Week 4: Data Transformation

Objective: In this lab session, you will learn basic data transformation techniques. We will use Google Colab - Python IDE for most of our tasks.

Go to https://colab.research.google.com/ to start a new Python notebook. You will need to log in to use the Google Colab. You can use your university email or your personal.

The dataset used is the Titanic dataset, which is available via Seaborn. It contains the following columns:

- survived (int): 0 = No, 1 = Yes.
- pclass (int): Passenger class (1st, 2nd, 3rd).
- sex (str): Gender (male, female).
- age (float): Age in years.
- sibsp (int): Number of siblings/spouses aboard.
- parch (int): Number of parents/children aboard.
- fare (float): Passenger fare.
- embarked (str): Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton).
- deck (str): Deck information (A, B, C, etc.), many missing values.
- embark_town (str): Town of embarkation (Cherbourg, Southampton, Queenstown).
- alive (str): Alive or deceased (yes, no).

```
#import all the useful libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
```

Task1: Load titanic data from Seaborn

```
# Load the Titanic dataset from seaborn
df = sns.load_dataset('titanic')
```

Task2: Identify the missing values in columns

1. Use `df.isnull().sum()` to calculate the number of missing values in each column and print out only the columns that have missing values.

2. Can you identify the type of missingness present for the columns with missing values? Would you classify them as MCAR (Missing Completely at Random), MAR (Missing at Random), or MNAR (Missing Not at Random)?

Task3: Handle Missing Data:

1. Use fillna to fill in missing numeric data using the mean. Replace the ‘age’ with every other float data type in the data.

```
# Fill missing values for 'age'
df['age'] = df['age'].fillna(df['age'].mean())
```

2. Use the fillna to fill in missing categorical data using mode.

```
df['deck'] = df['deck'].fillna(df['deck'].mode())

b. Modify the code for every other categorical variable with missing values, and that will include for features such as: embarked, deck and embark_town
```

Task4: Transforming the distribution of non-normal numeric features

1. Investigate the distribution of the parch features

```
plt.figure(figsize=(12, 6))

# Histogram
plt.subplot(1, 2, 1)
sns.histplot(df['parch'], bins=30, kde=False, color='blue',
edgecolor='black')
plt.title('Histogram of parch')
plt.xlabel('parch')
plt.ylabel('Frequency')

# KDE Plot
plt.subplot(1, 2, 2)
sns.kdeplot(df['parch'], fill=True, color='orange')
plt.title('KDE Plot of parch')
plt.xlabel('parch')
plt.ylabel('Density')

plt.tight_layout()
plt.show()
```

2. We can see the distribution is skewed, use a log transformation to modify the parch's distribution to a normal distribution

```
# from scipy import stats # incase you haven't installed stats before
# Apply Box-Cox Transformation
# Box-Cox requires positive data, so we ensure 'parch' is > 0
# Since 'parch' is count data, we add a small constant
df['parch_shifted'] = df['parch'] + 1  # Shift by 1 to avoid zeros
df['parch_boxcox'], _ = stats.boxcox(df['parch_shifted'])

# Step 3: Visualize the distributions using KDE
plt.figure(figsize=(16, 6))

# KDE plot for original parch
plt.subplot(1, 2, 1)
sns.kdeplot(df['parch'], fill=True, color='blue', alpha=0.6)
plt.title('KDE of Original Parch Distribution')
plt.xlabel('Original Parch')
plt.ylabel('Density')
```

```
# KDE plot for Box-Cox transformed parch
plt.subplot(1, 2, 2)
sns.kdeplot(df['parch_boxcox'], fill=True, color='orange', alpha=0.6)
plt.title('KDE of Box-Cox Transformed Parch Distribution')
plt.xlabel('Box-Cox Transformed Parch')
plt.ylabel('Density')

plt.tight_layout()
plt.show()
```

Task5: Data Normalization (Scaling Numeric Data)

Normalisation scales numeric data to a particular range (e.g., between 0 and 1).

1. Use the MinMaxScaler() to transform Age and Fare (float types) into the range [0,1].

```
scaler = MinMaxScaler()

# Apply Min-Max Scaling to 'age' and 'fare'
df[['age_scaled', 'fare_scaled']] = scaler.fit_transform(df[['age',
'fare']])
```

Task6: Data Enrichment

1. Creating Family Size:
   A feature combining `SibSp` (siblings/spouses aboard) and `Parch` (parents/children aboard) to get the total number of family members on board.

```
df['family_size'] = df['sibsp'] + df['parch'] + 1 # Including the
passenger themselves
```

2. Create Fare per Person:
   Adjusts the Fare feature by dividing it by the number of family members travelling together, providing a more person-centric view of fare costs.

```
df['fare_per_person'] = df['fare'] / df['family_size']
```

3. Create Is_child:
   A passenger is considered "is_child" if they have age < 18 years

```
df['is_child'] = (df['age'] < 18).astype(int)  # 1 if child, 0
otherwise
```

Task7: Outlier detection

1. Investigate outliers in numerical features, i.e., age, fare, sibp and parch, family_size

```
numerical_features = ['age', 'fare', 'sibsp', 'parch', 'family_size']
# Calculate Q1 (25th percentile) and Q3 (75th percentile) for each
numerical feature
Q1 = df[numerical_features].quantile(0.25)
Q3 = df[numerical_features].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers_iqr = ((df[numerical_features] < lower_bound) |
(df[numerical_features] > upper_bound))

# --- Visualization: Box Plots ---
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(y=df[feature])
    plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()
```