ML-Pipeline-Mastery

January 23, 2021

0.0.1 Machine Learning Pipelines Crash Course

Table of Contents

By

- Jesse E.Agbe(JCharis)
- Jesus Saves @JCharisTech

What is an ML Pipeline A Pipeline consists of a chain of processing elements or functions arranged so that the output of each element is the input of the next. It is a way of chaining functions and task to that are usually in a workflow. It is used in several fields such as Data Science, Machine Learning and DevOps ,Manufacturing as well as General Software development. It is a continous life cycle similar to how an assembly line works in the manufacturing industry.

ML Workflow/Data Science Life Cycle So what is an ML Pipeline? An Machine Learning Pipeline refers to + A means of automating the ML workflow + A way to codify and automate how we produce a usable ML model + An independently executable workflow of a complete ML task + The act of executing task in sequence automatically + Take data, Transform Data, Train and Build A Model to Achieve an Output + Use to package workflows or sequence of tasks

In summary it is the act of automating the normal ML workflow or Data Science life cycle in order to improve efficiency and to keep things well organised and reproducible.

Main Purpose:

- Is to allow you to increase the efficiency via controlled iteration cycle and maintainable model production and deployment.

Advantages of Using ML Pipelines

- It makes building models more efficient and simplified
- Helps cut redundant work

- Moves the product from just the model to the pipeline/workflow and this improves efficiency and scalablitity
- Easy to monitor each components
- Fast iteration cycle
- A Pipeline reduces the chance of error and saves time by automating repetitive tasks.
- Allows you to monitor and tune processes

Reason For ML Pipeline

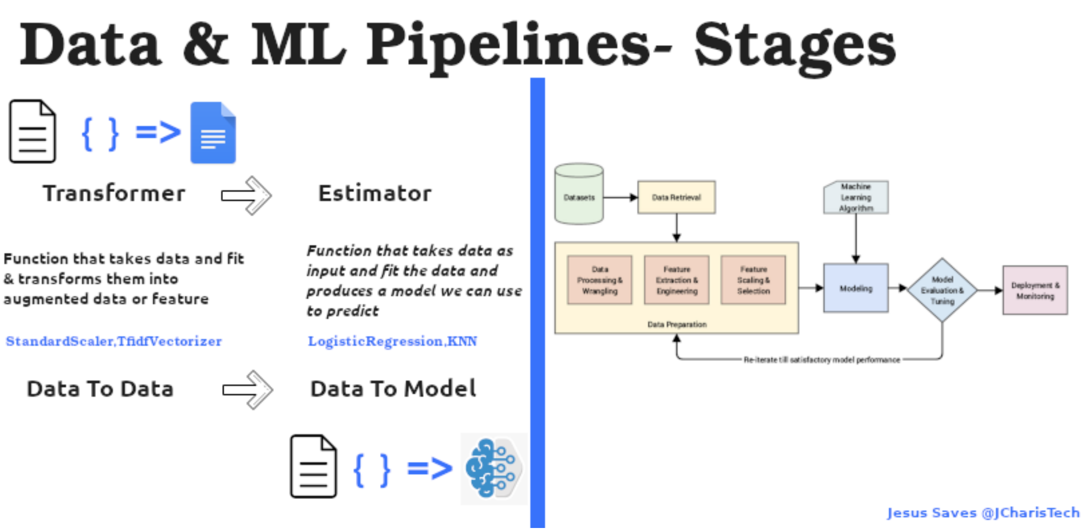
- Data Divergence: changing data means you have to change everything
- Volume: as you increase the same data you will still have to repeat the same basic steps of data preparation which can be reduntact
- Versioning: Changing data or model means you have to manually change every part of the scripts
- Data Version:identify and revert data changes and track them
- We reuse similar code for most things

0.0.2 Types of Pipeine in Datascience

There are several pipelines used in any Data Science Project. These types of pipeline depends on the problem and task at hand. The concept of using a pipeline can be applied in diverse fields + ETL/EDA Pipeline + NLP Pipeline + ML Pipeline + CI/CD Pipeline

0.0.3 Stages of ML Pipeline

Every ML Pipeline consist of several tasks which can be classified based on their input to output



flow.

Based on that we can have two main stages of an ML Pipeline which includes + Data to Data Pipeline - This involves the stage in which we take in or ingest data and then produce as an output data. - The main concept here is via the use of Transformers - a function that takes in data as input and convert/transforms that data into another form of data that is usually augmented or feature ready. - This starts from data ingestion to data cleaning to data transformation and verification as well as feature engineering

- Data to Model Pipeline

    - This is the stage in which we take in data (usually a prepared data) as input and then learns from it to produce a model.
    - We take in data and using an Estimator we fit the data and build a model that can be used on other dataset to perform predictive task
    - The main concept here is via the use of Estimators
        * An Estimator is a function that fit data and yields a model
        * eg All the ML Algorithms such as KNN, Logistic Regression,MLP etc

Use Cases Text Classification NLP Task

- Fetch/Ingest Data
- Text Cleaning
- Text Preprocessing
- Train Data to Build Model

Tips For Building ML Pipelines

- Modular instead of Monolith
- Add test to modular components
- Automate when needed
- Know where to get inputs and where to place result

Components of Pipelines Stages

- Transformer Stage: a transformer takes a dataset as input and produces a transformed/augmented dataset as output. It takes in data and convert it into a feature ready dataset. eg Tokenizer
- Estimator Stage: an estimator is fitted on an input dataset and produces a model that can be used to perform predictive task. eg Naive Bayes, Logistic Regression

Building A Simple ML Pipeline with Scikit Learn

- Scikit-learn
    - Transformers
        * StandardScaler
        * MinMaxScaler
        * Tokenizers
        * TfidfVectorizer
    - Estimators
        * Linear_models: LogisticRegression
        * Neigbours : KNN
- In Sklearn it is always transformers first then estimators
- Fit & Transform Before Fit & Predict

