Content from Deliverable_1.ipynb

```
# MSCS 634 - Project Deliverable 1
# Advanced Data Mining for Data-Driven Insights and Predictive Modeling
# Dataset: Stroke Prediction Dataset (Kaggle)

# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set display and style options
pd.set_option('display.max_columns', None)
sns.set(style='whitegrid')

# Load Dataset
df = pd.read_csv('/content/healthcare-dataset-stroke-data.csv')
```

```
# Inspect structure
print("Dataset shape:", df.shape)
df.head()

# Basic Info
df.info()
df.describe(include='all')

# Check missing values
print("\nMissing values per column:\n", df.isnull().sum())

# Data Cleaning
# Drop 'id' column (not useful for modeling)
df.drop(columns=['id'], inplace=True)

# Handle missing values in 'bmi' (replace with median)
df['bmi'].fillna(df['bmi'].median(), inplace=True)
```

```
Dataset shape: (5110, 12)
<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

Missing values per column:
 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
/tmp/ipython-input-2214330086.py:17: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the ope


  df['bmi'].fillna(df['bmi'].median(), inplace=True)
```

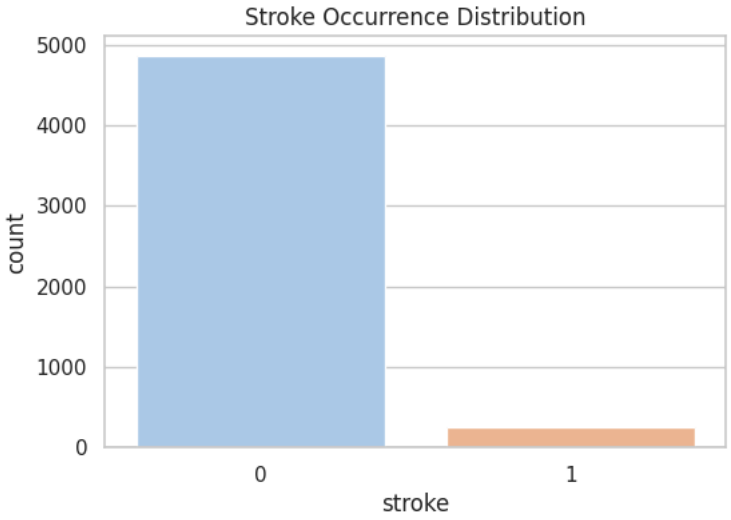
```
# Check for duplicates
duplicates = df.duplicated().sum()
print("Duplicate records:", duplicates)
df.drop_duplicates(inplace=True)

# Fix categorical inconsistencies (e.g., work_type typo)
df['work_type'].replace({'Govt_jov': 'Govt_job'}, inplace=True)

# Confirm cleaning
df.info()
```

