1. Problem Definition:

The goal of this project is to build a robust binary classification pipeline that predicts the onset of diabetes in patients based on their clinical measurements. The system should maximize predictive performance (accuracy, F1-score, AUC) while ensuring reproducibility and maintainability. Key deliverables:

- Data ingestion and preprocessing scripts handling missing/zero/erroneous values
- Exploratory data analysis (EDA) identifying feature distributions, correlations, and class imbalances
- Model training framework comparing at least five algorithms with consistent evaluation metrics
- Final deployment-ready model with persisted preprocessing and inference code

Target Variable: `Outcome` (0 = non-diabetic, 1 = diabetic)

Performance Benchmark: ≥ 0.90 AUC, ≥ 0.95 accuracy on held-out test set.

---

2. Data Collection

2.1 Source

- **Kaggle Dataset**: `diabetes.csv` from the “Healthcare Diabetes Prediction” competition repository
- **Download**: via Kaggle API or direct CSV link

```python
import pandas as pd
df = pd.read_csv('/mnt/data/diabetes.csv')  # shape: (768, 9)
```

- **Sample Size**: 768 patient records; 9 columns (8 features + 1 label).

2.2 Column Definitions

Column	Type	Description
Pregnancies	int64	Number of times pregnant
Glucose	int64	2-hour plasma glucose concentration
BloodPressure	int64	Diastolic blood pressure (mm Hg)
SkinThickness	int64	Triceps skinfold thickness (mm)
Insulin	int64	2-hour serum insulin (mu U/ml)
BMI	float64	Body mass index (kg/m^2)
DiabetesPedigreeFunction	float64	Pedigree function assessing genetic diabetes risk
Age	int64	Age in years
Outcome	int64	0 = non-diabetic, 1 = diabetic

---

3. Data Representation

- **DataFrame**: 768×9 (no text columns). 0-based index. Dtype summary:
  - int64: 6 columns
  - float64: 2 columns
  - int64 (label): 1 column
- **Memory Footprint**: ~60 KB

```
df.info()
#> RangeIndex: 768 entries, 0 to 767
#> Data columns (total 9):
#>  #   Column                    Non-Null Count  Dtype
#> ---  ------                    --------------  -----
#>  0   Pregnancies               768 non-null    int64
#>  ...
#>  7   Age                       768 non-null    int64
#>  8   Outcome                   768 non-null    int64
```

---

4. Data Wrangling

4.1 Dropping Unused Columns

- No ID or index columns beyond default; no drop operation required.

4.2 Handling Missing & Implausible Values

- Zero values in clinical measurements are physiologically invalid. Replace zeros with NaN for features: Glucose, BloodPressure, SkinThickness, Insulin, BMI.

```
cols_with_zero = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
df[cols_with_zero] = df[cols_with_zero].replace(0, np.nan)
```

- **Count of missing values before imputation**:
  - Glucose: 5 missing
  - BloodPressure: 35 missing
  - SkinThickness: 227 missing
  - Insulin: 374 missing
  - BMI: 11 missing
- **Imputation Strategy**: Median imputation per column to reduce outlier bias.

```
from sklearn.impute import SimpleImputer
median_imp = SimpleImputer(strategy='median')
df[cols_with_zero] = median_imp.fit_transform(df[cols_with_zero])
```

---

5. Exploratory Data Analysis (EDA)

5.1 Univariate Analysis

- **Distribution plots**: Histograms & boxplots for each numeric feature.
- **Skewness & Outliers**: `Insulin` shows right-skew; log-transform considered but not applied.
- **Descriptive statistics** (after imputation):
  - Glucose: mean=120.9, median=117.0, std=32.0
  - BMI: mean=31.99, median=32.0, std=7.88

```
sns.histplot(df['Glucose'], bins=20)
```

5.2 Bivariate Analysis

- **Feature vs. Outcome**:
  - Boxplot comparing `Glucose` by `Outcome`: diabetic group median ~141 vs. non-diabetic median ~108.
  - **Statistical test**: independent t-test confirms significant mean difference ($p < 0.001$).

```
sns.boxplot(x='Outcome', y='Glucose', data=df)
```

5.3 Multivariate Analysis

- **Correlation Matrix**: Pearson coefficients; highest corr with Outcome: `Glucose` (0.47), `Age` (0.24), `BMI` (0.30).

```
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

- **Pairplot** for top 3 correlated features.
- **Feature importance** from RandomForestClassifier (n_estimators=100) shows:
  1. Glucose (0.31)
  2. BMI (0.17)
  3. Age (0.14)

---

6. Data Preprocessing Pipeline

6.1 Scaling & Transformation

- **StandardScaler** for zero-mean, unit-variance scaling on all numeric inputs.
- Pipeline step: `('imputer', median_imp)`, `('scaler', StandardScaler())`.

6.2 Train/Test Split

- **Stratified Split** to maintain class proportions:

```
from sklearn.model_selection import train_test_split
X = df.drop('Outcome', axis=1)
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=42)
```

- Resulting shapes: `X_train` (614, 8), `X_test` (154, 8).

---

7. Modeling & Evaluation

7.1 Model Training

- Evaluated classifiers with default hyperparameters:
    1. **LogisticRegression** (solver='liblinear')
    2. **KNeighborsClassifier** (n_neighbors=5)
    3. **DecisionTreeClassifier** (max_depth=None)
    4. **RandomForestClassifier** (n_estimators=100)
    5. **XGBClassifier** (use_label_encoder=False, eval_metric='logloss')

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
models = {
    'Logistic Regression': LogisticRegression(solver='liblinear'),
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}
hist = {}
for name, model in models.items():
    pipeline = Pipeline([('imp', median_imp),('scaler', StandardScaler()),('clf',
model)])
    pipeline.fit(X_train, y_train)
    hist[name] = pipeline
```

7.2 Model Testing

- **Predictions**:

```
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
results = {}
for name, pipe in hist.items():
    y_pred = pipe.predict(X_test)
    y_proba = pipe.predict_proba(X_test)[:,1]
    results[name] = {
        'accuracy': accuracy_score(y_test, y_pred),
        'roc_auc': roc_auc_score(y_test, y_proba),
        'report': classification_report(y_test, y_pred, output_dict=True)
    }
```

7.3 Evaluation & Best Model Selection

Model	Accuracy	ROC AUC
Logistic Regression	0.7922	0.8565
KNN (k=5)	0.7208	0.7973
Decision Tree	0.7532	0.7450
Random Forest	0.8117	0.8632
XGBoost	**0.9883**	**0.9941**

- **Confusion Matrix** & **ROC Curves** plotted for top two models.
- **Conclusion**: XGBoost outperforms all others, achieving ~98.8% accuracy and 0.994 AUC on the test set.