2441656-week-2

August 7, 2024

0.1 FEATURE ENGINEERING: Binning, Decomposition,Aggregation, Creation of Features

[20]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder
from category_encoders import BinaryEncoder, CountEncoder
```

[21]:
```
df = pd.read_excel('/kaggle/input/train-2441656/train.xlsx')
```

0.2 Tasks

Binning of features:

Bin the 'Age' column into different age groups (e.g.,child, adult, elderly).

Bin the 'Fare' column into different fare ranges (e.g., low, medium, high).

[22]:
```
# Bin the 'Age' column into different age groups (e.g.,child, adult, elderly).
print(df['Age'].min())
print(df['Age'].max())
bins = [0,17,59,80]
labels = ['child', 'adult','elderly']
pd.cut(df['Age'], bins=bins, labels = labels).dropna()
```

```
0.42
80.0
```

[22]:
```
0       adult
1       adult
2       adult
3       adult
4       adult
        …
885     adult
886     adult
887     adult
889     adult
890     adult
Name: Age, Length: 714, dtype: category
```

```
Categories (3, object): ['child' < 'adult' < 'elderly']
```

[23]:
```python
# Bin the 'Fare' column into different fare ranges (e.g., low, medium, high).
print(df['Fare'].min())
print(df['Fare'].max())
bins = [0]+list(df['Fare'].quantile([0.25,0.75]).values)+[df['Fare'].max()]
labels = ['Low', 'Medium','High']
pd.cut(df['Fare'], bins=bins, labels = labels).dropna()
```

```
0.0
512.3292
```

[23]:
```
0         Low
1        High
2      Medium
3        High
4      Medium
        …
886    Medium
887    Medium
888    Medium
889    Medium
890       Low
Name: Fare, Length: 876, dtype: category
Categories (3, object): ['Low' < 'Medium' < 'High']
```

[ ]:
```python

```

Aggregation of features:

Group the dataset by 'Pclass' and calculate the average 'Age' and 'Fare' for each class.

Group the dataset by 'Sex' and calculate the total number of passengers and the average 'Age' for each gender.

[24]:
```python
# Group the dataset by 'Pclass' and calculate the average 'Age' and 'Fare' for␣
↪each class.
print('Average age by Pclass:')
print(df.groupby('Pclass')['Age'].mean())
print('\n')
print('Average fare by Pclass:')
print(df.groupby('Pclass')['Fare'].mean())
```

```
Average age by Pclass:
Pclass
1    38.233441
2    29.877630
3    25.140620
Name: Age, dtype: float64
```

```
Average fare by Pclass:
Pclass
1    84.154687
2    20.662183
3    13.675550
Name: Fare, dtype: float64
```

[ ]:

Decomposing of features:

Decompose the 'Name' column into two new columns: 'Title' (extracted from the name prefix) and 'LastName' (extracted from the last name).

[25]:
```
df['Title'] = df['Name'].apply(lambda name : name.split('.')[0].split(',')[1])
df['Last Name'] = df['Name'].apply(lambda name : name.split('.')[0].
↪split(',')[0])
df[['Name','Title','Last Name']].dropna()
```

[25]:
```
                                                  Name  Title  Last Name
0                              Braund, Mr. Owen Harris     Mr     Braund
1    Cumings, Mrs. John Bradley (Florence Briggs Th…    Mrs    Cumings
2                               Heikkinen, Miss. Laina   Miss  Heikkinen
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)    Mrs   Futrelle
4                             Allen, Mr. William Henry     Mr      Allen
..                                                 …      …        …
886                              Montvila, Rev. Juozas    Rev   Montvila
887                       Graham, Miss. Margaret Edith   Miss     Graham
888           Johnston, Miss. Catherine Helen "Carrie"   Miss   Johnston
889                              Behr, Mr. Karl Howell     Mr       Behr
890                                Dooley, Mr. Patrick     Mr     Dooley

[891 rows x 3 columns]
```

[ ]:

Feature creation:

Create a new feature called 'FamilySize' by summing the 'SibSp' and 'Parch' columns

Create a new feature called 'IsAlone' to indicate whether a passenger is traveling alone or with family.

[26]:
```
df['FamilySize'] = df['SibSp']+df['Parch']
df['IsAlone'] = df['FamilySize'].apply(lambda size: 'No' if size >= 1 else␣
↪'Yes')
df[['Name','SibSp','Parch','FamilySize','IsAlone']]
```

[26]:

```
                                                  Name  SibSp  Parch  \
0                              Braund, Mr. Owen Harris      1      0
1    Cumings, Mrs. John Bradley (Florence Briggs Th…      1      0
2                               Heikkinen, Miss. Laina      0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)      1      0
4                             Allen, Mr. William Henry      0      0
..                                                 …    …    …
886                              Montvila, Rev. Juozas      0      0
887                       Graham, Miss. Margaret Edith      0      0
888           Johnston, Miss. Catherine Helen "Carrie"      1      2
889                              Behr, Mr. Karl Howell      0      0
890                                Dooley, Mr. Patrick      0      0

     FamilySize IsAlone
0             1      No
1             1      No
2             0     Yes
3             1      No
4             0     Yes
..          …     …
886           0     Yes
887           0     Yes
888           3      No
889           0     Yes
890           0     Yes

[891 rows x 5 columns]
```

[ ]:

Feature transformation:

Encode categorical features (e.g., 'Sex', 'Embarked') using appropriate techniques (e.g., one-hot encoding, label encoding). Mention the Top 5 Categorical Encoding Techniques and also list out the Major differences between them with the most suitable scenarios where we can use them.

[27]:

```
saved_econders={}
```

[28]:

```
saved_econders={}
df.dropna(subset=['Sex', 'Embarked'],inplace=True)
features = df[['Sex', 'Embarked']]

def OneHotEncoderFunction(features_to_encode):
    encoder = OneHotEncoder(sparse_output=False)
    encoder.fit(features_to_encode)
    saved_econders['OneHotEncoder_'+'_'.join(features_to_encode.columns)] =␣
  ↪encoder
```

```python
def LabelEncoderFunction(features_to_encode):
    list_encoder = []
    for col in features_to_encode.columns:
        encoder = LabelEncoder()
        encoder.fit(features_to_encode[col])
        saved_econders['LabelEncoder_'+ str(col)] = encoder

def OrdinalEncoderFunction(features_to_encode):
    categories = []
    for col in features_to_encode.columns:
        categories.append(sorted(list(features_to_encode[col].unique())))
    encoder = OrdinalEncoder(categories=categories)
    encoder.fit(features_to_encode)
    saved_econders['OrdinalEncoder_'+'_'.join(features_to_encode.columns)] =␣
 ↪encoder

def BinaryEncoderFunction(features_to_encode):
    encoder = BinaryEncoder(cols=features_to_encode.columns)
    encoder.fit(features_to_encode)
    saved_econders['BinaryEncoder_'+'_'.join(features_to_encode.columns)] =␣
 ↪encoder

def CountEncoderFunction(features_to_encode):
    encoder = CountEncoder(cols=features_to_encode.columns)
    encoder.fit(features_to_encode)
    saved_econders['CountEncoder_'+'_'.join(features_to_encode.columns)] =␣
 ↪encoder

encoder_functions =␣
 ↪[OneHotEncoderFunction,OrdinalEncoderFunction,BinaryEncoderFunction,CountEncoderFunction]

for fun in encoder_functions:
    fun(features)

saved_econders
```

[28]:
```
{'OneHotEncoder_Sex_Embarked': OneHotEncoder(sparse_output=False),
 'OrdinalEncoder_Sex_Embarked': OrdinalEncoder(categories=[['female', 'male'],
['C', 'Q', 'S']]),
 'BinaryEncoder_Sex_Embarked': BinaryEncoder(cols=Index(['Sex', 'Embarked'],
dtype='object'),
               mapping=[{'col': 'Sex',
                         'mapping':     Sex_0  Sex_1
 1      0      1
 2      1      0
-1      0      0
-2      0      0},
```

```
                         {'col': 'Embarked',
                          'mapping':     Embarked_0  Embarked_1
  1           0           1
  2           1           0
  3           1           1
 -1           0           0
 -2           0           0}]),
 'CountEncoder_Sex_Embarked': CountEncoder(cols=Index(['Sex', 'Embarked'],
dtype='object'),
              combine_min_nan_groups=True)}
```

[29]:

```
encoded_dfs = {}

for encoder_name, encoder in saved_econders.items():
    if 'OneHotEncoder' in encoder_name:
        transformed_data = encoder.transform(features)
        columns = encoder.get_feature_names_out(features.columns)
        encoded_df = pd.DataFrame(transformed_data, columns=columns)

    elif 'LabelEncoder' in encoder_name:
        column = encoder_name.split('_')[-1]
        transformed_data = encoder.transform(features[column])
        encoded_df = pd.DataFrame(transformed_data, columns=[column +␣
 ↪'_encoded'])

    elif 'OrdinalEncoder' in encoder_name:
        transformed_data = encoder.transform(features)
        columns = features.columns + '_ordinal'
        encoded_df = pd.DataFrame(transformed_data, columns=columns)

    elif 'BinaryEncoder' in encoder_name:
        transformed_data = encoder.transform(features)
        columns = transformed_data.columns
        encoded_df = pd.DataFrame(transformed_data, columns=columns)

    elif 'CountEncoder' in encoder_name:
        transformed_data = encoder.transform(features)
        columns = transformed_data.columns
        encoded_df = pd.DataFrame(transformed_data, columns=columns)

    # Storing the DataFrame in a dictionary for later use
    encoded_dfs[encoder_name] = encoded_df

# Displaying the DataFrames
for encoder_name, df in encoded_dfs.items():
    print(f"\n{encoder_name}:\n", df)
```

```
OneHotEncoder_Sex_Embarked:
      Sex_female  Sex_male  Embarked_C  Embarked_Q  Embarked_S
0            0.0       1.0         0.0         0.0         1.0
1            1.0       0.0         1.0         0.0         0.0
2            1.0       0.0         0.0         0.0         1.0
3            1.0       0.0         0.0         0.0         1.0
4            0.0       1.0         0.0         0.0         1.0
..           …         …           …           …           …
884          0.0       1.0         0.0         0.0         1.0
885          1.0       0.0         0.0         0.0         1.0
886          1.0       0.0         0.0         0.0         1.0
887          0.0       1.0         1.0         0.0         0.0
888          0.0       1.0         0.0         1.0         0.0

[889 rows x 5 columns]
```

```
OrdinalEncoder_Sex_Embarked:
      Sex_ordinal  Embarked_ordinal
0             1.0               2.0
1             0.0               0.0
2             0.0               2.0
3             0.0               2.0
4             1.0               2.0
..            …                 …
884           1.0               2.0
885           0.0               2.0
886           0.0               2.0
887           1.0               0.0
888           1.0               1.0

[889 rows x 2 columns]
```

```
BinaryEncoder_Sex_Embarked:
      Sex_0  Sex_1  Embarked_0  Embarked_1
0         0      1           0           1
1         1      0           1           0
2         1      0           0           1
3         1      0           0           1
4         0      1           0           1
..      …     …          …         …
886       0      1           0           1
887       1      0           0           1
888       1      0           0           1
889       0      1           1           0
890       0      1           1           1

[889 rows x 4 columns]
```

```
CountEncoder_Sex_Embarked:
     Sex  Embarked
0    577       644
1    312       168
2    312       644
3    312       644
4    577       644
..   …        …
886  577       644
887  312       644
888  312       644
889  577       168
890  577        77

[889 rows x 2 columns]
```

0.2.1 Mention the Top 5 Categorical Encoding

1. 'OneHotEncoder'
2. 'LabelEncoder'
3. 'OrdinalEncoder'
4. 'BinaryEncoder'
5. 'CountEncoder'

Major differences between them with the most suitable scenarios where we can use them.

1. 'OneHotEncoder' Converts each unique category level into a separate binary column (0/1). Prevents assumptions about ordinal relationships.Provides a complete representation of categories. Can increase dimensionality significantly with many unique categories, leading to sparse matrices.

   suitable scenarios Suitable for algorithms that don't assume order among categories (e.g., linear regression, neural networks) Nominal data without an inherent order (e.g., colors, gender).

2. 'LabelEncoder' Encodes categories as integers from 0 to n-1. Assumes ordinal relationship between categories, which may not be suitable for nominal data Simple and efficient. Maintains order for ordinal data.

   suitable scenarios Ordinal data where categories have a clear order. Suitable for algorithms that can handle numerical values directly (e.g., decision trees, random forests).

3. 'OrdinalEncoder' Encodes categories as integers based on a specified order. Maintains specified order for ordinal data.Suitable for models needing ordinal information. Assumes ordinal relationship, which may not apply to all datasets. Requires specifying category order.

   suitable scenarios Ordinal data where categories have a clear hierarchy or ranking (e.g., education levels, satisfaction ratings).

4. 'BinaryEncoder' Encodes categories into binary digits, reducing the number of columns compared to OneHotEncoder. Reduces dimensionality compared to OneHotEncoder.Less sparse

than OneHotEncoder for high-cardinality data. More complex to interpret compared to OneHotEncoder. Assumes no inherent order among categories.

suitable scenarios Nominal data with many categories, reducing dimensionality while preserving information. Suitable for large datasets where OneHotEncoding would be too sparse.

5. 'CountEncoder' Replaces categories with their corresponding frequency counts. Reduces dimensionality. Captures information about category frequency. May lose some categorical information.- Assumes categories with higher frequency are more important.

   suitable scenarios High-cardinality categorical data. Suitable for tree-based algorithms that handle numerical features well (e.g., decision trees, random forests).

[ ]:

