```
# Connect Colab with your Google Drive
#from google.colab import drive
#drive.mount('/content/drive')
```

```
# Importing a file from google drive
import pandas as pd
import os
os.chdir = '/content'
#This box of code is not needed if you use the above box to import the data set
```

```
df = pd.read_csv('diabetes.csv', na_values = ['Not Available'])
#This box of code is not needed if you use the first box to import the data set
```

```
df.shape
```

```
(768, 10)
```

```
df.head()
```

	Gender	Income	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	None-Binary	100K-200K	148	72	35	0	33.6	
1	None-Binary	50-100K	85	66	29	0	26.6	
2	Female	>200K	183	64	0	0	23.3	
3	Female	>50K	89	66	23	94	28.1	

```
# import packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# scikit - learn
# ! pip install scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.metrics import accuracy_score

# for bagging, boosting, and random forest
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, RandomForestClassifier
```

```
# missing values
df.isna().sum()
```

```
Gender                      0
Income                      0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
# If so ----- drop missing values
df = df.dropna()
df.shape
```

```
(768, 10)
```

```
# data types
df.dtypes
```

```
Gender                       object
Income                       object
Glucose                       int64
```

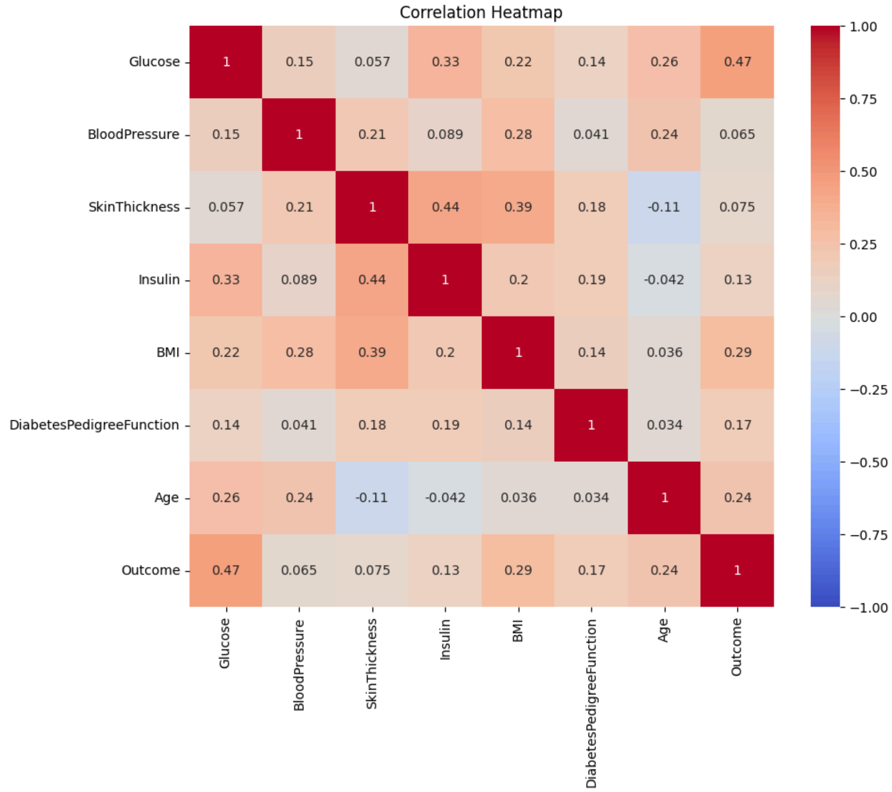
```
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

```
# summary statistics
df.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	120.894531	69.105469	20.536458	79.799479	31.992578	
std	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	140.250000	80.000000	32.000000	127.250000	36.600000	
max	199.000000	122.000000	99.000000	846.000000	67.100000	

```
# Correlation analysis
correlation_matrix = df.corr()
plt.figure(figsize = (10, 8))
sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm', vmin = -1, vmax = 1)
plt.title('Correlation Heatmap')
plt.show()
```

```
<ipython-input-12-96445fb39e5e>:2: FutureWarning: The default value of numeric_only in D
  correlation_matrix = df.corr()
```




```
# grouping 1
income_cat = df.groupby('Income')[['Outcome']].agg(['sum', 'mean', 'count']).reset_index()
income_cat

#Question which group of patients should recieve treatment based on the grouping?
#which group has the most amount of patients
# can ask us to sort based on the gender
```

	Income	Outcome		
		sum	mean	count
0	100K-200K	70	0.339806	206
1	50-100K	72	0.361809	199
2	>200K	65	0.351351	185
3	>50K	61	0.342697	178

```
# Grouping 2
Gender_cat = df.groupby('Gender')[['Outcome']].agg(['sum', 'mean', 'count']).reset_index()
sorted_Gender_cat = Gender_cat.sort_values(by=('Outcome', 'mean'), ascending = True)
print(sorted_Gender_cat)
```

```
          Gender Outcome
                     sum      mean count
0         Female      80  0.320000   250
```

```
2  None-Binary       97  0.337979   287
1         Male       91  0.393939   231
```

```
# Advanced Grouping 3
# Imagine you received a funding to support a specific age group of patients who are under high risky of diabetes.
# Which age group (>50;<=50) would you support?

new_ls = []
for x in range(len(df)):
  if df['Age'][x]> 50:
    new_ls.append('>50')
  else:
    new_ls.append('<51')
df['Age-50'] = new_ls
df.head()
```

	Gender	Income	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	None-Binary	100K-200K	148	72	35	0	33.6	
1	None-Binary	50-100K	85	66	29	0	26.6	
2	Female	>200K	183	64	0	0	23.3	

```
age_group = df.groupby('Age-50')[['Outcome']].agg(['sum', 'mean', 'count']).reset_index()
age_group
```

	Age-50	Outcome		
		sum	mean	count
0	<51	230	0.334789	687
1	>50	38	0.469136	81

```
# Double Check Data types
df.dtypes
```

```
Gender                       object
Income                       object
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
Age-50                       object
dtype: object
```

```
# Generate dummy variables
dummies = pd.get_dummies(df[['Gender', 'Income']],
                            prefix = ['Gender', 'Income']) * 1
dummies
```

	Gender_Female	Gender_Male	Gender_None-Binary	Income_100K-200K	Income_50-100K	Income_>200K	I
0	0	0	1	1	0	0	
1	0	0	1	0	1	0	
2	1	0	0	0	0	1	
3	1	0	0	0	0	0	
4	1	0	0	0	0	0	
...	...	...	...	...	...	...	
763	0	0	1	0	0	1	
764	0	0	1	1	0	0	
765	0	1	0	0	1	0	
766	0	0	1	0	0	0	
767	0	0	1	0	1	0	

```
# Define predictors and target
X = pd.concat([df[['Glucose', 'BloodPressure', 'SkinThickness',
                   'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']], dummies], axis = 1)
Y = df['Outcome']
```

```
# data partition
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

```
# Initialize the decision tree classifier
clf = DecisionTreeClassifier(random_state = 42)
# Train the classifier
clf.fit(X_train, Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Check sklearn version
import sklearn
print(sklearn.__version__)
```

```
1.2.2
```

```
#!pip uninstall scikit-learn -y
#!pip install -U scikit-learn
```

```
#after you run the code above and restart the session, put the cursor back into
#that box and then click runtime and run before
#you can do the above step at the very start of the code after you import the packages
```

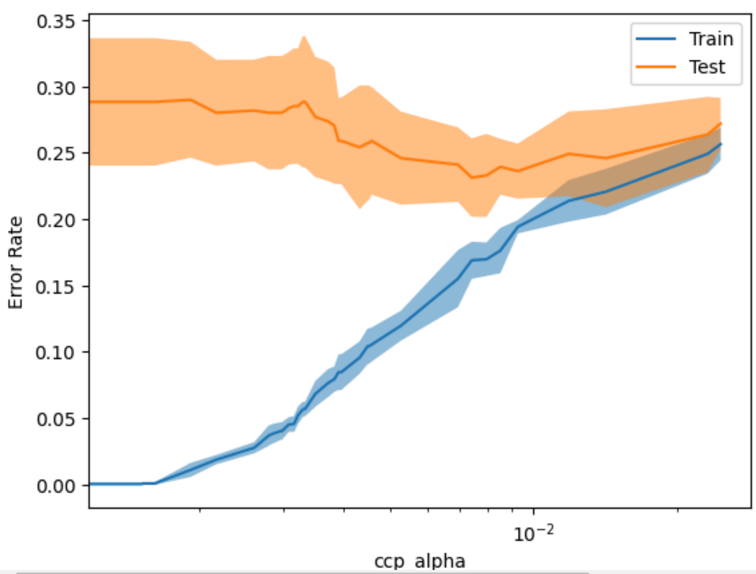
```
# ccp_alpha
from sklearn.model_selection import ValidationCurveDisplay, validation_curve
# Extract effective alphas for the full tree
path = clf.cost_complexity_pruning_path(X_train, Y_train)
ccp_alphas = path.ccp_alphas[:-1]
print(path)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-24-946f0c081052> in <cell line: 2>()
      1 # ccp_alpha
----> 2 from sklearn.model_selection import ValidationCurveDisplay, validation_curve
      3 # Extract effective alphas for the full tree
      4 path = clf.cost_complexity_pruning_path(X_train, Y_train)
      5 ccp_alphas = path.ccp_alphas[:-1]

ImportError: cannot import name 'ValidationCurveDisplay' from 'sklearn.model_selection'
(/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/__init__.py)

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
# Obtain train and test scores using validation_curve
train_scores, test_scores = validation_curve(
     clf, X_train, Y_train, param_name = 'ccp_alpha', param_range = ccp_alphas,
     cv = 5, scoring = 'accuracy'
 )

train_error_rates = 1- train_scores
test_error_rates = 1- test_scores
# Display the results using ValidationCurveDisplay
display = ValidationCurveDisplay(
    param_name = 'ccp_alpha', param_range = ccp_alphas,
    train_scores = train_error_rates, test_scores = test_error_rates, score_name = 'Accuracy'
)

display.plot()
plt.xscale('log')
plt.ylabel('Error Rate')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_plotting.py:98: RuntimeWarning: c
  return diff.max() / diff.min()
```



```
# create a table containing both error rate and ccp_alpha
mean_test_error = np.mean(test_error_rates, axis = 1)
std_test_error = np.std(test_error_rates, axis = 1)

# Create a DataFrame
df_ccp = pd.DataFrame({
    'ccp_alpha': ccp_alphas,
    'mean_test_score' : mean_test_error,
    'std_test_score' : std_test_error
})


pd.set_option('display.max_rows', None)
df_ccp
```

```
#the line in the middle shows the amount of errors that the model is making, pick
#a value towards the end of the list
```

20	0.002792	0.280221	0.042670
21	0.002800	0.280221	0.042670
22	0.002850	0.280221	0.042670
23	0.002961	0.280221	0.042670
24	0.002961	0.280221	0.042670
25	0.002969	0.280221	0.042670
26	0.002979	0.280221	0.042670
27	0.003078	0.283487	0.042378
28	0.003151	0.285113	0.043466
29	0.003162	0.285113	0.043466
30	0.003217	0.285113	0.043466
31	0.003294	0.288378	0.049407
32	0.003331	0.288378	0.049407
33	0.003500	0.276943	0.045307
34	0.003722	0.273691	0.044934
35	0.003833	0.270438	0.043720
36	0.003912	0.258990	0.032538
37	0.003965	0.258990	0.032538
38	0.004331	0.254112	0.046622
39	0.004501	0.257364	0.043418
40	0.004524	0.257364	0.043418
41	0.004586	0.258990	0.040502
42	0.005283	0.245928	0.035016
43	0.006950	0.241050	0.028033
44	0.007427	0.231294	0.029546
45	0.007974	0.232920	0.031264
46	0.008523	0.239424	0.021058
47	0.009271	0.236172	0.020741
48	0.011844	0.249180	0.031895
49	0.014170	0.245928	0.036855
50	0.023111	0.263774	0.028362
51	0.024597	0.271971	0.019417

