```
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
from sklearn.utils import check_X_y
```

Custom Estimator

```
class MostFrequentClassClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.most_frequent_ = None

    def fit(self, X, y):

        # Validate input X and target vector y
        X, y = check_X_y(X, y)

        # Ensure y is 1D
        y = np.ravel(y)

        # Manually compute the most frequent class
        unique_classes, counts = np.unique(y, return_counts=True)
        self.most_frequent_ = unique_classes[np.argmax(counts)]

        return self

    def predict(self, X):
        if self.most_frequent_ is None:
            raise ValueError("This classifier instance is not fitted
yet.")
        # Predict the most frequent class for each input sample
        return np.full(shape=(X.shape[0],),
fill_value=self.most_frequent_)

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Load data
iris = load_iris()
X, y = iris.data, iris.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)

# Initialize and fit the custom estimator
classifier = MostFrequentClassClassifier()
classifier.fit(X_train, y_train)

# Make predictions
#predictions = classifier.predict(X_test)
```

```
# Evaluate the custom estimator
print(f"Predicted class for all test instances: {predictions[0]}")
```

```
Predicted class for all test instances: 1
```

```
classifier.most_frequent_
```

```
1
```

```
from sklearn.model_selection import cross_val_score
```

```
cross_val_score(classifier, X_train, y_train)
```

```
array([0.34782609, 0.34782609, 0.31818182, 0.36363636, 0.36363636])
```

Scoing function

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.metrics import accuracy_score
import numpy as np

class MostFrequentClassClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.most_frequent_ = None

    def fit(self, X, y):
        # Ensure y is 1D
        y = np.ravel(y)

        # Compute the most frequent class
        unique_classes, counts = np.unique(y, return_counts=True)
        self.most_frequent_ = unique_classes[np.argmax(counts)]
        return self

    def predict(self, X):
        if self.most_frequent_ is None:
            raise ValueError("This classifier instance is not fitted
yet.")
        # Predict the most frequent class for each input sample
        return np.full(shape=(X.shape[0],),
fill_value=self.most_frequent_)

    def score(self, X, y):
        """Return the mean accuracy on the given test data and
labels."""
        # Ensure y is 1D
        y = np.ravel(y)

        # Generate predictions
        predictions = self.predict(X)
```

```python
        # Calculate and return the accuracy
        return accuracy_score(y, predictions)

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Load a dataset
iris = load_iris()
X, y = iris.data, iris.target

# Simplify to a binary classification problem
is_class_0_or_1 = y < 2
X_bin = X[is_class_0_or_1]
y_bin = y[is_class_0_or_1]

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_bin, y_bin,
test_size=0.2, random_state=42)

# Initialize and fit the custom classifier
classifier = MostFrequentClassClassifier()
classifier.fit(X_train, y_train)

# Evaluate the classifier using the score method
score = classifier.score(X_test, y_test)
print(f"Accuracy of the MostFrequentClassClassifier: {score}")
```

```
Accuracy of the MostFrequentClassClassifier: 0.4
```

Transformers

```python
from sklearn.datasets import make_regression
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Generate some data
X, y = make_regression(n_samples=100, n_features=2, noise=0.1,
random_state=42)

# Use the transformer directly
X_transformed = StandardScaler().fit_transform(X)


LinearRegression().fit(X_transformed, y)
```

```
LinearRegression()
```

Custom Transformer using Function Transformer

```
import numpy as np

def cube(x):

    return np.power(x,3)

from sklearn.preprocessing import FunctionTransformer

# Create the custom transformer
cube_transformer = FunctionTransformer(cube)

from sklearn.datasets import make_regression
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Generate some data
X, y = make_regression(n_samples=100, n_features=2, noise=0.1,
random_state=42)

# Use the transformer directly
X_transformed = cube_transformer.transform(X)

LinearRegression().fit(X_transformed, y)
```

```
LinearRegression()
```

Custom Transformer using BaseEstimator and TransformerMixin

```
from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np

class MedianIQRScaler(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.medians_ = None
        self.iqr_ = None

    def fit(self, X, y=None):
        # Calculate medians and interquartile range for each feature
        self.medians_ = np.median(X, axis=0)
        Q1 = np.percentile(X, 25, axis=0)
        Q3 = np.percentile(X, 75, axis=0)
        self.iqr_ = Q3 - Q1

        # Handle case where IQR is 0 to avoid division by zero during
transform
        self.iqr_[self.iqr_ == 0] = 1
        return self

    def transform(self, X):
        # Check if fit has been called
        if self.medians_ is None or self.iqr_ is None:
            raise RuntimeError("The transformer has not been fitted
yet.")

        # Scale features using median and IQR learned during fit
        return (X - self.medians_) / self.iqr_

from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=100, n_features=2, centers=3,
random_state=42)

# Initialize the transformer
scaler = MedianIQRScaler()

# Fit the scaler to the data
scaler.fit(X)

# Transform the data
X_scaled = scaler.transform(X)

# Check the first few rows of the transformed data
print("Transformed data (first 5 rows):")
print(X_scaled[:5])
```

```
Transformed data (first 5 rows):
[[-0.49872679 -0.71613207]
 [ 0.78423675 -0.08192868]
 [-0.03656645  0.52987512]
 [ 0.84159877 -0.09379661]
 [-0.3814692  -0.57206564]]
```

Column Transformer

```
import pandas as pd

# Define the data with numeric labels for sentiment
data = {
    "Social Media Platform": ["Twitter", "Facebook", "Instagram",
"Twitter", "Facebook",
                              "Instagram", "Twitter", "Facebook",
"Instagram", "Twitter"],
    "Review": ["Love the new update!", "Too many ads now", "Great for
sharing photos",
               "Newsfeed algorithm is biased", "Privacy concerns with
latest update",
               "Amazing filters!", "Too much spam", "Easy to connect
with friends",
               "Stories feature is fantastic", "Customer support
lacking"],
    "age": [21, 19, np.nan, 17, 24, np.nan, 30, 19, 16, 31],
    "Sentiment": [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]  # Numeric labels: 1
for Positive, 0 for Negative
}

# Create a DataFrame
df = pd.DataFrame(data)

print(df)
```

```
  Social Media Platform                               Review   age
Sentiment
0               Twitter                 Love the new update!  21.0
1
1              Facebook                     Too many ads now  19.0
0
2             Instagram             Great for sharing photos   NaN
1
3               Twitter         Newsfeed algorithm is biased  17.0
0
4              Facebook  Privacy concerns with latest update  24.0
0
5             Instagram                     Amazing filters!   NaN
1
6               Twitter                        Too much spam  30.0
0
7              Facebook         Easy to connect with friends  19.0
1
8             Instagram         Stories feature is fantastic  16.0
1
9               Twitter             Customer support lacking  31.0
0
```

```
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

# Define the column transformer
column_transformer = ColumnTransformer(
    transformers=[
        ('platform_ohe', OneHotEncoder(), ['Social Media Platform']),
        ('review_bow', CountVectorizer(), 'Review'),
        ('age_impute', SimpleImputer(),['age'])
    ],
    remainder='drop'  # Drop other columns not specified in
transformers
)
```

```
pd.DataFrame(column_transformer.fit_transform(df).toarray(),columns=co
lumn_transformer.get_feature_names_out())
```

```
   platform_ohe__Social Media Platform_Facebook  \
0                                           0.0   
1                                           1.0   
2                                           0.0   
3                                           0.0   
4                                           1.0   
5                                           0.0   
6                                           0.0   
7                                           1.0   
8                                           0.0   
9                                           0.0   

   platform_ohe__Social Media Platform_Instagram  \
0                                            0.0   
1                                            0.0   
2                                            1.0   
3                                            0.0   
4                                            0.0   
5                                            1.0   
6                                            0.0   
7                                            0.0   
8                                            1.0   
9                                            0.0   

   platform_ohe__Social Media Platform_Twitter  review_bow__ads  \
0                                          1.0              0.0   
1                                          0.0              1.0   
2                                          0.0              0.0   
3                                          1.0              0.0   
4                                          0.0              0.0   
5                                          0.0              0.0   
6                                          1.0              0.0   
7                                          0.0              0.0   
8                                          0.0              0.0   
9                                          1.0              0.0   

   review_bow__algorithm  review_bow__amazing  review_bow__biased  \
0                    0.0                  0.0                 0.0   
1                    0.0                  0.0                 0.0   
2                    0.0                  0.0                 0.0   
3                    1.0                  0.0                 1.0   
4                    0.0                  0.0                 0.0   
5                    0.0                  1.0                 0.0   
6                    0.0                  0.0                 0.0   
7                    0.0                  0.0                 0.0   
8                    0.0                  0.0                 0.0   
9                    0.0                  0.0                 0.0   
```

```
   review_bow__concerns  review_bow__connect
review_bow__customer  ...  \
0                    0.0                  0.0
0.0  ...
1                    0.0                  0.0
0.0  ...
2                    0.0                  0.0
0.0  ...
3                    0.0                  0.0
0.0  ...
4                    1.0                  0.0
0.0  ...
5                    0.0                  0.0
0.0  ...
6                    0.0                  0.0
0.0  ...
7                    0.0                  1.0
0.0  ...
8                    0.0                  0.0
0.0  ...
9                    0.0                  0.0
1.0  ...

   review_bow__sharing  review_bow__spam  review_bow__stories  \
0                   0.0               0.0                  0.0
1                   0.0               0.0                  0.0
2                   1.0               0.0                  0.0
3                   0.0               0.0                  0.0
4                   0.0               0.0                  0.0
5                   0.0               0.0                  0.0
6                   0.0               1.0                  0.0
7                   0.0               0.0                  0.0
8                   0.0               0.0                  1.0
9                   0.0               0.0                  0.0

   review_bow__support  review_bow__the  review_bow__to
review_bow__too  \
0                   0.0              1.0             0.0
0.0
1                   0.0              0.0             0.0
1.0
2                   0.0              0.0             0.0
0.0
3                   0.0              0.0             0.0
0.0
4                   0.0              0.0             0.0
0.0
5                   0.0              0.0             0.0
0.0
```

```
6                  0.0              0.0             0.0
1.0
7                  0.0              0.0             1.0
0.0
8                  0.0              0.0             0.0
0.0
9                  1.0              0.0             0.0
0.0

   review_bow__update  review_bow__with  age_impute__age
0                 1.0               0.0           21.000
1                 0.0               0.0           19.000
2                 0.0               0.0           22.125
3                 0.0               0.0           17.000
4                 1.0               1.0           24.000
5                 0.0               0.0           22.125
6                 0.0               0.0           30.000
7                 0.0               1.0           19.000
8                 0.0               0.0           16.000
9                 0.0               0.0           31.000

[10 rows x 38 columns]
```

Feature Union

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

# Generating a random dataset with 10 rows and 4 columns
np.random.seed(42)  # For reproducibility
data = np.random.randn(10, 4)

# Creating a DataFrame and naming the columns
df = pd.DataFrame(data, columns=['f1', 'f2', 'f3', 'y'])

df

         f1        f2        f3         y
0  0.496714 -0.138264  0.647689  1.523030
1 -0.234153 -0.234137  1.579213  0.767435
2 -0.469474  0.542560 -0.463418 -0.465730
3  0.241962 -1.913280 -1.724918 -0.562288
4 -1.012831  0.314247 -0.908024 -1.412304
5  1.465649 -0.225776  0.067528 -1.424748
6 -0.544383  0.110923 -1.150994  0.375698
7 -0.600639 -0.291694 -0.601707  1.852278
8 -0.013497 -1.057711  0.822545 -1.220844
9  0.208864 -1.959670 -1.328186  0.196861
```

```
from sklearn.pipeline import FeatureUnion
from sklearn.decomposition import PCA

# Define FeatureUnion
feature_union = FeatureUnion([
    ('scaler', StandardScaler()),  # Apply StandardScaler
    ('pca', PCA(n_components=2))   # Apply PCA, reduce to 2 components
])

X_transformed = feature_union.fit_transform(df.drop(columns=['y']))

pd.DataFrame(X_transformed,
columns=feature_union.get_feature_names_out())
```

```
   scaler__f1  scaler__f2  scaler__f3  pca__pca0  pca__pca1
0    0.815293    0.418360    0.947878  -1.025659  -0.425413
1   -0.282292    0.302777    1.873701  -1.772532  -0.358223
2   -0.635686    1.239158   -0.156427  -0.327888   1.038742
3    0.432718   -1.721587   -1.410206   1.911072  -0.689960
4   -1.451676    0.963905   -0.598312   0.193153   1.371662
5    2.270396    0.312856    0.371269  -0.511760  -0.891133
6   -0.748180    0.718778   -0.839795   0.484280   1.020731
7   -0.832663    0.233387   -0.293870   0.191723   0.583958
8    0.049080   -0.690119    1.121664  -0.726878  -0.811461
9    0.383011   -1.777515   -1.015903   1.584488  -0.838903
```