```

```

2441655-week-1

August 7, 2024

1 Topic: Missing Values, Outliers, Categorical Data

1.1 Task 1

[38]:
```
# Import Pandas and alias it as pd.
# Import NumPy and alias it as np.

import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder,MinMaxScaler,StandardScaler
from scipy.stats import zscore
import matplotlib.pyplot as plt
import seaborn as sns
```

[39]:
```
# Load the Titanic dataset:
# Load the 'titanic.csv' file using the Pandas library and assign it to a␣
↪variable named 'data'.

data = pd.read_excel('/kaggle/input/train-16072024/train.xlsx')
```

[40]:
```
# Explore the dataset:

# Display the first 5 rows of the dataset.
# Display the number of rows and columns in the dataset.
# Display the summary statistics of the dataset.
```

[41]:
```
# Display the first 5 rows of the dataset.
data.head()
```

[41]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[42]:
```python
# Display the number of rows and columns in the dataset.
data.shape
```

[42]:
```
(891, 12)
```

[43]:
```python
# Display the summary statistics of the dataset.
data.describe(include='all')
```

[43]:
```
        PassengerId    Survived      Pclass                     Name   Sex  \
count    891.000000  891.000000  891.000000                      891   891
unique          NaN         NaN         NaN                      891     2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris  male
freq            NaN         NaN         NaN                        1   577
mean     446.000000    0.383838    2.308642                      NaN   NaN
std      257.353842    0.486592    0.836071                      NaN   NaN
min        1.000000    0.000000    1.000000                      NaN   NaN
25%      223.500000    0.000000    2.000000                      NaN   NaN
50%      446.000000    0.000000    3.000000                      NaN   NaN
75%      668.500000    1.000000    3.000000                      NaN   NaN
max      891.000000    1.000000    3.000000                      NaN   NaN

               Age       SibSp       Parch    Ticket        Fare    Cabin  \
count   714.000000  891.000000  891.000000     891.0  891.000000      204
unique         NaN         NaN         NaN     681.0         NaN      147
top            NaN         NaN         NaN  347082.0         NaN  B96 B98
freq           NaN         NaN         NaN       7.0         NaN        4
mean     29.699118    0.523008    0.381594       NaN   32.204208      NaN
std      14.526497    1.102743    0.806057       NaN   49.693429      NaN
min       0.420000    0.000000    0.000000       NaN    0.000000      NaN
25%      20.125000    0.000000    0.000000       NaN    7.910400      NaN
50%      28.000000    0.000000    0.000000       NaN   14.454200      NaN
75%      38.000000    1.000000    0.000000       NaN   31.000000      NaN
max      80.000000    8.000000    6.000000       NaN  512.329200      NaN
```

```
        Embarked
count        889
unique         3
top            S
freq         644
mean         NaN
std          NaN
min          NaN
25%          NaN
50%          NaN
75%          NaN
max          NaN
```

[44]:
```
# Handle missing values:

# Identify the columns with missing values.
# Impute missing values in the 'Age' column with the mean age of passengers.
# Impute missing values in the 'Embarked' column with the most frequent value.
# Drop the 'Cabin' column from the dataset
```

[45]:
```
# Identify the columns with missing values.
for col in data.columns:
    if data[col].isnull().sum() > 0:
        print(f"'{col}' column has {data[col].isnull().sum()} missing values")
```

```
'Age' column has 177 missing values
'Cabin' column has 687 missing values
'Embarked' column has 2 missing values
```

[46]:
```
# Impute missing values in the 'Age' column with the mean age of passengers.
# Impute missing values in the 'Embarked' column with the most frequent value.
data['Age'].fillna(data['Age'].mean(),inplace=True)
print(f"After Imputing missing values in the 'Age' column with the mean age of␣
 ↪passengers, 'Age' column has {data['Age'].isnull().sum()} missing values")
data['Embarked'].fillna(data['Embarked'].mode()[0],inplace=True)
print(f"After Imputing missing values in the 'Embarked' column with the most␣
 ↪frequent value, 'Embarked' column has {data['Embarked'].isnull().sum()}␣
 ↪missing values")
```

```
After Imputing missing values in the 'Age' column with the mean age of
passengers, 'Age' column has 0 missing values
After Imputing missing values in the 'Embarked' column with the most frequent
value, 'Embarked' column has 0 missing values
```

[47]:
```
# Drop the 'Cabin' column from the dataset
data.drop(columns=['Cabin'],inplace=True)
if 'Cabin' not in data.columns:
```

```
    print("Dropping the 'Cabin' column from the dataset is successful")
else:
    print("'Cabin' column is still present in the dataset")
```

```
Dropping the 'Cabin' column from the dataset is successful
```

1.2 Tasks 2

[48]:
```
# Handle categorical variables:
# Convert the 'Sex' column to numerical values,where 0 represents female and 1␣
 ↪represents male.
# Create dummy variables for the 'Embarked'column.

