Requirements and questions

Exploratory Analysis(10 points)

- Are three any correlation between features? why or why not?
- Are there any outliers in the dataset? why or why not?
- Are there any normalization needed? why or why not?
- Are there any missing values? Why or why not?

Classification (25 points)

- Experiment the following algorithms/models: decision tree, random forest, adaboost, KNN, SVM, MLP, and Naive Baye
- For each model, train-test-split with 80% for train and 20% for test.
- For each algorithm, output the following performance measures: accuracy, precision, and recall

Visualization (5 points)

Pick at least one visualization for model perfomnace comparison.

Reflection (15 points)

- What is the best K in KNN? Only consider k in the range of 1-15. Use odd numbers only.
- What is the most important features found by decision tree and random forest?
- Which algorithm has the highest accuracy? Is there model overfitting for this algorithm? Why or why not?Perform 10-fold cross-validation with this algorithm and report the accuracy, precision and recall.

```
#pasted from URL
from ucimlrepo import fetch_ucirepo

# fetch dataset
iris = fetch_ucirepo(id=53)

# data (as pandas dataframes)
X = iris.data.features
y = iris.data.targets

# metadata
print(iris.metadata)

# variable information
print(iris.variables)
```

```
{'uci_id': 53, 'name': 'Iris', 'repository_url': 'https://archive.ics.uci
           name     role         type demographic  \
0  sepal length  Feature   Continuous        None
1   sepal width  Feature   Continuous        None
2  petal length  Feature   Continuous        None
3   petal width  Feature   Continuous        None
4         class   Target  Categorical        None

                                         description units missing_values
0                                               None    cm             no
1                                               None    cm             no
2                                               None    cm             no
3                                               None    cm             no
4  class of iris plant: Iris Setosa, Iris Versico...  None             no
```

```
correlation_matrix = X.corr(method='pearson')
print(correlation_matrix)

#yes there is a strong correlation between the pedal length
```

```
              sepal length  sepal width  petal length  petal width
sepal length      1.000000    -0.109369      0.871754     0.817954
sepal width      -0.109369     1.000000     -0.420516    -0.356544
petal length      0.871754    -0.420516      1.000000     0.962757
petal width       0.817954    -0.356544      0.962757     1.000000
```

```
import numpy as np

z_scores = np.abs((X - X.mean()) / X.std())
outliers = (z_scores > 3).any(axis=1)
print(X[outliers])
print("num of outliers:", outliers.sum())

#there is one outlier present this was found ing z_score to detmine the
```

```
    sepal length  sepal width  petal length  petal width
15           5.7          4.4           1.5          0.4
num of outliers: 1
```

```
# check for missing vals
dups = X.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))

# yes we have 3 duplicate rows found in hte X set the .duplicated() func
```

```
Number of duplicate rows = 3
```

```
print(X.isnull().sum())
# no there are no null vlaues we know this because of the function call wl
```

```
sepal length    0
sepal width     0
petal length    0
petal width     0
dtype: int64
```

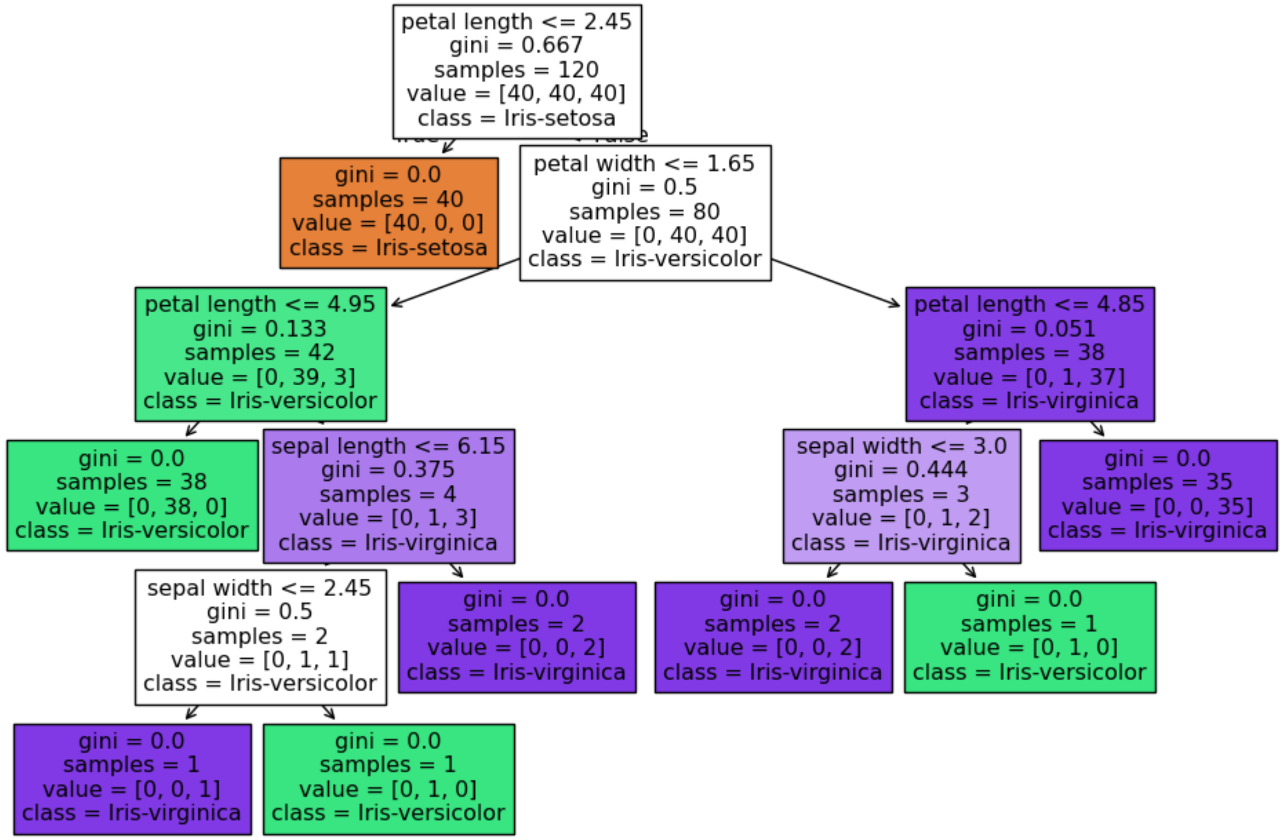
part 2: Classification

```
#global imports and data split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold, c
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_scor

# im just using the same train test split for each model

# y -> 1D numeric
y_1d = np.asarray(y).squeeze()
le = LabelEncoder()
y_enc = le.fit_transform(y_1d)

# split
X_train, X_test, y_train, y_test = train_test_split(X, y_enc, test_size=
```

```
#decision tree classsifier



# train
clf = tree.DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# eval
y_pred = clf.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred):.3f}")
print(f"precision (macro): {precision_score(y_test, y_pred, average='mac
print(f"recall (macro): {recall_score(y_test, y_pred, average='macro'):.

# visualization of each level of decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=X.columns, class_names=sorted(y1d.uniq
plt.show()

importances = pd.Series(clf.feature_importances_, index=X.columns).sort_
print("DTfeature importances:\n", importances)
```

```
accuracy: 0.933
precision (macro): 0.933
```

```
recall (macro): 0.933
```




```
DTfeature importances:
 petal length    0.558568
petal width     0.406015
sepal width     0.029167
sepal length    0.006250
dtype: float64
```

```
from sklearn.ensemble import RandomForestClassifier

# random forrest classsifier
```

```
rf_clf = RandomForestClassifier(n_estimators=200, random_state=42)
rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)

#  metrics
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average="macro", zero_division=0)
rec = recall_score(y_test, y_pred, average="macro", zero_division=0)
print(f"Accuracy: {acc:.3f} | Precision (macro): {prec:.3f} | Recall (mac


print(classification_report(y_test, y_pred))



# feature importances with names
importances = pd.Series(rf_clf.feature_importances_, index=X.columns).sor
print("feat importances:\n", importances)

#  10-fold CV with accuracy/precision/recall for hte later question
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_validate(
    rf_clf, X, y1d, cv=cv,
    scoring={"acc":"accuracy","prec":"precision_macro","rec":"recall_macr
```

```
Accuracy: 0.900 | Precision (macro): 0.902 | Recall (macro): 0.900
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.82      0.90      0.86        10
           2       0.89      0.80      0.84        10

    accuracy                           0.90        30
   macro avg       0.90      0.90      0.90        30
weighted avg       0.90      0.90      0.90        30

feat importances:
 petal length    0.453793
petal width     0.412449
sepal length    0.115873
sepal width     0.017885
dtype: float64
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier


# base learner
stump = DecisionTreeClassifier(max_depth=1, random_state=42)
adaboost = AdaBoostClassifier(estimator=stump, n_estimators=50,learning_


# evaluate / fit
adaboost.fit(X_train, y_train)
