Inconsistent Data

Practical 1&2 – W3

Learning Objectives

1. Inconsistent Data
   a. df.drop()
2. Encoding
   a. Encoding Nominal Categorical values using
      a. df.replace()
      b. df.cat.codes
   b. Encode Ordinal categorical values using one hot encoding.
3. Variance, Covariance, Correlation
   a. df.var()
   b. df.cov()
   c. df.corr()
4. Scaling
   a. Standard Scaling
   b. Min-Max Scaling

1. Inconsistent Data.

Task 1: Copy the jupyter notebook of week2 solution, available on Brightspace.

- Download “Week2_work.ipynb” from Brightspace and upload it to your Google Colab or open it in Jupyter.

Some rows need to be deleted depending upon some values in columns.

- Find unique values in columns ‘Payment_method’ and ‘Employees_status’.
- It is ok to have ‘Unemployed’ as category in Employee status and ‘Others’ in Payment method. But we don’t want to keep records of customers whose payment type is ‘others and employee status is ‘unemployed’.
- Drop rows where Employees_status is unemployment and Payment_method is other.

```
df[(df['Employees_status']=='Unemployment') &
(df['Payment_method']=='Other')].index
```

- The above code will return indexes of rows where Employees_status is unemployment and Payement_method is other.
- Pass it to the drop function to delete the rows.

```
df = df.drop(df[(df['Employees_status']=='Unemployment') &
(df['Payment_method']=='Other')].index)
```

Print df.shape before and after drop to find out if the rows were deleted.

2. Encoding

Task 2: Convert categorical data to numeric data

- Call df.info() function to find out categorical data. Data type 'Object' in pandas represent categorical data.
- Decide which categorical data is nominal and ordinal.

Nominal/Ordinal Data

- Find out the unique values in nominal data by calling df['your column name'].value_counts()
- There are three ways of convert categorical data to numeric.

1. **Method 1: Replace**

- Replace each unique value in nominal feature by a numeric value (manual encoding).

```
   df['Gender'] = df['Gender'].replace({'Female': 0, 'Male': 1})
```

- Just like Gender, replace the unique categories into numeric values of the following nominal features.
    - **Marital_status**
    - **Employees_status**
    - **Payment_method**

2. **Method 2: Cat.codes (Pandas library)**

- It is easier to use the replace function when the number of categories is small. If the number of categories is more than it is wise to use an automatic conversion using cat.codes.
- There are 50 different unique values in State_names nominal feature. Use cat.codes to convert states names to numeric values.

```
# Convert categorical data to numerical data using cat.codes
df['State_names'] = df['State_names'].astype('category')
df['State_names'] = df['State_names'].cat.codes
```

3. **Method 3: Ordinal Encoding (Scikitlearn library)**

```
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
oe = OrdinalEncoder()
df['State_names'] = oe.fit_transform(df[['State_names']])
```

Onehot Encoding

- Ordinal features have natural order in it. We typically prefer **OneHotEncoding** in situations where the categorical feature doesn't have any inherent order, and we want to prevent the model from interpreting the values as ordinal (having a ranking or magnitude). OneHotEncoding creates a binary column for each category, ensuring that the model treats each category independently.

a. Using Pandas:

```
# Convery ordinal categorical feature to numeric using dummy encoding
 # Get one hot encoding of columns 'Segment'
 one_hot = pd.get_dummies(df['Segment'])
 # Drop column 'Segment'as it is now encoded
 df = df.drop('Segment',axis = 1)
 # Join the encoded df
 df = df.join(one_hot)
```

b. Using Scikitlearn:

```
from sklearn.preprocessing import OneHotEncoder
# Initialize OneHotEncoder from sklearn
ohe = OneHotEncoder(sparse=False, drop='first')
# drop='first' to avoid the dummy variable trap (optional)

# Apply OneHotEncoder to the 'Segment' column
Ohe_coded = ohe.fit_transform(df[['Segment']])
# Convert the result into a DataFrame with proper column names
one_hot_df = pd.DataFrame(ohe_coded,
                  columns=ohe.get_feature_names_out(['Segment']))
# Drop the original 'Segment' column
df = df.drop('Segment', axis=1)
# Join the new one-hot encoded df back to the original
df = df.join(one_hot_df)
```

3. Variance, Covariance, Correlation

- Now we have removed missing values, inconsistent data and encoded categorical data to numeric data. It is time to find inter-relationships between features using variance, covariance, and correlation.

Task 1: Variance.

- Variance is a measure of how much the data for a variable varies from its mean.

```
df.var(numeric_only=None)#default option to include all variables

df.var(numeric_only=True) #include only numeric variables
```

Task 2: Covariance.

- Covariance is a measure of the relationship between 2 variables that is scale dependent, i.e., how much will a variable change when another variable changes.

```
df.cov()
```

Task 3: Correlation

- Correlation overcomes the lack of scale dependency that is present in covariance by standardizing the values. This standardization converts the values to the same scale.

```
df.corr()
```

Task 4: Correlation Heatmap visualization

```
# visualise correlation
import seaborn as sb
corr = df.corr().round(2)
sb.set (rc = {'figure.figsize':(12, 8)})
sb.heatmap(corr, cmap="Blues", annot=True)
```

4. Scaling

Task 1: Min-Max Scaling.

Scales all features between a given and value (e.g., 0 and 1)

```
from sklearn.preprocessing import MinMaxScaler

minmax_scaler = MinMaxScaler()

df_min_max = df.copy()

df_min_max =
pd.DataFrame(minmax_scaler.fit_transform(df_min_max.values),
columns=df_min_max.columns, index=df_min_max.index)

print("Scaled Dataset Using Min Max Scaler")
df_min_max.head()
```

Task 2: Standard Scaling.

- Per feature, subtract the mean value , scale by standard deviation.
- New feature has mean =0 and standard deviation=1, values can still be arbitrarily large.
- The new features have values between -1 and 1.

```
df_std = df.copy()
std_scaler = StandardScaler()

df_std = pd.DataFrame(std_scaler.fit_transform(df_std.values),
columns=df_std.columns, index=df_std.index)

print("Scaled Dataset Using StandardScaler")
df_std.head()
```