# Handle outliers:
# Identify the columns that may have outliers.
# Use appropriate techniques (e.g., Z-score, IQR) to identify and handle␣
 ↪outliers in the dataset.

# Data validation:
# Check for duplicate rows in the dataset.
# Remove any duplicate rows, if present.

# Data transformation:
# Create a new column called 'FamilySize' by
# summing the 'SibSp' and 'Parch' columns.
# Create a new column called 'Title' by extracting the
# titles from the 'Name' column.

# Data normalization:
# Normalize Data using appropriate scaling techniques (e.g., Min-Max scaling,␣
 ↪Standardization).
# Explain the Major Differences between Standardization and Normalization, and␣
 ↪When to
# use these Methods in real-world scenario based
# problems.
```

[49]:
```
# Handle categorical variables:
# Convert the 'Sex' column to numerical values,where 0 represents female and 1␣
 ↪represents male.
# Create dummy variables for the 'Embarked'column.

def label_ecoder(col):
    encoder = LabelEncoder()
    encoded_column = encoder.fit_transform(col)

    label_mapping = {label:index for index,label in enumerate(encoder.classes_)}
```

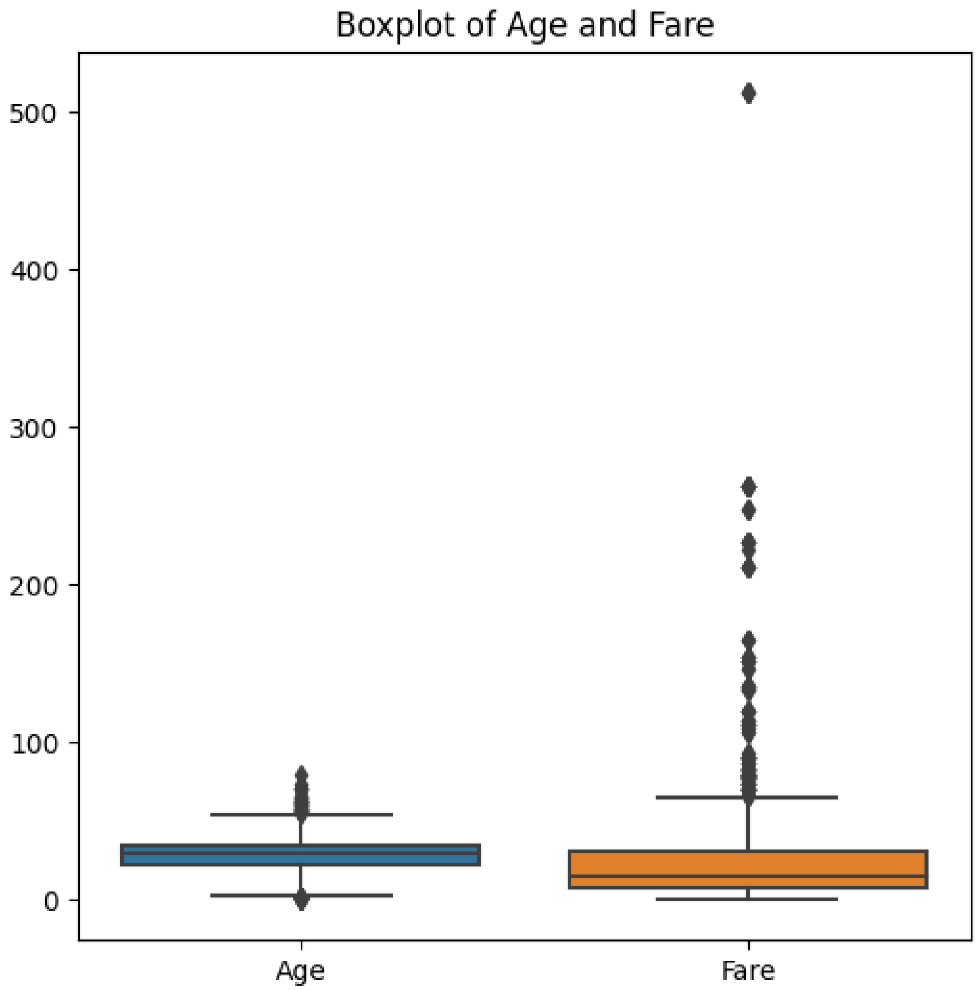
```
    print(f'label mapping of {col.name} column: {label_mapping}')

    return encoded_column,encoder


saved_encoders = {}

for col in ['Sex','Embarked']:
    encoded_column, encoder = label_ecoder(data[col])
    data[col] = encoded_column
    saved_encoders[col]=encoder
print(f'saved encoders: {saved_encoders}')
```

