In [28]:

```
#importing the dataset
import pandas as pd
df_mushroom=pd.read_csv("mushroom.txt",sep=",",header=None)
df_mushroom
```

Out[28]:

	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	18	19	20	21	22
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
5	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	k	n	g
6	e	b	s	w	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	n	m
7	e	b	y	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	s	m
8	p	x	y	w	t	p	f	c	n	p	...	s	w	w	p	w	o	p	k	v	g
9	e	b	s	y	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	s	m
10	e	x	y	y	t	l	f	c	b	g	...	s	w	w	p	w	o	p	n	n	g
11	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	k	s	m
12	e	b	s	y	t	a	f	c	b	w	...	s	w	w	p	w	o	p	n	s	g
13	p	x	y	w	t	p	f	c	n	k	...	s	w	w	p	w	o	p	n	v	u
14	e	x	f	n	f	n	f	w	b	n	...	f	w	w	p	w	o	e	k	a	g
15	e	s	f	g	f	n	f	c	n	k	...	s	w	w	p	w	o	p	n	y	u
16	e	f	f	w	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
17	p	x	s	n	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	g
18	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	n	s	u
19	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	n	s	u
20	e	b	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	s	m
21	p	x	y	n	t	p	f	c	n	n	...	s	w	w	p	w	o	p	n	v	g
22	e	b	y	y	t	l	f	c	b	k	...	s	w	w	p	w	o	p	n	s	m
23	e	b	y	w	t	a	f	c	b	w	...	s	w	w	p	w	o	p	n	n	m
24	e	b	s	w	t	l	f	c	b	g	...	s	w	w	p	w	o	p	k	s	m
25	p	f	s	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	n	v	g
26	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
27	e	x	y	w	t	l	f	c	b	w	...	s	w	w	p	w	o	p	n	n	m
28	e	f	f	n	f	n	f	c	n	k	...	s	w	w	p	w	o	p	k	y	u
29	e	x	s	y	t	a	f	w	n	n	...	s	w	w	p	w	o	p	n	v	d
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8094	e	b	s	g	f	n	f	w	b	g	...	s	w	w	p	w	t	p	w	n	g
8095	p	x	y	c	f	m	f	c	b	y	...	y	c	c	p	w	n	n	w	c	d
8096	e	k	f	w	f	n	f	w	b	w	...	s	w	w	p	w	t	p	w	n	g
8097	p	k	y	n	f	s	f	c	n	b	...	k	p	p	p	w	o	e	w	v	l
8098	p	k	s	e	f	y	f	c	n	b	...	k	w	p	p	w	o	e	w	v	d
8099	e	k	f	w	f	n	f	w	b	w	...	k	w	w	p	w	t	p	w	s	g
8100	e	f	s	n	f	n	a	c	b	o	...	s	o	o	p	n	o	p	b	v	l
8101	p	k	s	e	f	s	f	c	n	b	...	s	p	w	p	w	o	e	w	v	p
8102	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	n	c	l
8103	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	o	c	l
8104	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	n	y	l

	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	18	19	20	21	22
8104	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	n	v	l
8105	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	y	v	l
8106	e	k	s	n	f	n	a	c	b	o	...	s	o	o	p	o	o	p	n	v	l
8107	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	n	c	l
8108	p	k	y	e	f	y	f	c	n	b	...	s	p	w	p	w	o	e	w	v	l
8109	e	b	s	w	f	n	f	w	b	w	...	s	w	w	p	w	t	p	w	n	g
8110	e	x	s	n	f	n	a	c	b	o	...	s	o	o	p	o	o	p	n	v	l
8111	e	k	s	w	f	n	f	w	b	p	...	s	w	w	p	w	t	p	w	n	g
8112	e	k	s	n	f	n	a	c	b	o	...	s	o	o	p	n	o	p	b	v	l
8113	p	k	y	e	f	y	f	c	n	b	...	k	p	p	p	w	o	e	w	v	d
8114	p	f	y	c	f	m	a	c	b	y	...	y	c	c	p	w	n	n	w	c	d
8115	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	v	l
8116	p	k	y	n	f	s	f	c	n	b	...	k	p	w	p	w	o	e	w	v	l
8117	p	k	s	e	f	y	f	c	n	b	...	s	p	w	p	w	o	e	w	v	d
8118	p	k	y	n	f	f	f	c	n	b	...	s	p	w	p	w	o	e	w	v	d
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	b	c	l
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	b	v	l
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	b	c	l
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	w	v	l
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	c	l

8124 rows × 23 columns

In [29]:

```
#lets provide the column names to the dataset
df_mushroom.columns=["type","cap-shape","cap-surface","cap-color","bruises","odor","gill-attachment","gil
```

