The following steps will show the prediction of diabetes in a person on the basis of different features like insulin, bmi,etc

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('diabetes.csv')
```

In [3]:
```
df
```

Out[3]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

observation:- from we can see that, the values of features are in different ranges. So we have to standardize them after train test split

In [4]:
```
# check if standardization is required or not
df.std()
```

Out[4]:

	0
Pregnancies	3.369578
Glucose	31.972618
BloodPressure	19.355807
SkinThickness	15.952218
Insulin	115.244002
BMI	7.884160
DiabetesPedigreeFunction	0.331329
Age	11.760232
Outcome	0.476951

dtype: float64

from above it is clear that standardization is required

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
```

In [6]:
```python
#check for null values
df.isnull().sum()
```

Out[6]:

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

no null values

In [7]:
```python
df['Age'].groupby(df['Outcome']).value_counts()
```

Out[7]:

		count
Outcome	**Age**	
0	**22**	61
	21	58
	24	38
	25	34
	23	31
...	**...**	...
1	**67**	1
	61	1
	55	1
	48	1
	70	1

96 rows × 1 columns

dtype: int64

In [8]:
```python
df.groupby('Outcome')['Age'].describe()
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Outcome								
0	500.0	31.190000	11.667655	21.0	23.0	27.0	37.0	81.0
1	268.0	37.067164	10.968254	21.0	28.0	36.0	44.0	70.0

In [9]:
```python
df.groupby('Outcome').mean()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

this shows the mean of each feature regarding diabetic and none diabetic

In [10]:
```python
df.duplicated().sum()
```

Out[10]:
```
0
```

In [11]:
```python
df['Outcome'].value_counts()
```

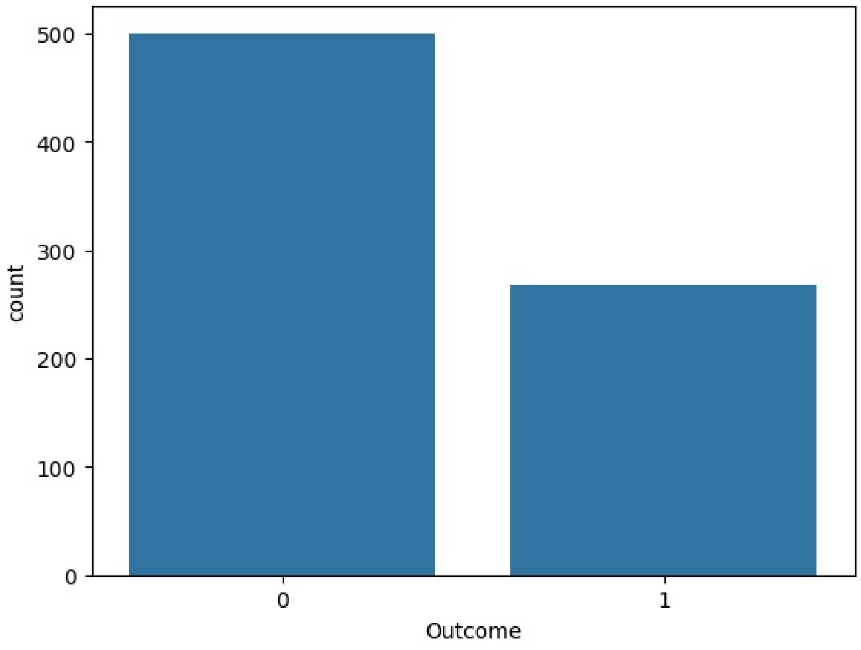
Out[11]:

	count
Outcome	
0	500
1	268

dtype: int64

In [12]:
```
sns.countplot(x='Outcome',data=df)
```

Out[12]:
```
<Axes: xlabel='Outcome', ylabel='count'>
```



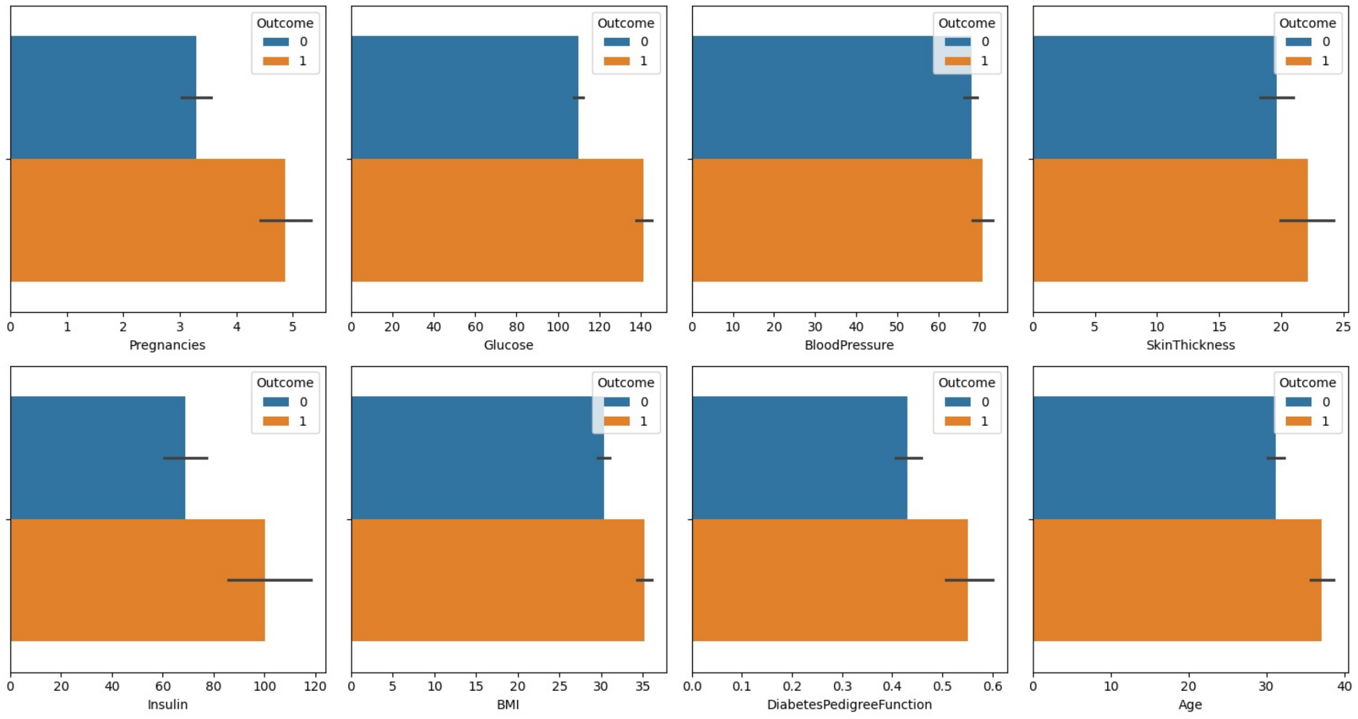
1--> Patient is Diabetic

0--> None Diabetic

to show the distribution of each feature with target feature(outcome)

In [13]:
```
#create subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 8))

#plot distribution by iterating through the feature col
feature_col = df.columns.drop('Outcome')
for index, feature in enumerate(feature_col):
    row = index // 4  # Calculate the row index
    col = index % 4   # Calculate the column index
    sns.barplot(data=df, x=feature, hue='Outcome', ax=axes[row, col])  # Plot on the correct subplot
    plt.tight_layout()
plt.show()
```



now doing standardization

```
In [14]: from sklearn.preprocessing import StandardScaler
```

```
In [15]: scaler=StandardScaler()
```

```
In [16]: x=df.drop('Outcome',axis=1)
```

```
In [17]: y=df['Outcome']
```

```
In [18]: # train test split
         from sklearn.model_selection import train_test_split
```

```
In [19]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state=42)
```

```
In [20]: # scaling train and test data
         x_train_scaled=scaler.fit_transform(x_train)
         x_test_scaled=scaler.transform(x_test)
```

as the label is binary, we will be using classification models

And evaluating them to see which one works better in comparison to other for diabetics prediction

Used Classification models:-

1. Logistic Regression
2. SVM
3. Decision Tree
4. Random Forest
5. Naive Bayes

Logistic Regression

```
In [21]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
```

```
In [22]: linear_model=LogisticRegression()
```

```
In [23]: linear_model.fit(x_train_scaled,y_train)
```

```
Out[23]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [24]: y_predict=linear_model.predict(x_test_scaled)
In [25]: log_accuracy=round(accuracy_score(y_test,y_predict),3)
         log_accuracy
```

```
Out[25]: 0.714
```

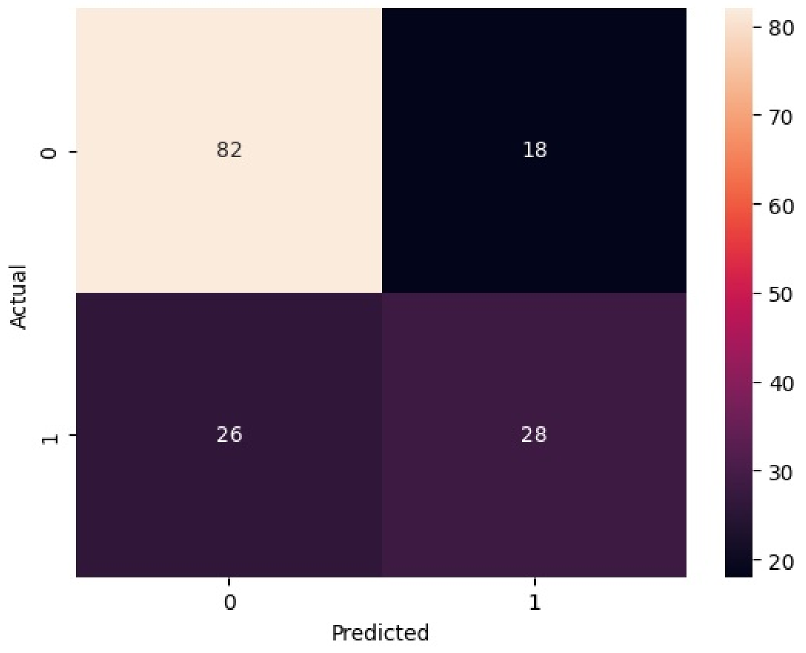
```
In [27]: log_matrix=confusion_matrix(y_test,y_predict)
         log_matrix
```

```
Out[27]: array([[82, 18],
                [26, 28]])
```

```
In [28]: log_precision=precision_score(y_test,y_predict)
         log_recall=recall_score(y_test,y_predict)
         log_f1=f1_score(y_test,y_predict)
```

```
In [29]: #visualize the confusion matrix
         sns.heatmap(log_matrix,annot=True)
         plt.ylabel('Actual')
         plt.xlabel('Predicted')
```

```
Out[29]: Text(0.5, 23.52222222222222, 'Predicted')
```



Support Vector Machine

```
In [30]: from sklearn.svm import SVC
```

```
In [32]: machine=SVC(kernel='linear')
```

```
In [33]: machine.fit(x_train_scaled,y_train)
```

```
Out[33]: ▾        SVC
         SVC(kernel='linear')
```

```
In [34]: y_predict=machine.predict(x_test_scaled)
         machine_accuracy=round(accuracy_score(y_test,y_predict),3)
```

```
In [35]: machine_matrix=confusion_matrix(y_test,y_predict)
         machine_precision=precision_score(y_test,y_predict)
         machine_recall=recall_score(y_test,y_predict)
         machine_f1=f1_score(y_test,y_predict)
         print(machine_matrix)
```

```
[[83 17]
 [26 28]]
```

Decision Tree

```
In [36]: from sklearn.tree import DecisionTreeClassifier
```

```
In [37]: decision_model=DecisionTreeClassifier()
```

```
In [38]: decision_model.fit(x_train_scaled,y_train)
```

Out[38]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [39]:
```
y_predict=decision_model.predict(x_test_scaled)
decision_accuracy=round(accuracy_score(y_test,y_predict),3)
```

In [40]:
```
decision_matrix=confusion_matrix(y_test,y_predict)
decision_precision=precision_score(y_test,y_predict)
decision_recall=recall_score(y_test,y_predict)
decision_f1=f1_score(y_test,y_predict)
print(decision_matrix)
```

```
[[87 13]
 [31 23]]
```

