Problem:

Describe a persona [classify] of wether the next patient that is administered basic laboratory tests would require to be admitted in to the hospital (incare patient) or can be treated within the out-patient department of the hospital itself.

Aims and Methods:

For this classification we aim to test a suite of supervisory classification algorithms, namely:

1. Logistic Regression
2. Support Vector Machine
3. K-Nearest Neighbour
4. Random Forest

We will use the **'SOURCE'** column that classifies each patient as **'IN'** or **'OUT'** as the target variable for the classification process.

Data Collection:

Dataset used is the Electronic Health Record (EHC) obtained from a private hospital in Indonesia. It contains the patients laboratory test results.

Exploratory Data Analysis:

In [2]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import shapiro, mstats, jarque_bera
sns.set()

# For pltoly graphs
import plotly.io as pio
pio.renderers.default = "notebook_connected"

from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

from plotly.subplots import make_subplots
import plotly.graph_objs as go
import plotly.figure_factory as ff

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix, roc_auc_score, accuracy_s

# To ignore unwanted warnings
import warnings
warnings.filterwarnings('ignore')

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Load the Data:

In [3]:
```
data  = pd.read_csv('data-ori.csv')
data.tail(5)
```

Out[3]:

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE	S
4407	32.8	10.4	3.49	8.1	72	29.8	31.7	94.0	92	
4408	33.7	10.8	3.67	6.7	70	29.4	32.0	91.8	92	
4409	33.2	11.2	3.47	7.2	235	32.3	33.7	95.7	93	
4410	31.5	10.4	3.15	9.1	187	33.0	33.0	100.0	98	
4411	33.5	10.9	3.44	5.8	275	31.7	32.5	97.4	99	

In [4]:
```
# Set Encoding labels for the target variable 'SORUCE'
data['SOURCE'] = data['SOURCE'].replace({'in': 1, 'out': 0})

# List all the features of the dataset
features = [col for col in data.columns if col != 'SOURCE']

#List the categorical and numerical features
num_feats = [numf for numf in features if data[numf].dtype != object]
cat_feats = [catf for catf in features if data[catf].dtype == object]
print(f'The feature table contains:\n1. {len(num_feats)} numerical feature[s] -', end=' ')
print(*num_feats, sep=', ')
print(f'2. {len(cat_feats)} categorical feature[s] -', end=' ')
print(*cat_feats)
```

```
The feature table contains:
1. 9 numerical feature[s] - HAEMATOCRIT, HAEMOGLOBINS, ERYTHROCYTE, LEUCOCYTE, THROMBOCYTE, MCH, MCHC, MC
V, AGE
2. 1 categorical feature[s] - SEX
```

Exploratory Data Analysis (EDA)

Features in dataset:

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4412 entries, 0 to 4411
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   HAEMATOCRIT   4412 non-null   float64
 1   HAEMOGLOBINS  4412 non-null   float64
 2   ERYTHROCYTE   4412 non-null   float64
 3   LEUCOCYTE     4412 non-null   float64
 4   THROMBOCYTE   4412 non-null   int64
 5   MCH           4412 non-null   float64
 6   MCHC          4412 non-null   float64
 7   MCV           4412 non-null   float64
 8   AGE           4412 non-null   int64
 9   SEX           4412 non-null   object
 10  SOURCE        4412 non-null   int64
dtypes: float64(7), int64(3), object(1)
memory usage: 379.3+ KB
```

- The dataset does not contain any missing/null values.
- For the feature list we can again see that the categorical varriable is 'SEX' as indicated by the 'object' data type.

In [6]:
```
# Unique values for each feature and target variables

data.nunique().to_frame('Unqiue Values')
```

Out[6]:

	Unqiue Values
HAEMATOCRIT	326
HAEMOGLOBINS	128
ERYTHROCYTE	433
LEUCOCYTE	276
THROMBOCYTE	554
MCH	189
MCHC	105
MCV	406
AGE	95
SEX	2
SOURCE	2

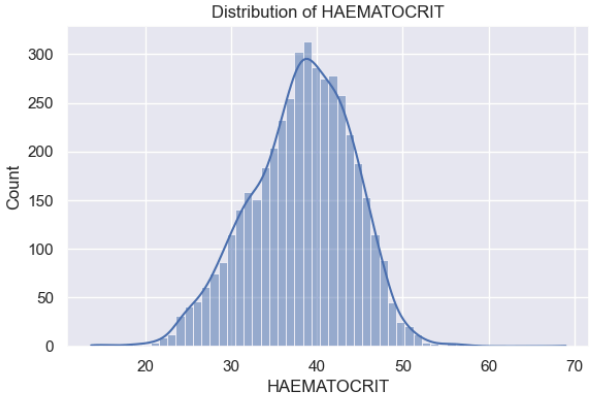
Descriptive Statistics:

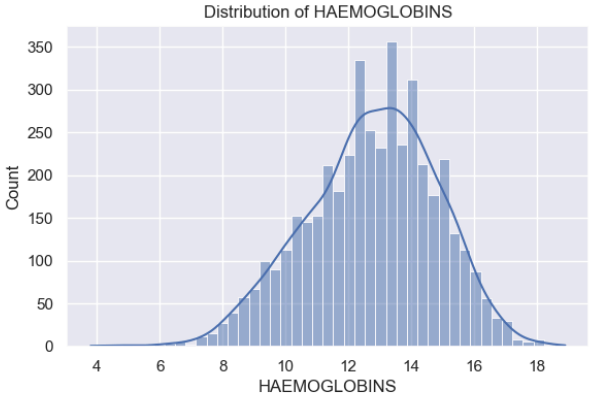
In [7]:
```
data.describe(include='all')
```

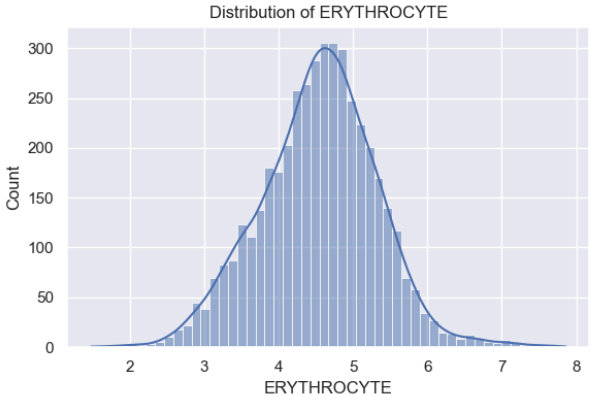
Out[7]:

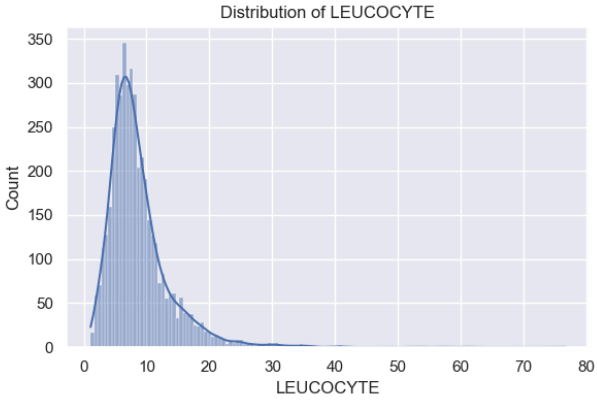
	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC
count	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	38.197688	12.741727	4.541260	8.718608	257.524479	28.234701	33.343042
std	5.974784	2.079903	0.784091	5.049041	113.972365	2.672639	1.228664
min	13.700000	3.800000	1.480000	1.100000	8.000000	14.900000	26.000000
25%	34.375000	11.400000	4.040000	5.675000	188.000000	27.200000	32.700000
50%	38.600000	12.900000	4.570000	7.600000	256.000000	28.700000	33.400000
75%	42.500000	14.200000	5.050000	10.300000	321.000000	29.800000	34.100000
max	69.000000	18.900000	7.860000	76.600000	1183.000000	40.800000	39.000000

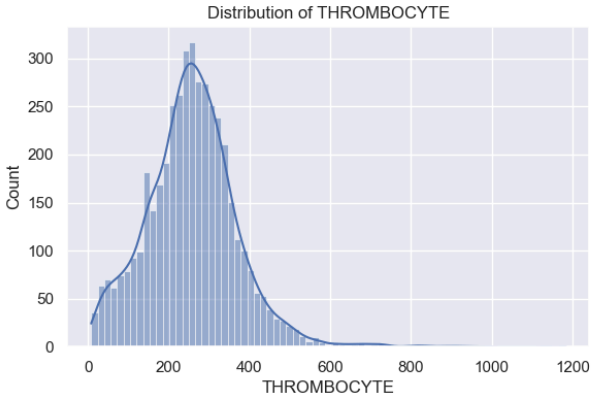
In [8]:
```
# Distribution of variables
numeric_columns = data.select_dtypes(include=[np.number]).columns
num_vars = len(numeric_columns)

# Calculate the number of rows needed in the subplot grid (2 columns grid here for more horizontal space
num_rows = (num_vars + 1) // 2

plt.figure(figsize=(12, num_rows * 4))  # Adjust the figure size based on the number of rows
for i, col in enumerate(numeric_columns):
    plt.subplot(num_rows, 2, i + 1)  # 2 columns
    sns.histplot(data[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

