The dataset is used to predict whether a person has a chance to get stroke or not using the inputs such as age, gender, work type, residential type, smoking status etc.

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report,ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier
```

Read dataset

```
df=pd.read_csv('/content/drive/MyDrive/dataset/full_data.csv')
df
```

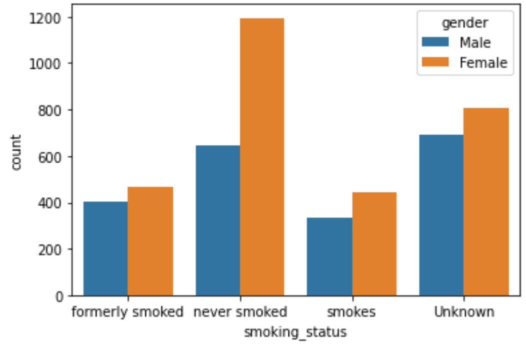
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
0	Male	67.0	0	1	Yes	Private	Urban	
2	Female	49.0	0	0	Yes	Private	Urban	

Learning dataset

Smoking status in males and females

```
sns.countplot(x='smoking_status',data=df,hue='gender')
```

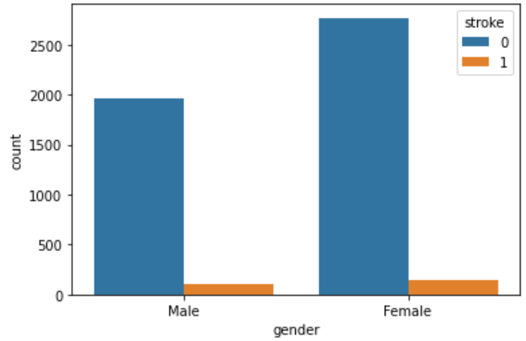
```
<Axes: xlabel='smoking_status', ylabel='count'>
```



Count of patients with respect to gender

```
sns.countplot(x='gender',data=df,hue='stroke')
```

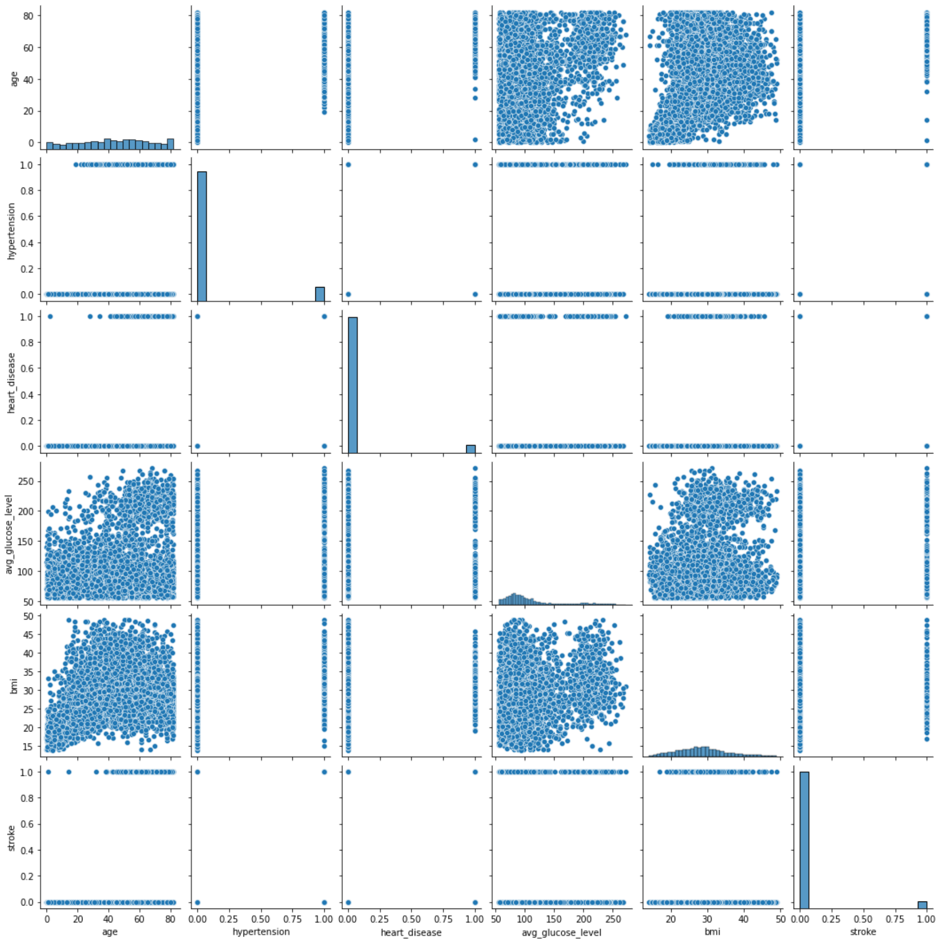
```
<Axes: xlabel='gender', ylabel='count'>
```



Pair relationship

```
sns.pairplot(df)
```

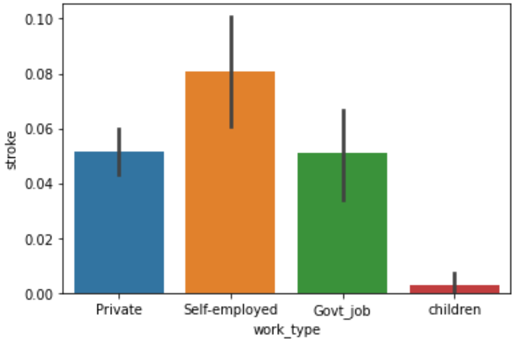
<seaborn.axisgrid.PairGrid at 0x7f3bb5f3b520>



Work type dependency on stroke

```
sns.barplot(x='work_type',y='stroke',data=df)
```

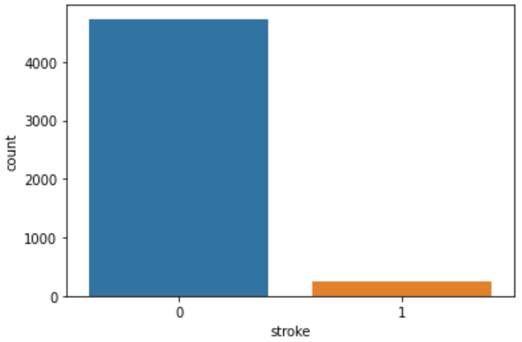
```
<Axes: xlabel='work_type', ylabel='stroke'>
```



Count of stroke patients in the dataset

```
sns.countplot(x='stroke',data=df)
```

```
<Axes: xlabel='stroke', ylabel='count'>
```



Checking datatypes

```
df.dtypes
```

```
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
```

Checking null values

```
df.isna().sum()
```

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

Changing datatype 'object'

```
le=LabelEncoder()
ls=['gender','ever_married','work_type','Residence_type','smoking_status']
for i in ls:
  df[i]=le.fit_transform(df[i])
```

```
df.dtypes
```

```
gender                 int64
age                  float64
hypertension           int64
heart_disease          int64
ever_married           int64
work_type              int64
Residence_type         int64
avg_glucose_level    float64
bmi                  float64
smoking_status         int64
stroke                 int64
dtype: object
```

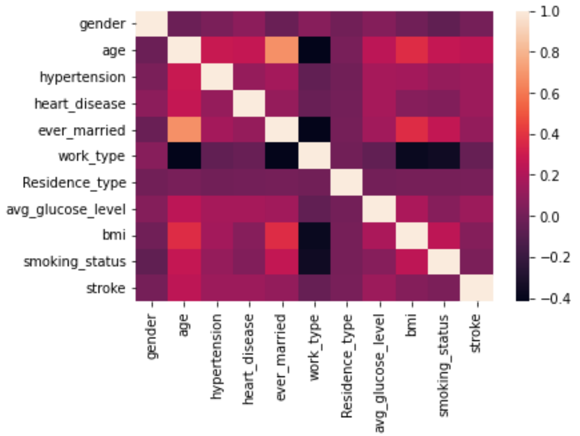
Checking correlation

```
df.corr()
```

	gender	age	hypertension	heart_disease	ever_married	work_type
gender	1.000000	-0.026538	0.021485	0.086476	-0.028971	0.065784
age	-0.026538	1.000000	0.278120	0.264852	0.677137	-0.415935
hypertension	0.021485	0.278120	1.000000	0.111974	0.164534	-0.061618
heart_disease	0.086476	0.264852	0.111974	1.000000	0.114765	-0.036943
ever_married	-0.028971	0.677137	0.164534	0.114765	1.000000	-0.406439
work_type	0.065784	-0.415935	-0.061618	-0.036943	-0.406439	1.000000
Residence_type	-0.004301	0.017155	-0.004755	0.002125	0.008191	-0.003524
avg_glucose_level	0.055796	0.236763	0.170028	0.166847	0.150724	-0.059658
bmi	-0.012093	0.373703	0.158762	0.060926	0.371690	-0.382418

```
sns.heatmap(df.corr())
```

```
<Axes: >
```



Seperating input and output

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

Scaling using Minmax Scaler

```
min=MinMaxScaler()
X=min.fit_transform(X)
```

Seperating training and testing dataset

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

K Nearest Neighbour classification

```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
y_pred
```

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

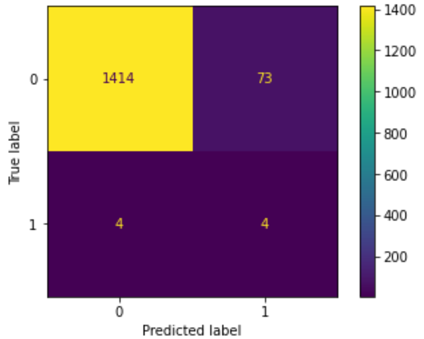
```
A1=accuracy_score(y_pred,y_test)
print(A1)
print(classification_report(y_pred,y_test))
print(ConfusionMatrixDisplay.from_predictions(y_pred,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1487
           1       0.05      0.50      0.09         8

    accuracy                           0.95      1495
   macro avg       0.52      0.73      0.53      1495
weighted avg       0.99      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb53261f0>
```



Using oversampling in knn

```
oversample=SMOTE(random_state=1)
X_os,y_os=oversample.fit_resample(X,y)
y_os.value_counts()
```

```
1    4733
0    4733
Name: stroke, dtype: int64
```

```
print(Counter(y_os))
```

```
Counter({1: 4733, 0: 4733})
```

```
X_trainos,X_testos,y_trainos,y_testos=train_test_split(X_os,y_os,test_size=0.2,random_state=1)
```

```
knnos=KNeighborsClassifier(n_neighbors=5)
knnos.fit(X_trainos,y_trainos)
y_predos=knnos.predict(X_testos)
y_predos
```

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

```
print(classification_report(y_testos,y_predos))
```

```
              precision    recall  f1-score   support

           0       0.98      0.81      0.89       958
           1       0.84      0.99      0.90       936

    accuracy                           0.90      1894
   macro avg       0.91      0.90      0.90      1894
weighted avg       0.91      0.90      0.90      1894
```

Accuracy reduced after oversampling

Support Vector Machine Algorithm

```
svm=SVC(kernel='poly')
svm.fit(X_train,y_train)
ypred2=svm.predict(X_test)
ypred2
```

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

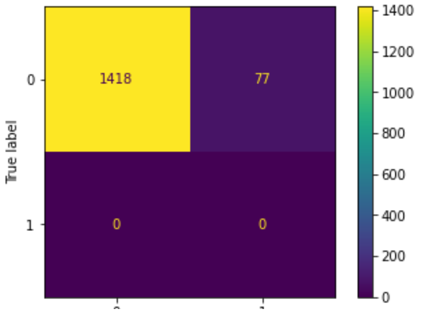
```
A2=accuracy_score(ypred2,y_test)
print(A2)
print(classification_report(ypred2,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred2,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1495
           1       0.00      0.00      0.00         0

    accuracy                           0.95      1495
   macro avg       0.50      0.47      0.49      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb5ea2f10>
```



Naive Bayes Algorithm

```
nb=GaussianNB()
nb.fit(X_train,y_train)
ypred3=nb.predict(X_test)
ypred3
```

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

```
A3=accuracy_score(ypred3,y_test)
print(A3)
print(classification_report(ypred3,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred3,y_test))
```

```
0.8655518394648829
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      1305
           1       0.43      0.17      0.25       190

    accuracy                           0.87      1495
   macro avg       0.66      0.57      0.59      1495
weighted avg       0.83      0.87      0.84      1495
```

Random Forest Classifier

```
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
ypred4=rf.predict(X_test)
ypred4
```

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

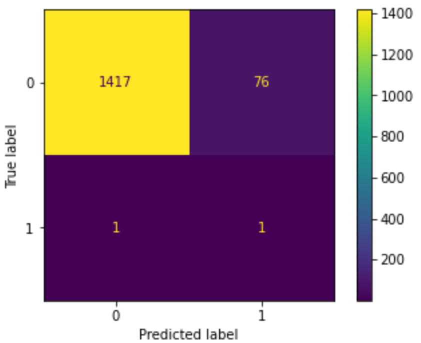
```
A4=accuracy_score(ypred4,y_test)
print(A4)
print(classification_report(ypred4,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred4,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1493
           1       0.01      0.50      0.03         2

    accuracy                           0.95      1495
   macro avg       0.51      0.72      0.50      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb51ac9d0>
```



Logistic Regression

```
lr=LogisticRegression()
lr.fit(X_train,y_train)
ypred5=lr.predict(X_test)
ypred5
```

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

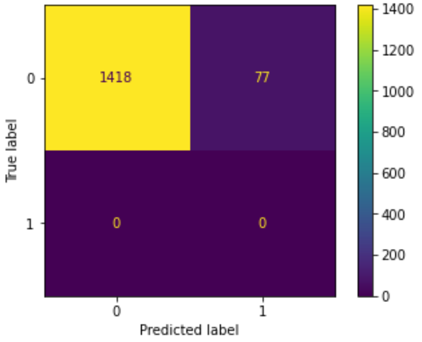
```
A5=accuracy_score(ypred5,y_test)
print(A5)
print(classification_report(ypred5,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred5,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1495
           1       0.00      0.00      0.00         0

    accuracy                           0.95      1495
   macro avg       0.50      0.47      0.49      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb537ce80>
```



Adaboost Classifier

```
abc=AdaBoostClassifier()
abc.fit(X_train,y_train)
ypred6=abc.predict(X_test)
ypred6
```

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

```
A6=accuracy_score(ypred6,y_test)
print(A6)
```

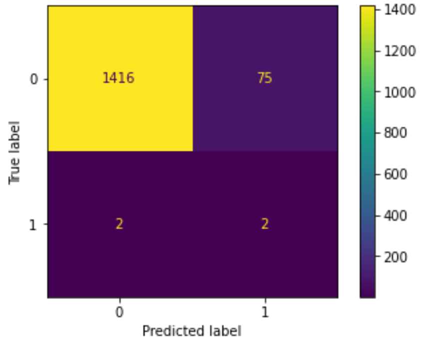
```
print(classification_report(ypred6,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred6,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1491
           1       0.03      0.50      0.05         4

    accuracy                           0.95      1495
   macro avg       0.51      0.72      0.51      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb70d1a30>
```



Extreme Gradient Booster Classifier

```
xgb=XGBClassifier()
xgb.fit(X_train,y_train)
ypred7=xgb.predict(X_test)
ypred7
```

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

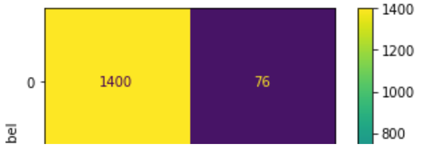
```
A7=accuracy_score(ypred7,y_test)
print(A7)
print(classification_report(ypred7,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred7,y_test))
```

```
0.9371237745819398
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1476
           1       0.01      0.05      0.02        19

    accuracy                           0.94      1495
   macro avg       0.50      0.50      0.49      1495
weighted avg       0.97      0.94      0.96      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb5069a90>
```



Decision Tree Classifier

```
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
ypred8=dt.predict(X_test)
ypred8
```

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

```
A8=accuracy_score(ypred8,y_test)
print(A8)
print(classification_report(ypred8,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred8,y_test))
```

```
0.9130434782608695
              precision    recall  f1-score   support
```

Creating dataframe to check accuracy

```
eval=pd.DataFrame({'model':['K Nearest Neighbour','Support Vector Machine','Naive Bayes','Random Forest','Logistic Regression','Adaboost','Extreme G
eval
```

	model	Accuracy
0	K Nearest Neighbour	94.849498
1	Support Vector Machine	94.849498
2	Naive Bayes	86.555184
3	Random Forest	94.849498
4	Logistic Regression	94.849498
5	Adaboost	94.849498
6	Extreme Gradient Boost	93.712375
7	Decision Tree	91.304348

Naive Bayes algorithm has lowest accuracy.

Checking with oversampling

```
oversample=SMOTE(random_state=1)
X_os1,y_os1=oversample.fit_resample(X,y)
y_os1.value_counts()
```

```
1    4733
0    4733
Name: stroke, dtype: int64
```

```
print(Counter(y_os1))
```

```
Counter({1: 4733, 0: 4733})
```

```
X_trainos1,X_testos1,y_trainos1,y_testos1=train_test_split(X_os1,y_os1,test_size=0.2,random_state=1)
```

```
nbos=GaussianNB()
nbos.fit(X_trainos1,y_trainos1)
y_predos1=nbos.predict(X_testos1)
y_predos1
```

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

```
print(accuracy_score(y_testos,y_predos))
print(classification_report(y_testos,y_predos))
```

```
0.8970432946145723
              precision    recall  f1-score   support

           0       0.98      0.81      0.89       958
           1       0.84      0.99      0.90       936

    accuracy                           0.90      1894
   macro avg       0.91      0.90      0.90      1894
weighted avg       0.91      0.90      0.90      1894
```

Accuracy increased to 89.7 from 86.5.

Checking for second lowest accuracy with Decision tree classifier with oversampling.

```
oversample=SMOTE(random_state=1)
X_os2,y_os2=oversample.fit_resample(X,y)
y_os2.value_counts()
```

```
1    4733
0    4733
Name: stroke, dtype: int64
```

```
print(Counter(y_os2))
```

```
Counter({1: 4733, 0: 4733})
```

```
X_trainos2,X_testos2,y_trainos2,y_testos2=train_test_split(X_os2,y_os2,test_size=0.2,random_state=1)
```

```
dtos=DecisionTreeClassifier()
dtos.fit(X_trainos,y_trainos)
y_predos2=dtos.predict(X_testos2)
y_predos2
```

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

```
print(classification_report(y_testos,y_predos2))
print(accuracy_score(y_testos,y_predos2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90       958
           1       0.89      0.90      0.90       936
```

```
    accuracy                           0.90      1894
   macro avg       0.90      0.90      0.90      1894
weighted avg       0.90      0.90      0.90      1894

0.8975712777191129
```

Accuray did not increase with oversampling.

Checking for the changes with stacking model.

Stacking model 1- by stacking low accuracy models in the set

```
estimators=[('dt',DecisionTreeClassifier()),('nb',GaussianNB())]
sc=StackingClassifier(estimators=estimators,final_estimator=XGBClassifier())
sc.fit(X_train,y_train)
ypred9=sc.predict(X_test)
```

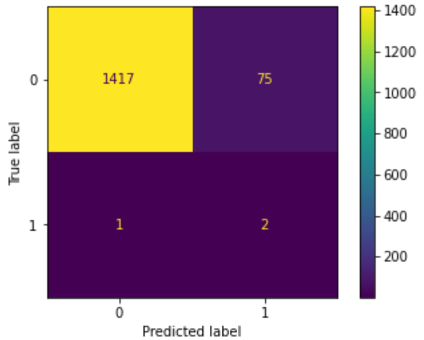
```
A9=accuracy_score(ypred9,y_test)
print(A9)
print(classification_report(ypred9,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred9,y_test))
```

```
0.9491638795986622
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1492
           1       0.03      0.67      0.05         3

    accuracy                           0.95      1495
   macro avg       0.51      0.81      0.51      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb6fb0bb0>
```



Stacking model 2- by stacking high accuracy models

```
estimators=[('knn',KNeighborsClassifier()),('svm',SVC()),('rf',RandomForestClassifier())]
sc1=StackingClassifier(estimators=estimators,final_estimator=LogisticRegression())
sc1.fit(X_train,y_train)
ypred10=sc1.predict(X_test)
```

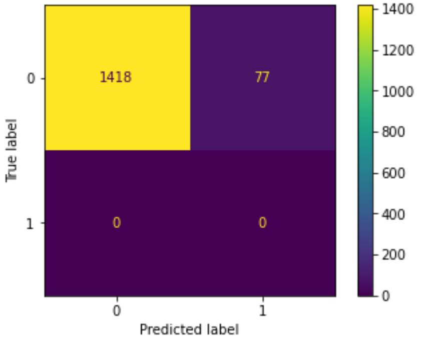
```
A10=accuracy_score(ypred10,y_test)
print(A10)
print(classification_report(ypred10,y_test))
print(ConfusionMatrixDisplay.from_predictions(ypred10,y_test))
```

```
0.948494983277592
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1495
           1       0.00      0.00      0.00         0

    accuracy                           0.95      1495
   macro avg       0.50      0.47      0.49      1495
weighted avg       1.00      0.95      0.97      1495

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3bb70a3af0>
```



Accuracy remained same for both stacking models

Checking Accuracy for eeach model

```
eval=pd.DataFrame({'model':['K Nearest Neighbour','Support Vector Machine','Naive Bayes','Random Forest','Logistic Regression','Adaboost','Extreme G
eval
```

	model	Accuracy
0	K Nearest Neighbour	94.849498
1	Support Vector Machine	94.849498
2	Naive Bayes	86.555184
3	Random Forest	94.849498
4	Logistic Regression	94.849498
5	Adaboost	94.849498

Plotting Accuracy

```
x=['knn','svm','nb','rf','lr','ab','xgb','dt','sc1','sc2']
y=[A1*100,A2*100,A3*100,A4*100,A5*100,A6*100,A7*100,A8*100,A9*100,A10*100]
plt.plot(x,y)
plt.xlabel('names')
plt.ylabel('results')
plt.title('Accuracy variation')
plt.show()
```