Vision

To produce globally competitive computer engineers, who are prepared to accept the challenges at professional level, while maintaining the core values.

Mission

- ✓ To create excellent teaching learning environment.
- ✓ To mould engineers with a strong foundation of scientific knowledge and engineering concepts. ✓ To enhance the acquired concepts and develop new technology through excellence in research.
- ✓ To assist nation building and elevating the quality of life of the people through leadership in professionalism, education, research and public services.

Programme Educational Objectives (PEO)

- ✓ To educate young aspirants with the fundamentals of engineering and knowledge of latest technologies.
- ✓ To encourage the students to remain updated by pursuing higher degree or certification programs.
- ✓ To assume management and leadership roles to contribute in socio-economic development of the nation.

G.H. Patel of College of Engineering and Technology
Department of Computer Engineering

A.Y. 2021-22(ODD), SEMESTER 7

SUBJECT CODE: 3170724

SUBJECT NAME: MACHINE LEARNING

INDEX

NAME: _____

ENROLMENT NO: _____ **BRANCH:** _____

Sr. No	Name of the Experiment	Page No.	Date	Marks	Signature
1	**Lab 1: File Handling Using Python. (Text file, csv file)** Implement simple Program to read and write content of file from text file and CSV file. Write a program to find frequency of each word in positive class and negative class from IMDB data set.				
2,3	**Lab 2: Data Analysis** Do the following Analysis for Automobile data • Get the data types of the data set • the shape of the dataset • Check for duplicate data if present remove duplicate record • Cleaning of the data : ▪ Find out if there are null fields ▪ **Missing Data** ▪ fill missing data of *normalised-losses, price, horsepower, peak-rpm, bore, stroke* with the respective column mean ▪ Fill missing data category Number of doors with the mode of the column i.e. Four • **Summary statistics of**				

	variable: Mean, median, mode, standad deviation, min, max , 25%, 75% • **Univariate Analysis:** ▪ **plot histrgram of curb-weight, enginesize, hoursepower,peak-rpm,price and write your findings from histrograms** ▪ **Plot value count plot for categorical variables:** engine-type, num-of-doors, fuel-type, body-style and write your findings from plots and write your findings • **Bivariate Analysis :Do** Price Analysis and write your findings ▪ **Numerical vs. Numerical** 1. Scatterplot 2. Line plot 3. Heatmap for correlation 4. Joint plot ▪ **Categorical vs. Numerical** ▪ 1. Bar chart 2. Violin plot 3. Categorical box plot 4.Swarm plot ▪ **Two Categorical Variables** ▪ 1. Bar chart 2. Grouped bar chart 3. Point plot • Handling outlier: ▪ Drop the outlier value ▪ Replace the outlier value using the IQR				
4	Implement Principal Component Analysis and also implement face recognition using PCA				
5	Prepare ttraining and testing data using following method 1) Holdout method 2) *K*-fold Cross-validation method 3) Leave-one-out cross-validation 4) Bootstrap sampling				
6	Write a python program to movie review sentiment analysis using Naïve Bayes classifier				
7	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.				
8	Implement face recognition using Support Vector machine				
9	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.				

10	Write a program to implement house price prediction using linear regression (implement using gradient decent)				
11	Write a Program to implement apriori				
12	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Do not use inbuilt function				
13,14	Implement following clustering algorithm 1) Kmeans clustering 2) Hierarchical clustering 3) Density based clustering				

SUBJECT CODE: 3170720

SUBJECT NAME: INFORMATION SECURITY

INDEX

Sr.	List of Assignment(s)	Page No.	Date	Marks	Signature
1	Assignment 1				
2	Assignment 2				
3	Assignment 3				

1. Implement simple program to read and write content of file from text file and csv file.

In [1]:
```
file = open("hello.txt","w")
file.write("Hello! I am Pankil tilvani")
file.close();
```

In [2]:
```
file=open("hello.txt","r")
str=file.readline()
print(str)
file.close()
```

```
Hello! I am Pankil tilvani
```

Write a program to find frequency of each word in positive classes and negative classes from IMDB data.

In [2]:
```
data={"Name:":['jack','Jill','Em','David'],
     "Profession":['Manager','Analyst','Data scientist','VP'],
     "Ecperience":[10,5,2,20],
     "Education":['MBA','BS','MS','MBA']}
data
```

Out[2]:
```
{'Name:': ['jack', 'Jill', 'Em', 'David'],
 'Profession': ['Manager', 'Analyst', 'Data scientist', 'VP'],
 'Ecperience': [10, 5, 2, 20],
 'Education': ['MBA', 'BS', 'MS', 'MBA']}
```

In [5]:
```
import pandas as pd
df=pd.DataFrame(data)
df.to_csv("1st.csv")
data=pd.read_csv("1st.csv")
data
```

Out[5]:

	Unnamed: 0	Name:	Profession	Ecperience	Education
0	0	jack	Manager	10	MBA
1	1	Jill	Analyst	5	BS
2	2	Em	Data scientist	2	MS
3	3	David	VP	20	MBA

In [7]:
```
import pandas as pd
imdb =pd.read_csv("IMDB Dataset.csv")
imdb
```

Out[7]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

In [8]:
```
imdb['sentiment'].value_counts()
```

Out[8]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

2.Analysis for Automobile Dataset

In [9]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as  plt
lol=pd.read_csv("Automobile_data.csv")
lol
```

Out[9]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepow
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	1
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0	1
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	1
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	1
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	1
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.4	23.0	1
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	1

205 rows × 26 columns

In [14]:
```
data=lol.replace('?',np.NAN)
data.isnull().sum()
```

Out[14]:
```
symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

In [10]:
```
lol.dtypes
```

Out[10]:
```
symboling             int64
normalized-losses    object
make                 object
fuel-type            object
aspiration           object
num-of-doors         object
body-style           object
drive-wheels         object
engine-location      object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight           int64
engine-type          object
num-of-cylinders     object
engine-size           int64
fuel-system          object
bore                 object
stroke               object
compression-ratio   float64
horsepower           object
peak-rpm             object
city-mpg              int64
highway-mpg           int64
price                object
dtype: object
```

In [11]:
```
lol.shape
```

Out[11]:
```
(205, 26)
```

In [18]:
```
temp=lol[lol['normalized-losses']!='?']
normalized_mean=temp['normalized-losses'].astype(int).mean()
lol['normalized-losses'] = lol['normalized-losses'].replace('?',normalized_mean).astype(int)

temp=lol[lol['price']!='?']
normalized_mean=temp['price'].astype(int).mean()
lol['price']=lol['price'].replace('?',normalized_mean).astype(int)

temp=lol[lol['horsepower']!='?']
normalized_mean=temp['horsepower'].astype(int).mean()
lol['horsepower']=lol['horsepower'].replace('?',normalized_mean).astype(int)

temp=lol[lol['peak-rpm']!='?']
normalized_mean=temp['peak-rpm'].astype(int).mean()
lol['peak-rpm']=lol['peak-rpm'].replace('?',normalized_mean).astype(int)

temp=lol[lol['bore']!='?']
normalized_mean=temp['bore'].astype(float).mean()
lol['bore']=lol['bore'].replace('?',normalized_mean).astype(float)

temp=lol[lol['stroke']!='?']
normalized_mean=temp['stroke'].astype(float).mean()
lol['stroke']=lol['stroke'].replace('?',normalized_mean).astype(float)
lol.head()
```

Out[18]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	122	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102
4	2	122	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115

5 rows × 26 columns

In [19]:
```
lol['num-of-doors']=lol['num-of-doors'].replace('?','four')
lol.head()
```

Out[19]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	122	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102
4	2	122	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115

5 rows × 26 columns

In [20]:
```
lol.describe()
```

Out[20]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower
count	205.000000	205.0	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	122.0	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329751	3.255423	10.142537	104.253659
std	1.245307	0.0	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.519219
min	-2.000000	122.0	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000
25%	0.000000	122.0	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000
50%	1.000000	122.0	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000
75%	2.000000	122.0	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000
max	3.000000	122.0	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000

In [19]:
```
lol['num-of-doors']=lol['num-of-doors'].replace('?','four')
lol.head()
```

Out[19]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	122	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102
4	2	122	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115

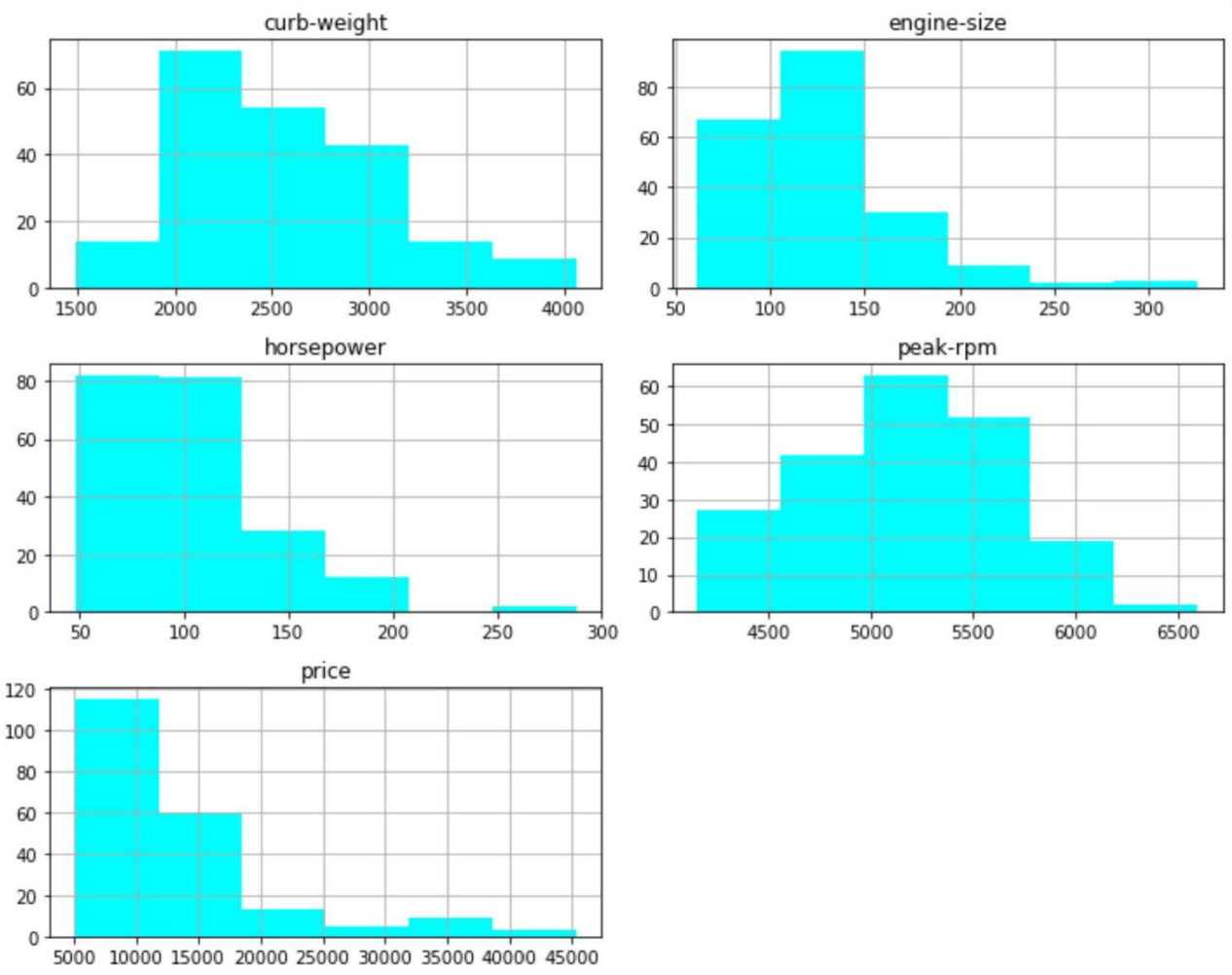
5 rows × 26 columns

In [20]:
```
lol.describe()
```

Out[20]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower
count	205.000000	205.0	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	122.0	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329751	3.255423	10.142537	104.253659
std	1.245307	0.0	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.519219
min	-2.000000	122.0	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000
25%	0.000000	122.0	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000
50%	1.000000	122.0	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000
75%	2.000000	122.0	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000
max	3.000000	122.0	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000

```
# Univariate Analysis
auto[['curb-weight','engine-size','horsepower','peak-rpm','price']].hist(figsize=(10,8),bin
plt.tight_layout()
plt.show()
```

