Model training with GitHub Actions

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

Dataset: Weather Prediction in Australia

- Binary classification
  - Predicts rainfall for tomorrow
- 5 Categorical Features
  - Location
  - WindGustDir
  - WindDir9am
  - WindDir3pm
  - RainToday
- 17 Numerical Features
  - MinTemp
  - MaxTemp
  - Rainfall
  - Evaporation
  - ...
  - WindGustSpeed
  - Cloud3pm
  - Temp9am
  - RISK_MM

[1] https://www.kaggle.com/datasets/rever3nd/weather-data

Modeling workflow

- Data preprocessing
  - Convert categorical features to numerical
  - Replace missing values of features
  - Scale features
- Random Forest Classifier
  - `max_depth = 2` , `n_estimators = 50`
- Standard metrics on test data
  - Performance plots
    - Confusion matrix plot

Data preparation: target encoding

```python
def target_encode_categorical_features(
    df: pd.DataFrame, categorical_columns: List[str], target_column: str
) -> pd.DataFrame:
    encoded_data = df.copy()

    # Iterate through categorical columns
    for col in categorical_columns:
        # Calculate mean target value for each category
        encoding_map = df.groupby(col)[target_column].mean().to_dict()

        # Apply target encoding
        encoded_data[col] = encoded_data[col].map(encoding_map)

    return encoded_data
```

[1] https://maxhalford.github.io/blog/target-encoding/

Imputing and Scaling

```python
def impute_and_scale_data(df_features: pd.DataFrame) -> pd.DataFrame:
    # Impute data with mean strategy
    imputer = SimpleImputer(strategy="mean")
    X_preprocessed = imputer.fit_transform(df_features.values)

    # Scale and fit with zero mean and unit variance
    scaler = StandardScaler()
    X_preprocessed = scaler.fit_transform(X_preprocessed)

    return pd.DataFrame(X_preprocessed, columns=df_features.columns)
```

Training

- Train/Test split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
  data.drop(TARGET_COLUMN), data[TARGET_COLUMN], random_state=1993)
```

- Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(
  max_depth=2, n_estimators=50, random_state=1993)
clf.fit(X_train, y_train)
```

Metrics

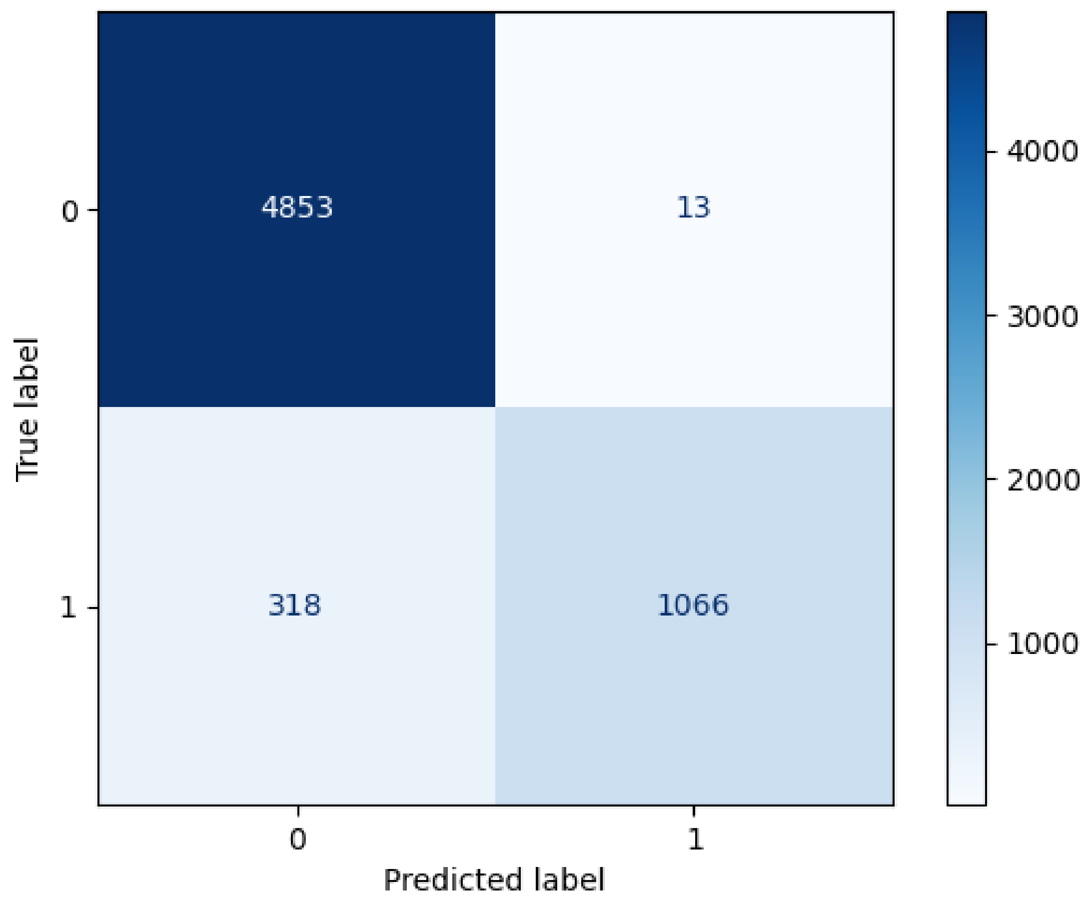
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Calculate predictions
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
# Calculate precision
precision = precision_score(y_test, y_pred)
# Calculate recall
recall = recall_score(y_test, y_pred)
# Calculate f1 score
f1 = f1_score(y_test, y_pred)
```

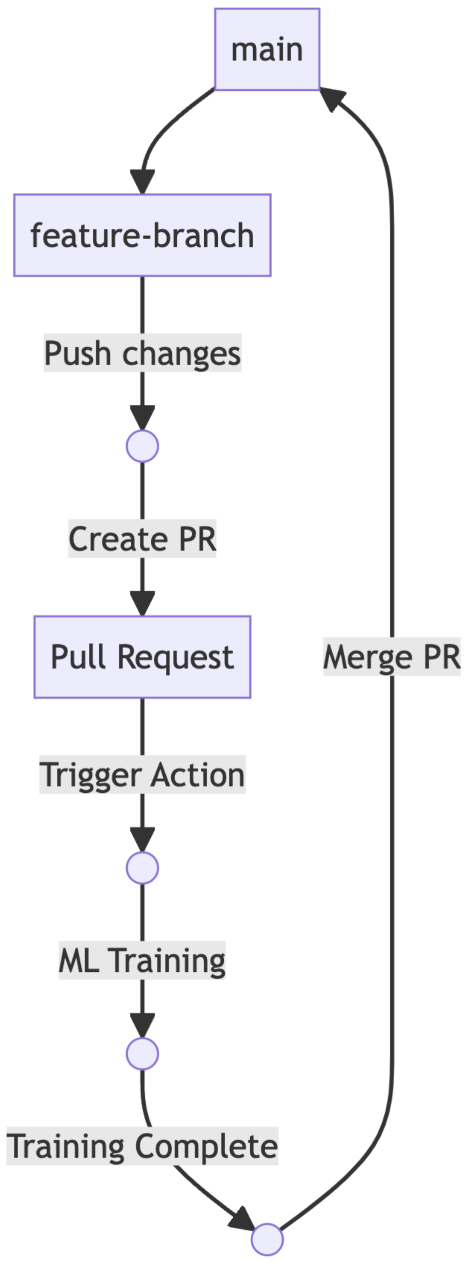
[1] https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

Plots

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test,cmap=plt.cm.Blues)
```



GitHub Actions Workflow



- Continuous Machine Learning (CML)
  - CI/CD tool for Machine Learning
  - GitHub Actions Integration
    - Provision training machines
    - Perform training and evaluation
    - Compare experiments
    - Monitor datasets
    - Visual reports

[1] https://cml.dev/ [2] https://martinfowler.com/bliki/FeatureBranch.html

CML commands

```
# Enable setup-cml action to be used later
- uses: iterative/setup-cml@v1
```

```
- name: Train model
  run: |
    # Your ML workflow goes here
    pip install -r requirements.txt
    python3 train.py
```

[1] https://www.markdownguide.org/basic-syntax/#images

CML commands

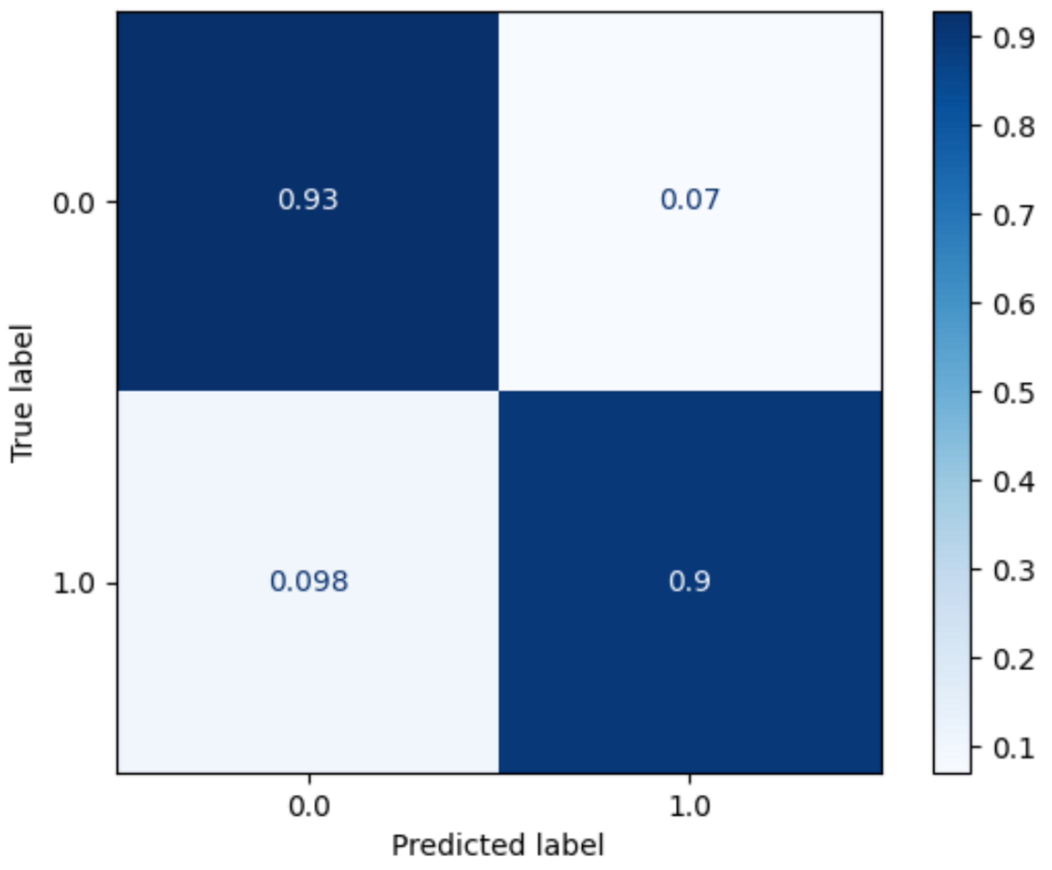
```yaml
- name: Write CML report
  run: |
    # Add results and plots to markdown
    cat results.txt >> report.md
    echo "![training graph](./graph.png)" >> report.md

    # Create comment from markdown report
    cml comment create report.md
  env:
    REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Output

github-actions bot commented on Apr 19

Accuracy: 0.916



Let's practice!

CI/CD FOR MACHINE LEARNING

Versioning datasets with Data Version Control

CI/CD FOR MACHINE LEARNING

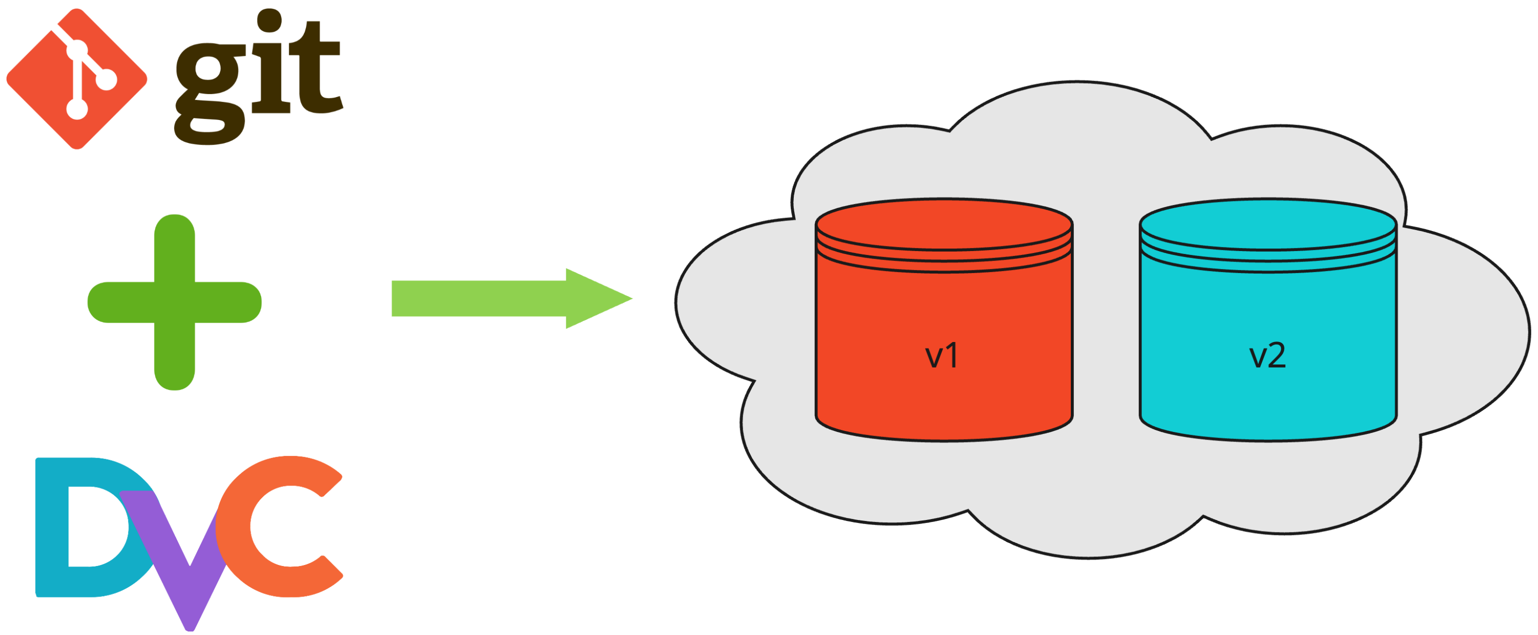
Ravi Bhadauria
Machine Learning Engineer

Why versioning data matters

- Versioning ensures a historical record of data changes
- Data versioning is crucial
  - Reproducibility
  - Experimentation and Iteration
  - Collaboration
  - Bug Tracking and Debugging
  - Model Monitoring and Maintenance
  - Auditing and Validation

Data Version Control (DVC)

- DVC: Data Version Control tool
  - Manages data and experiments
  - Similar to Git



- Git tracks metadata, DVC handles data versioning

DVC Storage

- Data stored separately
  - SSH, HTTP/HTTPS, Local File System
  - AWS, GCP, and Azure object storage
- Install locally

