SciKit-learn Data Preprocessing

1) Data Cleaning

from sklearn.impute import SimpleImputer

1.1 Cleaning Duplicate values

```
# Checking for duplicate values
data.duplicated().sum()

# Displaying duplicate records
data[data.duplicated() == True]

# Removing duplicate records
data.drop_duplicates(inplace=True)

data.duplicated().sum()
```

1.2 Removing columns with majority NaN values

```
# Checking for NaN values in df
data.isnull().sum().sum()

data.isnull().sum()

max_NaN_cols = data.isnull().sum()[data.isnull().sum() > 258].keys()
max_NaN_cols


data_copy = data.copy()

# Drop such columns from df
data_copy.drop(max_NaN_cols, axis=1, inplace=True)
```

1.3 Numerical Missing Value Imputation

```
# Getting numerical columns
num_cols = data_copy.select_dtypes(['int', 'float']).columns
num_cols

impute_mean = SimpleImputer(strategy='mean')
data_copy[num_cols] = impute_mean.fit_transform(data_copy[num_cols])

data_copy.isnull().sum().sum()
```

1.4 Categorical Missing Value Imputation

```
# Getting categorical columns
cat_cols = data_copy.select_dtypes(['object']).columns
cat_cols


impute_mode = SimpleImputer(strategy='most_frequent')
data_copy[cat_cols] = impute_mode.fit_transform(data_copy[cat_cols])

data_copy.isnull().sum().sum()


# Missing value imputation by median
impute_median = SimpleImputer(strategy='median')
data_copy[num_cols] = impute_median.fit_transform(data_copy[num_cols])

# Missing value imputation by constant
impute_const = SimpleImputer(strategy='constant', fill_value='Missing')
data_copy[cat_cols] = impute_const.fit_transform(data_copy[cat_cols])
```

2) Label Encoding

```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode the 'embark_town' column
data['embark_town'] = label_encoder.fit_transform(data['embark_town'])
```

3) Ordinal Encoding

```
from category_encoders import OrdinalEncoder

# Getting unique values of that column to encode in an order
data['class'].value_counts()

# Specify order
maplist = [{'col': 'class',
            'mapping': {'First': 0, 'Second': 1,'Third': 2}}]   # First, Second, Third are column values

ordinal_encode = OrdinalEncoder(mapping=maplist)

data = ordinal_encode.fit_transform(data)
```

5) One Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder

onehot_encoder = OneHotEncoder(sparse_output=False)

# List of categorical columns that have to be encoded
encode_col = ['sex', 'embarked', 'class', 'who', 'deck', 'embark_town', 'alive', 'adult_male',
              'alone']

# Perform one-hot encoding on the specified column
onehot_encoded = onehot_encoder.fit_transform(data[encode_col])

onehot_encoded      # Not a df, it's a numpy array

onehot_encoded.shape

# Create a DataFrame from the one-hot encoded data
onehot_encoded_df = pd.DataFrame(onehot_encoded, columns=column_names)
onehot_encoded_df

onehot_encoded_df.shape
```

Note:

- column_names should be a list of column names for newly created columns
- Here, we have to provide names for them manually
- Ex: Gender column contains Male and Female values, then the column name could be Gender_Male, Gender_Female

```
# Now this df has to be integrated with remaining numerical columns in original df

# Creating a copy of original df
data_copy = data.copy()

# Dropping those columns which will be encoded
data_copy.drop(encode_col, axis=1, inplace=True)

# Concatenating encoded df with other
data_copy = pd.concat([data_copy, onehot_encoded_df], axis=1)

data_copy.head()
```

6) Feature Scaling: Standardization

```
# Splitting data into dependent and target variable
x = data.drop('species', axis=1)
y = data['species']


scaler = StandardScaler()

x_scaled = scaler.fit_transform(x)
x_scaled

# Converting this numpy array back to dataframe
df_scaled = pd.DataFrame(x_scaled, columns=x.columns)
df_scaled.head()
```

7) Feature Scaling: Normalization

```
# Splitting data into dependent and target variable
x = data.drop('species', axis=1)
y = data['species']


scaler = MinMaxScaler()

x_scaled = scaler.fit_transform(x)
x_scaled

# Converting this numpy array back to dataframe
df_scaled = pd.DataFrame(x_scaled, columns=x.columns)
df_scaled.head()
```