8.3 Course Project: Milestone 4--Finalizing Your Results

In [1]:
```
#Load necessary libraries
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
import opendatasets as od
```

In [2]:
```
#Read csv into python dataframe
breast_cancer_df = pd.read_csv("Breast Cancer Prediction.csv")
breast_cancer_df.head(5)
```

Out[2]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

Data Preparation

Rename Columns

In [3]:
```
breast_cancer_df.columns = ['Sample_code_number','Clump_Thickness','Uniformity_Cell_Size
                            'Marginal_Adhesion','Single_Epithelial_Cell_Size','Bare_Nucl
                            'Normal_Nucleoli','Mitoses','Class']

breast_cancer_df.head(5)
```

Out[3]:

	Sample_code_number	Clump_Thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_
0	1000025	5	1	1	1	
1	1002945	5	4	4	5	
2	1015425	3	1	1	1	
3	1016277	6	8	8	1	
4	1017023	4	1	1	3	

I renamed columns by replacing spaces with '_' (underscore) for ease of column reference.

Check for null rows and/or columns

In [4]:
```
breast_cancer_df.isnull().sum()
```

Out[4]:
```
Sample_code_number              0
Clump_Thickness                 0
Uniformity_Cell_Size            0
```

```
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                    0
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

Check for duplicates

In [5]:
```
print('Dataframe before dropping duplicates :', breast_cancer_df.shape)
flight_data_df = breast_cancer_df.drop_duplicates() # 1,389 rows dropped
print('Dataframe after dropping duplicates :',breast_cancer_df.shape)
```

```
Dataframe before dropping duplicates : (683, 11)
Dataframe after dropping duplicates : (683, 11)
```

Check for Data classification

In [6]:
```
breast_cancer_df.Class.unique()

#2 for benign, 4 for malignant
```

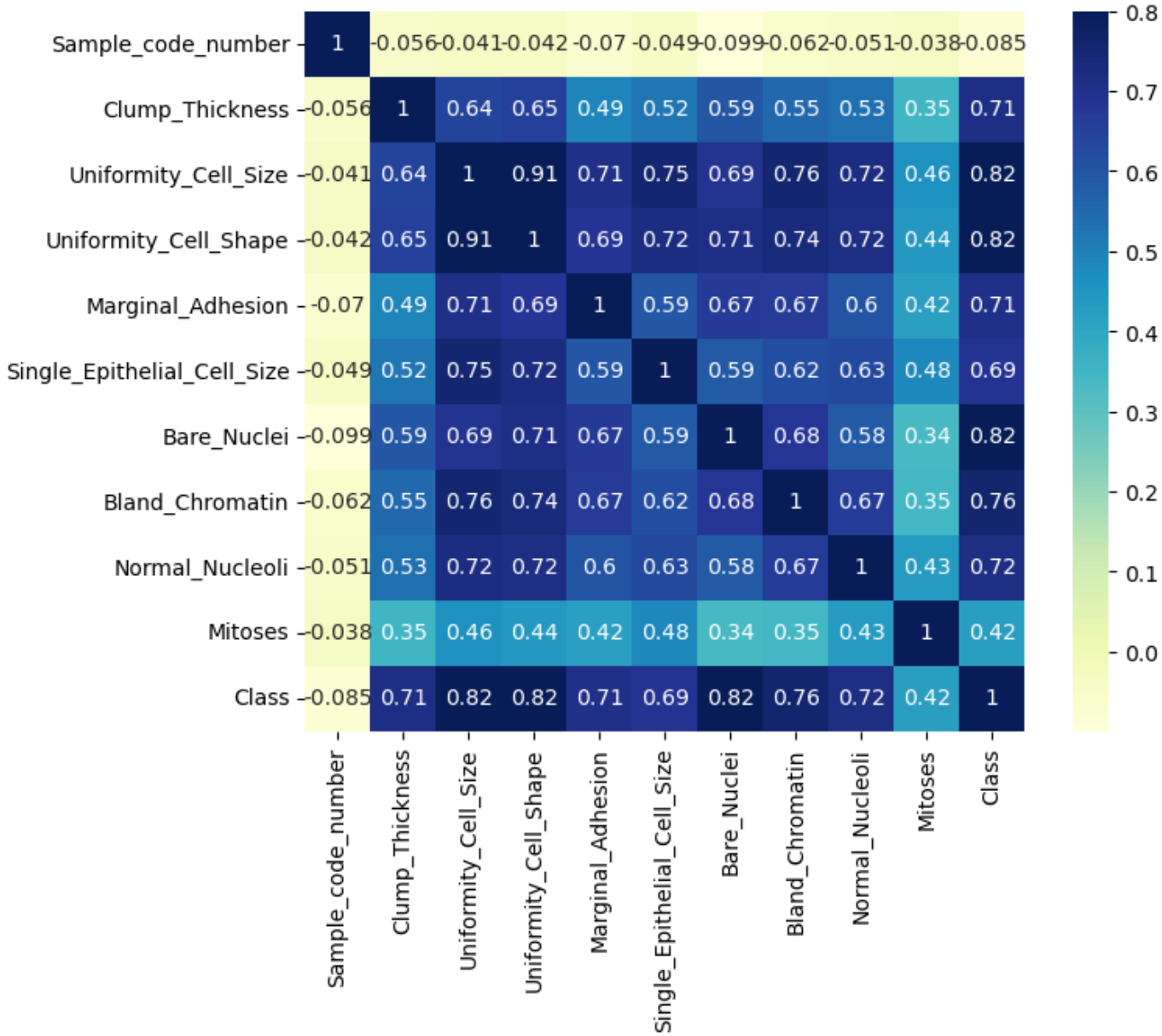
Out[6]:
```
array([2, 4], dtype=int64)
```

Data is classified into two classes benign and malignant

Visualizations

HEATMAP

In [7]:
```
corrmat = breast_cancer_df.corr()
f, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(corrmat, vmax=.8, square=True,annot=True,cmap='YlGnBu');
plt.show()
```



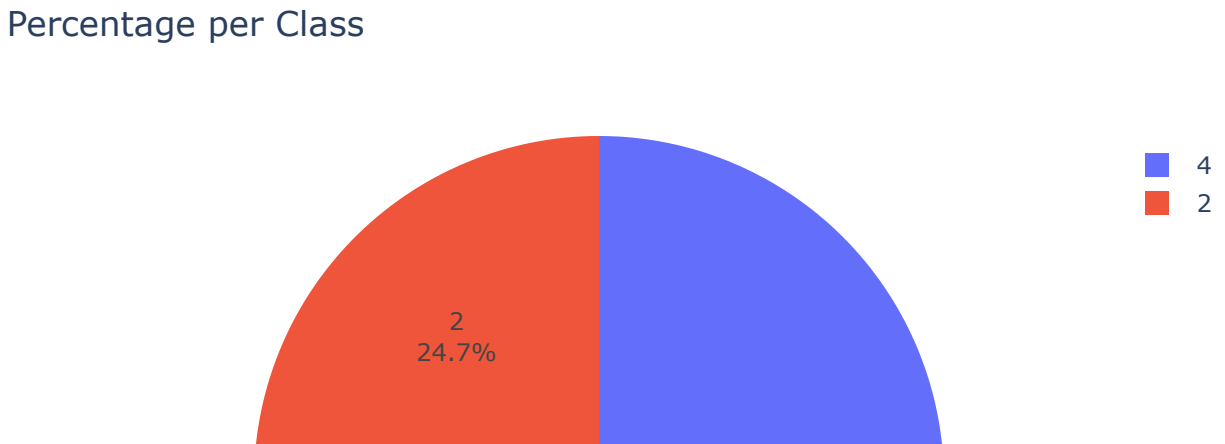
SCATTER PLOTS

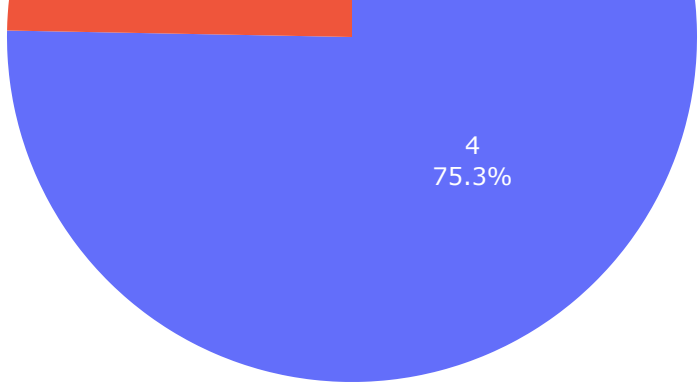
In [8]:
```
sns.set(font_scale=1)
sns.set_theme('notebook', style='dark')
plt.style.use("grayscale")

