In [138]:
```
import pandas as pd
import joblib
import numpy as Pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder,MinMaxScaler,StandardScaler,OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score ,confusion_matrix,precision_score
```

Task 1

In [99]:
```
#Task 1: Read given data into DataFrame in python. Perform Data cleaning.

#Data read from excel file
data = pd.read_excel('world_hapiness_2016.xlsx')
data

#cleaning data
data = data.loc[:, ~data.columns.duplicated()]
data.dropna(axis =1 ,how='all')
data.dropna(axis=0, how='all')
data.drop_duplicates()

data
```

Out[99]:

	Country	Region	Happiness Rank	Happiness Score	Lower Confidence Interval	Upper Confidence Interval	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)
0	Denmark	Western Europe	1	7.526	7.460	7.592	1.44178	1.16374	0.79504	0.57941	0.44453
1	Switzerland	Western Europe	2	7.509	7.428	7.590	1.52733	1.14524	0.86303	0.58557	0.41203
2	Iceland	Western Europe	3	7.501	7.333	7.669	1.42666	1.18326	0.86733	0.56624	0.14975
3	Norway	Western Europe	4	7.498	7.421	7.575	1.57744	1.12690	0.79579	0.59609	0.35776
4	Finland	Western Europe	5	7.413	7.351	7.475	1.40598	1.13464	0.81091	0.57104	0.41004
...	...	...	...	...	...	...	...	...	...	...	...
		Sub-									

Task 2

In [112]:
```
#Task 2: After data cleaning, you are required to prepare your dataset for training.
#• Separate features and labels.
#• Feature scaling/Normalization
#• Perform Label Encoding
#• Split dataset into training and testing data

#drop the values

x1 = data.drop(['Trust (Government Corruption)','Generosity'],axis=1)
y1 = data[['Trust (Government Corruption)','Generosity',]]

y_1 =np.array(y1['Trust (Government Corruption)'])
y_2 =np.array(y1['Generosity'])




numeric_feature = ['Happiness Rank','Happiness Score','Lower Confidence Interval','Upper Confidence Interval'
catego_feature =['Country' ,'Region']
catego_feature
```

Out[112]:
```
['Country', 'Region']
```

In [123]:
```
numeric_transfomer = Pipeline(steps=[('scalar', StandardScaler())])
catego_transfomer = Pipeline(steps=[('encoder', OneHotEncoder())])
```

In [126]:
```
preprocessor = ColumnTransformer(
transformers =[
    ('num',numeric_transfomer,numeric_feature),
    ('cat',catego_transfomer,catego_feature)
])
```

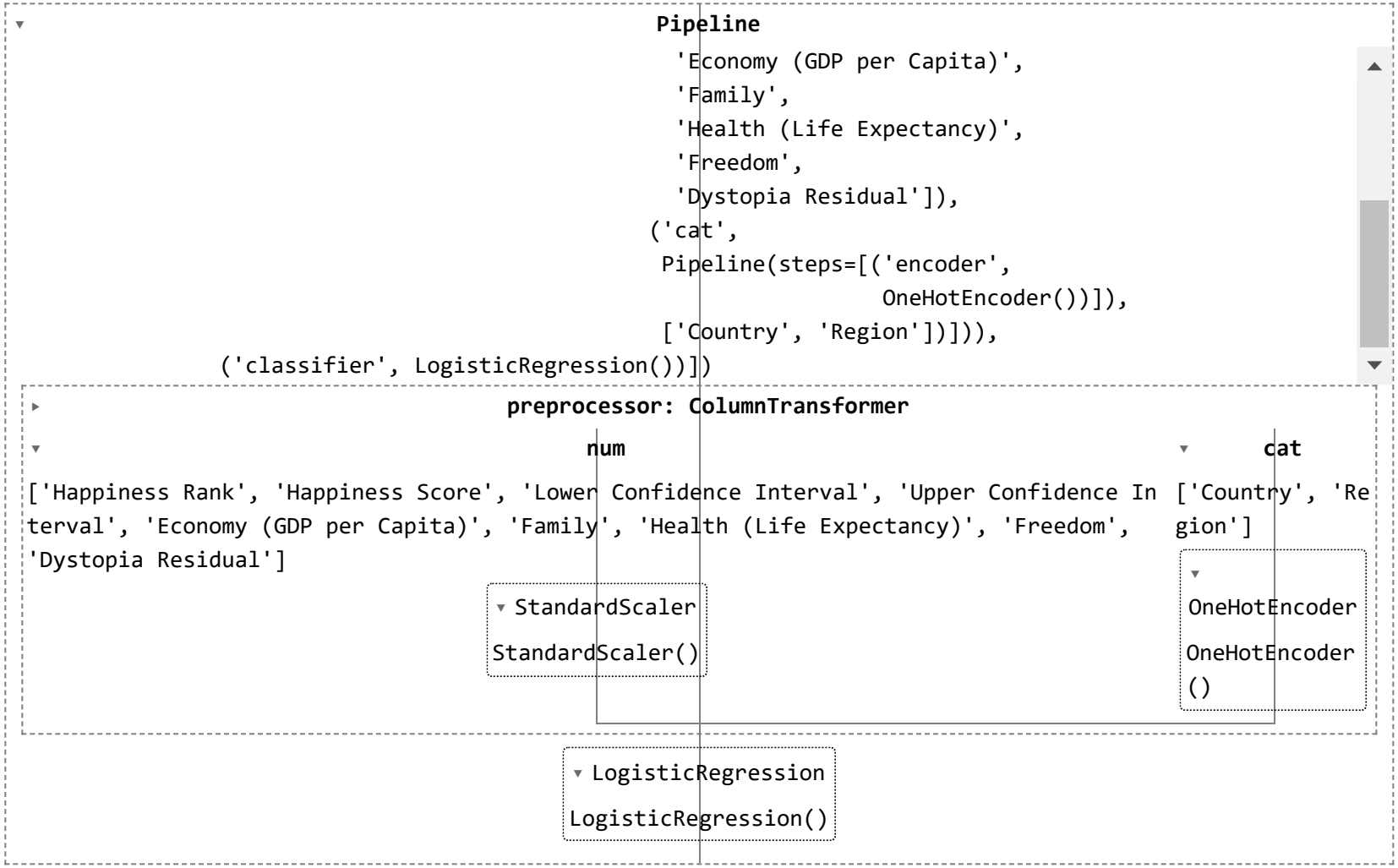
In [129]:
```
#encode the values
labelEncoder = LabelEncoder()
y_1_encod = labelEncoder.fit_transform(y_1)
y_2_encod = labelEncoder.fit_transform(y_2)
```

