| | | | |
|---|---|---|---|
| **Course Name:** | **Data Analysis Laboratory (216H03L501 )** | **Semester:** | **V** |
| **Date of Performance:** | **28 / 07 / 2025** | **DIV/ Batch No:** | **D2** |
| **Student Name:** | **Shreyans Tatiya** | **Roll No:** | **16010123325** |

## Experiment No: 2

**Title: Implement data pre-processing using python on real world dataset**

### Objectives of the Experiment:

To understand the need for data preprocessing before analysis or model building

### COs to be achieved:

CO1: Understand basic concepts of data analytics to solve real-world problems

### Books/ Journals/ Websites referred:

1. https://pandas.pydata.org/
2. https://numpy.org/
3. This experiment

### Theory:

Pandas is a powerful open-source Python library used for data manipulation and analysis. It provides data structures like DataFrame and Series which allow efficient handling of structured data. With pandas, operations like filtering, grouping, merging, reshaping, and cleaning data become intuitive and fast.

### Problem statement/ Tasks

**Program:**

```
import pandas as pd

import numpy as np

# Sample data
```

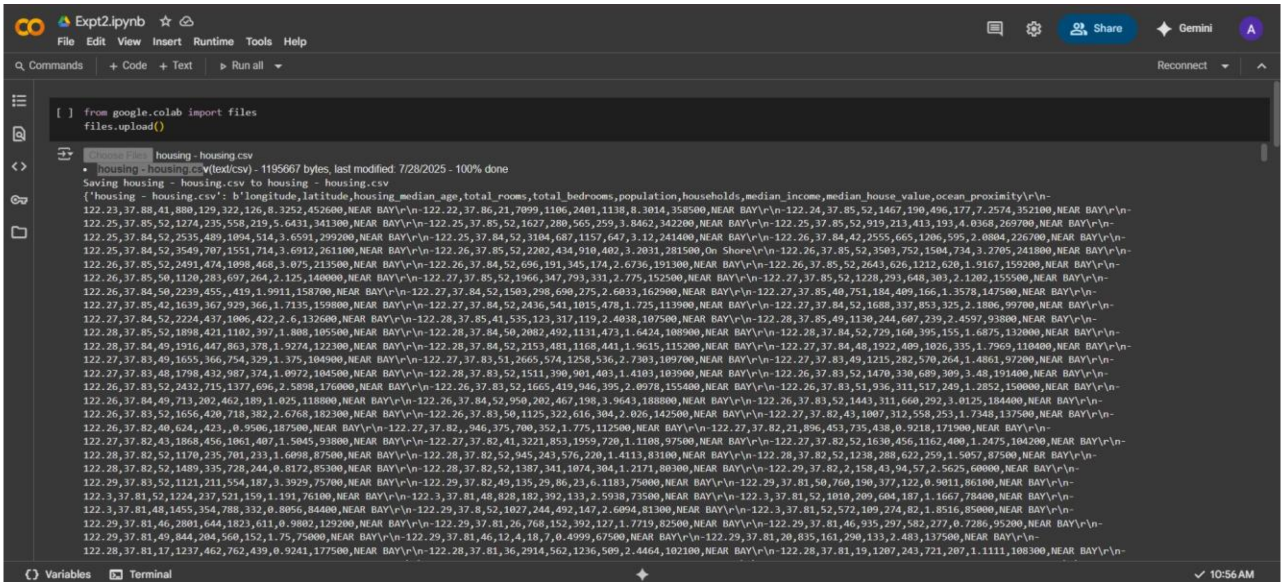
```
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve'],
    'age': [25, np.nan, 30, 22, 35],
    'gender': ['F', 'M', 'M', 'M', 'F'],
    'income': [50000, 60000, 75000, np.nan, 80000]
}
df = pd.DataFrame(data)
# Display the original data
print("Original DataFrame:")
print(df)
# User-defined function for discretization
def discretize_age(age):
    if age < 30:
        return 'Young'
    elif age >= 30 and age < 40:
        return 'Middle-aged'
    else:
        return 'Old'
# To view result
df['age_group'] = df['age'].apply(discretize_age)
# Handling missing values (NaN)
# Fill missing values in 'age' with the mean age
```

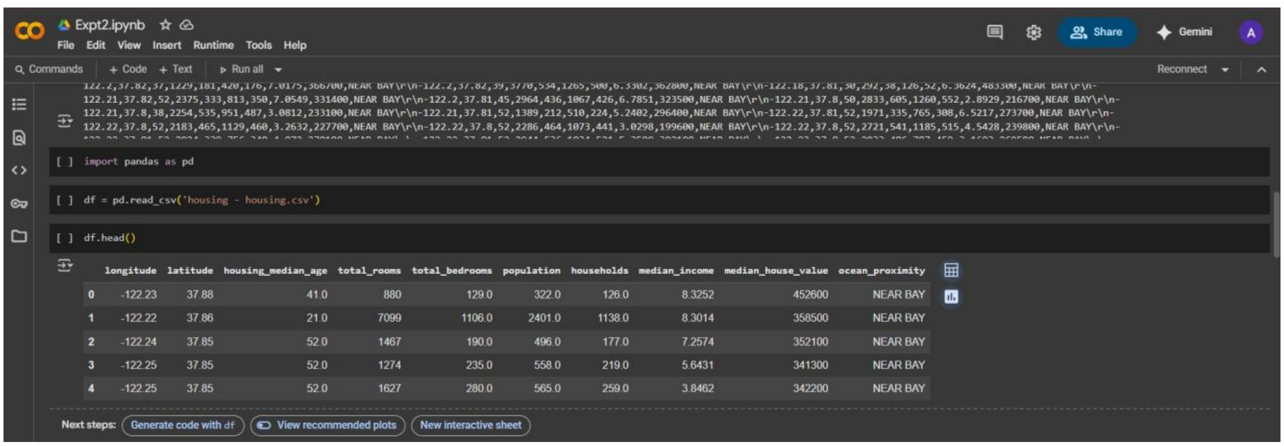
```
mean_age = df['age'].mean()

df['age'].fillna(mean_age, inplace=True)

# Apply discretization function to 'age' column

df['age_category'] = df['age'].apply(discretize_age)

# Drop rows with missing values in any column

df.dropna(inplace=True)

# Convert categorical variables (gender) to numerical

df['gender'] = df['gender'].map({'F': 0, 'M': 1})

# find employee with maximum salary

df.loc[df['income'].idxmax(), 'name']

#find youngest employee

df.loc[df['age'].idxmax(), 'name']

# Data normalization Min -Max

# Normalize 'income' column to range [0, 1]

min_income = df['income'].min()

max_income = df['income'].max()

df['income_normalized'] = (df['income'] - min_income) / (max_income - min_income)


# Display cleaned, preprocessed, and discretized data

print("\nCleaned, Preprocessed, and Discretized DataFrame:")

print(df)
```

| |
|---|
| **Task: Download the real time data set and implement data preprocessing techniques on the real time data set**<br><br>**Source of the dataset (URL):**<br><br>**Platform used by the student:**<br><br>**Following points should be written by students**<br><br>Different steps in Data Preprocessing:<br><br>• **Finding missing, null values**<br>• **Replacing missing, null values with statistical parameters**<br>• **Encoding categorical data if needed (Write user defined function)**<br>• **Normalization (Write user defined function)**<br>• **Discretization (Write user defined function)** |
| |

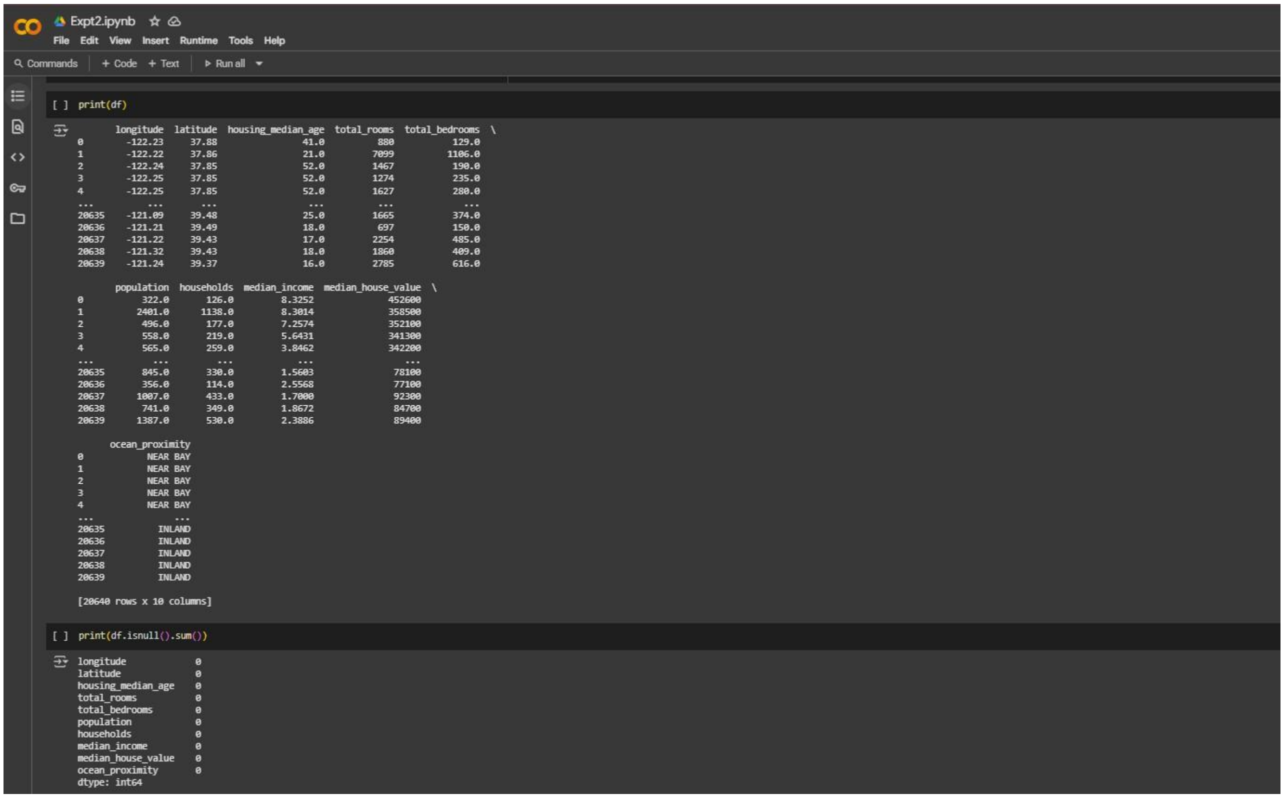
| **Code :** |
|---|
| https://colab.research.google.com/drive/146p8irTZTIqJb8HW3xQjYKUTnaHUviXp?usp=sharing |

| **Output:** |
|---|
| |


Expt2.ipynb
from google.colab import files
files.upload()
housing - housing.csv
housing - housing.csv(text/csv) - 1195667 bytes, last modified: 7/28/2025 - 100% done
Saving housing - housing.csv to housing - housing.csv
Variables
Terminal
10:56 AM


Expt2.ipynb
import pandas as pd
df = pd.read_csv('housing - housing.csv')
df.head()
longitude latitude housing_median_age total_rooms total_bedrooms population households median_income median_house_value ocean_proximity
0 -122.23 37.88 41.0 880 129.0 322.0 126.0 8.3252 452600 NEAR BAY
1 -122.22 37.86 21.0 7099 1106.0 2401.0 1138.0 8.3014 358500 NEAR BAY
2 -122.24 37.85 52.0 1467 190.0 496.0 177.0 7.2574 352100 NEAR BAY
3 -122.25 37.85 52.0 1274 235.0 558.0 219.0 5.6431 341300 NEAR BAY
4 -122.25 37.85 52.0 1627 280.0 565.0 259.0 3.8462 342200 NEAR BAY
Next steps: Generate code with df View recommended plots New interactive sheet


Expt2.ipynb
print(df.isnull().sum())
longitude 0
latitude 0
housing_median_age 2
total_rooms 0
total_bedrooms 208
population 1
households 1
median_income 0
median_house_value 0
ocean_proximity 1
dtype: int64
df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_bedrooms'].mean())
df['housing_median_age'] = df['housing_median_age'].fillna(df['housing_median_age'].mean())
df['population'] = df['population'].fillna(df['population'].mean())
df['households'] = df['households'].fillna(df['households'].mean())
# Mode produces a series types, we need to use [0] to get only the value, else it'll give us dtype and more areas we don't require
df['ocean_proximity'] = df['ocean_proximity'].fillna(df['ocean_proximity'].mode()[0])



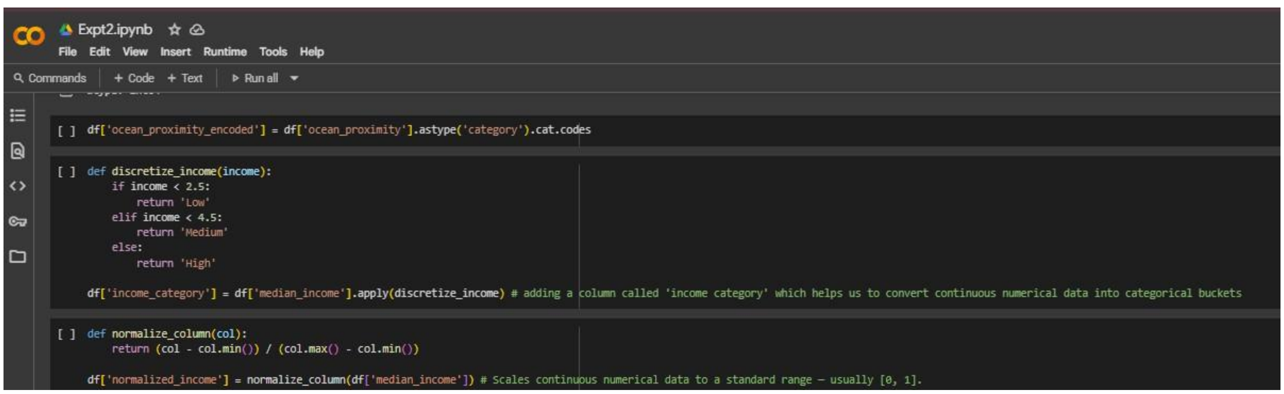
Expt2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
[ ] print(df)
```

```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0          880           129.0
1        -122.22     37.86                21.0         7099          1106.0
2        -122.24     37.85                52.0         1467           190.0
3        -122.25     37.85                52.0         1274           235.0
4        -122.25     37.85                52.0         1627           280.0
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0         1665           374.0
20636    -121.21     39.49                18.0          697           150.0
20637    -121.22     39.43                17.0         2254           485.0
20638    -121.32     39.43                18.0         1860           409.0
20639    -121.24     39.37                16.0         2785           616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252              452600
1          2401.0      1138.0         8.3014              358500
2           496.0       177.0         7.2574              352100
3           558.0       219.0         5.6431              341300
4           565.0       259.0         3.8462              342200
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603               78100
20636       356.0       114.0         2.5568               77100
20637      1007.0       433.0         1.7000               92300
20638       741.0       349.0         1.8672               84700
20639      1387.0       530.0         2.3886               89400

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]
```

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

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

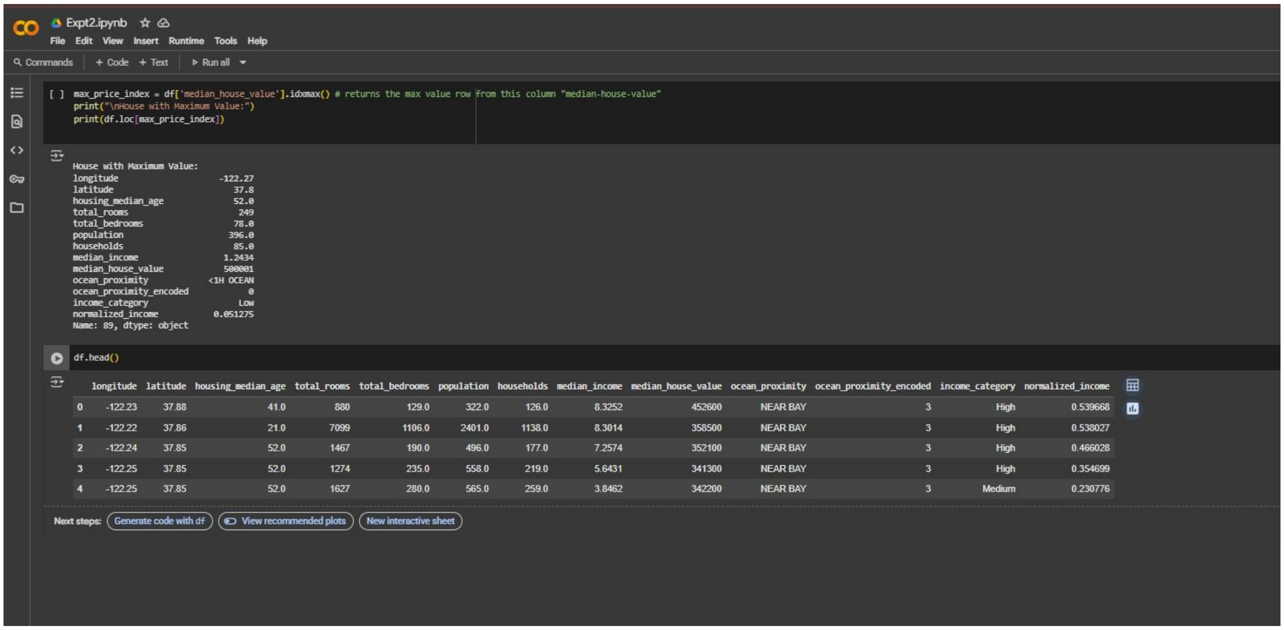


Expt2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
[ ] df['ocean_proximity_encoded'] = df['ocean_proximity'].astype('category').cat.codes
```

```
[ ] def discretize_income(income):
        if income < 2.5:
            return 'Low'
        elif income < 4.5:
            return 'Medium'
        else:
            return 'High'

    df['income_category'] = df['median_income'].apply(discretize_income) # adding a column called 'income category' which helps us to convert continuous numerical data into categorical buckets
```

```
[ ] def normalize_column(col):
        return (col - col.min()) / (col.max() - col.min())

    df['normalized_income'] = normalize_column(df['median_income']) # Scales continuous numerical data to a standard range – usually [0, 1].
```



Expt2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
max_price_index = df['median_house_value'].idxmax() # returns the max value row from this column "median-house-value"
print("\nHouse with Maximum Value:")
print(df.loc[max_price_index])
```

```
House with Maximum Value:
longitude                    -122.27
latitude                        37.8
housing_median_age              52.0
total_rooms                      249
total_bedrooms                  78.0
population                     396.0
households                      85.0
median_income                 1.2434
median_house_value            500001
ocean_proximity            <1H OCEAN
ocean_proximity_encoded            0
income_category                  Low
normalized_income           0.051275
Name: 89, dtype: object
```

```
df.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value | ocean_proximity | ocean_proximity_encoded | income_category | normalized_income |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880 | 129.0 | 322.0 | 126.0 | 8.3252 | 452600 | NEAR BAY | 3 | High | 0.539668 |
| 1 | -122.22 | 37.86 | 21.0 | 7099 | 1106.0 | 2401.0 | 1138.0 | 8.3014 | 358500 | NEAR BAY | 3 | High | 0.538027 |
| 2 | -122.24 | 37.85 | 52.0 | 1467 | 190.0 | 496.0 | 177.0 | 7.2574 | 352100 | NEAR BAY | 3 | High | 0.466028 |
| 3 | -122.25 | 37.85 | 52.0 | 1274 | 235.0 | 558.0 | 219.0 | 5.6431 | 341300 | NEAR BAY | 3 | High | 0.354699 |
| 4 | -122.25 | 37.85 | 52.0 | 1627 | 280.0 | 565.0 | 259.0 | 3.8462 | 342200 | NEAR BAY | 3 | Medium | 0.230776 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

## Post Lab Subjective/Objective type Questions:

**Q.1 What are some common challenges encountered during data cleaning? How did you handle missing values in the provided dataset?**

**Ans:**

Common challenges in data cleaning include:

- Missing values
- Inconsistent data types
- Outliers
- Duplicate entries

In the provided dataset, we handled missing values by:
Using the mode (most frequent value) to fill missing entries in the ocean_proximity column:
df['ocean_proximity'] = df['ocean_proximity'].fillna(df['ocean_proximity'].mode()[0])

**Q.2 Explain the importance of data normalization in the context of machine learning models. How does normalizing benefit the analysis?**

**Ans:**

- Normalization scales numerical features to a common range (typically between 0 and 1).
- Without normalization, features with larger values can dominate those with smaller values, leading to biased results.
- In this lab, normalization helped us compare median_income values more effectively:
  df['normalized_income'] = (df['median_income'] - df['median_income'].min()) / (df['median_income'].max() - df['median_income'].min())

**Q.3 Discuss why it's essential to convert categorical variables like 'gender' into numerical representations.**

**Ans:** Categorical variables (e.g., gender, ocean_proximity) contain text labels that machine learning models **cannot process directly**. Converting them to numerical form allows:

- Better compatibility with models
- Logical comparisons and distance calculations

For example, we used **label encoding** to convert the ocean_proximity column:

df['ocean_proximity_encoded'] = df['ocean_proximity'].astype('category').cat.codes

This makes the data model-friendly while preserving category information.

| **Conclusion:** |
| --- |
| In this lab, I learned how essential data cleaning and preprocessing steps are for preparing real-world datasets. I handled missing values, encoded categorical features, normalized numerical data, and extracted useful insights. |