CA_5

May 7, 2024

1 CA5

1.0.1 Imports

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪RandomizedSearchCV
from sklearn.cross_decomposition import PLSRegression
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error
```

1.0.2 Reading data

[ ]:
```
train_df = pd.DataFrame(pd.read_csv("train.csv"))
test_df = pd.DataFrame(pd.read_csv("test.csv"))

train_df[:10], train_df.size, test_df.size
```

[ ]:
```
(   Length (cm)  Width (cm)  Weight (g)  Pericarp Thickness (mm)  Seed Count  \
 0        17.37        5.42       94.30                     4.90      193.93
 1        27.78        4.75      262.71                     6.56      186.29
 2         6.17        3.51       66.72                     7.96      298.81
 3         6.12        6.07       51.24                     4.57       39.36
 4        28.58        4.84      166.51                     3.07      194.07
 5        13.80        8.64      189.24                     4.32       72.27
 6         9.15        8.40       74.69                     1.70        8.45
 7        12.40       10.68      159.72                    10.28      239.54
 8        18.90        7.66       57.72                     4.63       87.10
 9        35.57        8.82      633.31                     6.92      244.27
```

	Capsaicin Content	Vitamin C Content (mg)	Sugar Content	Moisture Content
0	3.21	173.59	6.15	88.59
1	8.19	100.41	2.36	111.20
2	4.69	125.91	6.75	72.98
3	2.76	143.54	5.93	63.93
4	7.01	193.76	2.85	88.19
5	2.38	140.15	4.60	117.75
6	6.22	68.89	4.02	101.63
7	11.08	28.62	4.32	94.90
8	6.74	34.05	0.65	82.13
9	0.84	312.63	0.84	84.36

	Firmness	color	Harvest Time
0	3.40	red	Midday
1	5.45	green	Midday
2	2.77	red	Midday
3	1.62	yellow	Midday
4	3.99	red	Midday
5	3.21	yellow	Morning
6	5.54	yellow	Midday
7	5.56	green	Midday
8	4.32	yellow	Midday
9	2.63	red	Morning

	Average Daily Temperature During Growth (celcius)
0	8.68
1	22.44
2	24.99
3	13.05
4	27.08
5	24.95
6	31.34
7	18.53
8	16.14
9	21.27

	Average Temperature During Storage (celcius)	Scoville Heat Units (SHU)
0	5-6	0.00
1	NaN	0.00
2	NaN	455995.06
3	NaN	0.00
4	NaN	0.00
5	NaN	0.00
6	NaN	70571.10
7	NaN	0.00
8	NaN	31362.49

```
9                                             NaN                      0.00   ,
15000,
11200)
```

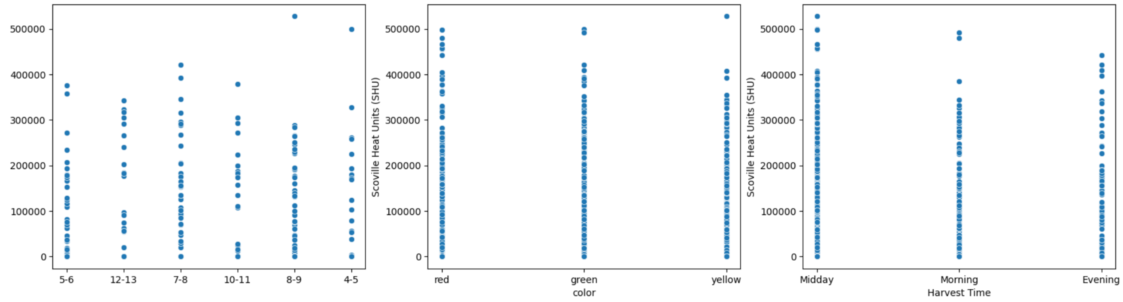
[ ]:
```python
train_df.dtypes
```

[ ]:
```
Length (cm)                                          float64
Width (cm)                                           float64
Weight (g)                                           float64
Pericarp Thickness (mm)                              float64
Seed Count                                           float64
Capsaicin Content                                    float64
Vitamin C Content (mg)                               float64
Sugar Content                                        float64
Moisture Content                                     float64
Firmness                                             float64
color                                                 object
Harvest Time                                          object
Average Daily Temperature During Growth (celcius)    float64
Average Temperature During Storage (celcius)          object
Scoville Heat Units (SHU)                            float64
dtype: object
```

I see that there are three columns with categorical values - these need to be altered to numerical values for the classifier to be able to calculate results. I investigate these columns further:

1.0.3 Data cleaning

[ ]:
```python
fig, ax = plt.subplots(1, 3, figsize=(20, 5))
sns.scatterplot(x=train_df["Average Temperature During Storage (celcius)"].
 ↪values, y=train_df["Scoville Heat Units (SHU)"].values, ax=ax[0])
sns.scatterplot(x=train_df["color"], y=train_df["Scoville Heat Units (SHU)"],␣
 ↪ax=ax[1])
sns.scatterplot(x=train_df["Harvest Time"], y=train_df["Scoville Heat Units␣
 ↪(SHU)"], ax=ax[2])
plt.show()
```



[ ]:
```python
"""
Since the categorical features show so little correlation with the target␣
↪variable, I decide to remove the columns "Average Temperature During
Storage (celsius)" and "Harvest Time". Logically, it seems unlikely that␣
↪storage temperature and harvest time should have an impact on spice.
Moreover, the plots show no direct correlation between these variables and the␣
↪target variable, and the storage temperature column has very many NaN-values.
Alternatively, I could have altered the values to numerical, e.g. by setting␣
↪the value of temperature to the mean (5.5 for 5-6) and giving the
time-of-day variable index labels - if they had proven themselves useful.␣
↪However, I decide to keep the colour-column for now, as it could
be somehow descriptive in combination with other physical features. I'll then␣
↪assign index labels (0, 1, 2) to the colours.
"""

columns_to_drop = ["Average Temperature During Storage (celcius)", "Harvest␣
↪Time"]
train_df = train_df.drop(columns_to_drop, axis=1) # I'm dropping columns here,␣
↪not rows!
test_df = test_df.drop(columns_to_drop, axis=1)

colour_mapping = {"red": 0, "green": 1, "yellow": 2}
train_df["color"] = train_df["color"].replace(colour_mapping)
test_df["color"] = test_df["color"].replace(colour_mapping)
print(train_df.dtypes)

train_df["Scoville Heat Units (SHU)"].max()
```

```
Length (cm)                                          float64
Width (cm)                                           float64
Weight (g)                                           float64
Pericarp Thickness (mm)                              float64
Seed Count                                           float64
Capsaicin Content                                    float64
Vitamin C Content (mg)                               float64
Sugar Content                                        float64
Moisture Content                                     float64
Firmness                                             float64
color                                                float64
Average Daily Temperature During Growth (celcius)    float64
Scoville Heat Units (SHU)                            float64
dtype: object
```

```
C:\Users\kroel\AppData\Local\Temp\ipykernel_16904\178743181.py:15:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
```

