Just Build It!

Tips for Making MLOps and ML Engineering **Real**

Andy McMahon*
ML Engineering Lead
NatWest Group, Edinburgh, Scotland

MLOps Community Meetup

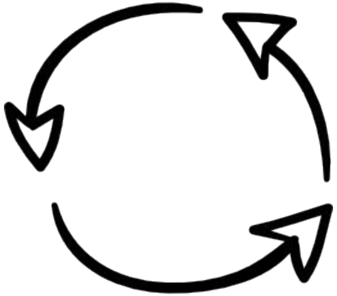
**speaking in a personal capacity*

Some Preamble

- My talk title is slightly tongue-in-cheek
- Objective: Let go of the fear of getting started
- Some phrases that can capture my sentiment:
    - Have a bias for action/execution is key
    - Bootstrap your capability
- In a nutshell - attempt what you are **not ready to do** and you will learn faster!

MLOps and MLEng

ML Eng: Getting your data science models into production by applying enough software engineering.



MLOps: The end-to-end lifecycle management of models and solutions with ML in them.

MLOps and ML Eng

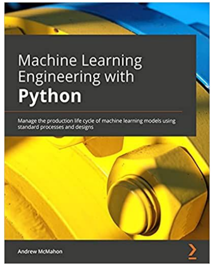
Software Engineering ⟷ **ML Engineering**

Disciplines with tools, tech, processes and best practice - part of the *means*

DevOps ⟷ **MLOps**

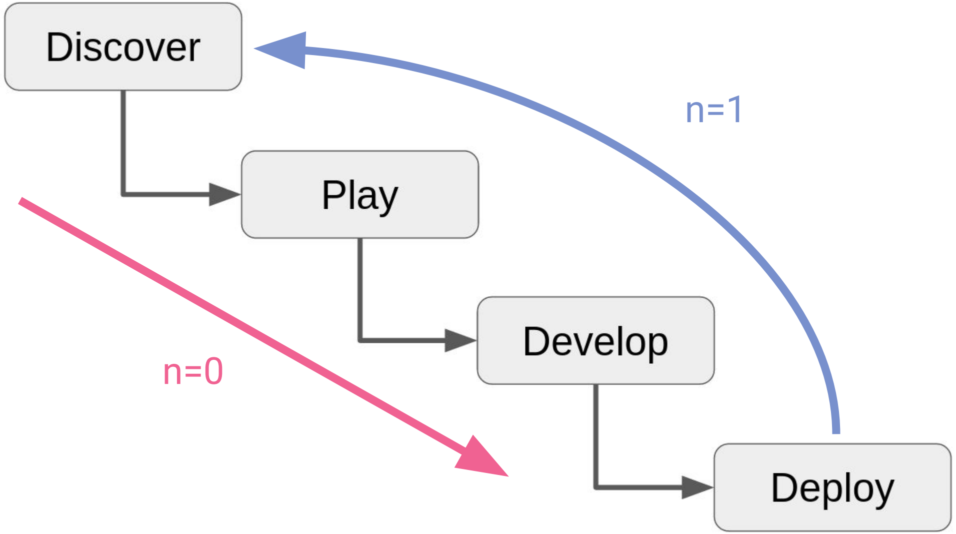
Culture of blended responsibilities - the *ends*

MLOps as Induction



1. Get your first model into production (n=0). **This is how you learn ML Engineering.**

2. Get the (n+1)th model into production given that the nth model is in production. **This is how you learn MLOps.**



Challenge: Analysis Paralysis

mlflow

dataiku

neptune.ai

ALGORITHMIA

fiddler

Kubeflow

truera

Kedro

ALIBI

DataRobot

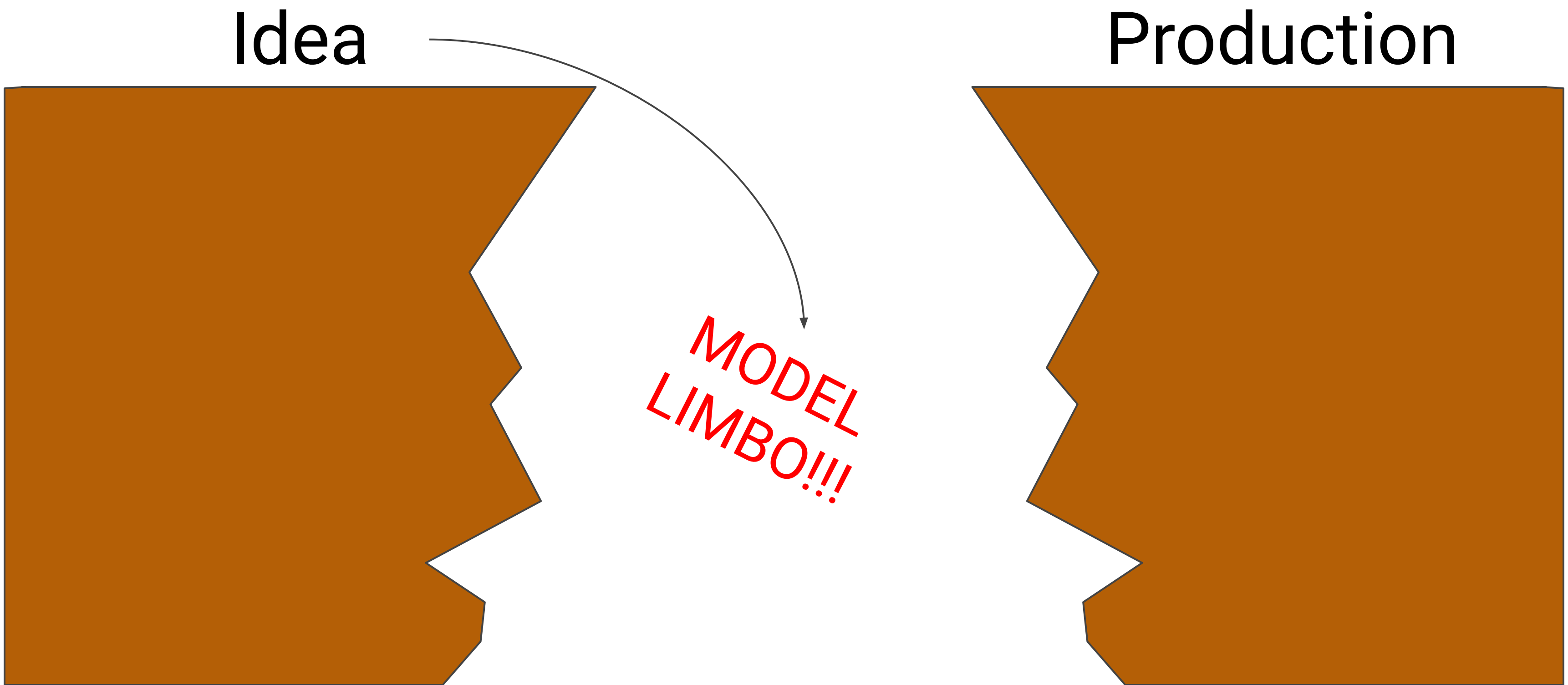
comet

TensorBoard

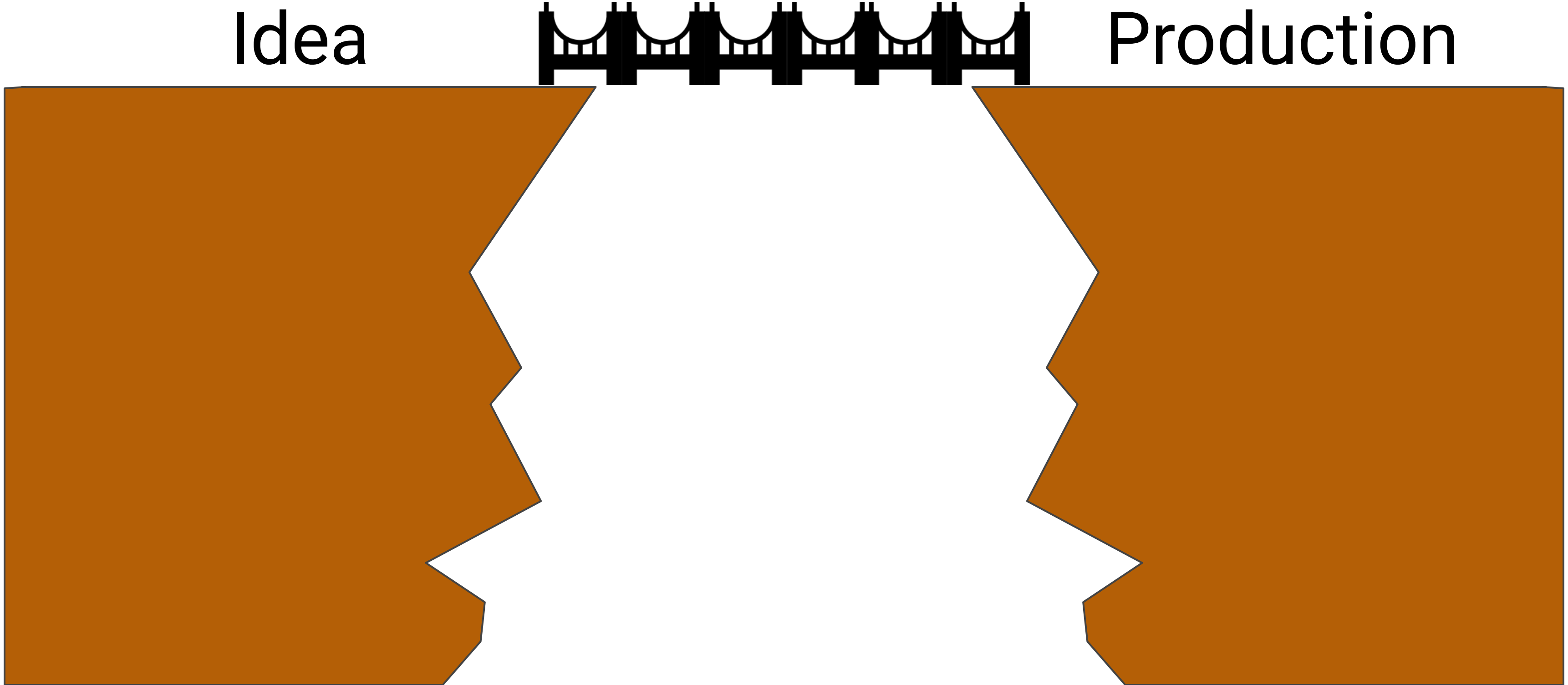
Arthur

So What Do We Do?

The Chasm

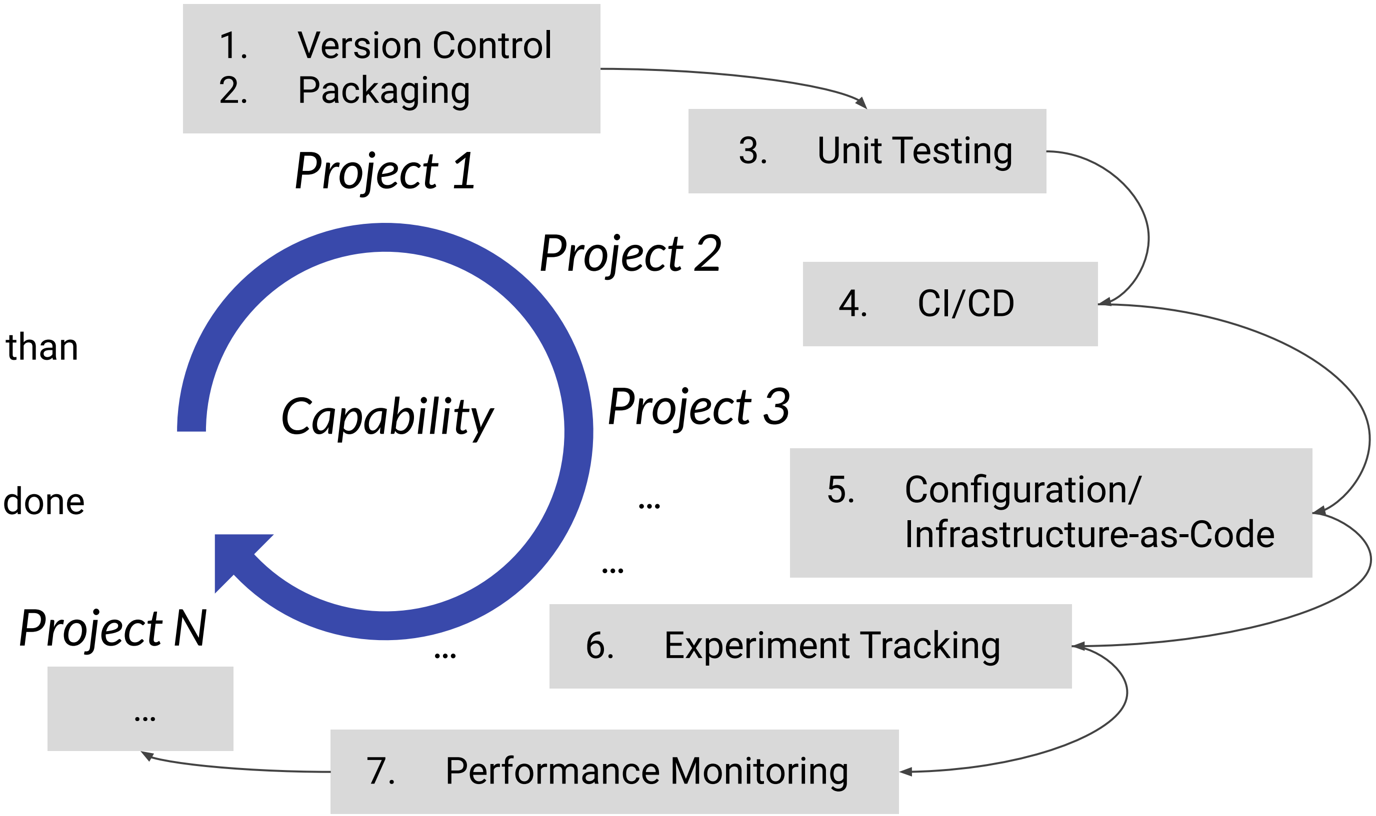


The Chasm



Bootstrapping

- Start small then ratchet up
- Working is better than perfect
- You will never be done



Getting The (n=0)th Model Into Production

ML Eng 101 - Production

What?

- An isolated environment with strict controls, defined risk profile, service level agreements ….
- The place where your ‘real’ inference happens
- Where request/response traffic goes

How?

- Forget about notebooks* (sorry everyone, but it’s true)
- We package, we lint, we test, we build, we version control
- We follow a clear route-to-live with appropriate promotion mechanisms

Done.

**We can fight about this one later …*

ML Eng 102 - The Code

Prep for Prophet

```
df.rename(columns= {'Datetime': 'ds', 'AEP_MW': 'y'}, inplace=True)
```

```
df['ds']=df['ds'].astype('datetime64[ns]')
```

```
df.dtypes
```

```
#Initialize Split Class, we'll split our data 5 times for cv
ts_splits = TimeSeriesSplit(n_splits=5)
```

Train and Forecast

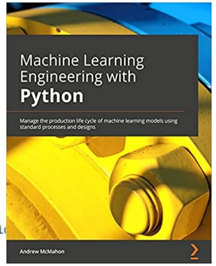
```
tmp = time_split_train_test(df.sort_values('ds', ascending=True).iloc[-1000:], ts_splits)
```

```
tmp.head()
```

Plot

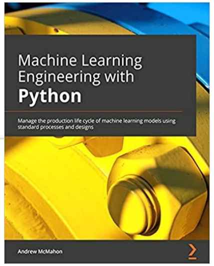
```
nrow = 5; ncol = 1;
fig, axs = plt.subplots(nrows=nrow, ncols=ncol, figsize=(20,30))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i, ax in enumerate(fig.axes):
    split_rmse = tmp[(tmp['split']==i) & (tmp['train']==False)]['rmse'].iloc[0]

    ax.set_title('Split '+str(i)+' - RMSE: '+"{:.2f}".format(split_rmse))

    tmp[(tmp['split']==i) & (tmp['train']==True)].plot(x='ds', y='y', ax=ax, color='blue', marker='o')
    tmp[(tmp['split']==i) & (tmp['train']==False)].plot(x='ds', y='y', ax=ax, color='red', marker='o')
    tmp[(tmp['split']==i) & (tmp['train']==False)].plot(x='ds', y='yhat', ax=ax, color='orange', marker='^')
```

- OOP or Functional
- Separation of Concerns
- Keep it Simple Stupid
- Unit-test friendly



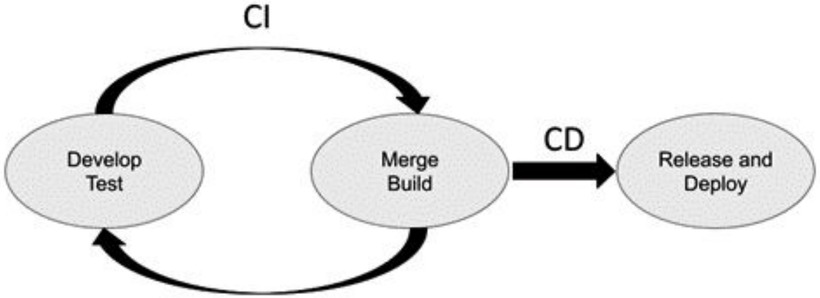

```
class Trainer(object):

    def __init__(self, forecast_session):
        self.forecast_session = forecast_session
        self.df_pred_metadata = self.get_df_pred_metadata()
        self.latest_predictor = self.get_latest_predictor()

    def get_df_pred_metadata(self):
        predictor_metadata = self.forecast_session.forecast.list_predictors()['Predi
        df_pred_metadata = pd.DataFrame.from_records(predictor_metadata)
        return df_pred_metadata

    def get_latest_predictor(self):
        latest_predictor = self.df_pred_metadata.sort_values(by='CreationTime', ascending=False).loc[0].to_dict()
        return latest_predictor

    def latest_predictor_in_tolerance(self, tolerance_days=2):
        train_time_elapsed_days = (
                datetime.datetime.now() - self.latest_predictor['CreationTime'].replace(tzinfo=None)
        ).days
        if train_time_elapsed_days < tolerance_days:
            return True
        else:
            return False

    def train_new_predictor(self):
        PREDICTOR_NAME = PREDICTOR_BASE_NAME + datetime.datetime.now().strftime(format='%Y_%m_%d_%H_%M')
        train_response = self.forecast_session.forecast.create_predictor(PredictorName=PREDICTOR_NAME,
                                                                         AlgorithmArn=ALGORITHM_ARN,
                                                                         ForecastHorizon=7,
                                                                         PerformAutoML=False,
                                                                         PerformHPO=False,
                                                                         InputDataConfig={
                                                                             "DatasetGroupArn": DATASET_GROUP_ARN},
                                                                         FeaturizationConfig={
                                                                             "ForecastFrequency": DATASET_FREQUENCY}
                                                                         )
        return train_response

    def create_latest_forecast(self):
        FORECAST_NAME = FORECAST_BASE_NAME + datetime.datetime.now().strftime(format='%Y_%m_%d_%H_%M')
        create_forecast_response = self.forecast_session.forecast.create_forecast(
            ForecastName=FORECAST_NAME,
            PredictorArn=self.latest_predictor['PredictorArn'])
        return create_forecast_response
```

ML Eng 103 - The Deployment



CI/CD Is Your Friend

- Github Actions, Jenkins, AWS CodePipeline/CodeBuild …
- Automate, automate, automate!
- Checks and balances like minimum test coverage or even data quality check passes

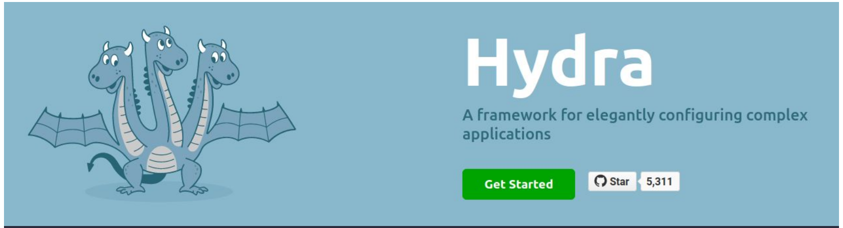



```yaml
# This is a basic workflow to help you get started with Actions

name: Upload DAGS to S3

# Controls when the action will run.
on:
  # Triggers the workflow on push or pull request events but only for the main branch
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

jobs:
  deploy:
    name: Upload DAGS to Amazon S3
    runs-on: ubuntu-latest

    steps:
    - name: Checkout
      uses: actions/checkout@v2

    - name: Configure AWS credentials from account
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: us-east-1

    - name: Copy files to bucket with the AWS CLI
      run: |
        aws s3 cp ./dags s3://github-actions-ci-cd-tests --recursive --include "*.py"
```

ML Eng 104 - The Deployment (Configuration)

Configuration Is Your Other Friend

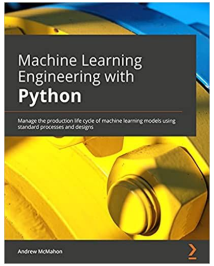
- I love YAML!
- Separate out what is instance specific and what is generic application logic/code/modelling
- Reduce complexity of deployments needed for configuration changes!



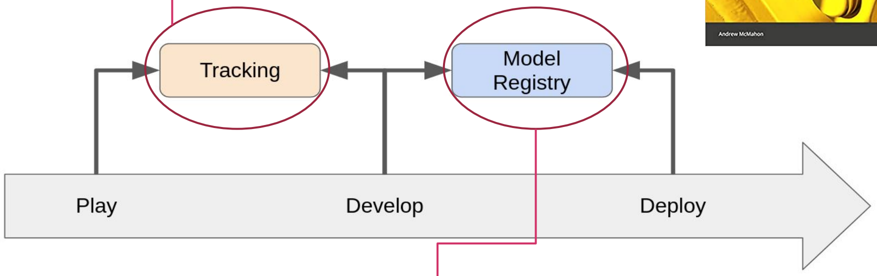

```python
@dataclass
class MySQLConfig:
    host: str = "localhost"
    port: int = 3306

@dataclass
class UserInterface:
    title: str = "My app"
    width: int = 1024
    height: int = 768

@dataclass
class MyConfig:
    db: MySQLConfig = MySQLConfig()
    ui: UserInterface = UserInterface()

cs = ConfigStore.instance()
cs.store(name="config", node=MyConfig)

@hydra.main(config_path=None, config_name="config")
def my_app(cfg: MyConfig) -> None:
    print(f"Title={cfg.ui.title}, size={cfg.ui.width}x{cfg.ui.height} pixels")

if __name__ == "__main__":
    my_app()
```

