Лабораторна робота #1

Базові алгоритми класифікації з використанням бібліотеки Scikit-learnt

Завдання роботи

1. [+] Завантажити дані, вивести назви колонок і розмір датасета
2. [+] Опрацювати пропуски (по можливості заповнити їх або видалити)
3. [+] Візуалізувати дані: побудувати графік (heatmap), що відображає кореляції ознак між собою і з цільовою змінною (розміткою); побудувати гістограми розподілу ознак і boxplot-и ознак відносно цільової змінної (якщо ознак занадто багато обмежитися декількома)
4. [+] Нормалізувати дані
5. Провести навчання наступних класифікаторів:

- [+] kNN
- [+] дерево ухвалення рішень
- [+] SVM
- [+] Random Forest
- [+] AdaBoost

Підібрати оптимальні параметри • для kNN • для SVM за допомогою GridSearch підібрати оптимальні «C» і «gamma» Серед обраних оптимальних моделей кожного класу вибрати найкращу.

[+] Відобразити sklearn.metrics.classification_report і sklearn.metrics.confusion_matrix

Хід роботи

Для роботи я вирішив обрати наступний датасет - https://www.kaggle.com/datasets/erdemtaha/cancer-data, оскільки вже використовував його у попередніх курсах із машинного навчання.

In [133…
```
import pandas as pd

data = pd.read_csv('data/Cancer_Data.csv', delimiter=',')
```

In [134…
```
print("Column Names:")
print(data.columns.tolist())
print("\nDataset Size:")
print(data.shape)
```

```
Column Names:
['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 's
moothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symme
try_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'are
a_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symm
etry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_wors
t', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'conca
ve points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32']

Dataset Size:
(569, 33)
```

In [135…

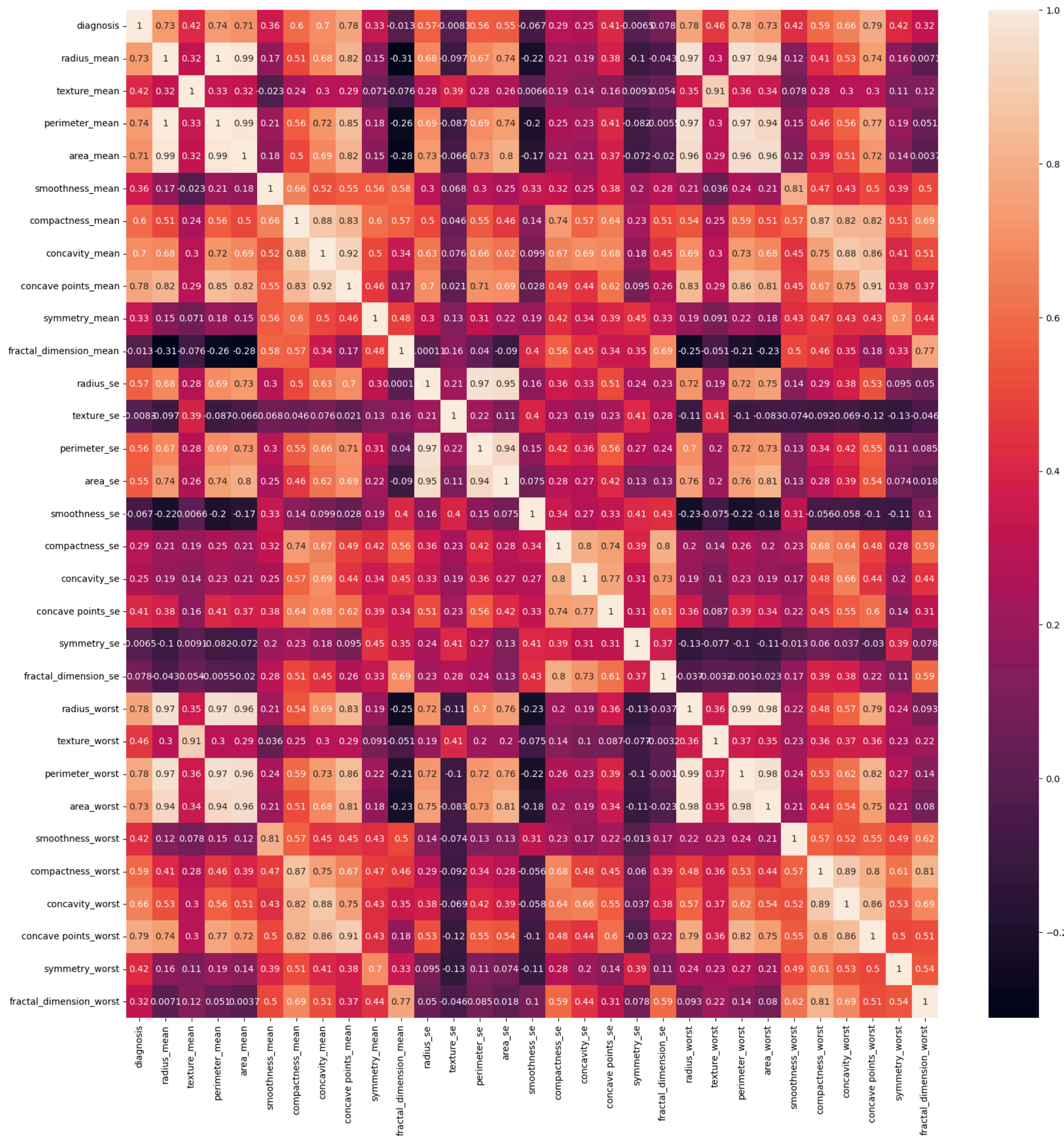
```
data["diagnosis"] = data["diagnosis"].replace({"B": 0, "M": 1})
print("\nSurv_status: \n0 = Benign cancer \n1 = Malignant cancer\n")

y_column = 'diagnosis'
data = data.dropna(subset=[y_column])

data = data.drop(columns=['id', "Unnamed: 32"], errors='ignore')

X_data = data.drop(y_column, axis=1)
X_column = X_data.columns

print("\nProcessed Data Size:")
print(data.shape)
```

```
Surv_status:
0 = Benign cancer
1 = Malignant cancer


Processed Data Size:
(569, 31)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_5344\319634148.py:1: FutureWarning: Downc
asting behavior in `replace` is deprecated and will be removed in a future version.
To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', Tru
e)`
  data["diagnosis"] = data["diagnosis"].replace({"B": 0, "M": 1})
```

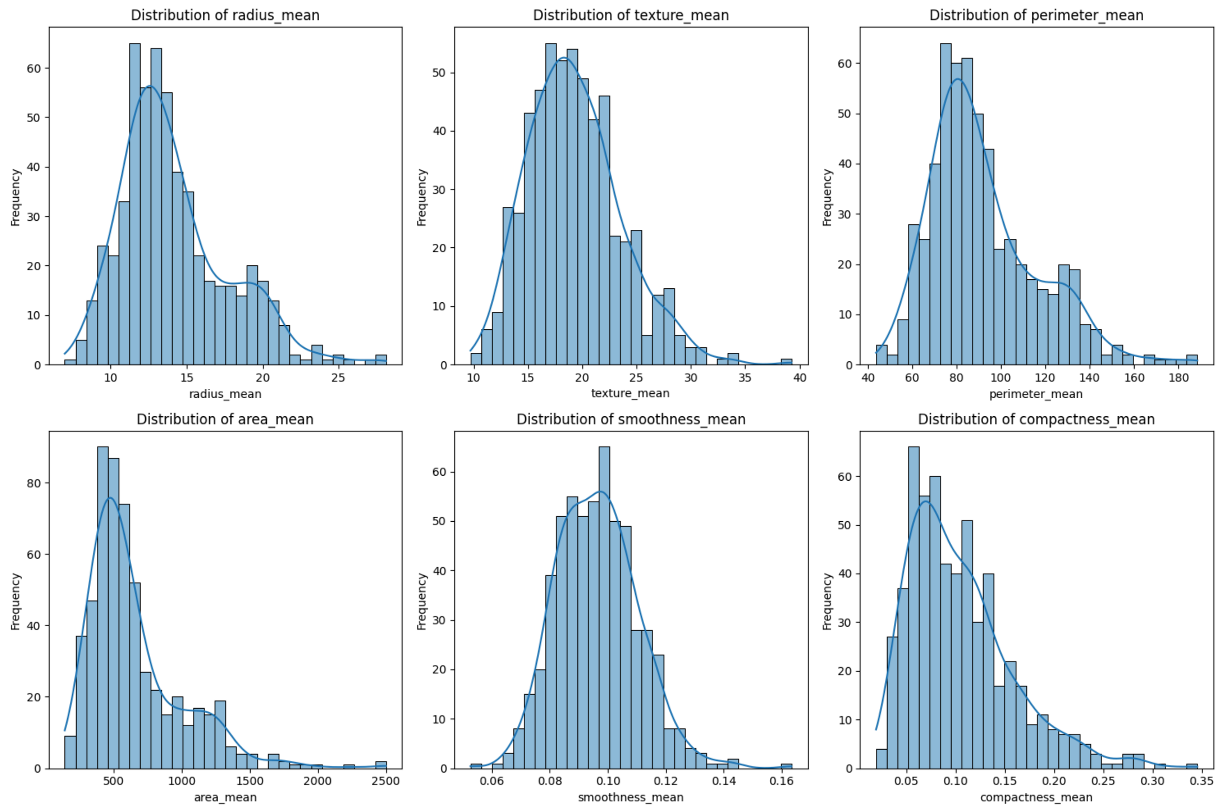
In [136…

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

plt.figure(figsize=(20, 20))
sns.heatmap(data.corr(), annot=True)
plt.show()
```



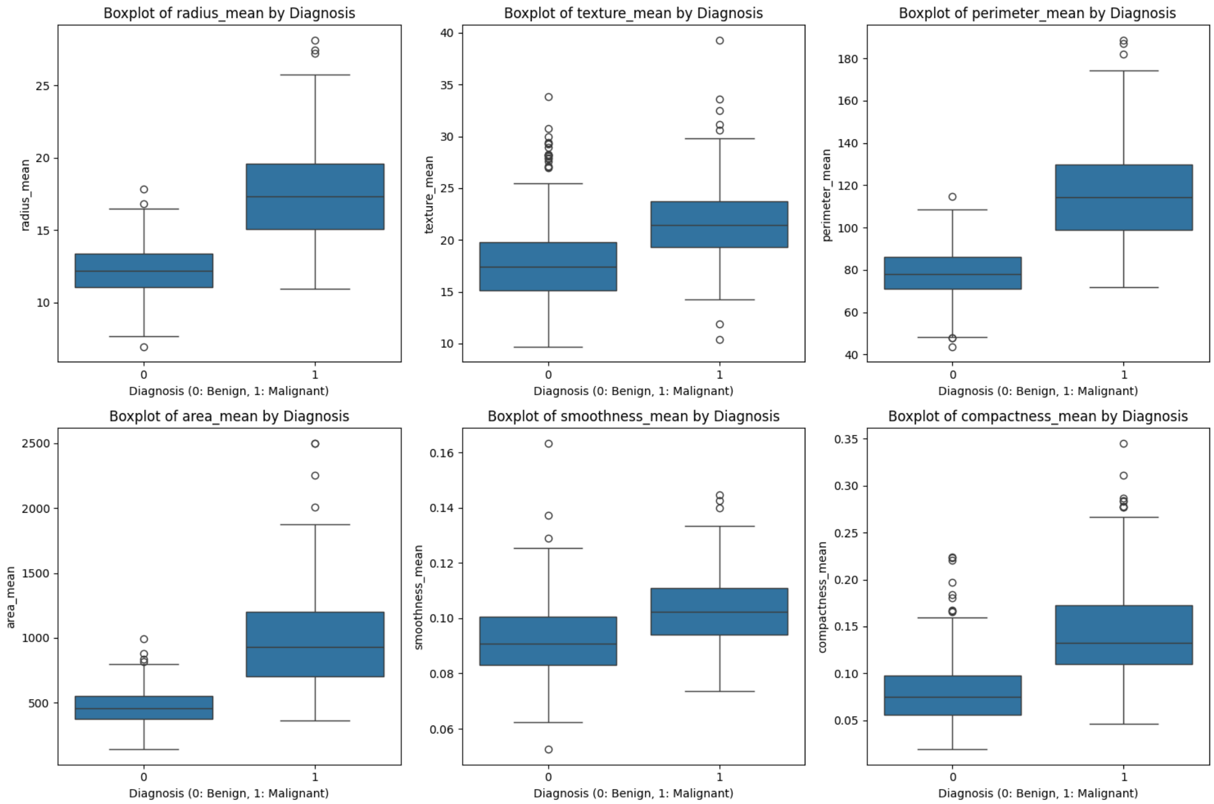
In [137…