y_pred = adaboost.predict(X_test)

print(f"accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/ensemble/_weight_boosting
  warnings.warn(
accuracy: 0.9333
[[10  0  0]
 [ 0  9  1]
 [ 0  1  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      0.90      0.90        10
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold, cross_validate


cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

rows = []
for k in range(1, 16, 2):
    knn = Pipeline([
        ("scaler", StandardScaler()),
```

```
            ("clf", KNeighborsClassifier(n_neighbors=k))])

        # fit on train | evaluate on test
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        acc  = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, average="macro", zero_division=
        rec  = recall_score(y_test, y_pred, average="macro", zero_division=0)
        print(f"k={k} | Test  Acc: {acc:.3f} | Prec(m): {prec:.3f} | Rec(m): 

        # 10-fold CV on  data
        scores = cross_validate(
            knn, X, y_enc, cv=cv,
            scoring={"acc":"accuracy", "prec":"precision_macro", "rec":"recal


        cv_acc  = scores["test_acc"].mean()
        cv_prec = scores["test_prec"].mean()
        cv_rec  = scores["test_rec"].mean()

        rows.append([k, acc, prec, rec, cv_acc, cv_prec, cv_rec])

    # summary table
    results_df = pd.DataFrame(rows, columns=[
        "k", "test_acc", "test_prec", "test_rec", "cv_acc", "cv_prec", "cv_re

    print("\nsummary (sorted by CV accuracy..):\n", results_df)

    # pick best k by CV accuracy
    best_row = results_df.sort_values(
        ["cv_acc", "test_acc", "k"], ascending=[False, False, True]).iloc[0]

    best_k = int(best_row["k"])
    print(f"\nbest k by CV accuracy: {best_k} "
          f"(CV acc={best_row['cv_acc']:.3f}, test Acc={best_row['test_acc']:
```

```
k=1 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
k=3 | Test  Acc: 0.933 | Prec(m): 0.944 | Rec(m): 0.933
k=5 | Test  Acc: 0.933 | Prec(m): 0.944 | Rec(m): 0.933
k=7 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
k=9 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
k=11 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
k=13 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
k=15 | Test  Acc: 0.967 | Prec(m): 0.970 | Rec(m): 0.967
```

```
Summary (sorted by CV accuracy):
    k  test_acc  test_prec  test_rec    cv_acc   cv_prec    cv_rec
6  13  0.966667   0.969697  0.966667  0.973333  0.977778  0.973333
7  15  0.966667   0.969697  0.966667  0.960000  0.966032  0.960000
2   5  0.933333   0.944444  0.933333  0.960000  0.969841  0.960000
3   7  0.966667   0.969697  0.966667  0.960000  0.968254  0.960000
5  11  0.966667   0.969697  0.966667  0.960000  0.969841  0.960000
4   9  0.966667   0.969697  0.966667  0.953333  0.964286  0.953333
1   3  0.933333   0.944444  0.933333  0.946667  0.957619  0.946667
0   1  0.966667   0.969697  0.966667  0.940000  0.946587  0.940000

Best k by CV accuracy: 13 (CV Acc=0.973, Test Acc=0.967)
```

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC


# SVM with scaling
svm_clf = Pipeline([("scaler", StandardScaler()),
    ("svc", SVC(kernel="rbf", C=1.0, gamma="scale", random_state=42))])

# fit / predict
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

# evaluation
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average="macro", zero_division=0)
rec = recall_score(y_test, y_pred, average="macro", zero_division=0)

print(f"accuracy: {acc:.3f} | Precision (macro): {prec:.3f} | Recall (mac
print("confusion matrix:\n", confusion_matrix(y_test, y_pred))
print("classification report:\n", classification_report(y_test, y_pred, ta
```

```
Accuracy: 0.967 | Precision (macro): 0.970 | Recall (macro): 0.967
Confusion matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
Classification report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      0.90      0.95        10
 Iris-virginica       0.91      1.00      0.95        10

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

```
#MLP
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import  confusion_matrix

#  scale to MLP
mlp = Pipeline([
```

```
        ("scaler", StandardScaler()),
        ("mlp", MLPClassifier(
            hidden_layer_sizes=(64, 32),   # two layers
            activation="relu",
            solver="adam",
            alpha=1e-3,
            learning_rate="adaptive",
            max_iter=500,
            early_stopping=True,
            n_iter_no_change=10,
            random_state=42))
    ])

    # fit / evaluate
    mlp.fit(X_train, y_train)
    y_pred = mlp.predict(X_test)

    print("accuracy:", accuracy_score(y_test, y_pred))
    print("confusion matrix:\n", confusion_matrix(y_test, y_pred))
    print("report:\n", classification_report(y_test, y_pred))
```

```
accuracy: 0.6666666666666666
confusion matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0 10  0]]
report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.50      1.00      0.67        10
           2       0.00      0.00      0.00        10

    accuracy                           0.67        30
   macro avg       0.50      0.67      0.56        30
weighted avg       0.50      0.67      0.56        30

/usr/local/lib/python3.12/dist-packages/sklearn/preprocessing/_label.py:1
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.p
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.p
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.p
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
#Naive Baye
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB(var_smoothing=1e-9)
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)
print("accuracy:", accuracy_score(y_test, y_pred))
print("confusion matrix:\n", confusion_matrix(y_test, y_pred))
print("report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9666666666666667
Confusion matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      0.90      0.95        10
 Iris-virginica       0.91      1.00      0.95        10

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```

Start coding or generate with AI.

Reflection questions:

- What is the best K in KNN? Only consider k in the range of 1-15. Use odd numbers only. The best vlaue for K is 3, 5 and 9 performed the best with perfect accuracy. Other vlaues of K are close but not as high as those three.
- What is the most important features found by decision tree and random forest? from randmforrest the most important feature is: `pedal` and for hte decision tree the most important feature is `pedal length` with the width following in both.
- Which algorithm has the highest accuracy? Is there model overfitting for this algorithm? Why or why not?Perform 10-fold cross-validation with this algorithm and report the accuracy, precision and recall.

the model with the highest accruacy was the Knearest neighbors model with K = (3, 5 and 9) all achieving perfect accuracy (no 10-fold validation yet), yes there was some over fitting but not too much, after performing the 10-fold CV the model achieved `CV Acc=0.973, Test Acc=0.967` these results are close to the origional output. see below for log table or all is avaible at the code block above.

Output logs for both vlaues of k after 10-fold CV:

`k test_acc test_prec test_rec cv_acc cv_prec cv_rec`

`6 13 0.966667 0.969697 0.966667 0.973333 0.977778 0.973333`

`7 15 0.966667 0.969697 0.966667 0.960000 0.966032 0.960000`