sns.pairplot(breast_cancer_df, hue='Class',
             vars=['Clump_Thickness','Uniformity_Cell_Size','Uniformity_Cell_Shape',
                   'Marginal_Adhesion','Single_Epithelial_Cell_Size','Bare_Nuclei','Blan
                   'Normal_Nucleoli','Mitoses'])
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x14bb81f87f0>
```

PIE CHART

In [9]:
```
fig = px.pie(breast_cancer_df, values='Bare_Nuclei', names='Class', title='Percentage pe
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show("notebook")
```





We can see that the data is imbalanced with 75.3% malignant and 24.7% 2 benign.

Models

Train and Test split of data

In [181…
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score,confusion_matrix, c
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score,KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

In [182…
```
X = breast_cancer_df.drop(columns="Class").values
Y = breast_cancer_df["Class"].values
```

In [183…
```
X.shape, Y.shape
```

Out[183]:
```
((683, 10), (683,))
```

In [184…
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state=
```

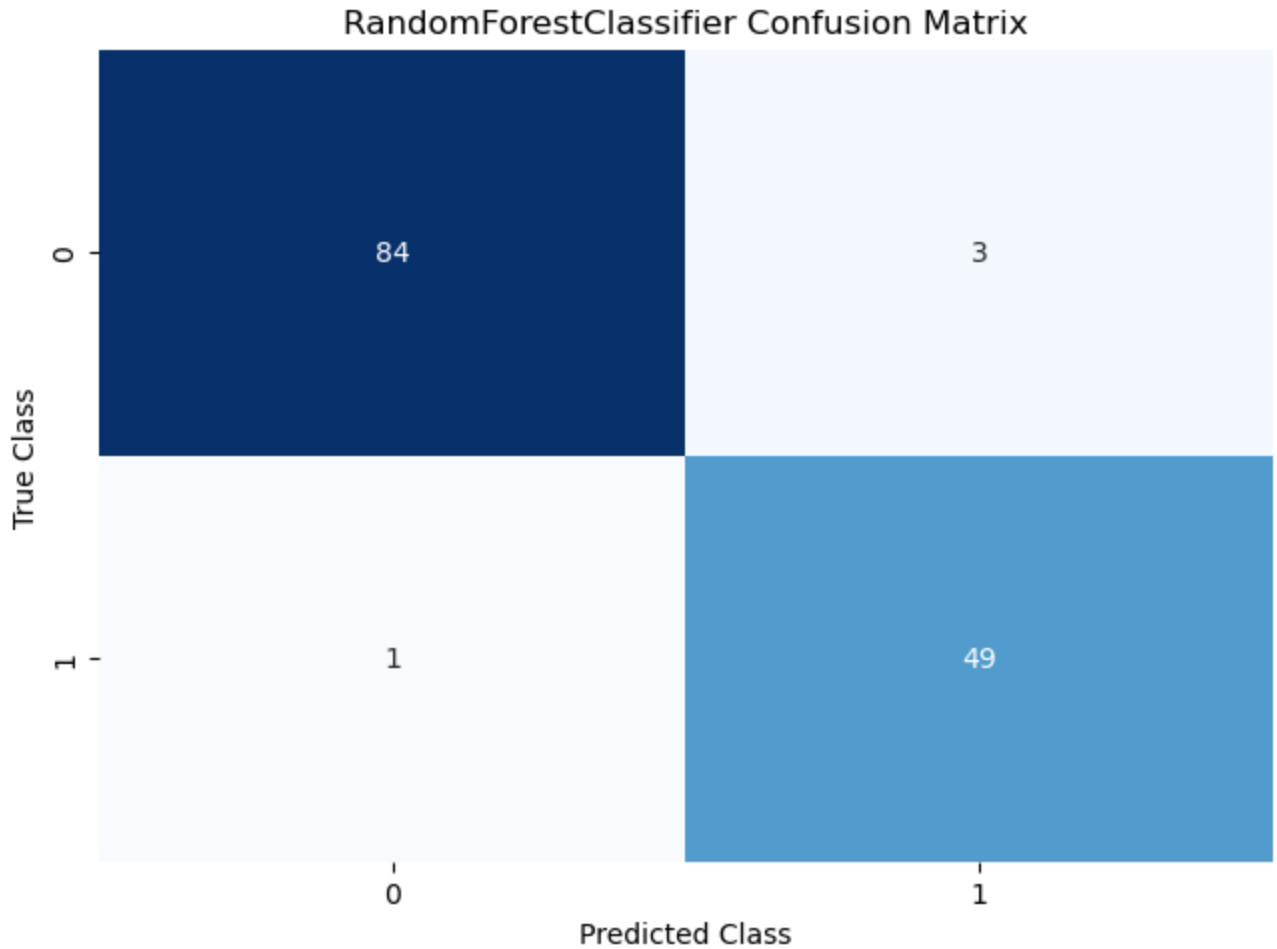
RandomForestClassifier with imbalanced data

