Fauzan Kamil

Link Github : https://github.com/Fauzan-Kamil/predict-diabetes-streamlit (https://github.com/Fauzan-Kamil/predict-diabetes-streamlit)

Dataset

Diabetes : https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset (https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset)

Penjelasan Atribut Dataset

- Pregnancies : Untuk menyatakan Jumlah kehamilan
- Glucose : Tingkat Glukosa dalam darah 2 jam dalam tes toleransi glukosa oral
- BloodPressure : Tekanan darah diastolik (mm Hg)
- SkinThickness : Ketebalan lipatan kulit pada triceps (mm)
- insulin : Tingkat insulin dalam darah
- BMI : Indeks massa tubuh (berat badan dalam kg/(tinggi dalam m)^2)
- DiabetesPedigreeFunction : Presentase keturunan diabetes
- Age : Umur (tahun)
- Outcome : Menyatakan hasil akhir 1 adalah Ya (terkena diabetes) dan 0 adalah Tidak (tidak terkena diabetes)

Import Library

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
import warnings
warnings.filterwarnings('ignore')
```

Data

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

Out[2]:

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

In [54]:
```
# Descriptive statistics
df.describe()
```

Out[54]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [55]:
```
# Info mengenai dataset diabetes
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 [56]:
```
# Cek missing value
df.isnull().sum()
```

Out[56]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [57]:
```
# Jumlah baris dan kolom
df.shape
```

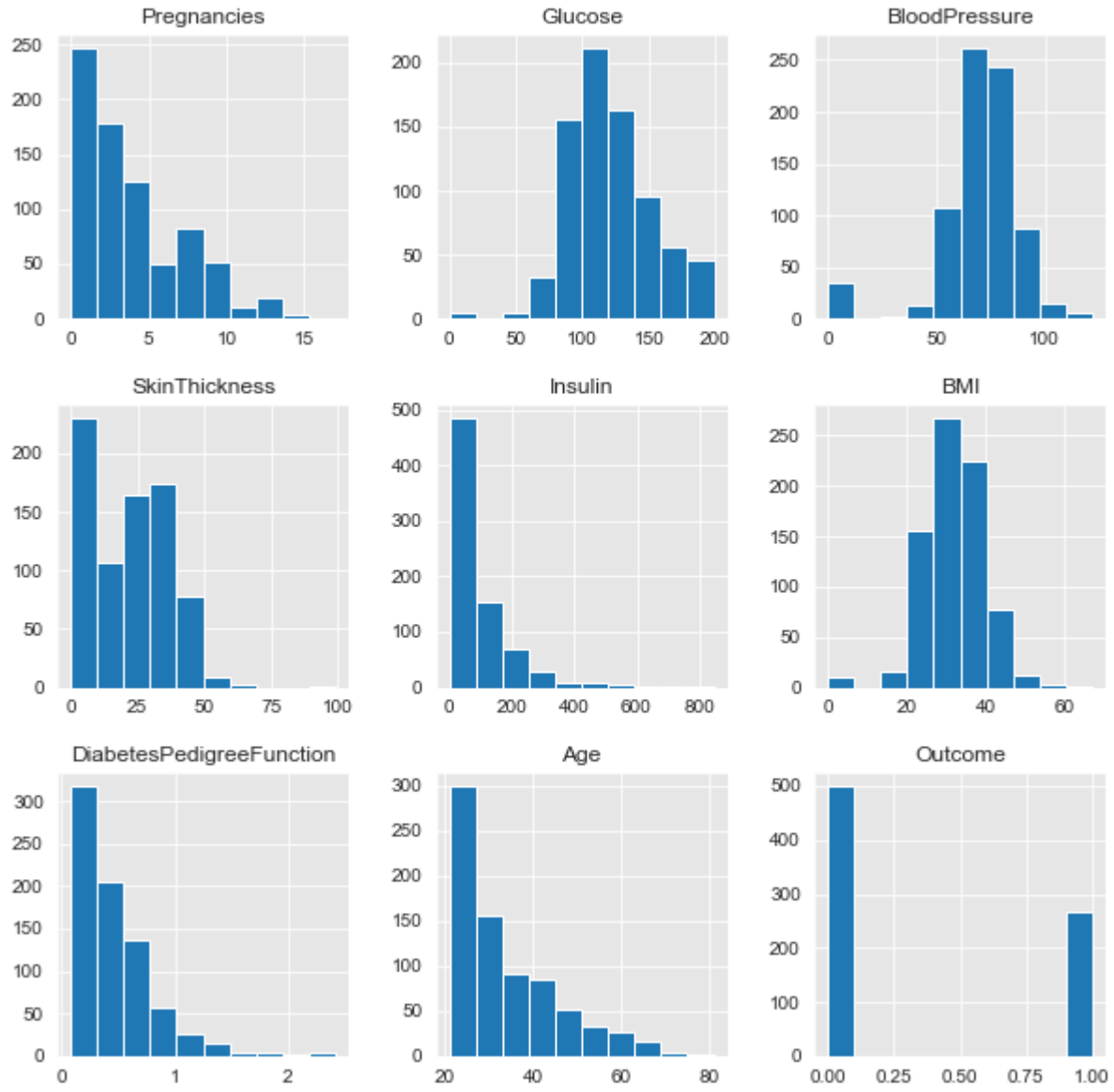
Out[57]:
```
(768, 9)
```

In [58]:
```
# Kolom yang ada di dataset
df.columns
```

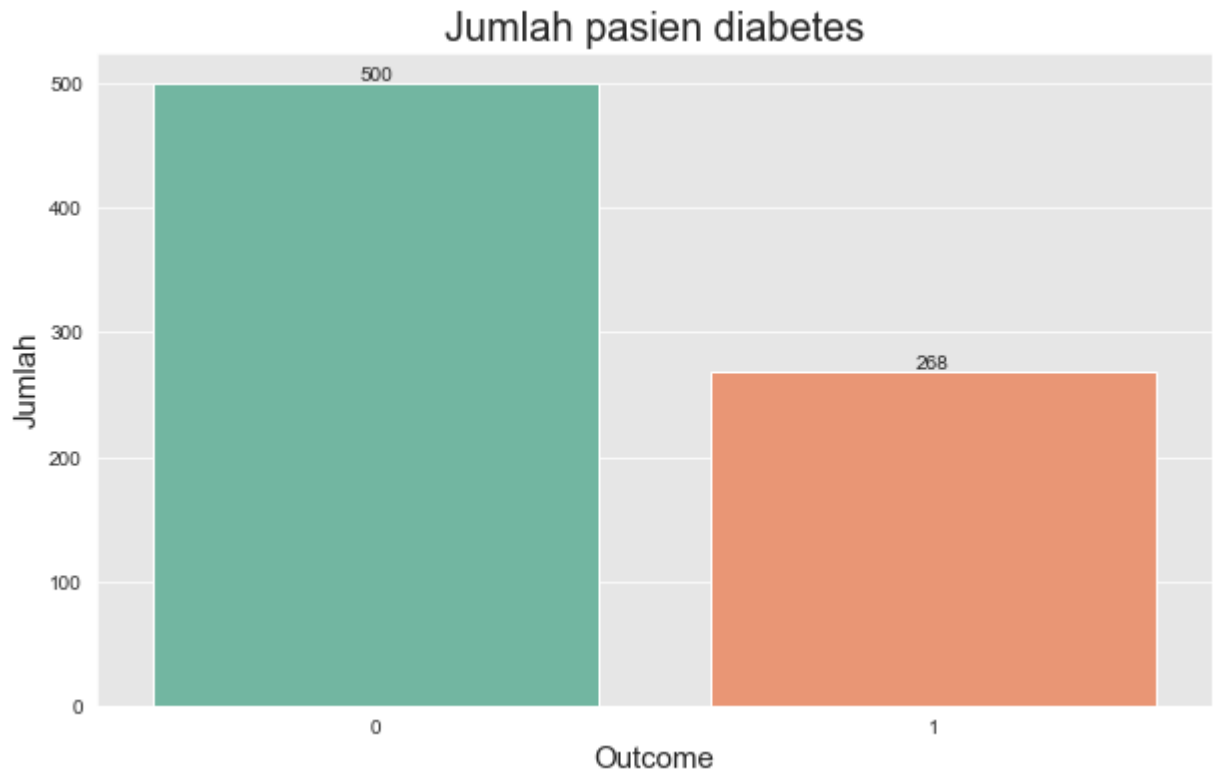
Out[58]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Data Visualization

