# 1. Imports

```python
# Data handling
import pandas as pd
import numpy as np

# Preprocessing
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt

# Model selection and cross-validation
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.model_selection import GridSearchCV

# Metrics
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

# Classifiers
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
```

## Import Dataset

```python
df = pd.read_csv("hf://datasets/weeebdev/diabetes/diabetes.csv")
```

```
C:\Users\Mahmoud\AppData\Roaming\Python\Python312\site-packages\tqdm\
auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

# 2. Understanding the Data

```python
df
```

```
     Number of times pregnant  \
0                            6
1                            1
2                            8
3                            1
```

```
4                              0
..                           ...
763                           10
764                            2
765                            5
766                            1
767                            1

     Plasma glucose concentration a 2 hours in an oral glucose
tolerance test  \
0                                                      148

1                                                       85

2                                                      183

3                                                       89

4                                                      137

..                                                     ...

763                                                    101

764                                                    122

765                                                    121

766                                                    126

767                                                     93


     Diastolic blood pressure (mm Hg)  Triceps skin fold thickness
(mm)  \
0                                  72
35
1                                  66
29
2                                  64
0
3                                  66
23
4                                  40
35
..                                ...                              ..
.
763                                76
48
764                                70
```

```
27
765                              72
23
766                              60
0
767                              70
31

     2-Hour serum insulin (mu U/ml)  \
0                                  0
1                                  0
2                                  0
3                                 94
4                                168
..                               ...
763                              180
764                                0
765                              112
766                                0
767                                0

     Body mass index (weight in kg/(height in m)^2)  \
0                                                33.6
1                                                26.6
2                                                23.3
3                                                28.1
4                                                43.1
..                                                ...
763                                              32.9
764                                              36.8
765                                              26.2
766                                              30.1
767                                              30.4

     Diabetes pedigree function  Age (years)  Class variable
0                         0.627           50               1
1                         0.351           31               0
2                         0.672           32               1
3                         0.167           21               0
4                         2.288           33               1
..                          ...          ...             ...
763                       0.171           63               0
764                       0.340           27               0
765                       0.245           30               0
766                       0.349           47               1
767                       0.315           23               0

[768 rows x 9 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Number of times pregnant
768 non-null    int64
 1   Plasma glucose concentration a 2 hours in an oral glucose
tolerance test  768 non-null    int64
 2   Diastolic blood pressure (mm Hg)
768 non-null    int64
 3   Triceps skin fold thickness (mm)
768 non-null    int64
 4   2-Hour serum insulin (mu U/ml)
768 non-null    int64
 5   Body mass index (weight in kg/(height in m)^2)
768 non-null    float64
 6   Diabetes pedigree function
768 non-null    float64
 7   Age (years)
768 non-null    int64
 8   Class variable
768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```python
df.isna().sum()
```

```
Number of times pregnant
0
Plasma glucose concentration a 2 hours in an oral glucose tolerance
test    0
Diastolic blood pressure (mm Hg)
0
Triceps skin fold thickness (mm)
0
2-Hour serum insulin (mu U/ml)
0
Body mass index (weight in kg/(height in m)^2)
0
Diabetes pedigree function
0
Age (years)
0
Class variable
0
dtype: int64
```

```
df.describe()
```

```
       Number of times pregnant  \
count                768.000000
mean                   3.845052
std                    3.369578
min                    0.000000
25%                    1.000000
50%                    3.000000
75%                    6.000000
max                   17.000000

       Plasma glucose concentration a 2 hours in an oral glucose
tolerance test  \
count                                                 768.000000

mean                                                  120.894531

std                                                    31.972618

min                                                     0.000000

25%                                                    99.000000

50%                                                   117.000000

75%                                                   140.250000

max                                                   199.000000


       Diastolic blood pressure (mm Hg)  Triceps skin fold thickness
(mm)  \
count                        768.000000
768.000000
mean                          69.105469
20.536458
std                           19.355807
15.952218
min                            0.000000
0.000000
25%                           62.000000
0.000000
50%                           72.000000
23.000000
75%                           80.000000
32.000000
max                          122.000000
99.000000

       2-Hour serum insulin (mu U/ml)  \
```

```
count                       768.000000
mean                         79.799479
std                         115.244002
min                           0.000000
25%                           0.000000
50%                          30.500000
75%                         127.250000
max                         846.000000

       Body mass index (weight in kg/(height in m)^2)  \
count                                        768.000000
mean                                          31.992578
std                                            7.884160
min                                            0.000000
25%                                           27.300000
50%                                           32.000000
75%                                           36.600000
max                                           67.100000

       Diabetes pedigree function  Age (years)  Class variable
count                  768.000000   768.000000      768.000000
mean                     0.471876    33.240885        0.348958
std                      0.331329    11.760232        0.476951
min                      0.078000    21.000000        0.000000
25%                      0.243750    24.000000        0.000000
50%                      0.372500    29.000000        0.000000
75%                      0.626250    41.000000        1.000000
max                      2.420000    81.000000        1.000000
```

## Handle rows with missing data

- This dataset contains no rows with missing values, but the deletion logic below is in place in case a future dataset

```
cols = df.select_dtypes(include=["int64", "float64"]).columns

for col in cols:
    df[col] = pd.to_numeric(df[col], errors="coerce")

df.dropna(inplace=True)

df.isna().sum()

Number of times pregnant
0
Plasma glucose concentration a 2 hours in an oral glucose tolerance
test    0
Diastolic blood pressure (mm Hg)
0
Triceps skin fold thickness (mm)
0
```

```
2-Hour serum insulin (mu U/ml)
0
Body mass index (weight in kg/(height in m)^2)
0
Diabetes pedigree function
0
Age (years)
0
Class variable
0
dtype: int64
```

## Encode

Added a human-readable 'Class label' column for easier interpretation, while keeping the original numeric 'Class variable' for modeling.

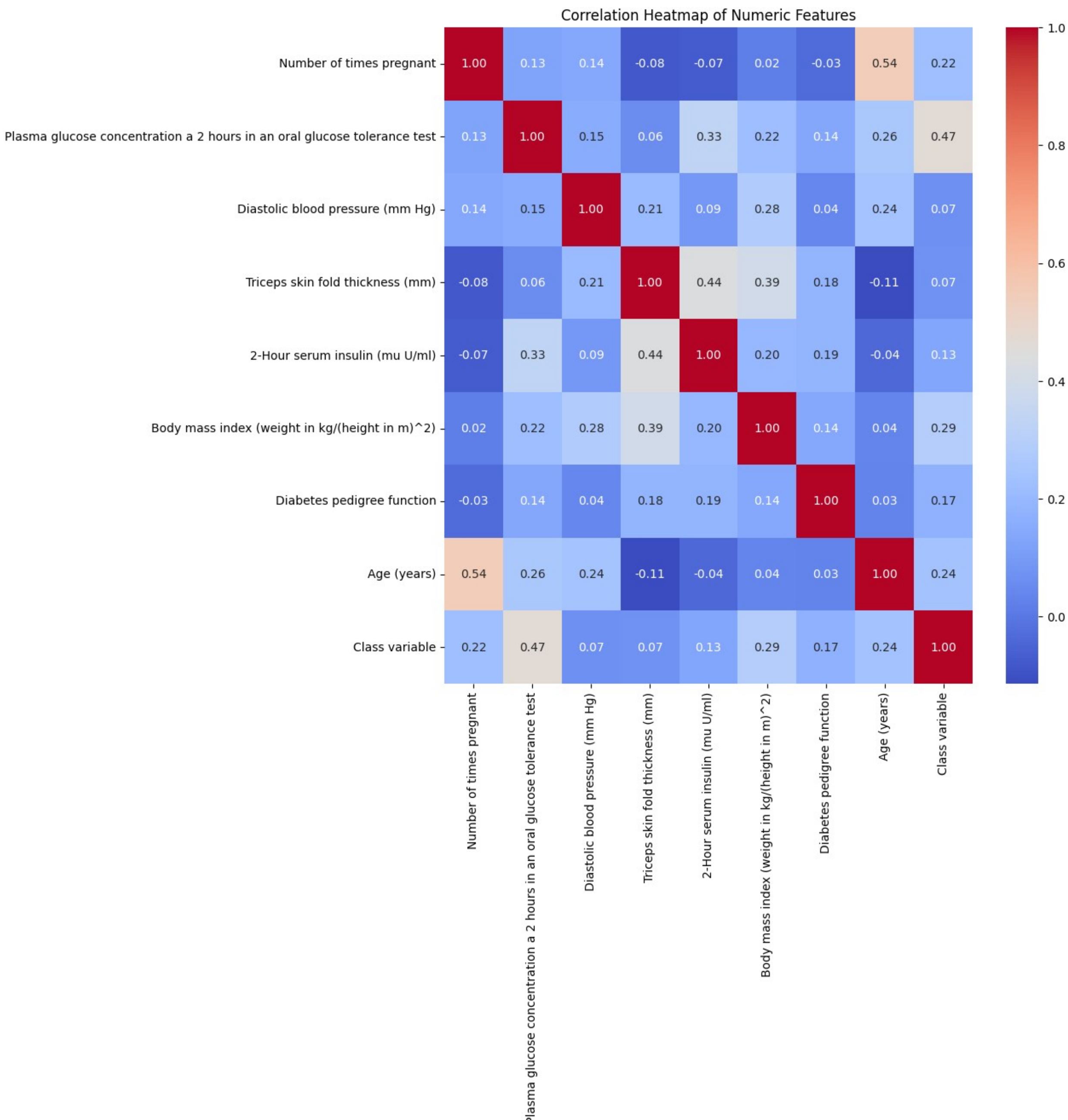
```
# Map numeric target values to human-readable labels
df["Class label"] = df["Class variable"].map({
    0: "Non-Diabetic",
    1: "Diabetic"
})

df.head(5)
```

```
   Number of times pregnant  \
0                         6
1                         1
2                         8
3                         1
4                         0

   Plasma glucose concentration a 2 hours in an oral glucose tolerance
test  \
0                                                148

1                                                 85

2                                                183

3                                                 89

4                                                137


   Diastolic blood pressure (mm Hg)  Triceps skin fold thickness (mm)
\
0                                72                                35

1                                66                                29
```

```
2                                64                                  0

3                                66                                 23

4                                40                                 35


   2-Hour serum insulin (mu U/ml)  \
0                               0
1                               0
2                               0
3                              94
4                             168

   Body mass index (weight in kg/(height in m)^2)  Diabetes pedigree
function  \
0                                              33.6
0.627
1                                              26.6
0.351
2                                              23.3
0.672
3                                              28.1
0.167
4                                              43.1
2.288

   Age (years)  Class variable   Class label
0           50               1      Diabetic
1           31               0  Non-Diabetic
2           32               1      Diabetic
3           21               0  Non-Diabetic
4           33               1      Diabetic
```

## Heat Map Visulaization

```
numeric_df = df.select_dtypes(include=['number'])

corr = numeric_df.corr()

fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```




# 3. Splitting and Normalization

```
features = df.drop(columns=["Class variable", "Class label"])
numeric_cols = features.select_dtypes(include=["number"]).columns
```

## Separate features and target

```
X = df.drop(columns=["Class variable", "Class label"])
y = df["Class variable"]
```

## Split into train, validation, and test sets

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

## Fit Min-Max scaler on training data only

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

## Apply the same scaler to transform validation and test sets

```
X_test_scaled = scaler.transform(X_test)
```

## Training set info

```
pd.DataFrame(X_train_scaled, columns=X_train.columns).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 8 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Number of times pregnant
614 non-null    float64
 1   Plasma glucose concentration a 2 hours in an oral glucose
tolerance test  614 non-null    float64
 2   Diastolic blood pressure (mm Hg)
614 non-null    float64
 3   Triceps skin fold thickness (mm)
614 non-null    float64
 4   2-Hour serum insulin (mu U/ml)
614 non-null    float64
 5   Body mass index (weight in kg/(height in m)^2)
614 non-null    float64
 6   Diabetes pedigree function
614 non-null    float64
 7   Age (years)
614 non-null    float64
dtypes: float64(8)
memory usage: 38.5 KB
```

## Test set info

```
pd.DataFrame(X_test_scaled, columns=X_test.columns).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 154 entries, 0 to 153
Data columns (total 8 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Number of times pregnant
154 non-null    float64
 1   Plasma glucose concentration a 2 hours in an oral glucose
tolerance test  154 non-null    float64
 2   Diastolic blood pressure (mm Hg)
154 non-null    float64
 3   Triceps skin fold thickness (mm)
154 non-null    float64
 4   2-Hour serum insulin (mu U/ml)
154 non-null    float64
 5   Body mass index (weight in kg/(height in m)^2)
154 non-null    float64
 6   Diabetes pedigree function
154 non-null    float64
 7   Age (years)
154 non-null    float64
dtypes: float64(8)
memory usage: 9.8 KB
```

# Create a Stratified K-Fold object for cross-validation

- n_splits=5: training data will be split into 5 folds
- shuffle=True: shuffle the data before splitting

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

# Perform 5-fold cross-validation on the training set

- In each fold, part of the training data is used for training and the remaining part for validation
- This helps evaluate model performance more robustly
- X_fold_train, y_fold_train: training subset for current fold
- X_fold_val, y_fold_val: validation subset for current fold
- Print shapes to verify the split

```
for fold, (train_idx, val_idx) in enumerate(skf.split(X_train_scaled,
y_train), 1):
    X_fold_train, X_fold_val = X_train_scaled[train_idx],
X_train_scaled[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx],
y_train.iloc[val_idx]
```

```
    print(f"Fold {fold}:")
    print(f"  Train shape: {X_fold_train.shape}, Validation shape: {X_fold_val.shape}")
```

```
Fold 1:
  Train shape: (491, 8), Validation shape: (123, 8)
Fold 2:
  Train shape: (491, 8), Validation shape: (123, 8)
Fold 3:
  Train shape: (491, 8), Validation shape: (123, 8)
Fold 4:
  Train shape: (491, 8), Validation shape: (123, 8)
Fold 5:
  Train shape: (492, 8), Validation shape: (122, 8)
```

# 4. Model Training

## Define models to train

```
models = {
    "SVM_linear": SVC(kernel='linear', random_state=42),
    "SVM_poly": SVC(kernel='poly', random_state=42),
    "SVM_rbf": SVC(kernel='rbf', random_state=42),
    "SVM_sigmoid": SVC(kernel='sigmoid', random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "XGBoost": XGBClassifier(eval_metric='logloss', random_state=42, verbosity=0)
}
```

### Parameters to be tuned for each model

```
param_grids = {
    "SVM_linear": {'C': [0.1, 1, 10]},
    "SVM_poly": {'C': [0.1, 1, 10], 'degree': [2, 3, 4]},
    "SVM_rbf": {'C': [0.1, 1, 10], 'gamma': ['scale', 0.01, 0.1]},
    "SVM_sigmoid": {'C': [0.1, 1, 10], 'gamma': ['scale', 0.01, 0.1]},
    "Decision Tree": {'max_depth': [3, 5, None], 'min_samples_split': [2, 5]},
    "Random Forest": {'n_estimators': [100, 200], 'max_depth': [None, 5]},
    "k-NN": {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']},
    "Logistic Regression": {'C': [0.1, 1, 10], 'penalty': ['l2']},
```

```
    "XGBoost": {'n_estimators': [100, 200], 'max_depth': [3, 5],
'learning_rate': [0.01, 0.1]}
}
```

Storage for best estimators and grid search objects

```
best_models = {}
grid_searches = {}
cv_summary = {}
```