Random Forest

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [42]:
```
random_model=RandomForestClassifier()
```

In [43]:
```
random_model.fit(x_train_scaled,y_train)
```

Out[43]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [44]:
```
y_predict=random_model.predict(x_test_scaled)
random_accuracy=round(accuracy_score(y_test,y_predict),3)
```

In [45]:
```
random_matrix=confusion_matrix(y_test,y_predict)
random_precision=precision_score(y_test,y_predict)
random_recall=recall_score(y_test,y_predict)
random_f1=f1_score(y_test,y_predict)
print(random_matrix)
```

```
[[81 19]
 [24 30]]
```

Naive Bayes

In [46]:
```
from sklearn.naive_bayes import BernoulliNB
naive_model=BernoulliNB()
```

In [47]:
```
naive_model.fit(x_train_scaled,y_train)
```

Out[47]:
```
▾ BernoulliNB
BernoulliNB()
```

In [48]:
```
y_predict=naive_model.predict(x_test_scaled)
naive_accuracy=round(accuracy_score(y_test,y_predict),3)
```

In [49]:
```
naive_matrix=confusion_matrix(y_test,y_predict)
naive_precision=precision_score(y_test,y_predict)
naive_recall=recall_score(y_test,y_predict)
naive_f1=f1_score(y_test,y_predict)
print(naive_matrix)
```

```
[[80 20]
 [26 28]]
```

In [50]:
```
#comparision table
data={'model':['logistic reg','svm','decision tree','random forest','naive bayes'],
      'precision':[log_precision,machine_precision,decision_precision,random_precision,naive_precision],
      'recall':[log_recall,machine_recall,decision_recall,random_recall,naive_recall],
      'f1':[log_f1,machine_f1,decision_f1,random_f1,naive_f1],
      'accuracy':[log_accuracy,machine_accuracy,decision_accuracy,random_accuracy,naive_accuracy]}
comparison_df=pd.DataFrame(data)
```

In [51]:
```
comparison_df
```

Out[51]:

	model	precision	recall	f1	accuracy
0	logistic reg	0.608696	0.518519	0.560000	0.714
1	svm	0.622222	0.518519	0.565657	0.721
2	decision tree	0.638889	0.425926	0.511111	0.714
3	random forest	0.612245	0.555556	0.582524	0.721
4	naive bayes	0.583333	0.518519	0.549020	0.701

as false negative(missing the diabeties) is more crucial --> recall should be more for less FN(priority)

as false positive(incorrectly diagnosing a disease) is less crucial than FN --> precision should be more for less FP

In [52]:
```
machine_matrix
```

Out[52]:
```
array([[83, 17],
       [26, 28]])
```

In [53]:
```
random_matrix
```

Out[53]:
```
array([[81, 19],
       [24, 30]])
```

Evaluation Metric -->

as F1-score balances Precision and Recall, we would be considering F1-Score for model evaluation

So according to that Random Forest would be best fit for Prediction of diabetes in a patient

In [54]:
```
# define hyperparameter grid
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

In [55]:
```
# initialize gridsearch
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=5,
                           scoring='f1')
```

In [56]:
```
#fit model
grid_search.fit(x_train_scaled, y_train)
```

Out[56]:
```
▸             GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

In [58]:
```
#best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
print(best_params)
print(best_model)
```

```
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
RandomForestClassifier(min_samples_leaf=2, random_state=42)
```

In [59]:
```
# evaluate the best model
y_pred = best_model.predict(x_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7402597402597403
```

after tuning the accuracy of random forest model has increased from 0.72 to 0.74

In [60]:
```
f1_value=f1_score(y_test,y_pred)
f1_value
```

Out[60]:
```
0.6
```

In [61]:
```
# predictive system
```

```python
input_data=(6,104,74,18,156,29.9,0.722,41)

#converting to array
data_array=np.asarray(input_data)

#reshapping
reshaped_data=data_array.reshape(1,-1)

#standardize the data as data would be in raw form and need to be standardized
data=scaler.transform(reshaped_data)

#predicting
prediction=random_model.predict(data)

#checking
if prediction[0]==0:
  print('patient is not diabetic')
else:
  print('patient is diabetic')
```

```
patient is diabetic
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, b
ut StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js