```
pip install dvc
```

Initializing DVC

- Initialize Git `git init`
- Initialize DVC

```
-> dvc init
Initialized DVC repository.
You can now commit the changes to git.
```

- Sets up DVC project files, ready to version data

```
.dvc
|- .gitignore
|- config
|- tmp
```

Adding Files to DVC

- Add data files using `dvc add <file>` command

```
-> dvc add data.csv
```

- `.dvc` placeholders containing file metadata are generated
  - Each DVC tracked file has its corresponding `.dvc` file (`data.csv -> data.csv.dvc`)
  - Checked into Git to manage data versions
- DVC cache is populated in `.dvc/cache`

DVC data files

- Data file

```
-> cat data.csv
This is a sample data file.
```

- `data.csv.dvc` file

```
-> cat data.csv.dvc
 outs:
 - md5: ea9972ac9f8fa321ea74969e93acc196
   size: 28
   hash: md5
   path: data.csv
```

Summary

- Data versioning ensures reproducibility and collaboration
- DVC: tool for managing data versioning, works with Git
- Initialize DVC with `dvc init`, add files with `dvc add <file>`
- `.dvc` files store metadata, Git tracks versions

Let's practice!

CI/CD FOR MACHINE LEARNING

Interacting with DVC remotes

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

Understanding DVC Remotes

- DVC Remotes: Location for Data Storage
- Similar to Git remotes, but for *cached* data
- Benefits of using remotes
  - Synchronize large files and directories
  - Centralize or distribute data storage
  - Save local space
- Supported storage types
  - Amazon S3, GCS, Google Drive
  - SSH, HTTP, local file systems

Setting Up Remotes

- Set remotes using `dvc remote add` command
- For AWS

```
dvc remote add myAWSremote s3://mybucket
```

- Customizations can be done with `dvc remote modify`

```
dvc remote modify myAWSremote connect_timeout 300
```

- `.dvc/config` change

```
 ['remote "myAWSremote"']
     url = s3://mybucket
     connect_timeout = 300
```

Local and Default Remotes

- Local remotes are used for rapid prototyping
- Use system directories or Network Attached Storage

```
dvc remote add mylocalremote /tmp/dvc
```

- Set default remotes with `-d` flag

```
dvc remote add -d mylocalremote /tmp/dvc
```

- Default remote assigned in the `core` section of `.dvc/config`

```
[core]
remote = mylocalremote
```

Uploading and Retrieving Data

- Commands to transfer data
  - Push to remote: `dvc push <target>`
  - Pull from remote: `dvc pull <target>`
- Similar to `git push` and `git pull`
  - `.dvc` is tracked by Git, not DVC
- Target can be individual files `dvc push data.csv`
  - Or entire cache `dvc push`
- Override default remote with `-r` flag

```
dvc push -r myAWSremote data.csv
```

Tracking Data Changes

- Change data file contents, then add dataset changes

```
dvc add /path/to/data/datafile
```

- Commit changed `.dvc` file to Git

```
git add /path/to/datafile.dvc
git commit /path/to/datafile.dvc -m "Dataset updates"
```

- Push metadata to Git

