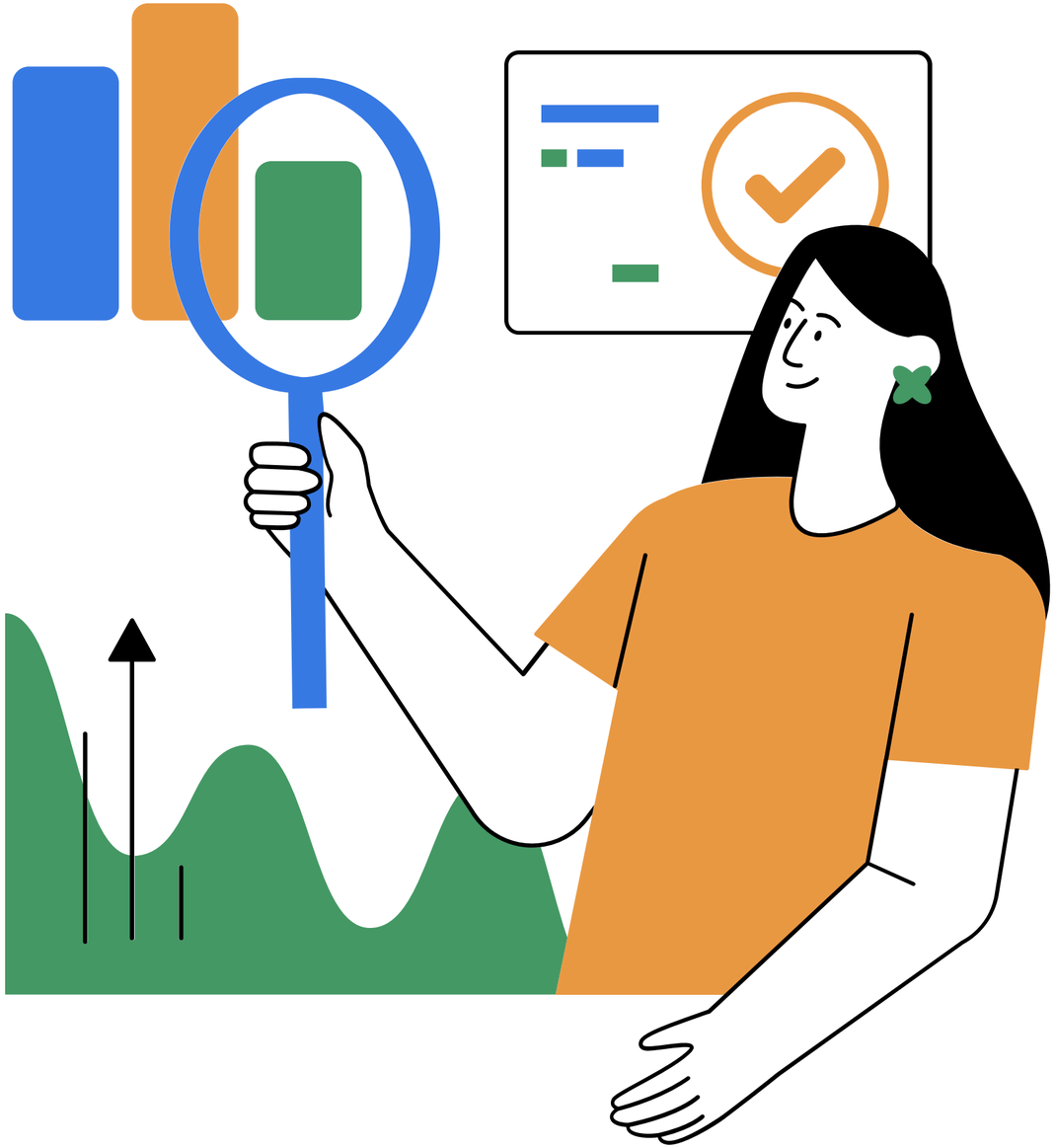
Stroke Risk Prediction Using Machine Learning



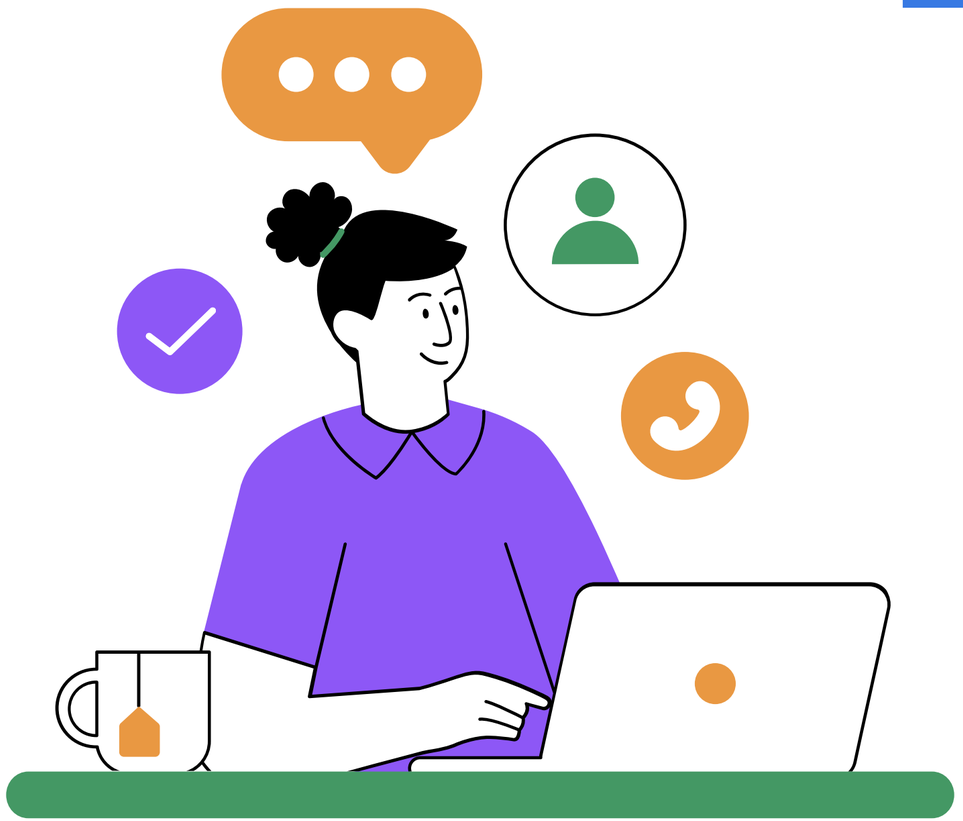
GROUP 06

Introduction

To develop a machine learning model that predicts the likelihood of a person having a stroke based on health and demographic data.

Dataset Source: Kaggle – Stroke Prediction Dataset

Problem Statement



Main Objective

- Build a model to predict whether a person is at risk of stroke

Secondary Goals

- Analyze key health and lifestyle factors that contribute to stroke
- Provide visual insights for medical decision support

Why This Dataset?

- Medical Relevance

  -Stroke is a leading cause of death and disability worldwide

- Diverse Features

  -Includes both numerical and categorical

- Use Case Potential

  -Early stroke risk screening

  -Feature importance for health interventions

Data Exploration

- **Dataset Overview (Summary)**
  Total records: ~5,000 rows

- **Target variable:**
  stroke (0 = No Stroke, 1 = Stroke)
  stroke is highly imbalanced
  ~95% of records are stroke = 0
  ~5% of records are stroke = 1

- **11 features including:**

1. Numerical: age, avg_glucose_level, bmi
2. Categorical: gender, work_type, smoking_status, etc.

Install these libraries

```
In [1]: !pip install pandas numpy matplotlib seaborn scikit-learn
        #use pandas for data handling and analysis
        #numpy for numerical operations and array support
        #matplotlib & seaborn for data visualization
        #scikit-learn for building and evaluating ml models.
```

for data handling and visualization

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

        #use pd,np are common aliases for padas and numpy
        #use plt and sns are used for creating plots and visualizations
```

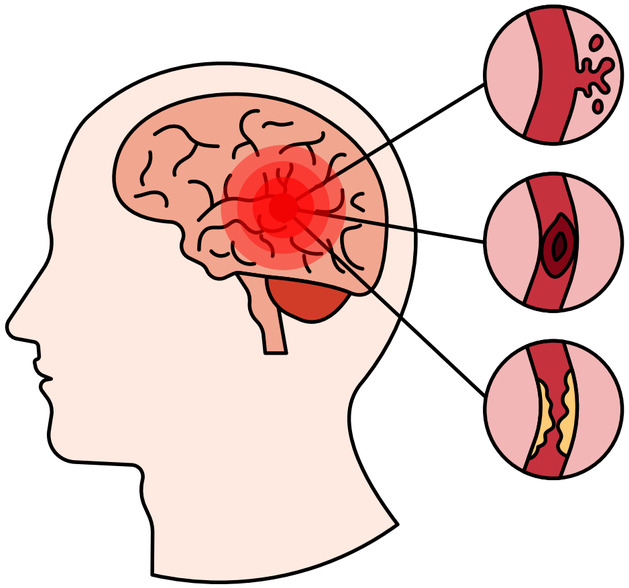
Load Data Set

```
In [3]: df = pd.read_csv('strokedata.csv') #loads the dataset in to pandas DataFrame named df
```

show first five rows

```
In [4]: print(df.head())  #shows the first 5 rows of the dataset.
```

```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

      work_type Residence_type  avg_glucose_level   bmi   smoking_status
0       Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2       Private          Rural             105.92  32.5     never smoked
3       Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```



give the summary of the dataframe

```
In [5]: print(df.info()) #gives the summary of the DataFrame
        #(Useful to identify missing values and understand what type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
None
```