In [30]:

```
df_mushroom
```

Out[30]:

	type	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spo pri co
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	
5	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	
6	e	b	s	w	t	a	f	c	b	g	...	s	w	w	p	w	o	p	
7	e	b	y	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	
8	p	x	y	w	t	p	f	c	n	p	...	s	w	w	p	w	o	p	
9	e	b	s	y	t	a	f	c	b	g	...	s	w	w	p	w	o	p	
10	e	x	y	y	t	l	f	c	b	g	...	s	w	w	p	w	o	p	
11	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	
12	e	b	s	y	t	a	f	c	b	w	...	s	w	w	p	w	o	p	
13	p	x	y	w	t	p	f	c	n	k	...	s	w	w	p	w	o	p	
14	e	x	f	n	f	n	f	w	b	n	...	f	w	w	p	w	o	e	
15	e	s	f	g	f	n	f	c	n	k	...	s	w	w	p	w	o	p	
16	e	f	f	w	f	n	f	w	b	k	...	s	w	w	p	w	o	e	
17	p	x	s	n	t	p	f	c	n	n	...	s	w	w	p	w	o	p	

	type	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spor pri co
18	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	
19	e	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	
20	e	b	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	
21	p	x	y	n	t	p	f	c	n	n	...	s	w	w	p	w	o	p	
22	e	b	y	y	t	l	f	c	b	k	...	s	w	w	p	w	o	p	
23	e	b	y	w	t	a	f	c	b	w	...	s	w	w	p	w	o	p	
24	e	b	s	w	t	l	f	c	b	g	...	s	w	w	p	w	o	p	
25	p	f	s	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	
26	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	
27	e	x	y	w	t	l	f	c	b	w	...	s	w	w	p	w	o	p	
28	e	f	f	n	f	n	f	c	n	k	...	s	w	w	p	w	o	p	
29	e	x	s	y	t	a	f	w	n	n	...	s	w	w	p	w	o	p	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8094	e	b	s	g	f	n	f	w	b	g	...	s	w	w	p	w	t	p	
8095	p	x	y	c	f	m	f	c	b	y	...	y	c	c	p	w	n	n	
8096	e	k	f	w	f	n	f	w	b	w	...	s	w	w	p	w	t	p	
8097	p	k	y	n	f	s	f	c	n	b	...	k	p	p	p	w	o	e	
8098	p	k	s	e	f	y	f	c	n	b	...	k	w	p	p	w	o	e	
8099	e	k	f	w	f	n	f	w	b	w	...	k	w	w	p	w	t	p	
8100	e	f	s	n	f	n	a	c	b	o	...	s	o	o	p	n	o	p	
8101	p	k	s	e	f	s	f	c	n	b	...	s	p	w	p	w	o	e	
8102	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	
8103	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	
8104	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	
8105	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	
8106	e	k	s	n	f	n	a	c	b	o	...	s	o	o	p	o	o	p	
8107	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	
8108	p	k	y	e	f	y	f	c	n	b	...	s	p	w	p	w	o	e	
8109	e	b	s	w	f	n	f	w	b	w	...	s	w	w	p	w	t	p	
8110	e	x	s	n	f	n	a	c	b	o	...	s	o	o	p	o	o	p	
8111	e	k	s	w	f	n	f	w	b	p	...	s	w	w	p	w	t	p	
8112	e	k	s	n	f	n	a	c	b	o	...	s	o	o	p	n	o	p	
8113	p	k	y	e	f	y	f	c	n	b	...	k	p	p	p	w	o	e	
8114	p	f	y	c	f	m	a	c	b	y	...	y	c	c	p	w	n	n	
8115	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	
8116	p	k	y	n	f	s	f	c	n	b	...	k	p	w	p	w	o	e	
8117	p	k	s	e	f	y	f	c	n	b	...	s	p	w	p	w	o	e	
8118	p	k	y	n	f	f	f	c	n	b	...	s	p	w	p	w	o	e	
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	

8124 rows × 23 columns

In [31]:

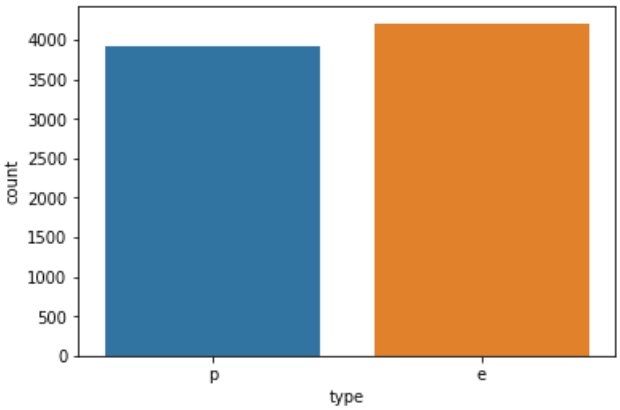
```
df_mushroom.isnull().sum()
```

Out[31]:

```
type                         0
cap-shape                    0
cap-surface                  0
cap-color                    0
bruises                      0
odor                         0
gill-attachment              0
gill-spacing                 0
gill-size                    0
gill-color                   0
stalk-shape                  0
stalk-root                   0
stalk-surface-above-ring     0
stalk-surface-below-ring     0
stalk-color-above-ring       0
stalk-color-below-ring       0
veil-type                    0
veil-color                   0
ring-number                  0
ring-type                    0
spore-print-color            0
population                   0
habitat                      0
dtype: int64
```

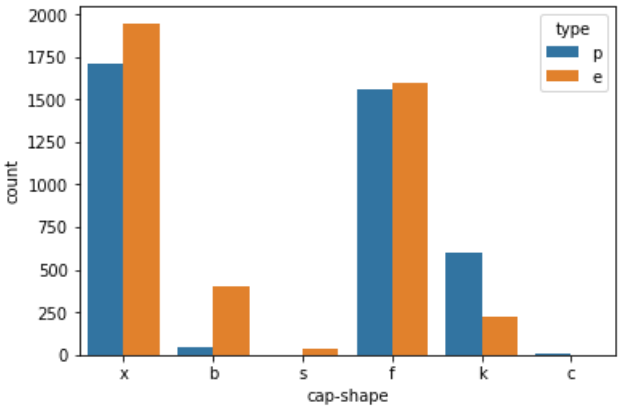
In [32]:

```
#Lets chcek the count of each type of mushroom graphically
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x="type",data=df_mushroom)
plt.show()
```



In [33]:

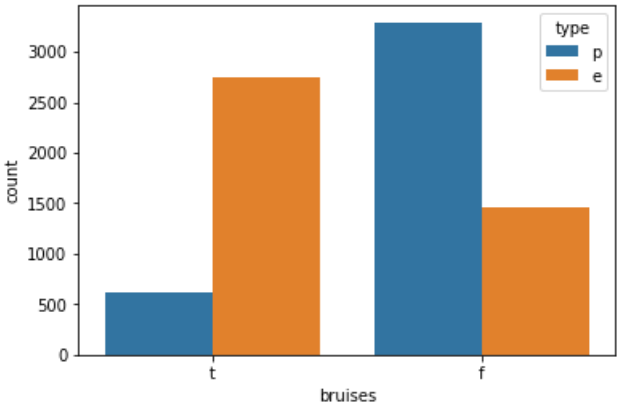
```
#Lets chcek whether the count basis of cap shape
sns.countplot(x="cap-shape",hue="type",data=df_mushroom)
plt.show()
# with cap-shaped as x and f both have alsmost equal count of poisonous and not edible mushrooms
#with cap shaped as k poisonous are more
```



In [34]:

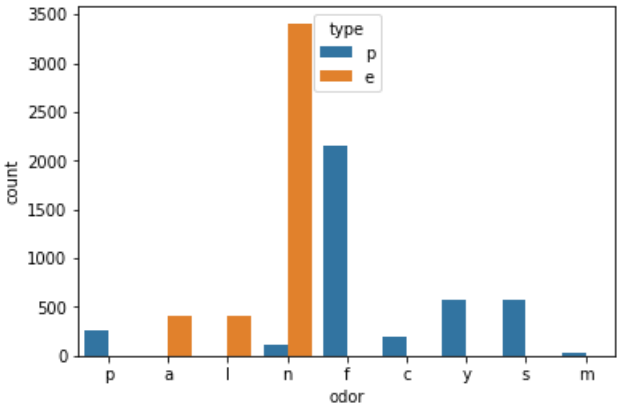
```
#lets chcek the count on the basis of bruises
sns.countplot(x="bruises",hue="type",data=df_mushroom)
plt.show()
#Here we come to know that mushrooms without bruises are in very high number poisonous
```

```
#we can say almost all the mushrooms without bruises are poisonous if we leave 500 instances
```



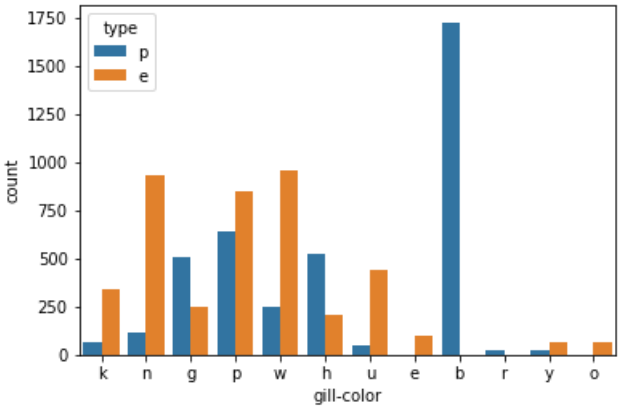
In [35]:

```
#lets chcek the count on the basis of "odor"
sns.countplot(x="odor",hue="type",data=df_mushroom)
plt.show()
#Here we come to know that edible mushrooms are basically has odor almond ,anise and no smell
#mximum number of edible mushrooms have no smell
#maximum number of poisonous mushrooms have foul smell
```



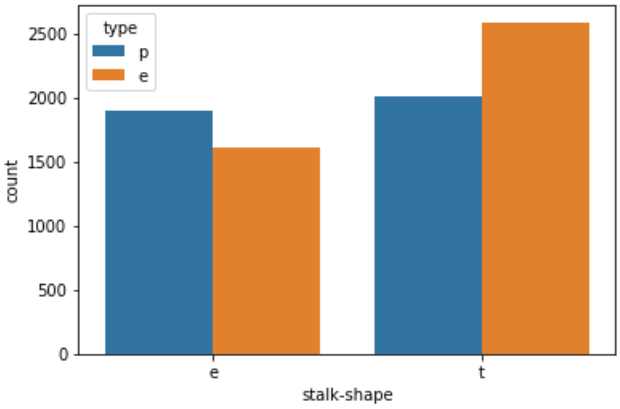
In [36]:

```
#Lets chcek the count on basis of gill-color
sns.countplot(x="gill-color",hue="type",data=df_mushroom)
plt.show()
#from here we can conclude all mushrooms with gill color buff are poisonous and they are very high numbe.
#So we should restrict ourselves before eating mushrooms havng gill color buff
```



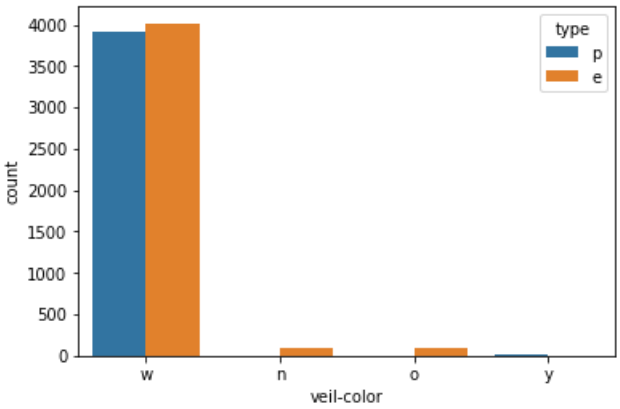
In [37]:

```
#Lets check the count on basis of stalk-shape
sns.countplot(x="stalk-shape",hue="type",data=df_mushroom)
plt.show()
```



In [38]:

```
#lets chcek the count on basis of veil-color
sns.countplot(x="veil-color",hue="type",data=df_mushroom)
plt.show()
#we cant conclude any thing from here as maximum of both types are from veil color white
#we can drop this column also as it has very less variance
```

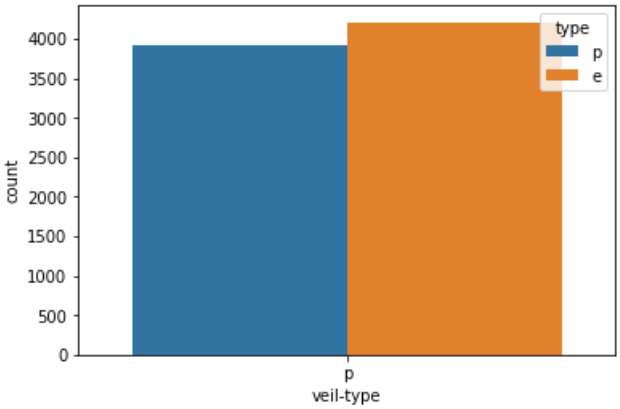


In [39]:

```
#lets drop the column veil-color as concluded from above
df_mushroom.drop(columns=["veil-color"],inplace=True)
```

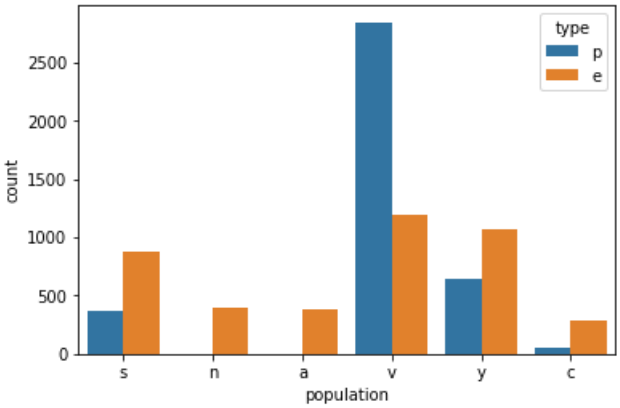
In [40]:

```
#lets chcek the count on basis of veil-type
sns.countplot(x="veil-type",hue="type",data=df_mushroom)
plt.show()
#so we can say veil type aslo does not prvide any clear distinction on its own
```



In [41]:

```
#lets chcek the counts on the basis of population
sns.countplot(x="population",hue="type",data=df_mushroom)
plt.show()
```



In [49]:

```
#Let's now divide the dataset into input and output
df_x=df_mushroom.drop(columns=["type"])
y=df_mushroom[["type"]]
```

In [50]:

```
#we will convert the input into into integers using get_dummies
df_x=pd.get_dummies(df_x,drop_first=True)
```

In [51]:

```
#lets chcek the shape
df_x.shape
```

Out[51]:

```
(8124, 92)
```

In [52]:

```
#Lets bring input dataset features to same scale
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(df_x)
x=sc.transform(df_x)
x=pd.DataFrame(x,columns=df_x.columns)
```

```
F:\anaconda3\lib\site-packages\sklearn\preprocessing\data.py:645: DataConversionWarning: Data with input
dtype uint8 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
F:\anaconda3\lib\site-packages\ipykernel_launcher.py:5: DataConversionWarning: Data with input dtype
uint8 were all converted to float64 by StandardScaler.
  """
```

In [53]:

```
#lets use labelencoder to convert target class into integers
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
le.fit(y)
y=le.transform(y)
y
```

```
F:\anaconda3\lib\site-packages\sklearn\preprocessing\label.py:219: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)
F:\anaconda3\lib\site-packages\sklearn\preprocessing\label.py:252: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for exa
mple using ravel().
  y = column_or_1d(y, warn=True)
```

Out[53]:

```
array([1, 0, 0, ..., 0, 1, 0])
```

In [62]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=55)
principalComponents = pca.fit_transform(x)
#principalDf = pd.DataFrame(data = principalComponents
#             , columns = ['principal component 1', 'principal component 2'])
```

In [63]:

```
#Lets chcek information retained after dimensionality reduction
sum(pca.explained_variance_ratio_)
#therefore we reduced the dimensions from 92 to 55
```

Out[63]:

```
0.9588343260200264
```

In [64]:

```
#We will use f1 score as the metrics as it is balanced dataset problem
#Maximum f1 score in between random states 42 to 100
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
def maxf1_score(clf,df_x,y):
    maxf=0
    rs=0
    for r_state in range(42,100):
        x_train,x_test,y_train,y_test=train_test_split(df_x, y,random_state = r_state,test_size=0.20,stra
        clf.fit(x_train,y_train)
        y_pred=clf.predict(x_test)
        tmp=f1_score(y_test,y_pred)
        print("random state :",r_state," and f1 score: ",tmp)
        if tmp>maxf:
            maxf=tmp
            rs=r_state
    print("maximum f1_score is at random state :",rs," and it is :",maxf)
```

In [65]:

```
#lets use logistic regression
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")
lg_clf=LogisticRegression()
maxf1_score(lg_clf,principalComponents,y)
```

```
random state : 42  and f1 score:  0.9987212276214833
random state : 43  and f1 score:  0.9993610223642173
random state : 44  and f1 score:  0.9980806142034548
random state : 45  and f1 score:  1.0
random state : 46  and f1 score:  1.0
random state : 47  and f1 score:  1.0
random state : 48  and f1 score:  0.9993610223642173
random state : 49  and f1 score:  1.0
random state : 50  and f1 score:  0.9993610223642173
random state : 51  and f1 score:  1.0
random state : 52  and f1 score:  1.0
random state : 53  and f1 score:  1.0
random state : 54  and f1 score:  0.9987212276214833
random state : 55  and f1 score:  0.9993610223642173
random state : 56  and f1 score:  1.0
random state : 57  and f1 score:  0.9993610223642173
random state : 58  and f1 score:  1.0
random state : 59  and f1 score:  0.9993610223642173
random state : 60  and f1 score:  0.9993610223642173
random state : 61  and f1 score:  0.9993610223642173
random state : 62  and f1 score:  0.9993610223642173
random state : 63  and f1 score:  0.9993610223642173
random state : 64  and f1 score:  1.0
random state : 65  and f1 score:  0.9993610223642173
random state : 66  and f1 score:  1.0
random state : 67  and f1 score:  0.9987212276214833
random state : 68  and f1 score:  1.0
random state : 69  and f1 score:  1.0
random state : 70  and f1 score:  0.9993610223642173
random state : 71  and f1 score:  1.0
random state : 72  and f1 score:  0.9993610223642173
random state : 73  and f1 score:  0.9993610223642173
random state : 74  and f1 score:  0.9993610223642173
random state : 75  and f1 score:  1.0
random state : 76  and f1 score:  0.9993610223642173
random state : 77  and f1 score:  1.0
random state : 78  and f1 score:  1.0
random state : 79  and f1 score:  1.0
random state : 80  and f1 score:  1.0
random state : 81  and f1 score:  0.9993610223642173
random state : 82  and f1 score:  1.0
random state : 83  and f1 score:  1.0
random state : 84  and f1 score:  1.0
random state : 85  and f1 score:  1.0
random state : 86  and f1 score:  0.9993610223642173
random state : 87  and f1 score:  1.0
random state : 88  and f1 score:  1.0
random state : 89  and f1 score:  0.9987212276214833
random state : 90  and f1 score:  0.9993610223642173
random state : 91  and f1 score:  1.0
random state : 92  and f1 score:  0.9993610223642173
random state : 93  and f1 score:  1.0
random state : 94  and f1 score:  0.9993610223642173
random state : 95  and f1 score:  0.9993610223642173
random state : 96  and f1 score:  0.9993610223642173
random state : 97  and f1 score:  0.9993610223642173
random state : 98  and f1 score:  0.9993610223642173
random state : 99  and f1 score:  0.9993610223642173
maximum f1_score is at random state : 45  and it is : 1.0
```

In [66]:

```
#Lets use cross_val_score and evaluate the logistic regression model
from sklearn.model_selection import cross_val_score
print("Mean f1 score for logistic classifier: ",cross_val_score(lg_clf,principalComponents,y,cv=5,scoring
print("standard deviation in f1 score for logistic classifier: ",cross_val_score(lg_clf,principalComponer
print(cross_val_score(lg_clf,principalComponents,y,cv=5,scoring="f1"))
```

```
Mean f1 score for logistic classifier:  0.937253953921053
standard deviation in f1 score for logistic classifier:  0.0915546470934307
[0.92465753 0.99936102 0.99872123 1.         0.76352999]
```

In [67]:

```
#Lets use KNN
#For KNN we need to know the best value of k using grid search
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings("ignore")
```

```
kc=KNeighborsClassifier()
neighbors={"n_neighbors":range(1,30)}
clf = GridSearchCV(kc, neighbors, cv=5,scoring="f1")
clf.fit(principalComponents,y)
clf.best_params_
```

Out[67]:

```
{'n_neighbors': 2}
```

In [68]:

```
kc=KNeighborsClassifier(n_neighbors=2)
maxf1_score(kc,principalComponents,y)
```

