Diabetes Prediction — Using a Local CSV

This notebook uses a local CSV file `pima_diabetes_dataset.csv` (placed in the same folder). It performs a full ML workflow with explanations:

- Load local CSV
- Inspect data and handle missing values
- Preprocess (replace invalid zeros, impute using median, scale features)
- Train 5 classifiers (Logistic Regression, Decision Tree, Random Forest, KNN, SVC)
- Evaluate models (accuracy, ROC-AUC, classification report, confusion matrix)
- Plot model comparison and give short assignment notes

Run the notebook cells top-to-bottom. The notebook is written for beginners with comments and brief explanations.

1) Imports and helper functions

```
# Imports
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt

print('Libraries imported')
```

```
Libraries imported
```

2) Load the local CSV dataset

```
# Load local CSV (ensure the file is in the same folder as this notebook)
csv_path = 'pima_diabetes_dataset.csv'  # change path if needed
df = pd.read_csv(csv_path)
print('Loaded shape:', df.shape)
df.head()
```

```
Loaded shape: (10, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3) Basic inspection

```
print(df.info())
print('\nSummary statistics:\n', df.describe().T)
print('\nTarget value counts:\n', df['Outcome'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
```

```
 0   Pregnancies               10 non-null     int64  
 1   Glucose                   10 non-null     int64  
 2   BloodPressure             10 non-null     int64  
 3   SkinThickness             10 non-null     int64  
 4   Insulin                   10 non-null     int64  
 5   BMI                       10 non-null     float64
 6   DiabetesPedigreeFunction  10 non-null     float64
 7   Age                       10 non-null     int64  
 8   Outcome                   10 non-null     int64  
dtypes: float64(2), int64(7)
memory usage: 852.0 bytes
None

Summary statistics:
                          count      mean         std     min      25%  \
Pregnancies               10.0    4.4000    3.502380   0.000   1.2500   
Glucose                   10.0  127.3000   40.075068  78.000  95.5000   
BloodPressure             10.0   59.8000   25.654976   0.000  53.5000   
SkinThickness             10.0   19.9000   17.978073   0.000   0.0000   
Insulin                   10.0   89.3000  169.937276   0.000   0.0000   
BMI                       10.0   27.7100   11.259016   0.000  25.8500   
DiabetesPedigreeFunction  10.0    0.5078    0.654114   0.134   0.1755   
Age                       10.0   35.9000   11.873874  21.000  29.2500   
Outcome                   10.0    0.6000    0.516398   0.000   0.0000   

                             50%      75%      max  
Pregnancies                4.00    7.500   10.000  
Glucose                  120.50  145.250  197.000  
BloodPressure             66.00   71.500   96.000  
SkinThickness             26.00   34.250   45.000  
Insulin                    0.00   92.500  543.000  
BMI                       29.30   32.950   43.100  
DiabetesPedigreeFunction   0.24    0.558    2.288  
Age                       31.50   45.750   54.000  
Outcome                    1.00    1.000    1.000  

Target value counts:
 Outcome
1    6
0    4
Name: count, dtype: int64
```

4) Robust target handling (ensure y is numeric 0/1)

```python
# Robust target mapping (handles string labels if present)
y = df['Outcome']
import numpy as np
if not np.issubdtype(pd.Series(y).dtype, np.number):
    print('Non-numeric target detected. Examples:', pd.Series(y).unique()[:20])
    mapping = {
        'tested_negative': 0, 'tested_positive': 1,
        'negative': 0, 'positive': 1,
        'No': 0, 'Yes': 1,
        'no': 0, 'yes': 1,
        'NEGATIVE': 0, 'POSITIVE': 1,
        '0': 0, '1': 1
    }
    y_mapped = pd.Series(y).map(mapping)
    if y_mapped.isna().any():
        coerced = pd.to_numeric(pd.Series(y), errors='coerce')
        if coerced.isna().any():
            missing = pd.Series(y)[y_mapped.isna()].unique()
            raise ValueError(f'Cannot automatically map these target labels: {missing}. Add them to mapping.')
        else:
            y = coerced.astype(int)
    else:
        y = y_mapped.astype(int).values
else:
    y = pd.Series(y).astype(int).values

print('Final target unique values:', np.unique(y))
```

```
Final target unique values: [0 1]
```

5) Preprocessing: replace invalid zeros, impute, and scale

```
# Features
X = df.drop(columns=['Outcome']).copy()

# Columns where 0 is medically invalid and likely represents missing values
cols_with_zero_invalid = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# Replace zeros with NaN in those columns
X[cols_with_zero_invalid] = X[cols_with_zero_invalid].replace(0, np.nan)
print('Missing counts after replacing zeros:')
print(X[cols_with_zero_invalid].isna().sum())