In [185…
```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
```

In [186…
```
#Build the confusion matrix
matrix = confusion_matrix(y_test, y_pred )

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)
plt.style.use("default")

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
```

```
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[84  3]
 [ 1 49]]
```



In [187…

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(f'ROC-AUC score : {roc_auc_score(y_test, y_pred)}')
print(f'Accuracy score : {accuracy_score(y_test, y_pred)}')
```

```
              precision    recall  f1-score   support

           2       0.99      0.97      0.98        87
           4       0.94      0.98      0.96        50

    accuracy                           0.97       137
   macro avg       0.97      0.97      0.97       137
weighted avg       0.97      0.97      0.97       137

[[84  3]
 [ 1 49]]
ROC-AUC score : 0.9727586206896552
Accuracy score : 0.9708029197080292
```

SMOTE to balance the imbalanced data

In [188…

```
smote = SMOTE()
x, y = smote.fit_resample(X, Y)
```

In [189…

```
#Split the smote (balanced) data into train and test subsets:
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x,y,test_size = 0.2, ran
```

RandomForestClassifier with balanced data

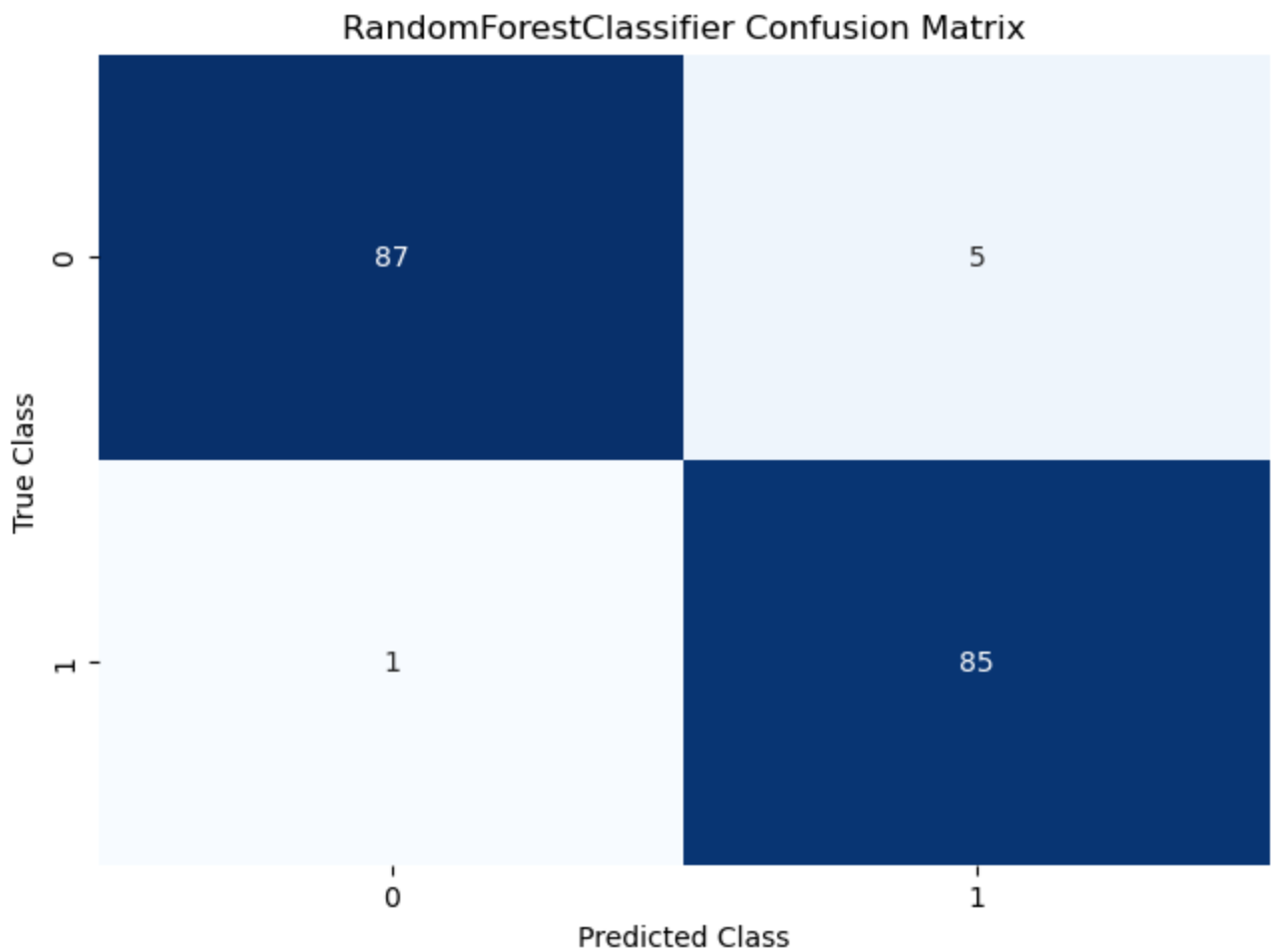
In [190…

```
rfc = RandomForestClassifier()
rfc.fit(x_train_sm, y_train_sm)

y_pred_sm = rfc.predict(x_test_sm)
```

In [191…

```
#Build the confusion matrix
matrix_sm = confusion_matrix(y_test_sm, y_pred_sm)

print(matrix_sm)

# Create pandas dataframe
df_sm = pd.DataFrame(matrix_sm)

# Create a heatmap
sns.heatmap(df_sm, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[87  5]
 [ 1 85]]
```



In [192…

```
print(classification_report(y_test_sm, y_pred_sm))
print(confusion_matrix(y_test_sm, y_pred_sm))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_sm)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_sm)}')
```

```
              precision    recall  f1-score   support

           2       0.99      0.95      0.97        92
           4       0.94      0.99      0.97        86

    accuracy                           0.97       178
```

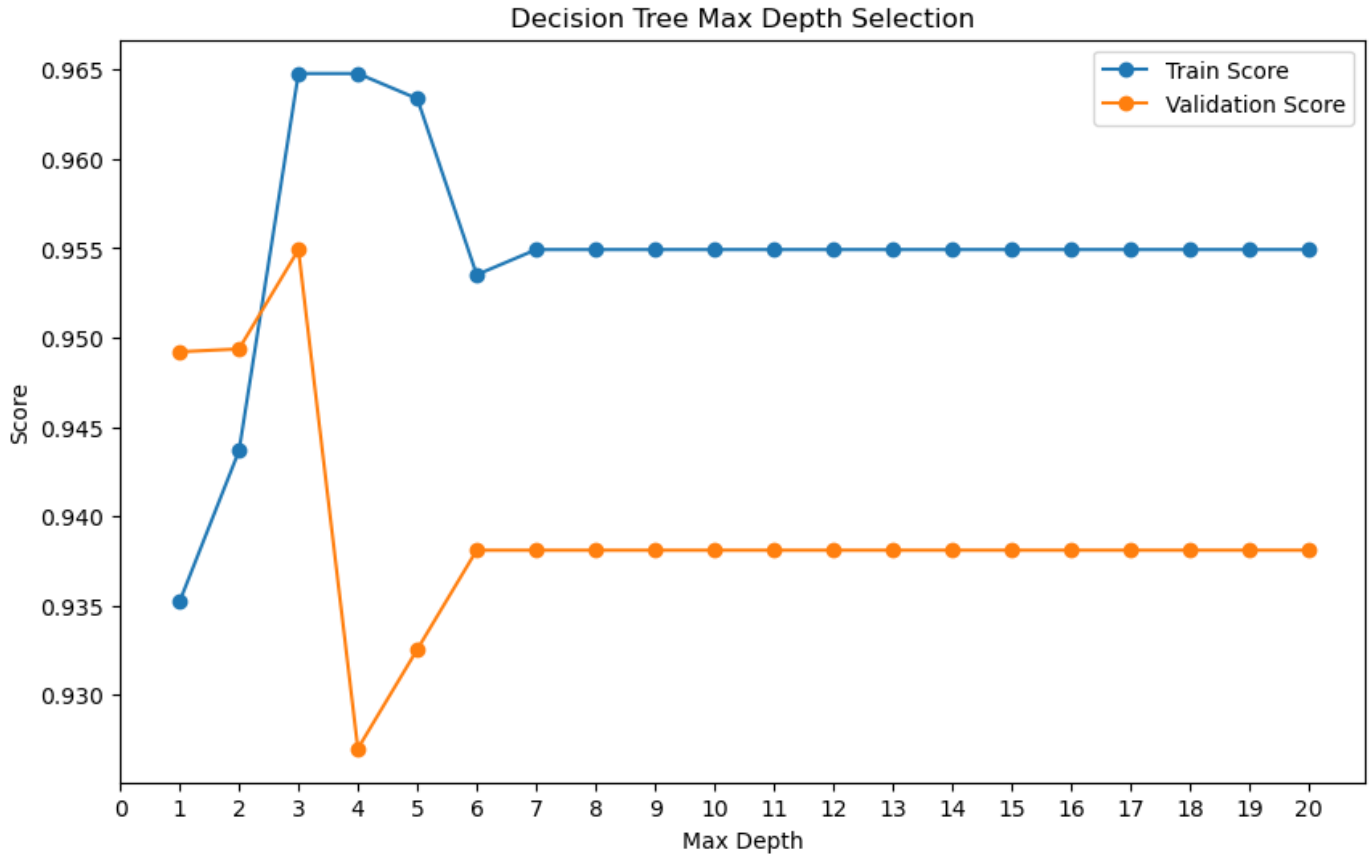
```
   macro avg       0.97      0.97      0.97       178
weighted avg       0.97      0.97      0.97       178

[[87  5]
 [ 1 85]]
ROC-AUC score : 0.9670121334681496
Accuracy score : 0.9662921348314607
```

- For predicting cases of benign tumors (class 2), the model has high precision (0.99), meaning that when it predicts a tumor as benign, it is correct 99% of the time. The recall (sensitivity) is 0.95, indicating that the model captures 95% of the actual benign cases. The F1-score (a balance between precision and recall) is 0.97.
- For predicting cases of malignant tumors (class 4), the model has slightly lower precision (0.94) compared to class 2, but higher recall (0.99). This suggests that while the model might be slightly less precise in predicting malignant cases, it is very effective at capturing most of the actual malignant cases. The F1-score for class 4 is also 0.97.
- The accuracy of 0.97 indicates that the model correctly predicts around 97% of the instances in the dataset.

DecisionTreeClassifier

In [193…

```python
# Range of max depth values to try
max_depth_values = np.arange(1, 21)

# Lists to store cross-validation scores
train_scores = []
val_scores = []

# Perform cross-validation for different max depth values
for max_depth in max_depth_values:
    model = DecisionTreeClassifier(max_depth=max_depth, random_state=0)
    train_score = np.mean(cross_val_score(model, x_train_sm, y_train_sm, cv=5))
    val_score = np.mean(cross_val_score(model, x_test_sm, y_test_sm, cv=5))
    train_scores.append(train_score)
    val_scores.append(val_score)

# Plot the validation curve
plt.figure(figsize=(10, 6))
plt.plot(max_depth_values, train_scores, label='Train Score', marker='o')
plt.plot(max_depth_values, val_scores, label='Validation Score', marker='o')
plt.xticks(np.arange(0, 21, 1.0))
plt.xlabel('Max Depth')
plt.ylabel('Score')
plt.title('Decision Tree Max Depth Selection')
plt.legend()
plt.show()
```



The above plot shows that the best accuracy for the model is when the parameter max_depth is 3.

In [199…

```
# Use the DecisionTreeClassifier to fit data
clf = DecisionTreeClassifier(max_depth=3 ,random_state=0)
clf.fit(x_train_sm, y_train_sm)
```

Out[199]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [200…

```
#Predict y data with classifier:
y_pred_dtc = clf.predict(x_test_sm)
```

In [225…

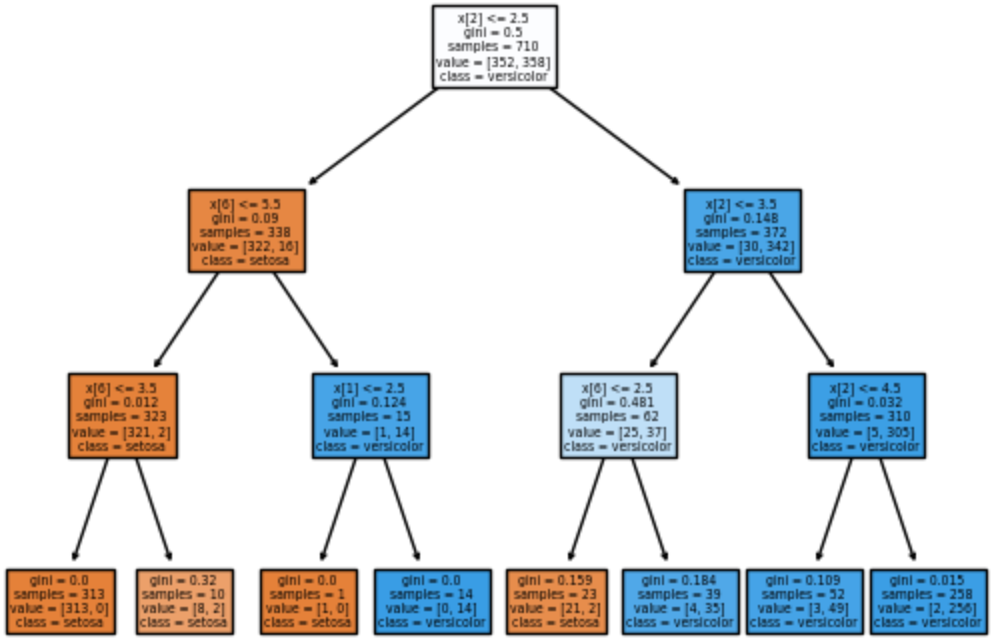
```
X
```

Out[225]:

```
array([[1000025,       5,       1, ...,       3,       1,       1],
       [1002945,       5,       4, ...,       3,       2,       1],
       [1015425,       3,       1, ...,       3,       1,       1],
       ...,
       [ 888820,       5,      10, ...,       8,      10,       2],
       [ 897471,       4,       8, ...,      10,       6,       1],
       [ 897471,       4,       8, ...,      10,       4,       1]],
      dtype=int64)
```

In [235…

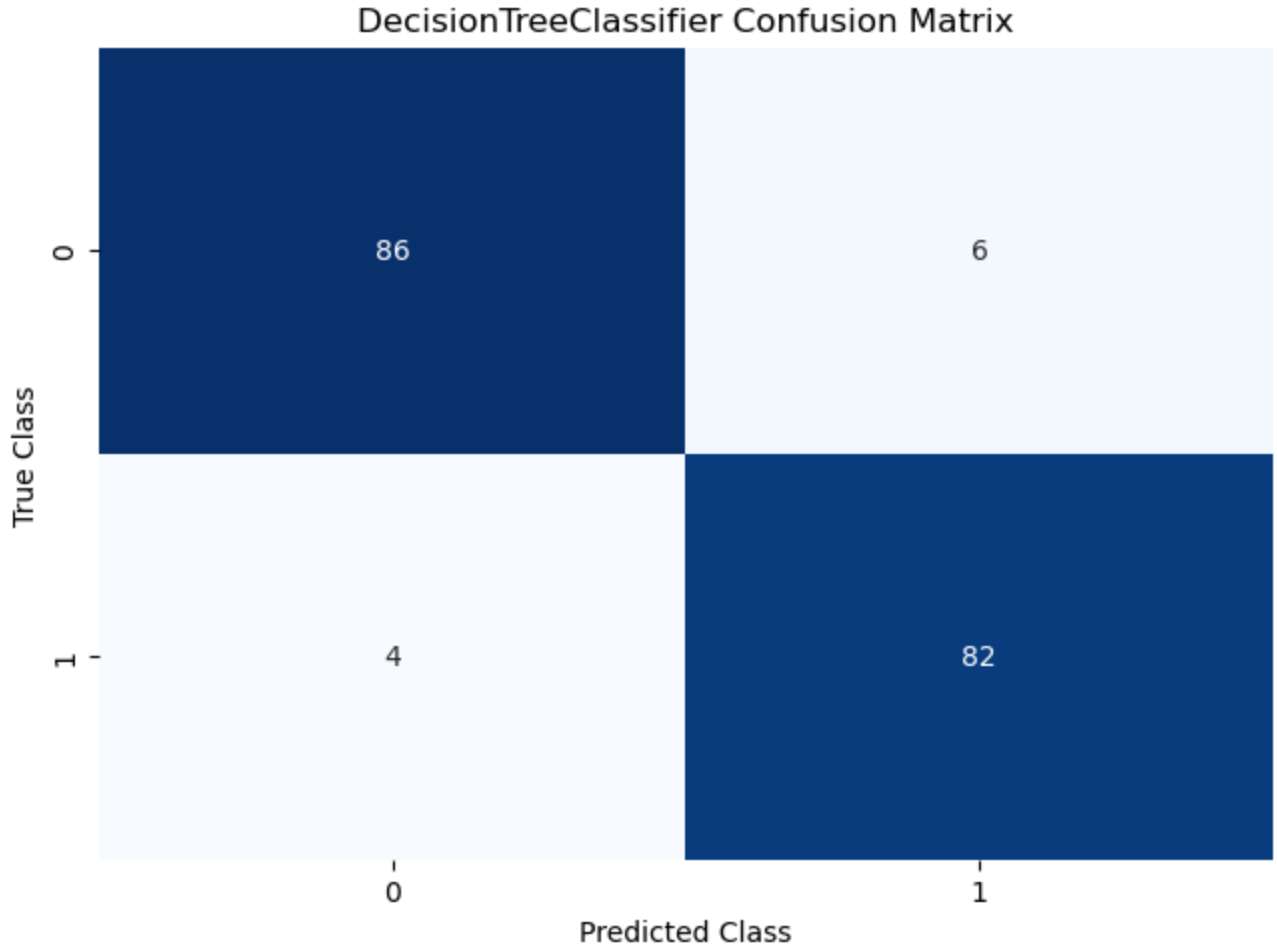
```
# Plot the decision tree
cn=['setosa', 'versicolor', 'virginica']
fig, axes = plt.subplots(nrows = 1,ncols = 1)
tree.plot_tree(clf,
               class_names=cn,
               filled = True);
fig.savefig('dtc.png')
```



In [202…

```
#Build the confusion matrix
matrix_sm = confusion_matrix(y_test_sm, y_pred_dtc)

print(matrix_sm)

# Create pandas dataframe
df_sm = pd.DataFrame(matrix_sm)

# Create a heatmap
sns.heatmap(df_sm, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("DecisionTreeClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[86  6]
 [ 4 82]]
```



In [203…

```
#Print results
print(classification_report(y_test_sm, y_pred_dtc))
print(confusion_matrix(y_test_sm, y_pred_dtc))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_dtc)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_dtc)}')
```

```
              precision    recall  f1-score   support

           2       0.96      0.93      0.95        92
           4       0.93      0.95      0.94        86

    accuracy                           0.94       178
   macro avg       0.94      0.94      0.94       178
weighted avg       0.94      0.94      0.94       178

[[86  6]
 [ 4 82]]
ROC-AUC score : 0.9441354903943378
Accuracy score : 0.9438202247191011
```

- The model performs reasonably well for both classes, with relatively high precision, recall, and F1-score values. It correctly identifies around 96% of benign cases (class 2) and 93% of malignant cases (class 4).
- The confusion matrix indicates that there were 6 false positive predictions and 4 false negative predictions. While the model is performing well overall, these misclassifications should be considered in the context of the application.
- The ROC-AUC score of 0.944 and accuracy score of 0.944 demonstrate the model's effectiveness in distinguishing between benign and malignant cases

KNN - KNeighborsClassifier

In [205…

```
k_range = list(range(1, 31))
weight_options = ['uniform', 'distance']
metric_options=['minkowski','euclidean','manhattan','hamming']
param_grid = dict(n_neighbors=k_range, weights=weight_options, metric=metric_options)
print(param_grid)

knn = KNeighborsClassifier()

grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', return_train_score=False
grid.fit(x, y)
```

```
{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30], 'weights': ['uniform', 'distance'], 'metric':
['minkowski', 'euclidean', 'manhattan', 'hamming']}
```

Out[205]:

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

In [206…

```
grid_mean_scores = grid.cv_results_['mean_test_score']
print(grid_mean_scores)
```

```
[0.58780644 0.58780644 0.56647089 0.58780644 0.59579928 0.59909346
 0.6026047  0.59794433 0.59136874 0.59798264 0.5857763  0.60137896
 0.56783708 0.60255363 0.55882278 0.60143003 0.56329162 0.60255363
 0.54977017 0.6081716  0.55085546 0.60591164 0.56326609 0.60707354
 0.54754852 0.60255363 0.54869765 0.59918284 0.53859806 0.60033197
 0.52618744 0.59919561 0.51713483 0.59581205 0.5148238  0.59466292
 0.51936925 0.59805924 0.51938202 0.59807201 0.53065628 0.59808478
 0.52950715 0.59581205 0.52611083 0.59356486 0.53059244 0.59693565
 0.53627426 0.59582482 0.5238764  0.59694842 0.52276558 0.5924668
 0.51824566 0.59583759 0.51943309 0.59020684 0.5182712  0.59020684
 0.58780644 0.58780644 0.56647089 0.58780644 0.59579928 0.59909346
 0.6026047  0.59794433 0.59136874 0.59798264 0.5857763  0.60137896
 0.56783708 0.60255363 0.55882278 0.60143003 0.56329162 0.60255363
 0.54977017 0.6081716  0.55085546 0.60591164 0.56326609 0.60707354
 0.54754852 0.60255363 0.54869765 0.59918284 0.53859806 0.60033197
 0.52618744 0.59919561 0.51713483 0.59581205 0.5148238  0.59466292
 0.51936925 0.59805924 0.51938202 0.59807201 0.53065628 0.59808478
 0.52950715 0.59581205 0.52611083 0.59356486 0.53059244 0.59693565
 0.53627426 0.59582482 0.5238764  0.59694842 0.52276558 0.5924668
 0.51824566 0.59583759 0.51943309 0.59020684 0.5182712  0.59020684
 0.59796987 0.59796987 0.58449949 0.60021706 0.60591164 0.60920582
 0.60824821 0.61031665 0.59587589 0.60924413 0.58914709 0.61378958
 0.56896067 0.61606231 0.55882278 0.61156793 0.56553882 0.61608784
 0.55201736 0.61721144 0.55199183 0.61944586 0.56326609 0.61949694
 0.54979571 0.61385342 0.55209397 0.61271706 0.53859806 0.61497702
 0.5273238  0.61048264 0.52053115 0.60709908 0.52044178 0.60708631
 0.51936925 0.60822268 0.51938202 0.611619   0.53065628 0.60823544
 0.52950715 0.6104954  0.52948161 0.60372829 0.53059244 0.60823544
 0.53739785 0.60486466 0.5238764  0.60600102 0.52388917 0.60487743
 0.51938202 0.60600102 0.5183095  0.60487743 0.5182712  0.60149387
 0.95610317 0.95610317 0.94593973 0.95271961 0.96625383 0.96625383
 0.96174668 0.96850102 0.96851379 0.96851379 0.96174668 0.96851379
 0.96624106 0.96624106 0.95948672 0.96399387 0.95948672 0.95948672
 0.95836313 0.96399387 0.96061032 0.96061032 0.95835036 0.96396834
 0.96173391 0.96285751 0.95722676 0.96172114 0.95835036 0.95835036
 0.95497957 0.96059755 0.95722676 0.95722676 0.95386874 0.96061032
 0.95836313 0.95836313 0.95611593 0.95836313 0.95948672 0.95948672
 0.95499234 0.95836313 0.95948672 0.96061032 0.95274515 0.95836313
```

```
 0.95611593 0.95723953 0.95049796 0.9572523  0.95162155 0.95274515
 0.94825077 0.95500511 0.94938713 0.95163432 0.94937436 0.95275792]
```

In [207…

```
pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
```

Out[207]:

	mean_test_score	std_test_score	params
0	0.587806	0.064817	{'metric': 'minkowski', 'n_neighbors': 1, 'wei...
1	0.587806	0.064817	{'metric': 'minkowski', 'n_neighbors': 1, 'wei...
2	0.566471	0.084582	{'metric': 'minkowski', 'n_neighbors': 2, 'wei...
3	0.587806	0.064817	{'metric': 'minkowski', 'n_neighbors': 2, 'wei...
4	0.595799	0.089004	{'metric': 'minkowski', 'n_neighbors': 3, 'wei...
...	...	...	...
235	0.955005	0.020677	{'metric': 'hamming', 'n_neighbors': 28, 'weig...
236	0.949387	0.023114	{'metric': 'hamming', 'n_neighbors': 29, 'weig...
237	0.951634	0.023010	{'metric': 'hamming', 'n_neighbors': 29, 'weig...
238	0.949374	0.022578	{'metric': 'hamming', 'n_neighbors': 30, 'weig...
239	0.952758	0.023420	{'metric': 'hamming', 'n_neighbors': 30, 'weig...

240 rows × 3 columns

In [208…

```
print(grid.best_score_)
print(grid.best_params_)
```

```
0.9685137895812053
{'metric': 'hamming', 'n_neighbors': 5, 'weights': 'uniform'}
```

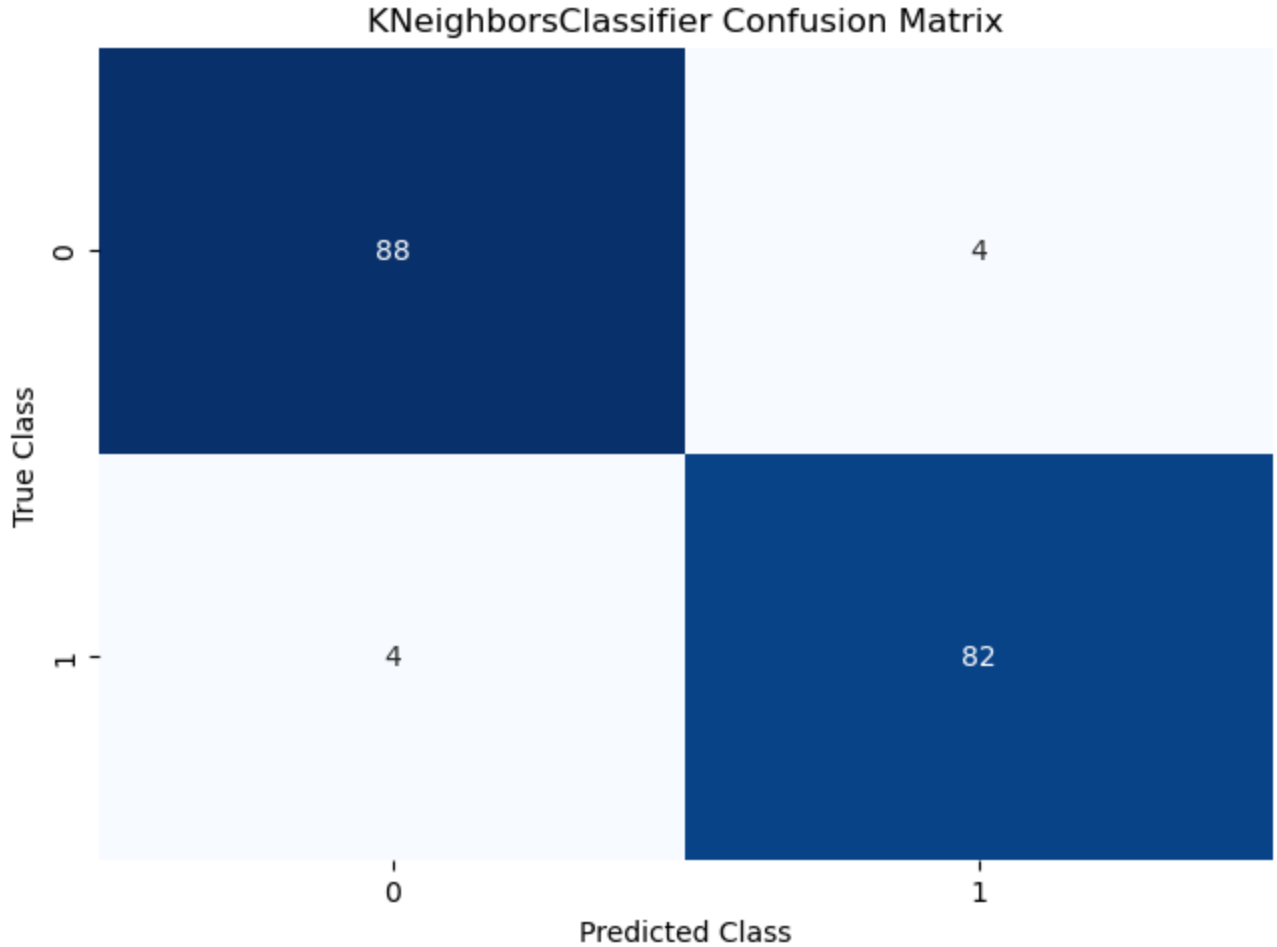
In [209…

```
# Using n_neighbors = 5 for best model performance
neighbors = KNeighborsClassifier(n_neighbors=5, weights='uniform',metric= 'hamming')
neighbors.fit(x_train_sm, y_train_sm)
y_pred_knn = neighbors.predict(x_test_sm)
```

In [210…

```
#Build the confusion matrix
matrix_sm = confusion_matrix(y_test_sm, y_pred_knn)

print(matrix_sm)

# Create pandas dataframe
df_sm = pd.DataFrame(matrix_sm)

# Create a heatmap
sns.heatmap(df_sm, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("KNeighborsClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[88  4]
 [ 4 82]]
```



In [211…

```
print(classification_report(y_test_sm, y_pred_knn))
print(confusion_matrix(y_test_sm, y_pred_knn))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_knn)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_knn)}')
```

```
              precision    recall  f1-score   support

           2       0.96      0.96      0.96        92
           4       0.95      0.95      0.95        86

    accuracy                           0.96       178
   macro avg       0.96      0.96      0.96       178
weighted avg       0.96      0.96      0.96       178

[[88  4]
 [ 4 82]]
ROC-AUC score : 0.955005055611729
Accuracy score : 0.9550561797752809
```

- The KNeighborsClassifier model performs well for both classes, with high precision, recall, and F1-score values. It correctly identifies around 96% of benign cases (class 2) and 95% of malignant cases (class 4).
- The confusion matrix indicates that there were 4 false positive predictions and 4 false negative predictions. These misclassifications should be considered in the context of the application.
- The ROC-AUC score of 0.955 and accuracy score of 0.955 demonstrate the model's effectiveness in distinguishing between benign and malignant cases.

Interpreting the Results:

Considering we are dealing with a small dataset with not many features, the initial assumption was that KNN would have better accuracy over the Random Forest model. But as we can see, the Random Forest model performs better than the KNN with a 96.6% accuracy. This outcome emphasizes the importance of empirical evaluation and testing assumptions. The Random Forest Classifier exhibited the highest accuracy of 96.6%, showcasing its proficiency in distinguishing between benign and malignant cases. It demonstrated robust precision, recall, and F1-scores for both classes, underscoring its overall effectiveness. In contrast, the Decision Tree achieved an accuracy of 94%, displaying slightly lower performance compared to the Random Forest. The KNN model, after fine-tuning its parameters, achieved an accuracy of 95.5%. This outcome makes us rethink assumptions and shows how complex factors affect models. Random Forest's success suggests it's great for breast cancer prediction, challenging our original idea.

Initial Conclusion and Recommendation:

Conclusion:

The primary objective of this project is to analyze the breast cancer dataset and develop a predictive model for breast cancer. By constructing and comparing different models, we aim to identify the most effective approach for this prediction task. We implemented three models: RandomForest, DecisionTree, and KNN, in order to determine the optimal model for breast cancer prediction. The ROC-AUC score and accuracy score of the models demonstrate the model's effectiveness in distinguishing between benign and malignant cases. All three models perform well, with accuracy scores in the range of 94 to 97%, indicating strong predictive capabilities. The Random Forest and KNN models show slightly higher precision, recall, and F1-scores compared to the Decision Tree. The ROC-AUC scores for all models are above 94, indicating their capacity to discriminate between classes. Overall, each model demonstrates effectiveness in predicting breast cancer based on the provided metrics. The Random Forest and KNN models particularly stand out with slightly higher performance, but the Decision Tree model is also competitive.

Recommendations:

I would like to build an API/model that could be used by patients to input their symptoms and be able to predict the possibility of a benign or malignant tumor. Although the Random Forest Classifier exhibited the highest level of accuracy, it may be beneficial to conduct a more comprehensive examination of feature importance and potential overfitting. Rigorous regression testing is crucial before deploying the API/model to minimize false positives and false negatives.

Limitations:

The study relies on a limited and potentially outdated dataset. A more recent and comprehensive dataset could provide a more accurate representation of current trends and factors influencing breast cancer prediction. Additionally, the dataset's features might not capture all relevant factors that contribute to breast cancer prediction. Additional clinical, genetic, or lifestyle-related features could enhance the model's accuracy.

Risks:

Predictive models in the medical domain pose a risk of false positives (classifying benign as malignant) and false negatives (missing malignant cases). False positives could trigger unnecessary distress and invasive procedures, straining healthcare resources. False negatives may lead to delayed treatment, harming patient outcomes and intervention efficacy. Thorough testing is vital to minimize these risks and ensure patient well-being.

References

1. Pmotta. (2021, June 6). Breast cancer prediction. Kaggle https://www.kaggle.com/code/pmotta/breast-cancer-prediction/input

2. Patel, J., Patel , U., Patel, R., & Shah, P. (2019, April 28). Breast Cancer Analysis. https://rstudio-pubs-static.s3.amazonaws.com/491489_b86f191488ab4ed0a37e7a95c839a8f4.html