Getting The (n+1)th Model into Production

MLOps 101 - Model Management

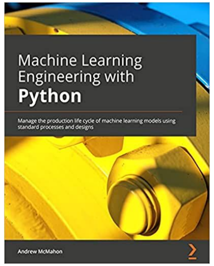


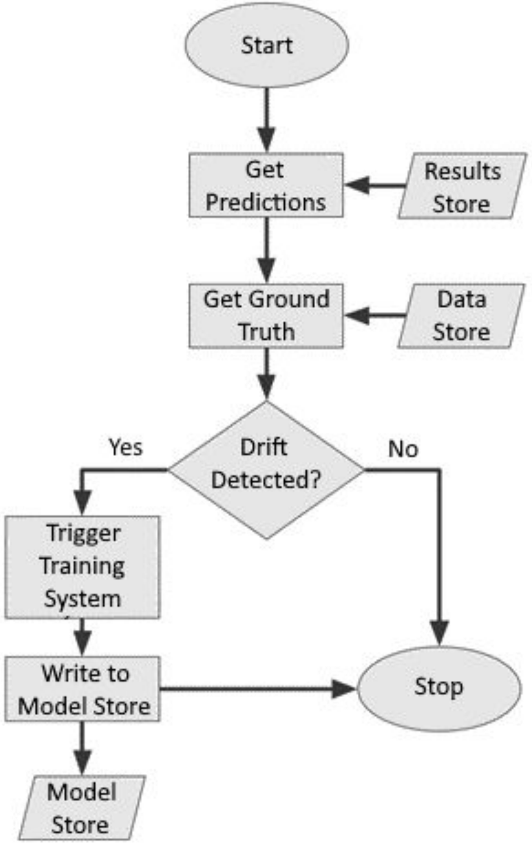

```python
with mlflow.start_run(run_name="YOUR_RUN_NAME") as run:
    params = {
        'tol': 1e-2,
        'solver': 'sag'
    }
    # Fit a ridge classifier after performing standard scaling
    std_scale_clf = make_pipeline(StandardScaler(), RidgeClassifier(**params))
    std_scale_clf.fit(X_train, y_train)
    y_pred_std_scale = std_scale_clf.predict(X_test)

    mlflow.log_metrics(
        {
            'accuracy': metrics.accuracy_score(y_test, y_pred_std_scale),
            'precision': metrics.precision_score(y_test, y_pred_std_scale, average='macro'),
            'f1': metrics.f1_score(y_test, y_pred_std_scale, average='macro'),
            'recall': metrics.recall_score(y_test, y_pred_std_scale, average='macro')
        }
    )

    mlflow.log_params(params)

    # Log the sklearn model and register as version 1
    mlflow.sklearn.log_model(
        sk_model=std_scale_clf,
        artifact_path="sklearn-model",
        registered_model_name="sk-learn-std-scale-clf"
    )
```




```python
# Transition the model stage to 'Staging'
client = MlflowClient()
client.transition_model_version_stage(
    name="sk-learn-std-scale-clf",
    version=1,
    stage="Staging"
)
# Transition the model stage to 'Production'
client = MlflowClient()
client.transition_model_version_stage(
    name="sk-learn-std-scale-clf",
    version=1,
    stage="Production"
)
```

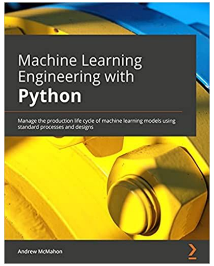
MLOps 102 - Performance Monitoring

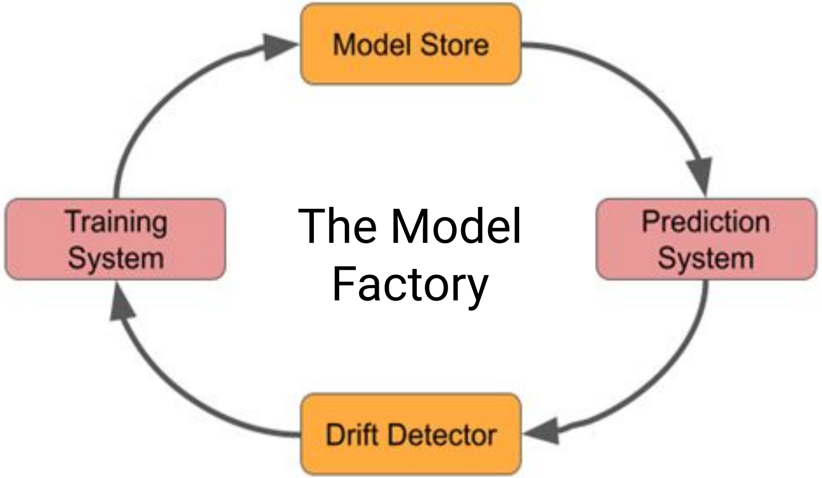




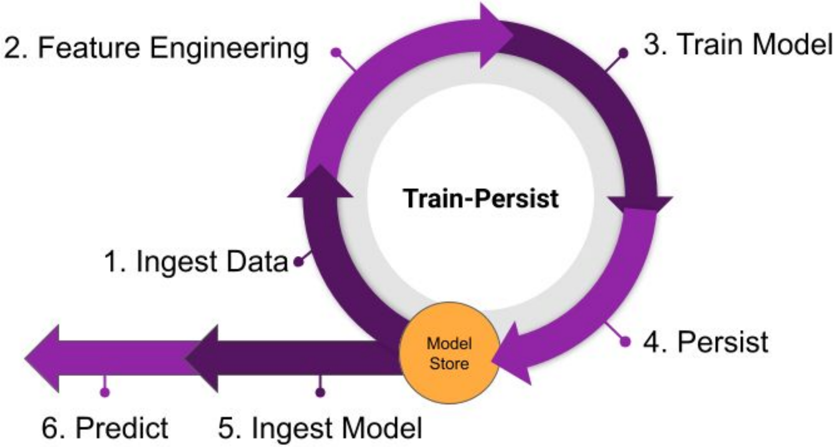

```python
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

import alibi
from alibi_detect.cd import ChiSquareDrift, TabularDrift
from alibi_detect.utils.saving import save_detector, load_detector
# Grab the data
wine_data = load_wine()
feature_names = wine_data.feature_names
X, y = wine_data.data, wine_data.target

# Make a 50/50 reference/test split
X_ref, X_test, y_ref, y_test = train_test_split(X, y,
                                                test_size=0.50,
                                                random_state=42)

# Initialise the detector
cd = TabularDrift(p_val=.05, X_ref=X_ref)
# Check for drift
preds = cd.predict(X_test)
labels = ['No', 'Yes']
print('Drift: {}'.format(labels[preds['data']['is_drift']]))
```

MLOps 103 - Bringing it Together



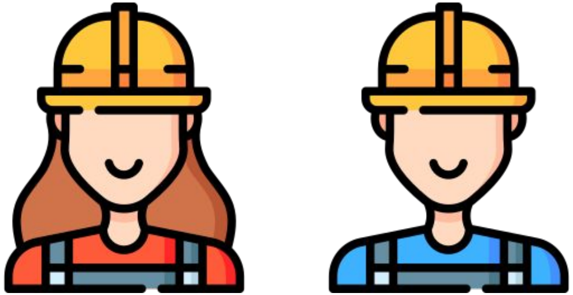


The Train-Persist Process

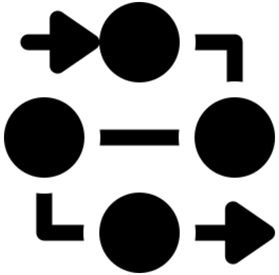


Other Tips & Tricks: The 4P's

The 4 P’s



People



Process

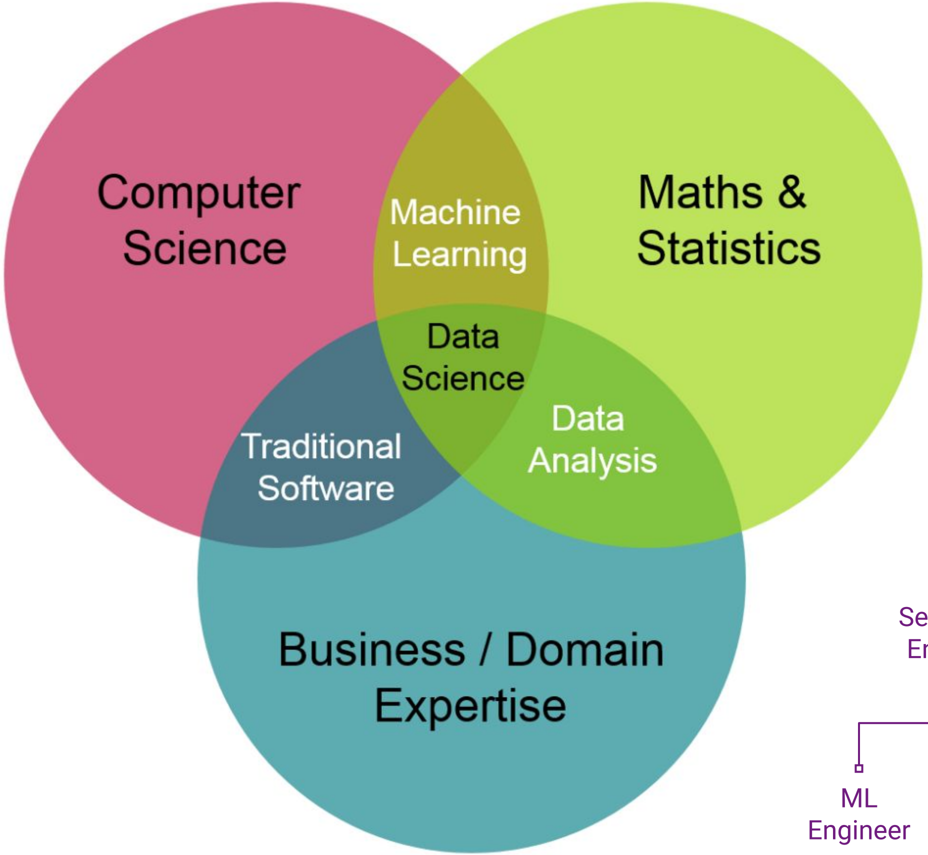
Product

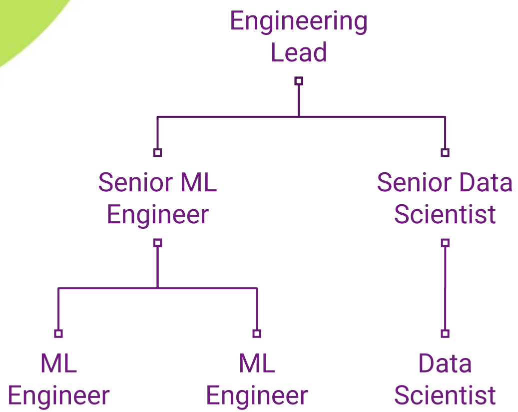
Pattern

People

People: Blended Teams > Unicorns

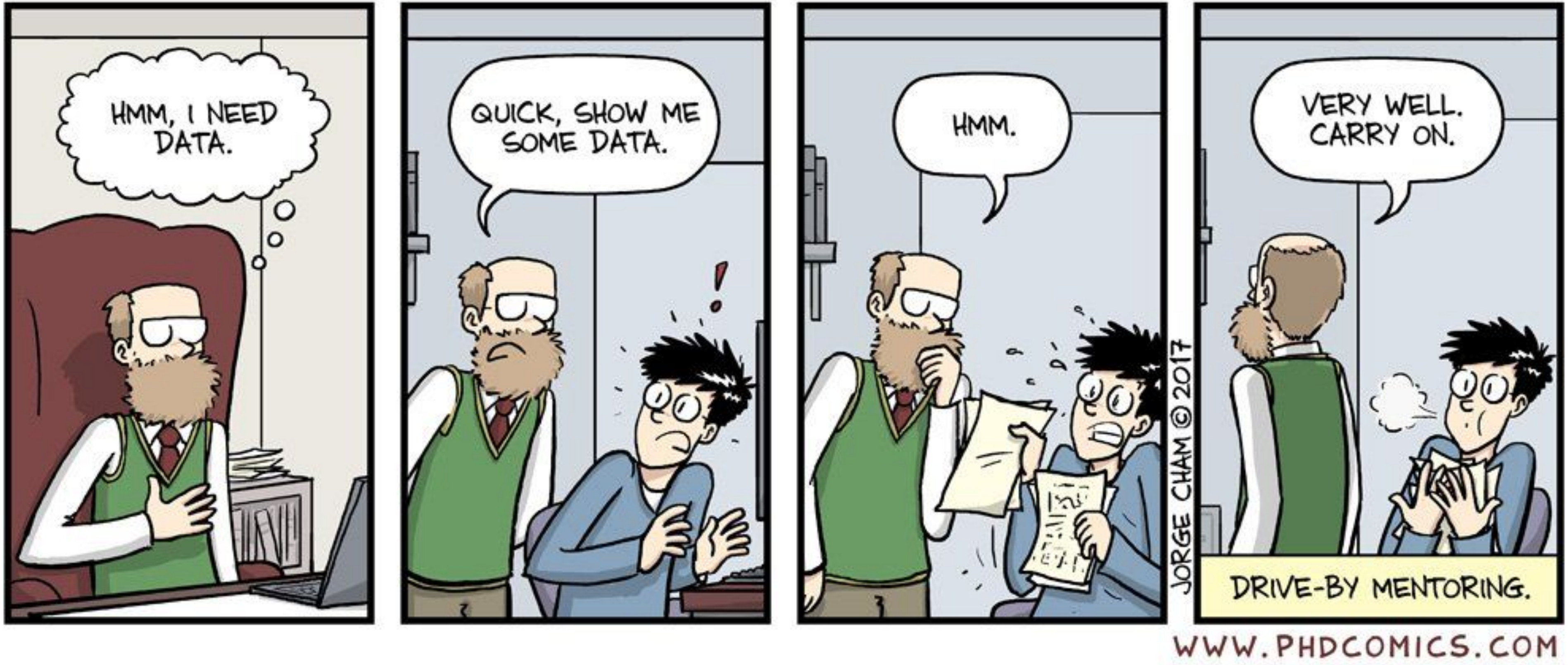
vs





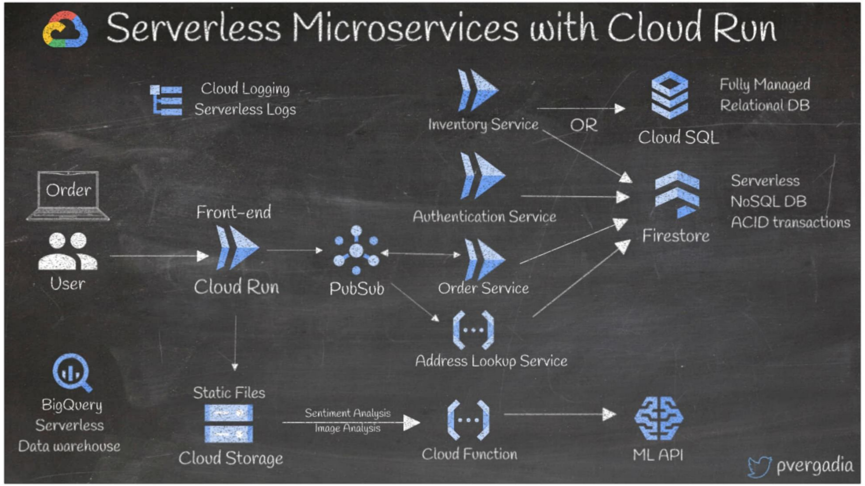
People: The Vision Should Be ...

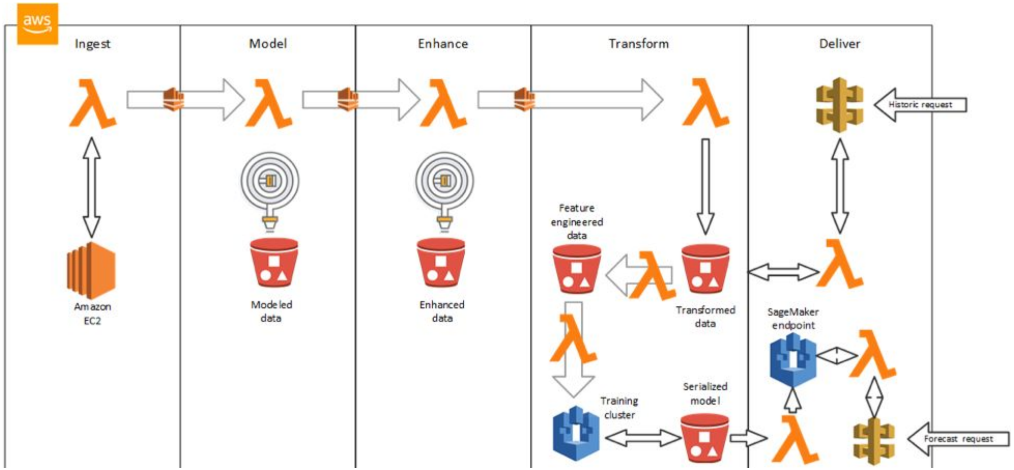
People: Know Your Customer



Patterns

Patterns: Reuse, Repeat, Recycle

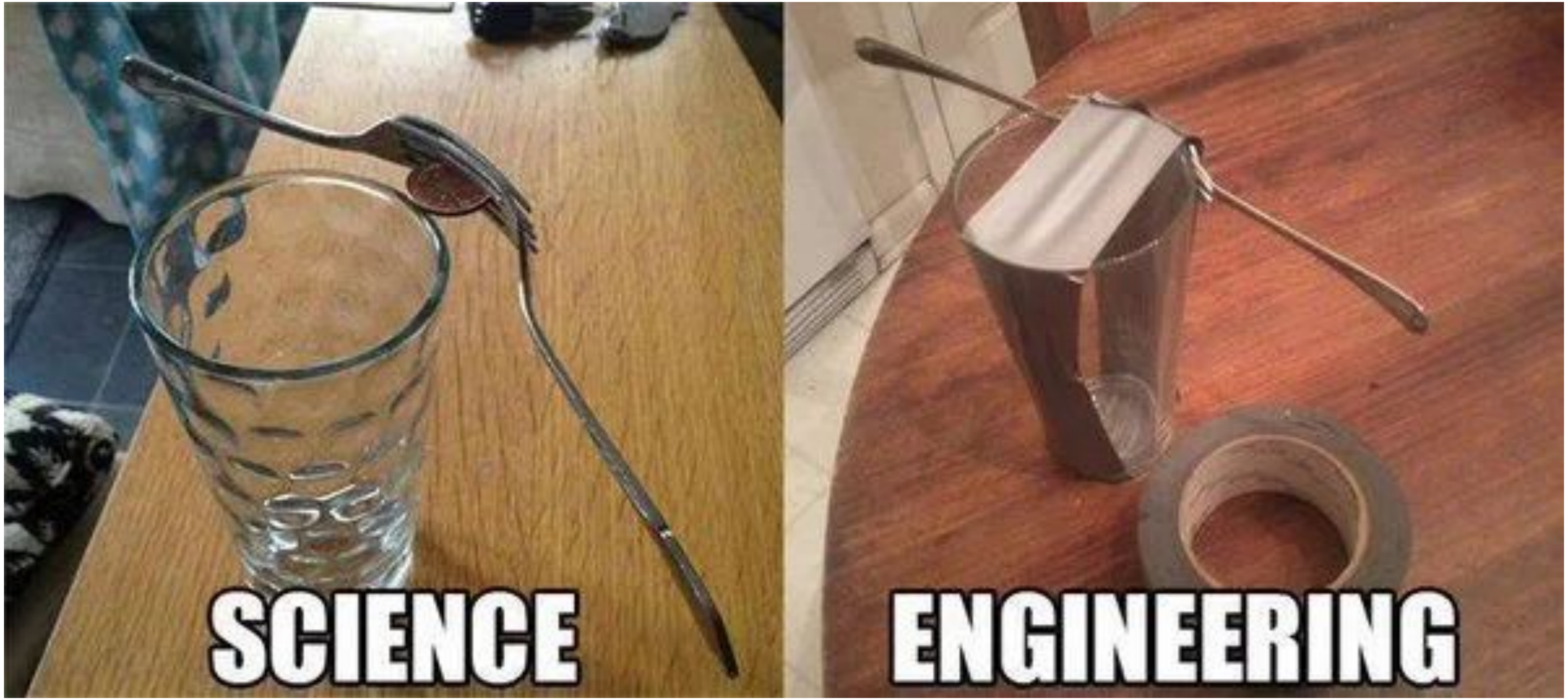




[13 sample architectures to kickstart your Google Cloud journey]

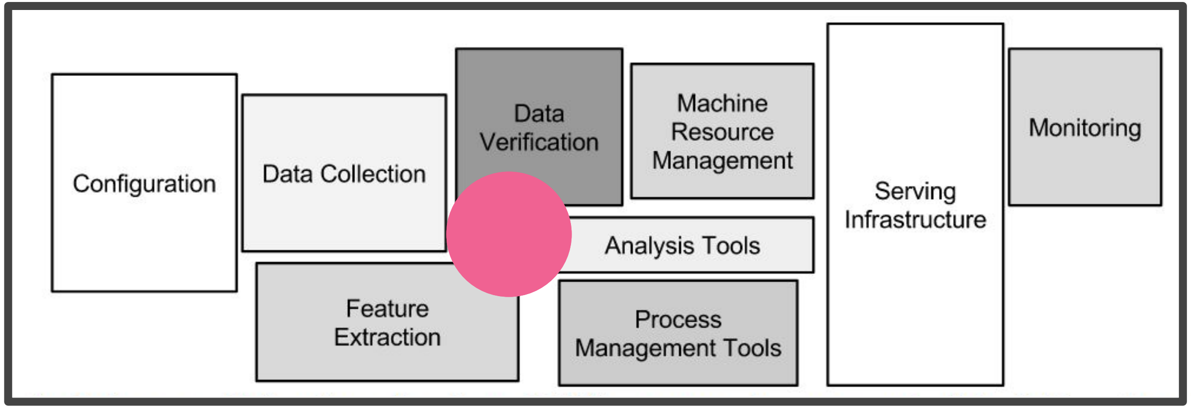
[Machine Learning Lens]
[AWS Well - Architected Framework]

Patterns: Science != Engineering



Product

Products != Models != Analyses



MLEng/MLOps

[Hidden Technical Debt in Machine Learning Systems, Google]

Products work: Test, test, test …

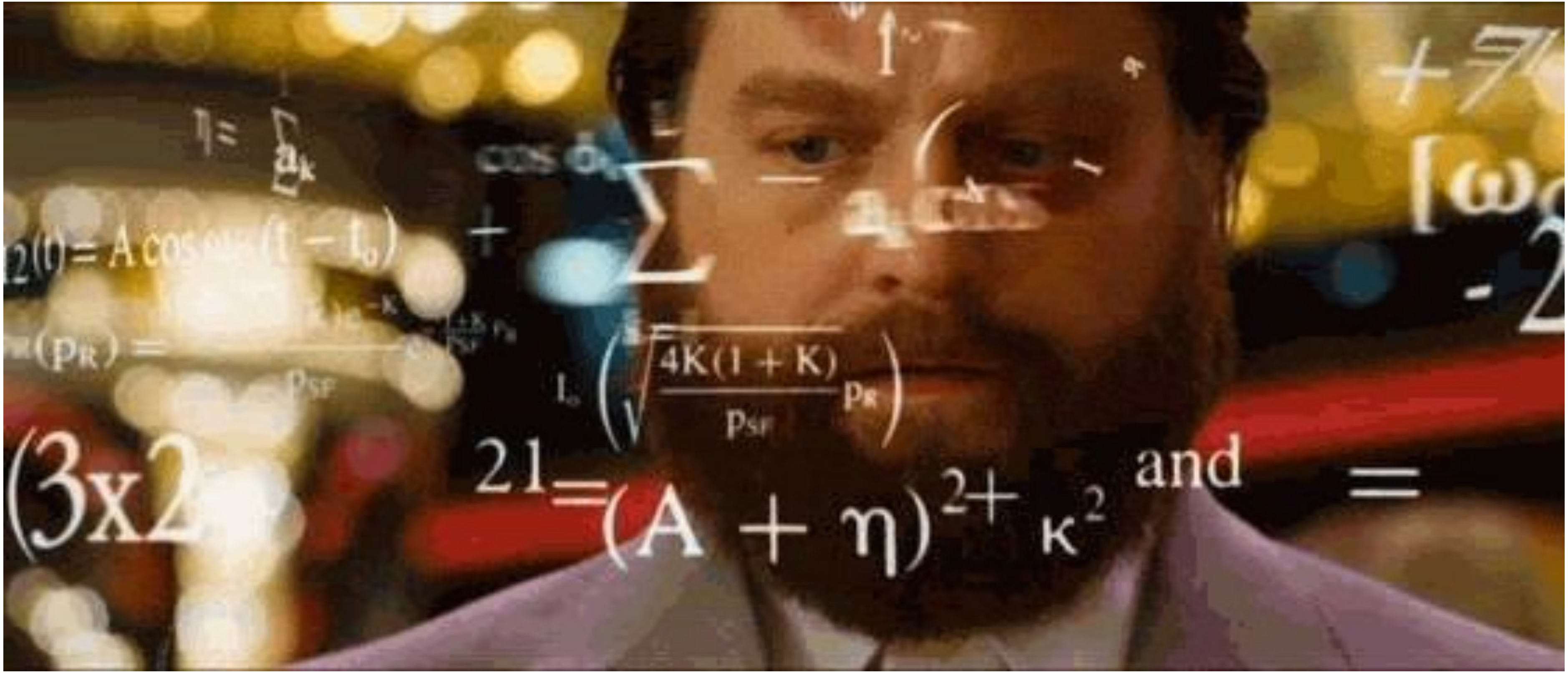
Products get delivered: Don’t reinvent the wheel …

Products come in a variety of shapes: Build for flexibility ...

Products != Models != Analyses

Process

Process: Machine Learning != Traditional Programming



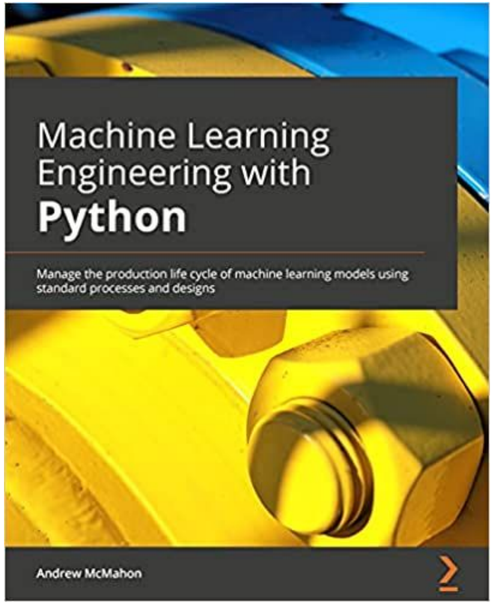
Thank you!

Some places you can find me (say hello!)

@electricweegie

AndyMc629

Andy McMahon

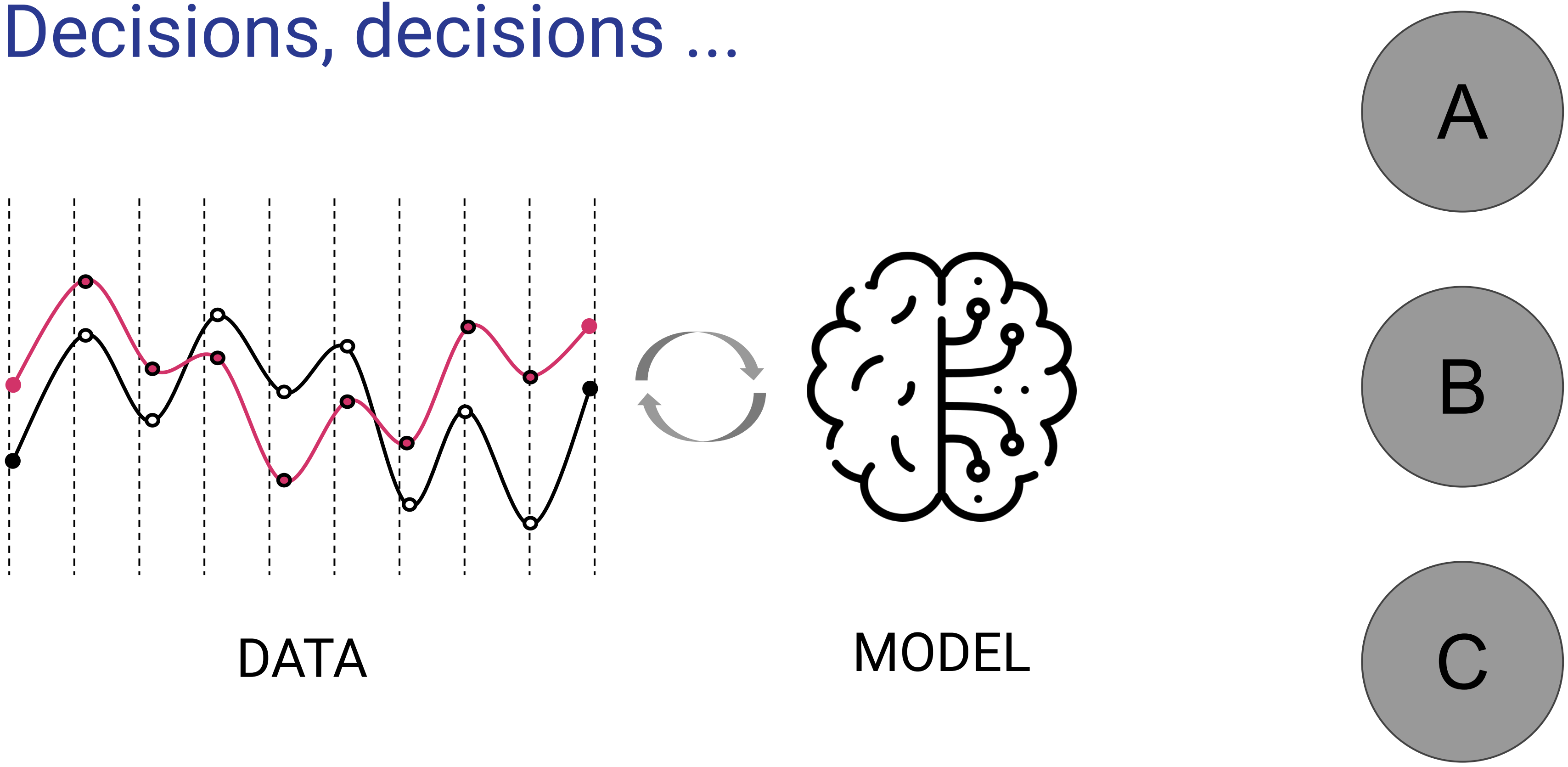


electricweegie.com

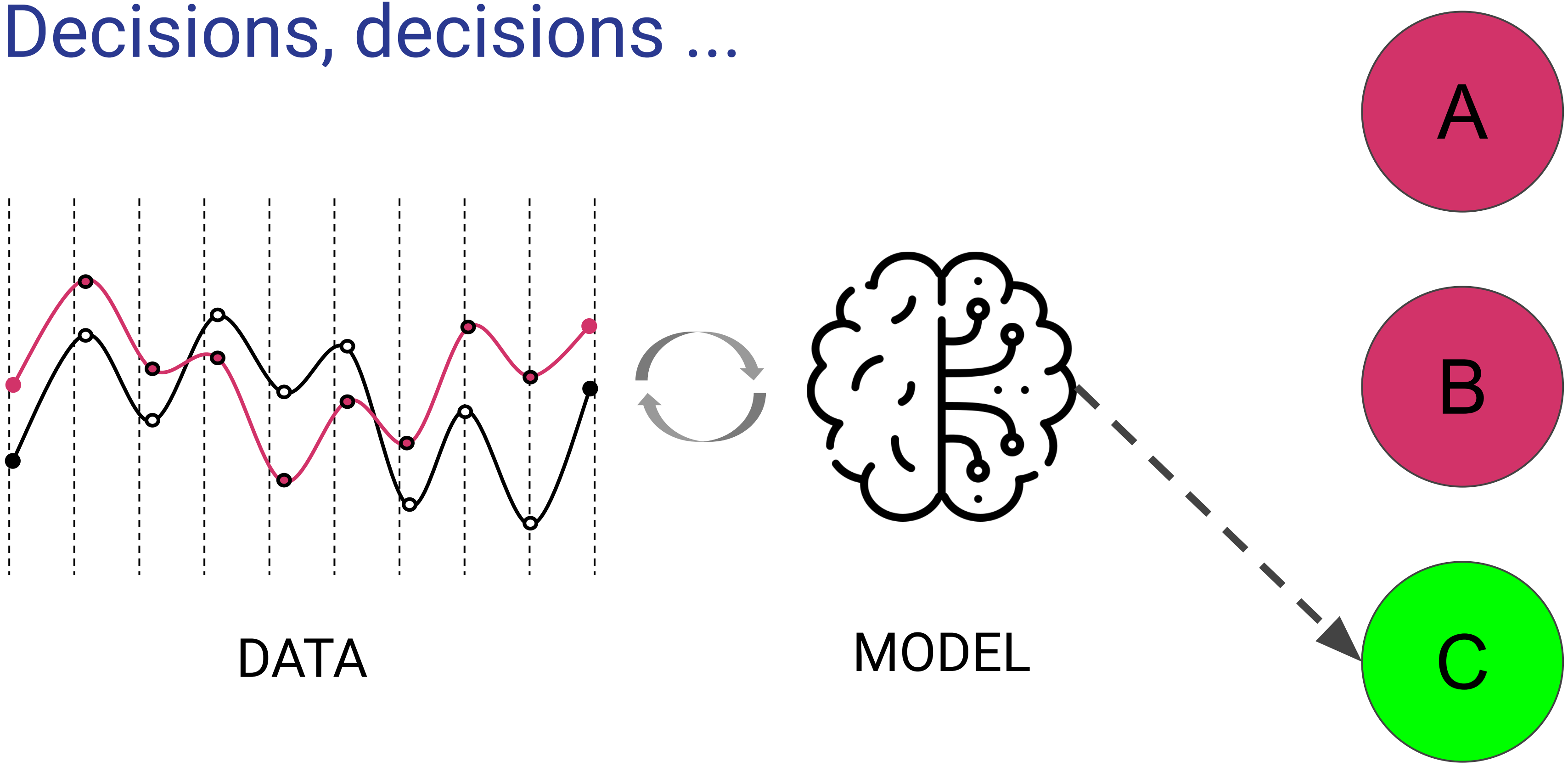
https://shows.acast.com/ai-right

Appendix

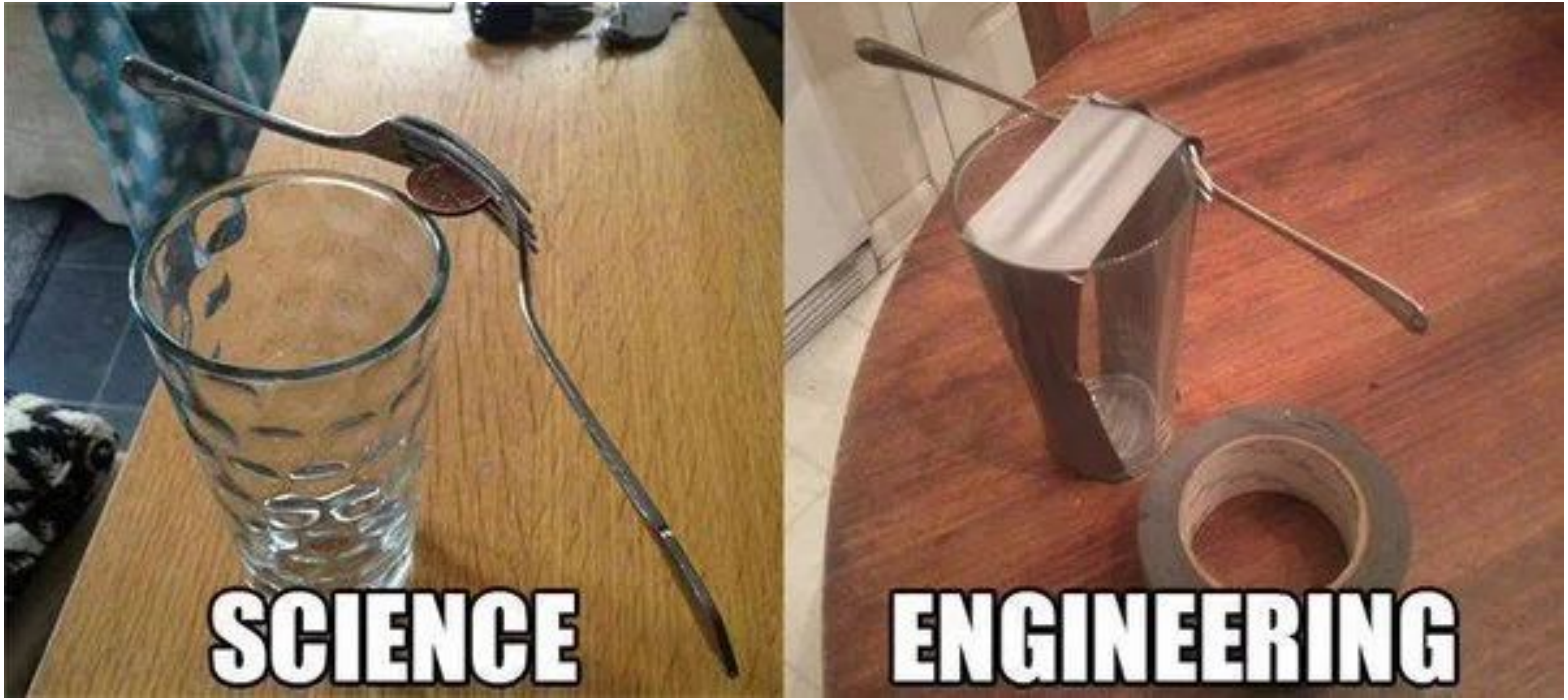
Decisions, decisions ...



Decisions, decisions ...



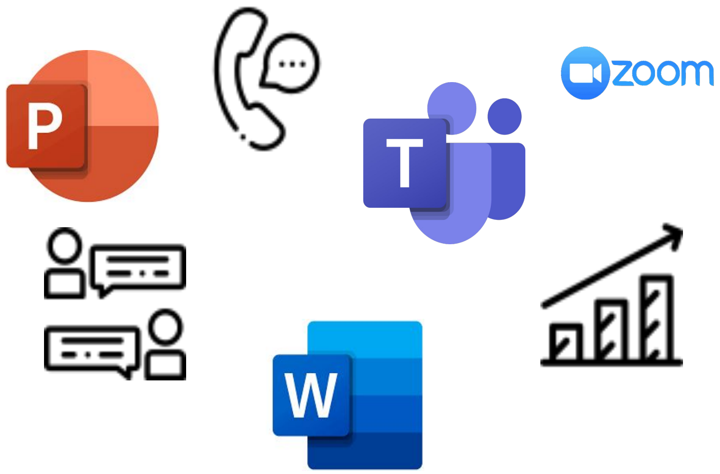
Science != Engineering



ML Dev Process

Discovery → PoC → Development → Deployment

- Work with subject matter experts and stakeholders
- Understand challenge
- Define end goal (KPIs, baseline performance)
- Sketch MVP implementation
- Summarise for sign off



ML Dev Process

Discovery → **PoC** → Development → Deployment

- Exploratory data analysis
- Identification of ‘no go’ scenarios (blockers for the project)
- Identification of other project risks
- Bare bones implementation
- Evidence that the solution is viable and will solve the problem (sign off again)

ML Dev Process

Discovery | PoC | **Development** | Deployment

- Dev sprints
- Git strategy
- Unit and Integration tests
- Model version control
- Baseline performance tests/results

ML Dev Process

Discovery → PoC → Development → **Deployment**

- Final packaging
- Deployment and update strategy
- Infrastructure instantiation
- Documentation completion
- Performance monitoring
- CI/CD pipeline definition

aws

kubernetes

Flask

docker

AWS Lambda

Azure

Google Cloud Platform