```
git push origin main
```

- Upload changed data file

```
dvc push
```

Let's practice!

CI/CD FOR MACHINE LEARNING

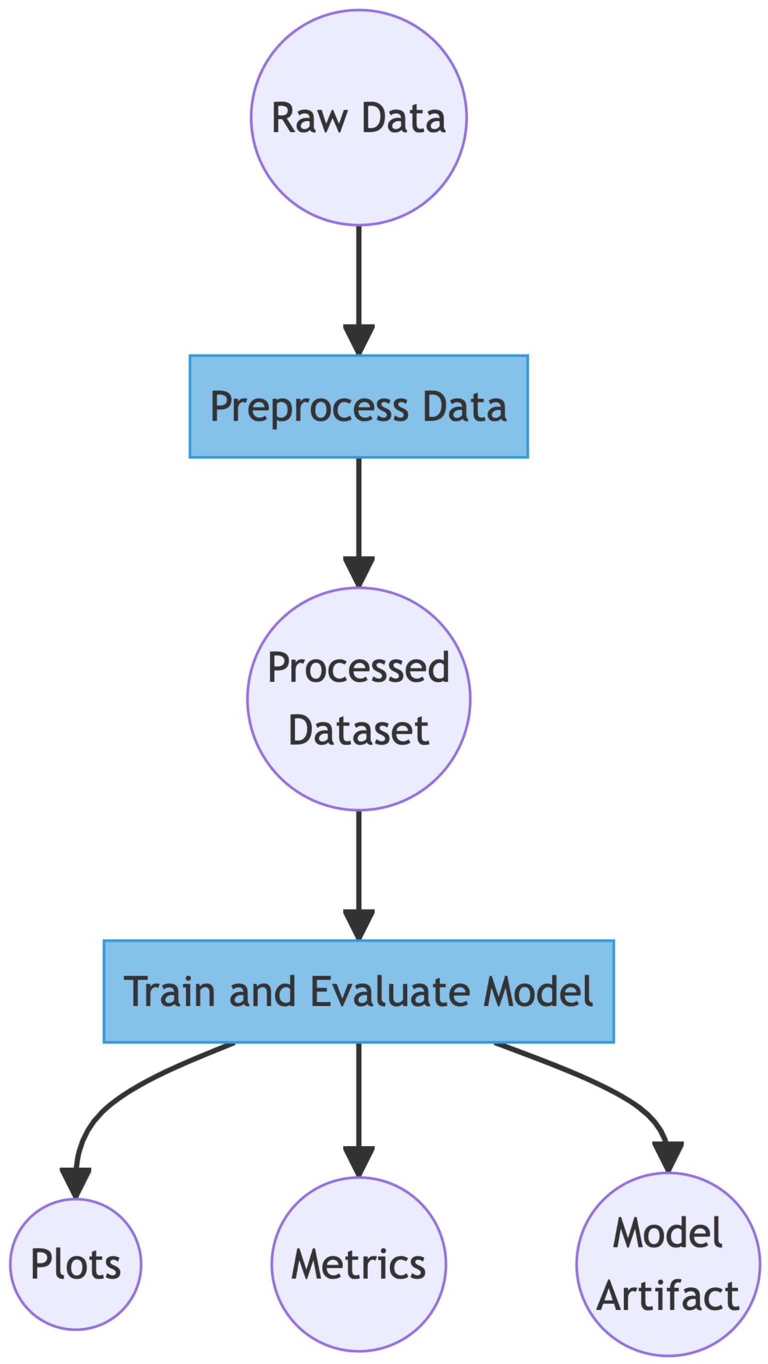
DVC Pipelines

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

The need for a data pipeline

- Versioning data alone is not very useful
- Tracking data lineage is important
  - Filtering, cleaning, and transformation
  - Model training
- Run only what's needed
- Steps in Directed Acyclic Graph (DAG)



DVC pipelines

- Sequence of stages defining ML workflow and dependencies
- Defined in `dvc.yaml` file
  - Input data and scripts (`deps`)
  - Stage execution commands (`cmd`)
  - Output artifacts (`outs`)
    - Special data e.g. `metrics` and `plots`
- Similar to the GitHub Actions workflow
  - Focused on ML tasks instead of CI/CD
  - Can be abstracted as a step in GHA

Defining pipeline stages

- Create stages using `dvc stage add`

```
dvc stage add \
-n preprocess \
-d raw_data.csv -d preprocess.py \
-o processed_data.csv \
python preprocess.py
```

- `dvc.yaml` contents

```
stages:
preprocess:
  cmd: python preprocess.py
  deps:
  - preprocess.py
  - raw_data.csv
  outs:
  - processed_data.csv
```

Dependency graphs

- Add a training step using output from previous step

```
dvc stage add \
-n train \
-d train.py -d processed_data.csv \
-o plots.png -o metrics.txt \
python train.py
```

```
stages:
  preprocess:
    cmd: python preprocess.py
    deps:
    - preprocess.py
    - raw_data.csv
    outs:
    - processed_data.csv
  train:
    cmd: python train.py
    deps:
    - processed_data.csv
    - train.py
    outs:
    - plots.png
```

Reproducing a pipeline

- Reproduce the pipeline using `dvc repro`

```
-> dvc repro
Running stage 'preprocess':
> python preprocess.py
Running stage 'train':
> python train.py
Updating lock file 'dvc.lock'
```

- A state file `dvc.lock` is generated
  - Similar to `.dvc` file, captures MD5 hashes
  - Commit to Git immediately to record state
    `git add dvc.lock && git commit -m "first pipeline repro"``

Using cached results

- Using cached results to speed up iteration

```
-> dvc repro
Stage 'preprocess' didn't change, skipping
Running stage 'train' with command: ...
```

Visualizing DVC pipeline

```
-> dvc dag
+------------+
| preprocess |
+------------+
       *
       *
       *
  +-------+
  | train |
  +-------+
```

[1] https://dvc.org/doc/command-reference/dag

Summary

- DVC pipelines are useful for
  - Rapid iteration due to caching
  - Reproducibility with `dvc.yaml` and `dvc.lock`
  - CI/CD enablement
- Generate pipelines with `dvc stage add`
- Reproduce/execute with `dvc repro`
- Visualize with `dvc dag`

Let's practice!

CI/CD FOR MACHINE LEARNING