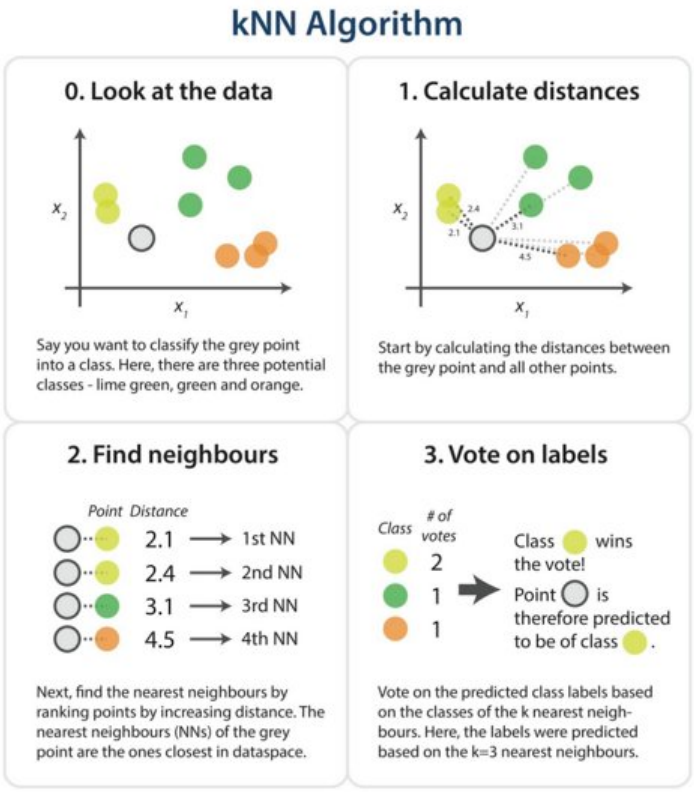
What is KNN Algorithm?

" ***K nearest neighbors or KNN Algorithm*** is a simple algorithm that uses the entire dataset in its **training** phase. Whenever a prediction is required for an unseen data instance, it searches through the entire training dataset for k-most similar instances and the data with the most similar instance is finally returned as the prediction."

How does a KNN Algorithm work?

The k-nearest neighbor algorithm uses a very simple approach to perform classification. When tested with a new example, it looks through the training data and finds the k training examples that are closest to the new example. It then assigns the most common class label (among those k-training examples) to the test example



1. Import Libraries

In [14]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

2. Read Data

In [83]:
```
df=pd.read_csv('diabetes.csv')
df.head()
```

Out[83]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3.Exploratory data analysis

1. we can see dataset columnns not have proper name

In [84]:
```
# total number of row and columns
df.shape
```

Out[84]: (768, 9)

Rename columns

col_names=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']
df.columns=col_names col_names

In [85]:
```
df.head()
```

Out[85]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Droping the 0th index row which have columns name

the `iloc[1:]` indexing is used to select all rows starting from index 1 (excluding the first row).

In [86]:
```
df = df.iloc[1:]
```

In [87]:
```
df.head(2)
```

Out[87]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

In [88]:
```
# Gives information about dataset like data type,columns,null value count,memory usage
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 767 entries, 1 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               767 non-null    int64  
 1   Glucose                   767 non-null    int64  
 2   BloodPressure             767 non-null    int64  
 3   SkinThickness             767 non-null    int64  
 4   Insulin                   767 non-null    int64  
 5   BMI                       767 non-null    float64
 6   DiabetesPedigreeFunction  767 non-null    float64
 7   Age                       767 non-null    int64  
 8   Outcome                   767 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [89]:
```
# basic statistic detail about the data
df.describe()
```

Out[89]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.0000
mean	3.842243	120.859192	69.101695	20.517601	79.903520	31.990482	0.471674	33.219035	0.3481
std	3.370877	31.978468	19.368155	15.954059	115.283105	7.889091	0.331497	11.752296	0.4766
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.0000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243500	24.000000	0.0000
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000	0.371000	29.000000	0.0000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.000000	1.0000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0000

In [90]:
```
df.describe().T
```

Out[90]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	767.0	3.842243	3.370877	0.000	1.0000	3.000	6.000	17.00
Glucose	767.0	120.859192	31.978468	0.000	99.0000	117.000	140.000	199.00
BloodPressure	767.0	69.101695	19.368155	0.000	62.0000	72.000	80.000	122.00
SkinThickness	767.0	20.517601	15.954059	0.000	0.0000	23.000	32.000	99.00
Insulin	767.0	79.903520	115.283105	0.000	0.0000	32.000	127.500	846.00
BMI	767.0	31.990482	7.889091	0.000	27.3000	32.000	36.600	67.10
DiabetesPedigreeFunction	767.0	0.471674	0.331497	0.078	0.2435	0.371	0.625	2.42
Age	767.0	33.219035	11.752296	21.000	24.0000	29.000	41.000	81.00
Outcome	767.0	0.348110	0.476682	0.000	0.0000	0.000	1.000	1.00

The quetion creeping out of the summary

Can minimum values of below listed columns are zeros (0) On these columns a values of zero does not make sense and thus indicates missing values following columns or variables have an invalid zero values

1. Glocose
2. BloodPressure
3. SkinThickness
4. Insulin
5. BMI

It is better to replace zeros with nan since after that count tham would be easier and zeros need to be replace with suitable

In [91]:
```
df_copy=df.copy(deep=True )
```

df: It refers to the original DataFrame that you have in your code or workspace.

copy(): It is a method available for DataFrames in pandas. This method is used to create a copy of the DataFrame.

deep=True: The deep parameter specifies whether to create a shallow copy (with deep=False) or a deep copy (with deep=True). In this case, deep=True is used, which means that a deep copy of the DataFrame is created.

In [104]:
```
dPressure','SkinThickness','Insulin','BMI']]=df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI'
```

df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] : This selects the subset of columns from **df_copy** that includes **'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', and 'BMI'.** The double square brackets [['...']] are used to select multiple columns by providing a list of column names.

= df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0, np.nan): This assigns the result of the replacement operation to the selected subset of columns. The **replace()** method is used to **replace specific values within a DataFrame. In this case, it replaces all occurrences of 0 with np.nan**, which stands for **"Not a Number"** and represents missing or undefined values.

showing the count of nan values

In [105]:
```
df_copy.isnull().sum()
```

Out[105]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   373
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

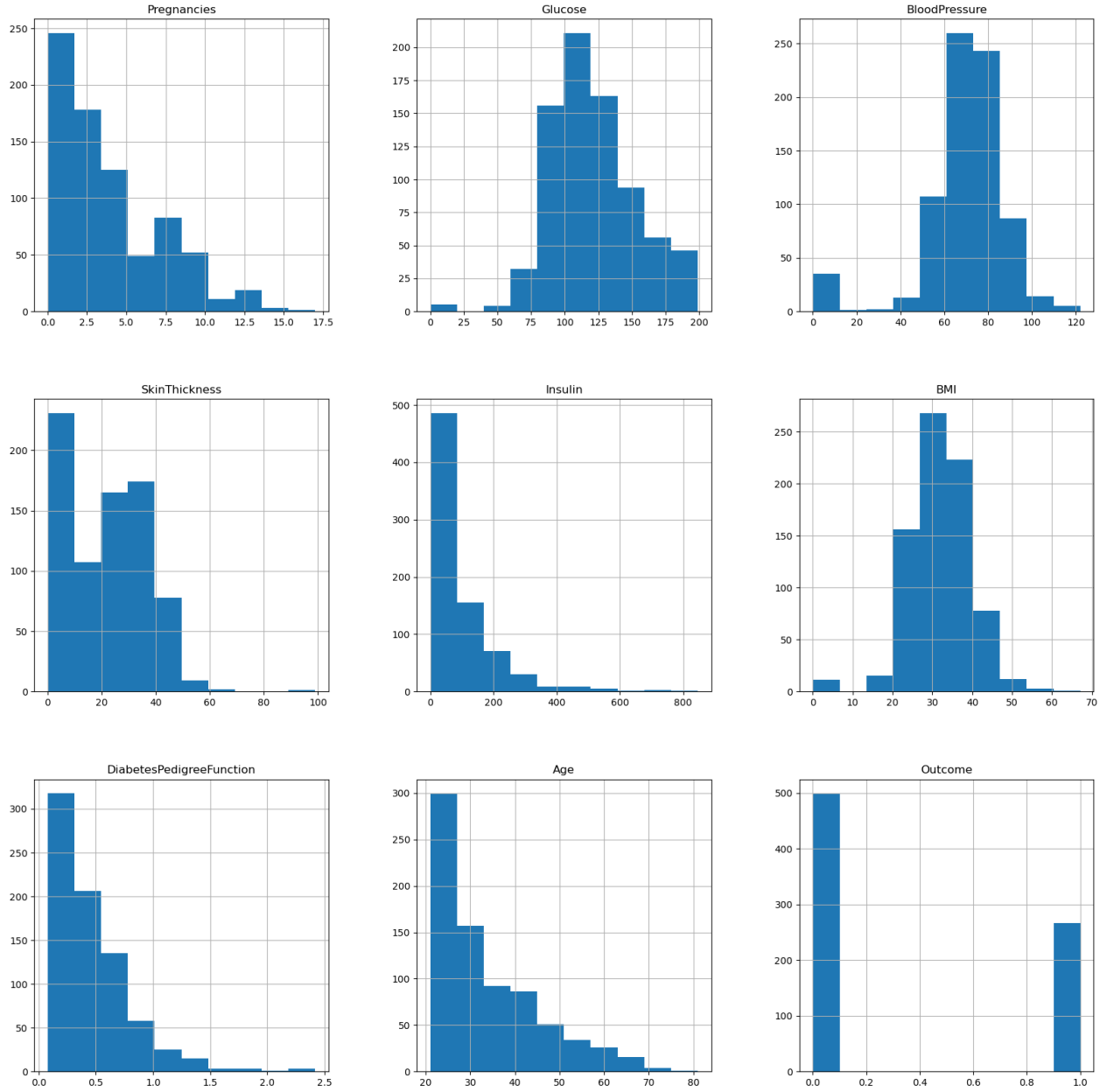
`df_copy` : It refers to the DataFrame on which the operation is performed.

`.isnull()` : It is a method available for DataFrames in pandas. It returns a DataFrame of the same shape as the original, where each element is a Boolean value indicating whether the corresponding element in the original DataFrame is missing (NaN) or not. Missing values are represented as True, and non-missing values are represented as False.

`.sum()` : It is a method chained to isnull() to calculate the sum of True values in each column. Since True is considered as 1 and False as 0 in numeric operations, summing the Boolean values gives the count of missing values (NaN) in each column.

Too fill this nan values the data distribution need to understand

In [106]:
```
p = df.hist(figsize = (20,20))
```



df: It refers to the DataFrame for which you want to create the histograms.

hist(): It is a method available for DataFrames in pandas. When applied to a DataFrame, it generates histograms for each numerical column by default.

figsize=(20,20): This parameter specifies the size of the figure in which the histograms will be plotted. The figsize argument takes a tuple of two values, (width, height), to define the dimensions of the figure. In this case, (20, 20) is used to create a square-shaped figure with a width and height of 20 units.

Aiming to impute nan values for the columns in accordance with their distribution

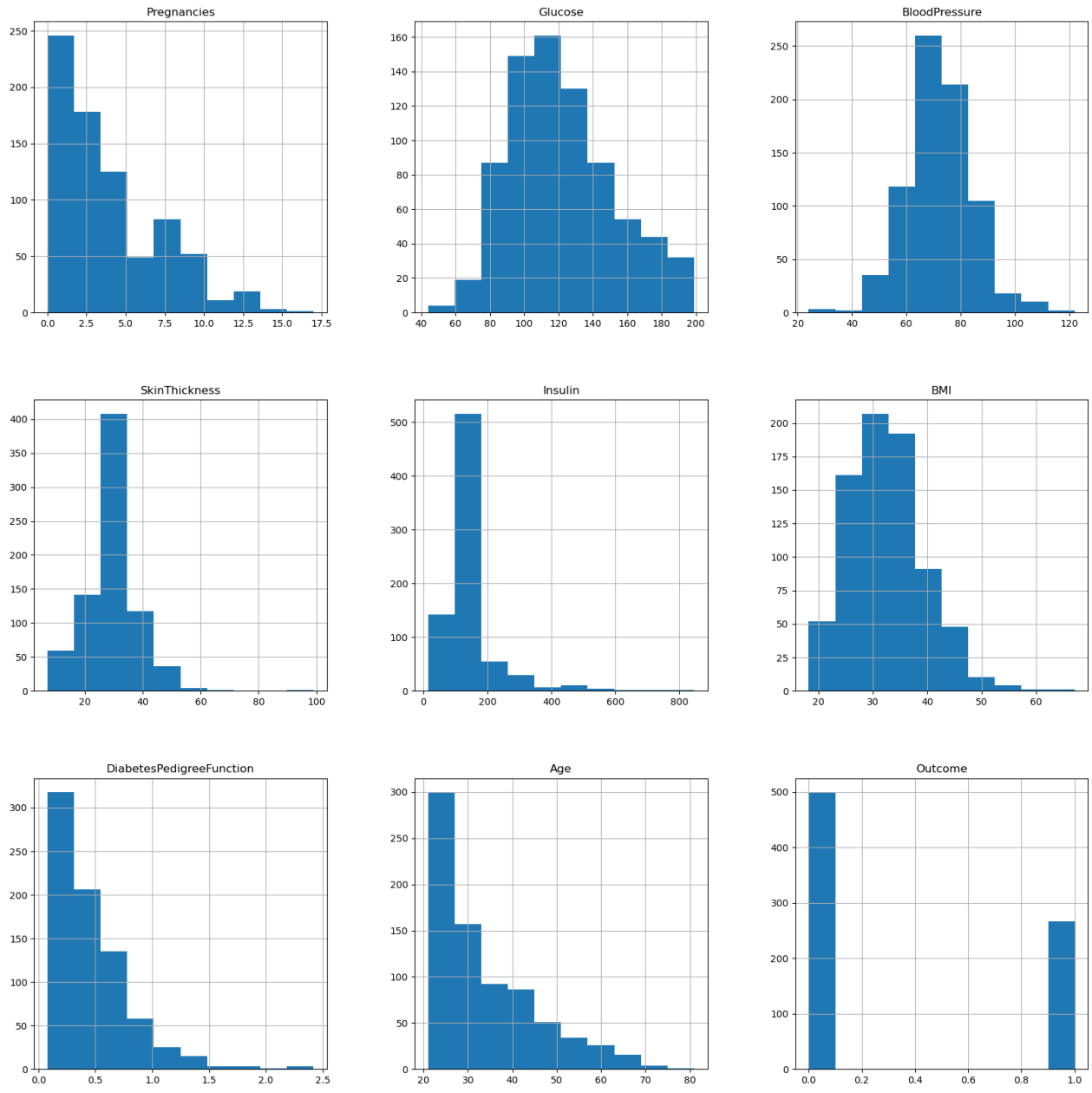
In [107]:
```
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(),inplace=True)
```

By executing **df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace=True)** , the missing values in the **'Glucose'** column of df_copy will be replaced with the **mean value** of the **'Glucose'** column. **This is a common approach to imputing missing values**

In [108]:
```
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(),inplace=True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(),inplace=True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(),inplace=True)
df_copy['BMI'].fillna(df_copy['BMI'].median(),inplace=True)
```

Ploting after nan remove

In [109]:
```
A=df_copy.hist(figsize=(20,20))
```



`df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace=True)` , the missing values in the *'SkinThickness'* column of df_copy will be replaced with the **median value** of the 'SkinThickness' column. Filling missing values with the median is a common imputation technique, especially when dealing with skewed data or in scenarios where outliers might affect the mean value.`

Skewness

A **left skewed distribution** has a long left tail .left shewed distribution are also called negatively distribution . Thats because there is long tail in the negative direction on the number line. The mean is also to the left of the peak .

If you have a left-skewed distribution in your data, it means that the majority of the data points are concentrated on the right side of the distribution, while the left side has a tail that extends towards the lower values. This indicates that there are fewer data points with lower values and more data points with higher values.

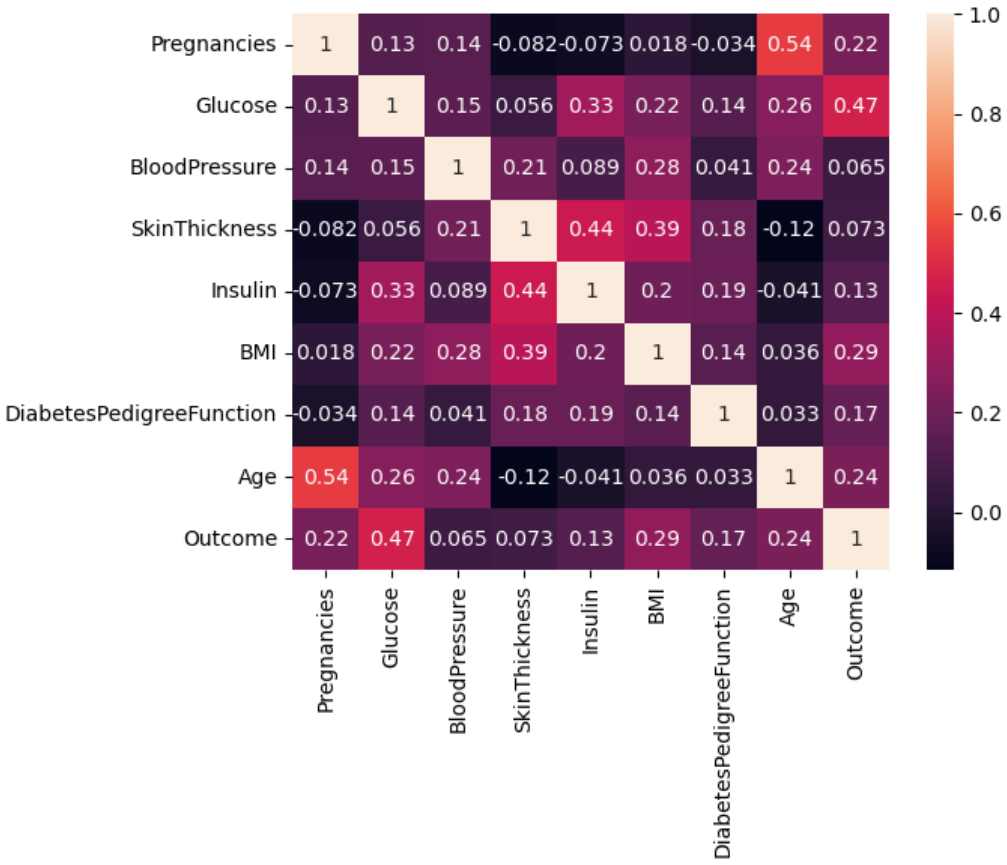
A **Right-Skewed distribution** has s long right tail. Right Skewed distribution are also called positive skewed distribution. That's because there is long tail in the positive direction on the number line .The mean is also to right of the peak.

If you have a right-skewed distribution in your data, it means that the majority of the data points are concentrated on the left side of the distribution, while the right side has a tail that extends towards the higher values. This indicates that there are fewer data points with higher values and more data points with lower values.

heatmap for the unclean dataset

In [118]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[118]: <AxesSubplot:>



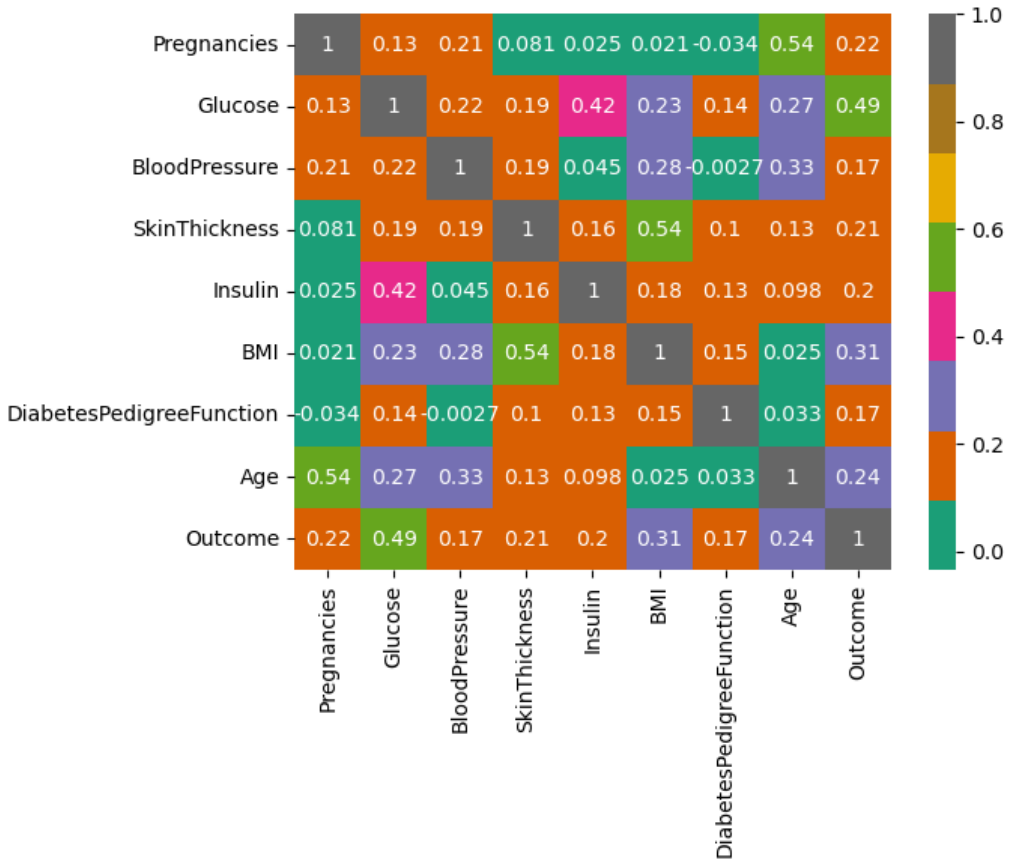
The code `sns.heatmap(df.corr(), annot=True)` generates a heatmap using the seaborn library to visualize the correlation matrix of a DataFrame `df`.

Here's a breakdown of the code:

- `df.corr()` calculates the pairwise correlation between columns of the DataFrame `df`. This generates a correlation matrix, where each value represents the correlation coefficient between two columns.
- `sns.heatmap()` is a seaborn function used to create a heatmap. It takes the correlation matrix as input.
- `annot=True` is an argument in `sns.heatmap()` that enables the annotation of the heatmap with the correlation values. This adds numerical values to each cell in the heatmap, representing the correlation coefficients.

In [122]:
```
sns.heatmap(df_copy.corr(),annot=True,cmap='Dark2')
```

Out[122]: <AxesSubplot:>

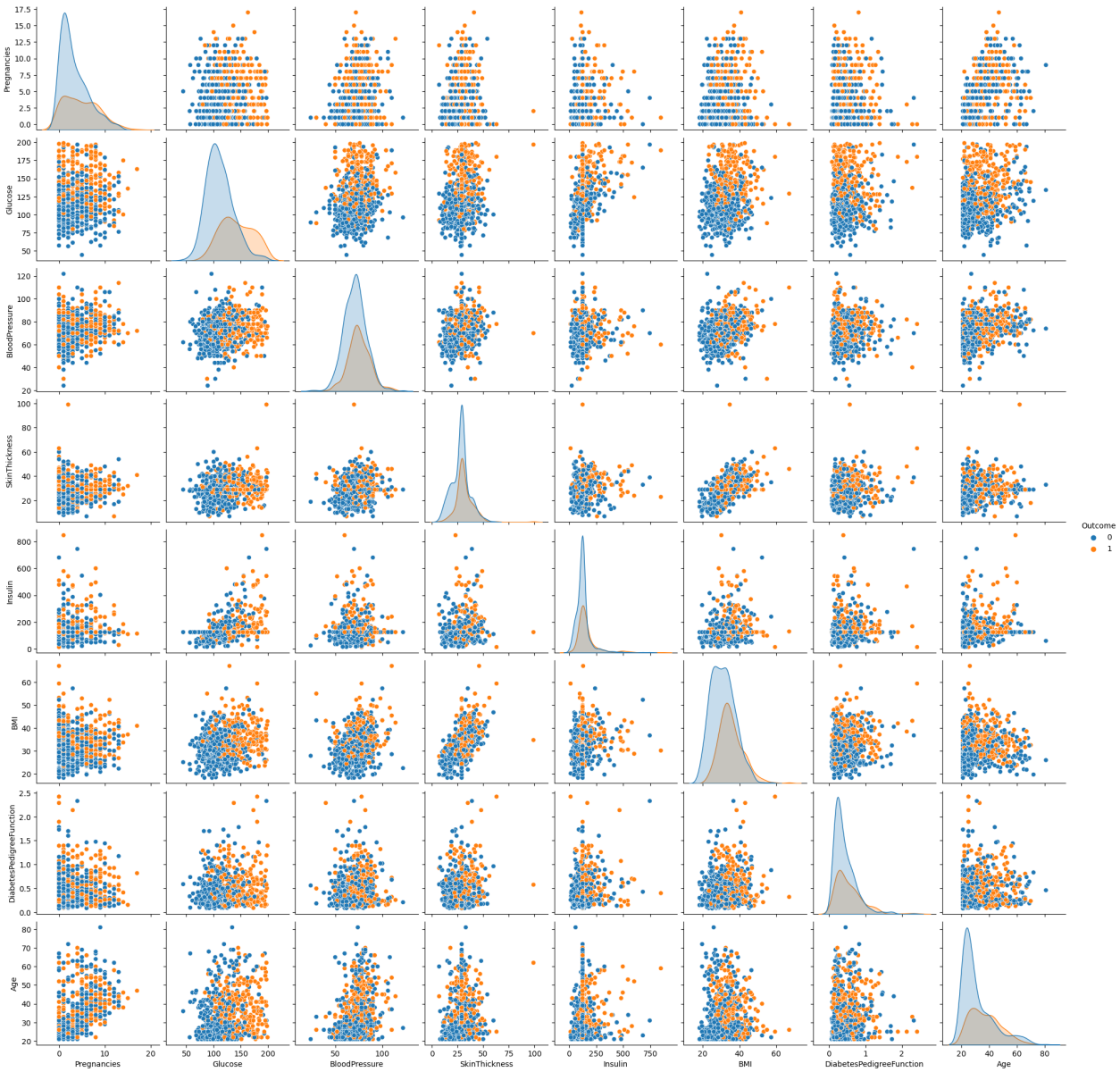


The `cmap='Dark2'` option sets the color map to 'Dark2', which is a predefined color map in seaborn. It is a qualitative color map that provides a set of distinct, contrasting colors suitable for categorical or discrete data.

Pairplot for clean dataset

In [123]:
```
sns.pairplot(df_copy,hue='Outcome')
```

Out[123]:
```
<seaborn.axisgrid.PairGrid at 0x22e5a59c940>
```



The code `sns.pairplot(df_copy, hue='Outcome')` generates a pair plot using the seaborn library to visualize pairwise relationships between variables in a DataFrame `df_copy`, while also differentiating the data points based on the 'Outcome' column.

Here's a breakdown of the code:

- `sns.pairplot()` is a seaborn function used to create a grid of scatter plots, where each variable is plotted against all other variables in the DataFrame.
- `df_copy` is the DataFrame containing the variables to be plotted.
- `hue='Outcome'` is an argument in `sns.pairplot()` that specifies the column to use for color differentiation. In this case, the 'Outcome' column is used to assign different colors to the data points based on their outcome.

In [124]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [127]:
```
X=df_copy.drop(['Outcome'],axis=1)
X
```

Out[127]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
5	5	116.0	74.0	29.0	125.0	25.6	0.201	30
...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
764	2	122.0	70.0	27.0	125.0	36.8	0.340	27
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30
766	1	126.0	60.0	29.0	125.0	30.1	0.349	47
767	1	93.0	70.0	31.0	125.0	30.4	0.315	23

767 rows × 8 columns

The code `X = df_copy.drop(['Outcome'], axis=1)` creates a new DataFrame `X` by dropping the 'Outcome' column from the `df_copy` DataFrame. The `drop()` function in pandas is used to remove specified columns from a DataFrame.

Here's a breakdown of the code:

- `df_copy` is the original DataFrame containing your data.
- `.drop(['Outcome'], axis=1)` is a method called on `df_copy` to drop the specified column(s). The `['Outcome']` parameter specifies the column(s) to be dropped. In this case, it drops the 'Outcome' column. The `axis=1` parameter indicates that we want to drop columns, not rows (where `axis=0` would be used).
- The resulting DataFrame, without the 'Outcome' column, is assigned to the variable `X`.

In [129]:
```
X=scaler.fit_transform(X)
X
```

Out[129]:
```
array([[-0.84372629, -1.20482941, -0.52956011, ..., -0.85144308,
        -0.36426474, -0.18894038],
       [ 1.23423997,  2.01661929, -0.69489941, ..., -1.33143942,
         0.60470064, -0.1037951 ],
       [-0.84372629, -1.0733417 , -0.52956011, ..., -0.63326292,
        -0.91968415, -1.0403932 ],
       ...,
       [ 0.343683  , -0.02144009, -0.03354219, ..., -0.90962445,
        -0.68423462, -0.27408566],
       [-0.84372629,  0.14291954, -1.02557802, ..., -0.34235605,
        -0.37030191,  1.17338414],
       [-0.84372629, -0.941854  , -0.1988815 , ..., -0.29872002,
        -0.47293375, -0.87010264]])
```

The code `X = scaler.fit_transform(X)` applies the `fit_transform()` method of the `StandardScaler` to scale the features in the `X` DataFrame. It standardizes the data by subtracting the mean and dividing by the standard deviation, based on the computed parameters from `scaler.fit()`.

In [141]:
```
cols=[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age
```

In [142]:
```
X=pd.DataFrame(X)
```

In [143]:
```
X.columns=cols
```

In [144]:
```
X.head()
```

Out[144]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	-0.843726	-1.204829	-0.529560	-0.011423	-0.181663	-0.851443	-0.364265	-0.188940
1	1.234240	2.016619	-0.694899	-0.011423	-0.181663	-1.331439	0.604701	-0.103795
2	-0.843726	-1.073342	-0.529560	-0.694122	-0.540538	-0.633263	-0.919684	-1.040393
3	-1.140579	0.504511	-2.678971	0.671276	0.316130	1.548539	5.482732	-0.018650
4	0.343683	-0.185800	0.131797	-0.011423	-0.181663	-0.996897	-0.817052	-0.274086

In [145]:
```
y=df_copy.Outcome
```

- `df_copy.Outcome` accesses the 'Outcome' column of the `df_copy` DataFrame, which contains the target variable values.
- The values from the 'Outcome' column are assigned to the variable `y`.

In [146]:
```
y.head()
```

Out[146]:
```
1    0
2    1
3    0
4    1
5    0
Name: Outcome, dtype: int64
```

In [149]:
```
from sklearn.model_selection import train_test_split as tts
X_train,X_test,y_train,y_test=tts(X,y,test_size=0.33,random_state=42)
```

K neighbors classifier

In [151]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [154]:
```
train_score=[]
test_score=[]
for i in range(1,15):
    knn=KNeighborsClassifier(i)
    knn.fit(X_train,y_train)
    train_score.append(knn.score(X_train,y_train))
    test_score.append(knn.score(X_test,y_test))
```

The code snippet you provided performs a loop to evaluate the performance of the K-Nearest Neighbors (KNN) classifier for different values of `i`, which represents the number of neighbors. The loop iterates from 1 to 14, and for each iteration, it performs the following steps:

1. Create an instance of the KNN classifier with `i` neighbors:
2. Fit the KNN classifier to the training data:
3. Calculate and store the training accuracy using the `score()` method:
4. Calculate and store the testing accuracy using the `score()` method:

At the end of the loop, you will have two lists: `train_score` and `test_score`, which contain the training and testing accuracies for each value of `i`, respectively.

In [157]:
```
max_train_score=max(train_score)
train_score_ind=[i for i,v in enumerate(train_score) if v==max_train_score]
print('max train score : {} % and K= {}'.format(max_train_score*100,list(map(lambda x: x+1 ,train_score_ind))))
```

```
max train score : 100.0 % and K= [1]
```

- `train_score` is a list that contains the training scores for different values of `i`, which were calculated in the previous loop.

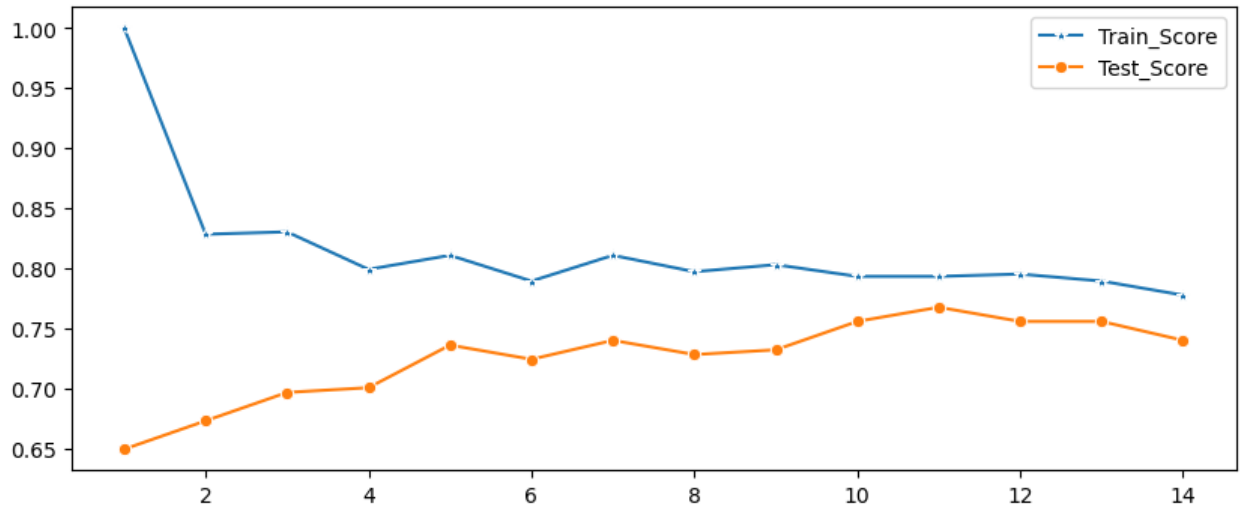
- `max_train_score` is a variable that will store the maximum training score.
- The code `train_score_ind = [i for i, v in enumerate(train_score) if v == max_train_score]` creates a list `train_score_ind` that contains the indices of the elements in the `train_score` list where the value is equal to `max_train_score`.
- The code `print('max train score : {} % and K= {}'.format(max_train_score*100, list(map(lambda x: x+1, train_score_ind))))` prints the maximum training score as a percentage and the corresponding values of `K` (number of neighbors) for which the maximum training score was achieved.

In [159]:
```
max_test_score=max(test_score)
test_score_ind=[i for i ,v in enumerate(test_score) if v==max_test_score]
print('max test score :{} % and k={}'.format(max_test_score*100,list(map(lambda x:x+1,test_score_ind))))
```

```
max test score :76.77165354330708 % and k=[11]
```

Result Visualization

In [161]:
```
plt.figure(figsize=(10,4))
p=sns.lineplot(range(1,15),train_score,marker='*',label='Train_Score')
p=sns.lineplot(range(1,15),test_score,marker='o',label='Test_Score')
```



In [162]:
```
knn=KNeighborsClassifier(13)
knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

Out[162]:
```
0.7559055118110236
```

The code `knn = KNeighborsClassifier(13)` creates an instance of the K-Nearest Neighbors (KNN) classifier with 13 neighbors.

In [165]:
```
from sklearn import metrics
confusion_matrix=metrics.confusion_matrix
```

In [168]:
```
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
pd.crosstab(y_test,y_pred,rownames=['True'],colnames=['Predicted'],margins=True)
```

Out[168]:

Predicted	0	1	All
True			
0	145	24	169
1	38	47	85
All	183	71	254

The code `y_pred = knn.predict(X_test)` is used to make predictions on the test data using the trained KNN classifier.

Here's how the code works:

- `X_test` represents the test data, which is a feature matrix.
- `knn.predict(X_test)` applies the trained KNN classifier (`knn`) to the test data and predicts the target variable (`y`) for the test data.

The code `cm = confusion_matrix(y_test, y_pred)` calculates the confusion matrix based on the true labels (`y_test`) and the predicted labels (`y_pred`).

Here's how the code works:

- `y_test` represents the true labels for the test data.
- `y_pred` contains the predicted labels for the test data using the KNN classifier.

The code `pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)` creates a cross-tabulation table that presents the predicted labels (`y_pred`) against the true labels (`y_test`).

Here's how the code works:

- `y_test` represents the true labels for the test data.
- `y_pred` contains the predicted labels for the test data using the KNN classifier.
- `rownames=['True']` sets the name for the row index of the table as "True".
- `colnames=['Predicted']` sets the name for the column index of the table as "Predicted".
- `margins=True` adds the row and column totals to the table.

By executing these code snippets, you will obtain the predicted labels (`y_pred`), the confusion matrix (`cm`), and the cross-tabulation table showing the predicted labels against the true labels.

Classification Report

In [169]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

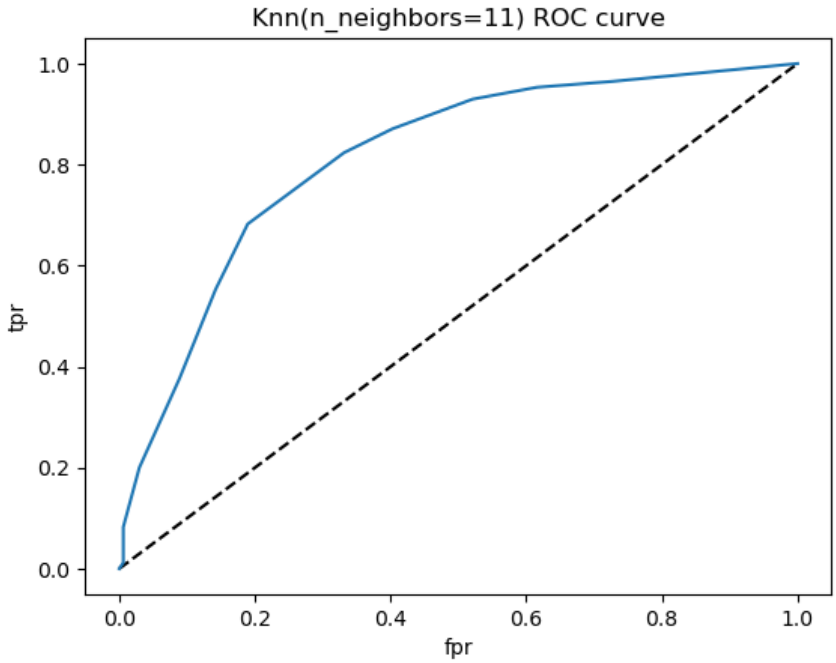
```
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       169
           1       0.66      0.55      0.60        85

    accuracy                           0.76       254
   macro avg       0.73      0.71      0.71       254
weighted avg       0.75      0.76      0.75       254
```

3. ROC - AUC

ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things (e.g If a patient has a disease or no). Better models can accurately distinguish between the two. Whereas, a poor model will have difficulties in distinguishing between the two

In [170]:
```
from sklearn.metrics import roc_curve
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [171]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=11) ROC curve')
plt.show()
```



Area under ROC curve

In [172]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

Out[172]: 0.8122868082144101

In [173]:
```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)

print("Best Score:" + str(knn_cv.best_score_))
print("Best Parameters: " + str(knn_cv.best_params_))
```

```
Best Score:0.7705967235378999
Best Parameters: {'n_neighbors': 25}
```

The code snippet you provided is used to print the best score and best parameters from a cross-validated grid search using the K-Nearest Neighbors (KNN) classifier.

Here's what the code does:

- `knn_cv.best_score_` retrieves the best score achieved during the cross-validation process. The best score represents the average performance metric (such as accuracy or F1-score) obtained across all folds and parameter combinations. It gives an indication of the model's overall performance.
- `knn_cv.best_params_` retrieves the best parameters found during the grid search. These parameters correspond to the combination of hyperparameters that yielded the best performance. The hyperparameters could include the number of neighbors (K), distance metric, or other settings specific to the KNN classifier.


thank
you