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)
```

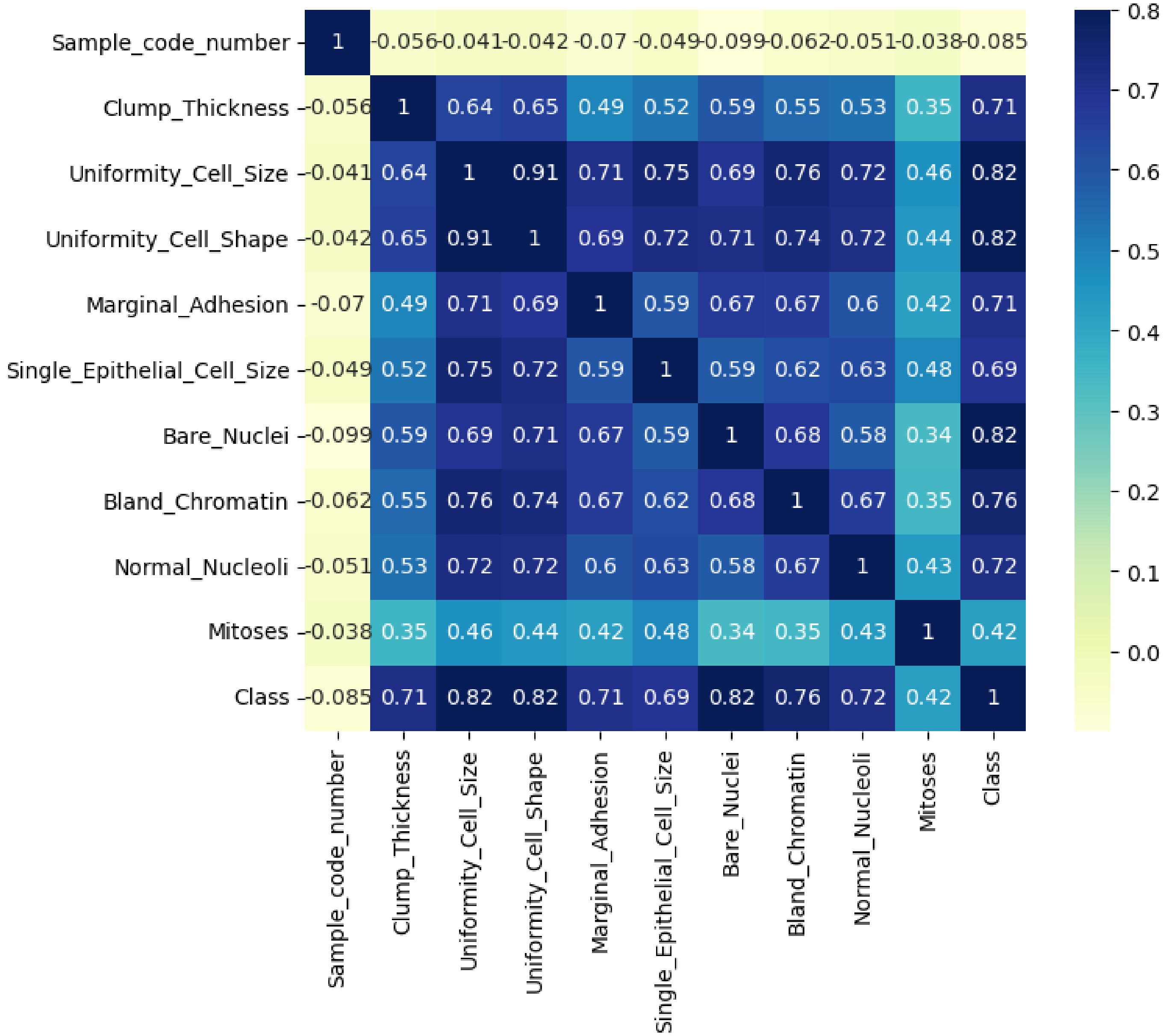
Data is classified into two classes benign and malignant

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]:

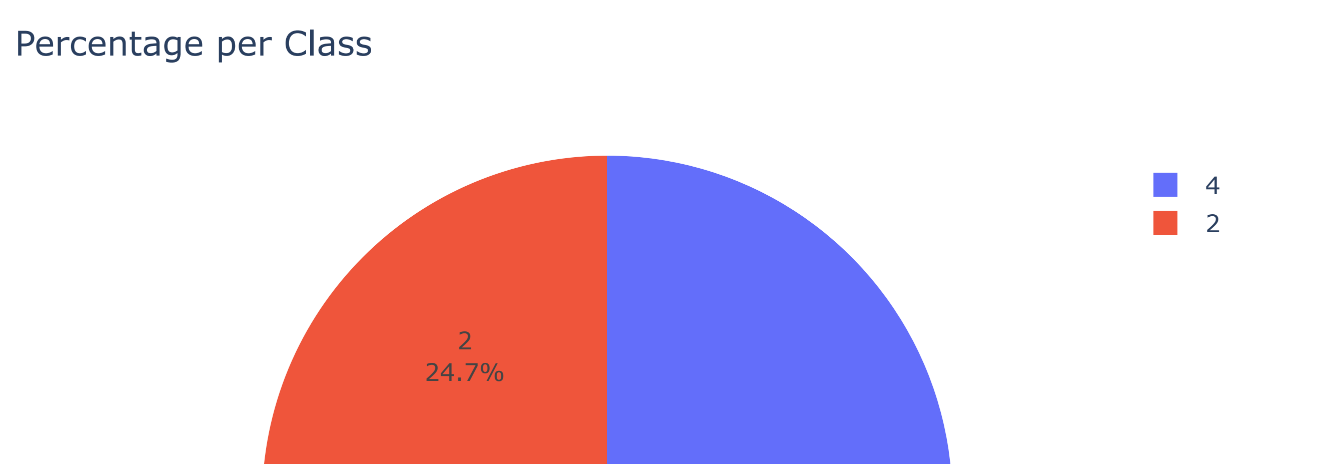
```
sns.set(font_scale=1)
sns.set_theme('notebook', style='dark')
plt.style.use("grayscale")

