30 Python Libraries to (Hugely) Boost Your Data Science Productivity

Avi Chawla

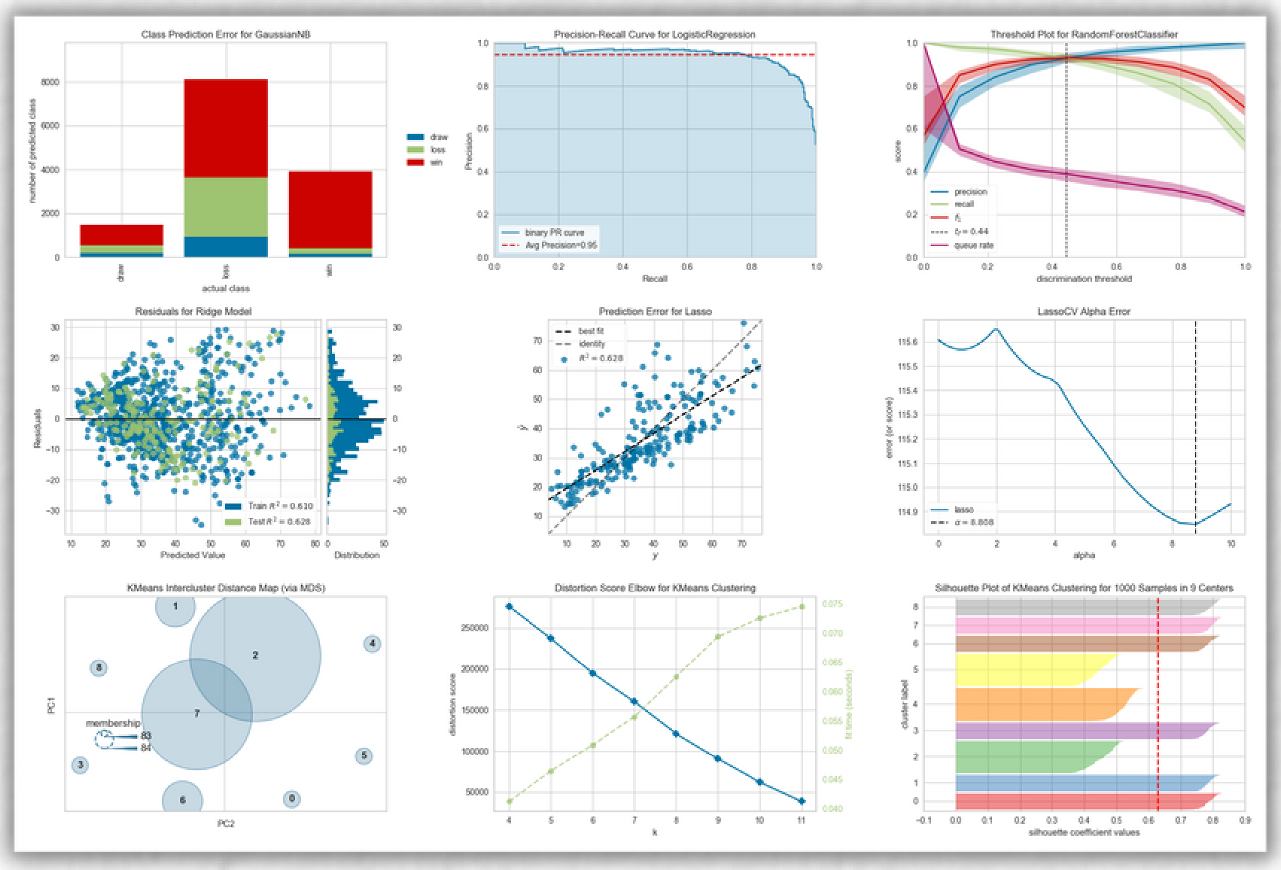
Daily Dose of Data Science

avichawla.substack.com

Data Science is much more than Pandas, NumPy and Sklearn.

Here are **30 open-source libraries** to upgrade your data game.

1. YellowBrick

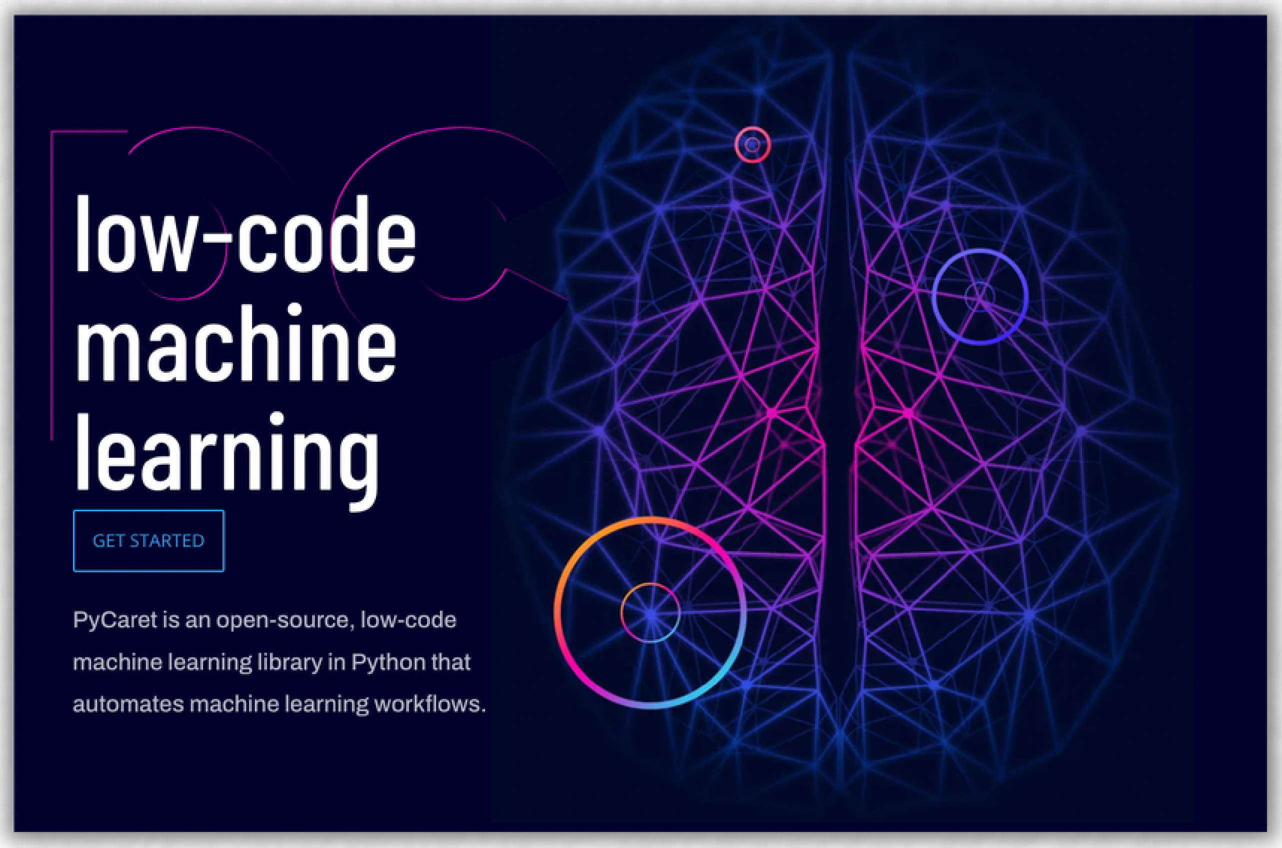


A suite of visualization and diagnostic tools for **faster model selection**.

Matplotlib Sklearn

avichawla.substack.com

2. PyCaret



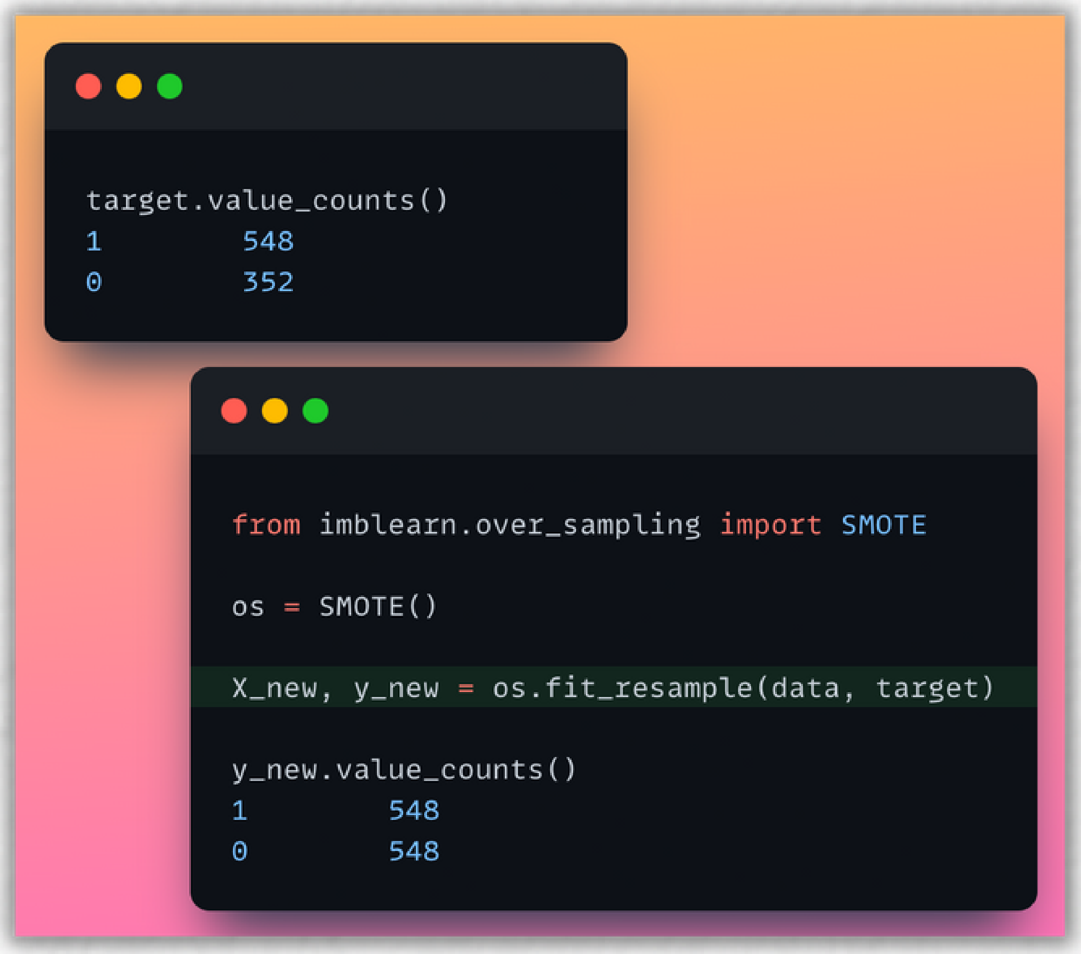
low-code machine learning

GET STARTED

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows.

Automate ML workflows with this **low-code library**.

3. imbalanced-learn

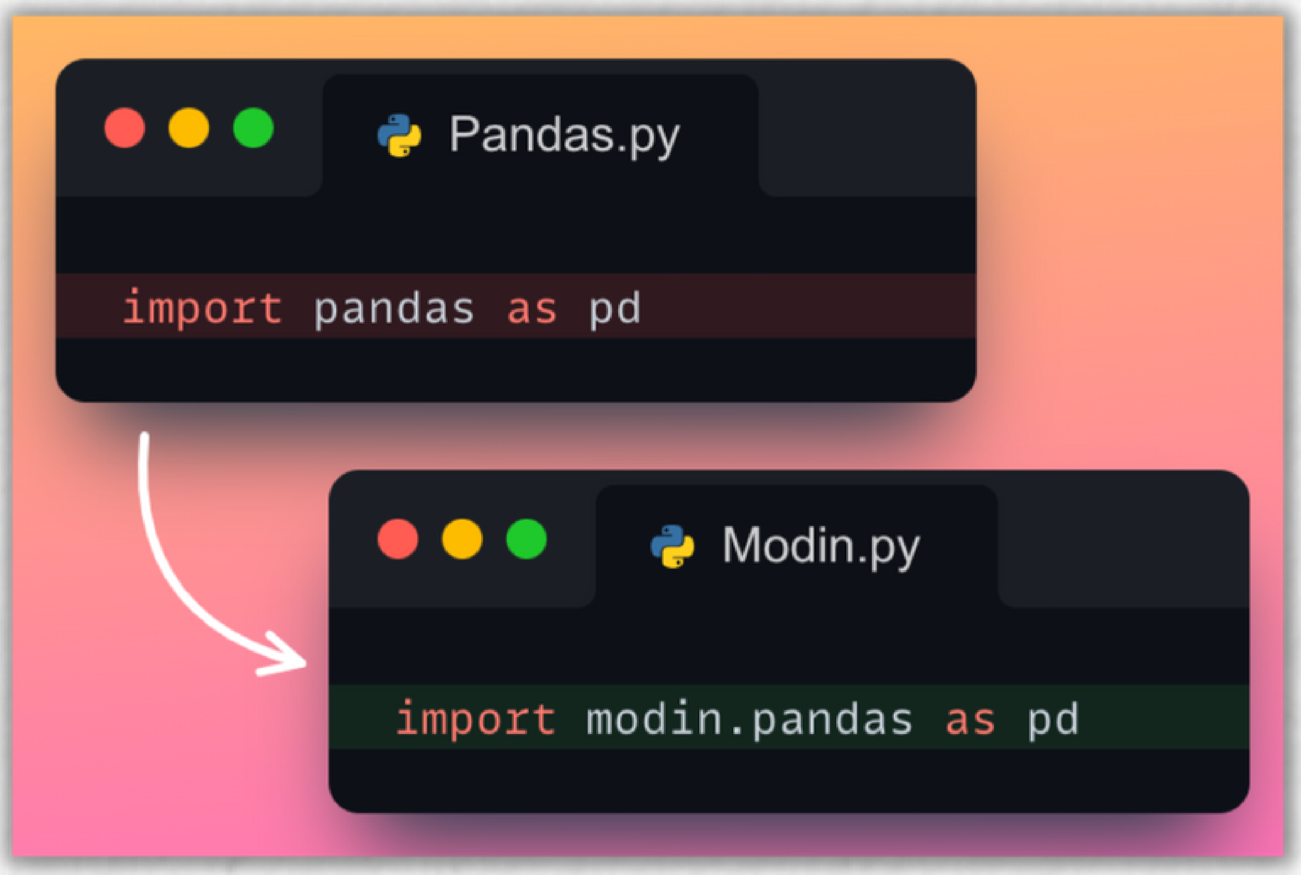


```
target.value_counts()
1          548
0          352
```

```
from imblearn.over_sampling import SMOTE

os = SMOTE()

X_new, y_new = os.fit_resample(data, target)

y_new.value_counts()
1          548
0          548
```

A variety of methods to handle class imbalance.

4. Modin

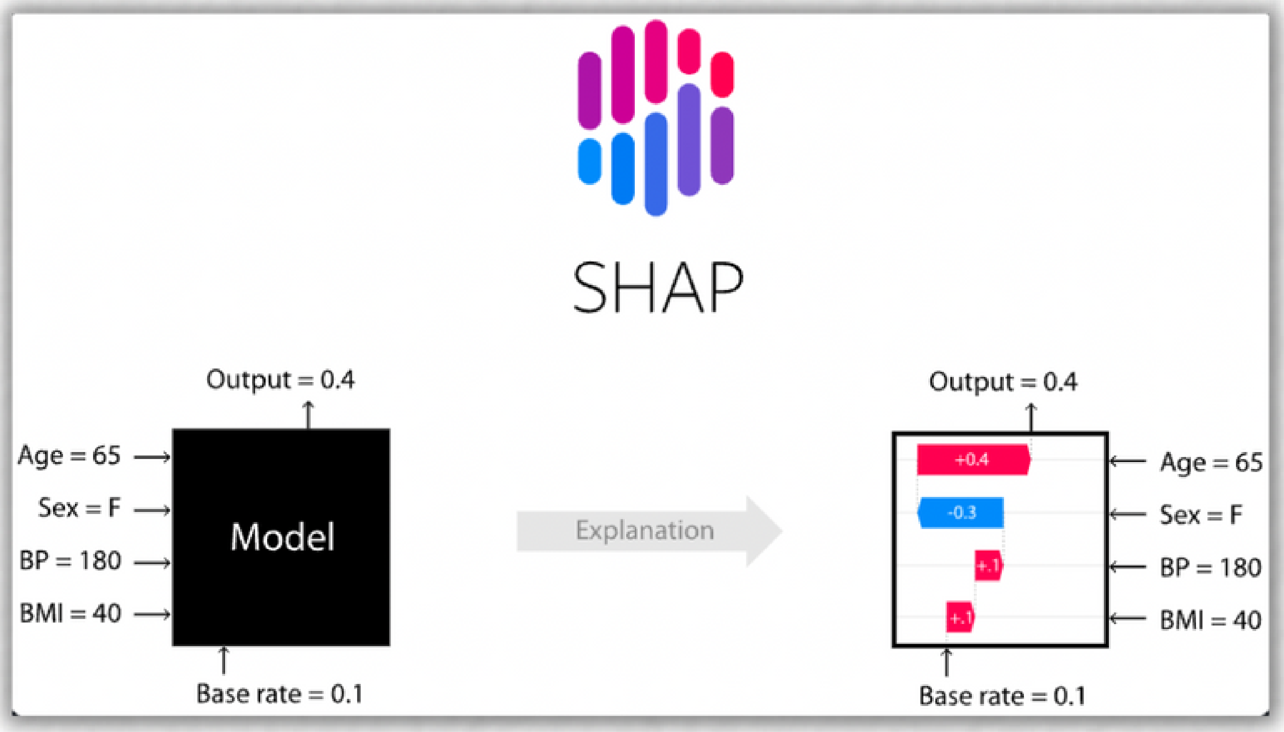


```
import pandas as pd
```

```
import modin.pandas as pd
```

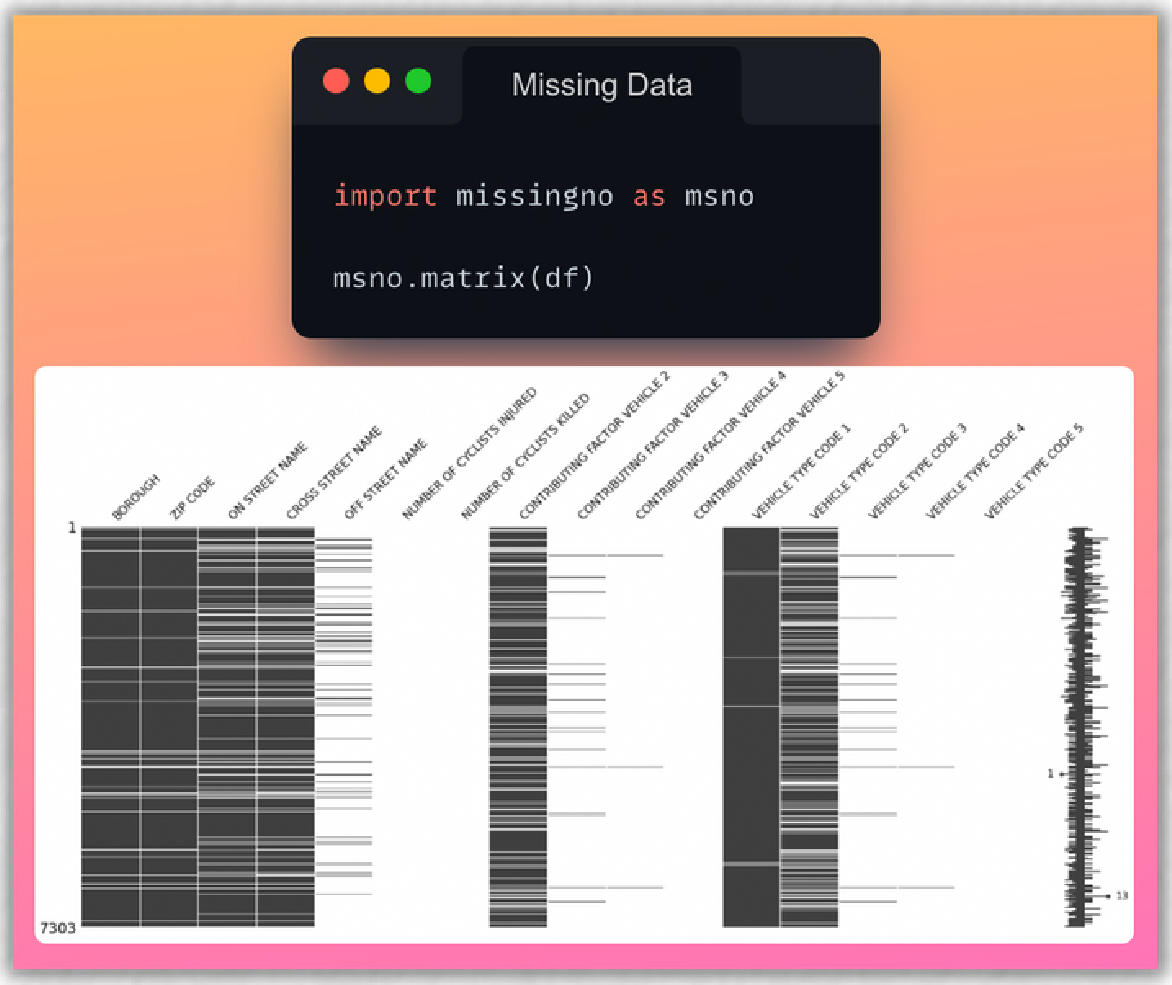
Boost Pandas' performance **up to 70x** by modifying the import.

5. SHAP



Explain the output of any ML model in few lines of code.

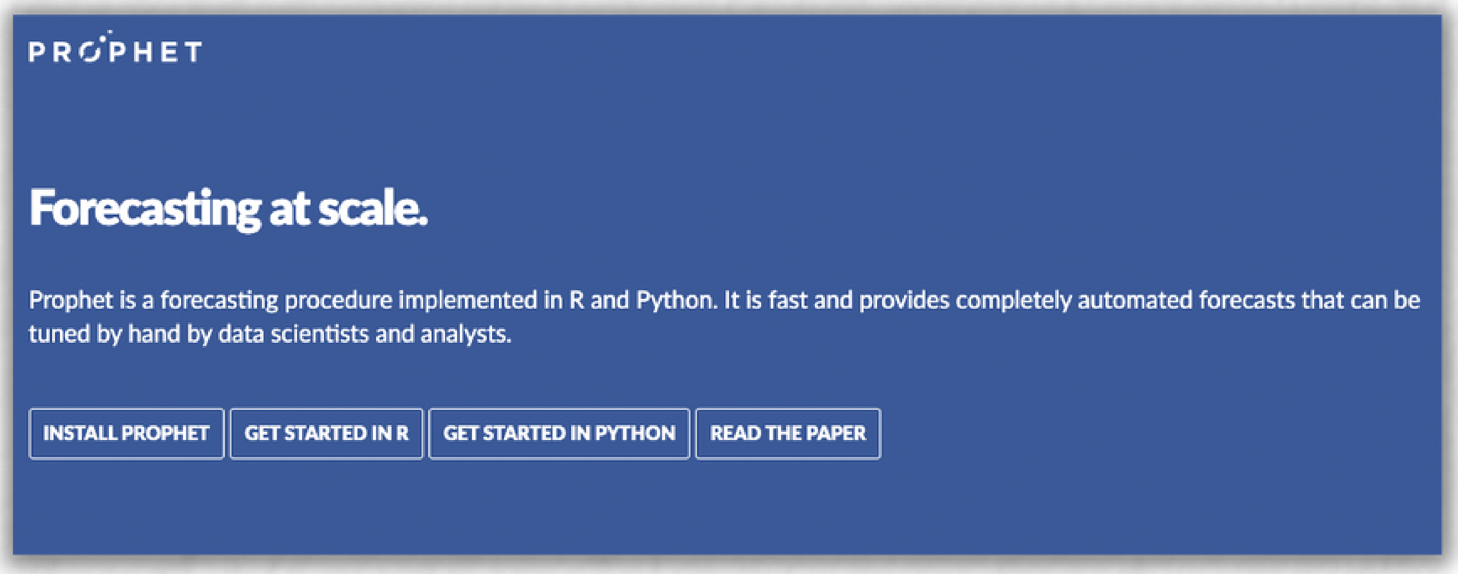
6. Missingno



```
import missingno as msno

msno.matrix(df)
```

Visualize missing values in your dataset with ease.

7. Prophet



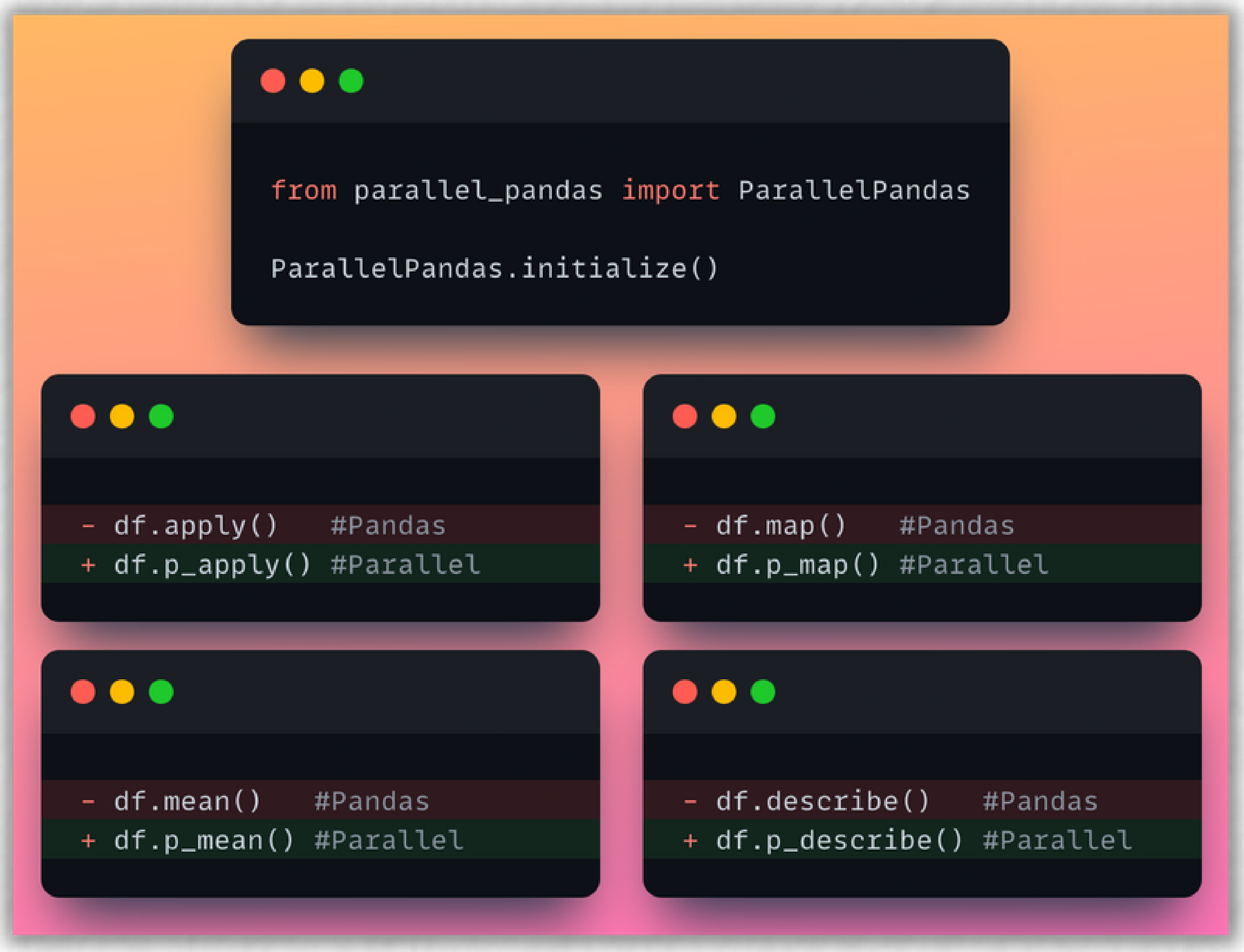
PROPHET

Forecasting at scale.

Prophet is a forecasting procedure implemented in R and Python. It is fast and provides completely automated forecasts that can be tuned by hand by data scientists and analysts.

Produce **high-quality forecasts** on time-series data.

8. Parallel-Pandas



```python
from parallel_pandas import ParallelPandas

ParallelPandas.initialize()
```

```diff
- df.apply()    #Pandas
+ df.p_apply() #Parallel
```

```diff
- df.map()    #Pandas
+ df.p_map() #Parallel
```

```diff
- df.mean()    #Pandas
+ df.p_mean() #Parallel
```

```diff
- df.describe()    #Pandas
+ df.p_describe() #Parallel
```

Parallelize Pandas across all CPU cores for **faster computation**.

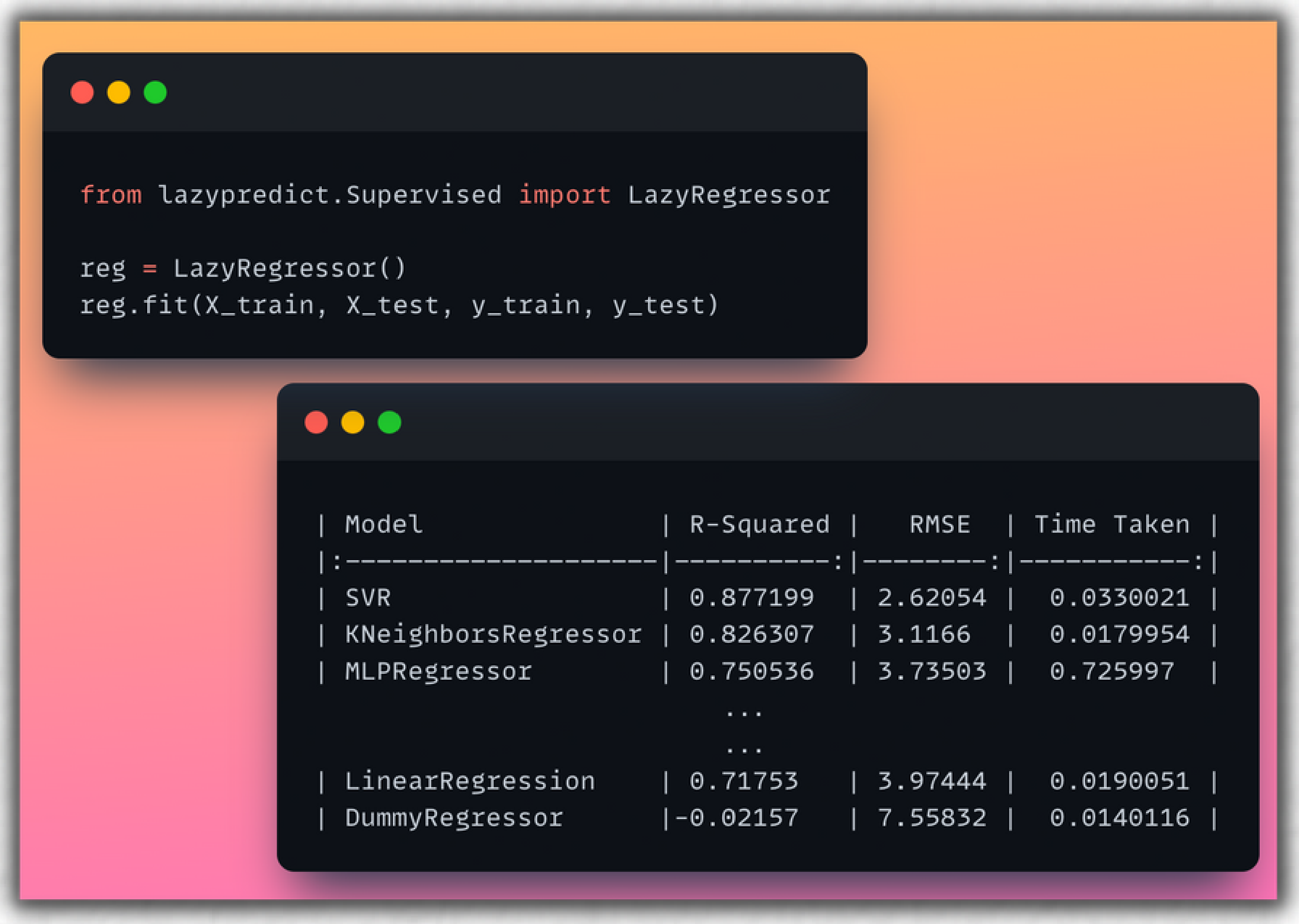
9. Featuretools

Featuretools

An open source python framework for automated feature engineering

Automated feature engineering for ML models.

10. Lazy Predict

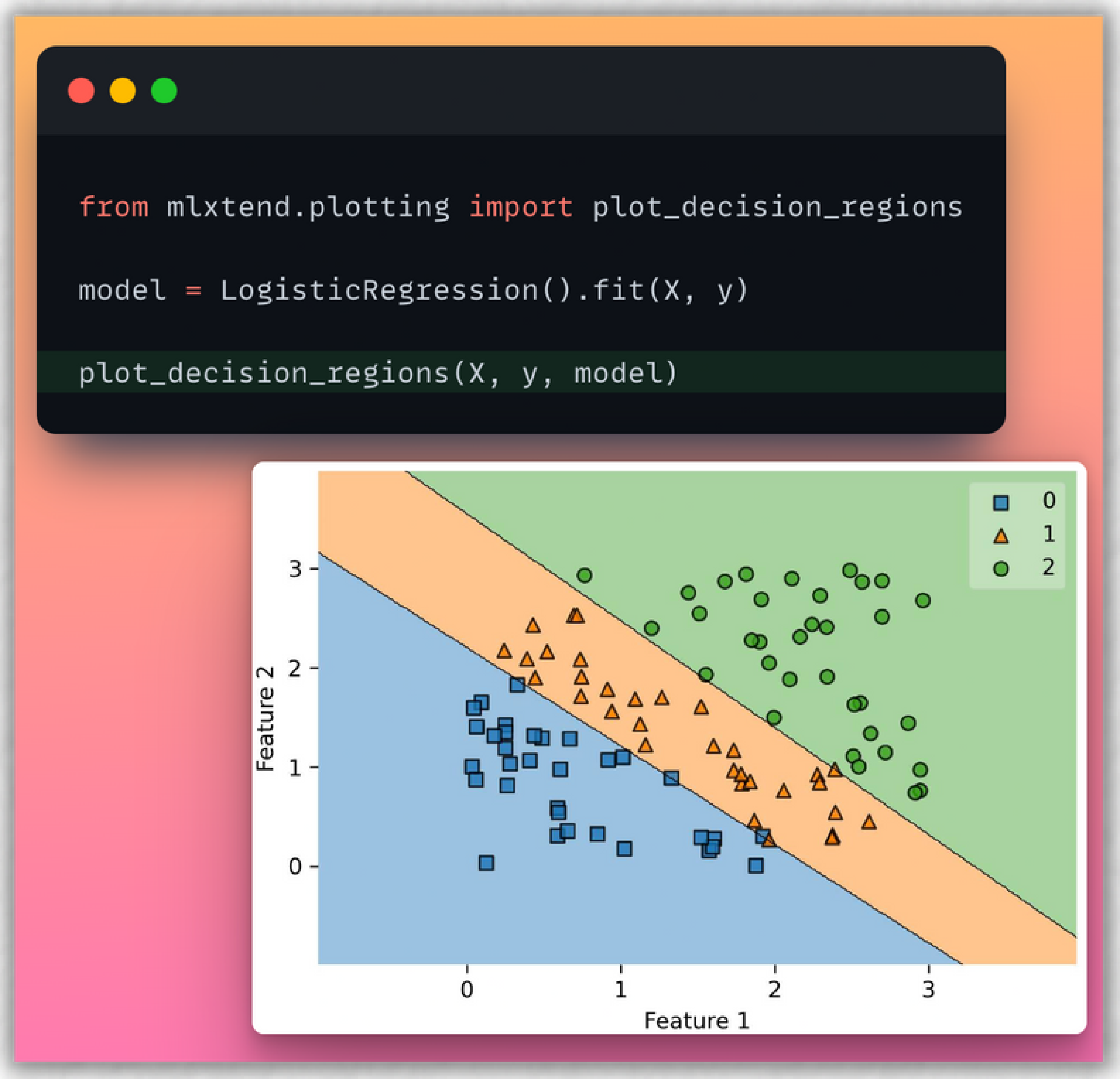


```
from lazypredict.Supervised import LazyRegressor

reg = LazyRegressor()
reg.fit(X_train, X_test, y_train, y_test)
```

```
| Model                | R-Squared |   RMSE   | Time Taken |
|:---------------------|----------:|---------:|-----------:|
| SVR                  | 0.877199  | 2.62054 |   0.0330021 |
| KNeighborsRegressor  | 0.826307  | 3.1166  |   0.0179954 |
| MLPRegressor         | 0.750536  | 3.73503 |   0.725997  |
                                 ...
                                 ...
| LinearRegression     | 0.71753   | 3.97444 |   0.0190051 |
| DummyRegressor       |-0.02157   | 7.55832 |   0.0140116 |
```

Train **30 machine learning models** in one line of code.

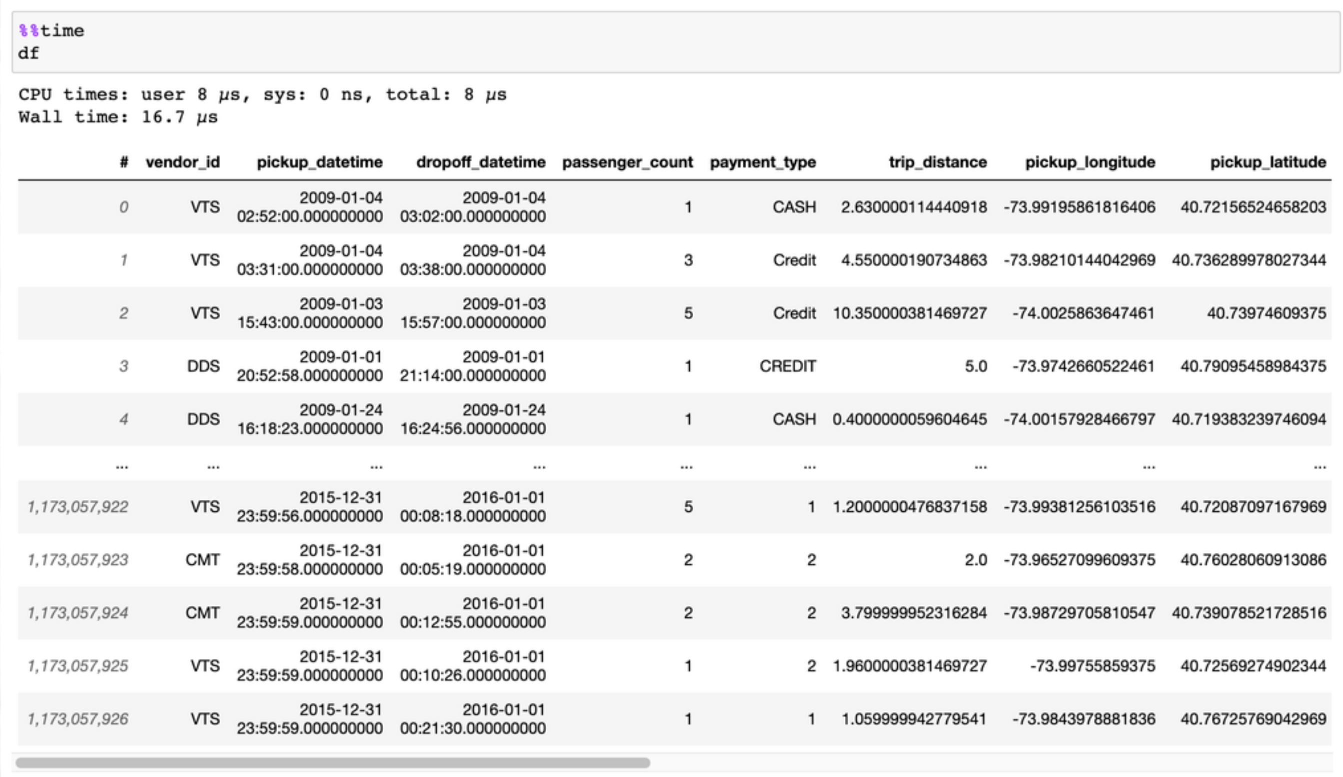
11. mlxtend



```
from mlxtend.plotting import plot_decision_regions

model = LogisticRegression().fit(X, y)

plot_decision_regions(X, y, model)
```

A collection of utility functions for **processing, evaluating, visualizing** models.

12. Vaex



```
%%time
df
```

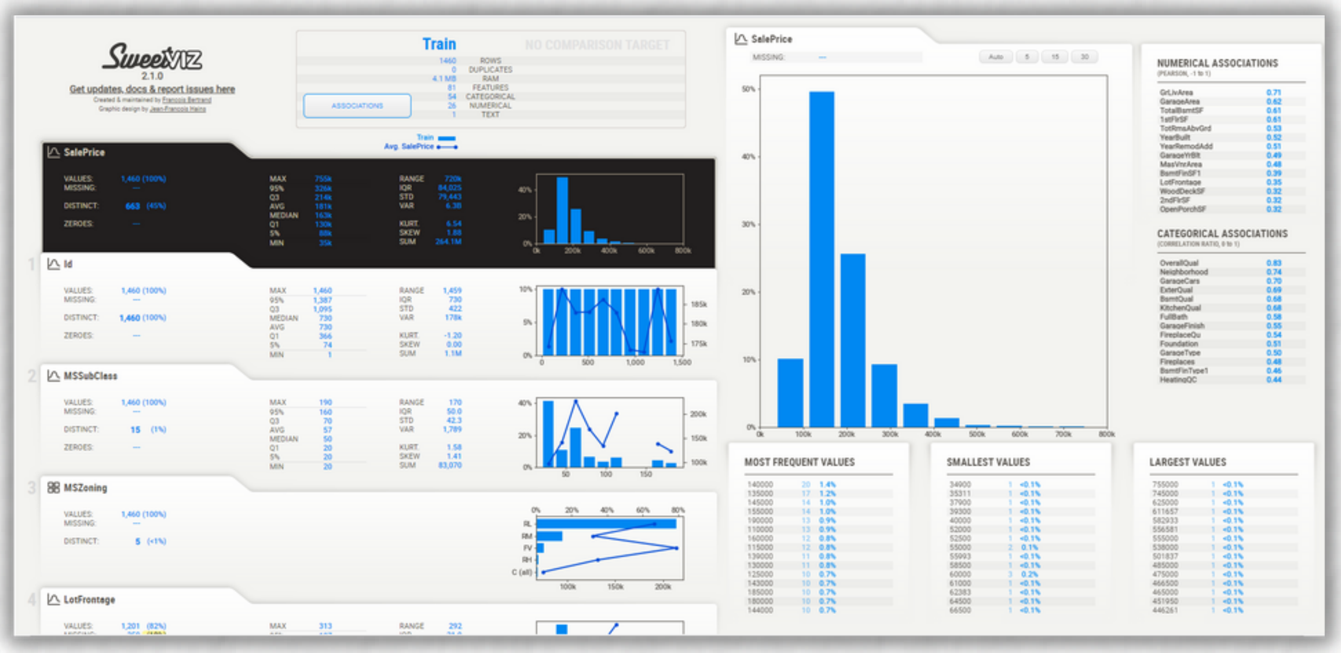
```
CPU times: user 8 µs, sys: 0 ns, total: 8 µs
Wall time: 16.7 µs
```

#	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	payment_type	trip_distance	pickup_longitude	pickup_latitude
0	VTS	2009-01-04 02:52:00.000000000	2009-01-04 03:02:00.000000000	1	CASH	2.630000114440918	-73.99195861816406	40.72156524658203
1	VTS	2009-01-04 03:31:00.000000000	2009-01-04 03:38:00.000000000	3	Credit	4.550000190734863	-73.98210144042969	40.736289978027344
2	VTS	2009-01-03 15:43:00.000000000	2009-01-03 15:57:00.000000000	5	Credit	10.350000381469727	-74.0025863647461	40.73974609375
3	DDS	2009-01-01 20:52:58.000000000	2009-01-01 21:14:00.000000000	1	CREDIT	5.0	-73.9742660522461	40.79095458984375
4	DDS	2009-01-24 16:18:23.000000000	2009-01-24 16:24:56.000000000	1	CASH	0.4000000059604645	-74.00157928466797	40.719383239746094
...	...	...	...	...	...	...	...	...
1,173,057,922	VTS	2015-12-31 23:59:56.000000000	2016-01-01 00:08:18.000000000	5	1	1.2000000476837158	-73.99381256103516	40.72087097167969
1,173,057,923	CMT	2015-12-31 23:59:58.000000000	2016-01-01 00:05:19.000000000	2	2	2.0	-73.96527099609375	40.76028060913086
1,173,057,924	CMT	2015-12-31 23:59:59.000000000	2016-01-01 00:12:55.000000000	2	2	3.799999952316284	-73.98729705810547	40.739078521728516
1,173,057,925	VTS	2015-12-31 23:59:59.000000000	2016-01-01 00:10:26.000000000	1	2	1.9600000381469727	-73.99755859375	40.72569274902344
1,173,057,926	VTS	2015-12-31 23:59:59.000000000	2016-01-01 00:21:30.000000000	1	1	1.059999942779541	-73.9843978881836	40.76725769042969

High performance package for **lazy Out-of-Core DataFrames.**

avichawla.substack.com

13. SweetViz



SweetViz 2.1.0

Get updates, docs & report issues here

Created & maintained by Francois Bertrand

Graphic design by Jean-Francois Hains

Train NO COMPARISON TARGET

1460	ROWS
0	DUPLICATES
4.1 MB	RAM
81	FEATURES
54	CATEGORICAL
26	NUMERICAL
1	TEXT

ASSOCIATIONS

```
SalePrice
VALUES:    1,460 (100%)    MAX     755k     RANGE  720k
MISSING:   —               95%     326k     IQR    84,025
DISTINCT:  663 (45%)       Q3      214k     STD    79,443
ZEROES:    —               AVG     181k     VAR    6.30
                           MEDIAN  163k     KURT.  6.54
                           Q1      130k     SKEW   1.88
                           5%      88k      SUM    264.1M
                           MIN     35k

1 Id
VALUES:    1,460 (100%)    MAX     1,460    RANGE  1,459
MISSING:   —               95%     1,387    IQR    730
DISTINCT:  1,460 (100%)    Q3      1,095    STD    422
ZEROES:    —               MEDIAN  730      VAR    178k
                           AVG     730      KURT.  -1.20
                           Q1      366      SKEW   0.00
                           5%      74       SUM    1.1M
                           MIN     1

2 MSSubClass
VALUES:    1,460 (100%)    MAX     190      RANGE  170
MISSING:   —               95%     160      IQR    50.0
DISTINCT:  15 (1%)         Q3      70       STD    42.3
ZEROES:    —               AVG     57       VAR    1,789
                           MEDIAN  50       KURT.  1.58
                           Q1      20       SKEW   1.41
                           5%      20       SUM    83,070
                           MIN     20

3 MSZoning
VALUES:    1,460 (100%)
MISSING:   —
DISTINCT:  5 (<1%)

4 LotFrontage
VALUES:    1,201 (82%)     MAX     313      RANGE  292
```

SalePrice

MISSING: —

NUMERICAL ASSOCIATIONS (PEARSON, -1 to 1)

GrLivArea	0.71
GarageArea	0.62
TotalBsmtSF	0.61
1stFlrSF	0.61
TotRmsAbvGrd	0.53
YearBuilt	0.52
YearRemodAdd	0.51
GarageYrBlt	0.49
MasVnrArea	0.48
BsmtFinSF1	0.39
LotFrontage	0.35
WoodDeckSF	0.32
2ndFlrSF	0.32
OpenPorchSF	0.32

CATEGORICAL ASSOCIATIONS (CORRELATION RATIO, 0 to 1)

OverallQual	0.83
Neighborhood	0.74
GarageCars	0.70
ExterQual	0.69
BsmtQual	0.68
KitchenQual	0.68
FullBath	0.58
GarageFinish	0.55
FireplaceQu	0.54
Foundation	0.51
GarageType	0.50
Fireplaces	0.48
BsmtFinType1	0.46
HeatingQC	0.44

MOST FREQUENT VALUES

140000	20	1.4%
135000	17	1.2%
145000	14	1.0%
155000	14	1.0%
190000	13	0.9%
110000	13	0.9%
160000	12	0.8%
115000	12	0.8%
139000	11	0.8%
130000	11	0.8%
125000	10	0.7%
143000	10	0.7%
185000	10	0.7%
180000	10	0.7%
144000	10	0.7%

SMALLEST VALUES

34900	1	<0.1%
35311	1	<0.1%
37900	1	<0.1%
39300	1	<0.1%
40000	1	<0.1%
52000	1	<0.1%
52500	1	<0.1%
55000	2	0.1%
55993	1	<0.1%
58500	1	<0.1%
60000	3	0.2%
61000	1	<0.1%
62383	1	<0.1%
64500	1	<0.1%
66500	1	<0.1%

LARGEST VALUES

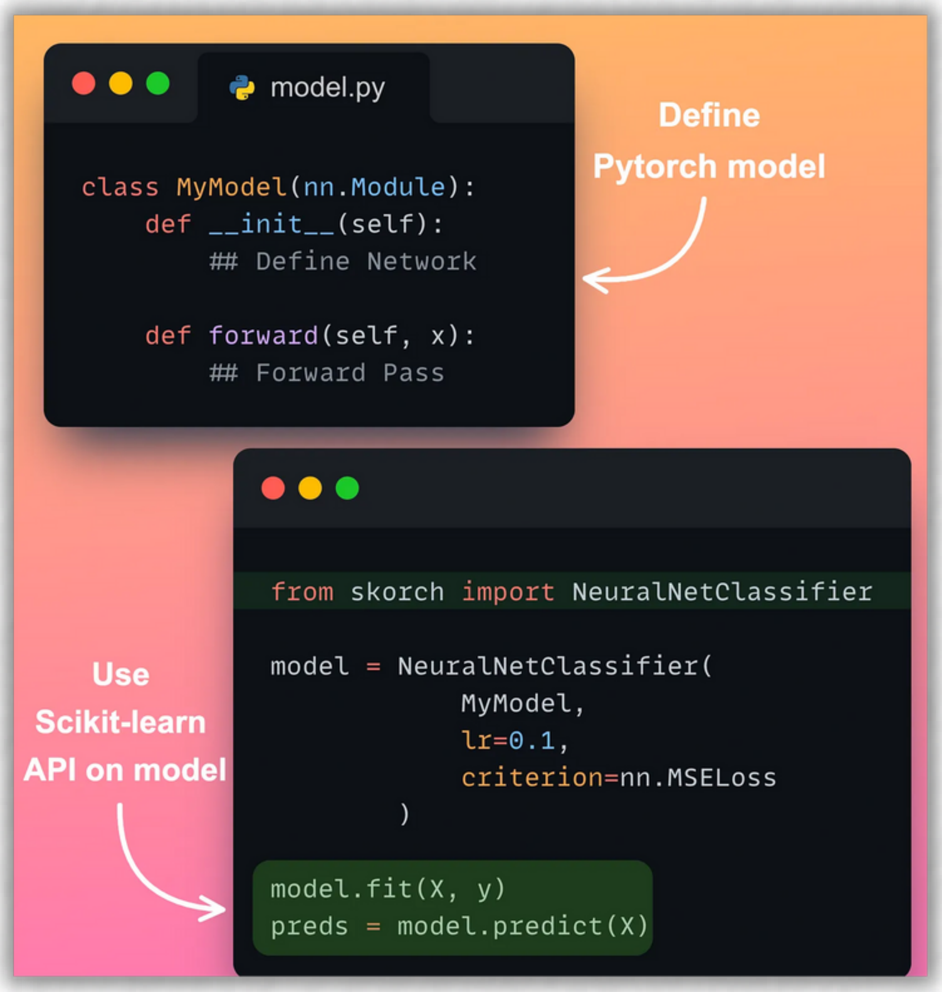
755000	1	<0.1%
745000	1	<0.1%
625000	1	<0.1%
611657	1	<0.1%
582933	1	<0.1%
556581	1	<0.1%
555000	1	<0.1%
538000	1	<0.1%
501837	1	<0.1%
485000	1	<0.1%
475000	1	<0.1%
466500	1	<0.1%
465000	1	<0.1%
451950	1	<0.1%
446261	1	<0.1%

In-depth EDA report

in two lines

of code.

14. Skorch



model.py

```
class MyModel(nn.Module):
    def __init__(self):
        ## Define Network

    def forward(self, x):
        ## Forward Pass
```

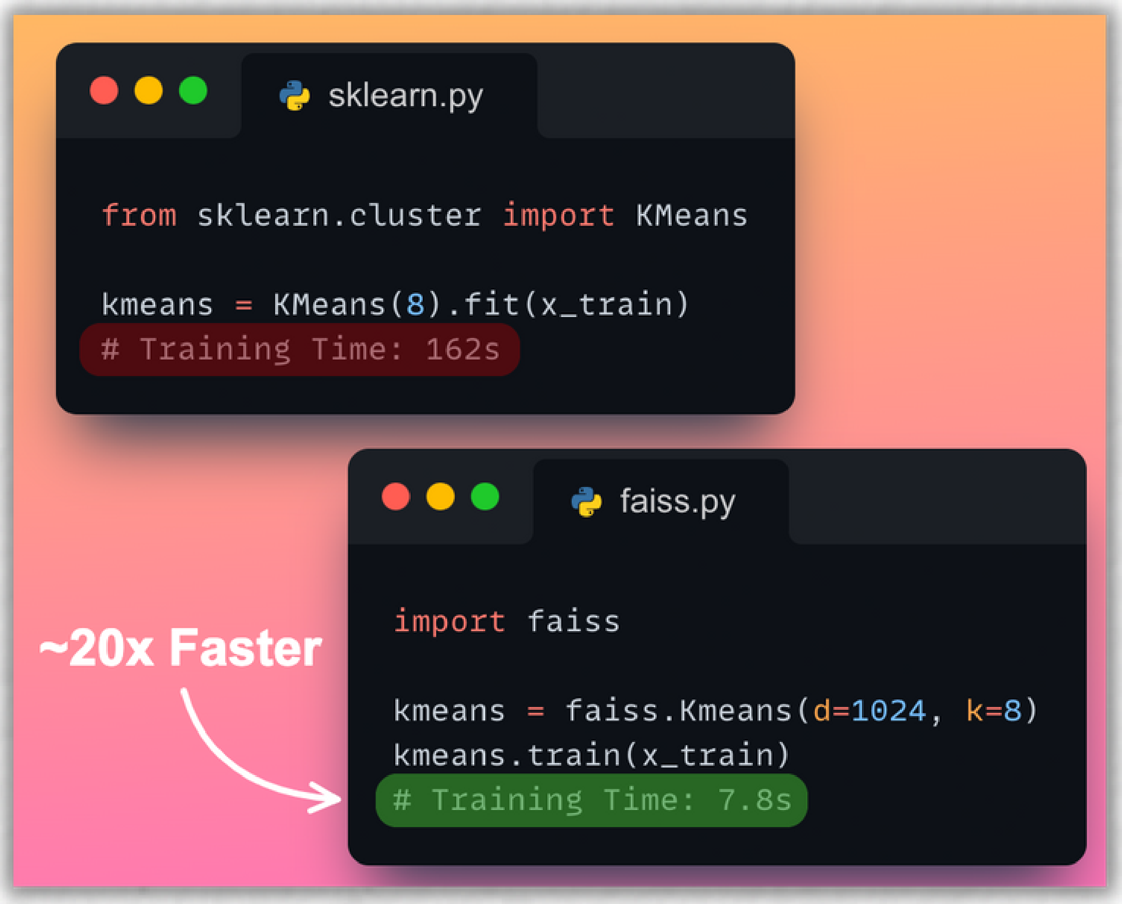
Define
Pytorch model

```
from skorch import NeuralNetClassifier

model = NeuralNetClassifier(
            MyModel,
            lr=0.1,
            criterion=nn.MSELoss
        )

model.fit(X, y)
preds = model.predict(X)
```

Use
Scikit-learn
API on model

Leverage the power of PyTorch with the elegance of sklearn.

15. Faiss



```python
from sklearn.cluster import KMeans

kmeans = KMeans(8).fit(x_train)
# Training Time: 162s
```

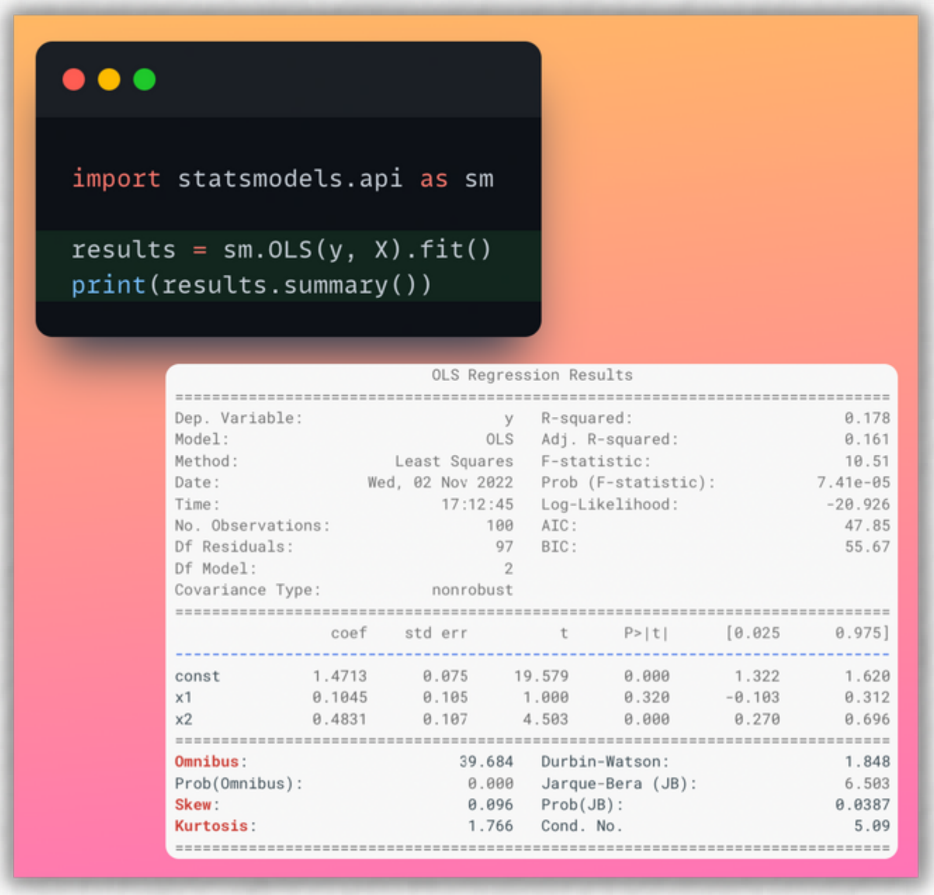
```python
import faiss

kmeans = faiss.Kmeans(d=1024, k=8)
kmeans.train(x_train)
# Training Time: 7.8s
```

~20x Faster

Efficient algorithms for **similarity search** and **clustering** dense vectors.

avichawla.substack.com

16. statsmodel

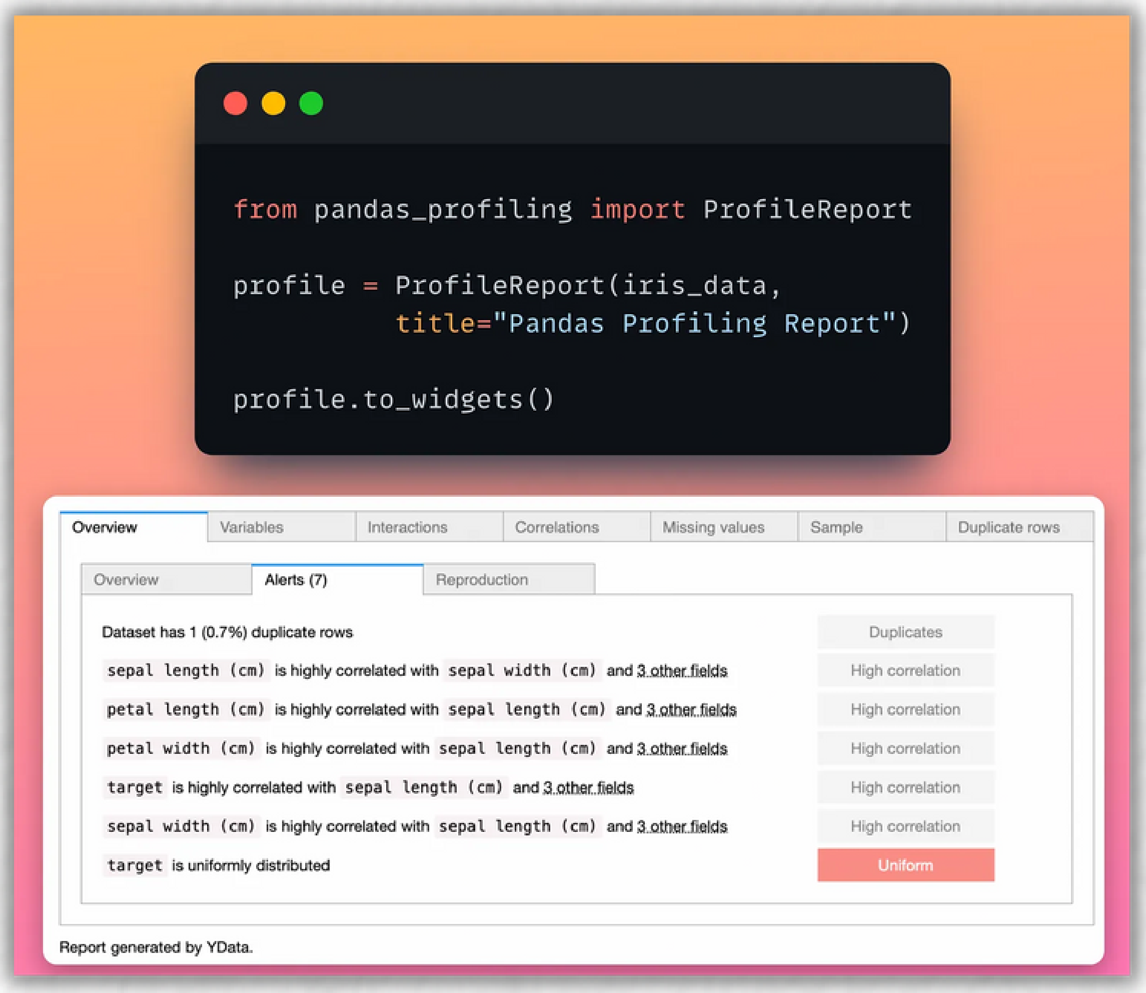


```python
import statsmodels.api as sm

results = sm.OLS(y, X).fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.178
Model:                            OLS   Adj. R-squared:                  0.161
Method:                 Least Squares   F-statistic:                     10.51
Date:                Wed, 02 Nov 2022   Prob (F-statistic):           7.41e-05
Time:                        17:12:45   Log-Likelihood:                -20.926
No. Observations:                 100   AIC:                             47.85
Df Residuals:                      97   BIC:                             55.67
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.4713      0.075     19.579      0.000       1.322       1.620
x1             0.1045      0.105      1.000      0.320      -0.103       0.312
x2             0.4831      0.107      4.503      0.000       0.270       0.696
==============================================================================
Omnibus:                       39.684   Durbin-Watson:                   1.848
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                6.503
Skew:                           0.096   Prob(JB):                       0.0387
Kurtosis:                       1.766   Cond. No.                         5.09
==============================================================================
```

Statistical testing and data exploration at fingertips.

17. Pandas-Profiling



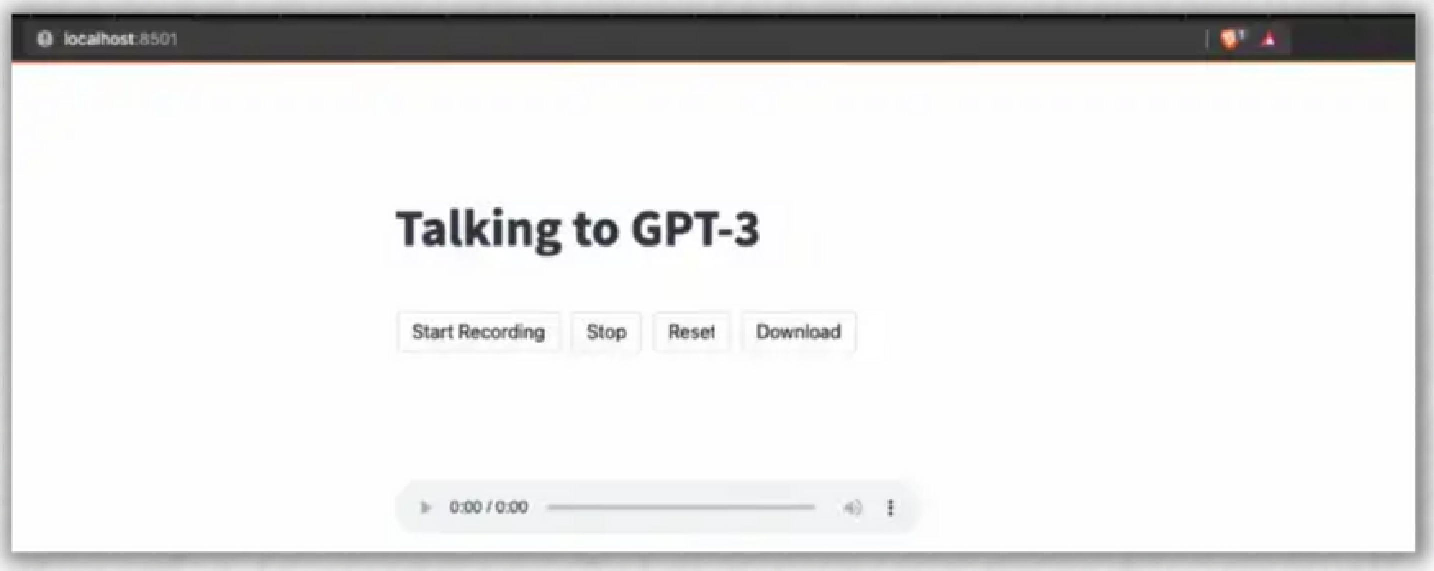
```python
from pandas_profiling import ProfileReport

profile = ProfileReport(iris_data,
          title="Pandas Profiling Report")

profile.to_widgets()
```

Dataset has 1 (0.7%) duplicate rows	Duplicates
`sepal length (cm)` is highly correlated with `sepal width (cm)` and 3 other fields	High correlation
`petal length (cm)` is highly correlated with `sepal length (cm)` and 3 other fields	High correlation
`petal width (cm)` is highly correlated with `sepal length (cm)` and 3 other fields	High correlation
`target` is highly correlated with `sepal length (cm)` and 3 other fields	High correlation
`sepal width (cm)` is highly correlated with `sepal length (cm)` and 3 other fields	High correlation
`target` is uniformly distributed	Uniform

Report generated by YData.

Generate a high-level **EDA report** of your data in no time.

18. Streamlit



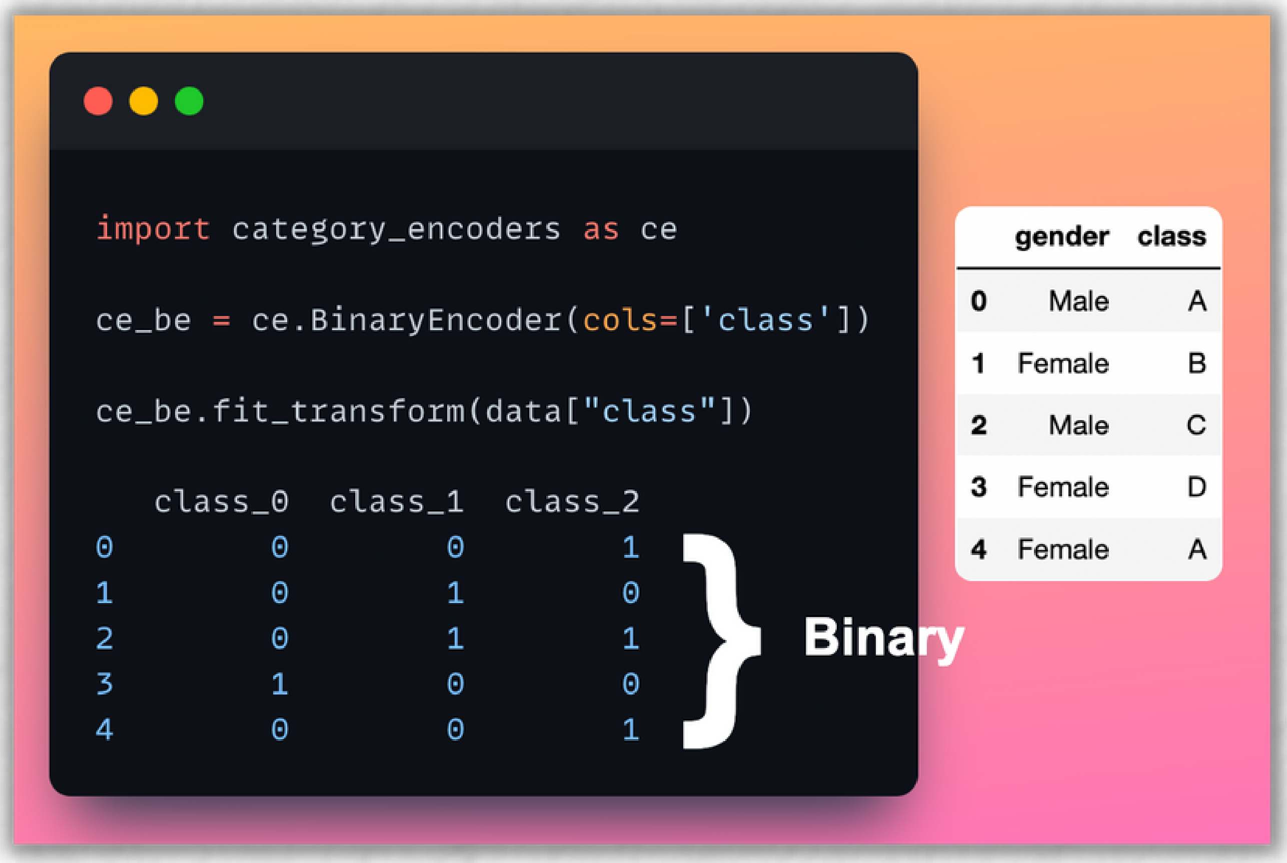
localhost:8501

Talking to GPT-3

Create and host data-based Python web apps

in few lines of code.

19. Category-encoders

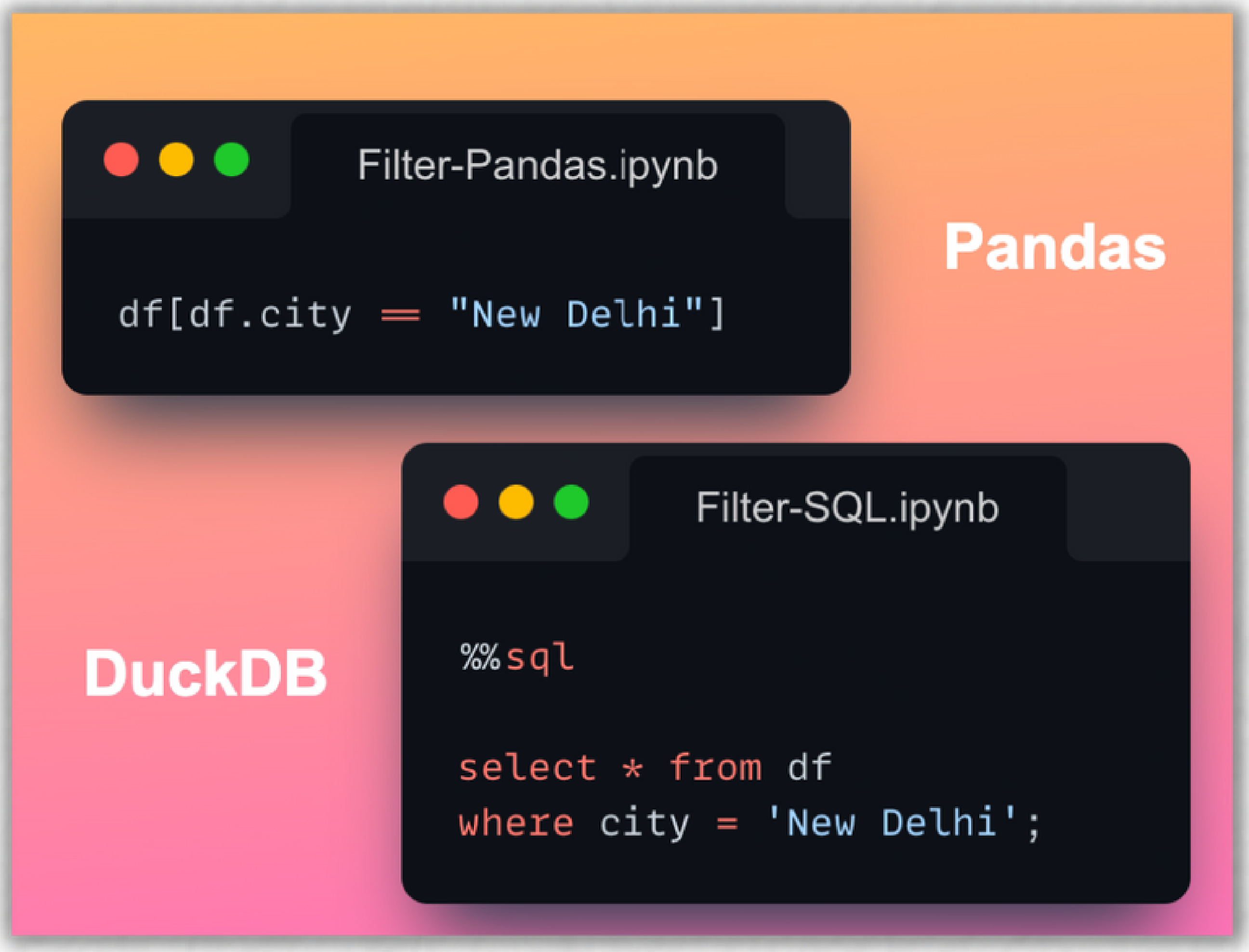


```
import category_encoders as ce

ce_be = ce.BinaryEncoder(cols=['class'])

ce_be.fit_transform(data["class"])

   class_0  class_1  class_2
0        0        0        1
1        0        1        0
2        0        1        1
3        1        0        0
4        0        0        1
```

	gender	class
0	Male	A
1	Female	B
2	Male	C
3	Female	D
4	Female	A

Over 15 categorical data encoders.

20. DuckDB

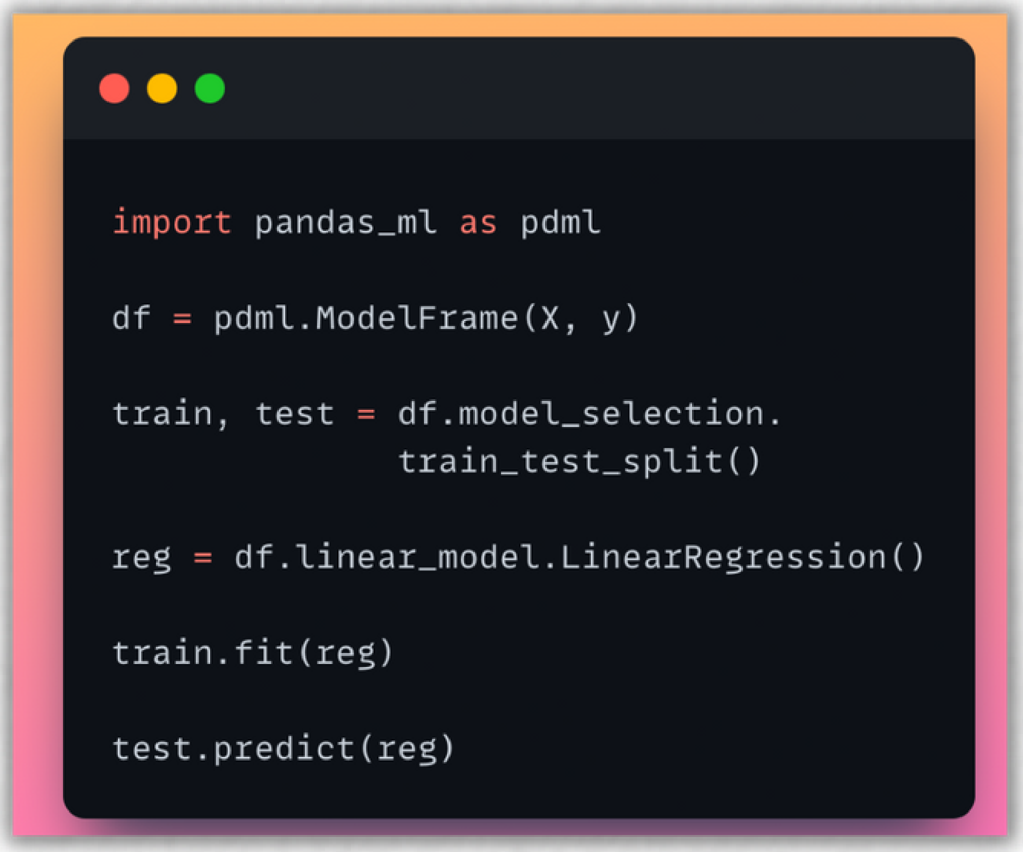


```
df[df.city == "New Delhi"]
```

```
%%sql

select * from df
where city = 'New Delhi';
```

Run SQL queries on DataFrame.

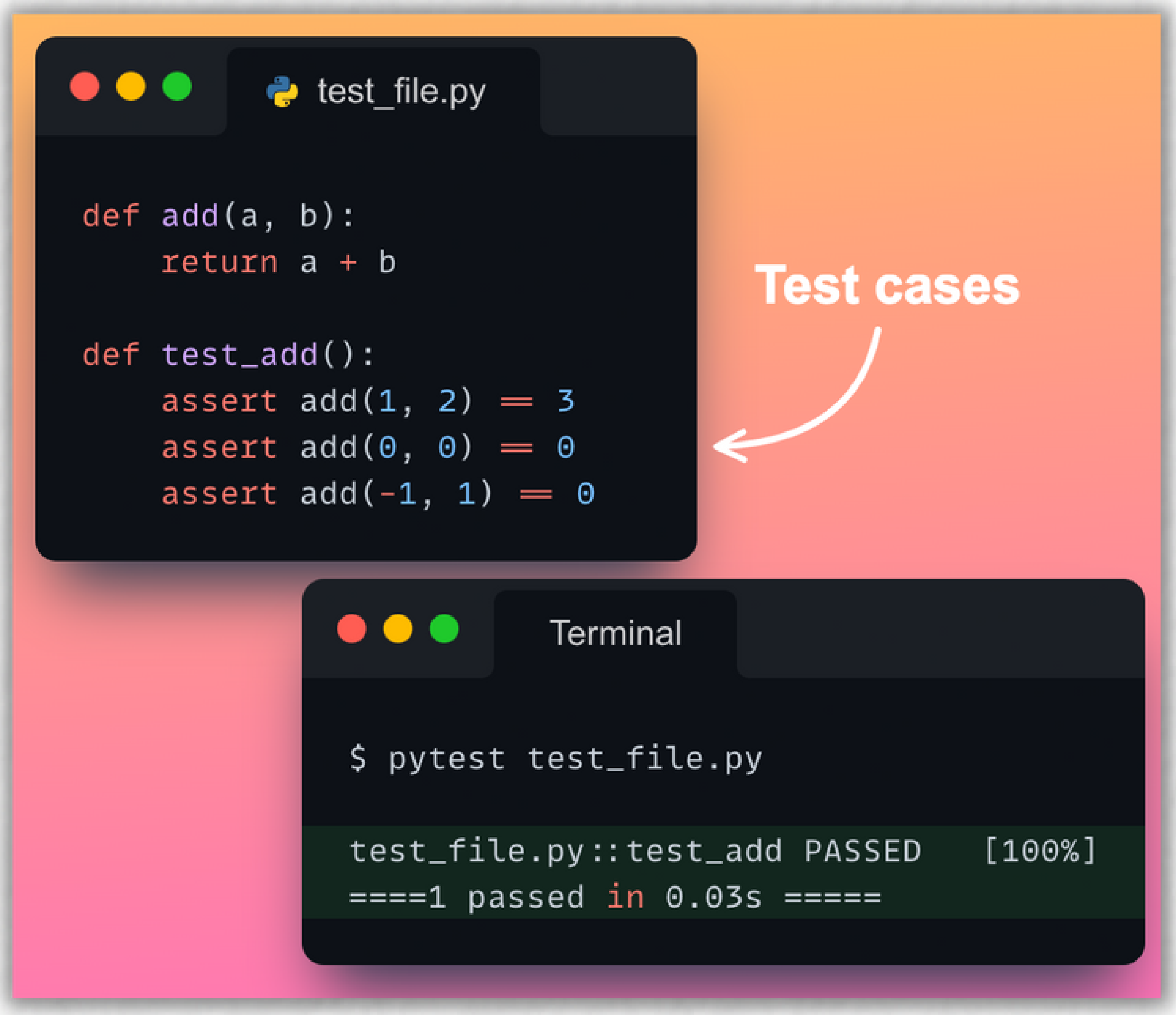
21. PandasML



```
import pandas_ml as pdml

df = pdml.ModelFrame(X, y)

train, test = df.model_selection.
              train_test_split()

reg = df.linear_model.LinearRegression()

train.fit(reg)

test.predict(reg)
```

Pandas data wrangling +
Sklearn algorithms +
Matplotlib visualization.

22. Pytest



test_file.py

```python
def add(a, b):
    return a + b

def test_add():
    assert add(1, 2) == 3
    assert add(0, 0) == 0
    assert add(-1, 1) == 0
```

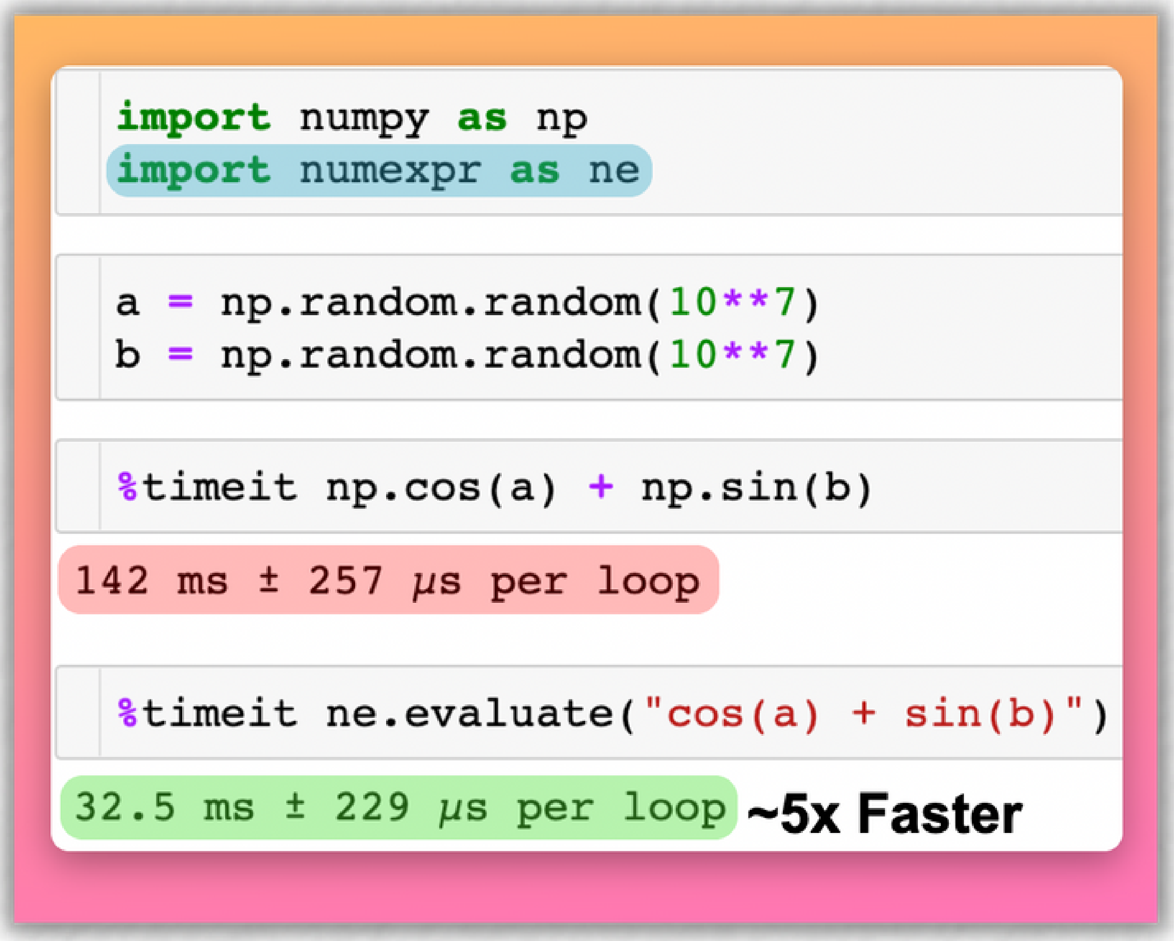
Terminal

```
$ pytest test_file.py

test_file.py::test_add PASSED   [100%]
====1 passed in 0.03s =====
```

Test cases

An **elegant testing framework** to test your code.

23. Numexpr



```
import numpy as np
import numexpr as ne
```

```
a = np.random.random(10**7)
b = np.random.random(10**7)
```

```
%timeit np.cos(a) + np.sin(b)
```

```
142 ms ± 257 µs per loop
```

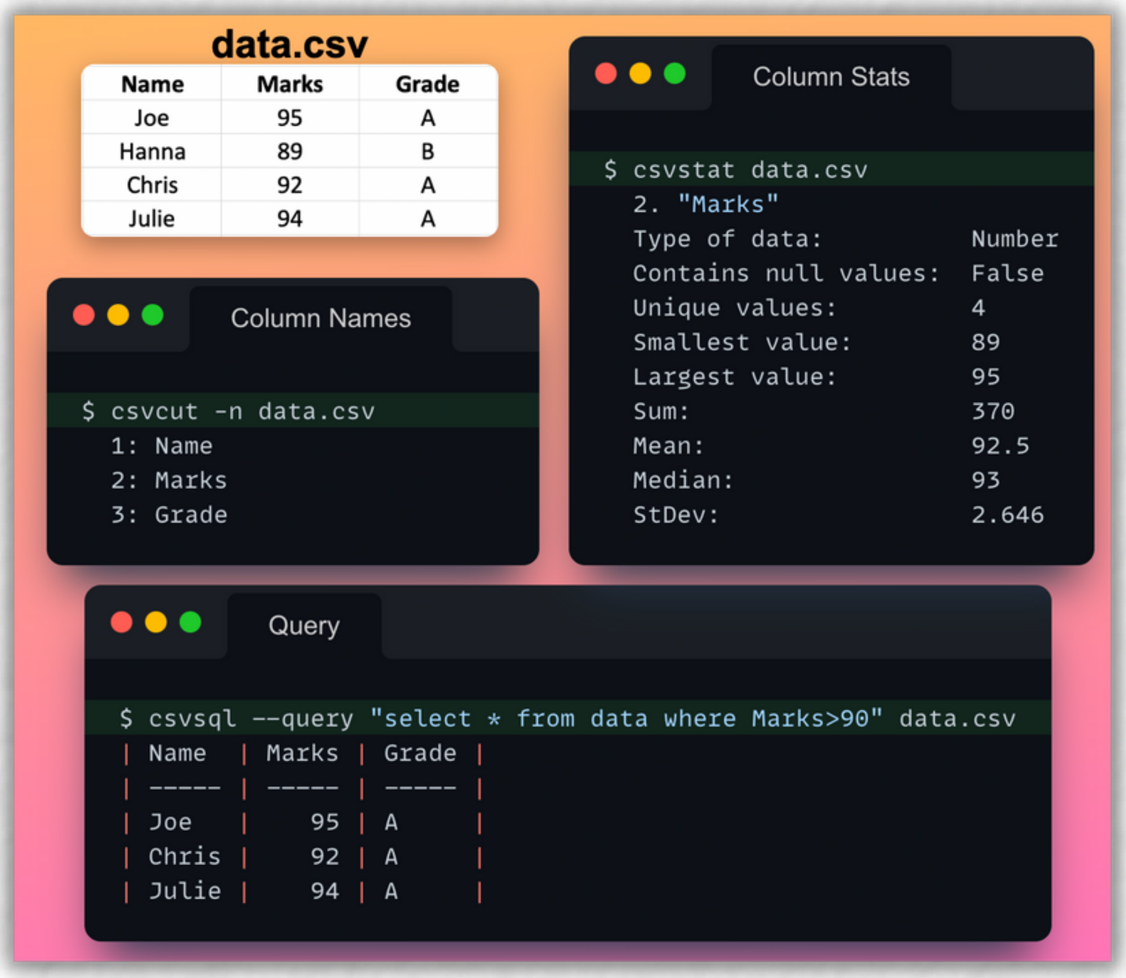
```
%timeit ne.evaluate("cos(a) + sin(b)")
```

```
32.5 ms ± 229 µs per loop
```

~5x Faster

Parallelize NumPy to all CPU cores for **20x speedup**.

24. CSV-Kit



Name	Marks	Grade
Joe	95	A
Hanna	89	B
Chris	92	A
Julie	94	A

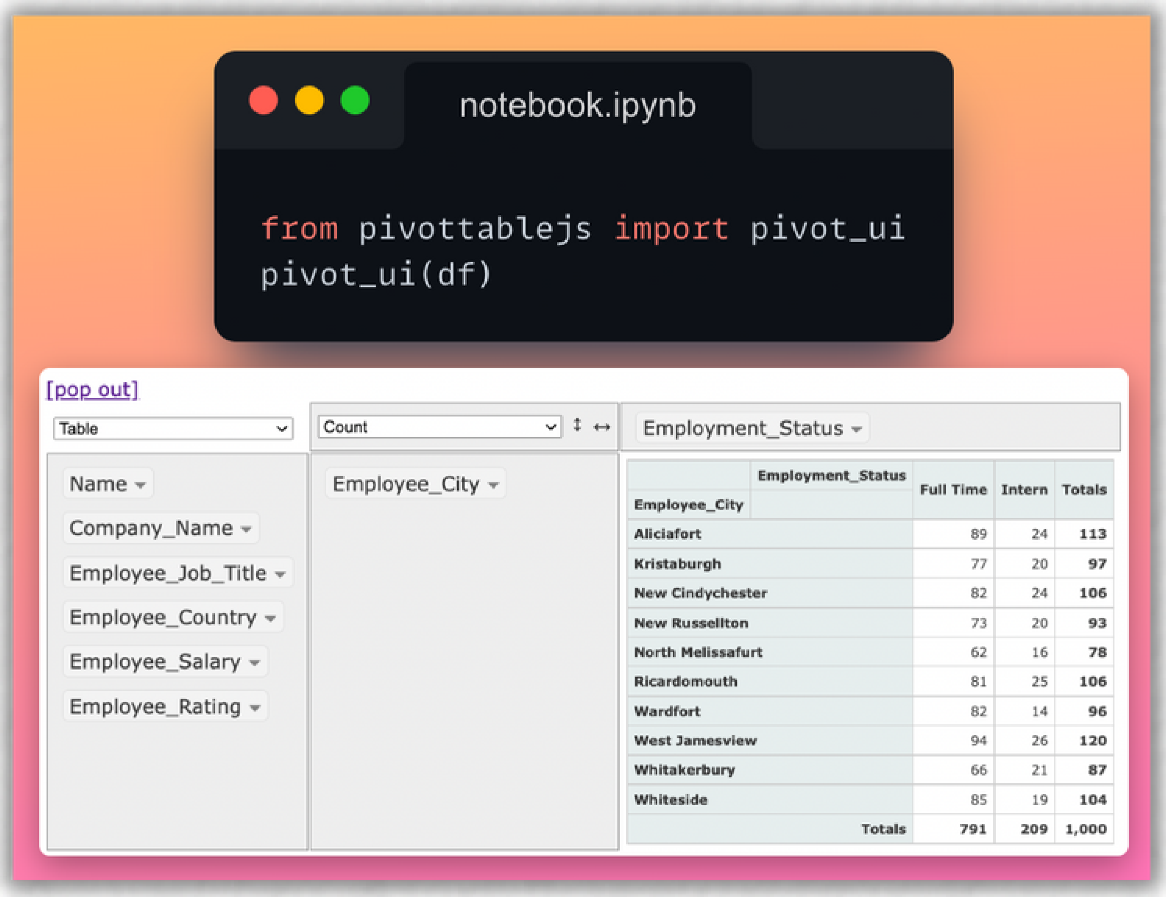
```
$ csvcut -n data.csv
  1: Name
  2: Marks
  3: Grade
```

```
$ csvstat data.csv
  2. "Marks"
  Type of data:           Number
  Contains null values:   False
  Unique values:          4
  Smallest value:         89
  Largest value:          95
  Sum:                    370
  Mean:                   92.5
  Median:                 93
  StDev:                  2.646
```

```
$ csvsql --query "select * from data where Marks>90" data.csv
| Name  | Marks | Grade |
| ----- | ----- | ----- |
| Joe   |    95 | A     |
| Chris |    92 | A     |
| Julie |    94 | A     |
```

Explore, query and describe CSV files from terminal.

25. PivotTableJS



notebook.ipynb

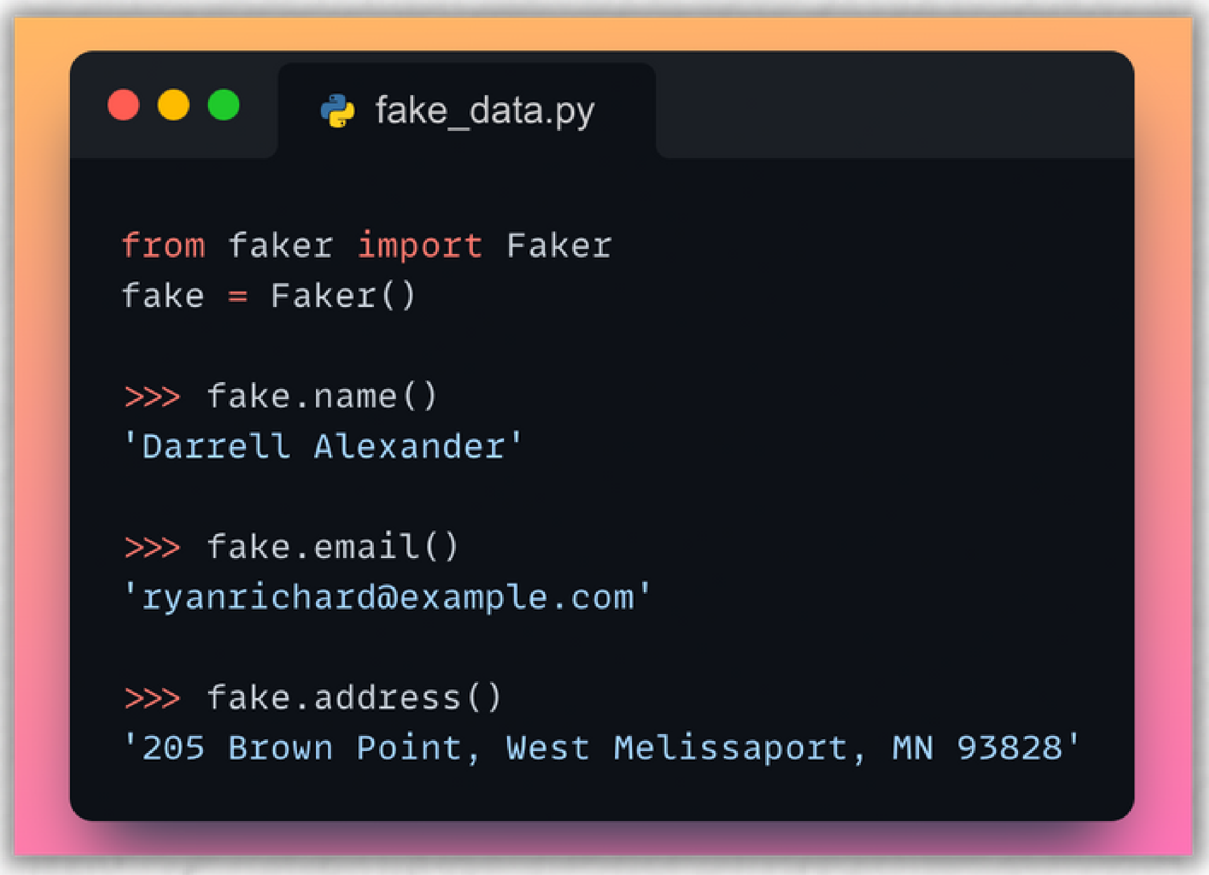
```
from pivottablejs import pivot_ui
pivot_ui(df)
```

[pop out]

	Employment_Status	Full Time	Intern	Totals
Employee_City				
Aliciafort		89	24	113
Kristaburgh		77	20	97
New Cindychester		82	24	106
New Russellton		73	20	93
North Melissafurt		62	16	78
Ricardomouth		81	25	106
Wardfort		82	14	96
West Jamesview		94	26	120
Whitakerbury		66	21	87
Whiteside		85	19	104
	Totals	791	209	1,000

Drap-n-drop tools to group, pivot, plot dataframe.

26. Faker

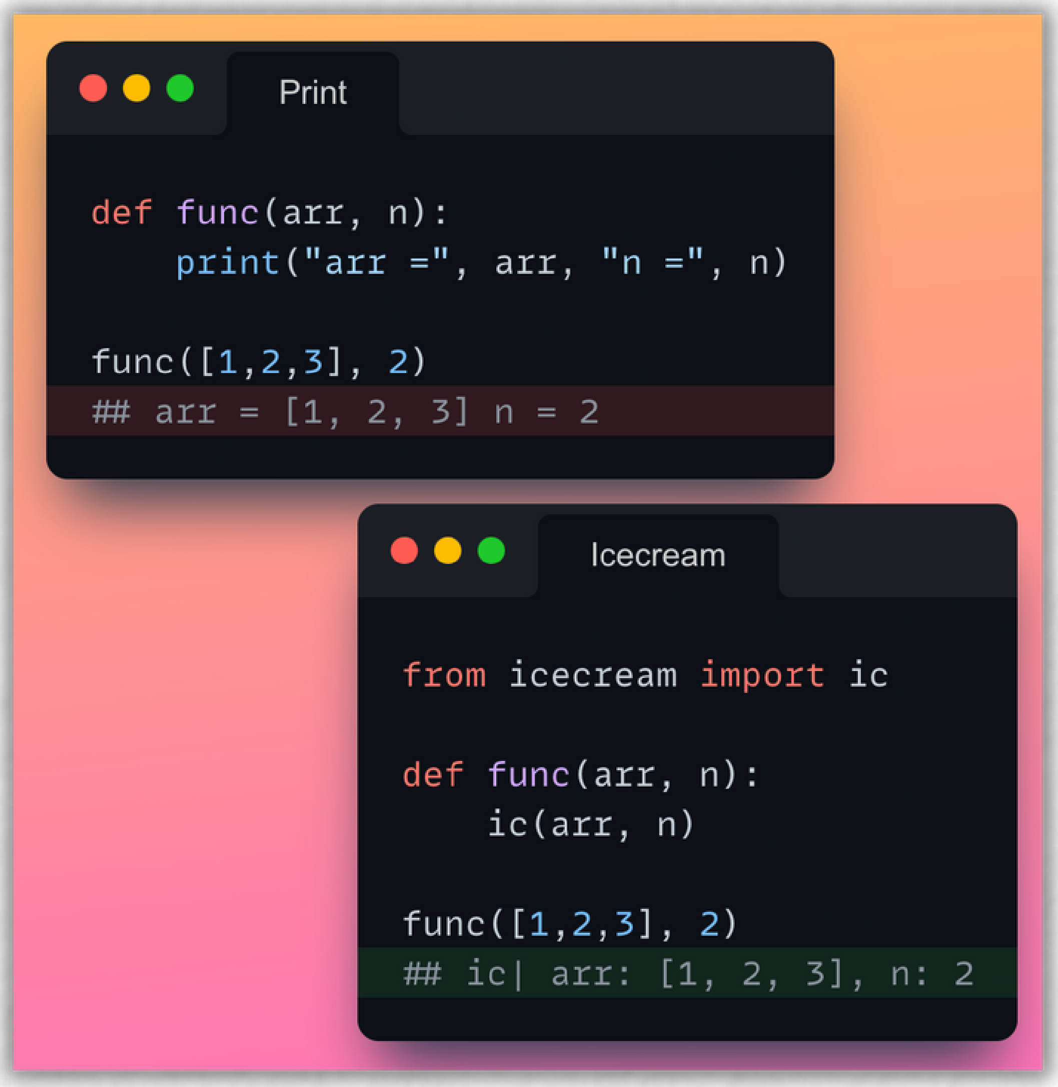


fake_data.py

```
from faker import Faker
fake = Faker()

>>> fake.name()
'Darrell Alexander'

>>> fake.email()
'ryanrichard@example.com'

>>> fake.address()
'205 Brown Point, West Melissaport, MN 93828'
```

Generate **fake yet meaningful data** in seconds.