In [59]:
```
# histogram
df.hist(figsize=(10,10))
plt.show()
```

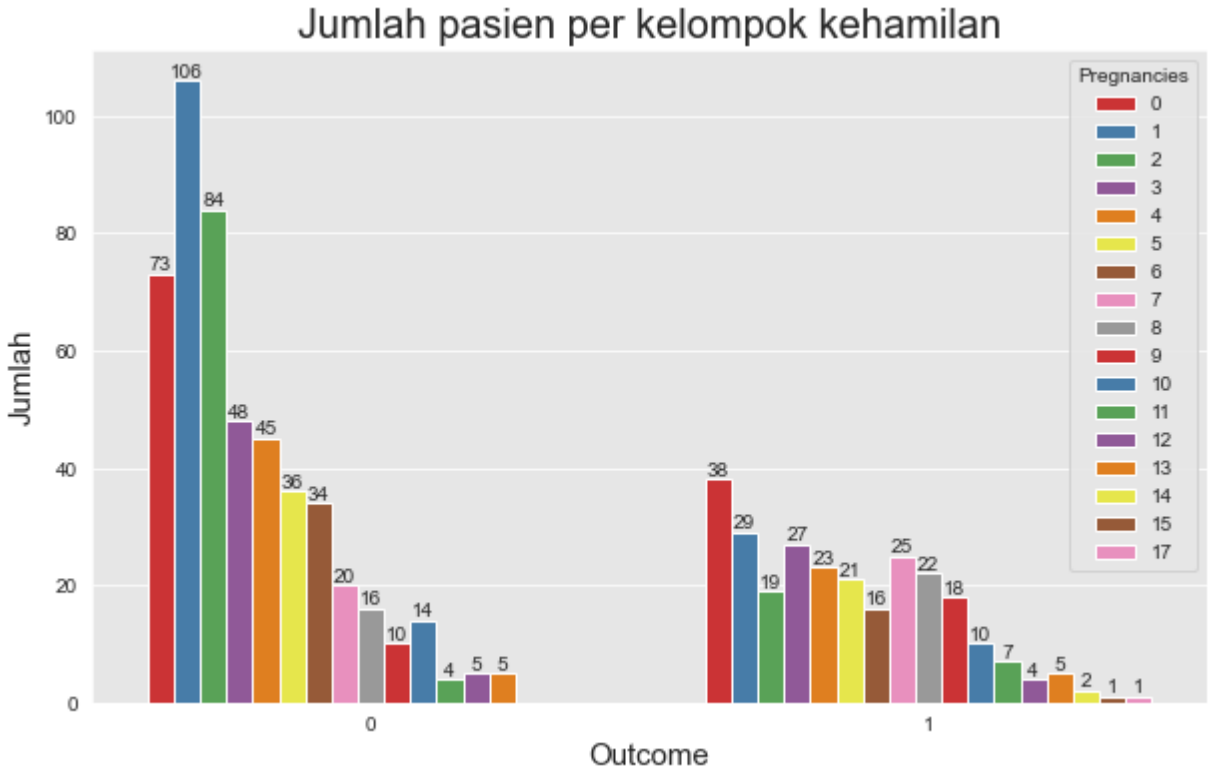


In [60]:
```
plt.figure(figsize=(10,6))
a = sns.countplot(x='Outcome', data=df, palette='Set2')
for j in a.containers:
    a.bar_label(j, label_type='edge')
plt.title('Jumlah pasien diabetes', fontsize=20)
plt.xlabel('Outcome', fontsize=15)
plt.ylabel('Jumlah', fontsize=15)
plt.show()
```



Bisa dilihat dari grafik diatas bahwa banyak orang yang tidak terkena diabetes dan sedikit orang yang terkena diabetes yaitu 268 orang.

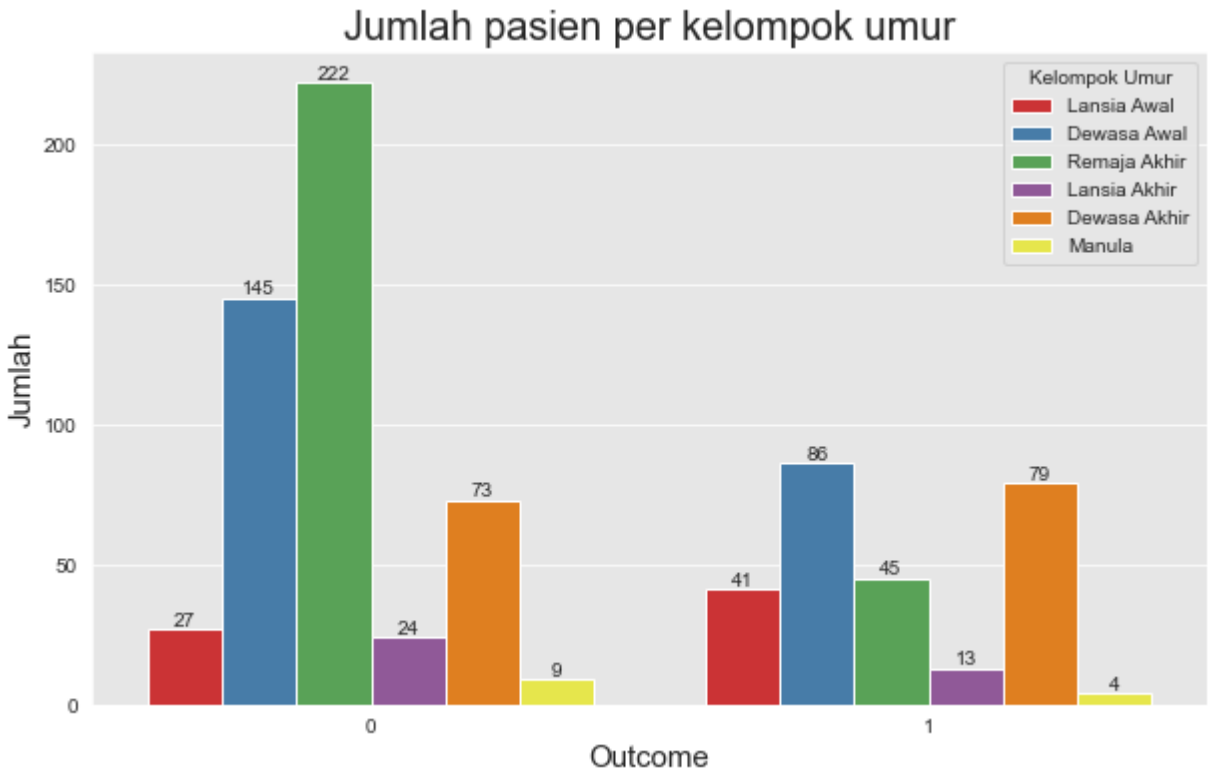
In [61]:
```
plt.figure(figsize=(10,6))
a = sns.countplot(x='Outcome', hue='Pregnancies', data=df, palette='Set1')
for j in a.containers:
    a.bar_label(j, label_type='edge')
plt.title('Jumlah pasien per kelompok kehamilan', fontsize=20)
plt.xlabel('Outcome', fontsize=15)
plt.ylabel('Jumlah', fontsize=15)
plt.show()
```



Jumlah pasien perkelompok kehamilan yang paling banyak terkena diabets adalah 0 kehamilan dengan jumlah 38 orang lalu diikuti dengan 3 kehamilan dengan jumlah 27 orang dan yang paling sedikit adalah 17 kehamilan dengan jumlah 1 orang.

In [62]:
```
age_grup = []
for i in df['Age']:
    if i >= 17 and i <= 25:
        age_grup.append('Remaja Akhir')
    elif i >= 26 and i <= 35:
        age_grup.append('Dewasa Awal')
    elif i >= 36 and i <= 45:
        age_grup.append('Dewasa Akhir')
    elif i >= 46 and i <= 55:
        age_grup.append('Lansia Awal')
    elif i >= 56 and i <= 65:
        age_grup.append('Lansia Akhir')
    else:
        age_grup.append('Manula')
df['Age_grup'] = age_grup
```

In [63]:
```
plt.figure(figsize=(10,6))
a = sns.countplot(x='Outcome', hue='Age_grup', data=df, palette='Set1')
for j in a.containers:
    a.bar_label(j, label_type='edge')
plt.title('Jumlah pasien per kelompok umur', fontsize=20)
plt.xlabel('Outcome', fontsize=15)
plt.ylabel('Jumlah', fontsize=15)
plt.legend(loc='upper right', title='Kelompok Umur')
plt.show()
```



Banyak pasien yang terkena diabetes adalah yang berumur 26-35 tahun atau dewasa awal dengan jumlah 86 orang lalu diikutu dengan dewasa akhir yaitu 46-55 tahun dengan jumlah 79 orang dan yang paling sedikit adalah manula dengan jumlah 4 orang.