```
  train_df["color"] = train_df["color"].replace(colour_mapping)
C:\Users\kroel\AppData\Local\Temp\ipykernel_16904\178743181.py:16:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  test_df["color"] = test_df["color"].replace(colour_mapping)
```

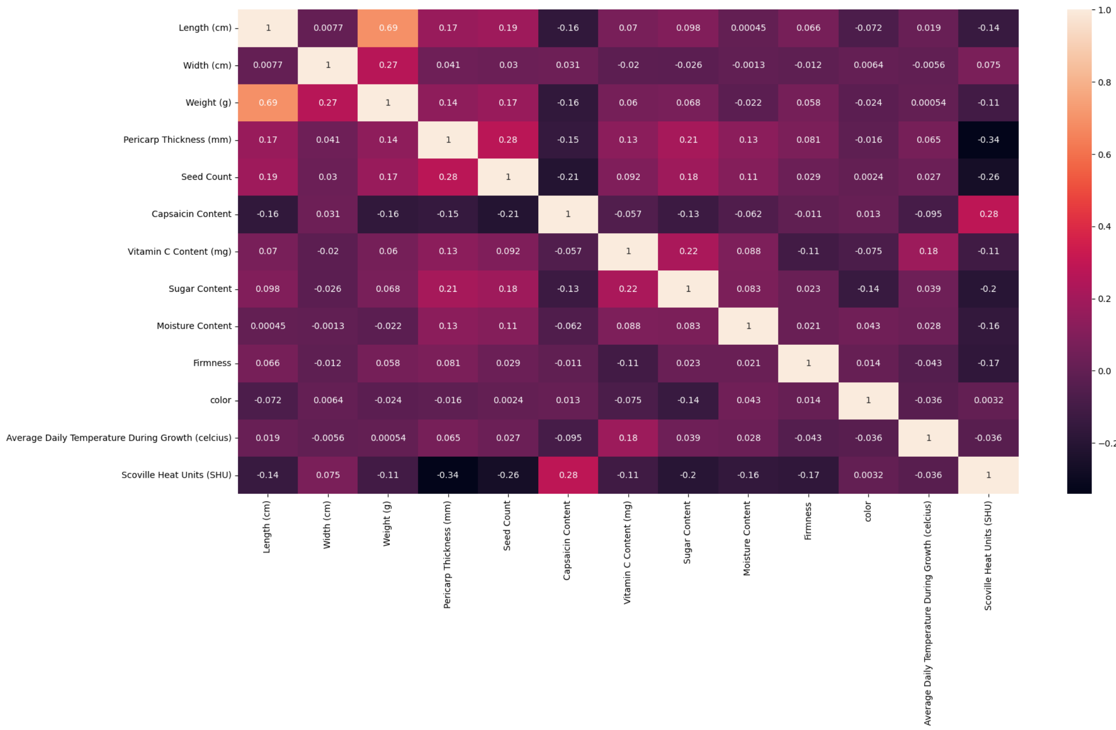
[ ]:
```
527639.86
```

[ ]:
```
train_df.isnull().sum(), test_df.isnull().sum()
```

[ ]:
```
(Length (cm)                                         1
 Width (cm)                                          1
 Weight (g)                                          1
 Pericarp Thickness (mm)                             2
 Seed Count                                          1
 Capsaicin Content                                   1
 Vitamin C Content (mg)                              0
 Sugar Content                                       1
 Moisture Content                                    0
 Firmness                                            1
 color                                               1
 Average Daily Temperature During Growth (celcius)   0
 Scoville Heat Units (SHU)                           0
 dtype: int64,
 Length (cm)                                         2
 Width (cm)                                          0
 Weight (g)                                          0
 Pericarp Thickness (mm)                             0
 Seed Count                                          0
 Capsaicin Content                                   1
 Vitamin C Content (mg)                              0
 Sugar Content                                       0
 Moisture Content                                    1
 Firmness                                            2
 color                                               0
 Average Daily Temperature During Growth (celcius)   0
 dtype: int64)
```

1.0.4 Data preprocessing and visualisation

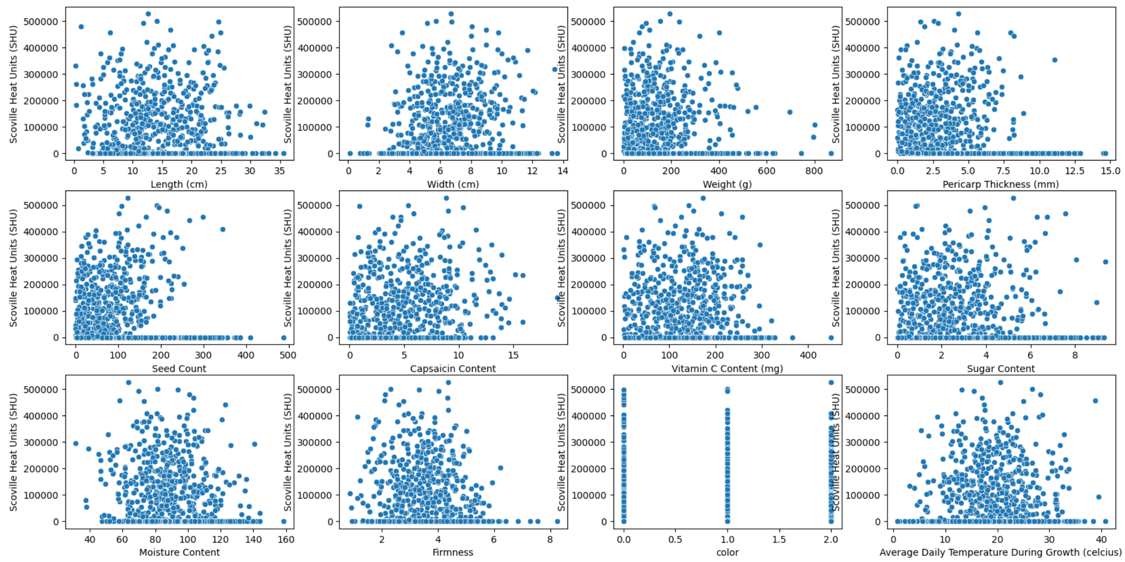
[ ]:
```
# Linear correlation map
plt.figure(figsize=(20, 10))
sns.heatmap(train_df.corr(), annot=True)
plt.show()
```



The linear correlation plot shows no significant linear correlation between any features of interest. I move on by plotting scatterplots of the target column together with the other columns to check for other kinds of correlation.

[ ]:
```python
fig, ax = plt.subplots(3, 4, figsize=(20, 10))
plot_row = 0
plot_column = 0
for feature in train_df.columns.drop("Scoville Heat Units (SHU)"):
    if feature == "Scoville Heat Units (SHU)":
        pass
    sns.scatterplot(x=train_df[feature], y=train_df["Scoville Heat Units (SHU)"], ax=ax[plot_row, plot_column])
    if plot_column == 3:
        plot_column = 0
        plot_row += 1
    else:
        plot_column += 1
plt.show()

train_df.columns
```



[ ]:
```
Index(['Length (cm)', 'Width (cm)', 'Weight (g)', 'Pericarp Thickness (mm)',
       'Seed Count', 'Capsaicin Content', 'Vitamin C Content (mg)',
       'Sugar Content', 'Moisture Content', 'Firmness', 'color',
       'Average Daily Temperature During Growth (celcius)',
       'Scoville Heat Units (SHU)'],
      dtype='object')
```

1.0.5 Comments on the visualisation

There seem to be no features here that are immediately revealing as to where the peppers spice is located on the Scoville scale, so I cannot draw any immediate conclusions on what further simplifications or alterations may be done to the dataset.

1.0.6 Train / dev split

[ ]:
```python
"""
I try creating a dataset containing only target values greater than 0 for the␣
↪logistic regression model to
train on for Task B. The idea is that this way, the models won't classify␣
↪samples as 0 at all.
The backside is, of course, that the training set without 0s is smaller than␣
↪half the size of the initial training set. We'll see!
"""

def binary_mapping(value):
    return 1 if value > 0 else 0

y = train_df["Scoville Heat Units (SHU)"].values
```

```
y_binary = np.array([binary_mapping(value) for value in y]) # Total binary y␣
↪for the very end
X = train_df.drop("Scoville Heat Units (SHU)", axis=1).values

train_df_gt0 = train_df[train_df["Scoville Heat Units (SHU)"] != 0]

y_gt0 = train_df_gt0["Scoville Heat Units (SHU)"].values
X_gt0 = train_df_gt0.drop("Scoville Heat Units (SHU)", axis=1).values

X_train, X_dev, y_train, y_dev = train_test_split(X, y, train_size=0.7,␣
↪random_state=42)
X_train_gt0, X_dev_gt0, y_train_gt0, y_dev_gt0 = train_test_split(X_gt0, y_gt0,␣
↪train_size=0.7, random_state=42)

y_train_binary = np.array([binary_mapping(value) for value in y_train])
y_dev_binary = np.array([binary_mapping(value) for value in y_dev])

X.size, X_gt0.size
```

[ ]: (12000, 5508)

1.0.7 Modelling

[ ]:
```
"""
This time I'm testing PLS and Linear regession for Task A and a RandomForest␣
↪for task B (in combination with one of the other two).
"""

# Regression analysis
pls = Pipeline([
    ('imputer', SimpleImputer()),
    ('pls', PLSRegression())
])

# Simple linear regression
lr = Pipeline([
    ('imputer', SimpleImputer()),
    ('lr', LinearRegression(n_jobs=-1))
])

# Ensemble classifier
rf = Pipeline([
    ('imputer', SimpleImputer()),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=.9)),
    ('rf', RandomForestClassifier(random_state=42, n_jobs=-1)),
])
```

```python
print(f"PLS parameters: {pls.get_params()}")
print(f"Randomforest parameters: {rf.get_params()}")
```

```
PLS parameters: {'memory': None, 'steps': [('imputer', SimpleImputer()), ('pls',
PLSRegression())], 'verbose': False, 'imputer': SimpleImputer(), 'pls':
PLSRegression(), 'imputer__add_indicator': False, 'imputer__copy': True,
'imputer__fill_value': None, 'imputer__keep_empty_features': False,
'imputer__missing_values': nan, 'imputer__strategy': 'mean', 'pls__copy': True,
'pls__max_iter': 500, 'pls__n_components': 2, 'pls__scale': True, 'pls__tol':
1e-06}
Randomforest parameters: {'memory': None, 'steps': [('imputer',
SimpleImputer()), ('scaler', StandardScaler()), ('pca', PCA(n_components=0.9)),
('rf', RandomForestClassifier(n_jobs=-1, random_state=42))], 'verbose': False,
'imputer': SimpleImputer(), 'scaler': StandardScaler(), 'pca':
PCA(n_components=0.9), 'rf': RandomForestClassifier(n_jobs=-1, random_state=42),
'imputer__add_indicator': False, 'imputer__copy': True, 'imputer__fill_value':
None, 'imputer__keep_empty_features': False, 'imputer__missing_values': nan,
'imputer__strategy': 'mean', 'scaler__copy': True, 'scaler__with_mean': True,
'scaler__with_std': True, 'pca__copy': True, 'pca__iterated_power': 'auto',
'pca__n_components': 0.9, 'pca__n_oversamples': 10,
'pca__power_iteration_normalizer': 'auto', 'pca__random_state': None,
'pca__svd_solver': 'auto', 'pca__tol': 0.0, 'pca__whiten': False,
'rf__bootstrap': True, 'rf__ccp_alpha': 0.0, 'rf__class_weight': None,
'rf__criterion': 'gini', 'rf__max_depth': None, 'rf__max_features': 'sqrt',
'rf__max_leaf_nodes': None, 'rf__max_samples': None,
'rf__min_impurity_decrease': 0.0, 'rf__min_samples_leaf': 1,
'rf__min_samples_split': 2, 'rf__min_weight_fraction_leaf': 0.0,
'rf__n_estimators': 100, 'rf__n_jobs': -1, 'rf__oob_score': False,
'rf__random_state': 42, 'rf__verbose': 0, 'rf__warm_start': False}
```

[ ]:
```python
# Setting up ranges for RandomizedSearch and GridSearch with cross-validation
components = np.arange(1, 10)

estimators = np.arange(1, 1300)
depths = np.arange(1, 15)
learning_rates = np.linspace(0, 1, 10)

pls_grid = {"pls__n_components": components}

rf_grid = {"rf__n_estimators": estimators, "rf__max_depth": depths}
```

[ ]:
```python
"""
Setting up the GridSearches and RandomizedSearches - I use RandomizedSearch on␣
↪models that use more or less continuous hyperparameter-ranges,
and GridSearch on the ones whose hyperparameters are tuned with discrete values.
```

```python
I use f1-score on the RandomForest, 'cause it's the binary step of Task B.
"""
rf_search = RandomizedSearchCV(estimator=rf,
                               param_distributions=rf_grid,
                               scoring='f1',
                               cv=10,
                               n_jobs=-1)

pls_search = GridSearchCV(estimator=pls,
                          param_grid=pls_grid,
                          scoring='neg_mean_absolute_error',
                          cv=10,
                          n_jobs=-1,
                          verbose=2)

lr_search = GridSearchCV(estimator=lr,
                        param_grid={}, # The linear regression has no grid, but␣
 ↪I sent it through the grid search for the sake of CV
                        scoring='neg_mean_absolute_error',
                        cv=10,
                        n_jobs=-1,
                        verbose=2)
```

[ ]:
```python
# RandomForest:
rf_search.fit(X_train, y_train_binary)
print(f'Best parameters for RandomForest: {rf_search.best_params_}')
print(f'Best score for RandomForest: {abs(rf_search.best_score_)}')

# PLS for y > 0:
pls_search.fit(X_train_gt0, y_train_gt0)
print(f'Best parameters for PLS y > 0: {pls_search.best_params_}')
print(f'Best score for PLS: {abs(pls_search.best_score_)}')

# PLS else:
pls_search.fit(X_train, y_train)
print(f'Best parameters for PLS: {pls_search.best_params_}')
print(f'Best score for PLS: {abs(pls_search.best_score_)}')

# Linear Regression for y > 0:
lr_search.fit(X_train_gt0, y_train_gt0)
print(f'Best parameters for LinReg y > 0: {lr_search.best_params_}')
print(f'Best score for LinReg: {abs(lr_search.best_score_)}')

# Linear Regression else:
lr_search.fit(X_train, y_train)
print(f'Best parameters for LinReg: {lr_search.best_params_}')
print(f'Best score for LinReg: {abs(lr_search.best_score_)}')
```

```
Best parameters for RandomForest: {'rf__n_estimators': 1240, 'rf__max_depth':
10}
Best score for RandomForest: 0.8536965826293821
Fitting 10 folds for each of 9 candidates, totalling 90 fits
Best parameters for PLS y > 0: {'pls__n_components': 3}
Best score for PLS: 79366.78664584653
Fitting 10 folds for each of 9 candidates, totalling 90 fits
Best parameters for PLS: {'pls__n_components': 2}
Best score for PLS: 69586.33778713846
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Best parameters for LinReg y > 0: {}
Best score for LinReg: 79444.3869130244
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Best parameters for LinReg: {}
Best score for LinReg: 69754.00576327537
```

1.0.8 Final evaluation

[ ]:
```
# Finally, I choose linear regression as the linear model, since it acchieved a␣
↪slightly higher score than PLS.
best_rf = rf_search.best_estimator_
lr = lr_search.best_estimator_

# Regular linear regression (Task A):
y_pred_linear = lr.predict(X_dev)

# Binary classification with RandomForest and linear classification with␣
↪regular linear regression (Task B):
y_pred_ensemble = best_rf.predict(X_dev) # These are now binary
indices_where_1 = np.where(y_pred_ensemble == 1)[0]
cont_dev = X_dev[indices_where_1]
cont_pred = lr.predict(cont_dev)

y_pred_ensemble[indices_where_1] = cont_pred

print(f"Dev mean absolute error linear: {mean_absolute_error(y_dev,␣
↪y_pred_linear)}")
print(f"Dev mean absolute error ensemble: {mean_absolute_error(y_dev,␣
↪y_pred_ensemble)}")
```

```
Dev mean absolute error linear: 66468.64899771604
Dev mean absolute error ensemble: 40234.983733333334
```

1.0.9 Kaggle submission

```
[ ]: """
I don't know if there's a more sufficient way to train and fit this␣
 ↪self-constructed ensemble, but I just left it this way.
In any case, this newest classifier beat my former score by quite a lot! I hope␣
 ↪it's okay
"""
best_rf.fit(X, y_binary)
X_test = test_df.values
y_pred = best_rf.predict(X_test)
indices_where_1 = np.where(y_pred == 1)[0]
cont_test = X_test[indices_where_1]
cont_pred = lr.predict(cont_test)

y_pred[indices_where_1] = cont_pred

submission_df = pd.DataFrame(y_pred, columns=['Scoville Heat Units (SHU)'])
submission_df.to_csv("Ensemble.csv", index_label="index")
```