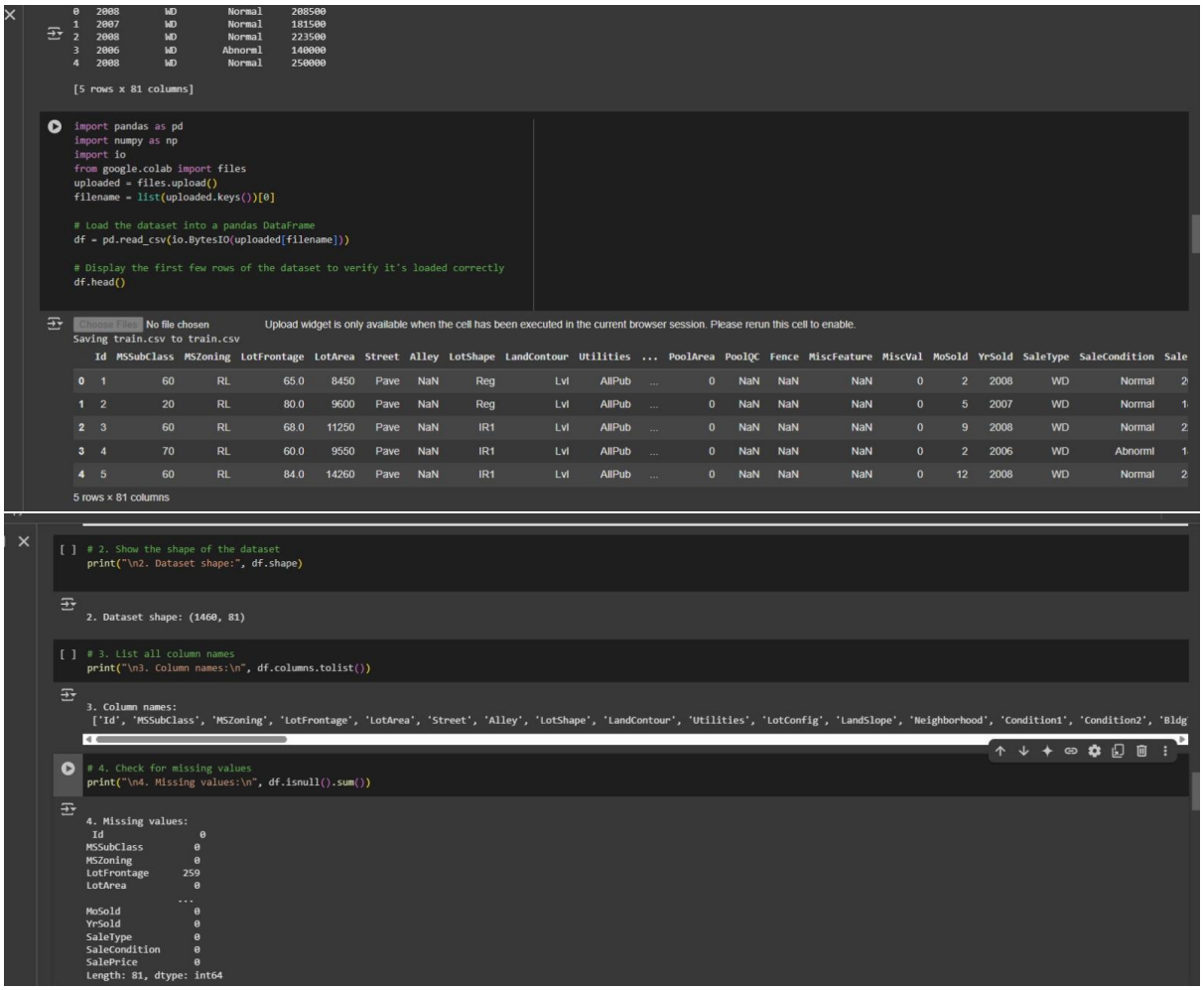
Name = PAYAL S BAJANTRI

PRN = 202401120064

BATCH = CS8-68

Data set = https://drive.google.com/file/d/1c26XS7KvN_IOnZwtUJrUZQRzZTCKzisz/view?usp=drivesdk



```
0   2008        WD         Normal    208500
1   2007        WD         Normal    181500
2   2008        WD         Normal    223500
3   2006        WD        Abnorml    140000
4   2008        WD         Normal    250000

[5 rows x 81 columns]
```

```
import pandas as pd
import numpy as np
import io
from google.colab import files
uploaded = files.upload()
filename = list(uploaded.keys())[0]

# Load the dataset into a pandas DataFrame
df = pd.read_csv(io.BytesIO(uploaded[filename]))

# Display the first few rows of the dataset to verify it's loaded correctly
df.head()
```

```
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving train.csv to train.csv
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	Sale
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	2
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	1
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	2
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	1
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	2

5 rows × 81 columns

```
[ ] # 2. Show the shape of the dataset
    print("\n2. Dataset shape:", df.shape)

2. Dataset shape: (1460, 81)

[ ] # 3. List all column names
    print("\n3. Column names:\n", df.columns.tolist())

3. Column names:
 ['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'Bldg

# 4. Check for missing values
print("\n4. Missing values:\n", df.isnull().sum())

4. Missing values:
 Id               0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                 ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

```
[ ] # 5. Display data types of each column
    print("\n5. Data types:\n", df.dtypes)
```

```
5. Data types:
 Id                int64
MSSubClass         int64
MSZoning          object
LotFrontage      float64
LotArea            int64
                   ...
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
Length: 81, dtype: object
```

```
# 6. Describe numerical columns
print("\n6. Statistical summary:\n", df.describe())
```

```
6. Statistical summary:
                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  ...
std       1.112799    30.202904     20.645407   181.066207   456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000  ...
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000  ...
```

```
# 7. Count of unique values in each column
print("\n7. Unique values per column:\n", df.nunique())
```

```
7. Unique values per column:
 Id               1460
MSSubClass         15
MSZoning            5
LotFrontage       110
LotArea          1073
                 ...
MoSold             12
YrSold              5
SaleType            9
SaleCondition       6
SalePrice         663
Length: 81, dtype: int64
```

```
] # 8. Correlation matrix (only numeric)
  numeric_cols = df.select_dtypes(include='number').columns
  print("\n8. Correlation matrix (numerical features only):\n", df[numeric_cols].corr())
```

```
8. Correlation matrix (numerical features only):
                      Id  MSSubClass  LotFrontage   LotArea  OverallQual  \
Id              1.000000    0.011156    -0.010601 -0.033226    -0.028365
MSSubClass      0.011156    1.000000    -0.386347 -0.139781     0.032628
LotFrontage    -0.010601   -0.386347     1.000000  0.426095     0.251646
LotArea        -0.033226   -0.139781     0.426095  1.000000     0.105806
OverallQual    -0.028365    0.032628     0.251646  0.105806     1.000000
OverallCond     0.012609   -0.059316    -0.059213 -0.005636    -0.091932
YearBuilt      -0.012713    0.027850     0.123349  0.014228     0.572323
YearRemodAdd   -0.021998    0.040581     0.088866  0.013788     0.550684
MasVnrArea     -0.050298    0.022936     0.193458  0.104160     0.411876
BsmtFinSF1     -0.005024   -0.069836     0.233633  0.214103     0.239666
BsmtFinSF2     -0.005968   -0.065649     0.049900  0.111170    -0.059119
BsmtUnfSF      -0.007940   -0.140759     0.132644 -0.002618     0.308159
TotalBsmtSF    -0.015415   -0.238518     0.392075  0.260833     0.537808
1stFlrSF        0.010496   -0.251758     0.457181  0.299475     0.476224
2ndFlrSF        0.005590    0.307886     0.080177  0.050986     0.295493
```

```python
# 9. Value counts for 'Neighborhood' if it exists
if 'Neighborhood' in df.columns:
    print("\n9. Value counts for 'Neighborhood':\n", df['Neighborhood'].value_counts())
else:
    print("\n9. 'Neighborhood' column not found.")
```

```
9. Value counts for 'Neighborhood':
 Neighborhood
NAmes      225
CollgCr    150
OldTown    113
Edwards    100
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     41
Timber      38
IDOTRR      37
ClearCr     28
SWISU       25
StoneBr     25
Blmngtn     17
MeadowV     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: count, dtype: int64
```

```python
# 10. Group by 'Neighborhood' and calculate average 'SalePrice'
if 'Neighborhood' in df.columns and 'SalePrice' in df.columns:
    print("\n10. Average SalePrice by Neighborhood:\n", df.groupby('Neighborhood')['SalePrice'].mean())
else:
    print("\n10. Cannot group by 'Neighborhood' (column missing).")
```

```
10. Average SalePrice by Neighborhood:
 Neighborhood
Blmngtn    194870.882353
Blueste    137500.000000
BrDale     104493.750000
BrkSide    124834.051724
ClearCr    212565.428571
CollgCr    197965.773333
Crawfor    210624.725490
Edwards    128219.700000
Gilbert    192854.506329
IDOTRR     100123.783784
MeadowV     98576.470588
Mitchel    156270.122449
NAmes      145847.080000
NPkVill    142694.444444
NWAmes     189050.068493
NoRidge    335295.317073
NridgHt    316270.623377
OldTown    128225.300885
SWISU      142591.360000
Sawyer     136793.135135
SawyerW    186555.796610
Somerst    225379.837209
StoneBr    310499.000000
Timber     242247.447368
Veenker    238772.727273
Name: SalePrice, dtype: float64
```

```python
# 11. Create a new column 'Price_per_SqFt'
if 'GrLivArea' in df.columns and 'SalePrice' in df.columns:
    df['Price_per_SqFt'] = df['SalePrice'] / df['GrLivArea']
    print("\n11. Added 'Price_per_SqFt' column.")
else:
    print("\n11. Columns 'GrLivArea' or 'SalePrice' missing.")
```

```
11. Added 'Price_per_SqFt' column.
```

```python
# 12. Find the property with the highest 'SalePrice'
if 'SalePrice' in df.columns:
    print("\n12. Property with highest SalePrice:\n", df.loc[df['SalePrice'].idxmax()])
else:
    print("\n12. 'SalePrice' column not found.")
```

```
12. Property with highest SalePrice:
 Id                      692
MSSubClass               60
MSZoning                 RL
LotFrontage           104.0
LotArea               21535
                      ...
YrSold                 2007
SaleType                WD
SaleCondition         Normal
SalePrice             755000
Price_per_SqFt    174.930491
Name: 691, Length: 82, dtype: object
```

```python
# 13. Find the property with the lowest 'SalePrice'
if 'SalePrice' in df.columns:
    print("\n13. Property with lowest SalePrice:\n", df.loc[df['SalePrice'].idxmin()])
else:
    print("\n13. 'SalePrice' column not found.")
```

```
13. Property with lowest SalePrice:
 Id                      496
MSSubClass               30
MSZoning            C (all)
LotFrontage            60.0
LotArea                7879
                      ...
YrSold                 2009
SaleType                 WD
SaleCondition       Abnorml
SalePrice             34900
Price_per_SqFt    48.472222
Name: 495, Length: 82, dtype: object
```

```python
# 14. Average 'SalePrice' for houses built after 2000
if 'YearBuilt' in df.columns and 'SalePrice' in df.columns:
    print("\n14. Average SalePrice for houses built after 2000:\n", df[df['YearBuilt'] > 2000]['SalePrice'].mean())
else:
    print("\n14. Required columns not found.")
```

```
14. Average SalePrice for houses built after 2000:
 244527.4587912088
```

```python
# 15. Number of houses with 'GarageArea' greater than 500
if 'GarageArea' in df.columns:
    print("\n15. Houses with GarageArea > 500:\n", df[df['GarageArea'] > 500].shape[0])
else:
    print("\n15. 'GarageArea' column not found.")
```

```
15. Houses with GarageArea > 500:
 629
```

```python
# 16. Median 'LotArea'
if 'LotArea' in df.columns:
    print("\n16. Median LotArea:", df['LotArea'].median())
else:
    print("\n16. 'LotArea' column not found.")
```

```
16. Median LotArea: 9478.5
```

```python
# 17. Standard deviation of 'SalePrice'
if 'SalePrice' in df.columns:
    print("\n17. SalePrice standard deviation:", df['SalePrice'].std())
else:
    print("\n17. 'SalePrice' column not found.")
```

```
17. SalePrice standard deviation: 79442.50288288663
```

```python
# 18. Normalize 'SalePrice' column
if 'SalePrice' in df.columns:
    df['SalePrice_Normalized'] = (df['SalePrice'] - df['SalePrice'].mean()) / df['SalePrice'].std()
    print("\n18. Normalized 'SalePrice' column added.")
else:
    print("\n18. 'SalePrice' column not found.")
```

```
18. Normalized 'SalePrice' column added.
```

```python
# 19. Top 5 most expensive neighborhoods
if 'Neighborhood' in df.columns and 'SalePrice' in df.columns:
    print("\n19. Top 5 expensive neighborhoods:\n", df.groupby('Neighborhood')['SalePrice'].mean().sort_values(ascending=False).head(5))
else:
    print("\n19. Required columns not found.")
```

```
19. Top 5 expensive neighborhoods:
 Neighborhood
NoRidge    335295.317073
NridgHt    316270.623377
StoneBr    310499.000000
Timber     242247.447368
Veenker    238772.727273
Name: SalePrice, dtype: float64
```

```python
# 20. Save the modified DataFrame to a new CSV file
df.to_csv('train_modified.csv', index=False)
print("\n20. Modified dataset saved as 'train_modified.csv'.")
```

```
20. Modified dataset saved as 'train_modified.csv'.
```