Assignment1_kwok

January 10, 2024

[1]:
```
#Import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# Set Pandas options to show all columns and rows
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)

# Make variable for input file
INFILE = "/Users/jck/Documents/MSDS 422/Unit 1/Assignment 1/HMEQ_Loss.csv"

# Read in the data file
df = pd.read_csv(INFILE, sep=',', header=0)
```

[2]:
```
# Print the first 5 rows of the data frame
df.head(5)
```

[2]:
```
   TARGET_BAD_FLAG  TARGET_LOSS_AMT  LOAN  MORTDUE     VALUE   REASON     JOB  \
0                1            641.0  1100  25860.0   39025.0  HomeImp   Other
1                1           1109.0  1300  70053.0   68400.0  HomeImp   Other
2                1            767.0  1500  13500.0   16700.0  HomeImp   Other
3                1           1425.0  1500      NaN       NaN      NaN     NaN
4                0              NaN  1700  97800.0  112000.0  HomeImp  Office

    YOJ  DEROG  DELINQ       CLAGE  NINQ  CLNO  DEBTINC
0  10.5    0.0     0.0   94.366667   1.0   9.0      NaN
1   7.0    0.0     2.0  121.833333   0.0  14.0      NaN
2   4.0    0.0     0.0  149.466667   1.0  10.0      NaN
3   NaN    NaN     NaN         NaN   NaN   NaN      NaN
4   3.0    0.0     0.0   93.333333   0.0  14.0      NaN
```

[3]:
```python
# Print the data shape, such as how many rows and columns
print(df.shape)
```

```
(5960, 14)
```

[4]:
```python
# Print the data types for each column
dt = df.dtypes
print(dt)
```

```
TARGET_BAD_FLAG      int64
TARGET_LOSS_AMT    float64
LOAN                 int64
MORTDUE            float64
VALUE              float64
REASON              object
JOB                 object
YOJ                float64
DEROG              float64
DELINQ             float64
CLAGE              float64
NINQ               float64
CLNO               float64
DEBTINC            float64
dtype: object
```

[5]:
```python
# Print the data frame statistics
print(df.describe())
```

```
       TARGET_BAD_FLAG  TARGET_LOSS_AMT          LOAN        MORTDUE  \
count      5960.000000      1189.000000   5960.000000    5442.000000
mean          0.199497     13414.576955  18607.969799   73760.817200
std           0.399656     10839.455965  11207.480417   44457.609458
min           0.000000       224.000000   1100.000000    2063.000000
25%           0.000000      5639.000000  11100.000000   46276.000000
50%           0.000000     11003.000000  16300.000000   65019.000000
75%           0.000000     17634.000000  23300.000000   91488.000000
max           1.000000     78987.000000  89900.000000  399550.000000

               VALUE          YOJ        DEROG       DELINQ        CLAGE  \
count    5848.000000  5445.000000  5252.000000  5380.000000  5652.000000
mean   101776.048741     8.922268     0.254570     0.449442   179.766275
std     57385.775334     7.573982     0.846047     1.127266    85.810092
min      8000.000000     0.000000     0.000000     0.000000     0.000000
25%     66075.500000     3.000000     0.000000     0.000000   115.116702
50%     89235.500000     7.000000     0.000000     0.000000   173.466667
75%    119824.250000    13.000000     0.000000     0.000000   231.562278
max    855909.000000    41.000000    10.000000    15.000000  1168.233561

              NINQ         CLNO      DEBTINC
```

```
count   5450.000000  5738.000000  4693.000000
mean       1.186055    21.296096    33.779915
std        1.728675    10.138933     8.601746
min        0.000000     0.000000     0.524499
25%        0.000000    15.000000    29.140031
50%        1.000000    20.000000    34.818262
75%        2.000000    26.000000    39.003141
max       17.000000    71.000000   203.312149
```

[6]:

```
# Print the number of missing values for each column
missing_values  =df.isnull().sum()
print(missing_values)
# After running the above code, we can see that there are 12 columns with␣
 ↪missing values.
```

```
TARGET_BAD_FLAG       0
TARGET_LOSS_AMT    4771
LOAN                  0
MORTDUE             518
VALUE               112
REASON              252
JOB                 279
YOJ                 515
DEROG               708
DELINQ              580
CLAGE               308
NINQ                510
CLNO                222
DEBTINC            1267
dtype: int64
```

[7]:

```
# Show which column is under object type and which is under numeric type
TARGET_B = "TARGET_BAD_FLAG"
TARGET_L = "TARGET_LOSS_AMT"

objList = []
numList = []
for i in dt.index :
    #print(" here is i .....", i , " ..... and here is the type", dt[i] )
    if i in ( [ TARGET_B, TARGET_L ] ) : continue
    if dt[i] in (["object"]) : objList.append( i )
    if dt[i] in (["float64","int64"]) : numList.append( i )

print(" OBJECTS ")
print(" ------- ")
for i in objList :
   print( i )
print(" ------- ")
```

```
print(" NUMBER ")
print(" ------- ")
for i in numList :
    print( i )
print(" ------- ")
```

```
 OBJECTS
 -------
REASON
JOB
 -------
 NUMBER
 -------
LOAN
MORTDUE
VALUE
YOJ
DEROG
DELINQ
CLAGE
NINQ
CLNO
DEBTINC
 -------
```

[8]:
```
# My idea is insert 0 for Target Loss Amount for my first step of data cleaning
df[TARGET_L] = df[TARGET_L].fillna(0)

missing_values2  =df.isnull().sum()
print(missing_values2)
'''
After setting the Target Loss Amount to 0, there are no longer any missing␣
↪values in that column.
I chose not to impute values for Target Loss Amount because it should be 0 if␣
↪the loan was repaid successfully.
Consequently, the count of columns with missing values has decreased to 11.
'''
```

```
TARGET_BAD_FLAG       0
TARGET_LOSS_AMT       0
LOAN                  0
MORTDUE             518
VALUE               112
REASON              252
JOB                 279
YOJ                 515
```

```
DEROG                708
DELINQ               580
CLAGE                308
NINQ                 510
CLNO                 222
DEBTINC             1267
dtype: int64
```

[8]:
```
'\nAfter setting the Target Loss Amount to 0, there are no longer any missing
values in that column. \nI chose not to impute values for Target Loss Amount
because it should be 0 if the loan was repaid successfully. \nConsequently, the
count of columns with missing values has decreased to 11.\n'
```

[9]:
```
# Next, lets handle the numeric columns with missing values

cols_with_missing = ['MORTDUE', 'VALUE', 'YOJ', 'DEROG', 'DELINQ', 'CLAGE',␣
 ↪'NINQ', 'CLNO', 'DEBTINC']
```

[10]:
```
'''
I observed that some data points are outliers, so I plan to employ the␣
 ↪Interquartile Range (IQR) method
to detect these outliers and substitute them, using the median value for␣
 ↪imputation.
'''
for col in cols_with_missing:
    # 1. Identify Outliers using the IQR method
    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

    # 2. Remove Outliers - Replace outliers in a copy of the column with NaN
    temp_col = df[col].copy()
    temp_col[(temp_col < lower_bound) | (temp_col > upper_bound)] = np.nan

    # 3. Calculate median of the column with outliers removed
    median_val = temp_col.median()

    # 4. Create new column for imputed values, fill missing values with the␣
 ↪calculated median
    df['IMP_'+col] = df[col].fillna(median_val)
```

[11]:
```
print(df.head(5))
```

```
   TARGET_BAD_FLAG  TARGET_LOSS_AMT  LOAN  MORTDUE    VALUE   REASON    JOB  \
0                1            641.0  1100  25860.0  39025.0  HomeImp  Other
1                1           1109.0  1300  70053.0  68400.0  HomeImp  Other
```

```
2                1            767.0  1500  13500.0   16700.0  HomeImp   Other
3                1           1425.0  1500      NaN       NaN      NaN     NaN
4                0              0.0  1700  97800.0  112000.0  HomeImp  Office

    YOJ  DEROG  DELINQ       CLAGE  NINQ  CLNO  DEBTINC  IMP_MORTDUE  \
0  10.5    0.0     0.0   94.366667   1.0   9.0      NaN      25860.0
1   7.0    0.0     2.0  121.833333   0.0  14.0      NaN      70053.0
2   4.0    0.0     0.0  149.466667   1.0  10.0      NaN      13500.0
3   NaN    NaN     NaN         NaN   NaN   NaN      NaN      63508.0
4   3.0    0.0     0.0   93.333333   0.0  14.0      NaN      97800.0

   IMP_VALUE  IMP_YOJ  IMP_DEROG  IMP_DELINQ   IMP_CLAGE  IMP_NINQ  IMP_CLNO  \
0    39025.0     10.5        0.0         0.0   94.366667       1.0       9.0
1    68400.0      7.0        0.0         2.0  121.833333       0.0      14.0
2    16700.0      4.0        0.0         0.0  149.466667       1.0      10.0
3    86908.0      7.0        0.0         0.0  172.432355       1.0      20.0
4   112000.0      3.0        0.0         0.0   93.333333       0.0      14.0

   IMP_DEBTINC
0    34.880462
1    34.880462
2    34.880462
3    34.880462
4    34.880462
```

[12]:
```python
missing_values3  =df.isnull().sum()
print(missing_values3)
# After running the above code, we can see that there are 2 columns with
  missing values.
```

```
TARGET_BAD_FLAG       0
TARGET_LOSS_AMT       0
LOAN                  0
MORTDUE             518
VALUE               112
REASON              252
JOB                 279
YOJ                 515
DEROG               708
DELINQ              580
CLAGE               308
NINQ                510
CLNO                222
DEBTINC            1267
IMP_MORTDUE           0
IMP_VALUE             0
IMP_YOJ               0
IMP_DEROG             0
```

```
IMP_DELINQ          0
IMP_CLAGE           0
IMP_NINQ            0
IMP_CLNO            0
IMP_DEBTINC         0
dtype: int64
```

[13]:
```
# Let's now address the missing values in the categorical columns.
categorical_cols_with_missing = ['REASON','JOB']
```

[14]:
```
for col in categorical_cols_with_missing :
    # 1. Fill Missing Values as a separate category
    df['TEMP_'+col] = df[col].fillna('Missing')

    # 2. One-Hot Encode in the temporary column
    encoded = pd.get_dummies(df['TEMP_'+col], prefix='OHE_'+col, dtype=int)
    # print(encoded.head(5))
    # 3. Merge the new one-hot encoded columns back with df
    df = pd.concat([df, encoded], axis=1)

    # 4.Remove the temporary column
    df.drop(columns='TEMP_'+col, inplace=True)

# After this loop, 'df' will have new one-hot encoded columns corresponding to␣
↪each category in the original columns.
```

[15]:
```
# Validate that all missing values have been handled
missing_values4  =df.isnull().sum()
print(missing_values4)
```

```
TARGET_BAD_FLAG        0
TARGET_LOSS_AMT        0
LOAN                   0
MORTDUE              518
VALUE                112
REASON               252
JOB                  279
YOJ                  515
DEROG                708
DELINQ               580
CLAGE                308
NINQ                 510
CLNO                 222
DEBTINC             1267
IMP_MORTDUE            0
IMP_VALUE              0
IMP_YOJ                0
IMP_DEROG              0
```

```
IMP_DELINQ             0
IMP_CLAGE              0
IMP_NINQ               0
IMP_CLNO               0
IMP_DEBTINC            0
OHE_REASON_DebtCon     0
OHE_REASON_HomeImp     0
OHE_REASON_Missing     0
OHE_JOB_Mgr            0
OHE_JOB_Missing        0
OHE_JOB_Office         0
OHE_JOB_Other          0
OHE_JOB_ProfExe        0
OHE_JOB_Sales          0
OHE_JOB_Self           0
dtype: int64
```

[16]:
```
print(missing_values4.tail(10))
# After running the code above, I noticed we have no more missing value
```

```
OHE_REASON_DebtCon    0
OHE_REASON_HomeImp    0
OHE_REASON_Missing    0
OHE_JOB_Mgr           0
OHE_JOB_Missing       0
OHE_JOB_Office        0
OHE_JOB_Other         0
OHE_JOB_ProfExe       0
OHE_JOB_Sales         0
OHE_JOB_Self          0
dtype: int64
```

[17]:
```
df.describe()
```

[17]:
```
       TARGET_BAD_FLAG  TARGET_LOSS_AMT          LOAN        MORTDUE  \
count      5960.000000      5960.000000   5960.000000    5442.000000
mean          0.199497      2676.163087  18607.969799   73760.817200
std           0.399656      7222.631500  11207.480417   44457.609458
min           0.000000         0.000000   1100.000000    2063.000000
25%           0.000000         0.000000  11100.000000   46276.000000
50%           0.000000         0.000000  16300.000000   65019.000000
75%           0.000000         0.000000  23300.000000   91488.000000
max           1.000000     78987.000000  89900.000000  399550.000000

               VALUE          YOJ        DEROG       DELINQ        CLAGE  \
count    5848.000000  5445.000000  5252.000000  5380.000000  5652.000000
mean   101776.048741     8.922268     0.254570     0.449442   179.766275
std     57385.775334     7.573982     0.846047     1.127266    85.810092
```

```
min      8000.000000     0.000000     0.000000     0.000000     0.000000
25%     66075.500000     3.000000     0.000000     0.000000   115.116702
50%     89235.500000     7.000000     0.000000     0.000000   173.466667
75%    119824.250000    13.000000     0.000000     0.000000   231.562278
max    855909.000000    41.000000    10.000000    15.000000  1168.233561

              NINQ         CLNO      DEBTINC    IMP_MORTDUE      IMP_VALUE  \
count  5450.000000  5738.000000  4693.000000    5960.000000    5960.000000
mean      1.186055    21.296096    33.779915   72869.716644  101496.649168
std       1.728675    10.138933     8.601746   42579.485794   56879.779380
min       0.000000     0.000000     0.524499    2063.000000    8000.000000
25%       0.000000    15.000000    29.140031   48139.000000   66489.500000
50%       1.000000    20.000000    34.818262   63508.000000   88310.500000
75%       2.000000    26.000000    39.003141   88200.250000  119004.750000
max      17.000000    71.000000   203.312149  399550.000000  855909.000000

           IMP_YOJ    IMP_DEROG   IMP_DELINQ    IMP_CLAGE     IMP_NINQ  \
count  5960.000000  5960.000000  5960.000000  5960.000000  5960.000000
mean      8.756166     0.224329     0.405705   179.387274     1.170134
std       7.259424     0.798458     1.079256    83.578832     1.653866
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       3.000000     0.000000     0.000000   117.371430     0.000000
50%       7.000000     0.000000     0.000000   172.432355     1.000000
75%      12.000000     0.000000     0.000000   227.143058     2.000000
max      41.000000    10.000000    15.000000  1168.233561    17.000000

          IMP_CLNO  IMP_DEBTINC  OHE_REASON_DebtCon  OHE_REASON_HomeImp  \
count  5960.000000  5960.000000         5960.000000         5960.000000
mean     21.247819    34.013874            0.659060            0.298658
std       9.951308     7.645985            0.474065            0.457708
min       0.000000     0.524499            0.000000            0.000000
25%      15.000000    30.763159            0.000000            0.000000
50%      20.000000    34.880462            1.000000            0.000000
75%      26.000000    37.949892            1.000000            1.000000
max      71.000000   203.312149            1.000000            1.000000

       OHE_REASON_Missing  OHE_JOB_Mgr  OHE_JOB_Missing  OHE_JOB_Office  \
count         5960.000000  5960.000000      5960.000000     5960.000000
mean             0.042282     0.128691         0.046812        0.159060
std              0.201248     0.334886         0.211254        0.365763
min              0.000000     0.000000         0.000000        0.000000
25%              0.000000     0.000000         0.000000        0.000000
50%              0.000000     0.000000         0.000000        0.000000
75%              0.000000     0.000000         0.000000        0.000000
max              1.000000     1.000000         1.000000        1.000000

       OHE_JOB_Other  OHE_JOB_ProfExe  OHE_JOB_Sales  OHE_JOB_Self
```

```
count    5960.000000    5960.000000    5960.000000    5960.000000
mean        0.400671       0.214094       0.018289       0.032383
std         0.490076       0.410227       0.134004       0.177029
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         1.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000
```