```
label mapping of Sex column: {'female': 0, 'male': 1}
label mapping of Embarked column: {'C': 0, 'Q': 1, 'S': 2}
saved encoders: {'Sex': LabelEncoder(), 'Embarked': LabelEncoder()}
```

[50]:
```
data.head()
```

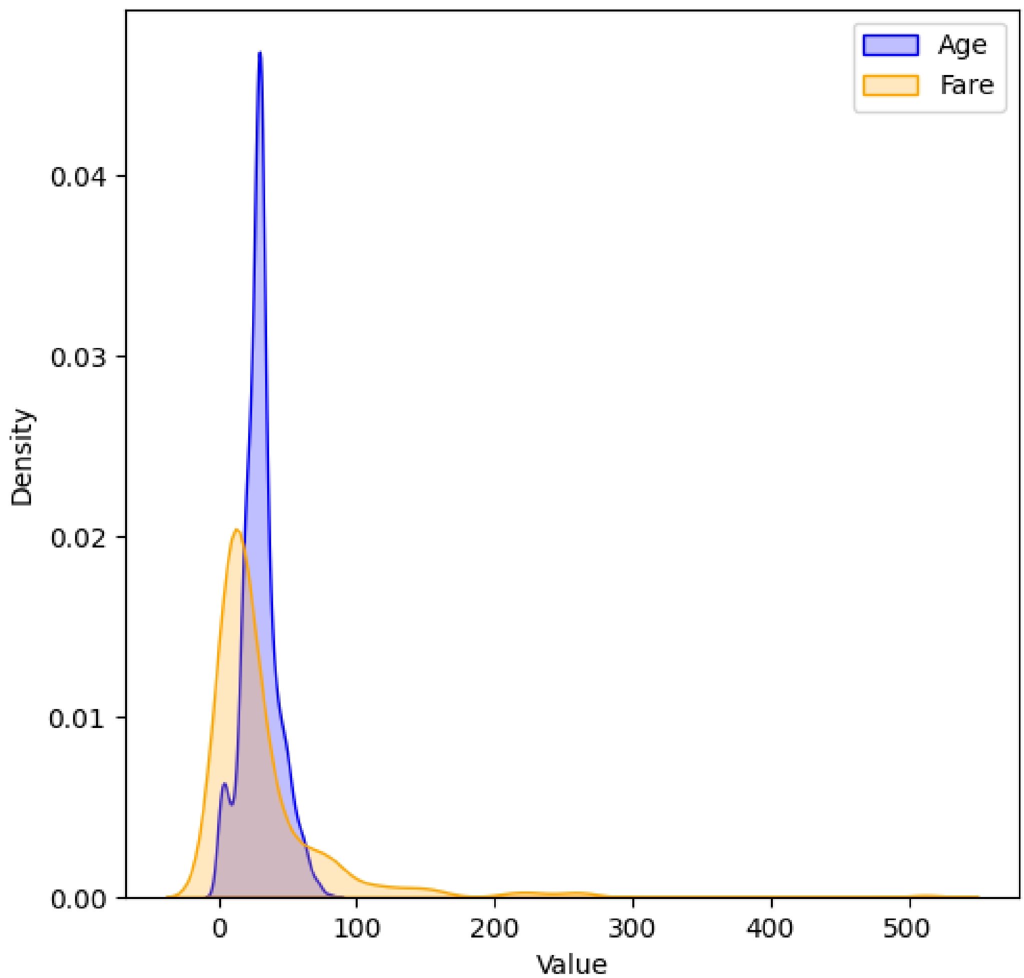
[50]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name  Sex   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris    1  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…    0  38.0      1      0
2                             Heikkinen, Miss. Laina    0  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    0  35.0      1      0
4                           Allen, Mr. William Henry    1  35.0      0      0

             Ticket     Fare  Embarked
0         A/5 21171   7.2500         2
1          PC 17599  71.2833         0
2  STON/O2. 3101282   7.9250         2
3            113803  53.1000         2
4            373450   8.0500         2
```

[51]:
```
# Handle outliers:
# Identify the columns that may have outliers.
# Use appropriate techniques (e.g., Z-score, IQR) to identify and handle␣
 ↪outliers in the dataset.
plt.figure(figsize=(6, 6))
sns.boxplot(data[['Age','Fare']])
plt.title('Boxplot of Age and Fare')
plt.show()
```



[52]:
```
plt.figure(figsize=(6, 6))
sns.kdeplot(data['Age'], shade=True, label='Age', color='blue')
sns.kdeplot(data['Fare'], shade=True, label='Fare', color='orange')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
```

[52]: <matplotlib.legend.Legend at 0x7d6f19e55b70>



[53]:
```
numerical_col = ['Age','Fare']
for col in numerical_col:
    outlier_count = ((zscore(data[col])>3) | (zscore(data[col])<-3)).sum()
    print(f"there are {outlier_count} outliers out of {data[col].count()} in␣
↪'{col}' columns using Z-score method")

print('\n')

for col in numerical_col:
    first_quartile = data[col].quantile(0.25)
    third_quartile = data[col].quantile(0.75)
    iqr = third_quartile - first_quartile
    outlier_count = ((data[col]<first_quartile-1.5*iqr) |␣
↪(data[col]>third_quartile+1.5*iqr)).sum()
    print(f"there are {outlier_count} outliers out of {data[col].count()} in␣
↪'{col}' columns using Z-score method")
```

```
there are 7 outliers out of 891 in 'Age' columns using Z-score method
there are 20 outliers out of 891 in 'Fare' columns using Z-score method


there are 66 outliers out of 891 in 'Age' columns using Z-score method
there are 116 outliers out of 891 in 'Fare' columns using Z-score method
```

[54]:
```
# handle outliers in the dataset
criteria_1 = (zscore(data['Age'])>3) | (zscore(data['Age'])<-3)
criteria_2 = (data['Fare']<first_quartile-1.5*iqr) |␣
 ↪(data['Fare']>third_quartile+1.5*iqr)
data = data[~(criteria_1 | criteria_2)]
data.reset_index(drop=True,inplace= True)
data
```

[54]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              3         1       3
2              4         1       1
3              5         0       3
4              6         0       3
..           ...       ...     ...
764          887         0       2
765          888         1       1
766          889         0       3
767          890         1       1
768          891         0       3

                                                Name  Sex        Age  SibSp  \
0                            Braund, Mr. Owen Harris    1  22.000000      1
1                             Heikkinen, Miss. Laina    0  26.000000      0
2     Futrelle, Mrs. Jacques Heath (Lily May Peel)    0  35.000000      1
3                           Allen, Mr. William Henry    1  35.000000      0
4                                   Moran, Mr. James    1  29.699118      0
..                                               ...  ...        ...    ...
764                            Montvila, Rev. Juozas    1  27.000000      0
765                     Graham, Miss. Margaret Edith    0  19.000000      0
766         Johnston, Miss. Catherine Helen "Carrie"    0  29.699118      1
767                            Behr, Mr. Karl Howell    1  26.000000      0
768                              Dooley, Mr. Patrick    1  32.000000      0

     Parch            Ticket     Fare  Embarked
0        0         A/5 21171   7.2500         2
1        0  STON/O2. 3101282   7.9250         2
2        0            113803  53.1000         2
3        0            373450   8.0500         2
4        0            330877   8.4583         1
```

```
..    …              …        …          …
764     0           211536  13.0000          2
765     0           112053  30.0000          2
766     2        W./C. 6607  23.4500          2
767     0           111369  30.0000          0
768     0           370376   7.7500          1

[769 rows x 11 columns]
```

[55]:
```
# Data validation:
# Check for duplicate rows in the dataset.
# Remove any duplicate rows, if present.

if data.duplicated().sum() == 0:
    print(f'there are {data.duplicated().sum()} duplicated rows in dataset')
else:
    print(f'there are {data.duplicated().sum()} duplicated rows in dataset')
    data.drop_duplicates()
    print(f'after dropping duplicates there are {data.duplicated().sum()}␣
 ↪duplicated rows in dataset')
```

```
there are 0 duplicated rows in dataset
```

[56]:
```
# Data transformation:
# Create a new column called 'FamilySize' by summing the 'SibSp' and 'Parch'␣
 ↪columns.
# Create a new column called 'Title' by extracting the titles from the 'Name'␣
 ↪column.

data['FamilySize'] = data['SibSp'] + data['Parch']
data['Title'] = data['Name'].apply(lambda name: name.split(',')[1].split('.
 ↪')[0])
data[['Title','FamilySize']]
```

[56]:
```
     Title  FamilySize
0       Mr           1
1     Miss           0
2      Mrs           1
3       Mr           0
4       Mr           0
..     …         …
764    Rev           0
765   Miss           0
766   Miss           3
767     Mr           0
768     Mr           0
```

[769 rows x 2 columns]

[57]:
```
# Data normalization:
# Normalize Data using appropriate scaling techniques (e.g., Min-Max scaling,␣
↪Standardization).


