Table of Contents

1. Introduction
2. Required Modules
3. Data Preprocessing
    - Key Features
    - Descriptive Statistics
    - Missing Values
    - Data Transformation
4. Exploratory Data Analysis
    - Visualization
    - Correlation
5. Model Building
    - Naive Bayes Classifier
    - K-Nearest Neighbours
6. Evaluation
7. Conclusion

1. Introduction

In this project, we aim to utilize both the extracted features from MRI images and data obtained from the ADNI (Alzheimer's Disease Neuroimaging Initiative) database to develop a classification model for Alzheimer's disease patients.

The objective of this project is to leverage a dataset consisting of 220 samples, each with 16 features extracted from MRI images, to train a classifier model. This model will enable the classification of subjects into two categories: those with cognitive normalcy and those diagnosed with Alzheimer's disease. By incorporating both MRI-based features and relevant data from the ADNI database, we aim to develop an accurate and reliable model that can aid in the early detection and diagnosis of Alzheimer's disease.

2. Required Modules

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

3. Data Preprocessing

In this section, we discuss the steps taken to preprocess the dataset. Through preprocessing steps, including data cleaning, handling missing values, and converting categorical variables into numerical formats, we will ensure the dataset is ready for analysis.

3.1 Key Features

The dataset contains the following information:

In [ ]:
```
# Load the data
data = pd.read_csv('Data_training_and_val_set_CNvsAD.csv')
data.head(-1)
```

Out[ ]:

	RID	BRAIN	EICV	VENTRICLES	LHIPPOC	RHIPPOC	LINFLATVEN	RINFLATVEN	LMIDTEM
0	1063	937159	1401690	38855.4	3427.13	3262.01	1045.280	1263.010	2.4456
1	184	901429	1235930	10081.2	3410.35	3693.69	529.756	391.364	2.5210
2	16	935565	1345440	21299.0	3621.49	3802.43	887.978	1135.170	2.4512
3	575	884775	1325760	69220.3	2802.91	3098.38	3215.570	3370.800	2.5934
4	403	1062950	1498230	32967.7	3416.55	4103.49	1648.410	1480.410	2.7854
...	...	...	...	...	...	...	...	...	
214	310	880382	1411200	55534.1	2555.54	1957.66	2486.780	3299.060	2.0484
215	712	856859	1323290	64639.2	2889.79	2612.19	3275.190	6056.760	2.1435
216	1377	1097900	1716660	76240.9	2661.54	3167.81	2745.350	4233.470	2.1388
217	1254	973188	1561140	90250.4	2653.06	2605.33	3005.910	5159.180	2.4302
218	1337	1105000	1589110	41649.7	2321.26	2855.26	1902.090	1429.610	2.3685

219 rows × 17 columns

3.2 Descriptive Statistics

Now we generate the summary statistics:

- Count: The number of non-missing values in each column.
- Mean: The average value of each column.
- Standard Deviation: A measure of the amount of variation or dispersion in each column.
- Minimum: The minimum value in each column.

 which 25% of the data falls.

- Median (50th Percentile or Q2): The middle value in each column. It represents the value below which 50% of the data falls.
- 75th Percentile (Q3): The value below which 75% of the data falls.
- Maximum: The maximum value in each column.

In [ ]:
```
data.describe()
```

Out[ ]:

	RID	BRAIN	EICV	VENTRICLES	LHIPPOC	RHIPPOC	LINFLATVEI
count	220.000000	2.200000e+02	2.200000e+02	220.000000	220.000000	220.000000	220.00000
mean	697.381818	9.732298e+05	1.455089e+06	46819.713000	3147.849182	3315.357955	1742.21697
std	410.637915	1.085988e+05	1.530882e+05	25260.106084	628.351722	645.180948	1031.87153
min	7.000000	6.345910e+05	1.059560e+06	7552.460000	1723.310000	1768.950000	296.26400
25%	373.500000	8.992745e+05	1.343640e+06	28324.100000	2673.720000	2751.300000	1001.20425
50%	691.000000	9.744355e+05	1.454475e+06	42132.200000	3182.570000	3381.970000	1473.67500
75%	1067.000000	1.044310e+06	1.565938e+06	59839.475000	3554.165000	3806.862500	2169.99000
max	1430.000000	1.303590e+06	1.949090e+06	152927.000000	5514.940000	5750.400000	6082.04000

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220 entries, 0 to 219
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   RID         220 non-null    int64
 1   BRAIN       220 non-null    int64
 2   EICV        220 non-null    int64
 3   VENTRICLES  220 non-null    float64
 4   LHIPPOC     220 non-null    float64
 5   RHIPPOC     220 non-null    float64
 6   LINFLATVEN  220 non-null    float64
 7   RINFLATVEN  220 non-null    float64
 8   LMIDTEMP    220 non-null    float64
 9   RMIDTEMP    220 non-null    float64
 10  LINFTEMP    220 non-null    float64
 11  RINFTEMP    220 non-null    float64
 12  LFUSIFORM   220 non-null    float64
 13  RFUSIFORM   220 non-null    float64
 14  LENTORHIN   220 non-null    float64
 15  RENTORHIN   220 non-null    float64
 16  DXCURREN    220 non-null    int64
dtypes: float64(13), int64(4)
memory usage: 29.3 KB
```

3.3 Missing Values

In [ ]:
```
# Check for missing values in the dataset
```

Out[ ]:
```
Age                 0
Gender              0
Polyuria            0
Polydipsia          0
sudden weight loss  0
weakness            0
Polyphagia          0
Genital thrush      0
visual blurring     0
Itching             0
Irritability        0
delayed healing     0
partial paresis     0
muscle stiffness    0
Alopecia            0
Obesity             0
class               0
dtype: int64
```

In [ ]:
```
#checking the data types of the columns
data.dtypes
```

Out[ ]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

3.4 Data Transformation

In [ ]:
```
# divide the data into two... for each group of the subjects
Demented = data[data.DXCURREN == 1]
Normal = data[data.DXCURREN == 0]
```

In [ ]:
```
Demented.head(-1)
```

Out[ ]:

	RID	BRAIN	EICV	VENTRICLES	LHIPPOC	RHIPPOC	LINFLATVEN	RINFLATVEN	LMIDTEM
110	829	837158	1210510	19389.3	2494.80	2114.81	954.183	1392.89	2.3485
111	76	920332	1396340	35435.0	2411.57	2583.68	1543.810	2816.11	2.143(
112	724	982159	1610120	117066.0	3429.70	3481.37	3681.200	4653.46	1.5625
113	1307	1014660	1485850	45363.9	3369.99	3720.33	1644.340	1351.26	2.7001
114	1101	753428	1118990	23837.3	2296.70	2699.53	1587.660	1344.40	2.2511
...	...	...	...	...	...	...	...	...	
214	310	880382	1411200	55534.1	2555.54	1957.66	2486.780	3299.06	2.0484
215	712	856859	1323290	64639.2	2889.79	2612.19	3275.190	6056.76	2.1435
216	1377	1097900	1716660	76240.9	2661.54	3167.81	2745.350	4233.47	2.1388
217	1254	973188	1561140	90250.4	2653.06	2605.33	3005.910	5159.18	2.4302
218	1337	1105000	1589110	41649.7	2321.26	2855.26	1902.090	1429.61	2.3685

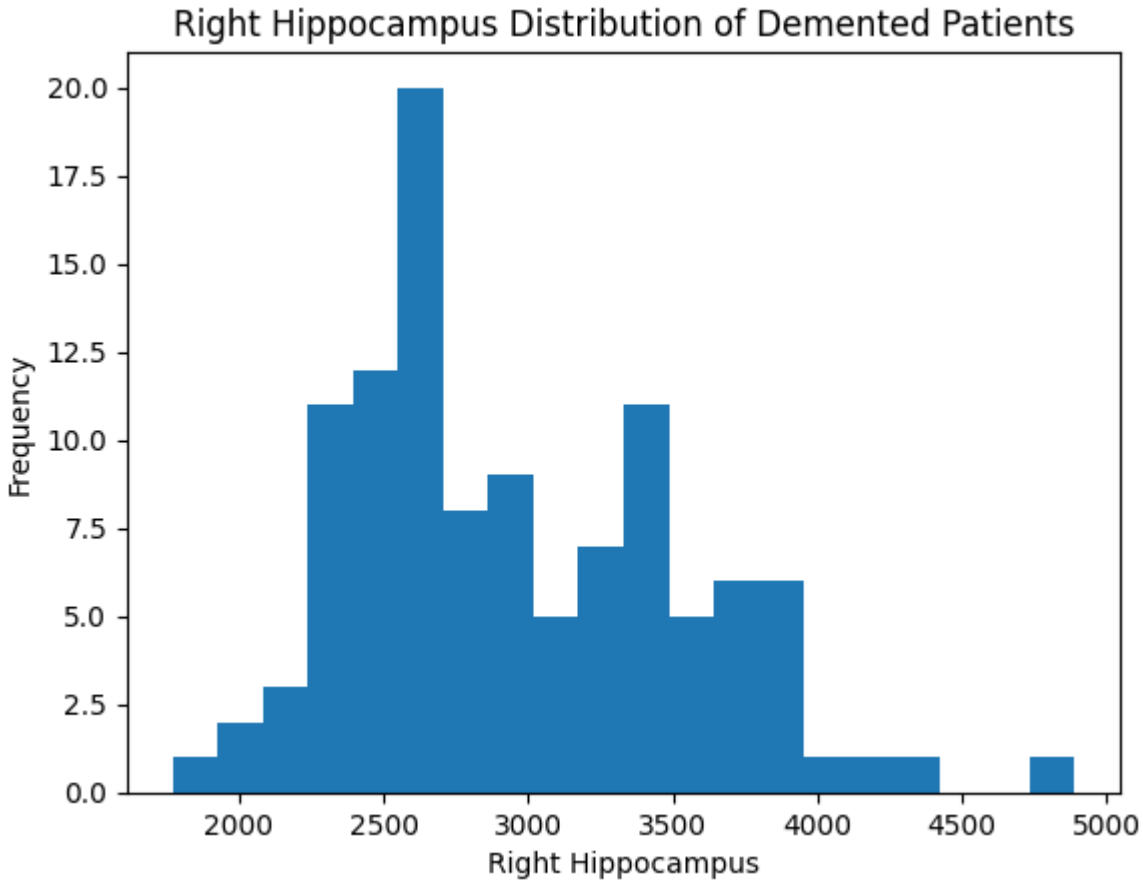
109 rows × 17 columns

4. Exploratory Data Analysis

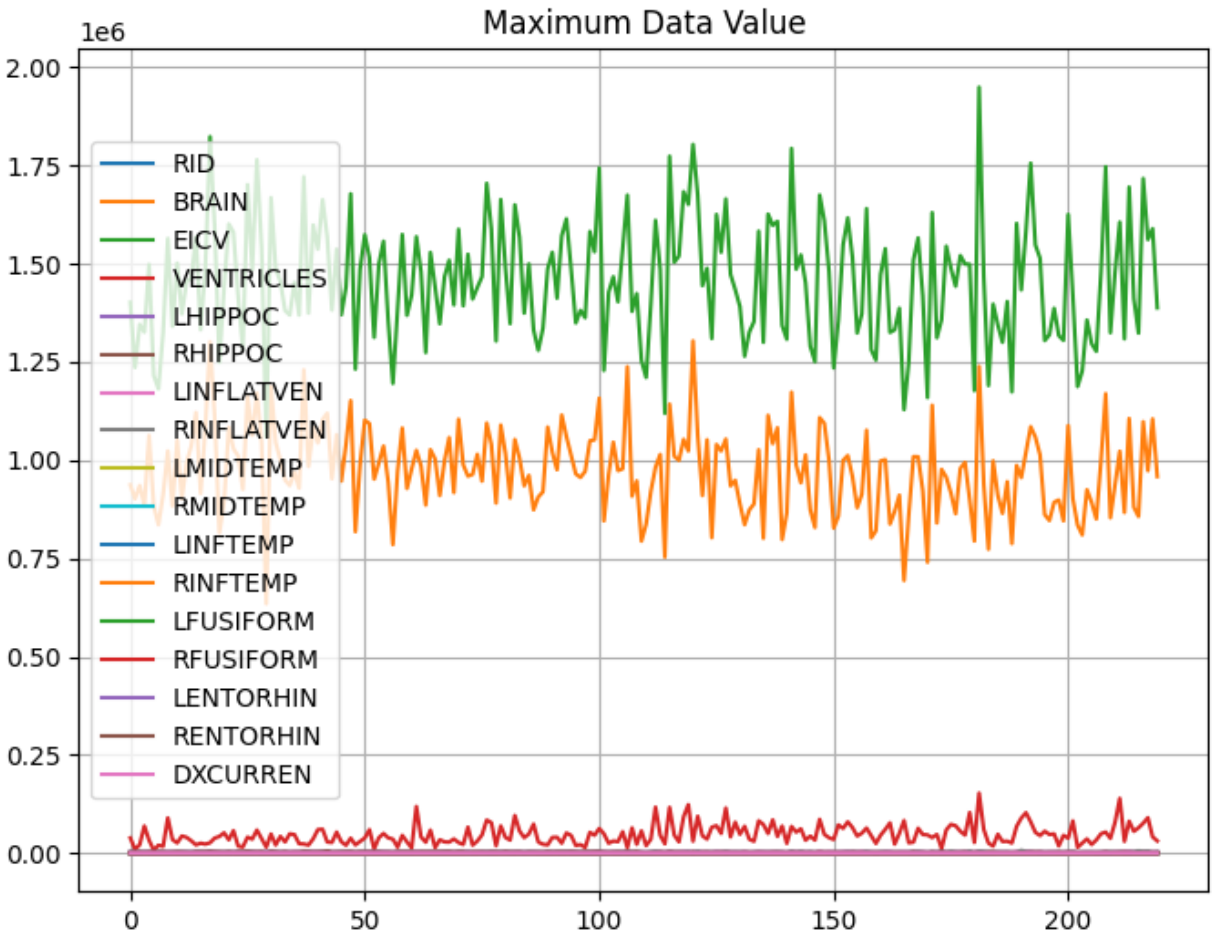
Exploratory data analysis will allow us to gain insights into the distribution of features, detect correlations, and uncover potential patterns and trends.

4.1 Visualization

In [ ]:
```
plt.hist(Demented['RHIPPOC'], bins=20)
plt.title('Right Hippocampus Distribution of Demented Patients')
plt.xlabel('Right Hippocampus')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
data.plot(kind = 'line', figsize = (8,6))
plt.title('Maximum Data Value')
plt.grid()
```



4.2 Correlation

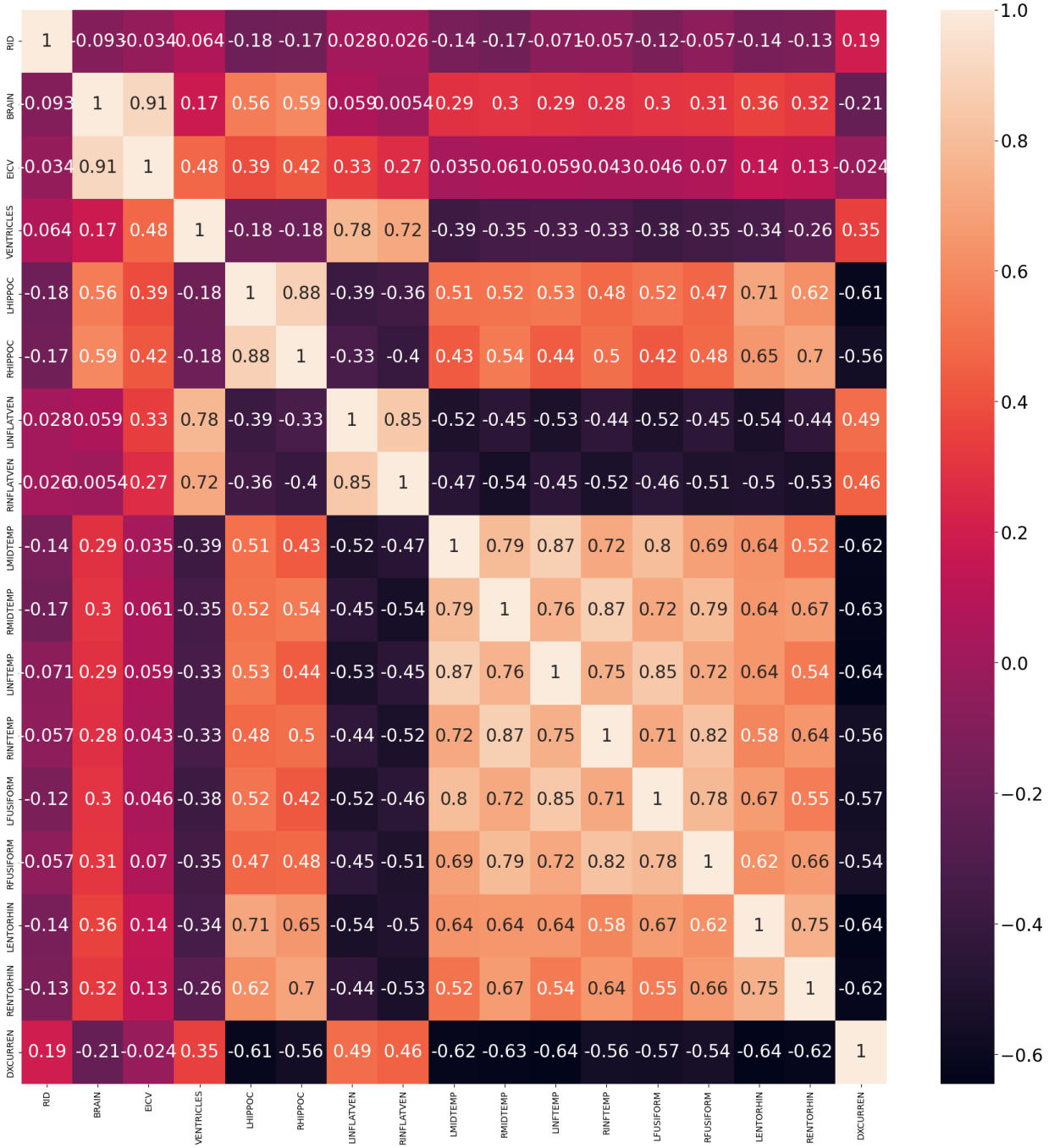
In [ ]:
```
data.corr()
```

Out[ ]:

	RID	BRAIN	EICV	VENTRICLES	LHIPPOC	RHIPPOC	LINFLATVEN	RINFLA
RID	1.000000	-0.093466	-0.033590	0.064066	-0.175928	-0.165794	0.027802	0.0
BRAIN	-0.093466	1.000000	0.914554	0.173406	0.555175	0.593095	0.058577	0.0
EICV	-0.033590	0.914554	1.000000	0.479153	0.388307	0.421946	0.332470	0.2
VENTRICLES	0.064066	0.173406	0.479153	1.000000	-0.182743	-0.178253	0.777208	0.7
LHIPPOC	-0.175928	0.555175	0.388307	-0.182743	1.000000	0.882356	-0.390813	-0.3
RHIPPOC	-0.165794	0.593095	0.421946	-0.178253	0.882356	1.000000	-0.326846	-0.4
LINFLATVEN	0.027802	0.058577	0.332470	0.777208	-0.390813	-0.326846	1.000000	0.8
RINFLATVEN	0.026366	0.005357	0.273800	0.717445	-0.358850	-0.402222	0.851397	1.0
LMIDTEMP	-0.140090	0.289149	0.034902	-0.387530	0.506860	0.430293	-0.519734	-0.4
RMIDTEMP	-0.170521	0.302821	0.061162	-0.347546	0.515082	0.543717	-0.453266	-0.5
LINFTEMP	-0.071395	0.291468	0.058630	-0.330725	0.532376	0.436532	-0.529090	-0.4
RINFTEMP	-0.057403	0.283498	0.043293	-0.334839	0.480592	0.499915	-0.444729	-0.5
LFUSIFORM	-0.124554	0.299817	0.046025	-0.380378	0.519097	0.423725	-0.519430	-0.4
RFUSIFORM	-0.056647	0.312397	0.069940	-0.353161	0.471420	0.482206	-0.448445	-0.5
LENTORHIN	-0.142876	0.356856	0.139978	-0.342013	0.709872	0.649987	-0.535279	-0.5
RENTORHIN	-0.128306	0.322380	0.128261	-0.261059	0.619150	0.698551	-0.435775	-0.5
DXCURREN	0.192822	-0.213242	-0.024218	0.345576	-0.614505	-0.564262	0.492914	0.4

In [ ]:

```
plt.figure(figsize=(22,22))
ax = sns.heatmap(data.corr(), annot=True, annot_kws={'size': 20})
col_ax = plt.gcf().axes[-1]
col_ax.tick_params(labelsize=20)
plt.show()
```



In [ ]:

```
# obtain the most correlated features from the dataset

cols = [col for col in data.columns[1:-1]] # exclude the individual identifier and dia
corr_feat = data[cols].corr() # get the correlation coefficients of all the features
corr_feat = corr_feat.unstack().sort_values(ascending=False) # unstack the dataframe a
corr_feat = corr_feat.drop_duplicates() # drop duplicates to exclude the same feature
corr_feat.head(10)
```

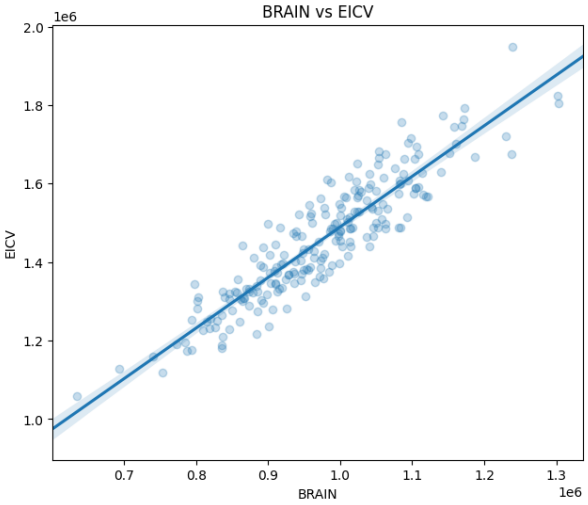
Out[ ]:
```
BRAIN       BRAIN          1.000000
            EICV           0.914554
LHIPPOC     RHIPPOC        0.882356
RMIDTEMP    RINFTEMP       0.874190
LINFTEMP    LMIDTEMP       0.871690
LINFLATVEN  RINFLATVEN     0.851397
LINFTEMP    LFUSIFORM      0.849473
RFUSIFORM   RINFTEMP       0.821934
LMIDTEMP    LFUSIFORM      0.803698
RMIDTEMP    RFUSIFORM      0.791022
dtype: float64
```

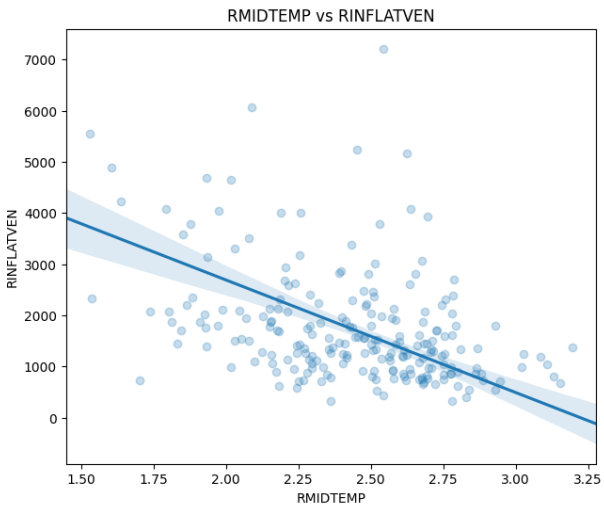
In [ ]:
```
corr_feat.tail(10)
```

Out[ ]:
```
LMIDTEMP    RINFLATVEN   -0.465602
LENTORHIN   RINFLATVEN   -0.500521
RFUSIFORM   RINFLATVEN   -0.511817
LINFLATVEN  LFUSIFORM    -0.519430
            LMIDTEMP     -0.519734
RINFLATVEN  RINFTEMP     -0.524645
LINFTEMP    LINFLATVEN   -0.529090
RINFLATVEN  RENTORHIN    -0.532744
LENTORHIN   LINFLATVEN   -0.535279
RMIDTEMP    RINFLATVEN   -0.542264
dtype: float64
```

In [ ]:
```
# set figure for the plots
plt.figure(figsize=[16, 6])

plt.subplot(1,2,1) # first plot... BRAIN vs EICV
sns.regplot(data = data, x = 'BRAIN', y = 'EICV',
            truncate=False, x_jitter=0.3, scatter_kws={'alpha':1/4})
plt.title('BRAIN vs EICV')

plt.subplot(1,2,2) # second plot... RMIDTEMP vs RINFLATVEN
sns.regplot(data = data, x = 'RMIDTEMP', y = 'RINFLATVEN',
            truncate=False, x_jitter=0.3, scatter_kws={'alpha':1/4})
plt.title('RMIDTEMP vs RINFLATVEN')
```

Out[ ]:
```
Text(0.5, 1.0, 'RMIDTEMP vs RINFLATVEN')
```





In [ ]:
```
# check the number of subjects in the two groups
```

Out[ ]:
```
0    110
1    110
Name: DXCURREN, dtype: int64
```

In [ ]:
```
# obtain the mean of each feature in both group
demented_mean = Demented[cols].describe().loc['mean']
demented_mean
```

Out[ ]:
```
BRAIN         9.501246e+05
EICV          1.451390e+06
VENTRICLES    5.552915e+04
LHIPPOC       2.762603e+03
RHIPPOC       2.952135e+03
LINFLATVEN    2.249683e+03
RINFLATVEN    2.253310e+03
LMIDTEMP      2.218401e+00
RMIDTEMP      2.258468e+00
LINFTEMP      2.285647e+00
RINFTEMP      2.290316e+00
LFUSIFORM     2.099536e+00
RFUSIFORM     2.104474e+00
LENTORHIN     2.499836e+00
RENTORHIN     2.589077e+00
Name: mean, dtype: float64
```

In [ ]:
```
normal_mean = Normal[cols].describe().loc['mean']
normal_mean
```

Out[ ]:
```
BRAIN         9.963349e+05
EICV          1.458788e+06
VENTRICLES    3.811028e+04
LHIPPOC       3.533096e+03
RHIPPOC       3.678581e+03
LINFLATVEN    1.234751e+03
RINFLATVEN    1.244294e+03
LMIDTEMP      2.564385e+00
RMIDTEMP      2.598034e+00
LINFTEMP      2.620976e+00
RINFTEMP      2.588706e+00
LFUSIFORM     2.389659e+00
RFUSIFORM     2.365876e+00
LENTORHIN     3.186655e+00
RENTORHIN     3.298458e+00
Name: mean, dtype: float64
```

In [ ]:
```
# perform two sample T-test to determine features that are significantly different bet
a = data[data["DXCURREN"]==0][cols]
b = data[data["DXCURREN"]==1][cols]

pvalue = pd.DataFrame(stats.ttest_ind(a=a, b=b)).T[1]
pvalue
```

Out[ ]:
```
0     1.464905e-03
1     7.209300e-01
2     1.444776e-07
3     3.110688e-24
4     6.790456e-20
5     7.197735e-15
6     7.144266e-13
7     5.668557e-25
8     2.069136e-25
9     1.167292e-26
10    2.104613e-19
11    3.829996e-20
12    9.731913e-18
13    3.026538e-27
14    9.028144e-25
Name: 1, dtype: float64
```

In [ ]:
```
# create a dataframe of the mean differences of the features for the two groups and th
mean_diff = pd.DataFrame()
mean_diff["features"] = cols
mean_diff["demented"] = list(demented_mean)
mean_diff["normal"] = list(normal_mean)
mean_diff["mean difference"] = mean_diff["demented"] - mean_diff["normal"]
mean_diff["p-values"] = list(pvalue)
mean_diff = mean_diff.sort_values("p-values", ascending=True) #sort the p-values from
mean_diff
```

Out[ ]:

	features	demented	normal	mean difference	p-values
13	LENTORHIN	2.499836e+00	3.186655e+00	-0.686819	3.026538e-27
9	LINFTEMP	2.285647e+00	2.620976e+00	-0.335329	1.167292e-26
8	RMIDTEMP	2.258468e+00	2.598034e+00	-0.339566	2.069136e-25
7	LMIDTEMP	2.218401e+00	2.564385e+00	-0.345984	5.668557e-25
14	RENTORHIN	2.589077e+00	3.298458e+00	-0.709381	9.028144e-25
3	LHIPPOC	2.762603e+03	3.533096e+03	-770.493091	3.110688e-24
11	LFUSIFORM	2.099536e+00	2.389659e+00	-0.290123	3.829996e-20
4	RHIPPOC	2.952135e+03	3.678581e+03	-726.445727	6.790456e-20
10	RINFTEMP	2.290316e+00	2.588706e+00	-0.298390	2.104613e-19
12	RFUSIFORM	2.104474e+00	2.365876e+00	-0.261402	9.731913e-18
5	LINFLATVEN	2.249683e+03	1.234751e+03	1014.932855	7.197735e-15
6	RINFLATVEN	2.253310e+03	1.244294e+03	1009.015918	7.144266e-13
2	VENTRICLES	5.552915e+04	3.811028e+04	17418.864909	1.444776e-07
0	BRAIN	9.501246e+05	9.963349e+05	-46210.290909	1.464905e-03
1	EICV	1.451390e+06	1.458788e+06	-7398.090909	7.209300e-01

In [ ]:
```
# obtain the significant features... these are features with p-values less than 0.05
sig_features = mean_diff[mean_diff["p-values"] <= 0.05]["features"]
```

Out[ ]:
```
13     LENTORHIN
9       LINFTEMP
8       RMIDTEMP
7       LMIDTEMP
14     RENTORHIN
3        LHIPPOC
11     LFUSIFORM
4        RHIPPOC
10      RINFTEMP
12     RFUSIFORM
5     LINFLATVEN
6     RINFLATVEN
2     VENTRICLES
0          BRAIN
Name: features, dtype: object
```

5. Model Building

In thise section we build predictive models using machine learning algorithms. First we have to split the dataset for training and testing.

In [ ]:
```
#dropping irrelevant feature
data.drop(['RID'], axis=1, inplace = True)
```

In [ ]:
```
input_cols = [col for col in data.columns[:-1]] # select the input columns

x = data[input_cols] # input data
y = data['DXCURREN'] # target

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=
print(X_train.shape,y_train.shape)
```

```
(176, 15) (176,)
```

In [ ]:
```
sc = StandardScaler() # create an instance of the scaler
train = sc.fit_transform(X_train) # fit the scaler on the training set
train_input = pd.DataFrame(train, columns=input_cols) # convert the scaled data array


val = sc.transform(X_test) # transform the scaler on the validation set
val_input = pd.DataFrame(val, columns=input_cols)

train_input.head()
```

Out[ ]:

	BRAIN	EICV	VENTRICLES	LHIPPOC	RHIPPOC	LINFLATVEN	RINFLATVEN	LMIDTEMP	RM
0	-1.197470	-0.898390	0.047310	-1.483716	-1.209035	0.810287	0.614007	-0.321485	
1	-0.003505	-0.653721	-1.390894	2.648516	2.374907	-1.066128	-0.885289	0.698345	
2	-0.612334	-0.740438	0.070481	-0.459276	-1.186119	-0.218392	0.150823	-0.164108	
3	0.209429	0.115838	0.123277	-1.126826	-1.226319	0.092440	0.249148	-0.087501	
4	0.951778	0.174839	-0.933264	1.113446	0.691902	-1.163315	-0.828359	1.578209	

5.1 Naive Bayes Classifier

In [ ]:
```
nb_clf = GaussianNB() # get instance of model
nb_clf.fit(train_input, y_train) # Train/Fit model

# use the train model to predict the validation set
y_pred_nb = nb_clf.predict(val_input)

accuracy = accuracy_score(y_test, y_pred_nb)
print(f"Classifier Accuracy: {accuracy*100:.1f}%")
```

```
Classifier Accuracy: 90.9%
```

5.2 K-Nearest Neighbours

In [ ]:
```
knn_clf = KNeighborsClassifier() # get instance of model
knn_clf.fit(train_input, y_train) # train/fit the model
y_pred_knn = knn_clf.predict(val_input) # get the target predictions
accuracy = accuracy_score(y_test, y_pred_knn) #obtain the accuracy score of the model

print(f"KNN Classifier Accuracy: {accuracy*100:.1f}%")
```

```
KNN Classifier Accuracy: 90.9%
```

In [ ]:
```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]

#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)#Create new k

knn_2 = KNeighborsClassifier()

#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)

#Fit the model
best_model = clf.fit(train_input, y_train)
y_pred = best_model.predict(val_input)
accuracy = accuracy_score(y_test, y_pred)

#Print The value of best Hyperparameters
print('Accuracy: ', accuracy)
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

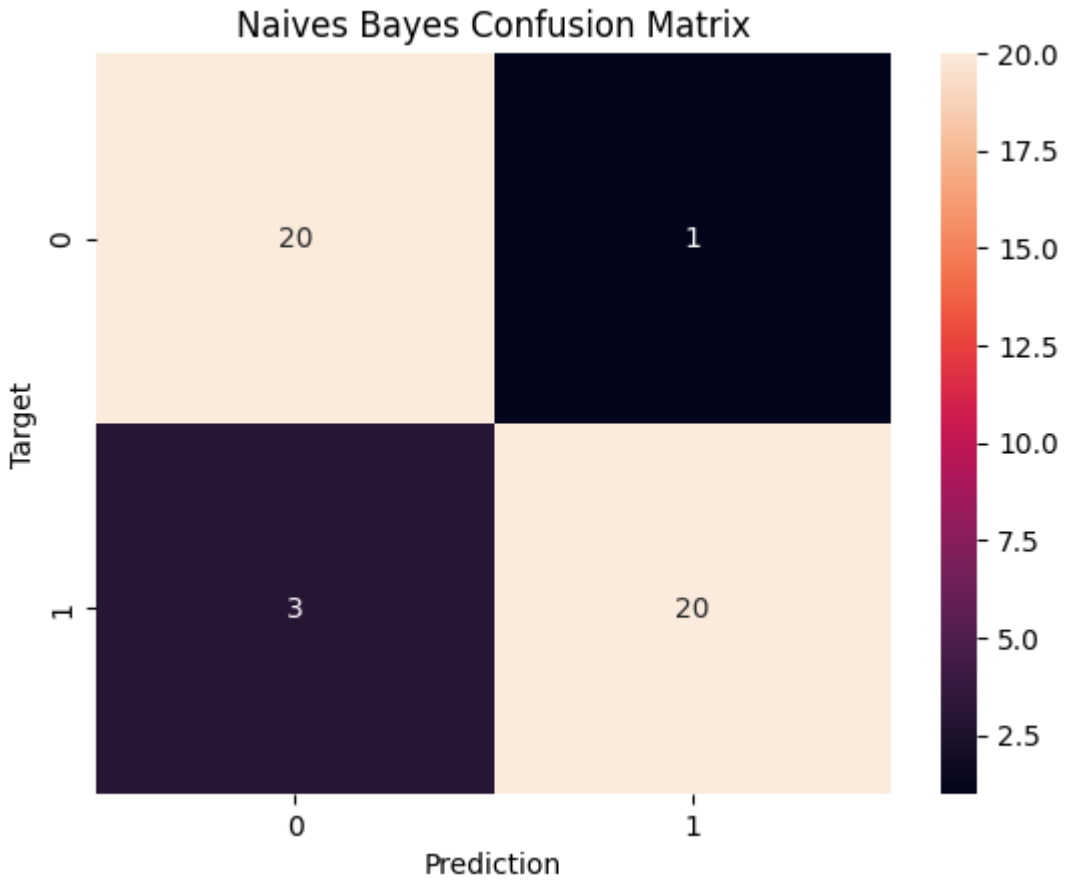
```
Accuracy:  0.9318181818181818
Best leaf_size: 1
Best p: 2
Best n_neighbors: 7
```

6. Evaluation

e of the models and compare them.

In [ ]:
```
#Confusion Matrix for Naives Bayes Classifier
nb_cf = confusion_matrix(y_test, y_pred_nb)

# plot an heatmap of the confusion matrix
plt.figure()
sns.heatmap(nb_cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Naives Bayes Confusion Matrix')
```

Out[ ]:
```
Text(0.5, 1.0, 'Naives Bayes Confusion Matrix')
```



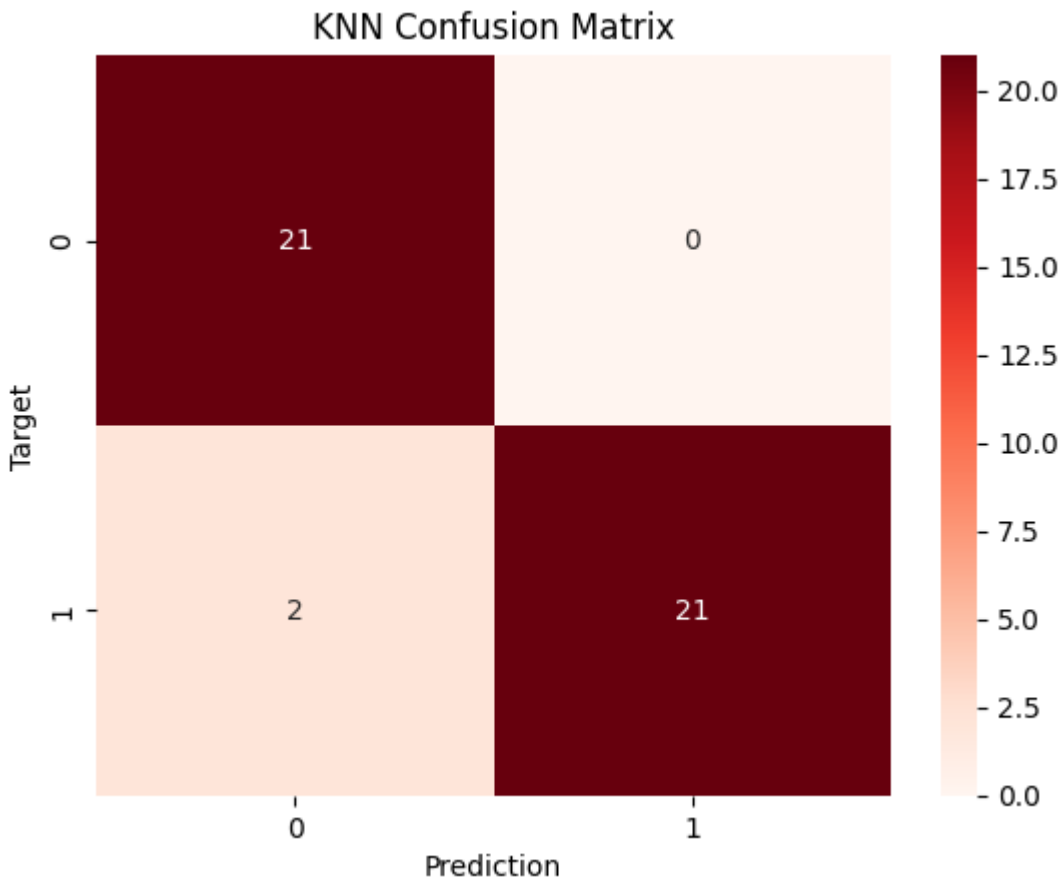
In [ ]:
```
# obtain true and false positive and negative data point
TP = nb_cf[1,1] # true positive
TN = nb_cf[0,0] # true negative
FP = nb_cf[1,0] # False positive
FN = nb_cf[0,1] # false negative

# determine the evaluation metrics

nb_accuracy = (TP+TN)/(TP+TN+FN+FP)
nb_sensitivity = TP / (TP + FN)
nb_specificity = TN / (TN + FP)
nb_precision = TP / (TP + FP)
nb_recall = TP / (TP + FN)


#calculate AUC of model
nb_auc = roc_auc_score(y_test, y_pred_nb)
```

```
# create a dictionary for the evaluation metrics
nb_perf = [{'Accuracy': nb_accuracy, 'Sensitivity': nb_sensitivity, 'Specificity': nb_
                  'Precision': nb_precision, 'AUC': nb_auc}]
nb_perf_df = pd.DataFrame.from_dict(nb_perf) # convert the dictionary to dataframe
nb_perf_df
```

Out[ ]:

	Accuracy	Sensitivity	Specificity	Precision	AUC
0	0.909091	0.952381	0.869565	0.869565	0.910973

In [ ]:
```
knn_clf2 = KNeighborsClassifier(leaf_size=1, p=2, n_neighbors=13)
knn_clf2.fit(train_input, y_train)
y_pred = knn_clf2.predict(val_input)

# generate the confusion matrix for the KNN classifier
knn_cf = confusion_matrix(y_test, y_pred)

# plot an heatmap of the confusion matrix
plt.figure()
sns.heatmap(knn_cf, cmap='Reds', annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('KNN Confusion Matrix');
```



In [ ]:
```
TP = knn_cf[1,1] # true positive
TN = knn_cf[0,0] # true negative
FP = knn_cf[1,0] # False positive
FN = knn_cf[0,1] # false negative
```



```
# determine the evaluation metrics
```

```
knn_accuracy = (TP+TN)/(TP+TN+FN+FP)
knn_sensitivity = TP / (TP + FN)
knn_specificity = TN / (TN + FP)
knn_precision = TP / (TP + FP)

#calculate AUC of model
knn_auc = roc_auc_score(y_test, y_pred)

# create a dictionary for the evaluation metrics
knn_perf = [{'Accuracy': knn_accuracy, 'Sensitivity': knn_sensitivity, 'Specificity':
                  'Precision': knn_precision, 'AUC': knn_auc}]
knn_perf_df = pd.DataFrame.from_dict(knn_perf) # convert the dictionary to dataframe
knn_perf_df
```

Out[ ]:

	Accuracy	Sensitivity	Specificity	Precision	AUC
0	0.954545	1.0	0.913043	0.913043	0.956522

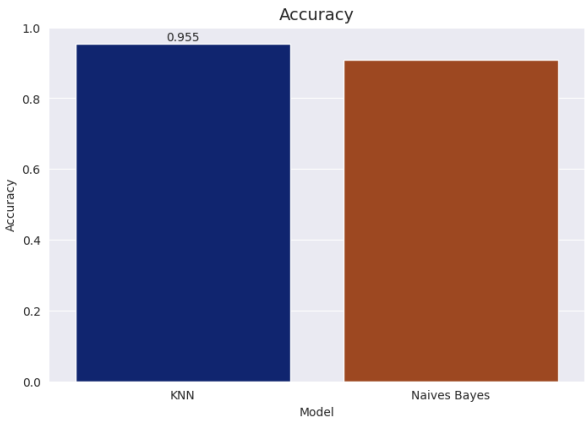
In [ ]:
```
metrics = {'Model': ['KNN', 'Naives Bayes'], 'Accuracy': [knn_accuracy, nb_accuracy],
           'Sensitivity': [knn_sensitivity, nb_sensitivity], 'Specificity': [knn_spec
           'Precision': [knn_precision, nb_precision], 'AUC': [knn_auc, nb_auc]}

# convert the metrics dictionary to dataframe
metrics_df = pd.DataFrame.from_dict(metrics)
metrics_df
```

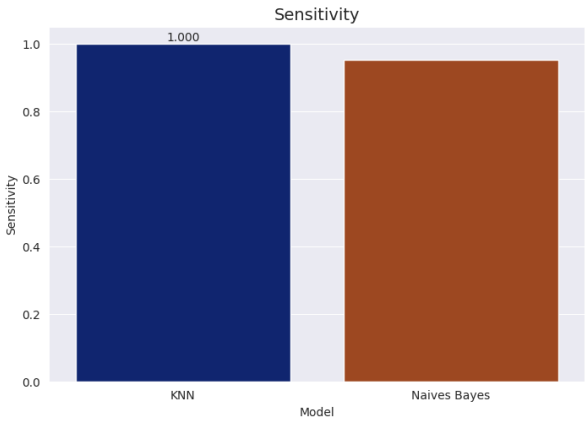
Out[ ]:

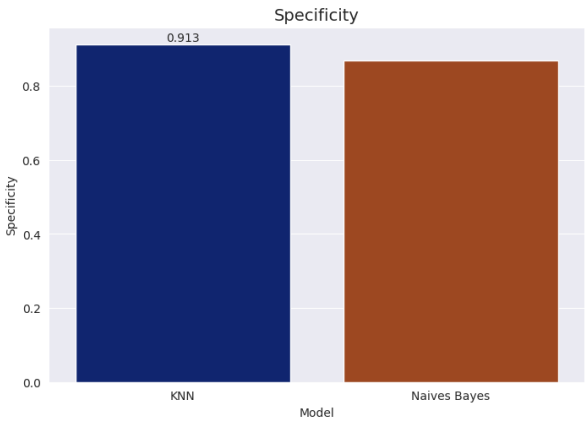
	Model	Accuracy	Sensitivity	Specificity	Precision	AUC
0	KNN	0.954545	1.000000	0.913043	0.913043	0.956522
1	Naives Bayes	0.909091	0.952381	0.869565	0.869565	0.910973

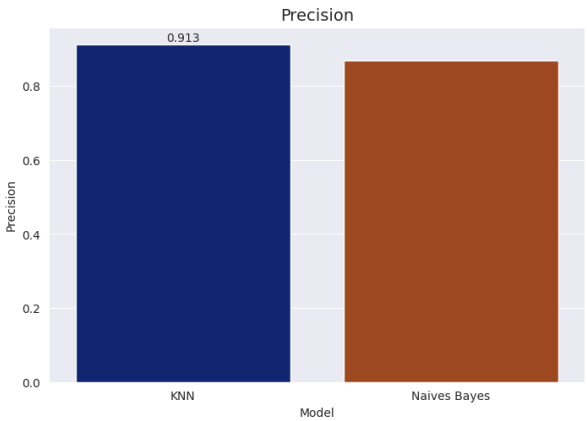
In [ ]:
```
# set figure for the plots
plt.figure(figsize=[18, 12])
sns.set_style('darkgrid')
color = sns.color_palette()[1] # plot the bars with the same colour for all categories

plt.subplot(2,2,1)
ax = sns.barplot(data=metrics_df, x='Model', y='Accuracy',palette='dark')
ax.bar_label(ax.containers[0], fmt='%.3f')
plt.title("Accuracy", fontsize=14)

plt.subplot(2,2,2)
ax = sns.barplot(data=metrics_df, x='Model', y='Sensitivity',palette='dark')
ax.bar_label(ax.containers[0], fmt='%.3f')
plt.title("Sensitivity", fontsize=14)

plt.subplot(2,2,3)
ax = sns.barplot(data=metrics_df, x='Model', y='Specificity',palette='dark')
ax.bar_label(ax.containers[0], fmt='%.3f')
plt.title("Specificity", fontsize=14)

plt.subplot(2,2,4)
ax = sns.barplot(data=metrics_df, x='Model', y='Precision',palette='dark')
ax.bar_label(ax.containers[0], fmt='%.3f')
plt.title("Precision", fontsize=14);
```

```
<ipython-input-57-61926c07bca8>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.barplot(data=metrics_df, x='Model', y='Accuracy',palette='dark')
<ipython-input-57-61926c07bca8>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.barplot(data=metrics_df, x='Model', y='Sensitivity',palette='dark')
<ipython-input-57-61926c07bca8>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.barplot(data=metrics_df, x='Model', y='Specificity',palette='dark')
<ipython-input-57-61926c07bca8>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.barplot(data=metrics_df, x='Model', y='Precision',palette='dark')
```









7. Conclusion

Based on the evaluation metrics, it was observed that the Naive Bayes Classifier outperformed the K-Nearest Neighbor (KNN) model in classifying patients with Alzheimer's disease. This

classification was achieved by utilizing both the extracted MRI images and additional data from the ADNI dataset.

To optimize the performance of the KNN classifier, a grid search technique was employed to identify the optimal combination of parameters that yielded the highest accuracy for the classifier.