```
features_to_plot = data.select_dtypes(include=['float64', 'int64']).columns[1:7]  #

plt.figure(figsize=(15, 10))
for feature in features_to_plot:
    plt.subplot(2, 3, list(features_to_plot).index(feature) + 1)
    sns.histplot(data[feature], bins=30, kde=True)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



In [138…

```
features_to_boxplot = data.select_dtypes(include=['float64', 'int64']).columns[1:7]

# Plot boxplots
plt.figure(figsize=(15, 10))
for feature in features_to_boxplot:
    plt.subplot(2, 3, list(features_to_boxplot).index(feature) + 1)
    sns.boxplot(x='diagnosis', y=feature, data=data)
    plt.title(f'Boxplot of {feature} by Diagnosis')
    plt.xlabel('Diagnosis (0: Benign, 1: Malignant)')
    plt.ylabel(feature)

plt.tight_layout()
plt.show()
```



In [139…

```
from sklearn.preprocessing import StandardScaler

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_data)

# Update the dataset with scaled features
data[X_column] = X_scaled

# Optionally, if you want to view the updated DataFrame
print("\nUpdated Data Sample:")
print(data.head())
```

```
Updated Data Sample:
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1     1.097064     -2.073335        1.269934   0.984375
1          1     1.829821     -0.353632        1.685955   1.908708
2          1     1.579888      0.456187        1.566503   1.558884
3          1    -0.768909      0.253732       -0.592687  -0.764464
4          1     1.750297     -1.151816        1.776573   1.826229

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0         1.568466          3.283515        2.652874             2.532475
1        -0.826962         -0.487072       -0.023846             0.548144
2         0.942210          1.052926        1.363478             2.037231
3         3.283553          3.402909        1.915897             1.451707
4         0.280372          0.539340        1.371011             1.428493

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0       2.217515  ...      1.886690      -1.359293         2.303601
1       0.001392  ...      1.805927      -0.369203         1.535126
2       0.939685  ...      1.511870      -0.023974         1.347475
3       2.867383  ...     -0.281464       0.133984        -0.249939
4      -0.009560  ...      1.298575      -1.466770         1.338539

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0    2.001237          1.307686           2.616665         2.109526
1    1.890489         -0.375612          -0.430444        -0.146749
2    1.456285          0.527407           1.082932         0.854974
3   -0.550021          3.394275           3.893397         1.989588
4    1.220724          0.220556          -0.313395         0.613179

   concave points_worst  symmetry_worst  fractal_dimension_worst
0              2.296076        2.750622                 1.937015
1              1.087084       -0.243890                 0.281190
2              1.955000        1.152255                 0.201391
3              2.175786        6.046041                 4.935010
4              0.729259       -0.868353                -0.397100

[5 rows x 31 columns]
```

Шаблон виводу метрик

Написав функцію для виводів, щоб не повторювати код.

In [140…

```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

def print_evaluation_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    print(f"Accuracy: {accuracy:.2f}")

    print("Classification Report:")
    print(classification_report(y_true, y_pred))

    print("Confusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
```

kNN

In [141…

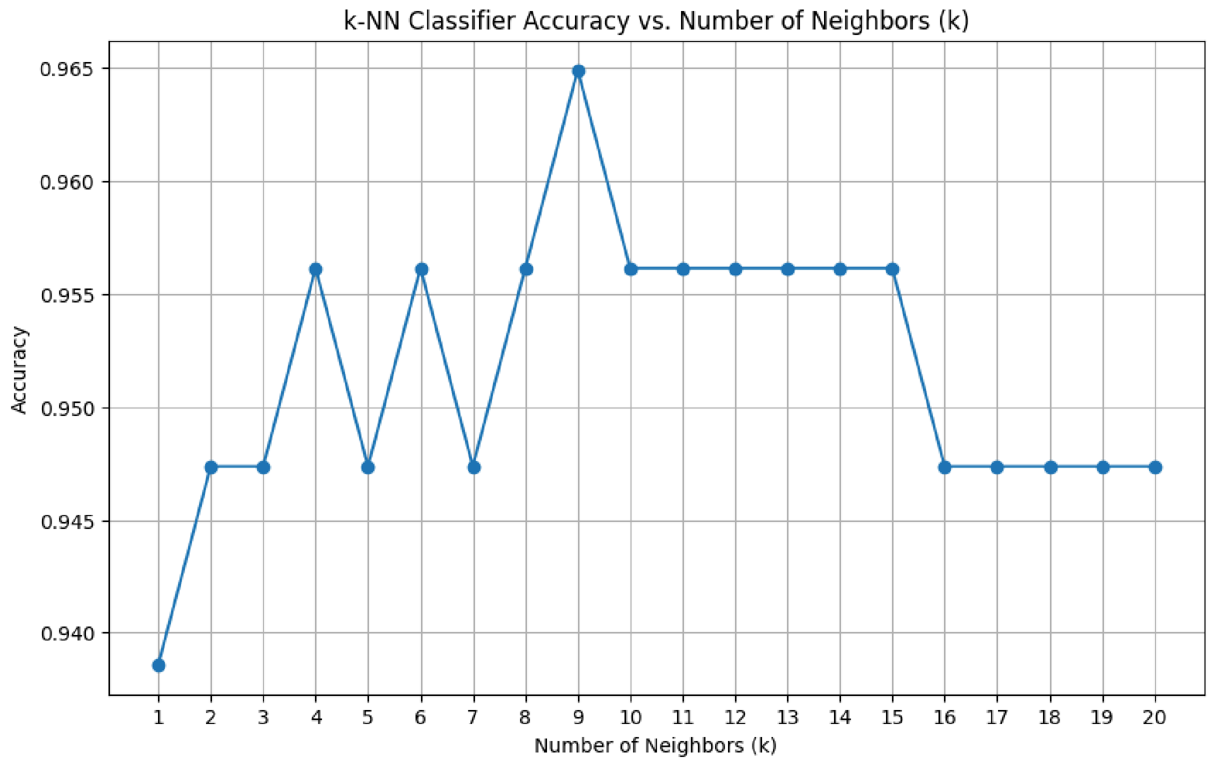
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Prepare the feature matrix (X) and target variable (y)
X_data = data.drop('diagnosis', axis=1)
y_data = data['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2,
```

In [142…

```
k_values = range(1, 21)  # Trying k from 1 to 20
accuracies = []

# Evaluate k-NN for different values of k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

    # print(f"Results for k = {k}:")
    # print_evaluation_metrics(y_test, y_pred)
    # print("\n" + "="*50 + "\n")
```

In [143…

```
# Plotting the accuracy vs. k
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.title('k-NN Classifier Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid()
plt.show()
```



GridSearch для підбору оптимальних параметрів

In [144…

```
param_grid = {
    'n_neighbors': range(1, 21),  # Testing neighbors from 1 to 25
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'metric': ['euclidean', 'manhattan'],  # You can add more metrics if desired
}

knn = KNeighborsClassifier()
```

In [145…

```
from sklearn.model_selection import train_test_split, GridSearchCV
# Perform Grid Search with cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=3, scoring='accuracy', verbose=1, n_
grid_search.fit(X_train, y_train)

# Output the best parameters and the best score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
Fitting 3 folds for each of 320 candidates, totalling 960 fits
Best Parameters: {'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 7, 'wei
ghts': 'uniform'}
Best Cross-Validation Score: 0.9647670500755199
```

In [146…

```
# Make predictions with the best parameters on the test set
best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)

print_evaluation_metrics(y_test, y_pred)
```

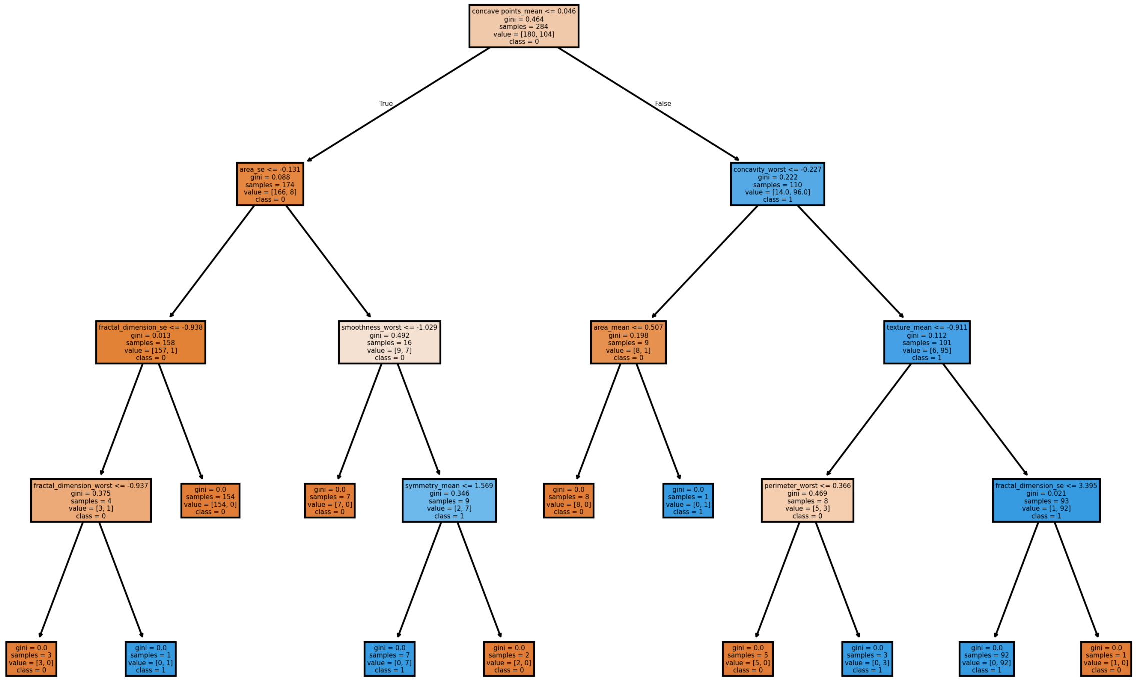
```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

