Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch:2023-2028	**Due date: 27.01.2025**

Experiment 1: Working with Python packages-Numpy, Scipy, Scikit-Learn, Matplotlib

Aim: To study and explore the fundamental Python libraries used in data science and machine learning, and to understand how different datasets can be analyzed and mapped to appropriate machine learning models using exploratory data analysis techniques.

Libraries Used:

- **NumPy**: Used for numerical computations and efficient handling of multi-dimensional arrays.
- **Pandas**: Used for data manipulation, cleaning, and analysis using DataFrames.
- **Matplotlib**: Used for creating visualizations such as line graphs, bar charts, and histograms.
- **Seaborn**: Used for advanced statistical data visualization with attractive and informative plots.

Objectives performed:

1 Iris Dataset

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder

from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif
from sklearn.model_selection import train_test_split
```

[5]:
```
df=pd.read_csv('Datasets/Iris.csv')
df.drop(columns=['Id'], inplace=True)

df.head()
```

```
[5]:    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
     0            5.1           3.5            1.4           0.2  Iris-setosa
     1            4.9           3.0            1.4           0.2  Iris-setosa
     2            4.7           3.2            1.3           0.2  Iris-setosa
     3            4.6           3.1            1.5           0.2  Iris-setosa
     4            5.0           3.6            1.4           0.2  Iris-setosa
```

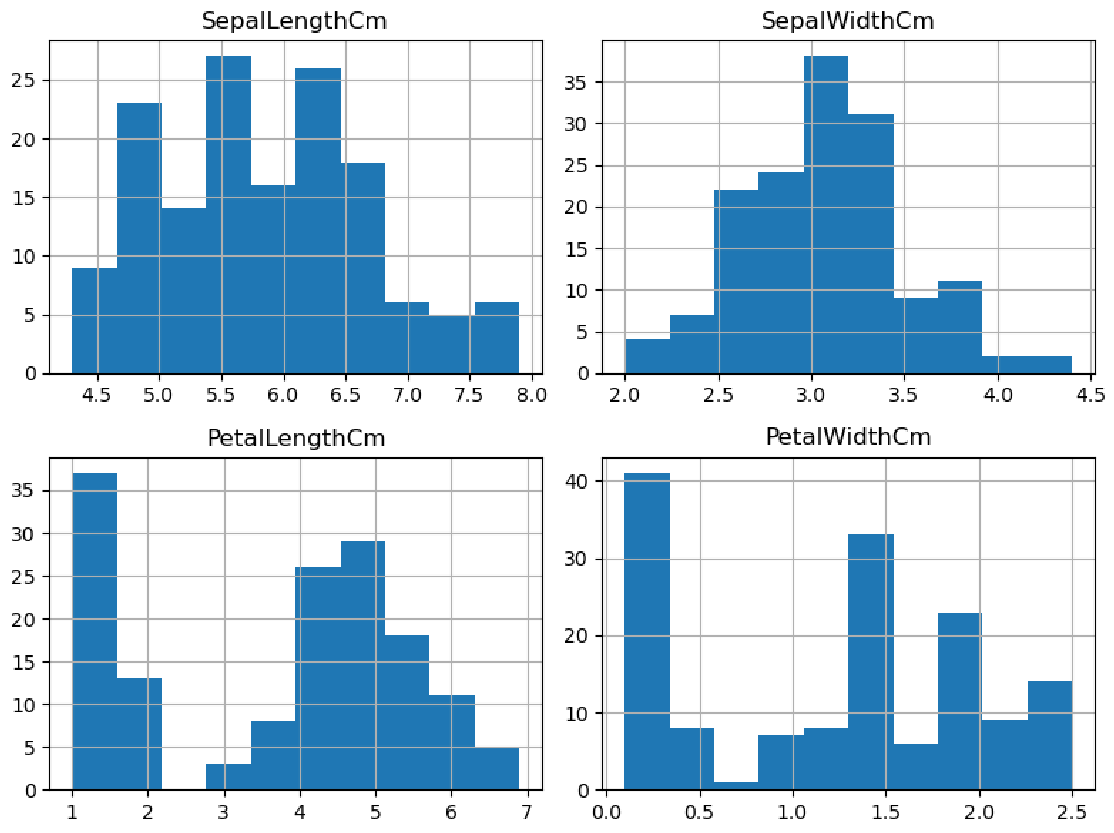
```
[6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

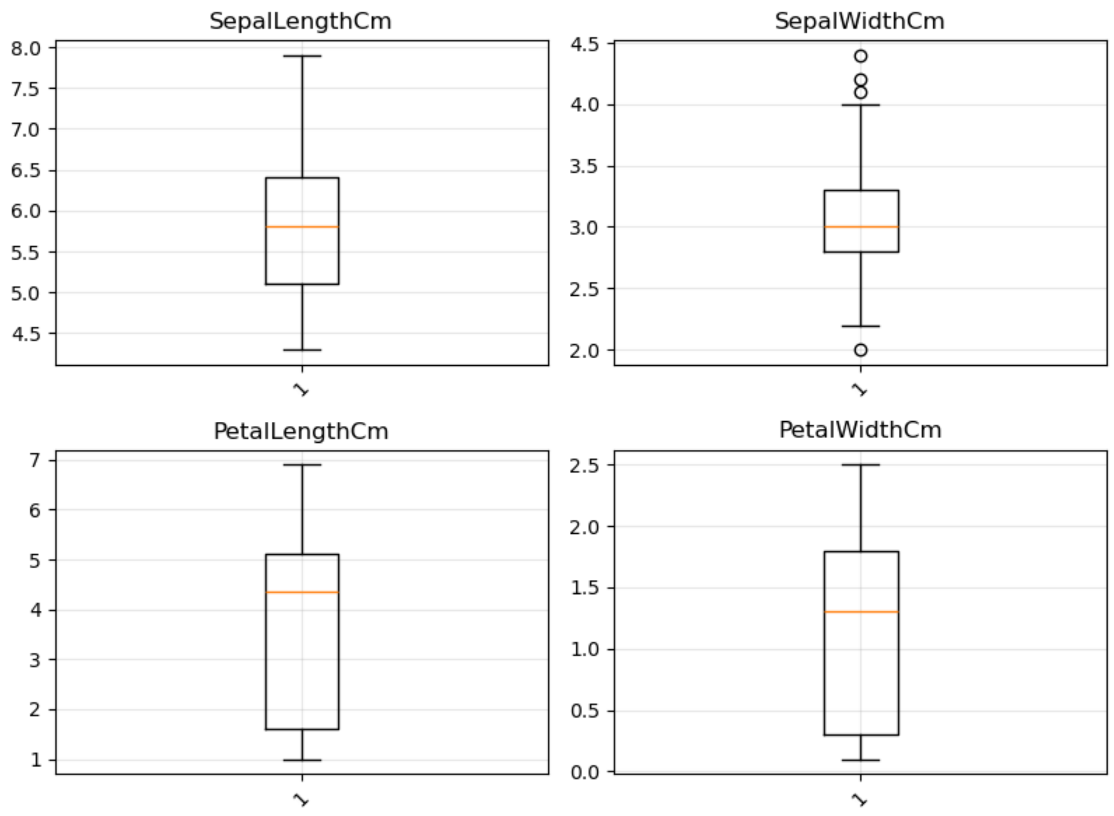
```
[7]: df.describe()
```

```
[7]:        SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
     count     150.000000    150.000000     150.000000    150.000000
     mean        5.843333      3.054000       3.758667      1.198667
     std         0.828066      0.433594       1.764420      0.763161
     min         4.300000      2.000000       1.000000      0.100000
     25%         5.100000      2.800000       1.600000      0.300000
     50%         5.800000      3.000000       4.350000      1.300000
     75%         6.400000      3.300000       5.100000      1.800000
     max         7.900000      4.400000       6.900000      2.500000
```

```
[8]: df[df.columns].hist(figsize=(8,6))
     plt.tight_layout()
     plt.savefig("histogram.png", dpi=300, bbox_inches="tight")
     plt.show()
```

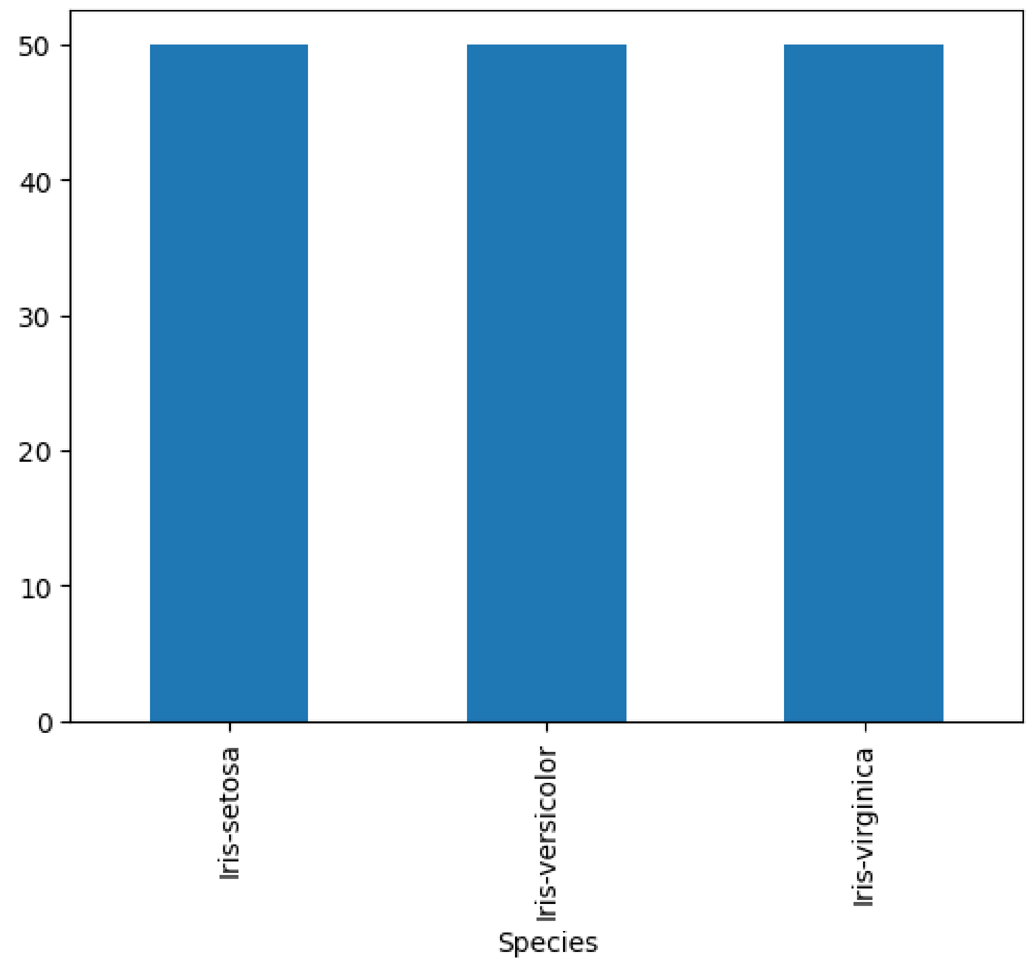


[9]:
```
num_cols = df.select_dtypes(include=['int64','float64']).columns

fig, axes = plt.subplots(2,2, figsize = (8,6))
axes = axes.flatten()

for i, col in enumerate(num_cols):
  axes[i].boxplot(df[col])
  axes[i].set_title(col)
  axes[i].tick_params(axis='x', rotation=45)
  axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig("box_plot.png", dpi=300, bbox_inches="tight")
plt.show()
```

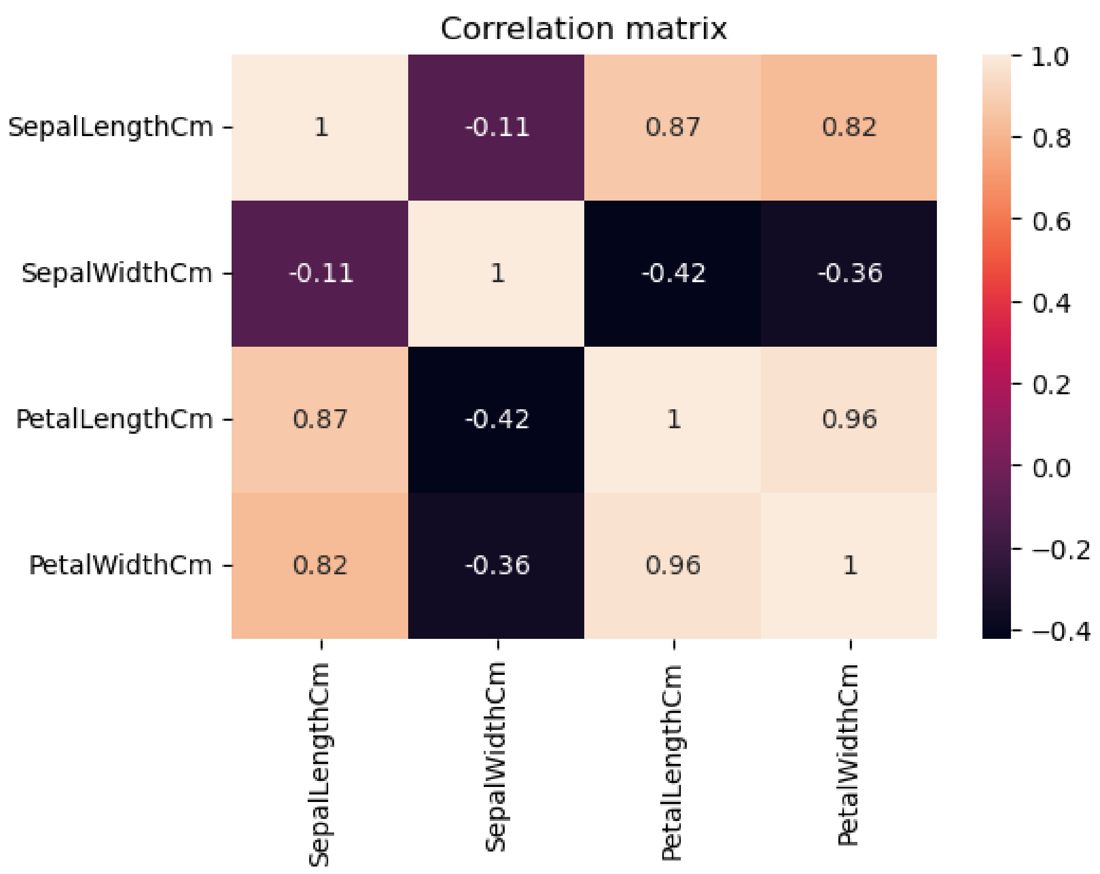


Not much outliers to be concerned about

[10]:
```
df['Species'].value_counts().plot(kind='bar')
axes[i].set_title("Species")
axes[i].set_xlabel('')
axes[i].set_ylabel('Count')
plt.savefig("bar_plot.png", dpi=300, bbox_inches="tight")
```



[11]:
```
plt.figure(figsize=(6,4))
sns.heatmap(df[num_cols].corr(), annot=True)
plt.title("Correlation matrix")
plt.savefig("correlational_matrix.png", dpi=300, bbox_inches="tight")
plt.show()
```



```
[ ]:
```

2 Loan Amount Prediction

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif
from sklearn.model_selection import train_test_split
```

[ ]:
```
#Loan dataset
df = pd.read_csv("/content/loan_data.csv")
print(df)
```

```
       person_age person_gender person_education  person_income  \
0            22.0        female           Master        71948.0
1            21.0        female      High School        12282.0
2            25.0        female      High School        12438.0
3            23.0        female         Bachelor        79753.0
4            24.0          male           Master        66135.0
...           ...           ...              ...            ...
44995        27.0          male        Associate        47971.0
44996        37.0        female        Associate        65800.0
44997        33.0          male        Associate        56942.0
44998        29.0          male         Bachelor        33164.0
44999        24.0          male      High School        51609.0

       person_emp_exp person_home_ownership  loan_amnt        loan_intent  \
0                   0                  RENT    35000.0           PERSONAL
1                   0                   OWN     1000.0          EDUCATION
2                   3              MORTGAGE     5500.0            MEDICAL
3                   0                  RENT    35000.0            MEDICAL
4                   1                  RENT    35000.0            MEDICAL
...               ...                   ...        ...                ...
44995               6                  RENT    15000.0            MEDICAL
44996              17                  RENT     9000.0    HOMEIMPROVEMENT
44997               7                  RENT     2771.0  DEBTCONSOLIDATION
44998               4                  RENT    12000.0          EDUCATION
44999               1                  RENT     6665.0  DEBTCONSOLIDATION

       loan_int_rate  loan_percent_income  cb_person_cred_hist_length  \
0              16.02                 0.49                         3.0
1              11.14                 0.08                         2.0
2              12.87                 0.44                         3.0
3              15.23                 0.44                         2.0
4              14.27                 0.53                         4.0
...              ...                  ...                         ...
44995          15.66                 0.31                         3.0
44996          14.07                 0.14                        11.0
44997          10.02                 0.05                        10.0
44998          13.23                 0.36                         6.0
44999          17.05                 0.13                         3.0

       credit_score previous_loan_defaults_on_file  loan_status
```

```
0                561                              No           1
1                504                             Yes           0
2                635                              No           1
3                675                              No           1
4                586                              No           1
...              ...                             ...         ...
44995            645                              No           1
44996            621                              No           1
44997            668                              No           1
44998            604                              No           1
44999            628                              No           1

[45000 rows x 14 columns]
```

[ ]:
```
print("Shape")
df.shape
```

```
Shape
```

[ ]:
```
(45000, 14)
```

[ ]:
```
print("Describe")
df.describe()
```

```
Describe
```

[ ]:
```
         person_age  person_income  person_emp_exp     loan_amnt  \
count  45000.000000   4.500000e+04    45000.000000  45000.000000
mean      27.764178   8.031905e+04        5.410333   9583.157556
std        6.045108   8.042250e+04        6.063532   6314.886691
min       20.000000   8.000000e+03        0.000000    500.000000
25%       24.000000   4.720400e+04        1.000000   5000.000000
50%       26.000000   6.704800e+04        4.000000   8000.000000
75%       30.000000   9.578925e+04        8.000000  12237.250000
max      144.000000   7.200766e+06      125.000000  35000.000000

       loan_int_rate  loan_percent_income  cb_person_cred_hist_length  \
count   45000.000000         45000.000000                45000.000000
mean       11.006606             0.139725                    5.867489
std         2.978808             0.087212                    3.879702
min         5.420000             0.000000                    2.000000
25%         8.590000             0.070000                    3.000000
50%        11.010000             0.120000                    4.000000
75%        12.990000             0.190000                    8.000000
max        20.000000             0.660000                   30.000000

       credit_score   loan_status
count  45000.000000  45000.000000
mean     632.608756      0.222222
```

```
std       50.435865      0.415744
min      390.000000      0.000000
25%      601.000000      0.000000
50%      640.000000      0.000000
75%      670.000000      0.000000
max      850.000000      1.000000
```

Outliers found! age = 144, person_income = 72 lakh

```
[ ]: print("columns")
     df.columns
```

```
columns
```

```
[ ]: Index(['person_age', 'person_gender', 'person_education', 'person_income',
            'person_emp_exp', 'person_home_ownership', 'loan_amnt', 'loan_intent',
            'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length',
            'credit_score', 'previous_loan_defaults_on_file', 'loan_status'],
           dtype='object')
```

```
[ ]: df.isnull().sum()
     (df.isnull().mean() * 100).sort_values(ascending=False)
```

```
[ ]: person_age                        0.0
     person_gender                     0.0
     person_education                  0.0
     person_income                     0.0
     person_emp_exp                    0.0
     person_home_ownership             0.0
     loan_amnt                         0.0
     loan_intent                       0.0
     loan_int_rate                     0.0
     loan_percent_income               0.0
     cb_person_cred_hist_length        0.0
     credit_score                      0.0
     previous_loan_defaults_on_file    0.0
     loan_status                       0.0
     dtype: float64
```

No Null values.

```
[ ]: #including previous_loan_defaults_on_file
     df['previous_loan_defaults_on_file'] = df['previous_loan_defaults_on_file'].
      ↪map({
         'Yes': 1,
         'No': 0
     })

     #column selection
```

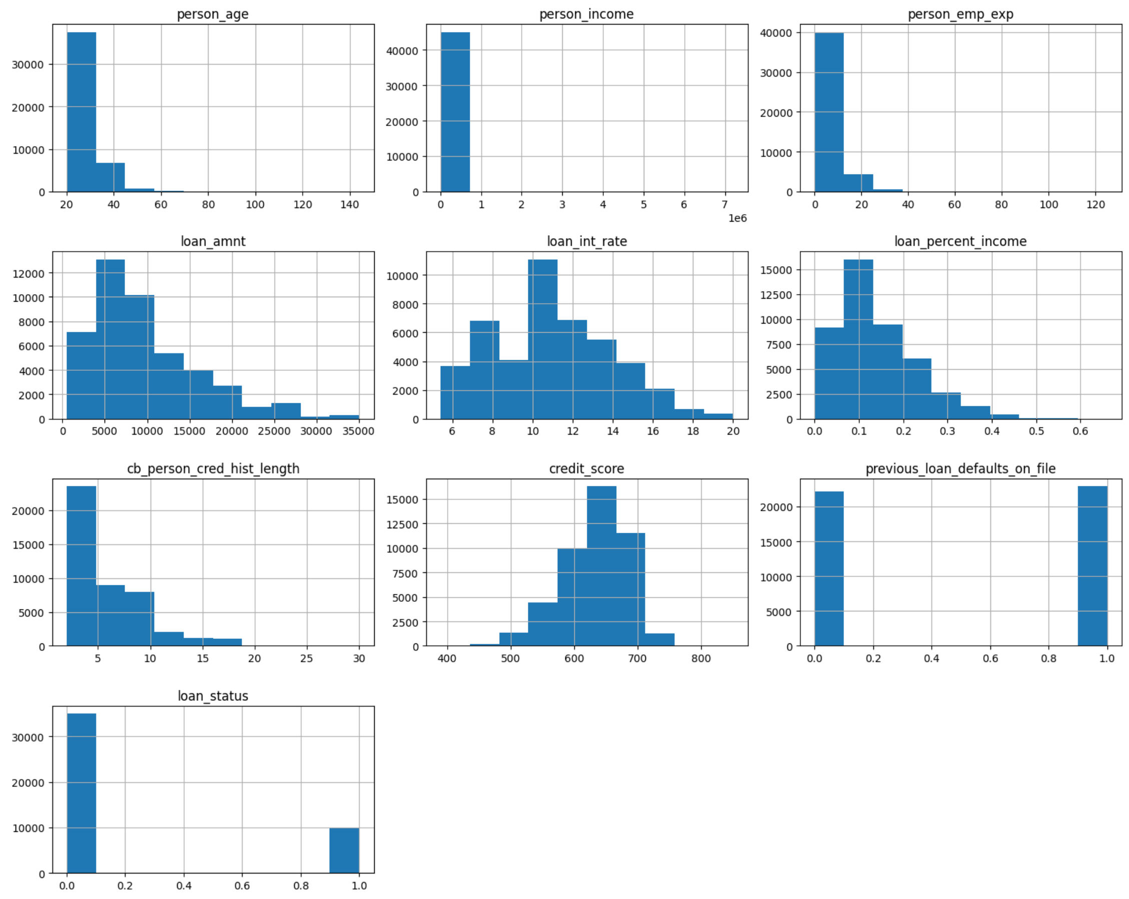
```
num_cols = df.select_dtypes(include=['int64','float64']).columns
cat_cols = df.select_dtypes(include=['object']).columns

num_cols, cat_cols
```

```
[ ]: (Index(['person_age', 'person_income', 'person_emp_exp', 'loan_amnt',
             'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length',
             'credit_score', 'previous_loan_defaults_on_file', 'loan_status'],
            dtype='object'),
      Index(['person_gender', 'person_education', 'person_home_ownership',
             'loan_intent'],
            dtype='object'))
```

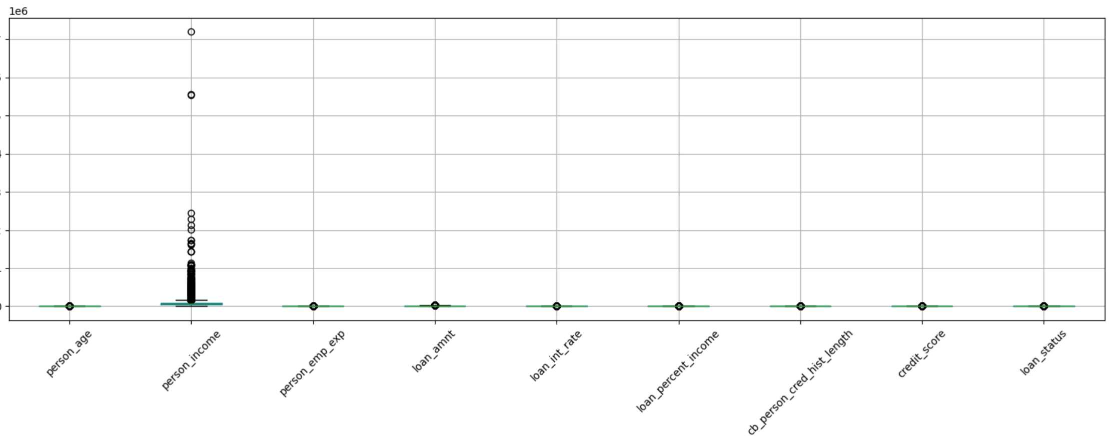
```
[ ]: df[num_cols].describe()
     #df[num_cols].median()
```

```
[ ]:          person_age  person_income  person_emp_exp     loan_amnt  \
     count  45000.000000   4.500000e+04    45000.000000  45000.000000
     mean      27.764178   8.031905e+04        5.410333   9583.157556
     std        6.045108   8.042250e+04        6.063532   6314.886691
     min       20.000000   8.000000e+03        0.000000    500.000000
     25%       24.000000   4.720400e+04        1.000000   5000.000000
     50%       26.000000   6.704800e+04        4.000000   8000.000000
     75%       30.000000   9.578925e+04        8.000000  12237.250000
     max      144.000000   7.200766e+06      125.000000  35000.000000

            loan_int_rate  loan_percent_income  cb_person_cred_hist_length  \
     count   45000.000000         45000.000000                45000.000000
     mean       11.006606             0.139725                    5.867489
     std         2.978808             0.087212                    3.879702
     min         5.420000             0.000000                    2.000000
     25%         8.590000             0.070000                    3.000000
     50%        11.010000             0.120000                    4.000000
     75%        12.990000             0.190000                    8.000000
     max        20.000000             0.660000                   30.000000