```
random state : 42  and f1 score:  0.9987212276214833
random state : 43  and f1 score:  1.0
random state : 44  and f1 score:  0.9987212276214833
random state : 45  and f1 score:  1.0
random state : 46  and f1 score:  1.0
random state : 47  and f1 score:  1.0
random state : 48  and f1 score:  1.0
random state : 49  and f1 score:  1.0
random state : 50  and f1 score:  1.0
random state : 51  and f1 score:  1.0
random state : 52  and f1 score:  1.0
random state : 53  and f1 score:  1.0
random state : 54  and f1 score:  0.9987212276214833
random state : 55  and f1 score:  1.0
random state : 56  and f1 score:  1.0
random state : 57  and f1 score:  1.0
random state : 58  and f1 score:  1.0
random state : 59  and f1 score:  1.0
random state : 60  and f1 score:  1.0
random state : 61  and f1 score:  1.0
random state : 62  and f1 score:  1.0
random state : 63  and f1 score:  1.0
random state : 64  and f1 score:  1.0
random state : 65  and f1 score:  1.0
random state : 66  and f1 score:  1.0
random state : 67  and f1 score:  0.9987212276214833
random state : 68  and f1 score:  1.0
random state : 69  and f1 score:  1.0
random state : 70  and f1 score:  1.0
random state : 71  and f1 score:  1.0
random state : 72  and f1 score:  1.0
random state : 73  and f1 score:  1.0
random state : 74  and f1 score:  1.0
random state : 75  and f1 score:  1.0
random state : 76  and f1 score:  1.0
random state : 77  and f1 score:  1.0
random state : 78  and f1 score:  1.0
random state : 79  and f1 score:  1.0
random state : 80  and f1 score:  1.0
random state : 81  and f1 score:  1.0
random state : 82  and f1 score:  1.0
random state : 83  and f1 score:  1.0
random state : 84  and f1 score:  1.0
random state : 85  and f1 score:  1.0
random state : 86  and f1 score:  1.0
random state : 87  and f1 score:  1.0
random state : 88  and f1 score:  1.0
random state : 89  and f1 score:  0.9987212276214833
random state : 90  and f1 score:  1.0
random state : 91  and f1 score:  1.0
random state : 92  and f1 score:  1.0
random state : 93  and f1 score:  1.0
random state : 94  and f1 score:  1.0
random state : 95  and f1 score:  1.0
random state : 96  and f1 score:  1.0
random state : 97  and f1 score:  1.0
random state : 98  and f1 score:  1.0
random state : 99  and f1 score:  1.0
maximum f1_score is at random state : 43  and it is : 1.0
```

In [69]:

```
#Lets use cross_val_score and evaluate the knn model
from sklearn.model_selection import cross_val_score
print("Mean f1 score for knn classifier: ",cross_val_score(kc,principalComponents,y,cv=5,scoring="f1").me
print("standard deviation in f1 score for knn classifier: ",cross_val_score(kc,principalComponents,y,cv=5
print(cross_val_score(kc,principalComponents,y,cv=5,scoring="f1"))
```

```
Mean f1 score for knn classifier:  0.9299924525874376
standard deviation in f1 score for knn classifier:  0.08513700975186633
[0.80941534 1.         0.99679693 1.         0.84375   ]
```

In [70]:

```
#lets use decision tree
from sklearn.tree import DecisionTreeClassifier
dc=DecisionTreeClassifier()
maxf1_score(dc,principalComponents,y)
```

```
random state : 42  and f1 score:  0.9968132568514977
random state : 43  and f1 score:  0.9942565411614551
random state : 44  and f1 score:  0.9987212276214833
random state : 45  and f1 score:  0.9942638623326959
random state : 46  and f1 score:  0.9968132568514977
random state : 47  and f1 score:  0.9980879541108987
random state : 48  and f1 score:  0.9987228607918263
random state : 49  and f1 score:  0.9987244897959183
random state : 50  and f1 score:  0.9974457215836526
random state : 51  and f1 score:  1.0
random state : 52  and f1 score:  0.9968051118210861
random state : 53  and f1 score:  0.9968091895341417
random state : 54  and f1 score:  0.9961783439490446
random state : 55  and f1 score:  0.9987228607918263
random state : 56  and f1 score:  0.998085513720485
random state : 57  and f1 score:  0.996173469387755
random state : 58  and f1 score:  0.998085513720485
random state : 59  and f1 score:  0.9974489795918366
random state : 60  and f1 score:  0.9993618379068284
random state : 61  and f1 score:  0.9961685823754789
random state : 62  and f1 score:  0.9961538461538463
random state : 63  and f1 score:  0.9968132568514977
random state : 64  and f1 score:  0.998085513720485
random state : 65  and f1 score:  0.9955156950672646
random state : 66  and f1 score:  0.9968010236724248
random state : 67  and f1 score:  0.9974457215836526
random state : 68  and f1 score:  0.9968132568514977
random state : 69  and f1 score:  0.9987228607918263
random state : 70  and f1 score:  0.9948914431673053
random state : 71  and f1 score:  0.9968091895341417
random state : 72  and f1 score:  0.9974489795918366
random state : 73  and f1 score:  0.9980830670926518
random state : 74  and f1 score:  0.9980879541108987
random state : 75  and f1 score:  0.998085513720485
random state : 76  and f1 score:  0.9993618379068284
random state : 77  and f1 score:  0.9936143039591315
random state : 78  and f1 score:  0.9955328653477983
random state : 79  and f1 score:  0.9968051118210861
random state : 80  and f1 score:  0.9955442393380012
random state : 81  and f1 score:  0.9968051118210861
random state : 82  and f1 score:  0.9948783610755442
random state : 83  and f1 score:  0.9955271565495207
random state : 84  and f1 score:  0.9987228607918263
random state : 85  and f1 score:  0.9974424552429666
random state : 86  and f1 score:  0.9987244897959183
random state : 87  and f1 score:  0.9993610223642173
random state : 88  and f1 score:  0.9968010236724248
random state : 89  and f1 score:  0.9974457215836526
random state : 90  and f1 score:  0.996173469387755
random state : 91  and f1 score:  0.9974489795918366
random state : 92  and f1 score:  0.9961685823754789
random state : 93  and f1 score:  0.9968091895341417
random state : 94  and f1 score:  0.9942784488239034
random state : 95  and f1 score:  0.998085513720485
random state : 96  and f1 score:  0.998085513720485
random state : 97  and f1 score:  0.9955442393380012
random state : 98  and f1 score:  0.998085513720485
random state : 99  and f1 score:  0.9993610223642173
maximum f1_score is at random state : 51  and it is : 1.0
```

In [72]:

```
#Lets use cross_val_score and evaluate the knn model
from sklearn.model_selection import cross_val_score
print("Mean f1 score for decision tree classifier: ",cross_val_score(dc,principalComponents,y,cv=5,scorin
print("standard deviation in f1 score for decision tree classifier: ",cross_val_score(dc,principalCompone
print(cross_val_score(dc,principalComponents,y,cv=5,scoring="f1"))
```

```
Mean f1 score for decision tree classifier:  0.9328057503410824
standard deviation in f1 score for decision tree classifier:  0.07171242171488959
[0.80485953 0.99745223 0.95744681 0.99808795 0.90690691]
```

In [73]:

```
#lets use ensemble calssifier such as random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
parameters={"n_estimators":[10,100,500]}
rf_clf=RandomForestClassifier()
clf = GridSearchCV(rf_clf, parameters, cv=5,scoring="f1")
clf.fit(principalComponents,y)
clf.best_params_
```

Out[73]:

```
{'n_estimators': 10}
```

In [74]:

```
rf_clf=RandomForestClassifier(n_estimators=10)
maxf1_score(rf_clf,principalComponents,y)
```

```
random state : 42  and f1 score:  0.9993610223642173
random state : 43  and f1 score:  1.0
random state : 44  and f1 score:  0.9993610223642173
random state : 45  and f1 score:  0.9987212276214833
random state : 46  and f1 score:  1.0
random state : 47  and f1 score:  1.0
random state : 48  and f1 score:  0.9993610223642173
random state : 49  and f1 score:  1.0
random state : 50  and f1 score:  1.0
random state : 51  and f1 score:  1.0
random state : 52  and f1 score:  1.0
random state : 53  and f1 score:  1.0
random state : 54  and f1 score:  1.0
random state : 55  and f1 score:  0.9993610223642173
random state : 56  and f1 score:  1.0
random state : 57  and f1 score:  1.0
random state : 58  and f1 score:  0.9980806142034548
random state : 59  and f1 score:  1.0
random state : 60  and f1 score:  0.9993610223642173
random state : 61  and f1 score:  0.9993610223642173
random state : 62  and f1 score:  1.0
random state : 63  and f1 score:  1.0
random state : 64  and f1 score:  1.0
random state : 65  and f1 score:  0.9993610223642173
random state : 66  and f1 score:  0.9993610223642173
random state : 67  and f1 score:  0.9987212276214833
random state : 68  and f1 score:  1.0
random state : 69  and f1 score:  1.0
random state : 70  and f1 score:  0.9993610223642173
random state : 71  and f1 score:  1.0
random state : 72  and f1 score:  0.9993610223642173
random state : 73  and f1 score:  0.9993610223642173
random state : 74  and f1 score:  1.0
random state : 75  and f1 score:  1.0
random state : 76  and f1 score:  0.9993610223642173
random state : 77  and f1 score:  1.0
random state : 78  and f1 score:  1.0
random state : 79  and f1 score:  1.0
random state : 80  and f1 score:  1.0
random state : 81  and f1 score:  0.9974391805377721
random state : 82  and f1 score:  1.0
random state : 83  and f1 score:  0.9993610223642173
random state : 84  and f1 score:  1.0
random state : 85  and f1 score:  0.9987212276214833
random state : 86  and f1 score:  1.0
random state : 87  and f1 score:  1.0
random state : 88  and f1 score:  1.0
random state : 89  and f1 score:  0.9987212276214833
random state : 90  and f1 score:  0.9993610223642173
random state : 91  and f1 score:  1.0
random state : 92  and f1 score:  1.0
random state : 93  and f1 score:  1.0
random state : 94  and f1 score:  1.0
random state : 95  and f1 score:  1.0
random state : 96  and f1 score:  0.9993610223642173
random state : 97  and f1 score:  0.9987212276214833
random state : 98  and f1 score:  1.0
random state : 99  and f1 score:  1.0
maximum f1_score is at random state : 43  and it is : 1.0
```

In [77]:

```
#lets again use cross val score
print("Mean f1 score for random forest classifier: ",cross_val_score(rf_clf,principalComponents,y,cv=5,sc
print("standard deviation in f1 score for random forest classifier: ",cross_val_score(rf_clf,principalCom
print(cross_val_score(rf_clf,principalComponents,y,cv=5,scoring="f1"))
```