Confusion Matrix:
[[68  3]
 [ 3 40]]
```

Дерево ухвалення рішень (Decision Tree) та ансамбль дерев ухвалення рішень (Random Forest)

Decision Tree

In [147…

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree

X_train, X_test, y_train, y_test = train_test_split(X_data, data[y_column], test_si

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

plt.figure(figsize=(12, 8), dpi=300)
plot_tree(clf, filled=True, feature_names=X_column, class_names=['0', '1'])
plt.show()

print_evaluation_metrics(y_test, predictions)
```




```
Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       177
           1       0.90      0.91      0.90       108

    accuracy                           0.93       285
   macro avg       0.92      0.92      0.92       285
weighted avg       0.93      0.93      0.93       285

Confusion Matrix:
[[166  11]
 [ 10  98]]
```

Тестовий код для того, щоб побачити аутпут predict_proba

In [148…

```
predict_proba = clf.predict_proba(X_test)
print(predict_proba)
```

```
[[0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
```

```
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[1. 0.]
[0. 1.]
[0. 1.]
```

```
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
```

```
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
```

```
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
```

```
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

Random Forest

In [149…

```
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X_data, data[y_column], test_si

rf_classifier = RandomForestClassifier(n_estimators=10, random_state=31337)
rf_classifier.fit(X_train, y_train)

predictions = rf_classifier.predict(X_test)

print_evaluation_metrics(y_test, predictions)
```

```
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       186
           1       0.96      0.80      0.87        99

    accuracy                           0.92       285
   macro avg       0.93      0.89      0.91       285
weighted avg       0.92      0.92      0.92       285

Confusion Matrix:
[[183   3]
 [ 20  79]]
```

AdaBoost

In [150…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import AdaBoostClassifier


X_data = data.drop('diagnosis', axis=1)
y_data = data['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2,
```

In [151…

```
ada_boost_model = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_boost_model.fit(X_train, y_train)
```

```
C:\Users\user\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra
8p0\LocalCache\local-packages\Python312\site-packages\sklearn\ensemble\_weight_boost
ing.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and wil
l be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

Out[151…

```
AdaBoostClassifier

AdaBoostClassifier(random_state=42)
```

In [152…

```python
y_pred = ada_boost_model.predict(X_test)

print_evaluation_metrics(y_test, y_pred)
```

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Confusion Matrix:
[[70  1]
 [ 2 41]]
```

SVM

In [153…

```python
X_data = data.drop('diagnosis', axis=1)
y_data = data['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2,
```

In [154…

```python
from sklearn.svm import SVC

# Set up the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],       # Regularization parameter
    'gamma': [0.001, 0.01, 0.1, 1],    # Kernel coefficient
}

svm = SVC(kernel='rbf', random_state=31337)
```

In [155…

```python
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', verbose=1, n_
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters: {'C': 10, 'gamma': 0.01}
Best Cross-Validation Score: 0.9736263736263737
```

In [156…

```python
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test)
```

```
print_evaluation_metrics(y_test, y_pred)
```

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114

Confusion Matrix:
[[71  0]
 [ 2 41]]
```