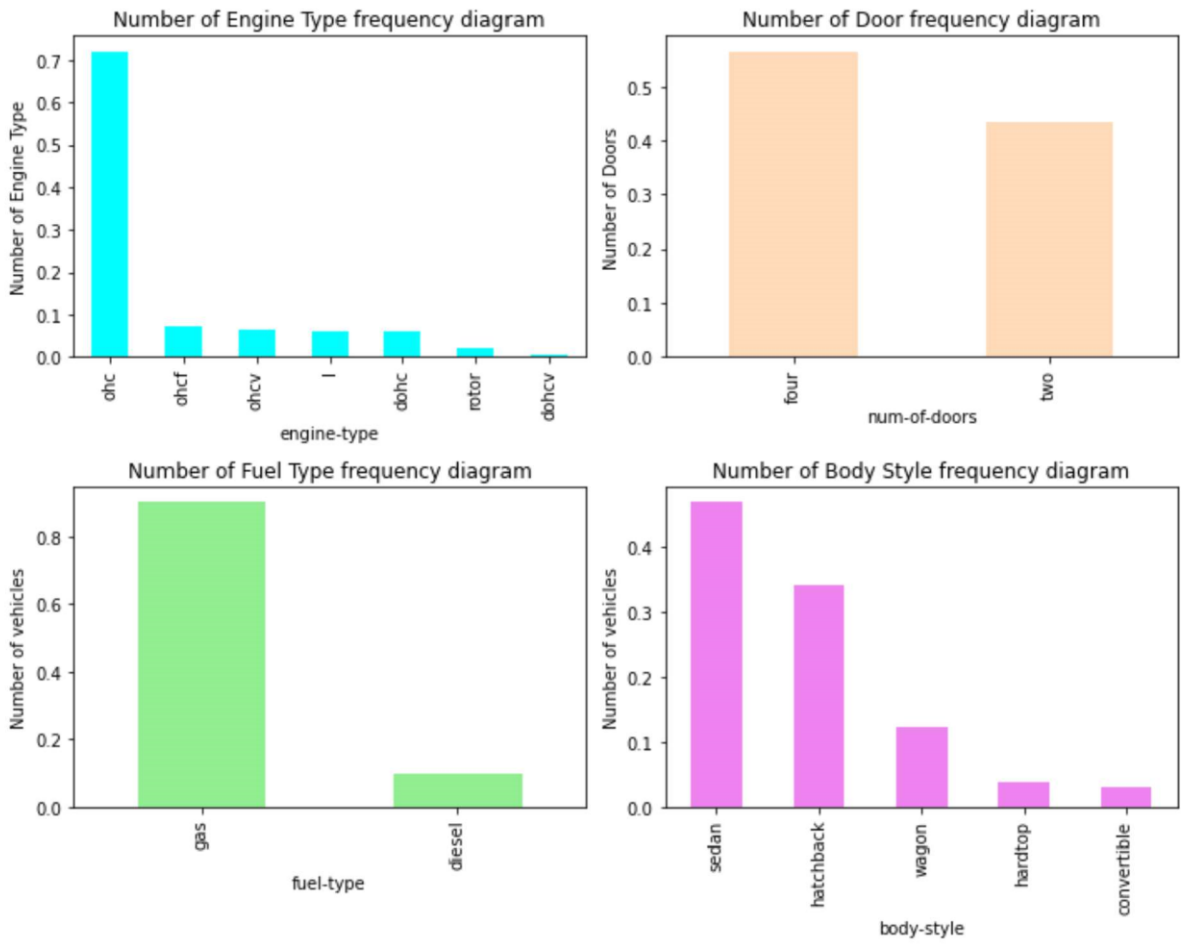


Findings

- Most of the car has a Curb Weight is in range 1900 to 3100
- The Engine Size is inrange 60 to 190
- Most vehicle has horsepower 50 to 125
- Most Vehicle are in price range 5000 to 18000
- Peak rpm is mostly distributed between 4600 to 5700

3

```
#Value count of respective columns
plt.figure(1)
plt.subplot(221)
auto['engine-type'].value_counts(normalize=True).plot(figsize=(10,8),kind='bar',color='cyan
plt.title("Number of Engine Type frequency diagram")
plt.ylabel('Number of Engine Type')
plt.xlabel('engine-type');

plt.subplot(222)
auto['num-of-doors'].value_counts(normalize=True).plot(figsize=(10,8),kind='bar',color='pea
plt.title("Number of Door frequency diagram")
plt.ylabel('Number of Doors')
plt.xlabel('num-of-doors');

plt.subplot(223)
auto['fuel-type'].value_counts(normalize= True).plot(figsize=(10,8),kind='bar',color='light
plt.title("Number of Fuel Type frequency diagram")
plt.ylabel('Number of vehicles')
plt.xlabel('fuel-type');

plt.subplot(224)
auto['body-style'].value_counts(normalize=True).plot(figsize=(10,8),kind='bar',color='viole
plt.title("Number of Body Style frequency diagram")
plt.ylabel('Number of vehicles')
plt.xlabel('body-style');
plt.tight_layout()
plt.show()
```

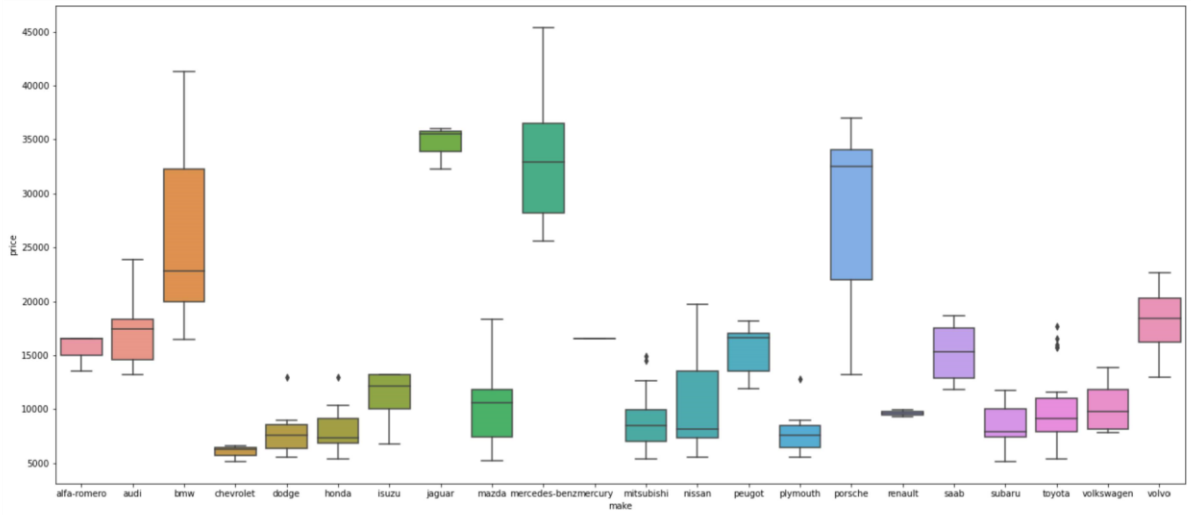


Findings

- More than 70 % of the vehicle has Ohc type of Engine
- 57% of the cars has 4 doors
- Gas is preferred by 85 % of the vehicles
- Most produced vehicle are of body style sedan around 48% followed by hatchback 32%

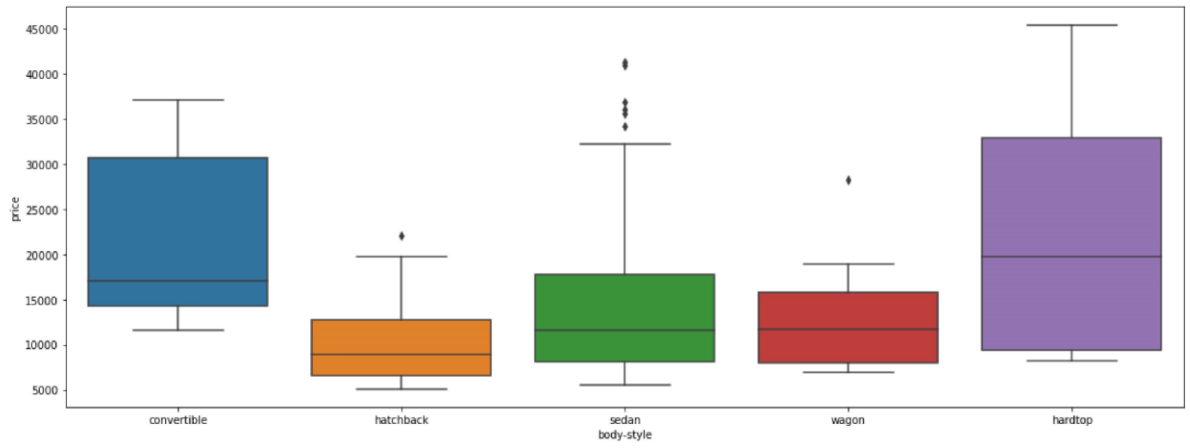
In [64]:

```
#Bivariate analysis (Price Analysis)
plt.rcParams['figure.figsize']=(23,10)
ax = sns.boxplot(x="make", y="price", data=auto)
```



In [65]:

```
plt.rcParams['figure.figsize']=(19,7)
ax = sns.boxplot(x="body-style", y="price", data=auto)
```

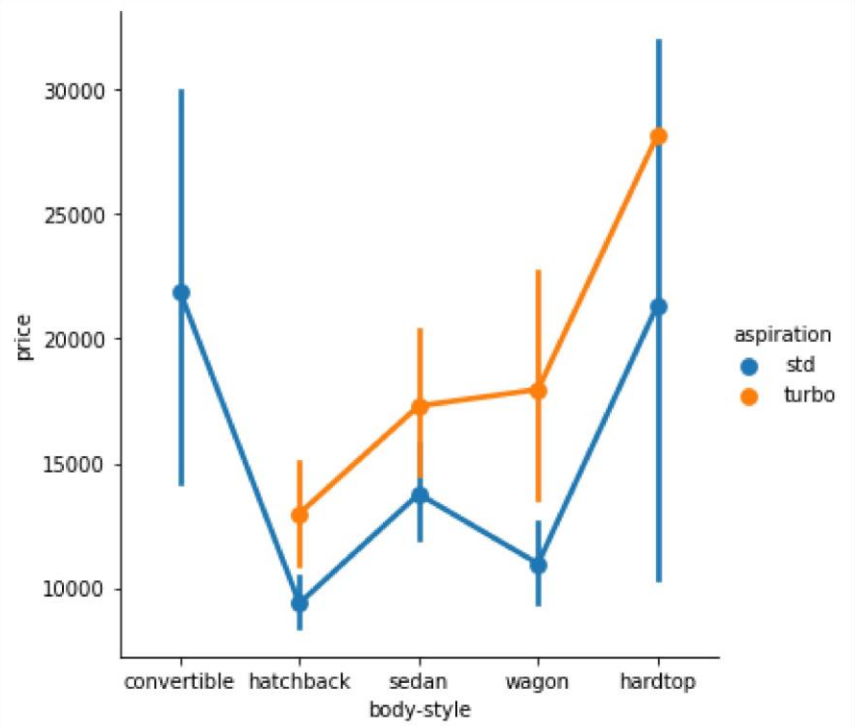


6

```
sns.catplot(data=auto, x="body-style", y="price", hue="aspiration" ,kind="point")
```

