Industrial Engineering Department

پروژه درس داده کاوی و کاربردهای آن

استاد: دکتر فراز صالحی

دستیار آموزشی: پویا ملکوتی

دانشجویان: محمدفاضل بصیری 99154201

مهسا کشاورزی نژاد 400150748

عنوان پروژه

Stroke Prediction

هدف اصلی پروژه

این دیتاست مربوط به اطلاعات افراد مختلفی با مشخصات مختلف می باشد که با توجه به شرایط مختلف مانند(سن ، شغل ، وضعیت Smoke ، نوع شغل و ...)دچار سکته شده اند یا نشده اند.

هدف ما از این پروژه پیاده سازی مدل یادگیری ماشین نظارت شده ای است (Classification)که بتواند افراد به دو دسته طبقه بندی کند.(گروه اول کسانی که احتمال وقوع سکته در آن ها وجود دارد و گروه دوم کسانی که احتمال وقوع سکته در آن ها وجود ندارد و یا کم است.)

دیتاست موجود از سایت Kaggle می باشد و از طریق لینک زیر قابل دانلود است.

https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset/data

توضیحات در مورد دیتاست و فیلد های موجود در دیتاست:

دیتاست انتخابی شامل فیلد های زیر بوده و اختصار به توضیح هر کدام از ستون ها میپردازیم.

- ID : این ستون نشان دهنده شماره هر رکورد بوده و مقدار آن برای هر سطر متفاوت است و در تحلیل ما کاربردی ندارد.(ارتباطی با پیش بینی ما ندارد.)
- Gender: این فیلد مربوط جنسیت افراد می باشد و افراد در دو دسته با جنسیت male و Female می باشند.
- Age: این فیلد مربوط به سن افراد می باشد و افراد در سنین مختلف مورد بررسی قرار گرفته اند.
- Hypertension : این فیلد افرادی که سابقه فشار خون دارند را مشخص میکند.(مقدار صفر یعنی فرد دارای سابقه فشار خون نیست و مقدار 1 یعنی فرد سابقه فشار خون دارد.)
- heart disease: این فیلد افرادی که سابقه بیماری قلبی دارند را مشخص می کند (مقدار یک به معنای سابقه بیماری قلبی و مقدار صفر به معنای عدم سابقه بیماری قلبی می باشد.)
- Ever Married : این فیلد سابقه ازدواج فرد را مورد بررسی قرار می دهد.
- Work Type: این فیلد نوع شغل فرد را بررسی می کند.
- Residence type: این فیلد نوع زندگی فرد از نظر جغرافیایی را بررسی می کند(شهری یا روستایی)

- AVG_glucose_level: این فیلد میانگین گلوکز موجود در خون را مشخص می کند.
- BMI شاخص توده بدنی هر فرد را نشان می دهد.
- Smoking status: وضعیت سیگار کشیدن افراد را برسی می کند.
- Stroke: ستون آخر مربوط به وضعیت سکته هر فرد با مشخصات متفاوت است که عدد یک به معنای این است که فرد دچار سکته شده است و مقدار صفر به معنای آن است که فرد دچار سکته نشده است.

برای پیش بینی این مسئله با توجه به دیتاست موجود و همچنین وجود ستون تارگت(ستون Stroke) استفاده از الگوریتم های Classification برای طبقه بندی اشخاص به دو دسته از افراد که احتمال وقوع سکته در آن ها زیاد است (افراد با label 1) و افرادی که احتمال وقوع سکته در آن ها کم است .(افراد با label 0) منطقی به نظر می رسد.

در ادامه به تشریح کد می پردازیم :

در قسمت اول Library های مورد نیاز برای پاکسازی و مصور سازی دیتا import می شود تا مورد استفاده قرار بگیرند.

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

در قسمت بعد دیتاست به عنوان یک دیتا فریم خوانده میشود و داده های تکراری حذف می شود و تعداد مقادیر null برای هر ستون مشخص می شود.

Read the Dataset and Remove duplicates

```
In [34]: df=pd.read_csv("healthcare-dataset-stroke-data.csv")
         df=df.drop_duplicates()
         print(df.isnull().sum())
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

همانگونه که مشخص است در ستون مربوط به BMI ، 201 مقدار null وجود دارد که با استفاده از یک روش باید این مقادیر خالی پر و یا حذف شوند.(طبق شکل زیر از مقدار میانگین برای پر کردن مقادیر خالی ستون BMI استفاده می کنیم.)

Replace the null value of the bmi with mean ¶

In [9]:
```
df.head(10)
df.shape
df['bmi']=df['bmi'].fillna(df['bmi'].mean())
df.head()
```

Out[9]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.600000	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.893237	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.400000	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.000000	never smoked	1

در ادامه دیتا تایپ مربوط به فیلدهای مختلف رو بررسی می کنیم و یک describe از داده های خود بدست می آوریم تا یک دید کلی از داده های خود داشته باشیم.

Check the datatypes and describe of the Dataset

In [11]:
```
df.dtypes
```

Out[11]:
```
id                   int64
gender              object
age                float64
hypertension         int64
heart_disease        int64
ever_married        object
work_type           object
Residence_type      object
avg_glucose_level  float64
bmi                float64
smoking_status      object
stroke               int64
dtype: object
```

In [12]:
```
df.describe()
```

Out[12]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.698018	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.800000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.400000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	32.800000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

در ادامه با استفاده از Interact میتوان داده را براساس فیلد های مختلف فیلتر کرد تا بتوان داده های براساس مقادیر خاصی از هر فیلد مشاهده کرد.

Use interact to filter the Data

```
In [14]: @interact
         def filter_work_type(work_type=list(df2.work_type.unique())):
             return df2[df2.work_type==work_type]
```

work_type Private

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
6	53882	Male	74.0	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5101	36901	Female	45.0	0	0	Yes	Private	Urban	97.95	24.5	Unknown	0
5102	45010	Female	57.0	0	0	Yes	Private	Rural	77.93	21.7	never smoked	0
5103	22127	Female	18.0	0	0	No	Private	Urban	82.85	46.9	Unknown	0
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0

2925 rows × 12 columns

با توجه به این که برخی از ستون های ما مثل ever_married، work_type ، residence_type و smoking_status مقادیر عددی نیستند و دیتا تایپ آن ها از جنس عددی نیستند باید مقادیر آن ها را به مقادیر عددی تبدیل کنیم تا بتوانیم مدل های خود روی این دیتاست فیت کنیم.

روش های مختلفی برای این کار وجود دارد که یکی از روش ها استفاده از Lable Encoder در کتابخانه Sklearn می باشد.

همانگونه که در تصویر زیر مشخص است دیتا تایپ ستون های غیرعددی به مقادیر عددی تغییر پیدا می کنند تا مقادیر همه ستون ها از جنس مقادیر عددی باشد.

Lable encoding

In [25]:
```
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
df['ever_married']=l.fit_transform(df['ever_married'])
df['Residence_type']=l.fit_transform(df['Residence_type'])
df['work_type']=l.fit_transform(df['work_type'])
df['smoking_status']=l.fit_transform(df['smoking_status'])
df['gender']=l.fit_transform(df['gender'])
df.dtypes
```

Out[25]:
```
id                   int64
gender               int32
age                float64
hypertension         int64
heart_disease        int64
ever_married         int32
work_type            int32
Residence_type       int32
avg_glucose_level  float64
bmi                float64
smoking_status       int32
stroke               int64
dtype: object
```

تا اینجای کار مراحل زیر را طی کردیم :

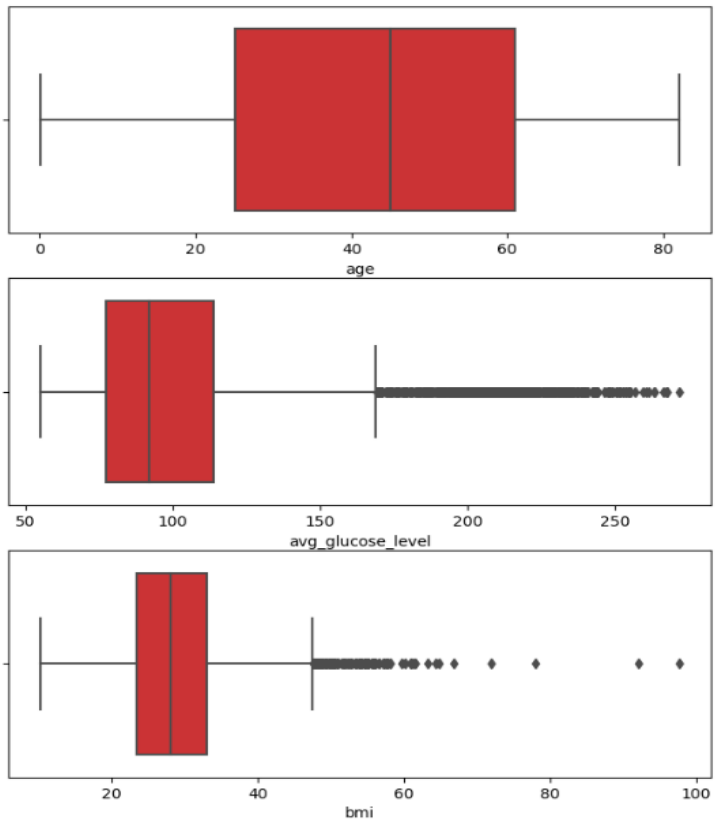
1. دیتافریم خود را تشکیل دادیم.
2. مقادیر تکراری دیتاست را حذف کردیم.
3. مقادیر خالی دیتا ست را پیدا کرده و جایگزین کردیم.
4. مقادیر غیر عددی ستون های مختلف را label گذاری کردیم تا به مقادیر عددی تبدیل شوند.

بصری سازی داده

در ادامه به بصری سازی داده ها می پردازیم تا یک نگاه کلی در ارتباط با داده های خود داشته باشیم تا بتوانیم تحلیل بهتری از داده های خود داشته باشیم.

با استفاده از box plot میتوانیم برای مقادیر عددی سه ستون مربوط به سن و میانگین سطح گلوکز وbmi را مورد بررسی قراردهیم و مقادیر Outlier را شناسایی کنیم.

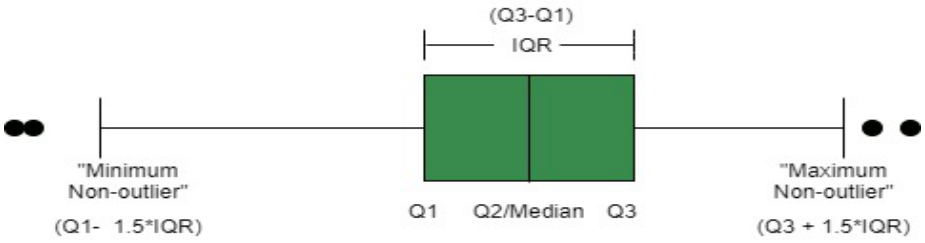
```
In [27]: plt.figure(figsize=(8,10))
for i,col in enumerate(df.select_dtypes(float).columns):
    plt.subplot(3,1,i+1)
    sns.boxplot(data=data,x=col,palette =sns.color_palette("Set1"))
```



همانگونه که در تصویر بالا مشخص است این دیتا سن بیشتر مربوط به افراد در بازه سنی حدود25 تا 60 سال است.

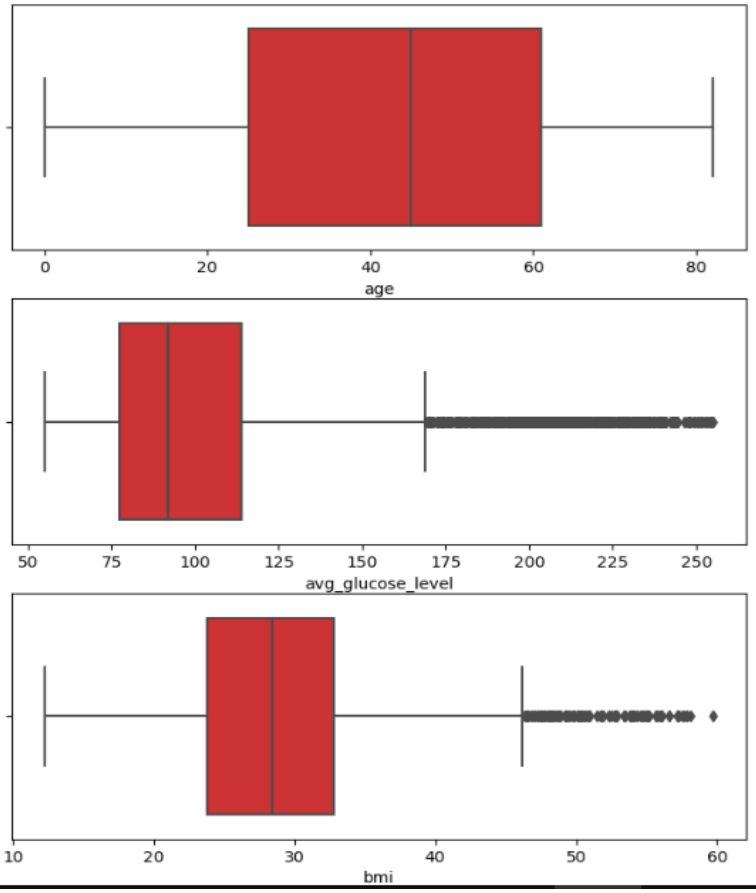
همانگونه که مشخص است avg glucose و bmi دارای مقادیر outlier هستند.

شکل زیر مربوط به راهنمای box plot می باشد .



در ادامه برای حذف دیتای outlier ، مقادیر bmi را بین 12 تا 60 درنظر میگیریم و avg glucose بالاتر از 250 را از دیتا فریم خود حذف می کنیم.

```
In [96]: df = df[(df["bmi"]<60)&(df["bmi"]>12)]
         df = df[(df["avg_glucose_level"]<255)]
         plt.figure(figsize=(8,10))
         for i,col in enumerate(df.select_dtypes(float).columns):
             plt.subplot(3,1,i+1)
             sns.boxplot(data=df,x=col,palette =sns.color_palette("Set1"))
```

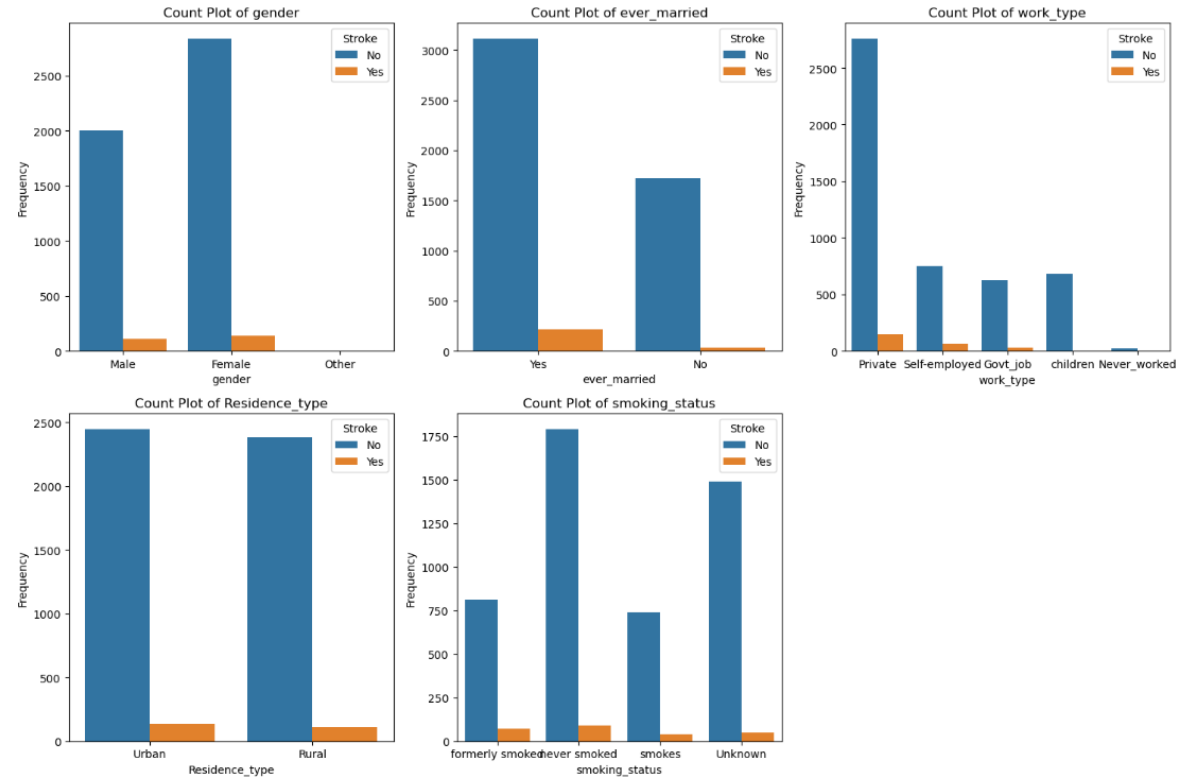


در ادامه با استفاده از کد زیر یک نگاه کلی از ارتباط هر feature با تارگت خود یعنی وقوع Stroke را مورد بررسی قرار میدهیم. برای مثال در نمودار اول از سمت چپ وقوع یا عدم وقوع stroke به

ازای جنسیت های مختلف مورد بررسی قرار میگیرد.

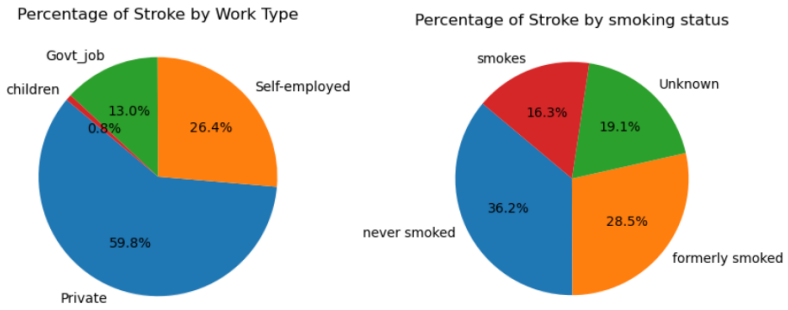
Count plot of data

```
112]: df2 = df2[(df2["bmi"]<60)&(df2["bmi"]>12)]
df2 = df2[(df2["avg_glucose_level"]<255)]

categorical_cols = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']


plt.figure(figsize=(15, 10))

for i, col in enumerate(categorical_cols, 1):
    plt.subplot(2, 3, i)
    sns.countplot(data=df2, x=col, hue='stroke')
    plt.title(f'Count Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.legend(title='Stroke', labels=['No', 'Yes'])

plt.tight_layout()
plt.show()

value_counts = {col: data[col].value_counts() for col in categorical_cols}
value_counts
```

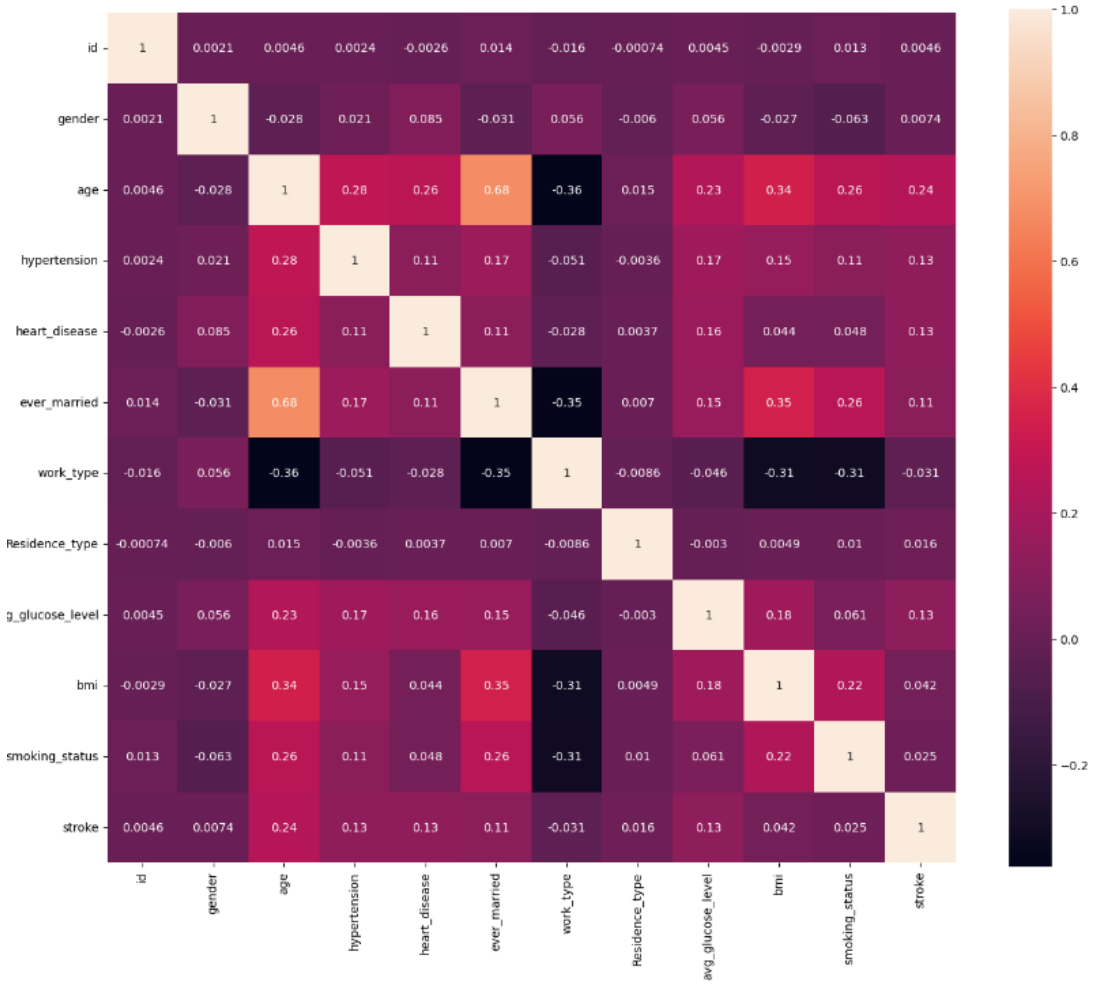


در ادامه pie chart هایی برای تفکیک بهتر feature ها و ارتباطشان با stroke رسم شده است.

و خروجی آن ها نمایش داده شده است.



بررسی correlation : همانگونه که مشخص است همبستگی stroke با سن و سابقه بیماری قلبی و فشار خون بیشتر است در این دیتاست بیشتر است.



جداسازی دیتای train و test و استاندارد سازی دیتا

برای فیت کردن مدل های یادگیری ماشین خود نیاز به تقسیم بندی دیتا به دو بخش train و test داریم تا بتوانیم مدل های خود را پیاده سازی کنیم و همچنین با استفاده از داده تست مدل خود را ارزیابی کنیم.

ابتدا دیتافریم رو به دو بخش x و y تقسیم بندی می کنیم . ستون stroke را به عنوان تارگت و بقیه ستون ها به جز ID را به عنوان feature های پیش بینی استفاده می کنیم.

select X and Y

```
In [198]: df=df.iloc[:,1:]
          x=df.iloc[:,:-1].values
          y=df['stroke'].values
          df
```

Out[198]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	1	2	1	228.69	36.600000	1	1
1	0	61.0	0	0	1	3	0	202.21	28.893237	2	1
2	1	80.0	0	1	1	2	0	105.92	32.500000	2	1
3	0	49.0	0	0	1	2	1	171.23	34.400000	3	1
4	0	79.0	1	0	1	3	0	174.12	24.000000	2	1
...	...	...	...	...	...	...	...	...	...	...	...
5105	0	80.0	1	0	1	2	1	83.75	28.893237	2	0
5106	0	81.0	0	0	1	3	1	125.20	40.000000	2	0
5107	0	35.0	0	0	1	3	0	82.99	30.600000	2	0
5108	1	51.0	0	0	1	2	0	166.29	25.600000	1	0
5109	0	44.0	0	0	1	0	1	85.28	26.200000	0	0

5081 rows × 11 columns

استاندارد سازی

از Standard Scaler برای استاندارد سازی داده استفاده می کنیم.

Standard the data

```
In [199]: from sklearn.preprocessing import StandardScaler
          s=StandardScaler()
          x=s.fit_transform(x)
```

در ادامه از train test split استفاده کرده و دیتای train و test خود را تفکیک می کنیم.(20 درصد از داده ها را به عنوان تست در نظر می گیریم.)

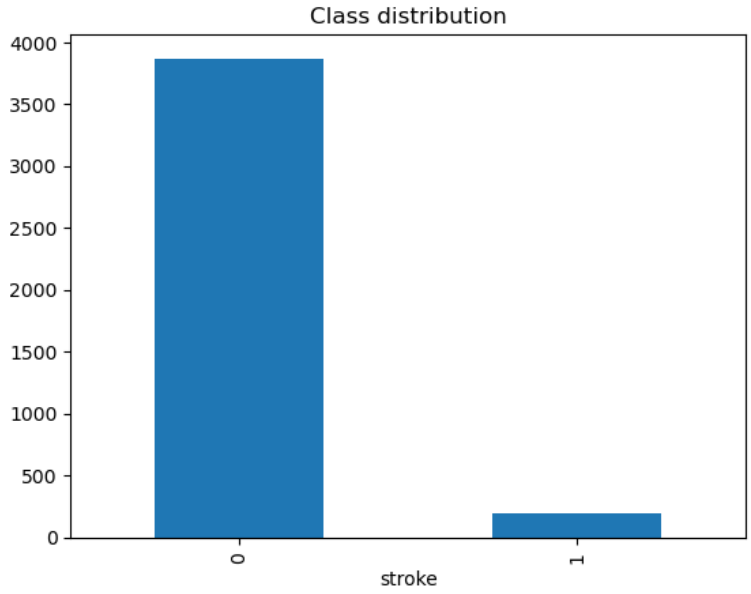
train test spliting

```
In [200]: 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)
```

همانگونه که در تصویر زیر مشخص است ستون مربوط به Stroke توزیع مناسبی ندارد و در دیتاست ما بیشتر دیتا مربوط به افرادی است که دچار Stroke نشده اند و این مسئله برای مدل ما مشکل ایجاد می کند.

توضیح: همانگونه که در نمودار زیر مشخص است y train ما بیشتر برابر مقدار صفر است و این موضوع سبب می شود که یادگیری با مشکل مواجه شود و ماشین به سبب یادگیری نادرست به خوبی عمل نکند.)

```
In [201]: pd.Series(y_train).value_counts().plot(kind='bar', title='Class distribution', xlabel='stroke')
Out[201]: <Axes: title={'center': 'Class distribution'}, xlabel='stroke'>
```

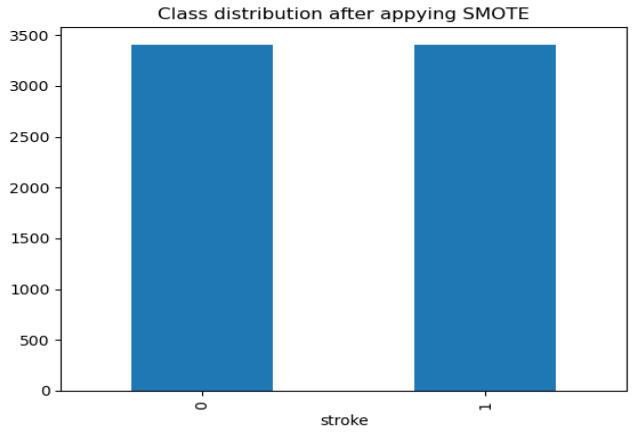


یکی از تکنیک هایی که برای اصلاح این عدم توازن قابل استفاده است ، تکنیک SMOTE می باشد که در ادامه به طور کوتاه مورد بررسی قرار میگیرد.

بعد از اعمال تکنیک SMOTE داده های Y train به شکل زیر می شود:

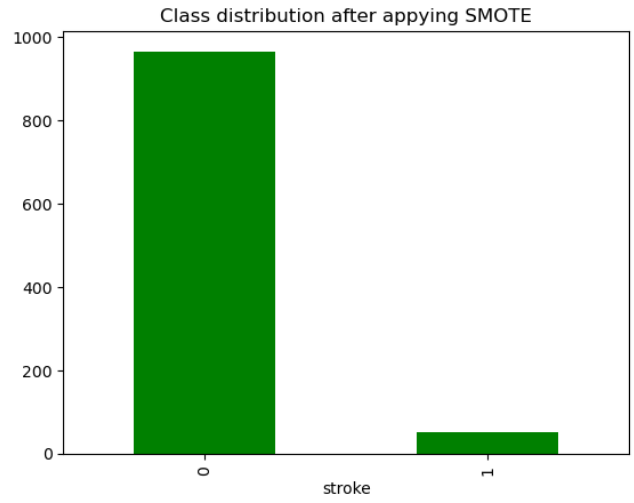
در این جا مشخص است که با استفاده از SMOTE داده ها Resample میشوند و توزیع منطقی تر می شود.

```
In [24]: from imblearn.over_sampling import SMOTE
         smote = SMOTE(random_state=42)

         x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)

         pd.Series(y_train_resampled).value_counts().plot(kind='bar', title='Class distribution after appying SMOTE', xlabel='stroke')

Out[24]: <Axes: title={'center': 'Class distribution after appying SMOTE'}, xlabel='stroke'>
```



طبق شکل زیر این داستان برای داده های تست ما نیز وجود دارد که با استفاده از همین تکنیک میتوان توزیع داده ها رو بهبود بخشید.

```
In [203]: pd.Series(y_test).value_counts().plot(kind='bar', title='Class distribution after appying SMOTE', xlabel='stroke',color='g')

Out[203]: <Axes: title={'center': 'Class distribution after appying SMOTE'}, xlabel='stroke'>
```

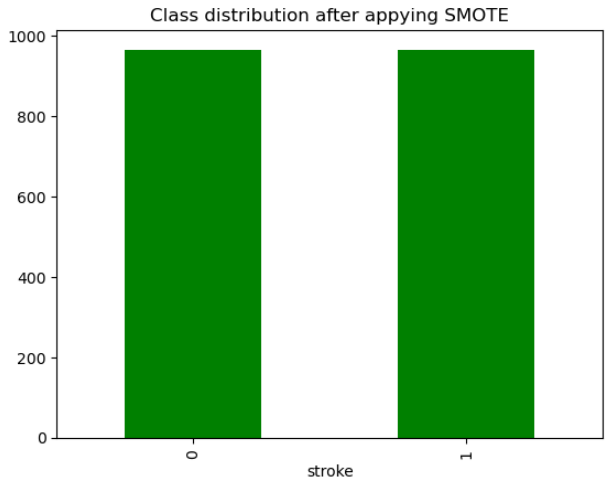


بعد از استفاده از تکنیک SMOTE در داده های تست طبق تصویر زیر مشکل نامتوازن بودن تارگت برطرف می شود.

Use SMOTE to distrobute the data for better analyze

```
In [234]: x_test_resampled, y_test_resampled = smote.fit_resample(x_test, y_test)
          pd.Series(y_test_resampled).value_counts().plot(kind='bar', title='Class distribution after appying SMOTE', xlabel='stroke',color
```

```
Out[234]: <Axes: title={'center': 'Class distribution after appying SMOTE'}, xlabel='stroke'>
```



در ادامه کد زیر جهت سهولت کار با x_train و y_train استفاده شده است.

```
In [235]: x_train=x_train_resampled
          x_test=x_test_resampled
          y_train=y_train_resampled
          y_test=y_test_resampled
```

در ادامه از بین مدل های موجود classification مدل ها رو روی دیتاست خود پیاده سازی میکنیم و عملکرد مدل را بررسی می کنیم.

- KNN

مدل اولی که انتخاب می کنیم مدل KNN می باشد که جزئیات مدل از طریق لینک زیر قابل مشاده می باشد.

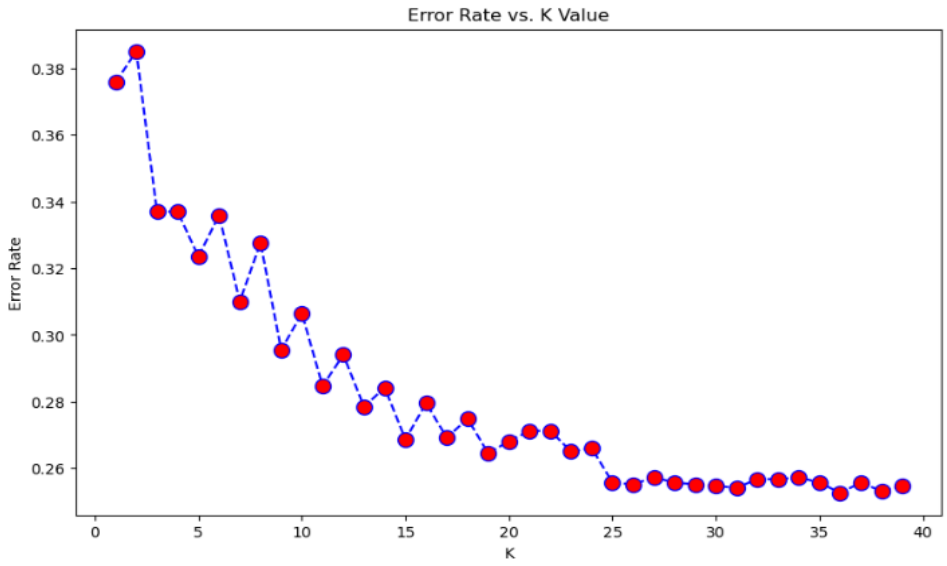
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

یکی از Hyperparameter های اصلی در مدل KNN تعیین مقدار K می باشد.

یکی از روش ها پیدا کردن مقدار K، فیت کردن مدل KNN به ازای K های مختلف در یک رنج می باشد .

نمودار زیر که بیانگر میزان ERROR (که اختلاف بین مقدار پیش بینی شده با مقدار واقعی دیتای تست است) به ازای K های مختلف را نشان میدهد. که به نظر میرسد به ازای K=25 مقدار ERROR کم میشود.

```
In [242]: from sklearn.neighbors import KNeighborsClassifier
          error_rate = []
          for i in range(1,40):

              knn = KNeighborsClassifier(n_neighbors=i)
              knn.fit(x_train,y_train)
              pred_i = knn.predict(x_test)
              error_rate.append(np.mean(pred_i != y_test))


          plt.figure(figsize=(10,6))
          plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
                   markerfacecolor='red', markersize=10)
          plt.title('Error Rate vs. K Value')
          plt.xlabel('K')
          plt.ylabel('Error Rate')

Out[242]: Text(0, 0.5, 'Error Rate')
```



در ادامه مدل با مشخصات زیر روی دیتا فیت می شود و دقت مدل بررسی می شود:

برای بررسی مدل های خود از Classification_Report و Confusion Matrix استفاده میکنیم که توضیحات مربوط به آن ها از دو لینک زیر قابل دسترسی است:

https://medium.com/@chanakapinfo/classification-report-explained-precision-recall-accuracy-macro-average-and-weighted-average-8cd358ee2f8a

https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

همان گونه که مشخص است دقت مدل ما در گزارش زیر قابل بررسی است.

In [382]:
```
knn=KNeighborsClassifier(n_neighbors=25)
knn_model=knn.fit(x_train,y_train)
knn_prediction=knn.predict(x_test)
knn_model
```

Out[382]:
```
KNeighborsClassifier(n_neighbors=25)
```

In [383]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,recall_score,precision_score
knn_accuracy=accuracy_score(y_test,knn_prediction)
knn_precision=precision_score(y_test,knn_prediction)
knn_recall=recall_score(y_test,knn_prediction)
knn_f1Score = f1_score(y_test, knn_prediction, average='weighted')
print('KNN_Accuracy:', knn_accuracy)
print('KNN_F1 score:', knn_f1Score)
```

```
KNN_Accuracy: 0.6922279792746114
KNN_F1 score: 0.6920374921157172
```

In [384]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, knn_prediction))
```

```
              precision    recall  f1-score   support

           0       0.70      0.67      0.68       965
           1       0.68      0.72      0.70       965

    accuracy                           0.69      1930
   macro avg       0.69      0.69      0.69      1930
weighted avg       0.69      0.69      0.69      1930
```

در ادامه Confusion matrix مربوط به مدل KNN را بررسی می کنیم.

همانگونه که از جدول زیر مشخص است مدل از تست ما 692 مورد که دچار Stroke شده اند را درست پیش بینی کرده است .

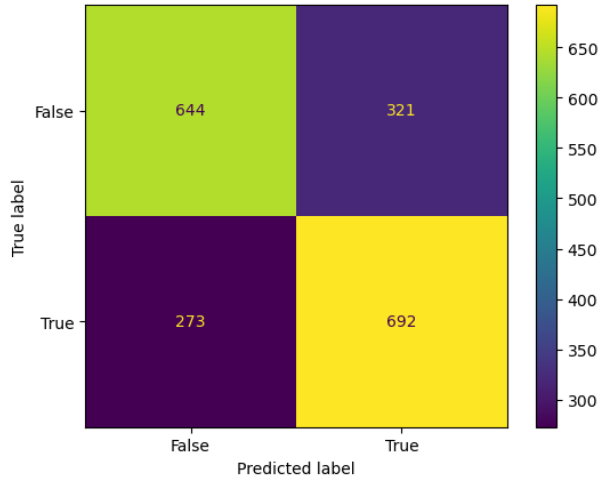
همچنین 644 مورد از کسانی هم که دچار Stroke نشده اند را به درستی تشخیص داده است.

اما مدل خطاهایی هم داشته است و 273 مورد که دچار سکته شده اند را به اشتباه تشخیص داده و پیش بینی کرده است که دچار سکته نشده اند.

همچنین 321 نفر هم دچار سکته نشده اند را به اشتباه در کلاسی قرار داده است که دچار سکته شده اند.

Confusion matrix of KNN model

```
In [387]: cm_knn=metrics.ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[False,True])
          cm_knn.plot()

Out[387]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x268a9f4fe50>
```



در مسئله با توجه به موضوع مسئله مقدار false negative های ما یعنی (جایی که احتمال سکته وجود دارد ولی مدل به اشتباه در کلاس صفر یعنی احتمال کم وقوع سکته طبقه بندی می کند) باید حداقل باشد.

در ادامه با همین روند مدل های مختلف را مورد بررسی قرار می دهیم و در نهایت مدل های مختلف را با یکدیگر مقایسه می کنیم.

Logistic regression

عملکرد مدل طبق گزارش زیر از مدل قبلی بهتر بود.

Logistic Regression ¶

```
from sklearn.linear_model import LogisticRegression
LO=LogisticRegression()
logistic=LO.fit(x_train,y_train)
logistic_predict=LO.predict(x_test)
logistic
```