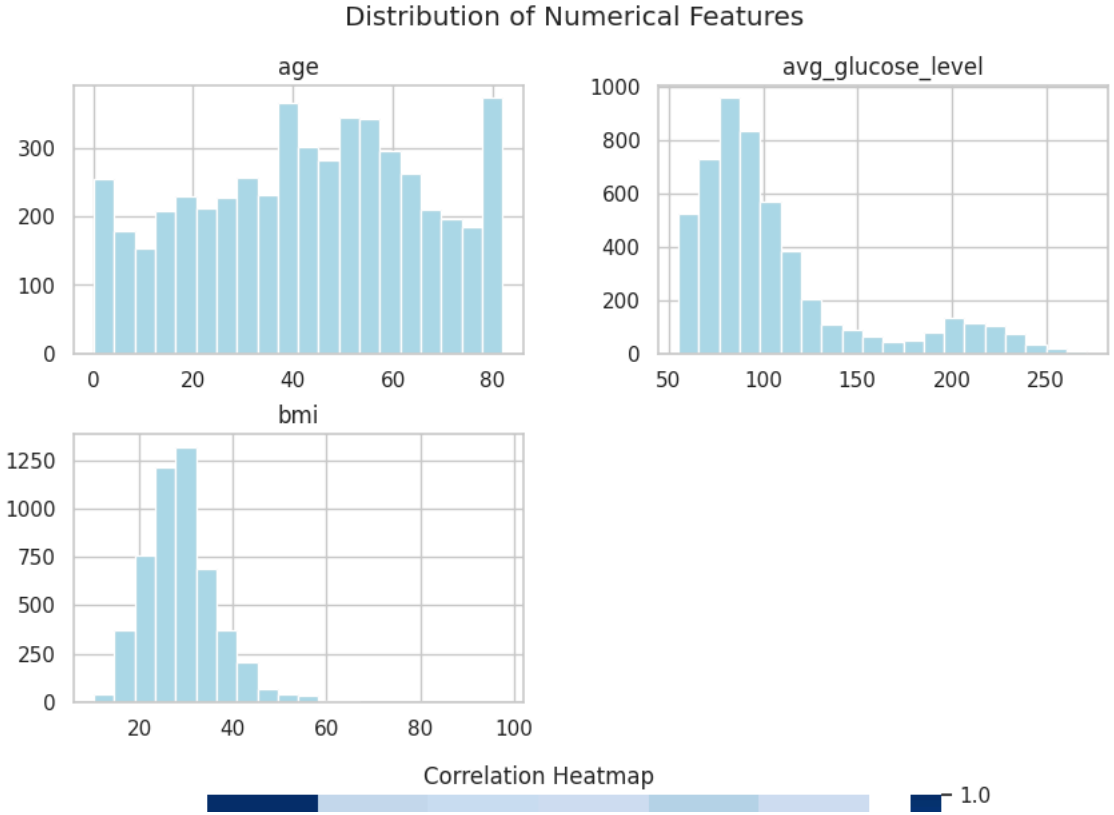
```
Duplicate records: 0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             5110 non-null   object
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64
 3   heart_disease      5110 non-null   int64
 4   ever_married       5110 non-null   object
 5   work_type          5110 non-null   object
 6   Residence_type     5110 non-null   object
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                5110 non-null   float64
 9   smoking_status     5110 non-null   object
 10  stroke             5110 non-null   int64
dtypes: float64(3), int64(3), object(5)
memory usage: 439.3+ KB
```

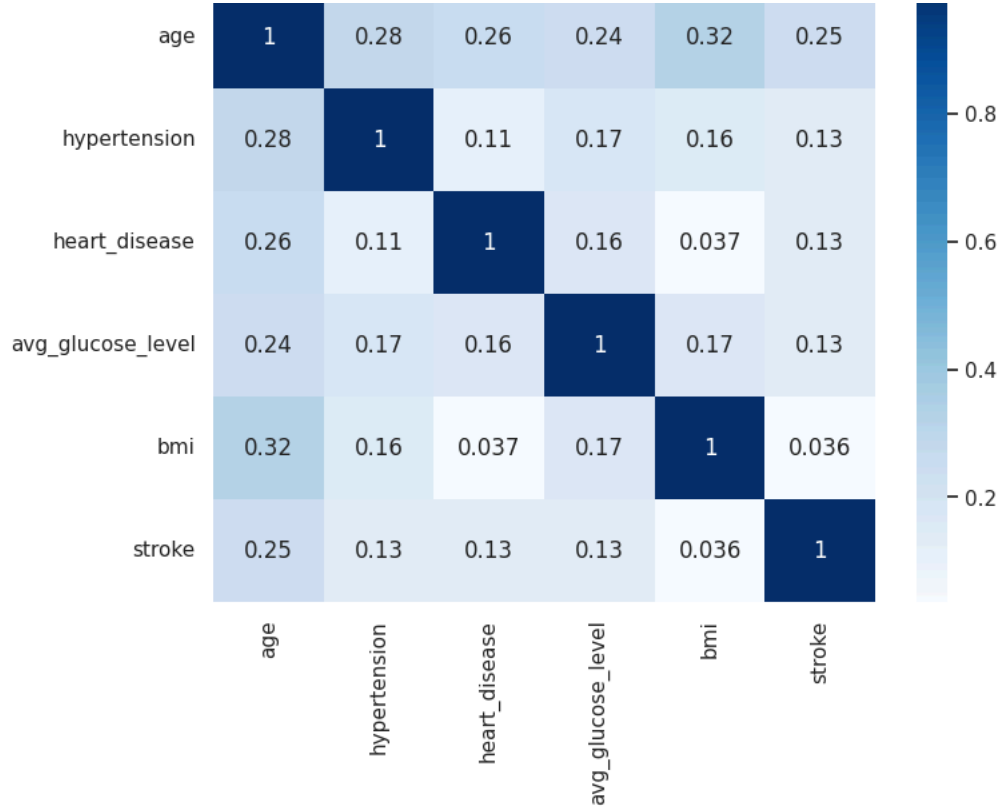
```
/tmp/ipython-input-1039141086.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the ope


  df['work_type'].replace({'Govt_jov': 'Govt_job'}, inplace=True)
```

```
# Exploratory Data Analysis (EDA)
#  Distribution of Target Variable
plt.figure(figsize=(6,4))
sns.countplot(x='stroke', data=df, palette='pastel')
plt.title('Stroke Occurrence Distribution')
plt.show()

#  Numeric Distributions
num_cols = ['age', 'avg_glucose_level', 'bmi']
df[num_cols].hist(bins=20, figsize=(10,6), color='lightblue')
plt.suptitle('Distribution of Numerical Features')
plt.show()

#  Correlation Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='Blues')
plt.title('Correlation Heatmap')
plt.show()

#  Relationship between Age, BMI, and Stroke
plt.figure(figsize=(7,5))
sns.boxplot(x='stroke', y='age', data=df, palette='coolwarm')
plt.title('Age vs Stroke')
plt.show()

plt.figure(figsize=(7,5))
sns.boxplot(x='stroke', y='bmi', data=df, palette='coolwarm')
plt.title('BMI vs Stroke')
plt.show()

#  Categorical Breakdown
plt.figure(figsize=(8,4))
sns.countplot(x='gender', hue='stroke', data=df)
plt.title('Gender vs Stroke')
plt.show()

plt.figure(figsize=(8,4))
sns.countplot(x='work_type', hue='stroke', data=df)
plt.title('Work Type vs Stroke')
plt.xticks(rotation=45)
plt.show()
```

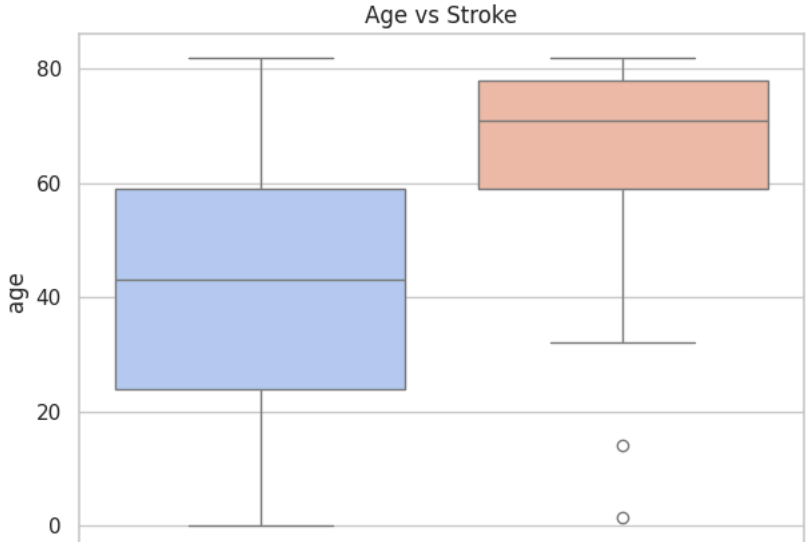
```
/tmp/ipython-input-1133343763.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='stroke', data=df, palette='pastel')
```

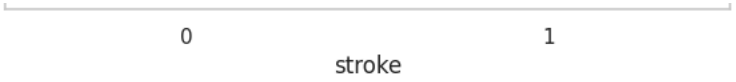




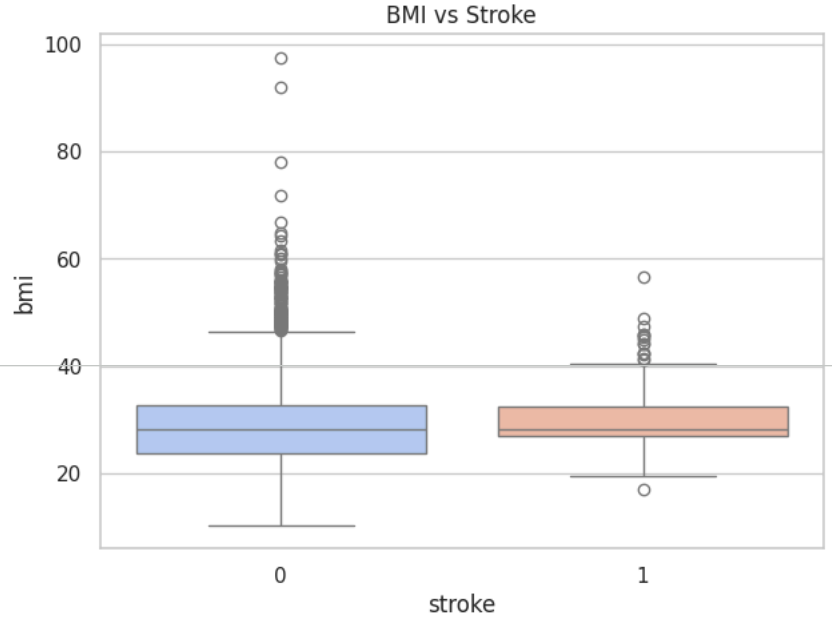


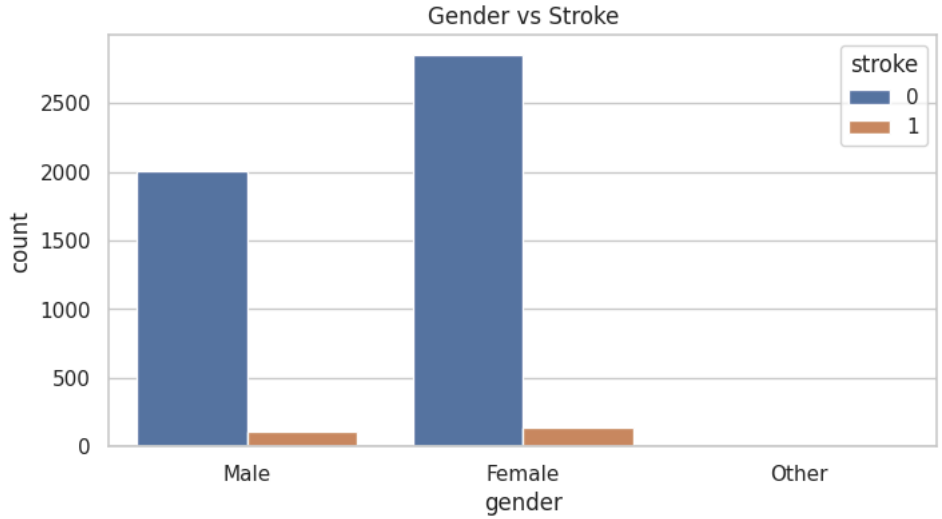

```
/tmp/ipython-input-1133343763.py:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='stroke', y='age', data=df, palette='coolwarm')
```

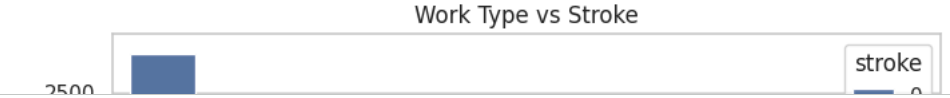





```
/tmp/ipython-input-1133343763.py:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='stroke', y='bmi', data=df, palette='coolwarm')
```







Deliverable 2 content

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

from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split, KFold, cross_validate
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score, mean_squared_error

data = load_diabetes()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

numeric_features = X.columns.tolist()

preprocessor = Pipeline([
    ("poly", PolynomialFeatures(degree=2, include_bias=False)),
    ("scaler", StandardScaler())
])

models = {
    "LinearRegression": Pipeline([("prep", preprocessor), ("reg", LinearRegression())]),
    "Ridge": Pipeline([("prep", preprocessor), ("reg", Ridge())]),
    "Lasso": Pipeline([("prep", preprocessor), ("reg", Lasso(max_iter=10000))])
}

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

cv = KFold(n_splits=5, shuffle=True, random_state=42)
results = {}

for name, model in models.items():
    cv_res = cross_validate(model, X_train, y_train, scoring=["r2","neg_mean_squared_error"], cv=cv)
    mse = -cv_res["test_neg_mean_squared_error"]
    rmse = np.sqrt(mse)
    results[name] = {
        "R2_CV": cv_res["test_r2"].mean(),
        "RMSE_CV": rmse.mean()
    }

print(pd.DataFrame(results).T)

test_results = {}
trained_models = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    trained_models[name] = model
    pred = model.predict(X_test)
    test_results[name] = {
        "R2 Test": r2_score(y_test, pred),
```

```
            "RMSE_Test": np.sqrt(mean_squared_error(y_test, pred))
        }

    test_df = pd.DataFrame(test_results).T
    print(test_df)

    best_model_name = test_df["R2_Test"].idxmax()
    best_model = trained_models[best_model_name]

    pred = best_model.predict(X_test)

    plt.figure(figsize=(6,6))
    plt.scatter(y_test, pred, alpha=0.6)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], "--")
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title("Actual vs Predicted")
    plt.show()

    residuals = y_test - pred
    plt.figure(figsize=(8,4))
    sns.histplot(residuals, kde=True)
    plt.title("Residuals")
    plt.show()

    plt.figure(figsize=(8,4))
    plt.scatter(pred, residuals, alpha=0.6)
    plt.axhline(0, linestyle="--", color="black")
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title("Residuals vs Predicted")
    plt.show()
```

Deliverable 3 content

```
                    R2_CV    RMSE_CV
LinearRegression  0.382059  60.465540
Ridge             0.382059  60.465540
Lasso             0.445447  57.133522
```

```
import warnings, sys, os

# Hide Python warnings
warnings.filterwarnings("ignore")

# Hide ALL stderr warnings from libraries like jupyter_client
sys.stderr = open(os.devnull, "w")
import logging
logging.disable(logging.CRITICAL)
```

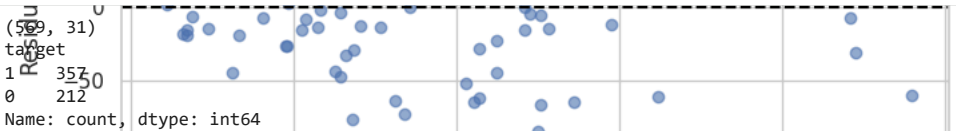
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc,
    accuracy_score, f1_score, classification_report
)
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN

# For association rules
from mlxtend.frequent_patterns import apriori, association_rules

# reproducibility
RANDOM_STATE = 42

print('libraries imported')
```

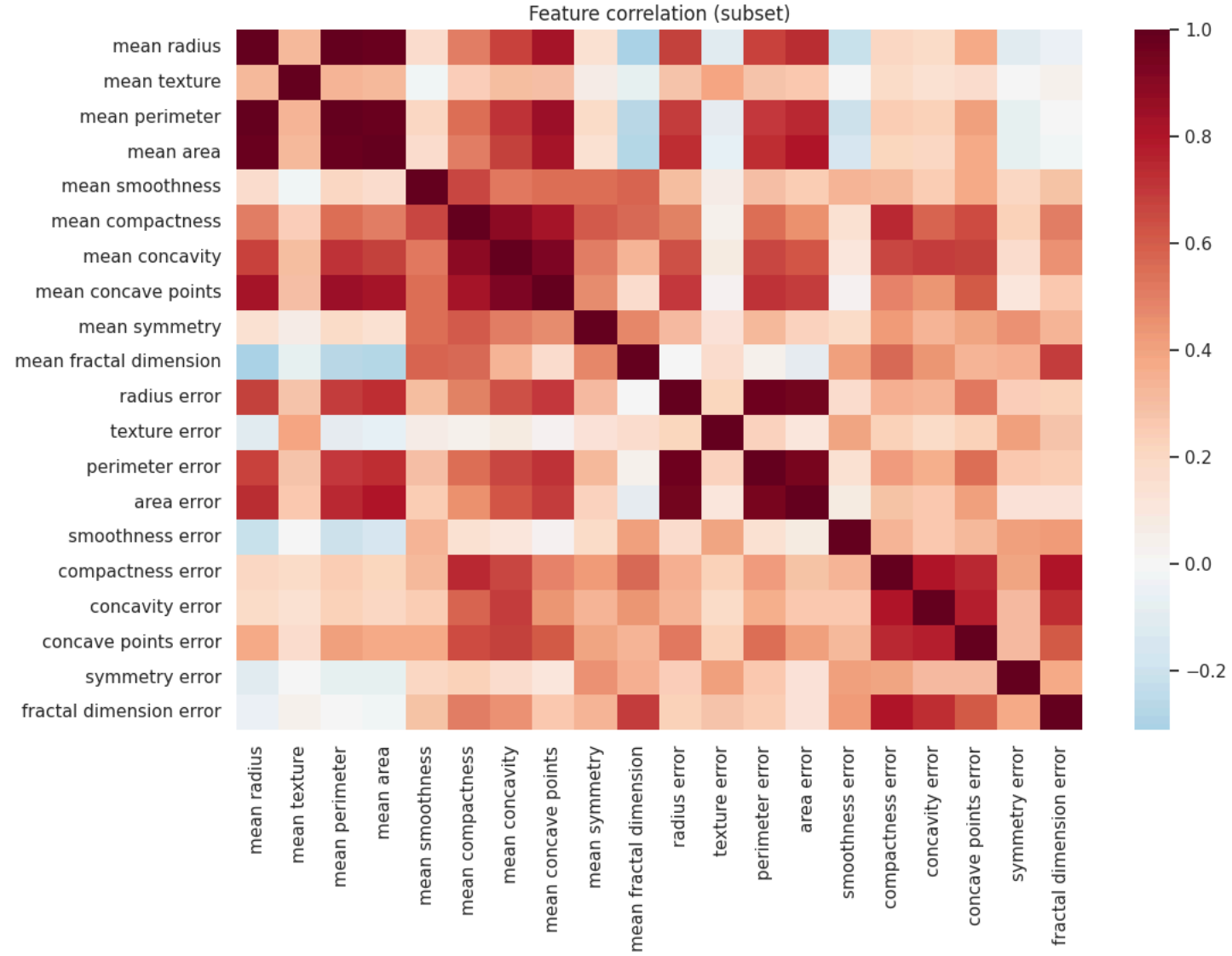
```
libraries imported
```

```
#Load dataset & exploratory data analysis
data = load_breast_cancer(as_frame=True)
df = data.frame.copy()

df['target'] = data.target

df.head()
# Cell: quick EDA
print(df.shape)
print(df['target'].value_counts())
df.describe().T.head()
# Cell: correlation heatmap (top 20 features by variance)
plt.figure(figsize=(12,8))
```

```
cols = df.columns[:-1]  # exclude target
corr = df[cols].corr()
# plot only a subset for readability
sns.heatmap(corr.iloc[:20,:20], cmap='RdBu_r', center=0)
plt.title('Feature correlation (subset)')
plt.show()
```

```
(569, 31)
target
1    357
0    212
Name: count, dtype: int64
```





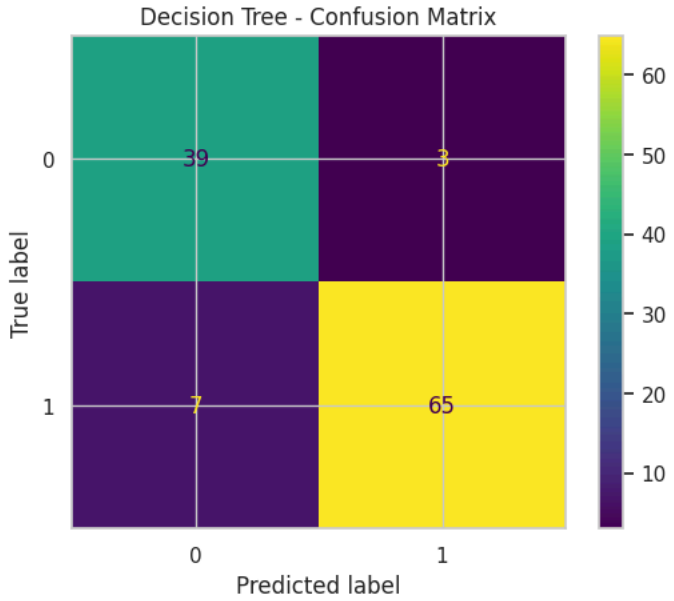
```
# Preprocessing
X = df.drop(columns=['target'])
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print('train/test shapes:', X_train.shape, X_test.shape)
```

```
train/test shapes: (455, 30) (114, 30)
```

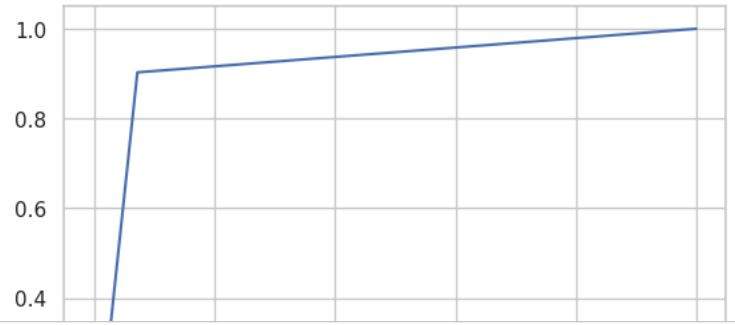
```
# train decision tree
dt = DecisionTreeClassifier(random_state=RANDOM_STATE)
dt.fit(X_train_scaled, y_train)

y_pred_dt = dt.predict(X_test_scaled)

# metrics
acc_dt = accuracy_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)
print('Decision Tree - Accuracy:', acc_dt, 'F1:', f1_dt)

# display confusion matrix
ConfusionMatrixDisplay.from_estimator(dt, X_test_scaled, y_test)
plt.title('Decision Tree - Confusion Matrix')
plt.show()

# ROC curve
y_proba_dt = dt.predict_proba(X_test_scaled)[:,1]
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_proba_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)
plt.plot(fpr_dt, tpr_dt, label=f'DT (AUC = {roc_auc_dt:.3f})')

# Cell: k-NN
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
acc_knn = accuracy_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
print('k-NN - Accuracy:', acc_knn, 'F1:', f1_knn)

ConfusionMatrixDisplay.from_estimator(knn, X_test_scaled, y_test)
plt.title('k-NN - Confusion Matrix')
plt.show()

# ROC for k-NN
y_proba_knn = knn.predict_proba(X_test_scaled)[:,1]
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_proba_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)
plt.plot(fpr_knn, tpr_knn, label=f'k-NN (AUC = {roc_auc_knn:.3f})')
```

```
# finish ROC plot
plt.plot([0,1],[0,1],'--',linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend()
plt.show()
```

```
Decision Tree - Accuracy: 0.9122807017543859 F1: 0.9285714285714286
```




```
k-NN - Accuracy: 0.956140350877193 F1: 0.9655172413793104
```




```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10, 20],
    'max_features': ['sqrt', 'log2'],
}

rf = RandomForestClassifier(random_state=RANDOM_STATE)

rs = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=10,
    scoring='f1',
    cv=5,
    random_state=42,
```