P16 Day 2025

Skrub - Machine learning with dataframes

Riccardo Cappuzzo

Inria P16

2025-10-14

https://skrub-data.org/skrub-materials/

Boost your productivity with skrub!

Skrub simplifies many tedious data preparation operations

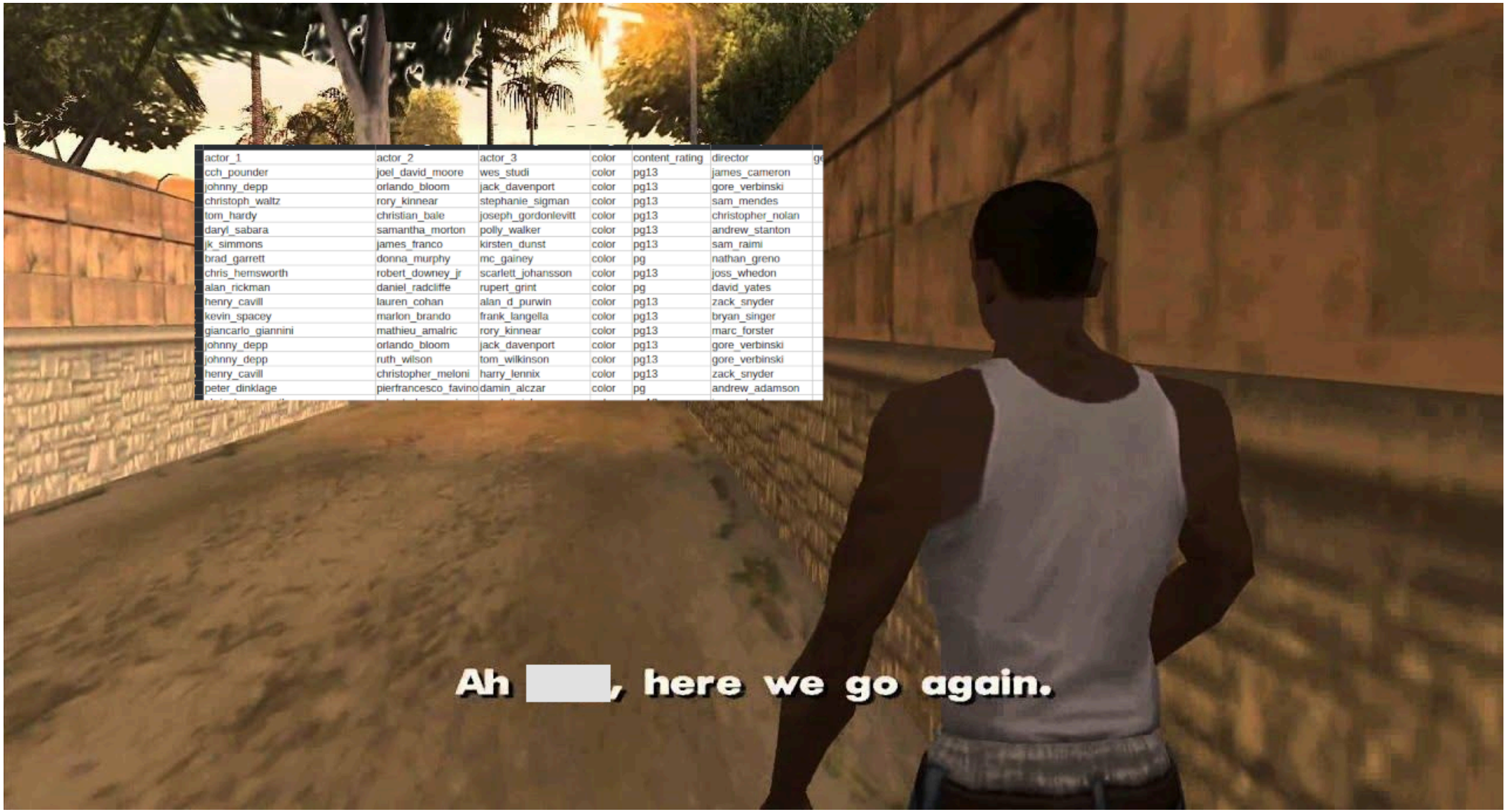
https://skrub-data.org/skrub-materials/

Skrub compatibility

- Skrub is fully compatible with pandas and polars
- Skrub transformers are fully compatible with scikit-learn

An example pipeline

1. Gather some data
2. Explore the data
3. Preprocess the data
4. Perform feature engineering
5. Build a scikit-learn pipeline
6. ???
7. Profit?



actor_1	actor_2	actor_3	color	content_rating	director	g
cch_pounder	joel_david_moore	wes_studi	color	pg13	james_cameron	
johnny_depp	orlando_bloom	jack_davenport	color	pg13	gore_verbinski	
christoph_waltz	rory_kinnear	stephanie_sigman	color	pg13	sam_mendes	
tom_hardy	christian_bale	joseph_gordonlevitt	color	pg13	christopher_nolan	
daryl_sabara	samantha_morton	polly_walker	color	pg13	andrew_stanton	
jk_simmons	james_franco	kirsten_dunst	color	pg13	sam_raimi	
brad_garrett	donna_murphy	mc_gainey	color	pg	nathan_greno	
chris_hemsworth	robert_downey_jr	scarlett_johansson	color	pg13	joss_whedon	
alan_rickman	daniel_radcliffe	rupert_grint	color	pg	david_yates	
henry_cavill	lauren_cohan	alan_d_purwin	color	pg13	zack_snyder	
kevin_spacey	marlon_brando	frank_langella	color	pg13	bryan_singer	
giancarlo_giannini	mathieu_amalric	rory_kinnear	color	pg13	marc_forster	
johnny_depp	orlando_bloom	jack_davenport	color	pg13	gore_verbinski	
johnny_depp	ruth_wilson	tom_wilkinson	color	pg13	gore_verbinski	
henry_cavill	christopher_meloni	harry_lennix	color	pg13	zack_snyder	
peter_dinklage	pierfrancesco_favino	damin_alczar	color	pg	andrew_adamson	

Exploring the data with `skrub`

```
1 from skrub import TableReport
2 TableReport(employee_salaries)
```

TableReport Preview

Main features:

- Obtain high-level statistics about the data
- Explore the distribution of values and find outliers
- Discover highly correlated columns
- Export and share the report as an `html` file

More examples

Data cleaning with pandas/polars: setup

Pandas | Polars

```python
import pandas as pd
import numpy as np

data = {
    "Int": [2, 3, 2],  # Multiple unique values
    "Const str": ["x", "x", "x"],  # Single unique value
    "Str": ["foo", "bar", "baz"],  # Multiple unique values
    "All nan": [np.nan, np.nan, np.nan],  # All missing values
    "All empty": ["", "", ""],  # All empty strings
    "Date": ["01 Jan 2023", "02 Jan 2023", "03 Jan 2023"],
}

df_pd = pd.DataFrame(data)
display(df_pd)
```

	Int	Const str	Str	All nan	All empty	Date
0	2	x	foo	NaN		01 Jan 2023
1	3	x	bar	NaN		02 Jan 2023
2	2	x	baz	NaN		03 Jan 2023

Nulls, datetimes, constant columns with pandas/polars

Pandas | Polars

```python
# Parse the datetime strings with a specific format
df_pd['Date'] = pd.to_datetime(df_pd['Date'], format='%d %b %Y')

# Drop columns with only a single unique value
df_pd_cleaned = df_pd.loc[:, df_pd.nunique(dropna=True) > 1]

# Function to drop columns with only missing values or empty strings
def drop_empty_columns(df):
    # Drop columns with only missing values
    df_cleaned = df.dropna(axis=1, how='all')
    # Drop columns with only empty strings
    empty_string_cols = df_cleaned.columns[df_cleaned.eq('').all()]
    df_cleaned = df_cleaned.drop(columns=empty_string_cols)
    return df_cleaned

# Apply the function to the DataFrame
df_pd_cleaned = drop_empty_columns(df_pd_cleaned)
```

Data cleaning with `skrub.Cleaner`

Pandas | Polars

```
from skrub import Cleaner
cleaner = Cleaner(drop_if_constant=True, datetime_format='%d %b %Y')
df_cleaned = cleaner.fit_transform(df_pd)
display(df_cleaned)
```

	Int	Str	Date
0	2	foo	2023-01-01
1	3	bar	2023-01-02
2	2	baz	2023-01-03

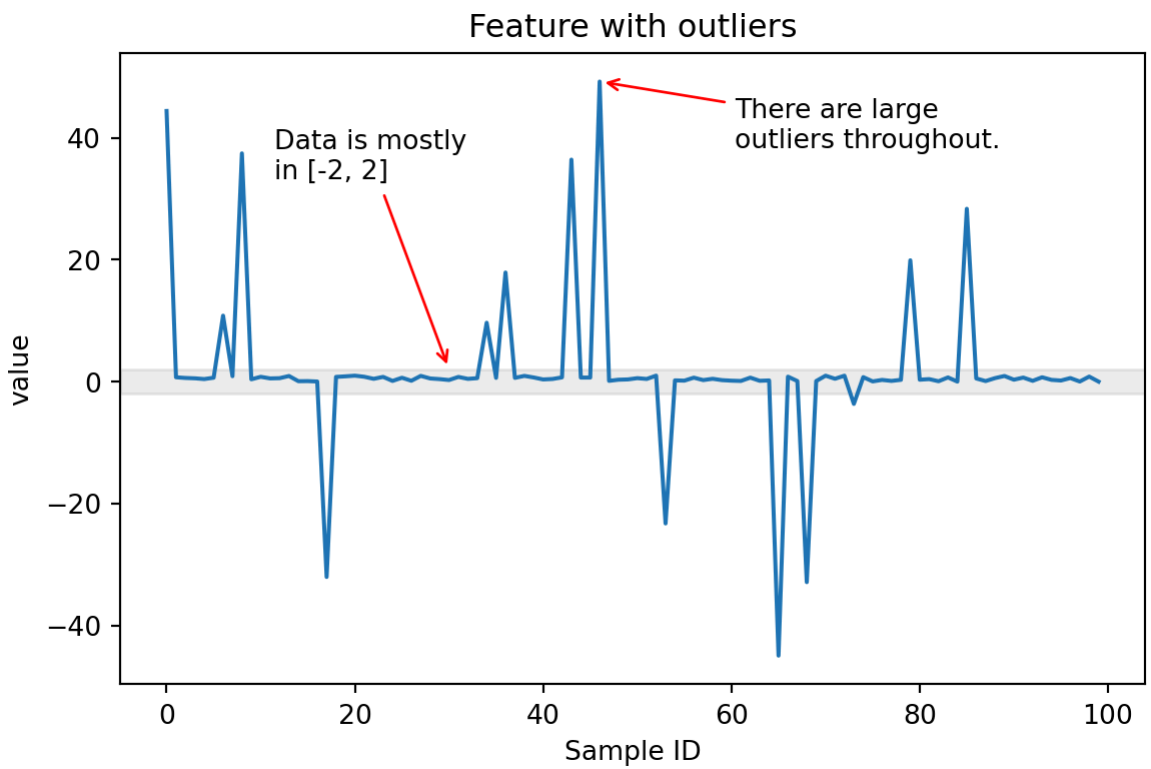
Encoding datetime features with `skrub.DatetimeEncoder`

```python
from skrub import DatetimeEncoder, ToDatetime

X_date = ToDatetime().fit_transform(df["date"])
de = DatetimeEncoder(resolution="second")
# de = DatetimeEncoder(periodic_encoding="spline")
X_enc = de.fit_transform(X_date)
print(X_enc)
```

shape: (3, 7)

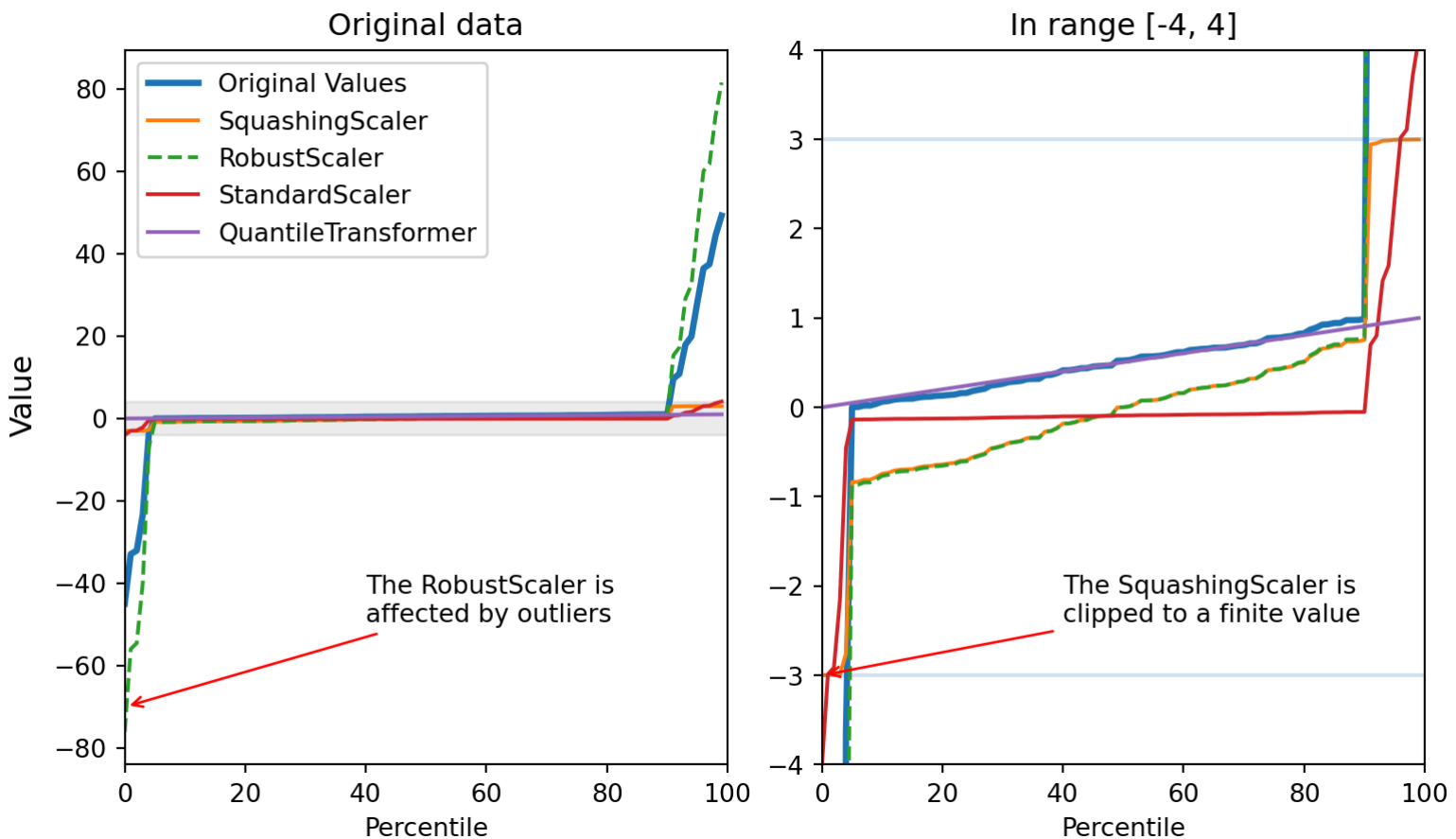
date_year	date_month	date_day	date_hour	date_minute	date_second	date_total_seconds
---	---	---	---	---	---	---
f32	f32	f32	f32	f32	f32	f32
2023.0	1.0	1.0	12.0	34.0	56.0	1.6726e9
2023.0	2.0	15.0	8.0	45.0	23.0	1.6765e9
2023.0	3.0	20.0	18.0	12.0	45.0	1.6793e9

Encoding numerical features with `skrub.SquashingScaler`



https://skrub-data.org/skrub-materials/

Encoding numerical features with `skrub.SquashingScaler`



https://skrub-data.org/skrub-materials/

Encoding categorical (string/text) features

Categorical features have a "**cardinality**": the number of unique values

- Low cardinality: `OneHotEncoder`
- High cardinality (>40 unique values): `skrub.StringEncoder`
- Text: `skrub.TextEncoder` and pretrained models from HuggingFace Hub

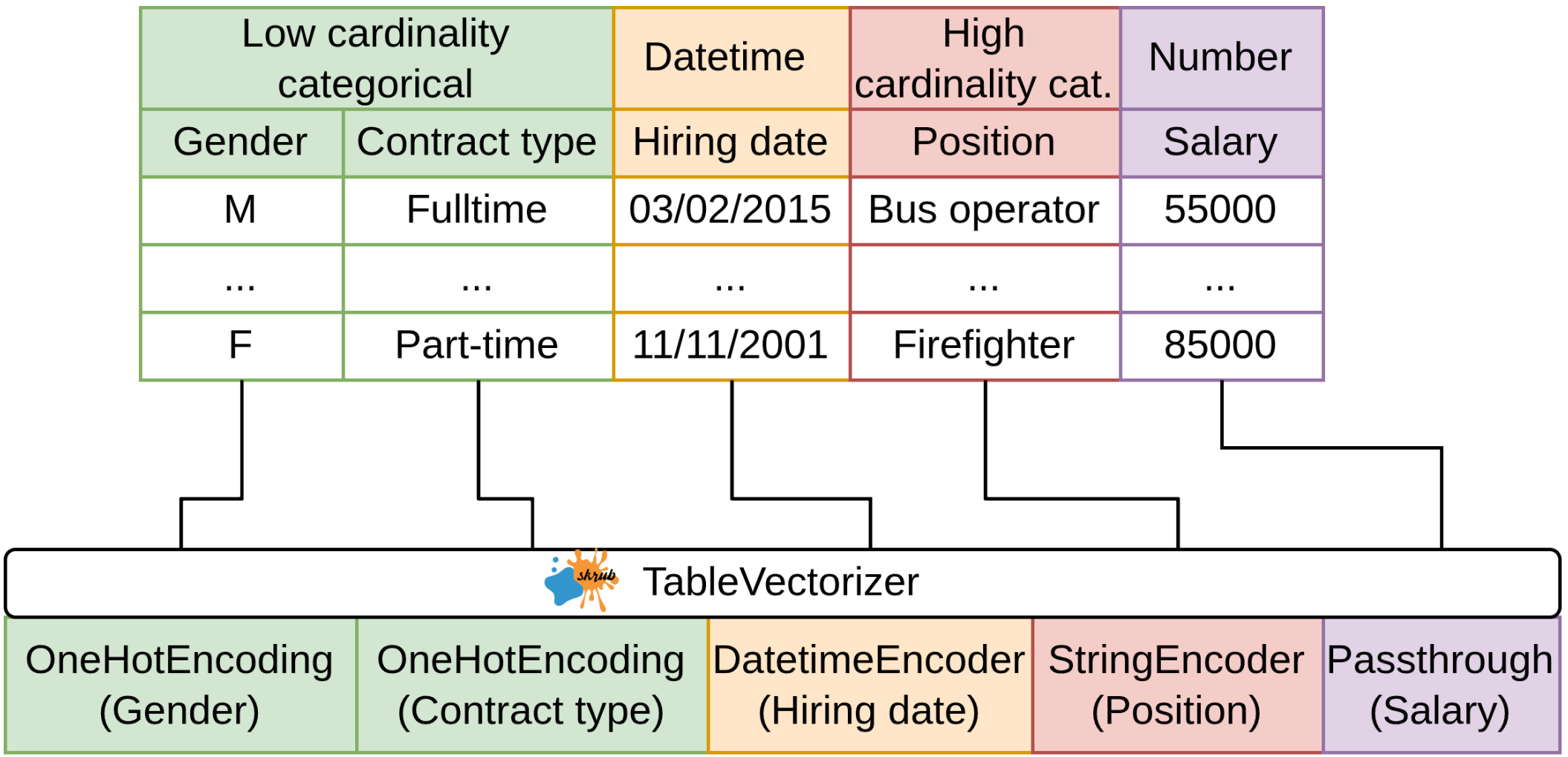
Encoding *all the features*: `TableVectorizer`

```
1 from skrub import TableVectorizer
2
3 table_vec = TableVectorizer()
4 df_encoded = table_vec.fit_transform(df)
```

- Apply the `Cleaner` to all columns
- Split columns by dtype and # of unique values
- Encode each column separately

Encoding *all the features*:

`TableVectorizer`



Build a predictive pipeline

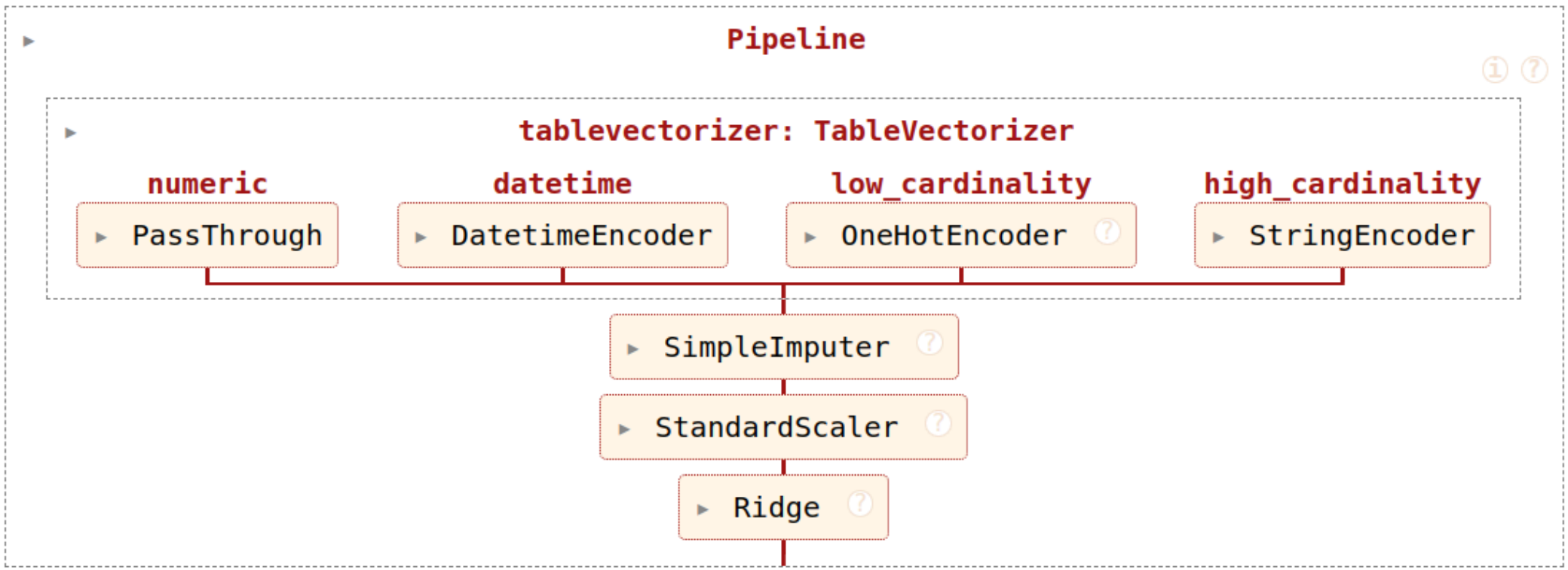
```
1 from sklearn.linear_model import Ridge
2 model = Ridge()
```

Build a predictive pipeline

```
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_selector as selector
from sklearn.compose import make_column_transformer

categorical_columns = selector(dtype_include=object)(employees)
numerical_columns = selector(dtype_exclude=object)(employees)

ct = make_column_transformer(
      (StandardScaler(),
       numerical_columns),
      (OneHotEncoder(handle_unknown="ignore"),
       categorical_columns))

model = make_pipeline(ct, SimpleImputer(), Ridge())
```

Build a predictive pipeline with `tabular_pipeline`

```
1 import skrub
2 from sklearn.linear_model import Ridge
3 model = skrub.tabular_pipeline(Ridge())
```



https://skrub-data.org/skrub-materials/

scikit learn

```python
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_selector as selector
from sklearn.compose import make_column_transformer

categorical_columns = selector(dtype_include=object)(employees)
numerical_columns = selector(dtype_exclude=object)(employees)

ct = make_column_transformer(
      (StandardScaler(),
       numerical_columns),
      (OneHotEncoder(handle_unknown="ignore"),
       categorical_columns))

model = make_pipeline(ct, SimpleImputer(), Ridge())
```

scikit learn + skrub

```python
import skrub
from sklearn.linear_model import Ridge
tl = skrub.tabular_learner(Ridge())
```

https://skrub-data.org/skrub-materials/

We now have a pipeline!

1. Gather some data
2. Explore the data
    - `TableReport`
3. Pre-process the data
    - `Cleaner`, `ToDatetime` ...
4. Perform feature engineering
    - `TableVectorizer`, `SquashingScaler`, `TextEncoder`, `StringEncoder`...
5. Build a scikit-learn pipeline
    - `tabular_pipeline`
6. ???
7. Profit 📈

What if this is not enough??

https://skrub-data.org/skrub-materials/

What if...

- Your data is spread over multiple tables?
- You want to avoid data leakage?
- You want to tune more than just the hyperparameters of your model?
- You want to guarantee that your pipeline is replayed exactly on new data?

When a normal pipeline is not enough...

... the `skrub` DataOps come to the rescue 🚒

DataOps...

- Extend the `scikit-learn` machinery to complex multi-table operations, and take care of data leakage
- Track all operations with a computational graph (a *Data Ops plan*)
- Are transparent and give direct access to the underlying object
- Allow tuning any operation in the Data Ops plan
- Guarantee that all operations are reproducible
- Can be persisted and shared easily

How do DataOps work, though?

DataOps **wrap** around *user operations*, where user operations are:

- any dataframe operation (e.g., merge, group by, aggregate etc.)
- scikit-learn estimators (a Random Forest, RidgeCV etc.)
- custom user code (load data from a path, fetch from an URL etc.)

Important

DataOps *record* user operations, so that they can later be *replayed* in the same order and with the same arguments on unseen data.

Starting with the `DataOps`

```
1 import skrub
2 data = skrub.datasets.fetch_credit_fraud()
3
4 baskets = skrub.var("baskets", data.baskets)
5 products = skrub.var("products", data.products) # add a new variable
6
7 X = baskets[["ID"]].skb.mark_as_X()
8 y = baskets["fraud_flag"].skb.mark_as_y()
```

- `baskets` and `products` represent inputs to the pipeline.
- Skrub tracks `X` and `y` so that training and test splits are never mixed.

Applying a transformer

```
from skrub import selectors as s

vectorizer = skrub.TableVectorizer(
    high_cardinality=skrub.StringEncoder()
)
vectorized_products = products.skb.apply(
    vectorizer, cols=s.all() - "basket_ID"
)
```

Executing dataframe operations

```
aggregated_products = vectorized_products.groupby(
    "basket_ID"
).agg("mean").reset_index()

features = X.merge(
    aggregated_products, left_on="ID", right_on="basket_ID"
)
features = features.drop(columns=["ID", "basket_ID"])
```

Applying a ML model

```
from sklearn.ensemble import ExtraTreesClassifier
predictions = features.skb.apply(
    ExtraTreesClassifier(n_jobs=-1), y=y
)
```

Inspecting the Data Ops plan

```
1 predictions.skb.full_report()
```

Execution report

Each node:

- Shows a preview of the data resulting from the operation
- Reports the location in the code where the code is defined
- Shows the run time of the node

Exporting the plan in a `learner`

The **Learner** is a stand-alone object that works like a scikit-learn estimator that takes a dictionary as input rather than just `X` and `y`.

```
learner = predictions.skb.make_learner(fitted=True)
```

Then, the `learner` can be pickled ...

```
import pickle

with open("learner.bin", "wb") as fp:
    pickle.dump(learner, fp)
```

... loaded and applied to new data:

```
with open("learner.bin", "rb") as fp:
    loaded_learner = pickle.load(fp)
data = skrub.datasets.fetch_credit_fraud(split="test")
new_baskets = data.baskets
new_products = data.products
loaded_learner.predict({"baskets": new_baskets, "products": new_products})
```

```
array([0, 0, 0, ..., 0, 0, 0], shape=(31549,))
```

Hyperparameter tuning in a Data Ops plan

Skrub implements four `choose_*` functions:

- `choose_from`: select from the given list of options
- `choose_int`: select an integer within a range
- `choose_float`: select a float within a range
- `choose_bool`: select a bool
- `optional`: chooses whether to execute the given operation

Tuning in `scikit-learn` can be complex

```
pipe = Pipeline([("dim_reduction", PCA()), ("regressor", Ridge())])
grid = [
    {
        "dim_reduction": [PCA()],
        "dim_reduction__n_components": [10, 20, 30],
        "regressor": [Ridge()],
        "regressor__alpha": loguniform(0.1, 10.0),
    },
    {
        "dim_reduction": [SelectKBest()],
        "dim_reduction__k": [10, 20, 30],
        "regressor": [Ridge()],
        "regressor__alpha": loguniform(0.1, 10.0),
    },
    {
        "dim_reduction": [PCA()],
        "dim_reduction__n_components": [10, 20, 30],
        "regressor": [RandomForestRegressor()],
        "regressor__n_estimators": loguniform(20, 200),
    },
    {
        "dim_reduction": [SelectKBest()],
        "dim_reduction__k": [10, 20, 30],
        "regressor": [RandomForestRegressor()],
        "regressor__n_estimators": loguniform(20, 200),
```

Tuning with Data Ops is simple!

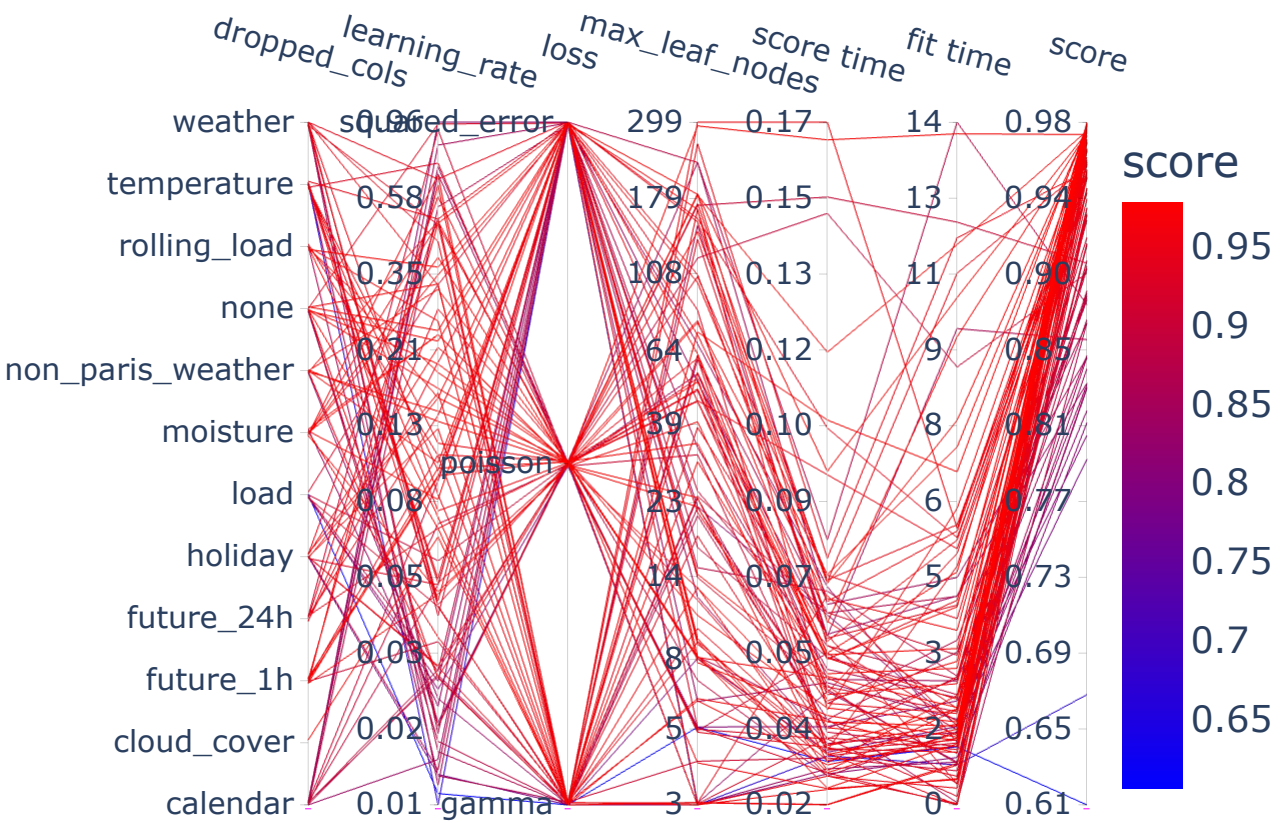
```
dim_reduction = X.skb.apply(
    skrub.choose_from(
        {
            "PCA": PCA(n_components=skrub.choose_int(10, 30)),
            "SelectKBest": SelectKBest(k=skrub.choose_int(10, 30))
        }, name="dim_reduction"
    )
)
regressor = dim_reduction.skb.apply(
    skrub.choose_from(
        {
            "Ridge": Ridge(alpha=skrub.choose_float(0.1, 10.0, log=True)),
            "RandomForest": RandomForestRegressor(
                n_estimators=skrub.choose_int(20, 200, log=True)
            )
        }, name="regressor"
    )
)
```

Run hyperparameter search

```
# fit the search
search = regressor.skb.make_randomized_search(
    scoring="roc_auc", fitted=True, cv=5
)

# save the best learner
best_learner = search.best_learner_
```

A parallel coordinate plot to explore hyperparameters

```
search = pred.skb.get_randomized_search(fitted=True)
search.plot_parallel_coord()
```



https://skrub-data.org/skrub-materials/

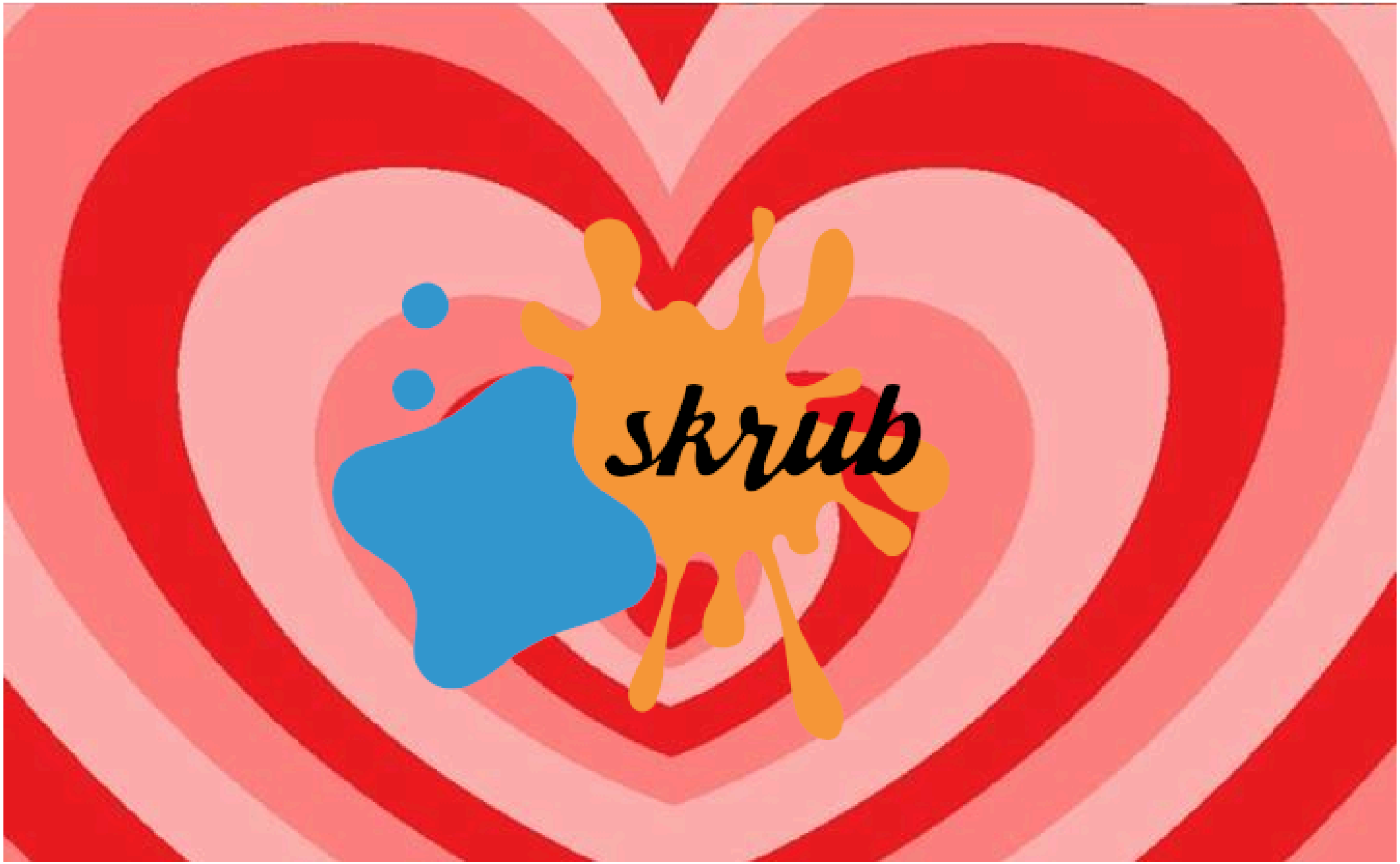
More information about the Data Ops

- Skrub example gallery
- Skrub user guide
- Tutorial on timeseries forecasting at Euroscipy 2025
- Kaggle notebook on the Titanic survival challenge

Wrapping up

https://skrub-data.org/skrub-materials/

https://skrub-data.org/skrub-materials/



https://skrub-data.org/skrub-materials/

Getting involved

Do you want to learn more?

- Skrub website
- Skrub materials website
- Discord server

Follow skrub on:

- Bluesky
- LinkedIn

Star skrub on GitHub, or contribute directly:

- GitHub repository

tl;dw: skrub

- interactive data exploration: `TableReport`
- automated pre-processing of pandas and polars dataframes: `Cleaner`
- powerful feature engineering: `TableVectorizer`, `tabular_pipeline`
- column- and dataframe-level operations: `ApplyToCols`, selectors
- DataOps, plans, hyperparameter tuning, (almost) no leakage