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'])
```

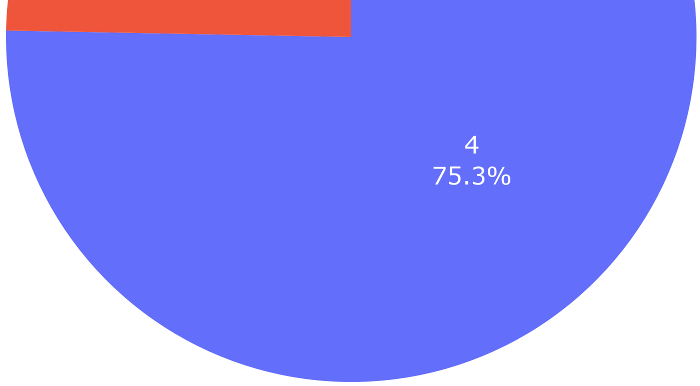
Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x17da0618c40>
```

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 [10]:
```
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.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

In [11]:
```
X = breast_cancer_df.drop(columns="Class").values
Y = breast_cancer_df["Class"].values
```

In [12]:
```
X.shape, Y.shape
```

Out[12]:
```
((683, 10), (683,))
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

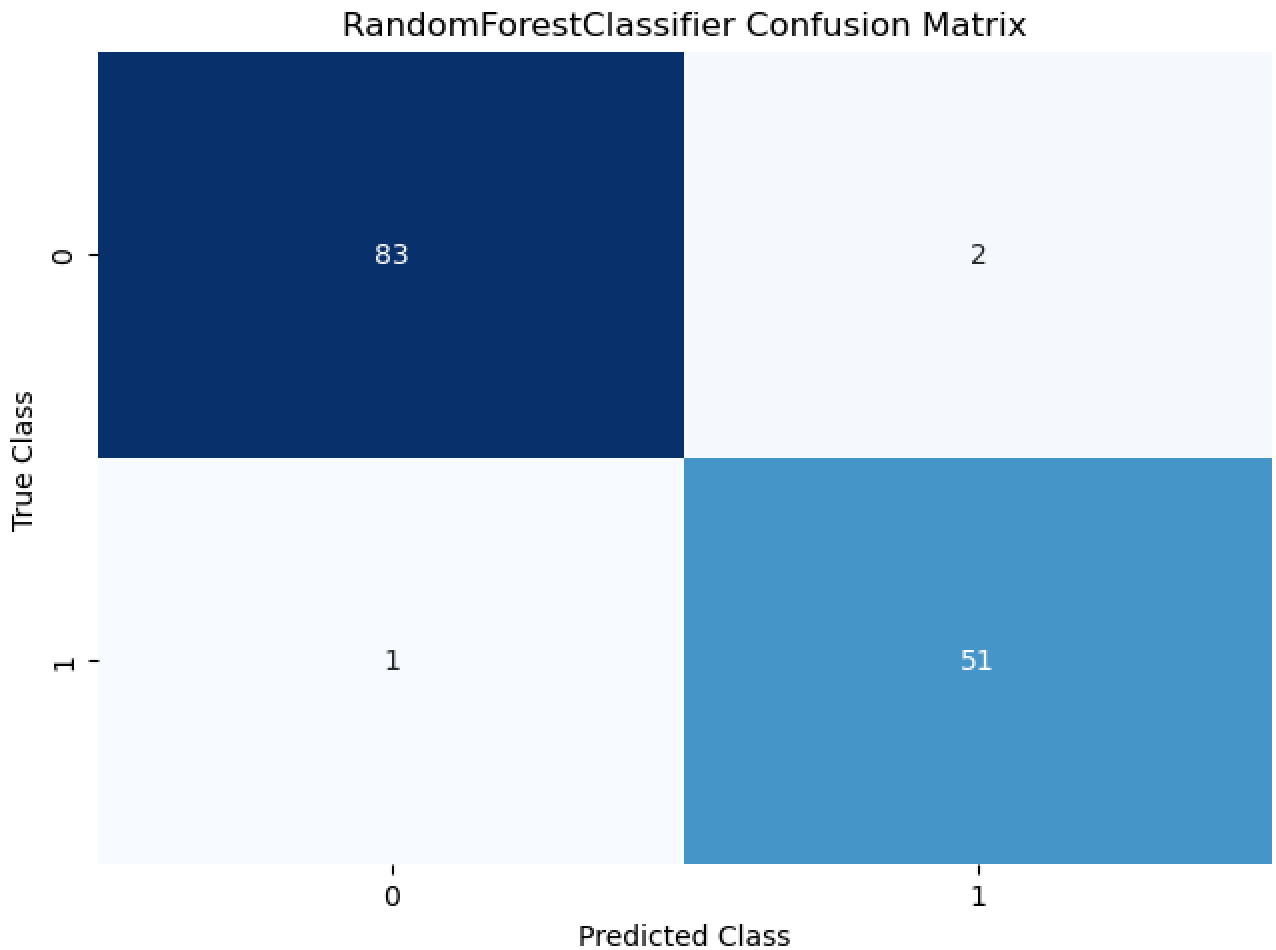
RandomForestClassifier with imbalanced data

In [14]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
```

In [15]:
```
#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()
```

```
[[83  2]
 [ 1 51]]
```



In [16]:
```
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      0.98      0.98        85
           4       0.96      0.98      0.97        52

    accuracy                           0.98       137
   macro avg       0.98      0.98      0.98       137
weighted avg       0.98      0.98      0.98       137

[[83  2]
 [ 1 51]]
ROC-AUC score : 0.9786199095022624
Accuracy score : 0.9781021897810219
```

SMOTE to balance the imbalanced data

In [17]:
```
smote = SMOTE()
x, y = smote.fit_resample(X, Y)
```

In [18]:
```
#Split the smote (balanced) data into random train and test subsets:
```

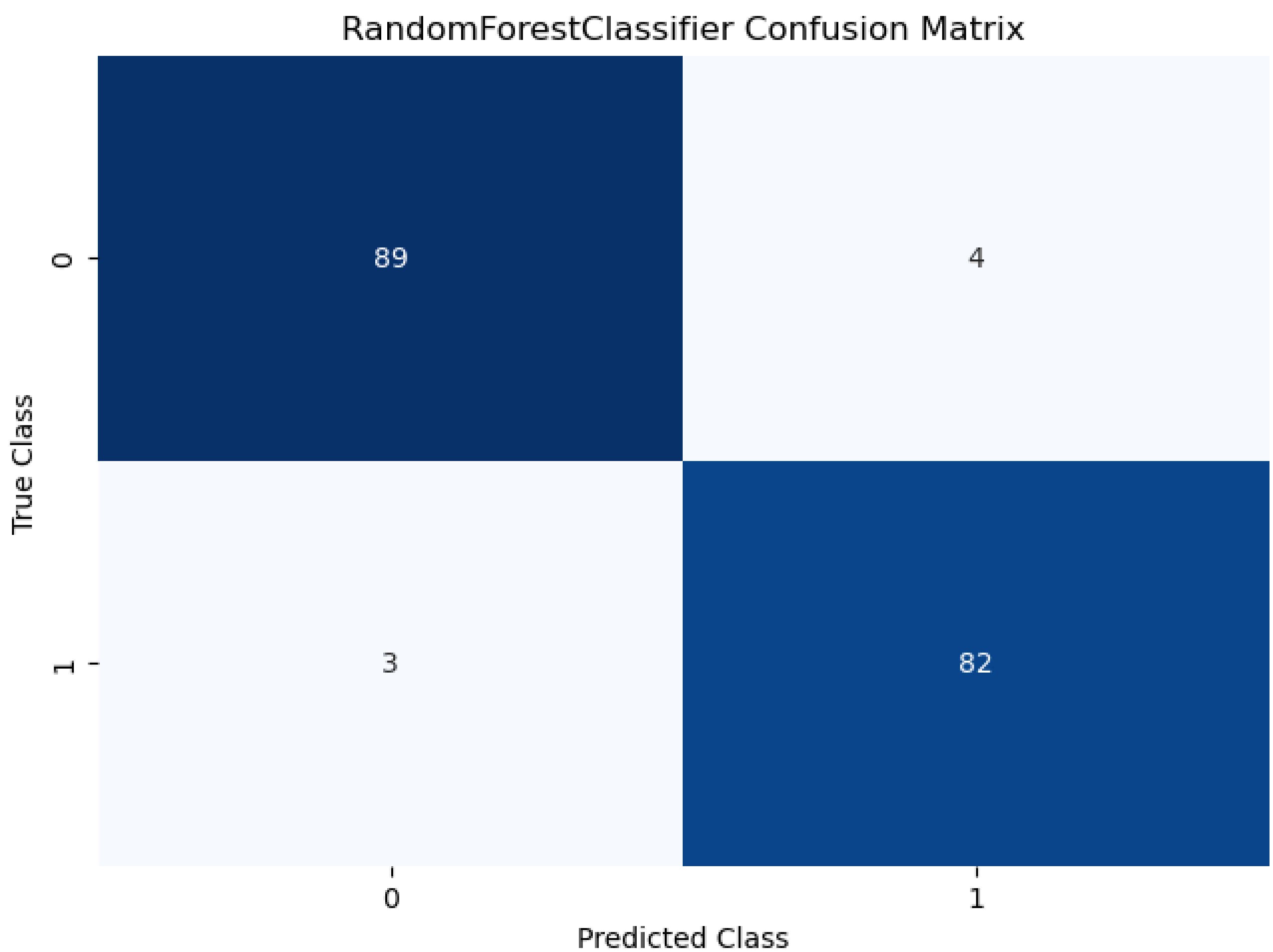
```
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x,y,test_size = 0.2)
```

RandomForestClassifier with balanced data

In [19]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train_sm, y_train_sm)

y_pred_sm = rfc.predict(x_test_sm)
```

In [20]:
```
#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()
```

```
[[89  4]
 [ 3 82]]
```



In [21]:
```
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       0.97      0.96      0.96        93
```

```
           4       0.95      0.96      0.96        85

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

[[89  4]
 [ 3 82]]
ROC-AUC score : 0.9608475648323846
Accuracy score : 0.9606741573033708
```

DecisionTreeClassifier

In [22]:

```
# Use the DecisionTreeClassifier to fit data
clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
clf.fit(x_train_sm, y_train_sm)
```

Out[22]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [23]:

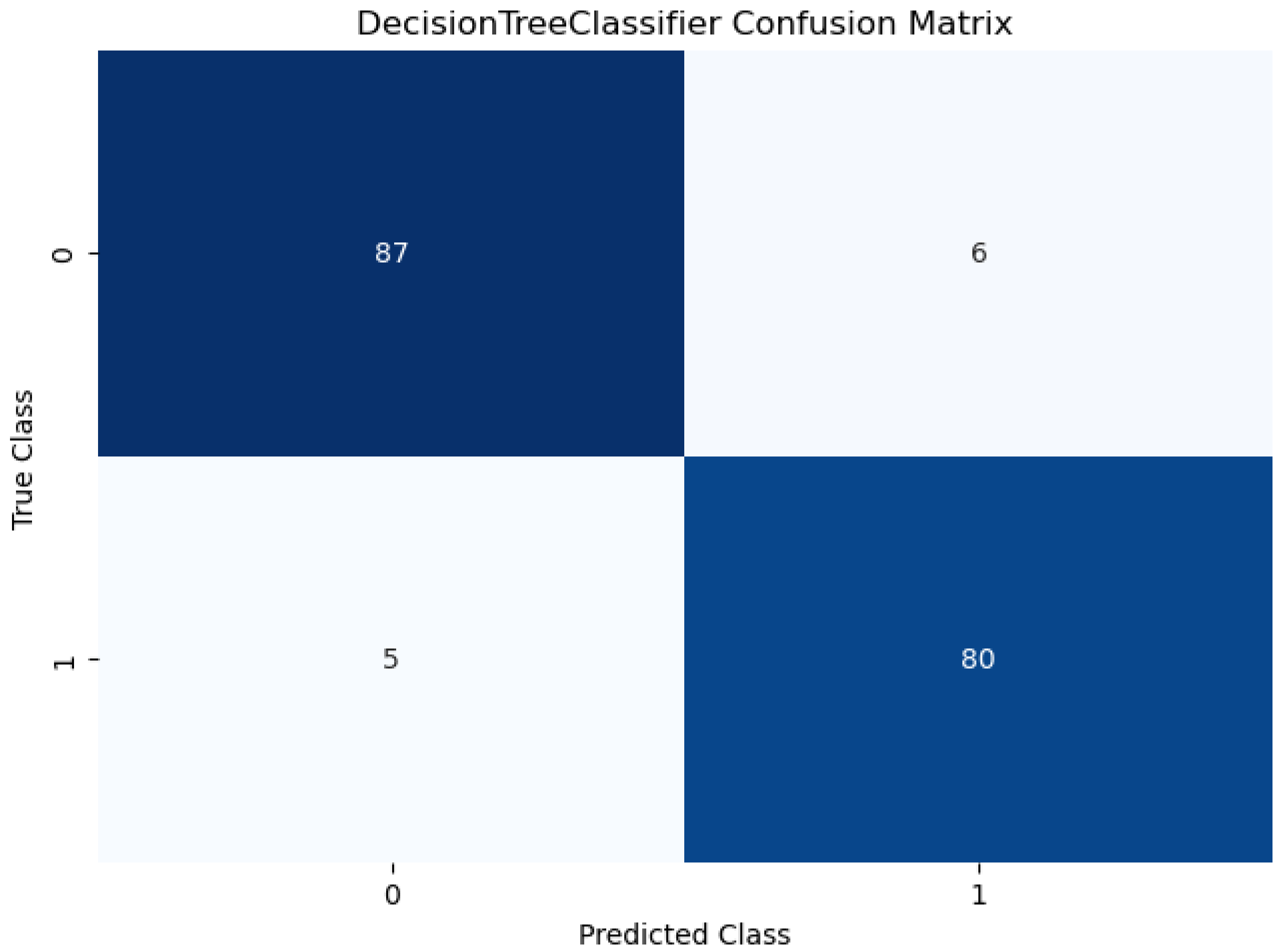
```
#Predict y data with classifier:
y_pred_dtc = clf.predict(x_test_sm)

#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       0.95      0.94      0.94        93
           4       0.93      0.94      0.94        85

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

[[87  6]
 [ 5 80]]
ROC-AUC score : 0.9383301707779887
Accuracy score : 0.9382022471910112
```

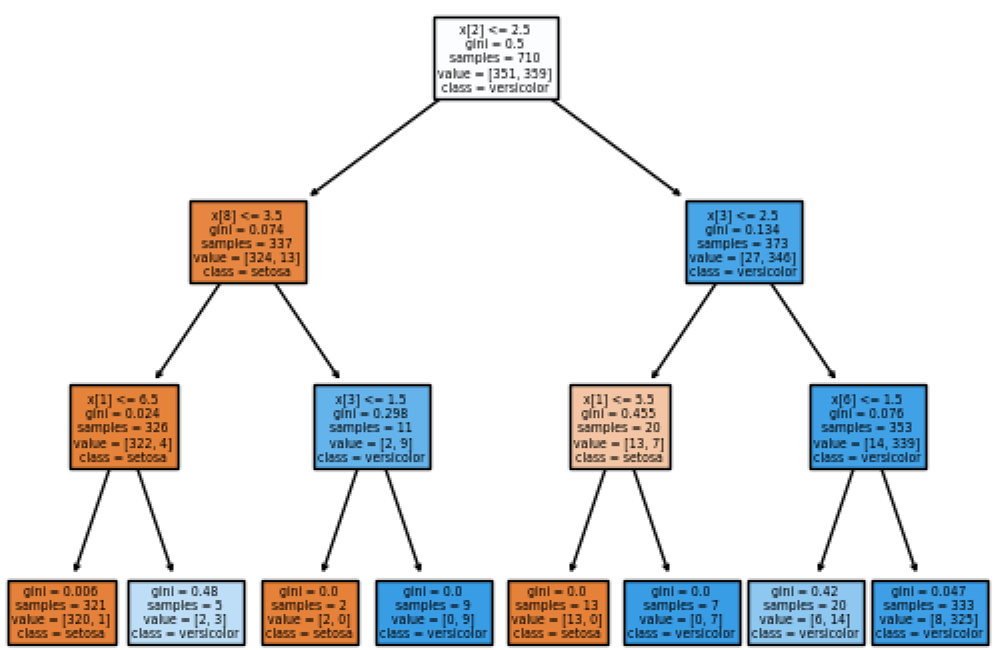
In [24]:

```
#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()
```

```
[[87  6]
 [ 5 80]]
```



In [25]:
```python
# Draw graph

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')
```



KNeighborsClassifier

In [26]:
```
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[26]:

GridSearchCV

estimator: KNeighborsClassifier

KNeighborsClassifier

In [27]:
```
grid_mean_scores = grid.cv_results_['mean_test_score']
print(grid_mean_scores)
```

```
[0.59011747 0.59011747 0.57208887 0.59011747 0.56763279 0.57545965
 0.55072778 0.57773238 0.55416241 0.56760725 0.52256129 0.56876915
 0.53504852 0.56650919 0.52146323 0.55975485 0.5282048  0.56200204
 0.51124872 0.55975485 0.52477017 0.56762002 0.51698161 0.56649642
 0.5102145  0.56424923 0.50684372 0.56424923 0.50455822 0.56084014
 0.51588355 0.56085291 0.50458376 0.56196374 0.49891471 0.56196374
 0.49777835 0.56197651 0.49336057 0.5631001  0.50007661 0.56086568
 0.48762768 0.56196374 0.50229826 0.56197651 0.48767875 0.56082737
 0.49891471 0.56421093 0.49449694 0.56195097 0.48094995 0.55858018
 0.48882789 0.55745659 0.48877681 0.55970378 0.48544433 0.55745659
 0.59011747 0.59011747 0.57208887 0.59011747 0.56763279 0.57545965
 0.55072778 0.57773238 0.55416241 0.56760725 0.52256129 0.56876915
 0.53504852 0.56650919 0.52146323 0.55975485 0.5282048  0.56200204
 0.51124872 0.55975485 0.52477017 0.56762002 0.51698161 0.56649642
 0.5102145  0.56424923 0.50684372 0.56424923 0.50455822 0.56084014
 0.51588355 0.56085291 0.50458376 0.56196374 0.49891471 0.56196374
 0.49777835 0.56197651 0.49336057 0.5631001  0.50007661 0.56086568
 0.48762768 0.56196374 0.50229826 0.56197651 0.48767875 0.56082737
 0.49891471 0.56421093 0.49449694 0.56195097 0.48094995 0.55858018
 0.48882789 0.55745659 0.48877681 0.55970378 0.48544433 0.55745659
 0.60363892 0.60363892 0.58111593 0.60363892 0.57438713 0.59348825
 0.55523493 0.59013023 0.55866956 0.58111593 0.52593207 0.58340143
 0.53617211 0.57777068 0.52371042 0.57212717 0.53499745 0.57437436
 0.51352145 0.57101634 0.5270429  0.57886874 0.52259959 0.57889428
 0.51359806 0.57437436 0.50684372 0.57775792 0.50681818 0.56985444
 0.51926711 0.57774515 0.50570735 0.5721144  0.50118744 0.573238
 0.49777835 0.5721144  0.49336057 0.57549796 0.5034474  0.57212717
 0.48875128 0.573238   0.50457099 0.57322523 0.4921859  0.56984168
 0.50117467 0.57433606 0.49449694 0.5720761  0.48320991 0.5709525
 0.48996425 0.56758172 0.48877681 0.5720761  0.48769152 0.56758172
 0.96732635 0.96732635 0.94032176 0.95829928 0.96851379 0.97076098
 0.96289581 0.96963739 0.96739019 0.96963739 0.96174668 0.96963739
 0.96963739 0.96963739 0.96288304 0.97076098 0.96737743 0.96737743
 0.96740296 0.97077375 0.965143   0.96739019 0.96177222 0.96740296
 0.96627937 0.96852656 0.96288304 0.96739019 0.96401941 0.96739019
 0.96177222 0.96739019 0.96627937 0.96852656 0.96064862 0.9662666
 0.96515577 0.96740296 0.96064862 0.96515577 0.96064862 0.96289581
 0.95726507 0.95838866 0.95951226 0.96063585 0.95726507 0.96064862
```

```
 0.95951226 0.95951226 0.95726507 0.95951226 0.95952503 0.96064862
 0.95614147 0.96064862 0.95952503 0.96177222 0.95726507 0.95952503]
```

In [28]:
```python
pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
```

Out[28]:

	mean_test_score	std_test_score	params
0	0.590117	0.071843	{'metric': 'minkowski', 'n_neighbors': 1, 'wei...
1	0.590117	0.071843	{'metric': 'minkowski', 'n_neighbors': 1, 'wei...
2	0.572089	0.077953	{'metric': 'minkowski', 'n_neighbors': 2, 'wei...
3	0.590117	0.071843	{'metric': 'minkowski', 'n_neighbors': 2, 'wei...
4	0.567633	0.101204	{'metric': 'minkowski', 'n_neighbors': 3, 'wei...
...	...	...	...
235	0.960649	0.026692	{'metric': 'hamming', 'n_neighbors': 28, 'weig...
236	0.959525	0.028498	{'metric': 'hamming', 'n_neighbors': 29, 'weig...
237	0.961772	0.026669	{'metric': 'hamming', 'n_neighbors': 29, 'weig...
238	0.957265	0.025978	{'metric': 'hamming', 'n_neighbors': 30, 'weig...
239	0.959525	0.025207	{'metric': 'hamming', 'n_neighbors': 30, 'weig...

240 rows × 3 columns

In [29]:
```python
print(grid.best_score_)
print(grid.best_params_)
```

```
0.9707737487231869
{'metric': 'hamming', 'n_neighbors': 10, 'weights': 'distance'}
```

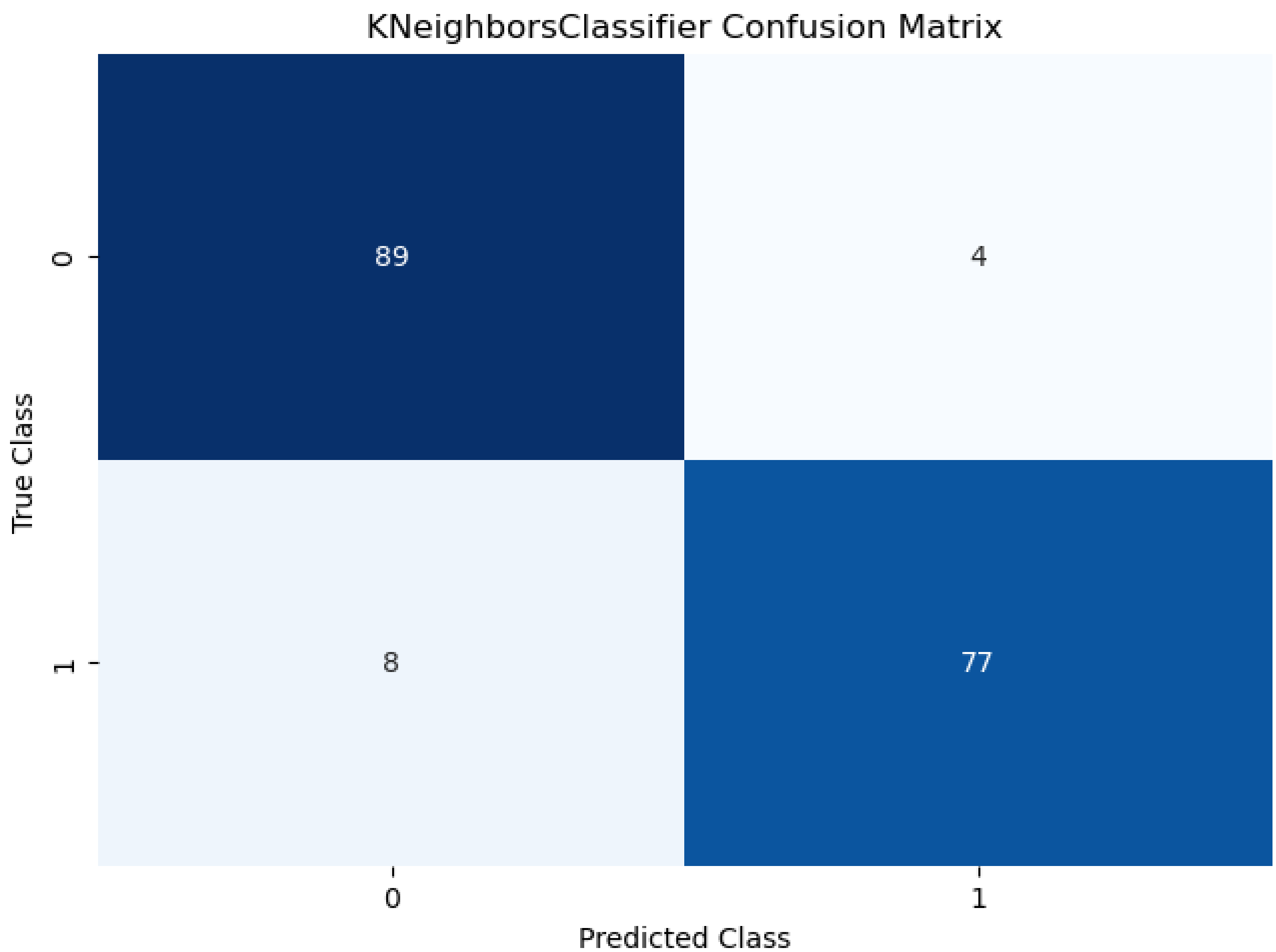
In [30]:
```python
# Using n_neighbors = 5 for best model performance
neighbors = KNeighborsClassifier(n_neighbors=5, weights='distance',metric= 'hamming')
neighbors.fit(x_train_sm, y_train_sm)
y_pred_knn = neighbors.predict(x_test_sm)

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       0.92      0.96      0.94        93
           4       0.95      0.91      0.93        85

    accuracy                           0.93       178
   macro avg       0.93      0.93      0.93       178
weighted avg       0.93      0.93      0.93       178

[[89  4]
 [ 8 77]]
ROC-AUC score : 0.9314358001265023
Accuracy score : 0.9325842696629213
```

In [31]:
```python
#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()
```

```
[[89  4]
 [ 8 77]]
```



Conclusion

The primary goal of the project is to analyze breast cancer dataset to be able to build a model that can help predict breast cancer. Building different models allows us to compare the performance to be able to select the best performing model.

Three models RandomForest, DecisionTree and KNN were implemented to compare the best performing model for breast cancer prediction.

Since the data was imbalanced, SMOTE technique was used to balance the data.

The difference in the performance between the models is minimal with over 97% accuracy across all three models. However, looking at the confusion matrix and accuracy score, the RandomForest model has the best performance over KNN and DecisionTree models.

Limitations

The dataset used is not recent and is also limited in terms of dataset size. In my opinion, the models should be tested with a more recent dataset with additional parameters that capture more symptoms that can be accounted for breast cancer prediction.

Recomendations:

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.

Risk:

Thourough regression testing will be required to publicise this model because there is a risk of false positives that could lead to unnecessary panic in patients.

References

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