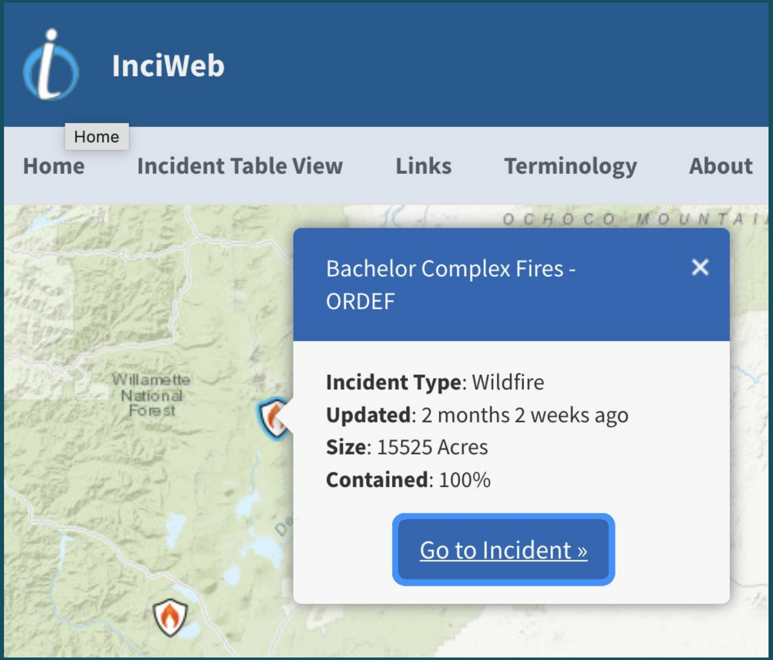
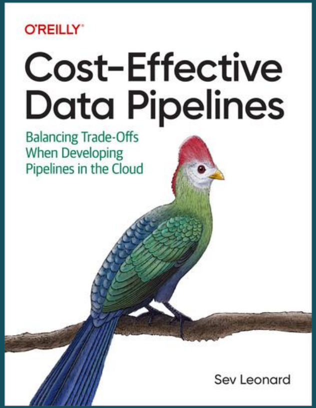
Sev Leonard
he/him/his
PyCascades 2025