Exploratory Data Analysis

01) Univariate non graphical

Objective: Understand the distribution and summary of single features without visualizations.

```
In [7]: #Univariate_non-graphical_EDA-Gender
        df['gender'].value_counts()
```

```
Out[7]: gender
        Female    2994
        Male      2115
        Other        1
        Name: count, dtype: int64
```

```
In [8]: #Univariate_non-graphical_EDA-Marital Staus
        df['ever_married'].value_counts()
```

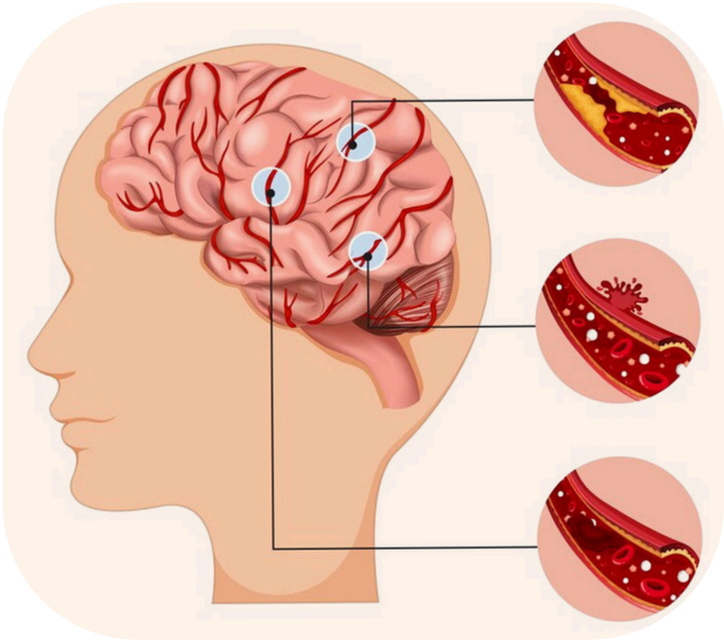
```
Out[8]: ever_married
        Yes    3353
        No     1757
        Name: count, dtype: int64
```

```
In [9]: #Univariate_non-graphical_EDA-Work Type
        df['work_type'].value_counts()
```

```
Out[9]: work_type
        Private          2925
        Self-employed     819
        children          687
        Govt_job          657
        Never_worked       22
        Name: count, dtype: int64
```

```
In [10]: #Univariate_non-graphical_EDA-Stroke
         df['stroke'].value_counts() #help to understand how age is distributed and see if it's skewed or normally distributed
```

```
Out[10]: stroke
         0    4861
         1     249
         Name: count, dtype: int64
```



Descriptive Statistics for numerical columns in the dataset

df.describe() → Summary stats of age, bmi, glucose

```
In [6]: df.describe() #Provides descriptive statistics for numerical columns in the dataset
```

Out[6]:

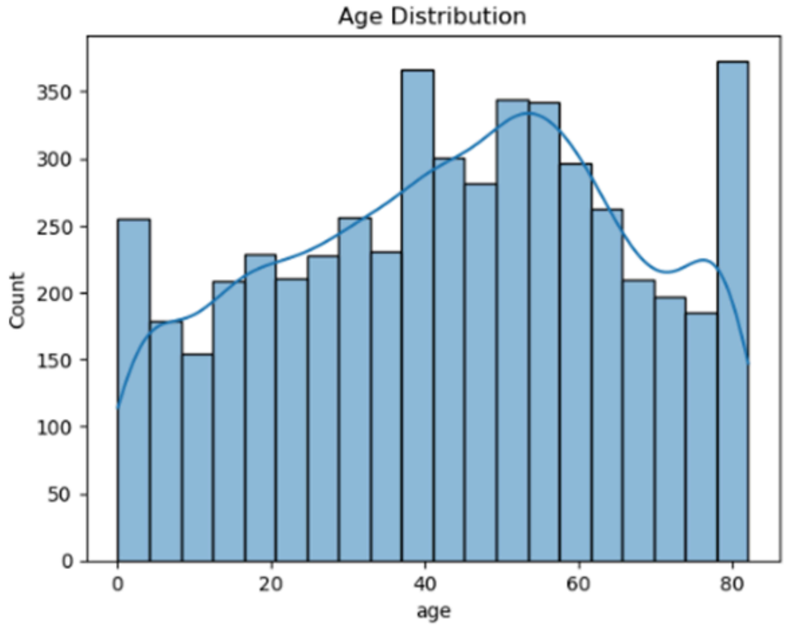
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

02) Univariate non graphical

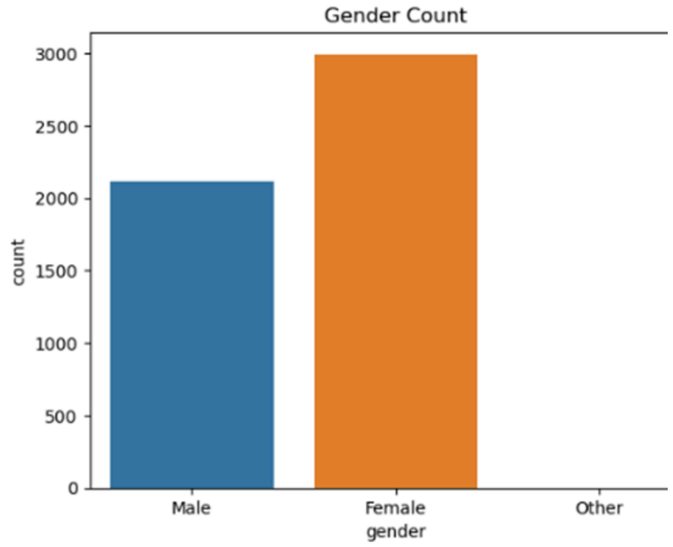
Objective: Visualize individual feature distributions to detect skewness or outliers.

Ex- Histogram of age, Countplot of stroke, gender

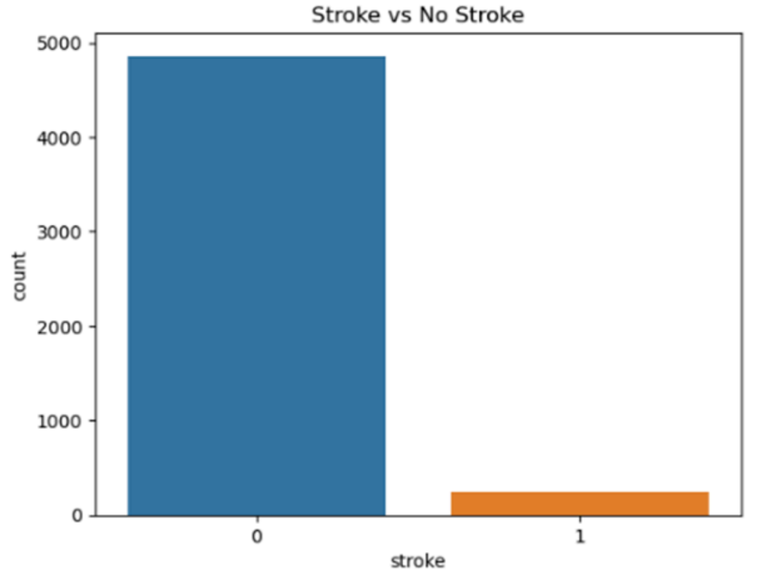
```
#UNIVARIATE_GRAPHICAL_EDA-Age Distribution
sns.histplot(df['age'], kde=True)
plt.title("Age Distribution")
plt.show()
```




```
12]: # Count plot for 'gender'
sns.countplot(x='gender', data=df)
plt.title("Gender Count")
plt.show()
```




```
[13]: # Count plot for 'stroke'
sns.countplot(x='stroke', data=df)
plt.title("Stroke vs No Stroke")
plt.show()
```



03) Multivariate non graphical EDA

- Objective: Analyze relationships between multiple features using data tables or statistics.
- Examples:

  pd.crosstab(df['hypertension'], df['stroke'])

  df.corr() → Correlation matrix of numerical features
- Observation: age and glucose show some correlation with stroke; gender has little effect.

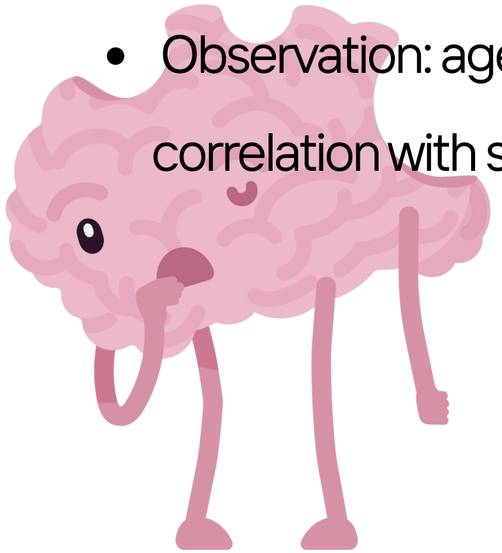
03) Multivariate non graphical EDA

- Objective: Analyze relationships between multiple features using data tables or statistics.
- Examples:

  pd.crosstab(df['hypertension'], df['stroke'])

  df.corr() → Correlation matrix of numerical features

- Observation: age and glucose show some correlation with stroke; gender has little effect.



In [14]:
```
#MULTIVARIATE NON-GRAPHICAL EDA
#Crosstab: stroke vs smoking_status
pd.crosstab(df['stroke'], df['smoking_status'], normalize='index')
```

Out[14]:

stroke	Unknown	formerly smoked	never smoked	smokes
0	0.307961	0.167661	0.370706	0.153672
1	0.188755	0.281124	0.361446	0.168675

In [15]:
```
# Correlation matrix for numerical features
# Computes Pearson correlation coefficients between numeric features
df[['age', 'bmi', 'avg_glucose_level', 'stroke']].corr()
```

Out[15]:

	age	bmi	avg_glucose_level	stroke
age	1.000000	0.333398	0.238171	0.245257
bmi	0.333398	1.000000	0.175502	0.042374
avg_glucose_level	0.238171	0.175502	1.000000	0.131945
stroke	0.245257	0.042374	0.131945	1.000000

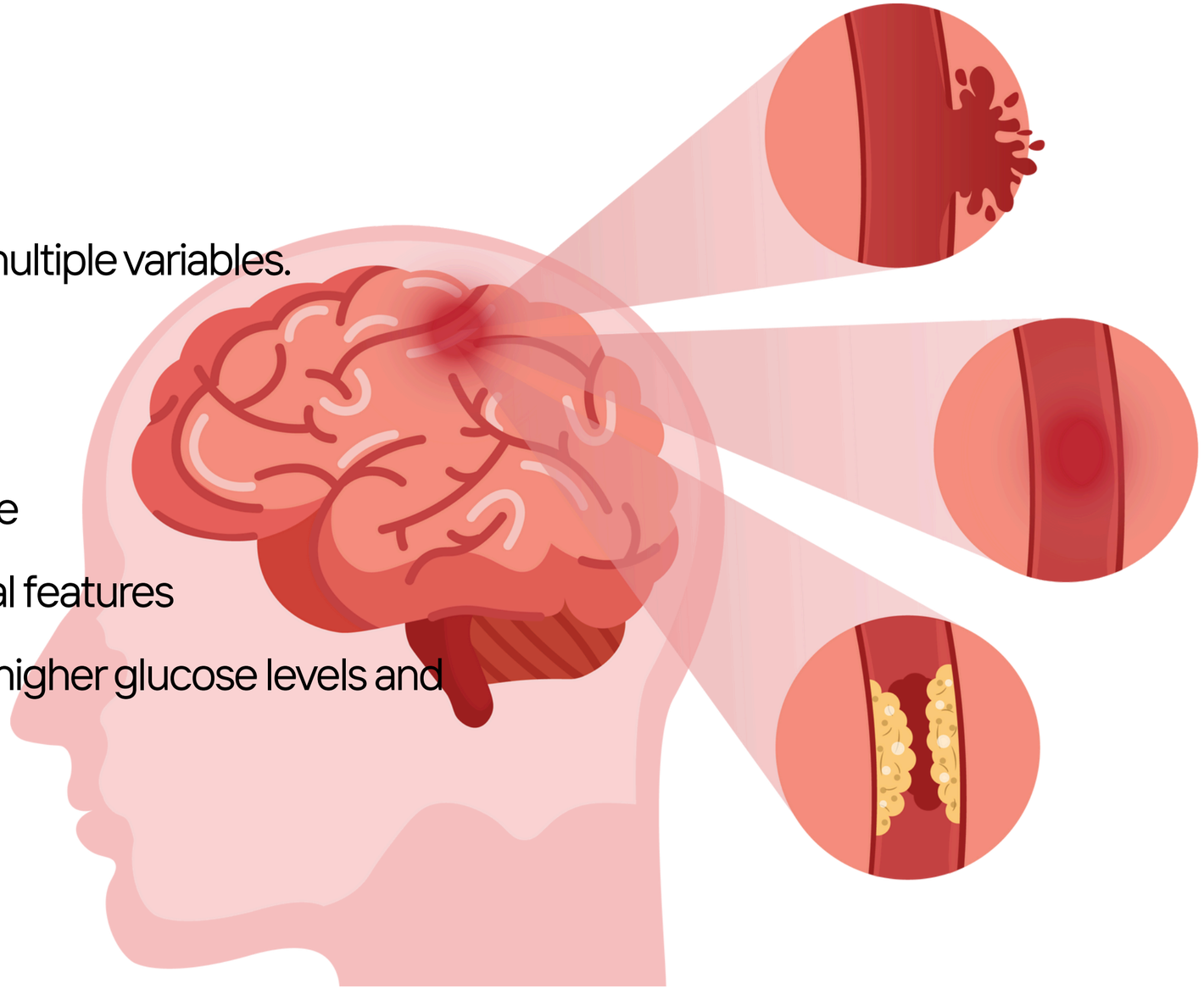
04) Multivariate Graphical EDA

- Objective: Visualize interactions between multiple variables.
- Examples (with visuals):

  Boxplot: age vs stroke

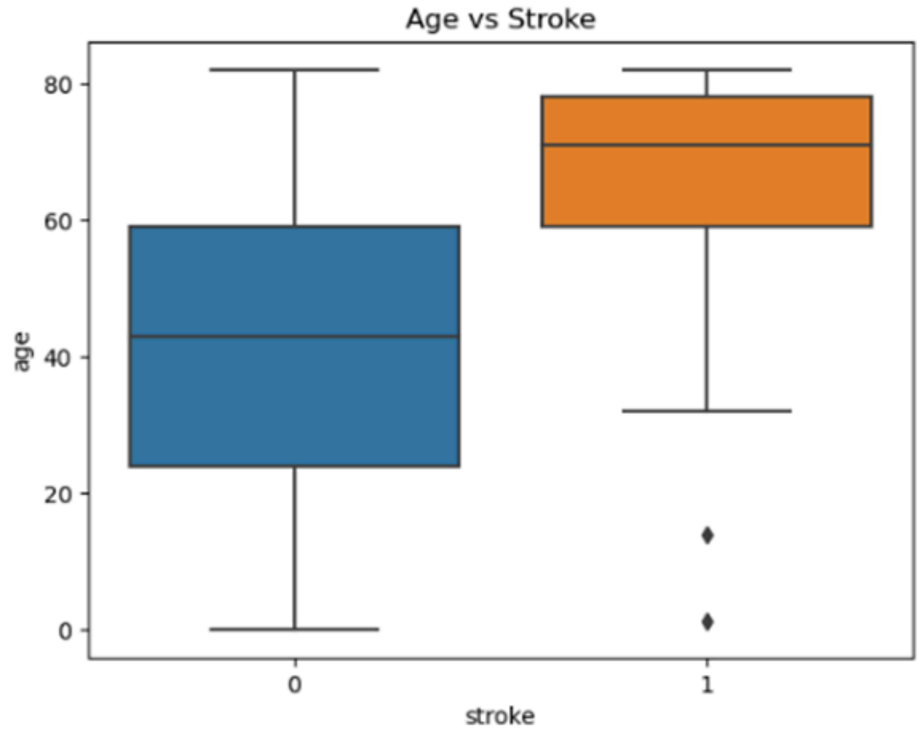
  Pairplot: age, bmi, glucose level and stroke

  Heatmap: Correlation between numerical features
- Observation: Stroke patients tend to have higher glucose levels and are mostly married or older.



Boxplot

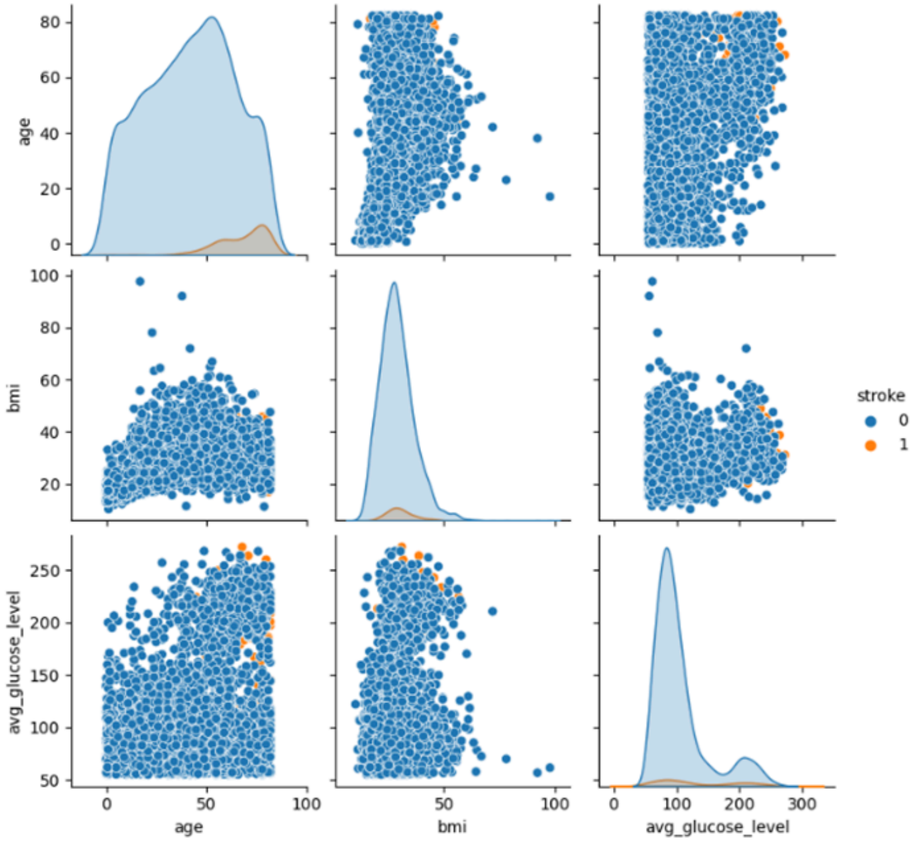
```
In [17]: #  Boxplot: age vs stroke
sns.boxplot(x='stroke', y='age', data=df)
plt.title("Age vs Stroke")
plt.show()
```



Pairplot

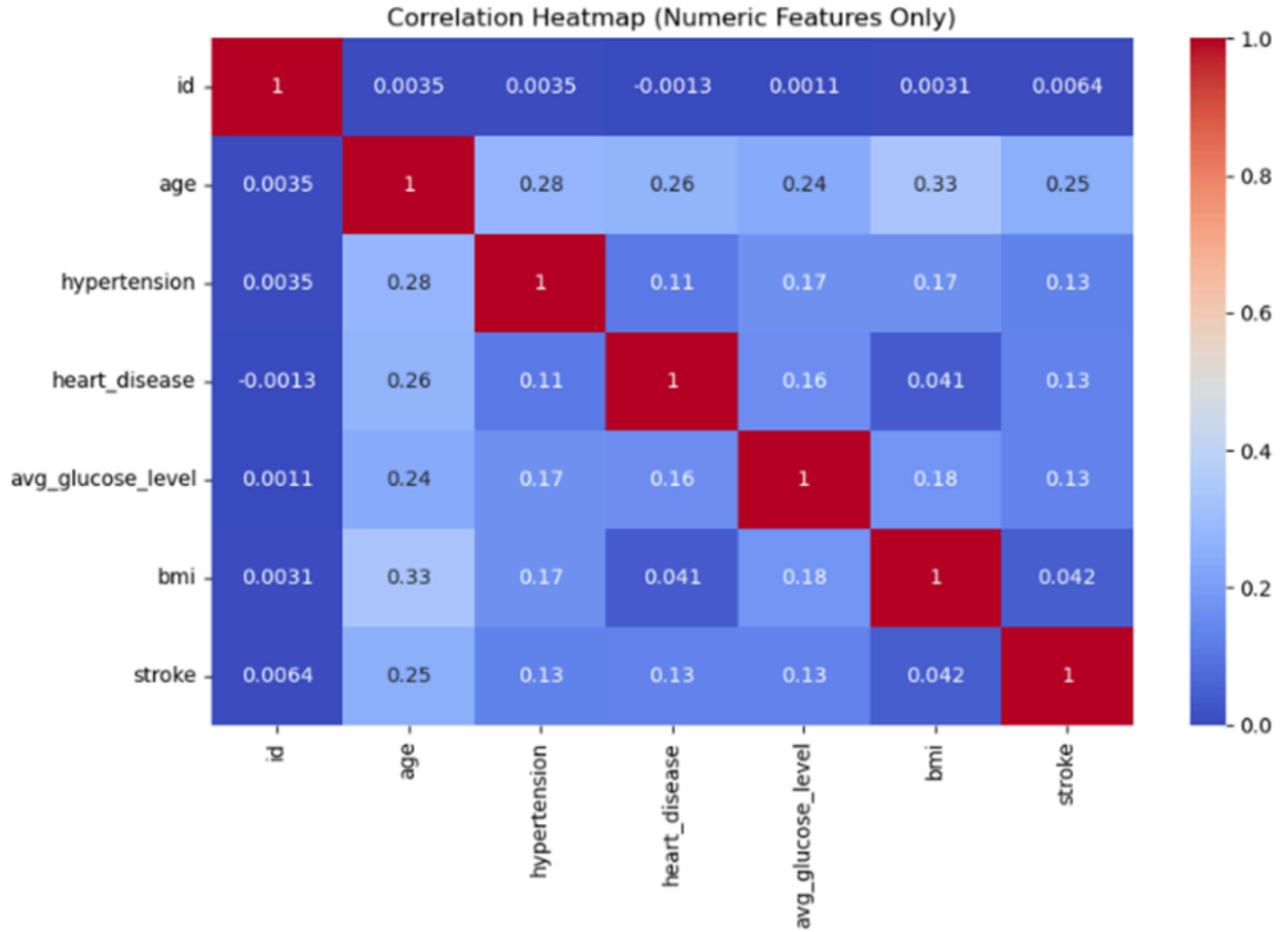
```
In [18]: #  Pairplot for selected features
sns.pairplot(df[['age', 'bmi', 'avg_glucose_level', 'stroke']], hue='stroke')
plt.show()
```

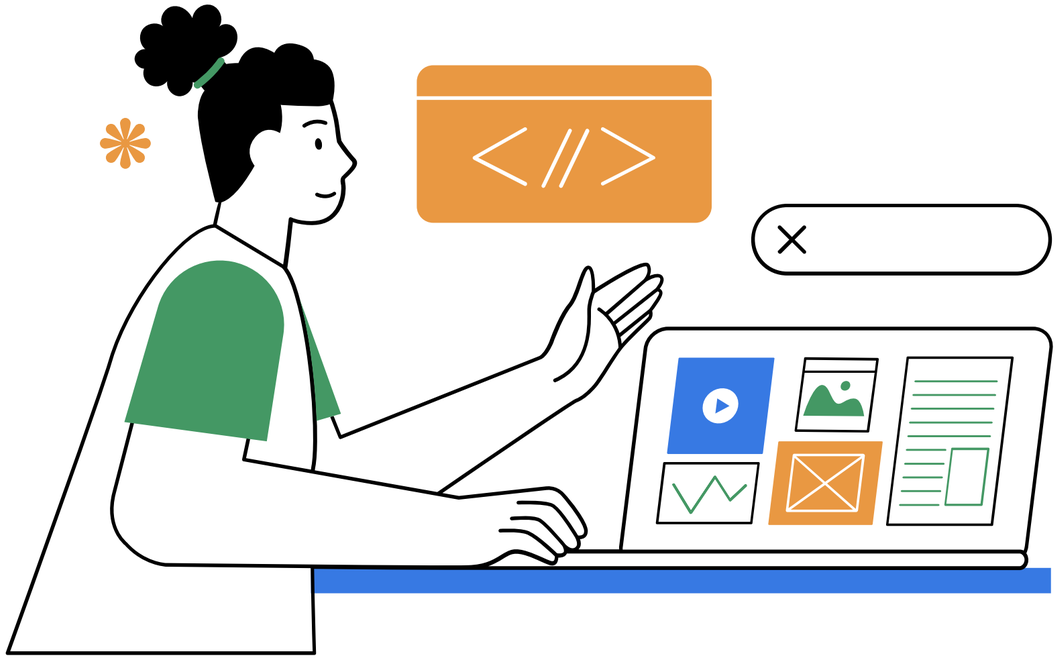
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Heatmap

In [16]:
```
#  MULTIVARIATE GRAPHICAL EDA
#Select only numeric columns
numeric_df = df.select_dtypes(include=['int64', 'float64'])
#  Heatmap of correlations (only numeric columns)
plt.figure(figsize=(10,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap (Numeric Features Only)")
plt.show()
```





Data Preprocessing

1. **Missing Value Handling**: Filled missing BMI values with mean
2. **Encoding Categorical Features**: Label encoding for binary categorical features(gender, ever_married, Residence_type), One-hot encoding for multi-class categorical variables (work_type, smoking_status)
3. **Feature Scaling**: StandardScaler used on numerical features (age, BMI, glucose_level)
4. **Train-Test Split**: 80% training, 20% testing

Handling Missing Values

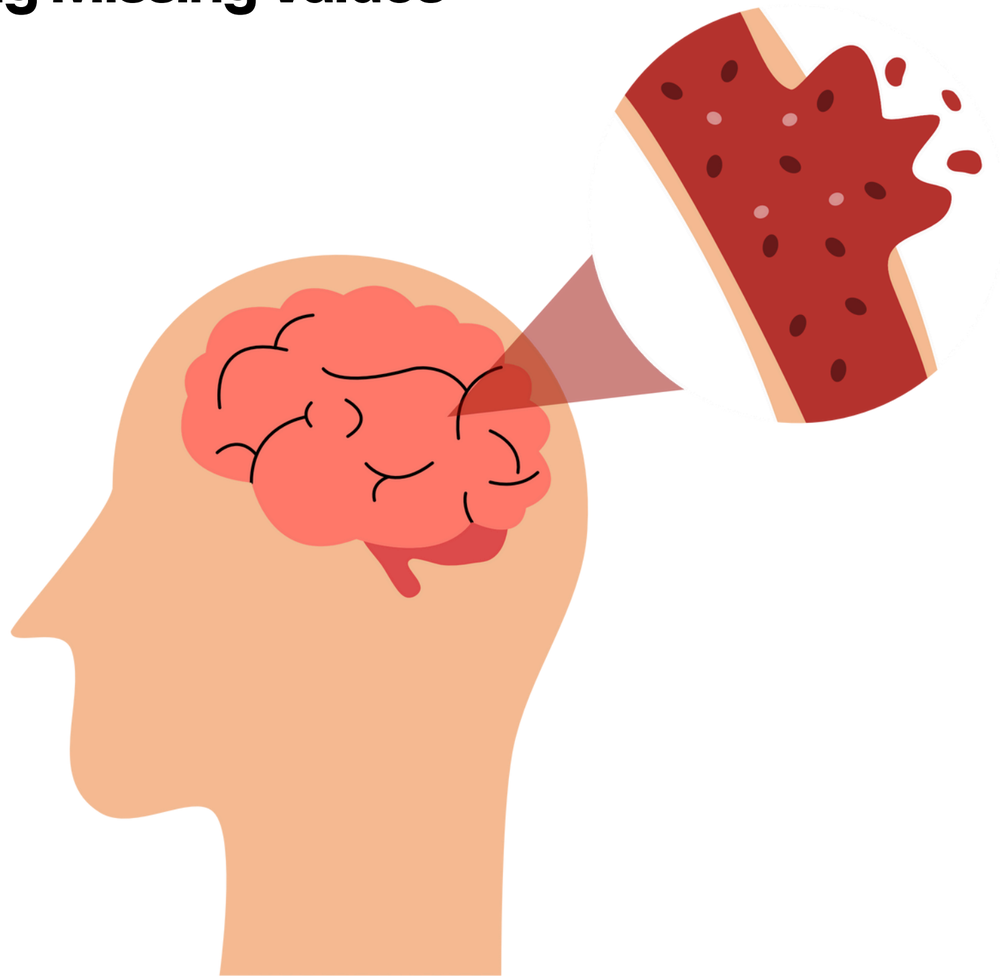
```
In [19]: #DATA PREPROCESSING
         #handling missing data(1)
         df.isnull().sum()
```

```
Out[19]: id                   0
         gender               0
         age                  0
         hypertension         0
         heart_disease        0
         ever_married         0
         work_type            0
         Residence_type       0
         avg_glucose_level    0
         bmi                201
         smoking_status       0
         stroke               0
         dtype: int64
```

```
In [20]: #  Fill missing BMI with mean
         df['bmi'].fillna(df['bmi'].mean(), inplace=True)
```

```
In [21]: df.isnull().sum()
```

```
Out[21]: id                   0
         gender               0
         age                  0
         hypertension         0
         heart_disease        0
         ever_married         0
         work_type            0
         Residence_type       0
         avg_glucose_level    0
         bmi                  0
         smoking_status       0
         stroke               0
         dtype: int64
```



Encoding Categorical Features

In [22]:
```
#Encode Categorical Variables(2)
from sklearn.preprocessing import LabelEncoder
```

In [23]:
```
#  Label Encoding for binary categorical columns
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])        # Male=1, Female=0 (or other depending on data)
df['ever_married'] = le.fit_transform(df['ever_married'])
df['Residence_type'] = le.fit_transform(df['Residence_type'])
```

In [24]:
```
#  One-hot encoding for other categorical features
df = pd.get_dummies(df, columns=['work_type', 'smoking_status'], drop_first=True)
```

Feature Scaling

In [25]:
```
#Scale Numeric Features(3)
from sklearn.preprocessing import StandardScaler
```

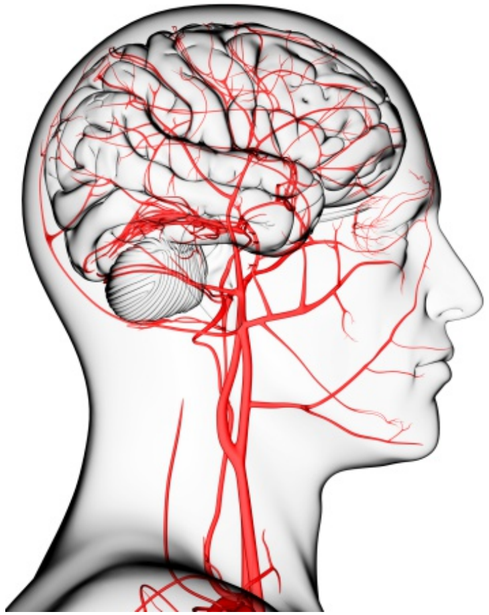
In [26]:
```
scaler = StandardScaler()
df[['age', 'avg_glucose_level', 'bmi']] = scaler.fit_transform(df[['age', 'avg_glucose_level', 'bmi']])
```

Train Test Splitting

In [27]:
```
#Split the data(4)
from sklearn.model_selection import train_test_split
```

In [28]:
```
# Separate features and label
X = df.drop('stroke', axis=1)
y = df['stroke']
```

In [29]:
```
#  Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```



Model Development

We selected five classification models for stroke prediction:

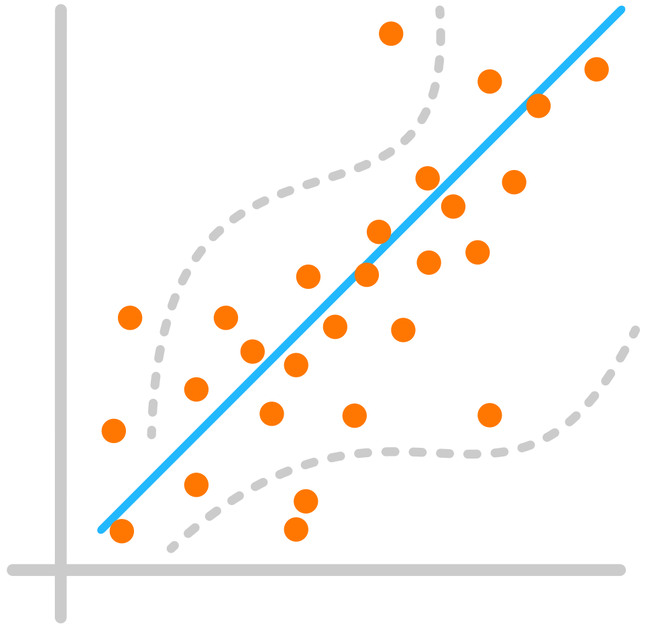
1. **Logistic Regression** – Simple, interpretable baseline
2. **Random Forest** – Robust tree-based model, handles imbalance
3. **XGBoost** – Gradient boosting, high performance
4. **K-Nearest Neighbors (KNN)** – Distance-based, requires scaling
5. **Support Vector Machine (SVM)** – Works well with small, high-dimensional data

- Each model offers different strengths in terms of bias, variance, and interpretability

Tools & Libraries Used

- pandas, numpy, seaborn, matplotlib – data handling and visualization
- scikit-learn – model training and evaluation
- xgboost – gradient boosting
- joblib – model saving

Logistic Regression



In [30]:
```
#MODEL FITTING-LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score
```

In [31]:
```
# Initialize and train the model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
```

Out[31]:
```
LogisticRegression(max_iter=1000)
```

In [32]:
```
# Predict on test set
y_pred_lr = log_reg.predict(X_test)
y_prob_lr = log_reg.predict_proba(X_test)[:, 1]
```

In [33]:
```
#EVALUATION - LOGISTIC REGRESSION
#  Classification Report
print("Logistic Regression Report:")
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       972
           1       0.00      0.00      0.00        50

    accuracy                           0.95      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.90      0.95      0.93      1022
```

In [34]:
```
# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))
```

```
Confusion Matrix:
 [[970   2]
 [ 50   0]]
```

In [35]:
```
#  Accuracy & ROC-AUC
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_lr))
```

```
Accuracy: 0.949119373776908
ROC-AUC Score: 0.7837242798353911
```

Random Forest

In [36]:
```
#MODEL FITTING - RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
```

In [37]:
```
#  Initialize and train
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Out[37]:
```
RandomForestClassifier(random_state=42)
```

In [38]:
```
#  Predict
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]
```

In [39]:
```
#EVALUATION - RANDOM FOREST
print("Random Forest Report:")
print(classification_report(y_test, y_pred_rf))
```

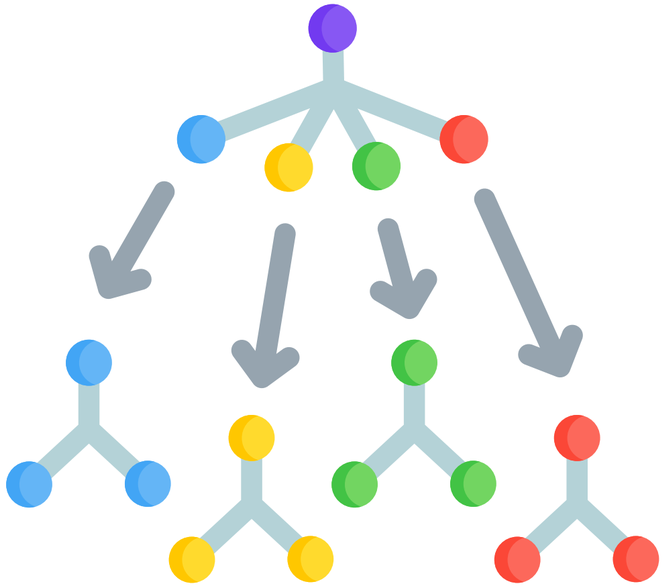
```
Random Forest Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       972
           1       1.00      0.02      0.04        50

    accuracy                           0.95      1022
   macro avg       0.98      0.51      0.51      1022
weighted avg       0.95      0.95      0.93      1022
```

In [40]:
```
# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

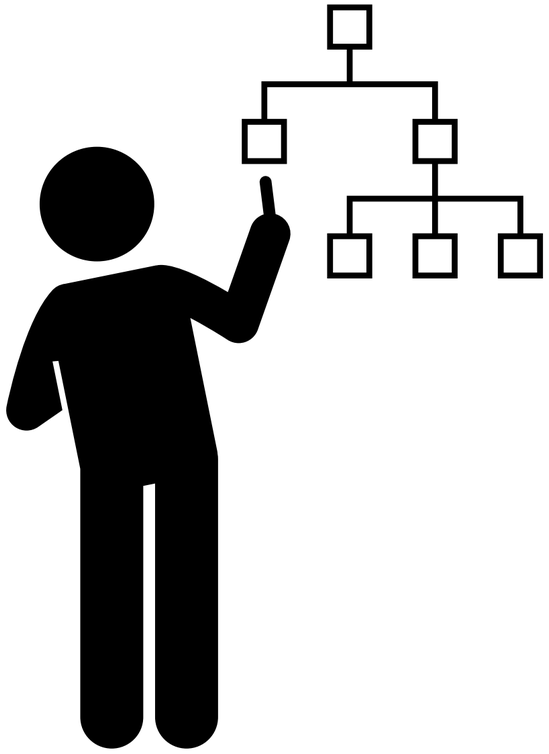
```
Confusion Matrix:
 [[972   0]
 [ 49   1]]
```

In [41]:
```
# Accuracy & ROC-AUC
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_rf))
```

```
Accuracy: 0.952054794520548
ROC-AUC Score: 0.8100205761316872
```



XGBoost Classifier



In [42]:
```
#MODEL FITTING-XGBoost Classifier
from xgboost import XGBClassifier
```

In [43]:
```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)
```

```
C:\Users\NETHMI APSARA\AppData\Roaming\Python\Python311\site-packages\xgboost\training.py:183:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Out[43]:

```
XGBClassifier(colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, ...)
```

In [44]:
```
y_pred_xgb = xgb.predict(X_test)
y_prob_xgb = xgb.predict_proba(X_test)[:, 1]
```

In [45]:
```
print("XGBoost Classifier Report:")
print(classification_report(y_test, y_pred_xgb))
```

```
XGBoost Classifier Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       972
           1       0.31      0.08      0.13        50

    accuracy                           0.95      1022
   macro avg       0.63      0.54      0.55      1022
weighted avg       0.92      0.95      0.93      1022
```

In [46]:
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))
```

```
Confusion Matrix:
 [[963   9]
 [ 46   4]]
```

In [47]:
```
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_xgb))
```

```
Accuracy: 0.9461839530332681
ROC-AUC Score: 0.8104732510288065
```

K-Nearest Neighbors

In [48]:
```
#MODEL FITTING-KNN
from sklearn.neighbors import KNeighborsClassifier
```

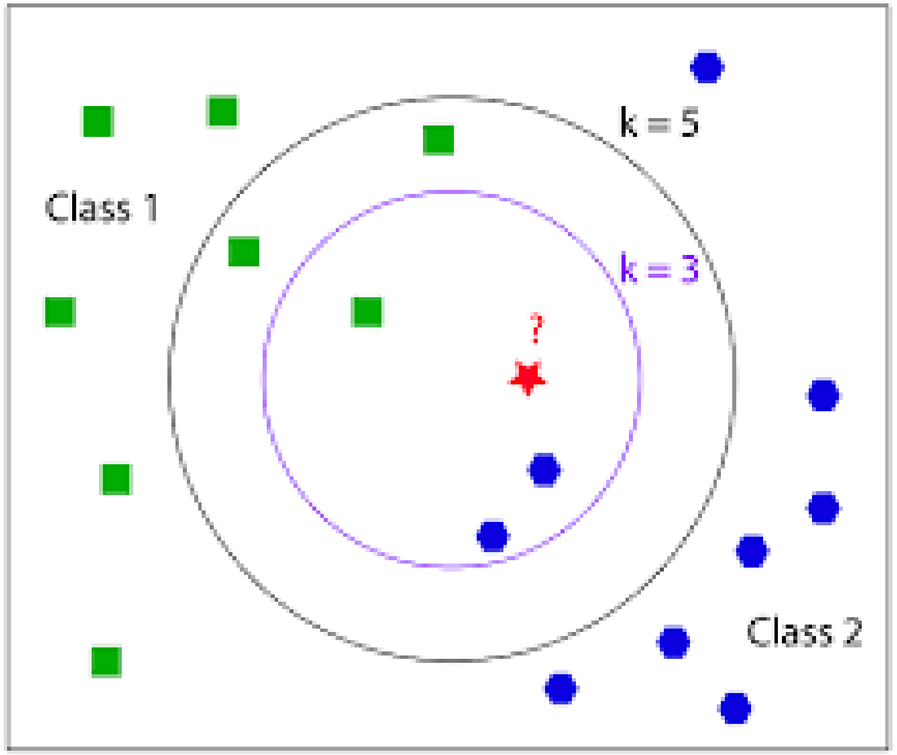
In [49]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[49]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

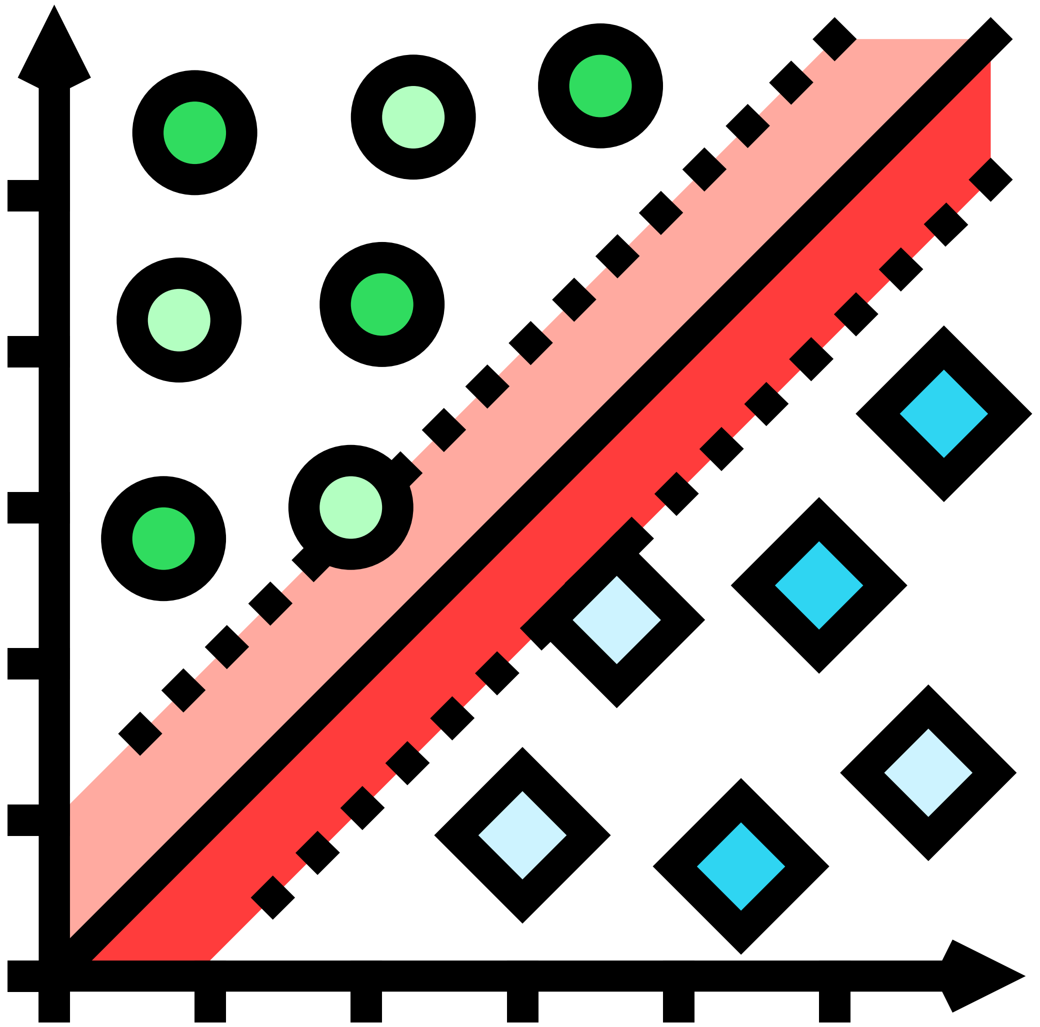
In [50]:
```
import numpy as np

# Ensure X_test is a NumPy array with correct dtype
X_test_knn = np.array(X_test).astype(np.float64)

# Predict
y_pred_knn = knn.predict(X_test_knn)
y_prob_knn = knn.predict_proba(X_test_knn)[:, 1]

# Evaluation
print("KNN Classifier Report:")
print(classification_report(y_test, y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_knn))
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X doe
sClassifier was fitted with feature names
  warnings.warn(
```

```
KNN Classifier Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       972
           1       0.00      0.00      0.00        50

    accuracy                           0.95      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.90      0.95      0.93      1022

Confusion Matrix:
 [[972   0]
 [ 50   0]]
Accuracy: 0.9510763209393346
ROC-AUC Score: 0.5155761316872428
```



Support Vector Machine



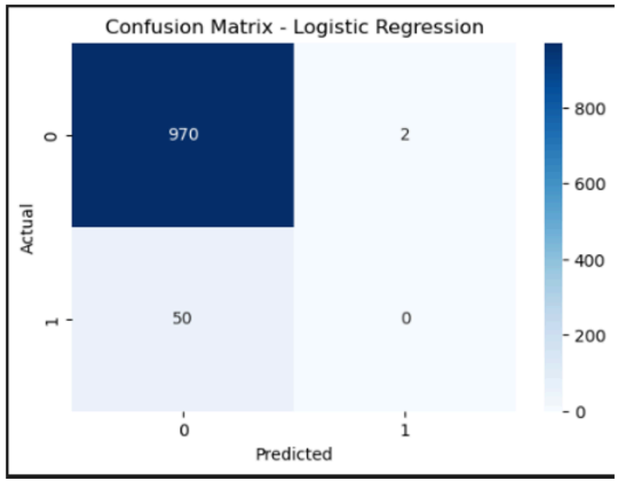
```
In [51]: from sklearn.svm import SVC

In [52]: svm = SVC(probability=True)  # Set to True for predict_proba
         svm.fit(X_train, y_train)

Out[52]: ▾ SVC
         SVC(probability=True)

In [53]: y_pred_svm = svm.predict(X_test)
         y_prob_svm = svm.predict_proba(X_test)[:, 1]

In [54]: print("SVM Classifier Report:")
         print(classification_report(y_test, y_pred_svm))

         SVM Classifier Report:
                       precision    recall  f1-score   support

                    0       0.95      1.00      0.97       972
                    1       0.00      0.00      0.00        50

             accuracy                           0.95      1022
            macro avg       0.48      0.50      0.49      1022
         weighted avg       0.90      0.95      0.93      1022

         C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: Undefine
         re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
         vior.
           _warn_prf(average, modifier, msg_start, len(result))
         C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: Undefine
         re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
         vior.
           _warn_prf(average, modifier, msg_start, len(result))
         C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: Undefine
         re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
         vior.
           _warn_prf(average, modifier, msg_start, len(result))

In [55]: print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))

         Confusion Matrix:
          [[972   0]
          [ 50   0]]

In [56]: print("Accuracy:", accuracy_score(y_test, y_pred_svm))
         print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_svm))

         Accuracy: 0.9510763209393346
         ROC-AUC Score: 0.5088065843621399
```

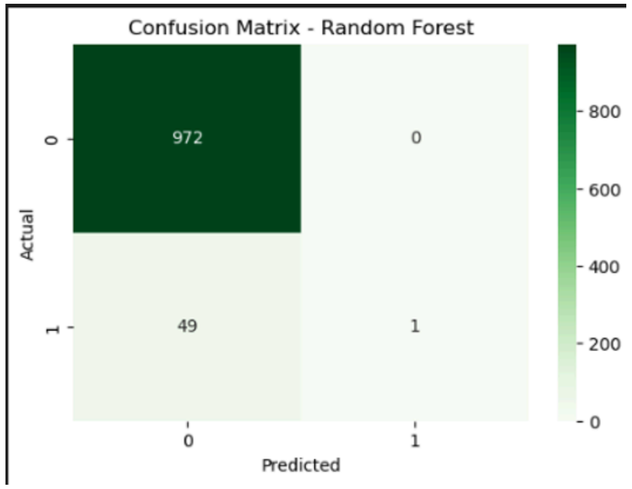
COMPARISON

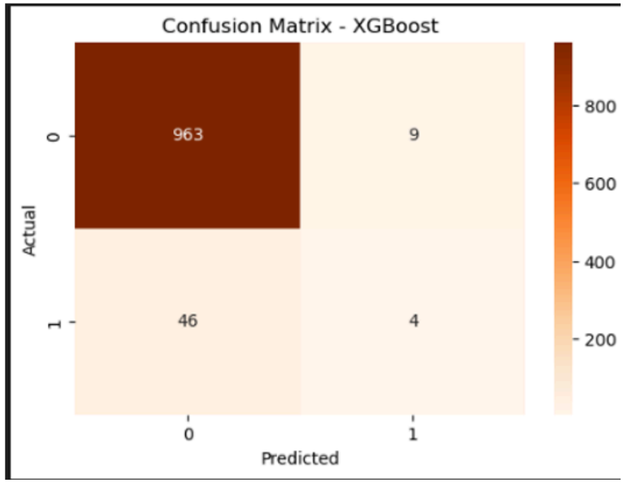
In [57]:
```python
# Model names
models = ['Logistic Regression', 'Random Forest', 'XGBoost', 'KNN', 'SVM']

# Accuracy scores
accuracies = [
    accuracy_score(y_test, y_pred_lr),
    accuracy_score(y_test, y_pred_rf),
    accuracy_score(y_test, y_pred_xgb),
    accuracy_score(y_test, y_pred_knn),
    accuracy_score(y_test, y_pred_svm)
]

# ROC-AUC scores
roc_aucs = [
    roc_auc_score(y_test, y_prob_lr),
    roc_auc_score(y_test, y_prob_rf),
    roc_auc_score(y_test, y_prob_xgb),
    roc_auc_score(y_test, y_prob_knn),
    roc_auc_score(y_test, y_prob_svm)
]

