

# Bootcamp Bringing ML Models into Production

Lesson 4:  
ML Best practices



**pyladies**  
amsterdam

 DEXTER

Alyona Galyeva

# Agenda

- Recap
- ML Best practices
  - Scope
  - Code structure
  - Tests
  - Pipelines
  - ML system monitoring
- Capstone: Energy use case



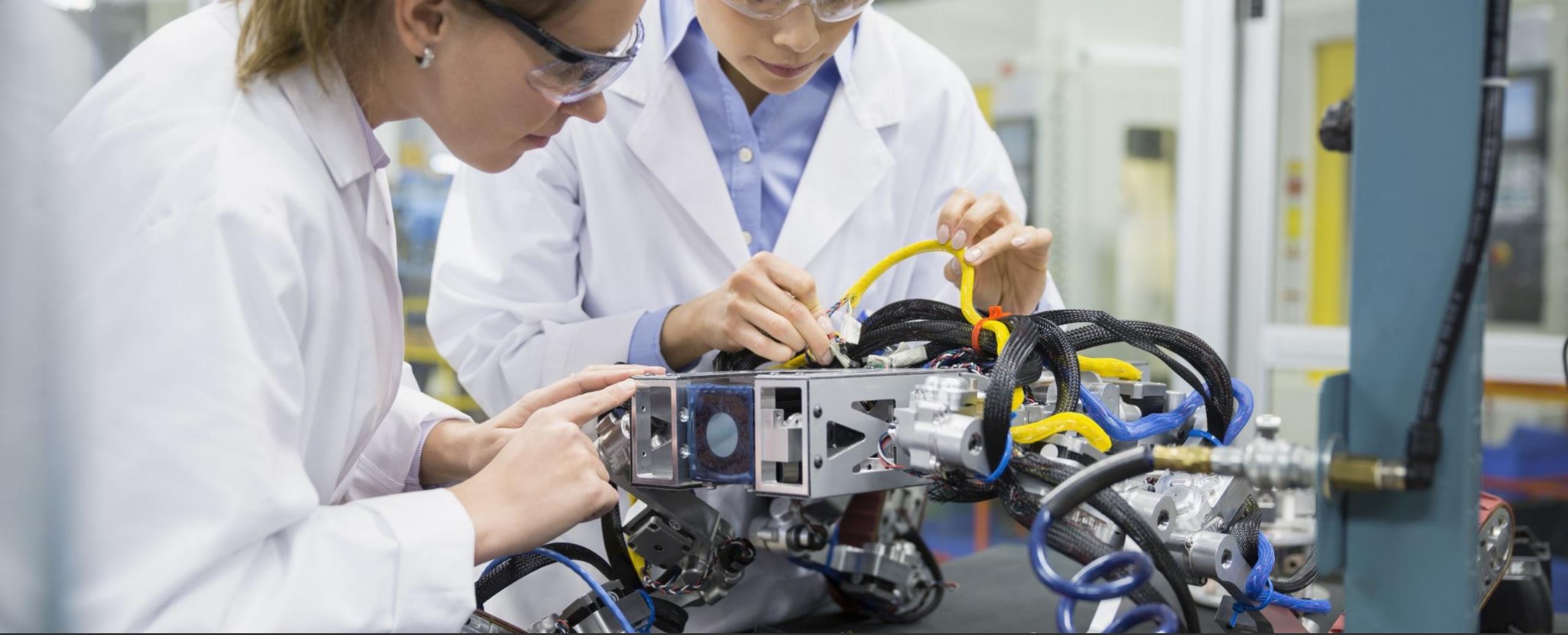
---

Recap

# Previous lesson

---

- Near real-time inference on Azure with Blob triggered Azure function:
  - create and deploy a simple Blob triggered Azure Function
  - develop the Blob triggered Azure Function locally with Azurite
  - deploy the Blob triggered Azure Function



# ML Best Practices



# Scope

# Scope

---

- define the main goal

# Scope

---

- define the main goal
- ML solution development:
  - visualize an ideal solution without any constraints
  - understand how the problem is solved
  - design with constraints in mind

# Scope

---

- define the main goal
- ML solution development:
  - visualize an ideal solution without any constraints
  - understand how the problem is solved
  - design with constraints in mind
- end to end MVP iterations



# Code structure

---

# notebooks vs modular code

```
## Multiple Linear Regression Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting Test Set
y_pred = regressor.predict(X_test)
from sklearn import metrics
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r2 = metrics.r2_score(y_test, y_pred)

results = pd.DataFrame([['Multiple Linear Regression', mae, mse, rmse, r2]],
                       columns = ['Model', 'MAE', 'MSE', 'RMSE', 'R2 Score'])

## Suport Vector Regression
'Necessary Standard Scaler '
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)

# Predicting Test Set
y_pred = regressor.predict(X_test)
from sklearn import metrics
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r2 = metrics.r2_score(y_test, y_pred)

model_results = pd.DataFrame([['Support Vector RBF', mae, mse, rmse, r2]],
                             columns = ['Model', 'MAE', 'MSE', 'RMSE', 'R2 Score'])
```

pin the libraries  
use docker

```
1 name: mlops_train
2 channels:
3   - defaults
4   - conda-forge
5 dependencies:
6   - python=3.8
7   - ipykernel
8   - matplotlib
9   - pandas
10  - scipy
11  - scikit-learn
12  - pip:
13    - azureml-core
14    - azureml-dataset-runtime[fuse]
15    - azureml-pipeline-core
16    - azureml-pipeline-steps
```

# docs, type hinting, pre-commit

---

```
def my_func(a, b):
    return a.lower() + b.upper()

#Rename function and its parameters
def mix_lower_upper_case(first_param, second_param):
    return first_param.lower() + second_param.upper()

#We can add type hints for readability purposes
def mix_lower_upper_case(first_param: str, second_param: str) -> str:
    """ Concatenate two strings, cast the first string to lowercase and the second string to uppercase"""
    return first_param.lower() + second_param.upper()

# Add casting to a str to make sure that the function returns what expected
def mix_lower_upper_case(first_param: str, second_param: str) -> str:
    """ Concatenate two strings, cast the first string to lowercase and the second string to uppercase"""
    return str(first_param).lower() + str(second_param).upper()
```

# print vs logging, error handling

---

```
1 def implementDivision(a, b):  
2     try:  
3         value = a/b  
4         print(value)  
5     except FileNotFoundError:  
6         print("The File Cannot be Found")  
7     except ZeroDivisionError:  
8         print("Number Cannot be Divided by Zero")  
9     except:  
10        print("This is the Generic Error")
```

# sys, argparse, configargparse, click, typer

---

```
parser = argparse.ArgumentParser()
parser.add_argument('--ff', '--feature_file', type=str, required=True)
parser.add_argument('--fcf', '--feature_class_file', type=str, required=True)
parser.add_argument('--of', '--outcome_file', type=str, required=True)
parser.add_argument('--op', '--output_prefix', type=str, required=True)
parser.add_argument('--m', '--model', type=str, required=True, choices=MODEL_NAMES)
```

# automation

---

```
remove-environment:  
    @echo ""  
    @echo "$(ccso)--- Remove conda environment $(ccend)"  
    conda env remove --name $(CONDA_ENV_NAME)  
  
remove-kernel:  
    @echo ""  
    @echo "$(ccso)--- Remove ipykernel from Jupyter lab $(ccend)"  
    $(CONDA_ACTIVATE) && jupyter kernelspec uninstall $(CONDA_ENV_NAME)
```

## Tests



# unit, integration, functional, regression tests

```
# positive case
def test_filter_not_equal_2():
    input_df = pd.DataFrame([(1, 2), (2, 8)], columns=["A", "B"])
    test_df = filter_not_equal(input_df, "B", 2)
    expected_df = pd.DataFrame([(2,8)], columns=["A", "B"])
    pd.testing.assert_frame_equal(test_df.reset_index(drop=True), expected_df.reset_index(drop=True))

# positive case
def test_filter_not_equal_0():
    input_df = pd.DataFrame([(1, 2), (2, 8)], columns=["A", "B"])
    test_df = filter_not_equal(input_df, "B", 0)
    expected_df = pd.DataFrame([(1, 2), (2, 8)], columns=["A", "B"])
    pd.testing.assert_frame_equal(test_df.reset_index(drop=True), expected_df.reset_index(drop=True))

# negative case
def test_filter_column_not_exist():
    with pytest.raises(KeyError):
        input_df = pd.DataFrame([(1, 2), (2, 8)], columns=["A", "B"])
        test_df = filter_not_equal(input_df, "C", 0)

def filter_not_equal(df: pd.DataFrame, column_name: str, val: int) -> pd.DataFrame:
    df = df.copy()
    new_df = df.loc[df[column_name] != val]
    return new_df
```

# data-specific tests

```
@MetaPandasDataset.column_map_expectation
def expect_column_value_word_counts_to_be_between(self, column, min_value=None, max_value=None):
    def count_words(string):
        word_list = re.findall("(\\S+)", string)
        return len(word_list)

    word_counts = column.map(lambda x: count_words(str(x)))

    if min_value is not None and max_value is not None:
        return word_counts.map(lambda x: min_value <= x <= max_value)
    elif min_value is not None and max_value is None:
        return word_counts.map(lambda x: min_value <= x)
    elif min_value is None and max_value is not None:
        return word_counts.map(lambda x: x <= max_value)
    else:
        return word_counts.map(lambda x: True)
```

# model-specific tests

```
def test_model_return_object():
    """
    Tests the returned object of the modeling function
    """
    X,y = random_data_constructor()
    scores = train_linear_model(X,y)

    #=====
    # TEST SUITE
    #=====

    # Check the return object type
    assert isinstance(scores, dict)

    # Check the length of the returned object
    assert len(scores) == 2

    # Check the correctness of the names of the returned dict keys
    assert 'Train-score' in scores and 'Test-score' in scores
```

Generates data  
and runs the  
function

Focused test suite  
checking returned  
object properties

# Pipelines



# Pipelines

---

- DataOps (get data, join, validate, prepare, split, feature engineering)

# Pipelines

---

- DataOps (get data, join, validate, prepare, split, feature engineering)
- Model training

# Pipelines

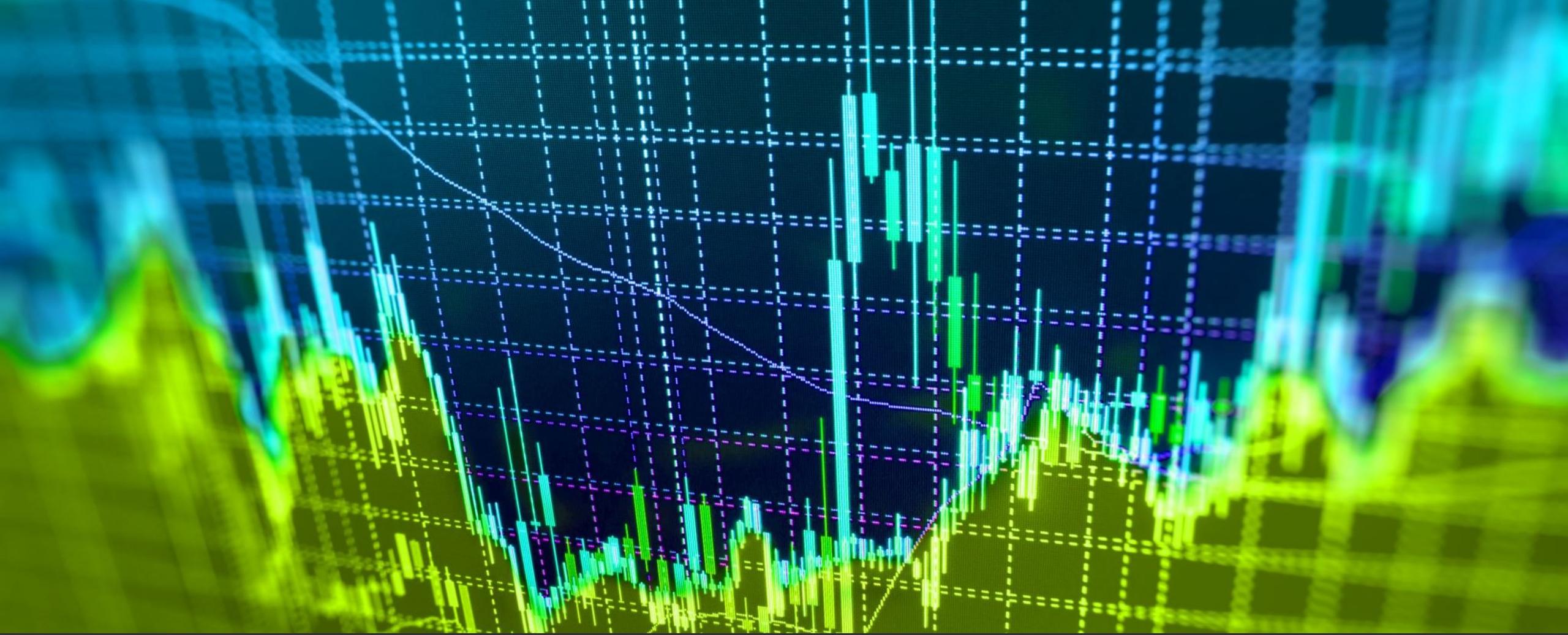
---

- DataOps (get data, join, validate, prepare, split, feature engineering)
- Model training
- Model evaluation (performance, latency, size, compute, interpretability, bias, time to develop, time to retrain, maintenance overhead)

# Pipelines

---

- DataOps (get data, join, validate, prepare, split, feature engineering)
- Model training
- Model evaluation (performance, latency, size, compute, interpretability, bias, time to develop, time to retrain, maintenance overhead)
- **Model deployment (build, release pipelines)**



ML system monitoring

# ML system monitoring

---

- overall ML system health
- model performance (data drift, feature drift, concept drift)
- monitoring solution (alert, inspect, act)

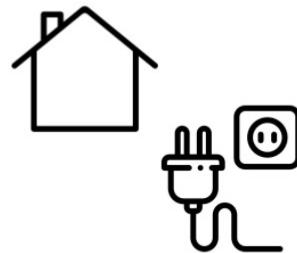


Capstone: Energy use case

## Dexter - PyLadies case

### OLD APPROACH

Determine  
energy consumption

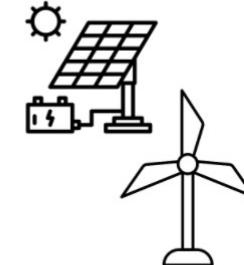


Adjust  
energy generation

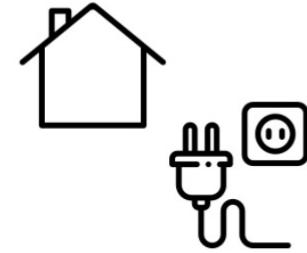


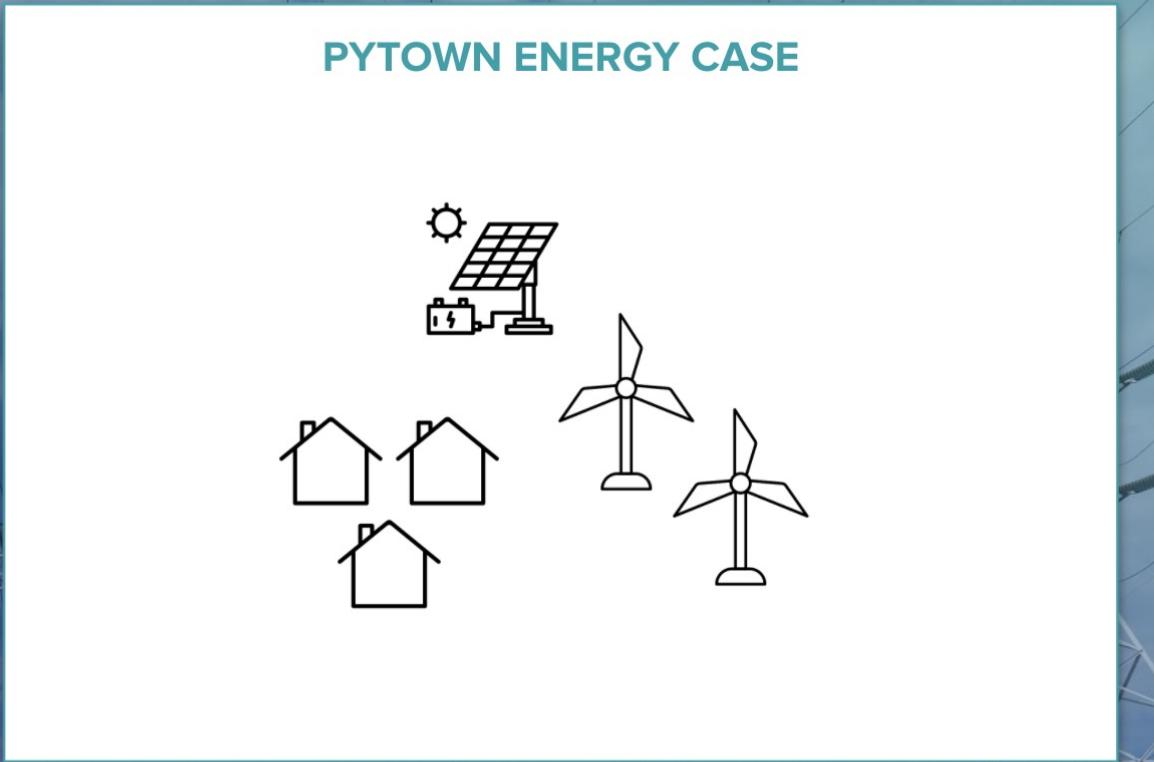
### NEW APPROACH

Determine  
energy generation



Adjust  
energy consumption





## Dexter-Pyladies energy case

Pytown is a village in the North of the Netherlands that wants to become fully self-sufficient with 100% renewable energy.



The village has built sufficient solar panels & wind turbines to produce their own energy demand.



The timing of the energy generation does not match the timing of demand. This raises a lot of problems for Pytown. Frequent blackouts are the order of the day. Can you help Pytown to adjust their demand to the energy generation?



# A dashboard can help Pytown to adjust their energy demand

The residents of Pytown need to be informed to be able to adjust their energy consumption



**Stop!**

Only use absolutely essential electricity

**Slow down!**

Use less electricity than you would normally use

**Go!**

Use as much electricity as you want



# First capstone task: Develop a solution to prevent power blackouts

Provide a weekly forecast in your dashboard

## Data processing

3 data sources:  
- energy load  
- wind & solar predictions  
- weather data

## Model training

Weekly forecast:  
- naive forecast  
- linear regression with Fourier features

## Evaluation

Optimize on MAPE with a sliding window approach

## Deployment

Deploy the best model as Azure Machine Learning Batch pipeline, scheduled every Monday at 4am

## Post-processing

Classify the advice on energy consumption & store in Azure Blob Storage

## Monitoring

Show your advice on energy consumption in your Power BI dashboard



## Second capstone task: Improve the solution to prevent blackouts

Prevent more blackouts by updating your prediction every 15 minutes

### Data processing

2 data sources:  
- energy load  
- wind & solar predictions

### Model training

Quarterly forecast:  
- naive forecast

### Evaluation

Optimize on MAPE with an expanding window approach

### Deployment

Deploy the model as blob-triggered Azure function that fires every 15 minutes

### Post-processing

Classify the advice on energy consumption & store in Azure Blob Storage

### Monitoring

Show your advice on energy consumption in your Power BI dashboard



## Working at Dexter



- Dynamic and young start-up
- Challenging software development & data science
- Contributing to the transition to renewable energy
- Fun team

Website: <https://dexterenergy.ai/>

Feel free to contact us with any questions

# Capstone

---

<https://github.com/pyladiesams/bootcamp-bringing-ML-models-into-production-intermediary-jun-aug2021/blob/master/bootcamp/capstone/capstone.md>



---

```
print(f"{user_name} thanks for watching")
```