Sardar Patel Institute of Technology,Mumbai

Department of Computer Science Engineering

B.E. Sem-VII- PE-IV (2024-2025)

IT 24 - AI in Healthcare

Experiment5: Data Transformation and preparation for analysis of healthcare data

Name: Adwait Purao **Date: 7/10/24**

Objective:

To understand and apply data transformation techniques for preparing healthcare data for analysis.

Link to Dataset :
https://www.kaggle.com/datasets/prasad22/healthcare-dataset/data

Link to Notebook:

https://colab.research.google.com/drive/1w8aKUO0BfcGsN6VhiqxGmCIDwM6mJlG3?usp=sharing

1.Introduction to Healthcare Data:

Dataset Information:

Each column provides specific information about the patient, their admission, and the healthcare services provided, making this dataset suitable for various data analysis and

modeling tasks in the healthcare domain. Here's a brief explanation of each column in the dataset -

- Name: This column represents the name of the patient associated with the healthcare record.
- Age: The age of the patient at the time of admission, expressed in years.
- Gender: Indicates the gender of the patient, either "Male" or "Female."
- Blood Type: The patient's blood type, which can be one of the common blood types (e.g., "A+", "O-", etc.).
- Medical Condition: This column specifies the primary medical condition or diagnosis associated with the patient, such as "Diabetes," "Hypertension," "Asthma," and more.
- Date of Admission: The date on which the patient was admitted to the healthcare facility.
- Doctor: The name of the doctor responsible for the patient's care during their admission.
- Hospital: Identifies the healthcare facility or hospital where the patient was admitted.
- Insurance Provider: This column indicates the patient's insurance provider, which can be one of several options, including "Aetna," "Blue Cross," "Cigna," "UnitedHealthcare," and "Medicare."
- Billing Amount: The amount of money billed for the patient's healthcare services during their admission. This is expressed as a floating-point number.
- Room Number: The room number where the patient was accommodated during their admission.
- Admission Type: Specifies the type of admission, which can be "Emergency," "Elective," or "Urgent," reflecting the circumstances of the admission.
- Discharge Date: The date on which the patient was discharged from the healthcare facility, based on the admission date and a random number of days within a realistic range.
- Medication: Identifies a medication prescribed or administered to the patient during their admission. Examples include "Aspirin," "Ibuprofen," "Penicillin," "Paracetamol," and "Lipitor."
- Test Results: Describes the results of a medical test conducted during the patient's admission. Possible values include "Normal," "Abnormal," or "Inconclusive," indicating the outcome of the test.

