PROJECT ON Medical INSURANCE PRICE PREDICTION

&

STATISTICAL MACHINE LEARNING CODINGS REPORT

2203A52151 JILLA.KIRTHAN

Under the Guidance of

DR. KIRAN ERANKI, PH. D(IITB),
Assoc.Prof., CSE,
S R University, Warangal

Submitted to

Department of Computer Science and Artificial Intelligence

SR University

Anantha Sagar(V), Hasanparthy(M), Hanam Konda (Dist.) – 506371

June 2023

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that the Statistical Machine Learning (STATML ASSIGNMENTS) course report entitled **"**Statistical Machine Learning Coding's Report**"** is a record of Bonafide work carried out by the student(s) Jilla. Kirthan bearing roll number 2203A52151 of Computer Science and Artificial Intelligence department during the academic year 2023-24.

Supervisor

(DR. KIRAN ERANKI)

ABSTRACT

The Statistical Machine Learning Lab is an immersive educational program that imparts essential skills in data analysis and statistical machine learning. This hands-on experience covers data preprocessing, exploratory data analysis, model selection, training, and evaluation, emphasizing model interpretability. Through practical exercises and real-world projects, participants gain proficiency in using machine learning techniques for tasks like classification, regression, and clustering. This lab equips students and professionals with the tools and knowledge needed to work effectively with data and build predictive models, making it a valuable resource for anyone seeking to leverage statistical machine learning in various domains.

The lab commenced with an introduction to the Python programming language and its libraries, which are pivotal tools for machine learning tasks. Participants were guided through the process of setting up their development environment and were introduced to key libraries like NumPy, Pandas, and Scikit-Learn. Proficiency in Python was crucial, as it served as the foundation for all subsequent machine learning endeavours. Following the foundational Python knowledge, the lab proceeded to explore the diverse landscape of machine learning algorithms.

INDEX

Medical Insurance Price Prediction

INTRODUCTION:

You must have heard some advertisements regarding medical insurance that promises to help financially in case of any medical emergency. One who purchases this type of insurance has to pay premiums monthly and this premium amount varies vastly depending upon various factors. In this article, we will try to extract some insights from a dataset that contains details about the background of a person who is purchasing medical insurance along with what amount of premium is charged to those individuals as well using machine learning in python

Problem Statement:

The goal of this project is to develop a machine learning model that can accurately predict medical insurance prices based on various individual attributes. Medical insurance is essential in covering the cost of healthcare, and understanding the factors that influence insurance charges can help individuals make informed decisions when purchasing insurance. By analysis dataset containing information about individuals and their insurance charges, we aim to build a predictive model that can assist in estimating insurance premiums.

Literature Review:

In the realm of healthcare and insurance, several studies have explored the factors influencing insurance charges. Previous research has highlighted age, smoking status, BMI, and geographical region as significant determinants of insurance costs. Machine learning techniques, such as Linear Regression, Support Vector Machines, Random Forest, and XG Boost, have been applied to predict insurance charges with varying degrees of success. This project draws upon these methodologies and aims to enhance predictive accuracy by analysis a comprehensive dataset and employing data preprocessing, feature engineering, and model selection techniques

Methodology:

1. Data Collection: The project utilizes a dataset containing information about individuals, including age, sex, BMI, number of children, smoking status, and region, as well as their corresponding insurance charges.

2. Data Preprocessing: Data preprocessing includes handling missing values, detecting and treating outliers, encoding categorical variables, and ensuring the data approximates a normal distribution. Outliers in the BMI column were treated by capping, and categorical variables were encoded.

3. Model Development: Several machine learning models are trained and evaluated, including Linear Regression, Support Vector Machine (SVR), Random Forest, Gradient Boosting, and XGBoost. Cross-validation is performed to assess model performance.

4. Feature Selection: The importance of features in predicting insurance charges is assessed, and non-essential columns are dropped for the final model.

5. Model Evaluation: Models are evaluated based on training accuracy, test accuracy, and cross-validation scores to identify the best-performing model.

6. Hyperparameter Tuning: For selected models, hyperparameter tuning is performed to optimize model performance.

CODE:

Importing Libraries and Dataset:

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- Pandas – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- Numpy – Numpy arrays are very fast and can perform large computations in a very short time.
- matplotlib and seaborn – This library is used to draw visualizations.

- Python3

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as pt
import warnings
warnings.filterwarnings("ignore")
```

Now let's use the panda's data frame to load the dataset and look at the first five rows of it.

- Python3

```
df=pd.read_csv("insurance.csv")
```

Output:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Data Set

Now, we can observe the data and its shape(rows x columns)

This dataset contains 1338 data points with 6 independent features and 1 target feature(charges).

- Python3

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Details about the columns of the dataset

From the above, we can see that the dataset contains 2 columns with float values 3 with categorical values and the rest contains integer values.

- Python3

```
df.describe()
```

Output:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Descriptive Statistical measures of the data

We can look at the *Descriptive Statistical measures* the continuous data available in the dataset.

Exploratory Data Analysis:

EDA is an approach to analysis the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations. While performing the EDA of this dataset we will try to look at what is the relation between the independent features that is how one affects the other.

- Python3

```
df.isnull().sum()
```

Output:

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

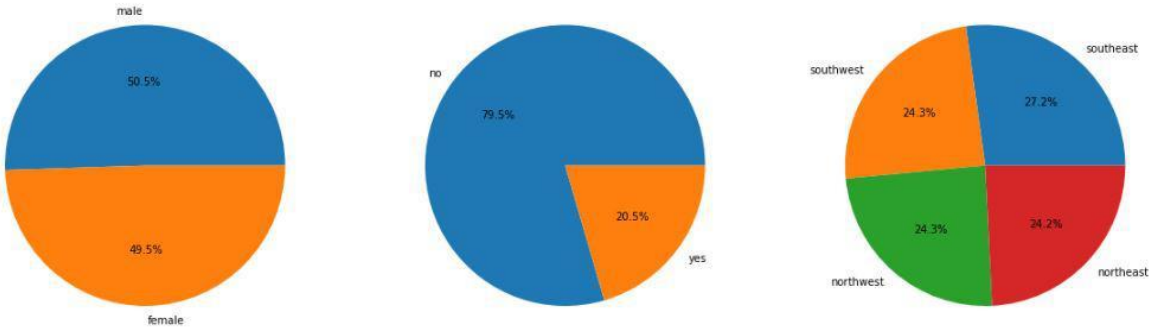
Count of the null values column wise

So, here we can conclude that there are no null values in the dataset given.

- Python3

```
features = ['sex', 'smoker', 'region']


plt.subplots(figsize=(20, 10))
for i, col in enumerate(features):
  plt.subplot(1, 3, i + 1)


  x = df[col].value_counts()
  plt.pie(x.values,
      labels=x.index,
      autopct='%1.1f%%')


plt.show()
```

Output:



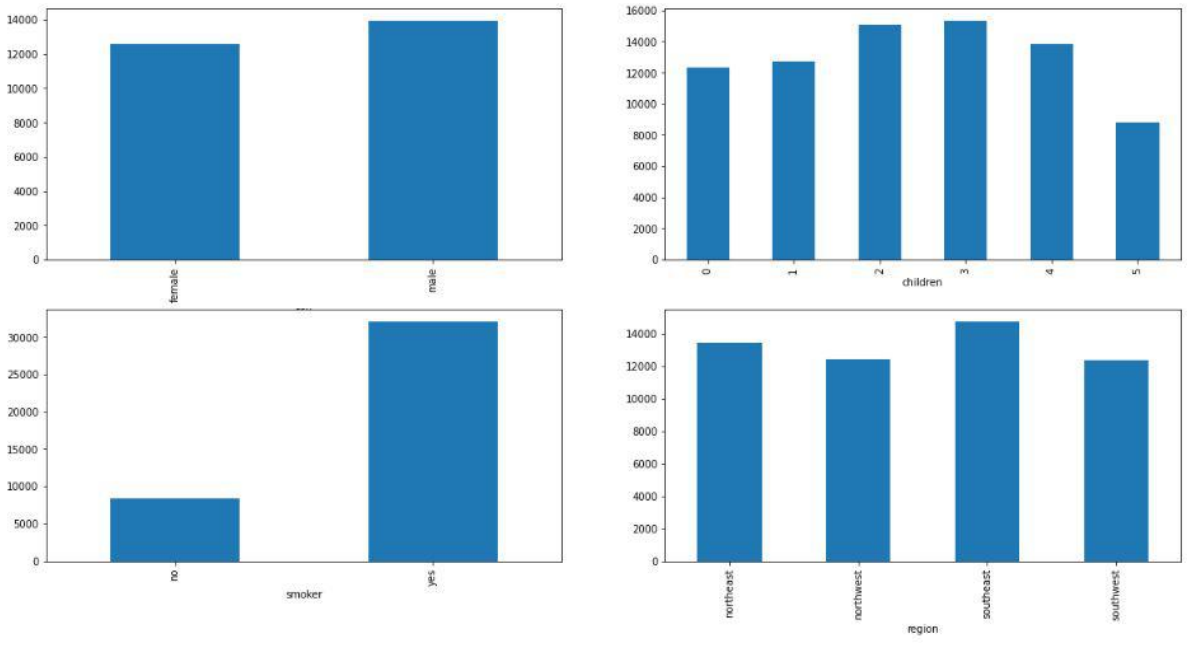
Pie chart for the sex, smoker, and region column

The data provided to us is equally distributed among the sex and the region columns but in the smoker column, we can observe a ratio of 80:20.

- Python3

```
features = ['sex', 'children', 'smoker', 'region']


plt.subplots(figsize=(20, 10))
for i, col in enumerate(features):
    plt.subplot(2, 2, i + 1)
    df.groupby(col).mean()['charges'].plot.bar()
plt.show()
```

Output:



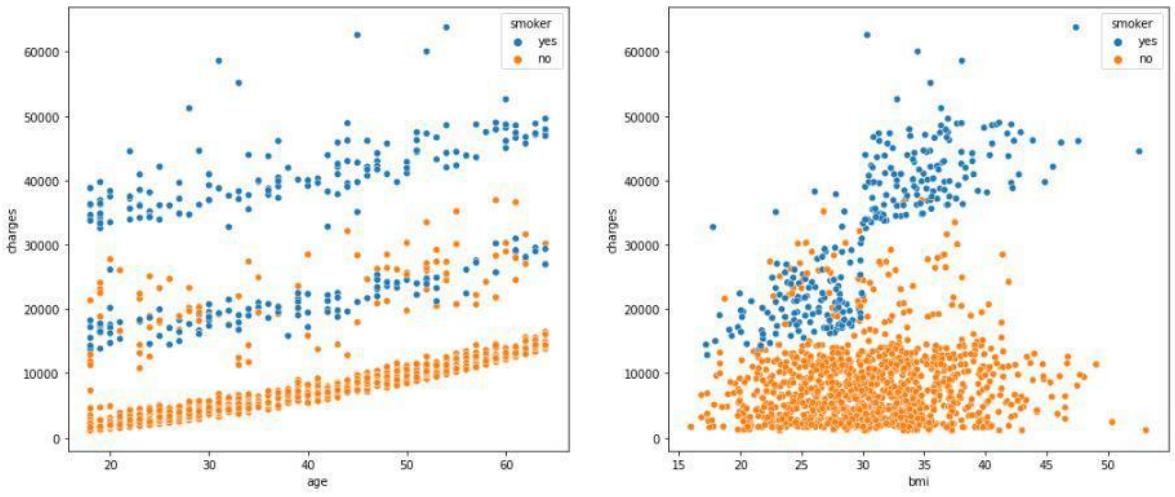
Comparison between charges paid between different groups

Now let's look at some of the observations which are shown in the above graphs:

- Charges are on the higher side for males as compared to females but the difference is not that much.
- Premium charged from the smoker is around thrice that which is charged from non-smokers.
- Charges are approximately the same in the given four regions.
- Python3

```
features = ['age', 'bmi']


plt.subplots(figsize=(17, 7))
for i, col in enumerate(features):
    plt.subplot(1, 2, i + 1)
    sb.scatterplot(data=df, x=col,
                   y='charges',
                   hue='smoker')
plt.show()
```

Output:



Scatter plot of the charges paid v/s age and BMI respectively

A clear distinction can be observed here between the charges that smokers have to pay. Also here as well we can observe that as the age of a person increases premium prices goes up as well.

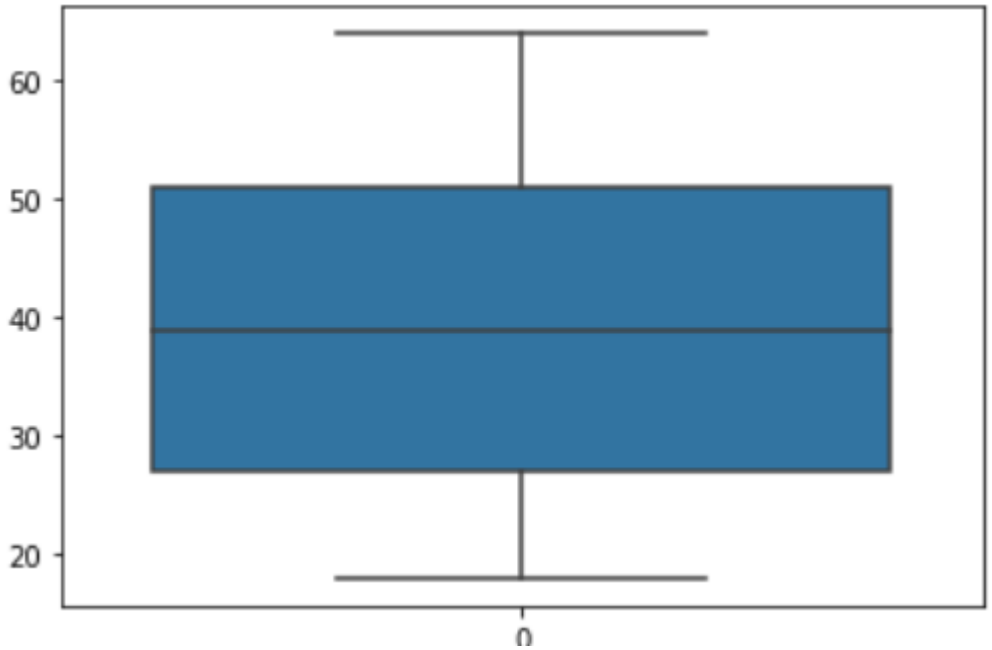
DATA PREPROCESSING:

Data preprocessing is technique to clean the unusual data like the missing values,wrong data,wrong format of data,duplicated data and the outliers.In this data we can observe that there are no missing values and wrong data.The only thing we can need to check is for duplicates and presence of outliers.

- Python3

```
df.drop_duplicates(inplace=True)

sns.boxplot(df['age'])
```

Output:

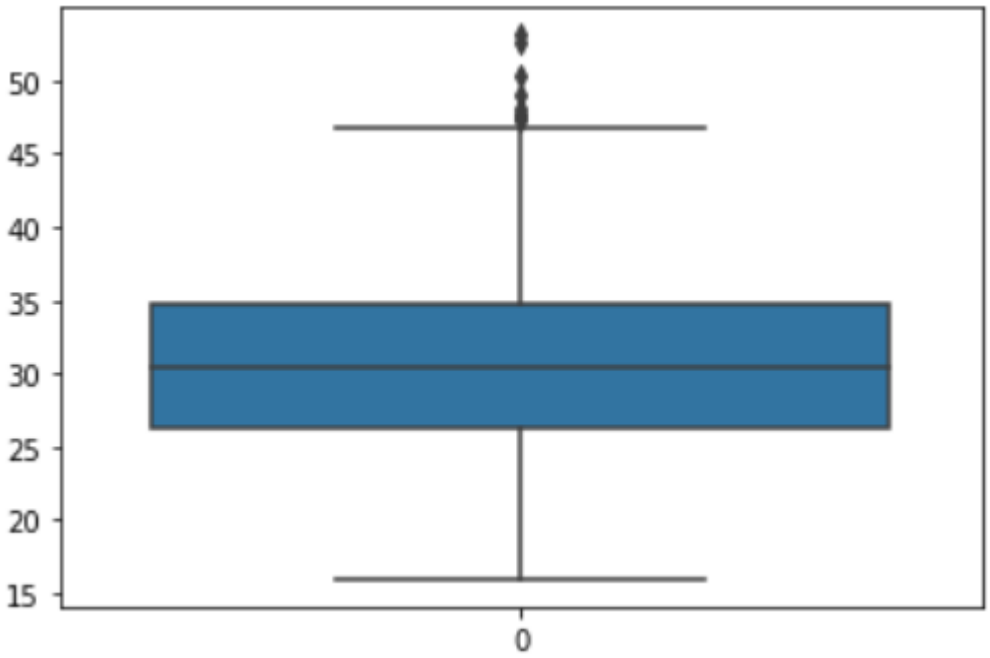


Boxplot of age

we can see that there are no outliers present in age column

- Python3

```
sns.boxplot(df['bmi'])
```



Box plot of bmi

Due to the presence of outliers present in bmi column we need to treat the outliers by replacing the values with mean as the bmi column consists of continuous data.

- Python3

```
Q1=df['bmi'].quantile(0.25)
Q2=df['bmi'].quantile(0.5)
Q3=df['bmi'].quantile(0.75)
iqr=Q3-Q1
lowlim=Q1-1.5*iqr
upplim=Q3+1.5*iqr
print(lowlim)
print(upplim)
```

Output:

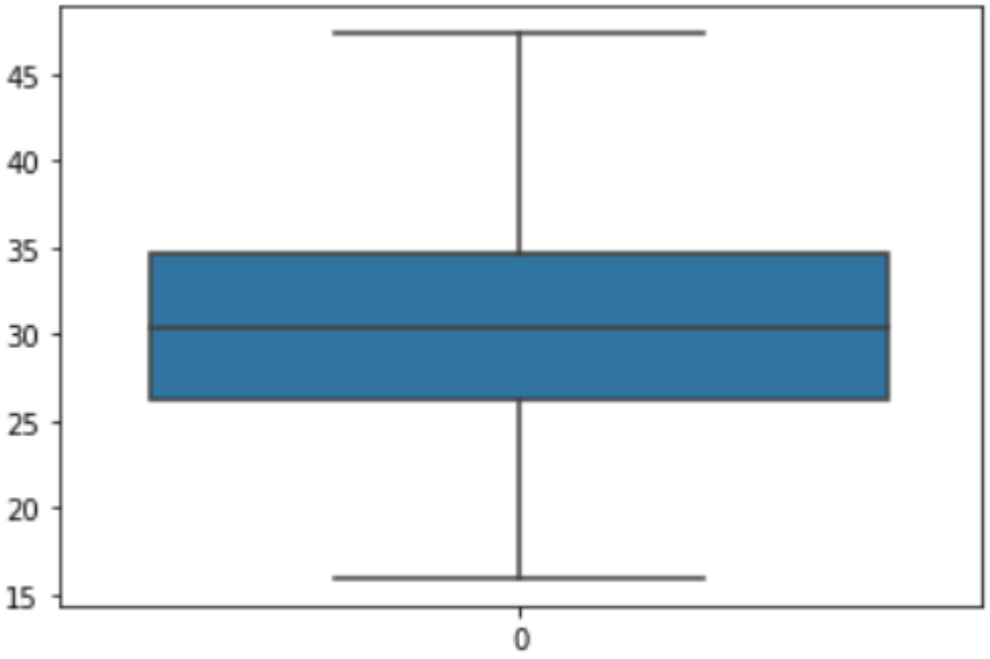
13.674999999999994

47.31500000000001

- Python3

```
from feature_engine.outliers import ArbitraryOutlierCapper
```

```
arb=ArbitraryOutlierCapper(min_capping_dict={'bmi':13.6749},max_capping_dict={'bmi':47.315})
df[['bmi']]=arb.fit_transform(df[['bmi']])
sns.boxplot(df['bmi'])
```



Box plot for bmi

Now we successfully treated the outliers.

Data Wrangling:

Data wrangling is a technique to ensure whether the data follow normal or standard distribution and encode the discrete data for prediction.

- Python3

```
df['bmi'].skew()
df['age'].skew()
```

Output:

0.23289153320569975

0.054780773126998195

Data in both the age and BMI column approximately follow a normal distribution which is a good point with respect to the model's learning.

Encoding:

encoding is to be done for discrete categorical data (sex,bmi,region).

- Python3

```
df['sex']=df['sex'].map({'male':0,'female':1})
df['smoker']=df['smoker'].map({'yes':1,'no':0})
df['region']=df['region'].map({'northwest':0, 'northeast':1,'southeast':2,'southwest':3})
```

Output:

	age	sex	bmi	children	smoker	region	charges
0	19	1	27.900	0	1	3	16884.92400
1	18	0	33.770	1	0	2	1725.55230
2	28	0	33.000	3	0	2	4449.46200
3	33	0	22.705	0	0	0	21984.47061
4	32	0	28.880	0	0	0	3866.85520
...	...	...	...	...	...	...	...
1333	50	0	30.970	3	0	0	10600.54830
1334	18	1	31.920	0	0	1	2205.98080
1335	18	1	36.850	0	0	2	1629.83350
1336	21	1	25.800	0	0	3	2007.94500
1337	61	1	29.070	0	1	0	29141.36030

1337 rows × 7 columns

Encoded Data

Now the discrete data is encoded and the data preprocessing and data wrangling part is completed Now we can go for model development.

Output:

- Python3

```
df.corr()
```

	age	sex	bmi	children	smoker	region	charges
age	1.000000	0.019814	0.111998	0.041536	-0.025587	0.001771	0.298308
sex	0.019814	1.000000	-0.044831	-0.017848	-0.076596	-0.008998	-0.058044
bmi	0.111998	-0.044831	1.000000	0.013692	0.003151	0.156937	0.199063
children	0.041536	-0.017848	0.013692	1.000000	0.007331	-0.002842	0.067389
smoker	-0.025587	-0.076596	0.003151	0.007331	1.000000	0.012736	0.787234
region	0.001771	-0.008998	0.156937	-0.002842	0.012736	1.000000	0.010767
charges	0.298308	-0.058044	0.199063	0.067389	0.787234	0.010767	1.000000

correlation matrix

Model Development:

There are so many state-of-the-art ML models available in academia but some model fits better to some problem while some fit better than other. So, to make this decision we split our data into Training and validation data .Then we use the validation data to choose the model with the highest performance.

- Python3

```
X=df.drop(['charges'],axis=1)
Y=df[['charges']]
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
l1=[]
l2=[]
l3=[]
cvs=0
for i in range(40,50):
 xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.2,random_state=i)
 lrmodel=LinearRegression()
 lrmodel.fit(xtrain,ytrain)
```

```
l1.append(lrmodel.score(xtrain,ytrain))
l2.append(lrmodel.score(xtest,ytest))
cvs=(cross_val_score(lrmodel,X,Y,cv=5,)).mean()
l3.append(cvs)
df1=pd.DataFrame({'train acc':l1,'test acc':l2,'cvs':l3})
 df1
```

Output:

	train acc	test acc	cvs
0	0.741659	0.778409	0.74707
1	0.756401	0.706267	0.74707
2	0.729542	0.806239	0.74707
3	0.754260	0.732791	0.74707
4	0.742966	0.779591	0.74707
5	0.753281	0.731769	0.74707
6	0.741261	0.776456	0.74707
7	0.731940	0.796173	0.74707
8	0.751915	0.741742	0.74707
9	0.756348	0.722565	0.74707

Scores for various random_state number

After dividing the data into training and validation data it is considered a better practice to achieve stable and fast training of the model.We have identified the best random_state number for this data set as 42 .Now we fix this random_state and try with different ml algorithms for better score or accuracy.

Now let's train some state-of-the-art machine learning models on the training data and then use the validation data for choosing the best out of them for prediction.

- Python3

```
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.2,random_state=42)
lrmodel=LinearRegression()
```

```
lrmodel.fit(xtrain,ytrain)
print(lrmodel.score(xtrain,ytrain))
print(lrmodel.score(xtest,ytest))
print(cross_val_score(lrmodel,X,Y,cv=5,).mean())
```

Output:

Linear Regression:

0.7295415541376445

0.8062391115570589

0.7470697972809902

- Python3

```
from sklearn.metrics import r2_score
svrmodel=SVR()
svrmodel.fit(xtrain,ytrain)
ypredtrain1=svrmodel.predict(xtrain)
ypredtest1=svrmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain1))
print(r2_score(ytest,ypredtest1))
print(cross_val_score(svrmodel,X,Y,cv=5,).mean())
```

Output:

SVR:

-0.10151474302536445

-0.1344454720199666

-0.10374591327267262

- Python3

```
rfmodel=RandomForestRegressor(random_state=42)
rfmodel.fit(xtrain,ytrain)
ypredtrain2=rfmodel.predict(xtrain)
```

```
ypredtest2=rfmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain2))
print(r2_score(ytest,ypredtest2))
print(cross_val_score(rfmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=RandomForestRegressor(random_state=42)
param_grid={'n_estimators':[10,40,50,98,100,120,150]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
rfmodel=RandomForestRegressor(random_state=42,n_estimators=120)
rfmodel.fit(xtrain,ytrain)
ypredtrain2=rfmodel.predict(xtrain)
ypredtest2=rfmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain2))
print(r2_score(ytest,ypredtest2))
print(cross_val_score(rfmodel,X,Y,cv=5,).mean())
```

Output:

Random Forest Regressor:

0.9738163260247533

0.8819423353068565

0.8363637309718952

Hyperparameter tuning:

{'n_estimators': 120}

0.9746383984429655

0.8822009842175969

0.8367438097052858

- Python3

```
gbmodel=GradientBoostingRegressor()
gbmodel.fit(xtrain,ytrain)
ypredtrain3=gbmodel.predict(xtrain)
ypredtest3=gbmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain3))
print(r2_score(ytest,ypredtest3))
print(cross_val_score(gbmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=GradientBoostingRegressor()
param_grid={'n_estimators':[10,15,19,20,21,50],'learning_rate':[0.1,0.19,0.2,0.21,0.8,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
gbmodel=GradientBoostingRegressor(n_estimators=19,learning_rate=0.2)
gbmodel.fit(xtrain,ytrain)
ypredtrain3=gbmodel.predict(xtrain)
ypredtest3=gbmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain3))
print(r2_score(ytest,ypredtest3))
print(cross_val_score(gbmodel,X,Y,cv=5,).mean())
```

Output:

```
Gradient Boosting Regressor:
0.8931345821166041
0.904261922040551
0.8549940291799407
Hyperparameter tuning
{'learning_rate': 0.2, 'n_estimators': 21}
0.8682397447116927
0.9017109716082661
0.8606041910125791
```

- Python3

```
xgmodel=XGBRegressor()
xgmodel.fit(xtrain,ytrain)
ypredtrain4=xgmodel.predict(xtrain)
ypredtest4=xgmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(xgmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=XGBRegressor()
param_grid={'n_estimators':[10,15,20,40,50],'max_depth':[3,4,5],'gamma':[0,0.15,0.3,0.5,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
xgmodel=XGBRegressor(n_estimators=15,max_depth=3,gamma=0)
xgmodel.fit(xtrain,ytrain)
ypredtrain4=xgmodel.predict(xtrain)
ypredtest4=xgmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(xgmodel,X,Y,cv=5,).mean())
```

Output:

XGB Regressor:

0.9944530188818493

0.8618686915522016

0.8104424308304893

Hyperparameter tuning:

{'gamma': 0, 'max_depth': 3, 'n_estimators': 15}

0.870691899927822

0.904151903449132

0.8600710679082143

Comapring All Models:

LinearRegression	**0.729**	**0.806**	**0.747**
Model	**Train Accuracy**	**Test Accuracy**	**CV Score**
SupportVectorMachine	-0.105	-0.134	0.103
RandomForest	0.974	0.882	0.836
GradientBoost	0.868	0.901	0.860
XGBoost	0.870	0.904	0.860

From the above table we can observe that XGBoost is the best model.Now we need to identify the important features for predicting of charges.

- Python3

```
feats=pd.DataFrame(data=grid.best_estimator_.feature_importances_,index=X.columns,columns=
['Importance'])
feats
```

Output:

	Importance
age	0.050547
sex	0.002721
bmi	0.092197
children	0.013500
smoker	0.834279
region	0.006756

Features Importance

- Python3

```
important_features=feats[feats['Importance']>0.01]
important_features
```

Output:

	Importance
age	0.050547
bmi	0.092197
children	0.013500
smoker	0.834279

Important Features

Final Model:

- Python3

```
df.drop(df[['sex','region']],axis=1,inplace=True)
Xf=df.drop(df[['charges']],axis=1)
X=df.drop(df[['charges']],axis=1)
xtrain,xtest,ytrain,ytest=train_test_split(Xf,Y,test_size=0.2,random_state=42)
finalmodel=XGBRegressor(n_estimators=15,max_depth=3,gamma=0)
finalmodel.fit(xtrain,ytrain)
ypredtrain4=finalmodel.predict(xtrain)
ypredtest4=finalmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(finalmodel,X,Y,cv=5,).mean())
```

Final Model:

Train accuracy: 0.870691899927822

Test accuracy: 0.904151903449132

CV Score: 0.8600710679082143

Save Model:

- Python3

```
from pickle import dump
dump(finalmodel,open('insurancemodelf.pkl','wb'))
```

Predict on new data:

- Python3

```
new_data=pd.DataFrame({'age':19,'sex':'male','bmi':27.9,'children':0,'smoker':'yes','region':'northeast'},
index=[0])
new_data['smoker']=new_data['smoker'].map({'yes':1,'no':0})
new_data=new_data.drop(new_data[['sex','region']],axis=1)
finalmodel.predict(new_data)
```

Output:

array([17483.12], dtype=float32

Results and Discussion:

The XGBoost model outperformed other models, achieving a test accuracy of 90.4% and a cross-validation score of 86.0%.

Important features influencing insurance charges were identified, emphasizing the role of factors like age and smoking status.

The project discussion highlights the significance of understanding these factors in making informed insurance choices.

Conclusion:

Out of all the models XG Boost model is giving the highest accuracy this means predictions made by this model are close to the real values as compared to the other model.

The dataset we have used here was small still the conclusion we drew from them were quite similar to what is observed in the real-life scenario. If we would have a bigger dataset then we will be able to learn even deeper patterns in the relation between the independent features and the premium charged from the buyers.

References:

- [Insert relevant references and data sources used in the project.]

CODE-1:

CODE ASSIGNMENTS 01: Exposing to various frameworks of StatML - Numpy, Pandas, Matplotlib, Seaborn, Tensorflow, Keras.

NUMPY

CODES:

```
lst1=[1,2,3]
array1=np.array(lst1)
array1
```

OUTPUT:

```
array([1, 2, 3])
```

```
print("array2 multiplied by array1: ", array1 * array2)
print("array2 divided by array1: ", array2 / array1)
print("array2 raised to the power of array1: ", array2 ** array1)
```

OUTPUT:

array2 multiplied by array1: [10 22 36]
array2 divided by array1: [10. 5.5 4.]
array2 raised to the power of array1: [10 121 1728]

```
# sine function
print("Sine: ",np.sin(array1))
# logarithm
print("Natural logarithm: ",np.log(array1))
print("Base-10 logarithm: ",np.log10(array1))
print("Base-2 logarithm: ",np.log2(array1))
# Exponential
print("Exponential: ",np.exp(array1))
```

OUTPUT:

```
Sine:  [0.84147098 0.90929743 0.14112001]
Natural logarithm:  [0.         0.69314718 1.09861229]
Base-10 logarithm:  [0.         0.30103    0.47712125]
Base-2 logarithm:  [0.        1.        1.5849625]
Exponential:  [ 2.71828183  7.3890561  20.08553692]
```

```
import numpy as np
print("a series of zero: ",np.zeros(7))
print("a series of ones: ",np.ones(9))
print("a series of arrange: ",np.arange(7,1))
print("a series of arrange: ",np.arange(0,11,2))
print("a series of arrange: ",np.arange(0,2,33))
print("a series of linspace: ",np.linspace(1,5,11))
```

OUTPUT:

```
a series of zero:  [0. 0. 0. 0. 0. 0. 0.]
a series of ones:  [1. 1. 1. 1. 1. 1. 1. 1. 1.]
a series of arrange:  []
a series of arrange:  [ 0  2  4  6  8 10]
a series of arrange:  [0]
a series of linspace:  [1.  1.4 1.8 2.2 2.6 3.  3.4 3.8 4.2 4.6 5. ]
```

```
my_mat = [[1,2,3],[4,5,6],[7,8,9]]
mat = np.array(my_mat)
print("Type/Class of this object:",type(mat))
print("Here is the matrix\n----------\n",mat,"\n----------")
```

OUTPUT:

```
Type/Class of this object: <class 'numpy.ndarray'>
Here is the matrix
----------
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
----------
```

```
my_tuple = np.array([(1.5,2,3), (4,5,6)])
mat_tuple = np.array(my_tuple)
print (mat_tuple)
```

OUTPUT:

```
[[1.5 2.  3. ]
 [4.  5.  6. ]]
```

```
print("Dimension of this matrix: ",mat.ndim,sep='')
print("Size of this matrix: ", mat.size,sep='')
print("Shape of this matrix: ", mat.shape,sep='')
print("Data type of this matrix: ", mat.dtype,sep='')
```

```
Dimension of this matrix: 2
Size of this matrix: 9
Shape of this matrix: (3, 3)
Data type of this matrix: int64
```

```
print("Vector of zeros: ",np.zeros(5))
print("Matrix of zeros: ",np.zeros((3,4)))
print("Vector of ones: ",np.ones(4))
print("Matrix of ones: ",np.ones((4,2)))
print("Matrix of 5's: ",5*np.ones((3,3)))
```

```
print("Identity matrix of dimension 2:",np.eye(2))
print("Identity matrix of dimension 4:",np.eye(4))
print("Random matrix of shape
(4,3):\n",np.random.randint(low=1,high=10,size=(4,3)))
```

OUTPUT:

```
Vector of zeros:  [0. 0. 0. 0. 0.]
Matrix of zeros:  [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Vector of ones:  [1. 1. 1. 1.]
Matrix of ones:  [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
Matrix of 5's:  [[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
Identity matrix of dimension 2: [[1. 0.]
 [0. 1.]]
Identity matrix of dimension 4: [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Random matrix of shape (4,3):
 [[9 7 4]
 [4 2 1]
 [9 6 1]
 [8 9 9]]
```

```
a = np.random.randint(1,100,30)
b = a.reshape(2,3,5)
c = a.reshape(6,5)
print ("Shape of a:", a.shape)
print ("Shape of b:", b.shape)
print ("Shape of c:", c.shape)
```

OUTPUT:

```
Shape of a: (30,)
Shape of b: (2, 3, 5)
Shape of c: (6, 5)
```

```
arr = np.arange(0,11)
print("Array:",arr)
print("Element at 7th index is:", arr[7])
print("Elements from 3rd to 5th index are:", arr[3:6])
print("Elements up to 4th index are:", arr[:4])
print("Elements from last backwards are:", arr[-1::-1])
print("3 Elements from last backwards are:", arr[-1:-6:-2])

arr2 = np.arange(0,21,2)
print("New array:",arr2)
print("Elements at 2nd, 4th, and 9th index are:", arr2[[2,4,9]]) # Pass
a list as a index to subset
```

OUTPUT:

```
Array: [ 0  1  2  3  4  5  6  7  8  9 10]
Element at 7th index is: 7
Elements from 3rd to 5th index are: [3 4 5]
Elements up to 4th index are: [0 1 2 3]
Elements from last backwards are: [10  9  8  7  6  5  4  3  2  1  0]
3 Elements from last backwards are: [10  8  6]
New array: [ 0  2  4  6  8 10 12 14 16 18 20]
Elements at 2nd, 4th, and 9th index are: [ 4  8 18]
```

```
mat = np.random.randint(10,100,15).reshape(3,5)
print("Matrix of random 2-digit numbers\n",mat)

print("\nDouble bracket indexing\n")
print("Element in row index 1 and column index 2:", mat[1][2])

print("\nSingle bracket with comma indexing\n")
print("Element in row index 1 and column index 2:", mat[1,2])
print("\nRow or column extract\n")

print("Entire row at index 2:", mat[2])
print("Entire column at index 3:", mat[:,3])

print("\nSubsetting sub-matrices\n")
print("Matrix with row indices 1 and 2 and column indices 3 and 4\n",
mat[1:3,3:5])
print("Matrix with row indices 0 and 1 and column indices 1 and 3\n",
mat[0:2,[1,3]])
```

OUTPUT:

```
Matrix of random 2-digit numbers
 [[13 78 24 15 48]
 [56 69 69 36 77]
 [55 40 58 41 35]]

Double bracket indexing

Element in row index 1 and column index 2: 69

Single bracket with comma indexing

Element in row index 1 and column index 2: 69

Row or column extract

Entire row at index 2: [55 40 58 41 35]
Entire column at index 3: [15 36 41]

Subsetting sub-matrices

Matrix with row indices 1 and 2 and column indices 3 and 4
 [[36 77]
 [41 35]]
```

```
Matrix with row indices 0 and 1 and column indices 1 and 3
 [[78 15]
 [69 36]]
```

```
mat = np.random.randint(10,100,15).reshape(3,5)
print("Matrix of random 2-digit numbers\n",mat)
print ("\nElements greater than 50\n", mat[mat>50])
```

OUTPUT:

```
Matrix of random 2-digit numbers
 [[39 35 28 62 21]
 [28 88 90 55 31]
 [92 33 63 51 51]]

Elements greater than 50
 [62 88 90 55 92 63 51 51]
```

```
mat>50
```

OUTPUT:

```
array([[False, False, False, True, False], [False, True, True, True,
False], [ True, False, True, True, True]])
```

```
mat1 = np.random.randint(1,10,9).reshape(3,3)
mat2 = np.random.randint(1,10,9).reshape(3,3)
print("\n1st Matrix of random single-digit numbers\n",mat1)
print("\n2nd Matrix of random single-digit numbers\n",mat2)

print("\nAddition\n", mat1+mat2)
print("\nMultiplication\n", mat1*mat2)
print("\nDivision\n", mat1/mat2)
print("\nLineaer combination: 3*A - 2*B\n", 3*mat1-2*mat2)

print("\nAddition of a scalar (100)\n", 100+mat1)

print("\nExponentiation, matrix cubed here\n", mat1**3)
print("\nExponentiation, sq-root using pow function\n",pow(mat1,0.5))
```

OUTPUT:

```
1st Matrix of random single-digit numbers
 [[1 8 4]
 [3 8 5]
 [9 1 2]]

2nd Matrix of random single-digit numbers
 [[5 9 5]
 [4 6 4]
 [8 8 4]]

Addition
 [[ 6 17  9]
 [ 7 14  9]
 [17  9  6]]
```

```
Multiplication
 [[ 5 72 20]
 [12 48 20]
 [72  8  8]]

Division
 [[0.2        0.88888889 0.8       ]
 [0.75       1.33333333 1.25      ]
 [1.125      0.125      0.5       ]]

Lineaer combination: 3*A - 2*B
 [[ -7   6   2]
 [  1  12   7]
 [ 11 -13  -2]]

Addition of a scalar (100)
 [[101 108 104]
 [103 108 105]
 [109 101 102]]

Exponentiation, matrix cubed here
 [[  1 512  64]
 [ 27 512 125]
 [729   1   8]]

Exponentiation, sq-root using pow function
 [[1.         2.82842712 2.        ]
 [1.73205081 2.82842712 2.23606798]
 [3.         1.         1.41421356]]
```

PANDAS

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

OUTPUT:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
import pandas as pd
st = pd.read_csv("/content/student_details.csv")
st.head()
```

OUTPUT:

	NAME	CLASS	RANK
0	Sunil	a1	1
1	Sunny	de	2
2	Shiva	gt	3
3	Rohith	s	4

```
datatxt = pd.read_table("/content/data2.txt")
datatxt.head()
```

OUTPUT:

