Productionizing Jupyter Notebooks

Antoni Ivanov
Versatile Data Kit Open Source Project
The role of Jupyter in the data world


https://odsc.medium.com/why-you-should-be-using-jupyter-notebooks-ea2e568c59f2
The role of Versatile Data Kit (VDK) in the data world

Develop

VDK SDK

```
extract_load_rest_calls.py

def run(job_input):
    response = requests.get("https://rest.com/calls")
    payload = response.json()

    job_input.send_object_for_ingestion(
        payload=payload,
        destination_table="rest_target_table")
```

```
transform_salesmart.sql

Insert into {mart_schema}.{sales_table}
SELECT
    s.product_id,
    s.transaction_date,
    s.quantity_sold * p.product_price
FROM {raw_schema}.{sale_transaction_table} as s
JOIN {raw_schema}.{products_table} p using product_id
```

Deploy and Monitor

Control Plane and Operations UI

https://github.com/vmware/versatile-data-kit
Jupyter
From Data Exploration to Production
Challenges

- Reproducibility: Non-Linear Execution and Hidden State Risks
- Code Organization: Irrelevant or debugging code
- Execution model: interactive kernel vs automated flow
- Automated Testing and CICD
- Version Control
Reproducibility: Non-Linear Execution and Hidden State Risks

\[
\begin{align*}
\text{co} &= \emptyset \\
\text{co} &= 1 \\
\text{co} &= 1
\end{align*}
\]
Reproducibility: Non-Linear Execution and Hidden State Risks
What can we do?

```python
import pandas as pd
# Read the data
url = "https://raw.githubusercontent.com/duyguHsmHsn/nps-data/main/nps_data.csv"
df = pd.read_csv(url)

df = df[df['User'] != 'testuser']

df.head()

<table>
<thead>
<tr>
<th>Date</th>
<th>User</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-01-01</td>
<td>miked97</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>lucy131</td>
<td>7</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david479</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david220</td>
<td>0</td>
</tr>
<tr>
<td>2023-01-02</td>
<td>alex467</td>
<td>9</td>
</tr>
</tbody>
</table>

job_input.send_tabular_data_for_ingestion(
    df.iter_tuples(index=False),
    destination_table="nps_data",
    column_names=df.columns.tolist()
)

%%wdsq
select * from nps_data

<table>
<thead>
<tr>
<th>Date</th>
<th>User</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-01-01</td>
<td>miked97</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>lucy131</td>
<td>7</td>
</tr>
</tbody>
</table>
```
What can we do?

Tagging VDK Cells

```python
import pandas as pd
# Read the data
df = pd.read_csv(url)

df = df[df['User'] != 'testuser']
df.head()
```

<table>
<thead>
<tr>
<th>Date</th>
<th>User</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-01-01</td>
<td>mike897</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>lucy131</td>
<td>7</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david479</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david220</td>
<td>0</td>
</tr>
<tr>
<td>2023-01-02</td>
<td>alex467</td>
<td>9</td>
</tr>
</tbody>
</table>

```python
job_input.send_tabular_data_for_ingestion(
    df.iterrows(index=False),
    destination_table="nps_data",
    column_names=df.columns.tolist()
)
```

```sql
\%vdksql
select * from nps_data
```

<table>
<thead>
<tr>
<th>Date</th>
<th>User</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-01-01</td>
<td>mike897</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>lucy131</td>
<td>7</td>
</tr>
</tbody>
</table>
What can we do?

Tagging VDK Cells

```
import pandas as pd

df = pd.read_csv(url)

df = df[df['User'] != 'testuser']

df.head()

job_input.send_tabular_data_for_ingestion(
    df.iterrows(index=False),
    destination_table="nps_data",
    column_names=df.columns.tolist()
)

wdksql
select * from nps_data
```

<table>
<thead>
<tr>
<th>Date</th>
<th>User</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-01-01</td>
<td>mike897</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>lucy131</td>
<td>7</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david479</td>
<td>5</td>
</tr>
<tr>
<td>2023-01-01</td>
<td>david220</td>
<td>0</td>
</tr>
<tr>
<td>2023-01-02</td>
<td>alex467</td>
<td>9</td>
</tr>
</tbody>
</table>
Reproducibility: Non-Linear Execution and Hidden State Risks

- Assign a "vdk" tag and a specific number to a cell.
- The number dictates the order in which the cell will be executed in production.

Benefits:

- Ensures only the tagged cells are executed, and in the determined sequence.
- Clearly defining the execution order.
- Detect when the current state is diverging from expected order.
- Test easily end-to-end before deployment (as we will see)
Challenges

✓ Reproducibility: Non-Linear Execution and Hidden State Risks

➢ Code Organization: Irrelevant or debugging code

➢ Execution model: interactive kernel vs automated flow

➢ Automated Testing and CICD

➢ Version Control
Code Organization: Irrelevant or debugging code

```python
[ ]: import pandas as pd
[ ]: url = "some-url"
    df = pd.read_csv(url)
[ ]: visualise(df)
```
Code Organization: Irrelevant or debugging code

VDK tags to the rescue again

```python
# Import all functions from the 'helper' module,
# which contains the necessary logic for classification and data visualization
from helper import visualize_data, classify_score

# Apply the classification function to the 'Score' column to determine the 'Type'
# Note: this cell might fail on its first run.
# If it does, simply run it again, and it should work as expected.
df.loc[:, 'Type'] = df['Score'].apply(classify_score)

# Check the DataFrame
df

# Visualise the types of users
visualize_data(df)

5.2 Data Ingestion

# Sending data for ingestion
job_input.send_tabular_data_for_ingestion(
    df.iteruples(index=True),
    destination_table="nps_data",
    column_names=df.columns.tolist()
)
```
Challenges

✓ Reproducibility: Non-Linear Execution and Hidden State Risks
✓ Code Organization: Irrelevant or debugging code
➢ Execution model: interactive kernel vs automated flow
➢ Automated Testing and CICD
➢ Version Control
Execution model: interactive kernel vs automated flow
Bad for automation, bad for being part of a workflow
Execution model: interactive kernel vs automated flow
Execution model: interactive kernel vs automated flow
Execution model: interactive kernel vs automated flow

- Reuse another notebook as a template (function)

```python
def run(job_input: IJobInput):
    args = dict(
        source_table="vm_new_data",
        target_table="dim_vm",
        timestamp_column="arrival_ts",
        id_column="vm_uuid",
    )
    job_input.execute_template("process-note-data-jupyter-notebook", args)
```

- Execute within a workflow

- Run automated tests (example coming later)
Challenges

✓ Reproducibility: Non-Linear Execution and Hidden State Risks
✓ Code Organization: Irrelevant or debugging code
✓ Execution model: interactive kernel vs automated flow

- Automated Testing and CICD
- Version Control
Automated Testing and CICD

How to Test Jupyter Notebooks with Pytest and Nbmake

Dec 14, 2021 — This tutorial describes how you can use the nbmake, a pytest plugin, in automated end-to-end testing of notebooks. jupyter notebook A Jupyter ...

Unit testing for notebooks | Databricks on AWS

How to call these functions from Python, R, Scala, and SQL notebooks. How to write unit tests in Python, R, and Scala by using the popular test nteract/testbook: ✔️ Unit test your Jupyter Notebooks ...

Previous attempts at unit testing notebooks involved writing the tests in the notebook itself. However, testbook will allow for unit tests to be run against …
Smoke (end-to-end) testing

{20:47}~ ➔ vdk run jupyter-notebook --arguments
On deploy VDK requires passing smoke test first
Opt out possible.
Automated testing with pytest
Using VDK testing library “vdk-test-utils”

```python
from vdk.internal.test_utils import CliEntryBasedTestRunner

list_of_plugins_i_am_using = []
runner = CliEntryBasedTestRunner(list_of_plugins_i_am_using)

result = runner.invoke(['run', 'path/to/your-data-job'])
cli_assert_equal(0, result)
assert 'expected_output' in result.output
```

Challenges

✓ Reproducibility: Non-Linear Execution and Hidden State Risks
✓ Code Organization: Irrelevant or debugging code
✓ Execution model: interactive kernel vs automated flow
✓ Automated Testing and CICD

➢ Version Control
Version Control Challenges

"cell_type": "code",
"execution_count": 2,
"id": "c948f9f2-1f7b-4d8c-aeca-9b380ded9775",
"metadata": {
  "pycharm": {
    "name": "\\n"
  },
  "tags": [
    "vdk"
  ]
},
"outputs": [
  {
    "ename": "NameError",
    "evalue": "name 'job_input' is not defined",
    "output_type": "error",
    "traceback": [
      "\n001b\r\n**** Traceback (most recent call last)****
\nCell \u001b[0;32m[2] \u001b[0m: \u001b[0;32m: \u001b[0m: name 'job_input' is not defined"
    ]
  }
],
"source": [
  "job_input.execute_query("DROP TABLE IF EXISTS rest_target_table;")"
]
Version Control Challenges

Without VDK

```json
358  {
359     "cell_type": "code",
360     "execution_count": null,
361     "id": "cc85260f-2457-4174-9788-f185b24dd821",
362     "metadata": {
363         "tags": []
364     }
365  },
366  "outputs": [],
367  "source": [
368      "run(job_input)"
369  ]
```

With VDK

```json
359  "outputs": [],
360  "source": [
361      "run(job_input)"
362  ]
```
From Data Exploration to Production

✓ Reproducibility: Non-Linear Execution and Hidden State Risks
✓ Code Organization: Irrelevant or debugging code
✓ Execution model: interactive kernel vs automated flow
✓ Automated Testing and CICD
✓ Version Control

Self-paced tutorial


Try it yourself
Thank You

Please take the survey.

https://www.linkedin.com/in/antoni-ivanov