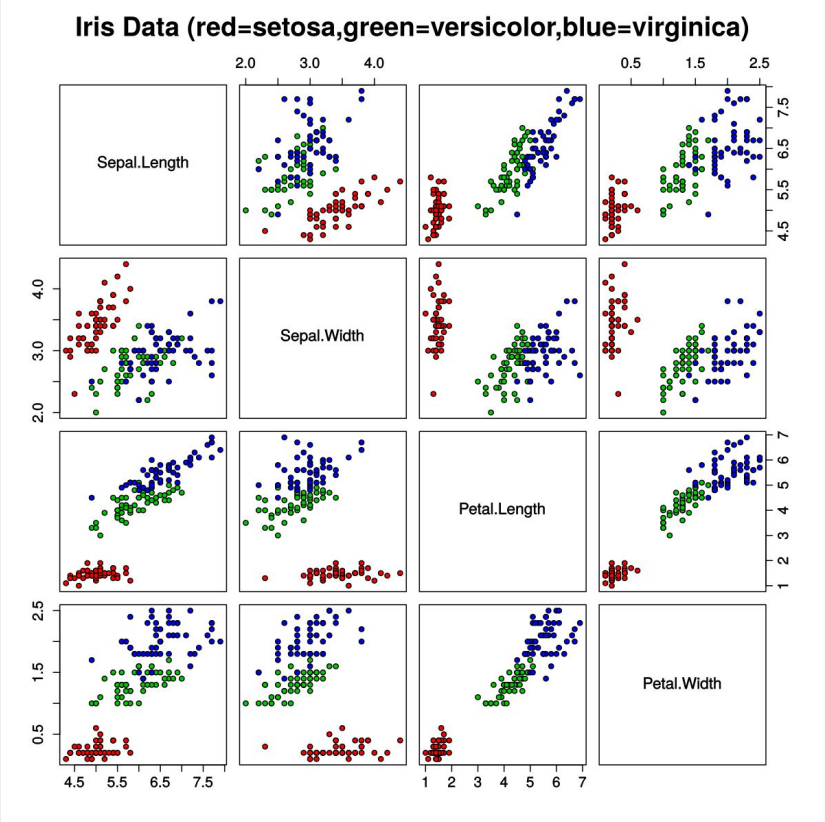


Pablo Alcain

Iris classification

Iris Versicolor **Iris Setosa** **Iris Virginica**

	s-l	s-w	p-l	p-w	variety
1	5.1	3.5	1.4	.2	Setosa
2	4.9	3	1.4	.2	Setosa
3	5	3.6	1.4	.2	Setosa
4	7	3.2	4.7	1.4	Versicolor
5	6.4	3.2	4.5	1.5	Versicolor
6	6.9	3.1	4.9	1.5	Versicolor
7	6.5	3	5.2	2	Virginica
8	6.2	3.4	5.4	2.3	Virginica
9	5.9	3	5.1	1.8	Virginica