```
# Prune the tree based on the optimnal ccp (best pruned tree)
clf3 = DecisionTreeClassifier(ccp_alpha = 0.014170)
clf3.fit(X_train, Y_train)
```

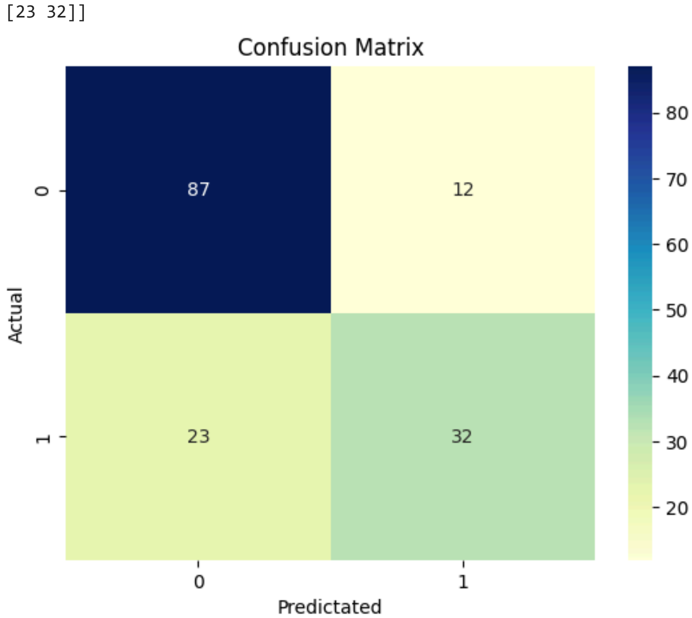
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(ccp_alpha=0.01417)
```

```
# Model Performance
from sklearn.metrics import accuracy_score
# model evaluation predicted class (start here)
Y_pred = clf3.predict(X_test)
Y_pred
# Predict probabilities for class 1 on the test data
y_prob = clf3.predict_proba(X_test)
y_prob1 = y_prob[:, 1]

accuracy_score(Y_test, Y_pred)
```

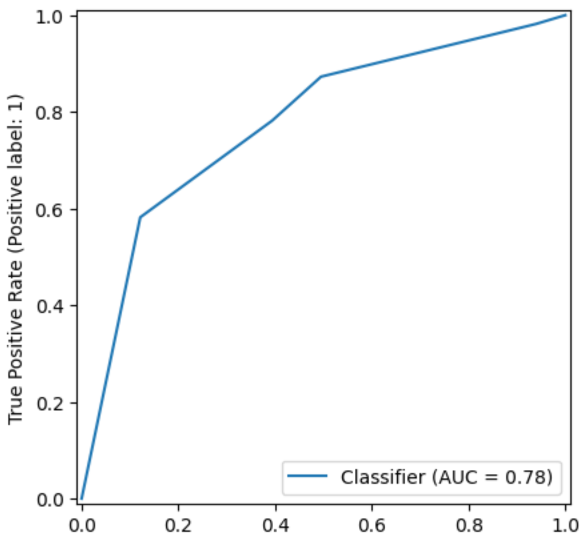
```
0.7727272727272727
```

```
# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)


import seaborn as sns
sns.heatmap(cm, annot = True, cmap = 'YlGnBu')
plt.xlabel('Predictated')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
[[87 12]
 [23 32]]
```



```
# ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import RocCurveDisplay

# Get predicted probabilities
probas_dt = clf3.predict_proba(X_test)[:, 1]

# plot individual roc curve
roc_display_dt = RocCurveDisplay.from_predictions(Y_test, probas_dt)
```




```
#use X.shape to find the upper bound of variables to use for max_features


# Instantiate Random Forest classifier
rf_clf = RandomForestClassifier(n_estimators = 50, random_state = 42, max_features = 2, oob_score = True)

# fine-tune random forest classifier
from sklearn.model_selection import GridSearchCV
param_grid = {'max_features': list(range(1, 15))}
# Narrow hyperparameter grid based on the results from randomized search
rf_grid_search = GridSearchCV(estimator = rf_clf, param_grid = param_grid, cv = 5, n_jobs = -1, verbose = 2)
rf_grid_search.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
list(range(1,14))
# he may ask to fine tune two different variables
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```