day-35-40-handeling-missing-value

May 26, 2025

#Note: - If Numaric Column contian <5% missing value and Randomly. Then you can use —>mean,median stratgy for handeling missing value

- but when not randomly use 'End of Distribution'–1)Q1-1.5IQR 2)Q1+1.5IQR where IQR=Q3-Q1
- When data is Numercal Catagorical and Missing value >10% .Then use Arbitary Missing value handeling or insert -1 where NaN value.
- When Data String Catagorical and Missing value >10% .Then use "missing/empty" string as a new Catagory to NaN value

```
# si=SimpleImputer(stratgy='constant',fill_value=-1 | "missing/empty")
X_train['new']=si.fit_transfrom(X_train)
```

```
[ ]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ]: df=pd.read_csv('/content/Titanic-Dataset.csv')
     df.head(3)
```

```
[ ]:    PassengerId  Survived  Pclass  \
     0            1         0       3
     1            2         1       1
     2            3         1       3

                                                    Name     Sex   Age  SibSp  \
     0                            Braund, Mr. Owen Harris    male  22.0      1
     1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
     2                             Heikkinen, Miss. Laina  female  26.0      0

        Parch            Ticket     Fare Cabin Embarked
     0      0         A/5 21171   7.2500   NaN        S
     1      0          PC 17599  71.2833   C85        C
     2      0  STON/O2. 3101282   7.9250   NaN        S
```

```
[ ]: df=df.iloc[:,[1,2,4,5,6,7,9,10,11]]
     df.head(3)
```

```
[ ]:    Survived  Pclass     Sex   Age  SibSp  Parch     Fare Cabin Embarked
     0         0       3    male  22.0      1      0   7.2500   NaN        S
     1         1       1  female  38.0      1      0  71.2833   C85        C
     2         1       3  female  26.0      0      0   7.9250   NaN        S
```

```
[ ]: df['Cabin'].isnull().sum()/df.shape[0]*100
```

```
[ ]: np.float64(77.10437710437711)
```

```
[ ]: df.drop('Cabin',axis=1,inplace=True)
     df.head(3)
```

```
[ ]:    Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
     0         0       3    male  22.0      1      0   7.2500        S
     1         1       1  female  38.0      1      0  71.2833        C
     2         1       3  female  26.0      0      0   7.9250        S
```

```
[ ]: from sklearn.model_selection import train_test_split
     from sklearn.impute import SimpleImputer,MissingIndicator
     from sklearn.preprocessing import OneHotEncoder,OrdinalEncoder,LabelEncoder
     from sklearn.preprocessing import StandardScaler,MinMaxScaler
     from sklearn.preprocessing import FunctionTransformer,PowerTransformer
     from sklearn.compose import ColumnTransformer,make_column_transformer
     from sklearn.pipeline import Pipeline,make_pipeline
     import scipy.stats as stats

     from sklearn.linear_model import LogisticRegression,LinearRegression
```