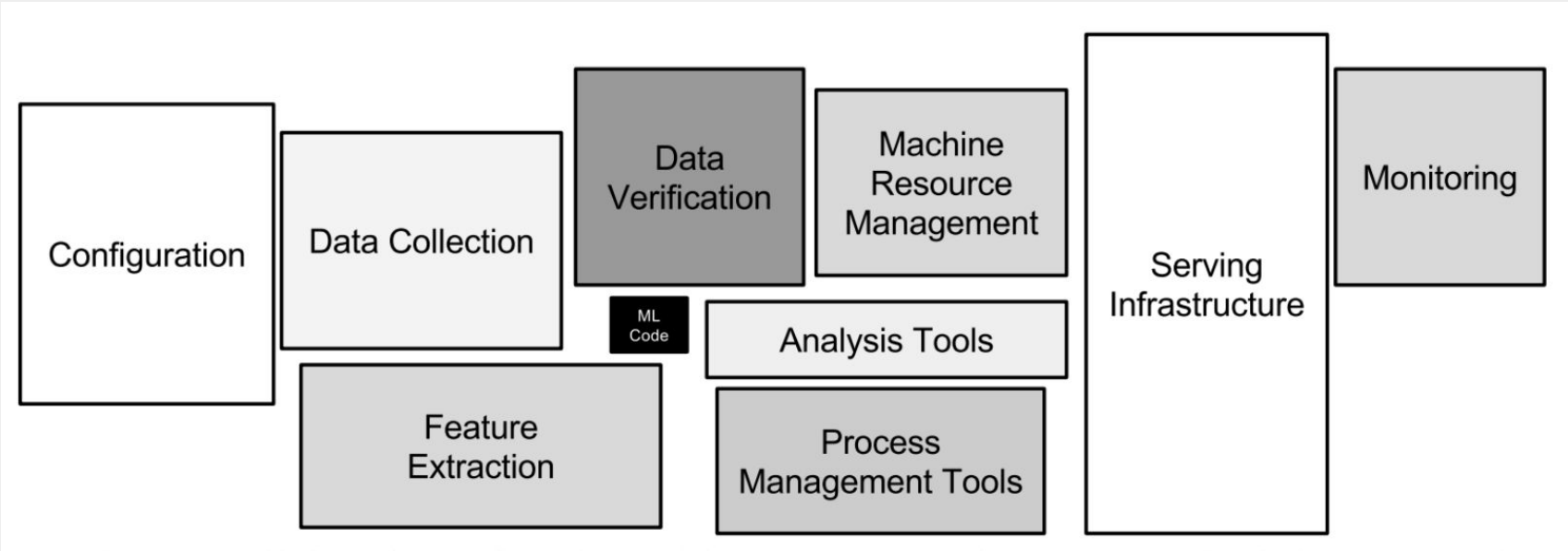


Iris classification: *playground*

```
import pandas as pd
from sklearn.feature selection import SelectKBest
from sklearn.linear model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline

dataset = pd.read csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                             ("clf", LogisticRegression())])
classifier.fit(features, target)
score = accuracy_score(target, classifier.predict(features))

print(f"I could predict irises' type with {score:2.2%}")
```

Iris classification: *production*

Data **preparation** Deployment (**CI/CD/CT**) Provisioning **infrastructure**

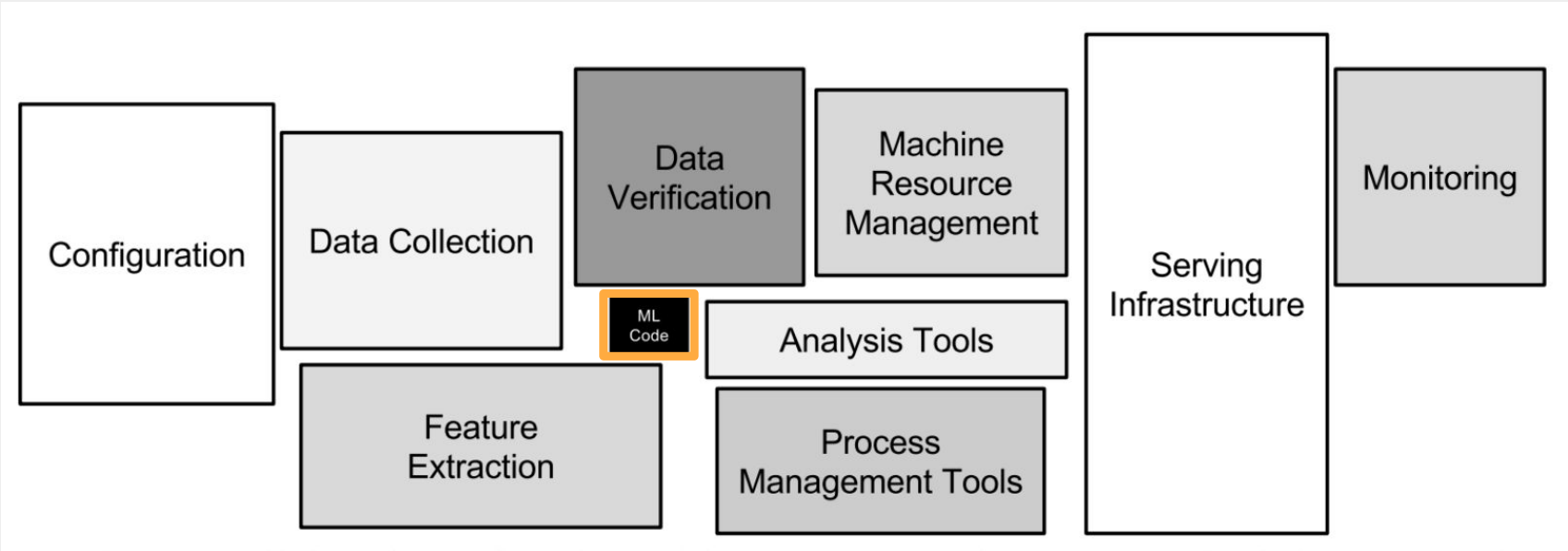


Hidden Technical Debt in Machine Learning Systems

Model **tracking** **Serving** capabilities ... and **more**

Iris classification: *production*

Data **preparation** Deployment (**CI/CD/CT**) Provisioning **infrastructure**



Hidden Technical Debt in Machine Learning Systems

Model **tracking** **Serving** capabilities ... and **more**

A Software Design solution for the productionization of ML code

Fast and **easy** exploration

Solutions are speculative and
models change too often

Declarative and **Intention revealing**

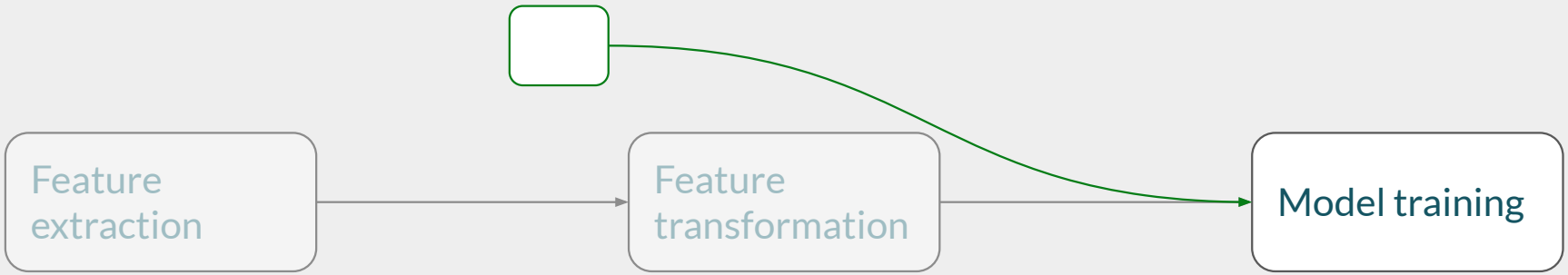
Seize the opportunity to use the code as documentation

-

Sensible **checkpoints**

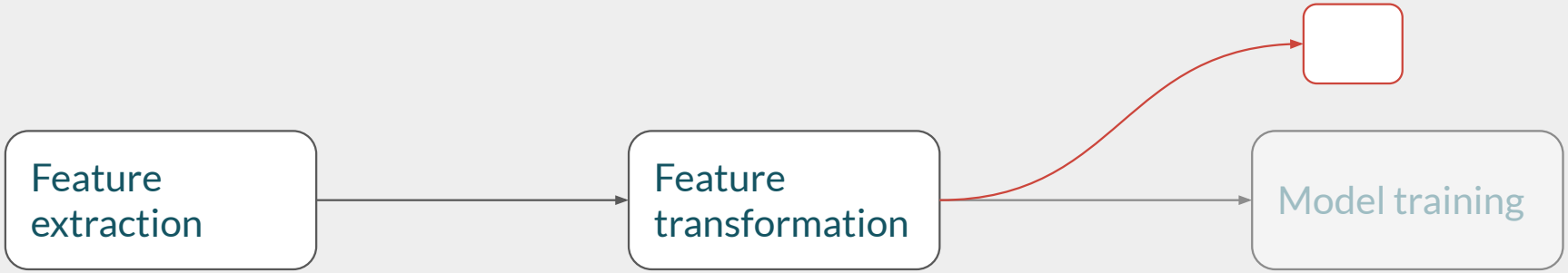
We can hook into and out of the workflow while experimenting

