Data Analysis

In [3]:

```
# Importing the necessary modules
import numpy as np
import pandas as pd
import seaborn as sns
from collections import Counter
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.utils import resample
from sklearn.decomposition import PCA
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
```

In [4]:

```
# Loading the dataset into memory using the pandas read_csv method
# Specifying the path to the dataset


# Reading the dataset into memory
df = pd.read_csv(dataset, delimiter=",")
# Dropping the unnecessary columns
df = df.drop("Unnamed: 32", axis = 1)

# Setting some basic parameters for the analysis
%matplotlib inline
plt.style.use('ggplot')

# Creating a new columns from the previous loaded dataframe
df = df[["radius_mean", "texture_mean", "perimeter_mean", "area_mean",
        "smoothness_mean", "compactness_mean", "concavity_mean", "concave points_mean",
"symmetry_mean", "fractal_dimension_mean",
        "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se", "compact
ness_se",
        "concavity_se", "concave points_se", "symmetry_se", "fractal_dimension_se", "ra
dius_worst", "texture_worst",
        "perimeter_worst", "area_worst", "smoothness_worst", "compactness_worst", "conc
avity_worst", "concave points_worst",
        "symmetry_worst", "fractal_dimension_worst", "diagnosis"]]

# Splitting the loaded dataframe into Input(X) and Output(y) features
# Splitting into input features and assigning it a variable X
X = df[["radius_mean", "texture_mean", "perimeter_mean", "area_mean",
        "smoothness_mean", "compactness_mean", "concavity_mean", "concave points_mean",
"symmetry_mean", "fractal_dimension_mean",
        "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se", "compact
ness_se",
        "concavity_se", "concave points_se", "symmetry_se", "fractal_dimension_se", "ra
dius_worst", "texture_worst",
        "perimeter_worst", "area_worst", "smoothness_worst", "compactness_worst", "conc
avity_worst", "concave points_worst",
        "symmetry_worst", "fractal_dimension_worst"]]

# Splitting into output feature and assigning it a label variable y
y = df["diagnosis"]

# Viewing the head of the filtered dataframe
df.head()
```

Out[4]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

5 rows × 31 columns

In [5]:

```
# Displaying the names of the columns
# Converting the names of the columns for the loaded dataframe into a numpy array
ColumnNames = df.columns.values

# Displaying the names of the columns
print(ColumnNames)
```

```
['radius_mean' 'texture_mean' 'perimeter_mean' 'area_mean'
 'smoothness_mean' 'compactness_mean' 'concavity_mean'
 'concave points_mean' 'symmetry_mean' 'fractal_dimension_mean'
 'radius_se' 'texture_se' 'perimeter_se' 'area_se' 'smoothness_se'
 'compactness_se' 'concavity_se' 'concave points_se' 'symmetry_se'
 'fractal_dimension_se' 'radius_worst' 'texture_worst' 'perimeter_worst'
 'area_worst' 'smoothness_worst' 'compactness_worst' 'concavity_worst'
 'concave points_worst' 'symmetry_worst' 'fractal_dimension_worst'
 'diagnosis']
```

In [6]:

```
# Describing the head of the loaded dataframe
df.describe()
```

Out[6]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compac
count	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	
std	3.524049	4.301036	24.298981	351.914129	0.014064	
min	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	

8 rows × 30 columns

In [7]:

```
# Displaying the shape of the input and output dataset
print('Input Shape: {}'.format(X.shape))
print('Output Shape: {}'.format(y.shape))
```

```
Input Shape: (569, 30)
Output Shape: (569,)
```

Data Pre-Processing

In [18]:

```
# Transforming the diagnosis column into a binary classification column
# where by 0 == "B"(Benign), and 1 == "M"(Malignant)
df["diagnosis"] = [1 if b == "M" else 0 for b in df["diagnosis"]]

# Checking the value counts for the Diagnosis column
df["diagnosis"].value_counts()

# Remember that 1 == Malignant, And 0 == Benign Type of Cancer
```

Out[18]:

```
0    357
1    212
Name: diagnosis, dtype: int64
```

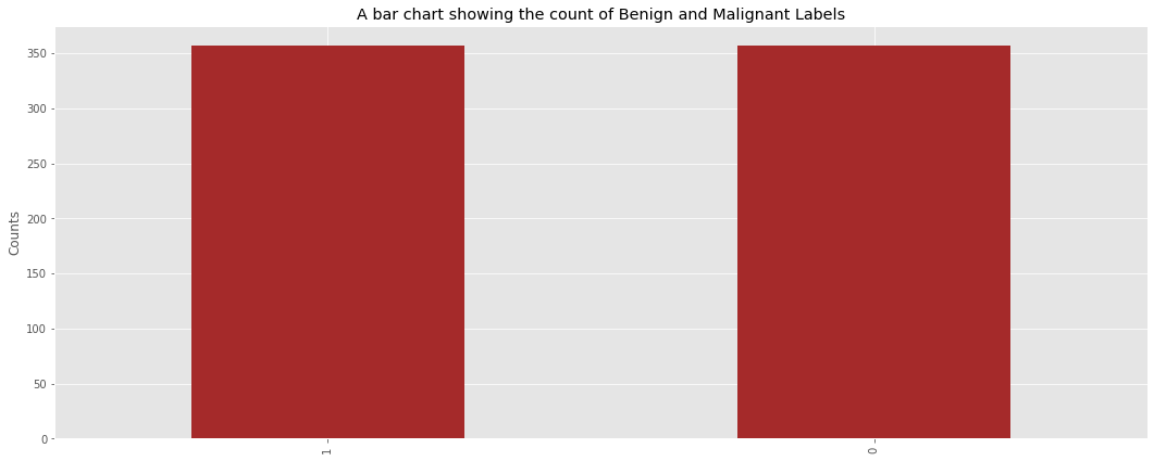
In [19]:

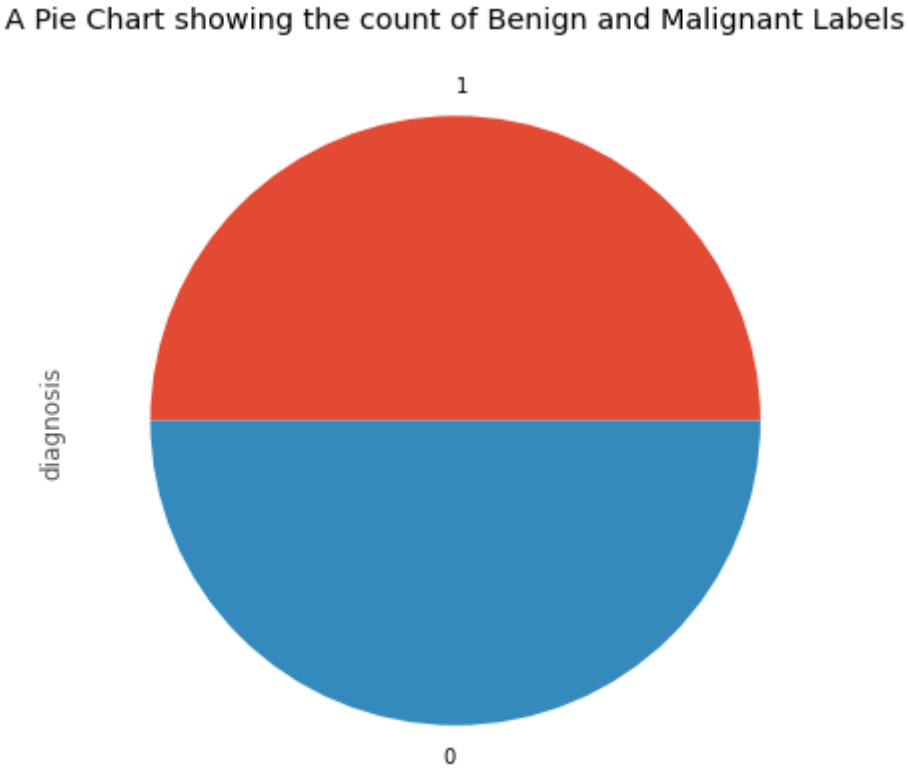
```
# Balancing the imbalanced dataset
# Separate majority and minority classes
df_majority = df[df["diagnosis"] == 0]
df_minority = df[df["diagnosis"] == 1]

# Upsample the minority class sample with replacement to
# Match the majority class reproducible results
df_minority_upsampled = resample(df_minority,
                                 replace = True,
                                 n_samples = 357,
                                 random_state = 123)

# Combine the majority class with upsampled minority class
Balanced_data = pd.concat([df_majority, df_minority_upsampled])

# Displaying the new class distribution
counter = Counter(Balanced_data["diagnosis"])
print(counter)
```

```
Counter({0: 357, 1: 357})
```

In [20]:

```
# Getting the value counts for the values in the diagnosis columns for the
# Balanced dataset
value_count = Balanced_data['diagnosis'].value_counts()

# Setting the figure size of the plot
plt.figure(figsize = (18, 7))

# Plotting the Count for the value counts in the diagnosis column of the balanced data
value_count.plot(kind = "bar", color = "brown")
plt.ylabel("Counts")
plt.title("A bar chart showing the count of Benign and Malignant Labels")
plt.grid(True)
plt.show()

# Plotting a pie chart of the imbalanced dataset
value_count.plot(kind = "pie", figsize=(18, 7))
plt.title("A Pie Chart showing the count of Benign and Malignant Labels")
plt.show()


# Printing the number of counts for the values of the labels in the diagnosis column
B, M = value_count
print("Number of Benign: {}".format(B))
print("Number of Malignant: {}".format(M))
```






```
Number of Benign: 357
Number of Malignant: 357
```

In [21]:

```
# Splitting the Balanced dataframe into input and output features
X = Balanced_data[["radius_mean", "texture_mean", "perimeter_mean", "area_mean",
        "smoothness_mean", "compactness_mean", "concavity_mean", "concave points_mean",
"symmetry_mean", "fractal_dimension_mean",
        "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se", "compact
ness_se",
        "concavity_se", "concave points_se", "symmetry_se", "fractal_dimension_se", "ra
dius_worst", "texture_worst",
        "perimeter_worst", "area_worst", "smoothness_worst", "compactness_worst", "conc
avity_worst", "concave points_worst",
        "symmetry_worst", "fractal_dimension_worst"]]

# Splitting into output feature and assigning it a label variable y
y = Balanced_data["diagnosis"]
```

Model Building

1). XGBoost Model

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominant competitive machine learning. XGBoost stands for Extreme Gradient Boosting.

In [26]:

```
# Displaying the shape of the data
print("Input Shape: {}, {}".format(X_train.shape, X_test.shape))
print("Output Shape: {}, {}".format(y_train.shape, y_test.shape))
```

```
Input Shape: (499, 30), (215, 30)
Output Shape: (499,), (215,)
```

In [27]:

```
# Fitting the model on training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

Out[27]:

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=N
one,
              validate_parameters=False, verbosity=None)
```

In [28]:

```
# Evualting the model to find how accurate it performs on it predictons
predictions = model.score(X_test, y_test)
print("The XGBoost Model is: {:.2f} accurate".format(predictions * 100))
```

```
The XGBoost Model is: 96.74 accurate
```