Data Preprocessing

February 20, 2025

1 Data Preprocessing Techniques using Pandas

STEPS IN DATA PROCESSING

Step 1:Import the necessary libraries

[ ]:
```
#importing libraries
import pandas as pd
import scipy
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Load the Dataset

[ ]:
```
#Load the Dataset
df=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML-DSE4/diabetes -␣
↪diabetes (1).csv')
print(df)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0        0  33.6
1              1     85.0           66.0           29.0        0  26.6
2              8    183.0           64.0            0.0        0  23.3
3              1     89.0           66.0           23.0       94  28.1
4              0    137.0           40.0           35.0      168  43.1
..           …        …            …            …       …    …
768            1     96.0           96.0           66.0        0  63.5
769           12      NaN            NaN           88.0        4  85.6
770           47     88.0           96.0            NaN       74  55.0
771            0    123.0           72.0            0.0        0  36.3
772            1    106.0           76.0            0.0        0  37.5

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
```

```
4                        2.288   33        1
..                         …   …      …
768                      0.368   47        1
769                      0.396  745        1
770                      0.396   14        1
771                      0.258   52        1
772                      0.197   26        0

[773 rows x 9 columns]
```

Check the data info

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 773 entries, 0 to 772
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               773 non-null    int64
 1   Glucose                   772 non-null    float64
 2   BloodPressure             772 non-null    float64
 3   SkinThickness             772 non-null    float64
 4   Insulin                   773 non-null    int64
 5   BMI                       773 non-null    float64
 6   DiabetesPedigreeFunction  773 non-null    float64
 7   Age                       773 non-null    int64
 8   Outcome                   773 non-null    int64
dtypes: float64(5), int64(4)
memory usage: 54.5 KB
```

We can also check the null values using df.null()

```
[ ]: df.isnull().sum()
```

```
[ ]: Pregnancies                 0
     Glucose                     1
     BloodPressure               1
     SkinThickness               1
     Insulin                     0
     BMI                         0
     DiabetesPedigreeFunction    0
     Age                         0
     Outcome                     0
     dtype: int64
```

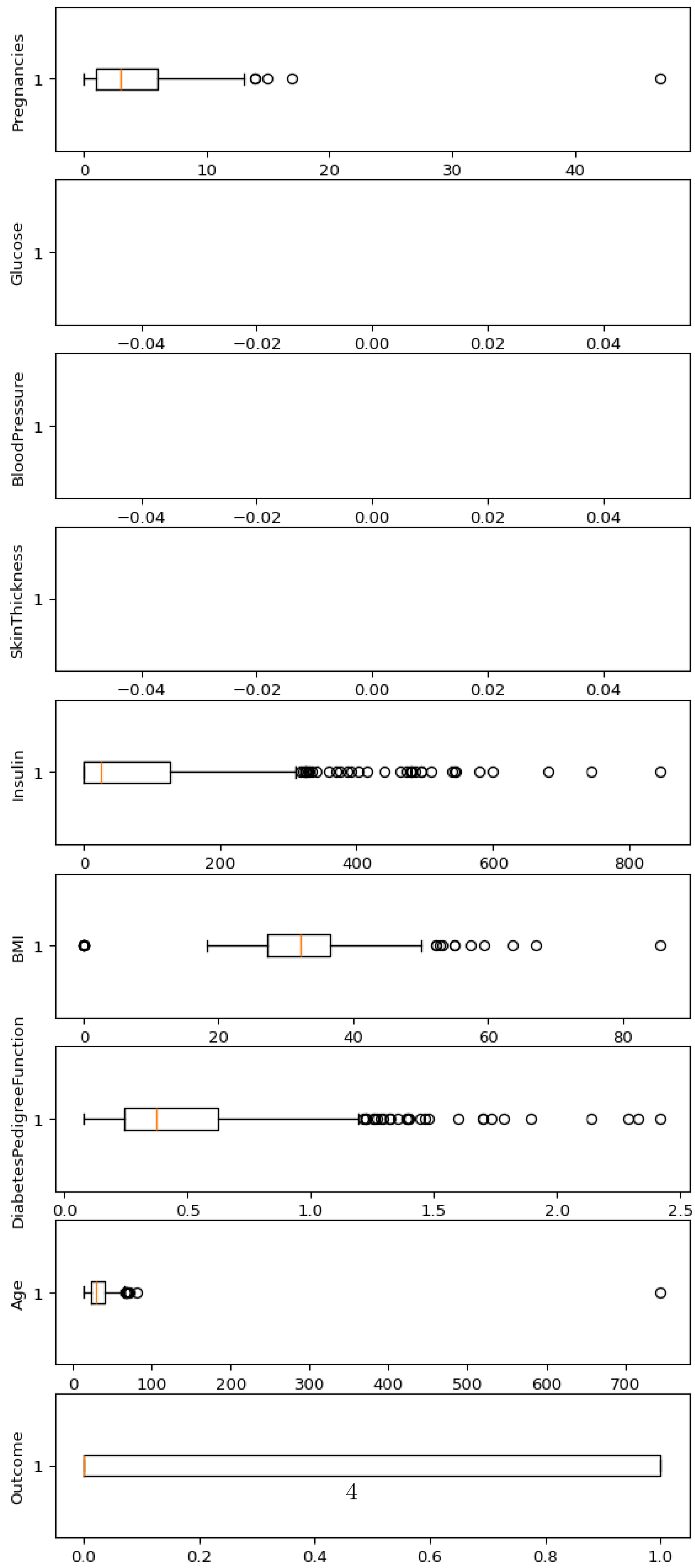
Step 3: Statistical Analysis

```
[ ]: df.describe()
```

[ ]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   773.000000  772.000000     772.000000     772.000000  773.000000
mean      3.899094  120.803109      69.187824      20.629534   79.384217
std       3.717019   31.928626      19.355764      16.211797  115.009658
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      63.500000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   25.000000
75%       6.000000  140.000000      80.000000      32.000000  126.000000
max      47.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  773.000000                773.000000  773.000000  773.000000
mean    32.145149                  0.470913   34.169470    0.351876
std      8.215319                  0.330534   28.178208    0.477865
min      0.000000                  0.078000   14.000000    0.000000
25%     27.300000                  0.244000   24.000000    0.000000
50%     32.100000                  0.371000   29.000000    0.000000
75%     36.600000                  0.624000   41.000000    1.000000
max     85.600000                  2.420000  745.000000    1.000000
```

Step 4: Check the Outliers

[ ]:
```python
#Box Plots
fig, axs= plt.subplots(9,1,dpi=95,figsize=(7,17))
i=0
for col in df.columns:
    axs[i].boxplot(df[col], vert=False)
    axs[i].set_ylabel(col)
    i+=1
plt.show()
```


Pregnancies
1
0 10 20 30 40
Glucose
1
−0.04 −0.02 0.00 0.02 0.04
BloodPressure
1
−0.04 −0.02 0.00 0.02 0.04
SkinThickness
1
−0.04 −0.02 0.00 0.02 0.04
Insulin
1
0 200 400 600 800
BMI
1
0 20 40 60 80
DiabetesPedigreeFunction
1
0.0 0.5 1.0 1.5 2.0 2.5
Age
1
0 100 200 300 400 500 600 700
Outcome
1
0.0 0.2 0.4 0.6 0.8 1.0

Step 5: Drop the Outliers

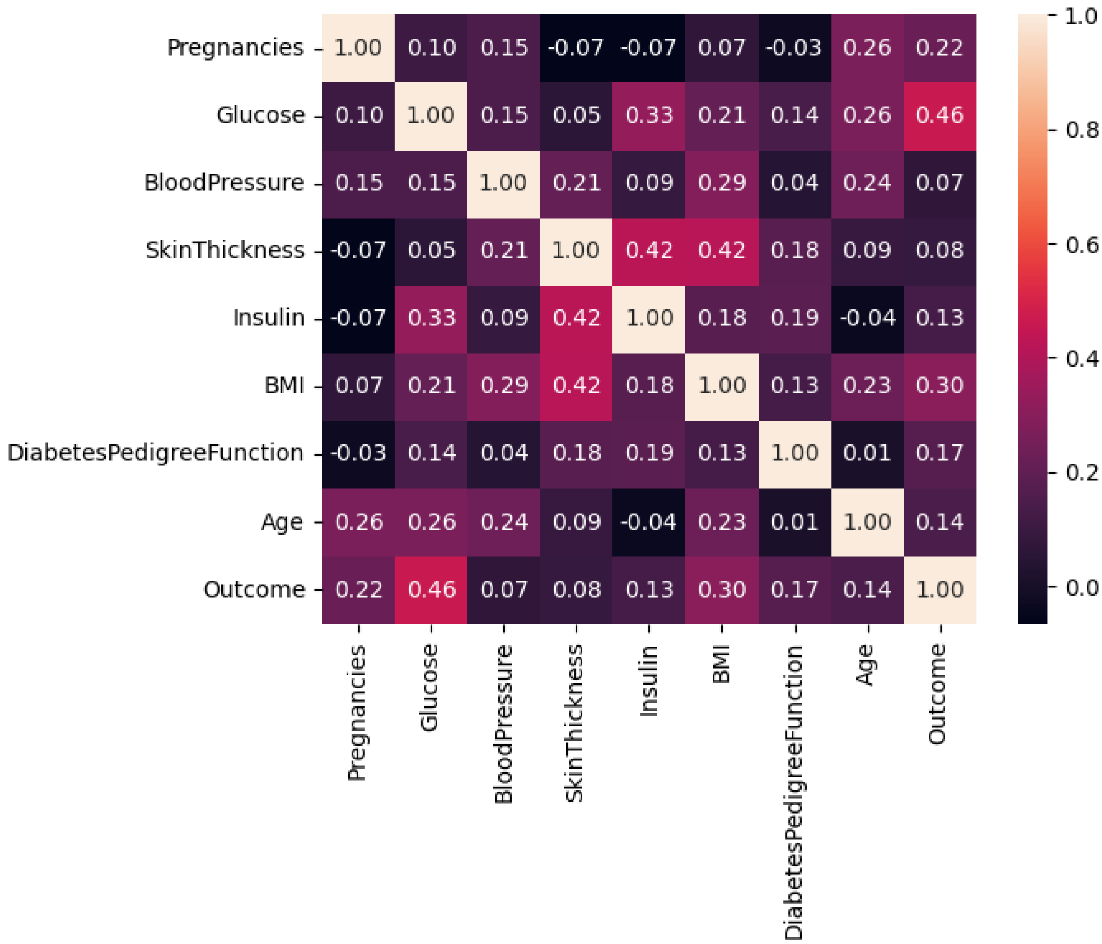
```
[ ]: # Identify the quartiles
q1, q3 = np.percentile(df['Insulin'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
clean_data = df[(df['Insulin'] >= lower_bound)
                & (df['Insulin'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['Pregnancies'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
clean_data = clean_data[(clean_data['Pregnancies'] >= lower_bound)
                        & (clean_data['Pregnancies'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['Age'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
clean_data = clean_data[(clean_data['Age'] >= lower_bound)
                        & (clean_data['Age'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['Glucose'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
```

```
# Drop the outliers
clean_data = clean_data[(clean_data['Glucose'] >= lower_bound)
                        & (clean_data['Glucose'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['BloodPressure'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (0.75 * iqr)
upper_bound = q3 + (0.75 * iqr)
# Drop the outliers
clean_data = clean_data[(clean_data['BloodPressure'] >= lower_bound)
                        & (clean_data['BloodPressure'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['BMI'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
clean_data = clean_data[(clean_data['BMI'] >= lower_bound)
                        & (clean_data['BMI'] <= upper_bound)]


# Identify the quartiles
q1, q3 = np.percentile(clean_data['DiabetesPedigreeFunction'], [25, 75])
# Calculate the interquartile range
iqr = q3 - q1
# Calculate the lower and upper bounds
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)

# Drop the outliers
clean_data = clean_data[(clean_data['DiabetesPedigreeFunction'] >= lower_bound)
                        & (clean_data['DiabetesPedigreeFunction'] <=
 ↪upper_bound)]
```

Step 5: Correlation

[ ]:
```
#correlation
corr = df.corr()
```

```
plt.figure(dpi=100)
sns.heatmap(df.corr(), annot=True, fmt= '.2f')
plt.show()
```



We can also compare by single columns in descending order

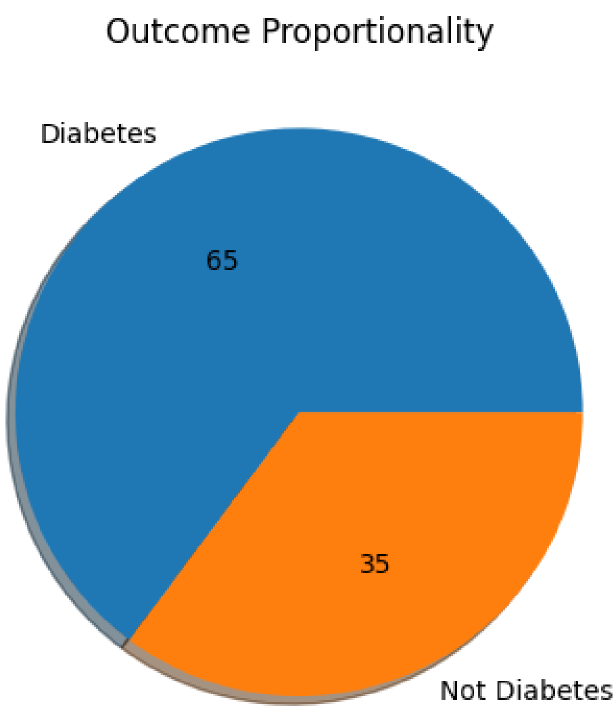
[ ]:
```
corr['Outcome'].sort_values(ascending = False)
```

[ ]:
```
Outcome                     1.000000
Glucose                     0.461537
BMI                         0.301947
Pregnancies                 0.221292
DiabetesPedigreeFunction    0.171100
Age                         0.144095
Insulin                     0.126696
SkinThickness               0.084325
BloodPressure               0.069464
```

```
Name: Outcome, dtype: float64
```

Step 6: Check the Outcomes Proportionality

```
[ ]: plt.pie(df.Outcome.value_counts(),
             labels= ['Diabetes', 'Not Diabetes'],
             autopct='%.f', shadow=True)
     plt.title('Outcome Proportionality')
     plt.show()
```



Step 7: Separate independent features and Target Variables

```
[ ]: # separate array into input and output components
     X = df.drop(columns =['Outcome'])
     Y = df.Outcome
```

Step 7: Normalization or Standardization

Normalization

1.Normalization works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.

2.Rescale your data using scikit-learn using the MinMaxScaler.

3.MinMaxScaler scales the data so that each feature is in the range [0, 1].

```
[ ]: # initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

# learning the statistical parameters for each of the data and transforming
rescaledX = scaler.fit_transform(X)
rescaledX[:5]
```

```
[ ]: array([[0.12765957, 0.74371859, 0.59016393, 0.35353535, 0.        ,
         0.39252336, 0.23441503, 0.04924761],
        [0.0212766 , 0.42713568, 0.54098361, 0.29292929, 0.        ,
         0.31074766, 0.11656704, 0.02325581],
        [0.17021277, 0.91959799, 0.52459016, 0.        , 0.        ,
         0.27219626, 0.25362938, 0.0246238 ],
        [0.0212766 , 0.44723618, 0.54098361, 0.23232323, 0.11111111,
         0.32827103, 0.03800171, 0.00957592],
        [0.        , 0.68844221, 0.32786885, 0.35353535, 0.19858156,
         0.50350467, 0.94363792, 0.02599179]])
```

Standardization

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

We can standardize data using scikit-learn with the StandardScaler class.

It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

```
[ ]: from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
rescaledX[:5]
```

```
[ ]: array([[ 0.56557845,  0.85235495,  0.14538301,  0.88699497, -0.69068648,
          0.1772047 ,  0.47253153,  0.56216421],
        [-0.78045647, -1.1220752 , -0.16480313,  0.51665418, -0.69068648,
         -0.67541369, -0.36302155, -0.11255229],
        [ 1.10399242,  1.94926059, -0.26819851, -1.2733263 , -0.69068648,
         -1.07736236,  0.60876301, -0.07704089],
        [-0.78045647, -0.99671455, -0.16480313,  0.14631339,  0.12716537,
         -0.49270975, -0.92005693, -0.46766623],
        [-1.04966346,  0.50761318, -1.50894306,  0.88699497,  0.77100619,
          1.33432966,  5.5009868 , -0.0415295 ]])
```

Hot Encoding in Data Preprocessing

[ ]:
```
dataset=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML-DSE4/
 ↪Project_College/Data.csv')
print(dataset)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

[ ]:
```
x=dataset.iloc[:,:-1].values
print(x)
dataset[['Country','Age']]
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

[ ]:
```
   Country   Age
0   France  44.0
1    Spain  27.0
2  Germany  30.0
3    Spain  38.0
4  Germany  40.0
5   France  35.0
6    Spain   NaN
7   France  48.0
8  Germany  50.0
9   France  37.0
```

[ ]:
```
y=dataset.iloc[:,3].values
print(y)
```

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

```
[ ]: from sklearn.preprocessing import LabelEncoder
label_encoder_x=LabelEncoder()
x[:,0]=label_encoder_x.fit_transform(x[:,0])
print(x)
```

```
[[0 44.0 72000.0]
 [2 27.0 48000.0]
 [1 30.0 54000.0]
 [2 38.0 61000.0]
 [1 40.0 nan]
 [0 35.0 58000.0]
 [2 nan 52000.0]
 [0 48.0 79000.0]
 [1 50.0 83000.0]
 [0 37.0 67000.0]]
```

```
[ ]: labelencoder_y=LabelEncoder()
y=labelencoder_y.fit_transform(y)
print(y)
```

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

```
[ ]: from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
[ ]: from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler()
x_train=sc_x.fit_transform(x_train)
x_test=sc_x.transform(x_test)

print(x_train)
print(x_test)
```

```
[[ 0.13483997  0.25315802         nan]
 [-0.94387981 -0.23014365  0.44897083]
 [ 1.21355975 -1.84114924 -1.41706417]
 [ 1.21355975         nan -1.0242147 ]
 [-0.94387981  1.54196248  1.62751925]
 [ 1.21355975 -0.0690431  -0.14030338]
 [-0.94387981  0.89756025  0.94003267]
 [-0.94387981 -0.55234477 -0.43494049]]
[[ 0.13483997 -1.35784756 -0.82778996]
 [ 0.13483997  1.8641636   2.02036872]]
```

```
[ ]:
```