#import libraries   
**import** **numpy** **as** **np**   
**import** **pandas** **as** **pd**   
**import** **matplotlib.pyplot** **as** **plt**   
**import** **seaborn** **as** **sns**

#Load the data   
**#from** **google.colab** **import** files # Use to load data on Google Colab #uploaded = files.upload() # Use to load data on Google Colab df = pd.read\_csv('data.csv')   
df.head(7)

*#Count the number of rows and columns in the data set*  
df.shape

**Number of Rows:** 569, **Number of Columns:** 33

#Count the empty (NaN, NAN, na) values in each column  
df.isna().sum()

**Remove the column ‘*Unnamed: 32*’** from the original data set since it adds no value.

*#Drop the column with all missing values (na, NAN, NaN)*  
*#NOTE: This drops the column Unnamed*  
df = df.dropna(axis=1)

**Get the new count** of the number of rows and columns.

*#Get the new count of the number of rows and cols*  
df.shape

Image for post

Image for post

**Number of Rows:** 569, **Number of Columns:** 32

**Get a count** of the number of patients with Malignant (M) cancerous and Benign (B) non-cancerous cells.

*#Get a count of the number of 'M' & 'B' cells*  
df['diagnosis'].value\_counts()

Image for post

**# of Cancerous Cells:** 212 and **# of Non-Cancerous Cells:** 357

**Visualize the counts**, by creating a count plot.

*#Visualize this count*   
sns.countplot(df['diagnosis'],label="Count")

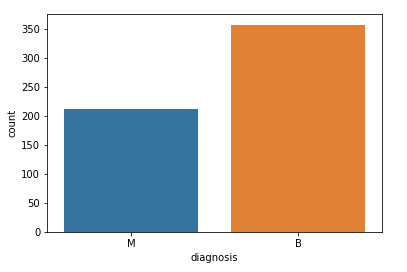
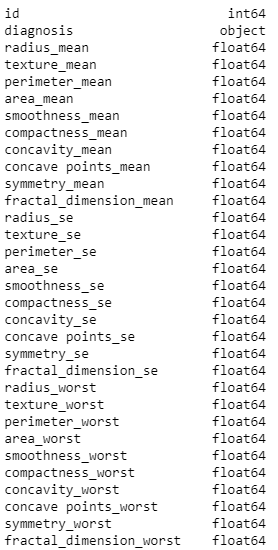


Chart displaying Malignant (cancerous) & Benign(non-cancerous) diagnosis

Look at the data types to see which **columns need to be transformed / encoded**. I can see from the data types that all of the columns/features are numbers except for the column ‘*diagnosis’,*which is categorical data represented as an object in python*.*

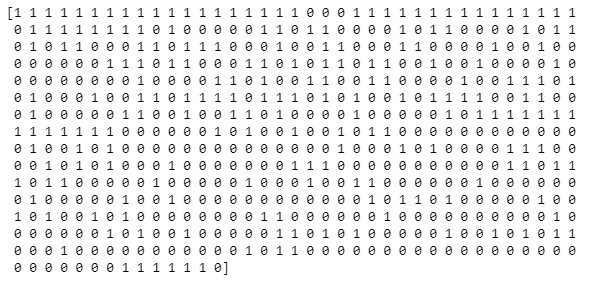
*#Look at the data types*   
df.dtypes



A list of the columns & their data types

**Encode the categorical data**. Change the values in the column ‘*diagnosis’* from **M and B** to **1 and 0** respectively, then print the results.

*#Encoding categorical data values (*  
**from** **sklearn.preprocessing** **import** LabelEncoder  
labelencoder\_Y = LabelEncoder()  
df.iloc[:,1]= labelencoder\_Y.fit\_transform(df.iloc[:,1].values)  
print(labelencoder\_Y.fit\_transform(df.iloc[:,1].values))



The encoded values of the feature/column diagnosis.

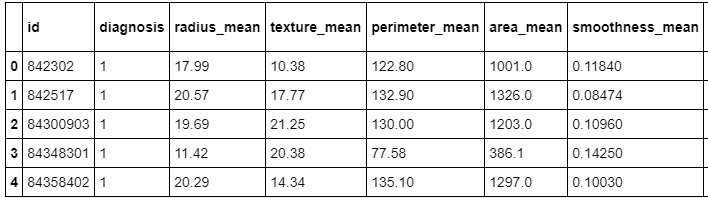
**Create a pair plot**. A “pairs plot” is also known as a scatter plot, in which one variable in the same data row is matched with another variable’s value.

sns.pairplot(df, hue="diagnosis")

Pair plot of all of the columns highlighting the diagnosis points in Orange (1) & Blue (0)

**Print the new data set** which now has only 32 columns. Print only the first 5 rows.

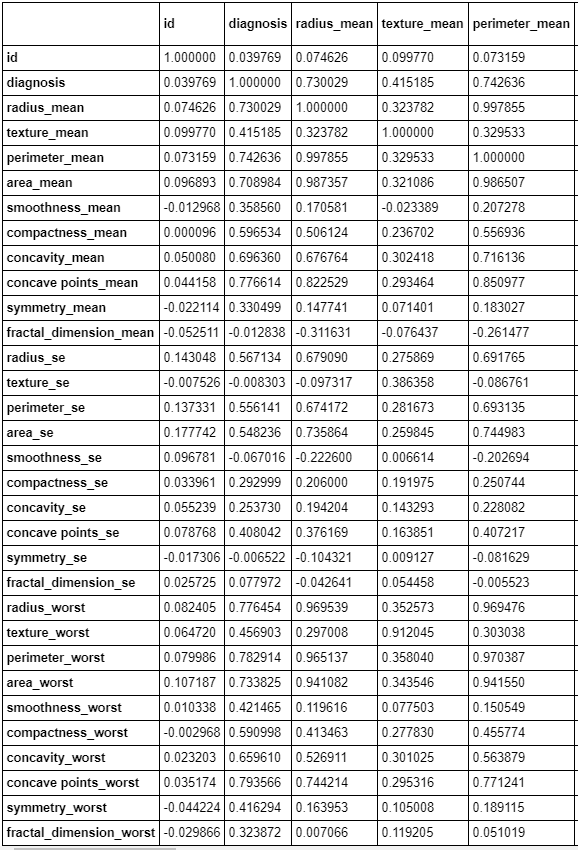
df.head(5)



5 rows of the new data set

Get the correlation of the columns.

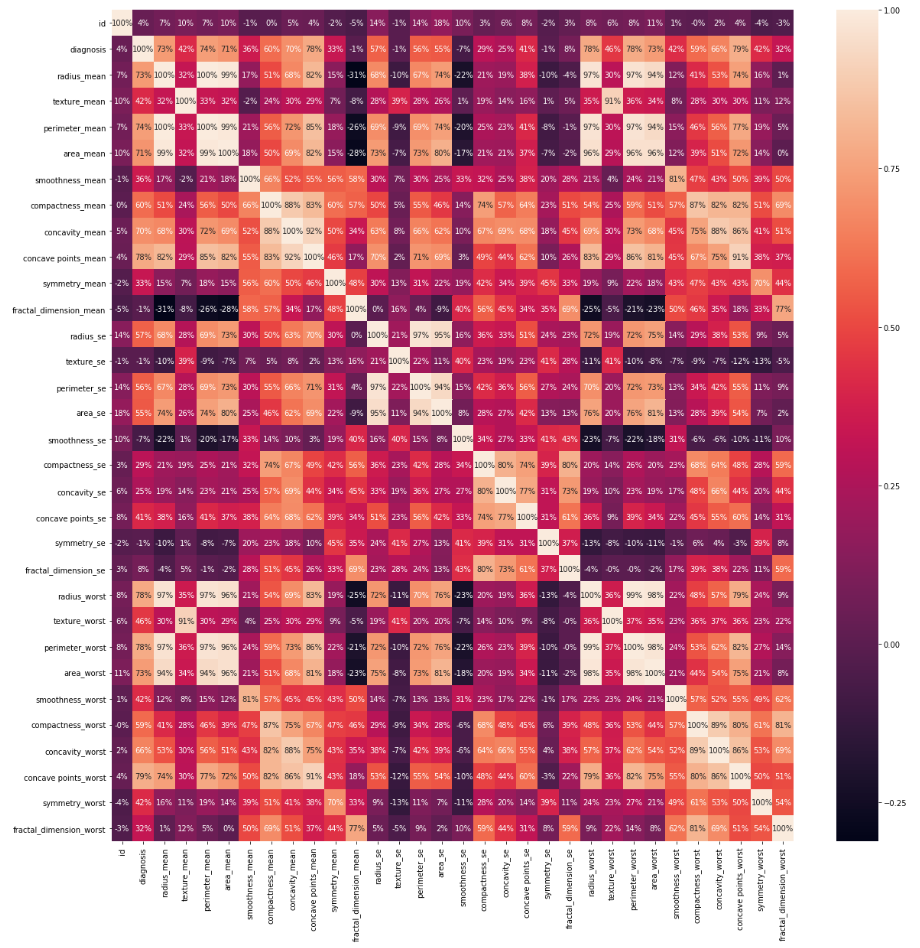
*#Get the correlation of the columns*  
df.corr()



Column correlation sample

**Visualize the correlation** by creating a heat map.

plt.figure(figsize=(20,20))   
sns.heatmap(df.corr(), annot=**True**, fmt='.0%')



Heat map of correlations

Now I am done exploring and cleaning the data. I will set up my data for the model by first **splitting the data set** into a feature data set also known as the independent data set (X), and a target data set also known as the dependent data set (Y).

X = df.iloc[:, 2:31].values   
Y = df.iloc[:, 1].values

**Split the data** again, but this time into 75% training and 25% testing data sets.

**from** **sklearn.model\_selection** **import** train\_test\_split  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.25, random\_state = 0)

**Scale the data** to bring all features to the same level of magnitude, which means the feature / independent data will be within a specific range for example 0–100 or 0–1.

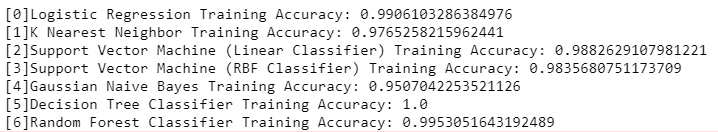
*#Feature Scaling*  
**from** **sklearn.preprocessing** **import** StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)

**Create a function** to hold many different models (e.g. *Logistic Regression*, *Decision Tree Classifier*, *Random Forest Classifier*) to make the classification. These are the models that will detect if a patient has cancer or not. Within this function I will also print the accuracy of each model on the training data.

**def** models(X\_train,Y\_train):  
   
 *#Using Logistic Regression*   
 **from** **sklearn.linear\_model** **import** LogisticRegression  
 log = LogisticRegression(random\_state = 0)  
 log.fit(X\_train, Y\_train)  
   
 *#Using KNeighborsClassifier*   
 **from** **sklearn.neighbors** **import** KNeighborsClassifier  
 knn = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)  
 knn.fit(X\_train, Y\_train)  
  
 *#Using SVC linear*  
 **from** **sklearn.svm** **import** SVC  
 svc\_lin = SVC(kernel = 'linear', random\_state = 0)  
 svc\_lin.fit(X\_train, Y\_train)  
  
 *#Using SVC rbf*  
 **from** **sklearn.svm** **import** SVC  
 svc\_rbf = SVC(kernel = 'rbf', random\_state = 0)  
 svc\_rbf.fit(X\_train, Y\_train)  
  
 *#Using GaussianNB*   
 **from** **sklearn.naive\_bayes** **import** GaussianNB  
 gauss = GaussianNB()  
 gauss.fit(X\_train, Y\_train)  
  
 *#Using DecisionTreeClassifier*   
 **from** **sklearn.tree** **import** DecisionTreeClassifier  
 tree = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)  
 tree.fit(X\_train, Y\_train)  
  
 *#Using RandomForestClassifier method of ensemble class to use Random Forest Classification algorithm*  
 **from** **sklearn.ensemble** **import** RandomForestClassifier  
 forest = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)  
 forest.fit(X\_train, Y\_train)   
 *#print model accuracy on the training data.*  
 print('[0]Logistic Regression Training Accuracy:', log.score(X\_train, Y\_train)) print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X\_train, Y\_train)) print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:', svc\_lin.score(X\_train, Y\_train)) print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:', svc\_rbf.score(X\_train, Y\_train)) print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X\_train, Y\_train)) print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X\_train, Y\_train)) print('[6]Random Forest Classifier Training Accuracy:', forest.score(X\_train, Y\_train))   
 **return** log, knn, svc\_lin, svc\_rbf, gauss, tree, forest

**Create the model** that contains all of the models, and look at the accuracy score on the training data for each model to classify if a patient has cancer or not.

model = models(X\_train,Y\_train)



The accuracy of each model on the training data