In this talk I will…





Addressing forest fire risk

Prevention

- Defensible space
- Fire-resistant building materials

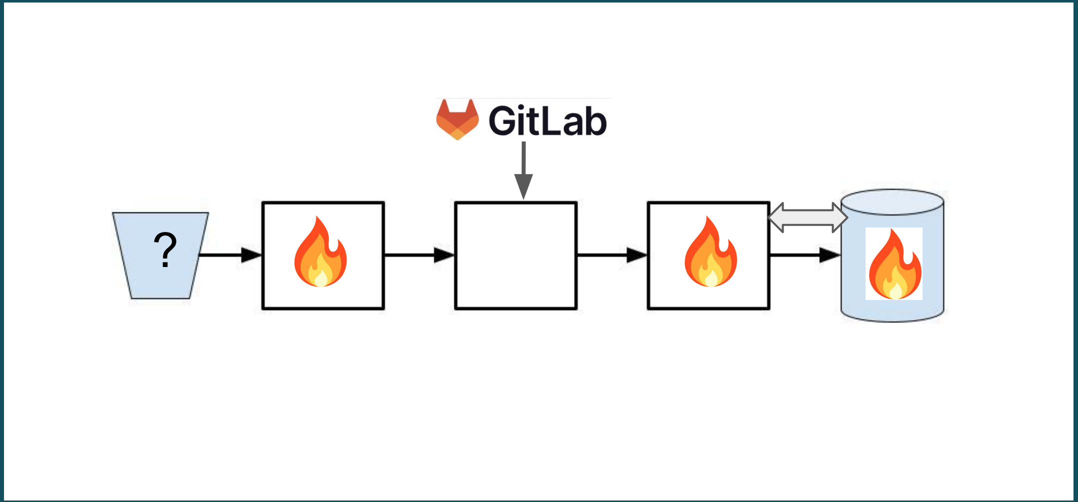
Monitoring

- BC Wildfire Map, IncіWeb (US)
- PurpleAir

Alerting

- OR Alerts, WA Alerts, BC Wildfire Service

Addressing data fire risk



Prevention

- Data validation
- Automated unit testing
- Defensive design

Monitoring

- Data quality
- Pipeline operation

Alerting

- Slack, PagerDuty

Data Quality is a system-wide endeavor

Create defensible space with data validation

What changes in data could cause pipeline failure or degrade data quality?

```python
from jsonschema import validate

schema = {
    "items": [{
        "description": {
            "type": "string"
        },

        "required": ["description"]
    }]
}
```

- Schema
- Catch bad data
- Descriptive logs

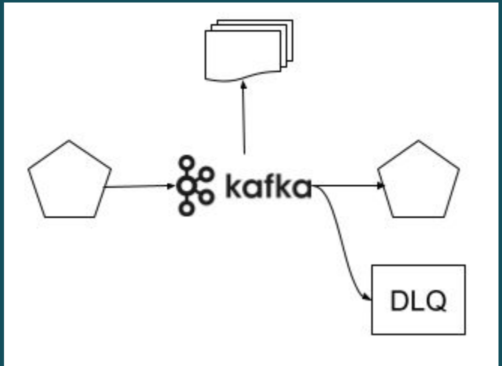
```
>>> validate([{}], schema)
[redacted]
...
jsonschema.exceptions.ValidationError: 'description' is a required property

>>> validate([{"description": 1234}], schema)
...
jsonschema.exceptions.ValidationError: 1234 is not of type 'string'
```

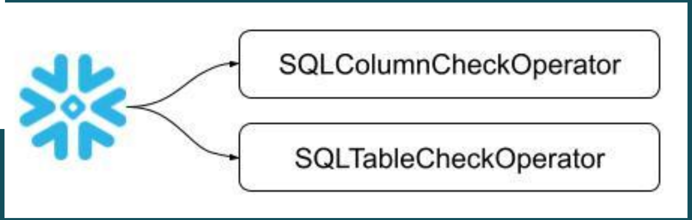
PySpark

```
spark.read.json(data_source,
                mode="PERMISSIVE",
                columnNameOfCorruptRecord="_corrupt_record")
```

	_corrupt_record	location	user
0	None	[26.91756, -82.07842]	pc@cats.xyz
1	None	[45.2341, 121.2351]	lucy@cats.xyz
2	{'user': 'scout@cats.xyz', 'location': [45.2341,}	None	None



Apache Airflow



databricks

```
@dlt.expect_or_drop("constraint_name", "id IS NOT NULL")
```

dbt

```
sources:
  - name: data_source
    tables:
    - name: source_table
      columns:
        - name: description
          tests:
            - not_null
```

Create defensible space with automated unit testing

Automation barriers:

- Code structure
- Maintaining test data

```
(spark
    .read(...)
    .withColumn(...)
    .withColumn(...)
    .drop(...)
    .groupBy(...)
    .agg(...)
    .write(...)
)
```

Benefits

- Readable
- Easy to change

Unit Testing

- Create a source file
- Read result file
- Compare to expected results
- Thorough testing can be difficult

Changes

- All code open to modification

```python
def preprocess(df):
    return (df
        .withColumn(...)
        .withColumn(...)
        .drop(...)
    )

def aggregate(df):
    return (df
        .groupBy(...)
        .agg(...)
    )
```

Benefits

- No files to manage
- Retain chaining ability

Unit Testing

- Compare test and expected dataframes
- Functions tested independently

Changes

- Changes isolated

```python
def a_good_dag():
    @task
    def get_configs():
        # Query an API
        # Get customer configurations
        # Map to a configs object
        return configs

    @task
    def do_data_stuff(configs):
        # Perform data stuff for all
        # customer configurations

    configs = get_configs()
    do_data_stuff(configs)

result = a_good_dag()
```

```python
from dag_tasks import get_configs, process_data

def a_good_dag():
    @task
    def get_configs():
        return get_configs()

    @task
    def do_data_stuff(configs):
        process_data(configs)

    configs = get_configs()
    do_data_stuff(configs)

result = a_good_dag()
```

Defensive design - Retries

Self-heal from intermittent issues

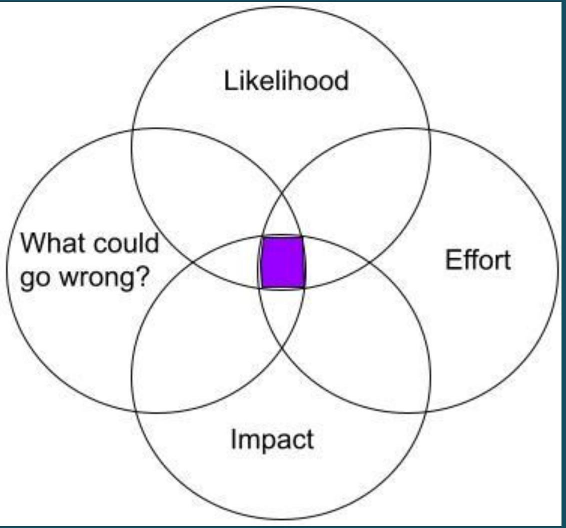
- Low level - resource interactions
- Task - retry from checkpoint
- Pipeline

Sending emails

```
@tenacity.retry(
    retry=retry_if_exception_type(ConnectionError),
    wait=wait_exponential(max=120),
    stop=stop_after_attempt(10),
    before_sleep=before_sleep_log(logger, logging.DEBUG),
    reraise=True,
)
def send_cust_email():
    ...
    requests.post(url=endpoint, json=request_body)


@task(retries=2, retry_delay=timedelta(minutes=20))
def send_email():
    send_cust_email()
```

Developing your custom data quality plan



Data Quality Plan - Clinical Trials Data Platform

- Used to come up with treatment options for patients
- Impact of poor data quality - degraded patient care
- System Considerations
  - Small data set
  - Postgres/SQL ELT
- Data Quality plan
  - Data validation - expected values, null checks
  - Manual adjudication of bad records
  - Automated unit testing
  - Manual data quality review

Data Quality Plan - Cybersecurity startup

- Filters cyber threats to only what is relevant for a customer
- Impact of poor data quality - some erosion of trust, adoption
- System considerations
  - Large scale data
  - Data from many different APIs, could change unexpectedly
  - Python, SQL, Airflow
- Data Quality plan
  - Validation on internal APIs only - schema match
  - Some automated unit tests
  - Automated retries

Thanks!

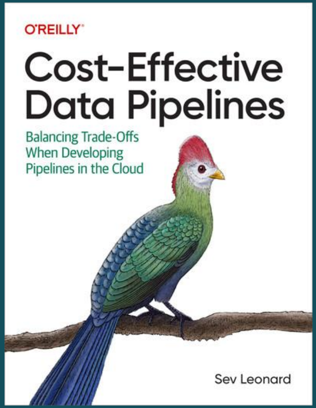
sev@thedatascout.com

https://www.linkedin.com/in/sevleonard/

https://github.com/gizm00/oreilly_dataeng_book/

https://pdxwlf.com/

PORTLAND WINTER LIGHT FESTIVAL
ANNIVERSARY
FEB 7-15, 2025
A WILLAMETTE LIGHT BRIGADE EVENT



References / Backup

References - Data Validation

Astronomer guide on data quality checks in Airflow

Databricks Data Quality Management Guide

Example dbt commands for a production deployment

dbt Data Quality Framework

Confluent Schema Registry

References - Unit Testing

Databricks unit testing for notebooks

Unit testing in Airflow

Unit testing models in dbt

Github actions for CI/CD testing Databricks Notebooks (preview)

Image sources

“Only you” image - https://arnoldzwicky.org/2016/02/01/only-you/