NAME: PASALA NEELIMA

REGNO : 24MDT1064

EXPERIMENT NO : 03, 04

Task : handling missing values

and visualization. And grid &

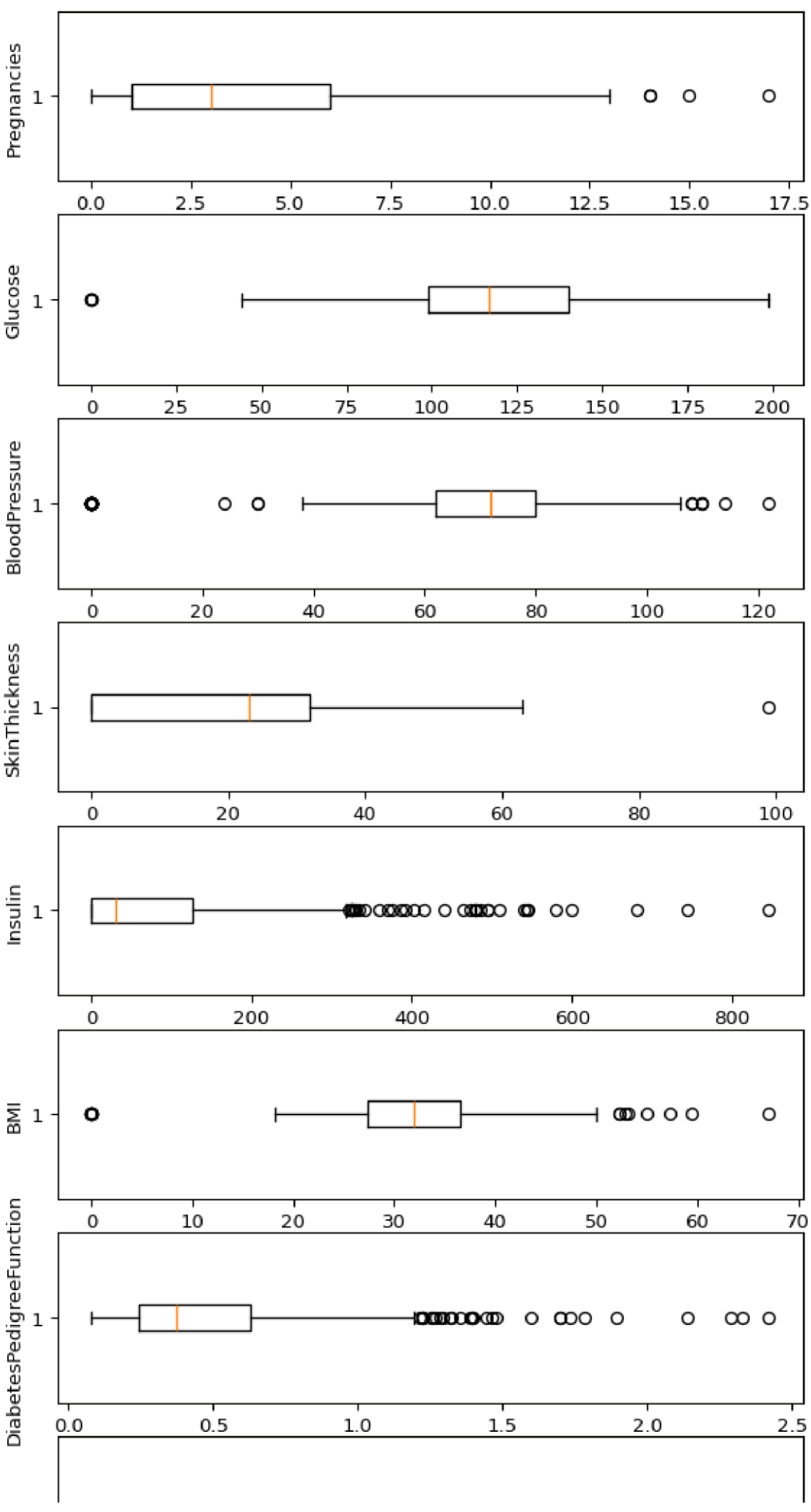
random search.

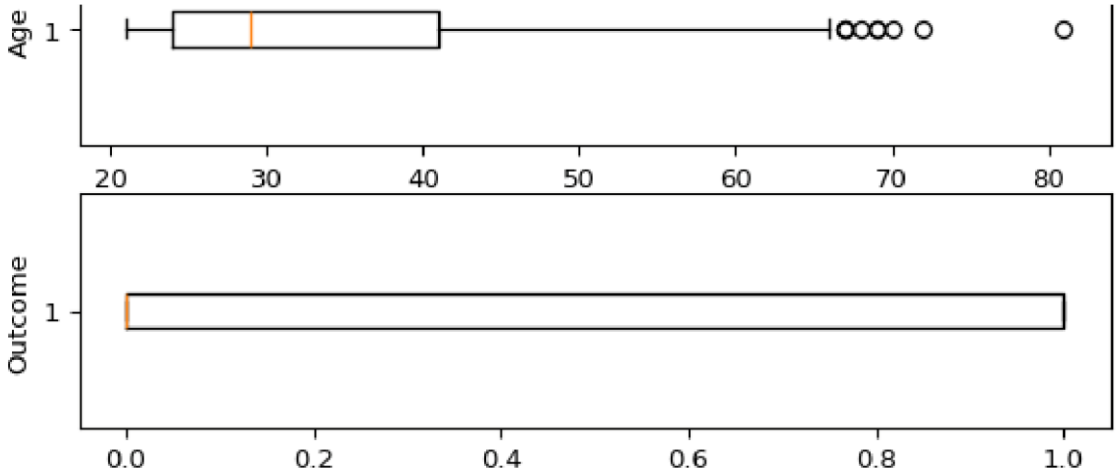
```
import pandas as pd import
scipy import numpy as np
import seaborn as sns import
matplotlib.pyplot as plt
```

```
df = pd.read_csv("/content/diabetes.csv")
print(df.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
# BOX PLOT fig,axs =
plt.subplots(9,1,dpi=95,figsize=(7,17)) i=0
# Filter out non-numeric columns before creating boxplots
numeric_cols =
df.select_dtypes(include=np.number).columns for col in
numeric_cols:   axs[i].boxplot(df[col],vert=False)
axs[i].set_ylabel(col)   i+=1 plt.show()
```


Pregnancies
1
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Glucose
1
0 25 50 75 100 125 150 175 200
BloodPressure
1
0 20 40 60 80 100 120
SkinThickness
1
0 20 40 60 80 100
Insulin
1
0 200 400 600 800
BMI
1
0 10 20 30 40 50 60 70
DiabetesPedigreeFunction
1
0.0 0.5 1.0 1.5 2.0 2.5

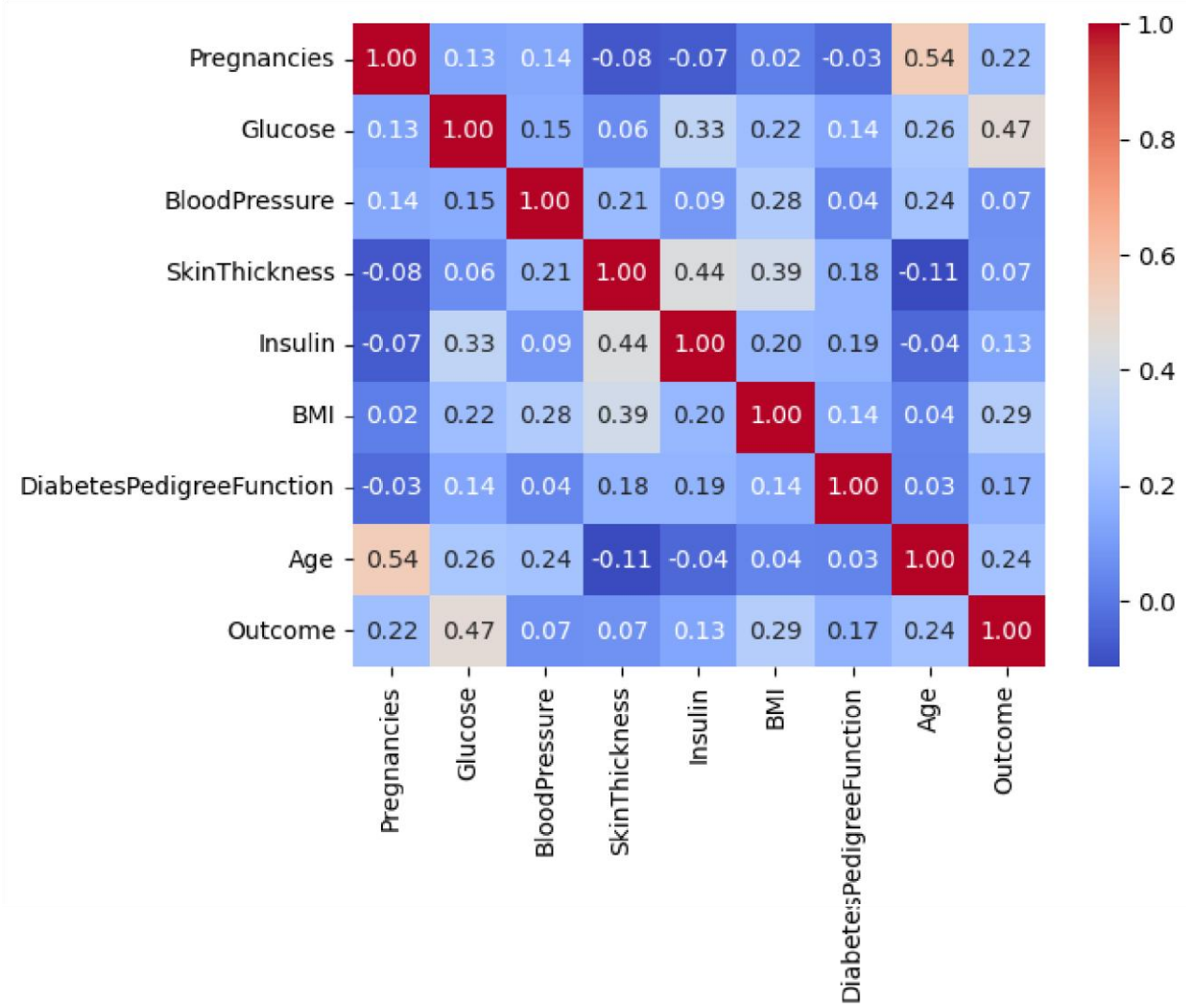



```
#correaltion corr=df.corr()
sns.heatmap(df.corr(),annot=True,fmt='.2f',cmap="coolwarm"
) print(corr)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

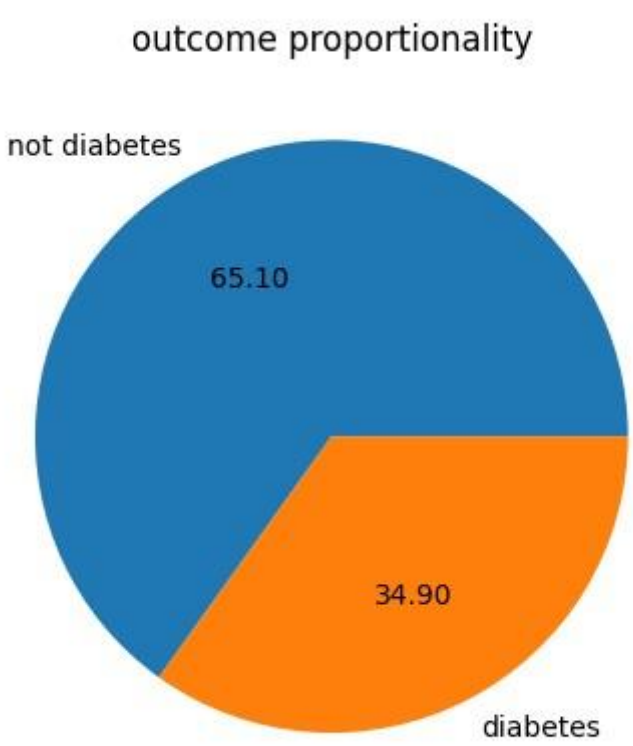
	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	-0.073535	0.017683	-0.033523
Glucose	0.331357	0.221071	0.137337
BloodPressure	0.088933	0.281805	0.041265
SkinThickness	0.436783	0.392573	0.183928
Insulin	1.000000	0.197859	0.185071
BMI	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	0.185071	0.140647	1.000000
Age	-0.042163	0.036242	0.033561
Outcome	0.130548	0.292695	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.239528	0.065068
SkinThickness	-0.113970	0.074752
Insulin	-0.042163	0.130548
BMI	0.036242	0.292695
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000




```
plt.pie(df.Outcome.value_counts(),          labels=['not
diabetes','diabetes'],autopct='%.2f') plt.title('outcome
proportionality')
```

```
plt.show()
```




```
#seperate  array into input and output components
X = df.drop(columns=['Outcome'])
Y = df.Outcome
X.head()
```

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

```
# drop

X = df.drop(columns=['Outcome'])
Y = df.Outcomeprint(X.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
```

```
1                    0.351   31
2                    0.672   32
3                    0.167   21
4                    2.288   33
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(X) rescaledX =
scaler.transform(X) print(rescaledX)
```

```
[[0.35294118 0.74371859 0.59016393 ... 0.50074516 0.23441503 0.48333333]
 [0.05882353 0.42713568 0.54098361 ... 0.39642325 0.11656704 0.16666667]
 [0.47058824 0.91959799 0.52459016 ... 0.34724292 0.25362938 0.18333333]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.390462   0.07130658 0.15      ]
 [0.05882353 0.63316583 0.49180328 ... 0.4485842  0.11571307 0.43333333]
 [0.05882353 0.46733668 0.57377049 ... 0.45305514 0.10119556 0.03333333]]
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = StandardScaler() rescaledX
= scaler.fit_transform(X)
rescaledX[:5]
```

```
array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
         0.20401277,  0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
        -0.68442195, -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
        -1.10325546,  0.60439732, -0.10558415],
       [-0.84488505, -0.99820778, -0.16054575,  0.15453319,  0.12330164,
        -0.49404308, -0.92076261, -1.04154944],
       [-1.14185152,  0.5040552 , -1.50468724,  0.90726993,  0.76583594,
         1.4097456 ,  5.4849091 , -0.0204964 ]])
```

```
import pandas as pd
import numpy as np

data = {'first test':[100,90,np.nan,95,75,87],
'second test':[30,45,56,np.nan,60,70],
       'third test':[np.nan,40,80,98,55,np.nan]}
```

	first test	second test	third test
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

4	75.0	60.0	55.0
5	87.0	70.0	NaN

```
df.dropna()
```

	first test	second test	third test
1	90.0	45.0	40.0
4	75.0	60.0	55.0

```
df = pd.DataFrame(data)
df
```

```
#fillinf missing values with
zero df=df.fillna(0) df
```

	first test	second test	third test
0	100.0	30.0	0.0
1	90.0	45.0	40.0
2	0.0	56.0	80.0
3	95.0	0.0	98.0
4	75.0	60.0	55.0
5	87.0	70.0	0.0

```
df.mean()
```

	0
first test	74.5
second test	43.5
third test	45.5

dtype: float64

```
df3=df.fillna(df.mean())
print('imputation using mean:')
print(df3)
```

```
imputation using mean:
   first test  second test  third test
0       100.0         30.0         0.0
1        90.0         45.0        40.0
2         0.0         56.0        80.0
3        95.0          0.0        98.0
4        75.0         60.0        55.0
5        87.0         70.0         0.0
```

```
df4=df.fillna(df.median())
print('imputation using median:')
print(df4)
```

```
imputation using median:
   first test  second test  third test
0       100.0         30.0         0.0
1         90.0         45.0        40.0
2         0.0         56.0        80.0
3         95.0          0.0        98.0
4         75.0         60.0        55.0
5         87.0         70.0         0.0
```

```
df5=df.ffill()
print('imputationusing next
value:') print(df5)
```

NAME: PASALA NEELIMA

REGNO : 24MDT1064 SLOT:

LU5,U6

EXPERIMENT NO : 04 DATE

04-02-2025

```
import pandas as pd df =
pd.read_csv("/content/iris.csv")
print(df.head())
```

```
  sepal_length  sepal_width  petal_length  petal_width species 0
    5.1          3.5           1.4          0.2  setosa
     1           4.9          3.0           1.4          0.2  setosa
     2           4.7          3.2           1.3          0.2  setosa
     3           4.6          3.1           1.5          0.2  setosa
     4           5.0          3.6           1.4          0.2  setosa
```

```
import numpy as np from sklearn import datasets from
sklearn.model_selection import train_test_split,GridSearchCV from
sklearn.svm import SVC
from sklearn.metrics import classification_report,accuracy_score


iris = datasets.load_iris()
x = iris.data y =
iris.target


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)


param_grid = {
    'C' : [0.1,1,10,100],
    'kernel':['linear','rbf','poly'],
    'gamma' :['scale','auto']
}


# setup the gridsearch cv

grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, verbose=3, n_jobs=-1, scoring='accuracy')
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```


```
#  GET THE BEST PARAMETERS AND BESTSCORE
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("best parameters:", best_params)
print("bestcross-validation score:",best_score)
```

```
best parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'} bestcross-validation
    score: 0.9583333333333334
```

```
best_model =  grid_search.best_estimator_
y_pred = best_model.predict(x_test)


#print classification report
print("classification report:")
print(classification_report(y_test,y_pred))
```

```
classification report:               precision
    recall  f1-score   support

     0       1.00      1.00      1.00        10
     1       1.00      1.00      1.00         9
```

```
2       1.00      1.00      1.00        11
    accuracy                           1.00        30
macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

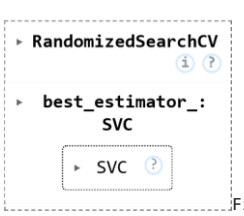
```
accuracy = accuracy_score(y_test,y_pred)

print("accuracy on test set:",accuracy)
```

```
accuracy on test set: 1.0
```

```
import numpy as np from sklearn import datasets from
sklearn.model_selection import train_test_split,RandomizedSearchCV from
sklearn.svm import SVC
from sklearn.metrics import classification_report,accuracy_score
from scipy.stats import uniform, randint


iris = datasets.load_iris()
x = iris.data y =
iris.target


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)


model = SVC()


param_distibutions = {
    'C' : uniform(0.1,100),
    'kernel':['linear','rbf','poly'],
    'gamma' :['scale','auto'] + list(uniform(0.001,1).rvs(size=10))

} Start coding or generate with AI.


#  setup the randomizedsearchcv
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_distibutions, n_iter=100, cv=5, verbose=2, n_jobs=-1, scor
random_search.fit(x_train, y_train)

#    GET    THE    BEST    PARAMETERS    AND    BESTSCORE
best_params_random     =     random_search.best_params_
best_score_random      =      random_search.best_score_
print("best      parameters:",      best_params_random)
print("bestcross-validation score:",best_score_random)
best_model_random   =     random_search.best_estimator_
y_pred_random = best_model_random.predict(x_test)
#print classification report print("classification
report:")
print(classification_report(y_test,y_pred_random))
accuracy_random = accuracy_score(y_test,y_pred_random)
print("accuracy on test set:",accuracy_random)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
     best parameters: {'C': 19.69828624191452, 'gamma': 0.07132602069540483, 'kernel':
     'rbf'} bestcross-validation score: 0.9666666666666668 classification report:
     precision    recall  f1-score   support

     0       1.00      1.00      1.00        10
     1       1.00      1.00      1.00         9           2       1.00      1.00      1.00
             11

         accuracy                           1.00        30
     macro avg       1.00      1.00      1.00        30
     weighted avg       1.00      1.00      1.00        30
     accuracy on test set: 1.0
```

```
random_search.fit(x_train, y_train)
```




```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```