Sensible **checkpoints**



We can hook into and out of the workflow while experimenting

Sensible **checkpoints**



We can hook into and out of the workflow while experimenting

Seamless tracking and monitoring

Tracking cannot pollute the intention revealing workspace we need for exploration

Fast and easy exploration

Sensible checkpoints

Declarative and intentional

Seamless tracking and monitoring

Productionizing a model

(A cautionary tale)

train.py

```
import...

dataset = pd.read csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                             ("clf", LogisticRegression())])
classifier.fit(features, target)
score = accuracy_score(target, classifier.predict(features))

print(f"I could predict irises' type with {score:2.2%}")
```

train.py

```
import...

dataset = pd.read_csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                             ("clf", LogisticRegression())])
classifier.fit(features, target)

with open("model.pkl", "wb") as model_file:
    pickle.dump(classifier, model_file)
```

predict.py

```python
import pickle

import pandas as pd
from sklearn.metrics import accuracy_score

dataset = pd.read csv("unseen iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

with open("model.pkl", "rb") as model file:
    classifier = pickle.load(model_file)

score = accuracy_score(target, classifier.predict(features))

print(f"I could predict irises' type with {score:2.2%}")
```

train.py

```
...
with open("model.pkl", "wb") as model_file:
    pickle.dump(classifier, model_file)
...
```

predict.py

```
...
with open("model.pkl", "rb") as model_file:
    classifier = pickle.load(model_file)
...
```

Exposing implementation details

train.py

```
from model import save_model
...
save_model(classifier, "model.pkl")
...
```

predict.py

```
from model import load_model
...
classifier = load_model("model.pkl")
...
```

Abstract them into functions

model.py

```python
import pickle

def load model(filename):
    with open(filename, "rb") as model_file:
        classifier = pickle.load(model_file)
    return classifier

def save model(classifier, filename):
    with open(filename, "wb") as model file:
        pickle.dump(classifier, model_file)
```

train.py

```python
from model import save model
...# all the modeling stuff
save_model(classifier, "model.pkl")
...
```

predict.py

```python
from model import load_model
...
classifier = load_model("model.pkl")
...
```

model.py

```python
import pickle

def load model(filename):
    with open(filename, "rb") as model_file:
        classifier = pickle.load(model_file)
    return classifier

def save model(classifier, filename):
    with open(filename, "wb") as model file:
        pickle.dump(classifier, model_file)
```

train.py

```python
from model import save model
...# all the modeling stuff
save_model(classifier, "model.pkl")
...
```

predict.py

```python
from model import load_model
...
classifier = load_model("model.pkl")
...
```

model.py

```python
import pickle

def load model(filename):
    with open(filename, "rb") as model_file:
        classifier = pickle.load(model_file)
    return classifier

def save model(classifier, filename):
    with open(filename, "wb") as model file:
        pickle.dump(classifier, model_file)
```

train.py

```python
from model import save model
...# all the modeling stuff
save_model(classifier, "model.pkl")
...
```

predict.py

```python
from model import load_model
...
classifier = load_model("model.pkl")
...
```

train.py

```
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

from model import save_model

dataset = pd.read csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                             ("clf", LogisticRegression())])
classifier.fit(features, target)

save_model(classifier, "model.pkl")
```

train.py

```
import pandas as pd
from sklearn.feature selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

from model import save_model

dataset = pd.read csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                             ("clf", LogisticRegression())])
classifier.fit(features, target)

save_model(classifier, "model.pkl")
```

The fall

train.py

```
import pandas as pd

from model import save_model, train_model

dataset = pd.read_csv("iris.csv")
features = dataset[["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]

classifier = fit_model(features, target)

save_model(classifier, "model.pkl")
```

model.py

```
...
def fit_model(features, target):
    classifier = Pipeline(steps=[("select", SelectKBest(k=2)),
                                 ("clf", LogisticRegression())])
    classifier.fit(features, target)
    return classifier

def load_model(filename):
...

def save_model(classifier, filename):
...
```

train.py

```python
import pandas as pd

from model import Model

dataset = pd.read_csv("iris.csv")
features = dataset[
    ["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]
model = Model()
model.fit(features, target)
model.save("model.pkl")
```

model.py

```python
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=2)),
                   ("clf", LogisticRegression())])

    def fit(self, features, target):
        self.classifier.fit(features, target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb") as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

train.py

```python
import pandas as pd

from model import Model

dataset = pd.read_csv("iris.csv")
features = dataset[
    ["s-l", "s-w", "p-l", "p-w"]]
target = dataset["variety"]
model = Model()
model.fit(features, target)
model.save("model.pkl")
```

model.py

```python
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=2)),
                   ("clf", LogisticRegression())])

    def fit(self, features, target):
        self.classifier.fit(features, target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb") as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

dataset.py

```python
import pandas as pd


class Dataset:
    def __init__(self, filename):
        dataset = pd.read_csv(filename)
        self.features = dataset[
            ["s-l", "s-w", "p-l", "p-w"]]
        self.target = dataset["variety"]
```

train.py

```python
from dataset import Dataset
from model import Model

iris_dataset = Dataset("iris.csv")
model = Model()
model.fit(iris_dataset)
model.save("model.pkl")
```

model.py

```python
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=2)),
                   ("clf", LogisticRegression())])

    def fit(self, dataset):
        self.classifier.fit(dataset.features,
                            dataset.target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb") as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

dataset.py

```python
import pandas as pd


class Dataset:
    def __init__(self, filename):
        dataset = pd.read_csv(filename)
        self.features = dataset[
            ["s-l", "s-w", "p-l", "p-w"]]
        self.target = dataset["variety"]
```

model.py

```python
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=2)),
                   ("clf", LogisticRegression())])

    def fit(self, dataset):
        self.classifier.fit(dataset.features,
                            dataset.target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb") as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

train.py

```python
from dataset import Dataset
from model import Model

iris_dataset = Dataset("iris.csv")
model = Model()
model.fit(iris_dataset)
model.save("model.pkl")
```

Requirements change:
we don’t want to pick the best 2

Requirements change:
we don't want to pick the best 2

model.py

```
class Model:
   def __init__(self, number_of_selected):
       self.classifier = Pipeline(
           steps=[("select", SelectKBest(k=number_of_selected)),
                  ("clf", LogisticRegression())])

   def fit(self, dataset):
       self.classifier.fit(dataset.features, dataset.target)

   def load(self, filename):
       with open(filename, "rb") as model_file:
           self.classifier = pickle.load(model_file)

   def save(self, filename):
       with open(filename, "wb") as model_file:
           pickle.dump(self.classifier, model_file)

   def predict(self, features):
       return self.classifier.predict(features)
```

Requirements change:
sometimes we don’t want to pick

Requirements change:
sometimes we don't want to pick

model.py

```
class Model:
    def __init__(self, number_of_selected=2):
        if number_of_selected is None:
            self.classifier = LogisticRegression()
        else:
            self.classifier = Pipeline(
                steps=[("select", SelectKBest(k=number_of_selected)),
                       ("clf', (LogisticRegression()))])
...
```