Pipeline

```
import pandas as pd
import numpy as np

# Generating a random dataset with 10 rows and 4 columns
np.random.seed(42)  # For reproducibility
data = np.random.randn(10, 4)

# Creating a DataFrame and naming the columns
df = pd.DataFrame(data, columns=['f1', 'f2', 'f3', 'y'])

df
```

```
         f1        f2        f3         y
0  0.496714 -0.138264  0.647689  1.523030
1 -0.234153 -0.234137  1.579213  0.767435
2 -0.469474  0.542560 -0.463418 -0.465730
3  0.241962 -1.913280 -1.724918 -0.562288
4 -1.012831  0.314247 -0.908024 -1.412304
5  1.465649 -0.225776  0.067528 -1.424748
6 -0.544383  0.110923 -1.150994  0.375698
7 -0.600639 -0.291694 -0.601707  1.852278
8 -0.013497 -1.057711  0.822545 -1.220844
9  0.208864 -1.959670 -1.328186  0.196861
```

```
from sklearn.pipeline import Pipeline

# Define FeatureUnion
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Apply StandardScaler
    ('pca', PCA(n_components=2))
])

pd.DataFrame(pipeline.fit_transform(X),
columns=pipeline.get_feature_names_out())
```

```
         pca0      pca1
0   -2.264703  0.480027
1   -2.080961 -0.674134
2   -2.364229 -0.341908
3   -2.299384 -0.597395
4   -2.389842  0.646835
..        ...       ...
145  1.870503  0.386966
146  1.564580 -0.896687
147  1.521170  0.269069
148  1.372788  1.011254
149  0.960656 -0.024332

[150 rows x 2 columns]
```

Slightly Complex Example

```
import pandas as pd

# Define the data with numeric labels for sentiment
data = {
    "Social Media Platform": ["Twitter", "Facebook", "Instagram",
"Twitter", "Facebook",
                              "Instagram", "Twitter", "Facebook",
"Instagram", "Twitter"],
    "Review": ["Love the new update!", "Too many ads now", "Great for
sharing photos",
               "Newsfeed algorithm is biased", "Privacy concerns with
latest update",
               "Amazing filters!", "Too much spam", "Easy to connect
with friends",
               "Stories feature is fantastic", "Customer support
lacking"],
    "age": [21, 19, np.nan, 17, 24, np.nan, 30, 19, 16, 31],
    "Sentiment": [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]  # Numeric labels: 1
for Positive, 0 for Negative
}

# Create a DataFrame
```

```
df = pd.DataFrame(data)

print(df)

  Social Media Platform                               Review   age
Sentiment
0               Twitter                 Love the new update!  21.0
1
1              Facebook                     Too many ads now  19.0
0
2             Instagram             Great for sharing photos   NaN
1
3               Twitter         Newsfeed algorithm is biased  17.0
0
4              Facebook  Privacy concerns with latest update  24.0
0
5             Instagram                     Amazing filters!   NaN
1
6               Twitter                        Too much spam  30.0
0
7              Facebook         Easy to connect with friends  19.0
1
8             Instagram         Stories feature is fantastic  16.0
1
9               Twitter             Customer support lacking  31.0
0

def count_words(reviews):
    # Count the number of words in each review
    # Assuming reviews is a 1D array-like of text strings
    return np.array([len(review.split()) for review in
reviews]).reshape(-1, 1)

from sklearn.preprocessing import FunctionTransformer

# Create the FunctionTransformer using the count_words function
word_count_transformer = FunctionTransformer(count_words)

feature_union = FeatureUnion([
    ('word_count', word_count_transformer),
    ('bag_of_words', CountVectorizer())
])

column_transformer = ColumnTransformer(
    transformers=[
        ('age_imputer', SimpleImputer(strategy='mean'), ['age']),
        ('platform_ohe', OneHotEncoder(), ['Social Media Platform']),
        ('review_processing', feature_union, 'Review')
    ],
    remainder='drop'  # Drop other columns not specified here
)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MaxAbsScaler
from sklearn.feature_selection import SelectKBest,chi2

final_pipeline = Pipeline(steps=[
    ('col_transformer', column_transformer),
    ('scaler', MaxAbsScaler()),
    ('selector', SelectKBest(score_func=chi2,k=10)),
    ('classifier', LogisticRegression())
])

final_pipeline.fit(df.drop(columns=['Sentiment']), df['Sentiment'])
```

```
Pipeline(steps=[('col_transformer',
                 ColumnTransformer(transformers=[('age_imputer',
                                                  SimpleImputer(),
['age']),
                                                 ('platform_ohe',
                                                  OneHotEncoder(),
                                                  ['Social Media
Platform']),
                                                 ('review_processing',

FeatureUnion(transformer_list=[('word_count',

FunctionTransformer(func=<function count_words at 0x7ea951525000>)),

('bag_of_words',

CountVectorizer())]),
                                                  'Review')])),
                ('scaler', MaxAbsScaler()),
                ('selector',
                 SelectKBest(score_func=<function chi2 at
0x7ea9515a3520>)),
                ('classifier', LogisticRegression())])
```