In [64]:
```
BMI_grup = []
for i in df['BMI']:
    if i >= 0 and i <= 18.5:
        BMI_grup.append('Kurus')
    elif i >= 18.6 and i <= 22.9:
        BMI_grup.append('Normal')
    elif i >= 23 and i <= 24.9:
        BMI_grup.append('Gemuk')
    elif i >= 25 and i <= 29.9:
        BMI_grup.append('Obesitas')
    else:
        BMI_grup.append('Obesitas II')
```

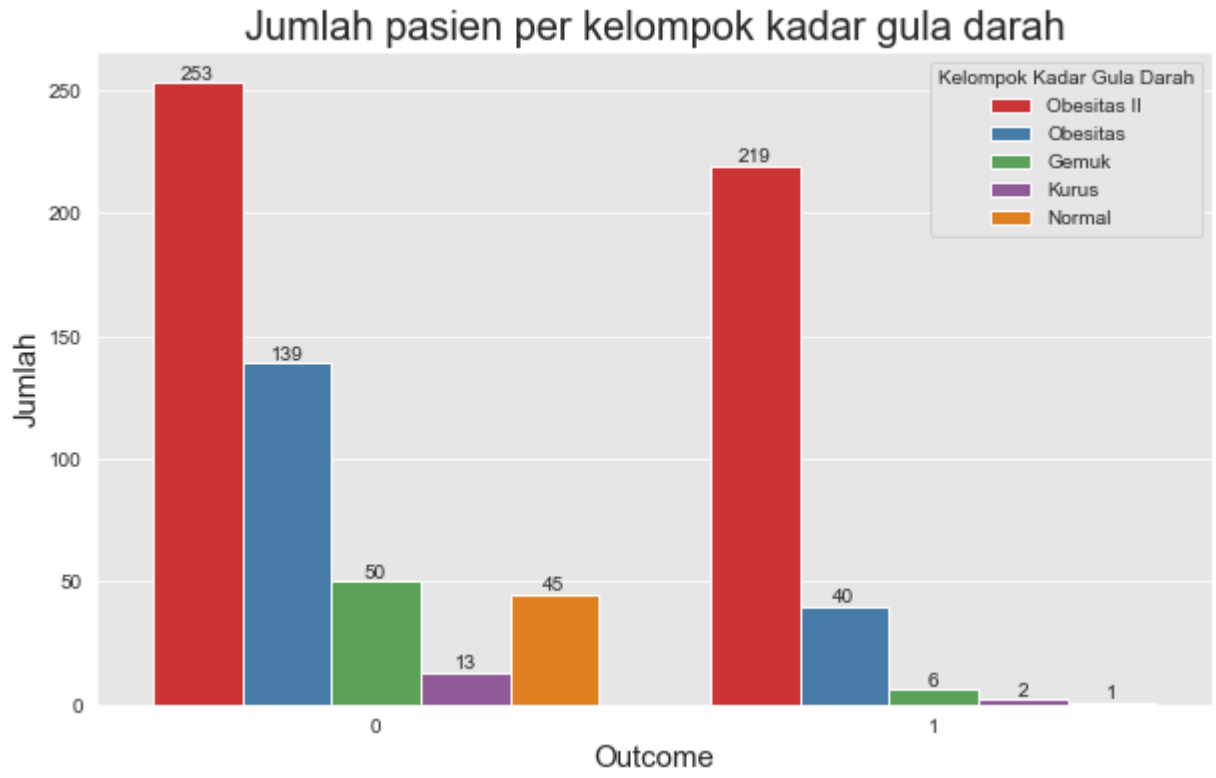
In [65]:
```
df['BMI_grup'] = BMI_grup
df
```

Out[65]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 11 columns

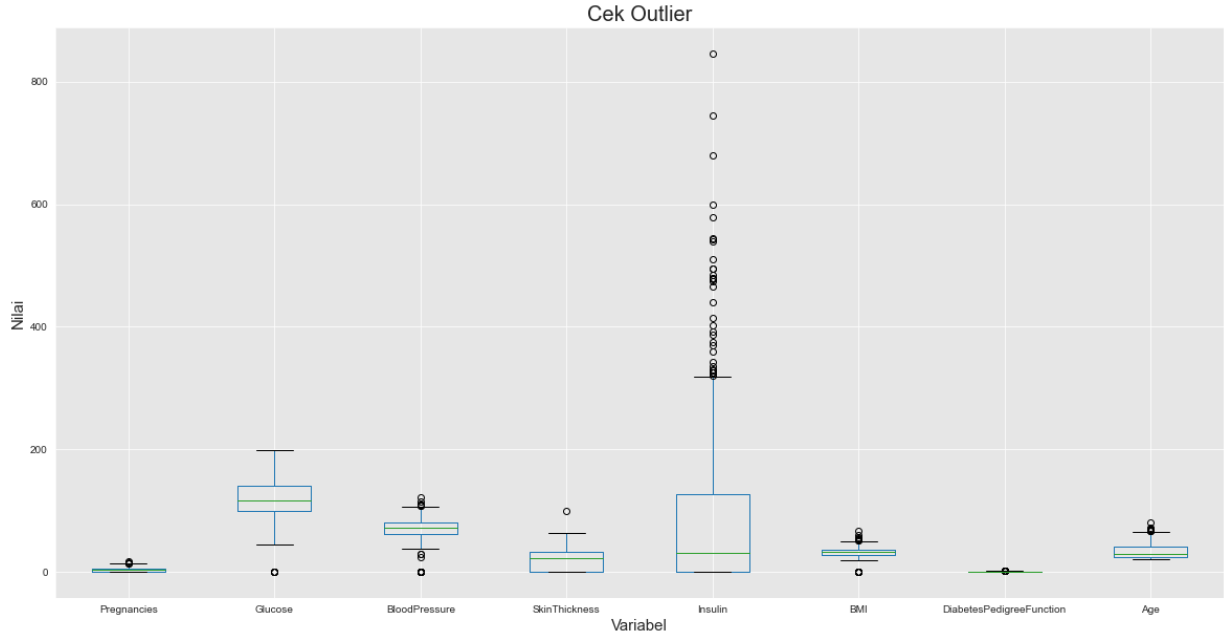
In [66]:
```
plt.figure(figsize=(10,6))
a = sns.countplot(x='Outcome', hue='BMI_grup', data=df, palette='Set1')
for j in a.containers:
    a.bar_label(j, label_type='edge')
plt.title('Jumlah pasien per kelompok kadar gula darah', fontsize=20)
plt.xlabel('Outcome', fontsize=15)
plt.ylabel('Jumlah', fontsize=15)
plt.legend(loc='upper right', title='Kelompok Kadar Gula Darah')
plt.show()
```



Berdasarkan kelompok BMI yang paling banyak terkena diabetes adalah yang memiliki BMI lebih dari 30 (Obesitasa II) dengan jumlah 219 orang lalu diikuti dengan BMI 25 - 29.9 (Obesitas) dengan jumlah 40 orang.

Outlier Detection

In [67]:
```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'B
df[cols].boxplot(figsize=(20,10))
plt.title('Cek Outlier', fontsize=20)
plt.xlabel('Variabel', fontsize=15)
plt.ylabel('Nilai', fontsize=15)
plt.show()
```



In [68]:
```
# Cek outlier
# Cek Outlier dengan IQR
def outlier_iqr(data):
    outliers = []
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1
    batas_bawah = q1 - 1.5 * iqr
    batas_atas = q3 + 1.5 * iqr
    return batas_bawah, batas_atas
    for i in data:
        if i < batas_bawah or i > batas_atas:
            outliers.append(i)
    return outliers
data_outlier = {}
for col in cols:
    data_outlier[col] = outlier_iqr(df[col])
    print('Outlier (',col,') : ',len(data_outlier[col]),' outlier',data_outlier[c
```

```
Outlier ( Pregnancies ) :  2  outlier (-6.5, 13.5)
Outlier ( Glucose ) :  2  outlier (37.125, 202.125)
Outlier ( BloodPressure ) :  2  outlier (35.0, 107.0)
Outlier ( SkinThickness ) :  2  outlier (-48.0, 80.0)
Outlier ( Insulin ) :  2  outlier (-190.875, 318.125)
Outlier ( BMI ) :  2  outlier (13.35, 50.550000000000004)
Outlier ( DiabetesPedigreeFunction ) :  2  outlier (-0.32999999999999996, 1.2)
Outlier ( Age ) :  2  outlier (-1.5, 66.5)
```

In [69]:
```
# Handling insulin
q1 = df['Insulin'].quantile(0.25)
q3 = df['Insulin'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['Insulin'] = np.where(df['Insulin'] > batas_atas, batas_atas, df['Insulin'])
df['Insulin'] = np.where(df['Insulin'] < batas_bawah, batas_bawah, df['Insulin'])
```

```
batas_bawah :  -190.875
batas_atas :  318.125
```

In [70]:
```
# Handling blodplassure
q1 = df['BloodPressure'].quantile(0.25)
q3 = df['BloodPressure'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['BloodPressure'] = np.where(df['BloodPressure'] > batas_atas, batas_atas, df['
df['BloodPressure'] = np.where(df['BloodPressure'] < batas_bawah, batas_bawah, df
```

```
batas_bawah :  35.0
batas_atas :  107.0
```

In [71]:
```
# Handling skinthickness
q1 = df['SkinThickness'].quantile(0.25)
q3 = df['SkinThickness'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['SkinThickness'] = np.where(df['SkinThickness'] > batas_atas, batas_atas, df['
df['SkinThickness'] = np.where(df['SkinThickness'] < batas_bawah, batas_bawah, df
```

```
batas_bawah :  -48.0
batas_atas :  80.0
```

In [72]:
```
# Handling BMI
q1 = df['BMI'].quantile(0.25)
q3 = df['BMI'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['BMI'] = np.where(df['BMI'] > batas_atas, batas_atas, df['BMI'])
df['BMI'] = np.where(df['BMI'] < batas_bawah, batas_bawah, df['BMI'])
```

```
batas_bawah :  13.35
batas_atas :  50.550000000000004
```

In [73]:
```
# Handling Glucose
q1 = df['Glucose'].quantile(0.25)
q3 = df['Glucose'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['Glucose'] = np.where(df['Glucose'] > batas_atas, batas_atas, df['Glucose'])
df['Glucose'] = np.where(df['Glucose'] < batas_bawah, batas_bawah, df['Glucose'])
```

```
batas_bawah :  37.125
batas_atas :  202.125
```

In [74]:
```
# Handling Age
q1 = df['Age'].quantile(0.25)
q3 = df['Age'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['Age'] = np.where(df['Age'] > batas_atas, batas_atas, df['Age'])
df['Age'] = np.where(df['Age'] < batas_bawah, batas_bawah, df['Age'])
```

```
batas_bawah :  -1.5
batas_atas :  66.5
```

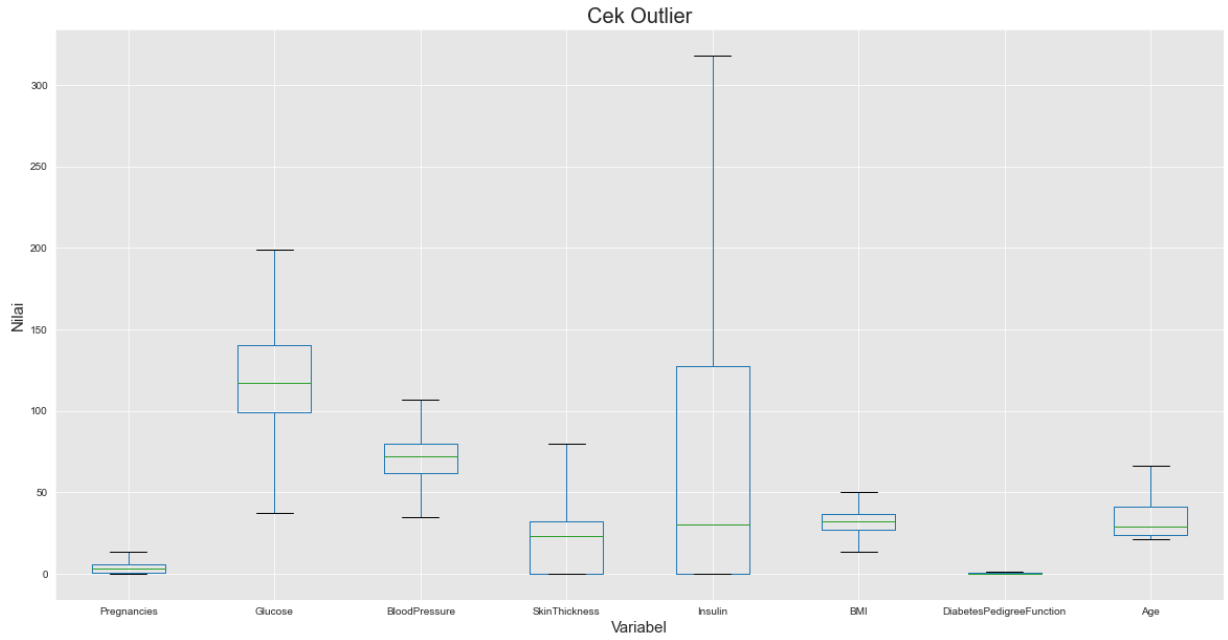
In [75]:
```
# handling diabetespedigreefunction
q1 = df['DiabetesPedigreeFunction'].quantile(0.25)
q3 = df['DiabetesPedigreeFunction'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['DiabetesPedigreeFunction'] = np.where(df['DiabetesPedigreeFunction'] > batas_
df['DiabetesPedigreeFunction'] = np.where(df['DiabetesPedigreeFunction'] < batas_
```

```
batas_bawah :  -0.32999999999999996
batas_atas :  1.2
```

In [76]:
```
# Handling Pregnancies
q1 = df['Pregnancies'].quantile(0.25)
q3 = df['Pregnancies'].quantile(0.75)
iqr = q3 - q1
batas_bawah = q1 - 1.5 * iqr
batas_atas = q3 + 1.5 * iqr
print('batas_bawah : ',batas_bawah)
print('batas_atas : ',batas_atas)
df['Pregnancies'] = np.where(df['Pregnancies'] > batas_atas, batas_atas, df['Preg
df['Pregnancies'] = np.where(df['Pregnancies'] < batas_bawah, batas_bawah, df['Pr
```

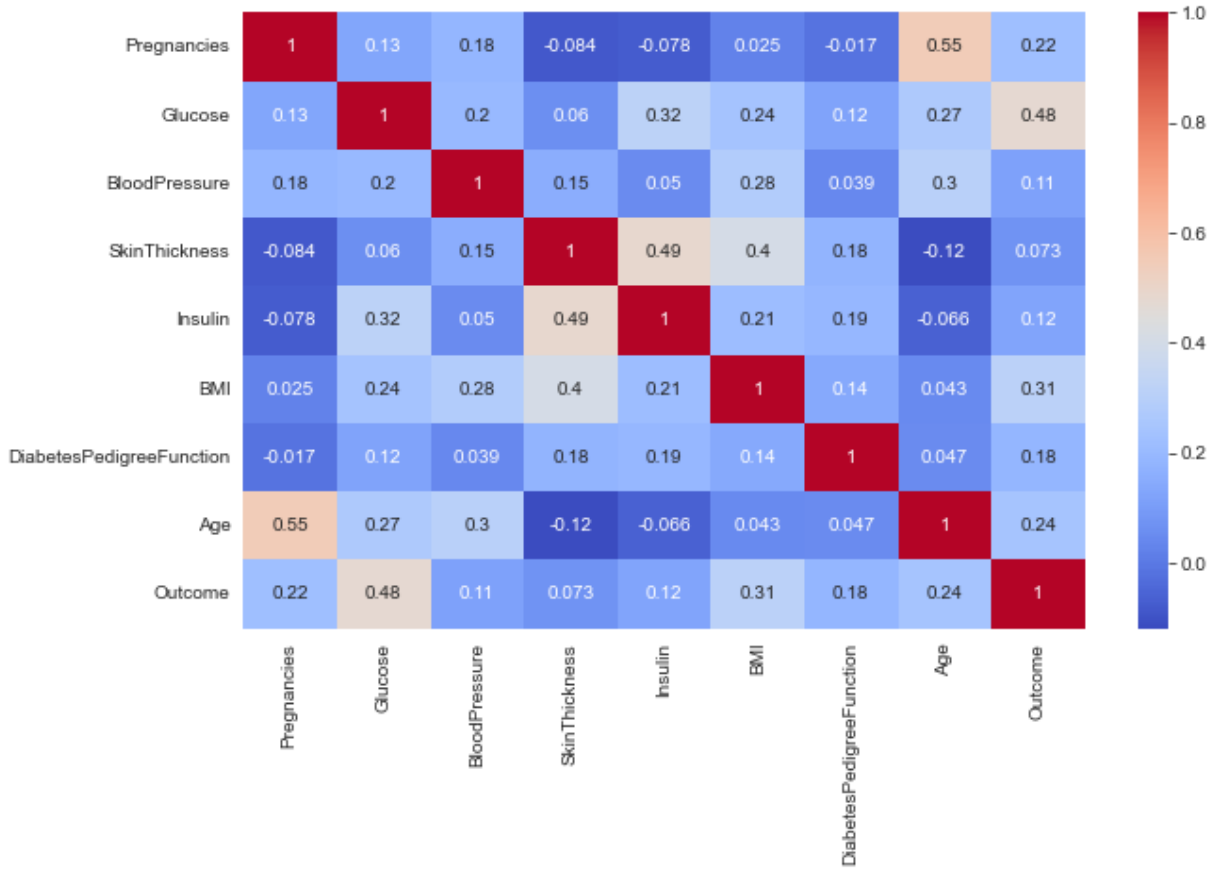
```
batas_bawah :  -6.5
batas_atas :  13.5
```

In [77]:
```
# Setelah di handling
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'B
df[cols].boxplot(figsize=(20,10))
plt.title('Cek Outlier', fontsize=20)
plt.xlabel('Variabel', fontsize=15)
plt.ylabel('Nilai', fontsize=15)
plt.show()
```



Data Preprocessing

In [78]:
```
# Cek korelasi
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



In [79]:
```
# Simpan dataset
#df.to_csv('diabetes_clean.csv', index=False)
```

In [80]:
```
# Train test split
X = df.drop(['Outcome', 'Age_grup', 'BMI_grup'], axis=1)
y = df['Outcome']
```

Oversampling

In [81]:
```
y.value_counts()
```

Out[81]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [82]:
```
# Imbalance data
from imblearn.over_sampling import SMOTE
from collections import Counter
smote = SMOTE()
X, y = smote.fit_resample(X, y)
print(sorted(Counter(y).items()))
```

```
[(0, 500), (1, 500)]
```

Feature Scaling

In [83]:
```
# MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
print(X)
```

```
[[0.44444444 0.68494208 0.51388889 ... 0.54435484 0.48930481 0.63736264]
 [0.07407407 0.2957529  0.43055556 ... 0.3561828  0.24331551 0.21978022]
 [0.59259259 0.9011583  0.40277778 ... 0.26747312 0.52941176 0.24175824]
 ...
 [0.4149944  0.79427823 0.70414983 ... 0.51703568 0.69913331 0.42843822]
 [0.22247413 0.81471383 0.54164777 ... 0.44494367 0.16961385 0.21985496]
 [0.39073446 0.68123413 0.68819209 ... 0.80745094 0.34205092 0.34230831]]
```

Splitting Dataset

In [84]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2
```

In [85]:
```
X_train
```

Out[85]:
```
array([[0.07407407, 0.38841699, 0.43055556, ..., 0.50134409, 0.32620321,
        0.46153846],
       [0.50481487, 0.76307334, 0.26645832, ..., 0.46400536, 0.45475156,
        0.39255496],
       [0.2962963 , 0.5984556 , 0.51388889, ..., 0.28091398, 0.17736185,
        0.85714286],
       ...,
       [0.07407407, 0.83320463, 0.45833333, ..., 0.78091398, 0.55614973,
        0.15384615],
       [0.07407407, 0.30810811, 0.59722222, ..., 0.57123656, 0.02049911,
        0.02197802],
       [0.2962963 , 0.2957529 , 0.31944444, ..., 0.38844086, 0.20320856,
        0.15384615]])
```

In [86]:
```
y_train
```

Out[86]:
```
390    0
847    1
93     1
236    1
858    1
      ..
118    0
334    0
409    1
225    0
482    0
Name: Outcome, Length: 640, dtype: int64
```

In [87]:
```
# Total data
print('Total data X         : ', len(X))
print('Total data y         : ', len(y))
print('=====================================')
# Train data
print('Total data X_train   : ', len(X_train))
print('Total data y_train   : ', len(y_train))
print('=====================================')
# Validation data
print('Total data X_val     : ', len(X_val))
print('Total data y_val     : ', len(y_val))
print('=====================================')
# Test data
print('Total data X_test    : ', len(X_test))
print('Total data y_test    : ', len(y_test))
```

```
Total data X         :  1000
Total data y         :  1000
=====================================
Total data X_train   :  640
Total data y_train   :  640
=====================================
Total data X_val     :  160
Total data y_val     :  160
=====================================
Total data X_test    :  200
Total data y_test    :  200
```

Modeling

In [88]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import cross_val_score, cross_val_predict
# Model Logistic Regression
from sklearn.linear_model import LogisticRegression
# Model SVM
from sklearn.svm import SVC
```

Logistic Regression

In [89]:
```
model_lr = LogisticRegression(solver='liblinear', random_state=42, max_iter=100)
model_lr.fit(X_train, y_train)
```

Out[89]:
```
LogisticRegression(random_state=42, solver='liblinear')
```

In [90]:
```
preds_lr = model_lr.predict(X_test)
preds_lr[-5:]
```

Out[90]:
```
array([1, 1, 0, 0, 1], dtype=int64)
```

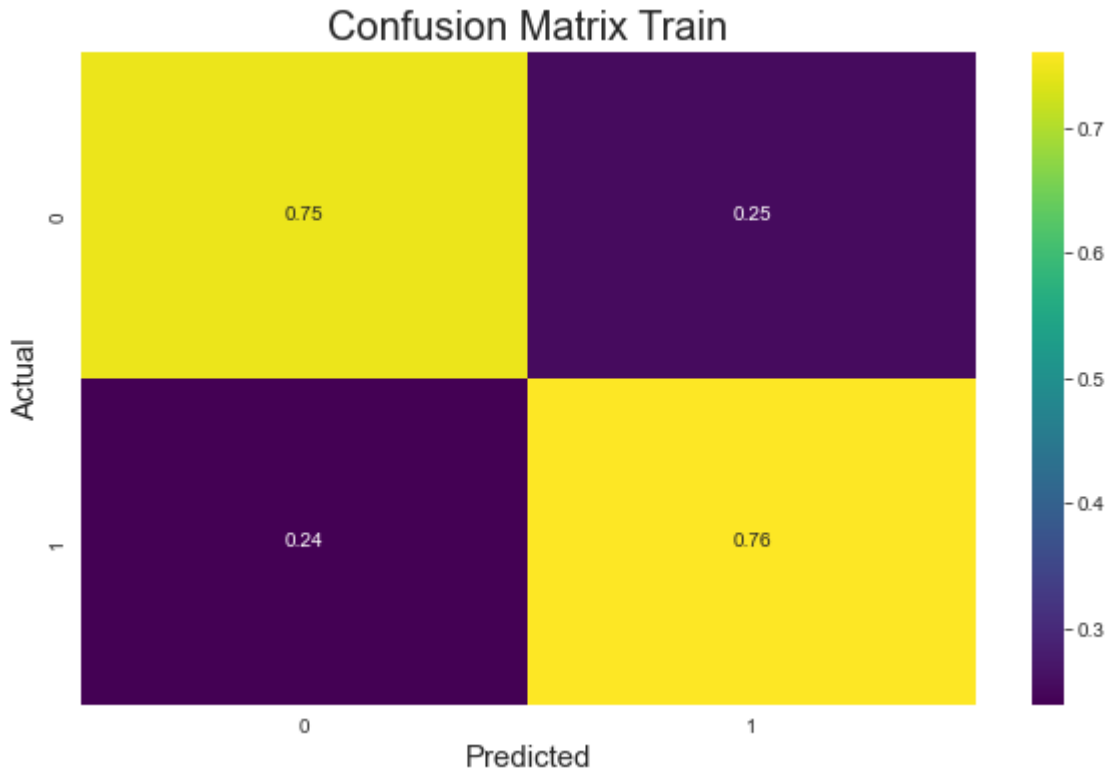
Cross Validation

In [91]:
```
def pred_and_plot(input, target, name=''):
    predict = model_lr.predict(input)
    print('Accuracy Score : ', accuracy_score(predict, target))
    print('Precision Score : ', precision_score(predict, target))
    print('Recall Score : ', recall_score(predict, target))

    cm = confusion_matrix(predict, target, normalize='true')
    plt.figure(figsize=(10,6))
    sns.heatmap(cm, annot=True, cmap='viridis')
    plt.title('Confusion Matrix ' + name, fontsize=20)
    plt.xlabel('Predicted', fontsize=15)
    plt.ylabel('Actual', fontsize=15)
    plt.show()
```

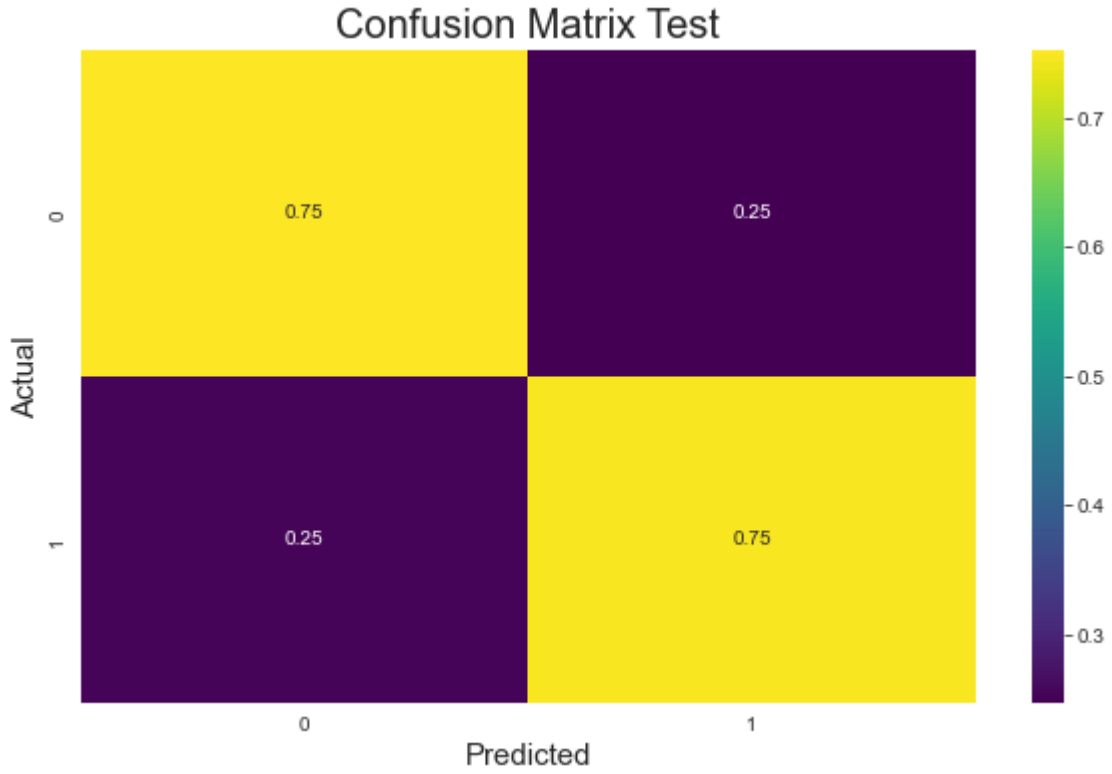
In [92]:
```
pred_and_plot(X_train,  y_train, 'Train')
```

