In [3]:
```python
#1.Debug the given code
import pandas as pd

data = {'Feature1': ['10', '20', 'Thirty'],  # 'Thirty' is not a valid number
        'Feature2': [5.5, 6.7, 8.9]}

df = pd.DataFrame(data)
df['Feature1'] = pd.to_numeric(df['Feature1'], errors='coerce')  # Convert invalid to NaN
df['Feature1'] = df['Feature1'].fillna(0).astype(int)  # Fill NaN with 0 (or choose another value)

print(df)
```

```
   Feature1  Feature2
0        10       5.5
1        20       6.7
2         0       8.9
```

In [6]:
```python
#2.
import pandas as pd

data = {'A': [1, 2, None],  # Missing value
        'B': [4, None, 6]}  # Missing value

df = pd.DataFrame(data)
mean_value = df.mean()
df.fillna(mean_value, inplace=True)  # Error: fillna() does not modify in place
# fillna() should be used with inplace=True or assign the result back to df

print(df)
```

```
     A    B
0  1.0  4.0
1  2.0  5.0
2  1.5  6.0
```

In [ ]:
```python
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])   # X is 2D, correct shape
y = np.array([2, 4, 6, 8, 10])

model = LinearRegression()
model.fit(X, y)

print(model.predict(np.array([[6], [7]])))  # Added missing parenthesis
```

```
[12. 14.]
```

In [15]:
```python
#4.
from sklearn.preprocessing import StandardScaler
import numpy as np

data = np.array([[10], [20], [30], [40], [50]]) # Error: Should be 2D

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```

```
[[-1.41421356]
 [-0.70710678]
 [ 0.        ]
 [ 0.70710678]
 [ 1.41421356]]
```

In [16]:
```python
#5.
from sklearn.linear_model import LogisticRegression
```

```
X = [[1, 2], [3, 4], [5, 6]]
y = [1, 0, 1] # Error: Labels should be numeric

model = LogisticRegression()
model.fit(X, y)
```

Out[16]:

LogisticRegression

LogisticRegression()

In [18]:

```
#6.import pandas as pd
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({'Category': ['A', 'B', 'C', 'A']})

encoder = OneHotEncoder(sparse_output=False)
encoded = encoder.fit_transform(df[['Category']])
print(encoded)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
```

In [19]:

```
#6.
from sklearn.model_selection import train_test_split

X = [[1, 2], [3, 4], [5, 6], [7, 8]]
y = [0, 1, 0, 1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print("X_train:", X_train)
print("y_train:", y_train)
```

```
X_train: [[1, 2], [3, 4], [5, 6]]
y_train: [0, 1, 0]
```

In [20]:

```
#7.from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

X_train = [[1, 2], [3, 4], [5, 6]]
y_train = ["yes", "no", "yes"] # Error: LogisticRegression expects numerical labels

encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train)  # Convert labels to numeric

model = LogisticRegression()
model.fit(X_train, y_train_encoded)

print(model.predict([[2, 3]]))
```

```
[1]
```

In [21]:

```
#8.
import numpy as np
import pandas as pd

# Creating a dataset with missing values
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Creating a dataset with missing values
X_train = np.array([[1, 2], [3, np.nan], [5, 6]])
y_train = np.array([10, 20, 30])
```

```
# Handling missing values by replacing NaN with the column mean
imputer = SimpleImputer(strategy="mean")
X_train_imputed = imputer.fit_transform(X_train)

# Train the model
model = LinearRegression()
model.fit(X_train_imputed, y_train)

# Make a prediction
prediction = model.predict([[3, 4]])
print("Predicted Output:", prediction)
```

```
Predicted Output: [20.]
```

In [24]:

```
#9.
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train) # Error: String labels not allowed
 # Convert to numbers

model.fit(X_train_imputed, y_train_encoded)
import numpy as np
print(model.predict(np.array([[3, 4]])))
```

```
[1.]
```

In [26]:

```
#10.
from sklearn.svm import SVC
import numpy as np
from sklearn.preprocessing import StandardScaler

X_train = np.array([[1, 100], [2, 200], [3, 300]])
y_train = np.array([0, 1, 0])

model = SVC()
model.fit(X_train, y_train)
print(model.predict(np.array([[1, 150]])))  # Unreliable output due to large-scale difference

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

model.fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(np.array([[1, 150]]))
print(model.predict(X_test_scaled))
```

```
[0]
[0]
```

In [ ]: