# Problem Statement

This dataset contains around 770 samples of data containing information about people's health parameters, e.g. Glucose level, Blood-Pressure, Age, BMI index, Skin thickness, no of pregnancies etc.. Our aim is to build a model that accurately classifies individuals with diabetes based on the given health parameters.

# Reading the data and checking statistical parameters.

In [3]:
```
import numpy as np
import pandas as pd
```

In [4]:
```
df=pd.read_csv('/content/diabetes.csv')
df
```

Out[4]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabe |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | |

768 rows × 9 columns

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

**The given dataset does not contain any null values.**

In [6]:
```
# Checking statistical measures[Average,Standard Deviation,Quartiles] of eac
df.describe()
```

Out[6]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | |
|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 76 |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 3 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 2 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 3 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 3 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 6 |

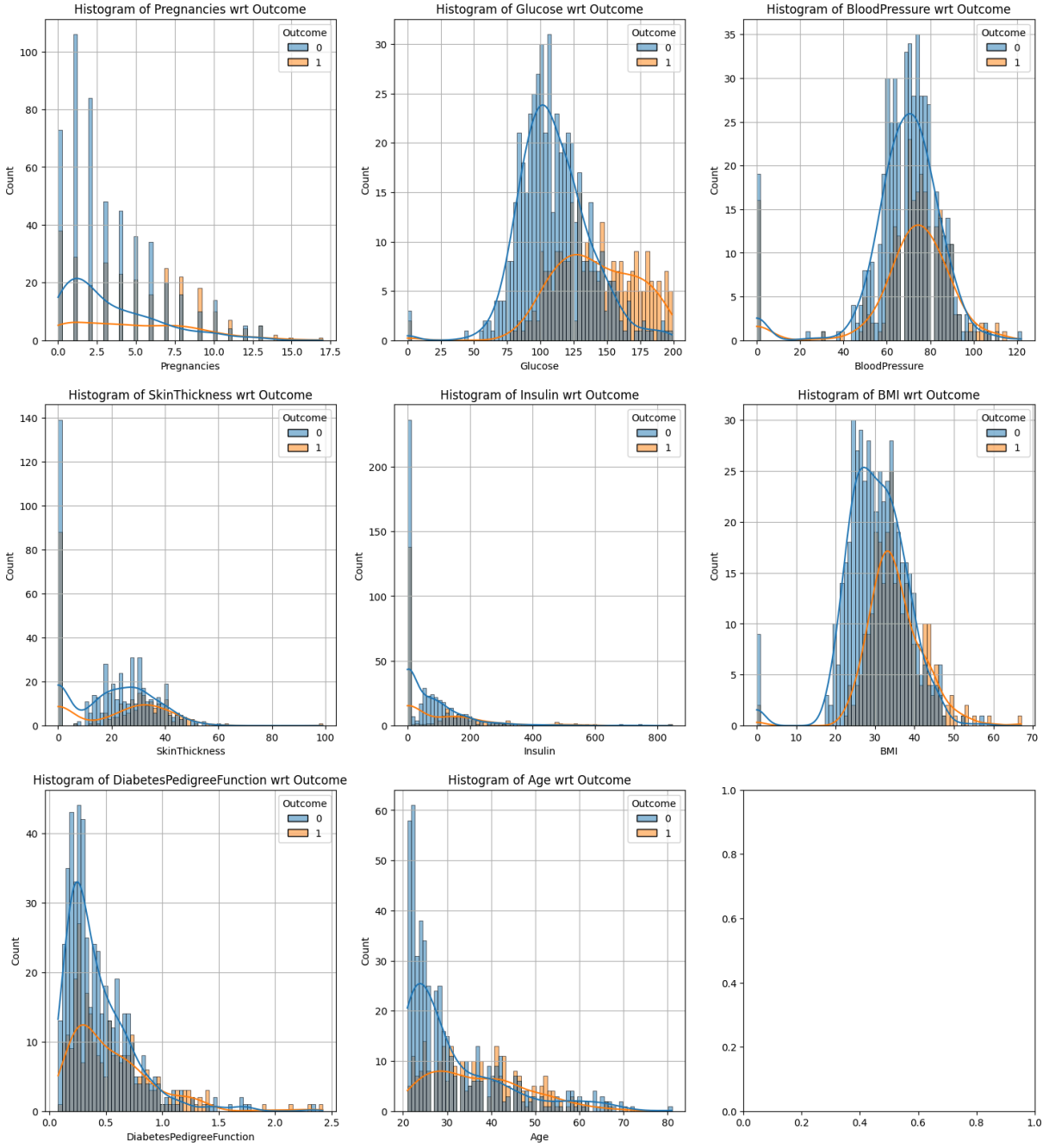
# Data visualization

1. Plotting Histograms of each numerical features wrt the target variable, i.e. 'Outcome'.
2. Plotting Boxplot for observing outliers in numerical variables.
3. Generating a pair plot to investigate how features influence each other.
4. Finally a Correlation table to numerically visualize it.

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as pt
```

**1. Histogram**

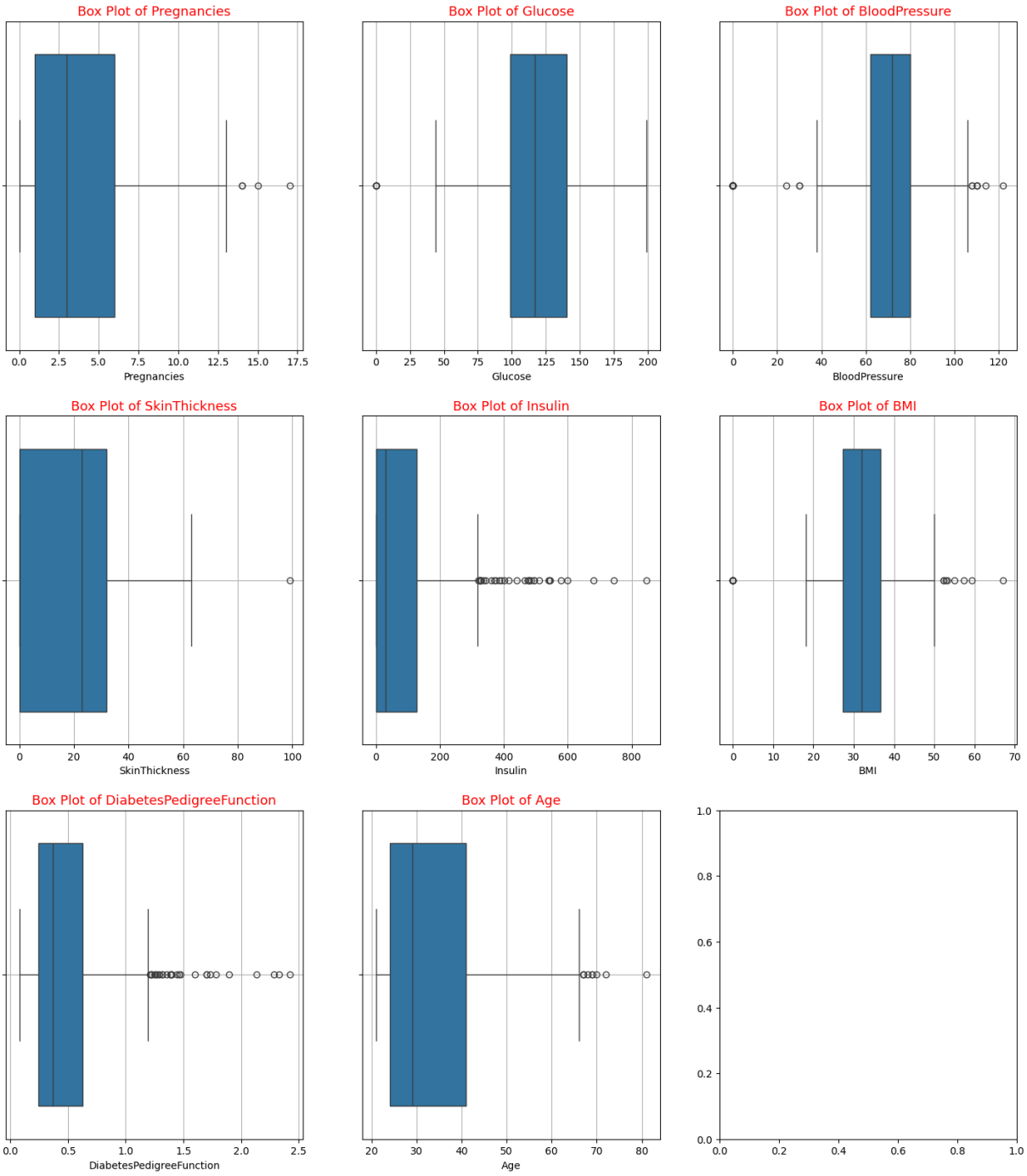
In [8]:
```
fig,axes=plt.subplots(3,3,figsize=(18,20))
for i,ax in enumerate(axes.flatten()):
    if i>=len(df.columns)-1:
        break
    sns.histplot(data=df,x=df[df.columns[i]],hue=df.Outcome,bins=70,ax=ax,kd
    ax.set(xlabel=df.columns[i])
    ax.set_title(f'Histogram of {df.columns[i]} wrt Outcome')
    ax.grid(True)
```




## 2. Boxplot

In [9]:
```
fig,axes=plt.subplots(3,3,figsize=(18,20))
for i,ax in enumerate(axes.flatten()):
    if i>=len(df.columns)-1:
        break
```

```
    sns.boxplot(data=df,x=df[df.columns[i]], ax=ax,)
    ax.set_title(f'Box Plot of {df.columns[i]}',color='red',fontsize=13,)
    ax.grid(True)
```



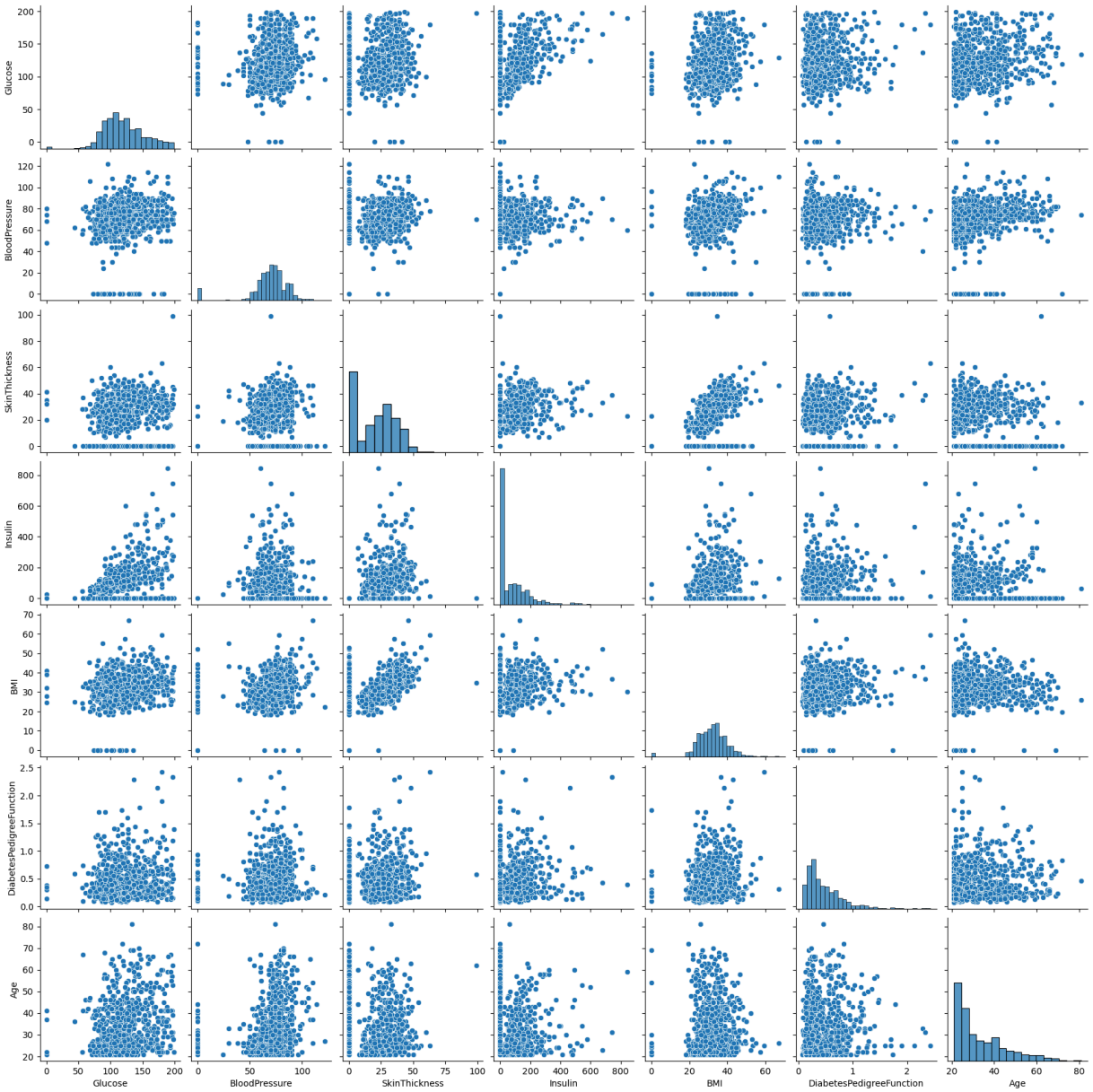

In [10]:
```
df.columns
```

Out[10]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

### 3. PairPlot

In [11]:
```
sns.pairplot(df.loc[:,['Glucose','BloodPressure','SkinThickness','Insulin','
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x7eb163bc7510>




## 4. Correlation Table

In [12]:
```
Correlation=df.loc[:,['Glucose','BloodPressure','SkinThickness','Insulin','E
Cm=sns.light_palette('Green',as_cmap=True)
Correlation.style.background_gradient(cmap=Cm)
```

Out[12]:

| | Glucose | BloodPressure | SkinThickness | Insulin |
|---|---|---|---|---|
| **Glucose** | 1.000000 | 0.152590 | 0.057328 | 0.331357 |
| **BloodPressure** | 0.152590 | 1.000000 | 0.207371 | 0.088933 |
| **SkinThickness** | 0.057328 | 0.207371 | 1.000000 | 0.436783 |
| **Insulin** | 0.331357 | 0.088933 | 0.436783 | 1.000000 |
| **BMI** | 0.221071 | 0.281805 | 0.392573 | 0.197859 |
| **DiabetesPedigreeFunction** | 0.137337 | 0.041265 | 0.183928 | 0.185071 |
| **Age** | 0.263514 | 0.239528 | -0.113970 | -0.042163 |

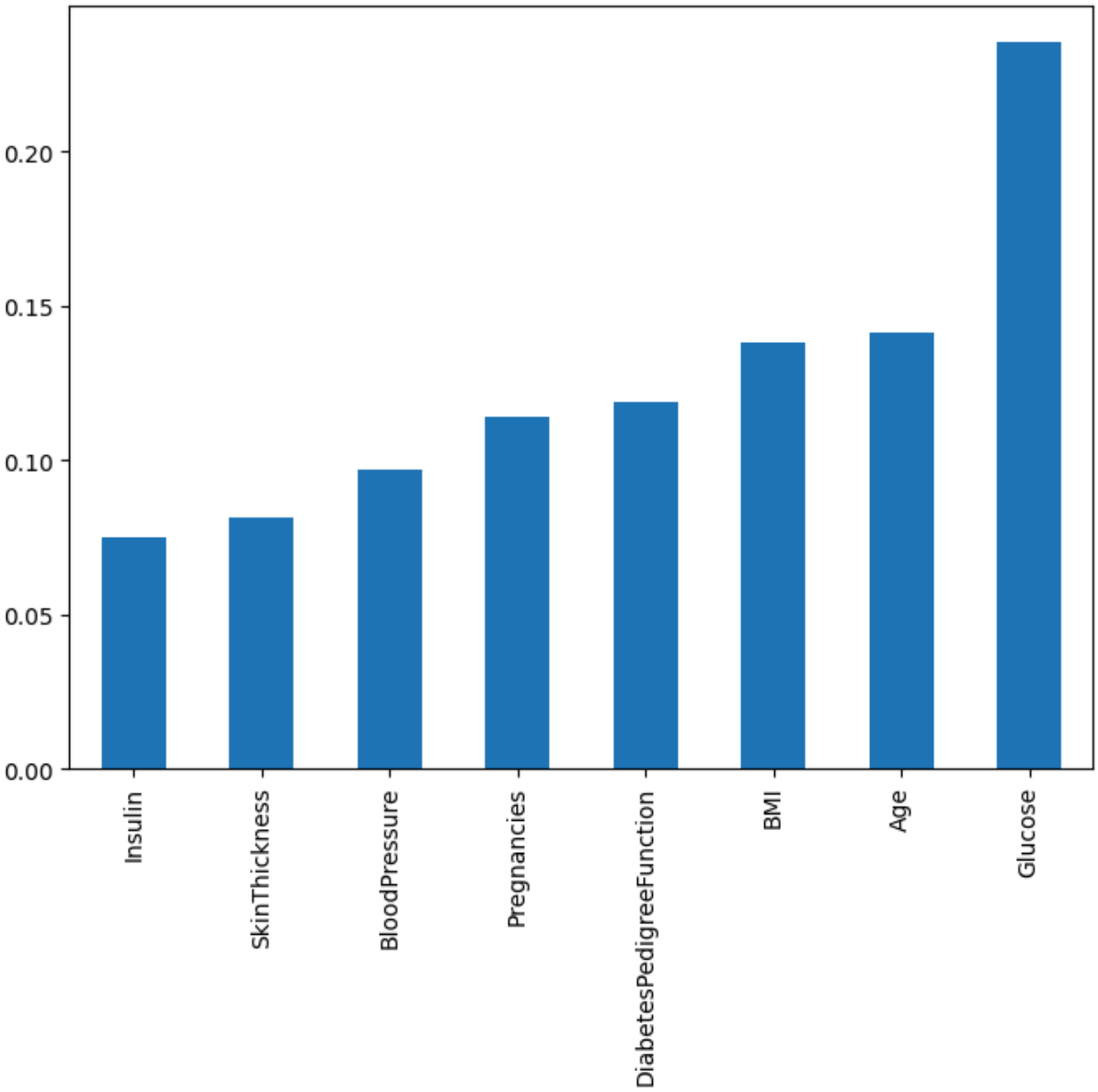
# Feature importence

This analysis aims to identify which features have the most predictive power or strongest relationship with the target variable in a model.

In [13]:
```
X = df.loc[:,df.columns!='Outcome']
Y = df.Outcome
```

In [14]:
```
from sklearn.ensemble import ExtraTreesClassifier
ETC=ExtraTreesClassifier()
ETC.fit(X,Y)
df_new=pd.Series(ETC.feature_importances_,index=X.columns)
plt.figure(figsize=(8,6))
df_new.sort_values().plot(kind='bar')
```

Out[14]: <Axes: >




**Cleaning the most importent features[Glucose, BMI, DiabetesPedigreeFunction] using IQR analysis**

In [15]:
```
Q1=df['Glucose'].quantile(.25)
Q3=df['Glucose'].quantile(.75)
IQR=Q3-Q1
lower_b=Q1-1.5*IQR
upper_b=Q3+1.5*IQR
Df=df[df['Glucose'].between(lower_b,upper_b,inclusive='neither')]
Df
```

Out[15]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabe |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | |

763 rows × 9 columns

In [16]:

```
Q1=Df['BMI'].quantile(0.25)
Q3=Df['BMI'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*Q3
Df=Df[Df['BMI'].between(lower,upper,inclusive='neither')]
Df
```

Out[16]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabe |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | |

752 rows × 9 columns

In [17]:
```
Q1 = Df['DiabetesPedigreeFunction'].quantile(0.25)
Q3 = Df['DiabetesPedigreeFunction'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
Df = Df[Df['DiabetesPedigreeFunction'].between(lower,upper,inclusive='neithe
Df
```

Out[17]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabe |
|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | |
| **5** | 5 | 116 | 74 | 0 | 0 | 25.6 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | |

724 rows × 9 columns

## Splitting the dataset into Dependent and Independent variable.

In [18]:
```
X=Df.loc[:,Df.columns!='Outcome']
y=Df['Outcome']
```

In [19]:
```
print(X.shape)
print(y.shape)
```

```
(724, 8)
(724,)
```

In [20]:
```
Df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 724 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               724 non-null    int64  
 1   Glucose                   724 non-null    int64  
 2   BloodPressure             724 non-null    int64  
 3   SkinThickness             724 non-null    int64  
 4   Insulin                   724 non-null    int64  
 5   BMI                       724 non-null    float64
 6   DiabetesPedigreeFunction  724 non-null    float64
 7   Age                       724 non-null    int64  
 8   Outcome                   724 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 56.6 KB
```

***Here one feature named 'Pregnancies' have int64 format which is irrelevant, hence it is converted into 'object' type.***

In [21]:
```
Df.Pregnancies = Df.Pregnancies.astype('object')
Df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 724 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               724 non-null    object 
 1   Glucose                   724 non-null    int64  
 2   BloodPressure             724 non-null    int64  
 3   SkinThickness             724 non-null    int64  
 4   Insulin                   724 non-null    int64  
 5   BMI                       724 non-null    float64
 6   DiabetesPedigreeFunction  724 non-null    float64
 7   Age                       724 non-null    int64  
 8   Outcome                   724 non-null    int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 56.6+ KB
<ipython-input-21-65439048d904>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Df.Pregnancies = Df.Pregnancies.astype('object')
```

**Importing libraries for model building**

In [22]:
```
#pip install optree
```

In [23]:
```
from sklearn.preprocessing import StandardScaler,LabelEncoder,OrdinalEncoder
from sklearn.model_selection import train_test_split,RandomizedSearchCV, Gri
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier, Ba
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report,confusion_m
from sklearn.compose import ColumnTransformer,make_column_transformer
from sklearn.pipeline import Pipeline,FeatureUnion
from sklearn.pipeline import make_pipeline
from sklearn import set_config
import tensorflow
from tensorflow.keras.layers import Input,Dense,Dropout
from tensorflow.keras.models import Sequential
```

In [24]:
```
X.Pregnancies = X.Pregnancies.astype('object')
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 724 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               724 non-null    object 
 1   Glucose                   724 non-null    int64  
 2   BloodPressure             724 non-null    int64  
 3   SkinThickness             724 non-null    int64  
 4   Insulin                   724 non-null    int64  
 5   BMI                       724 non-null    float64
 6   DiabetesPedigreeFunction  724 non-null    float64
 7   Age                       724 non-null    int64  
dtypes: float64(2), int64(5), object(1)
memory usage: 50.9+ KB
```

```
<ipython-input-24-faddb72c3ae5>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.Pregnancies = X.Pregnancies.astype('object')
```

# Building a Pipeline for custom data handling.

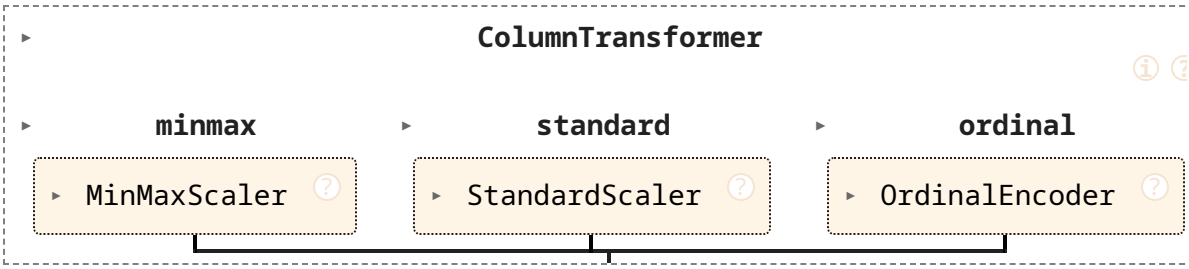
1. Numeric features are handled using Min-Max-Scaling and Standard-Scaling.
2. Categorical feature is handled by OrdinalEncoder.

In [25]:
```
numeric_features_mm = ['DiabetesPedigreeFunction', 'Age']
numeric_features_std = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
categorical_features = ['Pregnancies']
```

In [26]:
```
CT = ColumnTransformer(transformers=[
    ('minmax', MinMaxScaler(), numeric_features_mm),
    ('standard', StandardScaler(), numeric_features_std),
    ('ordinal', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_v
```

```
], remainder='drop')
CT
```

Out[26]:




# Splitting the data into Training and Testing sets for model traing and evaluation.

In [27]:
```
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=.20,random_state=
print(Xtrain.shape)
print(Xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(579, 8)
(145, 8)
(579,)
(145,)
```

In [28]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 724 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               724 non-null    object 
 1   Glucose                   724 non-null    int64  
 2   BloodPressure             724 non-null    int64  
 3   SkinThickness             724 non-null    int64  
 4   Insulin                   724 non-null    int64  
 5   BMI                       724 non-null    float64
 6   DiabetesPedigreeFunction  724 non-null    float64
 7   Age                       724 non-null    int64  
dtypes: float64(2), int64(5), object(1)
memory usage: 50.9+ KB
```

In [29]:
```
y.info()
```

```
<class 'pandas.core.series.Series'>
Index: 724 entries, 0 to 767
Series name: Outcome
Non-Null Count  Dtype
--------------  -----
724 non-null    int64
dtypes: int64(1)
memory usage: 11.3 KB
```

**Hyperparameter ranges.**

In [30]:
```
param_rs_gs_dt = {'model__criterion':['entropy','gini'],
            'model__max_depth': range(4,12),
            'model__splitter': ['best','random']}
param_rs_gs_knn = { 'model__n_neighbors': [i for i in range(3,15,2)],
                   'model__weights': ['uniform','distance'],
                   'model__metric': ['euclidean','manhattan','minkowski']}
```

**Setting different models to the predefined Pipeline to for training.**

In [31]:
```
models= {'Logistic Regression':Pipeline([('transformer',CT),('model',Logisti
         'Decision Tree Classifier':Pipeline([('transformer',CT),('model',De
         'KNN Classifier':Pipeline([('transformer',CT),('model',KNeighborsCl
         'RandomForest Classifier':Pipeline([('transformer',CT),('model',Ran
         'ExtraTree Classifier':Pipeline([('transformer',CT),('model',ExtraT
         'Naive Bayes':Pipeline([('transformer',CT),('model',GaussianNB())])
         'DecisionTreeClassifier_RSCV':RandomizedSearchCV(estimator=Pipeline
                                                          param_distributions
                                                          cv=7,
                                                          random_state=13,
                                                          n_iter=11),
         'DecisionTreeClassifier_GSDT':GridSearchCV(estimator=Pipeline([('tr
                                                    param_grid=param_rs_gs_dt
                                                    cv=7,),
         'KNNClassifier_RSCV':RandomizedSearchCV(estimator=Pipeline([('trans
                                                 param_distributions=param_rs
                                                 random_state=13,),
         'KNNClassifier_GSCV':GridSearchCV(estimator=Pipeline([('transformer
                                           param_grid=param_rs_gs_knn,
                                           cv =7,),
         'SVM':Pipeline([('transformer',CT),('model',SVC(kernel='poly',degre
         }
models.items()
```

Out[31]:
```
dict_items([('Logistic Regression', Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
                ('model', LogisticRegression())])), ('Decision Tree Classif
ier', Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
                ('model', DecisionTreeClassifier(criterion='entropy'))])),
('KNN Classifier', Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
```

```
                ('model', KNeighborsClassifier(n_neighbors=13))])), ('Rando
mForest Classifier', Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
                ('model',
                 RandomForestClassifier(criterion='entropy',
                                        n_estimators=121))])), ('ExtraTree
Classifier', Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
                ('model',
                 ExtraTreesClassifier(criterion='entropy', n_estimators=12
1,
                                      random_state=13))])), ('Naive Bayes',
Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                  ['DiabetesPedigreeFunctio
n',
                                                   'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
```

```
                                                  ('ordinal',
                                                   OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                  unknown_va
lue=-1),
                                                   ['Pregnancies'])])),
                 ('model', GaussianNB())])), ('DecisionTreeClassifier_RSCV',
RandomizedSearchCV(cv=7,
                   estimator=Pipeline(steps=[('transformer',
                                              ColumnTransformer(transformer
s=[('minmax',

MinMaxScaler(),

['DiabetesPedigreeFunction',

'Age']),

('standard',

StandardScaler(),

['Glucose',

'BloodPressure',

'SkinThickness',

'Insulin',

'BMI']),

('ordinal',

OrdinalEncoder(handle_unknown='use_encoded_value',

unknown_value=-1),

['Pregnancies'])])),
                                             ('model',
                                              DecisionTreeClassifier())]),
                   n_iter=11,
                   param_distributions={'model__criterion': ['entropy', 'gi
ni'],
                                        'model__max_depth': range(4, 12),
                                        'model__splitter': ['best', 'rando
m']},
                   random_state=13)), ('DecisionTreeClassifier_GSDT', GridS
earchCV(cv=7,
             estimator=Pipeline(steps=[('transformer',
                                        ColumnTransformer(transformers=[('m
inmax',
                                                                         Mi
nMaxScaler(),

['DiabetesPedigreeFunction',
```

```
'Age']),
                                                                ('s
tandard',
                                                                 St
andardScaler(),
['Glucose',
'BloodPressure',
'SkinThickness',
'Insulin',
'BMI']),
                                                                ('o
rdinal',
                                                                 Or
dinalEncoder(handle_unknown='use_encoded_value',
unknown_value=-1),
['Pregnancies'])])),
                                        ('model', DecisionTreeClassifier
())]),
             param_grid={'model__criterion': ['entropy', 'gini'],
                         'model__max_depth': range(4, 12),
                         'model__splitter': ['best', 'random']})), ('KNNCla
ssifier_RSCV', RandomizedSearchCV(estimator=Pipeline(steps=[('transformer',
                                              ColumnTransformer(transformer
s=[('minmax',
MinMaxScaler(),
['DiabetesPedigreeFunction',
'Age']),
('standard',
StandardScaler(),
['Glucose',
'BloodPressure',
'SkinThickness',
'Insulin',
'BMI']),
('ordinal',
OrdinalEncoder(handle_unknown='use_encoded_value',
```

```
unknown_value=-1),

['Pregnancies'])])),
                                            ('model',
                                             KNeighborsClassifier())]),
                   param_distributions={'model__metric': ['euclidean',
                                                          'manhattan',
                                                          'minkowski'],
                                        'model__n_neighbors': [3, 5, 7, 9,
11,
                                                               13],
                                        'model__weights': ['uniform',
                                                           'distance']},
                   random_state=13)), ('KNNClassifier_GSCV', GridSearchCV(c
v=7,
             estimator=Pipeline(steps=[('transformer',
                                        ColumnTransformer(transformers=[('m
inmax',
                                                                         Mi
nMaxScaler(),

['DiabetesPedigreeFunction',

'Age']),
                                                                        ('s
tandard',
                                                                         St
andardScaler(),

['Glucose',

'BloodPressure',

'SkinThickness',

'Insulin',

'BMI']),
                                                                        ('o
rdinal',
                                                                         Or
dinalEncoder(handle_unknown='use_encoded_value',

unknown_value=-1),

['Pregnancies'])])),
                                       ('model', KNeighborsClassifier())]),
             param_grid={'model__metric': ['euclidean', 'manhattan',
                                           'minkowski'],
                         'model__n_neighbors': [3, 5, 7, 9, 11, 13],
                         'model__weights': ['uniform', 'distance']})), ('SV
M', Pipeline(steps=[('transformer',
                     ColumnTransformer(transformers=[('minmax', MinMaxScaler(),
                                                      ['DiabetesPedigreeFunctio
n',
```

```
                                                  'Age']),
                                                 ('standard', StandardScale
r(),
                                                  ['Glucose', 'BloodPressur
e',
                                                   'SkinThickness', 'Insuli
n',
                                                   'BMI']),
                                                 ('ordinal',
                                                  OrdinalEncoder(handle_unk
nown='use_encoded_value',
                                                                 unknown_va
lue=-1),
                                                  ['Pregnancies'])])),
                ('model', SVC(kernel='poly', probability=True))])])
```

# Model training and comparing the performance of these Classification Models by ROC Curve AUC analysis.

In [32]:

```
plt.figure(figsize=(10,9))
for model_name,model in models.items():
    print(f"Fitting model: {model_name}")

    if isinstance(model,(GridSearchCV,RandomizedSearchCV)):
        model.fit(Xtrain,ytrain)
        best_model = model.best_estimator_
        print(f"Best parameters for {model_name}: {model.best_params_}")

    else:
        best_model = model
        best_model.fit(Xtrain,ytrain)

    y_pred = best_model.predict(Xtest)
    y_pred_proba = best_model.predict_proba(Xtest)[:,1]

    fpr,tpr,_ = roc_curve(ytest,y_pred_proba)
    roc_auc = auc(fpr,tpr)
    plt.plot(fpr,tpr,label= f'{model_name} AUC:{roc_auc:.2f}')

    print(f'Model Name:: {model_name}')
    print(f'Accuracy Score of {model_name} is {accuracy_score(ytest,y_pred)}
    print(classification_report(ytest,y_pred))
    print('*****'*27)
    print('*****'*27)
plt.plot([0,1],[0,1],label='Random Guessing')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('<-----False Positive Rate----->',fontsize=12)
plt.ylabel('<-----True Positive Rate----->',fontsize=12)
plt.title('Receiver Operating Characteristic (ROC) Curves',fontsize=16,color
plt.legend(loc='best')
```

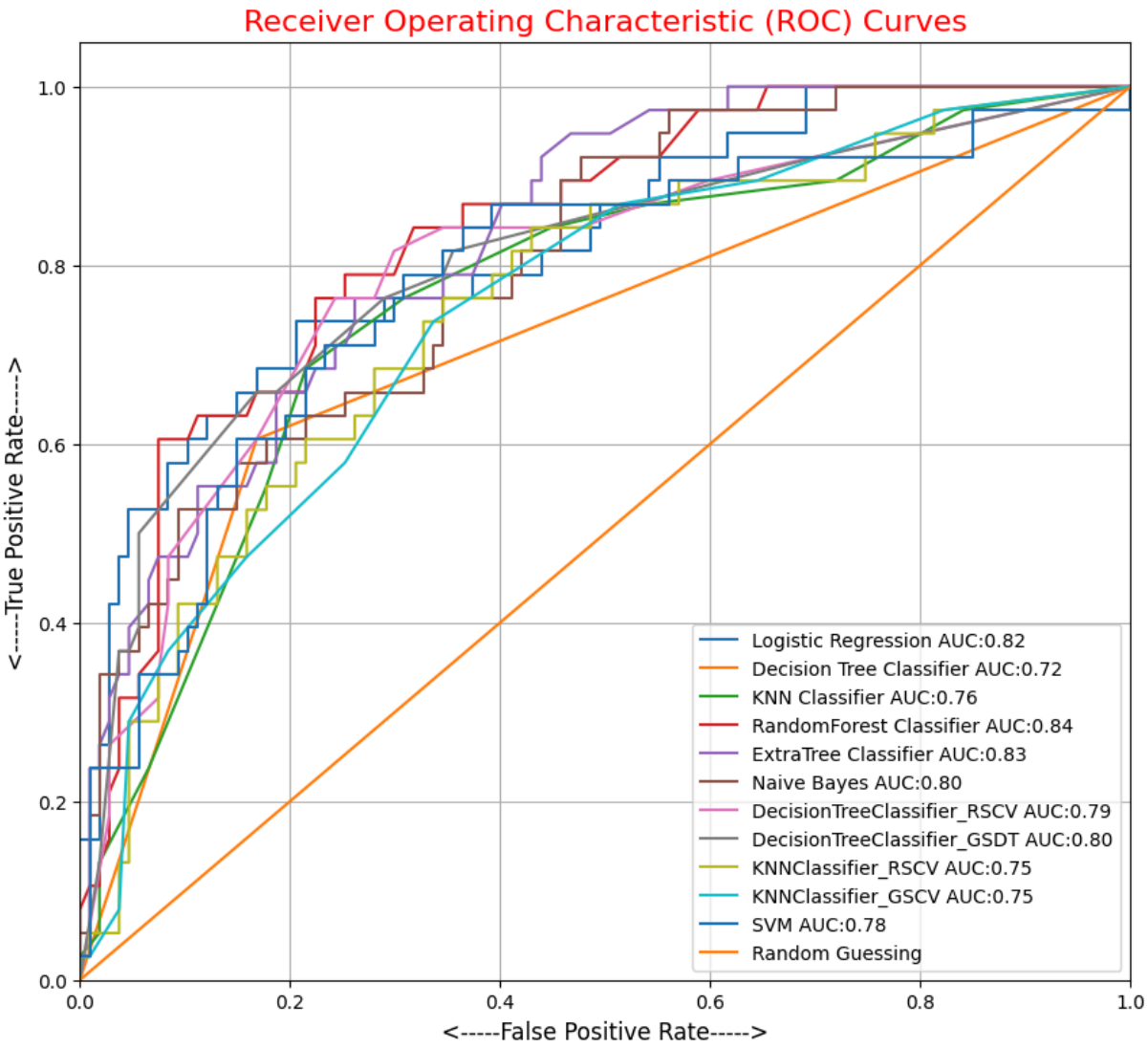
```
plt.grid(True)
plt.show()
```

```
Fitting model: Logistic Regression
Model Name:: Logistic Regression
Accuracy Score of Logistic Regression is 0.7931034482758621
              precision    recall  f1-score   support

           0       0.87      0.84      0.86       107
           1       0.60      0.66      0.62        38

    accuracy                           0.79       145
   macro avg       0.73      0.75      0.74       145
weighted avg       0.80      0.79      0.80       145

**************************************************************************************
************************************************************
**************************************************************************************
************************************************************
Fitting model: Decision Tree Classifier
Model Name:: Decision Tree Classifier
Accuracy Score of Decision Tree Classifier is 0.7724137931034483
              precision    recall  f1-score   support

           0       0.86      0.83      0.84       107
           1       0.56      0.61      0.58        38

    accuracy                           0.77       145
   macro avg       0.71      0.72      0.71       145
weighted avg       0.78      0.77      0.78       145

**************************************************************************************
************************************************************
**************************************************************************************
************************************************************
Fitting model: KNN Classifier
Model Name:: KNN Classifier
Accuracy Score of KNN Classifier is 0.7517241379310344
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       107
           1       0.53      0.55      0.54        38

    accuracy                           0.75       145
   macro avg       0.68      0.69      0.68       145
weighted avg       0.76      0.75      0.75       145

**************************************************************************************
************************************************************
**************************************************************************************
************************************************************
Fitting model: RandomForest Classifier
Model Name:: RandomForest Classifier
Accuracy Score of RandomForest Classifier is 0.7862068965517242
              precision    recall  f1-score   support

           0       0.87      0.83      0.85       107
           1       0.58      0.66      0.62        38
```

```
    accuracy                           0.79       145
   macro avg       0.73      0.74      0.73       145
weighted avg       0.80      0.79      0.79       145

********************************************************************************
************************************************************
********************************************************************************
************************************************************
Fitting model: ExtraTree Classifier
Model Name:: ExtraTree Classifier
Accuracy Score of ExtraTree Classifier is 0.7586206896551724
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       107
           1       0.53      0.61      0.57        38

    accuracy                           0.76       145
   macro avg       0.69      0.71      0.70       145
weighted avg       0.77      0.76      0.76       145

********************************************************************************
************************************************************
********************************************************************************
************************************************************
Fitting model: Naive Bayes
Model Name:: Naive Bayes
Accuracy Score of Naive Bayes is 0.7172413793103448
              precision    recall  f1-score   support

           0       0.85      0.75      0.80       107
           1       0.47      0.63      0.54        38

    accuracy                           0.72       145
   macro avg       0.66      0.69      0.67       145
weighted avg       0.75      0.72      0.73       145

********************************************************************************
************************************************************
********************************************************************************
************************************************************
Fitting model: DecisionTreeClassifier_RSCV
Best parameters for DecisionTreeClassifier_RSCV: {'model__splitter': 'rando
m', 'model__max_depth': 7, 'model__criterion': 'entropy'}
Model Name:: DecisionTreeClassifier_RSCV
Accuracy Score of DecisionTreeClassifier_RSCV is 0.7448275862068966
              precision    recall  f1-score   support

           0       0.90      0.74      0.81       107
           1       0.51      0.76      0.61        38

    accuracy                           0.74       145
   macro avg       0.70      0.75      0.71       145
weighted avg       0.80      0.74      0.76       145

********************************************************************************
************************************************************
```

```
**************************************************************************
************************************************************
Fitting model: DecisionTreeClassifier_GSDT
Best parameters for DecisionTreeClassifier_GSDT: {'model__criterion': 'gin
i', 'model__max_depth': 5, 'model__splitter': 'best'}
Model Name:: DecisionTreeClassifier_GSDT
Accuracy Score of DecisionTreeClassifier_GSDT is 0.7862068965517242
              precision    recall  f1-score   support

           0       0.87      0.83      0.85       107
           1       0.58      0.66      0.62        38

    accuracy                           0.79       145
   macro avg       0.73      0.74      0.73       145
weighted avg       0.80      0.79      0.79       145

**************************************************************************
************************************************************
**************************************************************************
************************************************************
Fitting model: KNNClassifier_RSCV
Best parameters for KNNClassifier_RSCV: {'model__weights': 'distance', 'mode
l__n_neighbors': 11, 'model__metric': 'manhattan'}
Model Name:: KNNClassifier_RSCV
Accuracy Score of KNNClassifier_RSCV is 0.7517241379310344
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       107
           1       0.53      0.47      0.50        38

    accuracy                           0.75       145
   macro avg       0.67      0.66      0.67       145
weighted avg       0.74      0.75      0.75       145

**************************************************************************
************************************************************
**************************************************************************
************************************************************
Fitting model: KNNClassifier_GSCV
Best parameters for KNNClassifier_GSCV: {'model__metric': 'manhattan', 'mode
l__n_neighbors': 11, 'model__weights': 'uniform'}
Model Name:: KNNClassifier_GSCV
Accuracy Score of KNNClassifier_GSCV is 0.7448275862068966
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       107
           1       0.51      0.47      0.49        38

    accuracy                           0.74       145
   macro avg       0.67      0.66      0.66       145
weighted avg       0.74      0.74      0.74       145

**************************************************************************
************************************************************
**************************************************************************
************************************************************
```

```
Fitting model: SVM
Model Name:: SVM
Accuracy Score of SVM is 0.7655172413793103
              precision    recall  f1-score   support

           0       0.78      0.95      0.86       107
           1       0.64      0.24      0.35        38

    accuracy                           0.77       145
   macro avg       0.71      0.60      0.60       145
weighted avg       0.74      0.77      0.72       145

********************************************************************************
************************************************************
********************************************************************************
************************************************************
```




***RandomForest Classifier performed the best with the AUC score of 0.84***

# Using Neural Network

In [33]:
```
transformed = CT.fit_transform(Df)
X= transformed
y=Df.Outcome
Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=.21,random_state=42
print(Xtrain.shape)
print(Xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(571, 8)
(153, 8)
(571,)
(153,)
```

In [34]:
```
import tensorflow as tf
```

# ANN model building

In [35]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(8,)),
    tf.keras.layers.Dense(50,activation='relu',kernel_regularizer=tf.keras.r
    tf.keras.layers.Dropout(.13),
    tf.keras.layers.Dense(100,activation='relu',kernel_regularizer=tf.keras.
    tf.keras.layers.Dropout(.13),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Par |
|---|---|---|
| dense (Dense) | (None, 50) | |
| dropout (Dropout) | (None, 50) | |
| dense_1 (Dense) | (None, 100) | 5 |
| dropout_1 (Dropout) | (None, 100) | |
| dense_2 (Dense) | (None, 1) | |

**Total params:** 5,651 (22.07 KB)

**Trainable params:** 5,651 (22.07 KB)

**Non-trainable params:** 0 (0.00 B)

# ANN model training

In [36]:
```
model.compile(optimizer='Adam',
              loss='binary_crossentropy',
              metrics=['accuracy'],)
M = model.fit(Xtrain,ytrain, validation_data=(Xtest,ytest), epochs=20,batch_
```

```
Epoch 1/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.5517 - loss: 0.7681 -
val_accuracy: 0.7255 - val_loss: 0.6532
Epoch 2/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6578 - loss: 0.6971 - v
al_accuracy: 0.7451 - val_loss: 0.5891
Epoch 3/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7209 - loss: 0.6237 - v
al_accuracy: 0.8039 - val_loss: 0.5587
Epoch 4/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7353 - loss: 0.5965 - v
al_accuracy: 0.7908 - val_loss: 0.5386
Epoch 5/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7431 - loss: 0.5794 - v
al_accuracy: 0.7974 - val_loss: 0.5285
Epoch 6/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7420 - loss: 0.5513 - v
al_accuracy: 0.7516 - val_loss: 0.5428
Epoch 7/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7251 - loss: 0.5665 - v
al_accuracy: 0.7908 - val_loss: 0.5226
Epoch 8/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7529 - loss: 0.5596 - v
al_accuracy: 0.7843 - val_loss: 0.5194
Epoch 9/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8079 - loss: 0.4852 - v
al_accuracy: 0.7908 - val_loss: 0.5274
Epoch 10/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7971 - loss: 0.4935 - v
al_accuracy: 0.7712 - val_loss: 0.5300
Epoch 11/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7737 - loss: 0.4933 - v
al_accuracy: 0.7908 - val_loss: 0.5175
Epoch 12/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7660 - loss: 0.5136 - v
al_accuracy: 0.7386 - val_loss: 0.5482
Epoch 13/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7482 - loss: 0.5455 - v
al_accuracy: 0.7843 - val_loss: 0.5126
Epoch 14/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7526 - loss: 0.5153 - v
al_accuracy: 0.7843 - val_loss: 0.5168
Epoch 15/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7572 - loss: 0.5074 - v
al_accuracy: 0.8105 - val_loss: 0.5027
Epoch 16/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7641 - loss: 0.5078 - v
al_accuracy: 0.7974 - val_loss: 0.5046
Epoch 17/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7544 - loss: 0.5278 - v
al_accuracy: 0.7582 - val_loss: 0.5235
Epoch 18/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8141 - loss: 0.4646 - v
al_accuracy: 0.7908 - val_loss: 0.5107
Epoch 19/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7492 - loss: 0.5387 - v
```

```
al_accuracy: 0.7908 - val_loss: 0.5048
Epoch 20/20
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7528 - loss: 0.5129 - v
al_accuracy: 0.7778 - val_loss: 0.5108
```

In [37]:
```
print("train score", model.evaluate(Xtrain,ytrain))
print("test score", model.evaluate(Xtest, ytest))
```

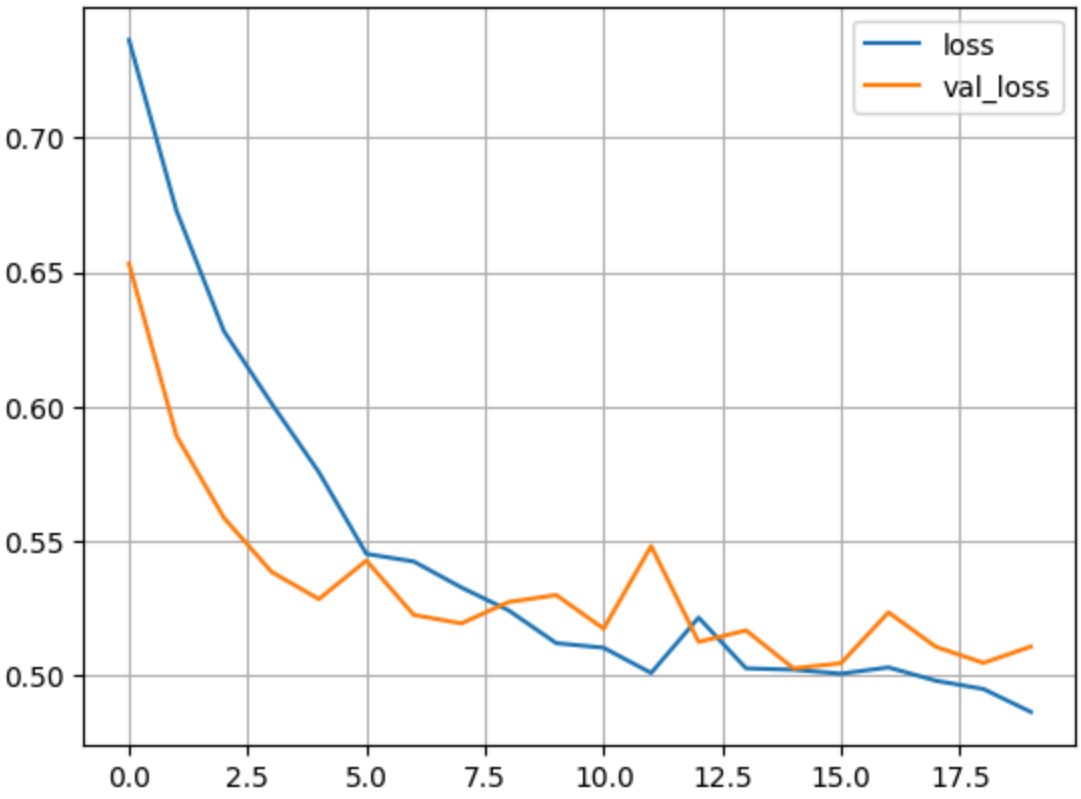
```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7793 - loss: 0.4810
train score [0.46941620111465454, 0.7845884561538696]
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7415 - loss: 0.5697
test score [0.5108239650726318, 0.7777777910232544]
```

# Neural Network model performance

1. Loss Curve. Training vs Validation/testing Loss.
2. Accuracy Curve. Training vs Validation/training Accuracy.
3. ROC curve.

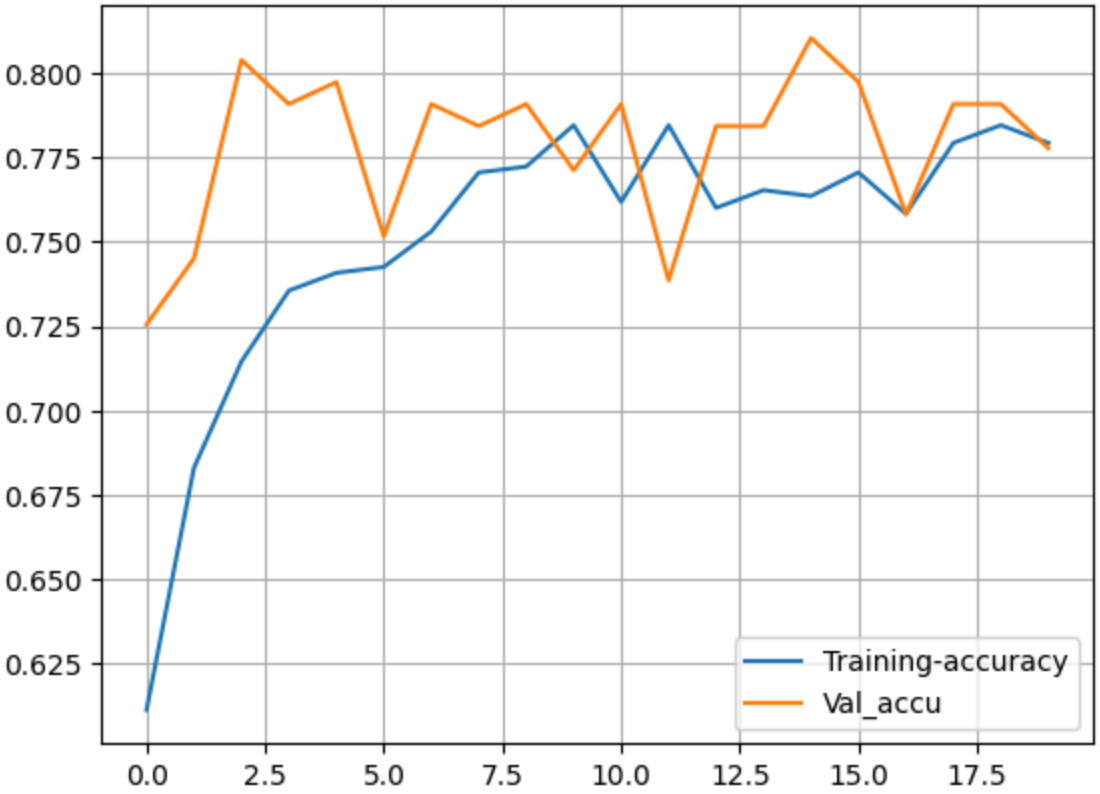
### 1. Loss Curve

In [38]:
```
plt.plot(M.history['loss'], label='loss')
plt.plot(M.history['val_loss'], label='val_loss')
plt.legend()
plt.grid(True)
plt.show()
```




### 2. Accuracy Curve

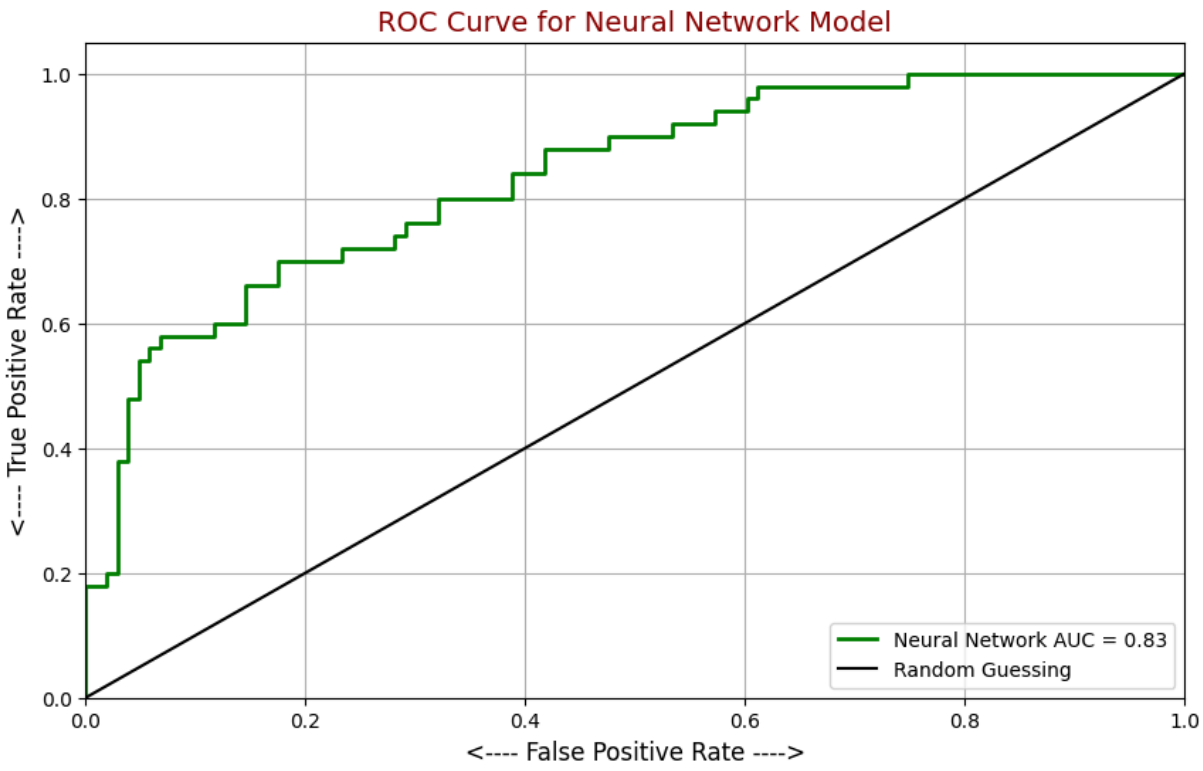
In [39]:
```
plt.plot(M.history['accuracy'], label='Training-accuracy')
plt.plot(M.history['val_accuracy'], label='Val_accu')
plt.legend()
plt.grid(True)
plt.show()
```




### 3. ROC Curve

In [40]:
```
y_pred_proba = model.predict(Xtest).ravel()

fpr, tpr, thresholds = roc_curve(ytest, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plotting
plt.figure(figsize=(10,6))
plt.plot(fpr, tpr, color='green', lw=2, label=f'Neural Network AUC = {roc_au
plt.plot([0, 1], [0, 1], color='black', linestyle='-', label='Random Guessir
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('<---- False Positive Rate ---->', fontsize=12)
plt.ylabel('<---- True Positive Rate ---->', fontsize=12)
plt.title('ROC Curve for Neural Network Model', fontsize=14, color='darkred'
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

**5/5** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step




## Prediction using ANN model.

In [41]:
```
new_data = pd.DataFrame([{
    'Pregnancies': '2',
    'Glucose': 120,
    'BloodPressure': 70,
    'SkinThickness': 20,
    'Insulin': 85,
    'BMI': 28.0,
    'DiabetesPedigreeFunction': 0.45,
    'Age': 30
}])

# Transforming the input using the already-fitted ColumnTransformer
transformed_input = CT.transform(new_data)

# Predicting the probability and class
predicted_proba = model.predict(transformed_input)[0][0]
predicted_class = int(predicted_proba >= 0.5)

print(f"Predicted Probability of Diabetes: {predicted_proba:.2f}")
print(f"Predicted Class (0 = No Diabetes, 1 = Diabetes): {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Predicted Probability of Diabetes: 0.05
Predicted Class (0 = No Diabetes, 1 = Diabetes): 0
```

## Conclusion

The accuracy of ANN model is 0.79 with ROC score of 0.84 same as RandomForest Classifier. These two are the best performing model in this dataset. Neural Network models often outperform traditional machine learning models when trained on large volumes of data. However, in our case, where the dataset is limited, the ANN model still achieves approximately the same efficiency as the RandomForest model in predicting the target class.