```
Accuracy Score :  0.753125
Precision Score :  0.7366771159874608
Recall Score :  0.7605177993527508
```



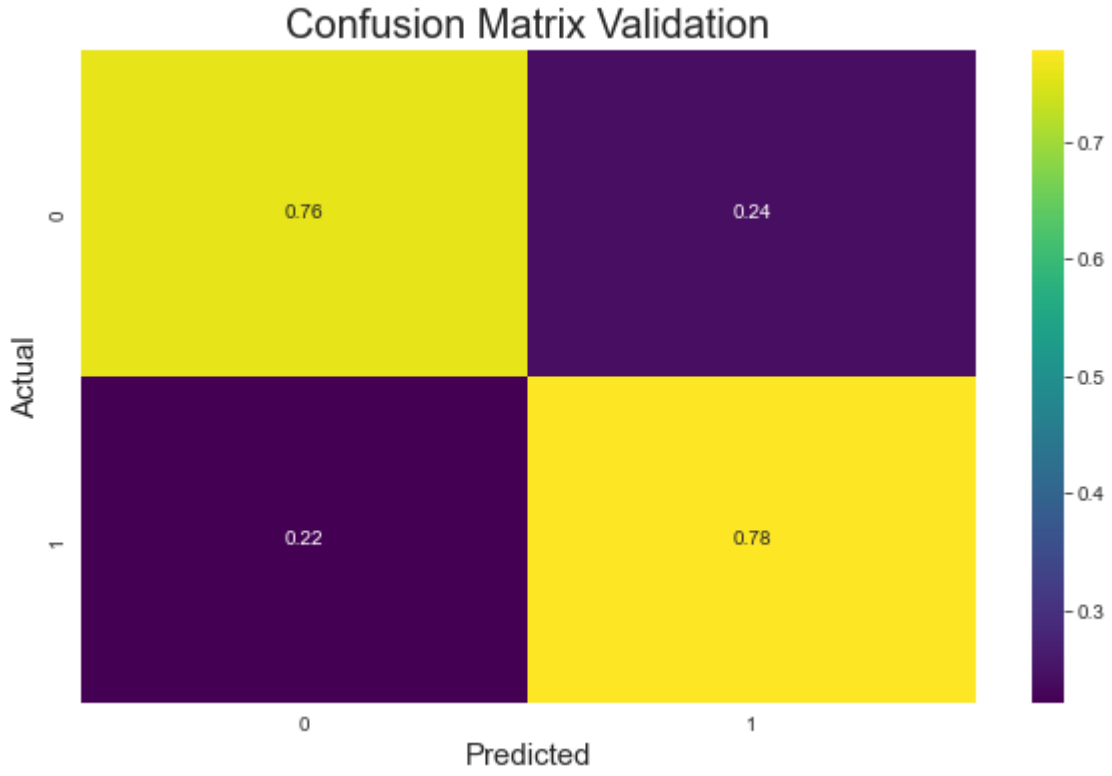
In [93]:
```
test_lr = pred_and_plot(X_test, y_test, 'Test')
```

```
Accuracy Score :  0.75
Precision Score :  0.7623762376237624
Recall Score :  0.7475728155339806
```



In [94]:
```
val_lr = pred_and_plot(X_val, y_val, 'Validation')
```

```
Accuracy Score :  0.76875
Precision Score :  0.75
Recall Score :  0.7792207792207793
```



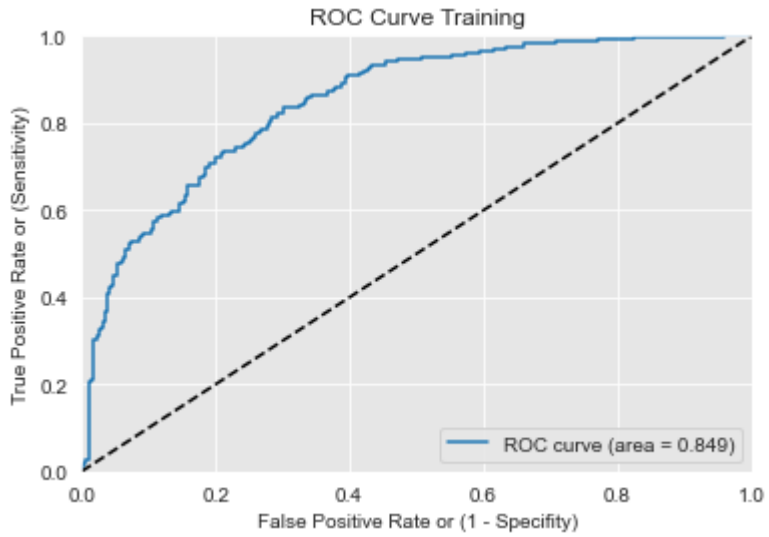
Basic Validation

In [95]:
```
def roc_curve_func(test, predd, nama=''):
  # Compute fpr, tpr, thresholds and roc auc
  fpr, tpr, thresholds = roc_curve(test, predd)
  roc_auc = roc_auc_score(test, predd)


  # Plot ROC curve
  plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)
  plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.0])
  plt.xlabel('False Positive Rate or (1 - Specifity)')
  plt.ylabel('True Positive Rate or (Sensitivity)')
  #plt.title('Receiver Operating Characteristic')
  plt.title('ROC Curve {}'.format(nama))
  plt.legend(loc="lower right")
```

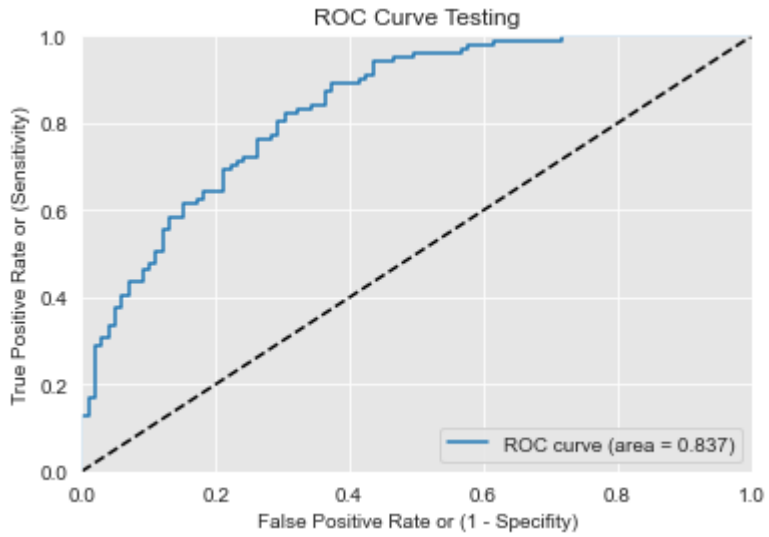
In [96]:
```
y_pred_proba = model_lr.predict_proba(X_train)[:,1]
```

In [97]:
```
roc_curve_func(y_train,y_pred_proba,'Training')
```



In [98]:
```
y_test_pred_proba_lr = model_lr.predict_proba(X_test)[:,1]
```

In [99]:
```
roc_curve_func(y_test, y_test_pred_proba_lr, 'Testing')
```



In [100]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, preds_lr))
```

```
              precision    recall  f1-score   support

           0       0.75      0.74      0.74        99
           1       0.75      0.76      0.75       101

    accuracy                           0.75       200
   macro avg       0.75      0.75      0.75       200
weighted avg       0.75      0.75      0.75       200
```

In [101]:
```
# Tingkat Error Model Logistic Regression Mae
mae_lr = mean_absolute_error(y_test, preds_lr)
print('Mean Absolute Error : ', mae_lr)
```

```
Mean Absolute Error :  0.25
```

Support Vector Machine

In [111]:
```
model_svm = SVC(C=100, gamma=1, kernel='rbf', probability=True)
model_svm.fit(X_train, y_train)
```

Out[111]:
```
SVC(C=100, gamma=1, probability=True)
```

In [112]:
```
preds_svm = model_svm.predict(X_test)
preds_svm[-5:]
```

Out[112]:
```
array([1, 1, 0, 0, 1], dtype=int64)
```

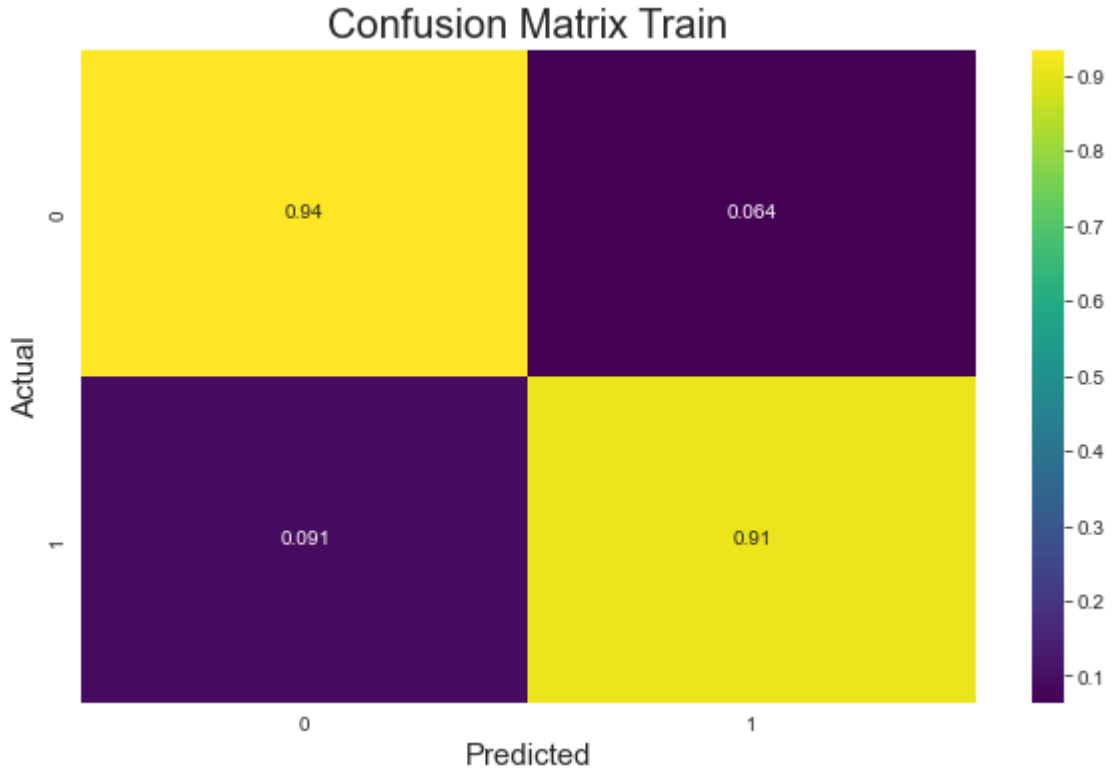
Cross Validation

In [113]:
```
def pred_and_plot(input, target, name=''):
    predict = model_svm.predict(input)
    print('Accuracy Score : ', accuracy_score(predict, target))
    print('Precision Score : ', precision_score(predict, target))
    print('Recall Score : ', recall_score(predict, target))

    cm = confusion_matrix(predict, target, normalize='true')
    plt.figure(figsize=(10,6))
    sns.heatmap(cm, annot=True, cmap='viridis')
    plt.title('Confusion Matrix ' + name, fontsize=20)
    plt.xlabel('Predicted', fontsize=15)
    plt.ylabel('Actual', fontsize=15)
    plt.show()
```

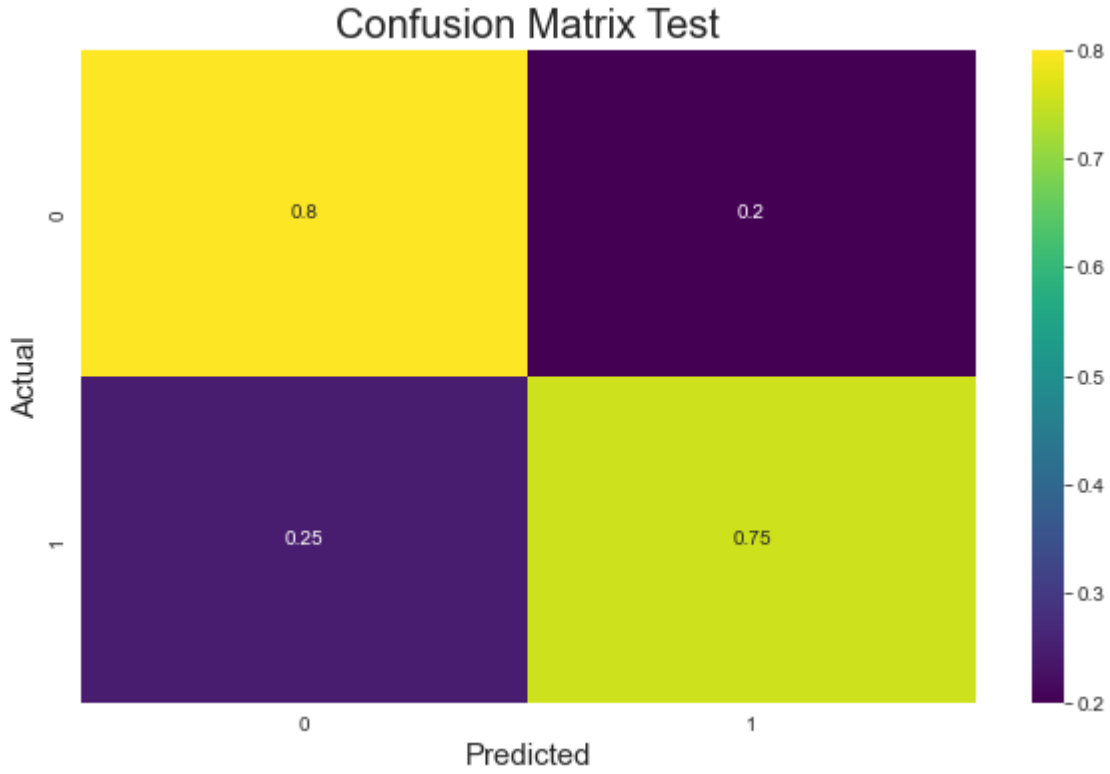
In [114]:
```
train_svm = pred_and_plot(X_train,  y_train, 'Train')
```

```
Accuracy Score :  0.921875
Precision Score :  0.9373040752351097
Recall Score :  0.9088145896656535
```



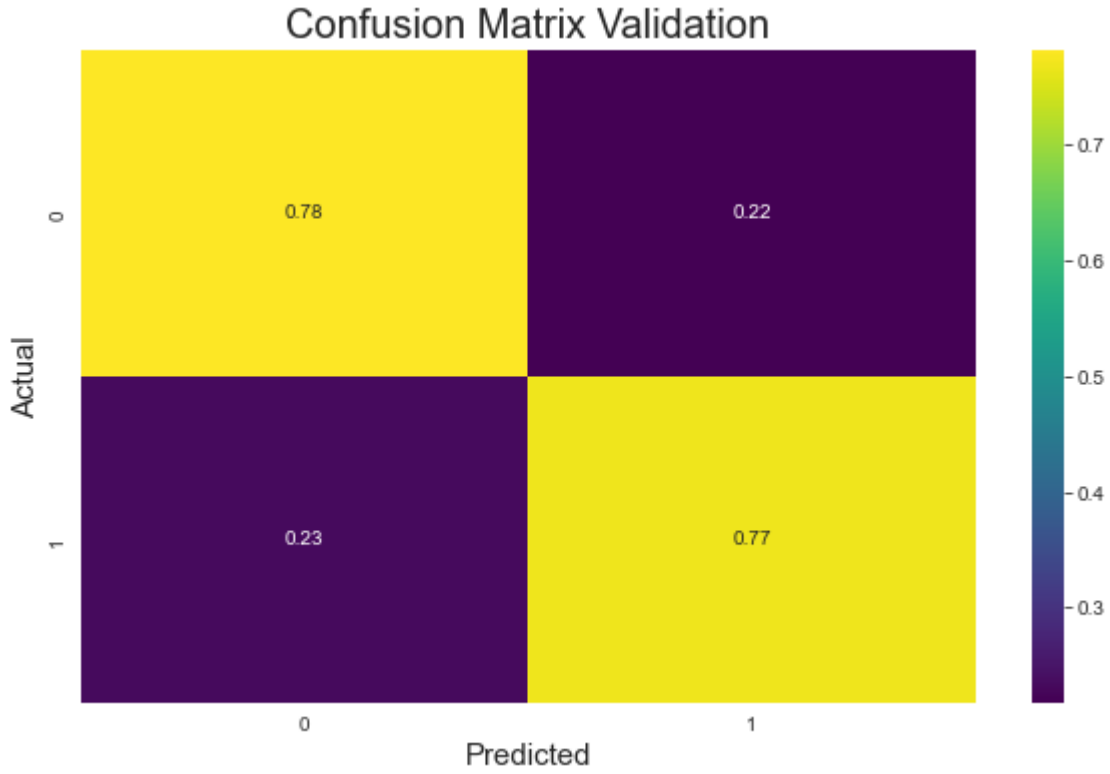
In [115]:
```
test_svm = pred_and_plot(X_test, y_test, 'Test')
```

```
Accuracy Score :  0.775
Precision Score :  0.8217821782178217
Recall Score :  0.7545454545454545
```



In [116]:
```
val_svm = pred_and_plot(X_val, y_val, 'Validation')
```

```
Accuracy Score :  0.775
Precision Score :  0.7875
Recall Score :  0.7682926829268293
```



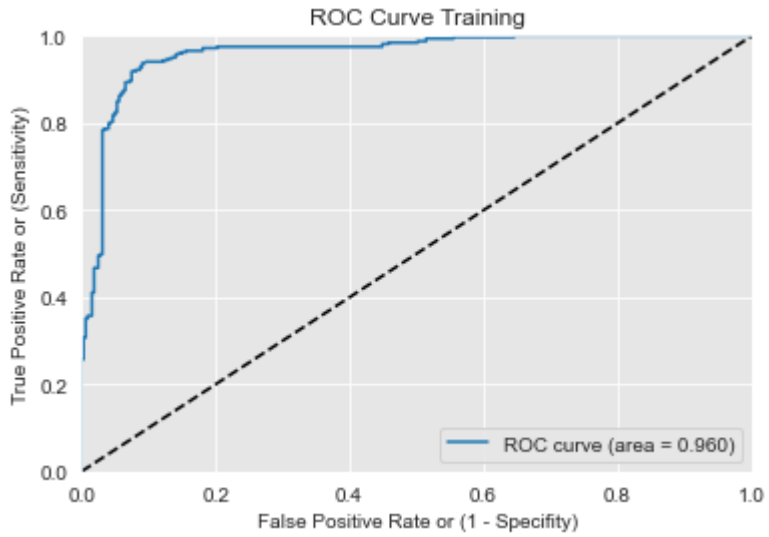
Basic Validation

In [117]:
```
def roc_curve_func(test, predd, nama=''):
  # Compute fpr, tpr, thresholds and roc auc
  fpr, tpr, thresholds = roc_curve(test, predd)
  roc_auc = roc_auc_score(test, predd)


  # Plot ROC curve
  plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)
  plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.0])
  plt.xlabel('False Positive Rate or (1 - Specifity)')
  plt.ylabel('True Positive Rate or (Sensitivity)')
  #plt.title('Receiver Operating Characteristic')
  plt.title('ROC Curve {}'.format(nama))
  plt.legend(loc="lower right")
```

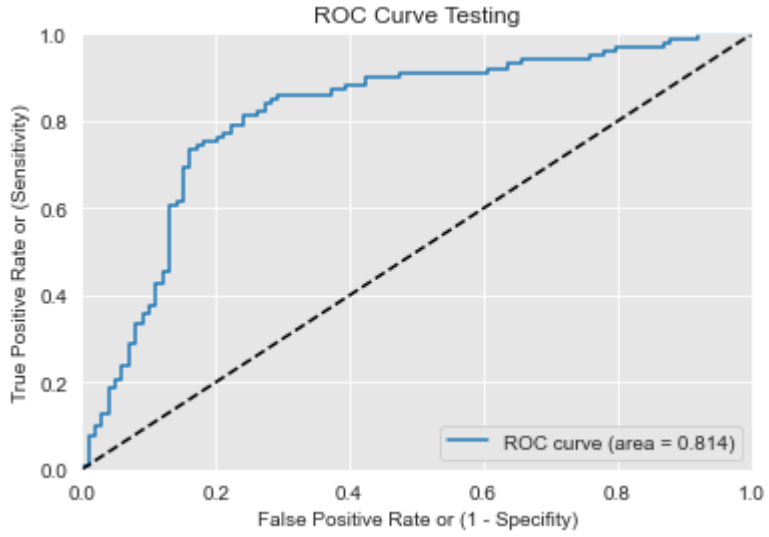
In [118]:
```
y_pred_proba_svm = model_svm.predict_proba(X_train)[:,1]
```

In [119]:
```
roc_curve_func(y_train,y_pred_proba_svm,'Training')
```



In [120]:
```
y_test_pred_proba_svm = model_svm.predict_proba(X_test)[:,1]
```

In [121]:
```
roc_curve_func(y_test, y_test_pred_proba_svm, 'Testing')
```



In [122]:
```
print(classification_report(y_test, preds_svm))
```

```
              precision    recall  f1-score   support

           0       0.80      0.73      0.76        99
           1       0.75      0.82      0.79       101

    accuracy                           0.78       200
   macro avg       0.78      0.77      0.77       200
weighted avg       0.78      0.78      0.77       200
```

In [131]:
```
# Tingkat Error Model SVM Mae
mae_svm = mean_absolute_error(y_test, preds_svm)
print('Mean Absolute Error  : ', mae_svm)
print('Recall Score         : ',recall_score(y_test, preds_svm))
print('Precision Score      : ', precision_score(y_test, preds_svm))
print('Accuracy Score       : ', accuracy_score(y_test, preds_svm))
print('F1 Score             : ', f1_score(y_test, preds_svm))
print('AuC Score            : ', roc_auc_score(y_test, preds_svm))
print('Roc Curve            : ', roc_curve(y_test, preds_svm))
```

```
Mean Absolute Error  :  0.225
Recall Score         :  0.8217821782178217
Precision Score      :  0.7545454545454545
Accuracy Score       :  0.775
F1 Score             :  0.7867298578199051
AuC Score            :  0.7745274527452746
Roc Curve            :  (array([0.        , 0.27272727, 1.        ]), array([0.
, 0.82178218, 1.        ]), array([2, 1, 0], dtype=int64))
```

Hyperparameter Tuning

Dengan membandingkan tingkat eror antara model SVM dan Logistict Regresi saya memilih SVM karena model SVM memiliki tingkat eror yang lebih kecil. Maka dari itu saya akan melakukan hyperparameter tuning pada model SVM. Dengan hypermater tuning saya mencoba untuk menaikan nilai recall, precision, dan f1 score dari model SVM.

In [124]:
```
from sklearn.model_selection import GridSearchCV
param_svm = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
grid_svm = GridSearchCV(SVC(), param_svm, verbose=3, n_jobs=-1, cv=5, scoring='ac
grid_svm.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

Out[124]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'linear']},
             scoring='accuracy', verbose=3)
```

In [125]:
```
pred_grid_svm = grid_svm.predict(X_test)
pred_grid_svm[-5:]
```

Out[125]:
```
array([1, 1, 0, 1, 1], dtype=int64)
```

In [126]:
```
print(classification_report(y_test, pred_grid_svm))
```

```
              precision    recall  f1-score   support

           0       0.78      0.72      0.75        99
           1       0.74      0.80      0.77       101

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.76       200
weighted avg       0.76      0.76      0.76       200
```

In [127]:
```
print('Best Parameters  : ', grid_svm.best_params_)
print('Best Score       : ', grid_svm.best_score_)
print('Best Estimator   : ', grid_svm.best_estimator_)
print('Best Index       : ', grid_svm.best_index_)
print('==========================================')
print('Akurasi Model    : ', accuracy_score(pred_grid_svm, y_test))
print('Precision Model  : ', precision_score(pred_grid_svm, y_test))
print('Recall Model     : ', recall_score(pred_grid_svm, y_test))
print('F1 Score Model   : ', f1_score(pred_grid_svm, y_test))
```

```
Best Parameters  :  {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
Best Score       :  0.8015625
Best Estimator   :  SVC(C=10, gamma=1)
Best Index       :  20
==========================================
Akurasi Model    :  0.76
Precision Model  :  0.801980198019802
Recall Model     :  0.7431192660550459
F1 Score Model   :  0.7714285714285715
```

Dari hasil hyperparameter tuning yang saya lakukan saya mendapatkan nilai recall, precision, dan f1 score yang lebih tinggi dari model SVM sebelumnya.

In [128]:
```
# Prediksi Data Baru
preganancies = int(input('Masukkan Jumlah Kehamilan     : '))
glucose = int(input('Masukkan Kadar Glukosa             : '))
blood_pressure = int(input('Masukkan Tekanan Darah      : '))
skin_thickness = int(input('Masukkan Ketebalan Kulit    : '))
insulin = int(input('Masukkan Insulin                   : '))
bmi = float(input('Masukkan BMI                         : '))
diabetes_pedigree_function = float(input('Masukkan Diabetes Pedigree Function : '
age = int(input('Masukkan Umur                          : '))
data = [[preganancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dia
data
```

Out[128]:
```
[[20, 50, 50, 41, 16, 18.0, 0.2, 25]]
```

In [129]:
```
if grid_svm.predict(data) == 1:
    print('Anda Terdiagnosa Diabetes')
else:
    print('Anda Tidak Terdiagnosa Diabetes')
```

```
Anda Tidak Terdiagnosa Diabetes
```

In [130]:
```
# Save Model SVM ke dalam File Pickle
import pickle
pkl_filename = "model_svm.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(model_svm, file)

# Load Model SVM dari File Pickle
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)

# Prediksi Data Baru dengan Model SVM yang sudah di Load dari File Pickle
score = pickle_model.score(X_test, y_test)
print("Test score: {0:.2f} %".format(100 * score))
y_predict = pickle_model.predict(X_test)
print("Predicted values:")
print(y_predict)
```

```
Test score: 77.50 %
Predicted values:
[0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0
 1 1 0 0 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0
 1 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1 1 0
 1 1 1 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0
 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1
 0 1 1 1 1 1 1 0 1 0 1 1 0 0 1]
```