            credit_score   loan_status
     count  45000.000000  45000.000000
     mean     632.608756      0.222222
     std       50.435865      0.415744
     min      390.000000      0.000000
     25%      601.000000      0.000000
     50%      640.000000      0.000000
     75%      670.000000      0.000000
     max      850.000000      1.000000
```

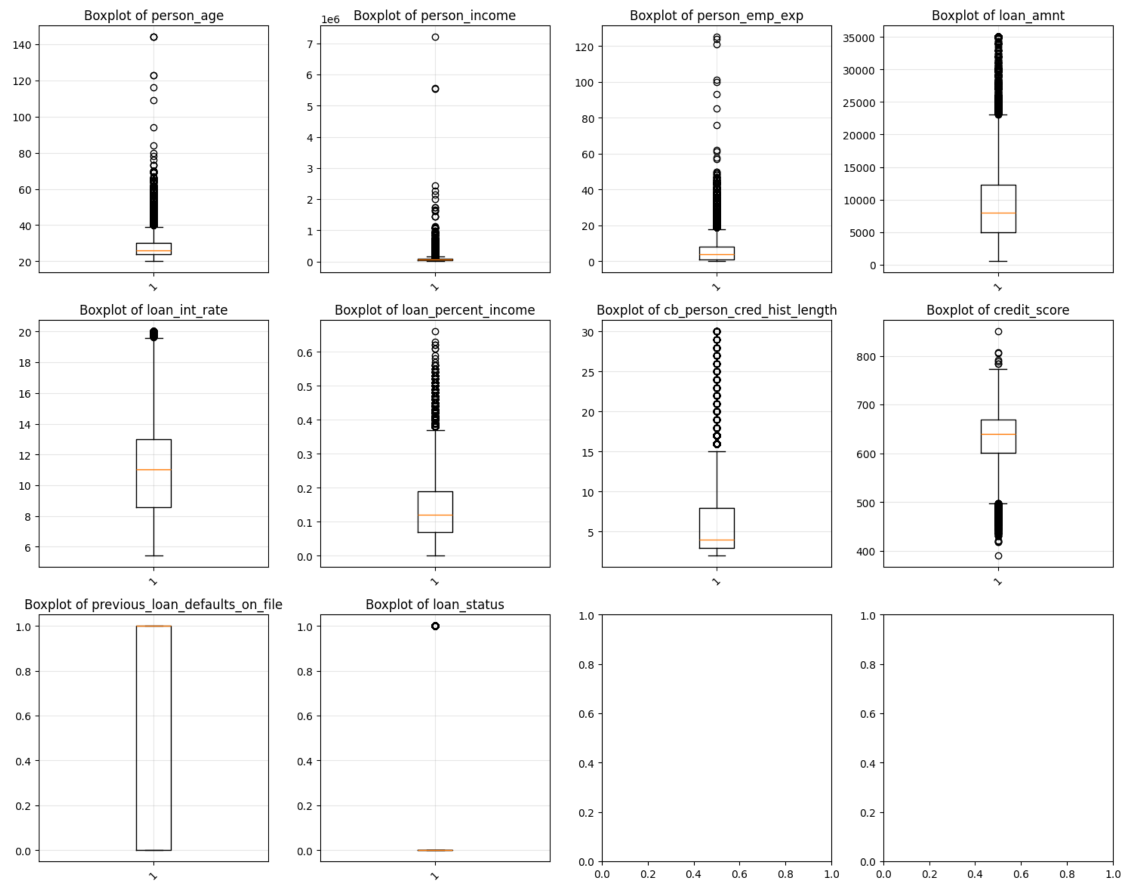
```
[ ]: df[num_cols].hist(figsize=(15,12))
     plt.tight_layout()
     plt.show()
```



```
[ ]: #box plot
     plt.figure(figsize=(15, 6))
     df[num_cols].boxplot(rot=45)
     plt.tight_layout()
     plt.show()
```



```
[ ]: fig, axes = plt.subplots(3, 4, figsize=(15, 12))
     axes = axes.flatten()

     for i, col in enumerate(num_cols):
         axes[i].boxplot(df[col])
         axes[i].set_title(f"Boxplot of {col}")
         axes[i].tick_params(axis='x', rotation=45)
         axes[i].grid(True, alpha=0.3)

     plt.tight_layout()
     plt.show()
```



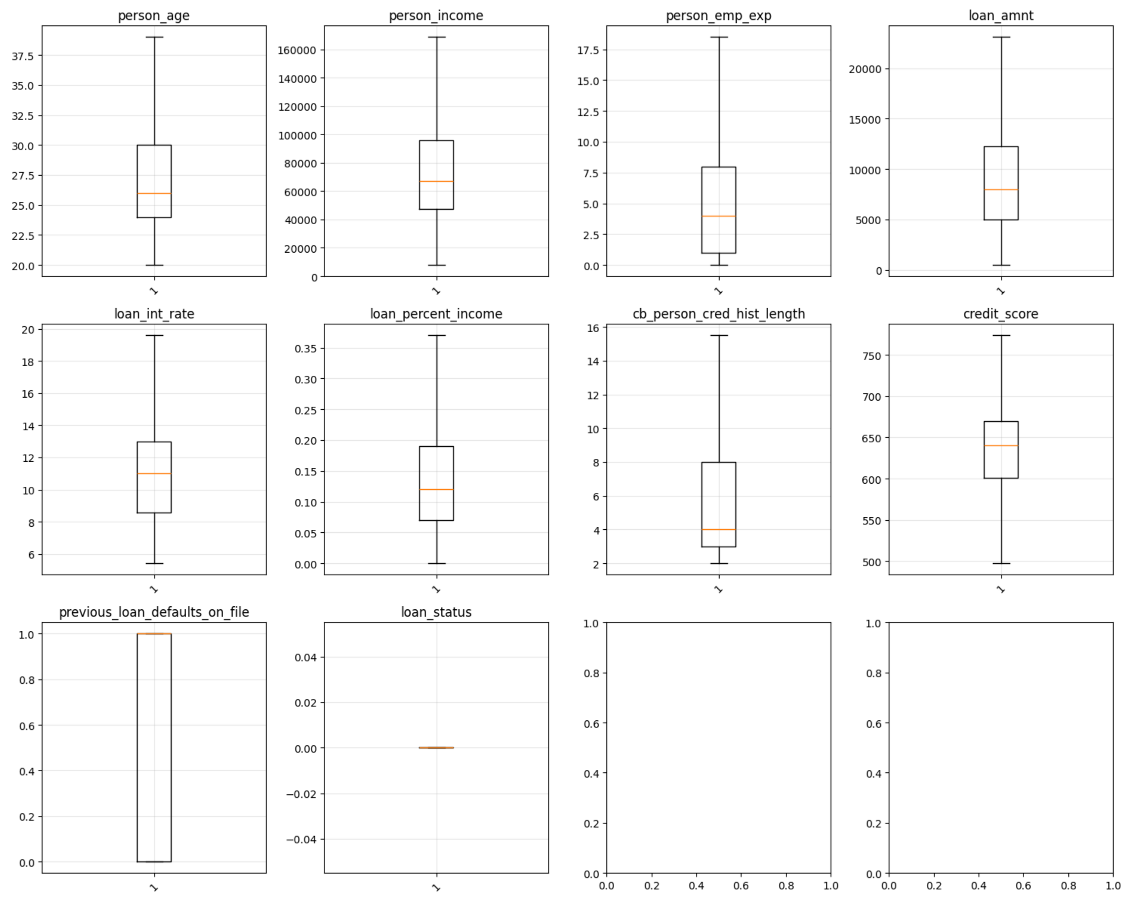
```
[ ]: outlier_summary = {}

     for col in num_cols:
         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

         lower_outliers = (df[col] < lower_bound).sum()
         upper_outliers = (df[col] > upper_bound).sum()

         outlier_summary[col] = {
             "Lower Outliers": lower_outliers,
             "Upper Outliers": upper_outliers,
             "Total Outliers": lower_outliers + upper_outliers
         }
```

```
outlier_df = pd.DataFrame(outlier_summary).T
outlier_df
```

```
[ ]:                                Lower Outliers  Upper Outliers  Total Outliers
     person_age                                  0            2188            2188
     person_income                               0            2218            2218
     person_emp_exp                              0            1724            1724
     loan_amnt                                   0            2348            2348
     loan_int_rate                               0             124             124
     loan_percent_income                         0             744             744
     cb_person_cred_hist_length                  0            1366            1366
     credit_score                              460               7             467
     previous_loan_defaults_on_file              0               0               0
     loan_status                                 0           10000           10000
```

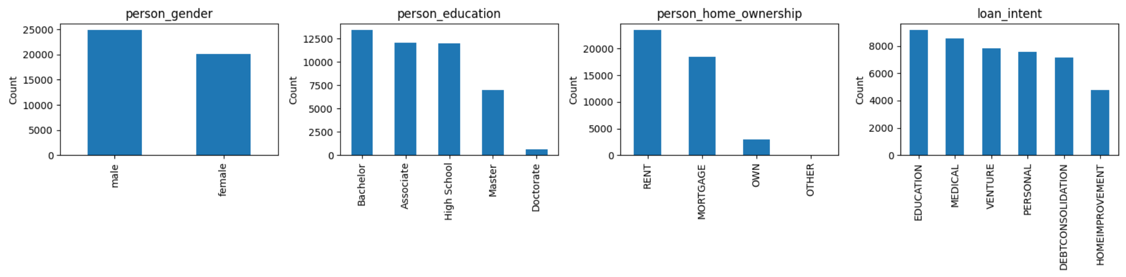
```
[ ]: #winorization
     df_capped = df.copy()

     for col in num_cols:
         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

         df_capped[col] = df_capped[col].clip(lower_bound, upper_bound)
```

```
[ ]: fig, axes = plt.subplots(3, 4, figsize=(15, 12))
     axes = axes.flatten()

     for i, col in enumerate(num_cols):
         axes[i].boxplot(df_capped[col])
         axes[i].set_title(col)
         axes[i].tick_params(axis='x', rotation=45)
         axes[i].grid(True, alpha=0.3)

     plt.tight_layout()
     plt.show()
```

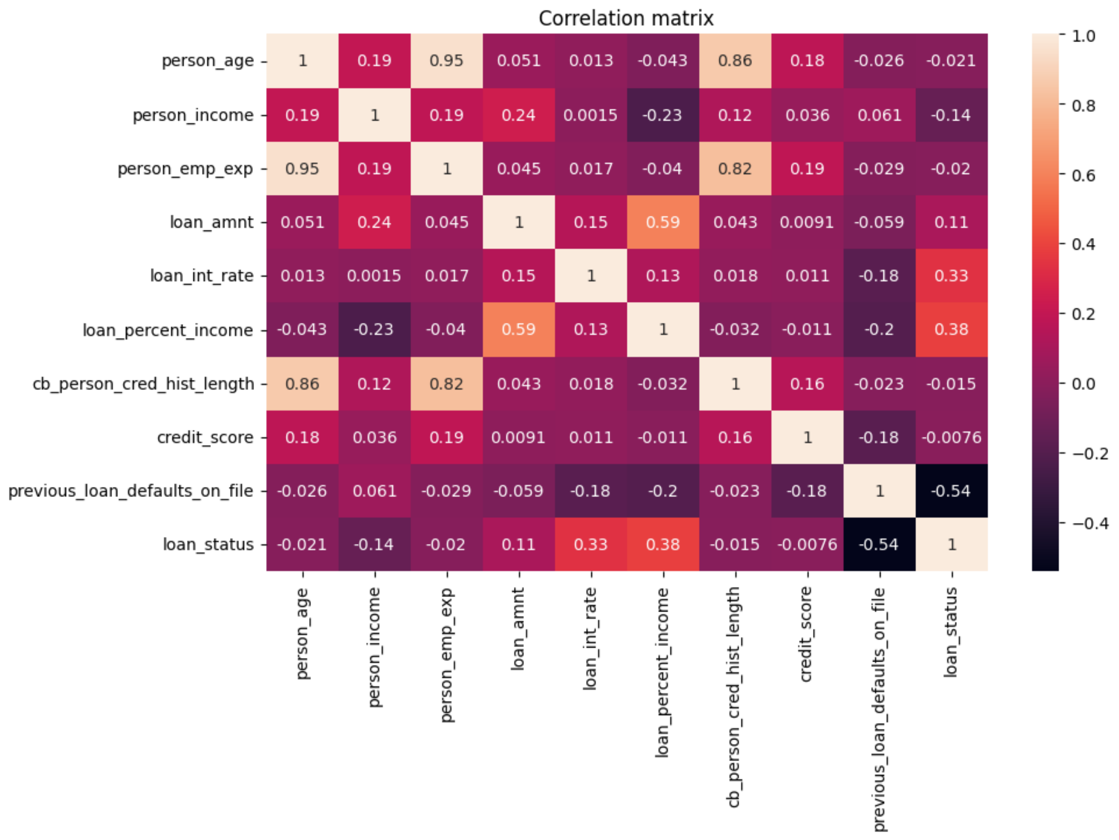


```
[ ]: ncols = 5
     nrows = (len(cat_cols) + ncols - 1) // ncols

     fig, axes = plt.subplots(
         nrows=nrows,
         ncols=ncols,
         figsize=(20, 4 * nrows)
     )

     # Make axes always a flat array
     if isinstance(axes, np.ndarray):
         axes = axes.flatten()
     else:
         axes = [axes]

     # Plot each categorical column
     for i, col in enumerate(cat_cols):
         df[col].value_counts().plot(kind='bar', ax=axes[i])
         axes[i].set_title(col)
```

```
    axes[i].set_xlabel('')
    axes[i].set_ylabel('Count')

# Remove any unused subplots
for j in range(len(cat_cols), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df[num_cols].corr(), annot=True)
plt.title("Correlation matrix")
plt.show()
```



	person_age	person_income	person_emp_exp	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_hist_length	credit_score	previous_loan_defaults_on_file	loan_status
person_age	1	0.19	0.95	0.051	0.013	-0.043	0.86	0.18	-0.026	-0.021
person_income	0.19	1	0.19	0.24	0.0015	-0.23	0.12	0.036	0.061	-0.14
person_emp_exp	0.95	0.19	1	0.045	0.017	-0.04	0.82	0.19	-0.029	-0.02
loan_amnt	0.051	0.24	0.045	1	0.15	0.59	0.043	0.0091	-0.059	0.11
loan_int_rate	0.013	0.0015	0.017	0.15	1	0.13	0.018	0.011	-0.18	0.33
loan_percent_income	-0.043	-0.23	-0.04	0.59	0.13	1	-0.032	-0.011	-0.2	0.38
cb_person_cred_hist_length	0.86	0.12	0.82	0.043	0.018	-0.032	1	0.16	-0.023	-0.015
credit_score	0.18	0.036	0.19	0.0091	0.011	-0.011	0.16	1	-0.18	-0.0076
previous_loan_defaults_on_file	-0.026	0.061	-0.029	-0.059	-0.18	-0.2	-0.023	-0.18	1	-0.54
loan_status	-0.021	-0.14	-0.02	0.11	0.33	0.38	-0.015	-0.0076	-0.54	1

[ ]:
```
y = df['loan_status']
X = df.drop(columns=['loan_status'])

num_cols = [col for col in num_cols if col != 'loan_status']
```

[ ]:
```
# ANOVA

selector = SelectKBest(score_func=f_classif, k=5)
X_anova_selected = selector.fit_transform(X[num_cols], y)

anova_scores = pd.DataFrame({
    'Feature': num_cols,
    'ANOVA F-Score': selector.scores_
}).sort_values(by='ANOVA F-Score', ascending=False)

anova_scores
```

[ ]:
```
                          Feature  ANOVA F-Score
8  previous_loan_defaults_on_file   18824.727466
5             loan_percent_income    7824.794030
```

```
4                    loan_int_rate    5574.454260
1                    person_income     845.525887
3                        loan_amnt     528.213632
0                       person_age      20.763596
2                   person_emp_exp      18.883771
6       cb_person_cred_hist_length       9.926174
7                     credit_score       2.631606
```

[ ]:
```
selected_features = [
    'previous_loan_defaults_on_file',
    'loan_percent_income',
    'loan_int_rate',
    'person_income',
    'loan_amnt'
]
```

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(
    X,
    y,
    test_size=0.30,          # 30% → temp
    random_state=42,
    stratify=y               # IMPORTANT for imbalanced classes
)


X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=0.50,          # split 30% into 15% + 15%
    random_state=42,
    stratify=y_temp
)

print("Training set:", X_train.shape)
print("Validation set:", X_val.shape)
print("Test set:", X_test.shape)
```

```
Training set: (31500, 13)
Validation set: (6750, 13)
Test set: (6750, 13)
```

[ ]:
```
#to verify class imbalance
def class_distribution(y, name):
    print(f"{name} class distribution:")
    print(y.value_counts(normalize=True))
    print()
```

```
class_distribution(y_train, "Train")
class_distribution(y_val, "Validation")
class_distribution(y_test, "Test")
```

```
Train class distribution:
loan_status
0    0.777778
1    0.222222
Name: proportion, dtype: float64

Validation class distribution:
loan_status
0    0.777778
1    0.222222
Name: proportion, dtype: float64

Test class distribution:
loan_status
0    0.777778
1    0.222222
Name: proportion, dtype: float64
```

Similar score, so there is no class imbalance

3 Predicting Diabetes

[ ]:
```
# DiabetesData
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder

from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif
from sklearn.model_selection import train_test_split
```

[ ]:
```
df=pd.read_csv('diabetes.csv')
df
```

[ ]:
```
        gender   age  hypertension  heart_disease smoking_history    bmi  \
0       Female  80.0             0              1           never  25.19
```

```
1      Female  54.0             0              0        No Info  27.32
2        Male  28.0             0              0          never  27.32
3      Female  36.0             0              0        current  23.45
4        Male  76.0             1              1        current  20.14
...       ...   ...           ...            ...            ...    ...
99995  Female  80.0             0              0        No Info  27.32
99996  Female   2.0             0              0        No Info  17.37
99997    Male  66.0             0              0         former  27.83
99998  Female  24.0             0              0          never  35.42
99999  Female  57.0             0              0        current  22.43

       HbA1c_level  blood_glucose_level  diabetes
0              6.6                  140         0
1              6.6                   80         0
2              5.7                  158         0
3              5.0                  155         0
4              4.8                  155         0
...            ...                  ...       ...
99995          6.2                   90         0
99996          6.5                  100         0
99997          5.7                  155         0
99998          4.0                  100         0
99999          6.6                   90         0

[100000 rows x 9 columns]
```

```
[ ]: print(df.columns)
```