1 After filling all the missing values, I want to compare the IMP value and the original value in histogram.

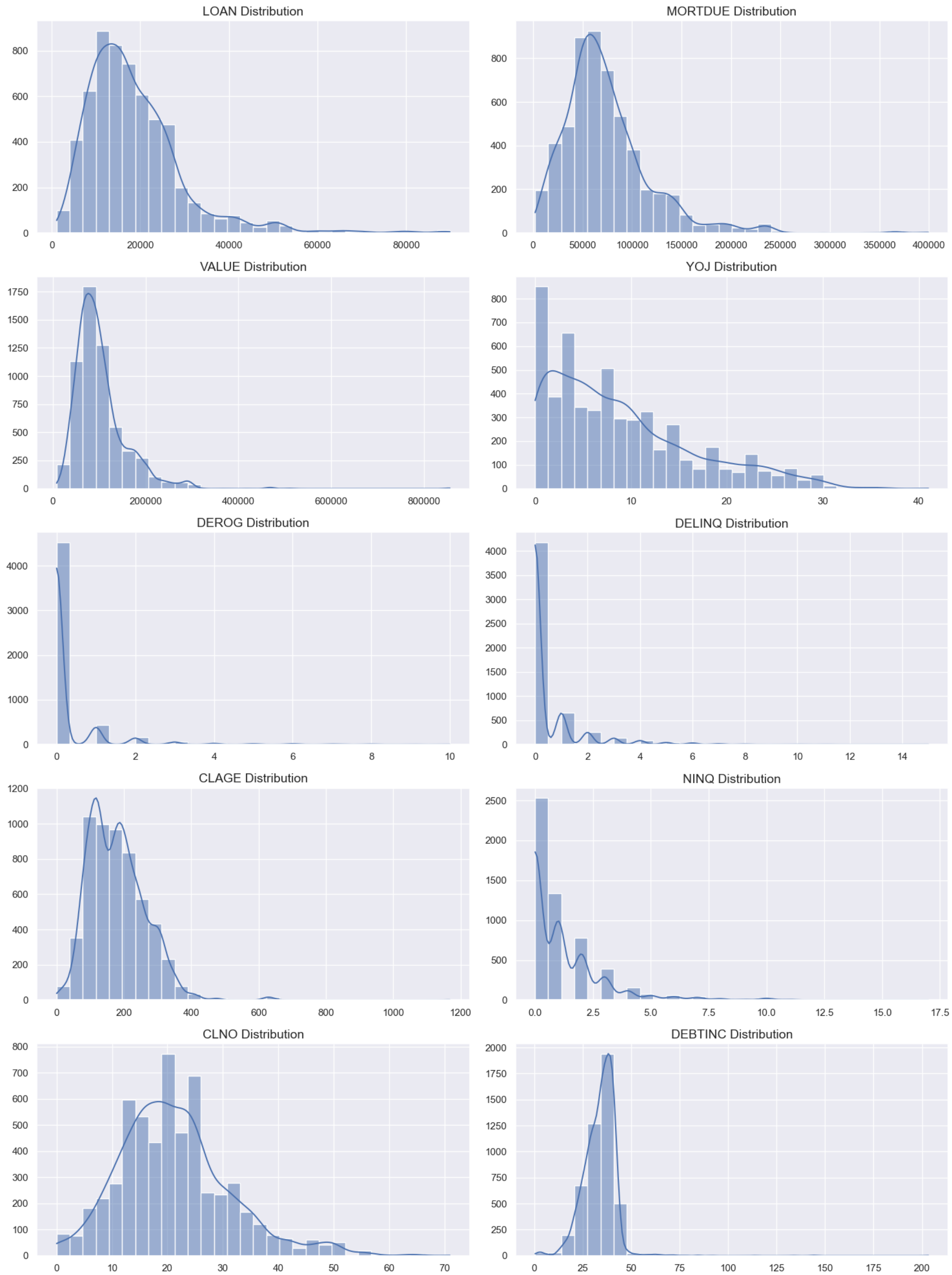
[18]:
```
# The list of key numerical variables for visualization
numerical_vars = ['LOAN', 'MORTDUE', 'VALUE', 'YOJ', 'DEROG', 'DELINQ',␣
 ↪'CLAGE', 'NINQ', 'CLNO', 'DEBTINC']

# Initialize the subplot function using matplotlib
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create a histogram for each numerical variable
for i, var in enumerate(numerical_vars):
    sns.histplot(df[var], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(var + ' Distribution', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjust the layout
plt.tight_layout()
plt.show()
```



[19]:

```
# Define the list of key numerical variables for visualization
```

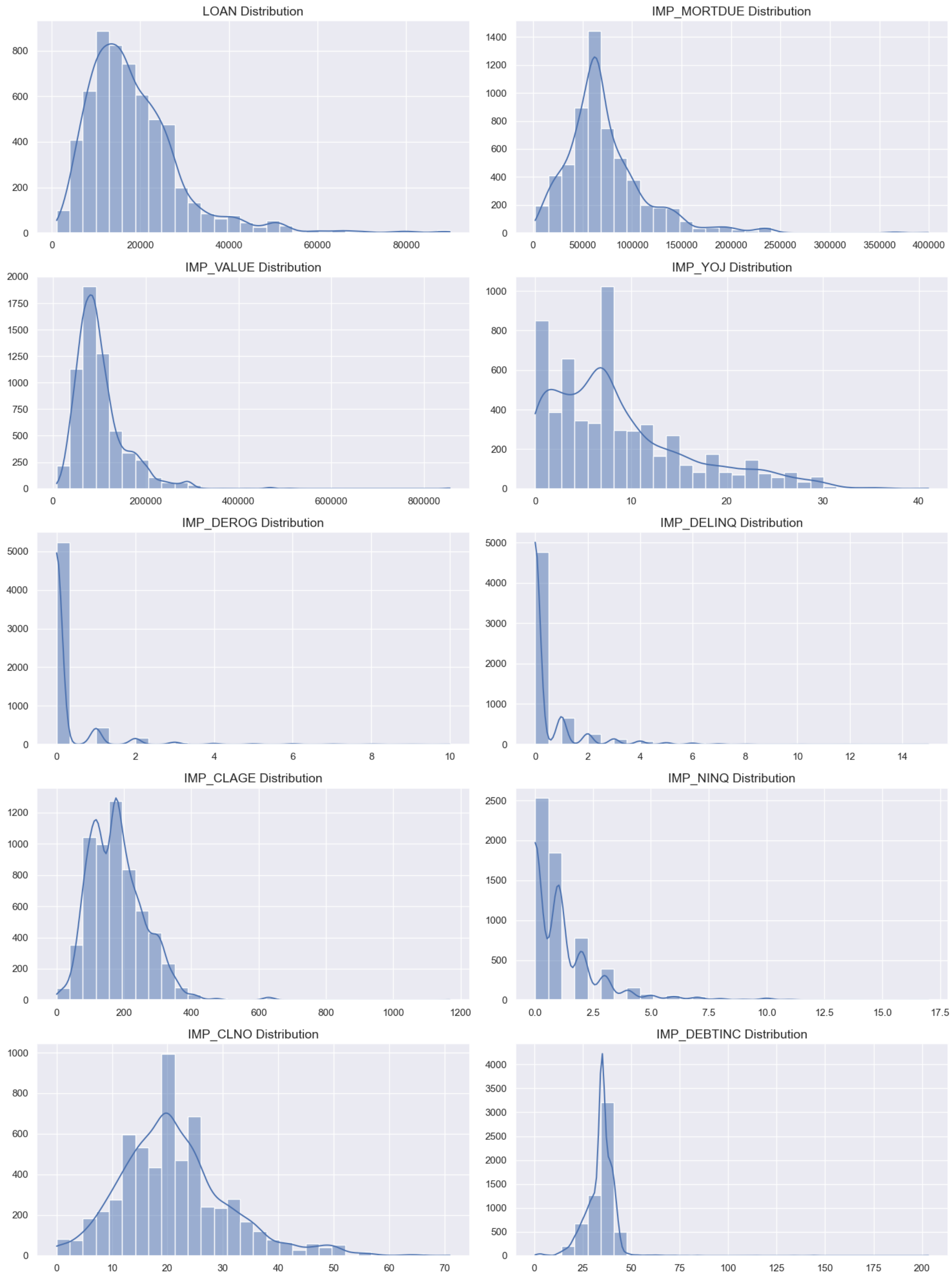
```
numerical_vars = ['LOAN', 'IMP_MORTDUE', 'IMP_VALUE', 'IMP_YOJ', 'IMP_DEROG',␣
↪'IMP_DELINQ', 'IMP_CLAGE', 'IMP_NINQ', 'IMP_CLNO', 'IMP_DEBTINC']

# Initialize the subplot function using matplotlib
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create a histogram for each numerical variable
for i, var in enumerate(numerical_vars):
    sns.histplot(df[var], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(var + ' Distribution', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjust the layout
plt.tight_layout()
plt.show()
```



[20]:
```
'''
Here are the observed findings from the histograms after filling missing values:
```

```
All the variables, including Loan Amount (LOAN), Mortgage Due (MORTDUE),␣
↪Property Value (VALUE),
Years of Employment (YOJ), Derogatory Reports (DEROG), Delinquent Credit Lines␣
↪(DELINQ),
Age of Oldest Credit Line (CLAGE), Number of Recent Credit Lines (NINQ), Number␣
↪of Credit Lines (CLNO)
, and Debt-to-Income Ratio (DEBTINC) show right-skewed distributions.
This means they mostly bunch up on the lower end with fewer instances␣
↪stretching out towards higher values.
The process of filling in missing data appears to have preserved the overall␣
↪shape of these distributions,
with no significant changes in their values.
'''
```

[20]: '\nHere are the observed findings from the histograms after filling missing values:\n\nAll the variables, including Loan Amount (LOAN), Mortgage Due (MORTDUE), Property Value (VALUE), \nYears of Employment (YOJ), Derogatory Reports (DEROG), Delinquent Credit Lines (DELINQ), \nAge of Oldest Credit Line (CLAGE), Number of Recent Credit Lines (NINQ), Number of Credit Lines (CLNO)\n, and Debt-to-Income Ratio (DEBTINC) show right-skewed distributions. \nThis means they mostly bunch up on the lower end with fewer instances stretching out towards higher values. \nThe process of filling in missing data appears to have preserved the overall shape of these distributions, \nwith no significant changes in their values.\n'

[21]:
```
'''
 I want to create a new dataframe with just the filled-in columns,
 and I'll make sure it doesn't have any columns with missing values , object␣
↪types, and target values.
'''
TARGET_COLUMNS = ['TARGET_BAD_FLAG', 'TARGET_LOSS_AMT']

# Identify numeric columns (exclude object type)
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# Exclude target columns from numeric columns
non_target_numeric_cols = [col for col in numeric_cols if col not in␣
↪TARGET_COLUMNS]

# Exclude columns with any missing values
final_cols = [col for col in non_target_numeric_cols if not df[col].isnull().
↪any()]

# Create a new DataFrame df2 with the specified columns from df
df2 = df[final_cols].copy()
```

```
# Now df2 contains only non-target, non-object, and non-missing value columns␣
↪from df

df2.head(5)
```

[21]:

```
   LOAN  IMP_MORTDUE  IMP_VALUE  IMP_YOJ  IMP_DEROG  IMP_DELINQ   IMP_CLAGE  \
0  1100      25860.0    39025.0     10.5        0.0         0.0   94.366667
1  1300      70053.0    68400.0      7.0        0.0         2.0  121.833333
2  1500      13500.0    16700.0      4.0        0.0         0.0  149.466667
3  1500      63508.0    86908.0      7.0        0.0         0.0  172.432355
4  1700      97800.0   112000.0      3.0        0.0         0.0   93.333333

   IMP_NINQ  IMP_CLNO  IMP_DEBTINC  OHE_REASON_DebtCon  OHE_REASON_HomeImp  \
0       1.0       9.0    34.880462                   0                   1
1       0.0      14.0    34.880462                   0                   1
2       1.0      10.0    34.880462                   0                   1
3       1.0      20.0    34.880462                   0                   0
4       0.0      14.0    34.880462                   0                   1

   OHE_REASON_Missing  OHE_JOB_Mgr  OHE_JOB_Missing  OHE_JOB_Office  \
0                   0            0                0               0
1                   0            0                0               0
2                   0            0                0               0
3                   1            0                1               0
4                   0            0                0               1

   OHE_JOB_Other  OHE_JOB_ProfExe  OHE_JOB_Sales  OHE_JOB_Self
0              1                0              0             0
1              1                0              0             0
2              1                0              0             0
3              0                0              0             0
4              0                0              0             0
```

[22]:

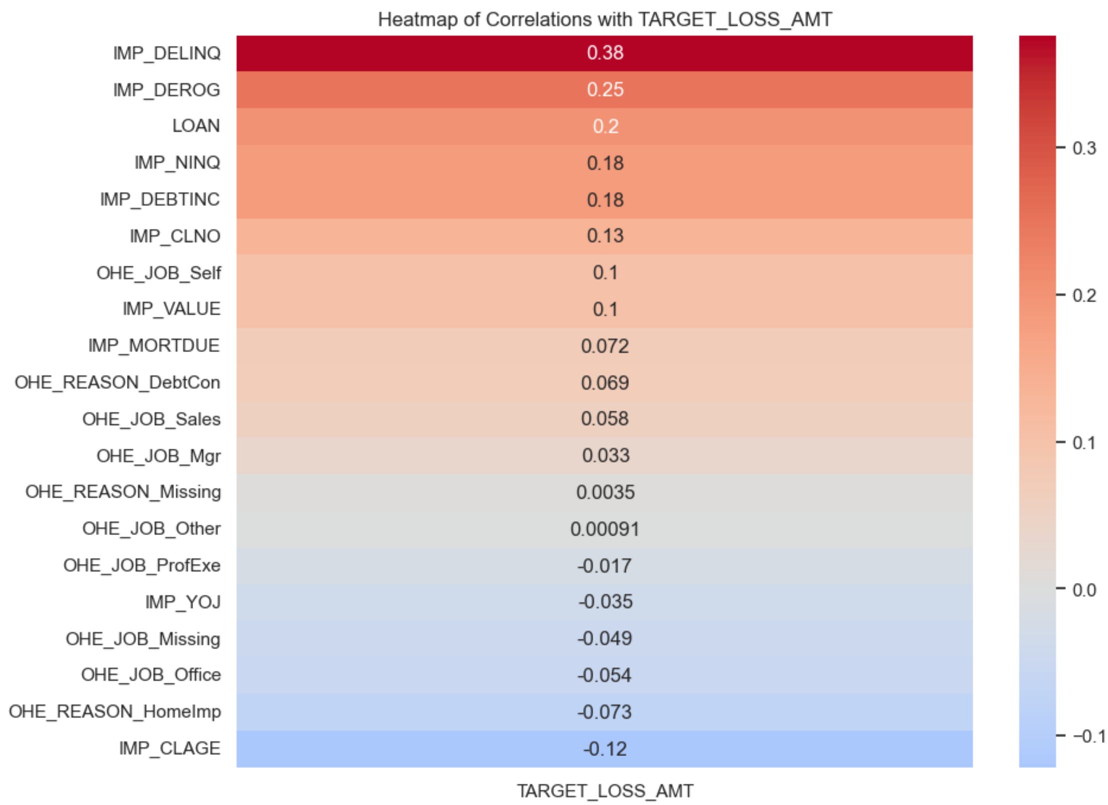
```
'''
After creating the df2 dataframe,
I want to calculate the correlation between each variable and the target␣
↪variable 'TARGET_LOSS_AMT'
'''
# Calculate the correlation of df2's variables with TARGET_LOSS_AMT
correlation_with_target_loss = df2.corrwith(df['TARGET_LOSS_AMT']).
↪sort_values(ascending=False)

# Calculate the correlation of each column in df2 with TARGET_LOSS_AMT from df
correlations = df2.apply(lambda col: col.corr(df['TARGET_LOSS_AMT']))
sorted_correlations = correlations.sort_values(ascending=False)
```

```python
# Print each column's correlation with TARGET_LOSS_AMT
print("Correlation of each column in df2 with TARGET_LOSS_AMT:")
print(sorted_correlations)

# I am trying to create a heatmap of the correlations with TARGET_LOSS_AMT
correlation_matrix = correlation_with_target_loss.
 ↪to_frame(name='TARGET_LOSS_AMT')
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Heatmap of Correlations with TARGET_LOSS_AMT')
plt.show()
```

```
Correlation of each column in df2 with TARGET_LOSS_AMT:
IMP_DELINQ            0.376198
IMP_DEROG             0.249934
LOAN                  0.199414
IMP_NINQ              0.183584
IMP_DEBTINC           0.182208
IMP_CLNO              0.134767
OHE_JOB_Self          0.101451
IMP_VALUE             0.100304
IMP_MORTDUE           0.071745
OHE_REASON_DebtCon    0.069198
OHE_JOB_Sales         0.057618
OHE_JOB_Mgr           0.033213
OHE_REASON_Missing    0.003464
OHE_JOB_Other         0.000908
OHE_JOB_ProfExe      -0.017369
IMP_YOJ              -0.034983
OHE_JOB_Missing      -0.048824
OHE_JOB_Office       -0.054159
OHE_REASON_HomeImp   -0.073194
IMP_CLAGE            -0.121376
dtype: float64
```



[23]:
```
'''
The heatmap provided displays the correlation coefficients between various
 ↪imputed variables and the TARGET_LOSS_AMT. Here are the findings:

Positive Correlations: Most variables show a positive correlation with
 ↪TARGET_LOSS_AMT, meaning as their values increase, the loss amount tends to
 ↪increase as well. The variable IMP_DELINQ has the strongest positive
 ↪correlation at 0.38, followed by IMP_DEROG at 0.25, suggesting that an
 ↪increase in delinquent credit lines or derogatory reports is associated with
 ↪a higher loss amount.

Negative Correlations: Two variables, IMP_YOJ and IMP_CLAGE, show a negative
 ↪correlation with TARGET_LOSS_AMT (at -0.035 and -0.12 respectively),
 ↪indicating that as the years of employment and the age of the oldest credit
 ↪line increase, the loss amount tends to decrease.

Weak Correlations:  Some variables like IMP_MORTDUE have a very low positive
 ↪correlation (0.072)
'''
```

[23]: '\nThe heatmap provided displays the correlation coefficients between various imputed variables and the TARGET_LOSS_AMT. Here are the findings:\n\nPositive Correlations: Most variables show a positive correlation with TARGET_LOSS_AMT, meaning as their values increase, the loss amount tends to increase as well. The variable IMP_DELINQ has the strongest positive correlation at 0.38, followed by IMP_DEROG at 0.25, suggesting that an increase in delinquent credit lines or derogatory reports is associated with a higher loss amount.\n\nNegative Correlations: Two variables, IMP_YOJ and IMP_CLAGE, show a negative correlation with TARGET_LOSS_AMT (at -0.035 and -0.12 respectively), indicating that as the years of employment and the age of the oldest credit line increase, the loss amount tends to decrease.\n\nWeak Correlations: Some variables like IMP_MORTDUE have a very low positive correlation (0.072) \n'

2 BINGO BONUS WORK

[24]:
```
'''
I am trying to fill in the missing value with different methods and see which␣
↪one is better

[Column Name]+ _Missing is replacing with Median
[Column Name]+ _Missing2 is replacing with Mean
IMP_ +[Column Name] is remove outliers and replace with Median
IMP2_ +[Column Name] is remove outliers and replace with Mean

'''
```

[24]: '\nI am trying to fill in the missing value with different methods and see which one is better\n\n[Column Name]+ _Missing is replacing with Median \n[Column Name]+ _Missing2 is replacing with Mean\nIMP_ +[Column Name] is remove outliers and replace with Median\nIMP2_ +[Column Name] is remove outliers and replace with Mean\n\n'

[25]:
```
#[Column Name]+ _Missing is replacing with Median
# Create flag columns and impute missing values with median
for col in cols_with_missing:
    df[col+'_MISSING'] = df[col].fillna(df[col].median(), inplace=False)
```

[26]:
```
#[Column Name]+ _Missing2 is replacing with Mean
# Create flag columns and impute missing values with mean
for col in cols_with_missing:
    df[col+'_MISSING2'] = df[col].fillna(df[col].mean(), inplace=False)
```

[27]:
```
#IMP2_ +[Column Name] is remove outliers and replace with Mean

for col in cols_with_missing:
    # 1. Identify Outliers using the IQR method
    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

    # 2. Remove Outliers - Replace outliers in a copy of the column with NaN
    temp_col = df[col].copy()
    temp_col[(temp_col < lower_bound) | (temp_col > upper_bound)] = np.nan

    # 3. Calculate mean of the column with outliers removed
    median_val = temp_col.mean()

    # 4. Create new column for imputed values, fill missing values with the␣
 ↪calculated median
    df['IMP2_'+col] = df[col].fillna(median_val)
```

[28]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_M = ['MORTDUE']

# Loop through each column to get and print the descriptive statistics for the␣
 ↪original, flag, and imputed columns
for col in cols_with_missing_M:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
 ↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for MORTDUE:
             MORTDUE  MORTDUE_MISSING  MORTDUE_MISSING2   IMP_MORTDUE  \
count    5442.000000      5960.000000       5960.000000   5960.000000
mean    73760.817200     73001.041812      73760.817200  72869.716644
std     44457.609458     42552.726779      42481.395689  42579.485794
min      2063.000000      2063.000000       2063.000000   2063.000000
25%     46276.000000     48139.000000      48139.000000  48139.000000
50%     65019.000000     65019.000000      69529.000000  63508.000000
75%     91488.000000     88200.250000      88200.250000  88200.250000
```

```
max     399550.000000    399550.000000     399550.000000  399550.000000

         IMP2_MORTDUE
count     5960.000000
mean     73227.437618
std      42516.565166
min       2063.000000
25%      48139.000000
50%      67623.862942
75%      88200.250000
max     399550.000000
```

[29]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_v = ['VALUE']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
for col in cols_with_missing_v:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for VALUE:
               VALUE  VALUE_MISSING  VALUE_MISSING2      IMP_VALUE  \
count    5848.000000    5960.000000     5960.000000    5960.000000
mean   101776.048741  101540.387423   101776.048741  101496.649168
std     57385.775334   56869.436682    56843.931566   56879.779380
min      8000.000000    8000.000000     8000.000000    8000.000000
25%     66075.500000   66489.500000    66489.500000   66489.500000
50%     89235.500000   89235.500000    90000.000000   88310.500000
75%    119824.250000  119004.750000   119004.750000  119004.750000
max    855909.000000  855909.000000   855909.000000  855909.000000

           IMP2_VALUE
count     5960.000000
```

```
mean    101604.342443
std      56857.473131
min       8000.000000
25%      66489.500000
50%      90000.000000
75%     119004.750000
max     855909.000000
```

[30]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_y = ['YOJ']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
for col in cols_with_missing_y:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for YOJ:
               YOJ  YOJ_MISSING  YOJ_MISSING2      IMP_YOJ     IMP2_YOJ
count  5445.000000  5960.000000   5960.000000  5960.000000  5960.000000
mean      8.922268     8.756166      8.922268     8.756166     8.889934
std       7.573982     7.259424      7.239301     7.259424     7.240065
min       0.000000     0.000000      0.000000     0.000000     0.000000
25%       3.000000     3.000000      3.000000     3.000000     3.000000
50%       7.000000     7.000000      8.000000     7.000000     8.000000
75%      13.000000    12.000000     12.000000    12.000000    12.000000
max      41.000000    41.000000     41.000000    41.000000    41.000000
```

[31]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_d = ['DEROG']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
```

```
for col in cols_with_missing_d:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for DEROG:
             DEROG  DEROG_MISSING  DEROG_MISSING2     IMP_DEROG    IMP2_DEROG
count  5252.000000    5960.000000     5960.000000  5960.000000  5960.000000
mean      0.254570       0.224329        0.254570     0.224329     0.224329
std       0.846047       0.798458        0.794198     0.798458     0.798458
min       0.000000       0.000000        0.000000     0.000000     0.000000
25%       0.000000       0.000000        0.000000     0.000000     0.000000
50%       0.000000       0.000000        0.000000     0.000000     0.000000
75%       0.000000       0.000000        0.000000     0.000000     0.000000
max      10.000000      10.000000       10.000000    10.000000    10.000000
```

[32]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_D = ['DELINQ']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
for col in cols_with_missing_D:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
↪imp2_col]

    # Calculate and print descriptive statistics
```

```
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for DELINQ:
            DELINQ  DELINQ_MISSING  DELINQ_MISSING2   IMP_DELINQ  IMP2_DELINQ
count  5380.000000     5960.000000      5960.000000  5960.000000  5960.000000
mean      0.449442        0.405705         0.449442     0.405705     0.405705
std       1.127266        1.079256         1.071002     1.079256     1.079256
min       0.000000        0.000000         0.000000     0.000000     0.000000
25%       0.000000        0.000000         0.000000     0.000000     0.000000
50%       0.000000        0.000000         0.000000     0.000000     0.000000
75%       0.000000        0.000000         0.449442     0.000000     0.000000
max      15.000000       15.000000        15.000000    15.000000    15.000000
```

[33]:

```
# One of the columns with missing values that I want to analyze
cols_with_missing_c = ['CLAGE']

# Loop through each column to get and print the descriptive statistics for the␣
 ↪original, flag, and imputed columns
for col in cols_with_missing_c:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
 ↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for CLAGE:
             CLAGE  CLAGE_MISSING  CLAGE_MISSING2    IMP_CLAGE   IMP2_CLAGE
count  5652.000000    5960.000000     5960.000000  5960.000000  5960.000000
mean    179.766275     179.440725      179.766275   179.387274   179.609230
std      85.810092      83.574697       83.563059    83.578832    83.565767
min       0.000000       0.000000        0.000000     0.000000     0.000000
25%     115.116702     117.371430      117.371430   117.371430   117.371430
50%     173.466667     173.466667      178.076005   172.432355   176.727344
75%     231.562278     227.143058      227.143058   227.143058   227.143058
max    1168.233561    1168.233561     1168.233561  1168.233561  1168.233561
```

[34]:
```
# One of the columns with missing values that I want to analyze
cols_with_missing_n = ['NINQ']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
for col in cols_with_missing_n:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for NINQ:
              NINQ  NINQ_MISSING  NINQ_MISSING2     IMP_NINQ    IMP2_NINQ
count  5450.000000   5960.000000    5960.000000  5960.000000  5960.000000
mean      1.186055      1.170134       1.186055     1.170134     1.166905
std       1.728675      1.653866       1.653046     1.653866     1.654232
min       0.000000      0.000000       0.000000     0.000000     0.000000
25%       0.000000      0.000000       0.000000     0.000000     0.000000
50%       1.000000      1.000000       1.000000     1.000000     0.962261
75%       2.000000      2.000000       2.000000     2.000000     2.000000
max      17.000000     17.000000      17.000000    17.000000    17.000000
```

[35]:
```
# One of the columns with missing values that I want to analyze
cols_with_missing_C = ['CLNO']

# Loop through each column to get and print the descriptive statistics for the␣
↪original, flag, and imputed columns
for col in cols_with_missing_C:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col
```

```
    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
 ↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for CLNO:
              CLNO  CLNO_MISSING  CLNO_MISSING2     IMP_CLNO    IMP2_CLNO
count  5738.000000   5960.000000    5960.000000  5960.000000  5960.000000
mean     21.296096     21.247819      21.296096    21.247819    21.254561
std      10.138933      9.951308       9.948280     9.951308     9.950521
min       0.000000      0.000000       0.000000     0.000000     0.000000
25%      15.000000     15.000000      15.000000    15.000000    15.000000
50%      20.000000     20.000000      21.000000    20.000000    20.181011
75%      26.000000     26.000000      26.000000    26.000000    26.000000
max      71.000000     71.000000      71.000000    71.000000    71.000000
```

[36]:
```
# One of the columns with missing values that I want to analyze
cols_with_missing_De = ['DEBTINC']

# Loop through each column to get and print the descriptive statistics for the␣
 ↪original, flag, and imputed columns
for col in cols_with_missing_De:
    # Define the column names for original, missing flags and imputed columns
    original_col = col
    missing_flag = col + '_MISSING'
    missing2_flag = col + '_MISSING2'
    imp_col = 'IMP_' + col
    imp2_col = 'IMP2_' + col

    # Select the columns to compare
    columns_to_compare = [original_col, missing_flag, missing2_flag, imp_col,␣
 ↪imp2_col]

    # Calculate and print descriptive statistics
    descriptive_stats = df[columns_to_compare].describe()
    print(f"Descriptive statistics for {col}:")
    print(descriptive_stats, "\n")
```

```
Descriptive statistics for DEBTINC:
           DEBTINC  DEBTINC_MISSING  DEBTINC_MISSING2  IMP_DEBTINC  \
count  4693.000000      5960.000000       5960.000000  5960.000000
mean     33.779915        34.000651         33.779915    34.013874
```

```
std        8.601746        7.644528        7.632713    7.645985
min        0.524499        0.524499        0.524499    0.524499
25%       29.140031       30.763159       30.763159   30.763159
50%       34.818262       34.818262       33.779915   34.880462
75%       39.003141       37.949892       37.949892   37.949892
max      203.312149      203.312149      203.312149  203.312149

       IMP2_DEBTINC
count   5960.000000
mean      33.779285
std        7.632713
min        0.524499
25%       30.763159
50%       33.776951
75%       37.949892
max      203.312149
```

[37]:

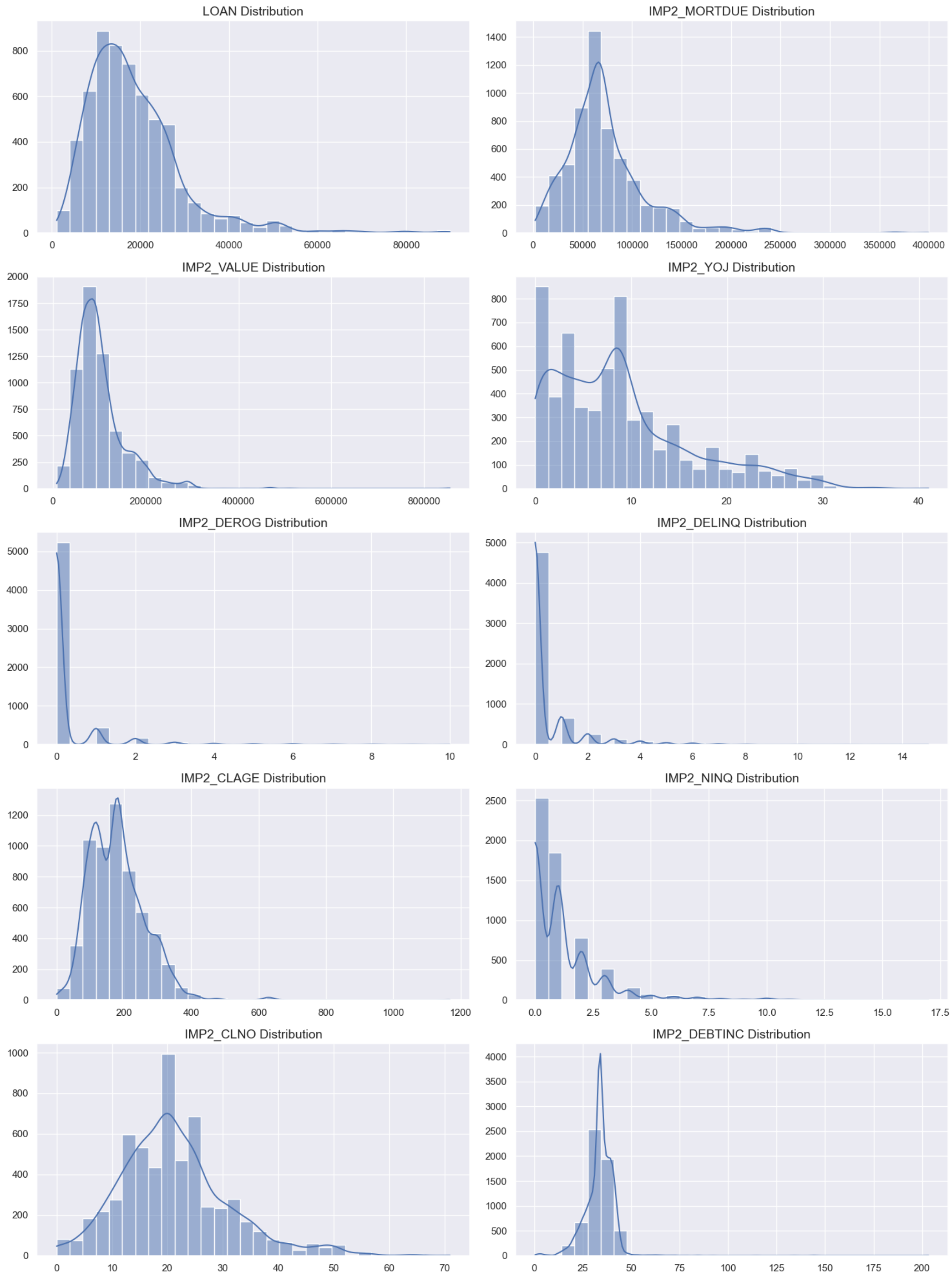
```
# Define the list of key numerical variables for visualization
numerical_vars = ['LOAN', 'IMP2_MORTDUE', 'IMP2_VALUE', 'IMP2_YOJ',␣
 ↪'IMP2_DEROG', 'IMP2_DELINQ', 'IMP2_CLAGE', 'IMP2_NINQ', 'IMP2_CLNO',␣
 ↪'IMP2_DEBTINC']

# Initialize the subplot function using matplotlib
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create a histogram for each numerical variable
for i, var in enumerate(numerical_vars):
    sns.histplot(df[var], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(var + ' Distribution', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjust the layout
plt.tight_layout()
plt.show()
```



[38]:
```
# Define the list of key numerical variables for visualization
```

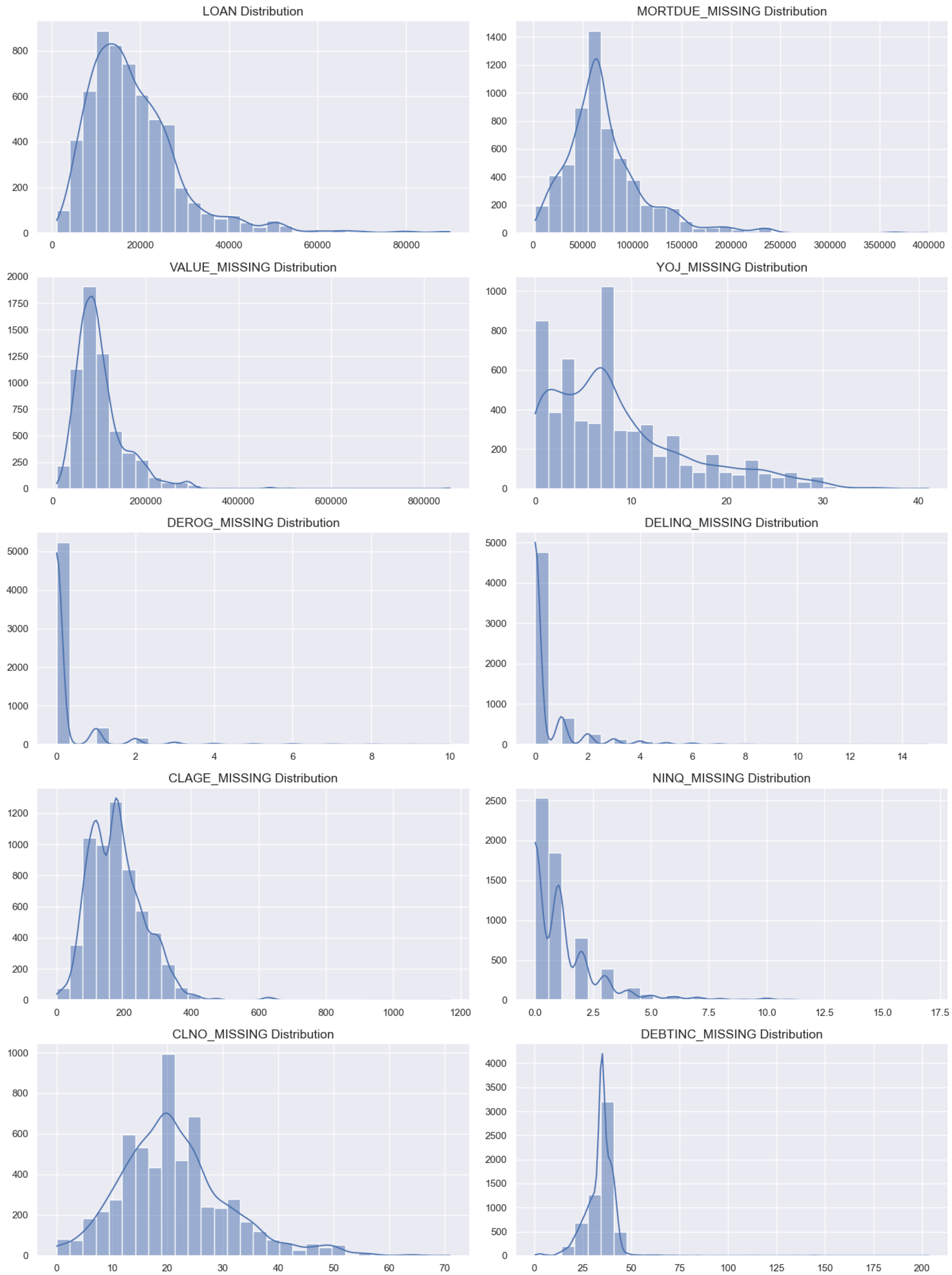
```
numerical_vars = ['LOAN', 'MORTDUE_MISSING', 'VALUE_MISSING', 'YOJ_MISSING',␣
↪'DEROG_MISSING', 'DELINQ_MISSING', 'CLAGE_MISSING', 'NINQ_MISSING',␣
↪'CLNO_MISSING', 'DEBTINC_MISSING']

# Initialize the subplot function using matplotlib
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create a histogram for each numerical variable
for i, var in enumerate(numerical_vars):
    sns.histplot(df[var], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(var + ' Distribution', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjust the layout
plt.tight_layout()
plt.show()
```



[39]:
```
# Define the list of key numerical variables for visualization
```

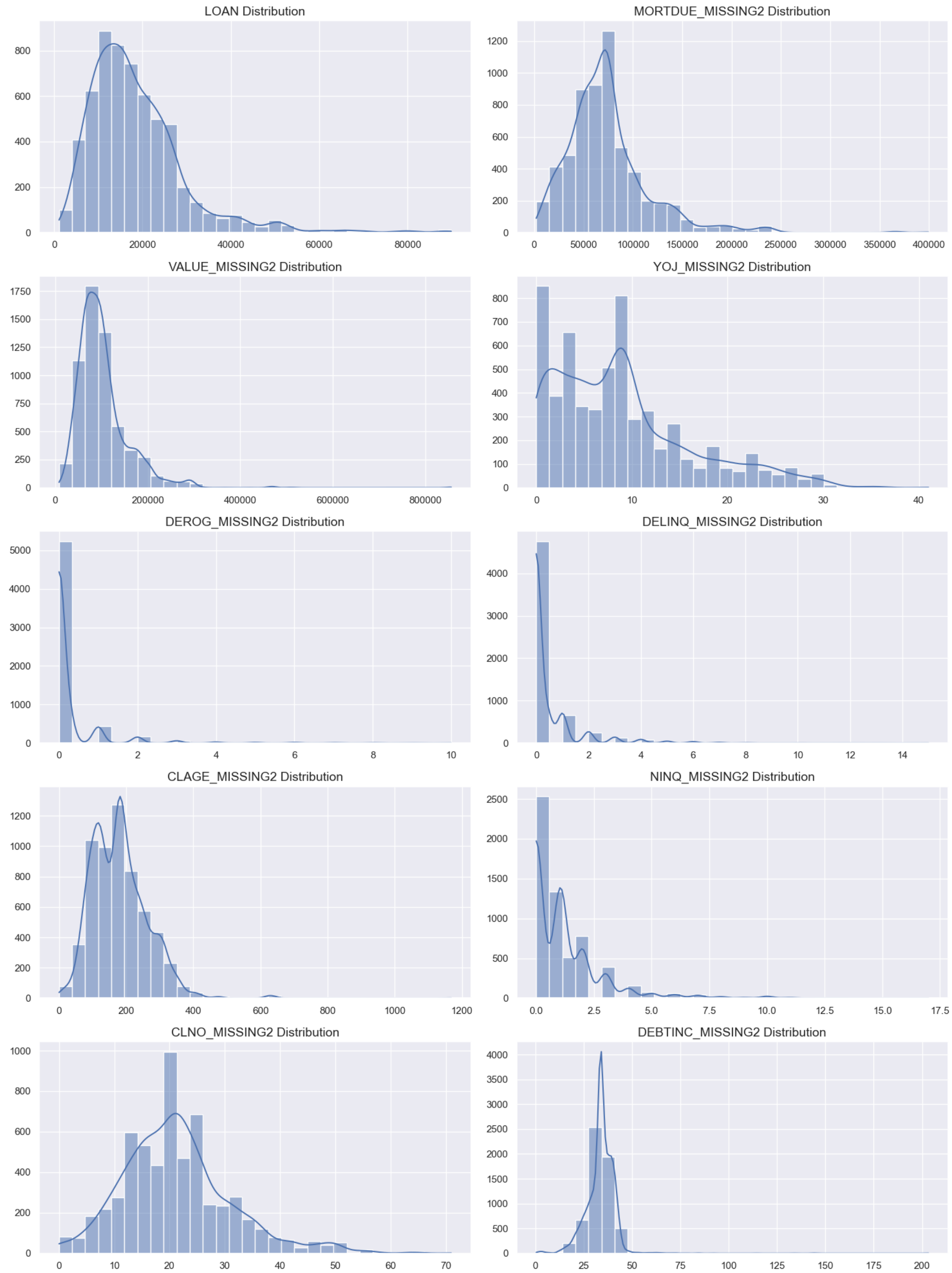
```
numerical_vars = ['LOAN', 'MORTDUE_MISSING2', 'VALUE_MISSING2', 'YOJ_MISSING2', 'DEROG_MISSING2', 'DELINQ_MISSING2', 'CLAGE_MISSING2', 'NINQ_MISSING2', 'CLNO_MISSING2', 'DEBTINC_MISSING2']

# Initialize the subplot function using matplotlib
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create a histogram for each numerical variable
for i, var in enumerate(numerical_vars):
    sns.histplot(df[var], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(var + ' Distribution', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjust the layout
plt.tight_layout()
plt.show()
```



[40]:

```
'''
After trying various ways to handle missing data, like:
```

```
[Column Name]+ _Missing is replacing with Median
[Column Name]+ _Missing2 is replacing with Mean
IMP_ +[Column Name] is remove outliers and replace with Median
IMP2_ +[Column Name] is remove outliers and replace with Mean
I noticed that the distributions and mean values stayed pretty similar across␣
↪these methods.
However, the approach where I took out the outliers and used the median,
labeled as "IMP_[Column Name]", seems to make the most sense to me.

***I've discussed with TA Logan, and he mentioned that it was acceptable,␣
↪despite the PDF cut off some of the output. ***
'''
```

[40]:
```
'\nAfter trying various ways to handle missing data, like:\n\n[Column Name]+
_Missing is replacing with Median \n[Column Name]+ _Missing2 is replacing with
Mean\nIMP_ +[Column Name] is remove outliers and replace with Median\nIMP2_
+[Column Name] is remove outliers and replace with Mean\nI noticed that the
distributions and mean values stayed pretty similar across these methods.
\nHowever, the approach where I took out the outliers and used the median,
\nlabeled as "IMP_[Column Name]", seems to make the most sense to
me.\n\n***I\'ve discussed with TA Logan, and he mentioned that it was
acceptable, despite the PDF cut off some of the output. ***\n'
```

[41]:
```
!jupyter nbconvert --to pdf Assignment1_kwok.ipynb
```