	Sunil	a1	1
0	Sunny	de	2
1	Shiva	gt	3
2	Rohith	s	4

```
dataxls = pd.read_excel("/content/Height_weight.xlsx")
dataxls
```

OUTPUT:

	Name	Height	Weight
0	Ashton	155	135
1	Kate	125	140
2	Bruce	178	210
3	Tom	181	165

	Name	Height	Weight
4	Bill	165	180

```
list_of=pd.read_html("https://en.wikipedia.org/wiki/2016_Summer_Olympics")
games=list_of[0]
games.head()
```

OUTPUT:

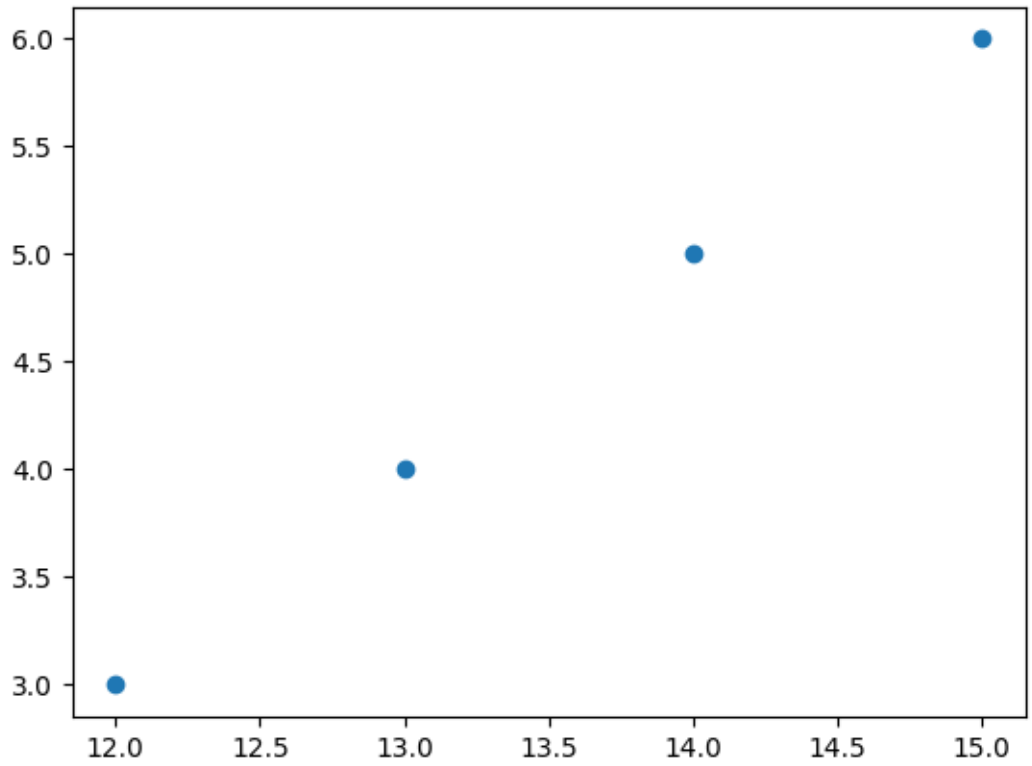
	0	1
0	Emblem of the 2016 Summer Olympics[a]	Emblem of the 2016 Summer Olympics[a]
1	Host city	Rio de Janeiro, Brazil
2	Motto	A New World (Portuguese: Um mundo novo)
3	Nations	207 (including IOA and EOR teams)[1]
4	Athletes	11,180 (6,146 men, 5,034 women)[1]

MATPLOTLIB

```
import matplotlib.pyplot as plt
people=['sunil','rohith','shiva','sai']
age = [12,13,14,15]
weight= [23,45,76,54]
height=[3,4,5,6]
plt.scatter(age,height)
plt.show
```

OUTPUT:

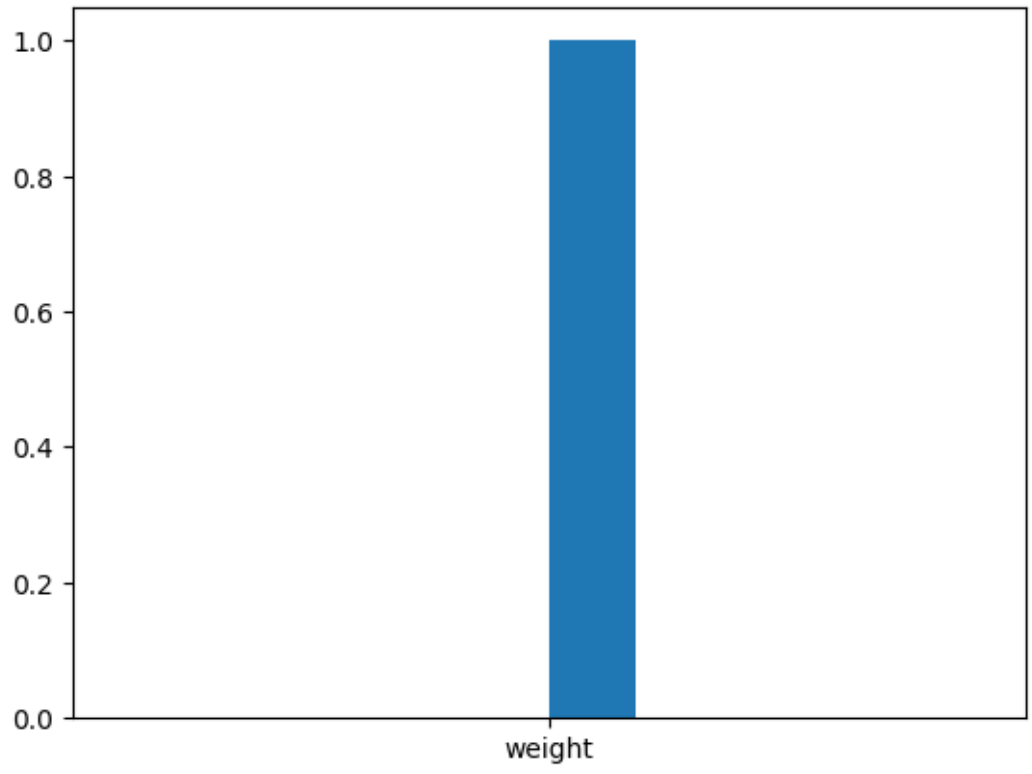
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
plt.hist("weight")
plt.show
```

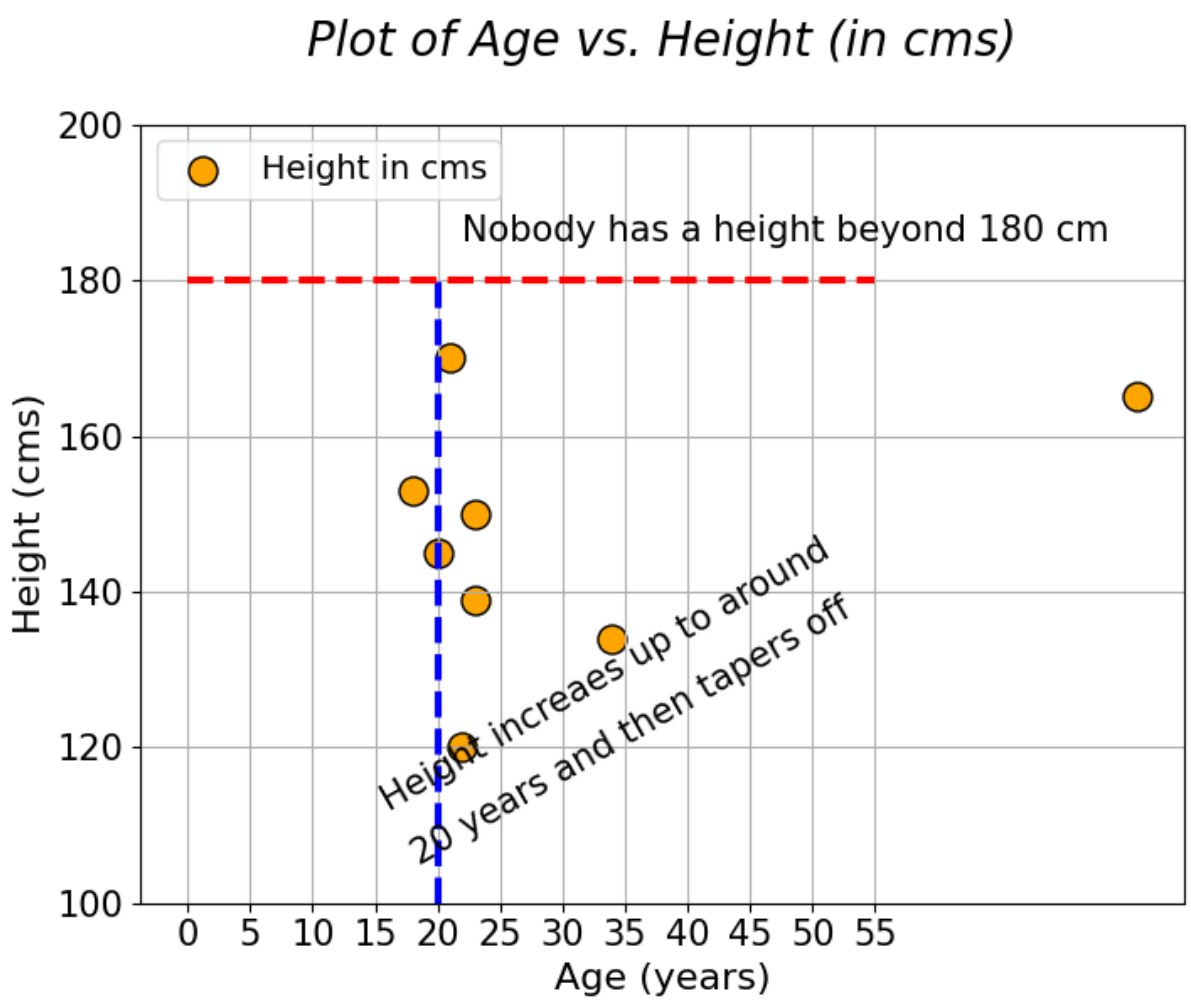
OUTPUT:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



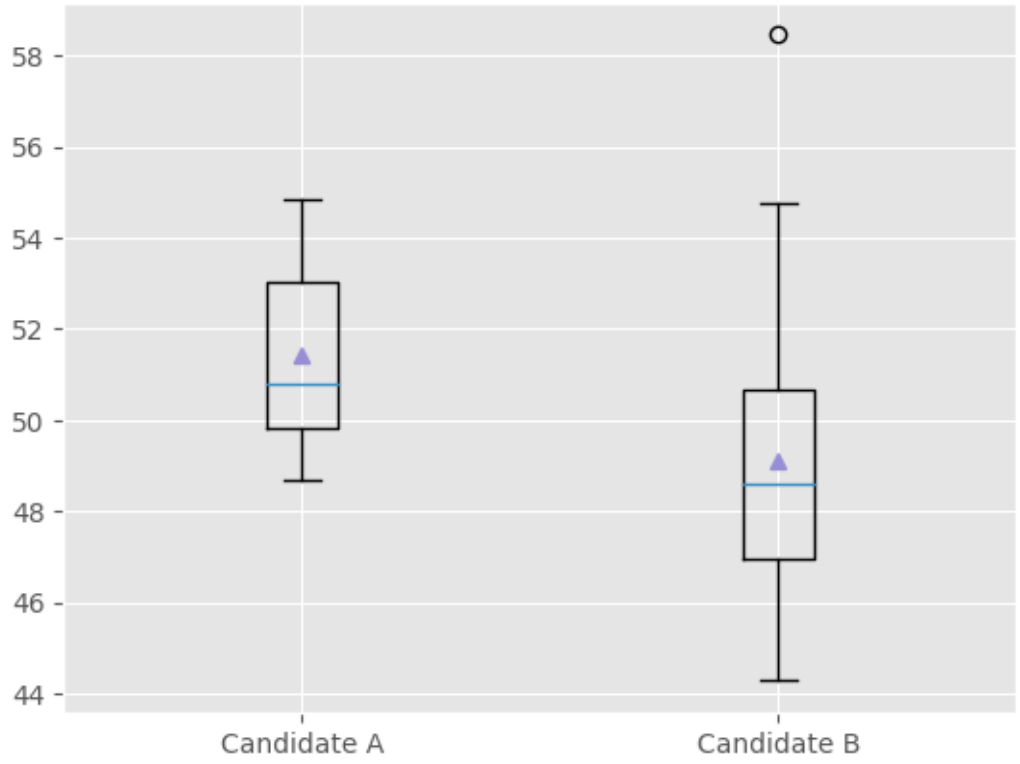
```
plt.figure(figsize=(8,6))
# Add a main title
plt.title("Plot of Age vs. Height (in cms)\n",fontsize=20,
fontstyle='italic')

# X- and Y-label with fontsize
plt.xlabel("Age (years)",fontsize=16)
plt.ylabel("Height (cms)",fontsize=16)
# Turn on grid
plt.grid (True)

# Set Y-axis limit
plt.ylim(100,200)

# X- and Y-axis ticks customization with fontsize and placement
plt.xticks([i*5 for i in range(12)],fontsize=15)
plt.yticks(fontsize=15)

# Main plotting function with choice of color, marker size, and marker
edge color
plt.scatter(x=a,y=h,c='orange',s=150,edgecolors='k')

# Adding bit of text to the plot
plt.text(x=15,y=105,s="Height increaes up to around \n20 years and then
tapers off",fontsize=15,
         rotation=30, linespacing=2)
plt.text(x=22,y=185,s="Nobody has a height beyond 180 cm",fontsize=15)

# Adding a vertical line
plt.vlines(x=20,ymin=100,ymax=180,linestyles='dashed',color='blue',lw=3
)

# Adding a horizontal line
plt.hlines(y=180,xmin=0,xmax=55,linestyles='dashed',color='red',lw=3)

# Adding a legend
plt.legend(['Height in cms'],loc=2,fontsize=14)

# Final show method
plt.show()
```

OUTPUT:



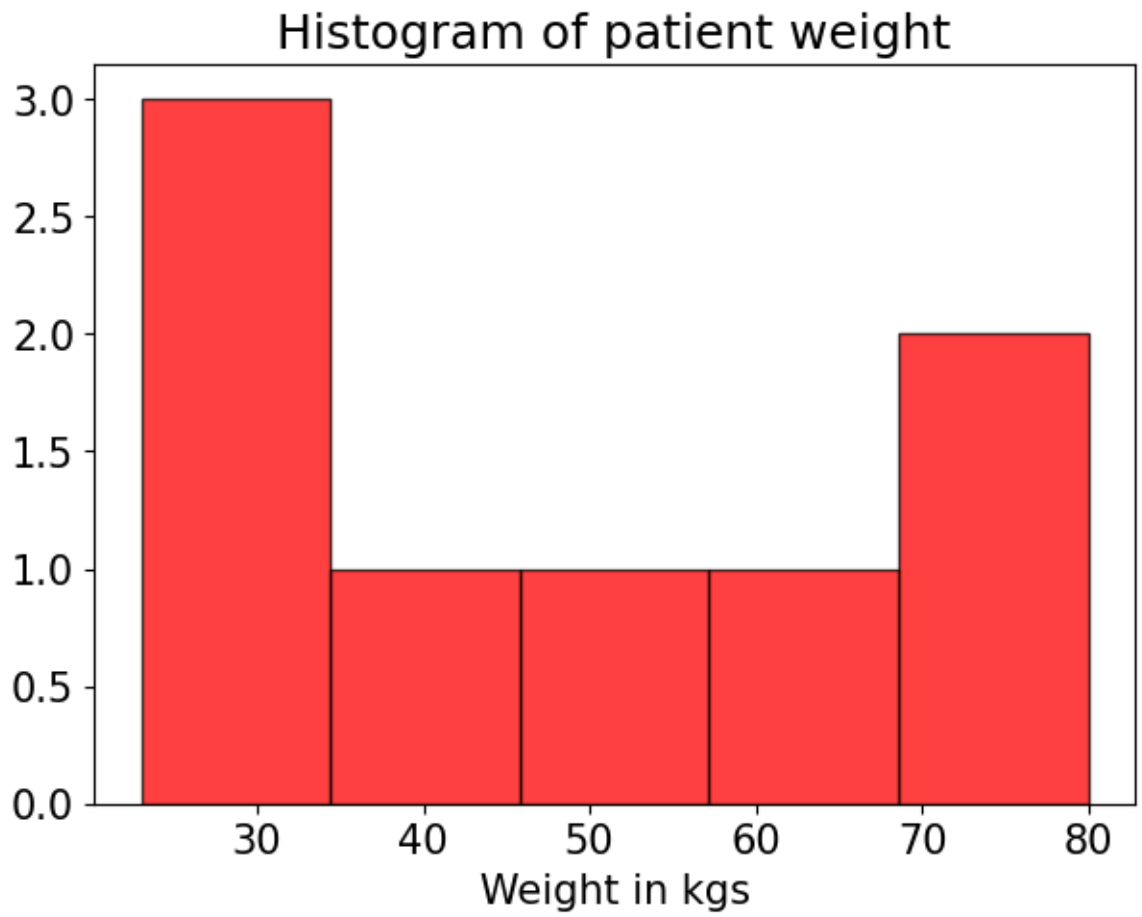

```
days = np.arange(1,31)
candidate_A = 50+days*0.07+2*np.random.randn(30)
candidate_B = 50-days*0.1+3*np.random.randn(30)


plt.style.use('ggplot')
# Note how to convert default numerical x-axis ticks to the list of
string by passing two lists
plt.boxplot(x=[candidate_A,candidate_B],showmeans=True)
plt.grid(True)
plt.xticks([1,2],['Candidate A','Candidate B'])
#plt.yticks(fontsize=15)
plt.show()
```

OUTPUT:




```
# Using a Histogram
plt.figure(figsize=(7,5))
# Main plot function 'hist'
plt.hist(w,color='red',edgecolor='k', alpha=0.75,bins=5)
plt.title("Histogram of patient weight",fontsize=18)
plt.xlabel("Weight in kgs",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```

OUTPUT:



SEABORN

```
import seaborn as sns
df = pd.read_csv('/content/wine.csv')
df.head()
```

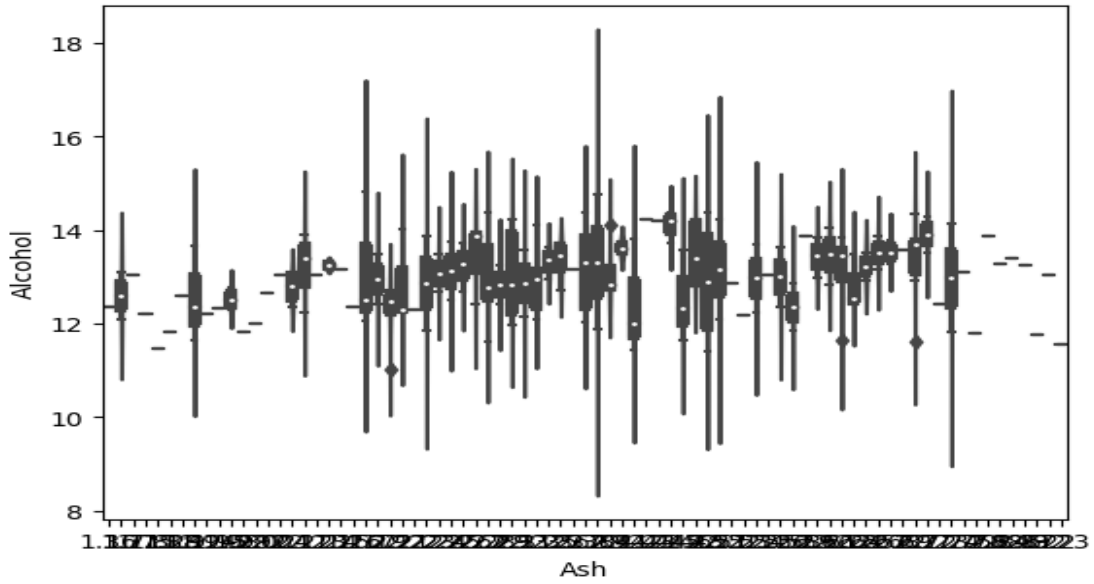
OUTPUT:

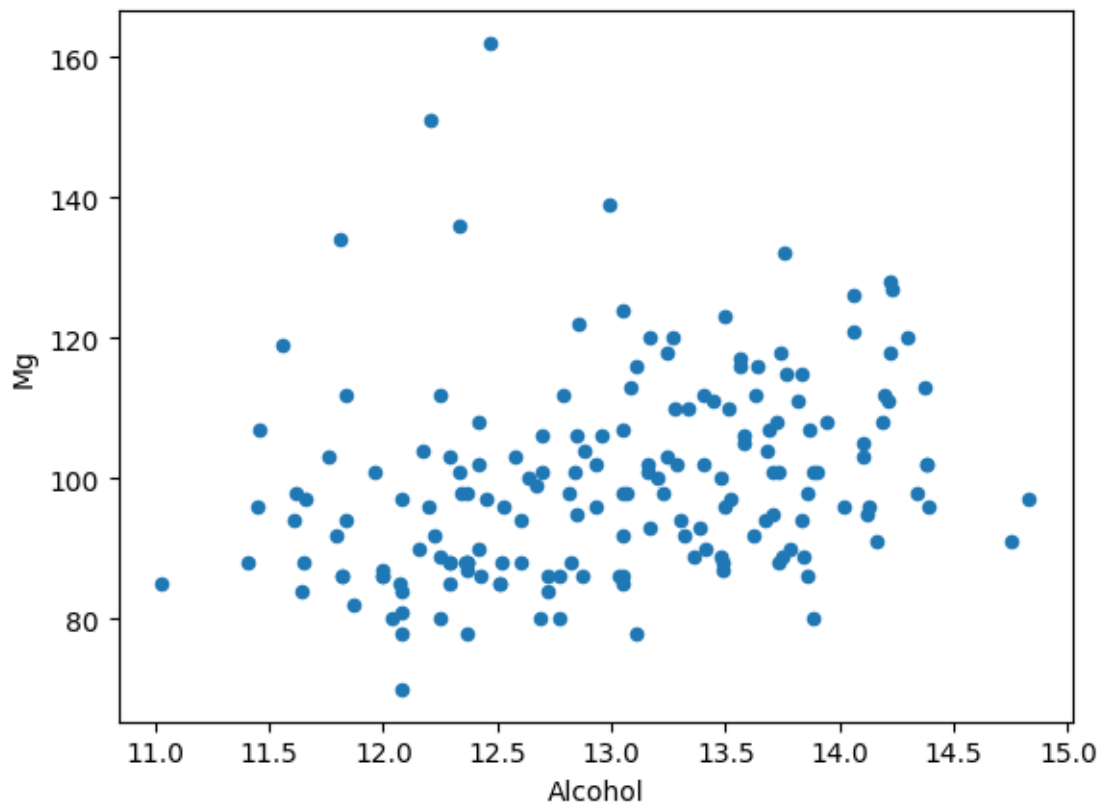
	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065

Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
# regplot

df.plot.scatter('Alcohol','Mg')
plt.show()
```

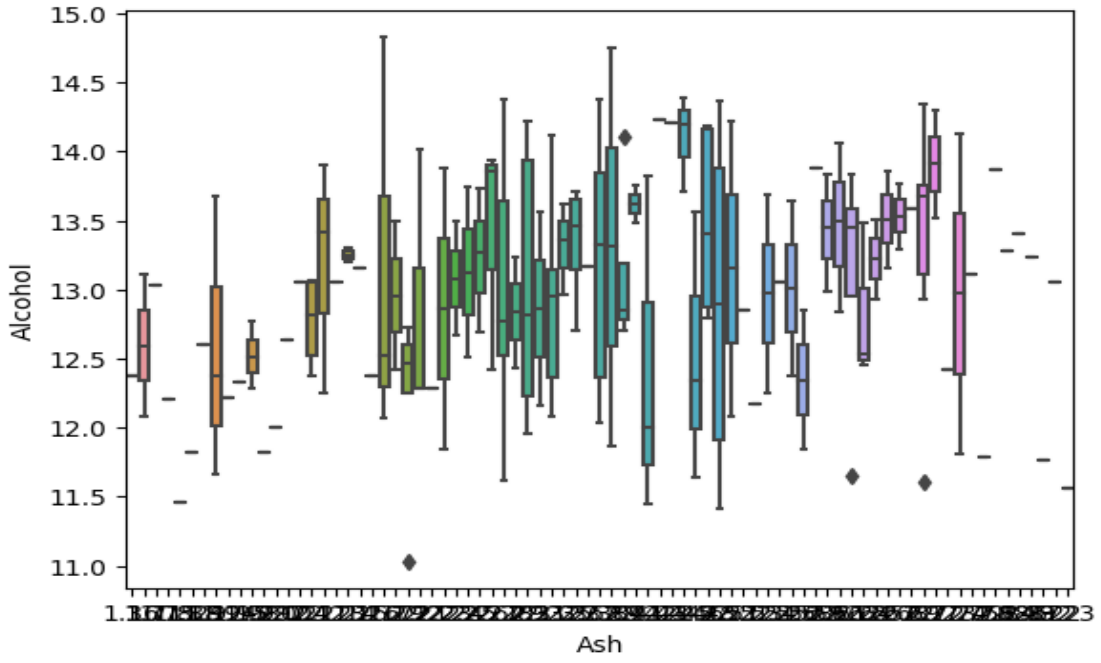
OUTPUT:






```
# Boxplot
sns.boxplot(x='Ash',y='Alcohol',data=df)
plt.show()
```

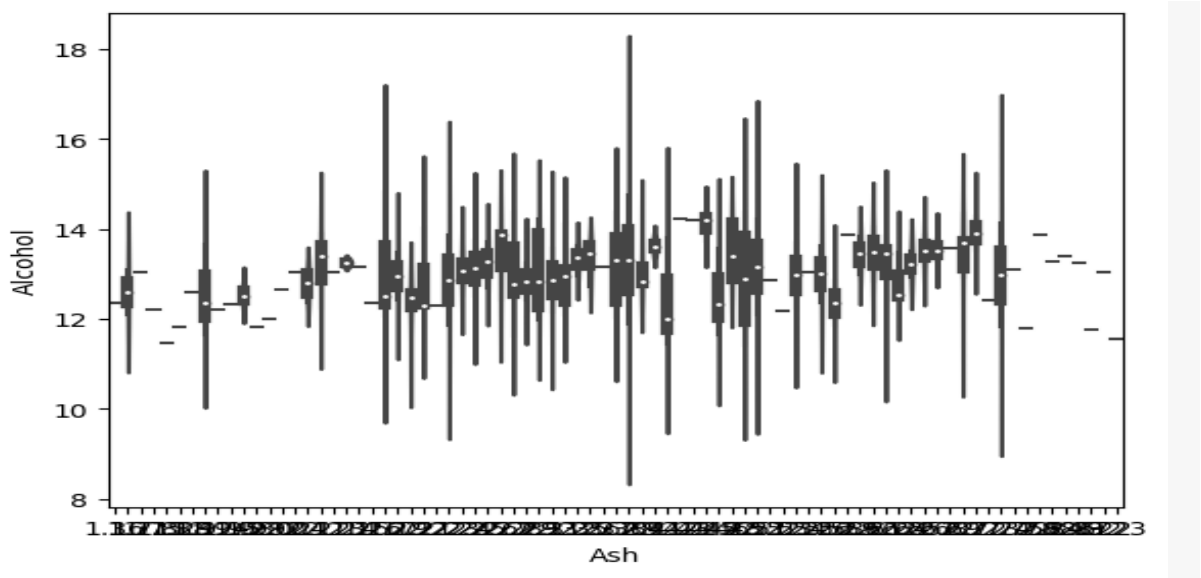
OUTPUT:




```
# Violinplot
```

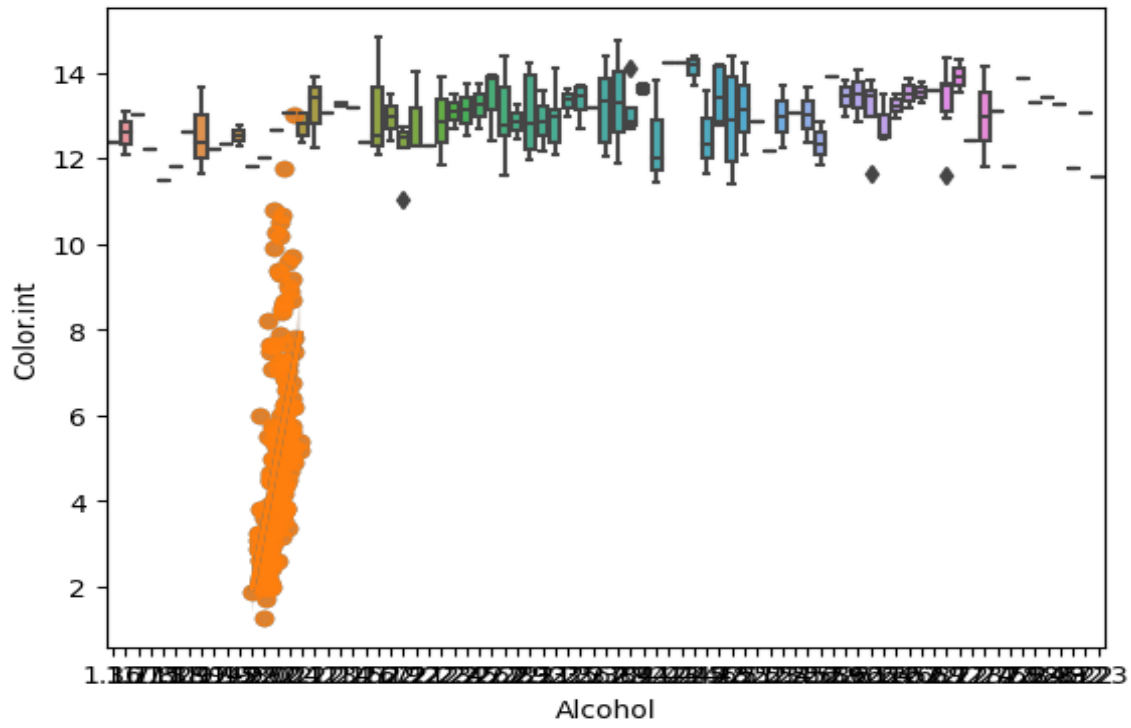
```
sns.violinplot(x='Ash',y='Alcohol',data=df)
plt.show()
```

OUTPUT:




```
# Regplot
sns.regplot(x='Alcohol',y='Color.int',data=df)
plt.show()
```

OUTPUT:

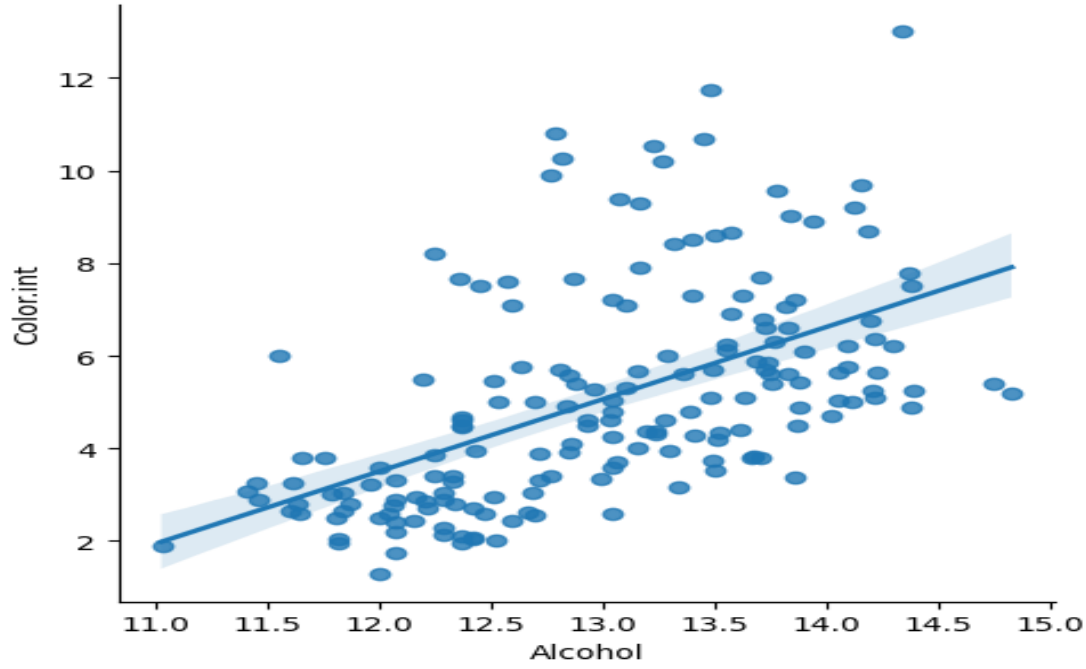


```
# using a pie chart
plt.pie(a)
plt.show()
```

OUTPUT:

```
# Implot
sns.lmplot(x='Alcohol',y='Color.int',data=df)
plt.show()
```

OUTPUT:



CODE-2:

CODE ASSIGNMENTS 02: Reading data and identifying variables, finding maximum likelihood values, density estimation

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal
%matplotlib inline
covariance = np.array([[0.14, -0.3, 0.0, 0.2],
                       [-0.3, 1.16, 0.2, -0.8],
                       [0.0, 0.2, 1.0, 1.0],
                       [0.2, -0.8, 1.0, 2.0]])
precision = np.linalg.inv(covariance)  print(precision)
```

OUTPUT:

```
[[ 60.   50.  -48.   38. ]
 [ 50.   50.  -50.   40. ]
 [-48.  -50.   52.4 -41.4]
 [ 38.   40.  -41.4  33.4]]
```

```
def generate_pair():
    return np.random.multivariate_normal([0.8, 0.8], [[0.1, -0.1],[-
0.1, 0.12]])
mu_t = generate_pair()
print(mu_t)
```

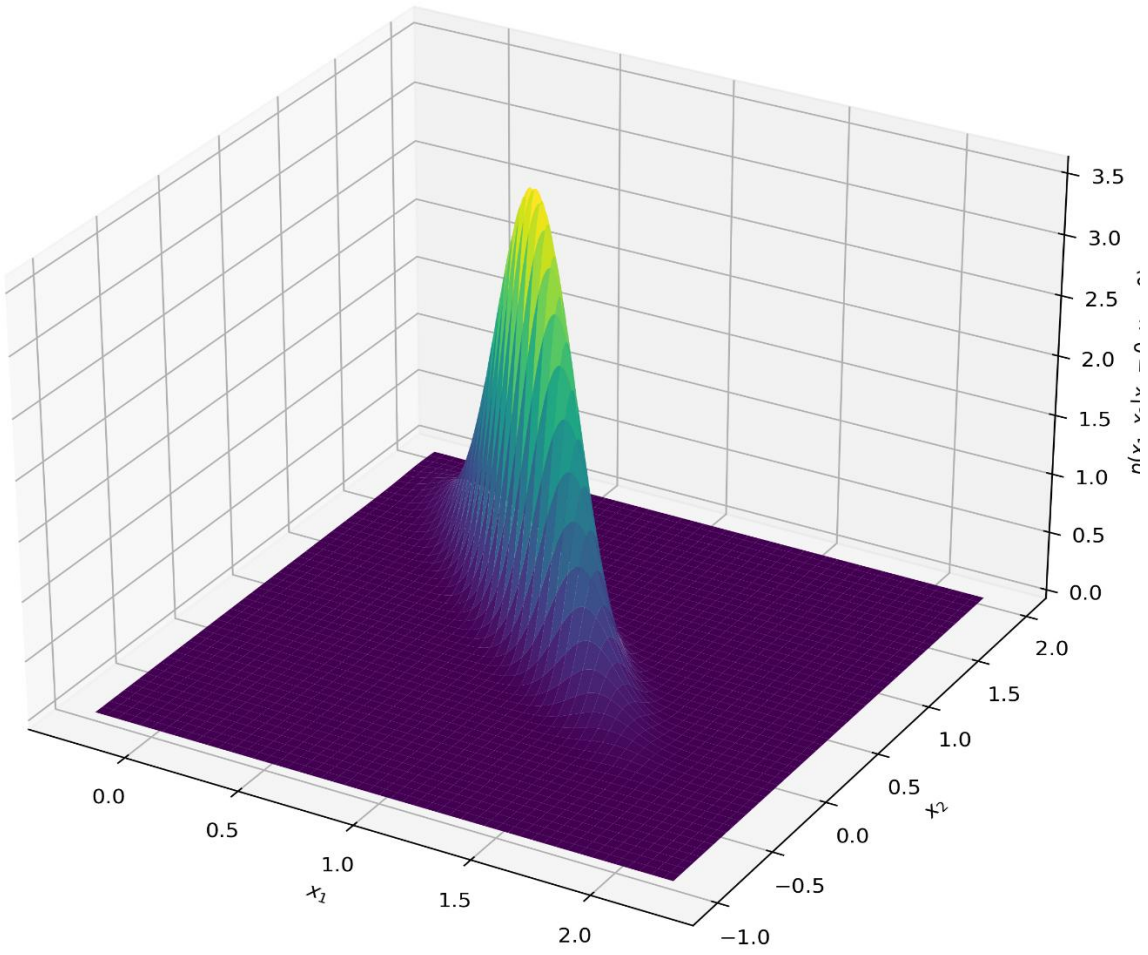
OUTPUT:

```
[0.29005878 0.89580309]
```

```
x, y = np.mgrid[-0.25:2.25:.01, -1:2:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x
pos[:, :, 1] = y
mu_p = [0.8, 0.8]
cov_p = [[0.1, -0.1], [-0.1, 0.12]]
z = multivariate_normal(mu_p, cov_p).pdf(pos)

fig = plt.figure(figsize=(10, 10), dpi=300)
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x, y, z, cmap=plt.cm.viridis)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
ax.set_zlabel('$p(x_1, x_2 | x_3=0, x_4=0)$')
```

```
plt.savefig('cond_mvg.png', bbox_inches='tight', dpi=300)
plt.show()
```

OUTPUT :




```
N=1000
import numpy as np
data = np.random.multivariate_normal([0.28, 1.18], [[2.0, 0.8], [0.8,
4.0]], N)
data
np.savetxt('data.txt',data)
```

```
import pandas as pd
data = pd.read_table('data.txt')
data = np.loadtxt('data.txt')
data
```

OUTPUT:

```
array([[ 1.39239302, 2.08421806], [ 0.02993624, 1.97107962], [-
0.53863737, 2.52411385], ..., [ 1.19302699, 5.7822559 ], [ 0.84779616,
1.34576194], [-0.45093109, -2.3587619 ]])
```

```
mu_ml = data.mean(axis=0)
x = data - mu_ml
cov_ml = np.dot(x.T, x) / N
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)
print(mu_ml)
print(cov_ml)
print(cov_ml_unbiased)
```

OUTPUT:

```
[0.25541076 1.18247388]
[[2.02413919 0.81163564]
 [0.81163564 4.17326348]]
[[2.02616535 0.81244808]
 [0.81244808 4.17744092]]
```

```
def seq_ml(data):
    mus = [np.array([[0], [0]])]
    for i in range(N):
        x_n = data[i].reshape(2, 1)
        mu_n = mus[-1] + (x_n-mus[-1]) / (i + 1)
        mus.append(mu_n)
    return mus
mus_ml = seq_ml(data)
print(mus_ml[-1])
```

OUTPUT:

```
[[0.24354648]
 [1.12578895]]
```

```
mu_p = np.array([[0.28], [1.18]])
cov_p = np.array([[0.1, -0.1], [-0.1, 0.12]])
cov_t = np.array([[2.0, 0.8], [0.8, 4.0]])
print(mu_p,cov_p,cov_t)
```