```
Mean f1 score for random forest classifier:  0.9244396437256592
standard deviation in f1 score for random forest classifier:  0.10768805513725976
[0.78200155 0.99936102 0.98771816 1.         0.81477627]
```

In [78]:

```
#Lets use SVM
from sklearn.svm import SVC
svc=SVC()
parameters={"kernel":["linear", "poly", "rbf"],"C":[0.001,0.01,0.1,1,10]}
clf = GridSearchCV(svc, parameters, cv=5,scoring="f1")
clf.fit(principalComponents,y)
clf.best_params_
```

Out[78]:

```
{'C': 0.1, 'kernel': 'poly'}
```

In [79]:

```
svc=SVC(kernel="poly",C=0.1)
maxf1_score(svc,principalComponents,y)
```

```
random state : 42  and f1 score:  0.9987212276214833
random state : 43  and f1 score:  0.9993610223642173
random state : 44  and f1 score:  0.9980806142034548
random state : 45  and f1 score:  1.0
random state : 46  and f1 score:  1.0
random state : 47  and f1 score:  1.0
random state : 48  and f1 score:  0.9993610223642173
random state : 49  and f1 score:  1.0
random state : 50  and f1 score:  0.9993610223642173
random state : 51  and f1 score:  1.0
random state : 52  and f1 score:  1.0
random state : 53  and f1 score:  1.0
random state : 54  and f1 score:  0.9987212276214833
random state : 55  and f1 score:  0.9993610223642173
random state : 56  and f1 score:  1.0
random state : 57  and f1 score:  0.9993610223642173
random state : 58  and f1 score:  0.9993610223642173
random state : 59  and f1 score:  0.9993610223642173
random state : 60  and f1 score:  0.9993610223642173
random state : 61  and f1 score:  0.9993610223642173
random state : 62  and f1 score:  0.9987212276214833
random state : 63  and f1 score:  0.9993610223642173
random state : 64  and f1 score:  1.0
random state : 65  and f1 score:  0.9987212276214833
random state : 66  and f1 score:  0.9993610223642173
random state : 67  and f1 score:  0.9987212276214833
random state : 68  and f1 score:  1.0
random state : 69  and f1 score:  1.0
random state : 70  and f1 score:  0.9987212276214833
random state : 71  and f1 score:  1.0
random state : 72  and f1 score:  0.9987212276214833
random state : 73  and f1 score:  0.9993610223642173
random state : 74  and f1 score:  0.9993610223642173
random state : 75  and f1 score:  1.0
random state : 76  and f1 score:  0.9993610223642173
random state : 77  and f1 score:  1.0
random state : 78  and f1 score:  1.0
random state : 79  and f1 score:  1.0
random state : 80  and f1 score:  1.0
random state : 81  and f1 score:  0.9974391805377721
random state : 82  and f1 score:  1.0
random state : 83  and f1 score:  0.9993610223642173
random state : 84  and f1 score:  1.0
random state : 85  and f1 score:  0.9993610223642173
random state : 86  and f1 score:  0.9993610223642173
random state : 87  and f1 score:  0.9993610223642173
random state : 88  and f1 score:  1.0
random state : 89  and f1 score:  0.9987212276214833
random state : 90  and f1 score:  0.9993610223642173
random state : 91  and f1 score:  1.0
random state : 92  and f1 score:  0.9987212276214833
random state : 93  and f1 score:  1.0
random state : 94  and f1 score:  0.9993610223642173
random state : 95  and f1 score:  0.9993610223642173
random state : 96  and f1 score:  0.9993610223642173
random state : 97  and f1 score:  0.9987212276214833
random state : 98  and f1 score:  0.9993610223642173
random state : 99  and f1 score:  0.9993610223642173
maximum f1_score is at random state : 45  and it is : 1.0
```

In [81]:

```
#lets again use cross val score
print("Mean f1 score for SVM classifier: ",cross_val_score(svc,principalComponents,y,cv=5,scoring="f1").m
print("standard deviation in f1 score for SVM classifier: ",cross_val_score(svc,principalComponents,y,cv=
print(cross_val_score(svc,principalComponents,y,cv=5,scoring="f1"))
```

```
Mean f1 score for SVM classifier:  0.9416779210445139
standard deviation in f1 score for SVM classifier:  0.07164626466068527
[0.80487805 0.99936102 0.95390782 1.         0.95024272]
```

In [84]:

```
#comparitively SVM is performing better so lets use svm
```

```
#random state 45
svc=SVC(kernel="poly",C=0.1)
x_train,x_test,y_train,y_test=train_test_split(principalComponents, y,random_state = 45,test_size=0.20,st
svc.fit(x_train,y_train)
y_pred=svc.predict(x_test)
```

In [85]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
print("Confusion matrix \n",confusion_matrix(y_test,y_pred))
print("f1 score is : ",f1_score(y_test,y_pred))
print("classification report \n",classification_report(y_test,y_pred))
print("AUC ROC Score: ",roc_auc_score(y_test,y_pred))
```

```
Confusion matrix
 [[842   0]
 [  0 783]]
f1 score is :  1.0
classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       842
           1       1.00      1.00      1.00       783

   micro avg       1.00      1.00      1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625

AUC ROC Score:  1.0
```