27. Icecream

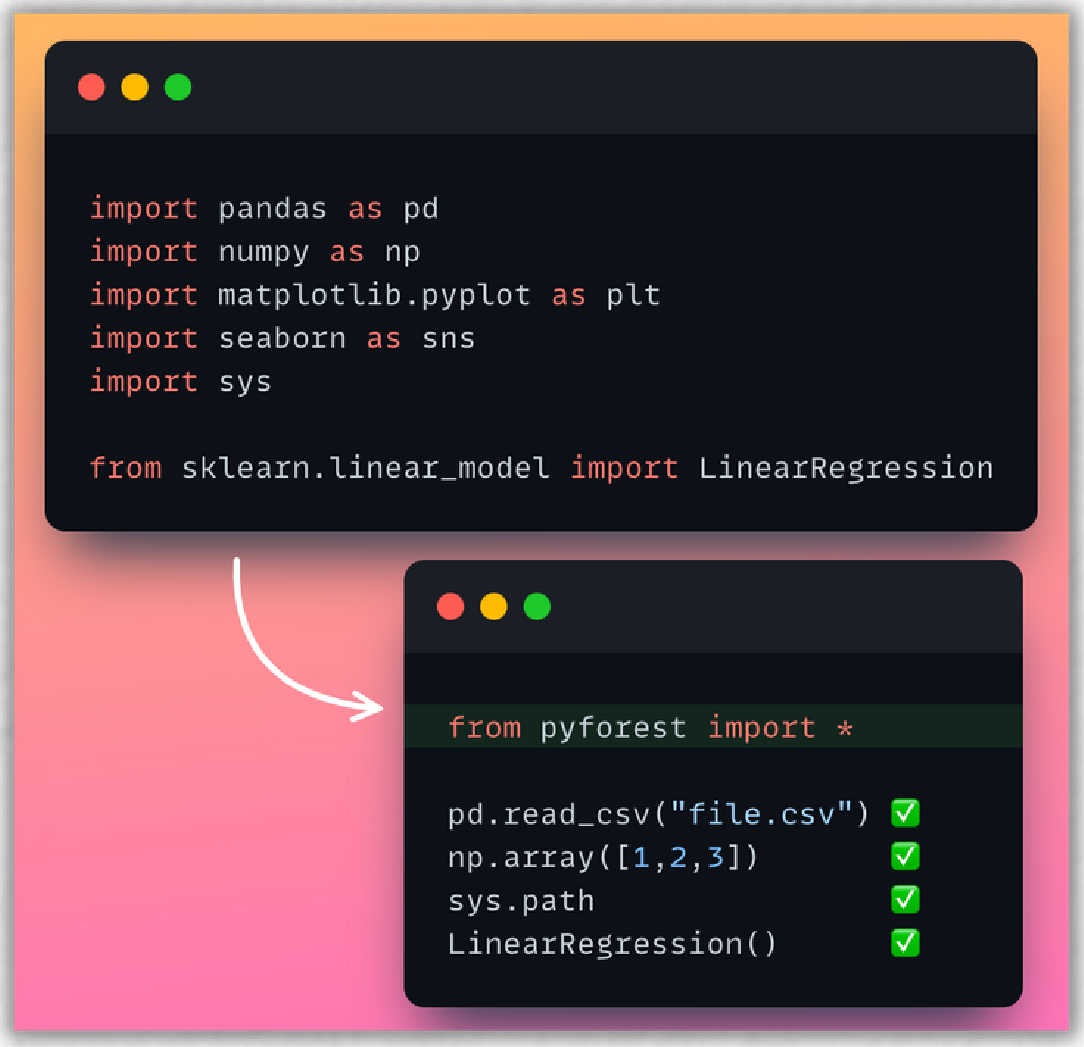


```
def func(arr, n):
    print("arr =", arr, "n =", n)

func([1,2,3], 2)
## arr = [1, 2, 3] n = 2
```

```
from icecream import ic

def func(arr, n):
    ic(arr, n)

func([1,2,3], 2)
## ic| arr: [1, 2, 3], n: 2
```

Don't debug with print(). Use icecream.

28. Pyforest

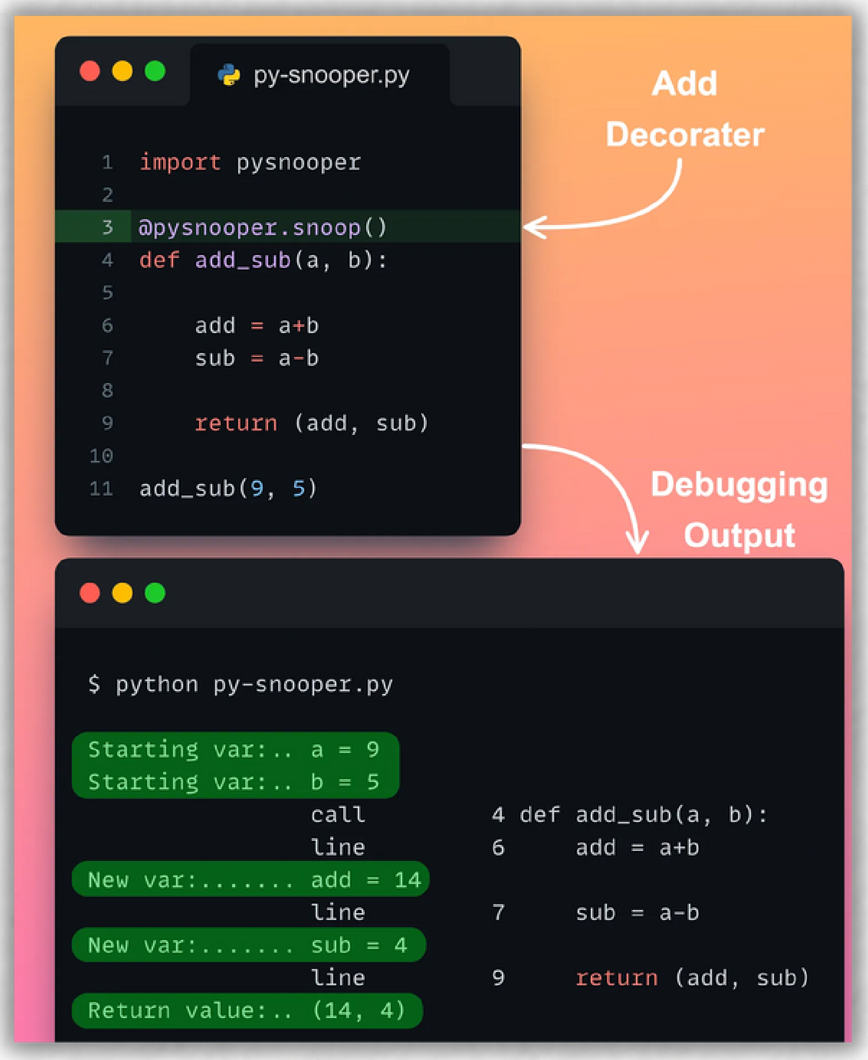


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys

from sklearn.linear_model import LinearRegression
```

```
from pyforest import *

pd.read_csv("file.csv") ✅
np.array([1,2,3])       ✅
sys.path                ✅
LinearRegression()      ✅
```

No need to write imports.
Automatic package import.

29. PySnooper



Add Decorater

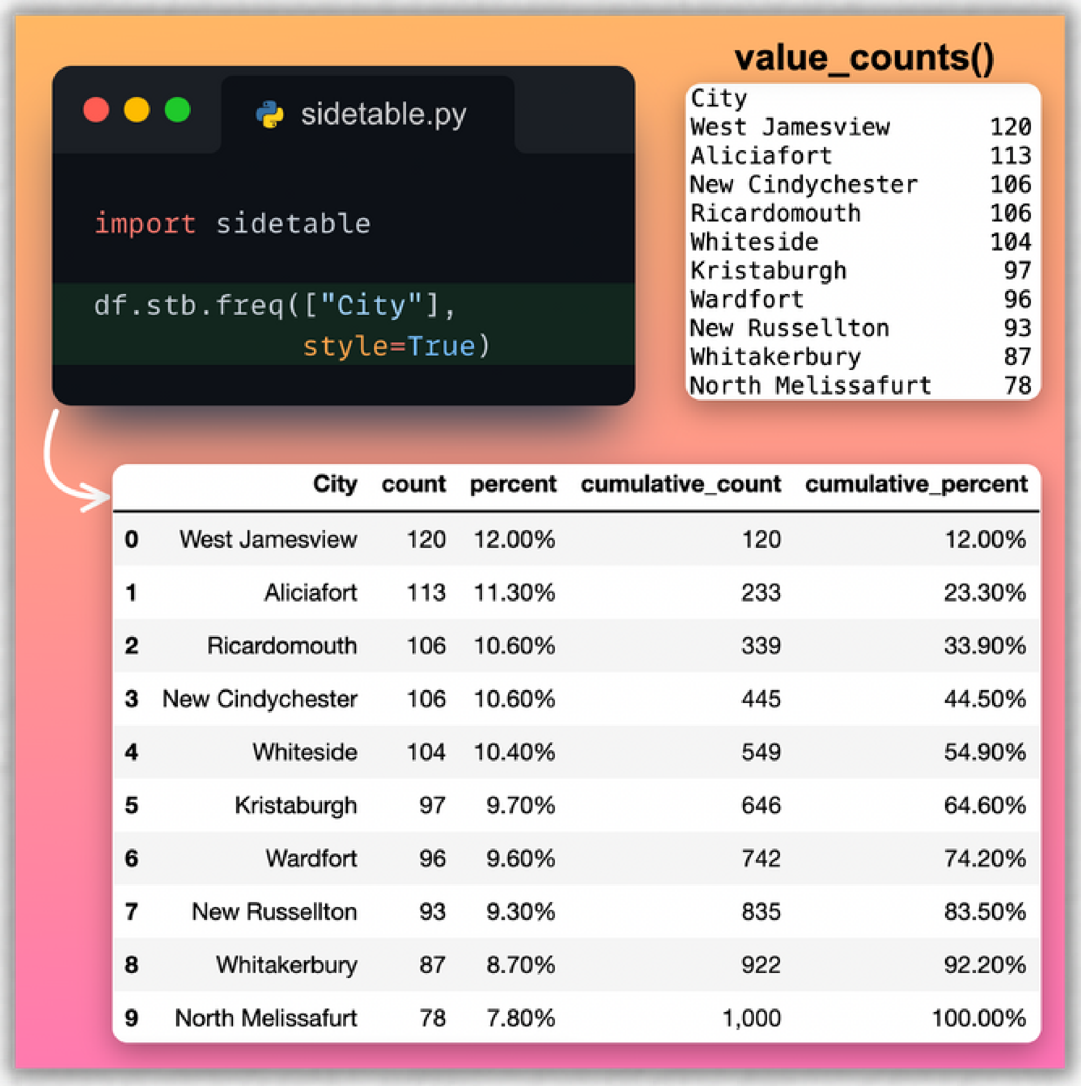
```python
import pysnooper

@pysnooper.snoop()
def add_sub(a, b):

    add = a+b
    sub = a-b

    return (add, sub)

add_sub(9, 5)
```

Debugging Output

```
$ python py-snooper.py

Starting var:.. a = 9
Starting var:.. b = 5
                call         4 def add_sub(a, b):
                line         6     add = a+b
New var:....... add = 14
                line         7     sub = a-b
New var:....... sub = 4
                line         9     return (add, sub)
Return value:.. (14, 4)
```

Profile your code. Track new variables, and their updates.

30. Sidetable



sidetable.py

```python
import sidetable

df.stb.freq(["City"],
            style=True)
```

value_counts()

```
City
West Jamesview        120
Aliciafort            113
New Cindychester      106
Ricardomouth          106
Whiteside             104
Kristaburgh            97
Wardfort               96
New Russellton         93
Whitakerbury           87
North Melissafurt      78
```

	City	count	percent	cumulative_count	cumulative_percent
0	West Jamesview	120	12.00%	120	12.00%
1	Aliciafort	113	11.30%	233	23.30%
2	Ricardomouth	106	10.60%	339	33.90%
3	New Cindychester	106	10.60%	445	44.50%
4	Whiteside	104	10.40%	549	54.90%
5	Kristaburgh	97	9.70%	646	64.60%
6	Wardfort	96	9.60%	742	74.20%
7	New Russellton	93	9.30%	835	83.50%
8	Whitakerbury	87	8.70%	922	92.20%
9	North Melissafurt	78	7.80%	1,000	100.00%

Supercharge Pandas' **value_counts()** method.

Hope that helped.

Daily Dose of Data Science

Checkout my daily newsletter to learn something new about Python and Data Science everyday.

avichawla.substack.com

Connect with me on LinkedIn.

https://www.linkedin.com/in/avi-chawla