OUTPUT:

```
[[0.28]
 [1.18]] [[ 0.1  -0.1 ]
 [-0.1   0.12]] [[2.  0.8]
 [0.8 4. ]]
```

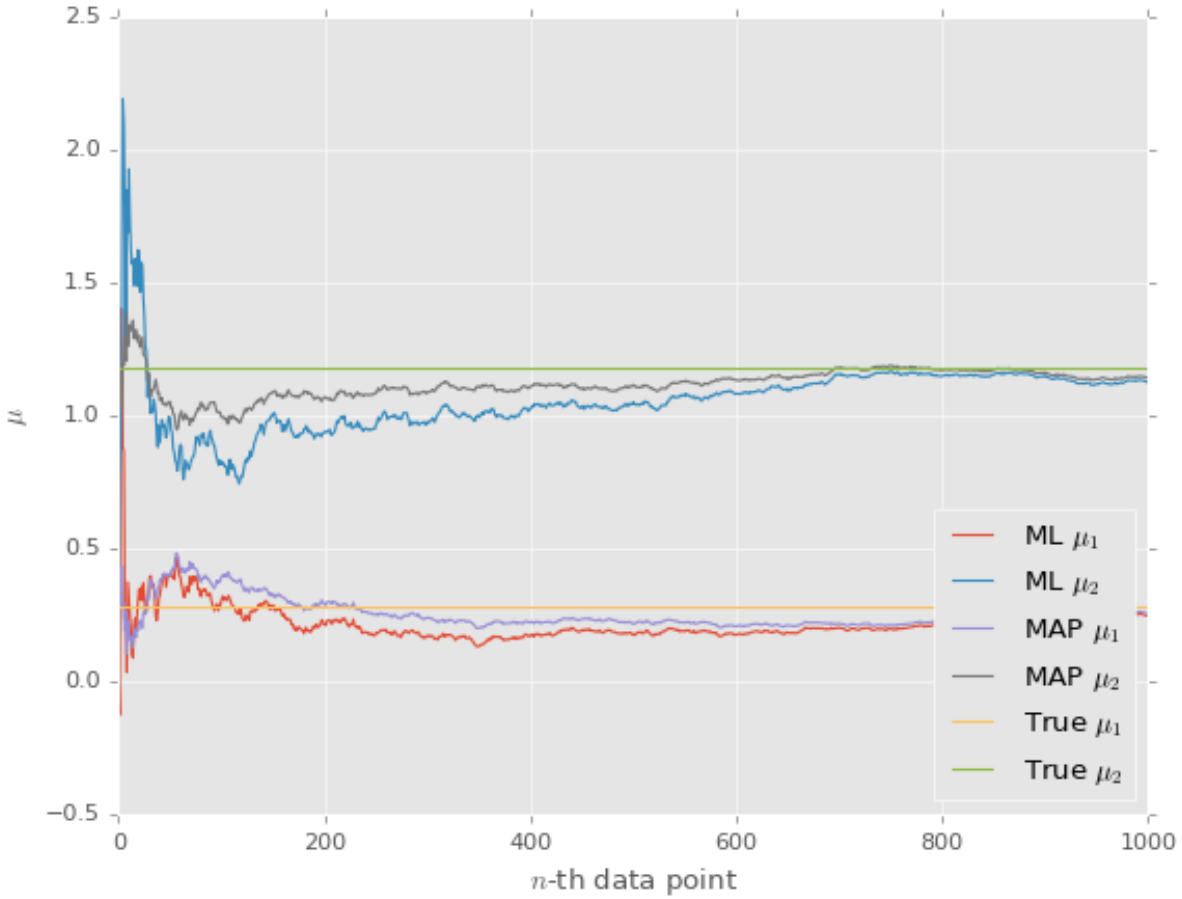
```
def seq_map(data, mu_p, cov_p, cov_t):
    mus, covs = [mu_p], [cov_p]
    for x in data:
        x_n = x.reshape(2, 1)
        cov_n = np.linalg.inv(np.linalg.inv(covs[-1]) +
np.linalg.inv(cov_t))
        mu_n = cov_n.dot(np.linalg.inv(cov_t).dot(x_n) +
np.linalg.inv(covs[-1]).dot(mus[-1]))
        mus.append(mu_n)
        covs.append(cov_n)

    return mus, covs
mus_map, covs_map = seq_map(data, mu_p, cov_p, cov_t)
print(mus_map[-1])
```

OUTPUT:

```
[[0.25320413]
 [1.14164058]]
```

```
X = np.arange(N+1)
mus1_ml = [mu[0] for mu in mus_ml]
mus2_ml = [mu[1] for mu in mus_ml]
mus1_map = [mu[0] for mu in mus_map]
mus2_map = [mu[1] for mu in mus_map]
mus1_t = [0.28] * (N+1)
mus2_t = [1.18] * (N+1)
plt.style.use('ggplot')
plt.plot(X, mus1_ml, label='ML $\mu_1$')
plt.plot(X, mus2_ml, label='ML $\mu_2$')
plt.plot(X, mus1_map, label='MAP $\mu_1$')
plt.plot(X, mus2_map, label='MAP $\mu_2$')
plt.plot(X, mus1_t, label='True $\mu_1$')
plt.plot(X, mus2_t, label='True $\mu_2$')
plt.xlabel('$n$-th data point')
plt.ylabel('$\mu$')
plt.legend(loc=4)
plt.savefig('seq_learning.png', bbox_inches='tight', dpi=300)
plt.show()
```

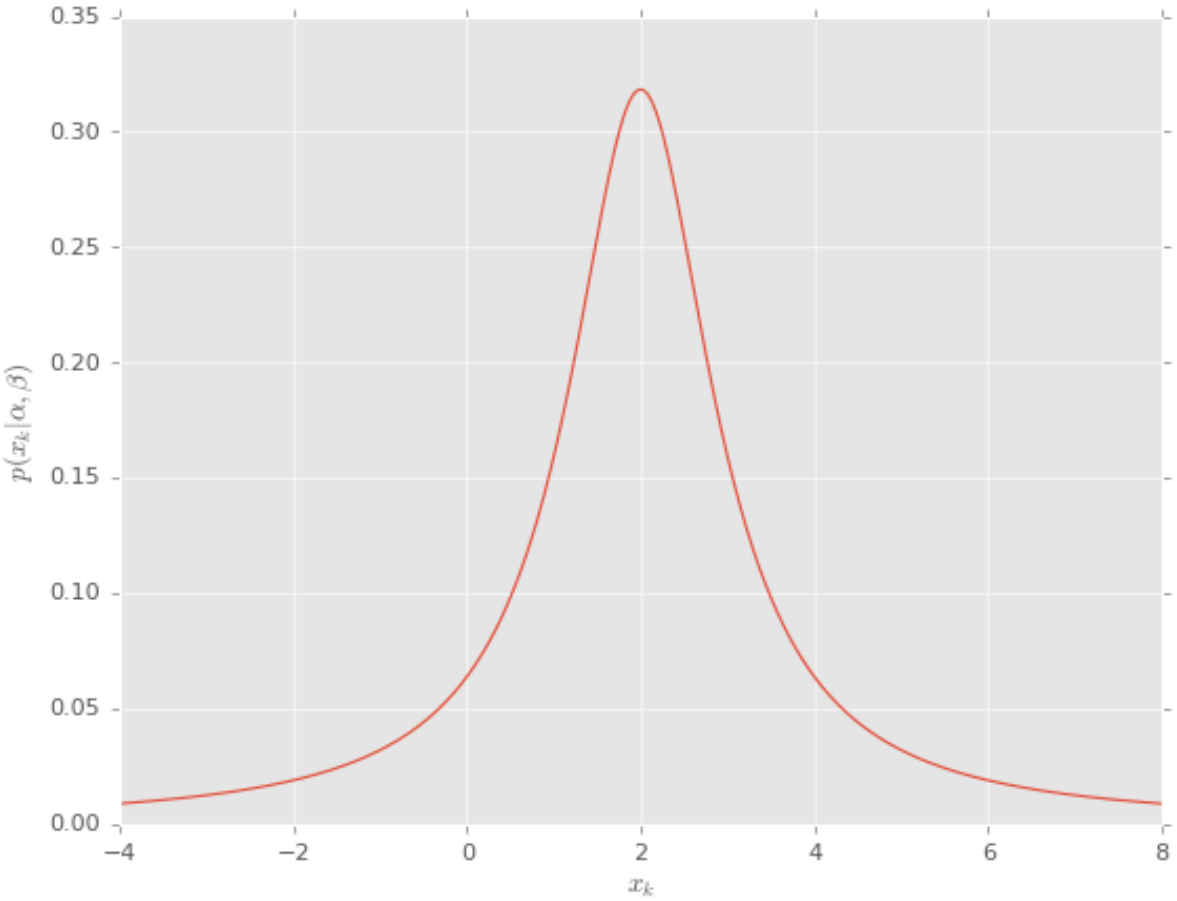
OUTPUT:




```
def p_xk(x, alpha, beta):
    return beta / (np.pi * (beta**2 + (x-alpha)**2))
```

```
x = np.linspace(-4, 8, num=1000)
probs = p_xk(x, 2, 1)
plt.plot(x, probs)
plt.xlabel('$x_k$')
plt.ylabel(r'$p(x_k | \alpha, \beta)$')
plt.savefig('prob_xk.png', bbox_inches='tight', dpi=300)
plt.show()
```

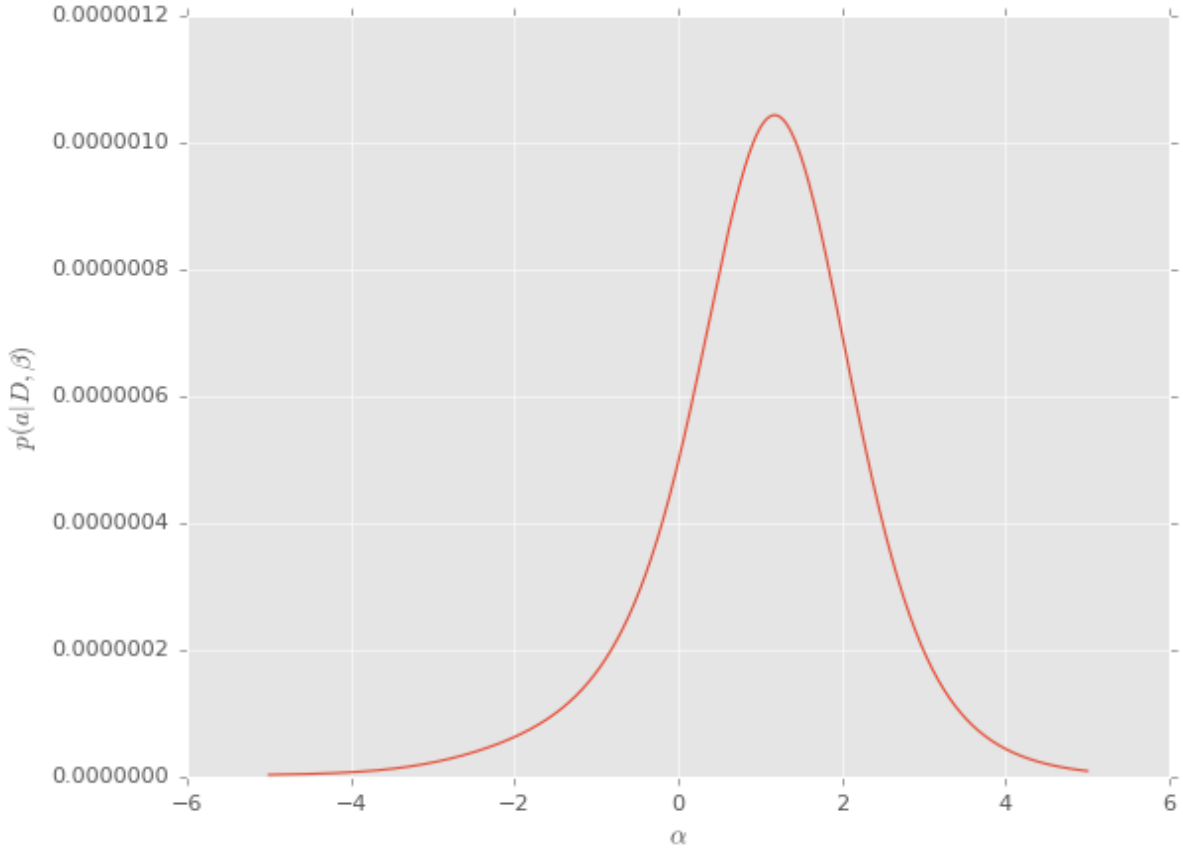
OUTPUT:




```
def p_a(x, alpha, beta):
    return np.product(beta / (np.pi * beta**2 + (x-alpha)**2))
```

```
D = np.array([4.8, -2.7, 2.2, 1.1, 0.8, -7.3])
alphas = np.linspace(-5, 5, num=1000)
beta = 1
likelihoods = [p_a(D, alpha, beta) for alpha in alphas]
plt.plot(alphas, likelihoods)
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$p(a | D, \beta)$')
plt.savefig('prob_a.png', bbox_inches='tight', dpi=300)
plt.show()
```

OUTPUT:




```
print(D.mean())
print(alphas[np.argmax(likelihoods)])
```

OUTPUT:

```
-0.18333333333333326
1.1761761761761758
```

```
alpha_t = np.random.uniform(0, 10)
beta_t = np.random.uniform(1, 2)
print(alpha_t, beta_t)
```

OUTPUT:

```
0.7361544757681782 1.6232156606522077
```

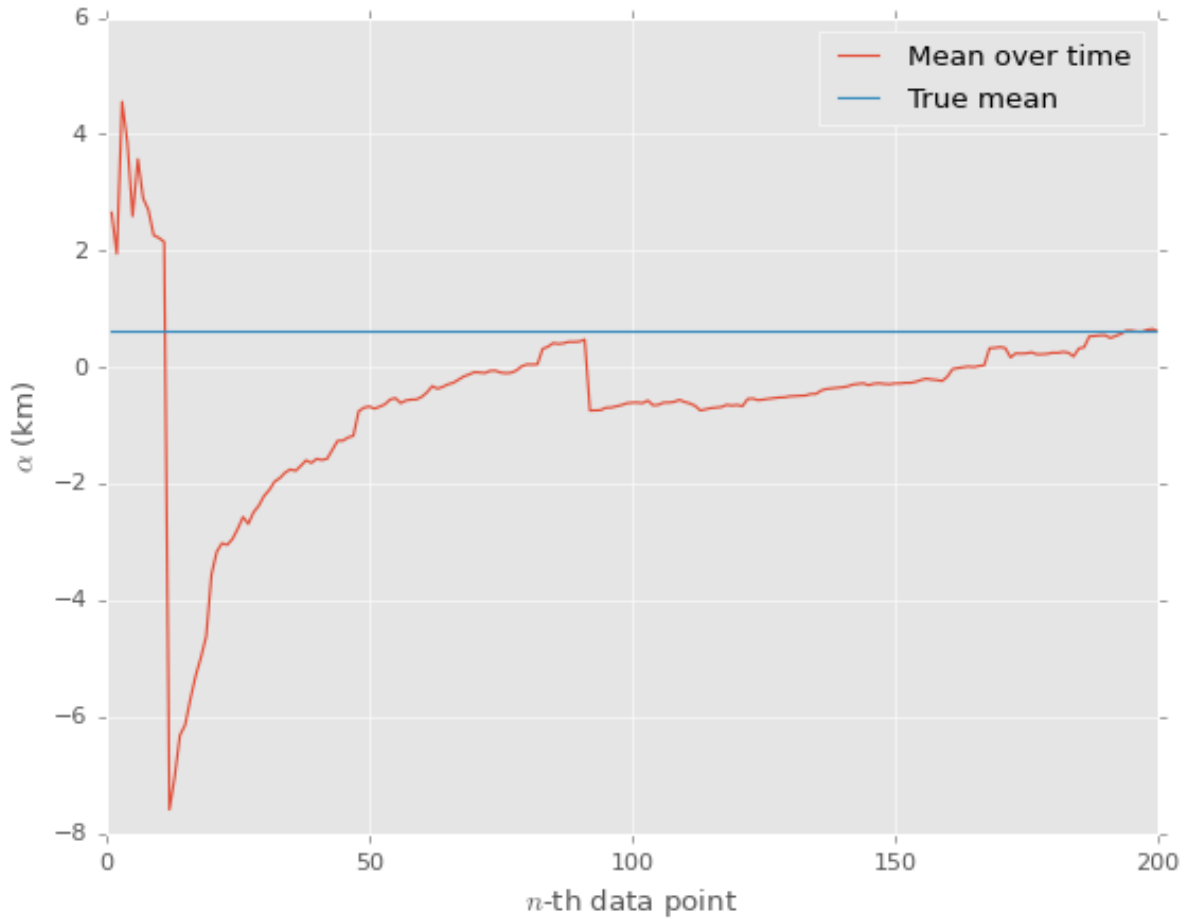
```
def location(angle, alpha, beta):
    return beta * np.tan(angle) + alpha
```

```
N = 200
```

```
angles = np.random.uniform(-np.pi/2, np.pi/2, N)
locations = np.array([location(angle, alpha_t, beta_t) for angle in
angles])
```

```
mus = [locations[:i + 1].mean() for i in range(N)]
mean = [locations.mean()] * (N)
X = np.arange(1, N + 1)
plt.style.use('ggplot')
plt.plot(X, mus, label='Mean over time')
plt.plot(X, mean, label='True mean')
plt.xlabel('$n$-th data point')
plt.ylabel(r'$\alpha$ (km)')
plt.legend()
plt.savefig('mean_x.png', bbox_inches='tight', dpi=300)
plt.show()
```

OUTPUT:




```
print(locations.mean())
```

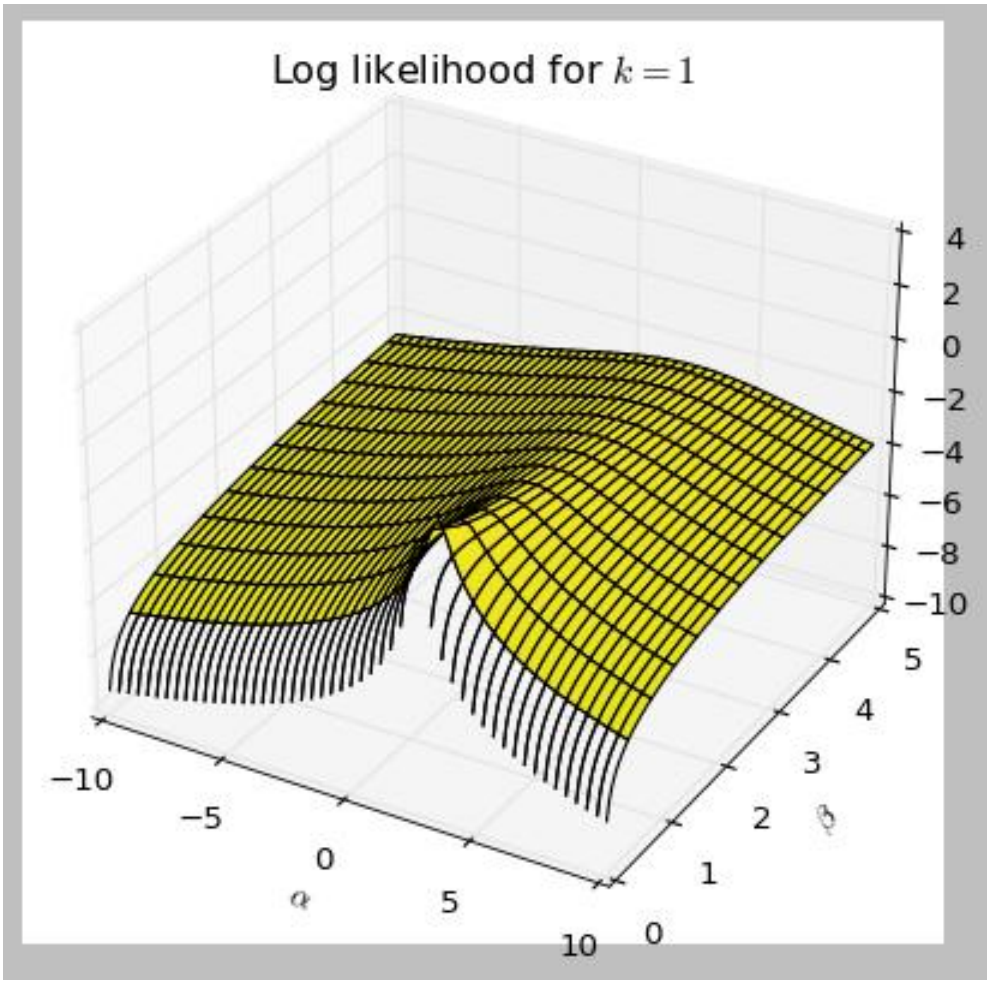
OUTPUT:

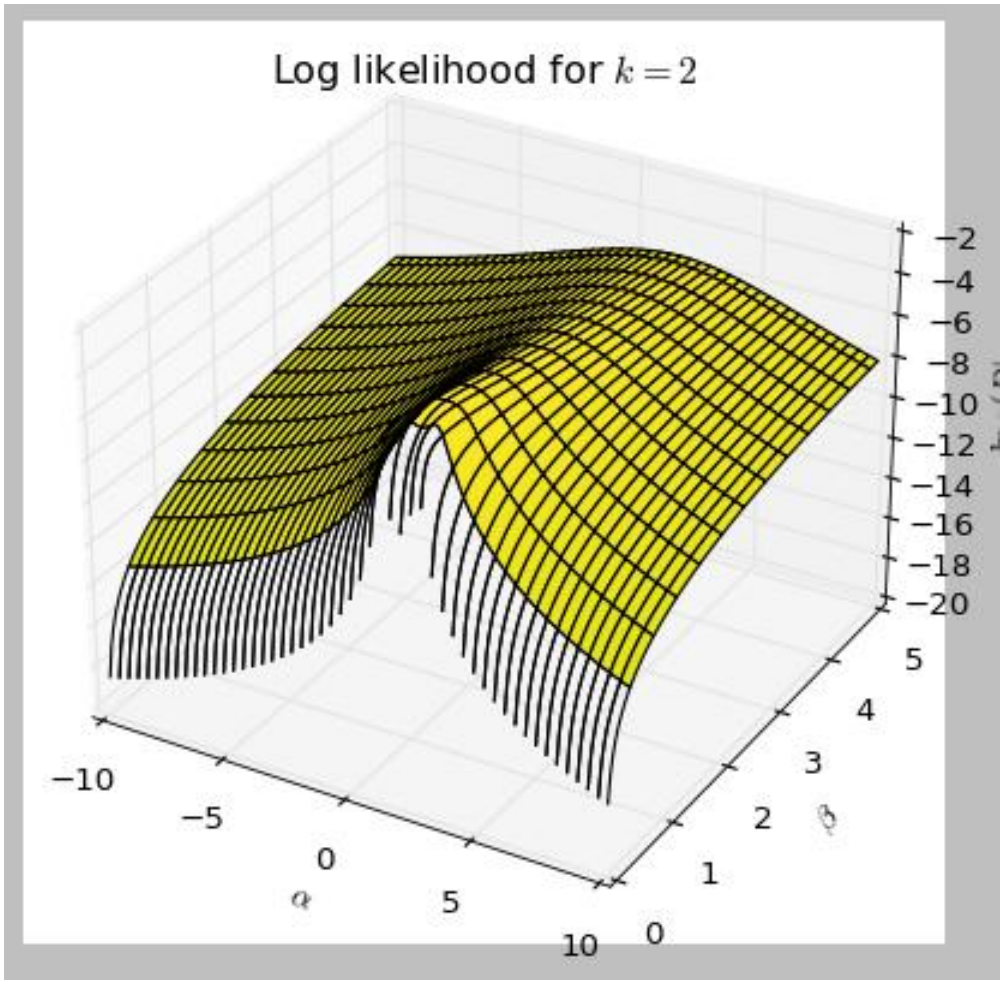
0.616993957733935

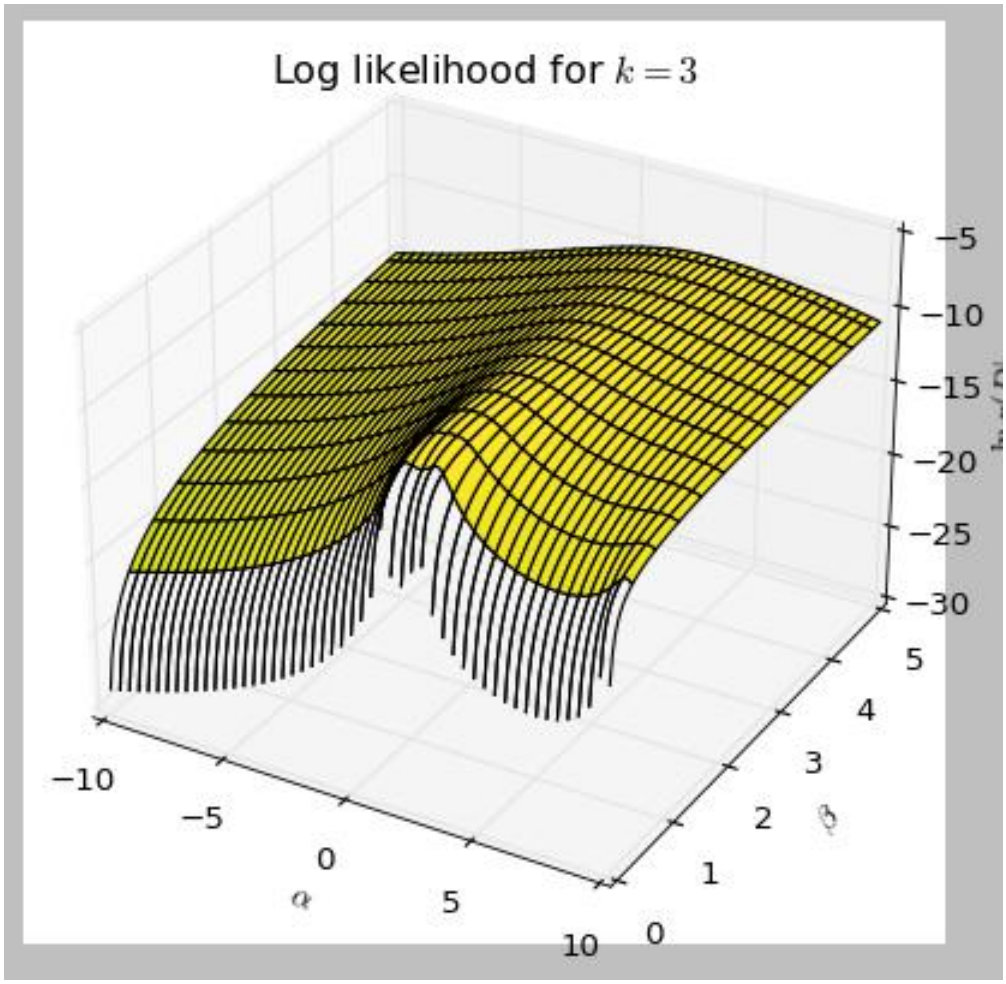
```
plt.style.use('classic')
ks = [1, 2, 3, 20]
alphas, betas = np.mgrid[-10:10:0.04, 0:5:0.04]
for k in ks:
    x = locations[:k]
    # We only have to calculate the constant once
    likelihood = k * np.log(betas/np.pi)
    for loc in x:
        likelihood -= np.log(betas**2 + (loc - alphas)**2)

    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.plot_surface(alphas, betas, likelihood, cmap=plt.cm.viridis,
vmin=-200, vmax=likelihood.max())
    plt.xlabel(r'$\alpha$')
    plt.ylabel(r'$\beta$')
    ax.set_zlabel('$\ln p(D | \alpha, \beta)$')
    plt.title('Log likelihood for $k = {}$'.format(k))
    plt.savefig('logl_{}.png'.format(k), bbox_inches='tight', dpi=300)
    plt.show()
```

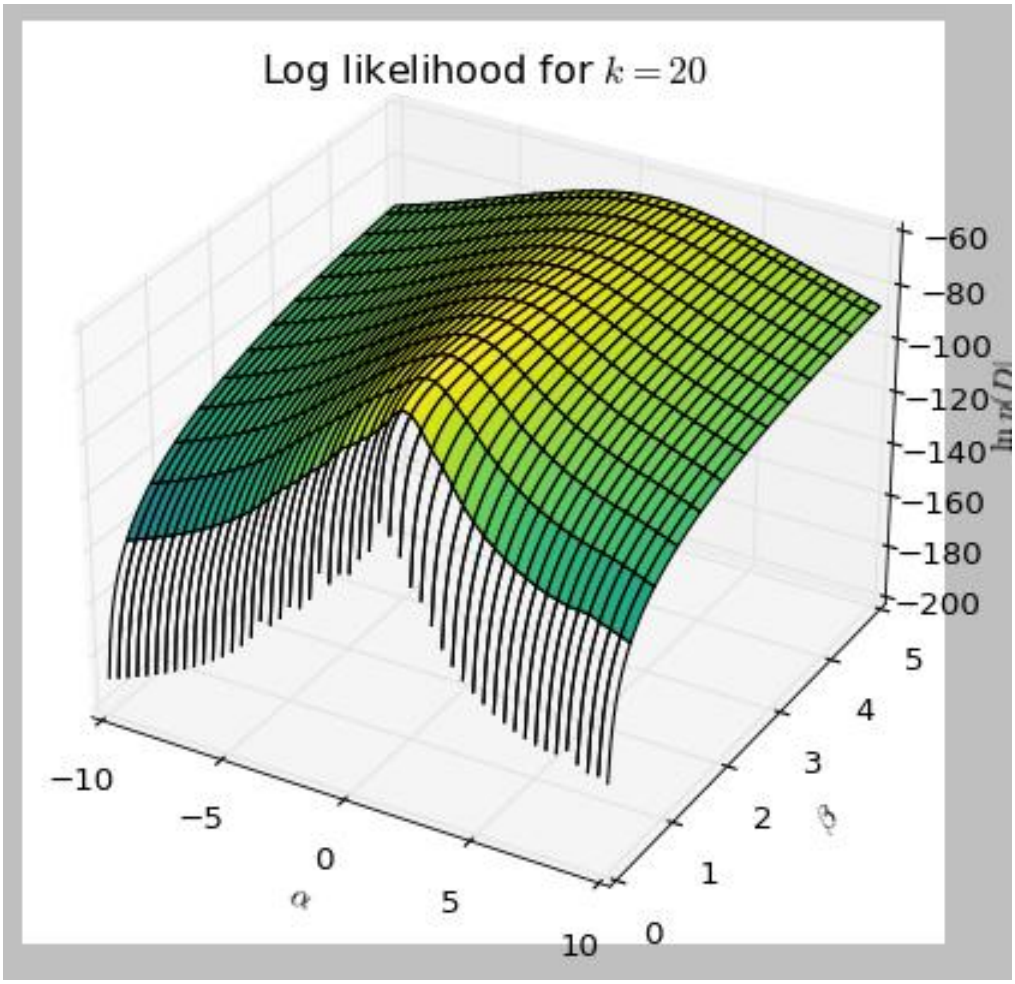
OUTPUT:

```
<ipython-input-64-157bd71ddba4>:7: RuntimeWarning: divide by zero
encountered in log
  likelihood = k * np.log(betas/np.pi)
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/proj3d.py:180:
RuntimeWarning: invalid value encountered in true_divide
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```


Log likelihood for $k = 1$
4
2
0
−2
−4
−6
−8
−10
−10
−5
0
5
10
a
0
1
2
3
4
5
b


Log likelihood for $k = 2$
−2
−4
−6
−8
−10
−12
−14
−16
−18
−20
−10
−5
0
5
10
a
0
1
2
3
4
5
b


Log likelihood for $k = 3$
−5
−10
−15
−20
−25
−30
−10
−5
0
5
10
a
0
1
2
3
4
5
b




```
from scipy.optimize import fmin

def log_likelihood(params, locations):
    alpha, beta = params
    likelihood = len(locations) * np.log(beta/np.pi)
    for loc in locations:
        likelihood -= np.log(beta**2 + (loc - alpha)**2)
    return -likelihood

def plot_maximize_logl(data, alpha_t, beta_t):
    alphas, betas = [], []
    x = np.arange(len(data))
    for k in x:
        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))
        alphas.append(alpha)
        betas.append(beta)

    plt.style.use('ggplot')
    plt.plot(x, alphas, label=r'$\alpha$')
    plt.plot(x ,betas, label=r'$\beta$')
    plt.plot(x, [alpha_t]*len(data), label=r'$\alpha_t$')
    plt.plot(x, [beta_t]*len(data), label=r'$\beta_t$')
```

```
    plt.xlabel('$k$')
    plt.ylabel('location (km)')
    plt.legend()
    plt.savefig('plots/min_logl.png', bbox_inches='tight', dpi=300)
    plt.show()

    print(alphas[-1], betas[-1])
```

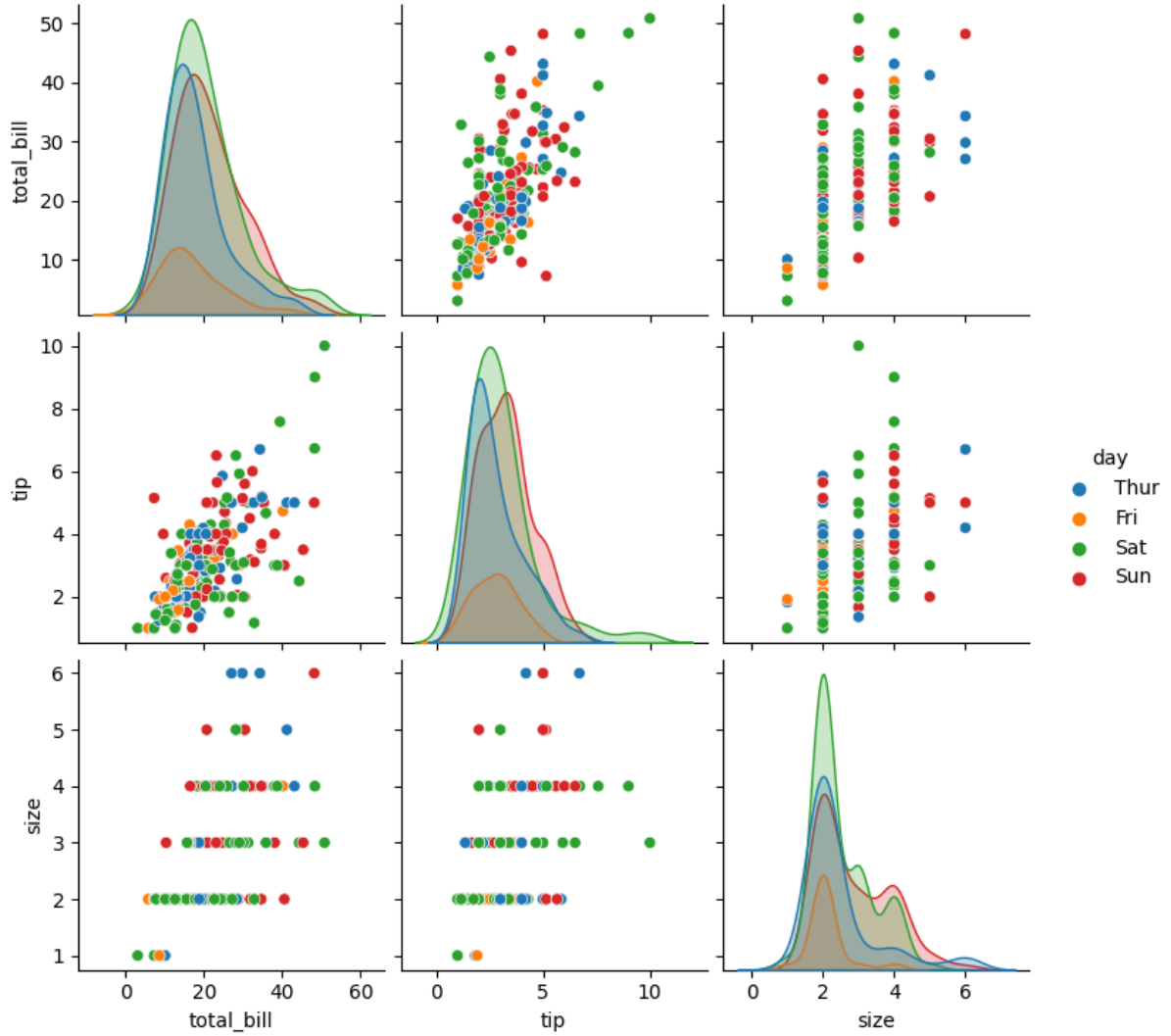
```
import pandas as pd
import numpy as np
a=pd.read_csv("/train.csv",sep=';')
print(a)
p=a['hour']
m=np.mean(p)
sd=np.std(p)#sigma value
var=np.var(p)#sigma square value
m, sd, var

import numpy as np
t=np.array(a['temp'])
tm=np.mean(t)
tsd=np.std(t)#sigma value/std.dev
tvar=np.var(t)#variance
x=29
l=np.log(np.sqrt(2*3.14))
e=np.log(tsd) #std.dev
f=(x-tm)**2 #mean
g=2*(tvar**2) #variance
h=f/g
i=-l-e
print(i-h)
```

OUTPUT:

```
-2.9860025235468406
```

```
import seaborn
import matplotlib.pyplot as plt
df = seaborn.load_dataset('tips')
seaborn.pairplot(df, hue='day')
plt.show()
```

OUTPUT:



CODE-3:

CODE ASSIGNMENTS 03: Implement Linear Regression Model Using US Housing Data

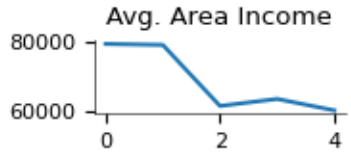
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

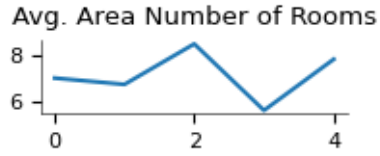
```
df =pd.read_csv("/content/USA_Housing.csv")
df.head()
```

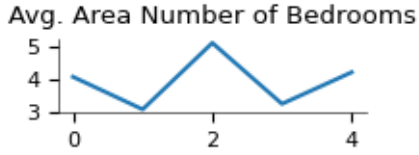
OUTPUT:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

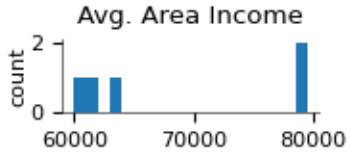
Values

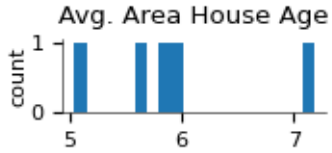


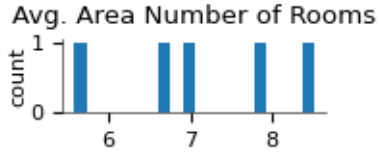


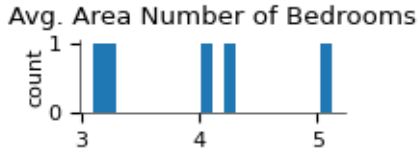


Distributions

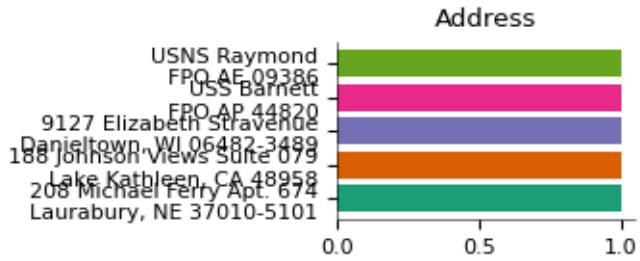




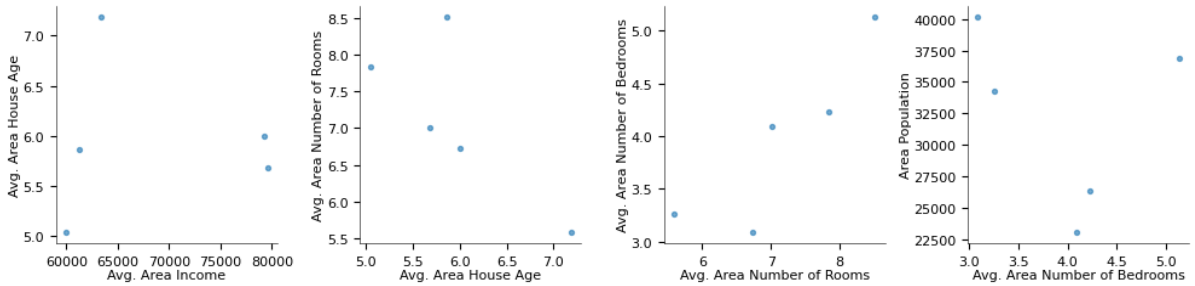




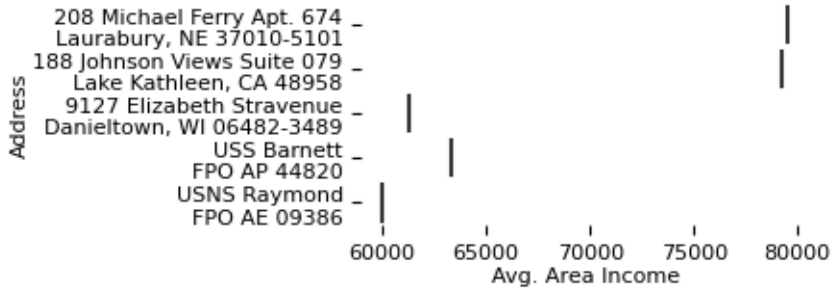
Categorical distributions

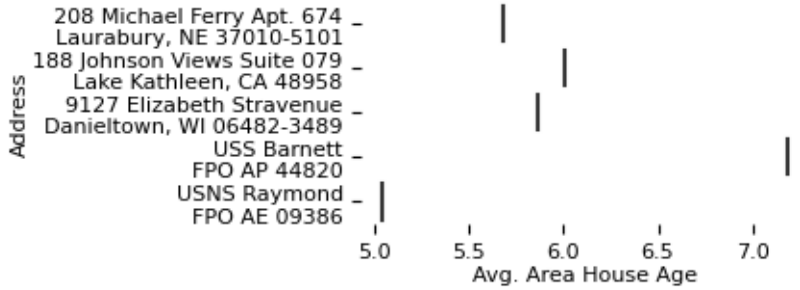


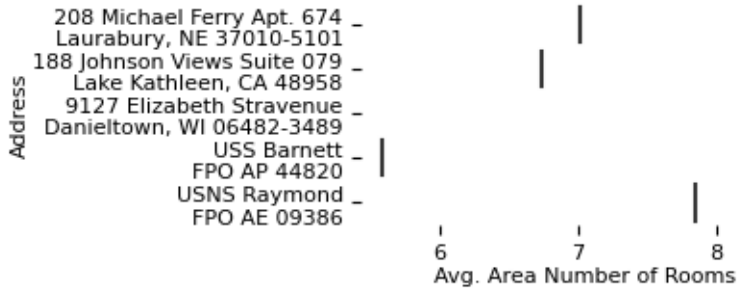
2-d distributions

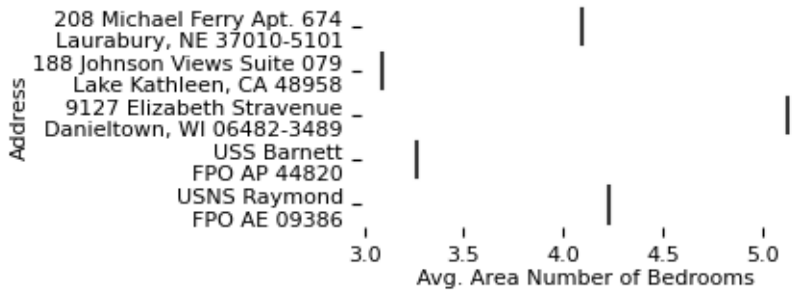


Faceted distributions










```
df.columns
```

OUTPUT:

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of
Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price',
'Address'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
from numpy.lib.function_base import percentile
df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

OUTPUT:

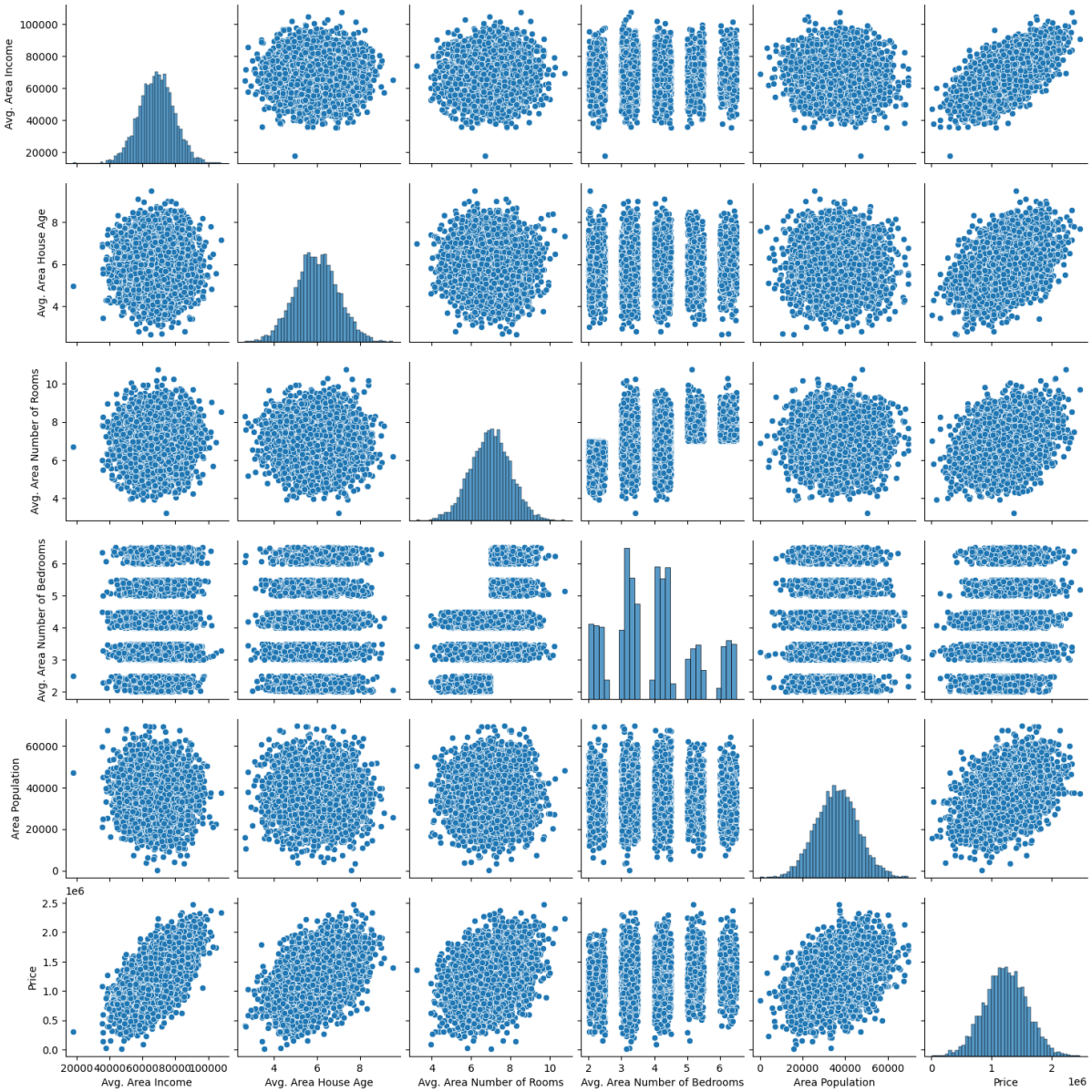
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
10%	55047.633980	4.697755	5.681951	2.310000	23502.845262	7.720318e+05
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
90%	82081.188283	7.243978	8.274222	6.100000	48813.618633	1.684621e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

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

OUTPUT:

```
<seaborn.axisgrid.PairGrid at 0x7a5280908820>
```




```
df.corr()
```

OUTPUT:

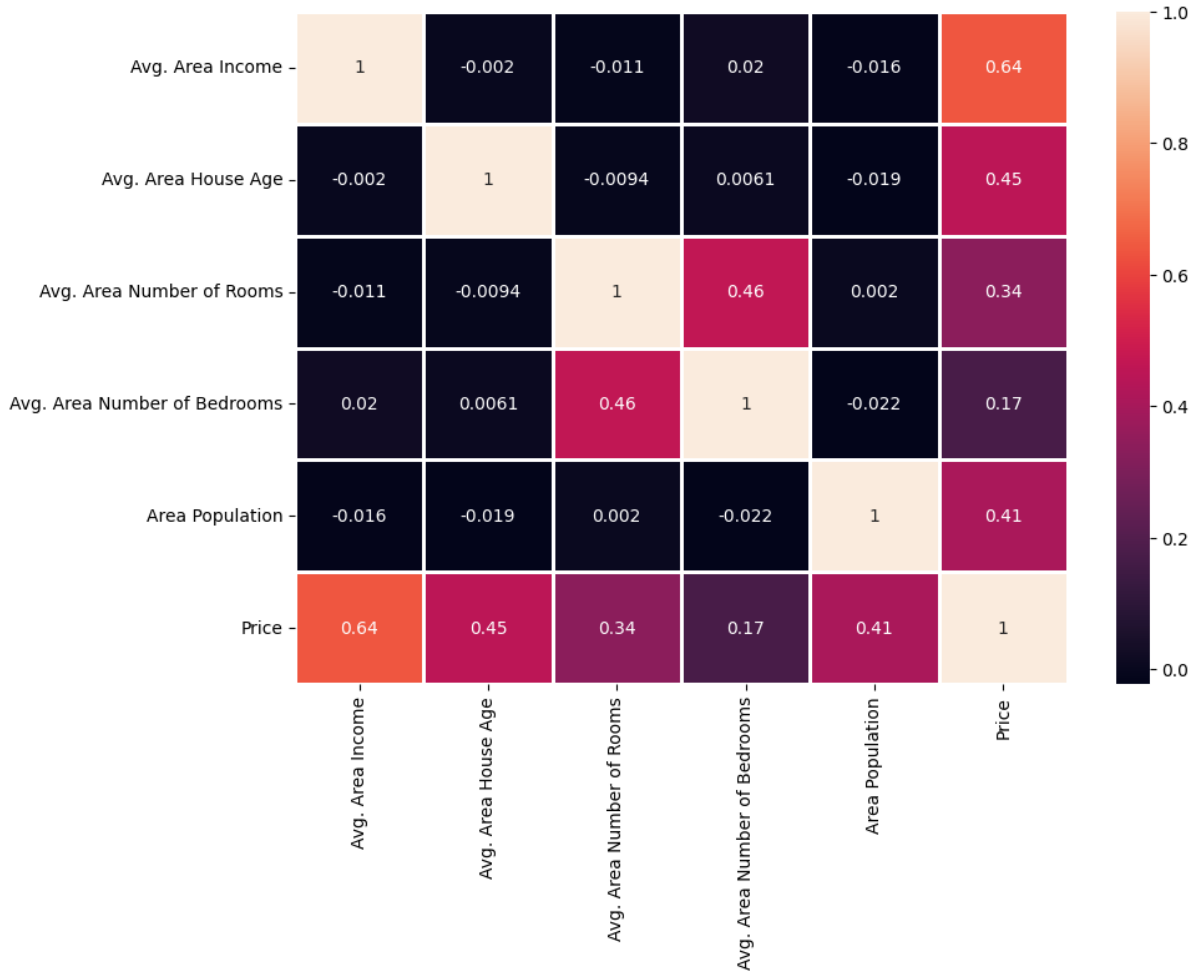
```
<ipython-input-13-2f6f6606aa2c>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  df.corr()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	0.171071	0.408556	1.000000

```
plt.figure(figsize=[10,7])
sns.heatmap(df.corr(),annot=True,linewidths=2)
```

OUTPUT:

```
<ipython-input-16-5fbd42e3bfca>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  sns.heatmap(df.corr(),annot=True,linewidths=2)
<Axes: >
```




```
l_column=list(df.columns)
len_feature =len(l_column)
l_column
```

OUTPUT:

```
['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of
Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price',
'Address']
```

```
x=df[l_column[0:len_feature-2]]
y=df[l_column[len_feature-2]]
print("feature set size:",x.shape)
print("variable size:",y.shape)
```

OUTPUT:

```
feature set size: (5000, 5)
variable size: (5000,)
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=123)
```

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
lm=LinearRegression()
```

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

OUTPUT:

 LinearRegression

```
LinearRegression()
```

```
print("The intercept term of the linear model:", lm.intercept_)
print("The coefficients of the linear model:", lm.coef_)
```

OUTPUT:

```
The intercept term of the linear model: -2631028.9017454907
The coefficients of the linear model: [2.15976020e+01 1.65201105e+05
1.19061464e+05 3.21258561e+03
 1.52281212e+01]
```

```
cdf = pd.DataFrame(data=lm.coef_, index=x_train.columns,
columns=["Coefficients"])
cdf
```

OUTPUT:

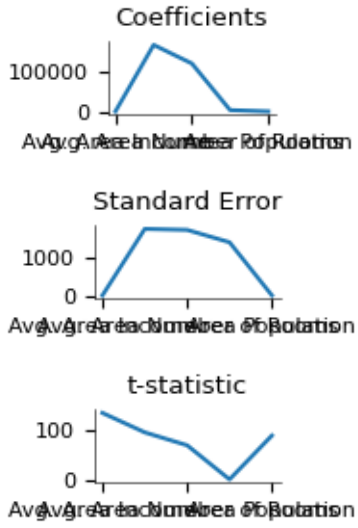
	Coefficients
Avg. Area Income	21.597602
Avg. Area House Age	165201.104954
Avg. Area Number of Rooms	119061.463868
Avg. Area Number of Bedrooms	3212.585606
Area Population	15.228121

```
n=x_train.shape[0]
k=x_train.shape[1]
dfN = n-k
train_pred=lm.predict(x_train)
train_error = np.square(train_pred - y_train)
sum_error=np.sum(train_error)
se=[0,0,0,0,0]
for i in range(k):
    r = (sum_error/dfN)
    r = r/np.sum(np.square(x_train[list(x_train.columns)[i]]-
x_train[list(x_train.columns)[i]].mean()))
    se[i]=np.sqrt(r)
cdf['Standard Error']=se
cdf['t-statistic']=cdf['Coefficients']/cdf['Standard Error']
cdf
```

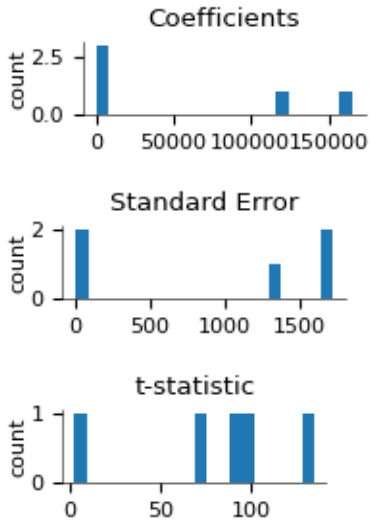
OUTPUT:

	Coefficients	Standard Error	t-statistic
Avg. Area Income	21.597602	0.160361	134.681505
Avg. Area House Age	165201.104954	1722.412068	95.912649
Avg. Area Number of Rooms	119061.463868	1696.546476	70.178722
Avg. Area Number of Bedrooms	3212.585606	1376.451759	2.333962
Area Population	15.228121	0.169882	89.639472

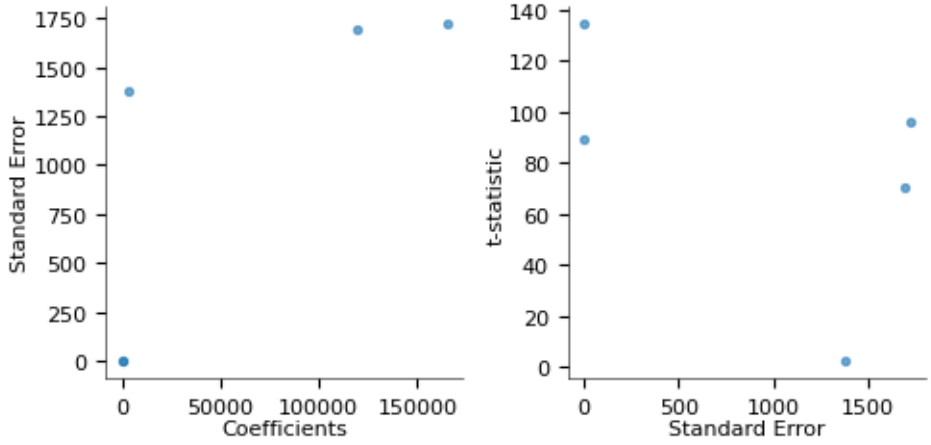
Values



Distributions



2-d distributions



CODE-4:

CODE ASSIGNMENTS 04: Implement Logistic Regression Model Using Titanic Ship Dataset

Linear regression using a pre-defined library. Comparative analysis of both implementations.

Import the required Python, Pandas, Matplotlib, Seaborn packages

```
import nbconvert
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

train = pd.read_csv('titanic_train.csv') # Training set is already available
train.head()
```

OUTPUT:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacqu	female	35.0	1	0	113803	53.1000	C123	S

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
				es Heath (Lily May Peel)								
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Check the data types of each feature(column) in the dataset.

```
t=train.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
d=train.describe()
d
```

OUTPUT:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000

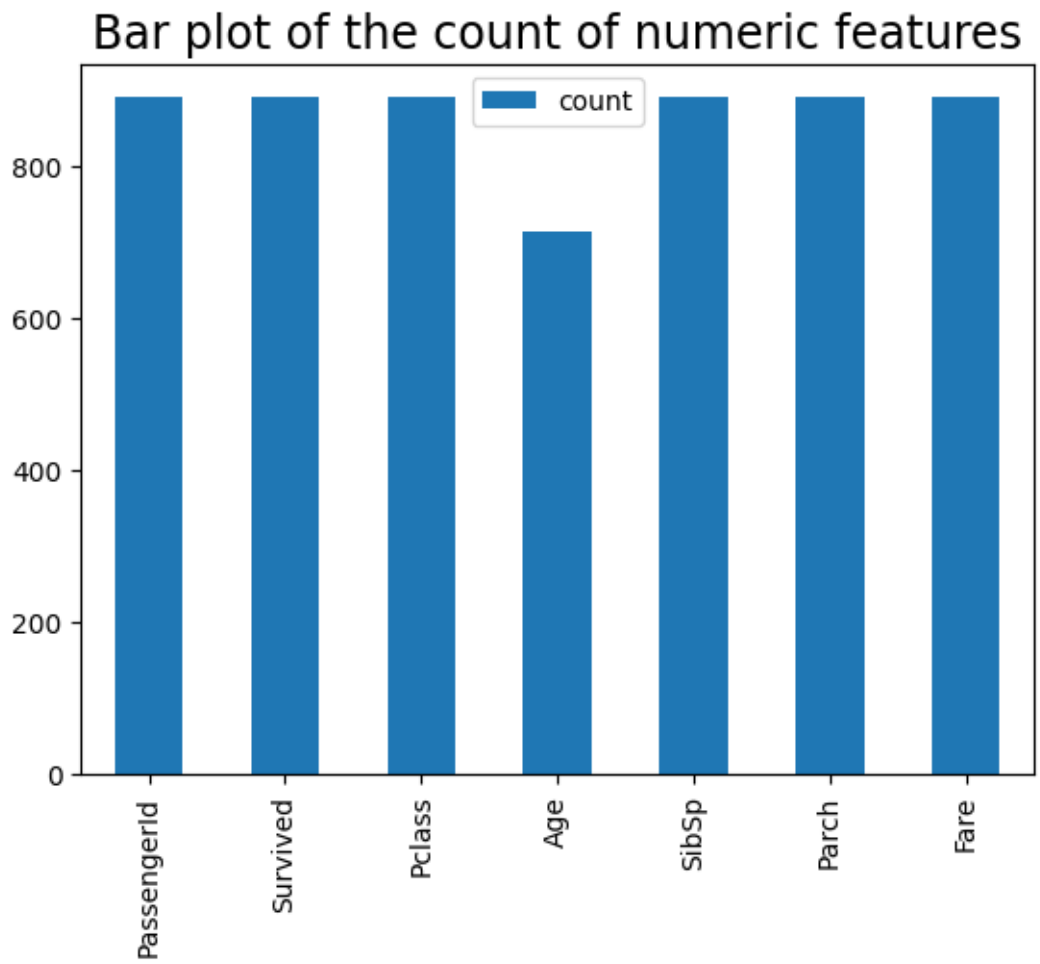
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Perform Exploratory analysis - plot numeric features, check relative size of survived/un-
survived, check if any pattern on gender. passenger class, class-wise survival rate, sibil-
ings, overall age distribution, class-wise age distribution - apply bar plot, histogram,
box plots to visualize

```
dT=d.T
dT.plot.bar(y='count')
plt.title("Bar plot of the count of numeric features",fontsize=17)
```

OUTPUT:

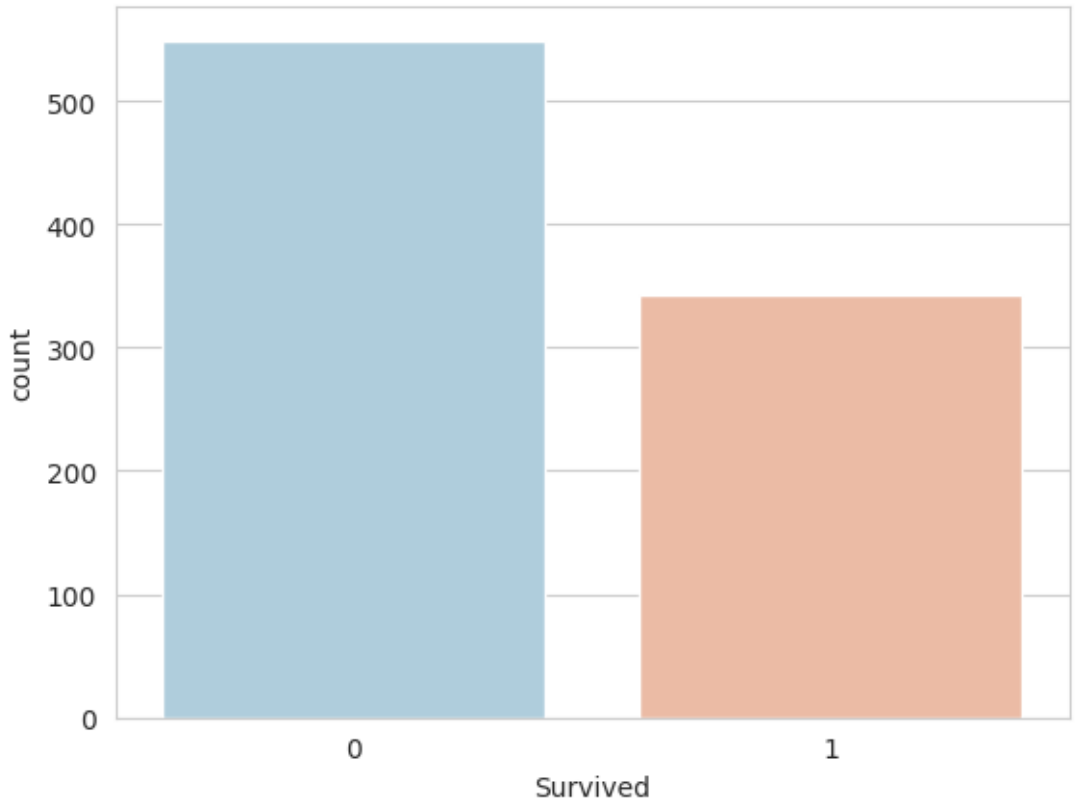
```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```

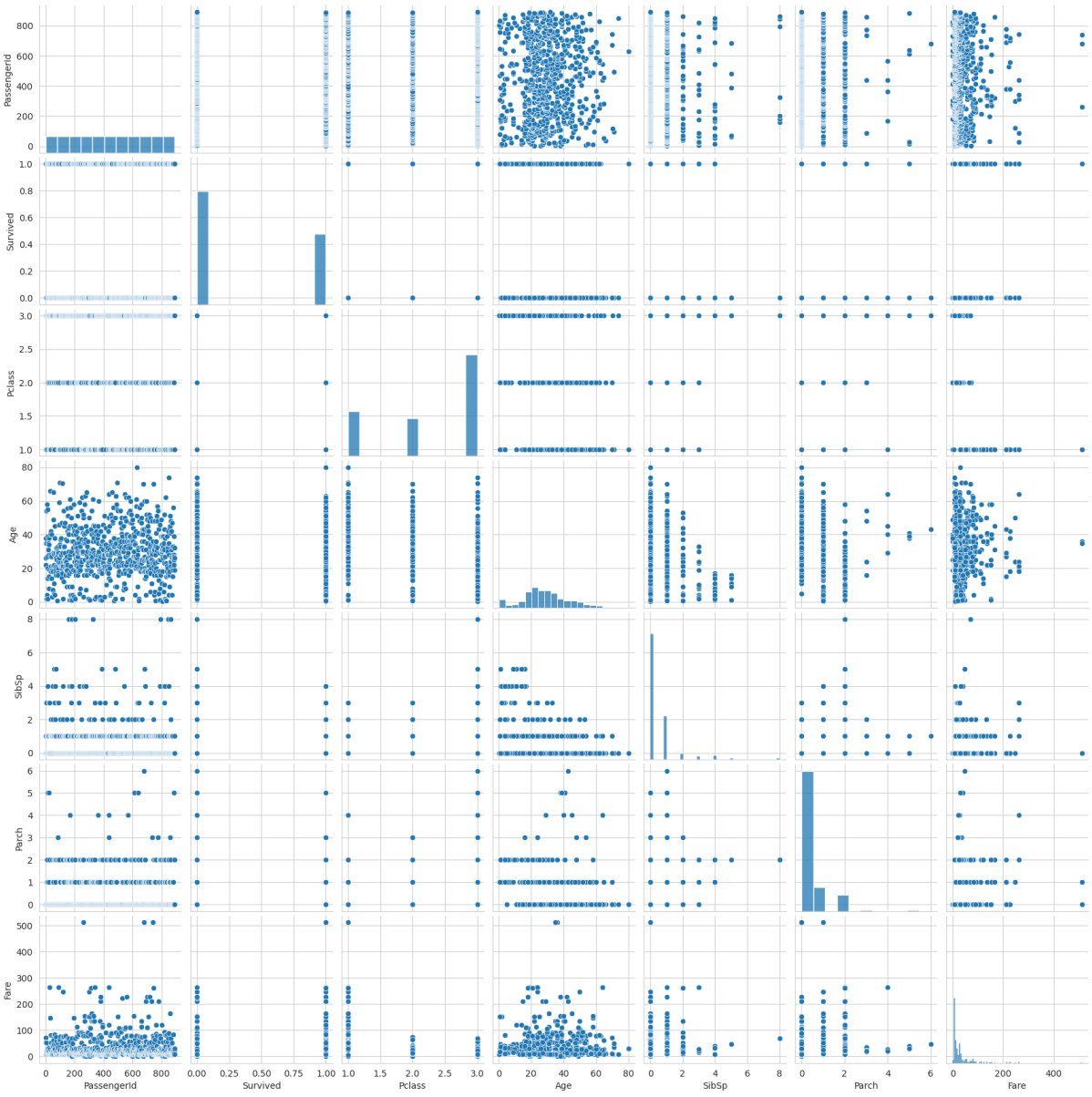



```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
sns.pairplot(train)
```

OUTPUT:

```
<seaborn.axisgrid.PairGrid at 0x7d97cb6284f0>
```

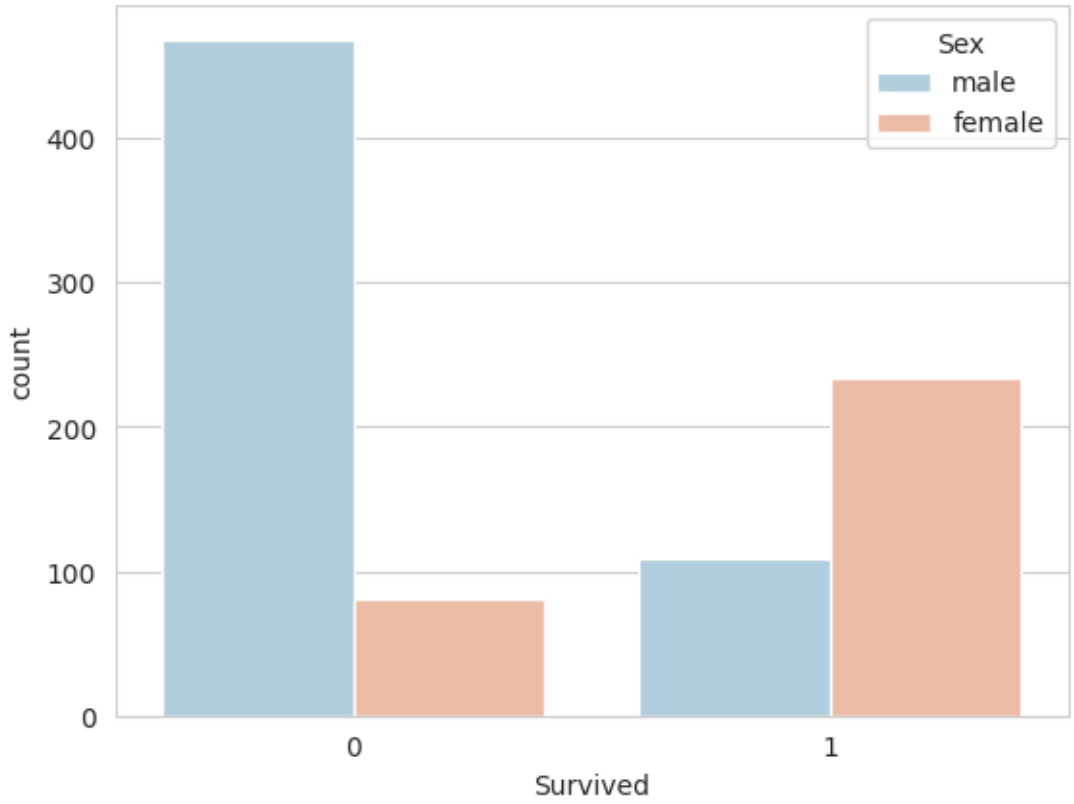





```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

OUTPUT:

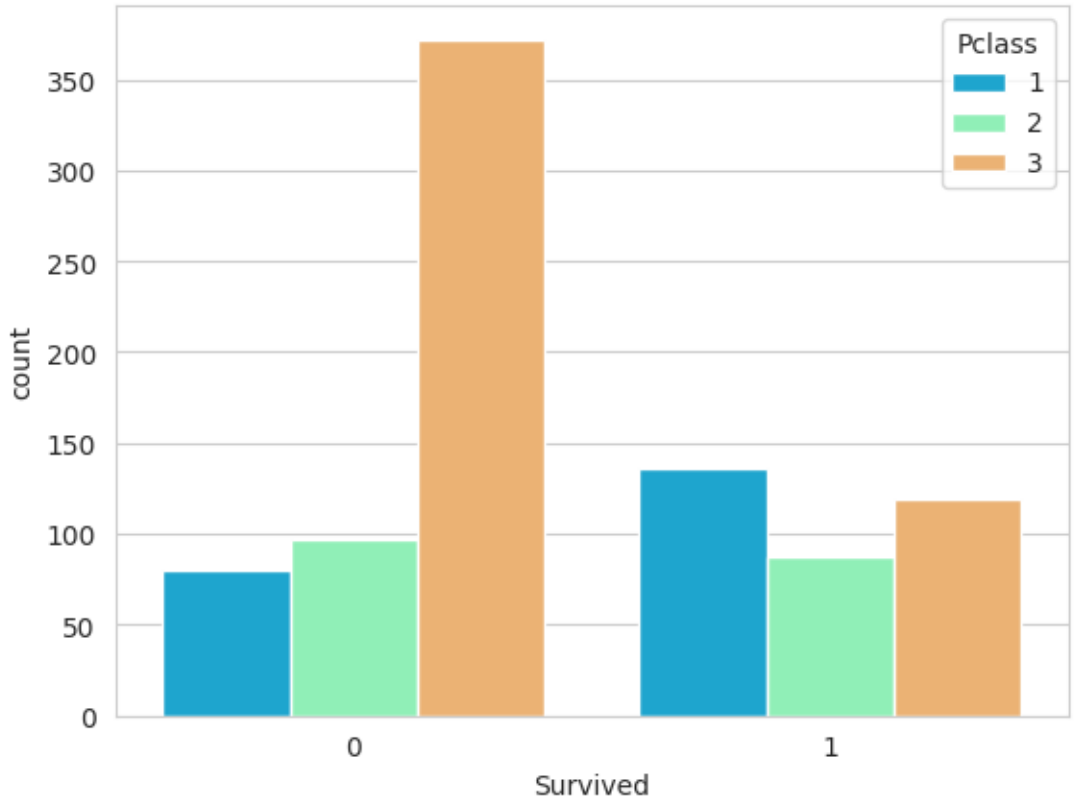
```
<Axes: xlabel='Survived', ylabel='count'>
```




```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

OUTPUT:

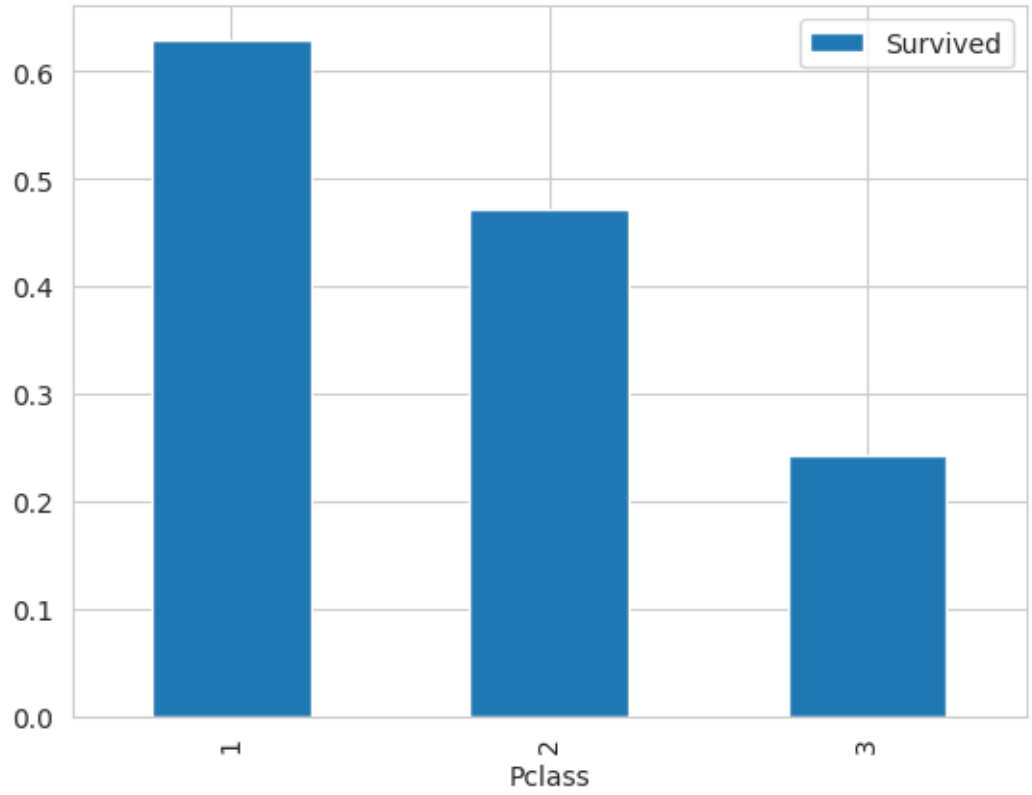
```
<Axes: xlabel='Survived', ylabel='count'>
```




```
f_class_survived=train.groupby('Pclass')['Survived'].mean()
f_class_survived = pd.DataFrame(f_class_survived)
f_class_survived
f_class_survived.plot.bar(y='Survived')
%%sns.countplot(x='Survived',data=f_class_survived,palette='rainbow')
plt.title("Fraction of passengers survived by class",fontsize=17)
```

OUTPUT:

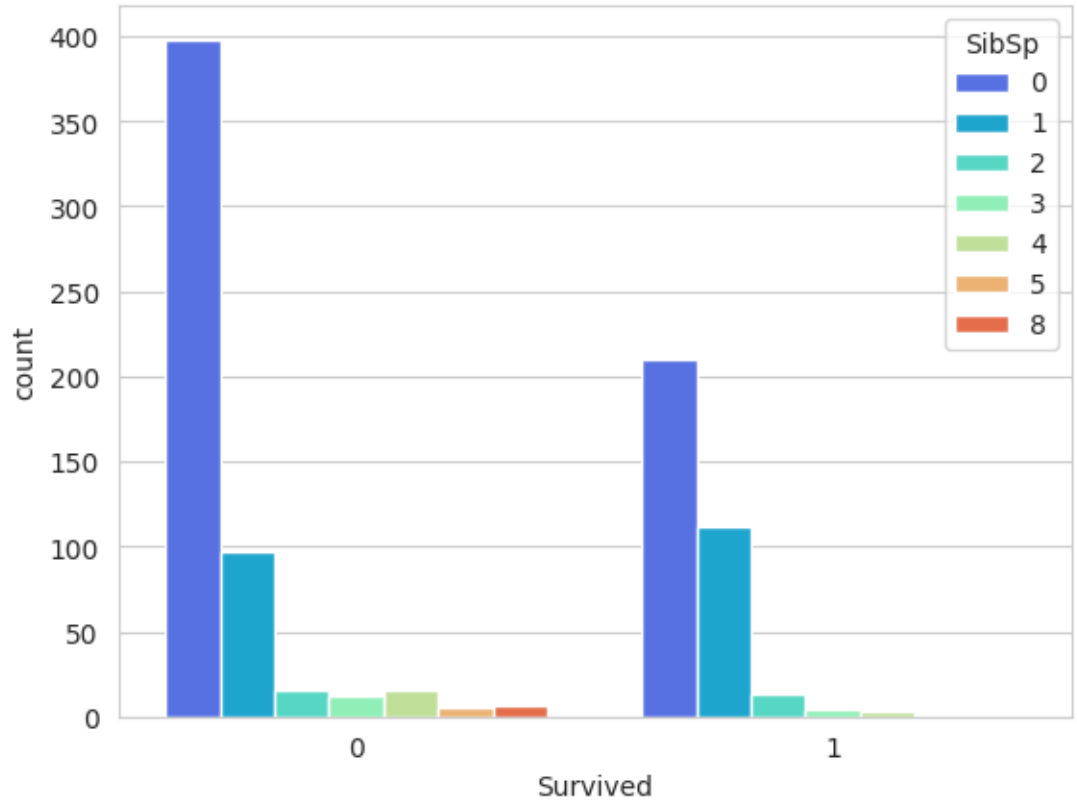
```
UsageError: Line magic function
`%%sns.countplot(x='Survived',data=f_class_survived,palette='rainbow')` not
found.
```




```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='SibSp',data=train,palette='rainbow')
```

OUTPUT:

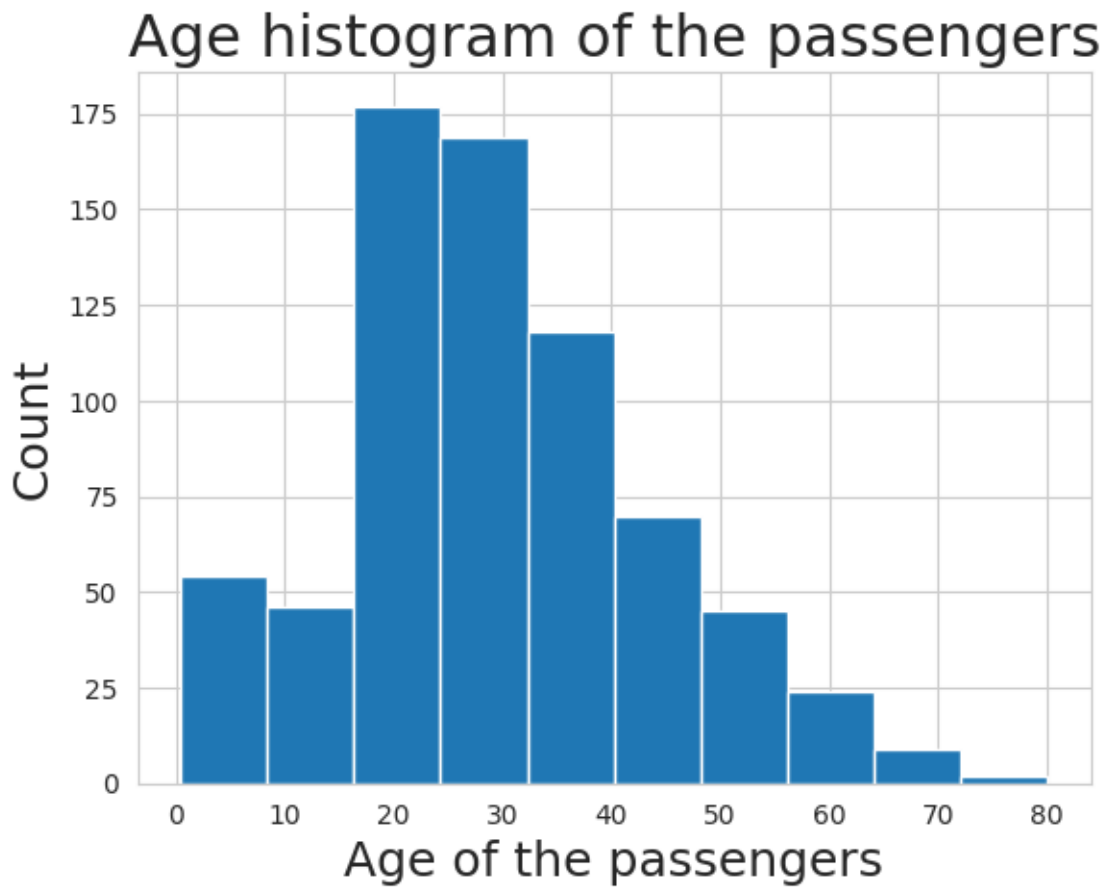
```
<Axes: xlabel='Survived', ylabel='count'>
```



```
plt.xlabel("Age of the passengers",fontsize=18)
plt.ylabel("Count",fontsize=18)
plt.title("Age histogram of the passengers",fontsize=22)
#train['Age'].hist(bins=30,color='darkred',alpha=0.7,figsize=(10,6))
train['Age'].hist()
```

OUTPUT:

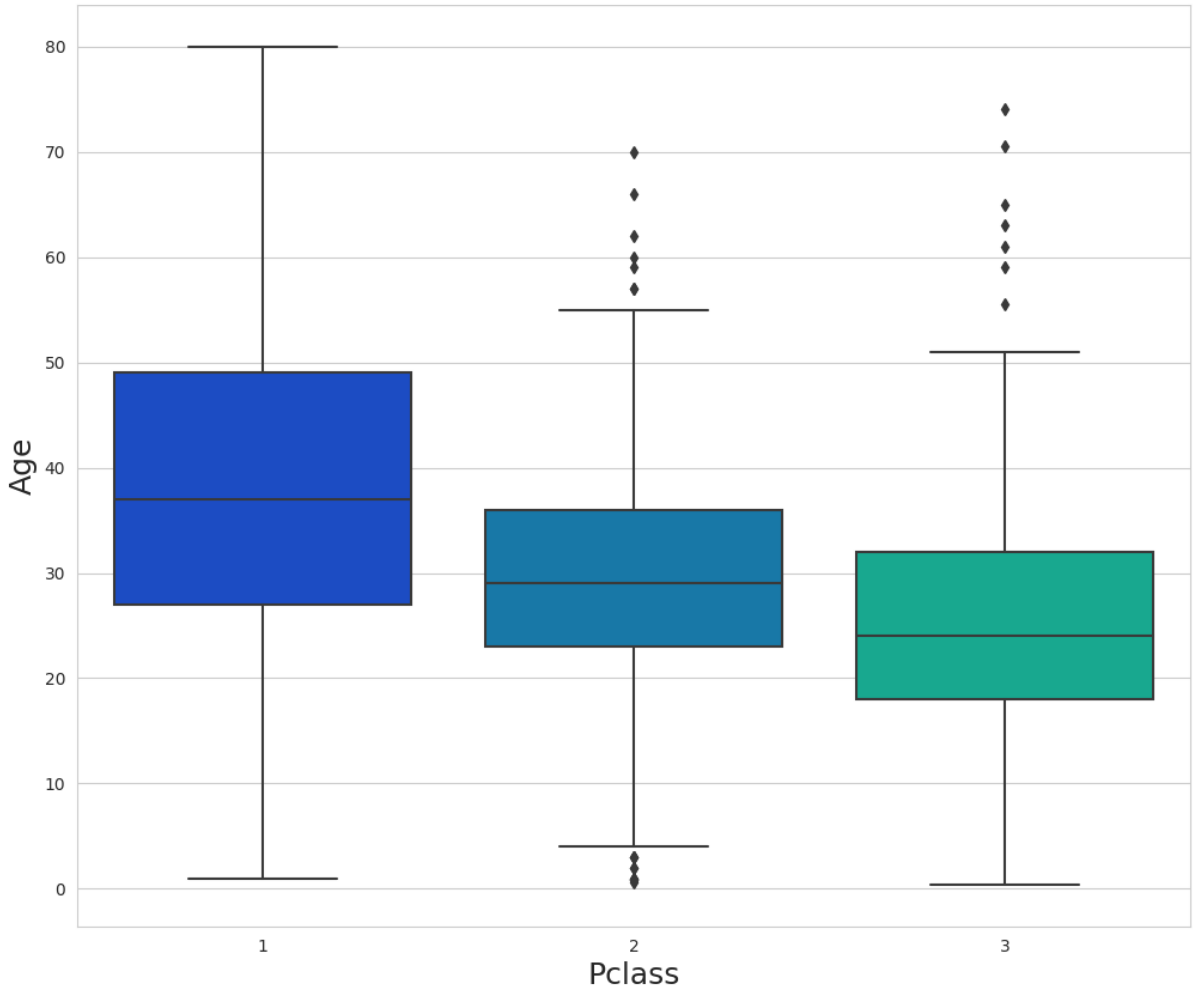
```
<Axes: title={'center': 'Age histogram of the passengers'}, xlabel='Age of
the passengers', ylabel='Count'>
```




```
plt.figure(figsize=(12, 10))
plt.xlabel("Passenger Class",fontsize=18)
plt.ylabel("Age",fontsize=18)
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

OUTPUT:

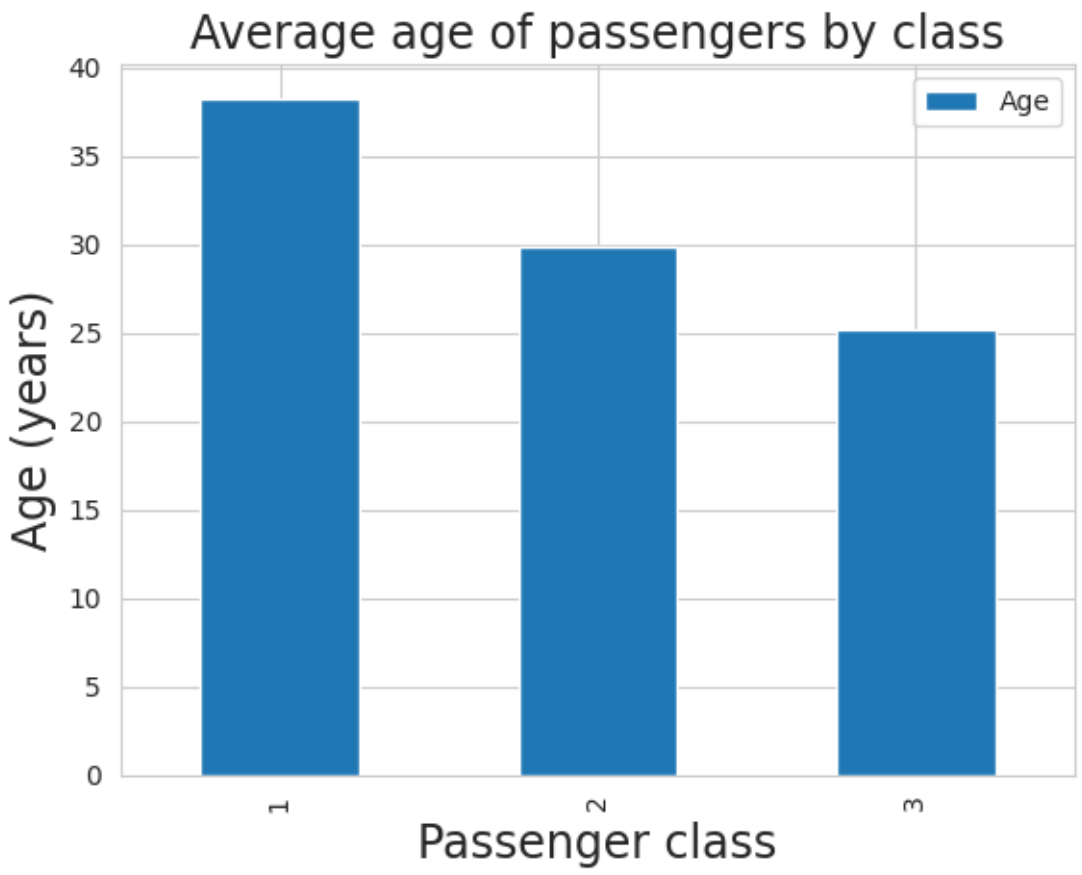
```
<Axes: xlabel='Pclass', ylabel='Age'>
```




```
f_class_Age=train.groupby('Pclass')['Age'].mean()
f_class_Age = pd.DataFrame(f_class_Age)
f_class_Age.plot.bar(y='Age')
plt.title("Average age of passengers by class",fontsize=17)
plt.ylabel("Age (years)", fontsize=17)
plt.xlabel("Passenger class", fontsize=17)
```

OUTPUT:

```
Text(0.5, 0, 'Passenger class')
```



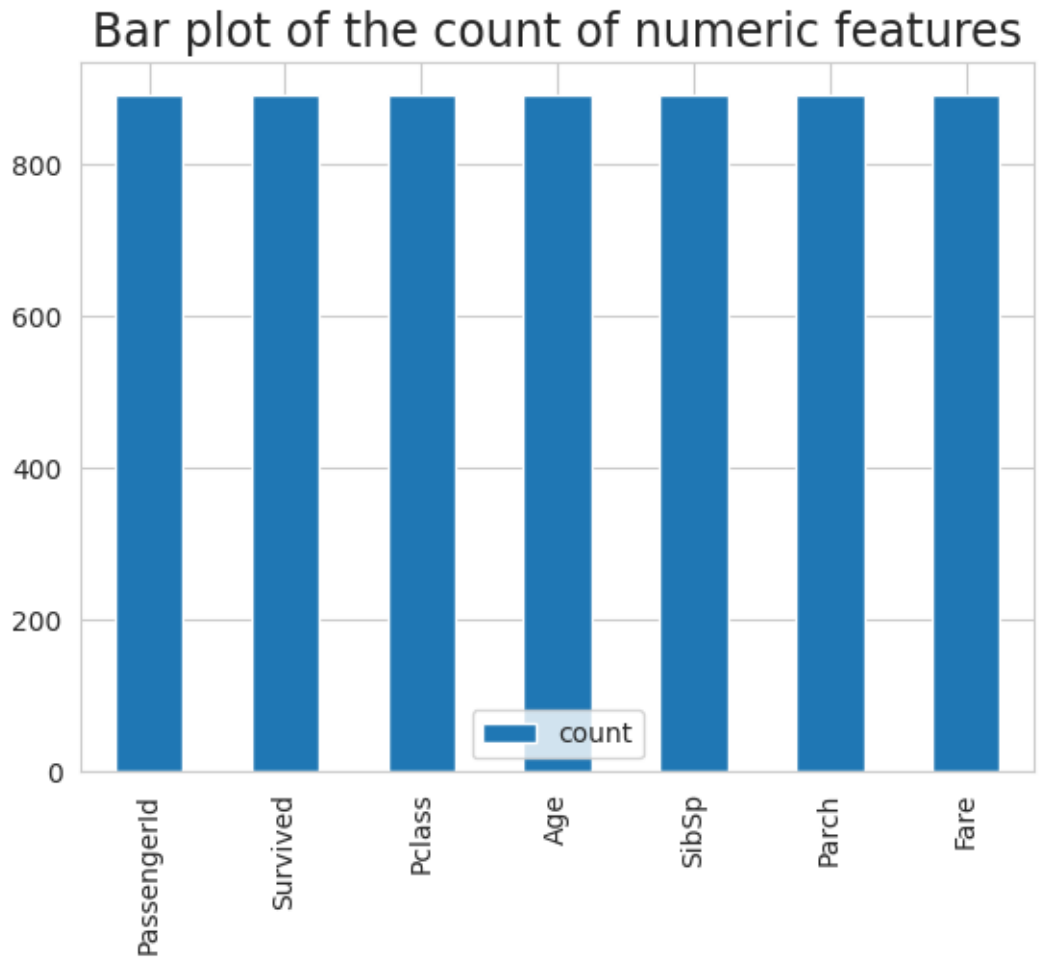

```
a=list(f_class_Age['Age'])

def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return a[0]

        elif Pclass == 2:
            return a[1]

        else:
            return a[2]

    else:
        return Age
```

```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
d=train.describe()
dT=d.T
dT.plot.bar(y='count')
```

```
plt.title("Bar plot of the count of numeric features",fontsize=17)
```

OUTPUT:

```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```




```
train.drop('Cabin',axis=1,inplace=True)
train.dropna(inplace=True)
train.head()
```

OUTPUT:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley	female	38.0	1	0	PC 17599	71.2833	C

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
				(Florence Briggs Th...							
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

```
train.drop(['PassengerId','Name','Ticket'],axis=1,inplace=True)
train.head()
```

OUTPUT:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Sex','Embarked'],axis=1,inplace=True)
train = pd.concat([train,sex,embark],axis=1)
train.head()
```

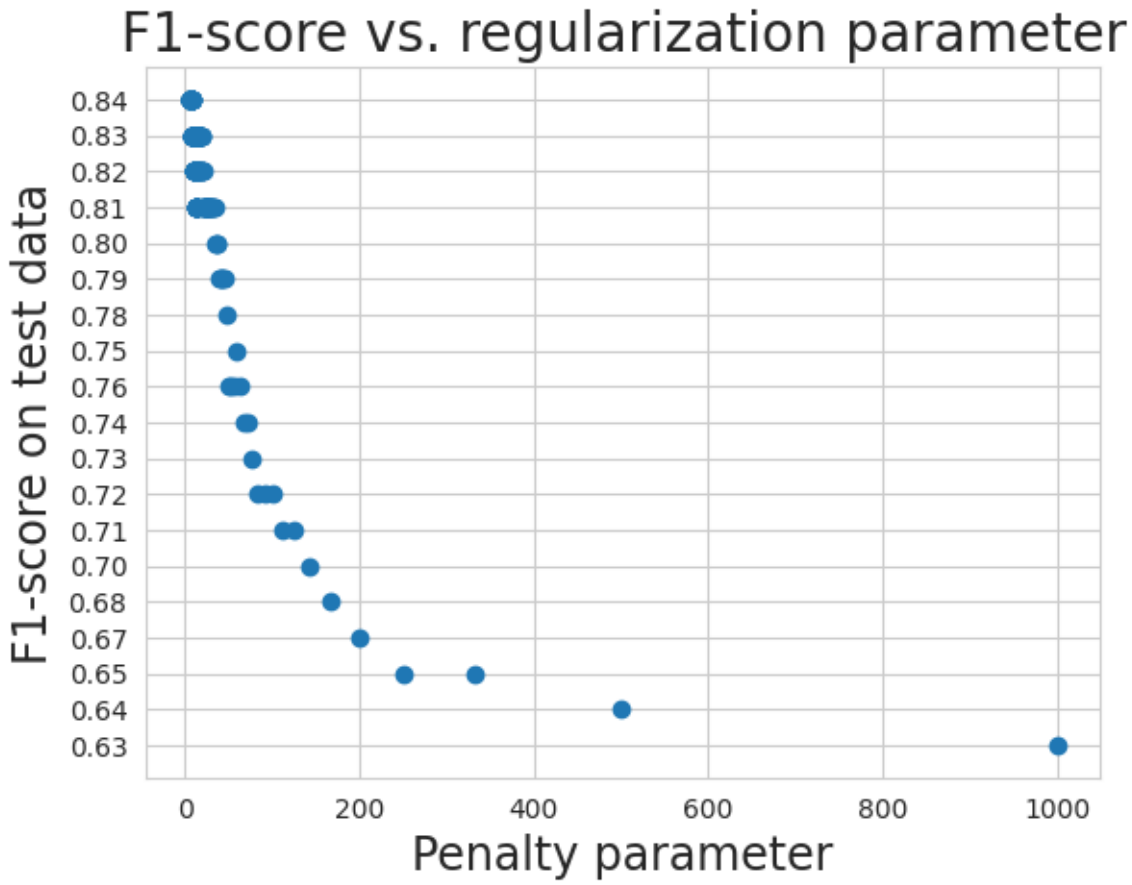
OUTPUT:

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'],
test_size=0.30,
                                                    random_state=111)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
nsimu=201
penalty=[0]*nsimu
logmodel=[0]*nsimu
predictions =[0]*nsimu
class_report = [0]*nsimu
f1=[0]*nsimu
for i in range(1,nsimu):
        logmodel[i] =(LogisticRegression(C=i/1000,tol=1e-4,
max_iter=int(1e6),
                                         n_jobs=4))
        logmodel[i].fit(X_train,y_train)
        predictions[i] = logmodel[i].predict(X_test)
        class_report[i] = classification_report(y_test,predictions[i])
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        penalty[i]=1000/i

plt.scatter(penalty[1:len(penalty)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. regularization parameter",fontsize=20)
plt.xlabel("Penalty parameter",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```

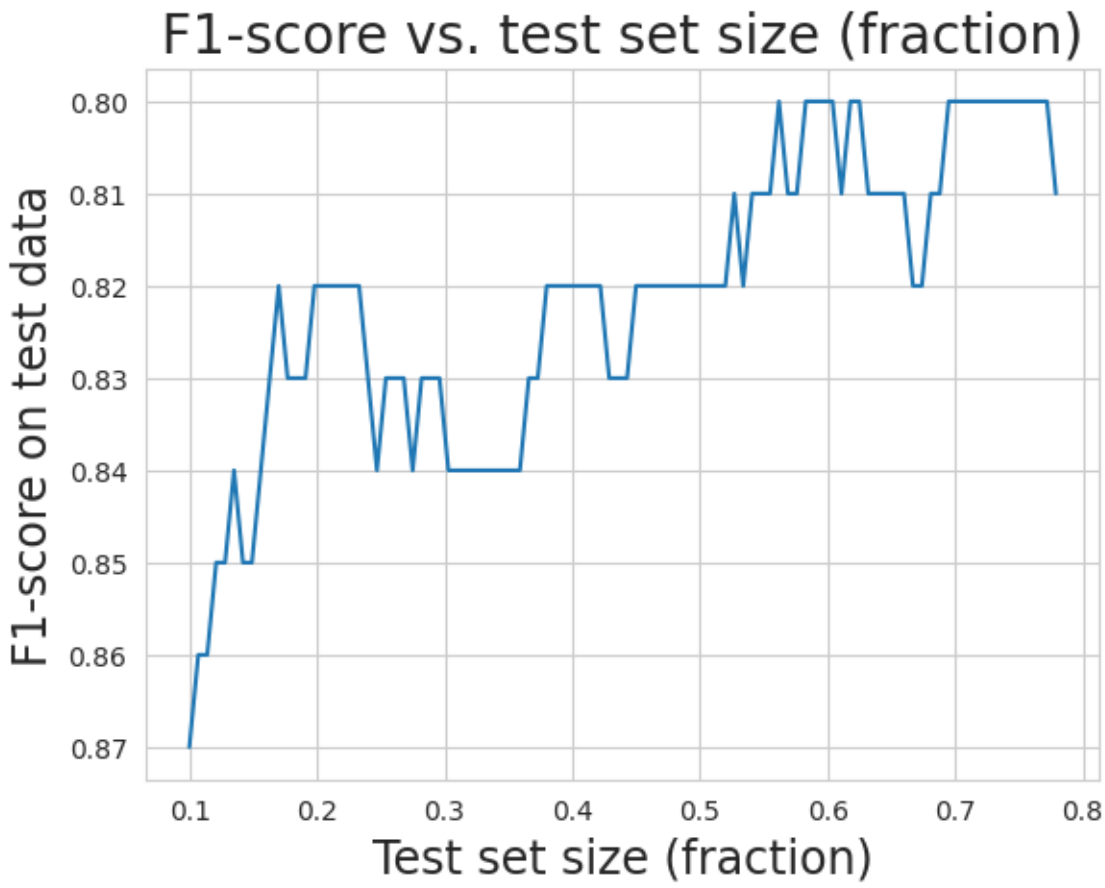
OUTPUT:




```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu
test_fraction =[0]*nsimu
for i in range(1,nsimu):
        X_train, X_test, y_train, y_test =
train_test_split(train.drop('Survived',axis=1),
                                                train['Survived'],
test_size=0.1+(i-1)*0.007,
                                                random_state=111)
        logmodel =(LogisticRegression(C=1,tol=1e-4,
max_iter=1000,n_jobs=4))
        logmodel.fit(X_train,y_train)
        predictions = logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        test_fraction[i]=0.1+(i-1)*0.007

plt.plot(test_fraction[1:len(test_fraction)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. test set size (fraction)",fontsize=20)
```

```
plt.xlabel("Test set size (fraction)",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```

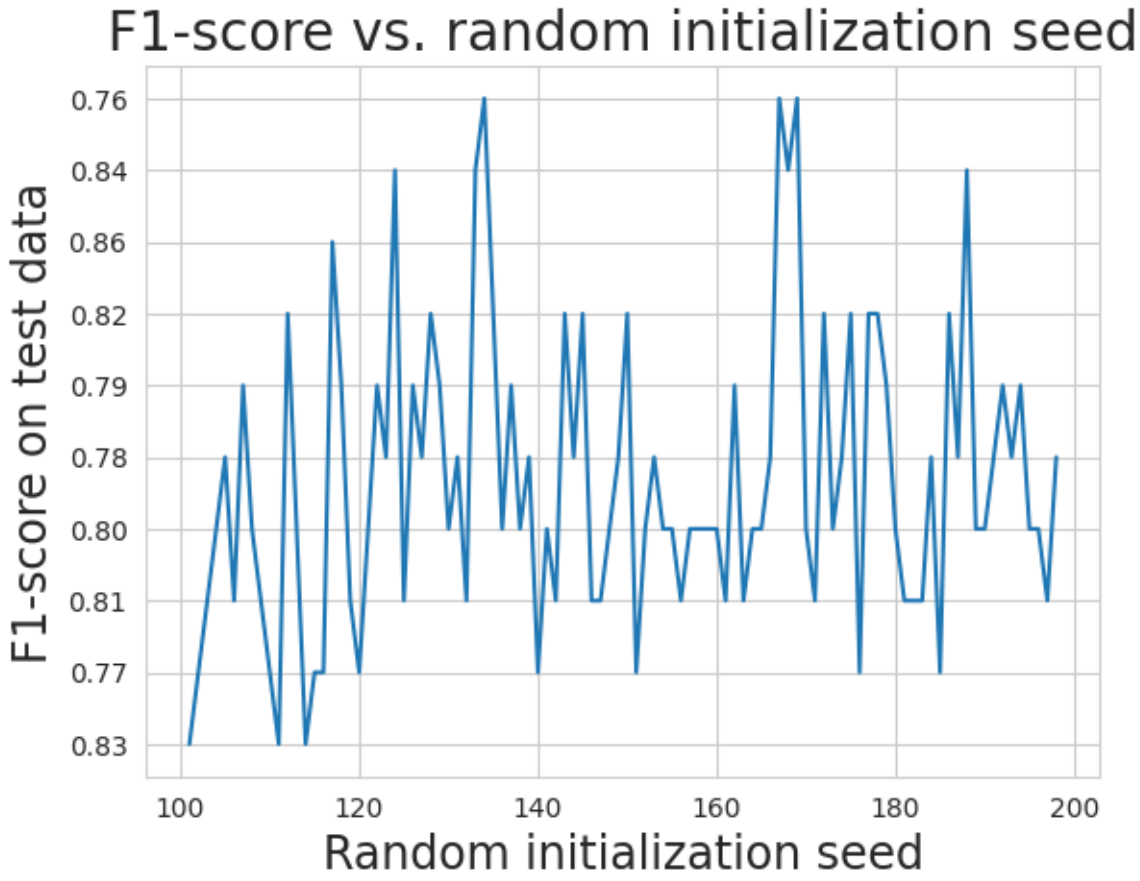
OUTPUT:




```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu
random_init =[0]*nsimu
for i in range(1,nsimu):
        X_train, X_test, y_train, y_test =
train_test_split(train.drop('Survived',axis=1),
                                                train['Survived'],
test_size=0.3,
                                                random_state=i+100)
        logmodel =(LogisticRegression(C=1,tol=1e-5,
max_iter=1000,n_jobs=4))
        logmodel.fit(X_train,y_train)
        predictions = logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        random_init[i]=i+100

plt.plot(random_init[1:len(random_init)-2],f1[1:len(f1)-2])
```

```
plt.title("F1-score vs. random initialization seed",fontsize=20)
plt.xlabel("Random initialization seed",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```

OUTPUT:



PART-B

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

df = pd.read_table("Classified_Data.txt",sep=',', index_col=0)
df.head()
```

OUTPUT:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

```
df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   WTT           1000 non-null   float64
 1   PTI           1000 non-null   float64
 2   EQW           1000 non-null   float64
 3   SBI           1000 non-null   float64
 4   LQE           1000 non-null   float64
 5   QWG           1000 non-null   float64
 6   FDJ           1000 non-null   float64
 7   PJF           1000 non-null   float64
 8   HQE           1000 non-null   float64
 9   NXJ           1000 non-null   float64
 10  TARGET CLASS  1000 non-null   int64  
dtypes: float64(10), int64(1)
memory usage: 93.8 KB
```

```
df.describe()
```

OUTPUT:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.949682	1.114303	0.834127	0.682099	1.032336	0.943534	0.963422	1.071960	1.158251	1.362725	0.500000
std	0.289635	0.257085	0.291554	0.229645	0.243413	0.256121	0.255118	0.288982	0.293738	0.204225	0.500025
min	0.174412	0.441398	0.170924	0.045027	0.315307	0.262389	0.295228	0.299476	0.365157	0.639693	0.000000
25%	0.742358	0.942071	0.615451	0.515010	0.870855	0.761064	0.784407	0.866306	0.934340	1.222623	0.000000
50%	0.940475	1.118486	0.813264	0.676835	1.035824	0.941502	0.945333	1.065500	1.165556	1.375368	0.500000
75%	1.163295	1.307904	1.028340	0.834317	1.198270	1.123060	1.134852	1.283156	1.383173	1.504832	1.000000
max	1.721779	1.833757	1.722725	1.634884	1.650050	1.666902	1.713342	1.785420	1.885690	1.893950	1.000000

```
l=list(df.columns)
l[0:len(l)-2]
```

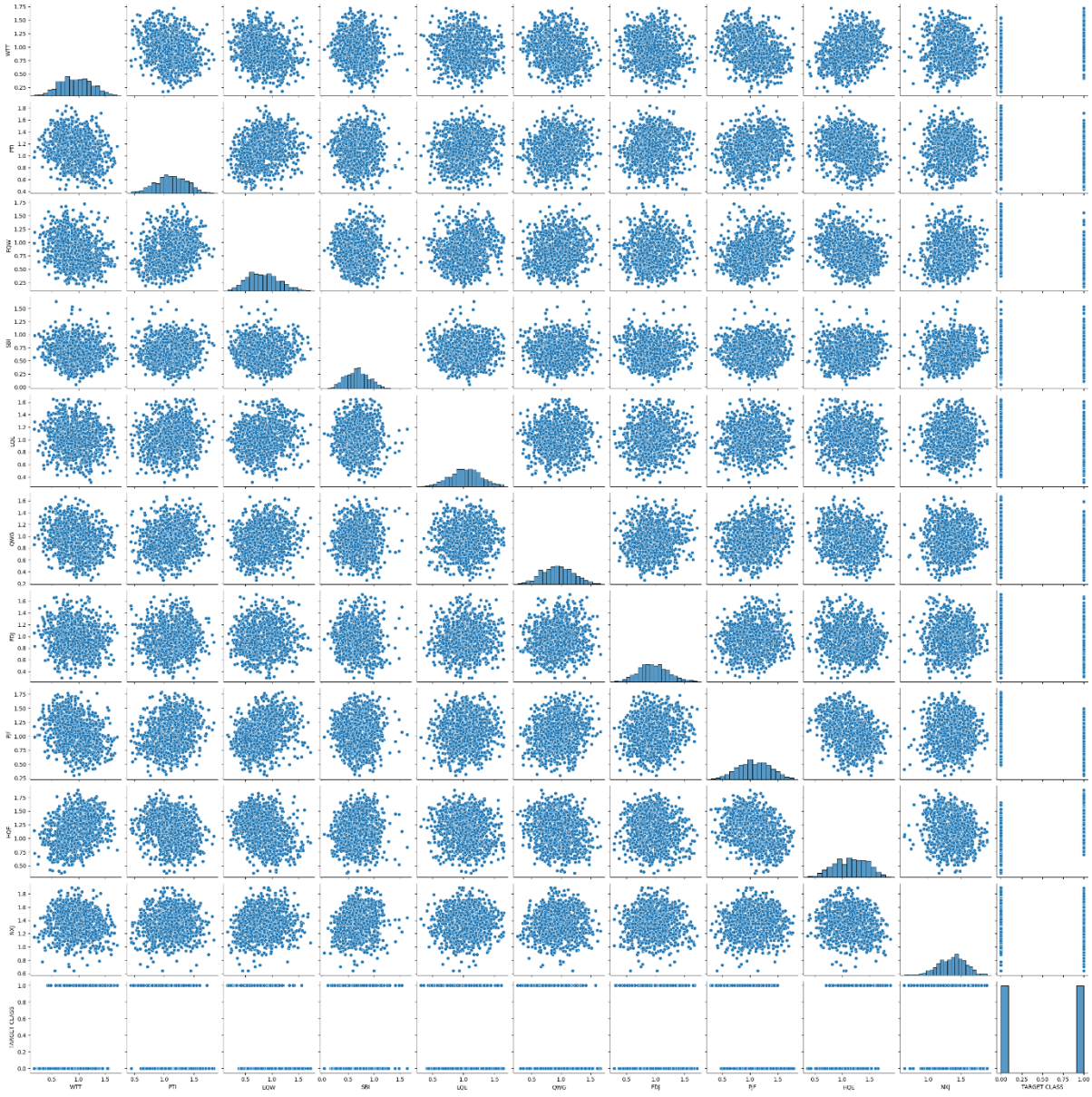
OUTPUT:

```
['WTT', 'PTI', 'EQW', 'SBI', 'LQE', 'QWG', 'FDJ', 'PJF', 'HQE']
```

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

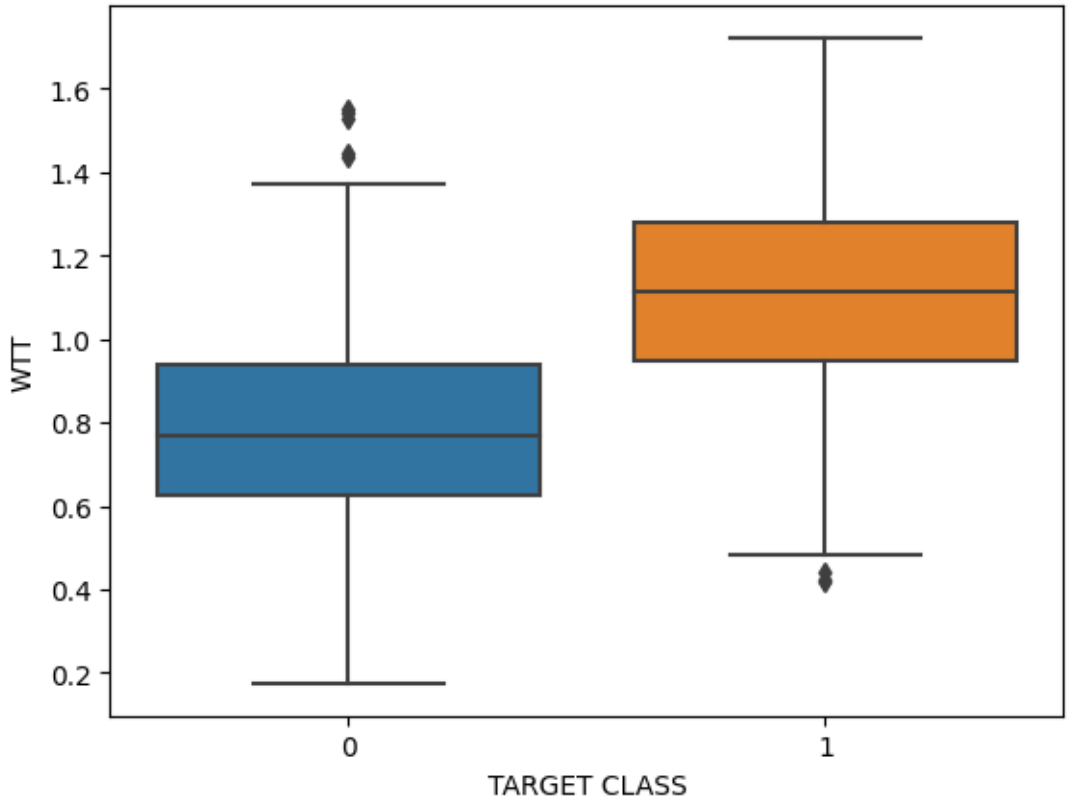
OUTPUT:

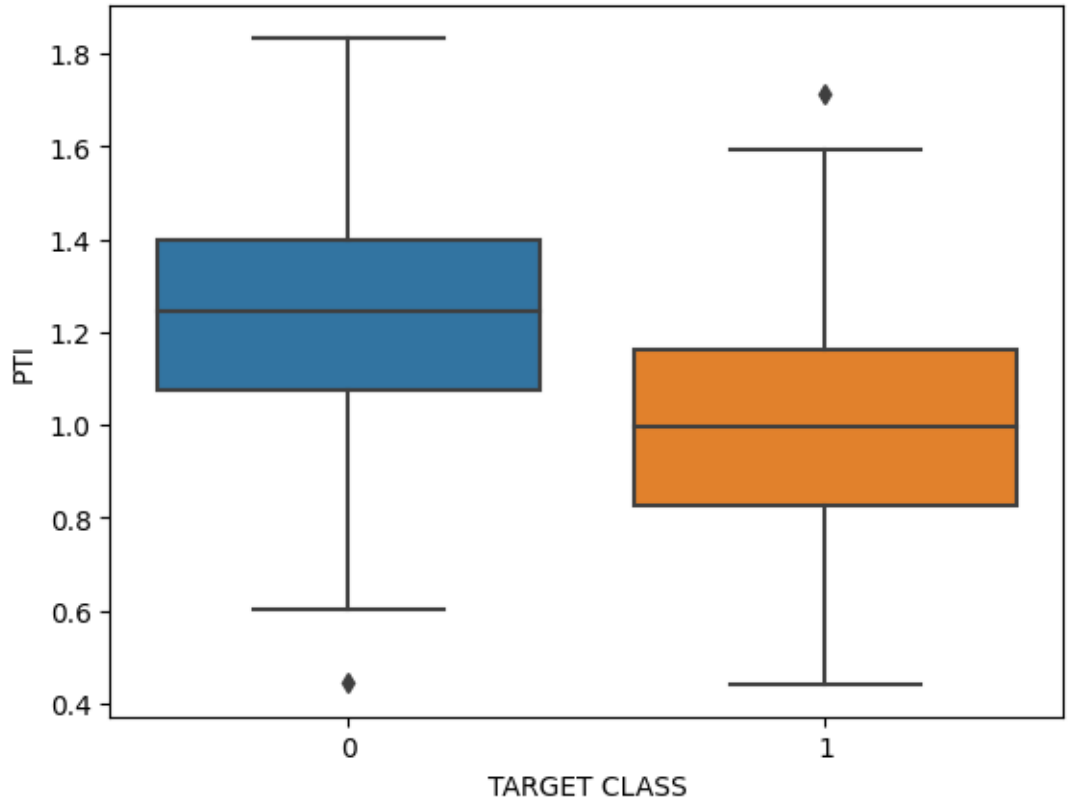
```
<seaborn.axisgrid.PairGrid at 0x797a43336cb0>
```

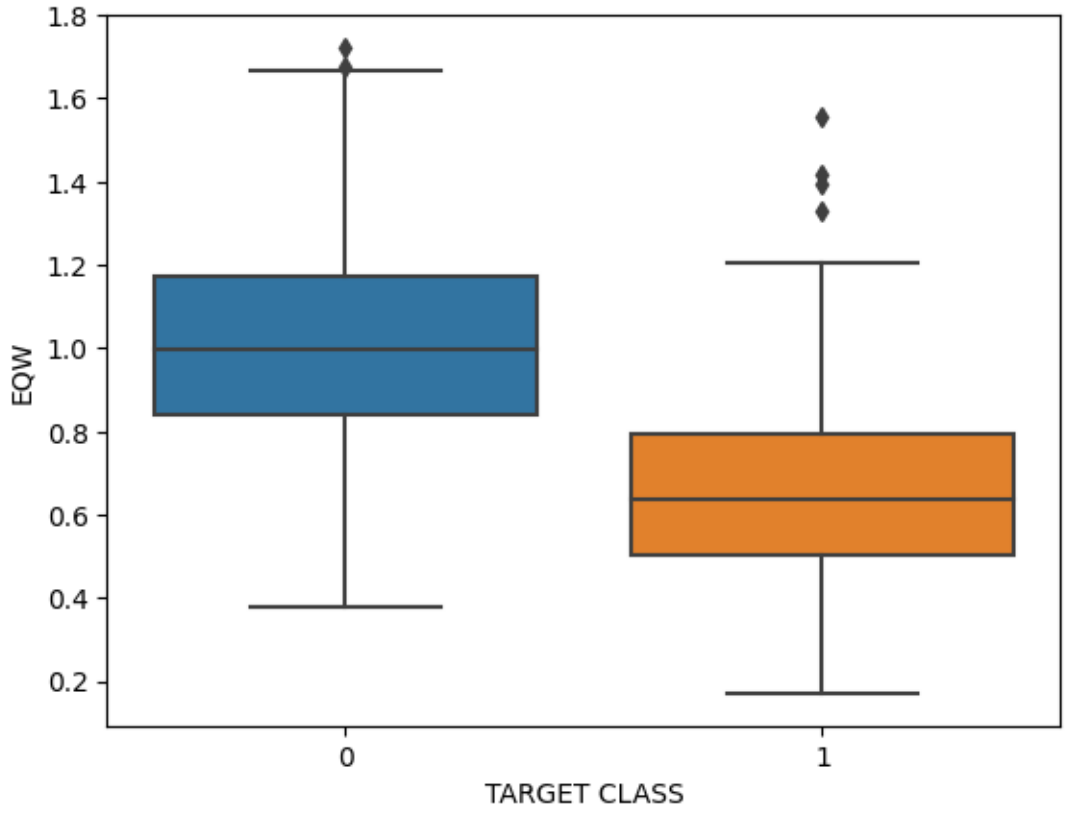


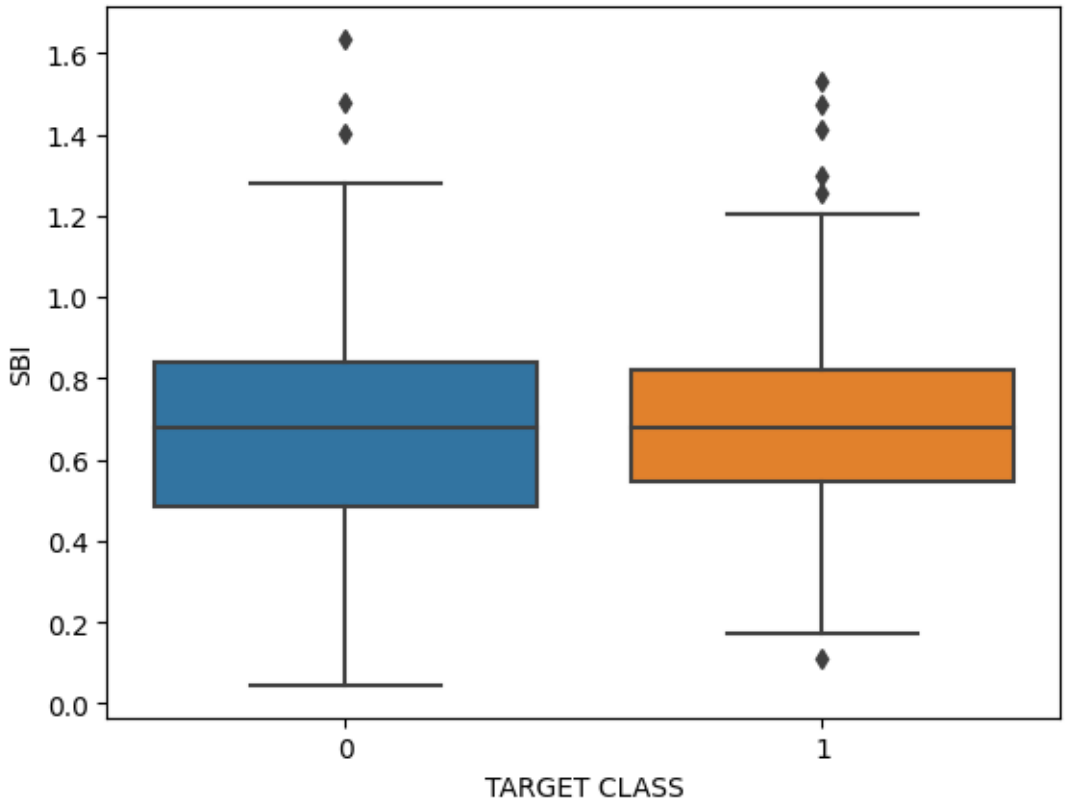
```
for i in range(len(l)-1):
    sns.boxplot(x='TARGET CLASS',y=l[i], data=df)
    plt.figure()
```

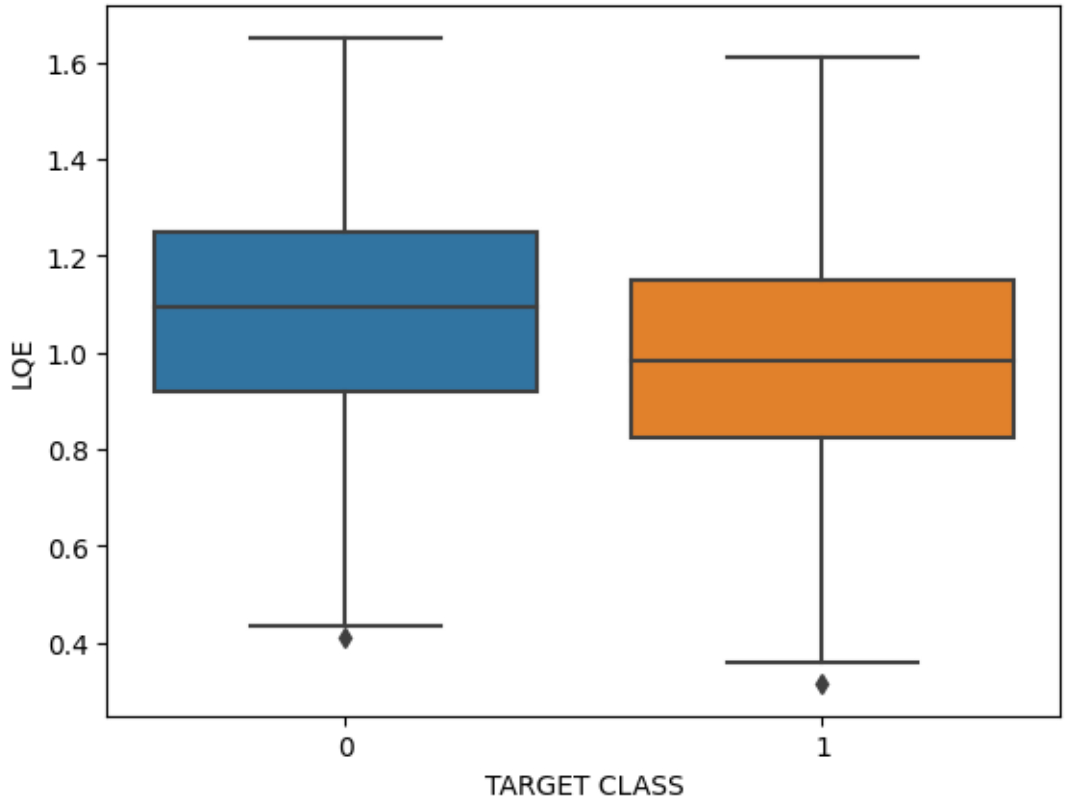
OUTPUT:

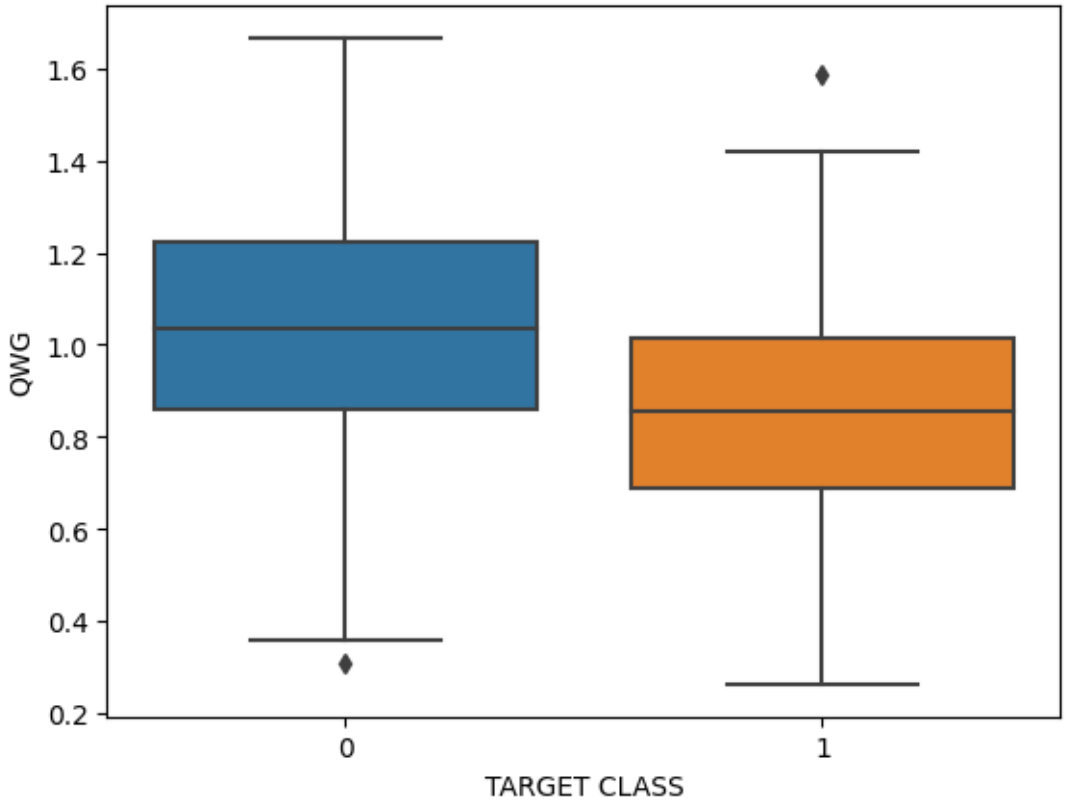

WTT
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
1
TARGET CLASS

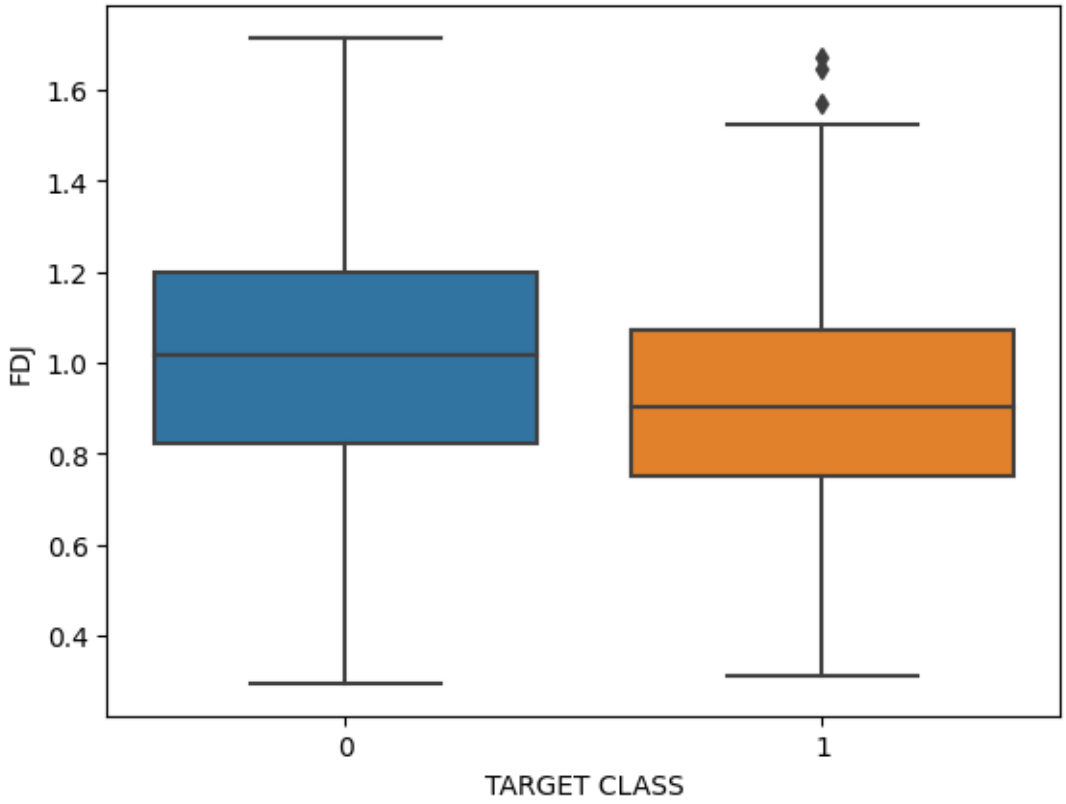

PTI
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
1
TARGET CLASS

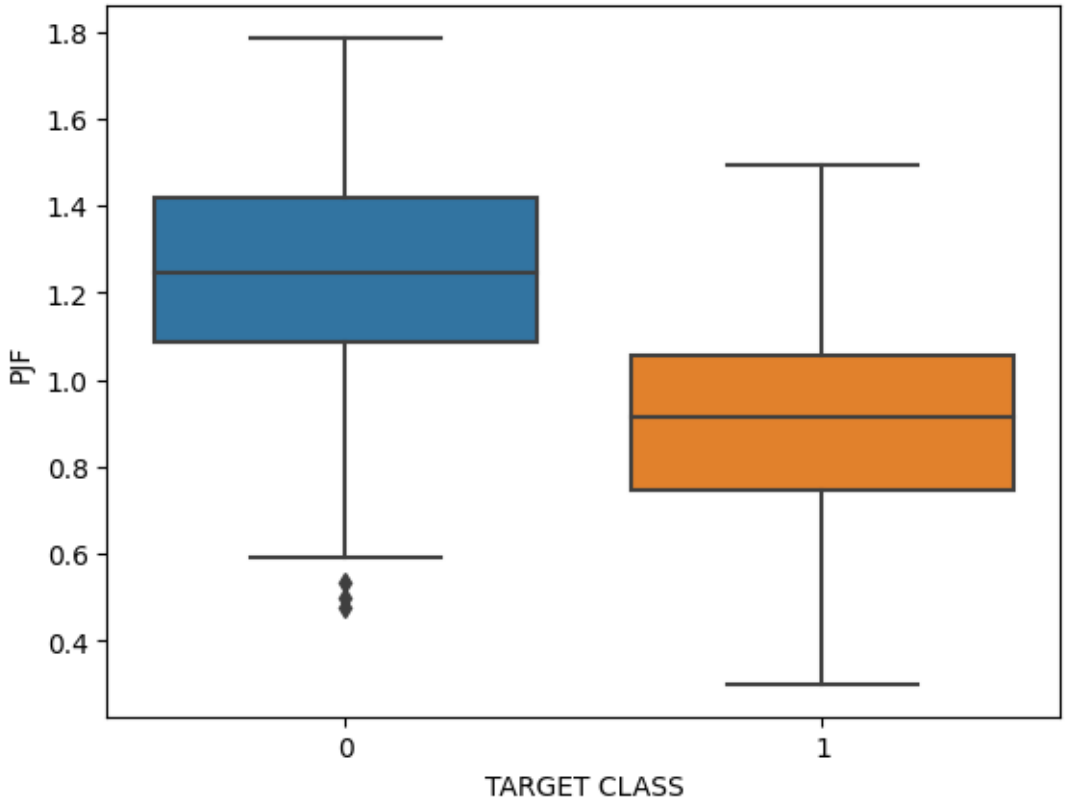

EQW
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
1
TARGET CLASS

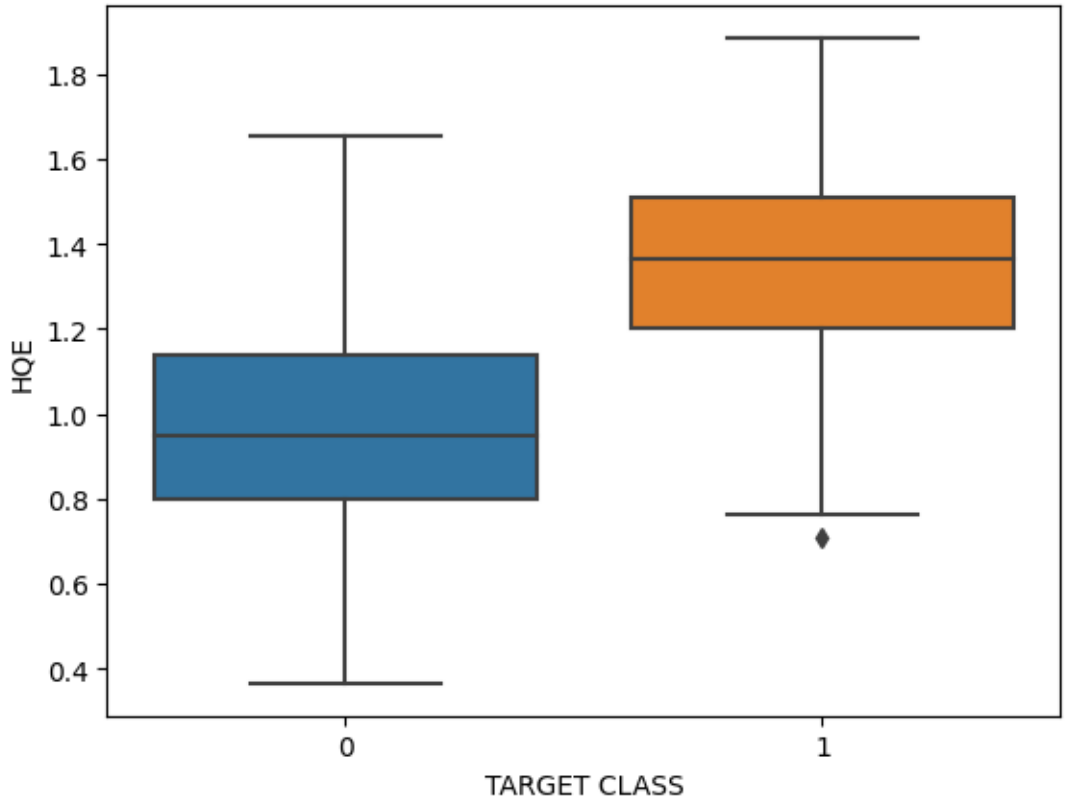

SBI
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
TARGET CLASS

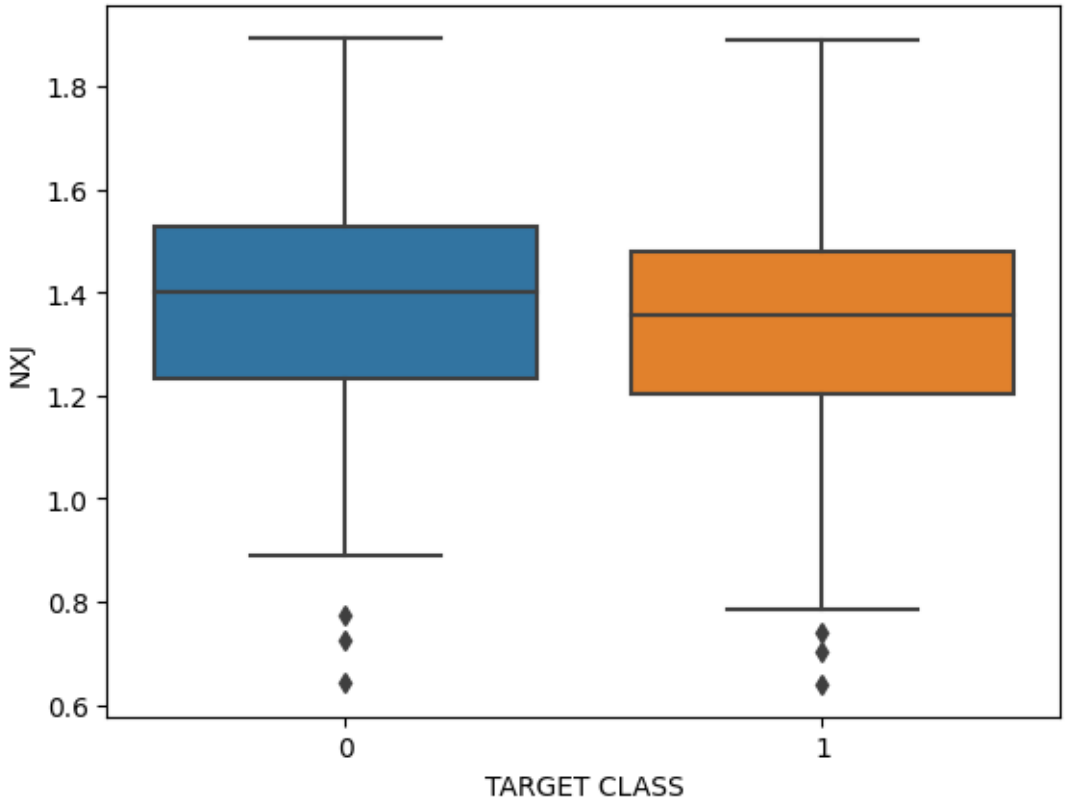

LQE
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
1
TARGET CLASS


QWG
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
1
TARGET CLASS


FDJ
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
1
TARGET CLASS


PJF
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
1
TARGET CLASS






```
<Figure size 640x480 with 0 Axes>
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

OUTPUT:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

```
from sklearn.model_selection import train_test_split
X = df_feat
y = df['TARGET CLASS']
X_train, X_test, y_train, y_test =
train_test_split(scaled_features,df['TARGET CLASS'],
                                                    test_size=0.50,
random_state=101)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
```

KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=1)
```

```
pred = knn.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
conf_mat=confusion_matrix(y_test,pred)
print(conf_mat)
```

OUTPUT:

```
[[233  17]
 [ 24 226]]
```

```
print(classification_report(y_test,pred))
```

OUTPUT:

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       250
           1       0.93      0.90      0.92       250

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500
```

```
print("Misclassification error rate:",round(np.mean(pred!=y_test),3))
```

OUTPUT:

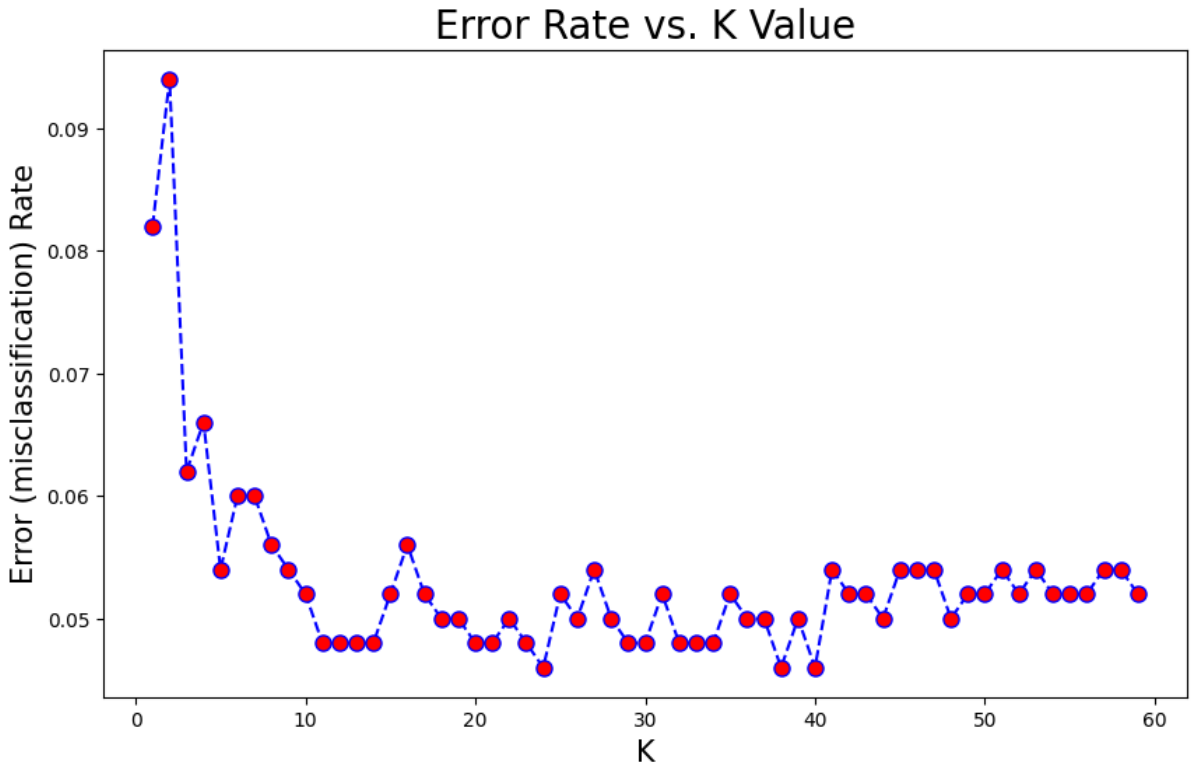
```
Misclassification error rate: 0.082
```

```
error_rate = []

# Will take some time
for i in range(1,60):

    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(1,60),error_rate,color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=8)
plt.title('Error Rate vs. K Value', fontsize=20)
plt.xlabel('K',fontsize=15)
plt.ylabel('Error (misclassification) Rate',fontsize=15)
```

OUTPUT:

```
Text(0, 0.5, 'Error (misclassification) Rate')
```



CODE-6:

CODE ASSIGNMENTS 06: *Implementation of a Project by taking a data set for any one of the Linear/Logistic/SVM/KNN Models*

Project on KNN-To Predict weather a person will have Diabetes or not

CODE :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("/content/diabetes.csv")
data
```

OUTPUT:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6	0.627	50	1	
1	1	85	66	29	0	26.6	0.351	31	0	
2	8	183	64	0	0	23.3	0.672	32	1	
3	1	89	66	23	94	28.1	0.167	21	0	
4	0	137	40	35	168	43.1	2.288	33	1	
...	...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0	
764	2	122	70	27	0	36.8	0.340	27	0	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
765	5	121	72	23	112	26.2	0.245	30	0	
766	1	126	60	0	0	30.1	0.349	47	1	
767	1	93	70	31	0	30.4	0.315	23	0	

768 rows × 9 columns

```
x = data.drop(['Outcome'], axis = 1)
x.head()
```

OUTPUT:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

```
y = data['Outcome']
y
```

OUTPUT:

```
0 1 1 0 2 1 3 0 4 1 .. 763 0 764 0 765 0 766 1 767 0 Name: Outcome,
Length: 768, dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x = scaler.fit_transform(x)
x
```

OUTPUT:

```
array([[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]])
```

```
y
```

OUTPUT :

```
0 1 1 0 2 1 3 0 4 1 .. 763 0 764 0 765 0 766 1 767 0 Name: Outcome,
Length: 768, dtype: int64
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3,
random_state=1)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(xtrain, ytrain)
```

OUTPUT:

KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=1)
```

```
ypred = knn.predict(xtest)
ypred
```

OUTPUT :

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

```
ytest
```

OUTPUT:

```
285 0 101 0 581 0 352 0 726 0 .. 241 0 599 0 650 0 11 1 214 1 Name:
Outcome, Length: 231, dtype: int64
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(ytest, ypred))
print(classification_report(ytest, ypred))
```

OUTPUT:

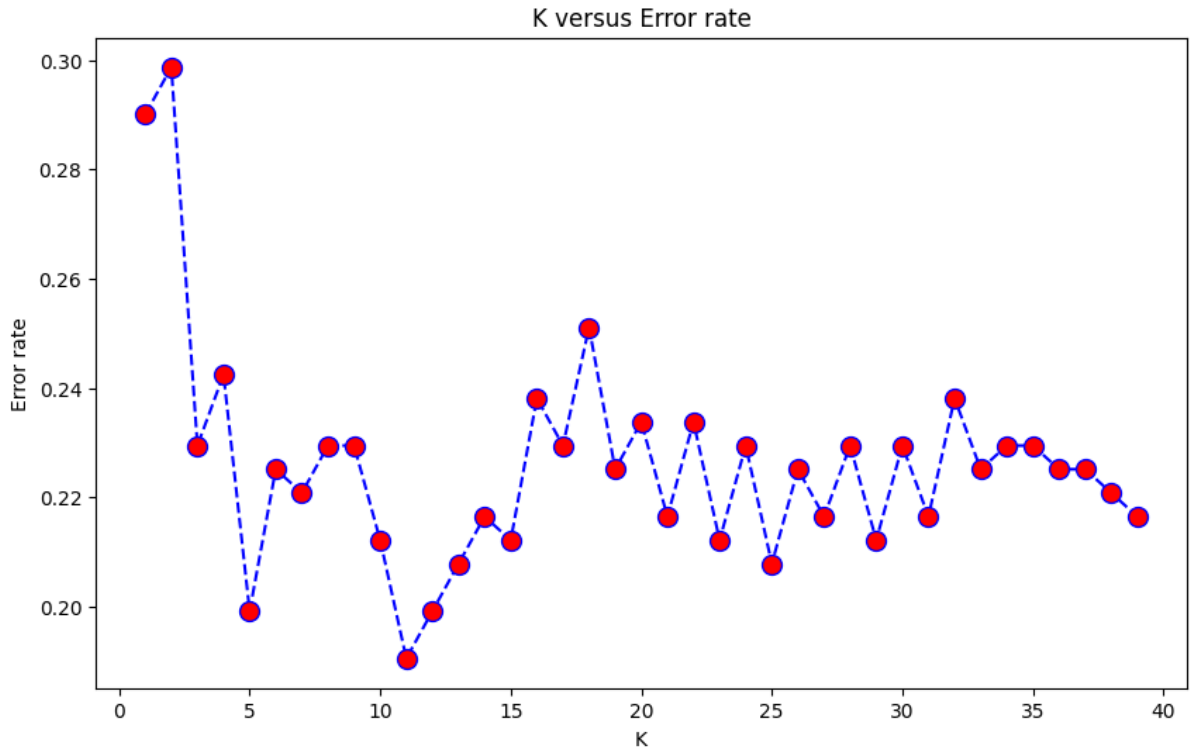
```
[[119  27]
 [ 40  45]]
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       146
           1       0.62      0.53      0.57        85

    accuracy                           0.71       231
   macro avg       0.69      0.67      0.68       231
weighted avg       0.70      0.71      0.70       231
```

```
error_rate = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(xtrain, ytrain)
    pred_i = knn.predict(xtest)

    error_rate.append(np.mean(pred_i != ytest))
```

```
plt.figure(figsize=(10, 6))

plt.plot(range(1, 40), error_rate, color='blue', linestyle='--',
markersize=10, markerfacecolor='red', marker='o')

plt.title('K versus Error rate')

plt.xlabel('K')
plt.ylabel('Error rate')
```

OUTPUT:

```
Text(0, 0.5, 'Error rate')
```



 # lowest error rate at " 11 "

```
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(xtrain, ytrain)
predictions = knn.predict(xtest)

print(confusion_matrix(ytest, ypred))
print(classification_report(ytest, ypred))
```

OUTPUT :

```
[[119  27]
 [ 40  45]]
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       146
           1       0.62      0.53      0.57        85

    accuracy                           0.71       231
   macro avg       0.69      0.67      0.68       231
weighted avg       0.70      0.71      0.70       231
```

Lab-6/2203A52151/sec-AB:

Lab06: *Logistic Regression using the pre-defined library. Analysis of different training and testing splits ranges*

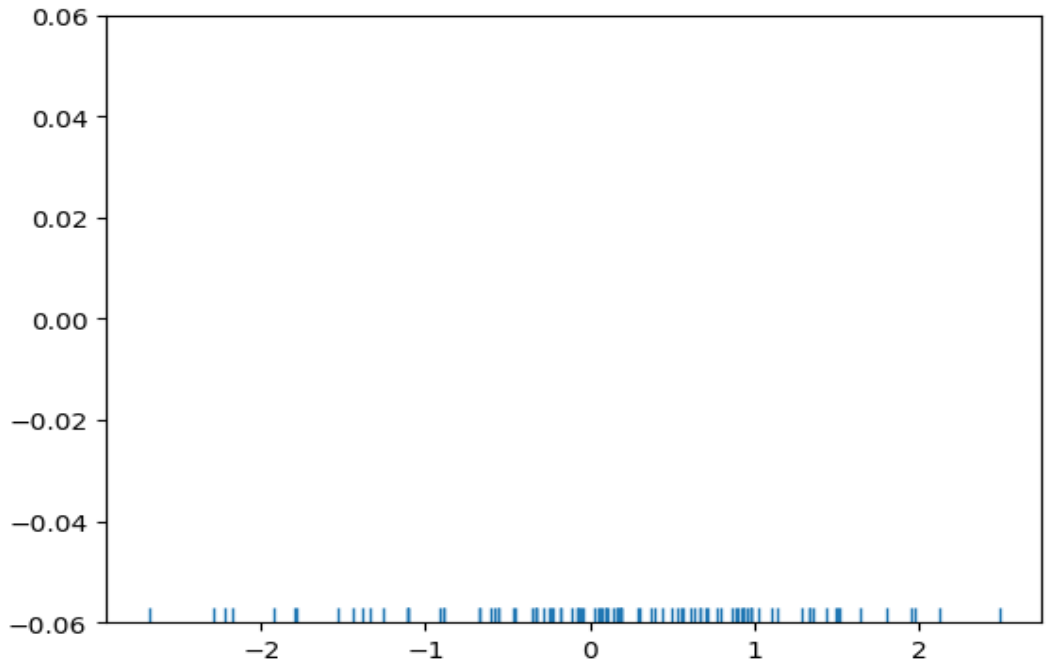
Implement Kernel Density Estimation for Feature Space.

OPTIMIZATION

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

```
#create dataset
dataset=np.random.randn(100)
#create another rugplot
sns.rugplot(dataset);
#setup the x axis for the plot
x_min=dataset.min()-2
x_max=dataset.max()+2
#100 equally spaced points from x_min to x_max
x_axis=np.linspace(x_min,x_max,100)
```

OUTPUT:




```
print(x_min,x_max)
```

OUTPUT:

```
-4.683674293857251 4.488173448116584
```

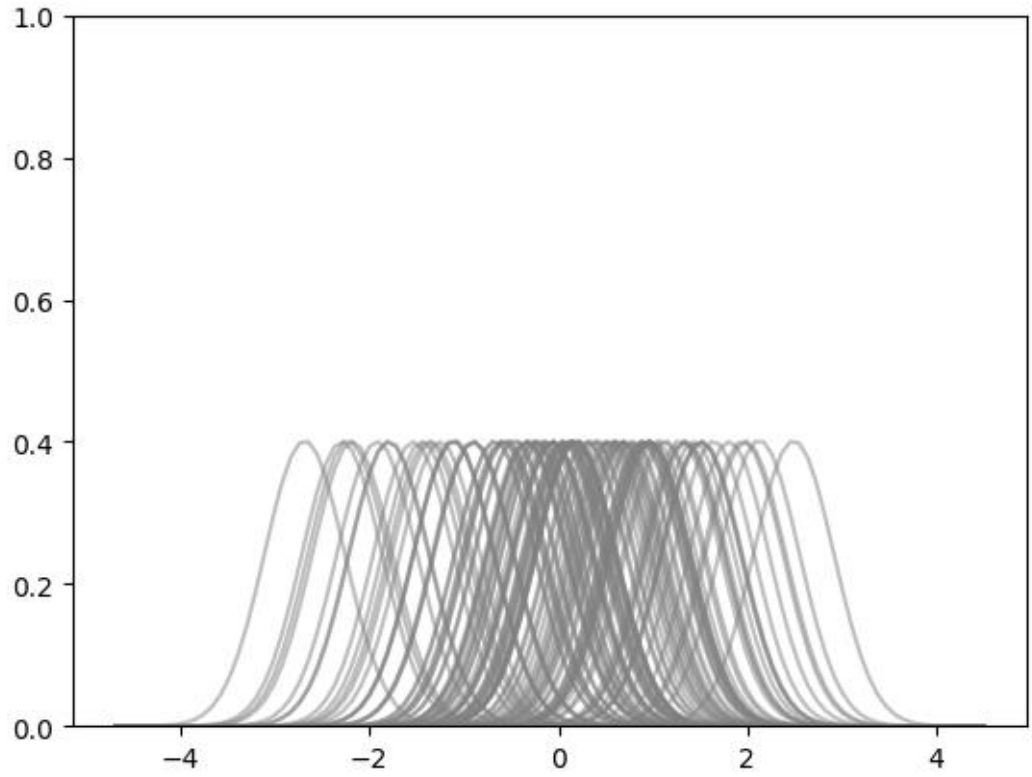
```
#setup the bandwith,for info on this:
url='http://en.wikipedia.org/wiki/Kernel\_density\_estimation\#Practical_estimation_of_the_bandwidth'
bandwidth=((4*dataset.std()**0.5)/(3*len(dataset)))**.2
bandwidth
```

OUTPUT:

```
0.42346006202198466
```

```
#create an empty kernel list
kernel_list=[]
#plot each basis function
for data_point in dataset:
  #create a kernel for each point
  kernel=stats.norm(data_point,bandwidth).pdf(x_axis)
  kernel_list.append(kernel)

  #scale for plotting
  kernel=kernel/kernel.max()
  kernel=kernel*.4
  plt.plot(x_axis,kernel,color='grey',alpha=0.5)

  plt.ylim(0,1)
```

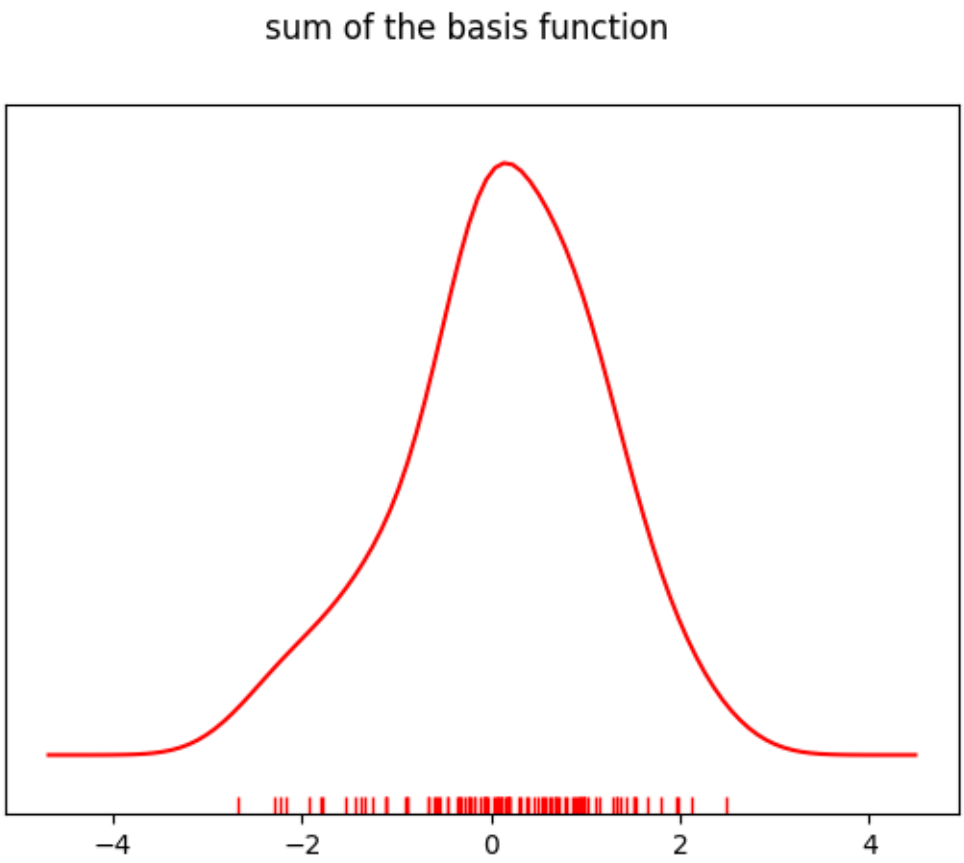
OUTPUT:




```
#to get the kde plot we can sum those basic functions
#plot the sum of the basis function
sum_of_kde=np.sum(kernel_list,axis=0)
#plot figure
fig=plt.plot(x_axis,sum_of_kde,color='red')
#add  the intial rugplot
sns.rugplot(dataset,c='red')
#get rid of y tick marks
plt.yticks([])
#set title
plt.suptitle("sum of the basis function")
```

OUTPUT:

```
Text(0.5, 0.98, 'sum of the basis function')
```



REGRESSION

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
!pip install cv
```

[]

```
!pip install cv
```

account_circle

```
Collecting cv
  Downloading cv-1.0.0-py3-none-any.whl (7.3 kB)
Installing collected packages: cv
Successfully installed cv-1.0.0
```

```
import numpy as np
import pandas as pd
import cv

from sklearn.datasets import fetch_california_housing

from sklearn import metrics
from sklearn import model_selection
from sklearn import linear_model

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rcParams.update({'font.size':16})
```

OUTPUT:

```
ERROR:root:Error disabling cv.imshow().
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-
packages/google/colab/_import_hooks/_cv2.py", line 83, in load_module
    cv_module.imshow,
AttributeError: module 'cv' has no attribute 'imshow'
```

```
housing = fetch_california_housing()
```

```
dir(housing)
```

OUTPUT:

```
['DESCR', 'data', 'feature_names', 'frame', 'target', 'target_names']
```

```
housing.data.shape
```

OUTPUT:

```
(20640, 8)
```

```
housing.target.shape
```

OUTPUT:

```
(20640,)
```

```
ridgereg=linear_model.Ridge()
```

```
X_train,X_test,y_train,y_test=model_selection.train_test_split(
```

```
    housing.data,housing.target,test_size=0.1,random_state=42
)
```

```
ridgereg.fit(X_train,y_train)
```

OUTPUT:

Ridge

```
Ridge()
```

```
metrics.mean_squared_error(y_train,ridgereg.predict(X_train))
```

OUTPUT:

0.5205320691482117

```
ridgereg.score(X_train,y_train)
```

OUTPUT:

```
0.6090109889897377
```

```
y_pred=ridgereg.predict(X_test)
```

```
metrics.mean_squared_error(y_test,y_pred)
```

OUTPUT:

0.5589961818225975

```
plt.figure(figsize=(10,6))
plt.plot(y_test,linewidth=3,label='ground truth')
plt.plot(y_pred,linewidth=3,label='predicted')
plt.legend(loc='best')
plt.xlabel('test data points')
plt.ylabel('target value')
```

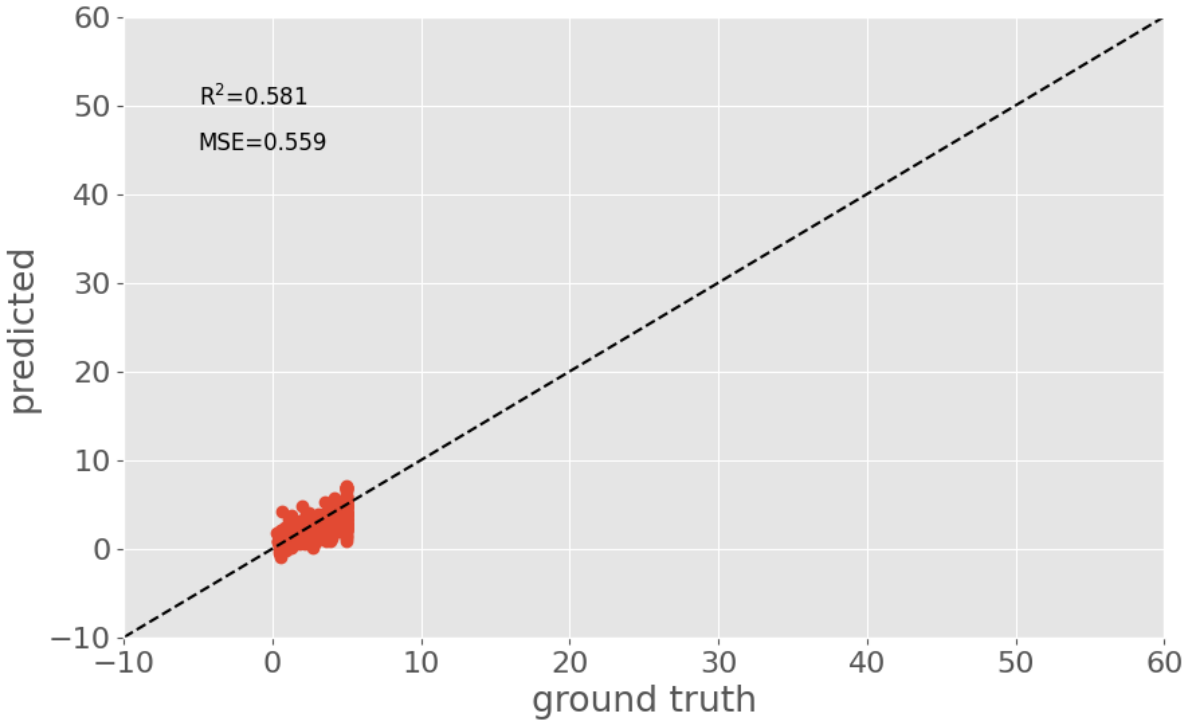
OUTPUT:

```
Text(0, 0.5, 'target value')
```


```
plt.figure(figsize=(10,6))
plt.plot(y_test,y_pred,'o')
plt.plot([-10,60],[-10,60],'k--')
plt.axis([-10,60,-10,60])
plt.xlabel('ground truth')
plt.ylabel('predicted')

scorestr=r'R$^2$=%.3f' % ridgereg.score(X_test,y_test)
errstr='MSE=%.3f' % metrics.mean_squared_error(y_test,y_pred)
plt.text(-5,50,scorestr,fontsize=12)
plt.text(-5,45,errstr,fontsize=12)
```

OUTPUT:

```
Text(-5, 45, 'MSE=0.559')
```


R²=0.581
MSE=0.559
predicted
ground truth
60
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
60

CODE-7:

CODE ASSIGNMENTS 07: *SVM and SVR for classification, regression,*
Implement Dimensionality Reduction using Principal Component Analysis (PCA)

Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
url="https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
column_names =
['sepal_length','sepal_width','petal_length','petal_width','class']
iris_data=pd.read_csv(url,names=column_names)
print(iris_data)
```

OUTPUT:

```
sepal_length  sepal_width  petal_length  petal_width           class
0             5.1          3.5           1.4          0.2     Iris-
setosa
1             4.9          3.0           1.4          0.2     Iris-
setosa
2             4.7          3.2           1.3          0.2     Iris-
setosa
3             4.6          3.1           1.5          0.2     Iris-
setosa
4             5.0          3.6           1.4          0.2     Iris-
setosa
..            ...          ...           ...          ...
...
145           6.7          3.0           5.2          2.3  Iris-
virginica
146           6.3          2.5           5.0          1.9  Iris-
virginica
147           6.5          3.0           5.2          2.0  Iris-
virginica
148           6.2          3.4           5.4          2.3  Iris-
virginica
149           5.9          3.0           5.1          1.8  Iris-
virginica

[150 rows x 5 columns]
```

```
from sklearn.preprocessing import StandardScaler
```

```
features=['sepal_length','sepal_width','petal_length','petal_width']
x= iris_data.loc[:,features].values
y= iris_data.loc[:,['class']].values
x = StandardScaler().fit_transform(x)
pca=PCA(n_components=2)
principalComponents=pca.fit_transform(x)
principalDataframe=pd.DataFrame(data=principalComponents,columns=['pc1'
,'pc2'])
targetDataframe=iris_data[['class']]
newDataframe=pd.concat([principalDataframe,targetDataframe],axis=1)
```

```
newDataframe
```

OUTPUT:

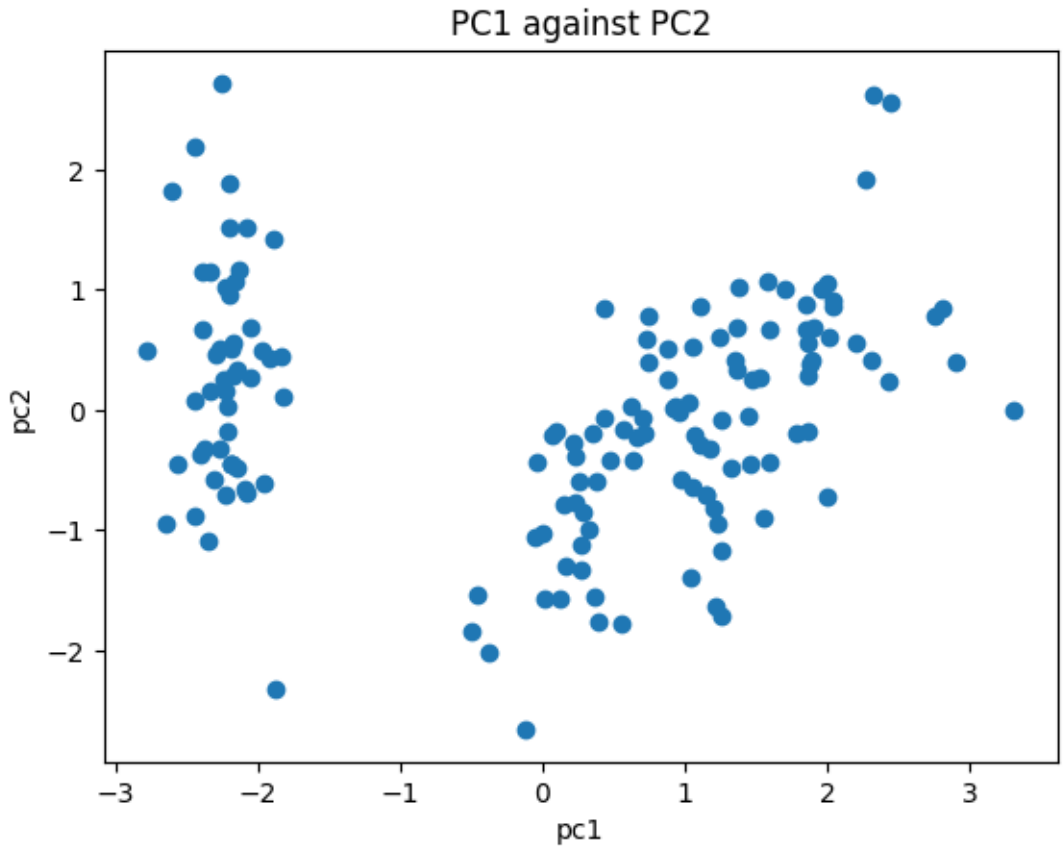
	pc1	pc2	class
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica
148	1.376391	1.016362	Iris-virginica
149	0.959299	-0.022284	Iris-virginica

150 rows × 3 columns

```
plt.scatter(principalDataframe.pc1,principalDataframe.pc2)
plt.title('PC1 against PC2')
plt.xlabel('pc1')
plt.ylabel('pc2')
```

OUTPUT:

```
Text(0, 0.5, 'pc2')
```



CODE-8:

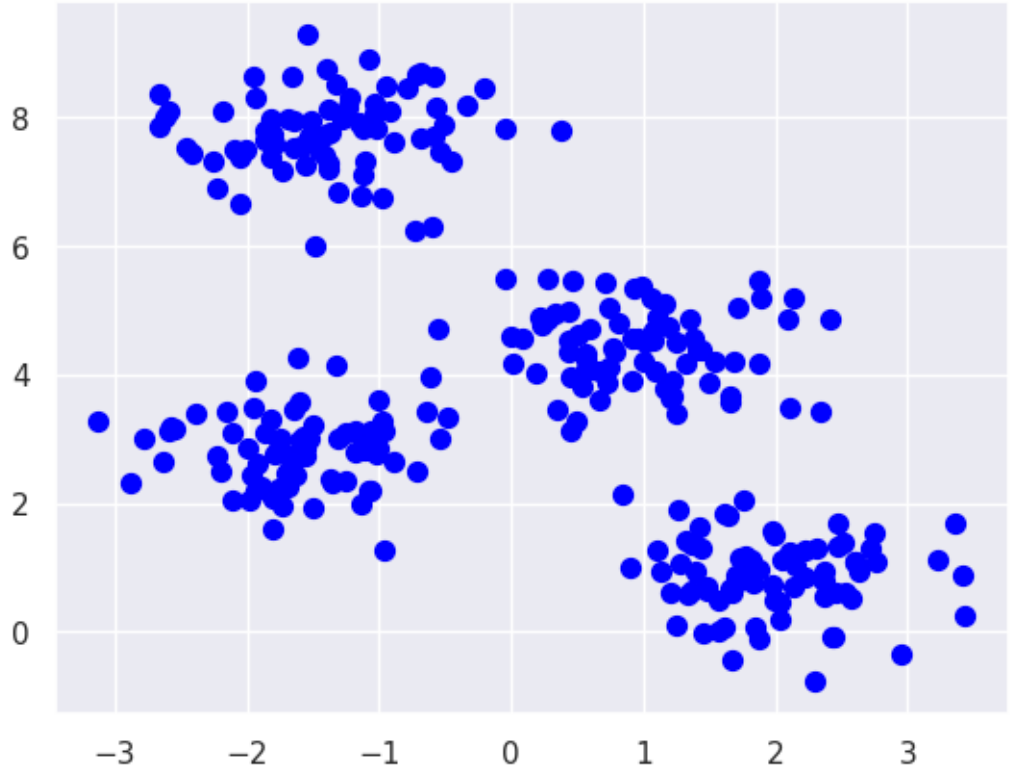
CODE ASSIGNMENTS 08: *L2 regularization using the predefined library, comparing the results with ordinary regression.*

Implement K-Means Clustering using Synthetic Data

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set() #plot styling
import numpy as np
```

```
from sklearn.datasets import make_blobs
X, y_true
=make_blobs(n_samples=300,centers=4,cluster_std=0.60,random_state=0)
plt.scatter(X[:,0],X[:,1],s=50,color='blue');
```

OUTPUT:

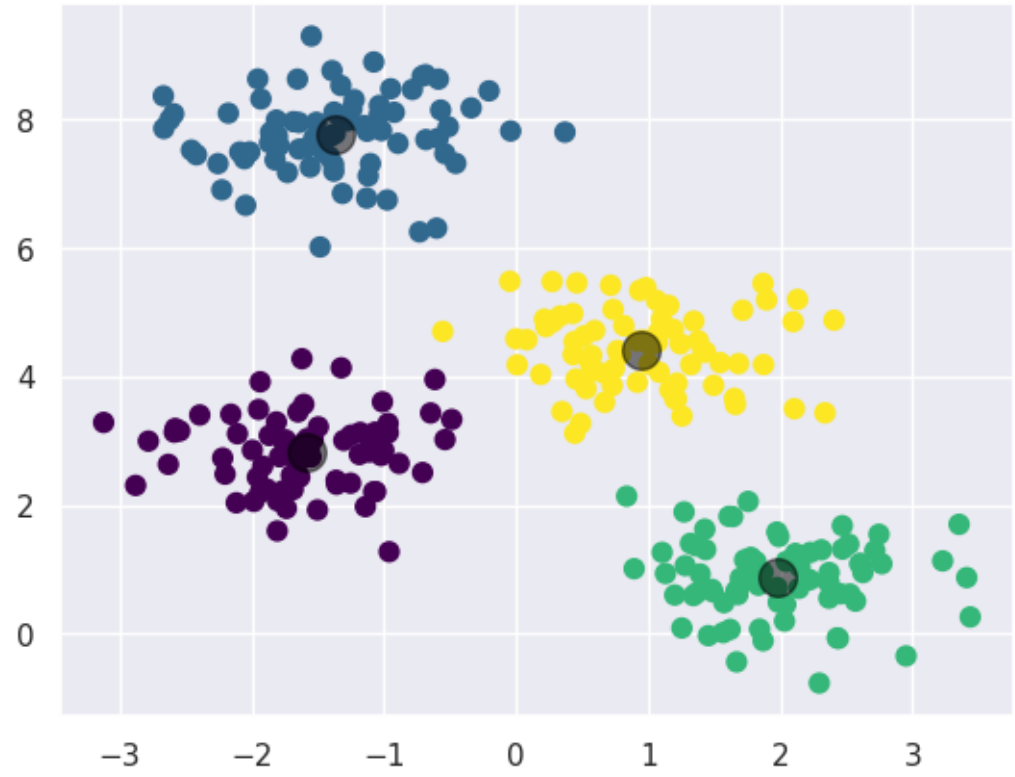



```
from sklearn.cluster import KMeans
Kmeans=KMeans(n_clusters=4,n_init=10)
Kmeans.fit(X)
```

```
y_Kmeans=Kmeans.predict(X)
```

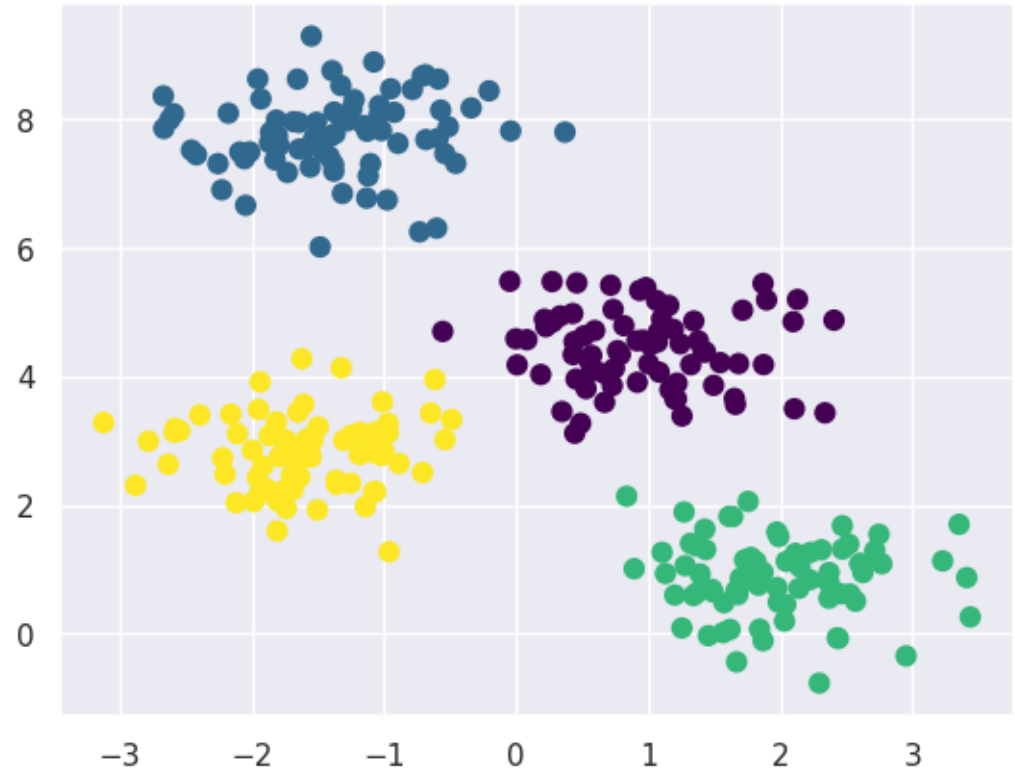
```
plt.scatter(X[:,0],X[:,1],c=y_Kmeans,s=50,cmap='viridis')

centers = Kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='black',s=200,alpha=0.5);
```

OUTPUT:



```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
# 1. Randomly choose clusters
  rng = np.random.RandomState(rseed)
  i = rng.permutation(X.shape[0])[:n_clusters]
  centers = X[i]
  while True:
  # 2a. Assign labels based on closest center
    labels = pairwise_distances_argmin(X, centers)
  # 2b. Find new centers from means of points
    new_centers = np.array([X[labels == i].mean(0)
    for i in range(n_clusters)])
  # 2c. Check for convergence
    if np.all(centers == new_centers):
      break
    centers = new_centers
  return centers, labels
centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels,
s=50, cmap='viridis');
```

OUTPUT:

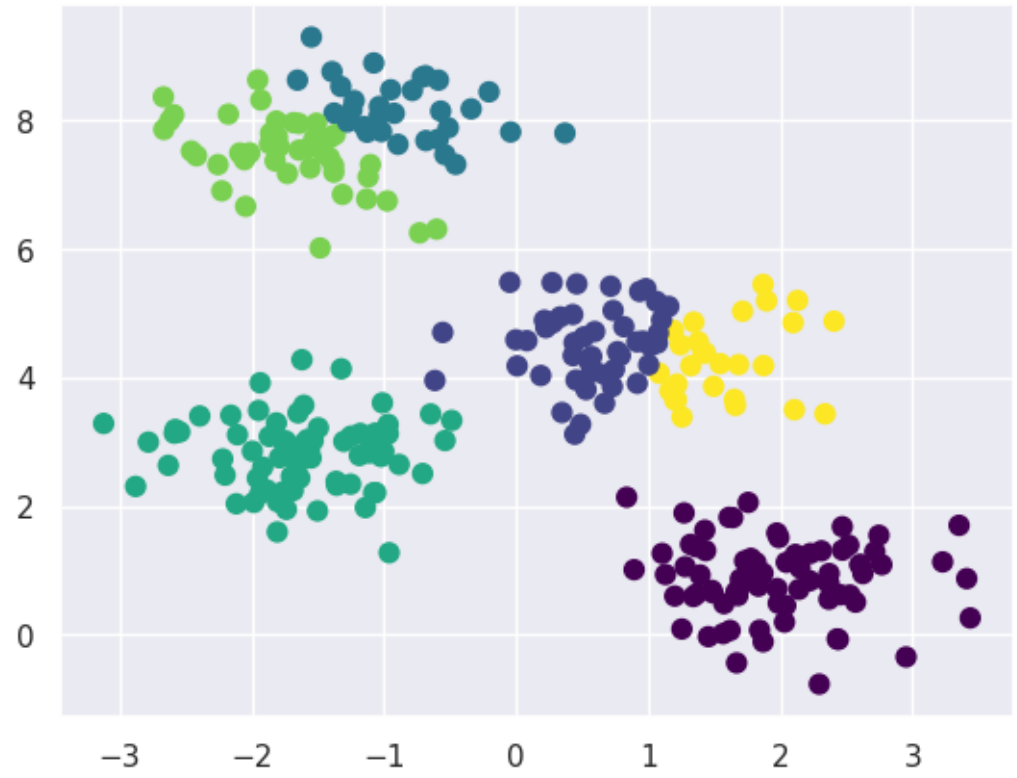


```
#sub-optimization of clusters
centers,labels =find_clusters(X,4,rseed=0)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

OUTPUT:

```
#how many clusters
labels = KMeans(6,random_state=0,n_init=10).fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

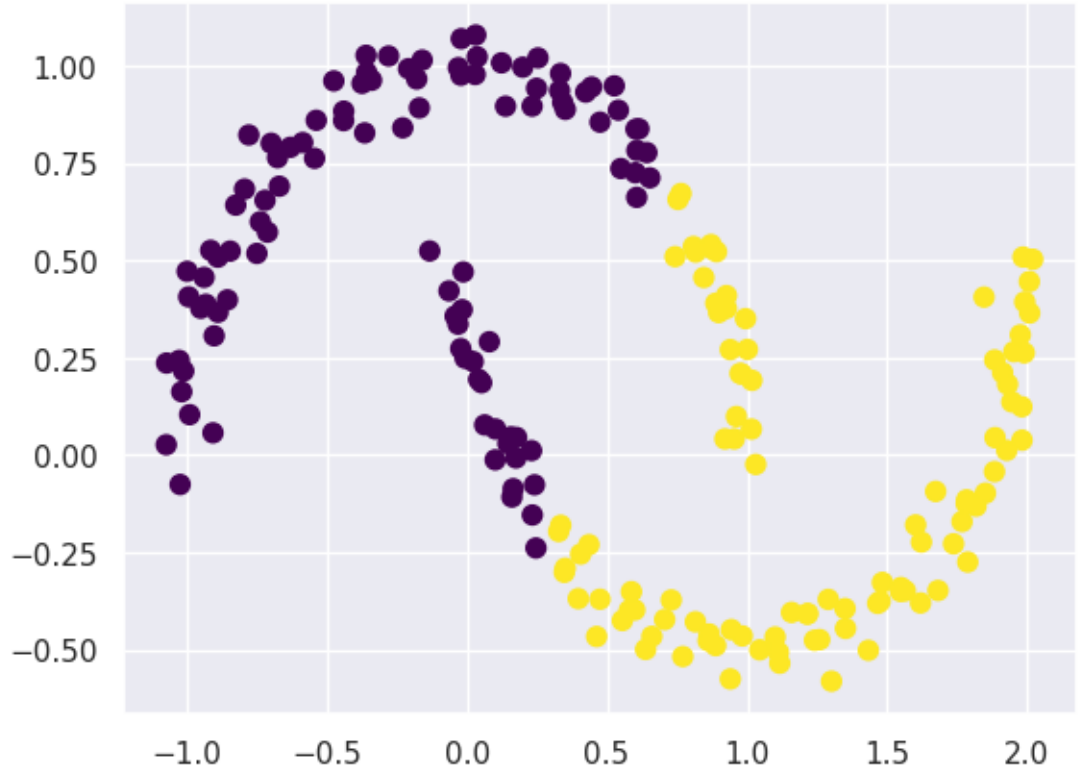
OUTPUT:




```
#limitations of K-means Algorithm
from sklearn.datasets import make_moons
X,y=make_moons(200,noise=.05,random_state=0)
```

```
labels =KMeans(2, random_state=0,n_init=10).fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

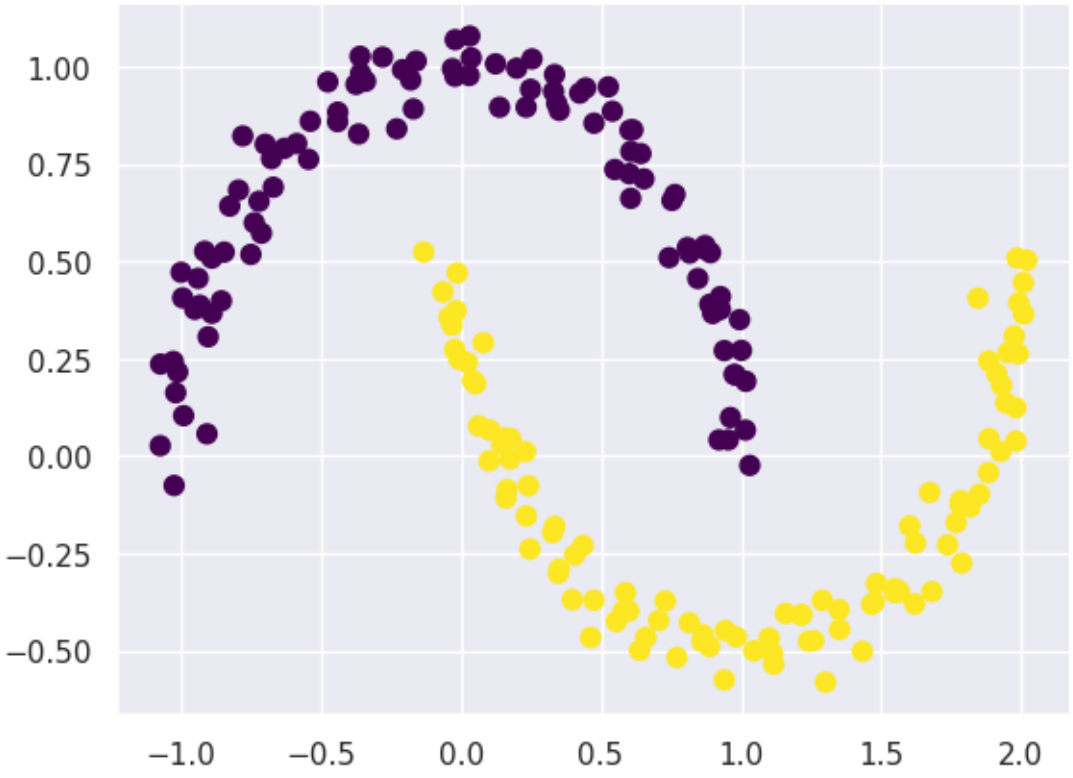
OUTPUT:




```
#kernal transformation
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```

OUTPUT:

```
/usr/local/lib/python3.10/dist-
packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Graph is
not fully connected, spectral embedding may not work as expected.
  warnings.warn(
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

CODE-9:

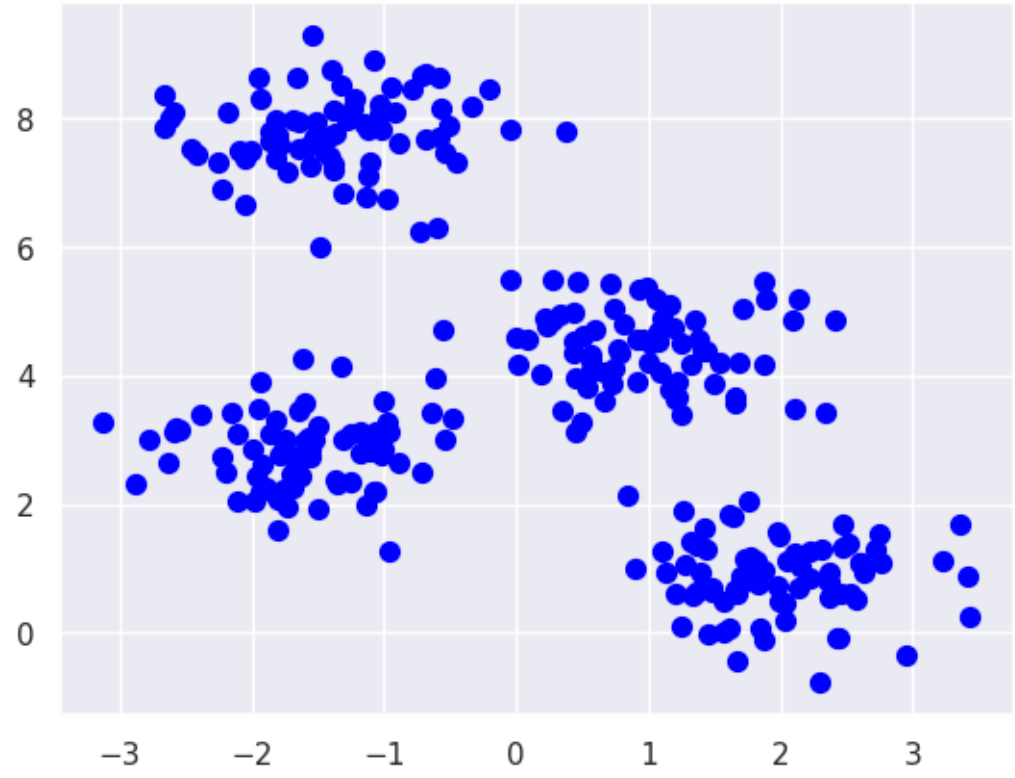
CODE ASSIGNMENTS 09: L2 regularization using the predefined library, comparing the results with ordinary regression.

Implement Gaussian Mixture Model using Synthetic Dataset

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set() #plot styling
import numpy as np
```

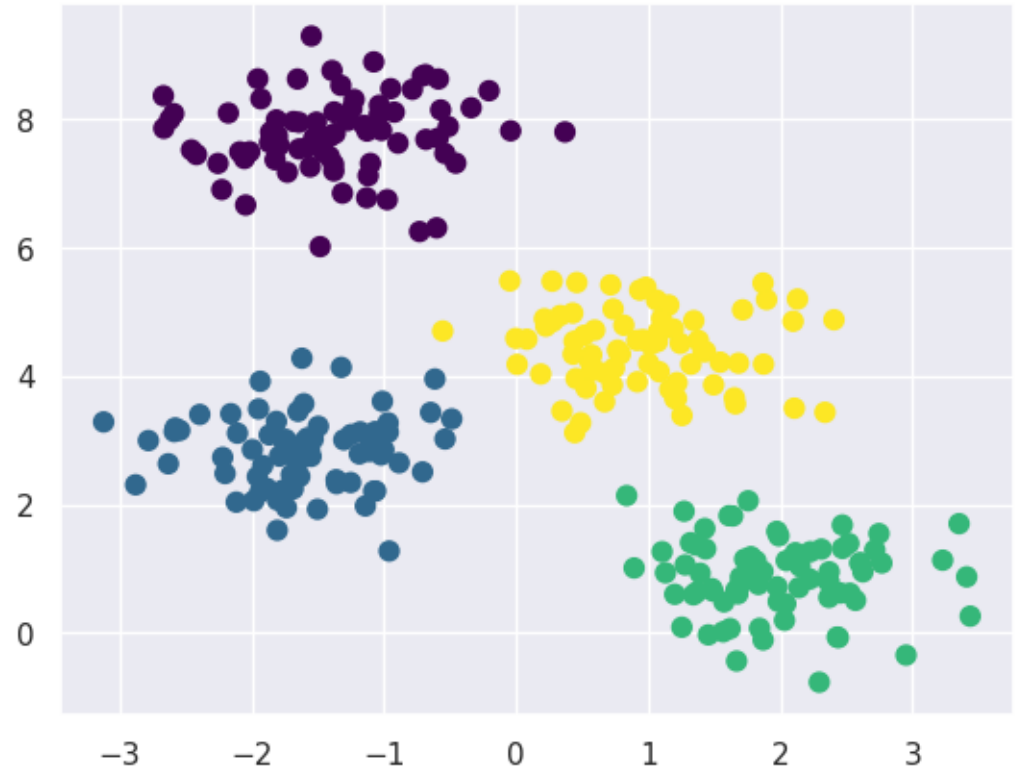
```
from sklearn.datasets import make_blobs
X, y_true
=make_blobs(n_samples=300,centers=4,cluster_std=0.60,random_state=0)
plt.scatter(X[:,0],X[:,1],s=50,color='blue');
```

OUTPUT:




```
#generlise to gaussian mixture models
from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=4).fit(X)
labels=gmm.predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

OUTPUT:



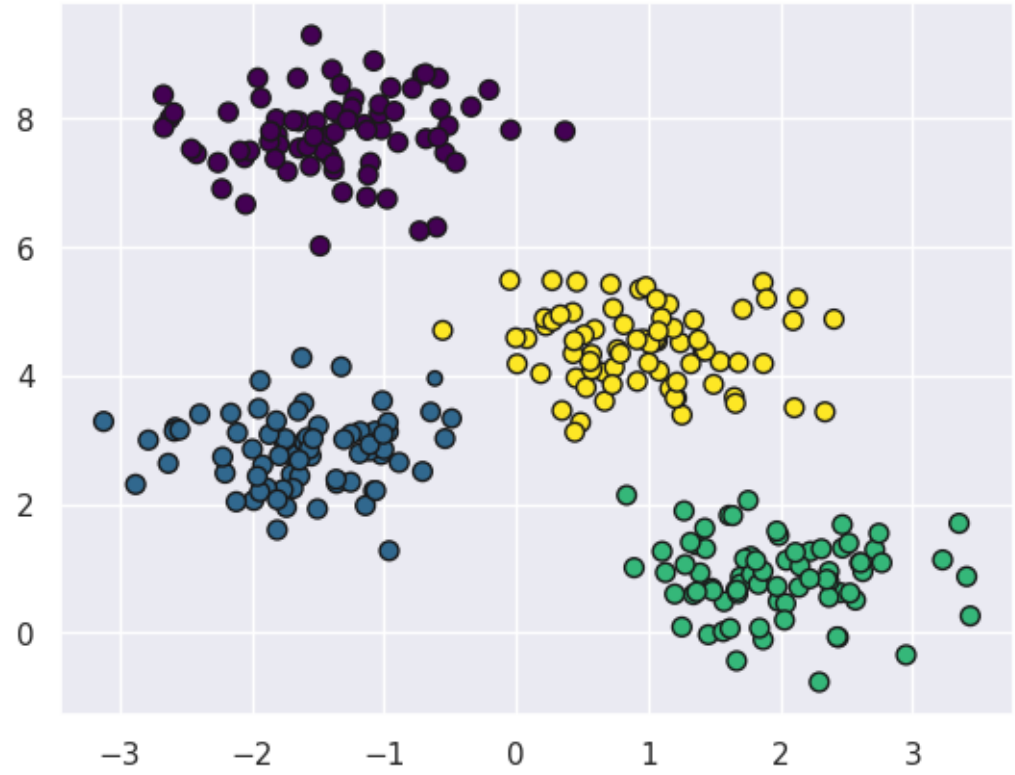

```
probs= gmm.predict_proba(X)
print(probs[:5].round(3))
```

OUTPUT:

```
[[0.    0.002 0.972 0.026]
 [1.    0.    0.    0.   ]
 [0.    0.    0.    1.   ]
 [1.    0.    0.    0.   ]
 [0.    0.    0.999 0.001]]
```

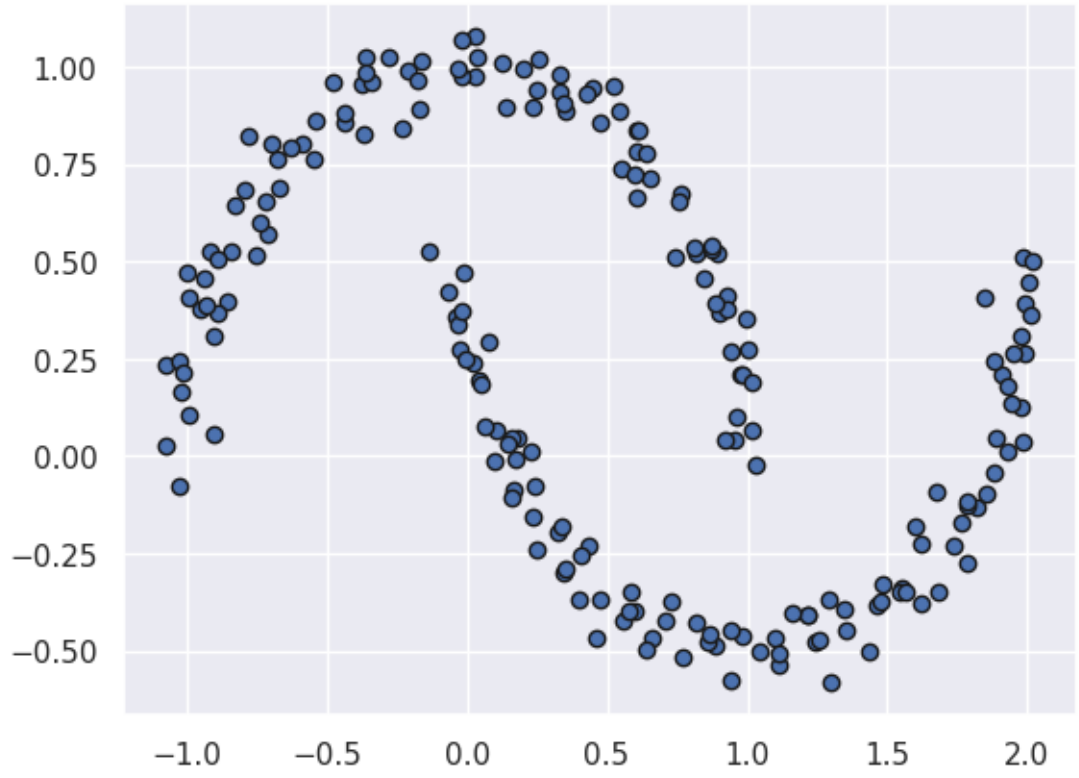
```
size=probs.max(1)/0.02
plt.scatter(X[:,0],X[:,1],c=labels,edgecolor='k',cmap='viridis',s=size)
;
```

OUTPUT:




```
from sklearn.datasets import make_moons
Xmoon,ymoon = make_moons(200,noise=0.05,random_state=0)
plt.scatter(Xmoon[:,0],Xmoon[:,1],edgecolor='k');
```

OUTPUT:



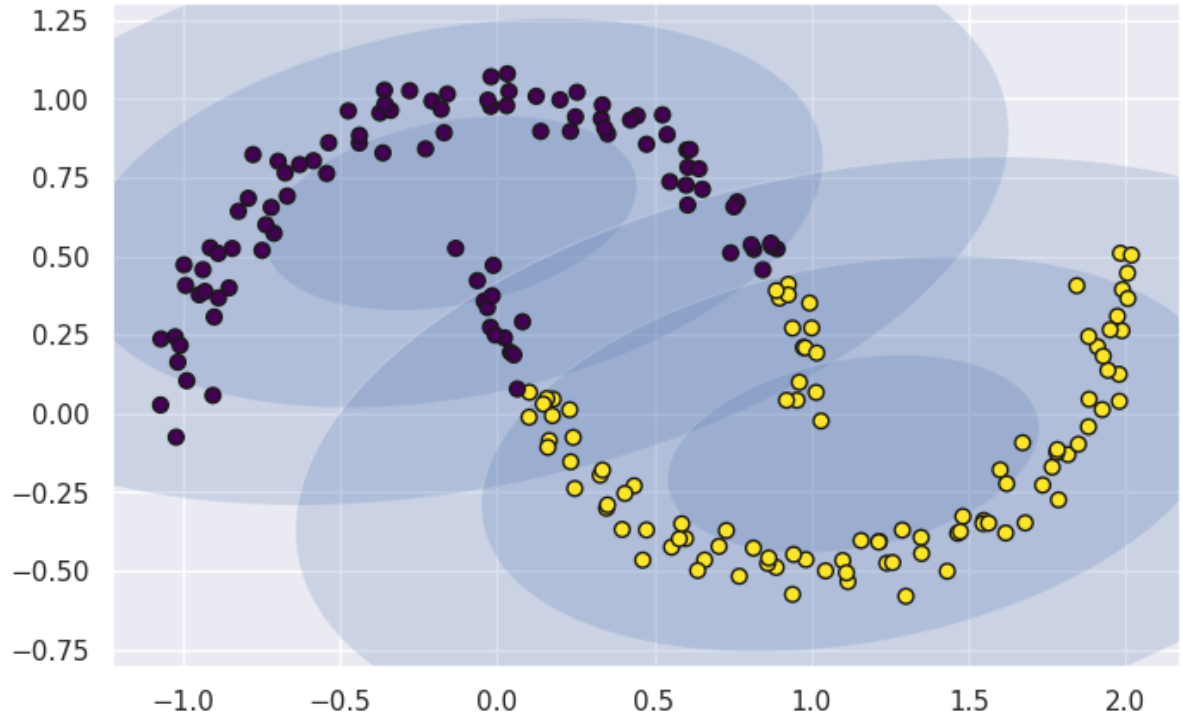

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))
```

```
def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis',
                   zorder=2, edgecolor='k')
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2, cmap='viridis',
edgecolor='k')
    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_,
gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

```
gmm2=
GaussianMixture(n_components=2,covariance_type='full',random_state=0)
plt.figure(figsize=(8,5))
plot_gmm(gmm2,Xmoon)
```

OUTPUT:

```
<ipython-input-56-4074ecdae225>:21: MatplotlibDeprecationWarning: Passing
the angle parameter of __init__() positionally is deprecated since
Matplotlib 3.6; the parameter will become keyword-only two minor releases
later.
  ax.add_patch(Ellipse(position, nsig * width, nsig * height,
```



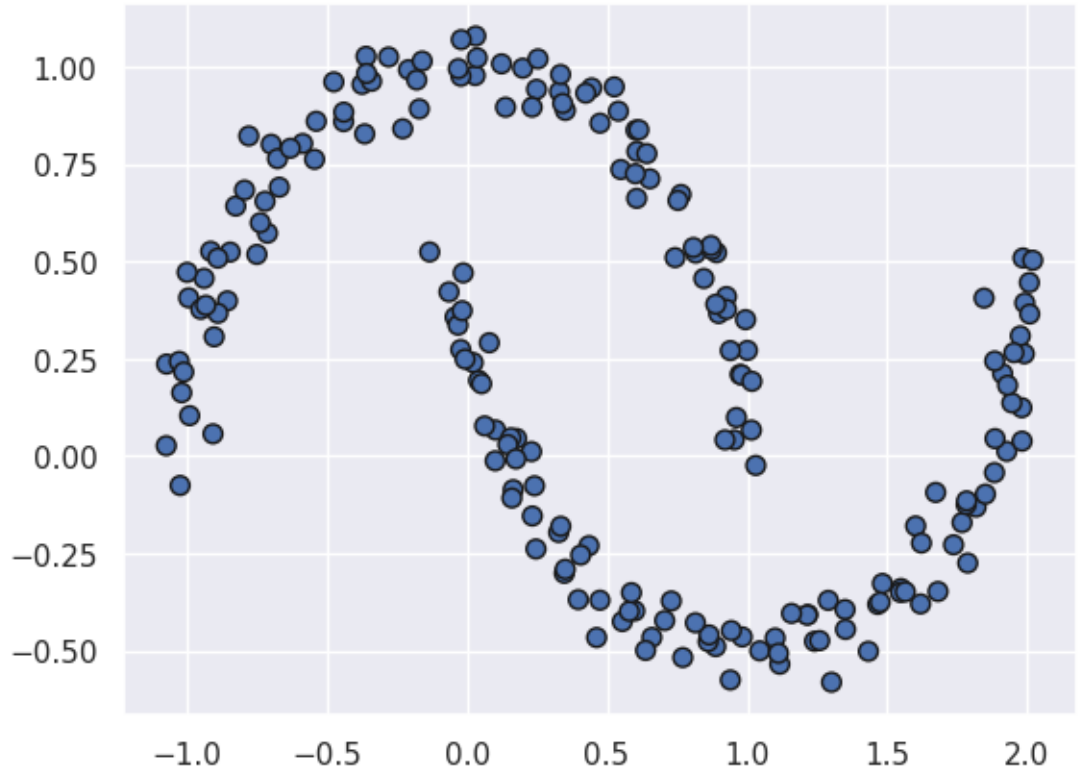
```
probs= gmm.predict_proba(Xmoon)
print(probs[:5].round(3))
```

OUTPUT:

```
[[0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]]
```

```
size = probs.max(1)/0.02
plt.scatter(Xmoon[:,0],Xmoon[:,1],edgecolor='k',s=size);
```

OUTPUT:



CODE-10:

CODE ASSIGNMENTS 10: Implement Support Vector Machine Classification using Breast Can-
cer Dataset

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

getting the data

```
from  sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

the dataset is presented in a dictionary form

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename', 'data_module'])
print(cancer['DESCR'])
```

OUTPUT:

```
. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the
class

    :Attribute Information:
        - radius (mean of distances from center to points on the
perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
```

```
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the
three
        worst/largest values) of these features were computed for each
image,
        resulting in 30 features.  For instance, field 0 is Mean
Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L.
Mangasarian
```

```
    :Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature
extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium
on
     Electronic Imaging: Science and Technology, volume 1905, pages
861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer
diagnosis and
     prognosis via linear programming. Operations Research, 43(4),
pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning
techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer
Letters 77 (1994)
     163-171.
```

```
cancer['feature_names']
```

OUTPUT:

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
'mean smoothness', 'mean compactness', 'mean concavity', 'mean concave
points', 'mean symmetry', 'mean fractal dimension', 'radius error',
```

```
'texture error', 'perimeter error', 'area error', 'smoothness error',
'compactness error', 'concavity error', 'concave points error',
'symmetry error', 'fractal dimension error', 'worst radius', 'worst
texture', 'worst perimeter', 'worst area', 'worst smoothness', 'worst
compactness', 'worst concavity', 'worst concave points', 'worst
symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

```
df.describe()
```

OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732	0.061867	0.018061

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
	24049	01036	298981	914129	40644	28133	79720	38803	27414	07060		33242	46258	602542	356993	28332	73336	08624	65732	61867	18061
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930	0.250400	0.071460

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.400000	1084.000000	0.146000	0.339100	0.382900	0.161400	0.317900	0.092080
max	28.110	39.280	188.50	2501.00	0.1634	0.3454	0.4268	0.2012	0.3040	0.0974	...	36.040	49.540	251.20	4254.00	0.2226	1.0580	1.2520	0.2910	0.6638	0.20750

mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0 0 0	0 0 0	0 0 0 0	0 0 0 0	0 0	0 0	0 0	0 0	0 0	4 0		0 0 0	0 0 0	0 0 0 0	0 0 0 0	0 0	0 0	0 0	0 0	0 0	0 0

8 rows × 30 columns

```
np.sum(pd.isnull(df).sum())
```

OUTPUT:

0

adding thetarget to the DataFrame

```
cancer['target']
```

OUTPUT:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1,
1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
cancer['target'].sum()
```

output:

357

```
adding the target data to the data frame
```

```
df['cancer'] = pd.DataFrame(cancer['target'])
df.head()
```

OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	cancer
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	cancer
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.070017	0.1812	0.056667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.18600	0.2750	0.089022	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.127900	0.2069	0.059999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.24300	0.3613	0.087578	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24144	0.105200	0.2597	0.097444	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.25755	0.6638	0.173000	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.058833	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.16250	0.2364	0.076788	0

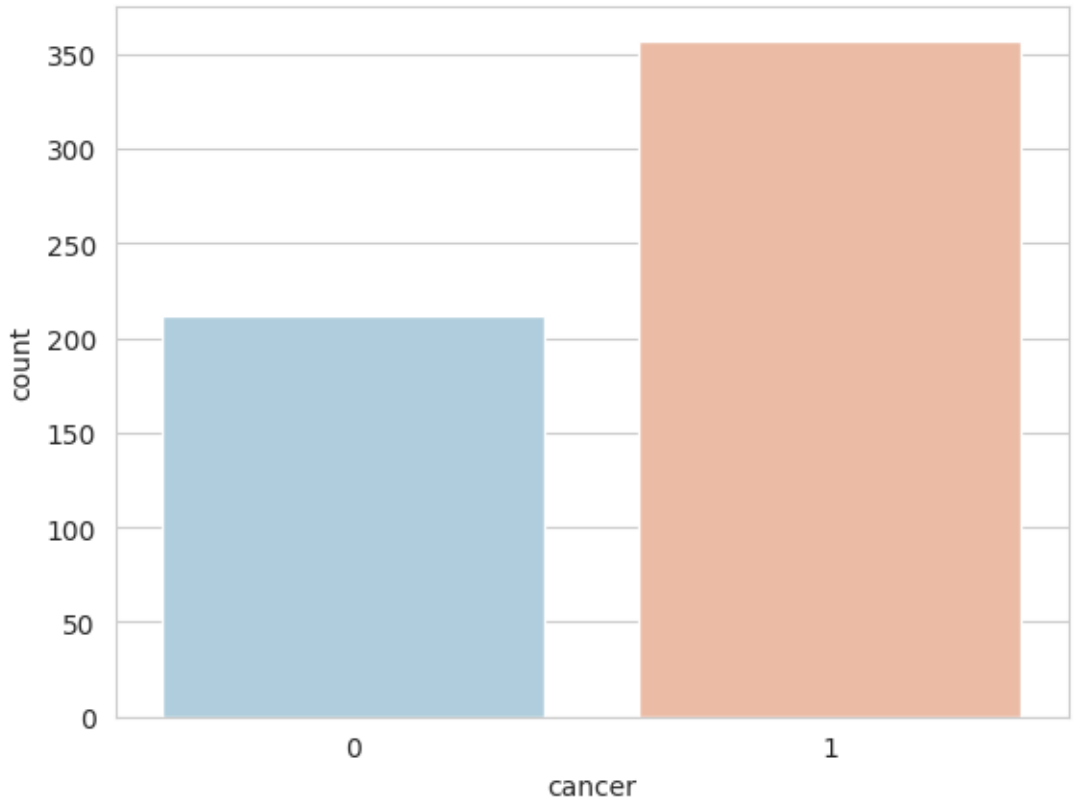
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	cancer
30																					

5 rows × 31 columns

```
sns.set_style('whitegrid')
sns.countplot(x='cancer',data=df,palette='RdBu_r')
```

OUTPUT:

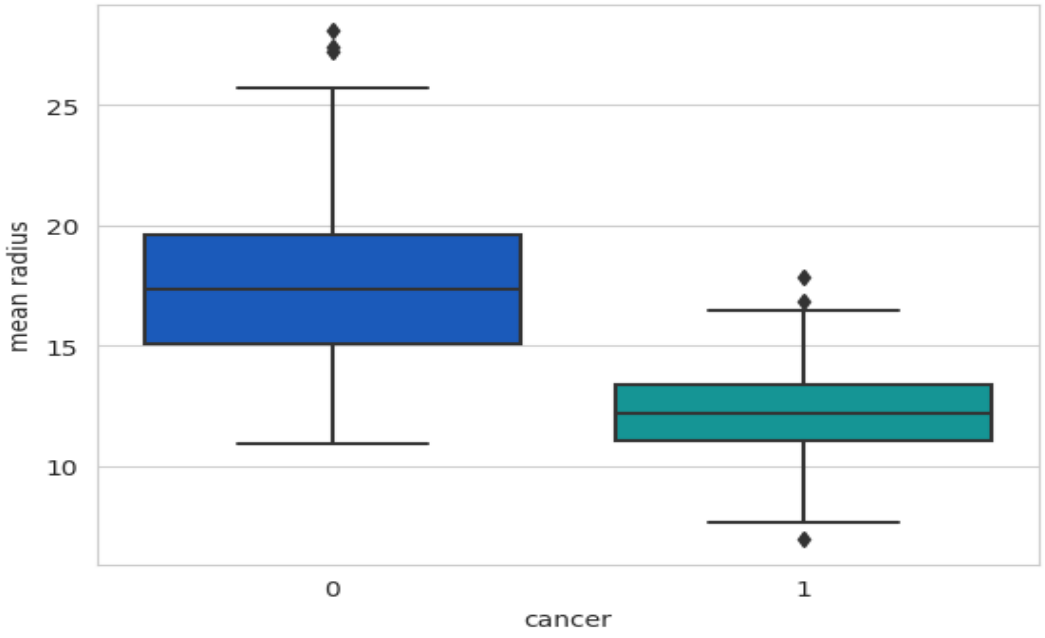
```
<Axes: xlabel='cancer', ylabel='count'>
```

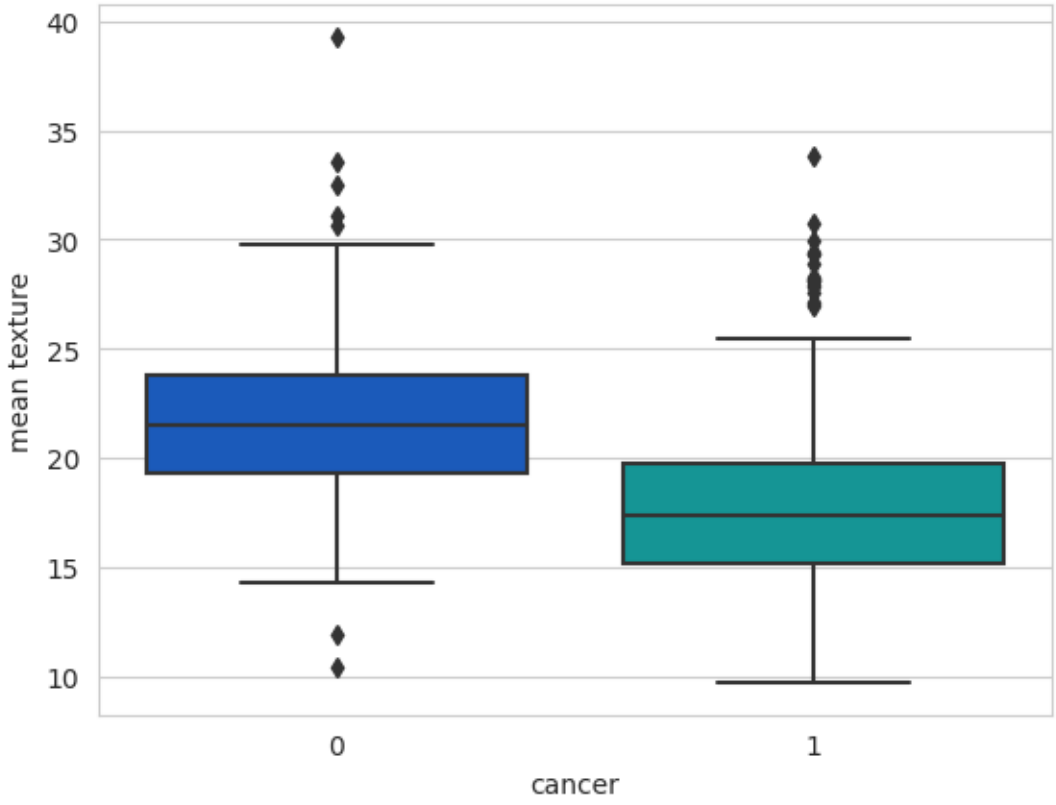


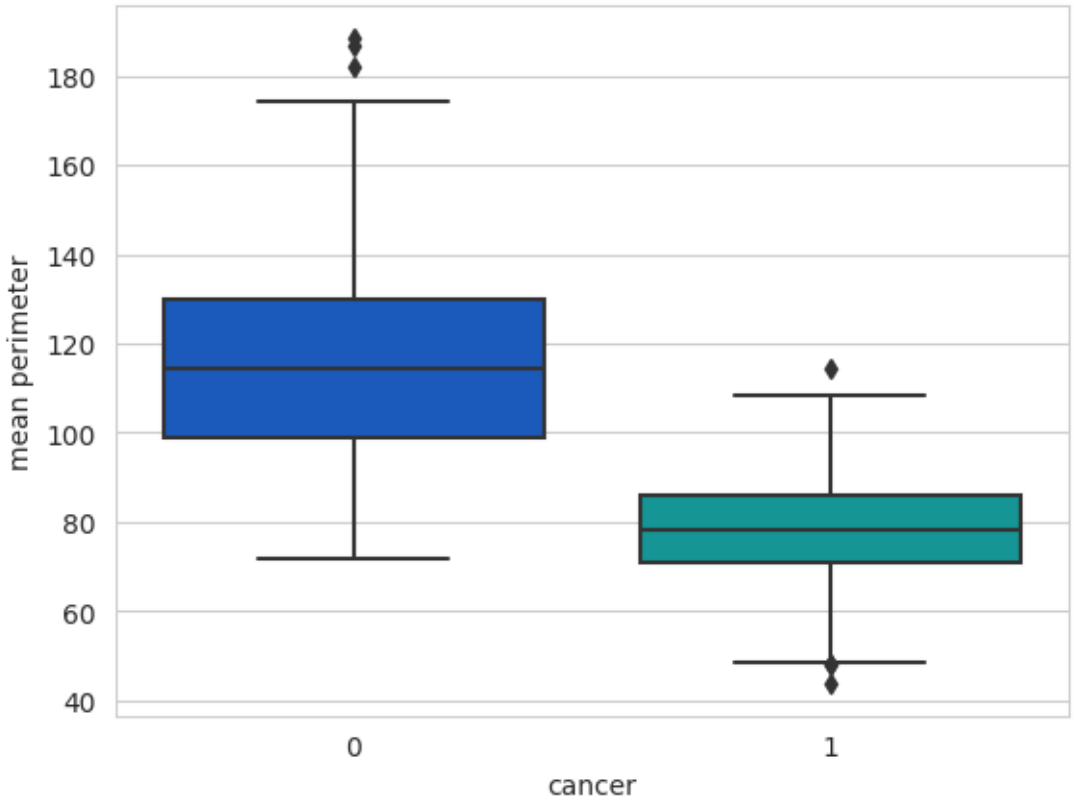
draw boxplots of all the mean features for 0 and cancer or not cancer

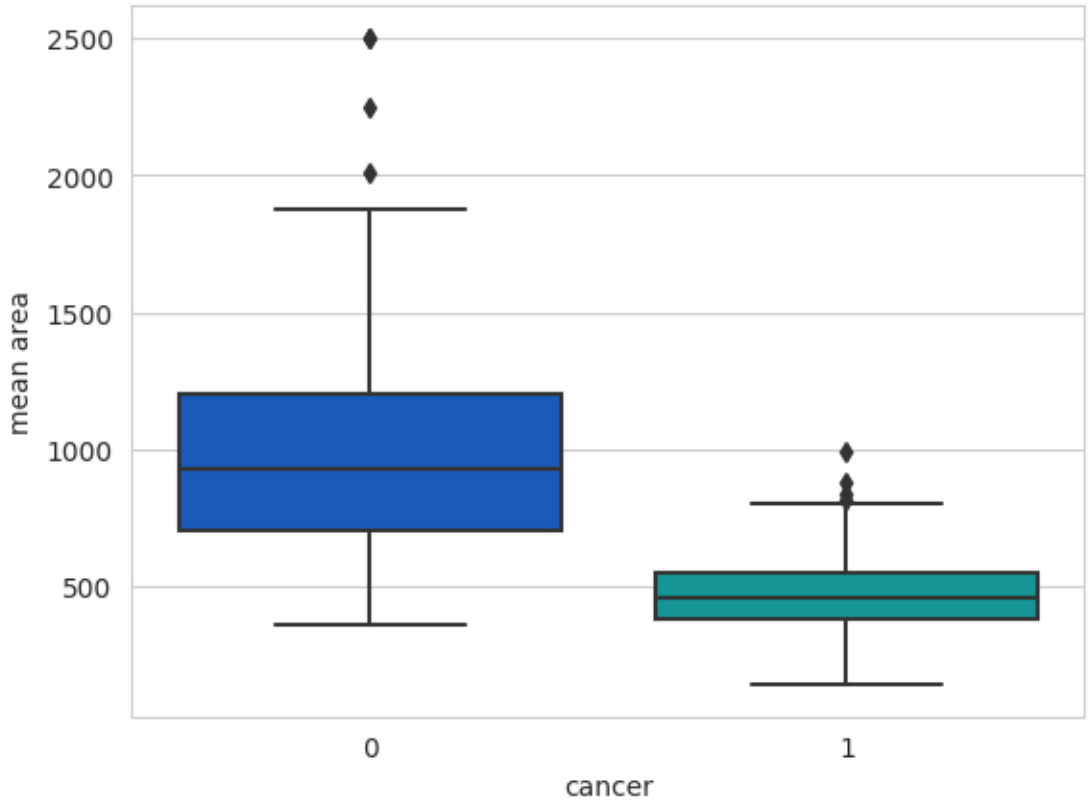
```
l=list(df.columns[0:10])
for i in range(len(l)-1):
    sns.boxplot(x='cancer',y=l[i], data=df, palette='winter')
    plt.figure()
```

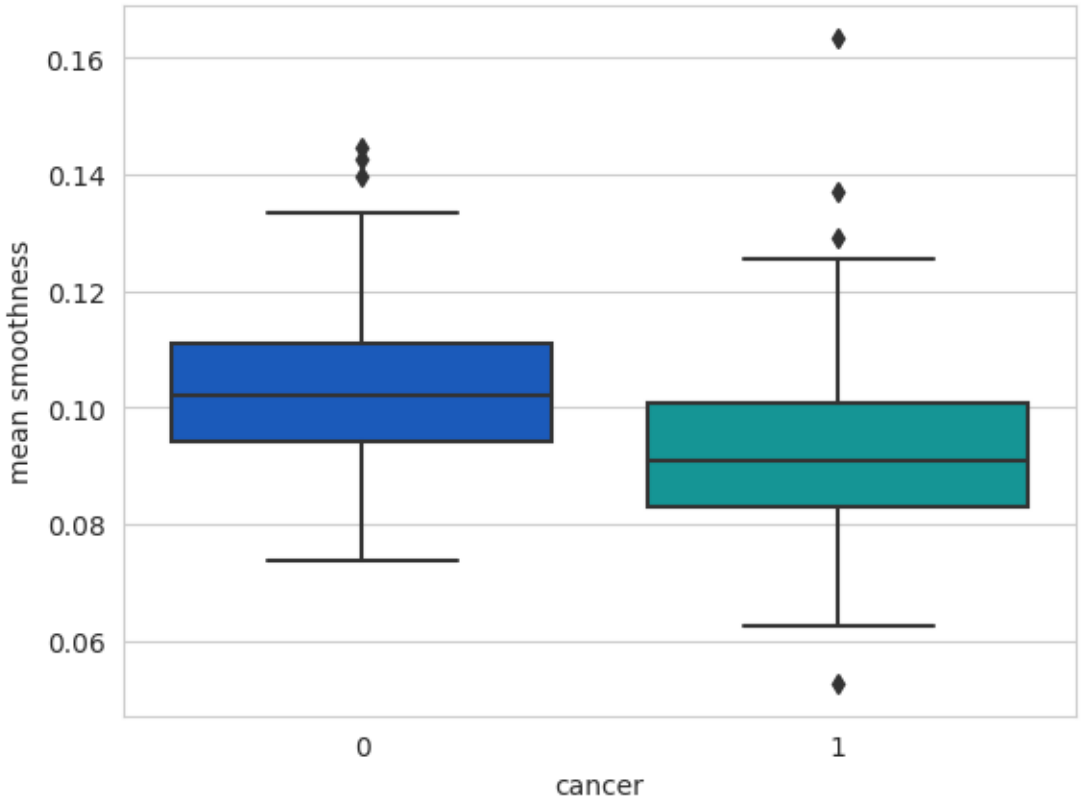
OUTPUT:

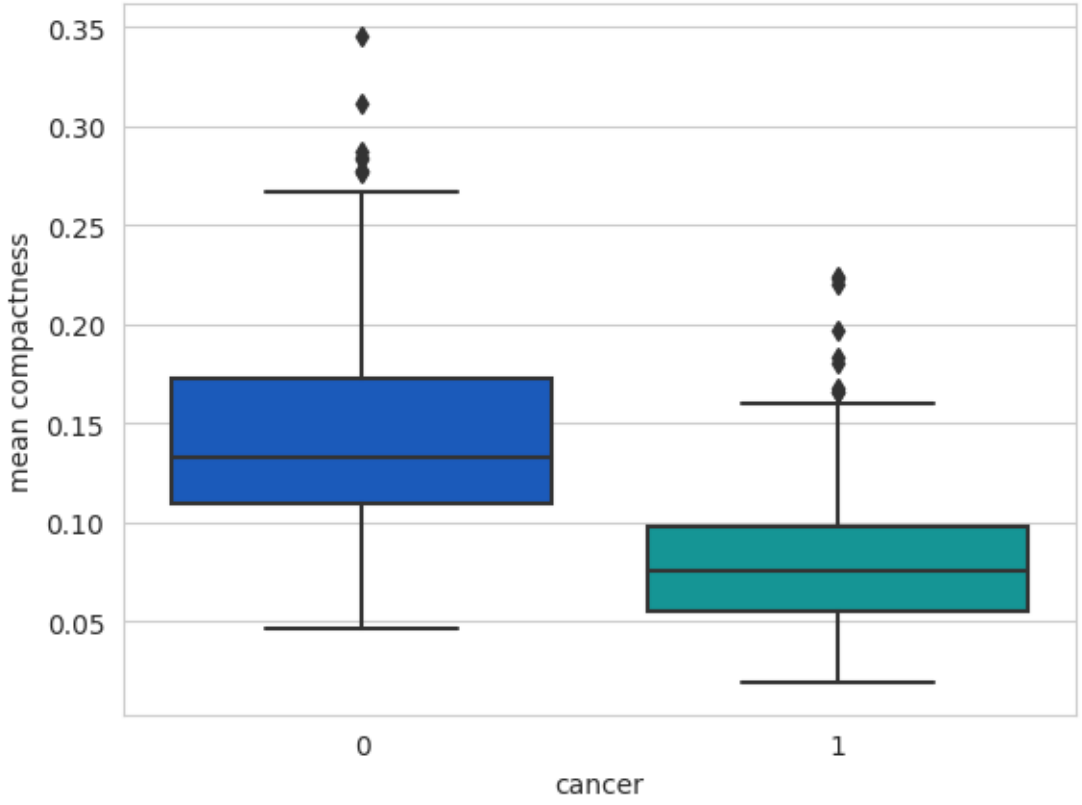

mean radius
25
20
15
10
0
1
cancer

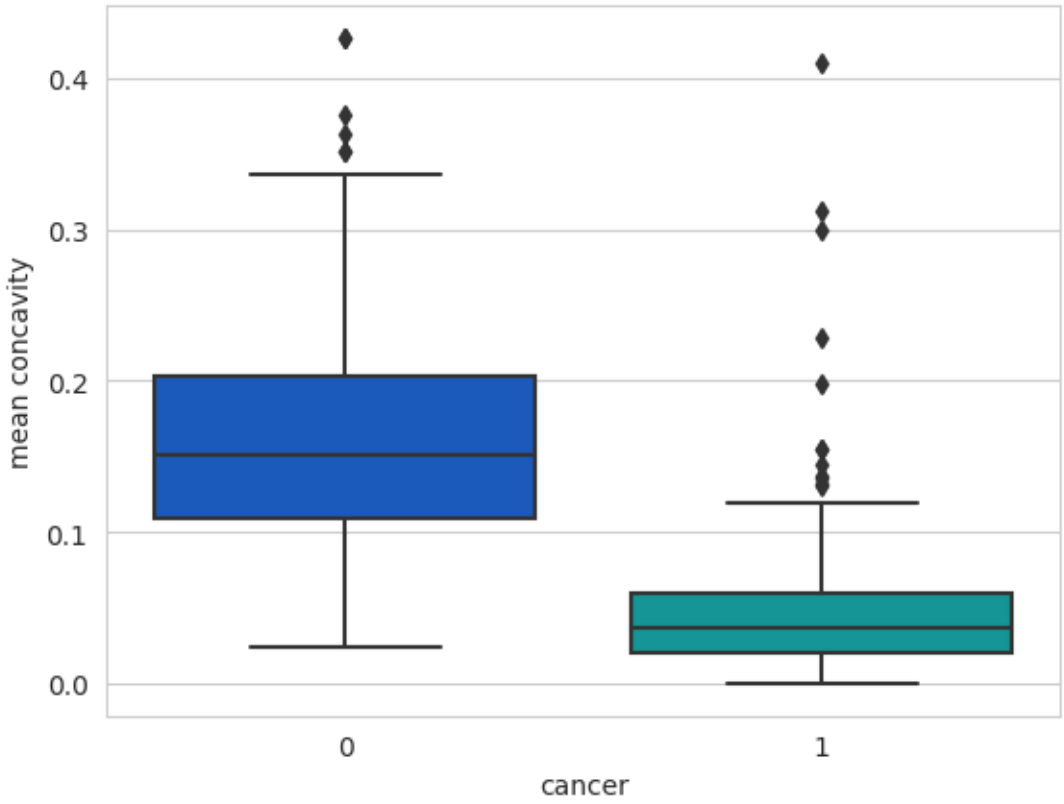

mean texture
40
35
30
25
20
15
10
0
1
cancer

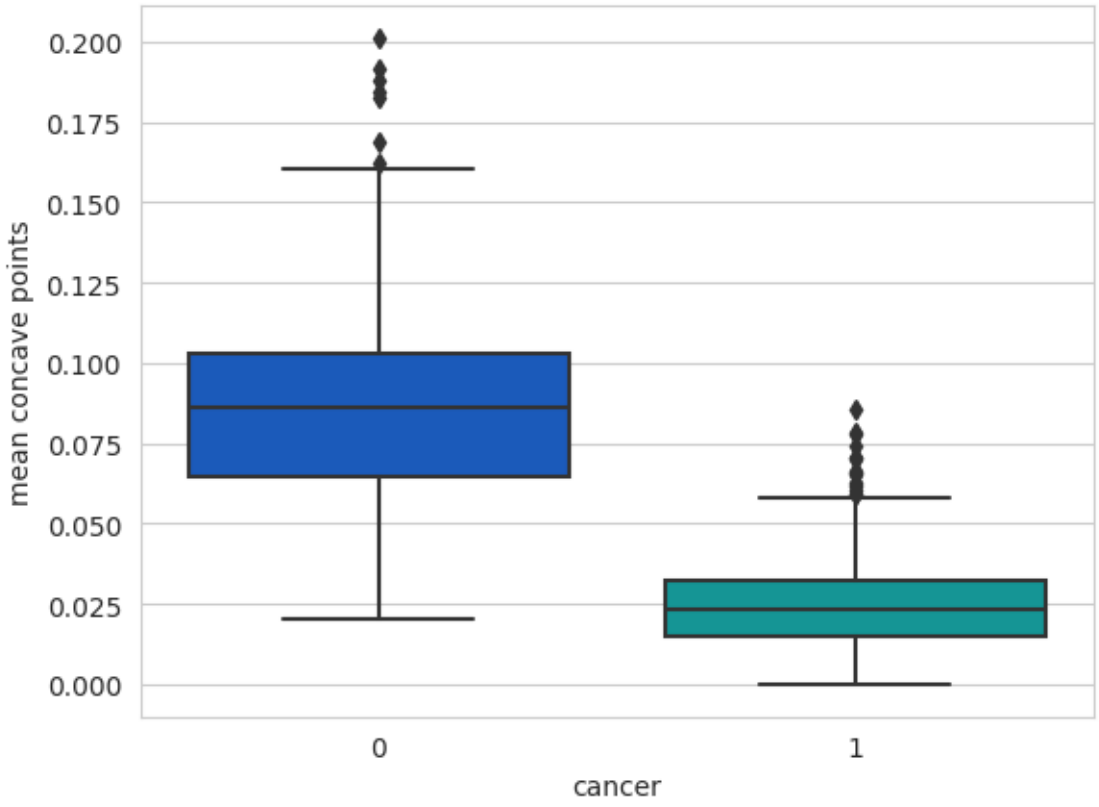

mean perimeter
180
160
140
120
100
80
60
40
0
1
cancer

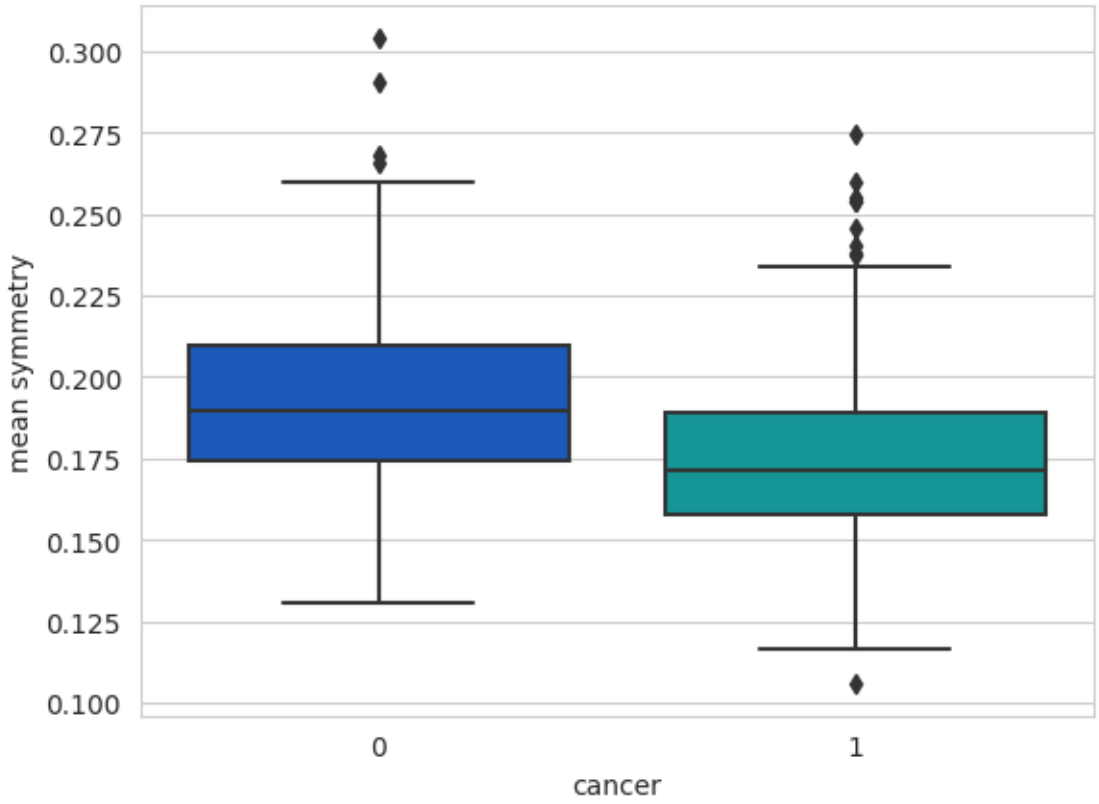

mean area
2500
2000
1500
1000
500
0
1
cancer


0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0
1
cancer


0.35
0.30
0.25
0.20
0.15
0.10
0.05
mean compactness
0
1
cancer


mean concavity
0.4
0.3
0.2
0.1
0.0
0
1
cancer


mean concave points
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
cancer

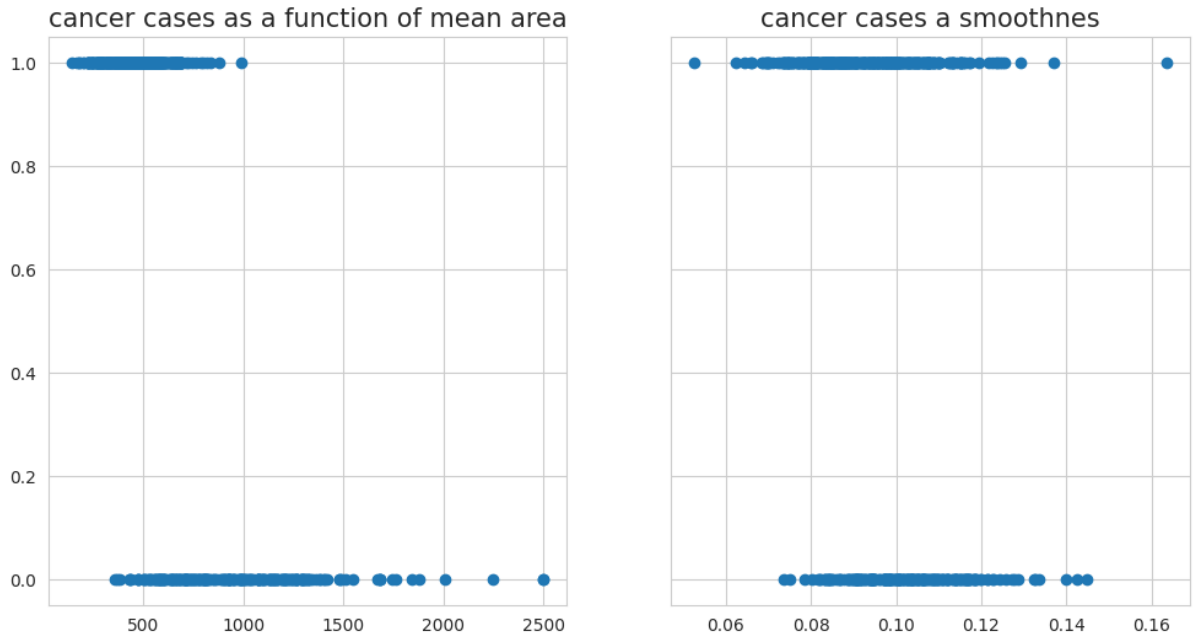



```
<Figure size 640x480 with 0 Axes>
```

```
f,(ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(12,6))
ax1.scatter(df['mean area'],df['cancer'])
ax1.set_title("cancer cases as a function of mean area",
              fontsize=15)
ax2.scatter(df['mean smoothness'],df['cancer'])
ax2.set_title("cancer cases a smoothnes",fontsize=15)
```

OUTPUT:

```
Text(0.5, 1.0, 'cancer cases a smoothnes')
```




```
df_feat = df.drop('cancer',axis=1)#define a dataframe with the feature
df_feat.head()
```

OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30001	0.14710	0.24190	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.18120	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.20690	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.25970	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
					50			20		44									75		00
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.18089	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.076678

5 rows × 30 columns

```
df_target = df['cancer']
df_target.head()
```

OUTPUT:

```
0 0 1 0 2 0 3 0 4 0 Name: cancer, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(df_feat, df_target,
                                                    test_size=0.30,
                                                    random_state=101)
```

```
y_train.head()
```

OUTPUT:

```
178 1 421 1 57 0 514 0 548 1 Name: cancer, dtype: int64
```

```
from sklearn.svm import SVC
```

```
model = SVC()
```

```
model.fit(X_train,y_train)
```

SVC

```
SVC()
```

predictions and evaluations

```
predictions = model.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,predictions))
```

OUTPUT:

```
[[ 56  10]
 [  3 102]]
```

```
print(classification_report(y_test,predictions))
```

OUTPUT:

```
precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
           1       0.91      0.97      0.94       105

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

```
param_grid = {'C': [0.1,1,10,100,1000],
              'gamma':[1,0.1,0.01,0.001,0.0001],'kernel':['rbf']}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=1)
```

```
grid.fit(X_train,y_train)
```

OUTPUT:

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

GridSearchCV
estimator: SVC
SVC

```
grid.best_params_
```

```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
grid.best_estimator_
```

OUTPUT:

SVC

```
SVC(C=1, gamma=0.0001)
```

```
grid_predictions = grid.predict(X_test)
```

```
print(confusion_matrix(y_test,grid_predictions))
```

OUTPUT:

```
[[ 59   7]
 [  4 101]]
```

```
print(classification_report(y_test,grid_predictions))
```

OUTPUT:

```
precision    recall  f1-score   support


           0       0.94      0.89      0.91        66
           1       0.94      0.96      0.95       105

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```