[1]:
```python
# Load EDA Pkgs
import pandas as pd
import numpy as np
```

[2]:
```python
# Load ML Pkgs
# Estimators
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# Transformers
from sklearn.preprocessing import StandardScaler,MinMaxScaler

# Utils
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

[27]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

[24]:
```python
import warnings
warnings.filterwarnings('ignore')
```

[4]:
```python
# Load Dataset
df = pd.read_csv("data/hcvdata.csv")
```

[5]:
```python
df.head()
```

[5]:
```
   Unnamed: 0       Category  Age Sex   ALB   ALP   ALT   AST   BIL    CHE  \
0           1  0=Blood Donor   32   m  38.5  52.5   7.7  22.1   7.5   6.93
1           2  0=Blood Donor   32   m  38.5  70.3  18.0  24.7   3.9  11.17
2           3  0=Blood Donor   32   m  46.9  74.7  36.2  52.6   6.1   8.84
3           4  0=Blood Donor   32   m  43.2  52.0  30.6  22.6  18.9   7.33
4           5  0=Blood Donor   32   m  39.2  74.1  32.6  24.8   9.6   9.15

   CHOL   CREA   GGT  PROT
0  3.23  106.0  12.1  69.0
1  4.80   74.0  15.6  76.5
2  5.20   86.0  33.2  79.3
3  4.74   80.0  33.8  75.7
4  4.32   76.0  29.9  68.7
```

[6]:
```python
df.columns
```

[6]:
```
Index(['Unnamed: 0', 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST',
       'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT'],
      dtype='object')
```

[7]:
```python
# Check Shape
df.shape
```

[7]: (615, 14)

[8]:
```python
# Data Cleaning
df['Category'].unique()
```

[8]:
```
array(['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',
       '2=Fibrosis', '3=Cirrhosis'], dtype=object)
```

[9]:
```python
cat_dict = {'0=Blood Donor':0, '0s=suspect Blood Donor':0, '1=Hepatitis':1,
       '2=Fibrosis':2, '3=Cirrhosis':3}
```

[10]:
```python
# Method 1
df['Category'].map(cat_dict)
```

[10]:
```
0      0
1      0
2      0
3      0
4      0
      ..
610    3
611    3
612    3
613    3
614    3
Name: Category, Length: 615, dtype: int64
```

[11]:
```python
# Method 2 using split
df['Category'].str.split("=")
```

[11]:
```
0      [0, Blood Donor]
1      [0, Blood Donor]
2      [0, Blood Donor]
3      [0, Blood Donor]
4      [0, Blood Donor]
             …
610      [3, Cirrhosis]
611      [3, Cirrhosis]
612      [3, Cirrhosis]
613      [3, Cirrhosis]
614      [3, Cirrhosis]
Name: Category, Length: 615, dtype: object
```

[14]:
```python
# Method 2 using split
df['Category'].str.split("=").str.get(0).str.replace('s','').astype(int)
```

[14]:
```
0      0
1      0
2      0
3      0
4      0
      ..
610    3
611    3
612    3
613    3
614    3
Name: Category, Length: 615, dtype: int64
```

[15]:
```
df['Category'] = df['Category'].map(cat_dict)
```

[18]:
```
# Change Gender
df['Sex'] = df['Sex'].map({'m':1,'f':0})
```

[19]:
```
df.columns
```

[19]:
```
Index(['Unnamed: 0', 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST',
       'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT'],
      dtype='object')
```

[20]:
```
df = df[['Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST',
       'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT','Category']]
```

[21]:
```
df.head()
```

[21]:

	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT	\
0	32	1	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106.0	12.1	69.0	
1	32	1	38.5	70.3	18.0	24.7	3.9	11.17	4.80	74.0	15.6	76.5	
2	32	1	46.9	74.7	36.2	52.6	6.1	8.84	5.20	86.0	33.2	79.3	
3	32	1	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7	
4	32	1	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76.0	29.9	68.7	

	Category
0	0
1	0
2	0
3	0
4	0

[37]:
```
# Check Missing Values
df.isnull().sum()
```

[37]:
```
Age          0
Sex          0
ALB          1
ALP         18
ALT          1
AST          0
BIL          0
CHE          0
CHOL        10
CREA         0
GGT          0
PROT         1
Category     0
dtype: int64
```

[38]:
```python
# Fill Missing Values with 0
df.fillna(0,inplace=True)
```

[39]:
```python
# Check Missing Values
df.isnull().sum()
```

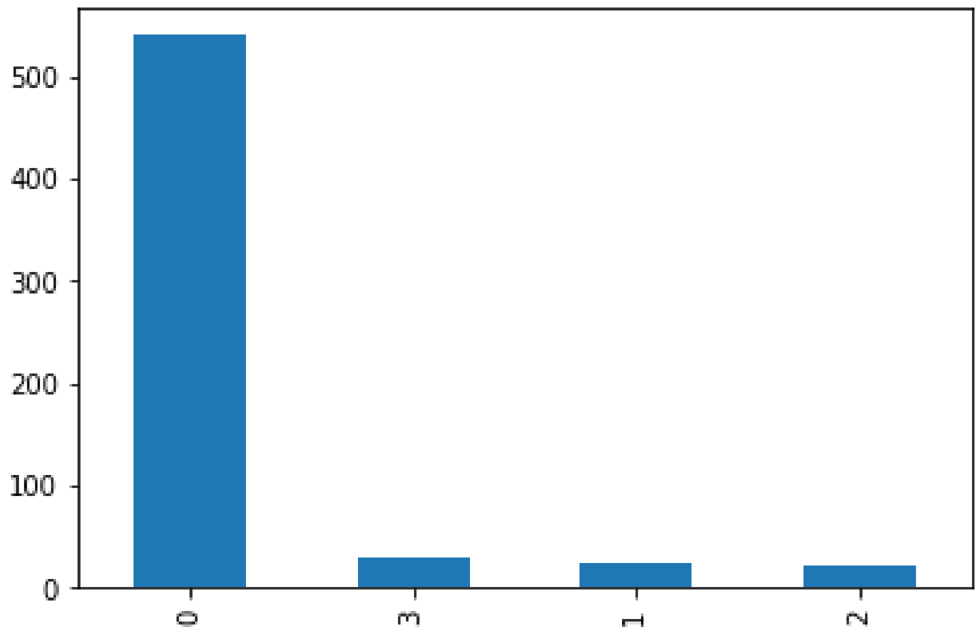
[39]:
```
Age         0
Sex         0
ALB         0
ALP         0
ALT         0
AST         0
BIL         0
CHE         0
CHOL        0
CREA        0
GGT         0
PROT        0
Category    0
dtype: int64
```

[40]:
```python
# Value Counts
df['Category'].value_counts()
```

[40]:
```
0    540
3     30
1     24
2     21
Name: Category, dtype: int64
```

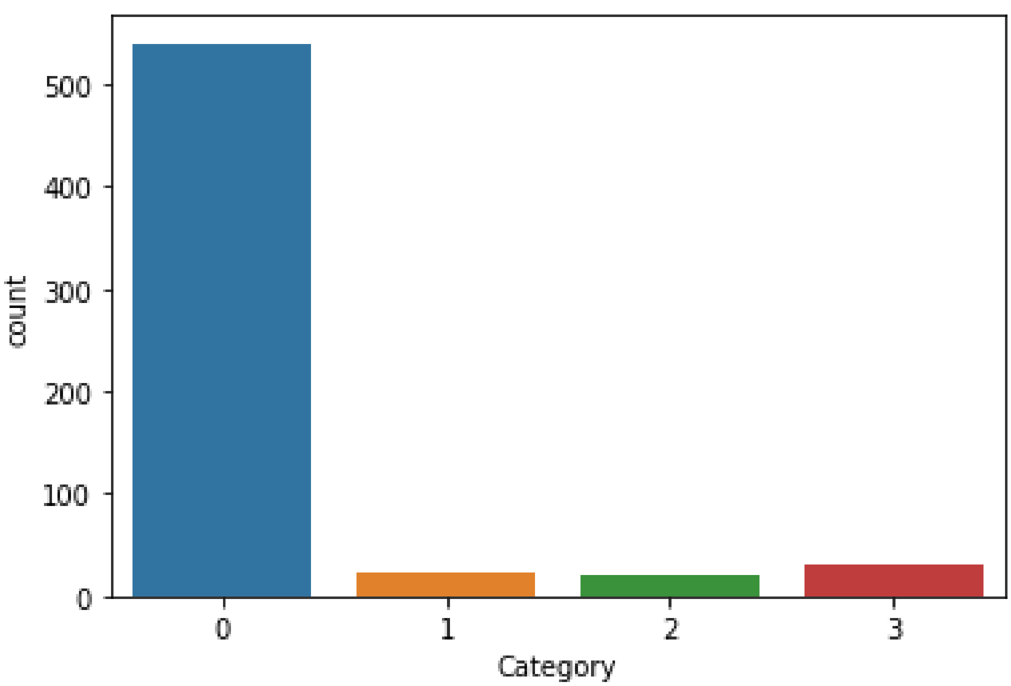
[41]:
```python
# Value Counts
df['Category'].value_counts().plot(kind='bar')
```

[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8faeb56190>



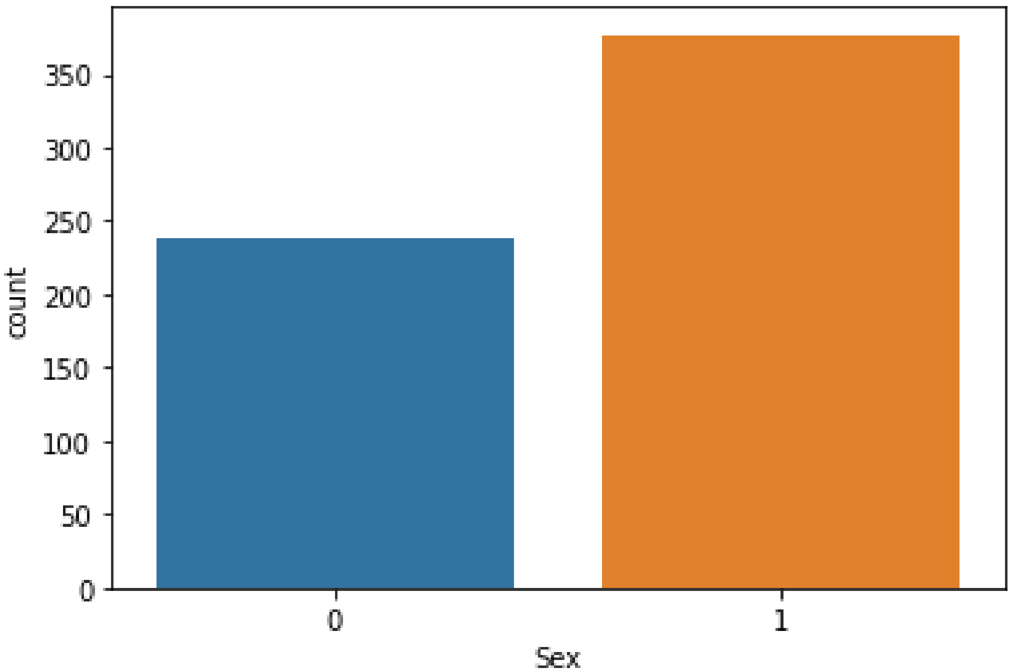
[42]:
```python
# Value Counts
sns.countplot(df['Category'])
```

[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8fae572fd0>



[43]:
```
# Value Counts
sns.countplot(df['Sex'])
```

[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8fae418850>



[44]:
```
# Features & Labels
Xfeatures = df.drop('Category',axis=1)
ylabels = df['Category']
```

[45]:
```
Xfeatures
```

[45]:
```
     Age  Sex   ALB    ALP   ALT    AST   BIL    CHE  CHOL   CREA    GGT  \
0     32    1  38.5   52.5   7.7   22.1   7.5   6.93  3.23  106.0   12.1
1     32    1  38.5   70.3  18.0   24.7   3.9  11.17  4.80   74.0   15.6
2     32    1  46.9   74.7  36.2   52.6   6.1   8.84  5.20   86.0   33.2
3     32    1  43.2   52.0  30.6   22.6  18.9   7.33  4.74   80.0   33.8
4     32    1  39.2   74.1  32.6   24.8   9.6   9.15  4.32   76.0   29.9
..   …  …   …    …    …  …    …    …  …    …     …
610   62    0  32.0  416.6   5.9  110.3  50.0   5.57  6.30   55.7  650.9
611   64    0  24.0  102.8   2.9   44.4  20.0   1.54  3.02   63.0   35.9
612   64    0  29.0   87.3   3.5   99.0  48.0   1.66  3.63   66.7   64.2
613   46    0  33.0    0.0  39.0   62.0  20.0   3.56  4.20   52.0   50.0
```

```
614   59    0  36.0    0.0  100.0   80.0  12.0   9.07  5.30   67.0   34.0

     PROT
0    69.0
1    76.5
2    79.3
3    75.7
4    68.7
..    …
610  68.5
611  71.3
612  82.0
613  71.0
614  68.0

[615 rows x 12 columns]
```

[46]:
```python
# Train Test Split
X_train,X_test,y_train,y_test = train_test_split(Xfeatures,ylabels,test_size=0.
↪3,random_state=42)
```

[ ]:
```python
# Building Model (Manual Step)
```

[47]:
```python
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

[47]:
```
LogisticRegression()
```

[48]:
```python
# Prediction Accuracy
print("Accuracy:", lr.score(X_test,y_test))
```

```
Accuracy: 0.9243243243243243
```

[49]:
```python
# Scale the data
# Normalize
minmax_scaler = MinMaxScaler()
Xfeatures_scaled = minmax_scaler.fit_transform(Xfeatures)
```

[50]:
```python
Xfeatures_scaled
```

[50]:
```
array([[0.22413793, 1.        , 0.46836983, …, 0.09149473, 0.01175743,
        0.76666667],
       [0.22413793, 1.        , 0.46836983, …, 0.0616189 , 0.01717203,
        0.85      ],
       [0.22413793, 1.        , 0.57055961, …, 0.07282233, 0.04439975,
        0.88111111],
       …,
```

```
       [0.77586207, 0.        , 0.35279805, …, 0.05480347, 0.09235767,
        0.91111111],
       [0.46551724, 0.        , 0.40145985, …, 0.04107926, 0.07038985,
        0.78888889],
       [0.68965517, 0.        , 0.4379562 , …, 0.05508356, 0.04563738,
        0.75555556]])
```

[54]:
```
# Convert to DataFrame
Xfeatures_scaled = pd.DataFrame(Xfeatures_scaled,columns=Xfeatures.columns)
```

[55]:
```
Xfeatures_scaled.head()
```

[55]:
```
        Age  Sex       ALB       ALP       ALT       AST       BIL       CHE  \
0  0.224138  1.0  0.468370  0.126020  0.023670  0.036694  0.026461  0.367578
1  0.224138  1.0  0.468370  0.168747  0.055334  0.044990  0.012243  0.650434
2  0.224138  1.0  0.570560  0.179309  0.111282  0.134014  0.020932  0.494997
3  0.224138  1.0  0.525547  0.124820  0.094067  0.038290  0.071485  0.394263
4  0.224138  1.0  0.476886  0.177868  0.100215  0.045310  0.034755  0.515677

       CHOL      CREA       GGT      PROT
0  0.334023  0.091495  0.011757  0.766667
1  0.496381  0.061619  0.017172  0.850000
2  0.537746  0.072822  0.044400  0.881111
3  0.490176  0.067221  0.045328  0.841111
4  0.446743  0.063486  0.039295  0.763333
```

[61]:
```
# Train Test Split
X_train_scaled,X_test_scaled,y_train_scaled,y_test_scaled =␣
↪train_test_split(Xfeatures_scaled,ylabels,test_size=0.3,random_state=42)
```

[62]:
```
lr_scaled = LogisticRegression()
lr_scaled.fit(X_train_scaled,y_train_scaled)
```

[62]:
```
LogisticRegression()
```

[64]:
```
# Prediction Accuracy
print("Accuracy:", lr_scaled.score(X_test_scaled,y_test_scaled))
```

```
Accuracy: 0.8594594594594595
```

[65]:
```
# Scale the data
# NScaling
std_scaler = StandardScaler()
Xfeatures_std = std_scaler.fit_transform(Xfeatures)
# Convert to DataFrame
Xfeatures_std = pd.DataFrame(Xfeatures_std,columns=Xfeatures.columns)
# Train Test Split
```

```
X_train_std,X_test_std,y_train_std,y_test_std =␣
↪train_test_split(Xfeatures_std,ylabels,test_size=0.3,random_state=42)
lr_std = LogisticRegression()
lr_std.fit(X_train_std,y_train_std)
```

[65]:
```
LogisticRegression()
```

[67]:
```
# Prediction Accuracy
print("Accuracy:", lr_std.score(X_test_std,y_test_std))
```

```
Accuracy: 0.9135135135135135
```

Narrative

- Normalizing the Dataset via Min Max Scaler reduce the model accuracy from 0.92 to 0.85
- Scaling via StandardScaler reduce the model accuracy from 0.92 to 0.91

[ ]:
```
# Via Pipeline
```

[68]:
```
# Create A LogisticRegression ML Pipeline
pipe_lr = Pipeline([
    ('scaler',StandardScaler()),
    ('lr',LogisticRegression())
])
```

[69]:
```
# Using the Pipeline
pipe_lr = pipe_lr.fit(X_train, y_train)
```

[70]:
```
# Methods
dir(pipe_lr)
```

[70]:
```
['__abstractmethods__',
 '__annotations__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
```

```
'__le__',
'__len__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__setstate__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_abc_impl',
'_check_fit_params',
'_check_n_features',
'_estimator_type',
'_final_estimator',
'_fit',
'_get_param_names',
'_get_params',
'_get_tags',
'_inverse_transform',
'_iter',
'_log_message',
'_more_tags',
'_pairwise',
'_replace_estimator',
'_repr_html_',
'_repr_html_inner',
'_repr_mimebundle_',
'_required_parameters',
'_set_params',
'_sk_visual_block_',
'_transform',
'_validate_data',
'_validate_names',
'_validate_steps',
'classes_',
'decision_function',
'fit',
'fit_predict',
'fit_transform',
'get_params',
'inverse_transform',
```

```
 'memory',
 'n_features_in_',
 'named_steps',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'score',
 'score_samples',
 'set_params',
 'steps',
 'transform',
 'verbose']
```

[71]:
```python
# Get the steps
pipe_lr.steps
```

[71]:
```
[('scaler', StandardScaler()), ('lr', LogisticRegression())]
```

[87]:
```python
# Get the steps as a Dictionary
pipe_lr.named_steps
```

[87]:
```
{'scaler': StandardScaler(), 'lr': LogisticRegression()}
```

[82]:
```python
# Get Params of A step
# [stringlabel][estimator/transformer]
dir(pipe_lr.steps[0][1].)
```

[82]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
```

```
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_check_n_features',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_repr_html_',
 '_repr_html_inner',
 '_repr_mimebundle_',
 '_reset',
 '_validate_data',
 'copy',
 'fit',
 'fit_transform',
 'get_params',
 'inverse_transform',
 'mean_',
 'n_features_in_',
 'n_samples_seen_',
 'partial_fit',
 'scale_',
 'set_params',
 'transform',
 'var_',
 'with_mean',
 'with_std']
```

[84]:
```
# Get Params of A step
# [stringlabel][estimator/transformer]
pipe_lr.steps[0][1].get_params()
```

[84]:
```
{'copy': True, 'with_mean': True, 'with_std': True}
```

[85]:
```
# Get Params of A step (LogisticRegression)
# [stringlabel][estimator/transformer]
pipe_lr.steps[1][1].get_params()
```

[85]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
```

```
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

[96]:
```
# Get Params of Entire Pipeline and Details
pipe_lr.get_params()
```

[96]:
```
{'memory': None,
 'steps': [('scaler', StandardScaler()), ('lr', LogisticRegression())],
 'verbose': False,
 'scaler': StandardScaler(),
 'lr': LogisticRegression(),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'lr__C': 1.0,
 'lr__class_weight': None,
 'lr__dual': False,
 'lr__fit_intercept': True,
 'lr__intercept_scaling': 1,
 'lr__l1_ratio': None,
 'lr__max_iter': 100,
 'lr__multi_class': 'auto',
 'lr__n_jobs': None,
 'lr__penalty': 'l2',
 'lr__random_state': None,
 'lr__solver': 'lbfgs',
 'lr__tol': 0.0001,
 'lr__verbose': 0,
 'lr__warm_start': False}
```

[86]:
```
print('Accuracy score: ', pipe_lr.score(X_test, y_test))
```

```
Accuracy score:  0.9135135135135135
```

Narrative

- Same as above

[89]:
```python
# Make Prediction
X_test.iloc[0].tolist()
```

[89]: [55.0, 1.0, 28.1, 65.5, 16.6, 17.5, 2.8, 5.58, 4.39, 65.0, 26.2, 62.4]

Reshape Dataset

- Reshape your data either using array.reshape(-1, 1)
- if your data has a single feature or array.reshape(1, -1) if it contains a single sample.

[92]:
```python
ex1 = [55.0, 1.0, 28.1, 65.5, 16.6, 17.5, 2.8, 5.58, 4.39, 65.0, 26.2, 62.4]
sample = np.array(ex1).reshape(1,-1)
```

[93]:
```python
pipe_lr.predict(sample)
```

[93]: array([0])

[95]:
```python
# Get Prediction Prob
pipe_lr.predict_proba(sample)
```

[95]: array([[9.94399719e-01, 8.44097028e-05, 3.28199154e-04, 5.18767258e-03]])

[ ]:
```python
### Make Pipeline
+ Construct a Pipeline from the given estimators. This is a shorthand for the␣
↪Pipeline constructor.
```

[151]:
```python
from sklearn.pipeline import make_pipeline
```

[152]:
```python
new_pipe_lr = make_pipeline(StandardScaler(),LogisticRegression())
```

[153]:
```python
new_pipe_lr
```

[153]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

Narrative

- Automatically uses the lowercase of the Transformer/Estimator as name for step

[154]:
```python
print("Make_Pipeline:",type(new_pipe_lr))
print("Pipeline:",type(pipe_lr))
```

```
Make_Pipeline: <class 'sklearn.pipeline.Pipeline'>
Pipeline: <class 'sklearn.pipeline.Pipeline'>
```

[155]:
```python
# Fit Data
new_pipe_lr.fit(X_train,y_train)
```

[155]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

[156]:
```
new_pipe_lr.score(X_test,y_test)
```

[156]:
```
0.9135135135135135
```

[162]:
```
X_train
```

[162]:
```
     Age  Sex   ALB   ALP   ALT   AST   BIL    CHE  CHOL  CREA   GGT  PROT
296   64    1  44.5  87.8  15.1  23.2  12.3   9.49  7.70  78.0  20.0  74.3
554   44    1  49.0  27.3  40.2  31.1  13.0   8.91  4.07  81.5  27.6  72.8
443   49    0  34.9  37.9  15.3  19.4   7.1   5.30  5.88  83.0   7.9  62.5
301   65    1  39.1  45.8  23.1  27.5   6.4   7.00  6.23  73.0  27.1  64.3
247   55    1  47.6  71.9  25.8  24.5   5.8   9.24  4.63  83.0  29.1  76.7
..   …  …   …    …    …    …    …    …    …    …    …    …
71    38    1  39.9  62.9  71.7  43.9  10.4  10.90  7.01  99.0  88.3  73.1
106   41    1  44.7  74.9  25.2  20.2   6.3  10.34  4.23  74.0  23.7  72.1
270   59    1  39.8  49.4  25.4  21.4  24.7   7.50  3.69  86.0  18.7  71.9
435   48    0  44.4  52.5  16.4  23.4   4.5   9.06  6.78  74.0  10.3  73.1
102   41    1  42.3  55.9  19.6  18.9  10.9   7.15  3.29  86.0  24.5  76.1

[430 rows x 12 columns]
```

[163]:
```
# Unscaling Data using Inverse Transform
pd.DataFrame(new_pipe_lr['standardscaler'].inverse_transform(X_train_std))
```

[163]:
```
             0         1          2          3          4          5  \
0    63.744790  0.967266  44.538843  83.883075  15.516404  22.554596
1    44.160360  0.967266  48.895353  33.429056  39.411141  31.294202
2    49.056468 -0.047864  35.244955  42.268934  15.706800  18.350734
3    64.724011  0.967266  39.311031  48.857145  23.132256  27.311597
4    54.931797  0.967266  47.539994  70.623258  25.702607  23.992759
..         …        …         …         …         …         …
425  38.285031  0.967266  40.085522  63.117702  69.398561  45.454578
426  41.222696  0.967266  44.732466  73.125111  25.131418  19.235758
427  58.848682  0.967266  39.988711  51.859367  25.321814  20.563293
428  48.077246 -0.047864  44.442032  54.444614  16.753980  22.775852
429  41.222696  0.967266  42.408994  57.280047  19.800321  17.797595

             6         7         8          9         10         11
0    12.184143  9.359019  7.735127  78.395562  20.785988  73.986818
1    12.866991  8.798228  4.132575  82.497775  26.768911  72.665663
2     7.111559  5.307788  5.928889  84.255867  11.260543  63.593736
3     6.428712  6.951485  6.276243  72.535256  26.375298  65.179121
4     5.843413  9.117299  4.688341  84.255867  27.949752  76.100665
..         …        …        …         …         …         …
```

```
425  10.330699  10.722322  7.050345  103.008845  74.553579  72.929894
426   6.331162  10.180868  4.291365   73.707317  23.698727  72.049124
427  24.280303   7.434926  3.755448   87.772050  19.762593  71.872971
428   4.575268   8.943260  6.822084   73.707317  13.149887  72.929894
429  10.818447   7.096518  3.358473   87.772050  24.328508  75.572203

[430 rows x 12 columns]
```

[165]:
```
newdata = pd.DataFrame(new_pipe_lr['standardscaler'].
 ↪inverse_transform(X_train_std),columns=Xfeatures.columns)
```

[ ]:
```
# Creating Your Own Transformer
+ FunctionTransformer
+ TransformerMixin
```

[164]:
```
# FunctionTransformer
from sklearn.preprocessing import FunctionTransformer
```

[166]:
```
newdata.head()
```

[166]:
```
         Age       Sex        ALB        ALP        ALT        AST        BIL  \
0  63.744790  0.967266  44.538843  83.883075  15.516404  22.554596  12.184143
1  44.160360  0.967266  48.895353  33.429056  39.411141  31.294202  12.866991
2  49.056468 -0.047864  35.244955  42.268934  15.706800  18.350734   7.111559
3  64.724011  0.967266  39.311031  48.857145  23.132256  27.311597   6.428712
4  54.931797  0.967266  47.539994  70.623258  25.702607  23.992759   5.843413

        CHE      CHOL       CREA        GGT       PROT
0  9.359019  7.735127  78.395562  20.785988  73.986818
1  8.798228  4.132575  82.497775  26.768911  72.665663
2  5.307788  5.928889  84.255867  11.260543  63.593736
3  6.951485  6.276243  72.535256  26.375298  65.179121
4  9.117299  4.688341  84.255867  27.949752  76.100665
```

[168]:
```
# Round them Every value
rounder = FunctionTransformer(np.round)
```

[169]:
```
newdata_rounded = rounder.transform(newdata)
```

[170]:
```
newdata_rounded.head()
```

[170]:
```
    Age  Sex   ALB   ALP   ALT   AST   BIL  CHE  CHOL  CREA   GGT  PROT
0  64.0  1.0  45.0  84.0  16.0  23.0  12.0  9.0   8.0  78.0  21.0  74.0
1  44.0  1.0  49.0  33.0  39.0  31.0  13.0  9.0   4.0  82.0  27.0  73.0
2  49.0 -0.0  35.0  42.0  16.0  18.0   7.0  5.0   6.0  84.0  11.0  64.0
3  65.0  1.0  39.0  49.0  23.0  27.0   6.0  7.0   6.0  73.0  26.0  65.0
4  55.0  1.0  48.0  71.0  26.0  24.0   6.0  9.0   5.0  84.0  28.0  76.0
```

[171]:
```
dir(rounder)
```

[171]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_check_input',
 '_check_inverse_transform',
 '_check_n_features',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_repr_html_',
 '_repr_html_inner',
 '_repr_mimebundle_',
 '_transform',
 '_validate_data',
 'accept_sparse',
 'check_inverse',
 'fit',
 'fit_transform',
 'func',
 'get_params',
```

```
 'inv_kw_args',
 'inverse_func',
 'inverse_transform',
 'kw_args',
 'set_params',
 'transform',
 'validate']
```

[172]:
```python
# Using Custom Transformer
from sklearn.base import TransformerMixin
```

[173]:
```python
class RounderTransformer(TransformerMixin):
    def fit(self,X,y=None):
        return self

    def transform(self,X):
        # Apply fxn
        rounded_df = np.round(X)
        return rounded_df
```

[174]:
```python
rtf = RounderTransformer()
rtf.transform(newdata)
```

[174]:
```
      Age  Sex   ALB   ALP   ALT   AST   BIL   CHE  CHOL   CREA   GGT  PROT
0    64.0  1.0  45.0  84.0  16.0  23.0  12.0   9.0   8.0   78.0  21.0  74.0
1    44.0  1.0  49.0  33.0  39.0  31.0  13.0   9.0   4.0   82.0  27.0  73.0
2    49.0 -0.0  35.0  42.0  16.0  18.0   7.0   5.0   6.0   84.0  11.0  64.0
3    65.0  1.0  39.0  49.0  23.0  27.0   6.0   7.0   6.0   73.0  26.0  65.0
4    55.0  1.0  48.0  71.0  26.0  24.0   6.0   9.0   5.0   84.0  28.0  76.0
..    …    …     …     …    …     …    …     …    …      …     …    …
425  38.0  1.0  40.0  63.0  69.0  45.0  10.0  11.0   7.0  103.0  75.0  73.0
426  41.0  1.0  45.0  73.0  25.0  19.0   6.0  10.0   4.0   74.0  24.0  72.0
427  59.0  1.0  40.0  52.0  25.0  21.0  24.0   7.0   4.0   88.0  20.0  72.0
428  48.0 -0.0  44.0  54.0  17.0  23.0   5.0   9.0   7.0   74.0  13.0  73.0
429  41.0  1.0  42.0  57.0  20.0  18.0  11.0   7.0   3.0   88.0  24.0  76.0

[430 rows x 12 columns]
```

EDA/ETL Pipeline

- Pandas pipe
  - Changes the dataframe
- Pdpipe
- Sklearn
  - compose column_transformer
  - base.TransformerMixin
  - FunctionTransformer

[145]:
```
df2 = pd.read_csv("data/hcvdata.csv")
```

[101]:
```
df2.head()
```

[101]:
```
   Unnamed: 0      Category  Age Sex   ALB   ALP   ALT   AST   BIL    CHE  \
0           1  0=Blood Donor   32   m  38.5  52.5   7.7  22.1   7.5   6.93
1           2  0=Blood Donor   32   m  38.5  70.3  18.0  24.7   3.9  11.17
2           3  0=Blood Donor   32   m  46.9  74.7  36.2  52.6   6.1   8.84
3           4  0=Blood Donor   32   m  43.2  52.0  30.6  22.6  18.9   7.33
4           5  0=Blood Donor   32   m  39.2  74.1  32.6  24.8   9.6   9.15

   CHOL   CREA   GGT  PROT
0  3.23  106.0  12.1  69.0
1  4.80   74.0  15.6  76.5
2  5.20   86.0  33.2  79.3
3  4.74   80.0  33.8  75.7
4  4.32   76.0  29.9  68.7
```

[102]:
```
df2['Category'].unique()
```

[102]:
```
array(['0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis',
       '2=Fibrosis', '3=Cirrhosis'], dtype=object)
```

[104]:
```
df2['Category'].value_counts()
```

[104]:
```
0=Blood Donor             533
3=Cirrhosis                30
1=Hepatitis                24
2=Fibrosis                 21
0s=suspect Blood Donor      7
Name: Category, dtype: int64
```

[125]:
```
def split_numbers_from_text(df_x):
    df_x['Category'] = df_x['Category'].str.split("=").str.get(0).str.
 ↪replace("s",'')
    return df_x
```

[126]:
```
ndf = split_numbers_from_text(df2)
```

[127]:
```
ndf.head()
```

[127]:
```
   Unnamed: 0 Category  Age Sex   ALB   ALP   ALT   AST   BIL    CHE  CHOL  \
0           1        0   32   m  38.5  52.5   7.7  22.1   7.5   6.93  3.23
1           2        0   32   m  38.5  70.3  18.0  24.7   3.9  11.17  4.80
2           3        0   32   m  46.9  74.7  36.2  52.6   6.1   8.84  5.20
3           4        0   32   m  43.2  52.0  30.6  22.6  18.9   7.33  4.74
4           5        0   32   m  39.2  74.1  32.6  24.8   9.6   9.15  4.32
```

	CREA	GGT	PROT
0	106.0	12.1	69.0
1	74.0	15.6	76.5
2	86.0	33.2	79.3
3	80.0	33.8	75.7
4	76.0	29.9	68.7

[135]:
```
# Group 1,2,3 as one class
def group_as_hep(x):
    if int(x) > 0:
        return 1
    else:
        return 0
```

[136]:
```
group_as_hep(3)
```

[136]: 1

[137]:
```
# Map Cate
def group_disease(df_x):
    df_x['Target'] = df_x['Category'].apply(lambda x:group_as_hep(x))
    return df_x
```

[146]:
```
# Using Pipe Let us join them
# df.pipe(fxn)
newdf = df2.pipe(split_numbers_from_text)
```

[147]:
```
newdf.head()
```

[147]:

	Unnamed: 0	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	\
0	1	0	32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	
1	2	0	32	m	38.5	70.3	18.0	24.7	3.9	11.17	4.80	
2	3	0	32	m	46.9	74.7	36.2	52.6	6.1	8.84	5.20	
3	4	0	32	m	43.2	52.0	30.6	22.6	18.9	7.33	4.74	
4	5	0	32	m	39.2	74.1	32.6	24.8	9.6	9.15	4.32	

	CREA	GGT	PROT
0	106.0	12.1	69.0
1	74.0	15.6	76.5
2	86.0	33.2	79.3
3	80.0	33.8	75.7
4	76.0	29.9	68.7

[148]:
```
# Original is changed
df2.head()
```

[148]:
```
   Unnamed: 0  Category  Age Sex   ALB   ALP   ALT   AST   BIL    CHE  CHOL  \
0           1         0   32   m  38.5  52.5   7.7  22.1   7.5   6.93  3.23
1           2         0   32   m  38.5  70.3  18.0  24.7   3.9  11.17  4.80
2           3         0   32   m  46.9  74.7  36.2  52.6   6.1   8.84  5.20
3           4         0   32   m  43.2  52.0  30.6  22.6  18.9   7.33  4.74
4           5         0   32   m  39.2  74.1  32.6  24.8   9.6   9.15  4.32

    CREA   GGT  PROT
0  106.0  12.1  69.0
1   74.0  15.6  76.5
2   86.0  33.2  79.3
3   80.0  33.8  75.7
4   76.0  29.9  68.7
```

[149]:
```python
df2.pipe(split_numbers_from_text).pipe(group_disease)
```

[149]:
```
     Unnamed: 0  Category  Age Sex   ALB    ALP    ALT    AST   BIL    CHE  \
0             1         0   32   m  38.5   52.5    7.7   22.1   7.5   6.93
1             2         0   32   m  38.5   70.3   18.0   24.7   3.9  11.17
2             3         0   32   m  46.9   74.7   36.2   52.6   6.1   8.84
3             4         0   32   m  43.2   52.0   30.6   22.6  18.9   7.33
4             5         0   32   m  39.2   74.1   32.6   24.8   9.6   9.15
..          …         …   …  ..    …      …      …      …     …      …
610         611         3   62   f  32.0  416.6    5.9  110.3  50.0   5.57
611         612         3   64   f  24.0  102.8    2.9   44.4  20.0   1.54
612         613         3   64   f  29.0   87.3    3.5   99.0  48.0   1.66
613         614         3   46   f  33.0    NaN   39.0   62.0  20.0   3.56
614         615         3   59   f  36.0    NaN  100.0   80.0  12.0   9.07

     CHOL   CREA    GGT  PROT  Target
0    3.23  106.0   12.1  69.0       0
1    4.80   74.0   15.6  76.5       0
2    5.20   86.0   33.2  79.3       0
3    4.74   80.0   33.8  75.7       0
4    4.32   76.0   29.9  68.7       0
..    …      …      …     …       …
610  6.30   55.7  650.9  68.5       1
611  3.02   63.0   35.9  71.3       1
612  3.63   66.7   64.2  82.0       1
613  4.20   52.0   50.0  71.0       1
614  5.30   67.0   34.0  68.0       1

[615 rows x 15 columns]
```

[142]:
```python
# Mapping Gender
def map_gender(df_x):
    gender_dict = {"f":0,"m":1}
```

```python
    df_x['Sex'] = df_x['Sex'].map(gender_dict)
    return df_x
```

[150]:

```python
df2.pipe(map_gender)
```

[150]:

```
     Unnamed: 0 Category  Age  Sex   ALB    ALP    ALT    AST   BIL    CHE  \
0             1        0   32    1  38.5   52.5    7.7   22.1   7.5   6.93
1             2        0   32    1  38.5   70.3   18.0   24.7   3.9  11.17
2             3        0   32    1  46.9   74.7   36.2   52.6   6.1   8.84
3             4        0   32    1  43.2   52.0   30.6   22.6  18.9   7.33
4             5        0   32    1  39.2   74.1   32.6   24.8   9.6   9.15
..          …        …   …   …    …     …     …     …    …     …
610         611        3   62    0  32.0  416.6    5.9  110.3  50.0   5.57
611         612        3   64    0  24.0  102.8    2.9   44.4  20.0   1.54
612         613        3   64    0  29.0   87.3    3.5   99.0  48.0   1.66
613         614        3   46    0  33.0    NaN   39.0   62.0  20.0   3.56
614         615        3   59    0  36.0    NaN  100.0   80.0  12.0   9.07

     CHOL   CREA    GGT  PROT  Target
0    3.23  106.0   12.1  69.0       0
1    4.80   74.0   15.6  76.5       0
2    5.20   86.0   33.2  79.3       0
3    4.74   80.0   33.8  75.7       0
4    4.32   76.0   29.9  68.7       0
..    …     …     …    …     …
610  6.30   55.7  650.9  68.5       1
611  3.02   63.0   35.9  71.3       1
612  3.63   66.7   64.2  82.0       1
613  4.20   52.0   50.0  71.0       1
614  5.30   67.0   34.0  68.0       1

[615 rows x 15 columns]
```

Using Pdpipe

Using Pdpipe

- AdHocStage - Define custom pipeline stages on the fly.
- ColDrop - Drop columns by name.
- ValDrop - Drop rows by by their value in specific or all columns.
- ValKeep - Keep rows by by their value in specific or all columns.
- ColRename - Rename columns.
- DropNa - Drop null values. Supports all parameter supported by pandas.dropna function.
- FreqDrop - Drop rows by value frequency threshold on a specific column.
- ColReorder - Reorder columns.
- RowDrop - Drop rows by callable conditions.
- Schematize - Learn a dataframe schema on fit and transform to it on future transforms.

- DropDuplicates - Drop duplicate values in a subset of columns.

[157]:
```
import pdpipe as pdp
```

[159]:
```
# Method
dir(pdp)
```

[159]:
```
['AdHocStage',
 'AggByCols',
 'ApplyByCols',
 'ApplyToRows',
 'Bin',
 'ColByFrameFunc',
 'ColDrop',
 'ColRename',
 'ColReorder',
 'DropDuplicates',
 'DropNa',
 'DropRareTokens',
 'DropTokensByLength',
 'DropTokensByList',
 'Encode',
 'FitOnly',
 'FreqDrop',
 'Log',
 'MapColVals',
 'OneHotEncode',
 'PdPipeline',
 'PdPipelineStage',
 'RegexReplace',
 'RemoveStopwords',
 'RowDrop',
 'Scale',
 'Schematize',
 'SnowballStem',
 'TfidfVectorizeTokenLists',
 'TokenizeText',
 'UntokenizeText',
 'ValDrop',
 'ValKeep',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
```

```
 '__path__',
 '__pdoc__',
 '__spec__',
 '__version__',
 'cond',
 'cq',
 'exceptions',
 'make_pdpipeline',
 'nltk_stages',
 'text_stages',
 'wrappers']
```

```
[ ]:
```