```
LogisticRegression()
```

```
logistic_accuracy=accuracy_score(y_test,logistic_predict)
logistic_f1Score = f1_score(y_test, logistic_predict, average='weighted')
logistic_recall=recall_score(y_test,logistic_predict)
logistic_precision=precision_score(y_test,logistic_predict)
print('logistic_Accuracy:', logistic_accuracy)
print('logistic_F1 score:', logistic_f1Score)
from sklearn.metrics import classification_report
print(classification_report(y_test, logistic_predict))
```

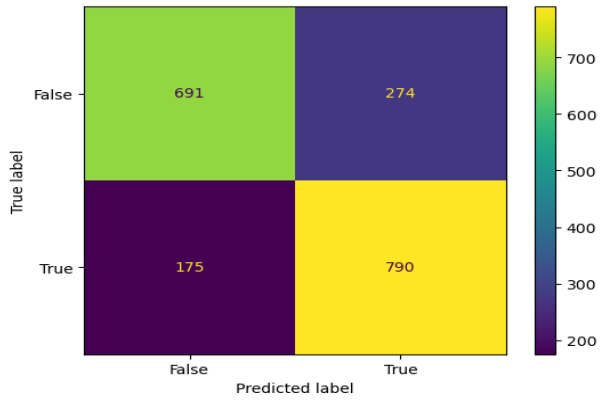
```
logistic_Accuracy: 0.7673575129533678
logistic_F1 score: 0.7667437664514458
              precision    recall  f1-score   support

           0       0.80      0.72      0.75       965
           1       0.74      0.82      0.78       965

    accuracy                           0.77      1930
   macro avg       0.77      0.77      0.77      1930
weighted avg       0.77      0.77      0.77      1930
```

In [396]:
```
cm_log = confusion_matrix(y_test, logistic_predict)
print('Confusion matrix:')
print(cm_log)
```

```
Confusion matrix:
[[691 274]
 [175 790]]
```

In [397]:
```
cm_knn=metrics.ConfusionMatrixDisplay(confusion_matrix=cm_log,display_labels=[False,True])
cm_knn.plot()
```

Out[397]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x268a9f4ef80>
```



DecisionTreeClassifier

یکی از روش های پیدا کردن هایپر پارامتر مناسب استفاده از GridSearchCV می باشد که به ازای پارامتر های مختلف مدل را فیت کرده و در نهایت بهترین پارامتر را از ببین پارامتر های موجود انتخاب می کند.

DecisionTreeClassifier

In [398]:
```
from sklearn.model_selection import GridSearchCV


param_grid = {
    'max_depth': [2, 4, 6, 8, 10],
  'min_samples_split': [2, 4, 6, 8, 10],
  'min_samples_leaf': [1, 2, 3, 4, 5]}

grid_search = GridSearchCV(DT1, param_grid=param_grid, cv=5)
grid_search.fit(x_train, y_train)


print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2}
Best score:  0.8626614987080103
```

In [400]:
```
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, min_samples_split= 2)
DT1=DecisionTreeClassifier()
dt=DT.fit(x_train,y_train)
dt_predict=DT.predict(x_test)
dt
```

Out[400]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, min_samples_leaf=5)
```

In [401]:
```
DecisionTree_accuracy=accuracy_score(y_test,dt_predict)
DecisionTree_precision=precision_score(y_test,dt_predict)
DecisionTree_recall=recall_score(y_test,dt_predict)
DecisionTree_f1Score = f1_score(y_test,dt_predict, average='weighted')
print('DecisionTree_Accuracy:', knn_accuracy)
print('DecisionTree_F1 score:', knn_f1Score)
```

```
DecisionTree_Accuracy: 0.6922279792746114
DecisionTree_F1 score: 0.6920374921157172
```

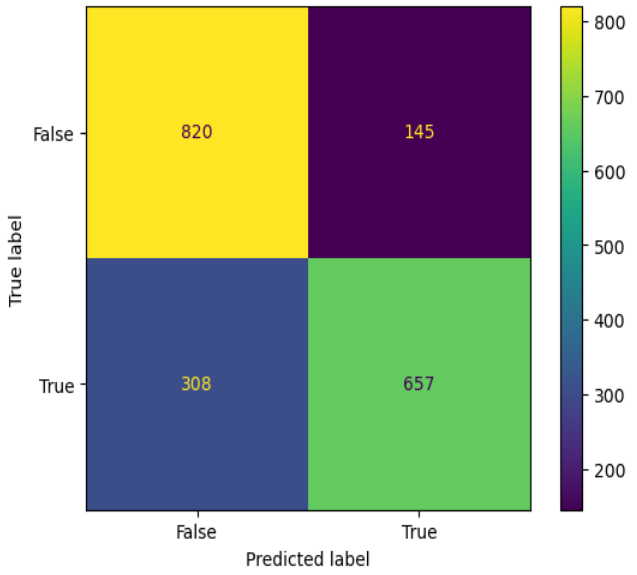
In [402]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, dt_predict))
```

```
              precision    recall  f1-score   support

           0       0.73      0.85      0.78       965
           1       0.82      0.68      0.74       965

    accuracy                           0.77      1930
   macro avg       0.77      0.77      0.76      1930
weighted avg       0.77      0.77      0.76      1930
```

```
cm_DecisionTree = confusion_matrix(y_test, dt_predict)
print('Confusion matrix:')
print(cm_DecisionTree)
cm_dt=metrics.ConfusionMatrixDisplay(confusion_matrix=cm_DecisionTree,display_labels=[False,True])
cm_dt.plot()
```

```
Confusion matrix:
[[820 145]
 [308 657]]
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x268aa08b970>
```



RandomForestClassifier

RandomForestClassifier

In [404]:
```
param_grid = {
   'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, None],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2]
}

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5)

grid_search.fit(x_train, y_train)

print(f"Best hyperparameters: {grid_search.best_params_}")
print(f"Accuracy: {grid_search.best_score_}")
```

```
Best hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Accuracy: 0.9620155038759691
```

In [406]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_depth=None,min_samples_leaf=1,min_samples_split=2,n_estimators=200)
rf_model=rf.fit(x_train,y_train)
rf_predict=rf.predict(x_test)
rf_model
```

Out[406]:
```
RandomForestClassifier(n_estimators=200)
```

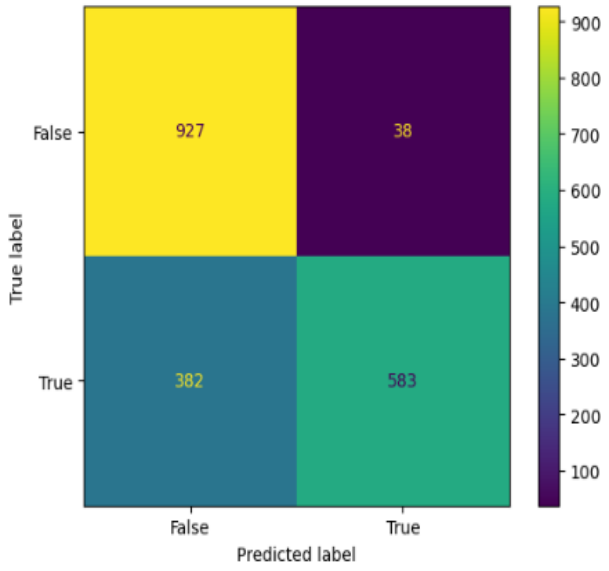
In [407]:
```
print(classification_report(y_test, rf_predict))
rf_accuracy=accuracy_score(y_test,rf_predict)
rf_precision=precision_score(y_test,rf_predict)
rf_recall=recall_score(y_test,rf_predict)
rf_f1score=f1_score(y_test,rf_predict)
cm_rf =confusion_matrix(y_test, rf_predict)
cm_rf=metrics.ConfusionMatrixDisplay(confusion_matrix=cm_rf,display_labels=[False,True])
cm_rf.plot()
```

```
              precision    recall  f1-score   support

           0       0.71      0.96      0.82       965
           1       0.94      0.60      0.74       965

    accuracy                           0.78      1930
   macro avg       0.82      0.78      0.78      1930
weighted avg       0.82      0.78      0.78      1930
```

In [407]:
```
print(classification_report(y_test, rf_predict))
rf_accuracy=accuracy_score(y_test,rf_predict)
rf_precision=precision_score(y_test,rf_predict)
rf_recall=recall_score(y_test,rf_predict)
rf_f1score=f1_score(y_test,rf_predict)
cm_rf =confusion_matrix(y_test, rf_predict)
cm_rf=metrics.ConfusionMatrixDisplay(confusion_matrix=cm_rf,display_labels=[False,True])
cm_rf.plot()
```

```
              precision    recall  f1-score   support

           0       0.71      0.96      0.82       965
           1       0.94      0.60      0.74       965

    accuracy                           0.78      1930
   macro avg       0.82      0.78      0.78      1930
weighted avg       0.82      0.78      0.78      1930
```

Out[407]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x268a9077d90>



XGBOOST (GradientBoostingClassifier)

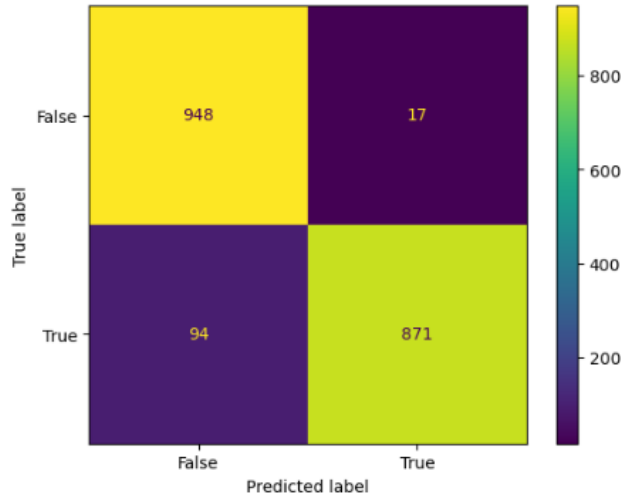
این مدل برروی دیتا ست ما نسبت به مدل های قبل عملکرد بهتری دارد.

XGBOOST (GradientBoostingClassifier)

In [408]:
```
from sklearn.ensemble import GradientBoostingClassifier

xgb_model = GradientBoostingClassifier(learning_rate=0.3,n_estimators=300)
xgb_model.fit(x_train, y_train)
xgb_pred = xgb_model.predict(x_test)
xgb_accuracy=accuracy_score(y_test,xgb_pred)
xgb_precision=precision_score(y_test,xgb_pred)
xgb_recall=recall_score(y_test,xgb_pred)
xgb_f1score=f1_score(y_test,xgb_pred)
print(classification_report(y_test, xgb_pred))
cm_xgb =confusion_matrix(y_test, xgb_pred)
cm_xgb=metrics.ConfusionMatrixDisplay(confusion_matrix=cm_xgb,display_labels=[False,True])
cm_xgb.plot()
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94       965
           1       0.98      0.90      0.94       965

    accuracy                           0.94      1930
   macro avg       0.95      0.94      0.94      1930
weighted avg       0.95      0.94      0.94      1930
```

Out[408]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x268a90af940>



در آخر مقایسه از مدل های مختلف در دو نمودار زیر آورده شده است و طبق نمودار زیر مدل XGBOOST عملکرد بهتری را بر روی دیتا ست ما دارد.

Accuracy, Precision, and Recall Comparison

```
In [409]: algorithms = ['KNeighbors', 'LogisticRegression', 'DecisionTree','RandomForest',
                        'GradientBoosting']
          accuracy = [knn_accuracy, logistic_accuracy,DecisionTree_accuracy,rf_accuracy,xgb_accuracy]
          precision = [knn_precision, logistic_precision,DecisionTree_precision,rf_precision,xgb_precision]
          recall = [knn_recall, logistic_recall,DecisionTree_recall,rf_recall,xgb_recall]

          bar_width = 0.25
          x = np.arange(len(algorithms))
          fig, ax = plt.subplots(figsize=(20, 5))
          accuracy_bars = ax.bar(x - bar_width, accuracy, bar_width, label='Accuracy', color='skyblue')
          precision_bars = ax.bar(x, precision, bar_width, label='Precision', color='lightcoral')
          recall_bars = ax.bar(x + bar_width, recall, bar_width, label='Recall', color='lightgreen')
          ax.set_xticks(x)
          ax.set_xticklabels(algorithms)
          ax.tick_params(axis='x', labelsize=10)
          ax.set_ylabel('Scores')
          ax.set_title('Accuracy, Precision, and Recall Comparison')
          ax.legend()
          plt.show()
```

F1 Score Comparison

```
11]: algorithms = ['KNeighbors', 'LogisticRegression', 'DecisionTree','RandomForest',
                   'GradientBoosting']

     f1_scores =[knn_f1Score, logistic_f1Score,DecisionTree_f1Score,rf_f1score,xgb_f1score]

     bar_width = 0.5
     x = np.arange(len(algorithms))
     fig, ax = plt.subplots(figsize=(20, 6))
     f1_score_bars = ax.bar(x, f1_scores, bar_width, label='F1 Score', color='darkred')
     ax.set_xticks(x)
     ax.set_xticklabels(algorithms)
     ax.set_ylabel('F1 Score')
     ax.set_title('F1 Score Comparison')
     ax.legend()
     plt.show()
```