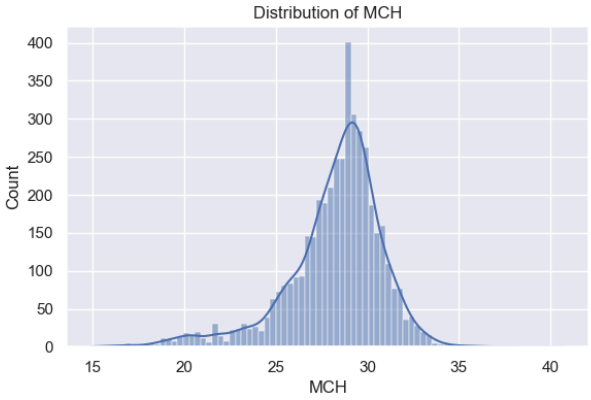

Distribution of HAEMATOCRIT
Count
HAEMATOCRIT

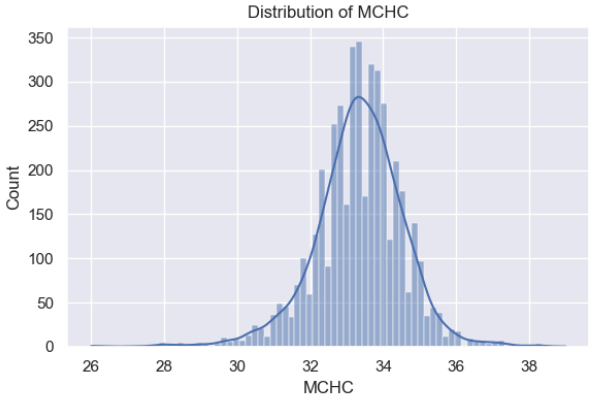

Distribution of HAEMOGLOBINS
Count
HAEMOGLOBINS

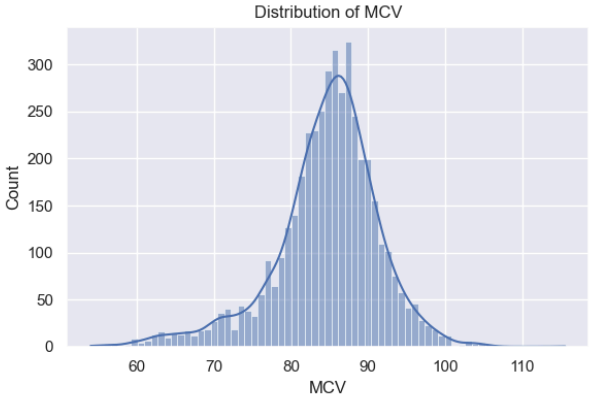

Distribution of ERYTHROCYTE
Count
ERYTHROCYTE

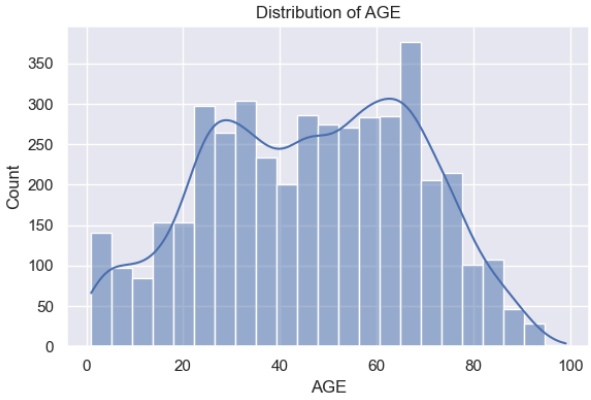

Distribution of LEUCOCYTE
Count
LEUCOCYTE

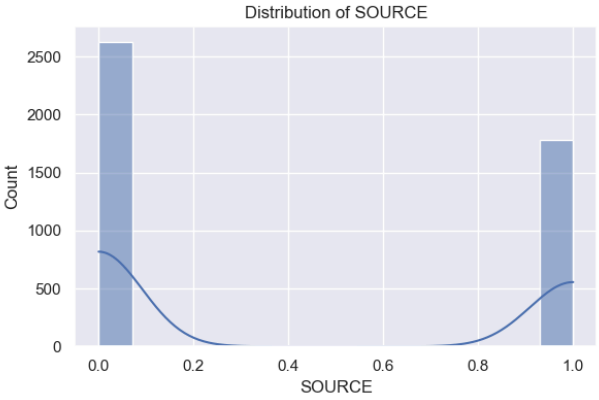

Distribution of THROMBOCYTE
Count
THROMBOCYTE


Distribution of MCH
Count
MCH


Distribution of MCHC
Count
MCHC


Distribution of MCV
Count
MCV

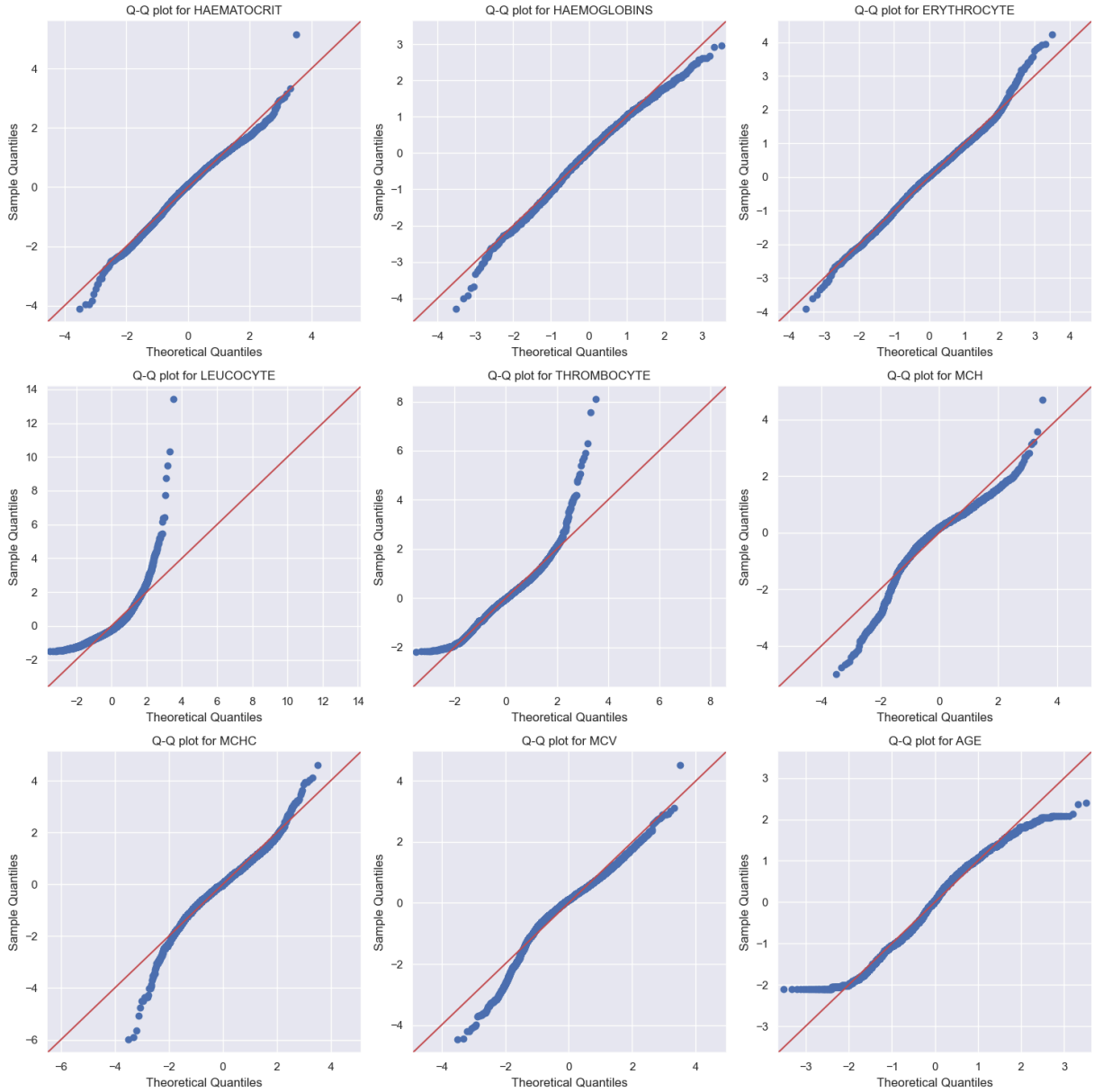

Distribution of AGE
Count
AGE


Distribution of SOURCE
Count
SOURCE

Normalization

Shapiro-Wilk Test/Jarque-Bera Test

In [9]:
```
# Assume df_robust is your DataFrame with robust scaling applied to the necessary columns
numeric_cols = ['HAEMATOCRIT', 'HAEMOGLOBINS', 'ERYTHROCYTE', 'LEUCOCYTE', 'THROMBOCYTE', 'MCH', 'MCHC',

# Initialize a dictionary to store test results
normality_results = {}

# Determine the grid size for subplots
n_cols = 3  # Number of columns in the grid
n_rows = (len(numeric_cols) + n_cols - 1) // n_cols  # Calculate rows needed, rounded up

# Create a figure with subplots
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(15, 5 * n_rows))
axes = axes.flatten()  # Flatten the axes array for easier iteration

# Loop through each column to plot
for i, column in enumerate(numeric_cols):
    # Perform Jarque-Bera test
    stat, p = jarque_bera(data[column])
    normality_results[column] = {'Statistic': stat, 'p-value': p}

    # Plot Q-Q plot in the respective subplot
    sm.qqplot(data[column], line='45', fit=True, ax=axes[i])
    axes[i].set_title(f'Q-Q plot for {column}')

# Adjust layout to prevent overlap
plt.tight_layout()

# Hide any unused subplots if the number of plots isn't a perfect fit for the grid
for ax in axes[len(numeric_cols):]:
    ax.set_visible(False)

# Show the plot grid
plt.show()
# Convert results to a DataFrame for better visualization
normality_test_df = pd.DataFrame(normality_results).T  # Transpose to have columns as headers
print(normality_test_df)
```



	Statistic	p-value
HAEMATOCRIT	70.846333	4.129659e-16
HAEMOGLOBINS	72.505472	1.801511e-16
ERYTHROCYTE	50.636894	1.010039e-11
LEUCOCYTE	63461.523424	0.000000e+00
THROMBOCYTE	3364.988832	0.000000e+00
MCH	2292.628795	0.000000e+00
MCHC	1366.448760	1.902926e-297
MCV	1079.133683	4.667649e-235
AGE	141.057522	2.342870e-31

As can be seen above, all variables are not normalized so we need normalization.

Solve long tail issues:

In [10]:
```
# Log Transformation (add 1 to shift all values away from zero)
data['LEUCOCYTE_log'] = np.log1p(data['LEUCOCYTE'])

# Winsorizing the data (limiting extreme values to the 95th percentile)
data['LEUCOCYTE_winsorized'] = mstats.winsorize(data['LEUCOCYTE'], limits=[0, 0.10])
```

In [11]:
```
# Log Transformation (add 1 to shift all values away from zero)
data['THROMBOCYTE_log'] = np.log1p(data['THROMBOCYTE'])

# Winsorizing the data (limiting extreme values to the 95th percentile)
data['THROMBOCYTE_winsorized'] = mstats.winsorize(data['THROMBOCYTE'], limits=[0, 0.05])
```

In [12]:
```
data
```

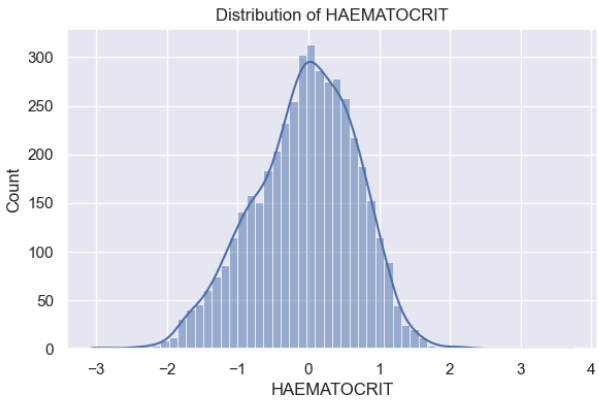
Out[12]:

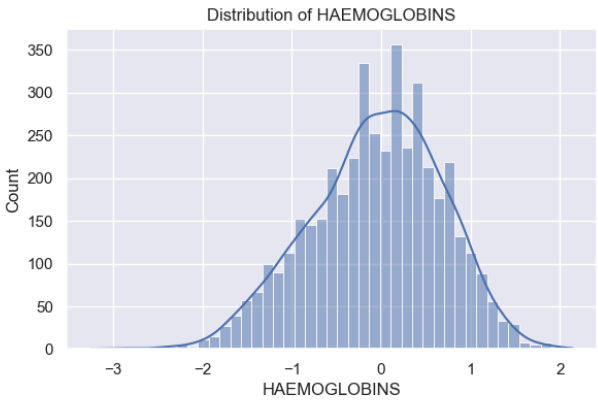
	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE	S
0	35.1	11.8	4.65	6.3	310	25.4	33.6	75.5	1	
1	43.5	14.8	5.39	12.7	334	27.5	34.0	80.7	1	
2	33.5	11.3	4.74	13.2	305	23.8	33.7	70.7	1	
3	39.1	13.7	4.98	10.5	366	27.5	35.0	78.5	1	
4	30.9	9.9	4.23	22.1	333	23.4	32.0	73.0	1	
...	...	...	...	...	...	...	...	...	...	
4407	32.8	10.4	3.49	8.1	72	29.8	31.7	94.0	92	
4408	33.7	10.8	3.67	6.7	70	29.4	32.0	91.8	92	
4409	33.2	11.2	3.47	7.2	235	32.3	33.7	95.7	93	
4410	31.5	10.4	3.15	9.1	187	33.0	33.0	100.0	98	
4411	33.5	10.9	3.44	5.8	275	31.7	32.5	97.4	99	

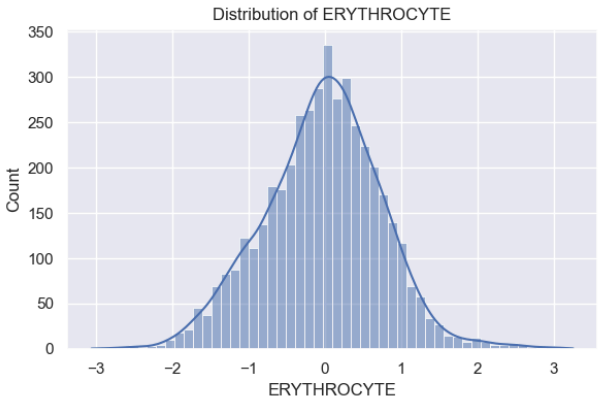
4412 rows × 15 columns

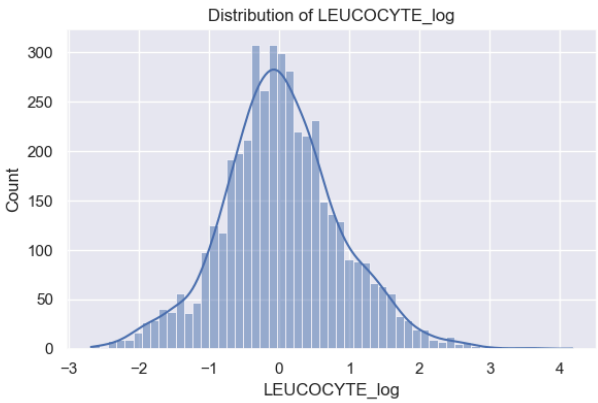
In [13]:
```
# Select numeric columns (excluding categorical columns like 'SEX' and 'SOURCE')
# numeric_cols = ['HAEMATOCRIT', 'HAEMOGLOBINS', 'ERYTHROCYTE', 'LEUCOCYTE_winsorized', 'THROMBOCYTE_wins
numeric_cols = ['HAEMATOCRIT', 'HAEMOGLOBINS', 'ERYTHROCYTE', 'LEUCOCYTE_log', 'THROMBOCYTE_log', 'MCH',

# # Min-Max Scaling
# min_max_scaler = MinMaxScaler()
# df_min_max = data.copy()
# df_min_max[numeric_cols] = min_max_scaler.fit_transform(data[numeric_cols])

# Z-score Normalization
standard_scaler = StandardScaler()
df_standard = data.copy()
df_standard[numeric_cols] = standard_scaler.fit_transform(data[numeric_cols])

# Robust Scaling
robust_scaler = RobustScaler()
df_robust = data.copy()
df_robust[numeric_cols] = robust_scaler.fit_transform(data[numeric_cols])
```

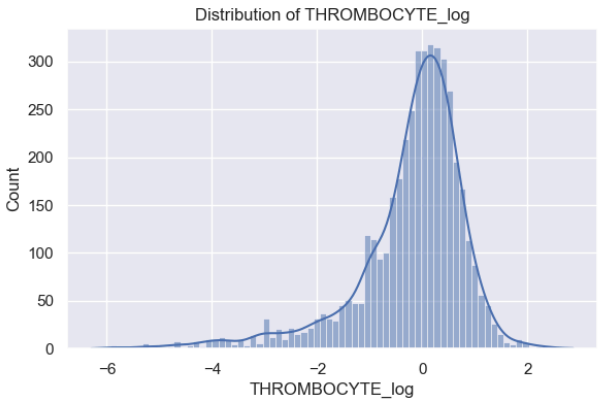
In [14]:
```
# Distribution of variables
num_vars = len(numeric_cols)

# Calculate the number of rows needed in the subplot grid (2 columns grid here for more horizontal space
num_rows = (num_vars + 1) // 2

plt.figure(figsize=(12, num_rows * 4))  # Adjust the figure size based on the number of rows
for i, col in enumerate(numeric_cols):
    plt.subplot(num_rows, 2, i + 1)  # 2 columns
    sns.histplot(df_robust[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

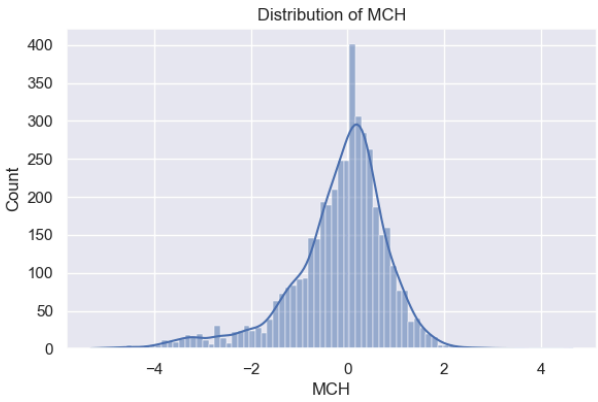

Distribution of HAEMATOCRIT
Count
HAEMATOCRIT

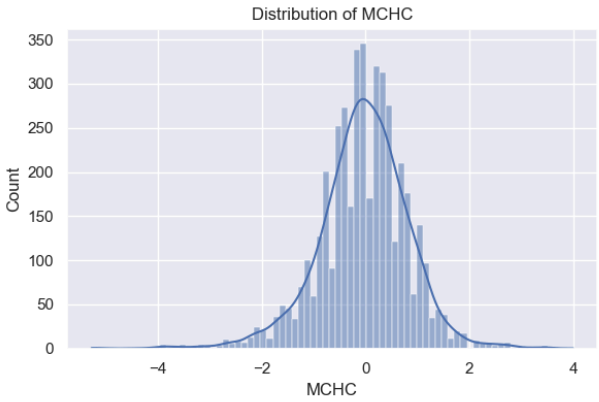

Distribution of HAEMOGLOBINS
Count
HAEMOGLOBINS

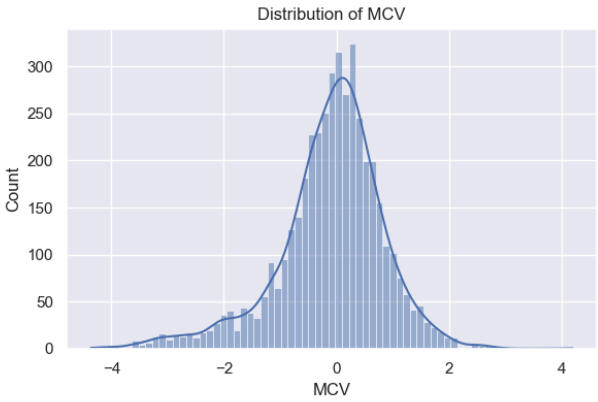

Distribution of ERYTHROCYTE
Count
ERYTHROCYTE

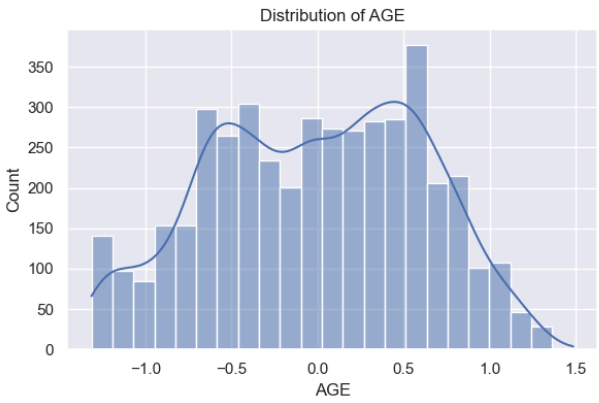

Distribution of LEUCOCYTE_log
Count
LEUCOCYTE_log


Distribution of THROMBOCYTE_log
Count
THROMBOCYTE_log


Distribution of MCH
Count
MCH


Distribution of MCHC
Count
MCHC


Distribution of MCV
Count
MCV


Distribution of AGE
Count
AGE

In [15]:
```
data
```

Out[15]:

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE	S
0	35.1	11.8	4.65	6.3	310	25.4	33.6	75.5	1	
1	43.5	14.8	5.39	12.7	334	27.5	34.0	80.7	1	
2	33.5	11.3	4.74	13.2	305	23.8	33.7	70.7	1	
3	39.1	13.7	4.98	10.5	366	27.5	35.0	78.5	1	
4	30.9	9.9	4.23	22.1	333	23.4	32.0	73.0	1	
...	...	...	...	...	...	...	...	...	...	
4407	32.8	10.4	3.49	8.1	72	29.8	31.7	94.0	92	
4408	33.7	10.8	3.67	6.7	70	29.4	32.0	91.8	92	
4409	33.2	11.2	3.47	7.2	235	32.3	33.7	95.7	93	
4410	31.5	10.4	3.15	9.1	187	33.0	33.0	100.0	98	
4411	33.5	10.9	3.44	5.8	275	31.7	32.5	97.4	99	

4412 rows × 15 columns

In [16]:
```
df_standard
```

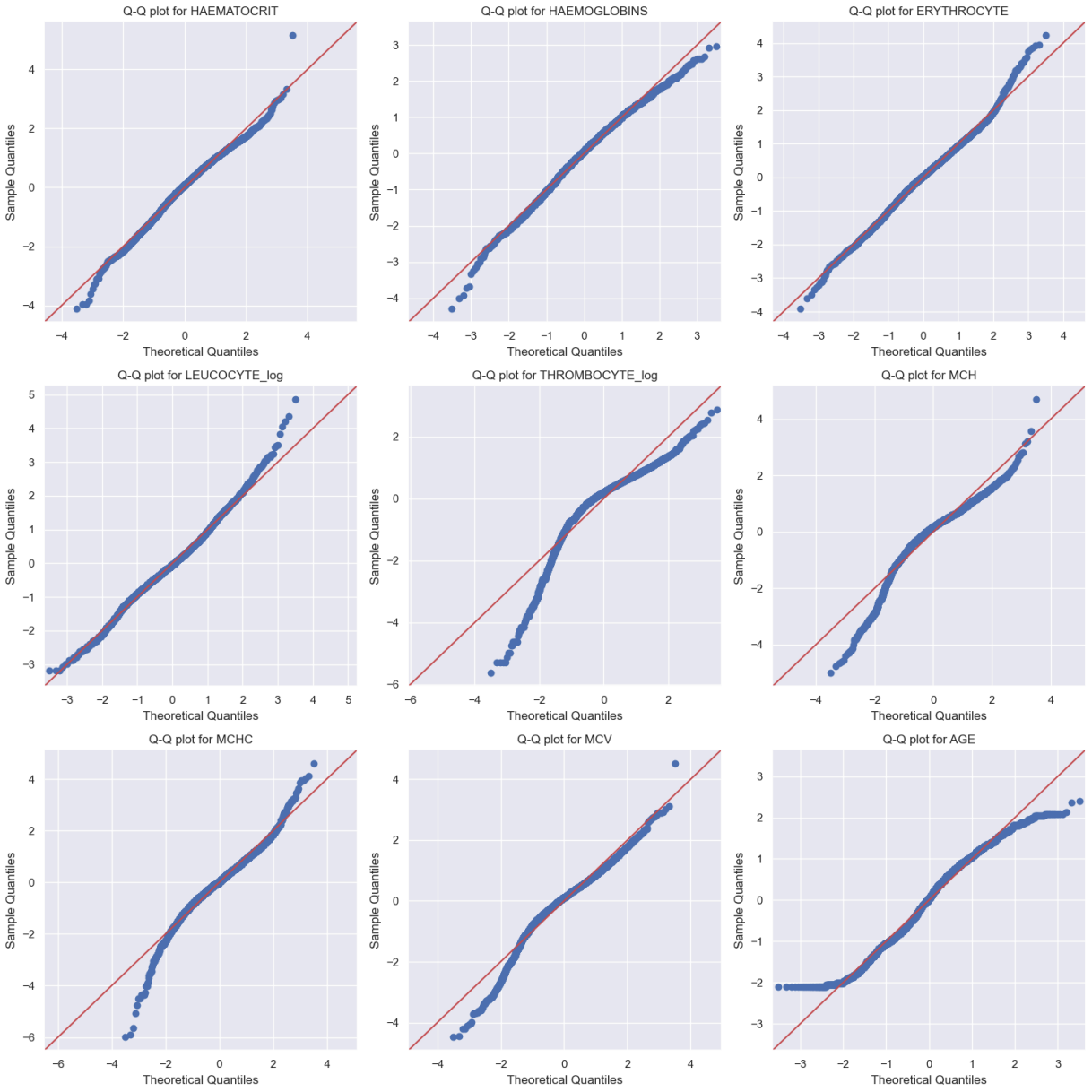
Out[16]:

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	M
0	-0.518519	-0.452826	0.138698	6.3	310	-1.060757	0.209160	-1.3287
1	0.887549	0.989713	1.082573	12.7	334	-0.274928	0.534754	-0.5705
2	-0.786341	-0.693249	0.253494	13.2	305	-1.659484	0.290558	-2.0286
3	0.151037	0.460782	0.559615	10.5	366	-0.274928	1.348738	-0.8913
4	-1.221553	-1.366434	-0.397014	22.1	333	-1.809166	-1.093215	-1.6932
...	...	...	...	...	...	...	...	
4407	-0.903514	-1.126011	-1.340889	8.1	72	0.585742	-1.337410	1.3687
4408	-0.752864	-0.933672	-1.111298	6.7	70	0.436060	-1.093215	1.0479
4409	-0.836558	-0.741334	-1.366399	7.2	235	1.521253	0.290558	1.6165
4410	-1.121119	-1.126011	-1.774561	9.1	187	1.783196	-0.279231	2.2435
4411	-0.786341	-0.885587	-1.404664	5.8	275	1.296730	-0.686223	1.8644

4412 rows × 15 columns

In [17]:
```
# Initialize a dictionary to store test results
normality_results = {}

# Determine the grid size for subplots
n_cols = 3  # Number of columns in the grid
n_rows = (len(numeric_cols) + n_cols - 1) // n_cols  # Calculate rows needed, rounded up

# Create a figure with subplots
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(15, 5 * n_rows))
axes = axes.flatten()  # Flatten the axes array for easier iteration

# Loop through each column to plot
for i, column in enumerate(numeric_cols):
    # Perform Jarque-Bera test
    stat, p = jarque_bera(df_robust[column])
    normality_results[column] = {'Statistic': stat, 'p-value': p}

    # Plot Q-Q plot in the respective subplot
```

```
    sm.qqplot(df_robust[column], line='45', fit=True, ax=axes[i])
    axes[i].set_title(f'Q-Q plot for {column}')

# Adjust layout to prevent overlap
plt.tight_layout()

# Hide any unused subplots if the number of plots isn't a perfect fit for the grid
for ax in axes[len(numeric_cols):]:
    ax.set_visible(False)

# Show the plot grid
plt.show()

# Convert results to a DataFrame for better visualization
normality_test_df = pd.DataFrame(normality_results).T  # Transpose to have columns as headers

print(normality_test_df)
```



	Statistic	p-value
HAEMATOCRIT	70.846333	4.129659e-16
HAEMOGLOBINS	72.505472	1.801511e-16
ERYTHROCYTE	50.636894	1.010039e-11
LEUCOCYTE_log	116.403108	5.289154e-26
THROMBOCYTE_log	5747.061476	0.000000e+00
MCH	2292.628795	0.000000e+00
MCHC	1366.448760	1.902926e-297
MCV	1079.133683	4.667649e-235
AGE	141.057522	2.342870e-31

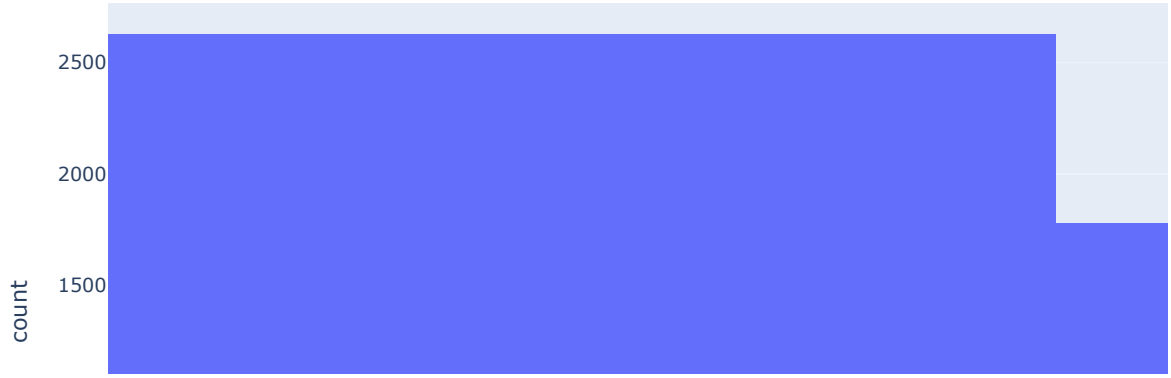
Univariate Analysis

Exploring each feature of the dataset

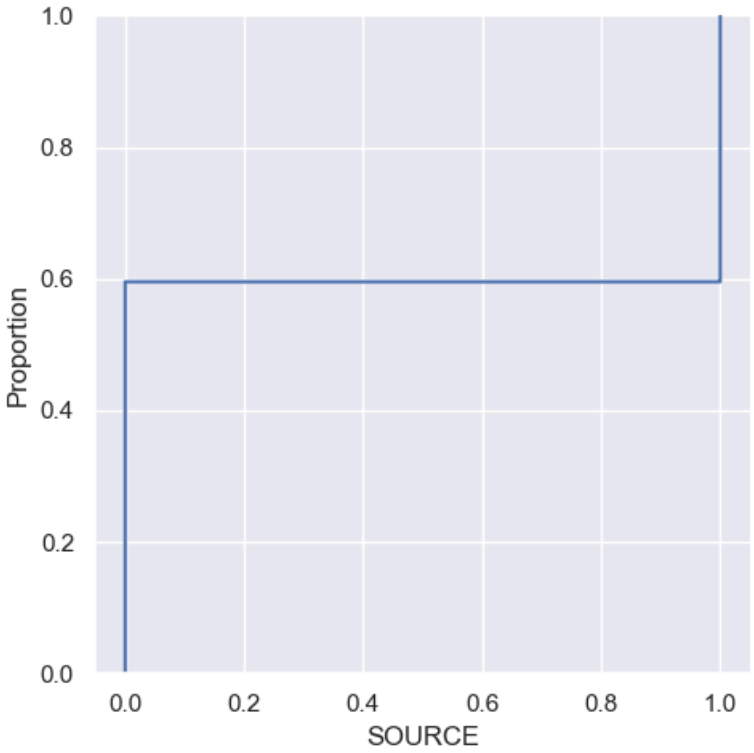
Target Variable

- **name:** 'SOURCE'
- **classes:** {'in': 1, 'out': 0}

In [18]:
```
import plotly.express as px

fig = px.histogram(data, x='SOURCE')
fig.show()
```



In [19]:
```
sns.displot(data['SOURCE'], kind='ecdf', stat='proportion');
```



- The proportion of classes is about 60:40 which is only mildy imbalanced and won't greatly bias the classification

Numerical Features

- HAEMATOCRIT
- HAEMOGLOBINS
- ERYTHROCYTE
- LEUCOCYTE
- THROMBOCYTE
- MCH
- MCHC
- MCV
- AGE

In [20]:
```
data[num_feats].describe()
```

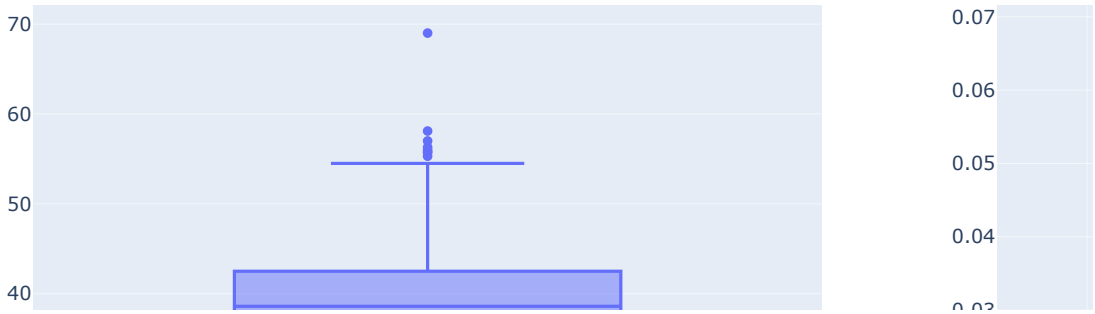
Out[20]:

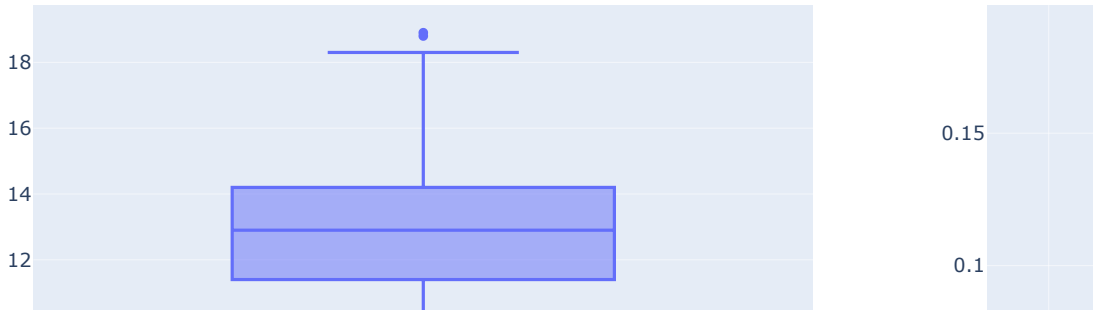
	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC
count	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000
mean	38.197688	12.741727	4.541260	8.718608	257.524479	28.234701	33.343042
std	5.974784	2.079903	0.784091	5.049041	113.972365	2.672639	1.228664
min	13.700000	3.800000	1.480000	1.100000	8.000000	14.900000	26.000000
25%	34.375000	11.400000	4.040000	5.675000	188.000000	27.200000	32.700000
50%	38.600000	12.900000	4.570000	7.600000	256.000000	28.700000	33.400000
75%	42.500000	14.200000	5.050000	10.300000	321.000000	29.800000	34.100000
max	69.000000	18.900000	7.860000	76.600000	1183.000000	40.800000	39.000000

Based on the descriptions of each of the feature variables, a few peculiar values are in:

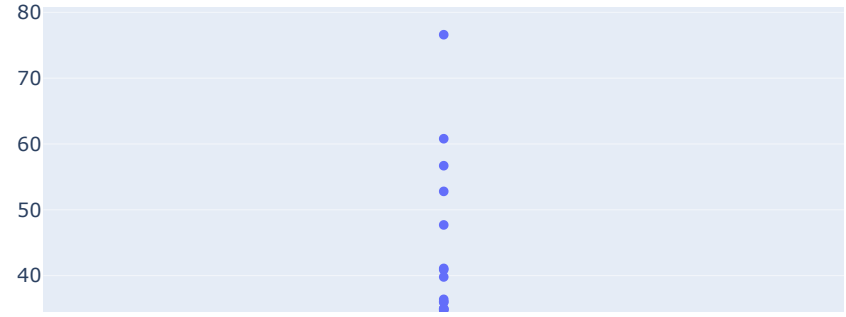
- **LEUCOCYTE** - max value is much higher than the mean value
- **THROMBOCYTE** - max and min values are very dispersed

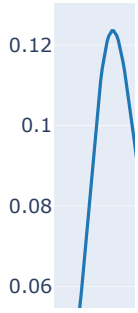
In [21]:
```
for column in num_feats:
    fig = make_subplots(rows=1, cols=2)

    #Box plot
    fig.add_trace(
    go.Box(y=data[column], name=column),
    row =1, col=1)

    # KDE plot
    hist_data = [data[column]]
    group_labels = [column]
    distplot = ff.create_distplot(hist_data, group_labels, show_hist=False, show_rug=False)
    for trace in distplot.data:
        fig.add_trace(trace, row=1, col=2)
    fig.show()
```



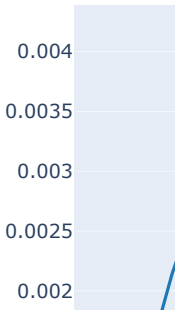

18
16
14
12
0.15
0.1

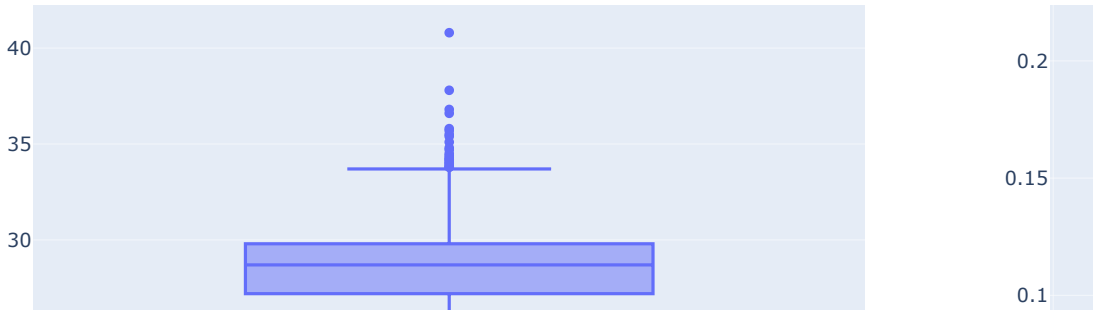

8
7
6
5
0.5
0.4
0.3

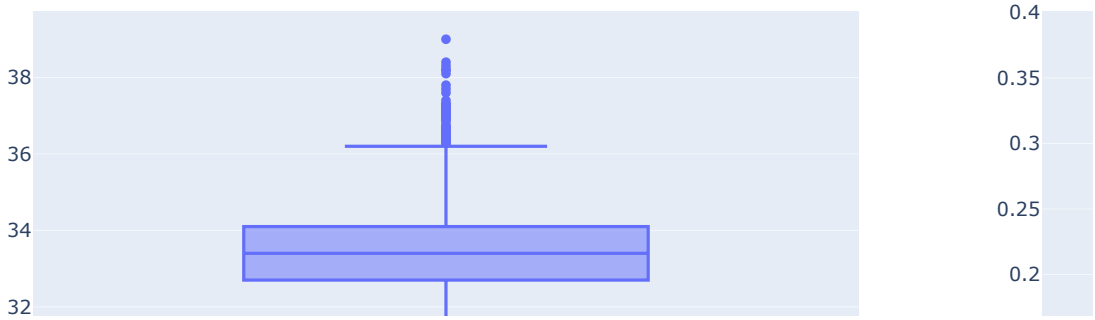

80
70
60
50
40

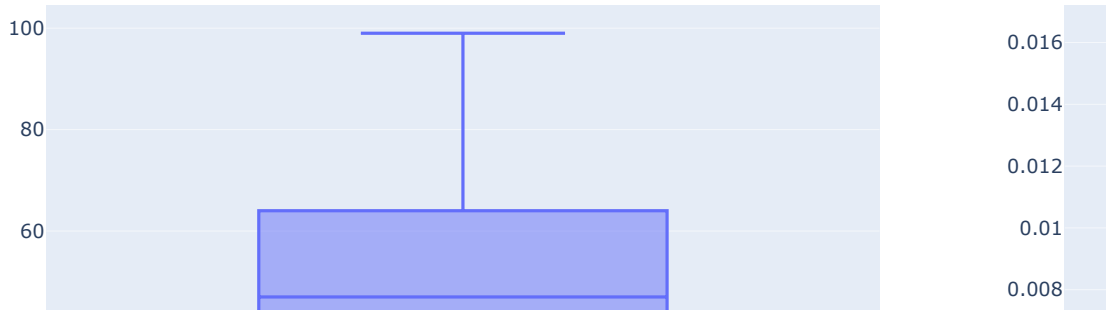

0.12
0.1
0.08
0.06


1200
1000
800
600


0.004
0.0035
0.003
0.0025
0.002


40
35
30
0.2
0.15
0.1


38
36
34
32
0.4
0.35
0.3
0.25
0.2



In [22]:
```
data.loc[data['LEUCOCYTE']==76.6]
```

Out[22]:

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE	S
3654	27.0	8.8	3.24	76.6	300	27.2	32.6	83.3	69	

In [23]:
```
data.loc[data['THROMBOCYTE']==1183]
```

Out[23]:

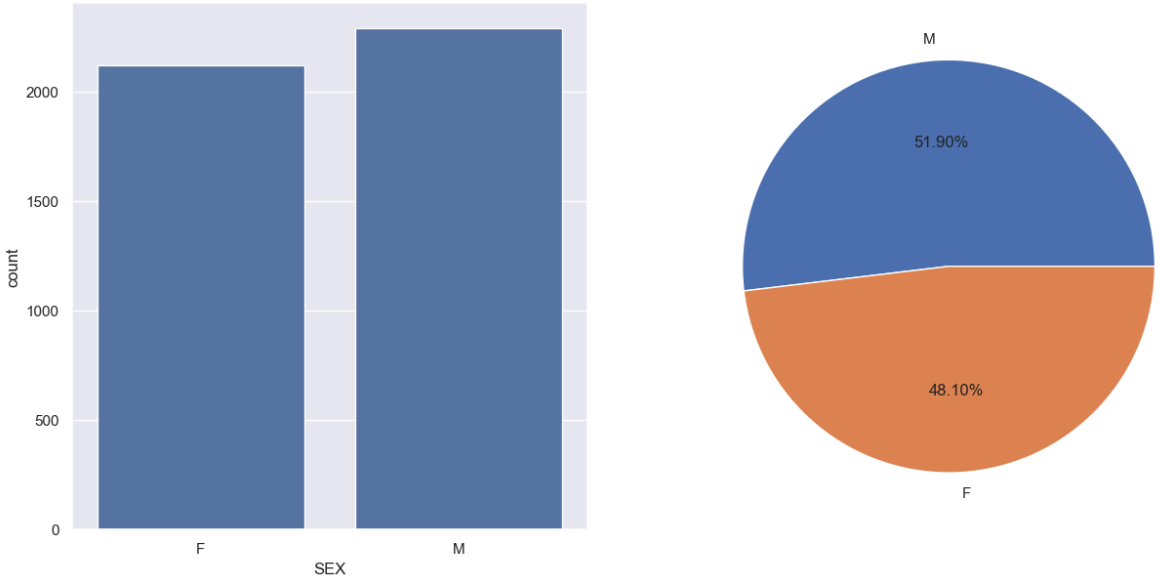
	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE	S
3777	43.5	13.7	4.53	17.9	1183	30.2	31.5	96.0	71	

- Most of the features follow an almost normal distribution with the exception of the LEUCOCYTE, THROMBOCYTE, and and AGE features.
- After inspecting the extreme values in 'LEUCOCYTE' and 'THROMBOCYTE' I have decided to keep them as the other values seem fine as this extreme values could be attributed to the reason they were visiting the hospital.

Categorical Variable:

- **name**: SEX
- **classes**: F, M

In [24]:
```
fig, ax = plt.subplots(1,2, figsize=(15,7))
sns.countplot(data, x='SEX', ax=ax[0])
plt.pie(data['SEX'].value_counts(), autopct='%0.2f%%', labels=['M', 'F'])
plt.show()
```

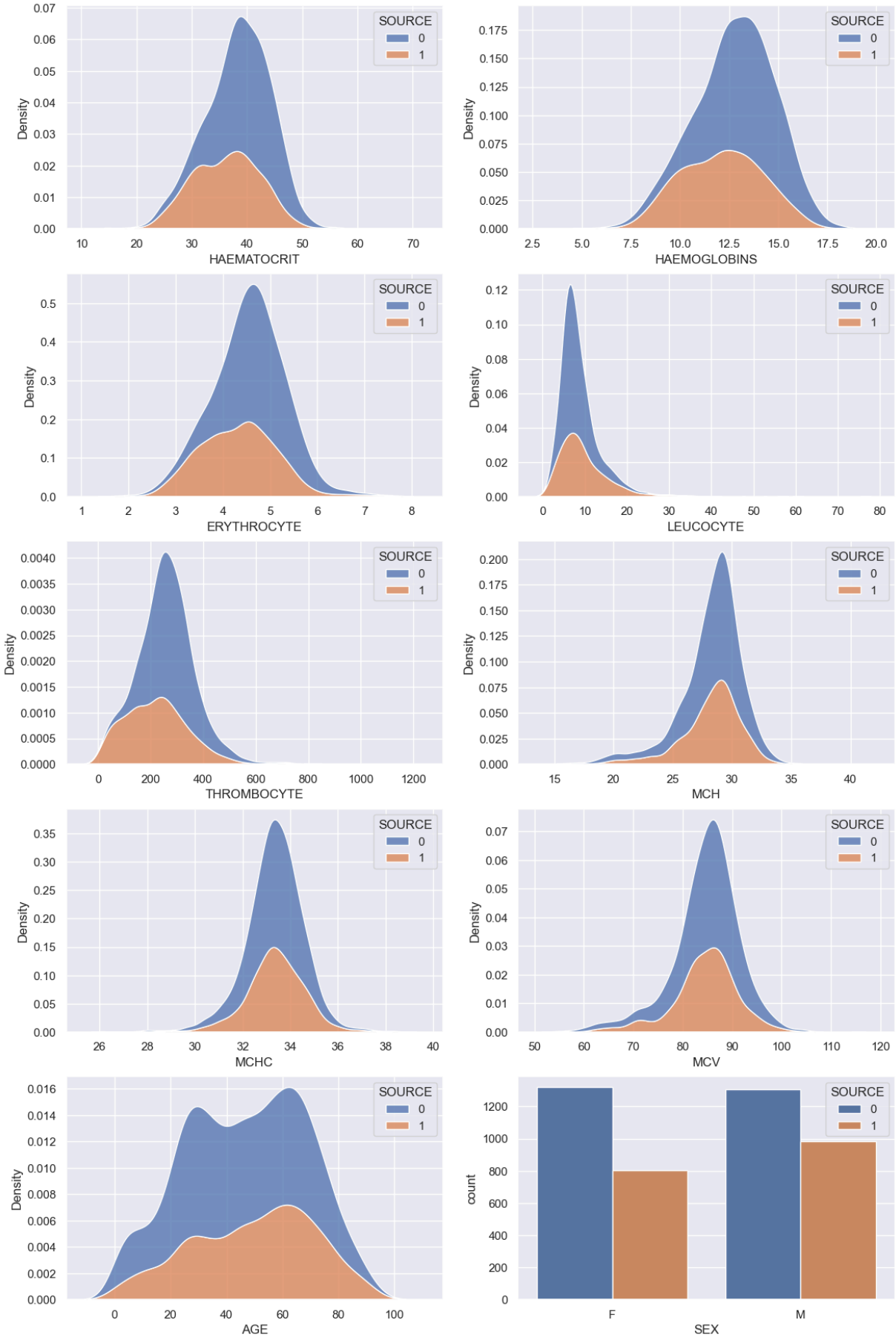


In [25]:
```
data['SEX'].value_counts()
```

Out[25]:
```
SEX
M    2290
F    2122
Name: count, dtype: int64
```

Bivariate Analysis:

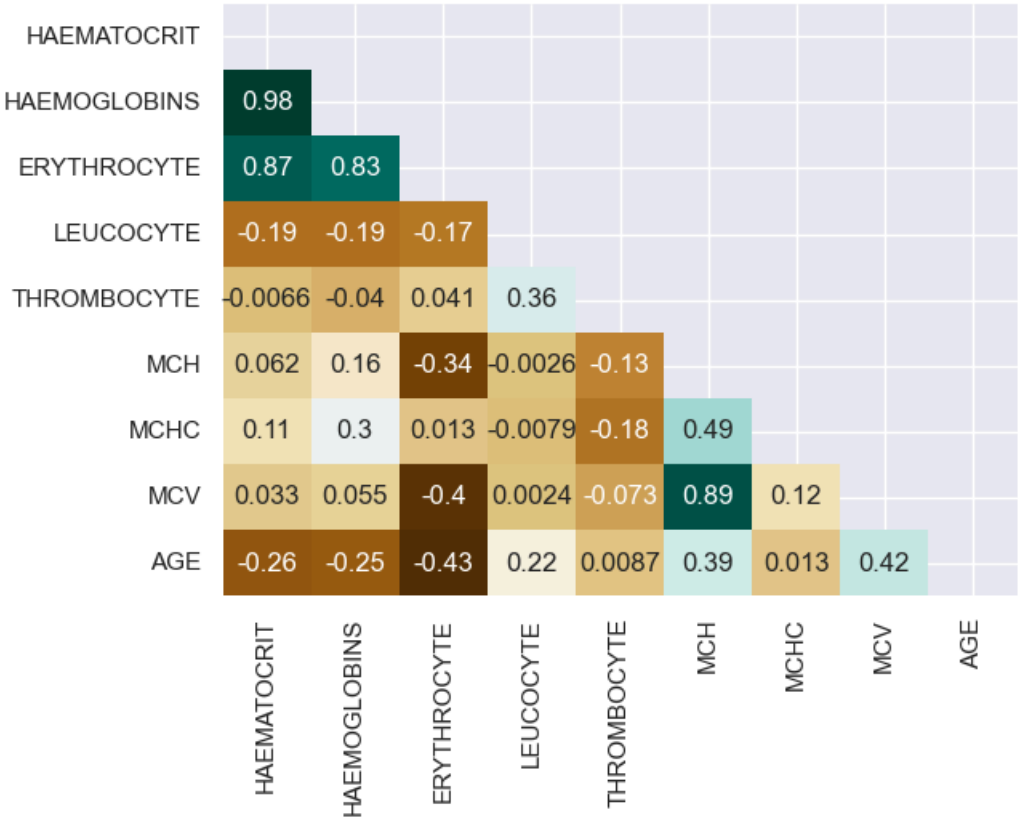
In [26]:
```
fig, axs = plt.subplots(5, 2, figsize=(14,22))
axes = [ax for rows in axs for ax in rows]
for idx, feat in enumerate(data[num_feats]):
    plot = sns.kdeplot(data=data, x=feat, hue='SOURCE', multiple='stack', ax=axes[idx])
sns.countplot(data=data, x='SEX', hue='SOURCE', ax=axs[4,1]);
```



- The features HAEMATOCRIT, HAEMOGLOBINS, ERYTHROCYTE, THROMBOCYTE, AGE show quite a difference in their distributions between the 'in' and 'out' patients, indicating that they serve as much better differentiators of the target variables than the rest.

- The categorical variables only tell us that there is a slightly higher proportion of males in in-patients than out-patients.
- There is a slightly higher older population that are admitted as in-patients but that would be expected and is not of much an insight for now.

In [27]:
```
corr = data[num_feats].corr(method='spearman')
triu_corr = np.triu(corr)
sns.heatmap(corr, annot=True, mask=triu_corr, cmap='BrBG', cbar=False)
plt.show()
```



- From the pairplots and correlation matrix we can see a high multicollinarity between 'HAEMATOCRIT' and 'HAEMOGLOBINS'(0.98) and 'MCH' and 'MCHC' (0.89). Even 'HAEMATOCRIT' and 'ERYTHROCYTE' (0.87), 'HAEMOGLOBINS' and 'ERYTHROCYTE' (0.83) have a relatively high level of correlation that can lead to overfitting of the model and lead to poor generalization performance on unseen data.

Feature Engineering:

In [28]:
```
## Train test split

# Feature set
X = data[features]
# Target set
y = data['SOURCE']

# Splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)

# train and test datasets dimensions
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[28]:
```
((3529, 10), (883, 10), (3529,), (883,))
```

In [29]:
```
# Label Encding

# Encode binary classes for the 'M' and 'F' categorical values
```

```
X_train.SEX.replace({'F':0, 'M': 1}, inplace=True)
X_test.SEX.replace({'F':0, 'M': 1}, inplace=True)
```

In [30]:
```
# Feature Scaling

# Based on the distribution and outliers observed in the univariate analysis we can resort to
# using a MinMaxScaler from sklearn

# minmax = MinMaxScaler(feature_range=(0,1))
scaler = MinMaxScaler(feature_range=(0,1))

X_train[num_feats] = scaler.fit_transform(X_train[num_feats]) #fit and transform the train set
X_test[num_feats] = scaler.transform(X_test[num_feats]) #transform the test test
```

Statistical Tests

1. Feature Selection

Check variable independence - correlation analysis: calculate the Pearson correlation coefficient for all pairs of continuous variables.

In [31]:
```
continuous_vars = data.select_dtypes(include=[float, int])
correlation_matrix = continuous_vars.corr()
print(correlation_matrix)
```

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE
HAEMATOCRIT	1.000000	0.973267	0.864989	-0.217218
HAEMOGLOBINS	0.973267	1.000000	0.818013	-0.208583
ERYTHROCYTE	0.864989	0.818013	1.000000	-0.186711
LEUCOCYTE	-0.217218	-0.208583	-0.186711	1.000000
THROMBOCYTE	-0.003562	-0.045441	0.025879	0.283262
MCH	0.083714	0.203248	-0.383666	-0.014840
MCHC	0.108098	0.314159	-0.048313	0.003194
MCV	0.054948	0.105379	-0.436041	-0.015885
AGE	-0.254799	-0.239480	-0.412858	0.191987
SOURCE	-0.271190	-0.255793	-0.232001	0.137359
LEUCOCYTE_log	-0.197946	-0.192624	-0.174717	0.925336
LEUCOCYTE_winsorized	-0.199219	-0.192023	-0.171886	0.882756
THROMBOCYTE_log	0.011820	-0.029977	0.022275	0.251929
THROMBOCYTE_winsorized	0.001435	-0.039913	0.037220	0.271965

	THROMBOCYTE	MCH	MCHC	MCV	AGE
HAEMATOCRIT	-0.003562	0.083714	0.108098	0.054948	-0.254799
HAEMOGLOBINS	-0.045441	0.203248	0.314159	0.105379	-0.239480
ERYTHROCYTE	0.025879	-0.383666	-0.048313	-0.436041	-0.412858
LEUCOCYTE	0.283262	-0.014840	0.003194	-0.015885	0.191987
THROMBOCYTE	1.000000	-0.138058	-0.198921	-0.075961	0.031064
MCH	-0.138058	1.000000	0.589830	0.931804	0.350166
MCHC	-0.198921	0.589830	1.000000	0.259672	0.041741
MCV	-0.075961	0.931804	0.259672	1.000000	0.395380
AGE	0.031064	0.350166	0.041741	0.395380	1.000000
SOURCE	-0.243202	-0.013308	0.017711	-0.022524	0.109533
LEUCOCYTE_log	0.363260	-0.008280	-0.006328	-0.004726	0.225457
LEUCOCYTE_winsorized	0.341447	-0.013967	0.002950	-0.015373	0.209678
THROMBOCYTE_log	0.897757	-0.108375	-0.200651	-0.039993	0.028940
THROMBOCYTE_winsorized	0.958161	-0.151053	-0.198687	-0.091751	0.018757

	SOURCE	LEUCOCYTE_log	LEUCOCYTE_winsorized
HAEMATOCRIT	-0.271190	-0.197946	-0.199219
HAEMOGLOBINS	-0.255793	-0.192624	-0.192023
ERYTHROCYTE	-0.232001	-0.174717	-0.171886
LEUCOCYTE	0.137359	0.925336	0.882756
THROMBOCYTE	-0.243202	0.363260	0.341447
MCH	-0.013308	-0.008280	-0.013967
MCHC	0.017711	-0.006328	0.002950
MCV	-0.022524	-0.004726	-0.015373
AGE	0.109533	0.225457	0.209678
SOURCE	1.000000	0.095410	0.124322
LEUCOCYTE_log	0.095410	1.000000	0.970036
LEUCOCYTE_winsorized	0.124322	0.970036	1.000000
THROMBOCYTE_log	-0.311279	0.341440	0.314627
THROMBOCYTE_winsorized	-0.280621	0.364698	0.338940

	THROMBOCYTE_log	THROMBOCYTE_winsorized
HAEMATOCRIT	0.011820	0.001435
HAEMOGLOBINS	-0.029977	-0.039913
ERYTHROCYTE	0.022275	0.037220
LEUCOCYTE	0.251929	0.271965
THROMBOCYTE	0.897757	0.958161
MCH	-0.108375	-0.151053
MCHC	-0.200651	-0.198687
MCV	-0.039993	-0.091751
AGE	0.028940	0.018757
SOURCE	-0.311279	-0.280621
LEUCOCYTE_log	0.341440	0.364698
LEUCOCYTE_winsorized	0.314627	0.338940
THROMBOCYTE_log	1.000000	0.931485
THROMBOCYTE_winsorized	0.931485	1.000000

Strong Positive Correlations: HAEMATOCRIT and HAEMOGLOBINS: A correlation of 0.973. 0.973 indicates almost a linear relationship. This is expected since hematocrit is the volume percentage of red blood cells in blood, and hemoglobin is a protein in those cells. MCH and MCV: A correlation of 0.932. 0.932 suggests that the mean corpuscular hemoglobin (MCH, the average mass of hemoglobin per red blood cell) and the mean corpuscular volume (MCV, the average volume of a red blood cell) are closely linked, which makes sense as both describe properties of red blood cells. Strong Negative Correlations:

ERYTHROCYTE and MCV: The −0.436. −0.436 correlation indicates that as the erythrocyte count increases, the mean corpuscular volume tends to decrease. This could suggest that higher red blood cell counts are associated with

smaller sized cells, potentially pointing to certain microcytic anemias. AGE and ERYTHROCYTE: A correlation of −0.413. −0.413 may imply that erythrocyte counts tend to decrease with age, which could be relevant in clinical settings. Moderate to Low Correlations: LEUCOCYTE and THROMBOCYTE: A correlation of 0.283. 0.283 might suggest a moderate association, indicating possible simultaneous responses to certain conditions like infections or inflammations. AGE and MCV: The positive correlation of 0.395. 0.395 could indicate changes in red blood cell size with aging.

In [32]:
```
# Chi-Square Test of Independence for SEX and SOURCE
contingency_table = pd.crosstab(data['SEX'], data['SOURCE'])
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
print(f"Chi-Square Statistic: {chi2}, p-value: {p}")
```

```
Chi-Square Statistic: 11.626002500279858, p-value: 0.0006503617798734924
```

Chi-Square Statistic: The Chi-Square statistic of 11.626 suggests that there is a disparity between the observed counts in the categories of SEX and SOURCE and what would be expected if they were independent of each other. A higher Chi-Square value generally indicates a stronger divergence from the null hypothesis of independence.
p-value: The p-value is a measure of the probability that the observed data (or something more extreme) would occur if the null hypothesis (in this case, the hypothesis that there is no association between SEX and SOURCE) were true. The p-value is 0.0006503617798734924, which is quite low.

Conclusion:
Since the p-value is less than 0.05 (a common threshold in statistical testing), we can reject the null hypothesis. This indicates that there is a statistically significant association between the variables SEX and SOURCE.
In simpler terms, the gender (SEX) of the subjects in our dataset appears to be related to the category of SOURCE. This could mean that whether the SOURCE is "in" or "out" depends to some extent on the gender, or vice versa.

In [33]:
```
# Chi-Square Test of Independence for SEX and SOURCE
contingency_table = pd.crosstab(data['HAEMATOCRIT'], data['HAEMOGLOBINS'])
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
print(f"Chi-Square Statistic: {chi2}, p-value: {p}")
```

```
Chi-Square Statistic: 182515.94615897577, p-value: 0.0
```

The p-value is very close to zero (0.0), which indicates strong evidence against the null hypothesis. It means that there is a significant association between 'HAEMATOCRIT' and 'HAEMOGLOBINS'.

The Chi-Square Statistic is considerably high, suggesting a strong association between the two variables.

In [34]:
```
# Chi-Square Test of Independence for SEX and SOURCE
contingency_table = pd.crosstab(data['MCH'], data['MCV'])
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
print(f"Chi-Square Statistic: {chi2}, p-value: {p}")
```

```
Chi-Square Statistic: 271188.93780701736, p-value: 0.0
```

The p-value is very close to zero (0.0), which indicates strong evidence against the null hypothesis. It means that there is a significant association between 'MCH' and 'MCV'.

The Chi-Square Statistic is considerably high, suggesting a strong association between the two variables.

In [35]:
```
#Columns dropped based on correlation of data that would render the feature values redundant

X_train.drop(['HAEMATOCRIT','MCH'], axis=1, inplace=True)
X_test.drop(['HAEMATOCRIT','MCH'], axis=1, inplace=True)
```

2. Does the inclusion of certain variables (e.g., AGE, SEX, and HAEMOGLOBINS) significantly improve the model's ability to explain the variation in the dependent variable (SOURCE)?"

In [36]:
```
# Define the models
model1 = sm.OLS.from_formula("SOURCE ~ 1", data=data)
model2 = sm.OLS.from_formula("SOURCE ~ AGE + SEX + HAEMOGLOBINS", data=data)

# Fit the models
result1 = model1.fit()
result2 = model2.fit()
```

```
# Perform the likelihood ratio test
lrt_stat = -2 * (result1.llf - result2.llf)
lrt_pval = stats.chi2.sf(lrt_stat, df=result2.df_model - result1.df_model)

print("Likelihood Ratio Test:")
print("Test Statistic:", lrt_stat)
print("p-value:", lrt_pval)
```

```
Likelihood Ratio Test:
Test Statistic: 394.30920842178966
p-value: 3.782675415956191e-85
```

The test statistic (394.31) indicates that the alternative model (Model 2) is significantly better than the null model (Model 1) in explaining the variation in the dependent variable (SOURCE).
The p-value (3.78e-85) is extremely small, which means that the probability of observing the test statistic (or a more extreme value) under the null hypothesis is virtually zero.
Therefore, we can reject the null hypothesis (Model 1) and conclude that the alternative model (Model 2) is a better fit to the data.
In other words, the inclusion of the additional variables (AGE, SEX, and HAEMATOCRIT) in the alternative model significantly improves the model's ability to explain the variation in SOURCE. This suggests that these variables are important predictors of SOURCE.

3. Does the inclusion of certain variables (e.g., ERYTHROCYTE, LEUCOCYTE, THROMBOCYTE, MCHC and MCV) significantly improve the model's ability to explain the variation in the dependent variable (SOURCE)?"

In [37]:
```
# Define the models
model1 = sm.OLS.from_formula("SOURCE ~ AGE + SEX + HAEMOGLOBINS", data=data)
model2 = sm.OLS.from_formula("SOURCE ~ HAEMOGLOBINS + ERYTHROCYTE + LEUCOCYTE + THROMBOCYTE + MCHC + MCV

# Fit the models
result1 = model1.fit()
result2 = model2.fit()

# Perform the likelihood ratio test
lrt_stat = -2 * (result1.llf - result2.llf)
lrt_pval = stats.chi2.sf(lrt_stat, df=result2.df_model - result1.df_model)

print("Likelihood Ratio Test:")
print("Test Statistic:", lrt_stat)
print("p-value:", lrt_pval)
```

```
Likelihood Ratio Test:
Test Statistic: 424.5929945748985
p-value: 1.481308996609937e-89
```

In [38]:
```
result1.summary()
```

Out[38]:

OLS Regression Results

Dep. Variable:	SOURCE	**R-squared:**	0.085
Model:	OLS	**Adj. R-squared:**	0.085
Method:	Least Squares	**F-statistic:**	137.4
Date:	Sat, 04 May 2024	**Prob (F-statistic):**	4.42e-85
Time:	17:08:02	**Log-Likelihood:**	-2922.8
No. Observations:	4412	**AIC:**	5854.
Df Residuals:	4408	**BIC:**	5879.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.1675	0.052	22.459	0.000	1.066	1.269
SEX[T.M]	0.1378	0.015	9.208	0.000	0.108	0.167
AGE	0.0008	0.000	2.431	0.015	0.000	0.001
HAEMOGLOBINS	-0.0685	0.004	-18.513	0.000	-0.076	-0.061

Omnibus:	27946.985	**Durbin-Watson:**	0.571
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	511.049
Skew:	0.368	**Prob(JB):**	1.06e-111
Kurtosis:	1.504	**Cond. No.**	389.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [39]:
```
result2.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	SOURCE	**R-squared:**	0.169
Model:	OLS	**Adj. R-squared:**	0.168
Method:	Least Squares	**F-statistic:**	112.2
Date:	Sat, 04 May 2024	**Prob (F-statistic):**	3.03e-171
Time:	17:08:02	**Log-Likelihood:**	-2710.5
No. Observations:	4412	**AIC:**	5439.
Df Residuals:	4403	**BIC:**	5497.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.4845	0.708	3.512	0.000	1.097	3.872
SEX[T.M]	0.0927	0.015	6.364	0.000	0.064	0.121
HAEMOGLOBINS	-0.0102	0.026	-0.392	0.695	-0.061	0.041
ERYTHROCYTE	-0.1465	0.072	-2.029	0.042	-0.288	-0.005
LEUCOCYTE	0.0148	0.001	10.125	0.000	0.012	0.018
THROMBOCYTE	-0.0013	6.4e-05	-19.611	0.000	-0.001	-0.001
MCHC	-0.0082	0.012	-0.683	0.494	-0.032	0.015
MCV	-0.0108	0.004	-2.593	0.010	-0.019	-0.003
AGE	0.0009	0.000	2.493	0.013	0.000	0.002

Omnibus:	1629.669	**Durbin-Watson:**	0.699
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	329.732
Skew:	0.400	**Prob(JB):**	2.51e-72
Kurtosis:	1.925	**Cond. No.**	3.14e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.14e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

The p-value is extremely close to zero, indicating strong evidence against the null hypothesis. It suggests that the more complex model (model2) provides a significantly better fit to the data compared to the simpler model (model1).
The Test Statistic is relatively high, indicating a substantial difference in fit between the two models.
In summary, the result suggests that including additional variables in the model significantly improves its ability to explain the variation in the response variable 'SOURCE'.

Classification Model

Following the classification with each of these methods, we will then evaluate each of these models using the following metrics:

- **Accuracy**: Provide the overall number of correct predictions divided by the total number of predictions.
- **Confusion Matrix**: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned).
- **Precision**: A measure of a classifiers exactness.

- **Recall**: A measure of a classifiers completeness
- **F2 Score** (or F-score): A weighted average of precision and recall, with more importance to recall as we want to avoid false-negative errors more than false-positive errors.

1. Logistic Regression:

The basis for choosing logistic regression is mainly given the fact that the target variable is binary in nature and we can use the logistic regression as a base model to compare with the other models.

In [40]:
```
logreg = LogisticRegression(random_state=42)

#Fit the model
logreg.fit(X_train, y_train)
```

Out[40]:
```
LogisticRegression
LogisticRegression(random_state=42)
```

In [41]:
```
logreg_precision, logreg_recall, logreg_f2score, logreg_support = precision_recall_fscore_support(y_test
logreg_accuracy = accuracy_score(y_test, logreg.predict(X_test))
logreg_rocauc = roc_auc_score(y_test, logreg.predict(X_test))
logreg_confusion_matrix = confusion_matrix(y_test, logreg.predict(X_test))
```

In [42]:
```
print(f'---Metrics---\n accuracy: {logreg_accuracy*100:.0f}%\n precision: {logreg_precision}\n\
 recall: {logreg_recall}\n f2_score: {logreg_f2score}\n\
 ROC AUC Score: {logreg_rocauc:0.2f}\n\n\
 confusion matrix:\n {logreg_confusion_matrix}')
```

```
---Metrics---
 accuracy: 70%
 precision: [0.6969697  0.70403587]
 recall: [0.87452471 0.43977591]
 f2_score: [0.83212735 0.47546941]
 ROC AUC Score: 0.66

 confusion matrix:
 [[460  66]
 [200 157]]
```

2. Support Vector Machine:

In [43]:
```
svm =SVC(kernel='rbf', C=14, random_state=42)
svm.fit(X_train,y_train)
```

Out[43]:
```
SVC
SVC(C=14, random_state=42)
```

In [44]:
```
svm_precision, svm_recall, svm_f2score, svm_support = precision_recall_fscore_support(y_test, svm.predic
svm_accuracy = accuracy_score(y_test, svm.predict(X_test))
svm_rocauc = roc_auc_score(y_test, svm.predict(X_test))
svm_confusion_matrix = confusion_matrix(y_test, svm.predict(X_test))
```

In [45]:
```
print(f'---Metrics---\n accuracy: {svm_accuracy*100:.1f}%\n precision: {svm_precision}\n\
 recall: {svm_recall}\n f2_score: {svm_f2score}\n\
 ROC AUC Score: {svm_rocauc:0.2f}\n\n\
 confusion matrix:\n {svm_confusion_matrix}')
```

```
---Metrics---
 accuracy: 75.1%
 precision: [0.75331126 0.74551971]
 recall: [0.86501901 0.58263305]
 f2_score: [0.8401034 0.609256 ]
 ROC AUC Score: 0.72

 confusion matrix:
 [[455  71]
 [149 208]]
```

3. K-Nearest Neighbor:

In [46]:
```
# Training the K-NN model on the Training set
knn = KNeighborsClassifier(n_neighbors=9, metric='minkowski', p=2)
knn.fit(X_train, y_train)
```

Out[46]:
```
KNeighborsClassifier(n_neighbors=9)
```

In [47]:
```
knn_precision, knn_recall, knn_f2score, knn_support = precision_recall_fscore_support(y_test, knn.predic
knn_accuracy = accuracy_score(y_test, knn.predict(X_test))
knn_rocauc = roc_auc_score(y_test, knn.predict(X_test))
knn_confusion_matrix = confusion_matrix(y_test, knn.predict(X_test))
```

In [48]:
```
print(f'---Metrics---\n accuracy: {knn_accuracy*100:.1f}%\n precision: {knn_precision}\n\
 recall: {knn_recall}\n f2_score: {knn_f2score}\n\
 ROC AUC Score: {knn_rocauc:0.2f}\n\n\
 confusion matrix:\n {knn_confusion_matrix}')
```

```
---Metrics---
 accuracy: 73.5%
 precision: [0.75       0.70568562]
 recall: [0.83269962 0.59103641]
 f2_score: [0.81473214 0.61088593]
 ROC AUC Score: 0.71

 confusion matrix:
 [[438  88]
 [146 211]]
```

4. Random Forest:

In [49]:
```
random_forest = RandomForestClassifier(n_estimators=40, random_state=42)

# Fit the model
random_forest.fit(X_train, y_train)
```

Out[49]:
```
RandomForestClassifier(n_estimators=40, random_state=42)
```

In [50]:
```
random_forest_precision, random_forest_recall, random_forest_f2score, random_forest_support = precision_
random_forest_accuracy = accuracy_score(y_test, random_forest.predict(X_test))
random_forest_rocauc = roc_auc_score(y_test, random_forest.predict(X_test))
random_forest_confusion_matrix = confusion_matrix(y_test, random_forest.predict(X_test))
```

In [51]:
```
print(f'---Metrics---\n accuracy: {random_forest_accuracy*100:.1f}%\n precision: {random_forest_precisio
 recall: {random_forest_recall}\n f2_score: {random_forest_f2score}\n\
 ROC AUC Score: {random_forest_rocauc:0.2f}\n\n\
 confusion matrix:\n {random_forest_confusion_matrix}')
```

```
---Metrics---
 accuracy: 76.0%
 precision: [0.7670068  0.74576271]
 recall: [0.85741445 0.6162465 ]
 f2_score: [0.83766716 0.63842136]
 ROC AUC Score: 0.74

 confusion matrix:
 [[451  75]
 [137 220]]
```

Model Optimization

Based the performance of the models the following models selected for optimization are:

a. Support Vector Machine:

In [52]:
```
# Hyperparameters
param_grid = {'C': [1, 5, 10, 20, 40, 50, 100],
              'kernel': ['rbf'],
              'degree':[1, 2, 4, 6]
             }

# Random search for best hyperparameters
search = GridSearchCV(SVC(random_state=42),
                          param_grid,
                          scoring='accuracy',
                          cv=3,
                          verbose=1)

search.fit(X_train, y_train)

# Best parameters for Support vector classifier
search.best_params_

best_svc = search.best_estimator_
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits
```

In [53]:
```
# Fit the model
best_svc.fit(X_train, y_train)

svm_precision_opt, svm_recall_opt, svm_f2score_opt, svm_support_opt = precision_recall_fscore_support(y_
svm_accuracy_opt = accuracy_score(y_test, best_svc.predict(X_test))


print(f'---Metrics---\n accuracy: {svm_accuracy_opt*100:.1f}%\n precision: {svm_precision_opt}\n\
 recall: {svm_recall_opt}\n f2_score: {svm_f2score_opt}')
```

```
---Metrics---
 accuracy: 74.7%
 precision: [0.75041322 0.74100719]
 recall: [0.86311787 0.57703081]
 f2_score: [0.83794758 0.60375147]
```

In [68]:
```
best_svc.coef_
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[68], line 1
----> 1 best_svc.coef_

File c:\Users\supha\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\svm\_base.py:656, i
n BaseLibSVM.coef_(self)
    649 """Weights assigned to the features when `kernel="linear"`.
    650
    651 Returns
    652 -------
    653 ndarray of shape (n_features, n_classes)
    654 """
    655 if self.kernel != "linear":
--> 656     raise AttributeError("coef_ is only available when using a linear kernel")
    658 coef = self._get_coef()
    660 # coef_ being a read-only property, it's better to mark the value as
    661 # immutable to avoid hiding potential bugs for the unsuspecting user.

AttributeError: coef_ is only available when using a linear kernel
```

b. K-Nearest Neighbor:

In [55]:
```
# GRID SEARCH
# Define the parameter values that should be searched
param_grid = {'n_neighbors': [5, 7],
              'metric': ['cosine'],
              'weights': ['distance'],
              'algorithm': ['auto'],
              'leaf_size': [10]}
```

```
# Create a base model
knn = KNeighborsClassifier()

# Instantiate the grid search model
grid_search = GridSearchCV(knn, param_grid, cv=3, verbose=1, n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Check the best parameters
print(grid_search.best_params_)

# Use the best estimator for our predictions on test
best_knn = grid_search.best_estimator_
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
{'algorithm': 'auto', 'leaf_size': 10, 'metric': 'cosine', 'n_neighbors': 7, 'weights': 'distance'}
```

In [56]:
```
# Fit the model
best_knn.fit(X_train, y_train)

knn_precision_opt, knn_recall_opt, knn_f2score_opt, knn_support_opt = precision_recall_fscore_support(y_
knn_accuracy_opt = accuracy_score(y_test, best_knn.predict(X_test))


print(f'---Metrics---\n accuracy: {knn_accuracy_opt*100:.1f}%\n precision: {knn_precision_opt}\n\
 recall: {knn_recall_opt}\n f2_score: {knn_f2score_opt}')
```

```
---Metrics---
 accuracy: 74.9%
 precision: [0.76666667 0.71565495]
 recall: [0.83079848 0.62745098]
 f2_score: [0.8171279  0.64330844]
```

c. Random Forest Classifier:

In [57]:
```
# Grid Search
# param_grid = {
#     'n_estimators': [100, 200, 350, 500],  # More options for number of trees
#     'max_features': ['sqrt', 'log2'],      # Two types of features to consider
#     'criterion': ['gini', 'entropy', 'log_loss'],  # Tree split criteria
#     'min_samples_leaf': [1, 2, 4],         # Minimum number of samples at leaf node
#     'min_samples_split': [2, 5, 7],        # Minimum number of samples to split
#     'max_depth': [None, 10, 20, 30],       # Maximum depth of the tree
#     'bootstrap': [True, False]             # Whether bootstrap samples are used
# }
param_grid = {'n_estimators': [350],
              'max_features': ['log2'],
              'criterion': ['gini', 'entropy', 'log_loss'],
             'min_samples_leaf': [1],
             'min_samples_split': [7],
              'max_depth' : [None],
             'bootstrap': [False]}

# Create a base model
random_forest = RandomForestClassifier(random_state=42)

# Instantiate the grid search model
grid_search = GridSearchCV(random_forest, param_grid, cv=3, verbose=1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Check the best parameters
print(grid_search.best_params_)

# You can now use the best estimator for further predictions
best_rf = grid_search.best_estimator_
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
{'bootstrap': False, 'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2', 'min_samples_lea
f': 1, 'min_samples_split': 7, 'n_estimators': 350}
```

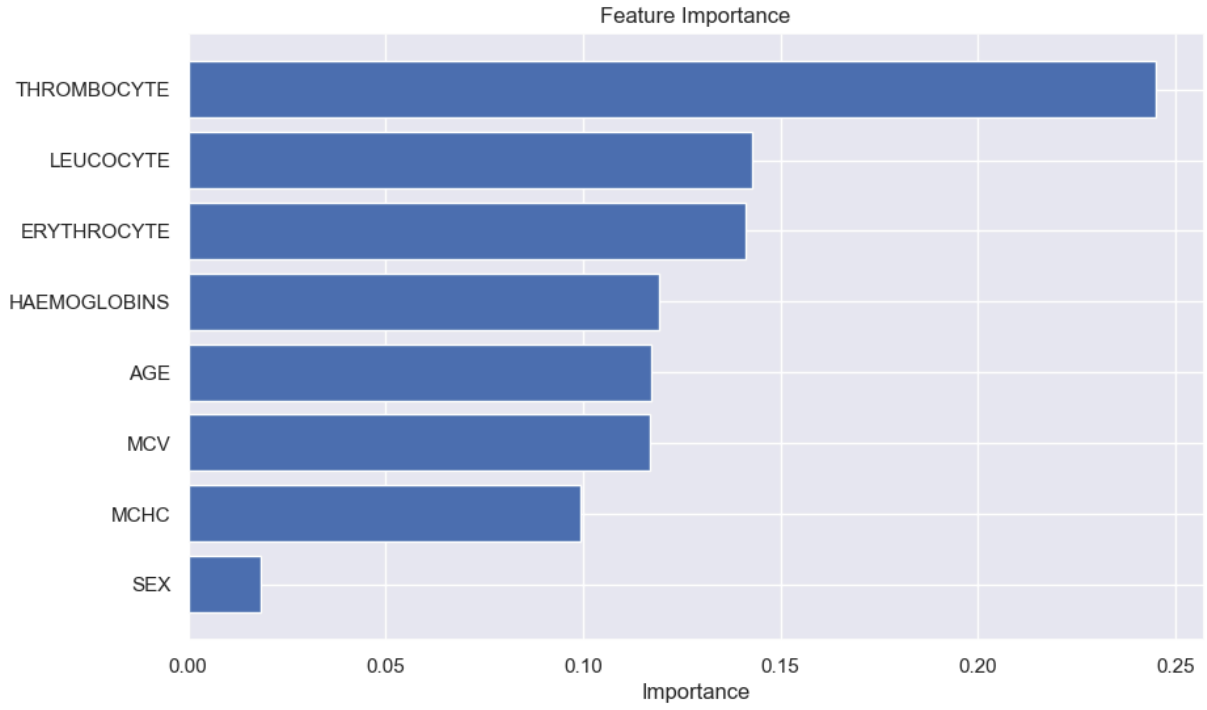
In [58]:
```
# Fit the model
best_rf.fit(X_train, y_train)
```

```
random_forest_precision_opt, random_forest_recall_opt, random_forest_f2score_opt, random_forest_support_
random_forest_accuracy_opt = accuracy_score(y_test, best_rf.predict(X_test))

print(f'---Metrics---\n accuracy: {random_forest_accuracy_opt*100:.1f}%\n precision: {random_forest_prec
 recall: {random_forest_recall_opt}\n f2_score: {random_forest_f2score_opt}')
```

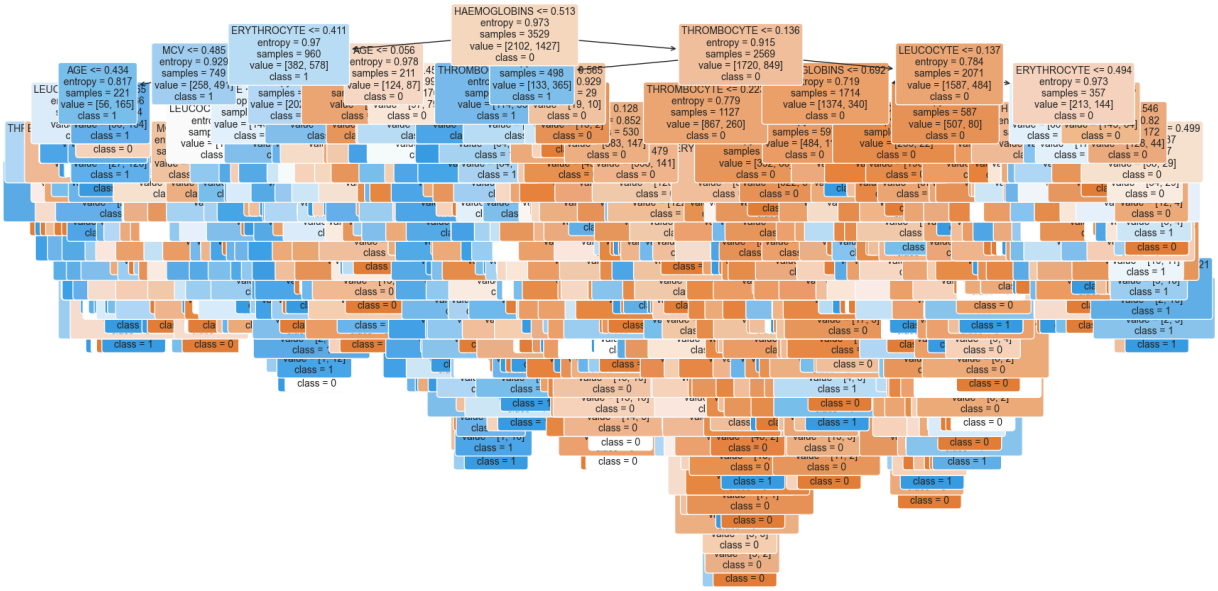
```
---Metrics---
 accuracy: 76.8%
 precision: [0.77816291 0.74836601]
 recall: [0.85361217 0.64145658]
 f2_score: [0.83737411 0.66032295]
```

Feature Importance

In [59]:
```
best_rf_feature_importance = pd.DataFrame({'features': best_rf.feature_names_in_, 'importance': best_rf.
best_rf_feature_importance.sort_values(
    by='importance',
    ascending=True,
    inplace=True
)
plt.figure(figsize=(10, 6))
plt.barh(best_rf_feature_importance['features'], best_rf_feature_importance['importance'])
plt.xlabel('Importance')
plt.title('Feature Importance')
plt.show()
```



In [60]:
```
from sklearn import tree

# Extract one of the trees from the forest
chosen_tree = best_rf.estimators_[0]

# Set the figure size
plt.figure(figsize=(20,10))

# Plot the decision tree
tree.plot_tree(chosen_tree,
               filled=True,
               feature_names=best_rf.feature_names_in_,
               class_names=best_rf.classes_.astype(str),
               rounded=True,
               fontsize=10)
plt.show()
```

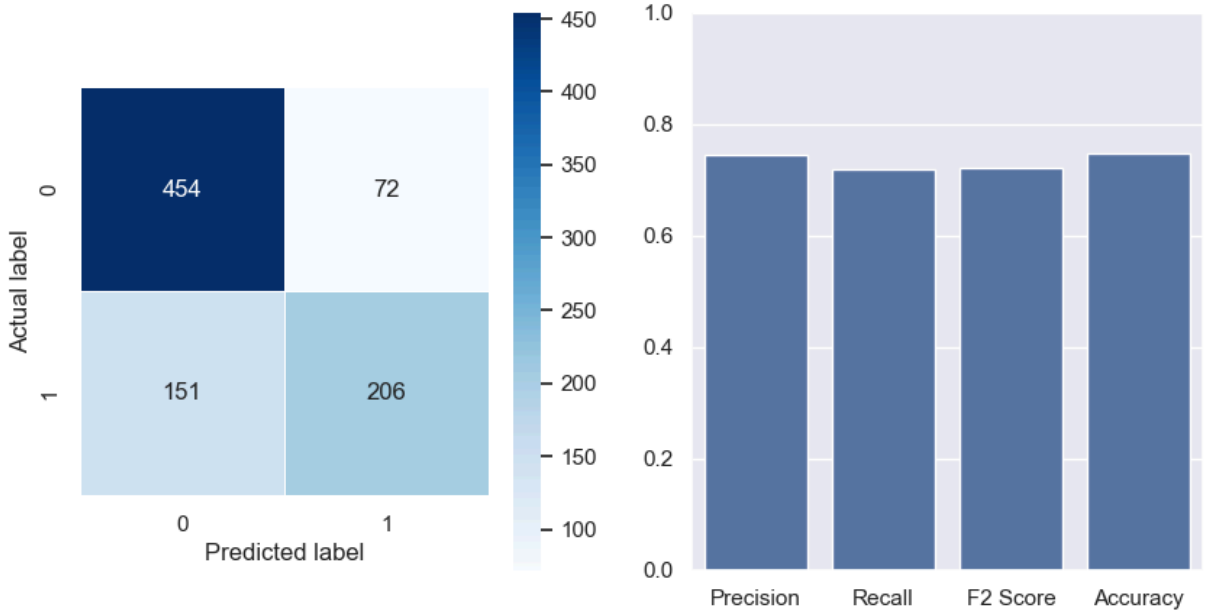


Results

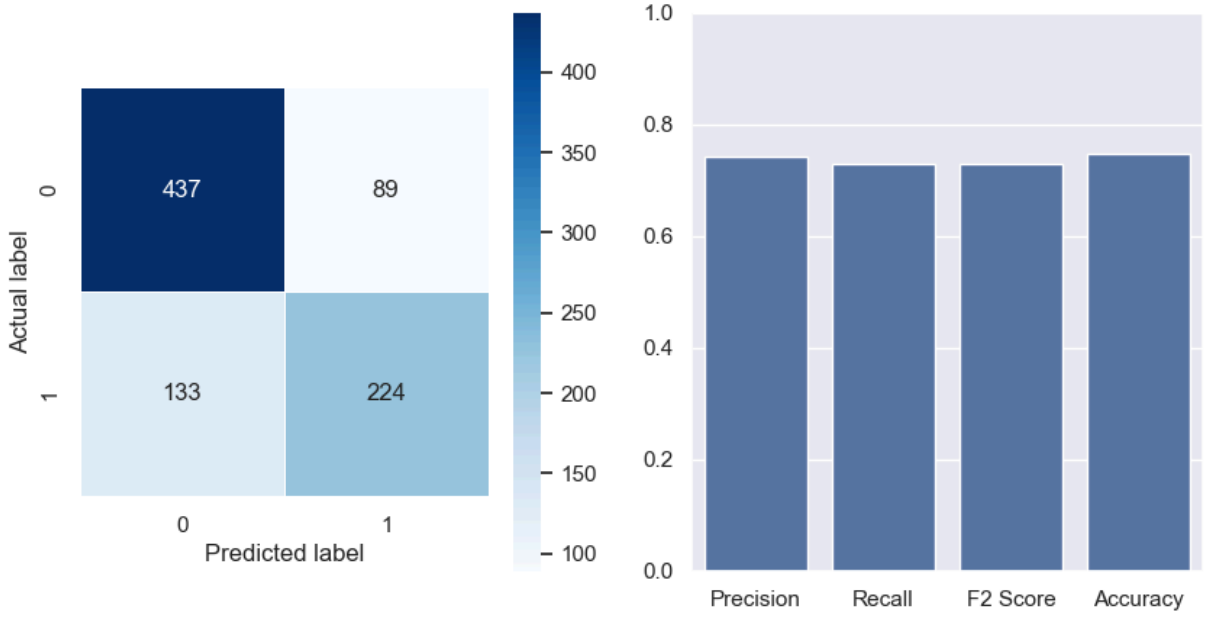
In [61]:
```
# Create a function to plot confusion matrix and classification metrics
def plot_cm(cm, precision, recall, f2score, accuracy, title):
    plt.figure(figsize=(10,5))

    plt.subplot(1, 2, 1)
    sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap = 'Blues');
    plt.ylabel('Actual label');
    plt.xlabel('Predicted label');

    plt.subplot(1, 2, 2)
    metrics = [precision, recall, f2score, accuracy]
    metric_names = ['Precision', 'Recall', 'F2 Score', 'Accuracy']
    sns.barplot(x=metric_names, y=metrics)
    plt.ylim(0, 1)

    plt.suptitle(title, size = 15)
```

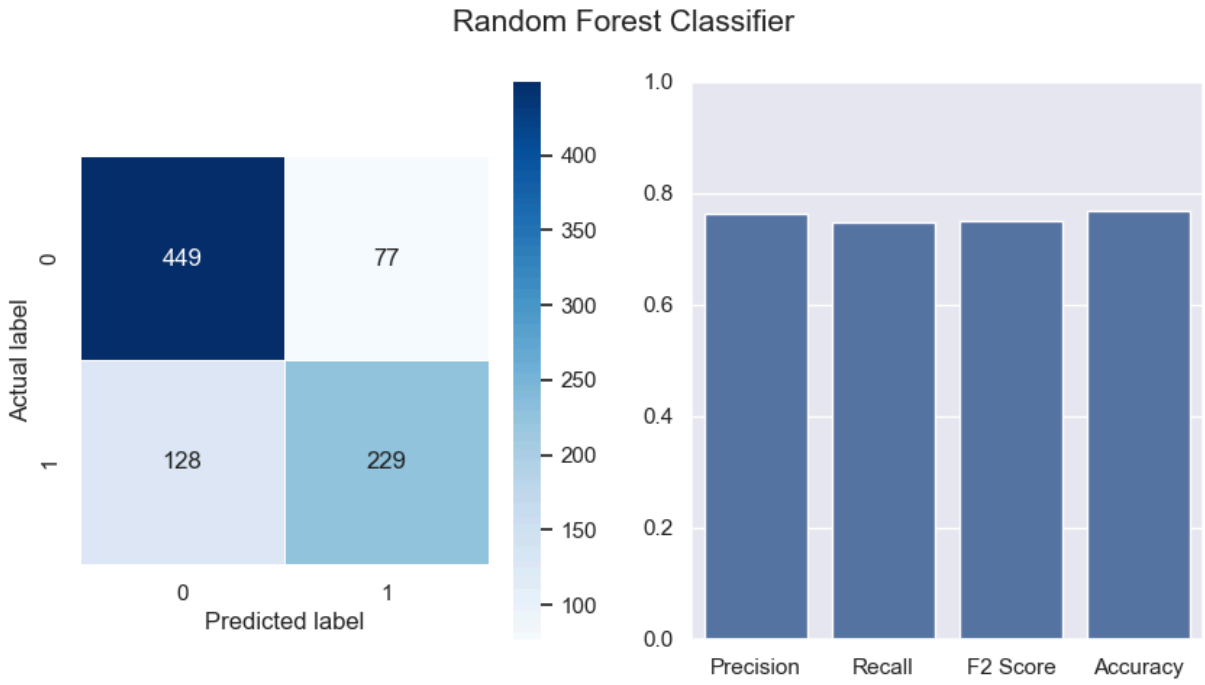
In [62]:
```
# Compute average metrics
svm_precision_opt_avg = np.mean(svm_precision_opt)
svm_recall_opt_avg = np.mean(svm_recall_opt)
svm_f2score_opt_avg = np.mean(svm_f2score_opt)

knn_precision_opt_avg = np.mean(knn_precision_opt)
knn_recall_opt_avg = np.mean(knn_recall_opt)
knn_f2score_opt_avg = np.mean(knn_f2score_opt)

random_forest_precision_opt_avg = np.mean(random_forest_precision_opt)
random_forest_recall_opt_avg = np.mean(random_forest_recall_opt)
random_forest_f2score_opt_avg = np.mean(random_forest_f2score_opt)

# Compute confusion matrices
cm_svm = confusion_matrix(y_test, best_svc.predict(X_test))
cm_knn = confusion_matrix(y_test, best_knn.predict(X_test))
cm_rf = confusion_matrix(y_test, best_rf.predict(X_test))

# Plot confusion matrices and classification metrics
plot_cm(cm_svm, svm_precision_opt_avg, svm_recall_opt_avg, svm_f2score_opt_avg, svm_accuracy_opt, 'Suppo
plot_cm(cm_knn, knn_precision_opt_avg, knn_recall_opt_avg, knn_f2score_opt_avg, knn_accuracy_opt, 'K-Nea
plot_cm(cm_rf, random_forest_precision_opt_avg, random_forest_recall_opt_avg, random_forest_f2score_opt_
```

Support Vector Machine



K-Nearest Neighbor



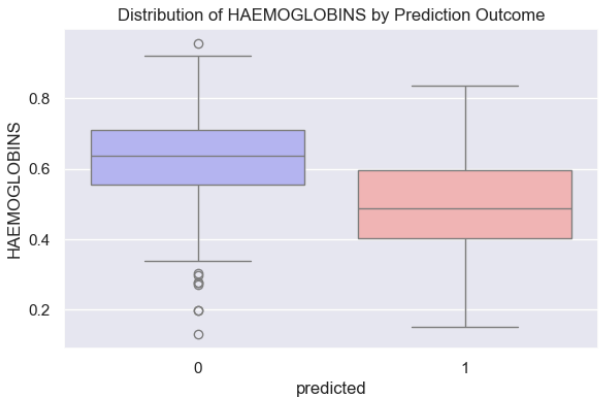


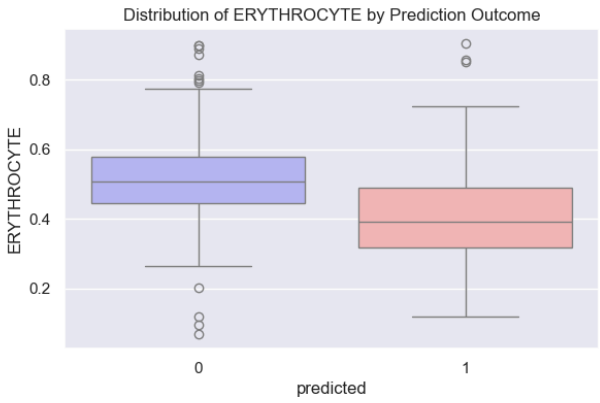
In [63]:
```
y_test_pred = best_svc.predict(X_test)
y_test_pred = pd.DataFrame({'predicted' : y_test_pred})
X_test.reset_index(drop=True, inplace=True)
X_y_test_pred = pd.concat([X_test, y_test_pred], axis=1)
X_y_test_pred.head()
```

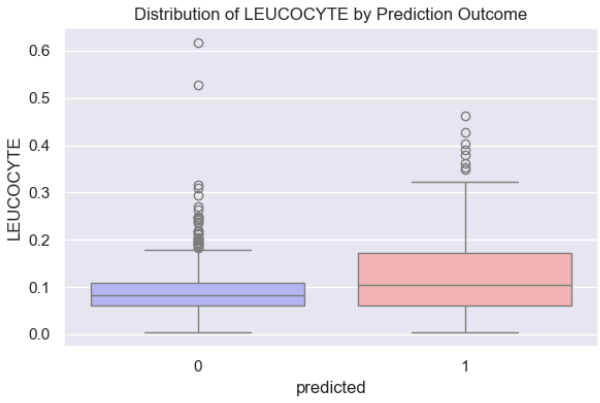
Out[63]:

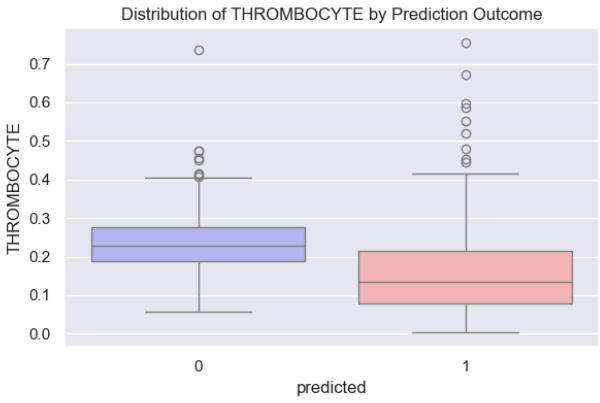
	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCHC	MCV	AGE	SEX	predicted
0	0.754967	0.625392	0.079470	0.200851	0.538462	0.488636	0.489796	1	0
1	0.701987	0.620690	0.033113	0.221277	0.600000	0.394481	0.102041	1	0
2	0.403974	0.297806	0.133775	0.232340	0.638462	0.511364	0.663265	1	1
3	0.245033	0.155172	0.119205	0.050213	0.661538	0.550325	0.765306	1	1
4	0.668874	0.500000	0.066225	0.205106	0.700000	0.500000	0.244898	0	0

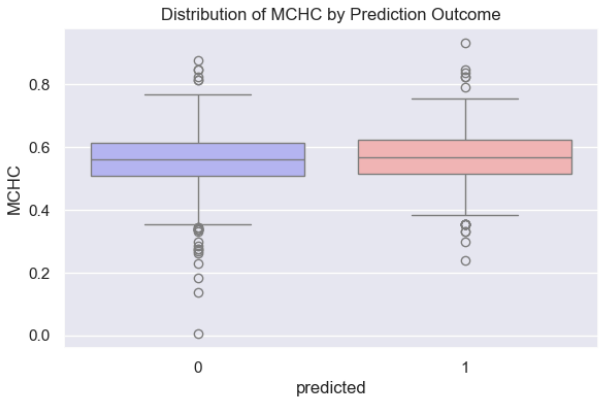
In [64]:
```
# Setting up the figure with subplots
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(12, 16))  # Adjust the figsize to fit your display
axes = axes.flatten()  # Flatten the axes array for easy iteration

# Box Plots for each feature
features = ['HAEMOGLOBINS', 'ERYTHROCYTE', 'LEUCOCYTE', 'THROMBOCYTE', 'MCHC', 'MCV', 'AGE']
for i, feature in enumerate(features):
    sns.boxplot(x='predicted', y=feature, data=X_y_test_pred, palette='bwr', ax=axes[i])
    axes[i].set_title(f'Distribution of {feature} by Prediction Outcome')

# If there's an empty subplot, hide it
if len(axes) > len(features):
    for ax in axes[len(features):]:
        ax.axis('off')

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

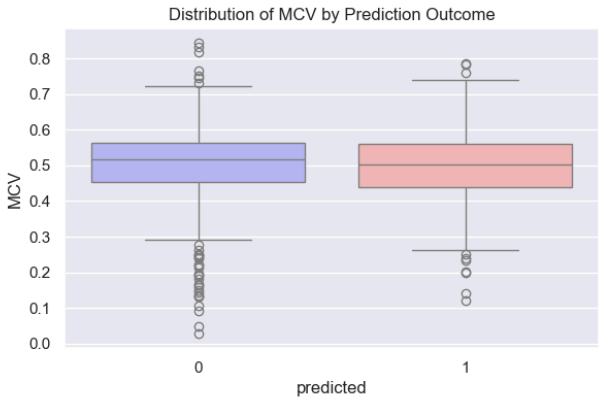


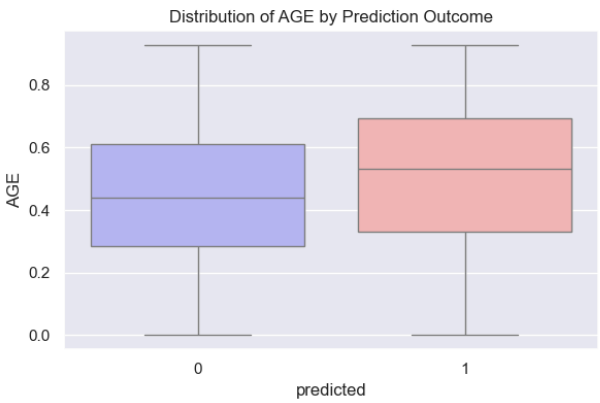












From the plots, we could see from the prediction that:

1. Haemoglobins (Protein containing iron that facilitates the transport of oxygen in red blood cells) and Erythrocyte (Red blood cell): The patients who are admitted as in-patients tend to have lower Haemoglobins and Erythrocyte because of

    - Increased Severity of Underlying Conditions: Low hemoglobin often signals severe or uncontrolled disease states, such as advanced kidney disease or cancers, which require intensive management and monitoring.
    - Complications of Anemia: Severe anemia can lead to significant complications requiring hospitalization, such as heart problems (like heart failure), severe fatigue, and other organ dysfunctions due to poor oxygenation.

    - Acute and Chronic Complications: With fewer red blood cells to transport oxygen, critical organs and tissues do not receive the oxygen they need, which can lead to complications such as organ failure, severe weakness, and in some cases, life-threatening conditions that necessitate immediate hospital care.
2. With the same reasons as above, We could see that the patients who are admitted as in-patients tend to have slightly lower MCHC (Mean Corpuscular Hemoglobin Concentration, Represents the average concentration of hemoglobin within red blood cells) and

MCV (Mean Corpuscular Volume or the average size of red blood cells)

3. Thrombocyte (Blood clotting palette): The patients who are admitted as in-patients tend to have lower Thrombocyte because of

    - Increased Bleeding Risk: The primary function of thrombocytes (platelets) is to help blood clot. A low platelet count can lead to easy or excessive bleeding, which might not stop on its own after an injury or surgery. This condition can manifest as frequent nosebleeds, gum bleeding, heavy menstrual periods, or severe bruising. In some cases, internal bleeding can occur, which can be life-threatening and requires immediate medical attention.
4. However, the patients who are admitted as in-patients tend to have higher Leucocyte (White blood cell) because:

    - Increased Risk of Thrombosis: One of the main risks associated with high thrombocyte counts is an increased likelihood of developing blood clots (thrombosis). These clots can form in arteries or veins and may lead to serious conditions such as deep vein thrombosis, pulmonary embolism, or stroke, depending on where the clot forms. Such events often require immediate hospital care.
    - Underlying Health Conditions: Secondary thrombocytosis can be a response to another condition, such as inflammation, infection, cancer, or anemia. Identifying and managing these underlying conditions often necessitates detailed evaluation and treatment in a hospital setting. Primary thrombocytosis, often due to a myeloproliferative disorder, requires careful monitoring and management to prevent complications.
5. Lastly, older-aged patients have a higher chance of being admitted.

- 1: In-patient
- 0: Out-patient

Interpreting the results:

- In terms of accuracy i.e., the overall number of correct predictions, Random Forest shows a clear superiority with 76.8% followed by KNN with 74.9% and SVM with 74.7%.
- Using the F2 score that bears more weight to the recall which shows the percentange of relevant data points correctly classified by the models, we again see Random Forest clearly win over KNN and SVM. Howevewr, SVM performs slightly better in predicting the out patient data (0.83) while KNN performs better in predicting the in patient data (0.64) between the two.

Discussion

The random forest is a sutiable model to aid in the patient classification task of patients to be admitted to the in patient and out patient units based on the electronic health records porvided. The features chosen after dropping the Haematocrit and MCH values still provided a valuable classification result.

However, there is further scope to improve with potentially a larger collection of patient records as well as further investigation of the dataset using factor analysis or other methods that can help reduce the dimensionality of the feature set.

Visualization