#Casually Implement of MissingIndicator

```
[ ]: df.head(2)
```

```
[ ]:    Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
     0         0       3    male  22.0      1      0   7.2500        S
     1         1       1  female  38.0      1      0  71.2833        C
```

```
[ ]: mi=MissingIndicator()
     df['new_age']=mi.fit_transform(df[['Age']])
     df.sample(10)
```

```
[ ]:      Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked  new_age
     246         0       3  female  25.0      0      0   7.7750        S    False
     358         1       3  female   NaN      0      0   7.8792        Q     True
     473         1       2  female  23.0      0      0  13.7917        C    False
     280         0       3    male  65.0      0      0   7.7500        Q    False
     807         0       3  female  18.0      0      0   7.7750        S    False
     53          1       2  female  29.0      1      0  26.0000        S    False
```

```
785          0       3    male  25.0      0      0   7.2500        S    False
7            0       3    male   2.0      3      1  21.0750        S    False
279          1       3  female  35.0      1      1  20.2500        S    False
240          0       3  female   NaN      1      0  14.4542        C     True
```

```
[ ]: df.shape
```

```
[ ]: (891, 9)
```

```
[ ]: X_train,X_test,y_train,y_test=train_test_split(df.
      ↪drop(['Survived','new_age'],axis=1),df['Survived'],test_size=0.
      ↪2,random_state=42)
     X_train.head(2)
```

```
[ ]:      Pclass   Sex   Age  SibSp  Parch  Fare Embarked
     331       1  male  45.5      0      0  28.5        S
     733       2  male  23.0      0      0  13.0        S
```

```
[ ]: age_pipe=Pipeline([
         ('imp_age',SimpleImputer(strategy='median'))
     ])
     fare_pipe=Pipeline([
         ('normalize_fare',PowerTransformer(method='yeo-johnson'))
     ])

     sex_emb_pipe=Pipeline([
         ('sex_imp',SimpleImputer(strategy='most_frequent')),
         ('ohe_sex_emb',OneHotEncoder(dtype=np.
      ↪int32,drop='first',sparse_output=False,handle_unknown='ignore'))
     ])
```

```
[ ]: CT1=make_column_transformer(
         (age_pipe,['Age']),
         (fare_pipe,['Fare']),
         (sex_emb_pipe,['Sex','Embarked']),
         remainder='passthrough'
     )
```

```
[ ]: CT1.fit_transform(X_train)
```

```
[ ]: array([[45.5       ,  0.47999826,  1.        , …,  1.        ,
              0.        ,  0.        ],
            [23.        , -0.28375264,  1.        , …,  2.        ,
              0.        ,  0.        ],
            [32.        , -0.77244668,  1.        , …,  3.        ,
              0.        ,  0.        ],
            …,
            [41.        , -0.20313477,  1.        , …,  3.        ,
              2.        ,  0.        ],
            [14.        ,  1.78386853,  0.        , …,  1.        ,
              1.        ,  2.        ],
            [21.        ,  1.40043606,  1.        , …,  1.        ,
              0.        ,  1.        ]])
```

[ ]:

```python
pipe_before=make_pipeline(CT1,StandardScaler(),LogisticRegression())

pipe_before.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('pipeline-1',
                                                  Pipeline(steps=[('imp_age',
SimpleImputer(strategy='median'))]),
                                                  ['Age']),
                                                 ('pipeline-2',
Pipeline(steps=[('normalize_fare',
PowerTransformer())]),
                                                  ['Fare']),
                                                 ('pipeline-3',
                                                  Pipeline(steps=[('sex_imp',
SimpleImputer(strategy='most_frequent')),
('ohe_sex_emb',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Sex', 'Embarked'])])),
                ('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

[ ]:
```python
y_pred=pipe_before.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

[ ]:
```
0.7988826815642458
```

[ ]:
```python
from sklearn.model_selection import cross_val_score
cross_val_score(pipe_before,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:
```
np.float64(0.7864572047670639)
```

##after Missing Indicator(builtin and manually)

[ ]:
```python
age_p=Pipeline([
    # ('mi_age',MissingIndicator()),
    ('imu_age',SimpleImputer(add_indicator=True))
])
```

[ ]:
```python
CT2=make_column_transformer(
    (age_p,['Age']),
    (fare_pipe,['Fare']),
    (sex_emb_pipe,['Sex','Embarked']),
    remainder='passthrough'
)
CT2.fit_transform(X_train)
```

[ ]:
```
array([[45.5       ,  0.        ,  0.47999826, …,  1.        ,
         0.        ,  0.        ],
       [23.        ,  0.        , -0.28375264, …,  2.        ,
         0.        ,  0.        ],
       [32.        ,  0.        , -0.77244668, …,  3.        ,
         0.        ,  0.        ],
       …,
       [41.        ,  0.        , -0.20313477, …,  3.        ,
         2.        ,  0.        ],
       [14.        ,  0.        ,  1.78386853, …,  1.        ,
         1.        ,  2.        ],
       [21.        ,  0.        ,  1.40043606, …,  1.        ,
         0.        ,  1.        ]])
```

[ ]:
```python
pipe_after=make_pipeline(CT2,StandardScaler(),LogisticRegression())
pipe_after.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
```

```
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('pipeline-1',
                                                  Pipeline(steps=[('imu_age',
SimpleImputer(add_indicator=True))]),
                                                  ['Age']),
                                                 ('pipeline-2',
Pipeline(steps=[('normalize_fare',
PowerTransformer())]),
                                                  ['Fare']),
                                                 ('pipeline-3',
                                                  Pipeline(steps=[('sex_imp',
SimpleImputer(strategy='most_frequent')),
('ohe_sex_emb',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Sex', 'Embarked'])])),
                ('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

[ ]:
```
y_pred=pipe_after.predict(X_test)
accuracy_score(y_pred,y_test)
```

[ ]:
```
0.8100558659217877
```

[ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(pipe_after,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:
```
np.float64(0.7864572047670639)
```

#check manually

[ ]:
```
df.head()
#True and False jodi bool type hoi tahole ML a Encode korte hobe na
```

[ ]:
```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked  new_age
0         0       3    male  22.0      1      0   7.2500        S    False
```

```
1          1       1  female  38.0       1      0  71.2833        C    False
2          1       3  female  26.0       0      0   7.9250        S    False
3          1       1  female  35.0       1      0  53.1000        S    False
4          0       3    male  35.0       0      0   8.0500        S    False
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    object
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  889 non-null    object
 8   new_age   891 non-null    bool
dtypes: bool(1), float64(2), int64(4), object(2)
memory usage: 56.7+ KB
```

[ ]:
```
X_train,X_test,y_train,y_test=train_test_split(df.
 ↪drop(['Survived'],axis=1),df['Survived'],test_size=0.2,random_state=42)
X_train.head(2)
```

[ ]:
```
     Pclass   Sex   Age  SibSp  Parch  Fare Embarked  new_age
331       1  male  45.5      0      0  28.5        S    False
733       2  male  23.0      0      0  13.0        S    False
```

[ ]:
```
age_pipe2=Pipeline([
    ('age_imp',SimpleImputer())
])
fare_pipe2=Pipeline([
    ('normalization_fare',PowerTransformer())
])

sex_emb_pipe2=Pipeline([
    ('sex_imp',SimpleImputer(strategy='most_frequent')),
    ('ohe_sex_emb',OneHotEncoder(dtype=np.
 ↪int32,drop='first',sparse_output=False,handle_unknown='ignore'))
])

CT3=make_column_transformer(
    (age_pipe2,['Age']),
    (fare_pipe2,['Fare']),
    (sex_emb_pipe2,['Sex','Embarked']),
    remainder='passthrough'
)

CT3.fit_transform(X_train)
```

[ ]:

```
array([[45.5, 0.47999826231989395, 1, …, 0, 0, False],
       [23.0, -0.28375263784654847, 1, …, 0, 0, False],
       [32.0, -0.7724466813890875, 1, …, 0, 0, False],
       …,
       [41.0, -0.20313476693661398, 1, …, 2, 0, False],
       [14.0, 1.7838685328829773, 0, …, 1, 2, False],
       [21.0, 1.400436058476949, 1, …, 0, 1, False]], dtype=object)
```

[ ]:

```
pipe3=make_pipeline(CT3,StandardScaler(),LogisticRegression())
pipe3.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('pipeline-1',
                                                  Pipeline(steps=[('age_imp',
SimpleImputer())]),
                                                  ['Age']),
                                                 ('pipeline-2',
Pipeline(steps=[('normalization_fare',
PowerTransformer())]),
                                                  ['Fare']),
                                                 ('pipeline-3',
                                                  Pipeline(steps=[('sex_imp',
SimpleImputer(strategy='most_frequent')),
('ohe_sex_emb',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                          ['Sex', 'Embarked'])])),
                ('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

```
[ ]: y_pred3=pipe3.predict(X_test)
     accuracy_score(y_pred3,y_test)
```

```
[ ]: 0.8100558659217877
```

```
[ ]: cross_val_score(pipe3,X_train,y_train,cv=5,scoring='accuracy').mean()
```

```
[ ]: np.float64(0.7864572047670639)
```

#Grid SearchCV: Find automatically better parameter for any Transformation which increase accuracy [SEE Latter]

```
[ ]: # from sklearn.model_selection import GridSearchCV
     # pram_grid={
     #     'A':[0.01, 0.1, 1, 10],
     #     'imp_age':['mean','median'],
     #     'imp_cat':['most_frequent','constant']
     # }
     # grid_search=GridSearchCV(pipe3,pram_grid,cv=5,scoring='accuracy')
     # grid_search.fit(X_train,y_train)
```

```
[ ]: # pram_grid = {
     #     'logisticregression__C': [0.01, 0.1, 1, 10],  # Assuming 'A' was intended␣
      ↪for LogisticRegression C
     #     'columntransformer__pipeline-1__simpleimputer__strategy':␣
      ↪['mean','median'], # Strategy for SimpleImputer in age_pipe2
     #     'columntransformer__pipeline-3__simpleimputer__strategy':␣
      ↪['most_frequent','constant'] # Strategy for SimpleImputer in sex_emb_pipe2
     # }

     # grid_search=GridSearchCV(pipe3, pram_grid, cv=5, scoring='accuracy')
     # grid_search.fit(X_train,y_train)
```

#Day-39:KNN Imputer

```
[ ]: X_train.head()
```

```
[ ]:      Pclass     Sex   Age  SibSp  Parch     Fare Embarked  new_age
     331       1    male  45.5      0      0  28.5000        S    False
     733       2    male  23.0      0      0  13.0000        S    False
     382       3    male  32.0      0      0   7.9250        S    False
```

```
704       3    male  26.0      1      0   7.8542        S    False
813       3  female   6.0      4      2  31.2750        S    False
```

[ ]:

```

```

[ ]:

```
X_train['Age'].isnull().sum()
```

[ ]:

```
np.int64(140)
```

[ ]:

```
#ChatGPT


from sklearn.pipeline import Pipeline
from sklearn.impute import KNNImputer
from sklearn.preprocessing import PowerTransformer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# 1. Load Titanic dataset
df = df
X = df.drop('Survived', axis=1)
y = df['Survived']

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 3. Select columns
numeric_cols = ['Age', 'Fare', 'SibSp', 'Parch']
categorical_cols = ['Sex', 'Embarked']

# 4. Step 1: Numeric imputation using KNN
knn_impute_pipe = Pipeline([
    ('knn_imputer', KNNImputer())
])

# 5. Step 2: PowerTransform only Fare
fare_power_pipe = Pipeline([
    ('power', PowerTransformer(method='yeo-johnson'))
])

# 6. Step 3: Categorical encode
cat_pipe = Pipeline([
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# 7. Combine: ColumnTransformer
preprocessor = ColumnTransformer([
    ('knn_num', knn_impute_pipe, numeric_cols),# shob golo column use kore just␣
 ↪akta column "Age" ei value Add korse[Onno kono Column k Affact kore nai. Tai␣
 ↪kaj korche]
                                                          #Ta o jodi kaj na kore Tahole␣
 ↪Fare Column k bad dia all Numeric column pass koro
    ('fare_power', fare_power_pipe, ['Fare']),
    ('cat', cat_pipe, categorical_cols)
])

# 8. Final Pipeline: Preprocessor + Model
model_pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('clf', LogisticRegression(max_iter=1000))
])

# 9. Train
model_pipeline.fit(X_train, y_train)

# 10. Predict & Evaluate
y_pred = model_pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.770949720670391
```

[ ]:

```
from sklearn.impute import KNNImputer
knn_age_pipe=Pipeline([
    ('knn_age',KNNImputer()) #just Age colum Pass korle hobe na All Numerical␣
 ↪Column Pass koro(with Encoded if have)
])
```

[ ]:

```
CT4=ColumnTransformer([
    ('knn_age_imp',knn_age_pipe,['Age']),#just Age ar opor KNN apply kore Fill␣
 ↪kore.But KNN All column ar distance ar opor bitte kore value ber korer kotha
    ('normalize_fare',fare_pipe,['Fare']),
    ('sex_emb_ohe',sex_emb_pipe,['Sex','Embarked']),
],remainder='passthrough')
```

[ ]:

```
pipe_knn=make_pipeline(
    CT4,StandardScaler(),LogisticRegression()
)
pipe_knn.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('knn_age_imp',
                                                  Pipeline(steps=[('knn_age',
KNNImputer())]),
                                                  ['Age']),
                                                 ('normalize_fare',
Pipeline(steps=[('normalize_fare',
PowerTransformer())]),
                                                  ['Fare']),
                                                 ('sex_emb_ohe',
                                                  Pipeline(steps=[('sex_imp',
SimpleImputer(strategy='most_frequent')),
('ohe_sex_emb',
OneHotEncoder(drop='first',
 dtype=<class 'numpy.int32'>,
 handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Sex', 'Embarked'])])),
                ('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

[ ]:

```
y_pred_knn=pipe_knn.predict(X_test)
accuracy_score(y_pred_knn,y_test)
```

[ ]:

```
0.8100558659217877
```

[ ]:

```
cross_val_score(pipe_knn,X_train,y_train,cv=5,scoring='accuracy').mean()
```

[ ]:

```
np.float64(0.7864572047670639)
```

[ ]:

```

```

#IterativeImputer:(use Lineare Model)

Iterative Imputer

? IterativeImputer
multivariate imputation

Missing value handling using Iterative Imputer (from `sklearn.impute`)

0.0.1 ?

`IterativeImputer`
multivariate imputation

0.0.2 : Python (Scikit-learn)

```
import pandas as pd
import numpy as np
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge  #

#
data = {
    'age': [25, np.nan, 35, 40, np.nan],
    'salary': [50000, 60000, np.nan, 80000, 75000],
    'experience': [2, 4, 5, np.nan, 3]
}
df = pd.DataFrame(data)

#
imputer = IterativeImputer(estimator=BayesianRidge(), max_iter=10, random_state=0)

#
df_imputed = imputer.fit_transform(df)

#     DataFrame
df_imputed = pd.DataFrame(df_imputed, columns=df.columns)

print(df_imputed)
```

0.0.3 Parameters

- `estimator`: (BayesianRidge, DecisionTreeRegressor, KNN)
- `max_iter`:
- `initial_strategy`: (mean, median, most_frequent)

- `random_state` :Reproducivility random_state

0.0.4 ?

-
- or
- Simple mean/median
-

0.1 (SimpleImputer)

0.1.1 :

- (slow)
- (StandardScaler)
- High Memory Consumed in for Server(deploy with main dataset for user missing value handeling.
- Overfitting: Iteration

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_iterative_imputer  #
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import PowerTransformer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# 1. Load Titanic dataset
df = df
X = df.drop('Survived', axis=1)
y = df['Survived']

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 3. Select columns
numeric_cols = ['Age', 'Fare', 'SibSp', 'Parch']
categorical_cols = ['Sex', 'Embarked']

# 4. Step 1: Numeric imputation using IterativeImputer
iter_impute_pipe = Pipeline([
    ('iter_imputer', IterativeImputer(random_state=42))
```

```python
])

# 5. Step 2: PowerTransform only Fare
fare_power_pipe = Pipeline([
    ('power', PowerTransformer(method='yeo-johnson'))
])

# 6. Step 3: Categorical encode
cat_pipe = Pipeline([
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# 7. Combine: ColumnTransformer
preprocessor = ColumnTransformer([
    ('iter_num', iter_impute_pipe, numeric_cols),
    ('fare_power', fare_power_pipe, ['Fare']),
    ('cat', cat_pipe, categorical_cols)
])

# 8. Final Pipeline: Preprocessor + Model
model_pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('clf', LogisticRegression(max_iter=1000))
])

# 9. Train
model_pipeline.fit(X_train, y_train)

# 10. Predict & Evaluate
y_pred = model_pipeline.predict(X_test)
print("Accuracy with Iterative Imputer:", accuracy_score(y_test, y_pred))
```

```
Accuracy with Iterative Imputer: 0.770949720670391
```

[ ]:

```python
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import PowerTransformer, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Load Titanic dataset
# Make sure df is already loaded properly with 'Survived' column
X = df.drop('Survived', axis=1)
```

```
y = df['Survived']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Column groups
numeric_cols = ['Age', 'SibSp', 'Parch']         # Only these go to␣
 ↪IterativeImputer
fare_col = ['Fare']                               # Separate transformer
categorical_cols = ['Sex', 'Embarked']

# Pipelines
iter_impute_pipe = Pipeline([
    ('iter_imputer', IterativeImputer(estimator=LogisticRegression(),␣
 ↪random_state=42))
])

fare_power_pipe = Pipeline([
    ('power', PowerTransformer(method='yeo-johnson'))
])

cat_pipe = Pipeline([
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# ColumnTransformer
preprocessor = ColumnTransformer([
    ('iter_num', iter_impute_pipe, numeric_cols),
    ('fare_power', fare_power_pipe, fare_col),
    ('cat', cat_pipe, categorical_cols)
])

# Final pipeline
model_pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('clf', LogisticRegression(max_iter=1000))
])

# Train
model_pipeline.fit(X_train, y_train)

# Predict & Evaluate
y_pred = model_pipeline.predict(X_test)
print("Accuracy with Iterative Imputer:", accuracy_score(y_test, y_pred))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-126-00f56afa5758> in <cell line: 0>()
     49
     50 # Train
---> 51 model_pipeline.fit(X_train, y_train)
     52
     53 # Predict & Evaluate

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in fit(self, X, y,␣
 ↪**params)
    652
    653         routed_params = self._check_method_params(method="fit",␣
 ↪props=params)
--> 654         Xt = self._fit(X, y, routed_params, raw_params=params)
    655         with _print_elapsed_time("Pipeline", self._log_message(len(self
 ↪steps) - 1)):
    656             if self._final_estimator != "passthrough":
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in _fit(self, X, y,␣
↪routed_params, raw_params)
    586             )
    587 
--> 588             X, fitted_transformer = fit_transform_one_cached(
    589                 cloned_transformer,
    590                 X,

/usr/local/lib/python3.11/dist-packages/joblib/memory.py in __call__(self,␣
↪*args, **kwargs)
    324 
    325     def __call__(self, *args, **kwargs):
--> 326         return self.func(*args, **kwargs)
    327 
    328     def call_and_shelve(self, *args, **kwargs):

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in␣
↪_fit_transform_one(transformer, X, y, weight, message_clsname, message, params)
   1549     with _print_elapsed_time(message_clsname, message):
   1550         if hasattr(transformer, "fit_transform"):
-> 1551             res = transformer.fit_transform(X, y, **params.
↪get("fit_transform", {}))
   1552         else:
   1553             res = transformer.fit(X, y, **params.get("fit", {})).
↪transform(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_set_output.py in␣
↪wrapped(self, X, *args, **kwargs)
    317     @wraps(f)
    318     def wrapped(self, X, *args, **kwargs):
--> 319         data_to_wrap = f(self, X, *args, **kwargs)
    320         if isinstance(data_to_wrap, tuple):
    321             # only wrap the first output for cross decomposition