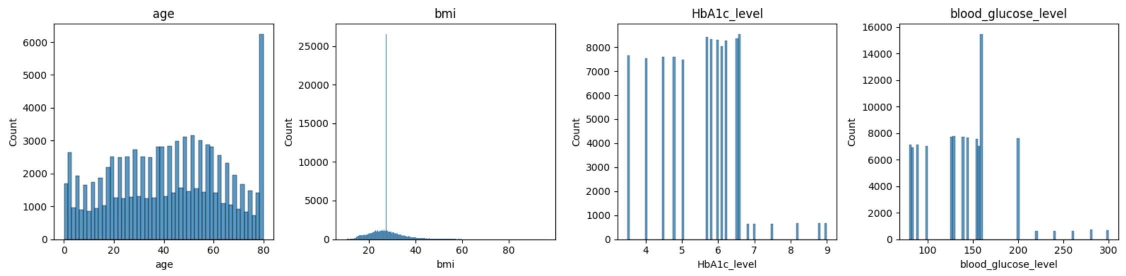
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'smoking_history',
       'bmi', 'HbA1c_level', 'blood_glucose_level', 'diabetes'],
      dtype='object')
```

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

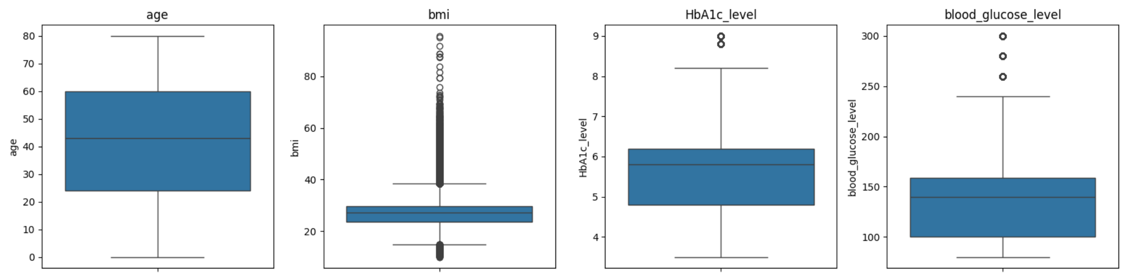
```
                 age  hypertension  heart_disease            bmi  \
count  100000.000000  100000.00000  100000.000000  100000.000000
mean       41.885856       0.07485       0.039420      27.320767
std        22.516840       0.26315       0.194593       6.636783
min         0.080000       0.00000       0.000000      10.010000
25%        24.000000       0.00000       0.000000      23.630000
50%        43.000000       0.00000       0.000000      27.320000
75%        60.000000       0.00000       0.000000      29.580000
max        80.000000       1.00000       1.000000      95.690000

         HbA1c_level  blood_glucose_level       diabetes
count  100000.000000        100000.000000  100000.000000
mean        5.527507           138.058060       0.085000
std         1.070672            40.708136       0.278883
```

```
min          3.500000          80.000000       0.000000
25%          4.800000         100.000000       0.000000
50%          5.800000         140.000000       0.000000
75%          6.200000         159.000000       0.000000
max          9.000000         300.000000       1.000000
```

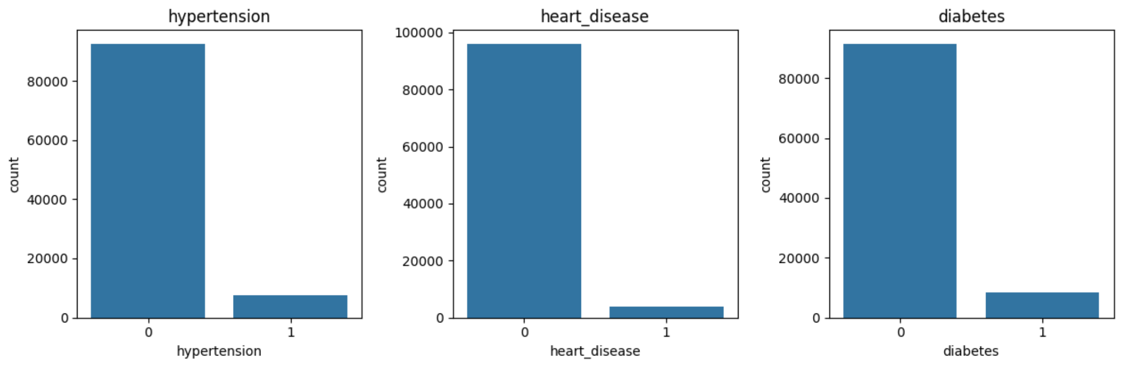
[ ]:
```
import math
num_cols = ['age', 'bmi', 'HbA1c_level', 'blood_glucose_level']
ncols=5
nrows=math.ceil(len(num_cols)/ncols)
fig,axes=plt.subplots(nrows=nrows,ncols=ncols,figsize=(20,4*nrows))
axes=axes.flatten()
for i,col in enumerate(num_cols):
    sns.histplot(df[col],ax=axes[i])
    axes[i].set_title(col)
for j in range(i+1,len(axes)):
    fig.delaxes(axes[j])
plt.tight_layout()
plt.show()
```



[ ]:
```
ncols=5
nrows=math.ceil(len(num_cols)/ncols)
fig,axes=plt.subplots(nrows=nrows,ncols=ncols,figsize=(20,4*nrows))
axes=axes.flatten()
for i,col in enumerate(num_cols):
    sns.boxplot(df[col],ax=axes[i])
    axes[i].set_title(col)
for j in range(i+1,len(axes)):
    fig.delaxes(axes[j])
plt.tight_layout()
plt.show()
```

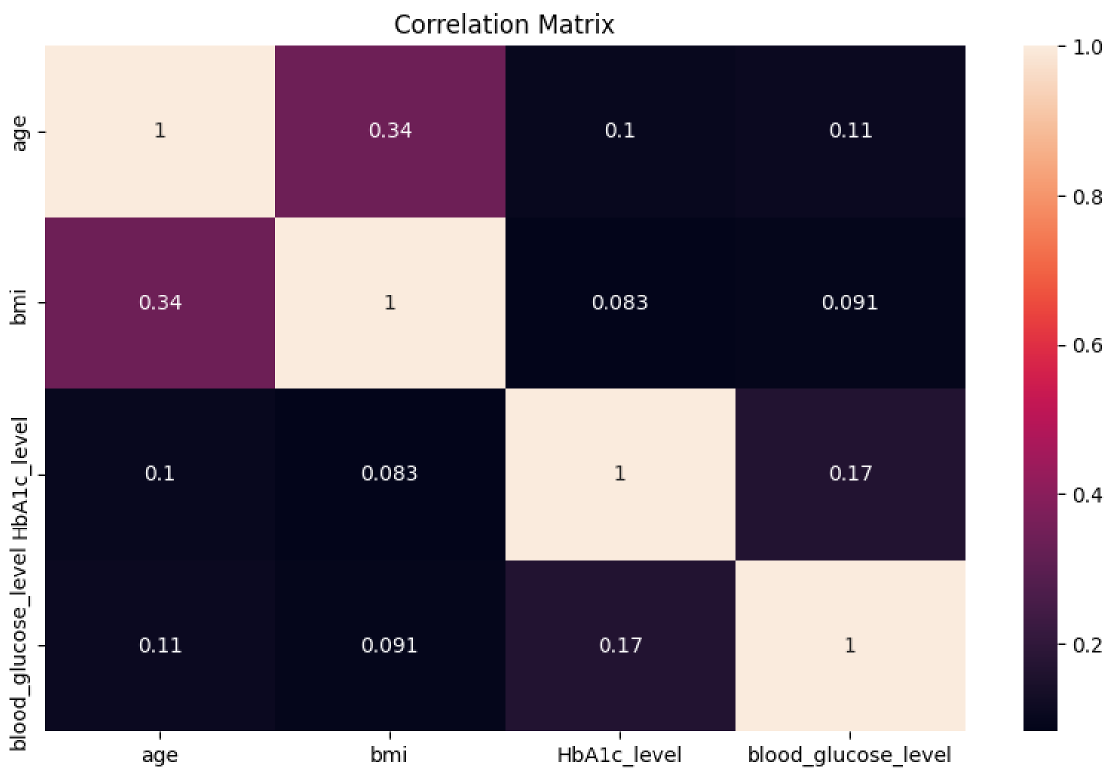


```
[ ]: cat_cols = ['hypertension', 'heart_disease', 'diabetes']

     fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20,4))
     axes = axes.flatten()
     for i, col in enumerate(cat_cols):
         sns.countplot(x=df[col], ax=axes[i])
         axes[i].set_title(col)


     for j in range(i+1, len(axes)):
         fig.delaxes(axes[j])

     plt.tight_layout()
     plt.show()
```



Heavy imbalanced

```
[ ]: plt.figure(figsize=(10, 6))
     sns.heatmap(df[num_cols].corr(), annot=True)
     plt.title("Correlation Matrix")
     plt.show()
```



4 Classification of Email Spam

[19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import math
from sklearn.feature_extraction.text import TfidfVectorizer
```

[20]:
```
df = pd.read_csv('spam_ham_dataset.csv')
df
```

[20]:
```
      Unnamed: 0 label                                               text  \
0            605   ham  Subject: enron methanol ; meter # : 988291\r\n...
1           2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see...
2           3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar...
3           4685  spam  Subject: photoshop , windows , office . cheap ...
4           2030   ham  Subject: re : indian springs\r\nthis deal is t...
...          ...   ...                                                ...
5166        1518   ham  Subject: put the 10 on the ft\r\nthe transport...
5167         404   ham  Subject: 3 / 4 / 2000 and following noms\r\nhp...
```

```
5168        2933   ham  Subject: calpine daily gas nomination\r\n>\r\n...
5169        1409   ham  Subject: industrial worksheets for august 2000...
5170        4807  spam  Subject: important online banking alert\r\ndea...

      label_num
0             0
1             0
2             0
3             1
4             0
...         ...
5166          0
5167          0
5168          0
5169          0
5170          1

[5171 rows x 4 columns]
```

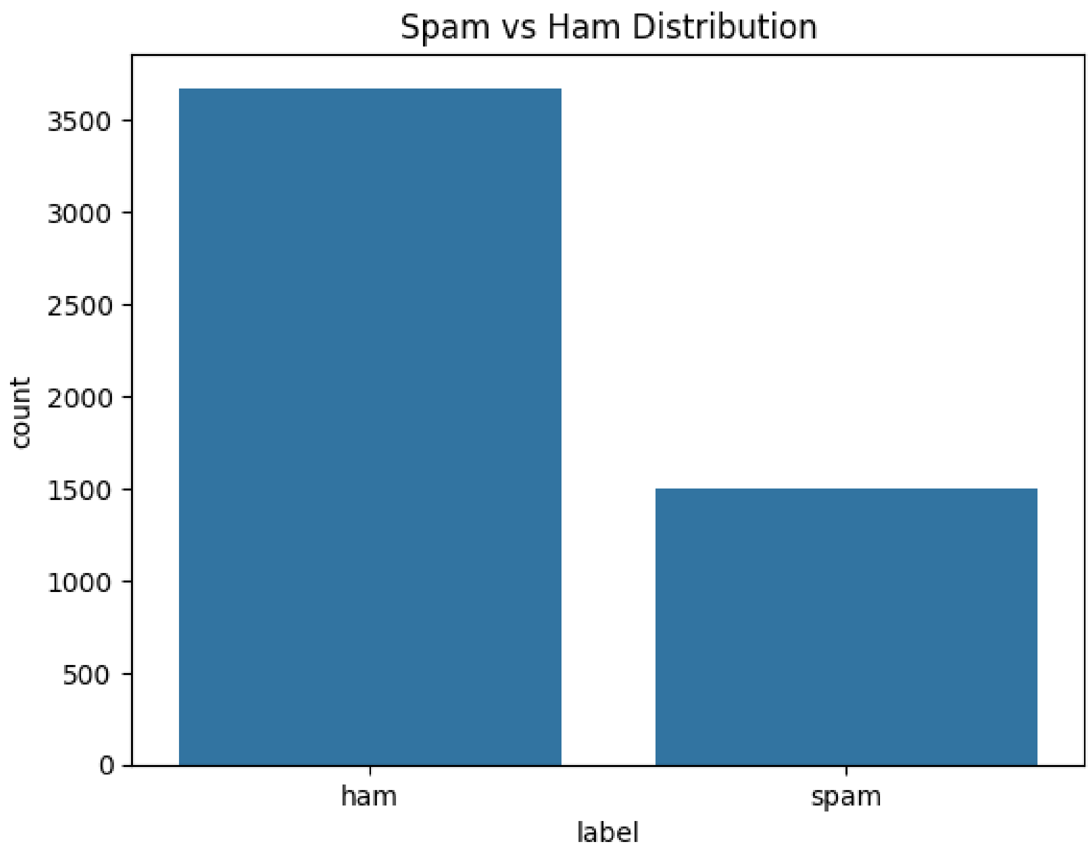
[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5171 entries, 0 to 5170
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  5171 non-null   int64
 1   label       5171 non-null   object
 2   text        5171 non-null   object
 3   label_num   5171 non-null   int64
dtypes: int64(2), object(2)
memory usage: 161.7+ KB
```

[22]:
```
df.describe()
```

[22]:
```
        Unnamed: 0    label_num
count  5171.000000  5171.000000
mean   2585.000000     0.289886
std    1492.883452     0.453753
min       0.000000     0.000000
25%    1292.500000     0.000000
50%    2585.000000     0.000000
75%    3877.500000     1.000000
max    5170.000000     1.000000
```

[23]:
```
sns.countplot(x='label', data=df)
plt.title('Spam vs Ham Distribution')
plt.show()
```



[24]:
```
import re
def clean_text(text):
  text = text.lower()
  text = re.sub(r"http\S+", "", text)
  text = re.sub(r"[^a-z\s]", "", text)
  return text

df['clean_text'] = df['text'].apply(clean_text)
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df['char_count'] = df['clean_text'].apply(len)

df.head()
```

[24]:
```
   Unnamed: 0 label                                               text  \
0         605   ham  Subject: enron methanol ; meter # : 988291\r\n...
1        2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see...
2        3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar...
3        4685  spam  Subject: photoshop , windows , office . cheap ...
4        2030   ham  Subject: re : indian springs\r\nthis deal is t...
```

```
   label_num                                         clean_text  word_count  \
0          0  subject enron methanol  meter   \r\nthis is a ...          49
1          0  subject hpl nom for january   \r\n see attache...          12
2          0  subject neon retreat\r\nho ho ho  we  re aroun...         460
3          1  subject photoshop  windows  office  cheap  mai...          44
4          0  subject re  indian springs\r\nthis deal is to ...          64

   char_count
0         302
1          80
2        2428
3         409
4         329
```

[25]:
```
from collections import Counter

all_words = " ".join(df['clean_text']).split()
common_words = Counter(all_words).most_common(20)
common_words
```

[25]:
```
[('the', 25613),
 ('to', 20332),
 ('ect', 13900),
 ('and', 12815),
 ('for', 10505),
 ('of', 10167),
 ('a', 9813),
 ('you', 8159),
 ('subject', 8060),
 ('in', 7699),
 ('on', 7308),
 ('hou', 7289),
 ('is', 7162),
 ('this', 7161),
 ('enron', 6555),
 ('i', 6379),
 ('be', 5060),
 ('that', 4767),
 ('we', 4339),
 ('from', 4191)]
```

[26]:
```
spam_words = " ".join(df[df['label']=="spam"]['clean_text']).split()
ham_words = " ".join(df[df['label']=="ham"]['clean_text']).split()
print("Spam words most common \n\n", Counter(spam_words).most_common(15))
print("\nNot Spam common words \n", Counter(ham_words).most_common(15))
```

```
Spam words most common
```

```
 [('the', 7254), ('to', 5160), ('and', 4903), ('of', 4490), ('a', 3787), ('in',
3129), ('you', 2794), ('for', 2523), ('this', 2283), ('is', 2256), ('your',
1946), ('subject', 1657), ('with', 1470), ('that', 1348), ('s', 1316)]

Not Spam common words
 [('the', 18359), ('to', 15172), ('ect', 13897), ('for', 7982), ('and', 7912),
('hou', 7281), ('enron', 6555), ('subject', 6403), ('on', 6049), ('a', 6026),
('of', 5677), ('you', 5365), ('i', 5241), ('is', 4906), ('this', 4878)]
```

[27]:
```
df.columns
```

[27]:
```
Index(['Unnamed: 0', 'label', 'text', 'label_num', 'clean_text', 'word_count',
       'char_count'],
      dtype='object')
```

[28]:
```
df.drop(columns=['Unnamed: 0'], inplace=True)
```

[29]:
```
vectorizer = TfidfVectorizer(
    stop_words='english',
    max_features=5000
)

X_text = vectorizer.fit_transform(df['clean_text'])
y = df['label_num']
```

[32]:
```
from sklearn.feature_selection import SelectKBest, chi2

chi2_scores, p_values = chi2(X_text, y)
```

[33]:
```
chi2_df = pd.DataFrame({
    'word': vectorizer.get_feature_names_out(),
    'chi2_score': chi2_scores,
    'p_value': p_values
})

chi2_df = chi2_df.sort_values(by='chi2_score', ascending=False)

chi2_df.head(15)
```

[33]:
```
         word  chi2_score       p_value
1447      ect  120.299194  5.440422e-28
2203     http  103.009438  3.335553e-24
1539    enron   90.682972  1.686383e-21
2187      hpl   77.925914  1.069787e-18
4967      xls   70.555826  4.474242e-17
2179      hou   65.285626  6.479222e-16
4956      www   60.633799  6.874349e-15
1163     deal   54.796253  1.336954e-13
```

```
2888     meter   52.872480  3.559216e-13
3174    online   51.076549  8.883323e-13
811      click   47.859892  4.577915e-12
2857      meds   44.986378  1.984098e-11
1911       gas   44.786746  2.197048e-11
320   attached   42.445020  7.269727e-11
4790    viagra   41.864447  9.782529e-11
```

5 Handwritten Character Recognition / MNIST

[19]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import math
from sklearn.feature_extraction.text import TfidfVectorizer
```

[20]:
```python
df = pd.read_csv('spam_ham_dataset.csv')
df
```

[20]:
```
      Unnamed: 0 label                                               text  \
0            605   ham  Subject: enron methanol ; meter # : 988291\r\n...
1           2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see...
2           3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar...
3           4685  spam  Subject: photoshop , windows , office . cheap ...
4           2030   ham  Subject: re : indian springs\r\nthis deal is t...
...          ...   ...                                                ...
5166        1518   ham  Subject: put the 10 on the ft\r\nthe transport...
5167         404   ham  Subject: 3 / 4 / 2000 and following noms\r\nhp...
5168        2933   ham  Subject: calpine daily gas nomination\r\n>\r\n...
5169        1409   ham  Subject: industrial worksheets for august 2000...
5170        4807  spam  Subject: important online banking alert\r\ndea...

      label_num
0             0
1             0
2             0
3             1
4             0
...         ...
5166          0
5167          0
5168          0
5169          0
5170          1
```

```
[5171 rows x 4 columns]
```

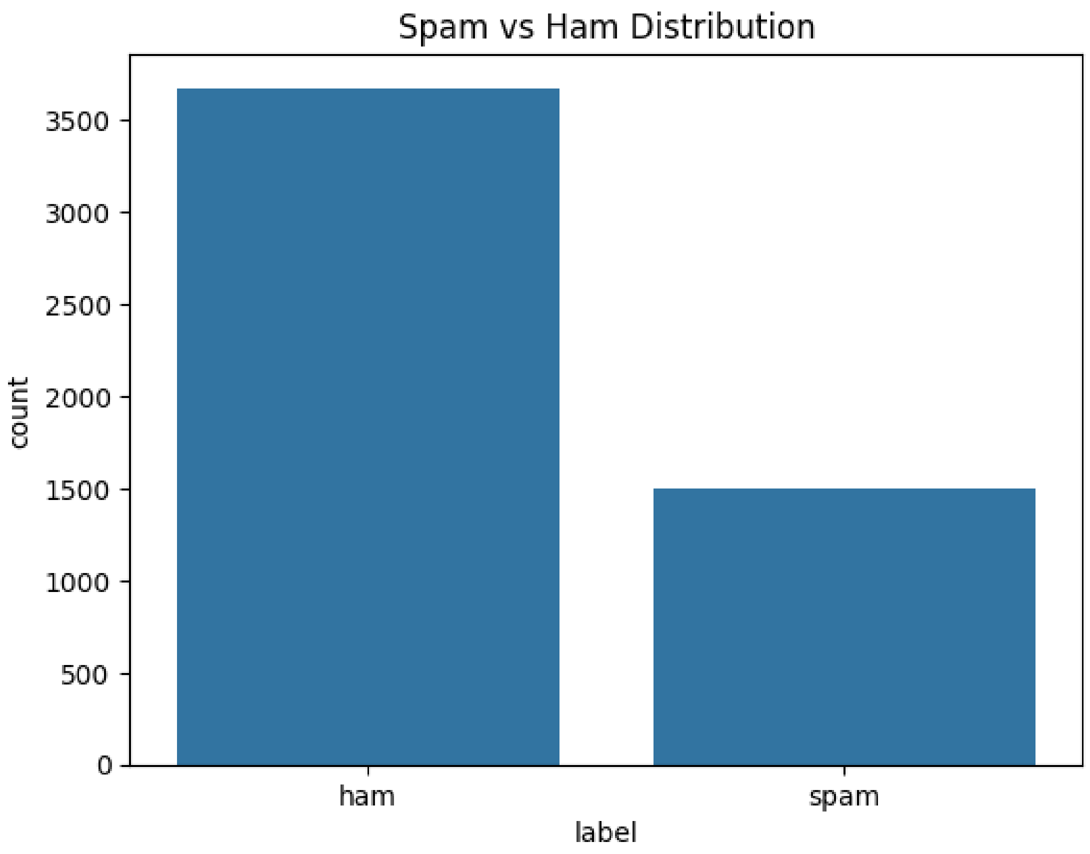
[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5171 entries, 0 to 5170
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  5171 non-null   int64
 1   label       5171 non-null   object
 2   text        5171 non-null   object
 3   label_num   5171 non-null   int64
dtypes: int64(2), object(2)
memory usage: 161.7+ KB
```

[22]:
```
df.describe()
```

[22]:
```
        Unnamed: 0    label_num
count  5171.000000  5171.000000
mean   2585.000000     0.289886
std    1492.883452     0.453753
min       0.000000     0.000000
25%    1292.500000     0.000000
50%    2585.000000     0.000000
75%    3877.500000     1.000000
max    5170.000000     1.000000
```

[23]:
```
sns.countplot(x='label', data=df)
plt.title('Spam vs Ham Distribution')
plt.show()
```



[24]:
```python
import re
def clean_text(text):
  text = text.lower()
  text = re.sub(r"http\S+", "", text)
  text = re.sub(r"[^a-z\s]", "", text)
  return text

df['clean_text'] = df['text'].apply(clean_text)
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
df['char_count'] = df['clean_text'].apply(len)

df.head()
```

[24]:
```
   Unnamed: 0 label                                               text  \
0         605   ham  Subject: enron methanol ; meter # : 988291\r\n...
1        2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see...
2        3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar...
3        4685  spam  Subject: photoshop , windows , office . cheap ...
4        2030   ham  Subject: re : indian springs\r\nthis deal is t...
```

```
   label_num                                         clean_text  word_count  \
0          0  subject enron methanol  meter   \r\nthis is a ...          49
1          0  subject hpl nom for january   \r\n see attache...          12
2          0  subject neon retreat\r\nho ho ho  we  re aroun...         460
3          1  subject photoshop  windows  office  cheap  mai...          44
4          0  subject re  indian springs\r\nthis deal is to ...          64

   char_count
0         302
1          80
2        2428
3         409
4         329
```

[25]:
```
from collections import Counter

all_words = " ".join(df['clean_text']).split()
common_words = Counter(all_words).most_common(20)
common_words
```

[25]:
```
[('the', 25613),
 ('to', 20332),
 ('ect', 13900),
 ('and', 12815),
 ('for', 10505),
 ('of', 10167),
 ('a', 9813),
 ('you', 8159),
 ('subject', 8060),
 ('in', 7699),
 ('on', 7308),
 ('hou', 7289),
 ('is', 7162),
 ('this', 7161),
 ('enron', 6555),
 ('i', 6379),
 ('be', 5060),
 ('that', 4767),
 ('we', 4339),
 ('from', 4191)]
```

[26]:
```
spam_words = " ".join(df[df['label']=="spam"]['clean_text']).split()
ham_words = " ".join(df[df['label']=="ham"]['clean_text']).split()
print("Spam words most common \n\n", Counter(spam_words).most_common(15))
print("\nNot Spam common words \n", Counter(ham_words).most_common(15))
```

```
Spam words most common
```

```
 [('the', 7254), ('to', 5160), ('and', 4903), ('of', 4490), ('a', 3787), ('in',
3129), ('you', 2794), ('for', 2523), ('this', 2283), ('is', 2256), ('your',
1946), ('subject', 1657), ('with', 1470), ('that', 1348), ('s', 1316)]

Not Spam common words
 [('the', 18359), ('to', 15172), ('ect', 13897), ('for', 7982), ('and', 7912),
('hou', 7281), ('enron', 6555), ('subject', 6403), ('on', 6049), ('a', 6026),
('of', 5677), ('you', 5365), ('i', 5241), ('is', 4906), ('this', 4878)]
```

[27]:
```
df.columns
```

[27]:
```
Index(['Unnamed: 0', 'label', 'text', 'label_num', 'clean_text', 'word_count',
       'char_count'],
      dtype='object')
```

[28]:
```
df.drop(columns=['Unnamed: 0'], inplace=True)
```

[29]:
```
vectorizer = TfidfVectorizer(
    stop_words='english',
    max_features=5000
)

X_text = vectorizer.fit_transform(df['clean_text'])
y = df['label_num']
```

[32]:
```
from sklearn.feature_selection import SelectKBest, chi2

chi2_scores, p_values = chi2(X_text, y)
```

[33]:
```
chi2_df = pd.DataFrame({
    'word': vectorizer.get_feature_names_out(),
    'chi2_score': chi2_scores,
    'p_value': p_values
})

chi2_df = chi2_df.sort_values(by='chi2_score', ascending=False)

chi2_df.head(15)
```

[33]:
```
          word  chi2_score       p_value
1447       ect  120.299194  5.440422e-28
2203      http  103.009438  3.335553e-24
1539     enron   90.682972  1.686383e-21
2187       hpl   77.925914  1.069787e-18
4967       xls   70.555826  4.474242e-17
2179       hou   65.285626  6.479222e-16
4956       www   60.633799  6.874349e-15
1163      deal   54.796253  1.336954e-13
```

```
2888     meter   52.872480  3.559216e-13
3174    online   51.076549  8.883323e-13
811      click   47.859892  4.577915e-12
2857      meds   44.986378  1.984098e-11
1911       gas   44.786746  2.197048e-11
320   attached   42.445020  7.269727e-11
4790    viagra   41.864447  9.782529e-11
```

Results and Discussions

Dataset	**Type of ML Task**	**Feature Selection Technique**	**Suitable ML Algorithm**
Iris Dataset	Classification	SelectKBest	Logistic Regression
Loan Amount Prediction	Regression	Correlation Coefficient	Linear Regression
Predicting Diabetes	Classification	SelectKBest	Ridge Regression
Classification of Email Spam	Classification	Correlation Coefficient	Naïve Bayes
Handwritten Character Recognition (MNIST)	Classification	PCA	CNN

Learning Practices

- Understand the fundamentals of data analysis using Python libraries such as NumPy, Pandas, Matplotlib, and Seaborn.
- Analyze datasets using descriptive statistics and exploratory data analysis techniques.
- Visualize data distributions and relationships using appropriate plots and charts..
- Identify suitable machine learning tasks such as classification and regression for different datasets.