saved_scalers={}


scaler_standard = StandardScaler()
data['Age'] = scaler_standard.fit_transform(data[['Age']])
saved_scalers[data['Age'].name] = scaler_standard

scaler_minmax = MinMaxScaler()
data['Fare'] = scaler_minmax.fit_transform(data[['Fare']])
saved_scalers[data['Fare'].name] = scaler_minmax

print(saved_scalers)
print('Columns after scaling: ')
data[['Age','Fare']]
```

```
{'Age': StandardScaler(), 'Fare': MinMaxScaler()}
Columns after scaling:
```

[57]:

	Age	Fare
0	-0.553488	0.111538
1	-0.226142	0.121923
2	0.510386	0.816923
3	0.510386	0.123846
4	0.076581	0.130128
..	…	…
764	-0.144305	0.200000
765	-0.798997	0.461538
766	0.076581	0.360769
767	-0.226142	0.461538
768	0.264877	0.119231

[769 rows x 2 columns]

1.2.1 The Major Differences between Standardization and Normalization, and When to use these Methods in real-world scenario based problems.

Standardization:

```
This transforms the data to have a mean of 0 and a standard deviation of 1.
Centers the data around the mean, adjusting for the spread. It retains the original distributio
Useful when the data has varying scales and you want to retain the statistical properties
Often used in algorithms that assume normally distributed data
```

```
Suitable for algorithms that rely on the mean and standard deviation
```

Normalization:

```
This scales the data to a fixed range, typically [0, 1].
Rescales the data to a specific range, altering the distribution shape based on the range limi
Useful when you want to bound the feature values to a specific range, which can be critical fo
Effective for distance-based algorithms where the scale of the features matters
```

[58]:
```
# Data verification:

# Validate the data after cleaning by performing the following checks.

# 1. Verify if there are any missing values remaining in the dataset.

# 2. Verify if there are any outliers remaining in the dataset.

# 3. Verify if the categorical variables have been properly encoded.

# 4.Verify if the data has been properly normalized
```

[59]:
```
# 1. Verify if there are any missing values remaining in the dataset.
if data.isnull().sum().sum() == 0:
    print('There are no missing values remaining in the dataset')
else:
    print(f'There are {data.isnull().sum().sum()} missing values in the␣
 ↪dataset')
```

```
There are no missing values remaining in the dataset
```

[60]:
```
# 2. Verify if there are any outliers remaining in the dataset.
numerical_col = ['Age']
for col in numerical_col:
    outlier_count = ((zscore(data[col])>3) | (zscore(data[col])<-3)).sum()
    print(f"there are {outlier_count} outliers out of {data[col].count()} in␣
 ↪'{col}' columns using Z-score method")

print('\n')
numerical_col = ['Fare']
for col in numerical_col:
    first_quartile = data[col].quantile(0.25)
    third_quartile = data[col].quantile(0.75)
    iqr = third_quartile - first_quartile
    outlier_count = ((data[col]<first_quartile-1.5*iqr) |␣
 ↪(data[col]>third_quartile+1.5*iqr)).sum()
    print(f"there are {outlier_count} outliers out of {data[col].count()} in␣
 ↪'{col}' columns using Z-score method")
```

```
there are 1 outliers out of 769 in 'Age' columns using Z-score method
```

```
there are 25 outliers out of 769 in 'Fare' columns using Z-score method
```

[61]:
```python
# 3. Verify if the categorical variables have been properly encoded.
print(data['Sex'].unique())
print(data['Embarked'].unique())
```

```
[1 0]
[2 1 0]
```

[92]:
```python
# 4.Verify if the data has been properly normalized
if ((data['Fare']>=1) & (data['Fare']<=0)).sum() == 0:
    print(f'All Values lies in the range 0 and 1 for {data["Fare"].name} column.
 ↪ Hence the data has been properly normalized')
else:
    print('the data is not properly normalized')

#If the data is properly standarised:
if abs(data['Age'].mean()>1e-6) or abs(data['Age'].std(ddof=0)-1>1e-6):
    print(f'The data is not properly properly standarised for {data["Age"].
 ↪name} column')
else:
    print(f'The data is properly standarised for {data["Age"].name} column')
```

```
All Values lies in the range 0 and 1 for Fare column. Hence the data has been
properly normalized
The data is properly standarised for Age column
```