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390 
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/compose/_column_transformer.py␣
↪in fit_transform(self, X, y, **params)
    999             routed_params = self._get_empty_routing()
   1000 
```

```
-> 1001         result = self._call_func_on_transformers(
   1002             X,
   1003             y,

/usr/local/lib/python3.11/dist-packages/sklearn/compose/_column_transformer.py in _call_func_on_transformers(self, X, y, func, column_as_labels, routed_params)
    908                 )
    909
--> 910             return Parallel(n_jobs=self.n_jobs)(jobs)
    911
    912         except ValueError as e:

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in __call__(self, iterable)
     75             for delayed_func, args, kwargs in iterable
     76         )
---> 77         return super().__call__(iterable_with_config)
     78
     79

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in __call__(self, iterable)
   1983             output = self._get_sequential_output(iterable)
   1984             next(output)
-> 1985             return output if self.return_generator else list(output)
   1986
   1987         # Let's create an ID that uniquely identifies the current call. If the

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in _get_sequential_output(self, iterable)
   1911                 self.n_dispatched_batches += 1
   1912                 self.n_dispatched_tasks += 1
-> 1913                 res = func(*args, **kwargs)
   1914                 self.n_completed_tasks += 1
   1915                 self.print_progress()

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in __call__(self, *args, **kwargs)
    137             config = {}
    138         with config_context(**config):
--> 139             return self.function(*args, **kwargs)
    140
    141
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in␣
 ↪_fit_transform_one(transformer, X, y, weight, message_clsname, message, params)
   1549     with _print_elapsed_time(message_clsname, message):
   1550         if hasattr(transformer, "fit_transform"):
-> 1551             res = transformer.fit_transform(X, y, **params.
 ↪get("fit_transform", {}))
   1552         else:
   1553             res = transformer.fit(X, y, **params.get("fit", {})).
 ↪transform(

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390 
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in␣
 ↪fit_transform(self, X, y, **params)
    728             )
    729             if hasattr(last_step, "fit_transform"):
--> 730                 return last_step.fit_transform(

    731                     Xt, y, **last_step_params["fit_transform"]
    732                 )

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_set_output.py in␣
 ↪wrapped(self, X, *args, **kwargs)
    317     @wraps(f)
    318     def wrapped(self, X, *args, **kwargs):
--> 319         data_to_wrap = f(self, X, *args, **kwargs)
    320         if isinstance(data_to_wrap, tuple):
    321             # only wrap the first output for cross decomposition

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390 
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/impute/_iterative.py in␣
 ↪fit_transform(self, X, y, **params)
    857                     n_features, feat_idx, abs_corr_mat
    858                 )
```

```
--> 859                 Xt, estimator = self._impute_one_feature(
    860                     Xt,
    861                     mask_missing_values,

/usr/local/lib/python3.11/dist-packages/sklearn/impute/_iterative.py in␣
 ↪_impute_one_feature(self, X_filled, mask_missing_values, feat_idx,␣
 ↪neighbor_feat_idx, estimator, fit_mode, params)
    425                 axis=0,
    426             )
--> 427             estimator.fit(X_train, y_train, **params)
    428 
    429         # if no missing values, don't predict

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390 
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py in␣
 ↪fit(self, X, y, sample_weight)
   1229             accept_large_sparse=solver not in ["liblinear", "sag",␣
 ↪"saga"],
   1230         )
-> 1231         check_classification_targets(y)
   1232         self.classes_ = np.unique(y)
   1233 

/usr/local/lib/python3.11/dist-packages/sklearn/utils/multiclass.py in␣
 ↪check_classification_targets(y)
    220         "multilabel-sequences",
    221     ]:
--> 222         raise ValueError(
    223             f"Unknown label type: {y_type}. Maybe you are trying to fit
 ↪a "
    224             "classifier, which expects discrete classes on a "

ValueError: Unknown label type: continuous. Maybe you are trying to fit a␣
 ↪classifier, which expects discrete classes on a regression target with␣
 ↪continuous values.
```