# Impute missing values with median
imputer = SimpleImputer(strategy='median')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Scale features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed), columns=X_imputed.columns)

print('\nAfter imputation and scaling — feature summary:')
print(X_scaled.describe().T)
```

```
Missing counts after replacing zeros:
Glucose          0
BloodPressure    1
SkinThickness    4
Insulin          6
BMI              1
dtype: int64

After imputation and scaling — feature summary:
                          count          mean       std       min       25%  \
Pregnancies                10.0 -8.881784e-17  1.054093 -1.324244 -0.948039
Glucose                    10.0  8.326673e-17  1.054093 -1.296735 -0.836434
BloodPressure              10.0 -3.552714e-16  1.054093 -1.890349 -0.136048
SkinThickness              10.0  5.780099e-16  1.054093 -1.985548 -0.178314
Insulin                    10.0 -5.551115e-17  1.054093 -0.630258 -0.291071
BMI                        10.0 -1.859624e-16  1.054093 -1.390562 -0.705533
DiabetesPedigreeFunction   10.0 -2.498002e-16  1.054093 -0.602372 -0.535495
Age                        10.0  1.110223e-16  1.054093 -1.322734 -0.590348

                               50%       75%       max
Pregnancies              -0.120386  0.932990  1.685402
Glucose                  -0.178860  0.472138  1.833316
BloodPressure            -0.028642  0.365181  2.119483
SkinThickness             0.038554  0.255422  2.255428
Insulin                  -0.291071 -0.291071  2.958823
BMI                      -0.048465  0.408221  2.300206
DiabetesPedigreeFunction -0.431555  0.080896  2.868759
Age                      -0.390606  0.874425  1.606811
```

6) Train/Test split

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.30, random_state=42, stratify=y)
print('Train shape:', X_train.shape)
print('Test shape:', X_test.shape)
print('Train class distribution:\n', pd.Series(y_train).value_counts())
```

```
Train shape: (7, 8)
Test shape: (3, 8)
Train class distribution:
 1    4
0    3
Name: count, dtype: int64
```

7) Train 5 models and evaluate

```
models = {
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
    'DecisionTree': DecisionTreeClassifier(random_state=42),
    'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'SVC': SVC(probability=True, random_state=42)
}
```

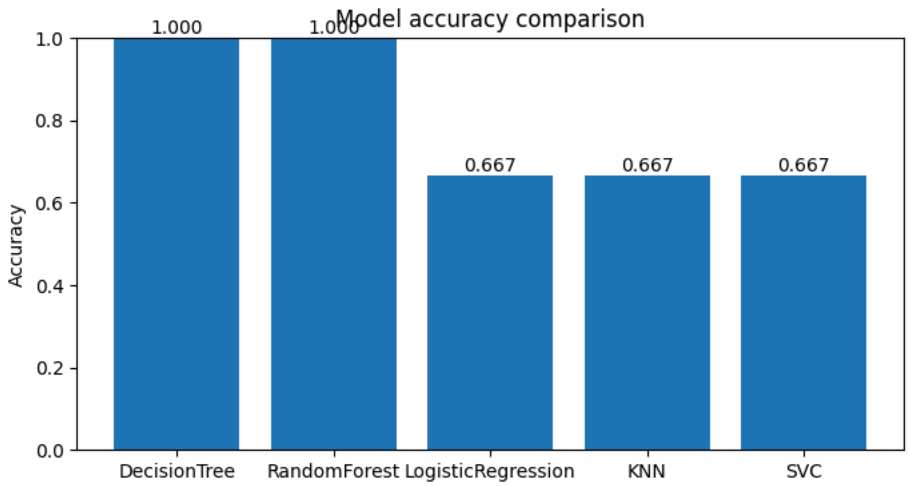
```
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    try:
        roc = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])
    except Exception:
        roc = None
    results.append({'model': name, 'accuracy': acc, 'roc_auc': roc})
    print(f'--- {name} ---')
    print('Accuracy:', round(acc, 4))
    print('Classification report:\n', classification_report(y_test, y_pred))
    print('Confusion matrix:\n', confusion_matrix(y_test, y_pred))
    print('\n')

results_df = pd.DataFrame(results).sort_values('accuracy', ascending=False).reset_index(drop=True)
print('Summary results:')
results_df
```

```
--- LogisticRegression ---
Accuracy: 0.6667
Classification report:
              precision    recall  f1-score   support
```

8) Plot model accuracies

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.bar(results_df['model'], results_df['accuracy'])
plt.title('Model accuracy comparison')
plt.ylabel('Accuracy')
plt.ylim(0,1)
for i, v in enumerate(results_df['accuracy']):
    plt.text(i, v+0.01, f'{v:.3f}', ha='center')
plt.show()
```




```
MetricWarning: Precision is ill-define
MetricWarning: Precision is ill-define
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
--- RandomForest ---
```

9) Assignment notes and suggestions