

Devanshu Surana

PC-23, Panel C, Batch C1
1032210755

ML Lab Assignment

Problem statement: Implement various pre-processing techniques on given dataset.

- Objectives:
1. To learn python programming with different modules/libraries.
 2. To understand the concept of exploratory data analysis.

Operations to be performed on dataset:

1. Importing Python modules/libraries
2. Importing data
3. Displaying data
4. Creating the independent and Dependent variables
5. Replacing missing value with meaningful value
6. Encoding Categorical data
7. Splitting the data into training and test set.
8. Doing feature scaling on data
9. Use any 3-4 graphs/plots.

FAQ's

Q1) List two common libraries for data manipulation.

Give an example for each library.

→ The two common libraries are:

1. Pandas: It is a powerful library for data manipulation and analysis. It provides data structures and functions for efficiently handling structured data.

2. NumPy: It is a fundamental package for numerical computing with python. It provides support for arrays, matrices, and mathematical functions.

Q2) Give an example on how ordinal data is handled in a machine learning algorithm.

→ 1) Use 'ordinal encoder' from sci-kit learn to convert ordinal categories into numerical values while preserving their order.

model.fit(x-train, y-train)

predictions = model.predict(x-test)

accuracy = accuracy-score(y-test, predictions)

print('Accuracy', accuracy)

Q3) Can one hot encoding be used for continuous data.

If yes, give an example.

→ One hot encoding is typically used for categorical data, where each category is represented by a binary vector. It is not typically used for continuous data because

continuous data implies a range of values rather than distinct categories.

Ex: import pandas as pd
 from sklearn.preprocessing import OneHotEncoder,
 KBinsDiscretizer

data = pd.DataFrame({'age': [22, 35, 42, 31, 28]})

discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal',
 strategy='uniform')

data_discretized = discretizer.fit_transform(data)

encoder = OneHotEncoder()

data_encoded = encoder.fit_transform(data_discretized)

print(data_encoded.toarray())

In this example:

- 1) We have a continuous variable 'age'.
- 2) We discretize the age values into three categories using 'KBins Discretizer'.
- 3) Then, we apply one hot encoding to the discretized data using 'One Hot Encoder'.
- 4) Finally, we print the one hot encoded array.

Q4) Why is it necessary to encode strings?

→ Encoding strings is necessary for various reasons such as ensuring data integrity during transmission, storage efficiency and facilitating compatibility between different systems.

Q5) State the significance of exploring data analysis.

→ 1) Understand data distribution

2) Detect Anomalies

3) Feature Selection

4) Data Cleaning

5) Visual Representation

6) Statistical Summary

Q6) Handling missing values of data is an important step in Data preprocessing.

→ It is crucial in data preprocessing as it ensures that the dataset is clean and accurate, preventing potential biases or errors in subsequent analysis or machine learning models.

Q7) State any 4 graphical techniques /plots used for exploratory data analysis.

→ 1) Histogram: Visualize the distribution of a single variable by dividing the data into bins and displaying the frequency of observation in each bin.

2) Scatter plots: Display the relationship between two continuous variables, helps to identify patterns and trends.

3) Box plots: Provides a summary of distribution of a dataset, including the minimum, first quartile, median, third quartile and maximum values.

u) Heatmaps: Represent the correlation matrix of variables in a color-coded format.

Q8) Describe the box and whisker plot.

→ A box-and-whisker plot, also known as box plot, is a graphical representation that displays the distribution of a dataset.

i) Box: The box in the middle represent the interquartile range (IQR), which is the range between the first quartile and third quartile.

2) Whisker: The line or whisker extend from box to minimum and maximum values with a certain range, often defined by a multiplier of the IQR.

Q9) Explain Central Tendency functions.

→ Central Tendency functions summarize the central or typical value of a dataset. Here are three common measures of central tendency:

1) Mean: avg value of dataset calculated by summing all values and by dividing by no. of observations.

2) Median: middle value of dataset when arranged in ascending order.

3) Mode: most frequent value of dataset.

Conclusion: Data collection, data preparation, handling various data types was studied, and EDA was performed.

• Aritmik adatokhoz alkalmazható módszerek (A statisztikai módszerekkel kapcsolatosan) (A statisztikai módszerekkel kapcsolatosan)

• Kategórikus adatokhoz alkalmazható módszerek (A kategórikus adatokhoz alkalmazható módszerekkel kapcsolatosan)

• Csoportosított adatokhoz alkalmazható módszerek (A csoportosított adatokhoz alkalmazható módszerekkel kapcsolatosan)

• Csatlakoztatott adatokhoz alkalmazható módszerek (A csatlakoztatott adatokhoz alkalmazható módszerekkel kapcsolatosan)

• Csatlakoztatott adatokhoz alkalmazható módszerek (A csatlakoztatott adatokhoz alkalmazható módszerekkel kapcsolatosan)

• Csatlakoztatott adatokhoz alkalmazható módszerek (A csatlakoztatott adatokhoz alkalmazható módszerekkel kapcsolatosan)

```

# importing libraries
import numpy as np
import pandas as pd

Data = {'Name': ['Alice', 'Bob', 'John', 'Lisa'],
        'Age': [21, 20, 22, 23],
        'City': ['San francisco', 'New year', 'Los angeles', 'Chicago']}

df = pd.DataFrame(Data)
print(df)

      Name  Age          City
0   Alice  21  San francisco
1     Bob  20       New year
2   John  22      Los angeles
3   Lisa  23        Chicago

# Handling Missing Values
from sklearn.preprocessing import LabelEncoder
Data = {'Name': ['Alice', 'Bob', 'John', None],
        'Age': [21, None, 22, 23],
        'City': ['San francisco', 'New year', 'Los angeles', 'Chicago']}

df = pd.DataFrame(Data)
print(df)

# Checking for missing value
print("Missing Values:\n", df.isnull())

# Dropping rows with missing values
df_cleaned = df.dropna()

# filling missing values with specified values
mean_value = df['Age'].mean()
df_filled = df.fillna(value={'Name': 'unknown',
                             'Age': df['Age'].mean()})

print("\nDataframe after dropping missing values\n", df_cleaned)
print("\nDataframe after filling missing values\n", df_filled)

# encoding Categorical Data
label_encoder = LabelEncoder()
df['Encoded_city'] =
label_encoder.fit_transform(df['City'].astype(str))
print("\n After Encoding City")
print(df)

      Name  Age          City
0   Alice  21.0  San francisco
1     Bob    NaN       New year
2   John  22.0      Los angeles

```

```

3 None 23.0 Chicago
Missing Values:
   Name    Age    City
0 False  False  False
1 False  True   False
2 False  False  False
3 True   False  False

Dataframe after dropping missing values
   Name    Age    City
0 Alice  21.0  San franscisco
2 John   22.0  Los angeles

Dataframe after filling missing values
   Name    Age    City
0 Alice  21.0  San franscisco
1 Bob    22.0  New year
2 John   22.0  Los angeles
3 unknown 23.0 Chicago

After Encoding City
   Name    Age    City  Encoded_city
0 Alice  21.0  San franscisco      3
1 Bob    NaN    New year        2
2 John   22.0  Los angeles      1
3 None   23.0  Chicago         0

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Encoding Categorical Data
categories = ['red', 'blue', 'green', 'yellow', 'white']

label_encoder = LabelEncoder()
numeric_labels = label_encoder.fit_transform(categories)

print(categories)
print(numeric_labels)

# Creating DataFrame
data = {'Name': ['Alice', 'Bob', 'John', None],
        'Age': [21, None, 22, 23],
        'City': ['San Francisco', 'New York', 'Los Angeles',
                 'Chicago']}
df = pd.DataFrame(data)

# Encoding 'City' column
label_encoder = LabelEncoder()
df['City_Encoded'] = label_encoder.fit_transform(df['City'])

```

```
X = df[['Age', 'City_Encoded']]  
y = df['Name']  
  
# Splitting the data into training and test sets  
x_train, x_test, y_train, y_test = train_test_split(X, y,  
test_size=0.2, random_state=42)  
  
print("Encoded DataFrame:")  
print(df)  
print("\nX_train:")  
print(x_train)  
print("\nX_test:")  
print(x_test)
```

```
['red', 'blue', 'green', 'yellow', 'white']  
[2 0 1 4 3]
```

Encoded DataFrame:

	Name	Age	City	City_Encoded
0	Alice	21.0	San Francisco	3
1	Bob	NaN	New York	2
2	John	22.0	Los Angeles	1
3	None	23.0	Chicago	0

X_train:

	Age	City_Encoded
3	23.0	0
0	21.0	3
2	22.0	1

X_test:

	Age	City_Encoded
1	NaN	2

```
# Independent variables (features)
```

```
X = df[['Name', 'Age', 'City']]
```

```
# Dependent variable (target)
```

```
y = df['City_Encoded']
```

```
print("Independent Variables (Features):\n", X)
```

```
print("Dependent Variable (Target):\n", y)
```

Independent Variables (Features):

	Name	Age	City
0	Alice	21.0	San Francisco
1	Bob	NaN	New York
2	John	22.0	Los Angeles
3	None	23.0	Chicago

Dependent Variable (Target):

```
0 3
```

```
1 2
```

```
2   1  
3   0  
Name: City_Encoded, dtype: int32  
from sklearn.preprocessing import StandardScaler  
import pandas as pd  
Data = {'Name': ['Alice', 'Bob', 'John', None],  
        'Age': [21, None, 22, 23],  
        'City': ['San francisco', 'New year', 'Los angeles', 'Chicago']}  
  
df = pd.DataFrame(Data)  
# Instantiate the StandardScaler  
scaler = StandardScaler()  
# Fit and transform the 'Age' column  
X = df[['Name', 'Age', 'City']]  
X['Age'] = scaler.fit_transform(X[['Age']]).fillna(X[['Age']].mean())  
print("Scaled Independent Variables (Features):\n", X)  
  
Scaled Independent Variables (Features):  
      Name      Age          City  
0  Alice -1.414214  San francisco  
1    Bob  0.000000      New year  
2   John  0.000000    Los angeles  
3   None  1.414214     Chicago  
  
import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LinearRegression  
  
df = pd.read_csv('House_data.csv')  
print(df)  
  
          date      price  bedrooms  bathrooms  
sqft_living \  
0  2014-05-02 00:00:00  3.130000e+05      3.0      1.50  
1340  
1  2014-05-02 00:00:00  2.384000e+06      5.0      2.50  
3650  
2  2014-05-02 00:00:00  3.420000e+05      3.0      2.00  
1930  
3  2014-05-02 00:00:00  4.200000e+05      3.0      2.25  
2000  
4  2014-05-02 00:00:00  5.500000e+05      4.0      2.50  
1940  
...  
...  
4595 2014-07-09 00:00:00      ...      ...      ...  
       ...      ...      ...  
           ...      ...      ...  
           ...      ...      ...  
           ...      ...      ...
```

1510						
4596	2014-07-09 00:00:00	5.343333e+05		3.0	2.50	
1460						
4597	2014-07-09 00:00:00	4.169042e+05		3.0	2.50	
3010						
4598	2014-07-10 00:00:00	2.034000e+05		4.0	2.00	
2090						
4599	2014-07-10 00:00:00	2.206000e+05		3.0	2.50	
1490						

	sqft_lot	floors	waterfront	view	condition	sqft_above	\
0	7912	1.5	0	0	3	1340	
1	9050	2.0	0	4	5	3370	
2	11947	1.0	0	0	4	1930	
3	8030	1.0	0	0	4	1000	
4	10500	1.0	0	0	4	1140	
...	
4595	6360	1.0	0	0	4	1510	
4596	7573	2.0	0	0	3	1460	
4597	7014	2.0	0	0	3	3010	
4598	6630	1.0	0	0	3	1070	
4599	8102	2.0	0	0	4	1490	

	sqft_basement	yr_built	yr_renovated	street
\				
0	0	1955	2005	18810 Densmore Ave N
1	280	1921	0	709 W Blaine St
2	0	1966	0	26206-26214 143rd Ave SE
3	1000	1963	0	857 170th Pl NE
4	800	1976	1992	9105 170th Ave NE
...
4595	0	1954	1979	501 N 143rd St
4596	0	1983	2009	14855 SE 10th Pl
4597	0	2009	0	759 Ilwaco Pl NE
4598	1020	1974	0	5148 S Creston St
4599	0	1990	0	18717 SE 258th St

	city	statezip	country
0	Shoreline	WA 98133	USA
1	Seattle	WA 98119	USA

```

2          Kent  WA 98042      USA
3        Bellevue  WA 98008      USA
4       Redmond  WA 98052      USA
...
4595      Seattle  WA 98133      USA
4596    Bellevue  WA 98007      USA
4597     Renton  WA 98059      USA
4598     Seattle  WA 98178      USA
4599   Covington  WA 98042      USA

```

[4600 rows x 18 columns]

```
print(df.describe())
```

	price	bedrooms	bathrooms	sqft_living	
count	4.600000e+03	4600.000000	4600.000000	4600.000000	
mean	5.519630e+05	3.400870	2.160815	2139.346957	
std	5.638347e+05	0.908848	0.783781	963.206916	
min	0.000000e+00	0.000000	0.000000	370.000000	
25%	3.228750e+05	3.000000	1.750000	1460.000000	
50%	4.609435e+05	3.000000	2.250000	1980.000000	
75%	6.549625e+05	4.000000	2.500000	2620.000000	
max	2.659000e+07	9.000000	8.000000	13540.000000	
\					
	floors	waterfront	view	condition	sqft_above
count	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000
mean	1.512065	0.007174	0.240652	3.451739	1827.265435
std	0.538288	0.084404	0.778405	0.677230	862.168977
min	1.000000	0.000000	0.000000	1.000000	370.000000
25%	1.000000	0.000000	0.000000	3.000000	1190.000000
50%	1.500000	0.000000	0.000000	3.000000	1590.000000
75%	2.000000	0.000000	0.000000	4.000000	2300.000000
max	3.500000	1.000000	4.000000	5.000000	9410.000000

```
count    sqft_basement      yr_built   yr_renovated
mean      4600.000000     4600.000000     4600.000000
std       312.081522     1970.786304     808.608261
std       464.137228     29.731848     979.414536
min       0.000000     1900.000000     0.000000
25%      0.000000     1951.000000     0.000000
50%      0.000000     1976.000000     0.000000
75%      610.000000     1997.000000     1999.000000
max      4820.000000     2014.000000     2014.000000

print(df.isnull().sum())
date          0
price         0
bedrooms      0
bathrooms     0
sqft_living    0
sqft_lot       0
floors         0
waterfront     0
view           0
condition       0
sqft_above      0
sqft_basement    0
yr_built        0
yr_renovated     0
street          0
city            0
statezip        0
country         0
dtype: int64

new_df = df.drop(labels=['street','city','statezip','country'],axis=1)
new_df = new_df.drop('date',axis=1)
new_df

      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors
0  3.130000e+05        3.0       1.50      1340     7912    1.5
1  2.384000e+06        5.0       2.50      3650     9050    2.0
2  3.420000e+05        3.0       2.00      1930     11947   1.0
3  4.200000e+05        3.0       2.25      2000     8030    1.0
4  5.500000e+05        4.0       2.50      1940    10500   1.0
...       ...       ...       ...       ...       ...       ...
```

4595	3.081667e+05	3.0	1.75	1510	6360	1.0
4596	5.343333e+05	3.0	2.50	1460	7573	2.0
4597	4.169042e+05	3.0	2.50	3010	7014	2.0
4598	2.034000e+05	4.0	2.00	2090	6630	1.0
4599	2.206000e+05	3.0	2.50	1490	8102	2.0

	waterfront	view	condition	sqft_above	sqft_basement	yr_built
0	0	0	3	1340	0	1955
1	0	4	5	3370	280	1921
2	0	0	4	1930	0	1966
3	0	0	4	1000	1000	1963
4	0	0	4	1140	800	1976
...
4595	0	0	4	1510	0	1954
4596	0	0	3	1460	0	1983
4597	0	0	3	3010	0	2009
4598	0	0	3	1070	1020	1974
4599	0	0	4	1490	0	1990

	yr_renovated
0	2005
1	0
2	0
3	0
4	1992
...	...
4595	1979
4596	2009
4597	0
4598	0
4599	0

[4600 rows x 13 columns]

```

X =
new_df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition']]
y = new_df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)

import matplotlib.pyplot as plt
y_pred = model.predict(X_test)

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()

```