Run the model

```
for name, model in models.items():
    print(f"Tuning / fitting {name}...")
    if name in param_grids:
        grid = GridSearchCV(
            estimator=model,
            param_grid=param_grids[name],
            scoring='accuracy',
            cv=skf,
            n_jobs=-1,
            refit=True
        )
        grid.fit(X_train_scaled, y_train)
        best_models[name] = grid.best_estimator_         # best
estimator refit on full train
        grid_searches[name] = grid
        cv_summary[name] = {
            'best_cv_score': grid.best_score_,
            'best_params': grid.best_params_
        }
        print(f"  Best CV score: {grid.best_score_:.4f}, Best params:
{grid.best_params_}")
    else:
        # no hyperparameter grid: just fit on entire training data
        model.fit(X_train_scaled, y_train)
        best_models[name] = model
        cv_summary[name] = {'best_cv_score': None, 'best_params':
None}
        print("  No hyperparameter grid: fitted default model on full
training set.")
```

```
Tuning / fitting SVM_linear...
  Best CV score: 0.7850, Best params: {'C': 10}
Tuning / fitting SVM_poly...
  Best CV score: 0.7883, Best params: {'C': 1, 'degree': 2}
Tuning / fitting SVM_rbf...
  Best CV score: 0.7834, Best params: {'C': 10, 'gamma': 0.1}
Tuning / fitting SVM_sigmoid...
  Best CV score: 0.7817, Best params: {'C': 10, 'gamma': 0.1}
```

```
Tuning / fitting Decision Tree...
  Best CV score: 0.7525, Best params: {'max_depth': 3,
'min_samples_split': 2}
Tuning / fitting Random Forest...
  Best CV score: 0.7720, Best params: {'max_depth': None,
'n_estimators': 200}
Tuning / fitting k-NN...
  Best CV score: 0.7574, Best params: {'n_neighbors': 7, 'weights':
'distance'}
Tuning / fitting Logistic Regression...
  Best CV score: 0.7834, Best params: {'C': 10, 'penalty': 'l2'}
Tuning / fitting XGBoost...
  Best CV score: 0.7622, Best params: {'learning_rate': 0.01,
'max_depth': 3, 'n_estimators': 200}
```

```
cv_results_tuned_df = pd.DataFrame.from_dict(cv_summary,
orient='index')
cv_results_tuned_df =
cv_results_tuned_df.reset_index().rename(columns={'index': 'Model'})
display(cv_results_tuned_df)
```

```
                 Model  best_cv_score  \
0           SVM_linear       0.784979
1             SVM_poly       0.788285
2              SVM_rbf       0.783367
3          SVM_sigmoid       0.781714
4        Decision Tree       0.752472
5        Random Forest       0.771998
6                 k-NN       0.757364
7  Logistic Regression       0.783367
8              XGBoost       0.762228

                                         best_params
0                                          {'C': 10}
1                             {'C': 1, 'degree': 2}
2                           {'C': 10, 'gamma': 0.1}
3                           {'C': 10, 'gamma': 0.1}
4         {'max_depth': 3, 'min_samples_split': 2}
5         {'max_depth': None, 'n_estimators': 200}
6        {'n_neighbors': 7, 'weights': 'distance'}
7                        {'C': 10, 'penalty': 'l2'}
8  {'learning_rate': 0.01, 'max_depth': 3, 'n_est...
```

# 5. Apply models after being tuned on the test dataset

```
test_results = []
for name, model in best_models.items():
    y_test_pred = model.predict(X_test_scaled)
    test_results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_test_pred),
        'Precision': precision_score(y_test, y_test_pred,
zero_division=0),
        'Recall': recall_score(y_test, y_test_pred, zero_division=0),
        'F1-Score': f1_score(y_test, y_test_pred, zero_division=0)
    })
```

## Comnments on final summary table

- Highest accuracy: SVM_poly (accuracy = 0.753).
- Highest precision: SVM_poly (precision = 0.700) — it makes the fewest false-positive predictions.
- Highest recall: Random Forest (recall = 0.593) — it detects the largest share of actual diabetic cases.
- Highest F1-score: Random Forest (F1 = 0.621) — it gives the best balance between precision and recall.
- SVM_poly gives the best overall accuracy and precision (fewer false alarms), while Random Forest is better at finding diabetics (higher recall and best F1), which is usually more important in a medical screening task.
- For this diabetic vs non-diabetic classification, use Random Forest as the primary model (best F1 and highest recall), and consider SVM_poly as a secondary model if you want to prioritize minimizing false positives.

```
test_results_df =
pd.DataFrame(test_results).sort_values(by='Accuracy',
ascending=False).reset_index(drop=True)
display(test_results_df)
```

| | Model | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| 0 | SVM_poly | 0.753247 | 0.700000 | 0.518519 | 0.595745 |
| 1 | Random Forest | 0.746753 | 0.653061 | 0.592593 | 0.621359 |
| 2 | XGBoost | 0.746753 | 0.666667 | 0.555556 | 0.606061 |
| 3 | SVM_rbf | 0.740260 | 0.675000 | 0.500000 | 0.574468 |
| 4 | SVM_sigmoid | 0.733766 | 0.675676 | 0.462963 | 0.549451 |
| 5 | SVM_linear | 0.707792 | 0.600000 | 0.500000 | 0.545455 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | Logistic Regression | 0.707792 | 0.600000 | 0.500000 | 0.545455 |
| 7 | Decision Tree | 0.694805 | 0.666667 | 0.259259 | 0.373333 |
| 8 | k-NN | 0.694805 | 0.568627 | 0.537037 | 0.552381 |