Requirements change:

wait, we don’t always want a logistic regression

Requirements change:

wait, we don't always want a logistic regression

model.py

```
class Model:
    def __init__(self, number_of_selected=2, classifier_type="log"):
        if classifier_type == "log":
            base_classifier = LogisticRegression()
        elif classifier_type == "nn":
            base_classifier = MLPClassifier()
        else:
            raise RuntimeError(f"Unknown type {classifier_type}")

        if number_of_selected is None:
            self.classifier = base_classifier
        else:
            self.classifier = Pipeline(
                steps=[("select", SelectKBest(k=number_of_selected)),
                       ("clf', base_classifier)])
```

Requirements change:

It looks like we don't actually know what model we want

model.py

```
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=number_of_selected)),
                   ("clf', (LogisticRegression())])
...
```

Requirements change:

It looks like we don't actually know what model we want

model.py

```
class Model:
    def __init__(self):
        self.classifier = Pipeline(
            steps=[("select", SelectKBest(k=number_of_selected)),
                   ("clf', (LogisticRegression()))])
...
```

model.py

```
class Model:
    def __init__(self, backend_model):
        self.classifier = backend_model
...
```

model.py

```
class Model:
    def __init__(self, backend_model):
        self.classifier = backend_model

    def fit(self, dataset):
        self.classifier.fit(dataset.features, dataset.target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb') as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

model.py

Dependency Injection

```
class Model:
    def __init__(self, backend_model):
        self.classifier = backend_model

    def fit(self, dataset):
        self.classifier.fit(dataset.features, dataset.target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb') as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

model.py

Dependency Injection

```
class Model:
    def __init__(self, backend_model):
        self.classifier = backend_model

    def fit(self, dataset):
        self.classifier.fit(dataset.features, dataset.target)

    def load(self, filename):
        with open(filename, "rb") as model_file:
            self.classifier = pickle.load(model_file)

    def save(self, filename):
        with open(filename, "wb') as model_file:
            pickle.dump(self.classifier, model_file)

    def predict(self, features):
        return self.classifier.predict(features)
```

train.py

```
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

from dataset import Dataset
from model import Model

iris_dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
model = Model(backend_model)
model.fit(iris_dataset)
model.save("model.pkl')
```

Remove a feature

dataset.py

```
import pandas as pd


class Dataset:
    def __init__(self, filename):
        dataset = pd.read_csv(filename)
        self.features = dataset[["s-l", "s-w", "p-l", "p-w"]]
        self.target = dataset["variety"]
```

train.py

```
iris_dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
model = Model(backend_model)
model.fit(iris_dataset)
model.save("model.pkl")
```

Remove a feature

dataset.py

```
import pandas as pd


class Dataset:
    def __init__(self, filename):
        dataset = pd.read_csv(filename)
        self.features = dataset[["s-l", "s-w", "p-l", "p-w"]]
        self.target = dataset["variety"]
```

train.py

```
iris_dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
model = Model(backend_model)
model.fit(iris_dataset)
model.save("model.pkl")
```

A-ha!

Beyond DI:
We are **exposing a known library** to the DS

train.py

```
iris_dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
model = Model(backend_model)
model.fit(iris_dataset)
model.save("model.pkl")
```

train.py

```
dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
target = dataset.dataframe["variety"]
features = dataset.dataframe[["s-l", "s-w", "p-l", "p-w"]]

model = Model(backend_model)
model.fit(features, target)
model.save("model.pkl")
```

dataset.py

```
import pandas as pd

class Dataset:
   def __init__(self, filename):
       self.dataframe = pd.read_csv(filename)
```

***abstraction*:** a simplification of something much more complicated that is going on under the covers.

Joel Spolsky

joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

abstraction: a simplification of something much more complicated that is going on under the covers.

Joel Spolsky

*All non-trivial **abstractions**, to some degree, are **leaky***

joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

train.py

```
dataset = Dataset("iris.csv")
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
target = dataset.dataframe["variety"]
features = dataset.dataframe[["s-l", "s-w", "p-l", "p-w"]]

model = Model(backend_model)
model.fit(features, target)
model.save("model.pkl")
```

You'll have to **compromise**

Let the abstractions **leak**, but **choose** how to

We have been building a **framework for code development** leveraging the knowledge of **python, pandas** and **sklearn** as interfaces between the user and the backend

Fast and **easy** exploration

train.py

```
dataset = Dataset("iris.csv")

backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
target = dataset.dataframe["variety"]
features = dataset.dataframe[["s-l", "s-w", "p-l", "p-w"]]

model = Model(backend_model)
model.fit(features, target)
model.save("model.pkl")
```

Fast and **easy** exploration

train.py

```
dataset = Dataset("iris.csv")
dataset.dataframe = dataset.dataframe[dataset.dataframe[ "s-l"] <= 5.0]
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ( "clf", LogisticRegression())])
target = dataset.dataframe[ "variety"]
features = dataset.dataframe[[ "s-l", "s-w", "p-l", "p-w"]]

model = Model(backend_model)
model.fit(features, target)
model.save("model.pkl")
```

Declarative and **Intention revealing**

train.py

```
dataset = Dataset("iris.csv")
dataset.remove_long_sepals(5.0)
backend_model = Pipeline(steps=[("select", SelectKBest(k=2)),
                                ("clf", LogisticRegression())])
target = dataset.dataframe["variety"]
features = dataset.dataframe[["s-l", "s-w", "p-l", "p-w"]]

model = Model(backend_model)
model.fit(features, target)
model.save("model.pkl")
```

dataset.py

```
class Dataset:
    ...
    def remove_long_sepals(self, max_length=5.0):
        self.dataframe = self.dataframe[self.dataframe["s-l"] <= max_length]
```

Sensible **checkpoints**

```
class Model:
    ...
    def fit(self, features, target):
        self.classifier.fit(features, target)
    ...
```

```
class Model:
    ...
    def fit(self, dataset):
        self.classifier.fit(dataset.features,
                            dataset.target)
    ...
```

Consider exposing to the application "basic" types instead of custom abstractions. This brings flexibility between checkpoints.

Sensible **checkpoints**

```
class Model:
    ...
    def fit(self, features, target):
        self.classifier.fit(features, target)
    ...
```

```
class Model:
    ...
    def fit(self, dataset):
        self.classifier.fit(dataset.features,
                            dataset.target)
    ...
```

Consider exposing to the application "basic" types instead of custom abstractions. This brings flexibility between checkpoints.

But be weary of **Primitive Obsession**

Seamless tracking and monitoring

logger.py

```
class TextLogger:
    def __init__(self, filename):
        self.filename = filename

    def log(self, line):
        with open(self.filename, "a") as log_file:
            print(line, file=log_file)
```

model.py

```
class Model:
    def __init__(self, backend_model, logger=TextLogger("log.txt")):
        self.classifier = backend_model
        self.logger = logger

    def fit(self, features, target):
        self.logger.log(f"Trained model with {len(features)} features")
        self.classifier.fit(features, target)
    ...
```

The API might be **wider** than what you expected

model.py

```
class Model:
    def __init__(
            self,
            backend_model=NoSKModel(),
            logger=TextLogger("log.txt")
        ):
        self.classifier = backend_model
        self.logger = logger

    @classmethod
    def from_file(cls, filename, logger=TextLogger("log.txt")):
        model = cls(logger=logger)
        model.load(filename)
        return model
...
```

The API might be **wider** than what you expected

dataset.py

```
class Dataset:
    def __init__(self, dataframe):
        self.dataframe = dataframe

    @classmethod
    def from_csv(cls, filename):
        dataframe = pd.read_csv(filename)
        return cls(dataframe)

    @classmethod
    def from_parquet(cls, filename):
        dataframe = pd.read_parquet(filename)
        return cls(dataframe)
...
```

Implementation detail lies in the eye of the beholder

Allow **flexibility** by letting the application code speak the language of the **developer**

Some abstractions will **leak**. Instead of fighting, consider channelling them.