Task 3

In [135]:
```
pipeline = Pipeline(steps=[
('preprocessor',preprocessor),
('classifier',LogisticRegression())])
```

In [136]:
```
pipeline.fit(x1,y_1_encod)
pipeline.fit(x1,y_2_encod)
```

Out[136]:



In [140]:
```
joblib.dump(pipeline, 'model.pkl')
joblib.dump(labelEncoder, 'labelEncoder.pkl')
```

Out[140]:
```
['labelEncoder.pkl']
```

In [ ]:
```
# Load the saved preprocessing and model files
pipeline = joblib.load('model.pkl')
labelEncoder = joblib.load('labelEncoder.pkl')
```

In [145]:
```
# Set handle_unknown='ignore' for the label encoder
labelEncoder.handle_unknown = 'ignore'

# Create a dictionary to map the label encoder classes to their original labels
class_mapping = {i: label for i, label in enumerate(labelEncoder.classes_)}

# Get input from the user
country = input("Enter the country: ")
region = input("Enter the region: ")
happiness_rank = int(input("Enter the happiness rank: "))
happiness_score = float(input("Enter the happiness score: "))
lower_confidence_interval = float(input("Enter the lower confidence interval: "))
upper_confidence_interval = float(input("Enter the upper confidence interval: "))
gdp_per_capita = float(input("Enter the GDP per capita: "))
family = float(input("Enter the family score: "))
life_expectancy = float(input("Enter the life expectancy score: "))
freedom = float(input("Enter the freedom score: "))
dystopia_residual = float(input("Enter the dystopia residual: "))

# Create a DataFrame with the user input
data = pd.DataFrame({
    'Country': [country],
    'Region': [region],
    'Happiness Rank': [happiness_rank],
    'Happiness Score': [happiness_score],
    'Lower Confidence Interval': [lower_confidence_interval],
    'Upper Confidence Interval': [upper_confidence_interval],
    'Economy (GDP per Capita)': [gdp_per_capita],
    'Family': [family],
    'Health (Life Expectancy)': [life_expectancy],
    'Freedom': [freedom],
    'Dystopia Residual': [dystopia_residual]
})

# Check if the country and region categories are known
if country not in labelEncoder.classes_:
    country = 'Other'
if region not in labelEncoder.classes_:
    region = 'Other'

# Preprocess the input data using the saved preprocessing steps
preprocessed_data = pipeline['preprocessor'].transform(data)
```

```
# Make predictions using the trained model
predicted_class = pipeline['classifier'].predict(preprocessed_data)

# Map the predicted class label to its original label
predicted_label = class_mapping[predicted_class[0]]

# Print the predicted label
print("Predicted Label:", predicted_label)
```

```
Enter the country: Western Europe
Enter the region: Denmark
Enter the happiness rank: 1
Enter the happiness score: 2
Enter the lower confidence interval: 1
Enter the upper confidence interval: 3
Enter the GDP per capita: 2
Enter the family score: 2
Enter the life expectancy score: 2
Enter the freedom score: 3
Enter the dystopia residual: 2

C:\Users\Barcha\AppData\Local\Temp\ipykernel_12512\2819581133.py:36: FutureWarning: elementwise compariso
n failed; returning scalar instead, but in the future will perform elementwise comparison
  if country not in labelEncoder.classes_:
C:\Users\Barcha\AppData\Local\Temp\ipykernel_12512\2819581133.py:38: FutureWarning: elementwise compariso
n failed; returning scalar instead, but in the future will perform elementwise comparison
  if region not in labelEncoder.classes_:

---------------------------------------------------------------------------
```

In [141]:

In [ ]: