Johnston630FinalProject

September 20, 2024

Cancer Prediction Analysis

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import neighbors
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
```

[3]:
```
df = pd.read_csv("The_Cancer_data_1500_V2.csv")

df.head()
```

[3]:

	Age	Gender	BMI	Smoking	GeneticRisk	PhysicalActivity	AlcoholIntake	CancerHistory	Diagnosis
0	58	1	16.085313	0	1	8.146251	4.148219	1	1
1	71	0	30.828784	0	1	9.361630	3.519683	0	0
2	48	1	38.785084	0	2	5.135179	4.728368	0	1
3	34	0	30.040296	0	0	9.502792	2.044636	0	0
4	62	1	35.479721	0	0	5.356890			

```
4       3.309849               0           1
```

Data Prep/Cleaning **********

[5]:
```
df.isna().sum()
```

[5]:
```
Age                 0
Gender              0
BMI                 0
Smoking             0
GeneticRisk         0
PhysicalActivity    0
AlcoholIntake       0
CancerHistory       0
Diagnosis           0
dtype: int64
```

[6]:
```
df = df[["Age", "Gender", "Smoking", "GeneticRisk", "CancerHistory",
         "Diagnosis", "PhysicalActivity", "AlcoholIntake", "BMI"]]

df
```

[6]:
```
      Age  Gender  Smoking  GeneticRisk  CancerHistory  Diagnosis  \
0      58       1        0            1              1          1
1      71       0        0            1              0          0
2      48       1        0            2              0          1
3      34       0        0            0              0          0
4      62       1        0            0              0          1
…     …      …       …          …             …          …
1495   62       1        0            0              0          1
1496   31       0        0            1              1          1
1497   63       1        1            1              0          1
1498   55       0        0            0              1          1
1499   67       1        0            0              1          0

      PhysicalActivity  AlcoholIntake        BMI
0             8.146251       4.148219  16.085313
1             9.361630       3.519683  30.828784
2             5.135179       4.728368  38.785084
3             9.502792       2.044636  30.040296
4             5.356890       3.309849  35.479721
…                …            …         …
1495          9.892167       1.284158  25.090025
1496          1.668297       2.280636  33.447125
1497          0.466848       0.150101  32.613861
1498          7.795317       1.986138  25.568216
```

```
1499          2.525860       2.856600  23.663104

[1500 rows x 9 columns]
```

[7]:
```
df.shape
```

[7]:
```
(1500, 9)
```

[8]:
```
df["Diagnosis"].value_counts()
```

[8]:
```
Diagnosis
0    943
1    557
Name: count, dtype: int64
```

[9]:
```
df.dtypes
```

[9]:
```
Age                  int64
Gender               int64
Smoking              int64
GeneticRisk          int64
CancerHistory        int64
Diagnosis            int64
PhysicalActivity   float64
AlcoholIntake      float64
BMI                float64
dtype: object
```

[10]:
```
df.describe()
```

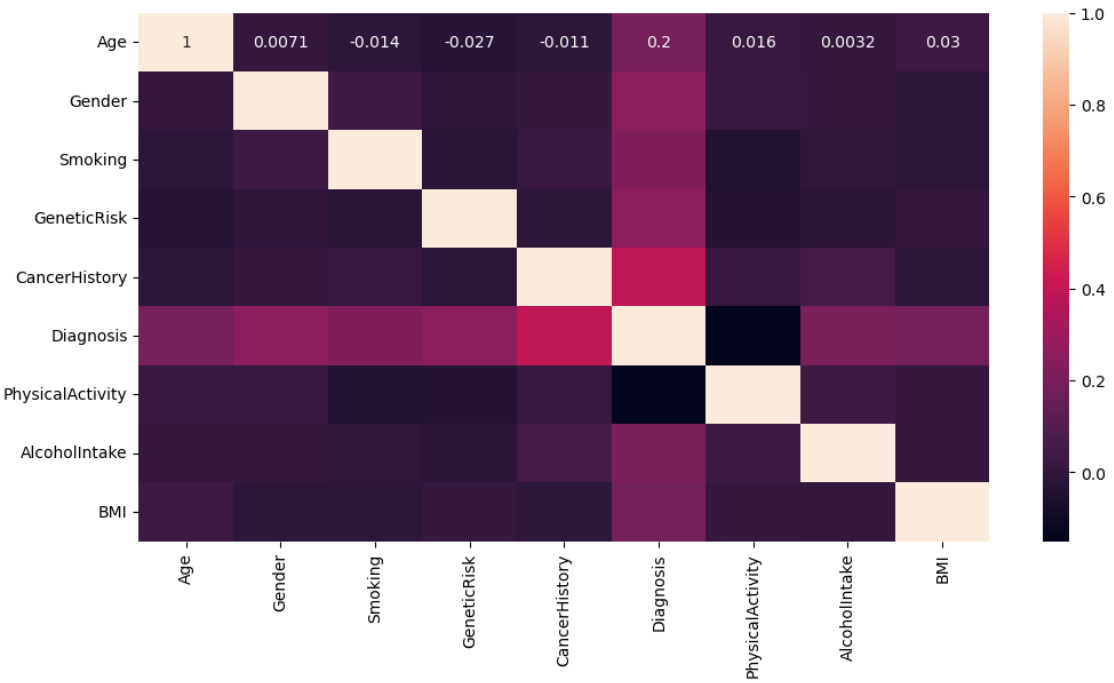
[10]:
```
                Age       Gender      Smoking  GeneticRisk  CancerHistory  \
count  1500.000000  1500.000000  1500.000000  1500.000000    1500.000000
mean     50.320000     0.490667     0.269333     0.508667       0.144000
std      17.640968     0.500080     0.443761     0.678895       0.351207
min      20.000000     0.000000     0.000000     0.000000       0.000000
25%      35.000000     0.000000     0.000000     0.000000       0.000000
50%      51.000000     0.000000     0.000000     0.000000       0.000000
75%      66.000000     1.000000     1.000000     1.000000       0.000000
max      80.000000     1.000000     1.000000     2.000000       1.000000

          Diagnosis  PhysicalActivity  AlcoholIntake          BMI
count  1500.000000       1500.000000    1500.000000  1500.000000
mean      0.371333          4.897929       2.417987    27.513321
std       0.483322          2.866162       1.419318     7.230012
min       0.000000          0.002410       0.001215    15.000291
25%       0.000000          2.434609       1.210598    21.483134
50%       0.000000          4.834316       2.382971    27.598494
```

```
75%       1.000000          7.409896       3.585624     33.850837
max       1.000000          9.994607       4.987115     39.958688
```

[11]:
```
plt.figure(figsize = (12,6))
sns.heatmap(df.corr(), annot = True)
```

[11]:
```
<Axes: >
```



The above correlation heatmap shows that the target variable (Diagnosis) has almost if not a complete lack of correlation at all. Due to this result the PhysicalActivity column can safely be classified as redundant and will be dropped from the dataset.

[13]:
```
df = df.drop(columns=["PhysicalActivity"])

df
```

[13]:

	Age	Gender	Smoking	GeneticRisk	CancerHistory	Diagnosis	\
0	58	1	0	1	1	1	
1	71	0	0	1	0	0	
2	48	1	0	2	0	1	
3	34	0	0	0	0	0	
4	62	1	0	0	0	1	
…	…	…	…	…	…	…	…
1495	62	1	0	0	0	1	
1496	31	0	0	1	1	1	

```
1497   63       1        1             1              0          1
1498   55       0        0             0              1          1
1499   67       1        0             0              1          0

      AlcoholIntake        BMI
0          4.148219  16.085313
1          3.519683  30.828784
2          4.728368  38.785084
3          2.044636  30.040296
4          3.309849  35.479721
…               …         …
1495       1.284158  25.090025
1496       2.280636  33.447125
1497       0.150101  32.613861
1498       1.986138  25.568216
1499       2.856600  23.663104

[1500 rows x 8 columns]
```

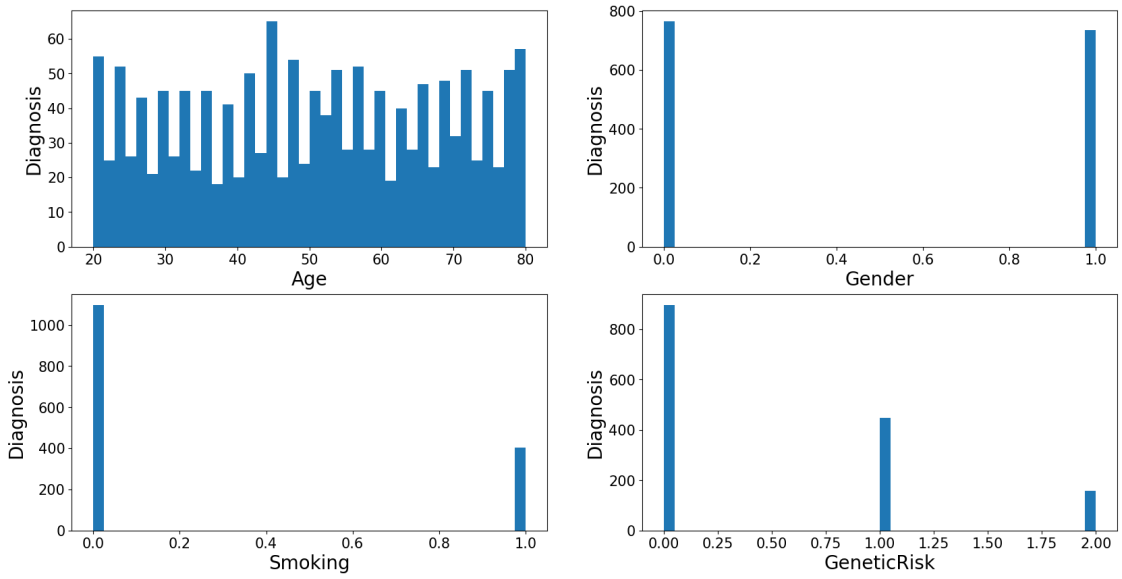
[14]:
```
df.shape
```

[14]:
```
(1500, 8)
```

[15]:
```
# Displays the relationship of some of the more important variables to␣
 ↪'Diagnosis'


important_variables = ["Age", "Gender", "Smoking", "GeneticRisk"]

xaxes = important_variables
yaxes = ['Diagnosis', 'Diagnosis', 'Diagnosis', 'Diagnosis']
plt.rcParams['figure.figsize'] = (20, 10)
fig, axes = plt.subplots(nrows = 2, ncols = 2)
axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(df[important_variables[idx]].dropna(), bins = 40)
    ax.set_xlabel(xaxes[idx], fontsize = 20)
    ax.set_ylabel(yaxes[idx], fontsize = 20)
    ax.tick_params(axis = 'both', labelsize = 15)
plt.show()
```



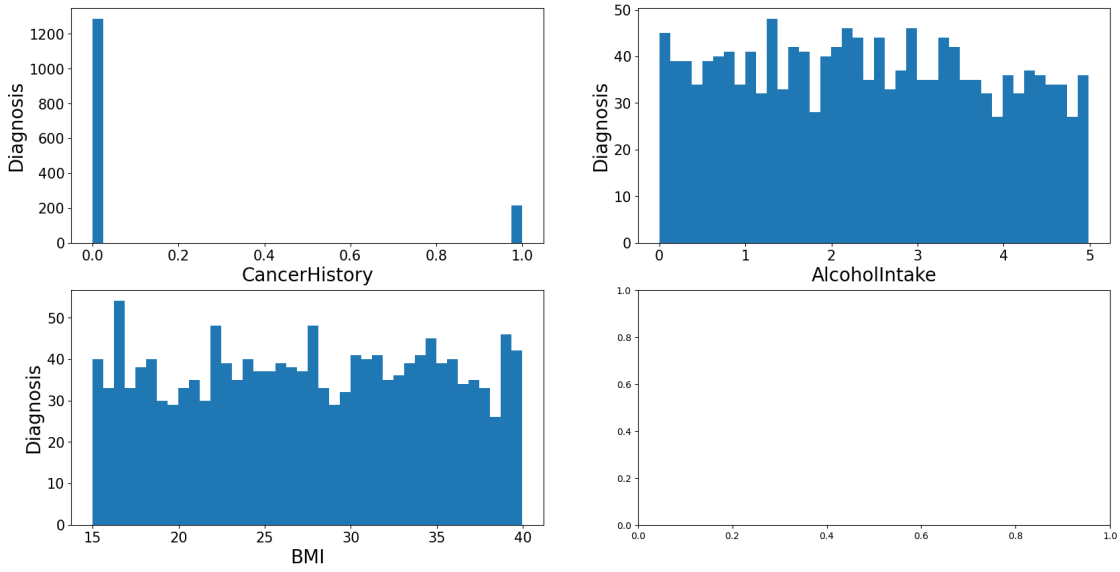
[16]:
```
# Displays the relationship of some of the more important variables to␣
↪'Diagnosis'


important_variables = ["CancerHistory", "AlcoholIntake", "BMI"]

xaxes = important_variables
yaxes = ['Diagnosis', 'Diagnosis', 'Diagnosis']
plt.rcParams['figure.figsize'] = (20, 10)
fig, axes = plt.subplots(nrows = 2, ncols = 2)
axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(df[important_variables[idx]].dropna(), bins = 40)
    ax.set_xlabel(xaxes[idx], fontsize = 20)
    ax.set_ylabel(yaxes[idx], fontsize = 20)
    ax.tick_params(axis = 'both', labelsize = 15)
plt.show()
```

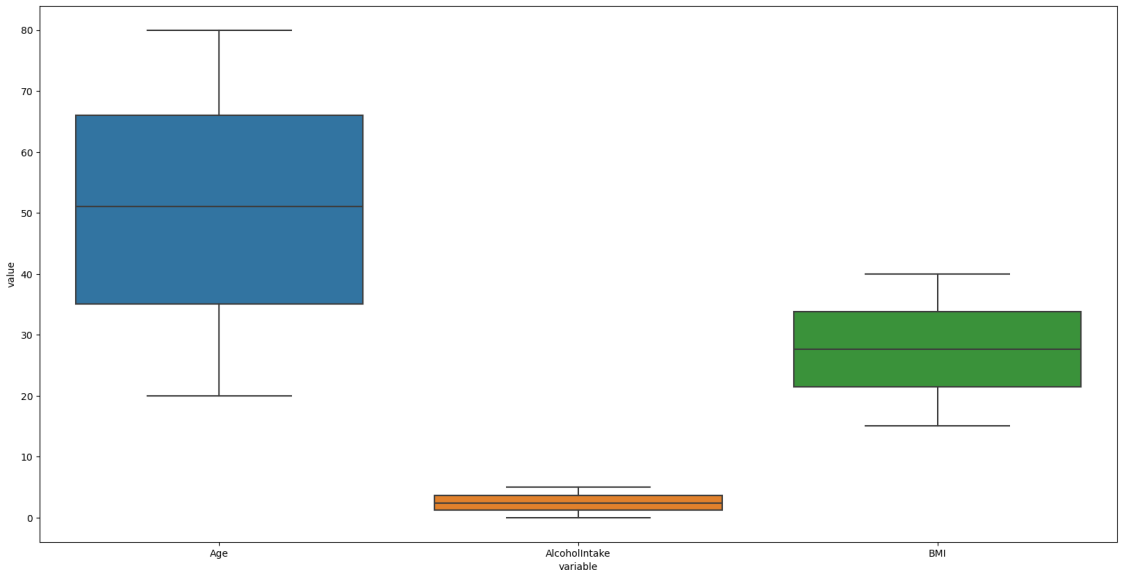
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[16], line 12
     10 axes = axes.ravel()
     11 for idx, ax in enumerate(axes):
---> 12     ax.hist(df[important_variables[idx]].dropna(), bins = 40)
     13     ax.set_xlabel(xaxes[idx], fontsize = 20)
     14     ax.set_ylabel(yaxes[idx], fontsize = 20)
```

```
IndexError: list index out of range
```



[18]:
```
# Boxplot to locate any outlier values within the dataset (Only non Boolean/O
  ↪or 1 variables)

outlier_df = pd.DataFrame(data = df, columns = ["Age", "AlcoholIntake", "BMI"])
sns.boxplot(x = "variable", y = "value", data = pd.melt(outlier_df))
plt.show()
```



The variable with the largest range of data is the Age column which was to be expected. All three variables have a balance of data ranging higher and lower than their median values.

LINEAR REGRESSION MODEL **********

[22]:
```
# Splits the df into test and train sets with the target variable being␣
↪'Diagnosis'

X = df.drop('Diagnosis', axis = 1)
y = df['Diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

[24]:
```
target_model = LinearRegression()
target_model.fit(X_train, y_train)
```

[24]:
```
LinearRegression()
```

[25]:
```
# Displays the relationship of the target variable with the rest of the columns

pd.DataFrame(target_model.coef_, X.columns, columns = ['Coef'])
```

[25]:

	Coef
Age	0.005513
Gender	0.236067
Smoking	0.250174
GeneticRisk	0.189886
CancerHistory	0.529226
AlcoholIntake	0.063934
BMI	0.013344

[28]:
```
# Displays the metrics of each set in regards to rmse, absolute error & r2 score

test_predictions = target_model.predict(X_test)
train_predictions = target_model.predict(X_train)

print('Test Metrics:')
print('R2', metrics.r2_score(y_test, test_predictions))
print('RMSE', metrics.mean_squared_error(y_test, test_predictions, squared =␣
↪False))
print('MAE', metrics.mean_absolute_error(y_test, test_predictions))

print('\nTrain Metrics:')
print('R2', metrics.r2_score(y_train, train_predictions))
print('RMSE', metrics.mean_squared_error(y_train, train_predictions, squared =␣
↪False))
```

```
print('MAE', metrics.mean_absolute_error(y_train, train_predictions))
```

```
Test Metrics:
R2 0.4125978367211943
RMSE 0.3749378495884677
MAE 0.3106657670885278

Train Metrics:
R2 0.45328745661898917
RMSE 0.35596952080498945
MAE 0.29214799871425223
```

The R2 scores indicate close to a 50/50 chance of accurate predictability for this models correlation of each variable. This score is higher than I would have liked but still very useful information. The RMSE and MAE scores of .29 - .36 are slightly better in terms of displaying confidence in the models predictive capabilities. The values that were particularly interesting in this model were the low levels of correlation between Alchohol Intake and BMI to Diagnosis.

OLS REGRESSION MODEL **********

[32]:
```
X = sm.add_constant(df[["Age", "Gender", "Smoking", "GeneticRisk",␣
  ↪"CancerHistory", "AlcoholIntake", "BMI"]])
y = df['Diagnosis']
model = sm.OLS(y, X).fit()
predictions = model.predict(X)
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Diagnosis   R-squared:                       0.446
Model:                            OLS   Adj. R-squared:                  0.443
Method:                 Least Squares   F-statistic:                     171.3
Date:                Sat, 14 Sep 2024   Prob (F-statistic):          4.98e-186
Time:                        02:23:46   Log-Likelihood:                -594.94
No. Observations:                1500   AIC:                             1206.
Df Residuals:                    1492   BIC:                             1248.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
const          -0.7651      0.049    -15.507      0.000      -0.862
-0.668
Age             0.0056      0.001     10.532      0.000       0.005
```

```
0.007
Gender            0.2317      0.019     12.429      0.000       0.195
0.268
Smoking           0.2434      0.021     11.577      0.000       0.202
0.285
GeneticRisk       0.1923      0.014     13.997      0.000       0.165
0.219
CancerHistory     0.5274      0.027     19.842      0.000       0.475
0.580
AlcoholIntake     0.0656      0.007      9.974      0.000       0.053
0.078
BMI               0.0125      0.001      9.714      0.000       0.010
0.015
==============================================================================
Omnibus:                       47.785   Durbin-Watson:                   1.845
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               48.281
Skew:                           0.410   Prob(JB):                     3.28e-11
Kurtosis:                       2.683   Cond. No.                         318.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

A high F-Statistic score and extremely low values of standard error across the dataset allow us to proceed with confidence in our current understanding of the dataset in regards to the two models presented so far.

KNN MODEL **********

[36]:
```
X = df.drop(columns = "Diagnosis").copy()
y = df["Diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 30)
```

[38]:
```
knn = neighbors.KNeighborsClassifier()

knn.fit(X_train, y_train)
```

[38]:
```
KNeighborsClassifier()
```

[40]:
```
knn.score(X_test, y_test)
```

[40]:
```
0.6746666666666666
```

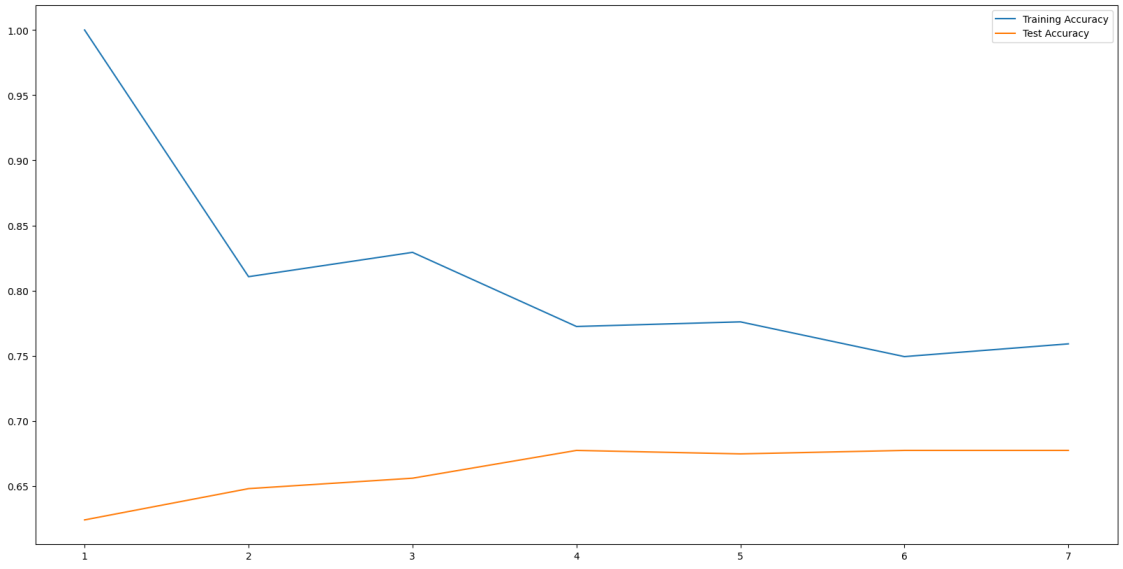
[42]:
```
no_neighbors = range(1,8)

training_accuracy = []
test_accuracy = []

for n in no_neighbors:
    knn = neighbors.KNeighborsClassifier(n_neighbors = n)
    knn.fit(X_train, y_train)
    training_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
```

[43]:
```
plt.plot(no_neighbors, training_accuracy, label = "Training Accuracy")
plt.plot(no_neighbors, test_accuracy, label = "Test Accuracy")
plt.legend()
plt.plot()
```

[43]: []



[44]:
```
training_accuracy[3]
```

[44]: 0.7724444444444445

[46]:
```
test_accuracy[3]
```

[46]: 0.6773333333333333

Our KNN MODEL shows us that this dataset is capable of predicting a group correlation or trend/pattern against the target variable at a rate of 77% according to our training accuracy score. This is a slight improvement from our previous models when evaluating the dataset at its original

state. However, I will still look to create more models in the hope of producing a more accurate model that can predict cancer trends/future diagnoses at a higher rate.

ADDITIONAL MODELS- LOGISTIC REGRESSION, DECISION TREE & RANDOM FOREST
****

[52]:
```
X = df.drop(columns = "Diagnosis").copy()
y = df["Diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
↪random_state = 0)
```

[54]:
```
X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().fit_transform(X_test)
```

[56]:
```
def models(X_Train, y_train):
    # Logistic Regression Model
    log = LogisticRegression(random_state = 0)
    log.fit(X_Train, y_train)


    # Decision Tree Model
    tree = DecisionTreeClassifier(random_state = 0, criterion = 'entropy')
    tree.fit(X_train, y_train)

    # Random Forest Model
    rforest = RandomForestClassifier(random_state = 0, criterion = 'entropy',␣
↪n_estimators = 10)
    rforest.fit(X_train, y_train)

    print("[0]: Logistic Regression Model Accuracy: ", log.score(X_train,␣
↪y_train))
    print("[1]: Decision Tree Model Accuracy: ", tree.score(X_train, y_train))
    print("[2]: Random Forest Model Accuracy: ", rforest.score(X_train,␣
↪y_train))


    return log, tree, rforest
```

[58]:
```
model_results = models(X_train, y_train)
```

```
[0]: Logistic Regression Model Accuracy:  0.8275
[1]: Decision Tree Model Accuracy:  1.0
[2]: Random Forest Model Accuracy:  0.9866666666666667
```

[ ]:
```
for i in range(len(model_results)):
    print("Model", i)
    print(classification_report(y_test, model_results[i].predict(X_test)))
    print("Accuracy Score: ", accuracy_score(y_test, model_results[i].
 ↪predict(X_test)))
```

As you can see above I was able to create a model with an improved level of accuracy compared to our previous results. The Random Forest Classifier Model produces a report high 88.3% level of accuracy.

RESULT VISUALIZATION (ONLY RANDOM FOREST MODEL) **********

[ ]:
```
X = df.drop(columns = "Diagnosis").copy()
y = df["Diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 0)
```

[ ]:
```
clf = RandomForestClassifier(n_estimators = 20, random_state=0)
```

[ ]:
```
clf.fit(X_train, y_train)
```

[ ]:
```
y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

[ ]:
```
len(clf.estimators_)
```

[ ]:
```
plt.figure(figsize=(20,20))
tree.plot_tree(clf.estimators_[2], filled = True)
plt.show()
```

RESULTS & INTERPRETATION/CONCLUSION & FURTHER RECOMMENDATIONS **********

During this project analysis I was able to locate the model type that best worked within this dataset for producing the most accurate prediction results possible towards the target variable. The highest prediction score was achieved through a Random Forest Classification model that held a project high 88% accuracy score. The tree displayed above offers a breakdown into which classifiers are most likely to lead to a positive or negative Diagnosis value. Whilst I believe that this analysis does contain a decent amount of value that could be applied to a medical scenario as was the original thought process when starting this project, I believe that it would be best to further build upon this work before it was brought forward for professional utilization. The first thing that I would recommend is to narrow down the variables utilized within the model. I decided to not utilize this method as I believed it could possibly skew the results. However, upon completion I believe that this would allow for the final visual model to be easier to evaluate. Also, I would continue to try

and test out other models that could possibly result in a 90%+ accuracy score for the model as the current one does have a 12% margin for error.

[ ]: