

## ✓ Data Preprocessing Theory

Preprocessing is the process of transforming raw data into a suitable format for further analysis or machine learning models. It improves data quality, ensures consistency, and enhances model performance. Preprocessing involves several steps, depending on the nature of the data and the intended use case.

---

### Steps in Preprocessing

#### 1. Data Collection

- Gather raw data from various sources such as databases, APIs, CSV files, or IoT devices.
- Ensure data completeness and reliability.

#### 2. Data Cleaning

- Handle missing values:
  - Remove rows/columns with missing data.
  - Impute missing values (mean, median, mode, interpolation).
- Remove duplicate records to avoid redundancy.
- Correct inconsistencies and errors (e.g., incorrect labels, typos).

#### 3. Data Transformation

- Convert data into a desired format for analysis.
- Apply normalization or standardization:
  - **Normalization (Min-Max Scaling)**: Scales values between 0 and 1.
  - **Standardization (Z-score Scaling)**: Transforms data to have a mean of 0 and standard deviation of 1.
- Encode categorical variables:
  - **One-Hot Encoding**: Converts categorical values into binary vectors.
  - **Label Encoding**: Assigns numerical labels to categories.

#### 4. Data Reduction

- Reduce dataset size without losing critical information.
- Use techniques like:

- **Dimensionality Reduction (PCA, LDA)**
- **Sampling (Random, Stratified)**
- **Aggregation (Grouping data)**

## 5. Data Splitting

- Divide data into training, validation, and testing sets.
- Common split ratios:
  - **Training Set (70-80%)**: Used to train the model.
  - **Validation Set (10-15%)**: Used for hyperparameter tuning.
  - **Test Set (10-15%)**: Used for final model evaluation.

## 6. Data Augmentation (For Image/Text Data)

- Apply transformations like rotation, flipping, cropping (for images).
- Use synonym replacement, back translation (for text data).

## ✓ Importing all the required libraries and set standard for output

```
import numpy as np
import pandas as pd

import kagglehub

import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 20)
pd.set_option('display.width', 800)

np.set_printoptions(threshold=np.inf, linewidth=np.inf)
```

## ✓ Definitions of utility functions

```
def print_as_pd(array, column_names=None):
    df = pd.DataFrame(array, columns=column_names)
    return df
```

## ✓ PREPROCESSING

## ✓ Importing the data set using kaggle

```
path = kagglehub.dataset_download("fedesoriano/stroke-prediction-dataset")
```

```
print("Path to dataset files:", path)
```

```
⇒ Path to dataset files: /root/.cache/kagglehub/datasets/fedesoriano/stroke-prediction-dat
```

## ✓ Reading the dataset using pandas and conversion to numpy array

```
df_pd = pd.read_csv(path+"/healthcare-dataset-stroke-data.csv")
```

```
col_names = df_pd.columns
```

```
df = df_pd.to_numpy()
```

```
print_as_pd(df)
```

```
⇒
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.2	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-	Rural	82.99	30.6	never	0

```
df = np.vstack([col_names,df])
```

```
print_as_pd(df)
```



	0	1	2	3	4	5	6	7
0	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
1	9046	Male	67.0	0	1	Yes	Private	Urban
2	51676	Female	61.0	0	0	Yes	Self-employed	Rural
3	31112	Male	80.0	0	1	Yes	Private	Rural
4	60182	Female	49.0	0	0	Yes	Private	Urban
...	...	...	...	...	...	...	...	...
5106	18234	Female	80.0	1	0	Yes	Private	Urban
5107	44873	Female	81.0	0	0	Yes	Self-employed	Urban
5108	19723	Female	35.0	0	0	Yes	Self-employed	Rural

## ✓ Removing useless column(s)

```
# Remove first column i.e. id

if(df[0,0] == 'id') :
    df = df.delete(df,0,axis=1)
print_as_pd(df)
```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_c
1	Male	67.0	0	1	Yes	Private	Urban	
2	Female	61.0	0	0	Yes	Self-employed	Rural	
3	Male	80.0	0	1	Yes	Private	Rural	
4	Female	49.0	0	0	Yes	Private	Urban	
...	...	...	...	...	...	...	...	
5106	Female	80.0	1	0	Yes	Private	Urban	
5107	Female	81.0	0	0	Yes	Self-employed	Urban	
5108	Female	35.0	0	0	Yes	Self-employed	Rural	

## ✓ Handling missing values : replacing with mean

```

headers = df[0]

numeric_data = df[1:].astype(object)

num_cols = [i for i in range(numeric_data.shape[1]) if np.issubdtype(type(numeric_data[0, i]), np.number)]

for col in num_cols:
    col_values = numeric_data[:, col].astype(float)
    col_mean = np.round(np.nanmean(col_values), 2)
    col_values[np.isnan(col_values)] = col_mean
    numeric_data[:, col] = col_values

df = np.vstack([headers, numeric_data])

print_as_pd(df)

```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_c
1	Male	67.0	0.0	1.0	Yes	Private	Urban	
2	Female	61.0	0.0	0.0	Yes	Self-employed	Rural	
3	Male	80.0	0.0	1.0	Yes	Private	Rural	
4	Female	49.0	0.0	0.0	Yes	Private	Urban	
...	...	...	...	...	...	...	...	
5106	Female	80.0	1.0	0.0	Yes	Private	Urban	
5107	Female	81.0	0.0	0.0	Yes	Self-employed	Urban	
5108	Female	35.0	0.0	0.0	Yes	Self-employed	Rural	

## ✓ Normalizing the data

$$norm(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

```
columns_to_normalize = ['age', 'avg_glucose_level', 'bmi']
col_indices = {col: np.where(df[0] == col)[0][0] for col in columns_to_normalize}
```

```
numeric_data = df[1:, list(col_indices.values())].astype(float)
```

```
min_vals = np.min(numeric_data, axis=0)
max_vals = np.max(numeric_data, axis=0)
normalized_data = (numeric_data - min_vals) / (max_vals - min_vals)
normalized_data = np.round(normalized_data,4)
```

```
for i, col in enumerate(col_indices.values()):
    df[1:, col] = normalized_data[:, i]
```

```
print_as_pd(df)
```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
1	Male	0.8169	0.0	1.0	Yes	Private	Urban	
2	Female	0.7437	0.0	0.0	Yes	Self-employed	Rural	
3	Male	0.9756	0.0	1.0	Yes	Private	Rural	
4	Female	0.5972	0.0	0.0	Yes	Private	Urban	
...	...	...	...	...	...	...	...	
5106	Female	0.9756	1.0	0.0	Yes	Private	Urban	
5107	Female	0.9878	0.0	0.0	Yes	Self-employed	Urban	
5108	Female	0.4263	0.0	0.0	Yes	Self-employed	Rural	

## ✓ Encoding text values in to numbers

### ✓ Label Encoding

```

categorical_columns = ['gender', 'ever_married', 'Residence_type']

column_indices = {col: np.where(df[0] == col)[0][0] for col in categorical_columns}

for col, index in column_indices.items():
    unique_values, encoded_values = np.unique(df[1:, index], return_inverse=True)
    df[1:, index] = encoded_values

print_as_pd(df)

```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
1	1	0.8169	0.0	1.0	1	Private		1
2	0	0.7437	0.0	0.0	1	Self-employed		0
3	1	0.9756	0.0	1.0	1	Private		0
4	0	0.5972	0.0	0.0	1	Private		1
...	...	...	...	...	...	...	...	...
5106	0	0.9756	1.0	0.0	1	Private		1
5107	0	0.9878	0.0	0.0	1	Self-employed		1
5108	0	0.4263	0.0	0.0	1	Self-employed		0

## ✓ One Hot Encoding

## ✓ replacing children with never worked

```
work_type_col_idx = np.where(df[0] == 'work_type')[0][0]
df[1:, work_type_col_idx] = np.where(df[1:, work_type_col_idx] == 'children', 'Never_worked')
print_as_pd(df)
```





	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
1	1	0.8169	0.0	1.0	1	Private		1
2	0	0.7437	0.0	0.0	1	Self-employed		0
3	1	0.9756	0.0	1.0	1	Private		0
4	0	0.5972	0.0	0.0	1	Private		1
...	...	...	...	...	...	...	...	...
5106	0	0.9756	1.0	0.0	1	Private		1
5107	0	0.9878	0.0	0.0	1	Self-employed		1
5108	0	0.4263	0.0	0.0	1	Self-employed		0

## ✓ encoding work\_type and smoking\_status

```

columns_to_encode = ['work_type', 'smoking_status']
column_indices = {col: np.where(df[col] == col)[0][0] for col in columns_to_encode}

def one_hot_encode(data, col_index):
    unique_values = np.unique(data[1:, col_index])
    one_hot_matrix = np.zeros((data.shape[0] - 1, len(unique_values)), dtype=int)

    for i, val in enumerate(data[1:, col_index]):
        one_hot_matrix[i, np.where(unique_values == val)[0][0]] = 1

    new_headers = [f"{data[0, col_index]}_{val}" for val in unique_values]

    return new_headers, one_hot_matrix

work_type_headers, work_type_encoded = one_hot_encode(df, column_indices['work_type'])
smoking_headers, smoking_encoded = one_hot_encode(df, column_indices['smoking_status'])

new_headers = np.concatenate((df[0], work_type_headers, smoking_headers))
new_data = np.hstack((df[1:], work_type_encoded, smoking_encoded))

df = np.vstack((new_headers, new_data))

```

```
print_as_pd(df)
```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
1	1	0.8169	0.0	1.0	1	Private		1
2	0	0.7437	0.0	0.0	1	Self-employed		0
3	1	0.9756	0.0	1.0	1	Private		0
4	0	0.5972	0.0	0.0	1	Private		1
...	...	...	...	...	...	...	...	...
5106	0	0.9756	1.0	0.0	1	Private		1
5107	0	0.9878	0.0	0.0	1	Self-employed		1
5108	0	0.4263	0.0	0.0	1	Self-employed		0
5109	1	0.6216	0.0	0.0	1	Private		0
5110	0	0.5361	0.0	0.0	1	Govt_job		1

5111 rows × 19 columns

## ✓ Removing redundant columns and rearrange columns

```
columns_to_remove = ['work_type', 'Residence_type', 'smoking_status']
column_indices_to_remove = [np.where(df[col] == col)[0][0] for col in columns_to_remove]

df = np.delete(df, column_indices_to_remove, axis=1)

stroke_index = np.where(df[0] == 'stroke')[0][0]

df = np.concatenate((df[:, :stroke_index], df[:, stroke_index + 1:], df[:, stroke_index:stroke_index+1]), axis=1)

print_as_pd(df)
```



	0	1	2	3	4	5	6	
0	gender	age	hypertension	heart_disease	ever_married	avg_glucose_level	bmi	work
1	1	0.8169	0.0	1.0	1	0.8013	0.3013	
2	0	0.7437	0.0	0.0	1	0.679	0.2129	
3	1	0.9756	0.0	1.0	1	0.2345	0.2543	
4	0	0.5972	0.0	0.0	1	0.536	0.2761	
...	...	...	...	...	...	...	...	
5106	0	0.9756	1.0	0.0	1	0.1322	0.2129	
5107	0	0.9878	0.0	0.0	1	0.3235	0.3402	
5108	0	0.4263	0.0	0.0	1	0.1287	0.2325	
5109	1	0.6216	0.0	0.0	1	0.5132	0.1753	
5110	0	0.5361	0.0	0.0	1	0.1392	0.1821	

5111 rows × 16 columns

## ✓ Visualization

### Importing Libraries and preprocessed Dataset

```
import pandas as pd
```

```
file_url = "https://raw.githubusercontent.com/Aayush-Dwivedi/stroke-prediction-dataset/main/p
df = pd.read_csv(file_url)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Fill missing values in BMI with the mean
df["bmi"].fillna(df["bmi"].mean(), inplace=True)
```

```
# Convert relevant columns to numeric types
df[['age', 'avg_glucose_level', 'bmi', 'stroke']] = df[['age', 'avg_glucose_level', 'bmi', 'stroke']].astype(float)
```



<ipython-input-87-1c516510fdcd>:9: FutureWarning: A value is trying to be set on a copy  
The behavior will change in pandas 3.0. This inplace method will never work because the  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col

```
df["bmi"].fillna(df["bmi"].mean(), inplace=True)
```



## Histograms For Numerical Features

# 1. Histograms for Numerical Features

```
import matplotlib.pyplot as plt
```

```
numerical_cols = ['age', 'avg_glucose_level', 'bmi']
```

```
df[numerical_cols].hist(figsize=(12, 6), bins=30, color='skyblue', edgecolor='black')
```

```
plt.suptitle("Distribution of Numerical Features")
```

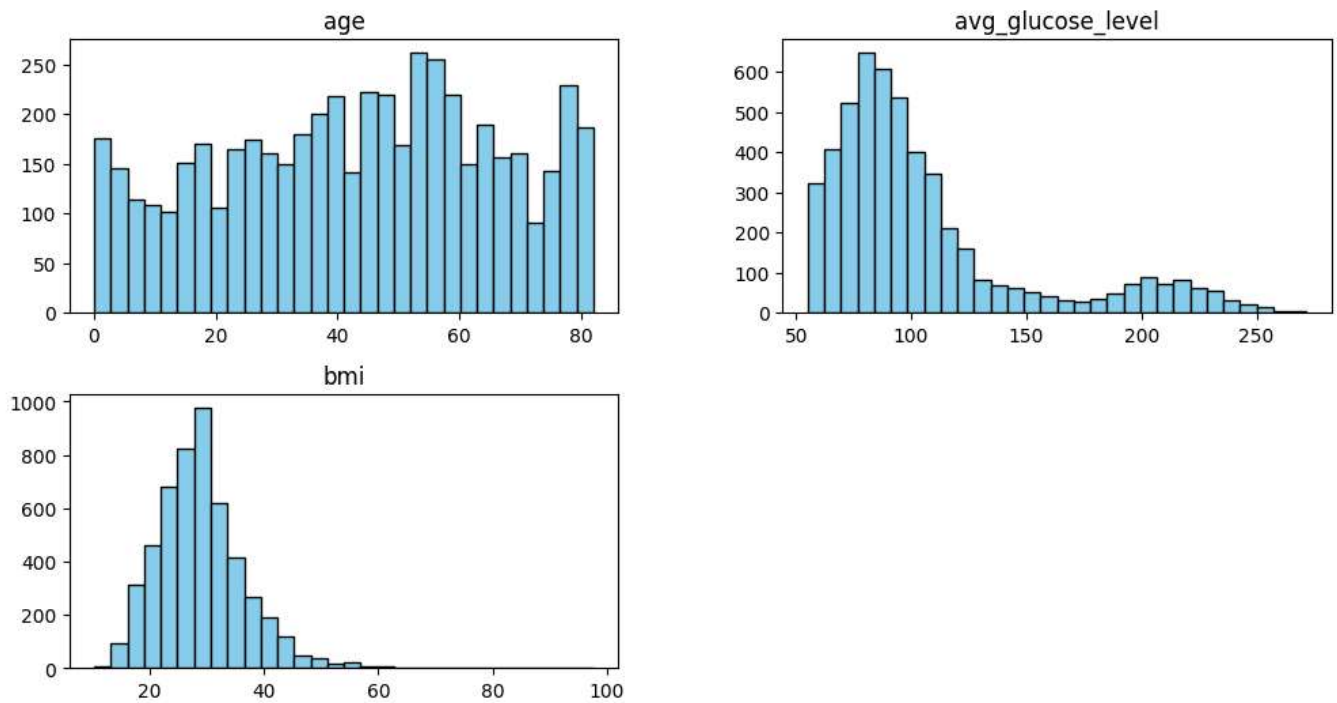
```
for ax in plt.gcf().axes:
```

```
    ax.grid(False)
```

```
plt.show()
```

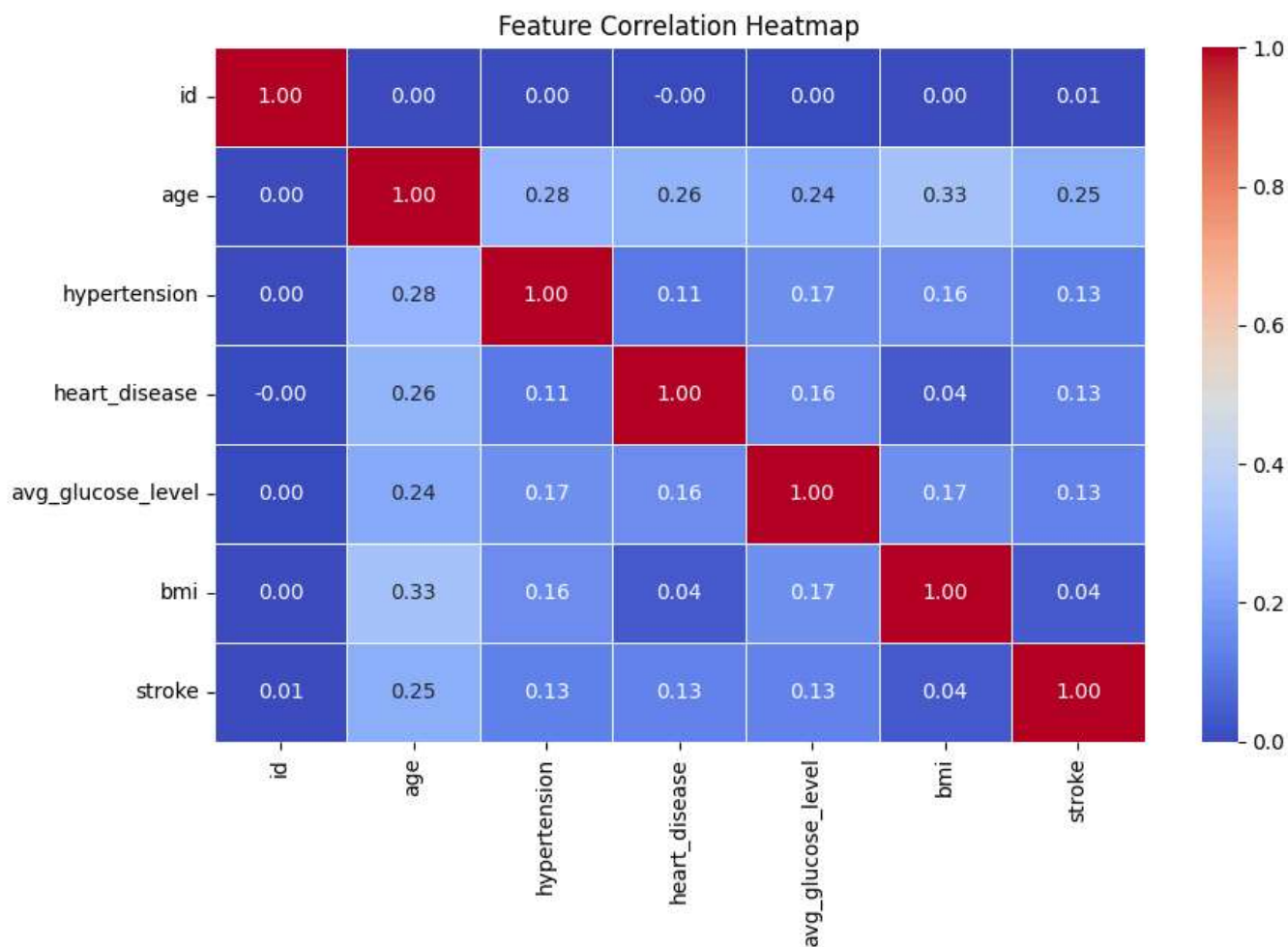


### Distribution of Numerical Features



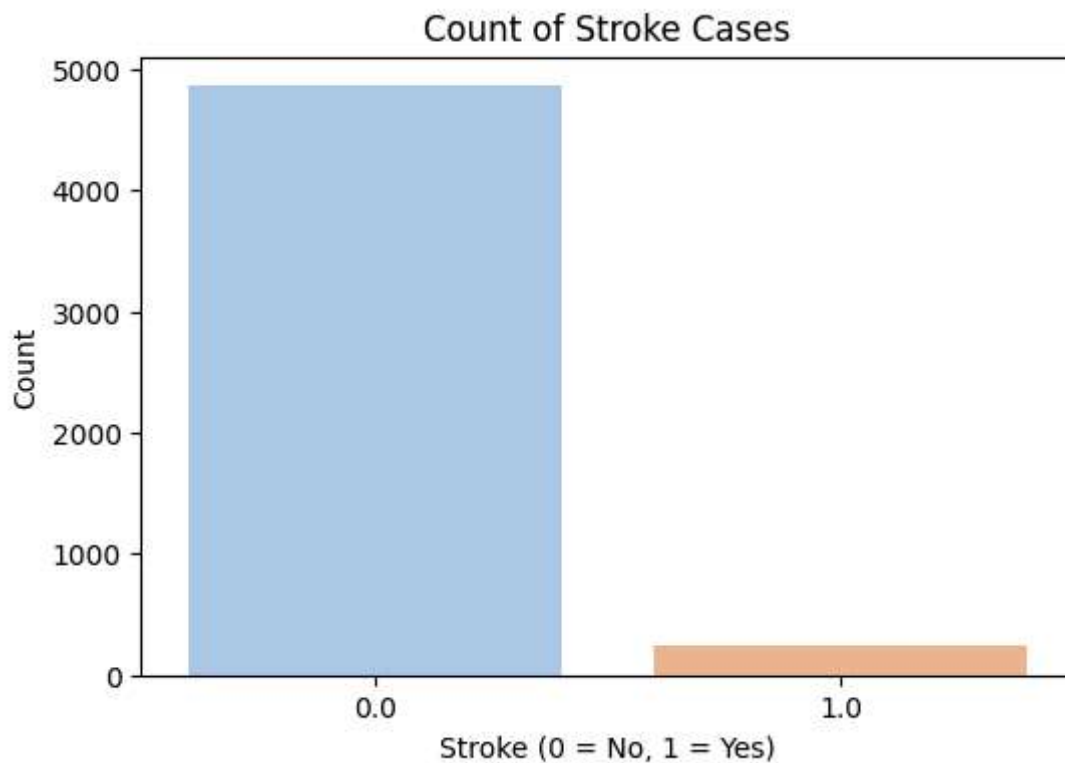
### Correlation Heatmap

```
# 2. Correlation Heatmap
plt.figure(figsize=(10, 6))
numeric_df = df.select_dtypes(include=['number']) # Select only numeric columns
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



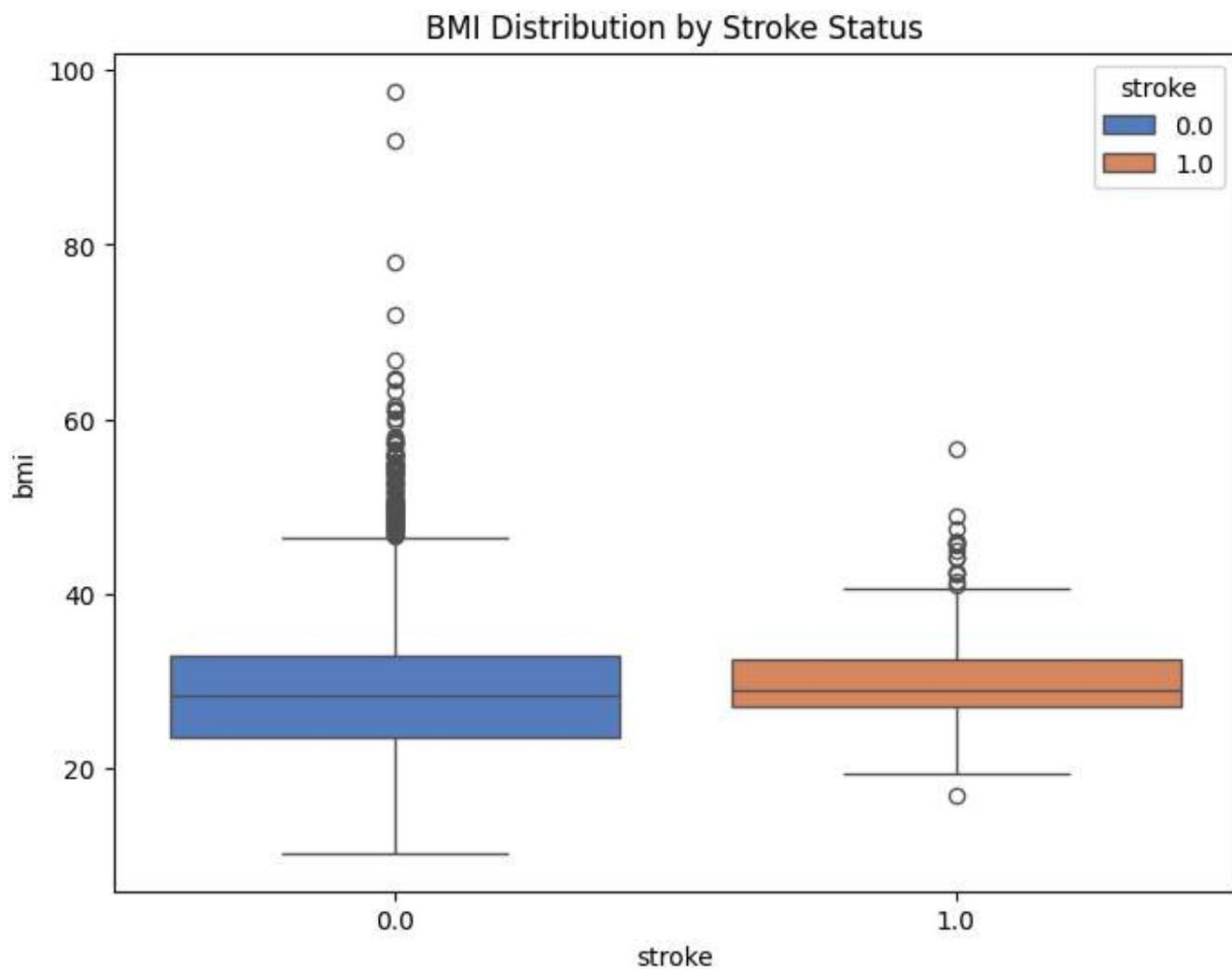
Bargraph for stroke cases count

```
# 3. Stroke Cases Count
plt.figure(figsize=(6, 4))
sns.countplot(x="stroke", hue="stroke", data=df, palette="pastel", legend=False) # Fix appl
plt.title("Count of Stroke Cases")
plt.xlabel("Stroke (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()
```



## Boxplot for BMI

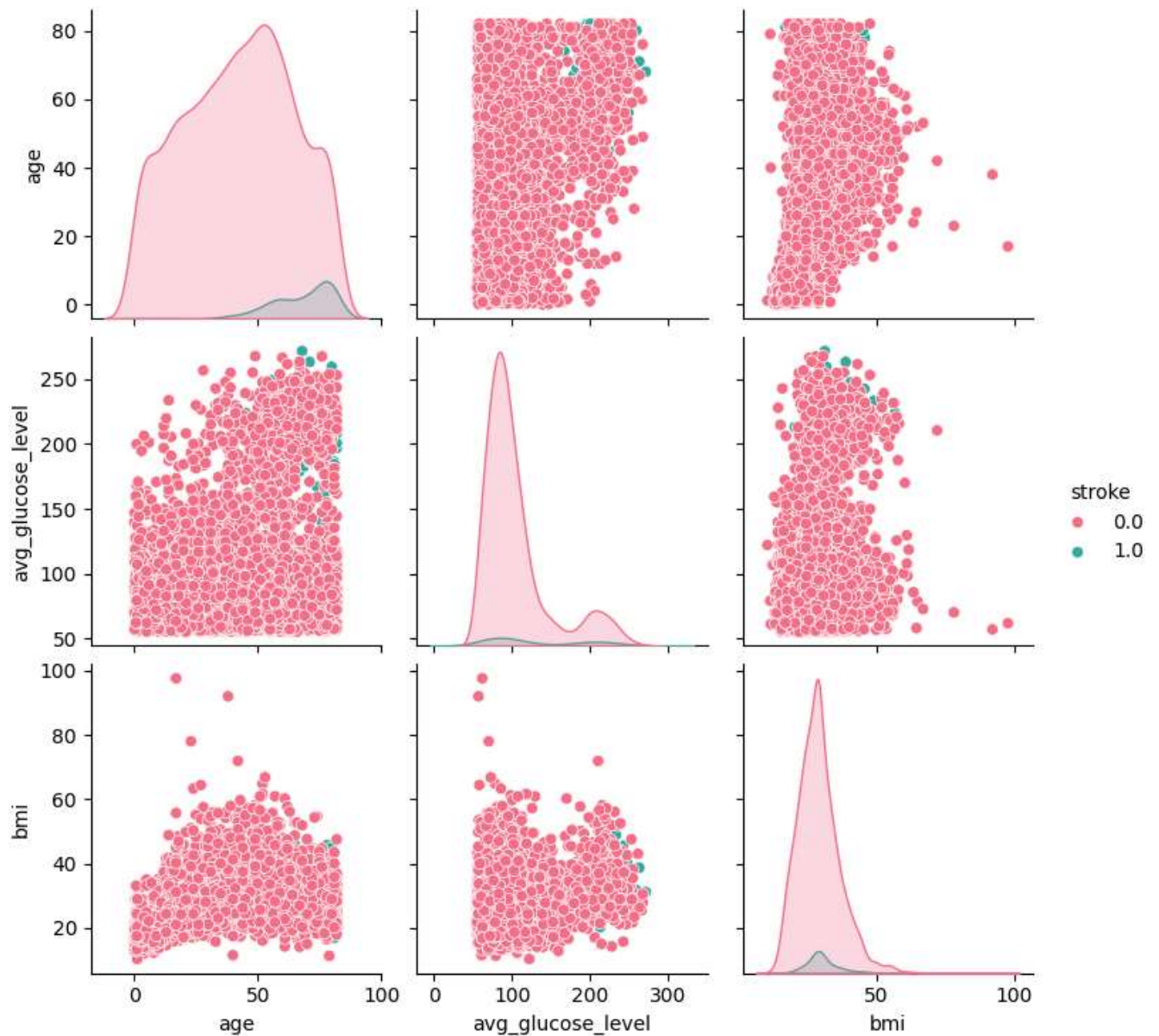
```
# 4. Boxplot of BMI by Stroke Status
plt.figure(figsize=(8, 6))
sns.boxplot(x="stroke", y="bmi", hue="stroke", data=df, palette="muted", dodge=False)
plt.title("BMI Distribution by Stroke Status")
plt.show()
```



## Pairplot

```
# 5. Pairplot for Important Features Colored by Stroke
sns.pairplot(df, hue="stroke", vars=['age', 'avg_glucose_level', 'bmi'], palette="husl")
plt.show()
```





## Categorical Feature Distribution

### ✓ Using data to text on a KNN classifier

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc
import pandas as pd
import numpy as np

# Define features and target
X = df.drop(columns=['stroke']) # Features
y = df['stroke'].astype(int) # Target variable

# Identify categorical and numerical columns
# Replace these with your actual column names
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

print(f"Categorical columns: {categorical_cols}")
print(f"Numerical columns: {numerical_cols}")

# Create preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_cols)
    ])

# Create KNN pipeline with preprocessing
knn_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier())
])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Define hyperparameters to tune
param_grid = {
    'classifier__n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'classifier__weights': ['uniform', 'distance'],
    'classifier__metric': ['euclidean', 'manhattan']
}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(knn_pipeline, param_grid, cv=5, scoring='roc_auc', n_jobs=-1, ver
grid_search.fit(X_train, y_train)

# Print best parameters
print("\nBest parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)
```

```
# Evaluate the model on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[: , 1] # For ROC AUC calculation

# Print classification metrics
print("\nTest set metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_prob))
print("\nClassification report:\n", classification_report(y_test, y_pred))
```


⇒ Categorical columns: ['gender', 'ever\_married', 'work\_type', 'Residence\_type', 'smoking\_  
Numerical columns: ['id', 'age', 'hypertension', 'heart\_disease', 'avg\_glucose\_level', '  
Fitting 5 folds for each of 28 candidates, totalling 140 fits

Best parameters: {'classifier\_\_metric': 'euclidean', 'classifier\_\_n\_neighbors': 3, 'clas  
Best cross-validation score: nan

Test set metrics:  
Accuracy: 0.9393346379647749  
ROC AUC: 0.5748045267489712

Classification report:

	precision	recall	f1-score	support
0	0.95	0.98	0.97	972
1	0.17	0.06	0.09	50
accuracy			0.94	1022
macro avg	0.56	0.52	0.53	1022
weighted avg	0.91	0.94	0.93	1022

/usr/local/lib/python3.11/dist-packages/sklearn/model\_selection/\_search.py:1108: UserWar  
warnings.warn(  
◀  ▶

## ✓ Confusion Matrix

Double-click (or enter) to edit

```
print("\nConfusion matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

⇒ Confusion matrix:  
[[957 15]  
[ 47 3]]

## ✓ Plot confusion matrix

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Stroke', 'Stroke'],
            yticklabels=['No Stroke', 'Stroke'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