2.Understanding the Dataset

Dataset Description:

```
import pandas as pd


def display_step_header(step_name):

    print(f"\n{'='*20} {step_name} {'='*20}")
```

```python
def load_data(file_path):
    display_step_header("1. Data Loading")
    df = pd.read_csv(file_path)
    print("Dataset Overview:")
    print(df.info())
    print("\nSample of the dataset:")
    print(df.head())

    print("\nBasic statistics of numerical columns:")
    print(df.describe())

    # Display value counts for categorical columns
    categorical_columns = df.select_dtypes(include=['object']).columns
    for col in categorical_columns:
        print(f"\nValue counts for {col}:")
        print(df[col].value_counts().head())

    return df


df = load_data('./healthcare_dataset.csv')
```

```
==================== 1. Data Loading ====================
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Name                55500 non-null  object
 1   Age                 55500 non-null  int64
 2   Gender              55500 non-null  object
 3   Blood Type          55500 non-null  object
 4   Medical Condition   55500 non-null  object
 5   Date of Admission   55500 non-null  object
 6   Doctor              55500 non-null  object
 7   Hospital            55500 non-null  object
 8   Insurance Provider  55500 non-null  object
 9   Billing Amount      55500 non-null  float64
 10  Room Number         55500 non-null  int64
 11  Admission Type      55500 non-null  object
 12  Discharge Date      55500 non-null  object
 13  Medication          55500 non-null  object
 14  Test Results        55500 non-null  object
dtypes: float64(1), int64(2), object(12)
memory usage: 6.4+ MB
None

Sample of the dataset:
            Name  Age  Gender Blood Type Medical Condition Date of Admission  \
0  Bobby JacksOn   30    Male         B-            Cancer        2024-01-31
1   LesLie TErRy   62    Male         A+           Obesity        2019-08-20
2    DaNnY sMitH   76  Female         A-           Obesity        2022-09-22
3   andrEw waTtS   28  Female         O+          Diabetes        2020-11-18
4  adrIENNE bEll   43  Female        AB+            Cancer        2022-09-19

            Doctor                    Hospital Insurance Provider  \
0    Matthew Smith             Sons and Miller         Blue Cross
1  Samantha Davies                     Kim Inc           Medicare
2 Tiffany Mitchell                    Cook PLC              Aetna
3      Kevin Wells  Hernandez Rogers and Vang,           Medicare
4   Kathleen Hanna                 White-White              Aetna

   Billing Amount  Room Number Admission Type Discharge Date   Medication  \
0    18856.281306          328         Urgent     2024-02-02  Paracetamol
1    33643.327287          265      Emergency     2019-08-26    Ibuprofen
2    27955.096079          205      Emergency     2022-10-07      Aspirin
3    37909.782410          450       Elective     2020-12-18    Ibuprofen
4    14238.317814          458         Urgent     2022-10-09   Penicillin

   Test Results
0        Normal
1  Inconclusive
2        Normal
3      Abnormal
4      Abnormal

Basic statistics of numerical columns:
                Age  Billing Amount   Room Number
count  55500.000000    55500.000000  55500.000000
```

```
Basic statistics of numerical columns:
                Age  Billing Amount   Room Number
count  55500.000000    55500.000000  55500.000000
mean      51.539459    25539.316097    301.134829
std       19.602454    14211.454431    115.243069
min       13.000000    -2008.492140    101.000000
25%       35.000000    13241.224652    202.000000
50%       52.000000    25538.069376    302.000000
75%       68.000000    37820.508436    401.000000
max       89.000000    52764.276736    500.000000

Value counts for Name:
Name
DAvId muNoZ        3
SOnYa aDams        2
terRY gONZaLeZ     2
JaCKsON BARbeR     2
doNALD aViLA       2
Name: count, dtype: int64

Value counts for Gender:
Gender
Male      27774
Female    27726
Name: count, dtype: int64

Value counts for Blood Type:
Blood Type
A-     6969
A+     6956
AB+    6947
AB-    6945
B+     6945
Name: count, dtype: int64

Value counts for Medical Condition:
Medical Condition
Arthritis       9308
Diabetes        9304
Hypertension    9245
Obesity         9231
Cancer          9227
Name: count, dtype: int64

Value counts for Date of Admission:
Date of Admission
2024-03-16    50
2022-07-24    49
2020-10-22    49
2021-12-28    48
2021-01-03    48
Name: count, dtype: int64

Value counts for Doctor:
Doctor
Michael Smith      27
Robert Smith       22
John Smith         22
Michael Johnson    20
```

```
Value counts for Doctor:
Doctor
Michael Smith      27
Robert Smith       22
John Smith         22
Michael Johnson    20
James Smith        20
Name: count, dtype: int64

Value counts for Hospital:
Hospital
LLC Smith      44
Ltd Smith      39
Johnson PLC    38
Smith Ltd      37
Smith PLC      36
Name: count, dtype: int64

Value counts for Insurance Provider:
Insurance Provider
Cigna               11249
Medicare            11154
UnitedHealthcare    11125
Blue Cross          11059
Aetna               10913
Name: count, dtype: int64

Value counts for Admission Type:
Admission Type
Elective     18655
Urgent       18576
Emergency    18269
Name: count, dtype: int64

Value counts for Discharge Date:
Discharge Date
2020-03-15    53
2021-12-13    51
2020-12-02    51
2023-04-29    51
2020-08-11    50
Name: count, dtype: int64

Value counts for Medication:
Medication
Lipitor        11140
Ibuprofen      11127
Aspirin        11094
Paracetamol    11071
Penicillin     11068
Name: count, dtype: int64

Value counts for Test Results:
Test Results
Abnormal        18627
Normal          18517
Inconclusive    18356
Name: count, dtype: int64
```

3.Handling Missing data in the dataset:

Methods of Handling Missing data

```
def handle_missing_data(df):
    display_step_header("2. Missing Data Handling")
    print("Missing values before handling:")
    print(df.isnull().sum())

    # For numerical columns, fill with median
    numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
    for col in numeric_columns:
        df[col].fillna(df[col].median(), inplace=True)

    # For categorical columns, fill with mode
    categorical_columns = df.select_dtypes(include=['object']).columns
    for col in categorical_columns:
        df[col].fillna(df[col].mode()[0], inplace=True)

    print("\nMissing values after handling:")
    print(df.isnull().sum())
    return df

df = handle_missing_data(df)
```

```
=================== 2. Missing Data Handling ===================
Missing values before handling:
Name                  0
Age                   0
Gender                0
Blood Type            0
Medical Condition     0
Date of Admission     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Discharge Date        0
Medication            0
Test Results          0
dtype: int64
<ipython-input-12-91a262dc2575>:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-12-91a262dc2575>:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].mode()[0], inplace=True)

Missing values after handling:
Name                  0
Age                   0
Gender                0
Blood Type            0
Medical Condition     0
Date of Admission     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Discharge Date        0
Medication            0
Test Results          0
dtype: int64
```

4.Categorical Data Encoding

```
from sklearn.preprocessing import LabelEncoder


def encode_categorical_data(df):
    display_step_header("3. Categorical Data Encoding")
    label_encoders = {}
    categorical_columns = df.select_dtypes(include=['object']).columns

    # Create a sample dataframe to show transformations
    sample_df = pd.DataFrame()

    for col in categorical_columns:
        label_encoders[col] = LabelEncoder()
        df[f'{col}_encoded'] = label_encoders[col].fit_transform(df[col])
```

```
        # Get unique categories and their encodings
        unique_categories = df[col].unique()[:3]  # Take only first 3 unique values
        unique_encodings = [label_encoders[col].transform([cat])[0] for cat in unique_categories]

        # Ensure sample_df has the correct length
        sample_df = pd.DataFrame({col: unique_categories})  # Create a DataFrame with the correct length

        sample_df[f'{col}_encoded'] = unique_encodings

        print(f"\nEncoding example for {col} (showing first 3 categories):")
        for orig, enc in zip(unique_categories, unique_encodings):
            print(f"{orig} -> {enc}")

    print("\nSample of original and encoded data:")
    print(sample_df.to_string(index=False))

    return df, label_encoders

df, label_encoders = encode_categorical_data(df)
```

```
=================== 3. Categorical Data Encoding ===================

Encoding example for Name (showing first 3 categories):
Bobby JacksOn -> 3068
LesLie TErRy -> 15211
DaNnY sMitH -> 6476

Encoding example for Gender (showing first 3 categories):
Male -> 1
Female -> 0

Encoding example for Blood Type (showing first 3 categories):
B- -> 5
A+ -> 0
A- -> 1

Encoding example for Medical Condition (showing first 3 categories):
Cancer -> 2
Obesity -> 5
Diabetes -> 3

Encoding example for Date of Admission (showing first 3 categories):
2024-01-31 -> 1729
2019-08-20 -> 104
2022-09-22 -> 1233

Encoding example for Doctor (showing first 3 categories):
Matthew Smith -> 26612
Samantha Davies -> 33648
Tiffany Mitchell -> 37828

Encoding example for Hospital (showing first 3 categories):
Sons and Miller -> 29933
Kim Inc -> 16012
Cook PLC -> 5473

Encoding example for Insurance Provider (showing first 3 categories):
Blue Cross -> 1
Medicare -> 3
Aetna -> 0

Encoding example for Admission Type (showing first 3 categories):
Urgent -> 2
Emergency -> 1
Elective -> 0

Encoding example for Discharge Date (showing first 3 categories):
2024-02-02 -> 1730
2019-08-26 -> 109
2022-10-07 -> 1247

Encoding example for Medication (showing first 3 categories):
Paracetamol -> 3
Ibuprofen -> 1
Aspirin -> 0

Encoding example for Test Results (showing first 3 categories):
Normal -> 2
Inconclusive -> 1
```

```
Encoding example for Test Results (showing first 3 categories):
Normal -> 2
Inconclusive -> 1
Abnormal -> 0

Sample of original and encoded data:
Test Results  Test Results_encoded
      Normal                     2
Inconclusive                     1
    Abnormal                     0
```

5. Outlier Detection and Treatment

```
def handle_outliers(df, numeric_columns):

    display_step_header("4. Outlier Detection and Treatment")


    for col in numeric_columns:

        Q1 = df[col].quantile(0.25)

        Q3 = df[col].quantile(0.75)

        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR

        upper_bound = Q3 + 1.5 * IQR


        # Print outlier information

        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

        print(f"\nOutliers in {col}:")

        print(f"Number of outliers: {len(outliers)}")

        print(f"Percentage                 of                 outliers:
{(len(outliers)/len(df))*100:.2f}%")

        print(f"Lower bound: {lower_bound:.2f}")

        print(f"Upper bound: {upper_bound:.2f}")


        # Cap the outliers
```

```
        df[f'{col}_cleaned']       =       df[col].clip(lower=lower_bound, upper=upper_bound)


        # Display statistics before and after

        print(f"\nStatistics for {col} before and after outlier treatment:")

        print(pd.DataFrame({

            'Original': df[col].describe(),

            'Cleaned': df[f'{col}_cleaned'].describe()

        }))


    return df


numeric_columns = ['Age', 'Billing Amount']

df = handle_outliers(df, numeric_columns)
```

```
==================== 4. Outlier Detection and Treatment ====================

Outliers in Age:
Number of outliers: 0
Percentage of outliers: 0.00%
Lower bound: -14.50
Upper bound: 117.50

Statistics for Age before and after outlier treatment:
           Original       Cleaned
count  55500.000000  55500.000000
mean      51.539459     51.539459
std       19.602454     19.602454
min       13.000000     13.000000
25%       35.000000     35.000000
50%       52.000000     52.000000
75%       68.000000     68.000000
max       89.000000     89.000000

Outliers in Billing Amount:
Number of outliers: 0
Percentage of outliers: 0.00%
Lower bound: -23627.70
Upper bound: 74689.43

Statistics for Billing Amount before and after outlier treatment:
           Original       Cleaned
count  55500.000000  55500.000000
mean   25539.316097  25539.316097
std    14211.454431  14211.454431
min    -2008.492140  -2008.492140
25%    13241.224652  13241.224652
50%    25538.069376  25538.069376
75%    37820.508436  37820.508436
max    52764.276736  52764.276736
```

6. Feature Scaling and Normalization

```
from sklearn.preprocessing import StandardScaler


def scale_features(df, numeric_columns):

    display_step_header("5. Feature Scaling")

    scaler = StandardScaler()
```

```
    scaled_columns = [col + '_scaled' for col in numeric_columns]
    df[scaled_columns] = scaler.fit_transform(df[numeric_columns])

    for original, scaled in zip(numeric_columns, scaled_columns):
        print(f"\nScaling results for {original}:")
        print(pd.DataFrame({
            'Original': df[original].describe(),
            'Scaled': df[scaled].describe()
        }))

    return df, scaler

df, scaler = scale_features(df, numeric_columns)
```

```
==================== 5. Feature Scaling ====================

Scaling results for Age:
           Original        Scaled
count  55500.000000  5.550000e+04
mean      51.539459  7.732753e-17
std       19.602454  1.000009e+00
min       13.000000 -1.966071e+00
25%       35.000000 -8.437519e-01
50%       52.000000  2.349424e-02
75%       68.000000  8.397259e-01
max       89.000000  1.911030e+00

Scaling results for Billing Amount:
           Original        Scaled
count  55500.000000  5.550000e+04
mean   25539.316097  5.703546e-17
std    14211.454431  1.000009e+00
min    -2008.492140 -1.938440e+00
25%    13241.224652 -8.653725e-01
50%    25538.069376 -8.772730e-05
75%    37820.508436  8.641834e-01
max    52764.276736  1.915723e+00
```

7. Feature Engineering

- Create new features (e.g., BMI from weight and height).

```
def engineer_features(df):

    display_step_header("6. Feature Engineering")



    # Age groups

    df['Age_Group'] = pd.cut(df['Age'], bins=[0, 18, 35, 50, 65,
100],

                             labels=['0-18', '19-35', '36-50', '51-
65', '65+'])

    print("\nAge Group Distribution:")

    print(df['Age_Group'].value_counts())
```

```python
    # Chronic condition flag
    df['Is_Chronic'] = df['Medical Condition'].apply(lambda x: 1 if
'chronic' in str(x).lower() else 0)
    print("\nChronic Condition Distribution:")
    print(df['Is_Chronic'].value_counts(normalize=True))


    # Cost per age
    df['Cost_Per_Age'] = df['Billing Amount'] / df['Age']
    print("\nCost Per Age Statistics:")
    print(df['Cost_Per_Age'].describe())


    return df


df = engineer_features(df)
```

```
==================== 6. Feature Engineering ====================

Age Group Distribution:
Age_Group
65+      16250
19-35    13644
51-65    12417
36-50    12301
0-18       888
Name: count, dtype: int64

Chronic Condition Distribution:
Is_Chronic
0    1.0
Name: proportion, dtype: float64

Cost Per Age Statistics:
count    55500.000000
mean       596.195820
std        467.843388
min        -49.140205
25%        257.961738
50%        495.787257
75%        787.818365
max       3886.670220
Name: Cost_Per_Age, dtype: float64
```

8. Data Preparation for Machine Learning Models:

- Train-Test Split

```
from sklearn.model_selection import train_test_split


def prepare_for_ml(df, target_column='Billing Amount'):

    display_step_header("7. Preparing Data for Machine Learning")


    numeric_features = df.select_dtypes(include=['int64', 'float64']).columns

    categorical_features = [col for col in df.columns if '_encoded' in col]
```

```
    features = list(numeric_features) + categorical_features
    features = [f for f in features if f != target_column]

    X = df[features]
    y = df[target_column]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    print("Selected features:")
    for f in features:
        print(f"- {f}")

    print(f"\nTarget variable statistics:")
    print(y.describe())

    print("\nData split sizes:")
    print(f"Training set: {X_train.shape}")
    print(f"Test set: {X_test.shape}")

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = prepare_for_ml(df)
```

```
==================== 7. Preparing Data for Machine Learning ====================
Selected features:
- Age
- Room Number
- Name_encoded
- Gender_encoded
- Blood Type_encoded
- Medical Condition_encoded
- Date of Admission_encoded
- Doctor_encoded
- Hospital_encoded
- Insurance Provider_encoded
- Admission Type_encoded
- Discharge Date_encoded
- Medication_encoded
- Test Results_encoded
- Age_cleaned
- Billing Amount_cleaned
- Age_scaled
- Billing Amount_scaled
- Is_Chronic
- Cost_Per_Age
- Name_encoded
- Gender_encoded
- Blood Type_encoded
- Medical Condition_encoded
- Date of Admission_encoded
- Doctor_encoded
- Hospital_encoded
- Insurance Provider_encoded
- Admission Type_encoded
- Discharge Date_encoded
- Medication_encoded
- Test Results_encoded

Target variable statistics:
count    55500.000000
mean     25539.316097
std      14211.454431
min      -2008.492140
25%      13241.224652
50%      25538.069376
75%      37820.508436
max      52764.276736
Name: Billing Amount, dtype: float64

Data split sizes:
Training set: (44400, 32)
Test set: (11100, 32)
```

Conclusion:

In this assignment, I learned how to effectively handle real-world data by applying essential preprocessing steps such as handling missing values, encoding categorical variables, detecting and treating outliers, scaling features, and engineering new features. I also gained insights into how these steps impact the data's quality and readiness for machine learning. Through each step, I enhanced my understanding of preparing datasets for analysis and model building, ensuring the data is clean, consistent, and well-structured for successful predictions.