# Print comparison
for i in range(len(models)):
    print(f"{models[i]} - Accuracy: {accuracies[i]:.4f} | ROC-AUC: {roc_aucs[i]:.4f}")
```

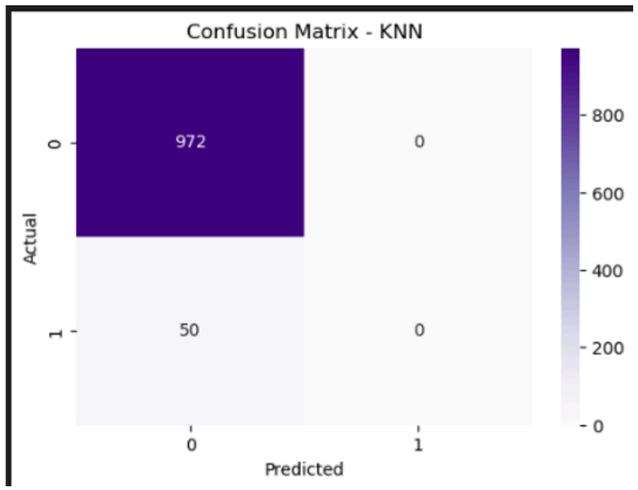
```
Logistic Regression - Accuracy: 0.9491 | ROC-AUC: 0.7837
Random Forest - Accuracy: 0.9521 | ROC-AUC: 0.8100
XGBoost - Accuracy: 0.9462 | ROC-AUC: 0.8105
KNN - Accuracy: 0.9511 | ROC-AUC: 0.5156
SVM - Accuracy: 0.9511 | ROC-AUC: 0.5088
```

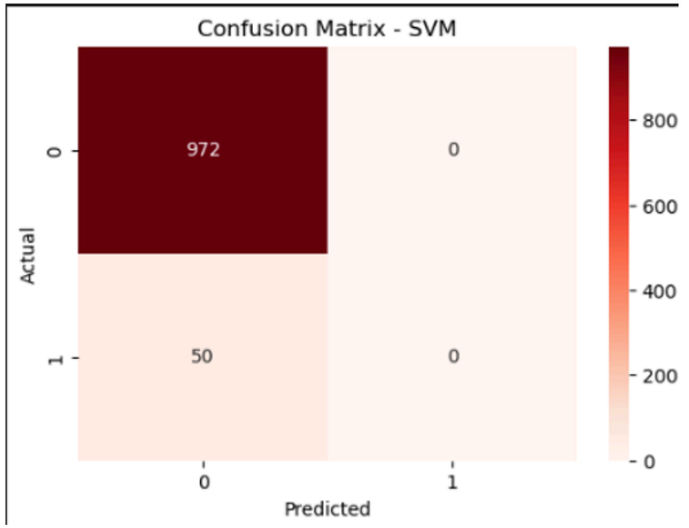
Comparison using Confusion Matrix

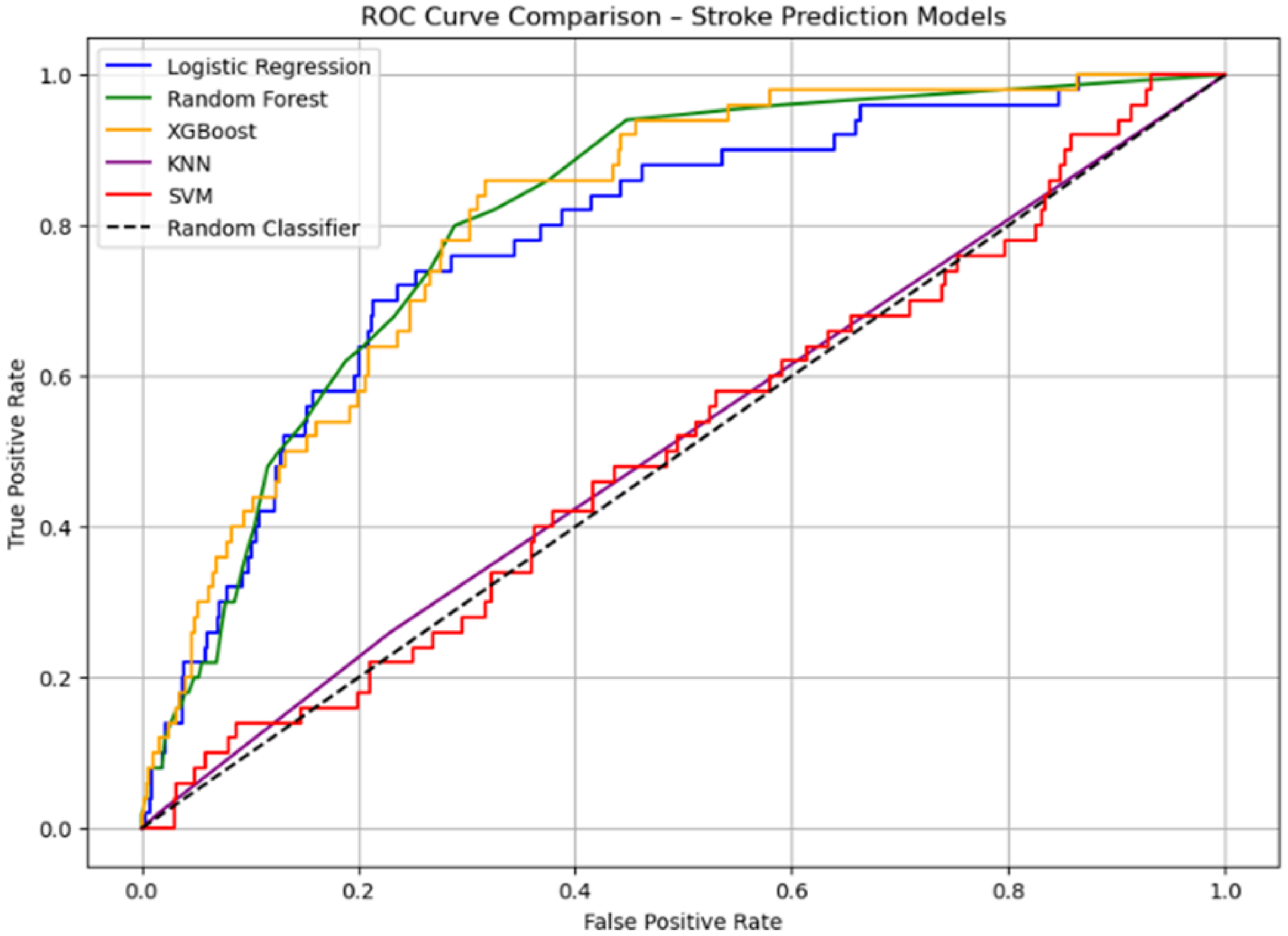


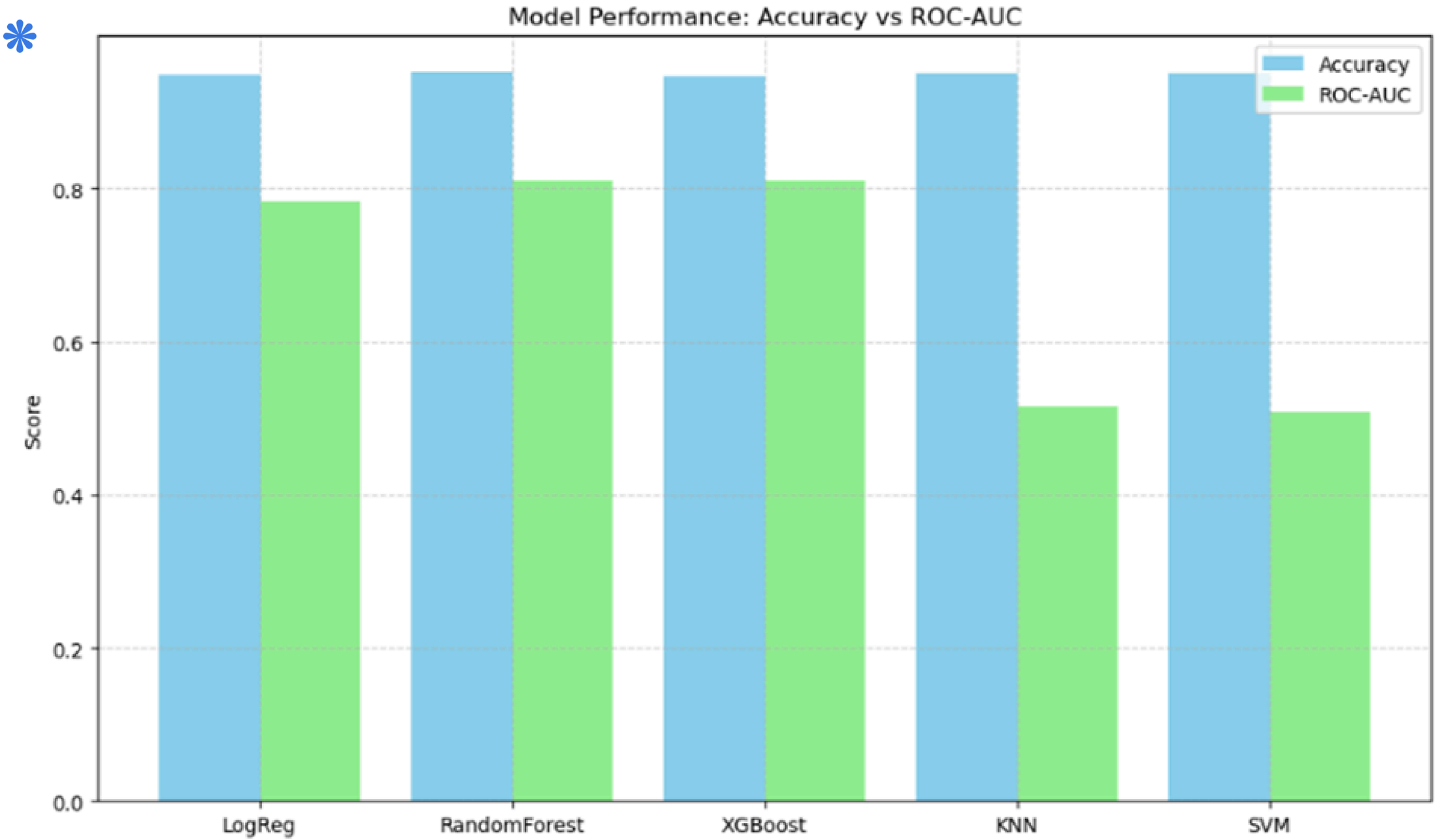









ROC Curve Comparison – Stroke Prediction Models
Logistic Regression
Random Forest
XGBoost
KNN
SVM
Random Classifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

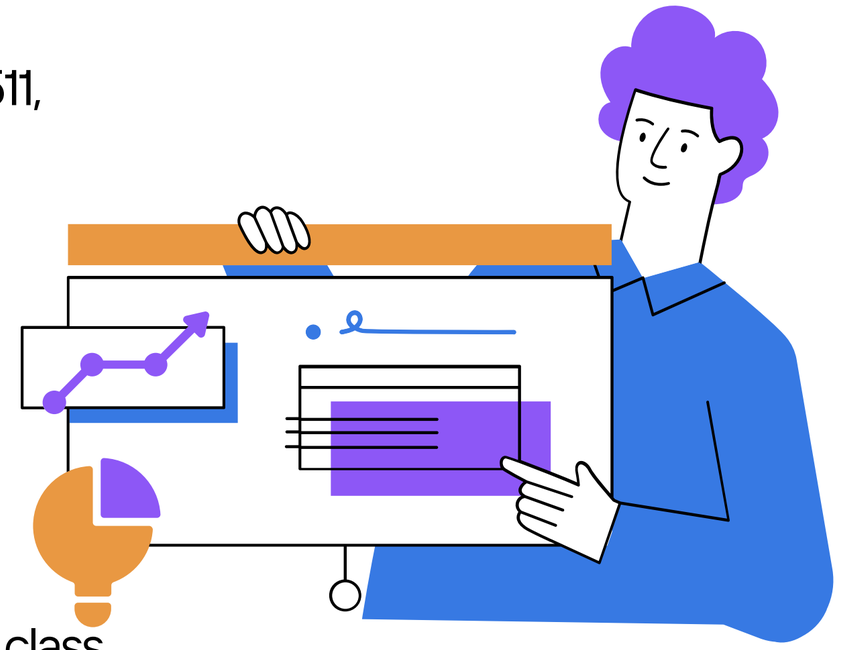

Model Performance: Accuracy vs ROC-AUC
Accuracy
ROC-AUC
Score
0.0
0.2
0.4
0.6
0.8
LogReg
RandomForest
XGBoost
KNN
SVM

Key Findings & Insights

Based on the provided results, the best-fit model can be determined by evaluating both accuracy and ROC-AUC (Receiver Operating Characteristic - Area Under Curve) scores, as these metrics provide insights into the model's predictive performance.

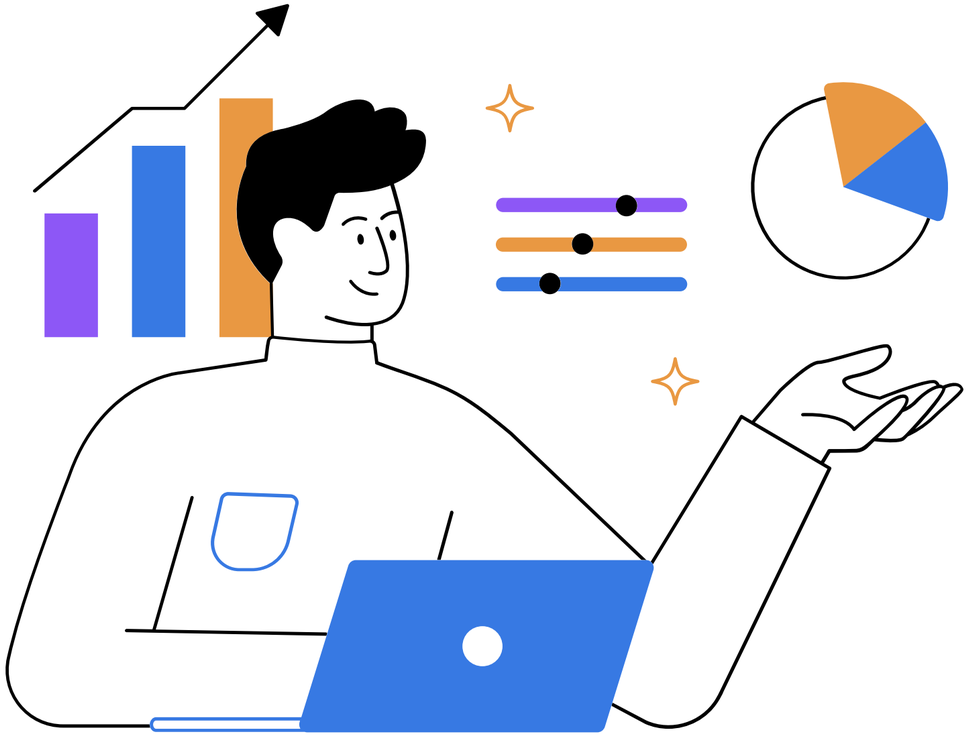
Analysis:

1. Accuracy:
   - Random Forest and SVM achieve the highest accuracy (0.9521 and 0.9511, respectively).
   - However, accuracy alone is not sufficient for evaluation, especially in imbalanced datasets.
2. ROC-AUC:
   - ROC-AUC measures the model's ability to distinguish between classes.
   - XGBoost has the highest ROC-AUC score (0.8105), followed closely by Random Forest (0.8100).
   - KNN and SVM have significantly lower ROC-AUC scores, indicating poor class separation despite high accuracy.



Conclusion

The best-fit model is XGBoost, as it achieves a strong balance between high accuracy (0.9462) and the highest ROC-AUC score (0.8105). Random Forest is a close second, but XGBoost slightly outperforms it in terms of ROC-AUC, which is crucial for evaluating classification models comprehensively.



References

- https://www.kaggle.com/
- https://www.techtarget.com/searchbusinessanalytics/definition/data-exploration
- https://www.techtarget.com/searchdatamanagement/definition/data-preprocessing
- https://www.ibm.com/think/topics/exploratory-data-analysis

Group Members

1. DNA De Silva - D/BCS/22/0019
2. PKI Chiranthana - D/BCS/22/0020
3. WAHS Kumarasiri - D/BCS/22/0024
4. WACL Bandara - D/BCE/22/0020
5. TP Muthukumara - D/BCE/22/0024

Thank You