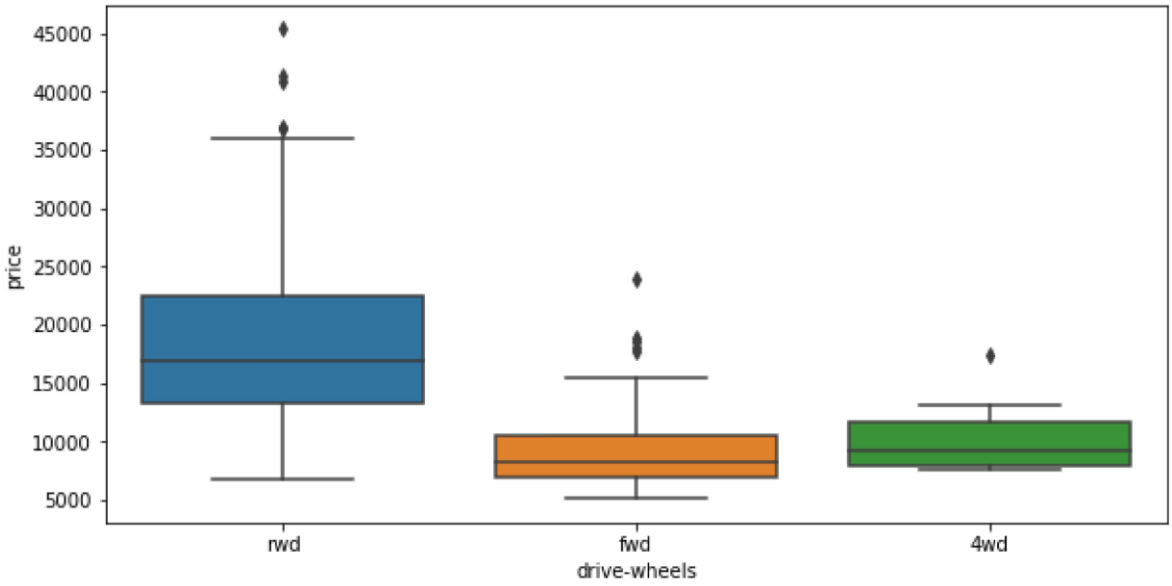
Out[66]:

```
<seaborn.axisgrid.FacetGrid at 0x2ab935d3dc0>
```



In [67]:

```
plt.rcParams['figure.figsize']=(10,5)
ax = sns.boxplot(x="drive-wheels", y="price", data=auto)
```



Findings

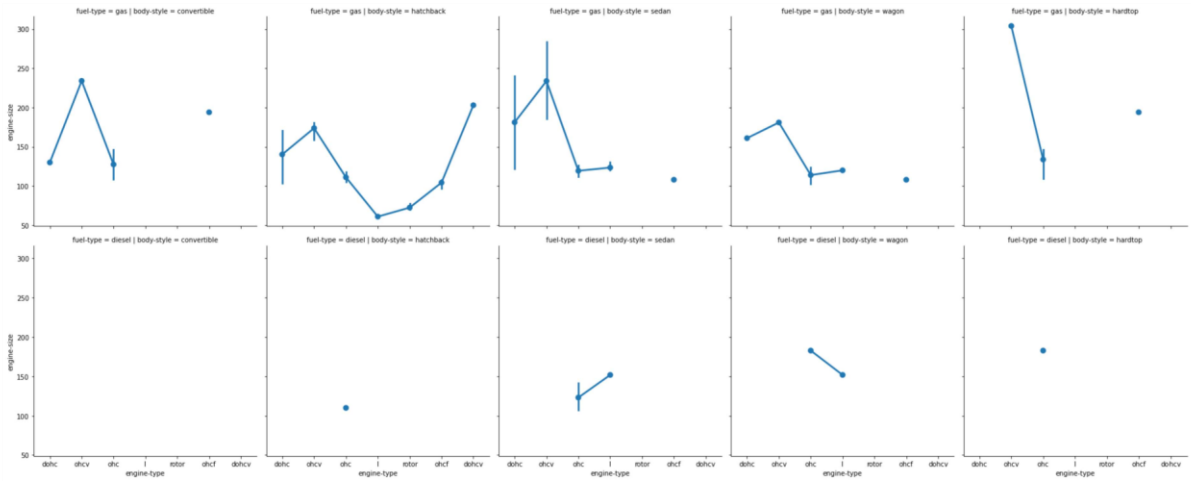
- Mercedez-Benz ,BMW, Jaguar, Porshe produce expensive cars more than 25000
- Chevrolet, Dodge, Honda, Mitsubishi, Nissan, Plymouth, Subaru, Toyata produce budget models with lower prices
- Most of the cars companies produce cars in range below 25000
- Hardtop model are expensive in prices followed by convertible and sedan body style
- Turbo models have higher prices than for the standard model
- Convertible has only standard edition with expensive cars
- Hatchback and Sedan turbo models are available below 20000
- RWD wheel drive vehicles have expensive prices

In [68]:

```
import warnings
warnings.filterwarnings("ignore")
sns.factorplot(data=auto, x="engine-type", y="engine-size", col="body-style",row="fuel-type
```

Out[68]:

```
<seaborn.axisgrid.FacetGrid at 0x2ab8d58c730>
```

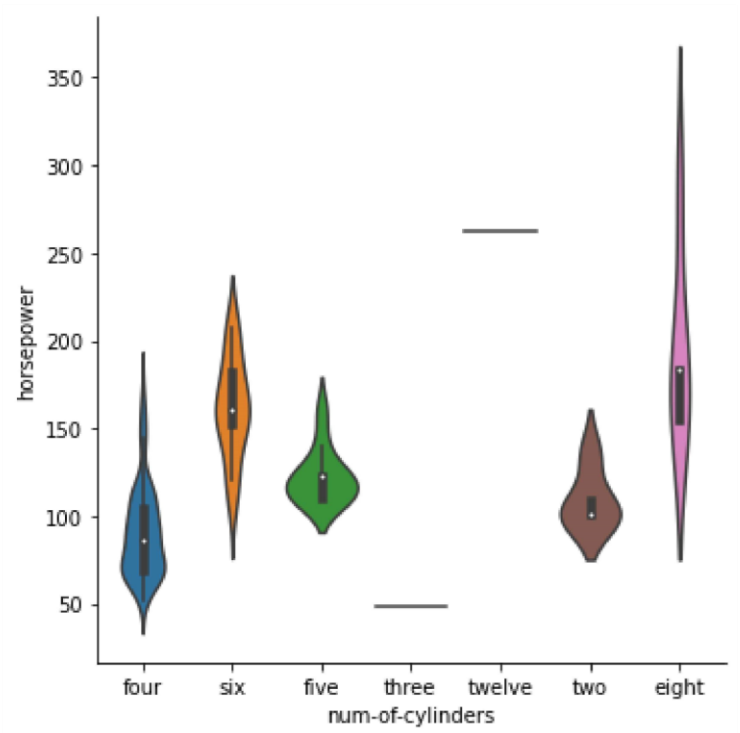


9

```
sns.catplot(data=auto, x="num-of-cylinders", y="horsepower",kind="violin")
```

Out[69]:

```
<seaborn.axisgrid.FacetGrid at 0x2ab92f964c0>
```



Findings

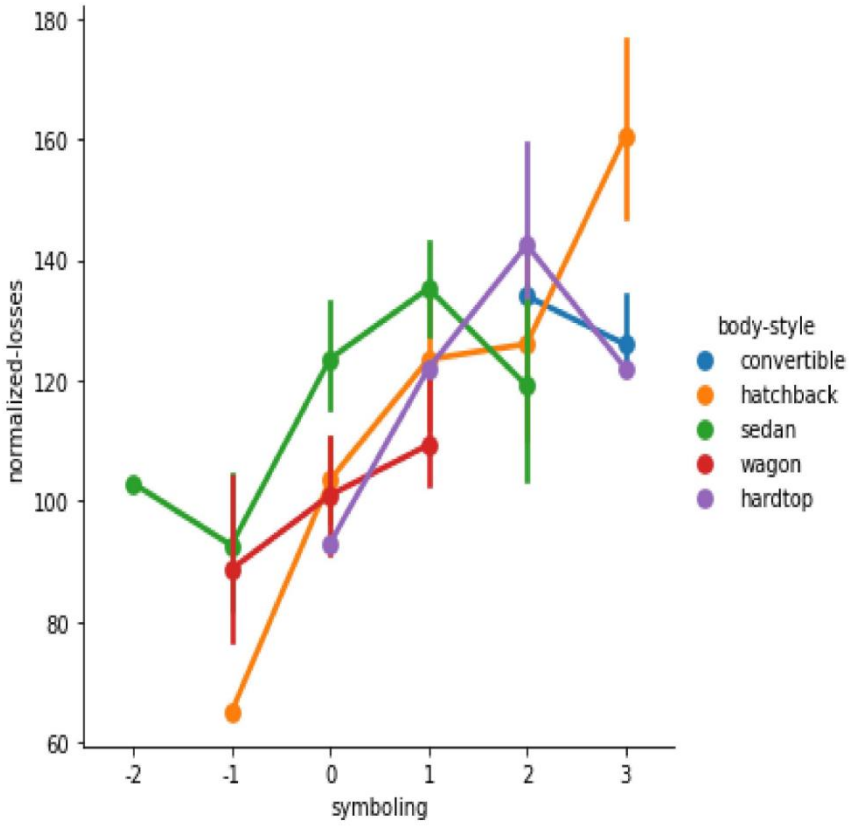
- OHC is the most used Engine Type both for diesel and gas
- Diesel vehicles have Engine type "ohc" and "l" and engine size ranges between 100 to 190
- Engine type OHCV has the bigger Engine size ranging from 155 to 300
- Body-style Hatchback uses max variety of Engine Type followed by sedan
- Body-style Convertible is not available with Diesel Engine type
- Vehicles with above 200 horsepower have Eight Twelve Six cyclinders

```
n  70
```

```
sns.catplot(data=auto, y="normalized-losses", x="symboling" , hue="body-style" ,kind="point
```

Out[70]:

```
<seaborn.axisgrid.FacetGrid at 0x2ab8aa05dc0>
```

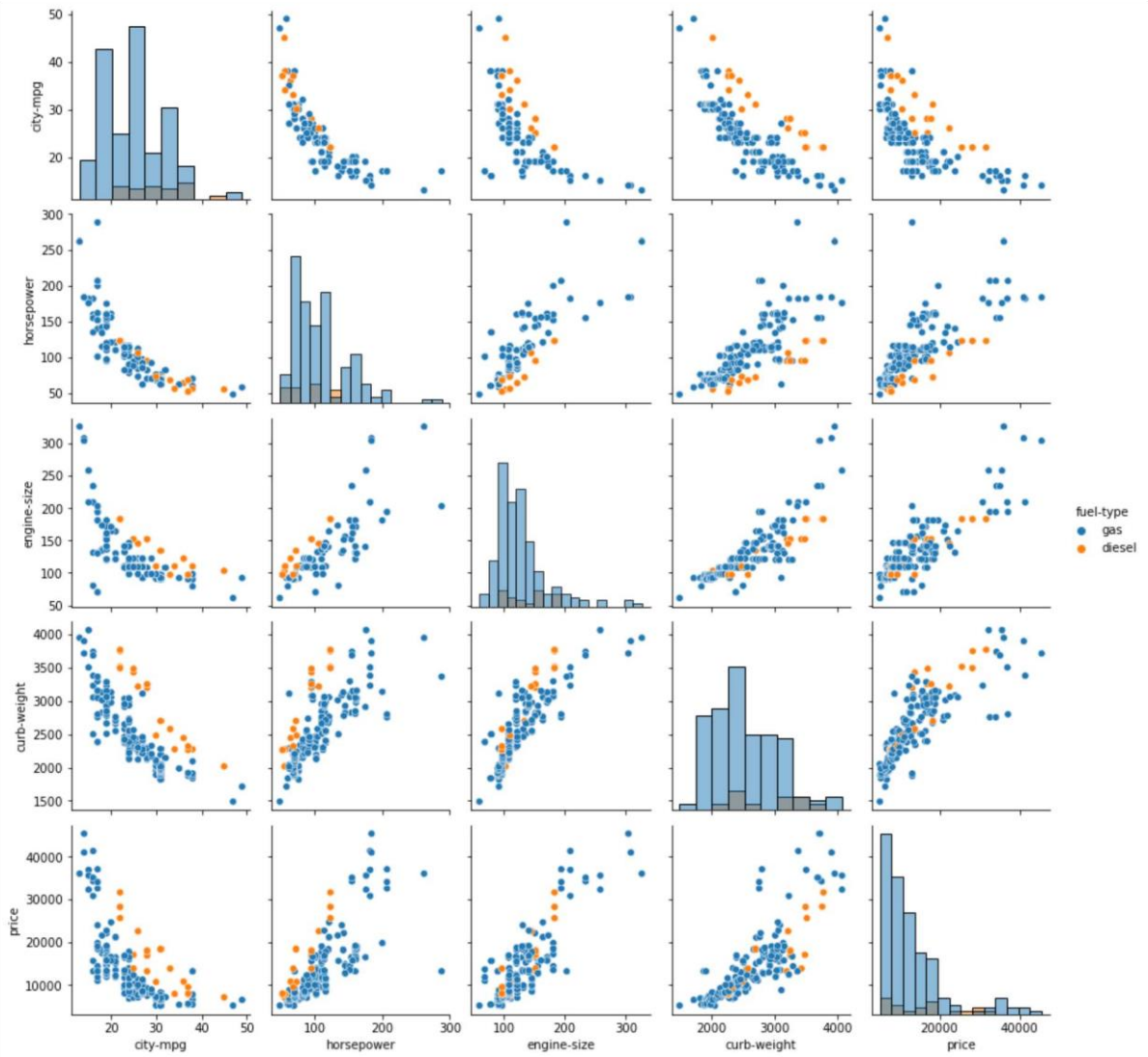


Losses Findings

- Increased in risk rating linearly increases in normalised losses in vehicle
- Convertible cars and Hardtop cars have mostly losses with risk rating above 0
- Hatchback cars have highest losses at risk rating 3
- Sedan and Wagon cars have losses even in less risk(safe) rating

71

```
g = sns.pairplot(auto[["city-mpg", "horsepower", "engine-size", "curb-weight","price", "fue
```



Findings

- Vehicle Mileage decreases with increase in Horsepower, Engine-size, Curb Weight
- As horsepower increases the engine size increases
- Curb Weight increases with the increase in Engine Size

Price Analysis

- Engine size and Curb-Weight are positively corelated with Price
- City-mpg is negatively corelated with price as increase in horsepower reduces the mileage

Implement Principal Component Analysis and Face Recognition using PCA

In [1]:

```
# Import matplotlib library
import matplotlib.pyplot as plt

# Import scikit-learn library
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC

import numpy as np
```

In [2]:

```
lfw_people = fetch_lfw_people(min_faces_per_person = 70, resize = 0.4)
n_samples, h, w = lfw_people.images.shape
X = lfw_people.data
n_features = X.shape[1]
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
print("Number of Data Samples: % d" % n_samples)
print("Size of a data sample: % d" % n_features)
print("Number of Class Labels: % d" % n_classes)
```

```
Number of Data Samples:  1288
Size of a data sample:  1850
Number of Class Labels:  7
```

In [4]:

```
def plot_gallery(images, titles, h, w, n_row = 3, n_col = 4):
    plt.figure(figsize =(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom = 0, left =.01, right =.99, top =.90, hspace =.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap = plt.cm.gray)
        plt.title(titles[i], size = 12)
        plt.xticks(())
        plt.yticks(())

def true_title(Y, target_names, i):
    true_name = target_names[Y[i]].rsplit(' ', 1)[-1]
    return 'true label: % s' % (true_name)

true_titles = [true_title(y, target_names, i)
for i in range(y.shape[0])]
plot_gallery(X, true_titles, h, w)
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
print("size of training Data is % d and Testing Data is % d" %(y_train.shape[0], y_test.sha
```

```
size of training Data is  966 and Testing Data is  322
```

In [10]:

```
from time import time

n_components = 150
t0 = time()
pca = PCA(n_components = n_components, svd_solver ='randomized', whiten = True).fit(X_train
print("done in % 0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in % 0.3fs" % (time() - t0))
```

```
done in  0.113s
Projecting the input data on the eigenfaces orthonormal basis
done in  0.016s
```

In [12]:

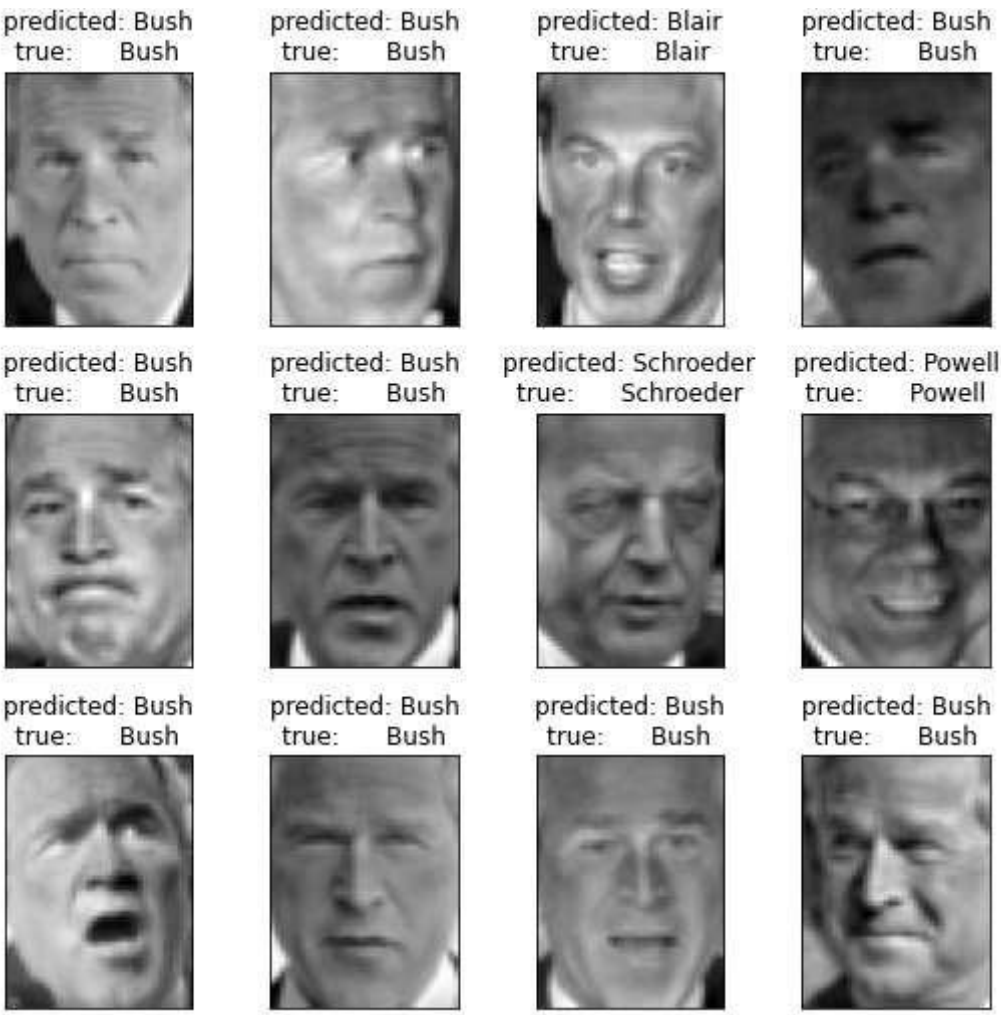
```
print("Sample Data point after applying PCA\n", X_train_pca[0])
print("Dimensions of training set = % s and Test Set = % s"%(X_train.shape, X_test.shape))
```

```
Sample Data point after applying PCA
 [-2.0756032  -1.0457916   2.1269364   0.03682139 -0.757567   -0.51736623
  0.8555065   1.0519369   0.45773777  0.01347729 -0.03963248  0.6387263
  0.48166943  2.3378541   1.7784541   0.13308632 -2.2713253  -4.4569674
  2.097628   -1.1379254   0.18843494 -0.3349755   1.1254572  -0.32402483
  0.14093065  1.0769353   0.7588309  -0.09976928  3.1199572   0.8837761
 -0.89338034  1.159604    1.4306276   1.6856381   1.3434589  -1.2591333
 -0.6391263  -2.3362787  -0.01371578 -1.4638852  -0.46884668 -1.0547469
 -1.3328594   1.1363696   2.2223487  -1.8015273  -0.30623782 -1.028425
  4.7736416   3.4598582   1.9259627  -1.3512428  -0.25882423  2.0104215
 -1.0565367   0.3608064   1.1711143   0.7575033   0.90065414  0.59932965
 -0.4653361   2.0981355   1.3455778   1.9331867   5.066952   -0.70533204
  0.60640717 -0.89904875 -0.21429199 -2.1076744  -1.6812096  -0.19653304
 -1.7456497  -3.055236    2.0528047   0.39445785  0.12771842  1.2109721
 -0.7921295  -1.3899782  -2.0339344  -2.792643    1.4827937   0.21270312
  0.25381833 -0.11067396  1.1729906   0.8053914   1.2692982   0.08991409
 -0.97994703  0.32077459  1.0446659   0.8445741   0.6040515  -0.51763964
 -1.328008   -1.0426265  -0.46996164  1.0556353   1.210372   -1.2927651
 -1.0693852  -1.9511477   0.727909    1.9512117  -1.444685    0.6527036
  1.4632903  -1.7049578   1.8737769  -0.63071793  1.1679231  -1.5509627
  0.1299906   2.212018    1.7003239   0.9123466  -1.1271588  -0.01025486
  0.04143495  2.003224    1.1336865  -0.6641279   0.27638003  0.7142704
  0.96128076 -0.9137842  -0.62422854  0.23606376  0.26360866 -1.7313025
 -0.73771745  0.7524248   0.80605334 -0.13806811 -1.4943777  -0.77290356
  0.0508016   0.15344012 -0.7018406   0.9447583  -2.0078738   0.2364678
 -0.14391083 -0.76907796  1.0371499  -0.4650051   2.2233624  -1.4806844 ]
Dimensions of training set = (966, 1850) and Test Set = (322, 1850)
```

In [13]:

```
print("Fitting the classifier to the training set")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5], 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01,
clf = GridSearchCV(SVC(kernel ='rbf', class_weight ='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)
print("done in % 0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)
print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in % 0.3fs" % (time() - t0))
print(classification_report(y_test, y_pred, target_names = target_names))
print("Confusion Matrix is:")
print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))
```

```
Fitting the classifier to the training set
done in  17.469s
Best estimator found by grid search:
SVC(C=1000.0, class_weight='balanced', gamma=0.005)
Predicting people's names on the test set
done in  0.062s
                   precision    recall  f1-score   support

     Ariel Sharon       0.78      0.54      0.64        13
     Colin Powell       0.82      0.88      0.85        60
  Donald Rumsfeld       0.85      0.63      0.72        27
    George W Bush       0.85      0.98      0.91       146
Gerhard Schroeder       0.95      0.80      0.87        25
      Hugo Chavez       1.00      0.60      0.75        15
       Tony Blair       0.97      0.78      0.86        36

         accuracy                           0.86       322
        macro avg       0.89      0.74      0.80       322
     weighted avg       0.87      0.86      0.85       322

Confusion Matrix is:
[[  7   1   0   5   0   0   0]
 [  1  53   1   5   0   0   0]
 [  1   3  17   6   0   0   0]
 [  0   3   0 143   0   0   0]
 [  0   1   0   3  20   0   1]
 [  0   3   0   2   1   9   0]
 [  0   1   2   5   0   0  28]]
```


predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Blair
true: Blair
predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Schroeder
true: Schroeder
predicted: Powell
true: Powell
predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Bush
true: Bush

Prepare training and testing data using:

In [49]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
from sklearn.utils import resample
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

Holdout method

In [39]:

```
foods = pd.read_csv("FAO database.csv", encoding='latin1')
pd.set_option('max_column', None)
usa_foods = foods.query('Area == "United States of America"')
usa_foods.head(3)
```

Out[39]:

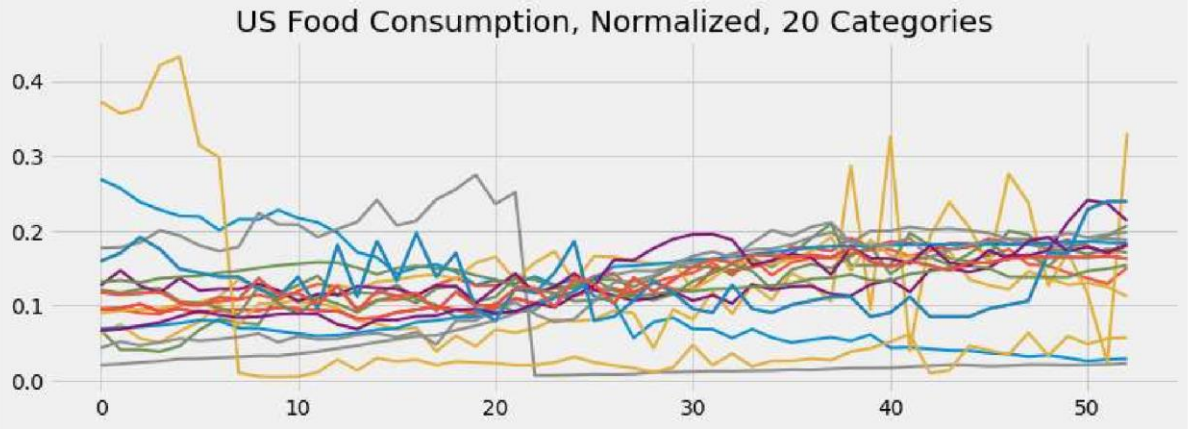
	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	long
20394	US	231	United States of America	2511	Wheat and products	5521	Feed	1000 tonnes	37.09	-
20395	US	231	United States of America	2511	Wheat and products	5142	Food	1000 tonnes	37.09	-
20396	US	231	United States of America	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	37.09	-

```
0
```

```
X = usa_foods.loc[:, ['Y{0}'.format(y) for y in range(1961, 2014)]].replace(0, np.nan)
X = X.dropna(thresh=30)
X = X.fillna(method='backfill')
X = normalize(X)

X = X[(np.max(X, axis=1) - np.min(X, axis=1)) / np.min(X, axis=1) < 10**2, :]

plt.style.use('fivethirtyeight')

fig = plt.figure(figsize=(12, 4))
for n in np.random.choice(range(X.shape[1]), size=20):
    plt.plot(X[n, :],linewidth=2)

plt.title('US Food Consumption, Normalized, 20 Categories')
```

Out[40]:

```
Text(0.5, 1.0, 'US Food Consumption, Normalized, 20 Categories')
```



In [41]:

```
X, y = X[:, :-1], X[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

In [42]:

```
clf = LinearRegression()
clf.fit(X_train, y_train)
y_test_hat = clf.predict(X_test)

from sklearn.metrics import median_absolute_error
median_absolute_error(y_test_hat, y_test)
```

Out[42]:

```
0.023716083349389613
```

```
3
```

```
median_absolute_error(clf.predict(X_train), y_train)
```

Out[43]:

```
0.00221303501937703
```

K-fold Cross-validation method

In [44]:

```
kf = KFold(n_splits=6)

scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    scores.append(
        median_absolute_error(clf.fit(X_train, y_train).predict(X_test), y_test)
    )

np.mean(scores)
```

Out[44]:

```
0.017993801387054878
```

Leave-one-out cross-validation

In [45]:

```
loo = LeaveOneOut()

scores = []
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    scores.append(
        median_absolute_error(clf.fit(X_train, y_train).predict(X_test), y_test)
    )

scores = np.array(scores)
np.mean(scores)
```

Out[45]:

```
0.02618378635961042
```

Bootstrap sampling

6

```
data = pd.read_csv('pima-indians-diabetes.csv')
data.head()
```

Out[46]:

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

In [47]:

```
n_iterations = 10
n_size = int(len(data) * 0.50)
```

In [50]:

```
#Bootstrap-ing
stats = list()
for i in range(n_iterations):
    train = resample(values, n_samples = n_size)
    test = np.array([x for x in values if x.tolist() not in train.tolist()])

    model = DecisionTreeClassifier()
    model.fit(train[:,:-1], train[:,-1])

    predictions = model.predict(test[:,:-1])
    score = accuracy_score(test[:,-1], predictions)

    print(score)
    stats.append(score)
```

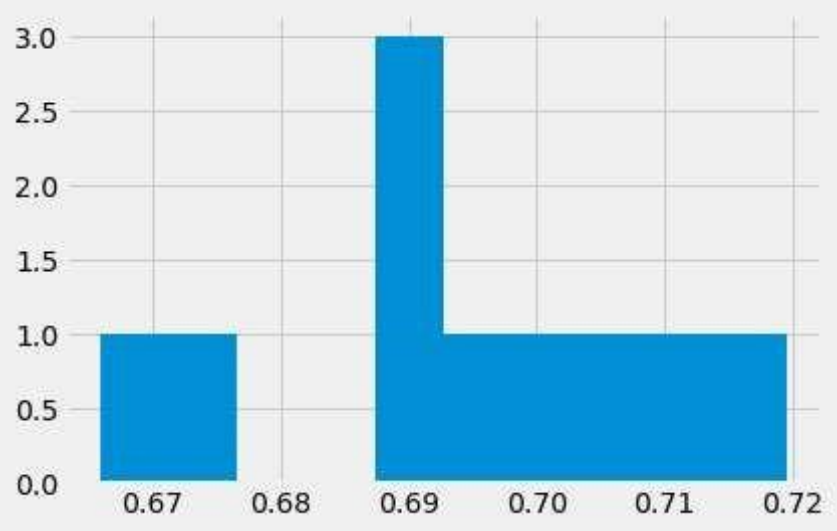
```
0.665938864628821
0.67170626349892
0.7194860813704497
0.6888412017167382
0.7098901098901099
0.7048458149779736
0.697228144989339
0.6911447084233261
0.6984815618221258
0.6895074946466809
```

In [51]:

```
plt.hist(stats)
plt.figure(figsize = (10,5))
```

Out[51]:

```
<Figure size 720x360 with 0 Axes>
```




```
<Figure size 720x360 with 0 Axes>
```

In [52]:

```
#Confidence Intervals
a = 0.95
p = ((1.0 - a)/2.0) * 100
lower = max(0.0, np.percentile(stats,p))

p = (a + ((1.0 - a)/ 2.0)) * 100
upper = min(1.0, np.percentile(stats,p))
print('%.1f confidence interval %.1f%% and %.1f%%' %(a*100, lower*100, upper*100))
```

```
95.0 confidence interval 66.7% and 71.7%
```

Movie review sentiment analysis using Naïve Bayes classifier

In [4]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import string
import re
import matplotlib.pyplot as plt
import math
from matplotlib import rc
from sklearn.model_selection import train_test_split
from collections import Counter, defaultdict
from bs4 import BeautifulSoup
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import nltk
from nltk.corpus import stopwords
%matplotlib inline

sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\rajba\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[4]:

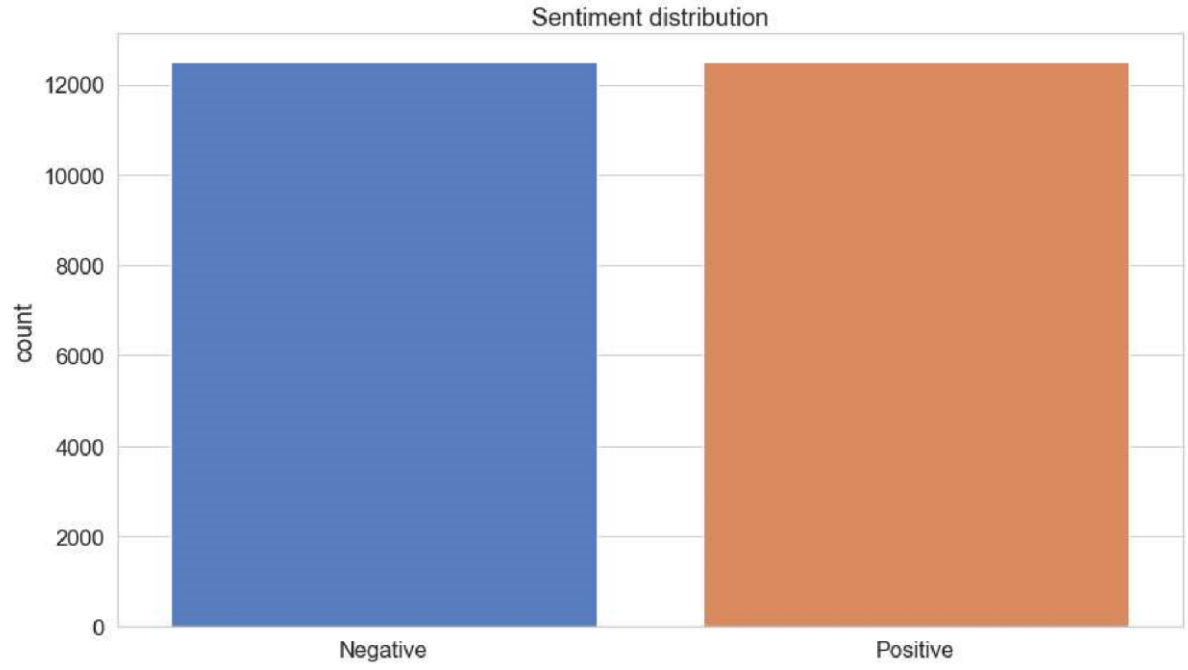
```
True
```

In [5]:

```
train = pd.read_csv("imdb_review_train.tsv", delimiter="\t")
test = pd.read_csv("imdb_review_test.tsv", delimiter="\t")
```

In [6]:

```
f = sns.countplot(x='sentiment', data=train)
f.set_title("Sentiment distribution")
f.set_xticklabels(['Negative', 'Positive'])
plt.xlabel("");
```



8]:

```
class Tokenizer:

    def clean(self, text):
        no_html = BeautifulSoup(text).get_text()
        clean = re.sub("[^a-z\s]+", " ", no_html, flags=re.IGNORECASE)
        return re.sub("(\s+)", " ", clean)


    def tokenize(self, text):
        clean = self.clean(text).lower()
        stopwords_en = stopwords.words("english")
        return [w for w in re.split("\W+", clean) if not w in stopwords_en]

class MultinomialNaiveBayes:

    def __init__(self, classes, tokenizer):
        self.tokenizer = tokenizer
        self.classes = classes

    def group_by_class(self, X, y):
        data = dict()
        for c in self.classes:
            data[c] = X[np.where(y == c)]
        return data

    def fit(self, X, y):
        self.n_class_items = {}
        self.log_class_priors = {}
        self.word_counts = {}
        self.vocab = set()

        n = len(X)
        grouped_data = self.group_by_class(X, y)
        for c, data in grouped_data.items():
            self.n_class_items[c] = len(data)
            self.log_class_priors[c] = math.log(self.n_class_items[c] / n)
            self.word_counts[c] = defaultdict(lambda: 0)

            for text in data:
                counts = Counter(self.tokenizer.tokenize(text))
                for word, count in counts.items():
                    if word not in self.vocab:
                        self.vocab.add(word)

                self.word_counts[c][word] += count

        return self

    def laplace_smoothing(self, word, text_class):
        num = self.word_counts[text_class][word] + 1
        denom = self.n_class_items[text_class] + len(self.vocab)
        return math.log(num / denom)

    def predict(self, X):
        result = []
        for text in X:
            class_scores = {c: self.log_class_priors[c] for c in self.classes}
            words = set(self.tokenizer.tokenize(text))
```

```
            for word in words:
                if word not in self.vocab: continue
                for c in self.classes:
                    log_w_given_c = self.laplace_smoothing(word, c)
                    class_scores[c] += log_w_given_c

            result.append(max(class_scores, key=class_scores.get))

        return result
```

In [9]:

```
X = train['review'].values
y = train['sentiment'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=RANDO
```

In [11]:

```
MNB = MultinomialNaiveBayes(classes=np.unique(y), tokenizer=Tokenizer()).fit(X_train, y_tra
```

In [12]:

```
y_hat = MNB.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[12]:

```
0.6342
```

In [13]:

```
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.60      0.79      0.68      2481
           1       0.70      0.48      0.57      2519

    accuracy                           0.63      5000
   macro avg       0.65      0.64      0.63      5000
weighted avg       0.65      0.63      0.63      5000
```

In [14]:

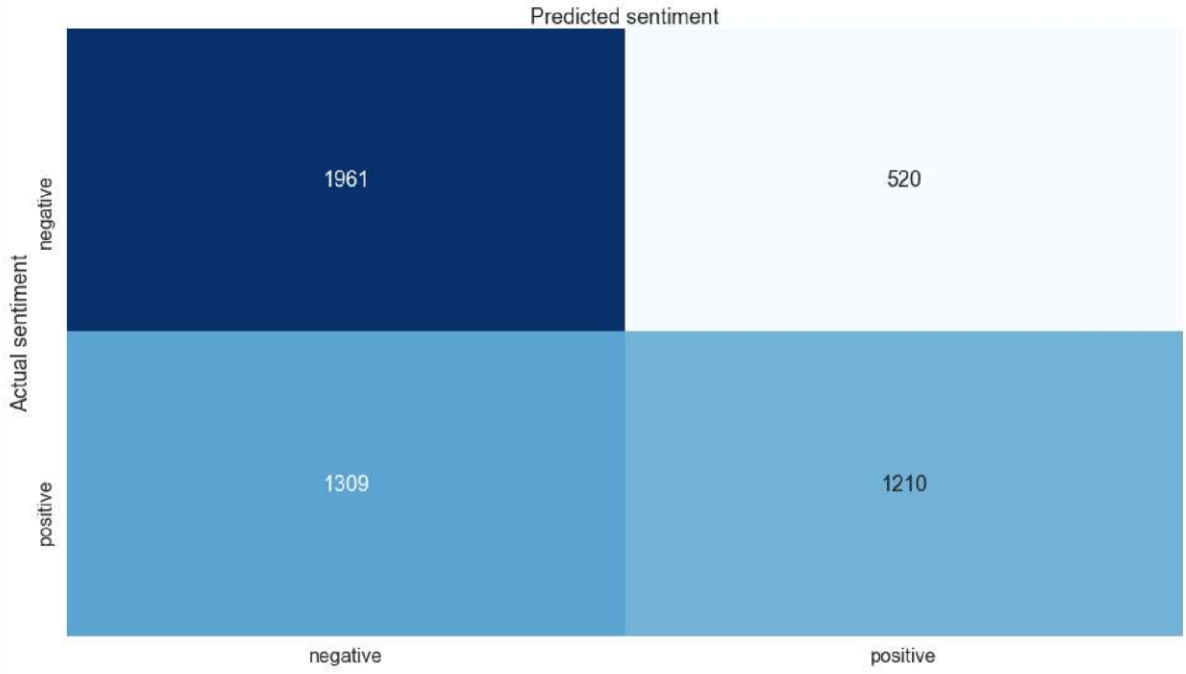
```
cnf_matrix = confusion_matrix(y_test, y_hat)
cnf_matrix
```

Out[14]:

```
array([[1961,  520],
       [1309, 1210]], dtype=int64)
```

5

```
class_names = ["negative", "positive"]
fig,ax = plt.subplots()

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="Blues", fmt="d", cbar=False, xtickl
ax.xaxis.set_label_position('top')
plt.tight_layout()
plt.ylabel('Actual sentiment')
plt.xlabel('Predicted sentiment');
```



Program to demonstrate the working of the decision tree based ID3 algorithm.

Example 1

In [9]:

```
import math
import pandas as pd
from operator import itemgetter
```

In [10]:

```
class DecisionTree:
    def __init__(self, df, target, positive, parent_val, parent):
        self.data = df
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = ''

    def _get_entropy(self, data):
        p = sum(data[self.target]==self.positive)
        n = data.shape[0] - p
        p_ratio = p/(p+n)
        n_ratio = 1 - p_ratio
        entropy_p = -p_ratio*math.log2(p_ratio) if p_ratio != 0 else 0
        entropy_n = - n_ratio*math.log2(n_ratio) if n_ratio !=0 else 0
        return entropy_p + entropy_n

    def _get_gain(self, feat):
        avg_info=0
        for val in self.data[feat].unique():
            avg_info+=self._get_entropy(self.data[self.data[feat] == val])*sum(self.data[fe
        return self._get_entropy(df) - avg_info

    def _get_splitter(self):
        self.splitter = max(self.gains, key = itemgetter(1))[0]

    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features]
            self._get_splitter()
            residual_columns = [k for k in self.data.columns if k != self.splitter]
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter]==val][residual_columns]
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val, self.split
                tmp_node.update_nodes()
                self.childs.append(tmp_node)
```

1

```
df = pd.read_csv('id3.csv')
df
```

Out[11]:

	Outlook	Temperature	Humidity	WindSpeed	Play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cool	Normal	Weak	Yes
5	Rainy	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rainy	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rainy	Mild	High	Strong	No

In [12]:

```
def print_tree(n):
    for child in n.childs:
        if child:
            print(child.__dict__.get('parent', ''))
            print(child.__dict__.get('parent_val', ''), '\n')
            print_tree(child)
```

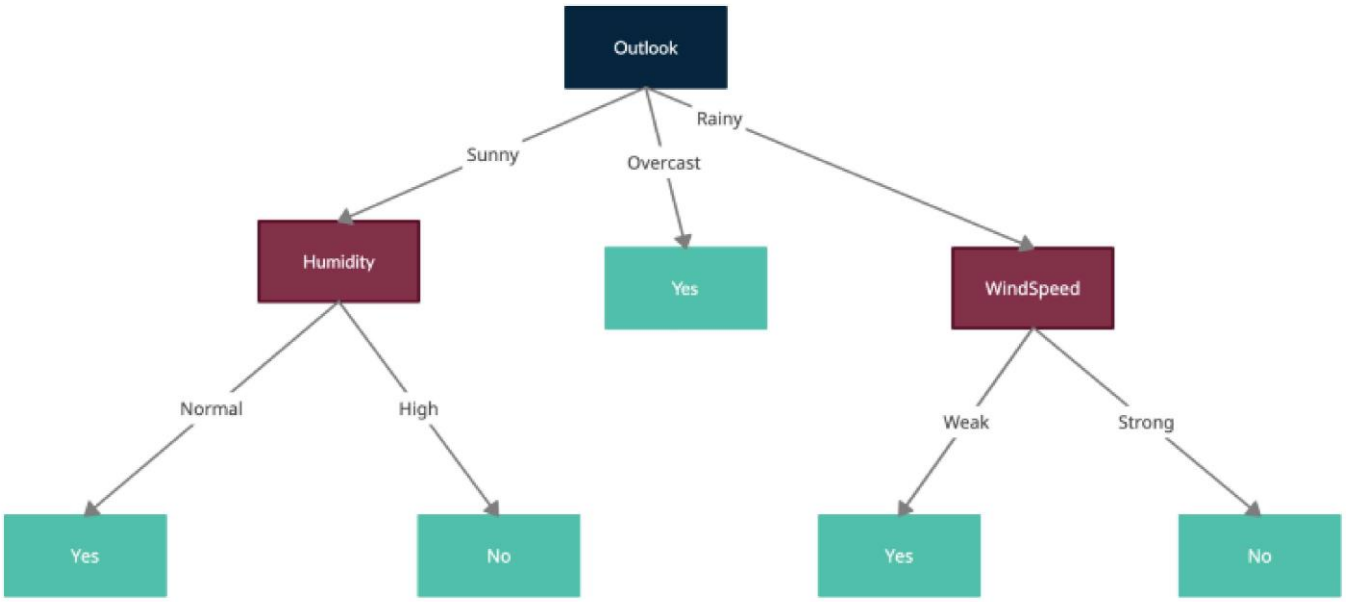
In [13]:

```
dt = DecisionTree(df, 'Play', 'Yes', '', '')
dt.update_nodes()
```

4

```
print_tree(dt)
```

```
Outlook
Sunny

Humidity
High

Humidity
Normal

Outlook
Overcast

Outlook
Rainy

WindSpeed
Weak

WindSpeed
Strong
```



Example 2

5

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree


iris = load_iris()

df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],columns= iris['feature_names']
df
```

Out[15]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

```
def compute_entropy(y):
    if len(y) < 2:
        return 0
    freq = np.array( y.value_counts(normalize=True) )
    return -(freq * np.log2(freq + 1e-6)).sum()

def compute_info_gain(samples, attr, target):
    values = samples[attr].value_counts(normalize=True)
    split_ent = 0
    for v, fr in values.iteritems():
        index = samples[attr]==v
        sub_ent = compute_entropy(target[index])
        split_ent += fr * sub_ent

    ent = compute_entropy(target)
    return ent - split_ent

class TreeNode:
    def __init__(self, node_name="", min_sample_num=10, default_decision=None):
        self.children = {}
        self.decision = None
        self.split_feat_name = None
        self.name = node_name
        self.default_decision = default_decision
        self.min_sample_num = min_sample_num

    def pretty_print(self, prefix=''):
        if self.split_feat_name is not None:
            for k, v in self.children.items():
                v.pretty_print(f"{prefix}:When {self.split_feat_name} is {k}")
        else:
            print(f"{prefix}:{self.decision}")

    def predict(self, sample):
        if self.decision is not None:
            print("Decision:", self.decision)
            return self.decision
        else:
            attr_val = sample[self.split_feat_name]
            child = self.children[attr_val]
            print("Testing ", self.split_feat_name, "->", attr_val)
            return child.predict(sample)

    def fit(self, X, y):
        if self.default_decision is None:
            self.default_decision = y.mode()[0]

        print(self.name, "received", len(X), "samples")
        if len(X) < self.min_sample_num:
            if len(X) == 0:
                self.decision = self.default_decision
                print("DECISION", self.decision)
            else:
                self.decision = y.mode()[0]
                print("DECISION", self.decision)
            return
        else:
            unique_values = y.unique()
            if len(unique_values) == 1:
```

```
            self.decision = unique_values[0]
            print("DECISION", self.decision)
            return
        else:
            info_gain_max = 0
            for a in X.keys():
                aig = compute_info_gain(X, a, y)
                if aig > info_gain_max:
                    info_gain_max = aig
                    self.split_feat_name = a
            print(f"Split by {self.split_feat_name}, IG: {info_gain_max:.2f}")
            self.children = {}
            for v in X[self.split_feat_name].unique():
                index = X[self.split_feat_name] == v
                self.children[v] = TreeNode(
                    node_name=self.name + ":" + self.split_feat_name + "==" + str(v),
                    min_sample_num=self.min_sample_num,
                    default_decision=self.default_decision)
                self.children[v].fit(X[index], y[index])
```

In [38]:

```
X = df.drop( "target", axis = 1)
y = df["target"]

t = TreeNode(min_sample_num=50)
t.fit(X, y)

attributes = ["sepal length (cm)","sepal width (cm)","petal length (cm)","petal width (cm)"
new_attributes = []
for a in attributes:
    new_a = "Quant4." + a
    df[new_a] = pd.qcut(df[a], q=4, labels=["q1", "q2", "q3", "q4"])
    new_attributes.append(new_a)
```

...

In [34]:

```
X = df[new_attributes]
y = df["target"]

t = TreeNode(min_sample_num=50)
t.fit(X, y)
```

```
 received 150 samples
Split by Quant4.petal length (cm), IG: 1.17
:Quant4.petal length (cm)==q1 received 44 samples
DECISION 0.0
:Quant4.petal length (cm)==q2 received 31 samples
DECISION 1.0
:Quant4.petal length (cm)==q3 received 41 samples
DECISION 1.0
:Quant4.petal length (cm)==q4 received 34 samples
DECISION 2.0
```

In [39]:

```
corr = 0
err_fp = 0
err_fn = 0
for (i, ct), tgt in zip(X.iterrows(), y):
    a = t.predict(ct)
    if a and not tgt:
        err_fp += 1
    elif not a and tgt:
        err_fn += 1
    else:
        corr += 1
```

...

In [22]:

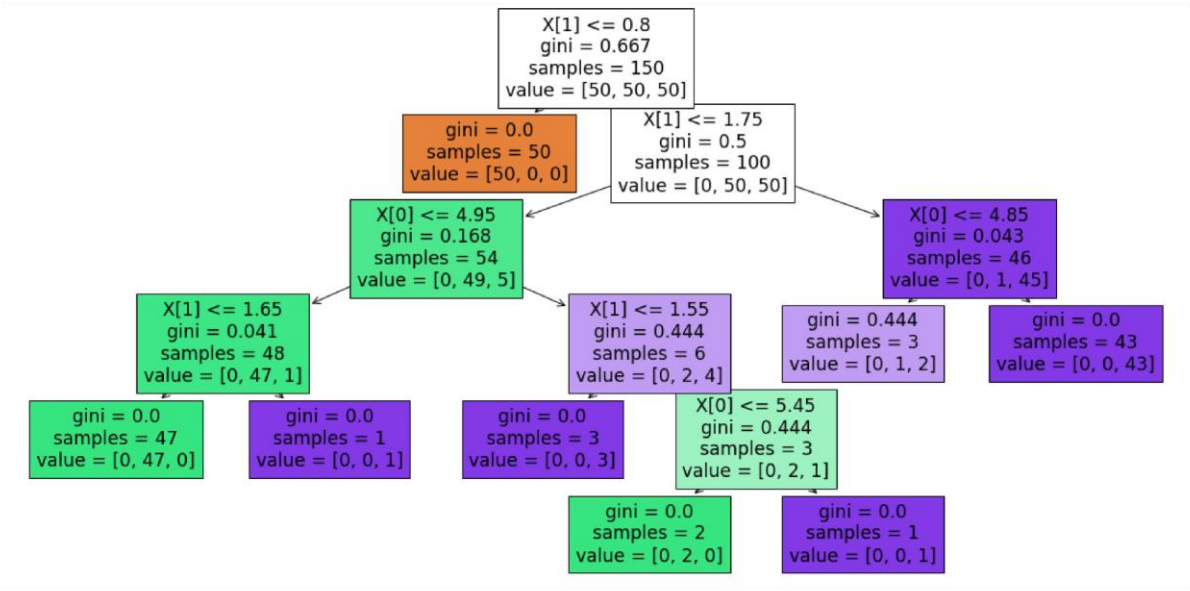
```
corr, err_fp, err_fn
```

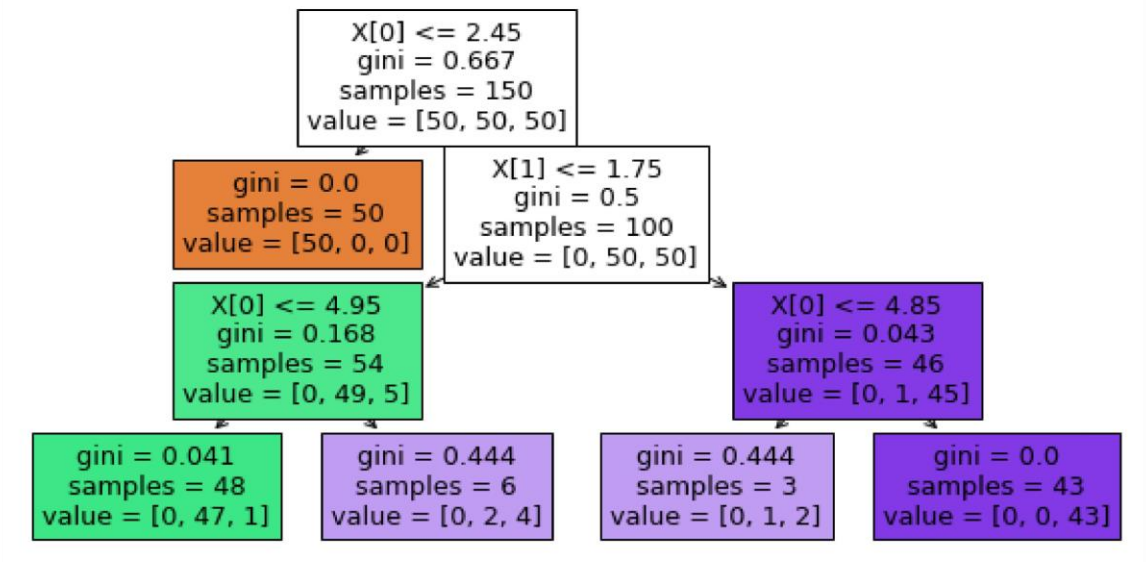
Out[22]:

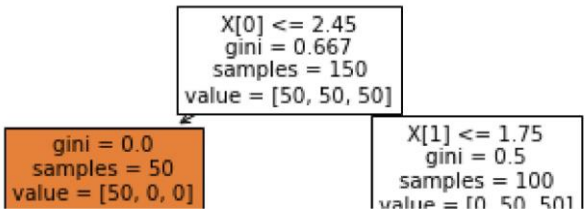
```
(144, 6, 0)
```

In [30]:

```
show_parms = DecisionTreeClassifier().fit(X, y)
plt.figure(figsize = (20,10))
plot_tree(show_parms, filled=True)

show_parms_max_depth_3 = DecisionTreeClassifier(max_depth=3).fit(X, y)
plt.figure(figsize = (10,5))
plot_tree(show_parms_max_depth_3, filled=True)

show_parms_max_depth_3 = DecisionTreeClassifier(min_impurity_decrease=0.01).fit(X, y)
plt.figure(figsize = (10,5))
plot_tree(show_parms_max_depth_3, filled=True)
plt.show()
```






X[0] <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
X[1] <= 1.75
gini = 0.5
samples = 100

Face recognition using Support Vector machine

In [35]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from sklearn.svm import SVC
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
```

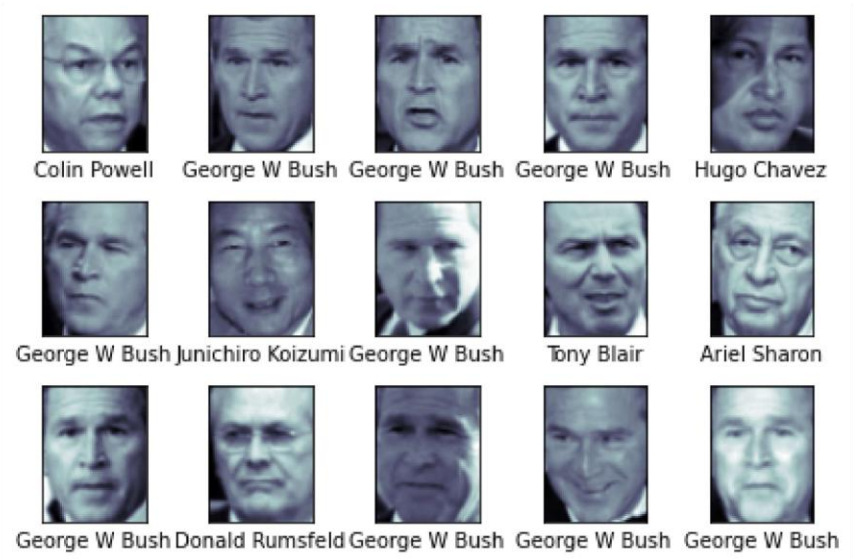
In [1]:

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

In [27]:

```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[], xlabel=faces.target_names[faces.target[i]])
fig.tight_layout()
```



In [33]:

```
pca = RandomizedPCA(n_components=150, whiten=True, random_state=42)
svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)

Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target, random_state=42)
```

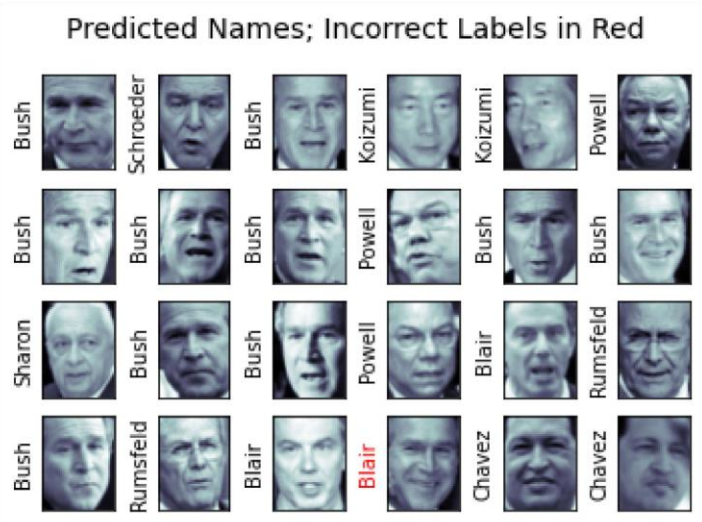
In [36]:

```
param_grid = {'svc__C': [1, 5, 10, 50], 'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid)

%time grid.fit(Xtrain, ytrain)
print(grid.best_params_)
```

```
Wall time: 53.4 s
{'svc__C': 10, 'svc__gamma': 0.001}
```

In [38]:

```
model = grid.best_estimator_
yfit = model.predict(Xtest)

fig, ax = plt.subplots(4, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
```



39]:

```
from sklearn.metrics import classification_report
print(classification_report(ytest, yfit, target_names=faces.target_names))
```

```
                   precision    recall  f1-score   support

     Ariel Sharon       0.65      0.73      0.69        15
     Colin Powell       0.80      0.87      0.83        68
  Donald Rumsfeld       0.74      0.84      0.79        31
    George W Bush       0.92      0.83      0.88       126
Gerhard Schroeder       0.86      0.83      0.84        23
      Hugo Chavez       0.93      0.70      0.80        20
Junichiro Koizumi       0.92      1.00      0.96        12
       Tony Blair       0.85      0.95      0.90        42

         accuracy                           0.85       337
        macro avg       0.83      0.84      0.84       337
     weighted avg       0.86      0.85      0.85       337
```

Implement k-Nearest Neighbour algorithm to classify the Iris data set

In [25]:

```
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import parallel_coordinates
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
```

In [3]:

```
data = pd.read_csv('Iris.csv')
data.shape
```

Out[3]:

```
(150, 6)
```

In [5]:

```
data.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

6]:

```
data.groupby('Species').size()
```

Out[6]:

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [9]:

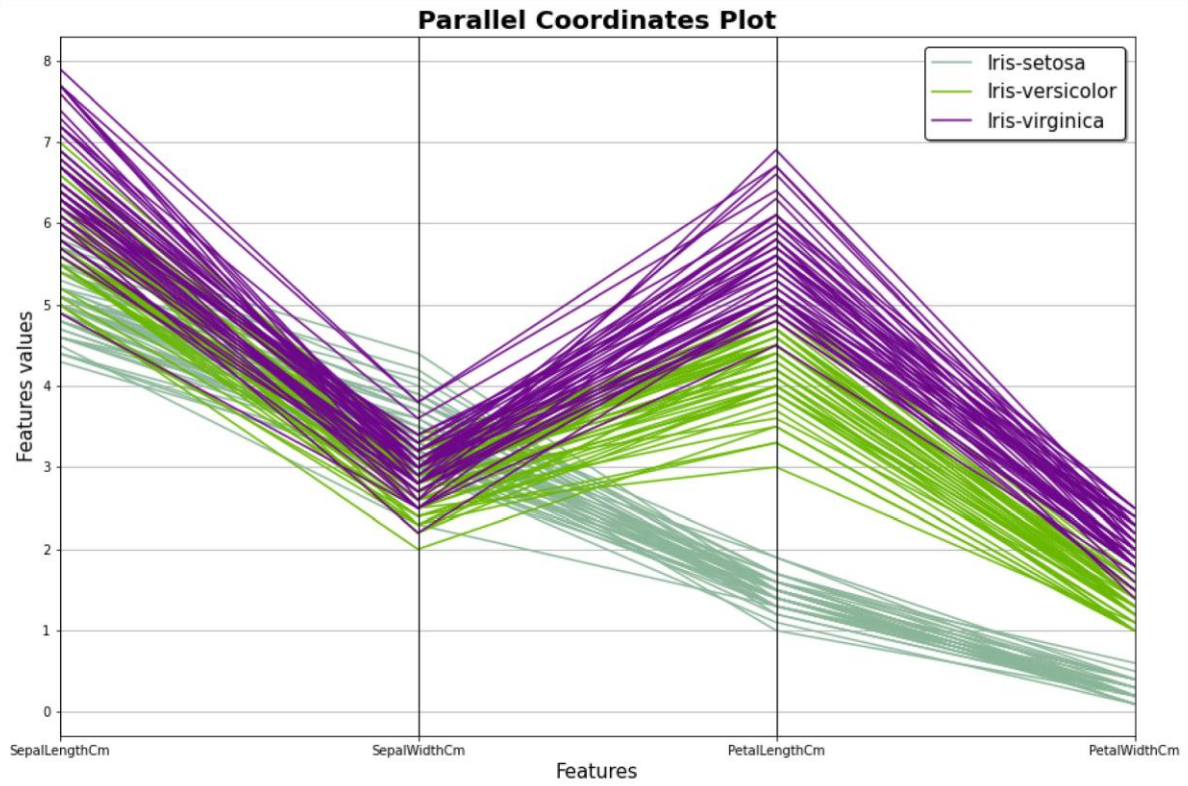
```
feature_columns = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']
X = data[feature_columns].values
y = data['Species'].values

le = LabelEncoder()
y = le.fit_transform(y)
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0
```

In [17]:

```
plt.figure(figsize=(15,10))
parallel_coordinates(data.drop("Id", axis=1), "Species")
plt.title('Parallel Coordinates Plot', fontsize=20, fontweight='bold')
plt.xlabel('Features', fontsize=15)
plt.ylabel('Features values', fontsize=15)
plt.legend(loc=1, prop={'size': 15}, frameon=True,shadow=True, facecolor="white", edgecolor
plt.show()
```



```
classifier = KNeighborsClassifier(n_neighbors=3)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

In [20]:

```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[20]:

```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

In [21]:

```
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
Accuracy of our model is equal 96.67 %.
```

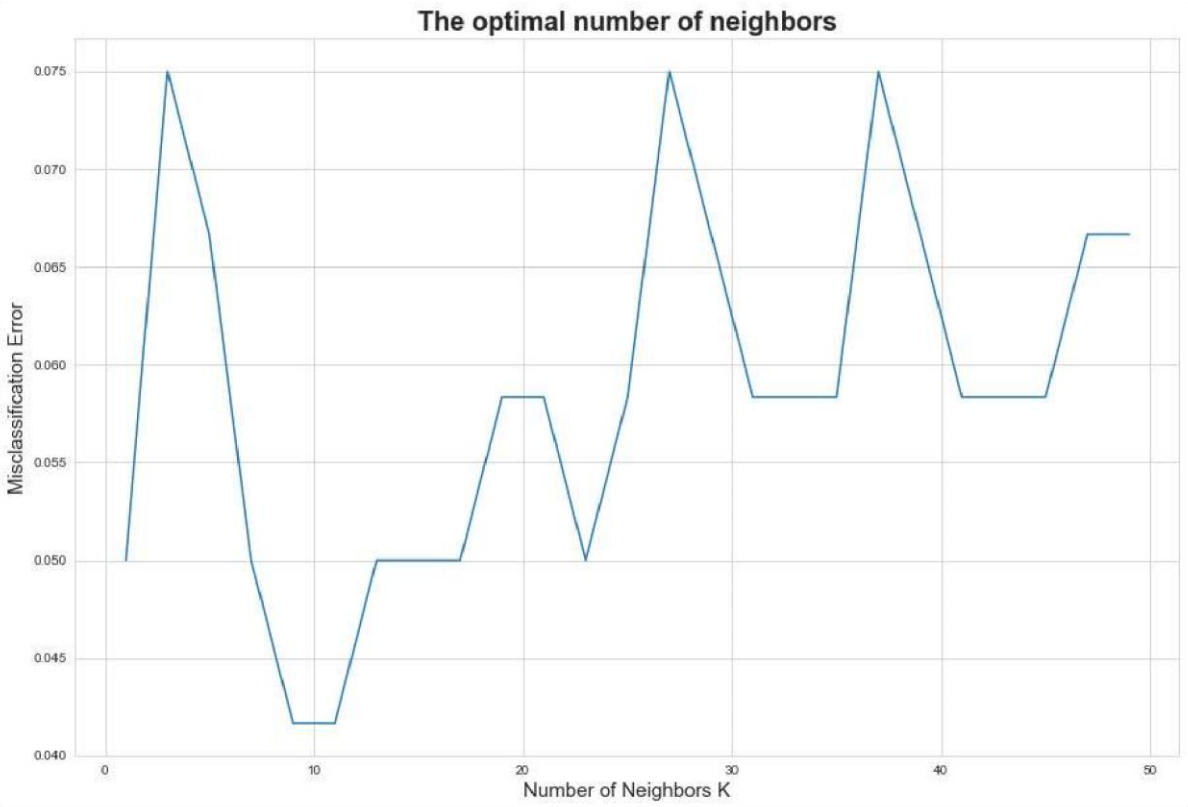
In [22]:

```
k_list = list(range(1,50,2))
cv_scores = []

for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
```

4

```
MSE = [1 - x for x in cv_scores]

plt.figure()
plt.figure(figsize=(15,10))
plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)

plt.show()

best_k = k_list[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d." % best_k)
```

```
<Figure size 432x288 with 0 Axes>
```




```
The optimal number of neighbors is 9.
```

```
#KNN Python code

class MyKNeighborsClassifier():

    def __init__(self, n_neighbors=5):
        self.n_neighbors=n_neighbors

    def fit(self, X, y):
        n_samples = X.shape[0]
        if self.n_neighbors > n_samples:
            raise ValueError("Number of neighbors can't be larger then number of samples in

        if X.shape[0] != y.shape[0]:
            raise ValueError("Number of samples in X and y need to be equal.")

        self.classes_ = np.unique(y)

        self.X = X
        self.y = y

    def predict(self, X_test):
        n_predictions, n_features = X_test.shape

        predictions = np.empty(n_predictions, dtype=int)

        for i in range(n_predictions):
            predictions[i] = single_prediction(self.X, self.y, X_test[i, :], self.n_neighbo

        return(predictions)

def single_prediction(X, y, x_train, k):
    n_samples = X.shape[0]

    distances = np.empty(n_samples, dtype=np.float64)

    for i in range(n_samples):
        distances[i] = (x_train - X[i]).dot(x_train - X[i])

    distances = sp.c_[distances, y]
    sorted_distances = distances[distances[:,0].argsort()]
    targets = sorted_distances[0:k,1]

    unique, counts = np.unique(targets, return_counts=True)
    return(unique[np.argmax(counts)])
```

In [27]:

```
my_classifier = MyKNeighborsClassifier(n_neighbors=3)

my_classifier.fit(X_train, y_train)

my_y_pred = my_classifier.predict(X_test)
```

8

```
accuracy = accuracy_score(y_test, my_y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
Accuracy of our model is equal 96.67 %.
```

House price prediction using Linear regression

In [9]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [10]:

```
housing_data =  pd.read_csv('home.txt', usecols = [0,2],names=['size','price'])
housing_data.head()
```

Out[10]:

	size	price
0	2104	399900
1	1600	329900
2	2400	369000
3	1416	232000
4	3000	539900

In [11]:

```
def gradient_descent(alpha, n_iters):
    X = housing_data['size'].values/1000
    y = housing_data['price'].values/1000
    m=len(X)
    theta0 = 0
    theta1 = 0
    for _ in range(n_iters):
        d_theta0 = []
        d_theta1 = []
        for i in range(m):
            d_theta0.append((theta0+theta1*X[i])-y[i])
            d_theta1.append(((theta0+theta1*X[i])-y[i])*X[i])

        theta0 = theta0-alpha*(1/m)*sum(d_theta0)
        theta1 = theta1-alpha*(1/m)*sum(d_theta1)

    return theta0,theta1
theta0,theta1 = gradient_descent(0.01,1000)
print('intercept_term(theta0):',theta0,'\n','bias_term(theta1):',theta1)
```

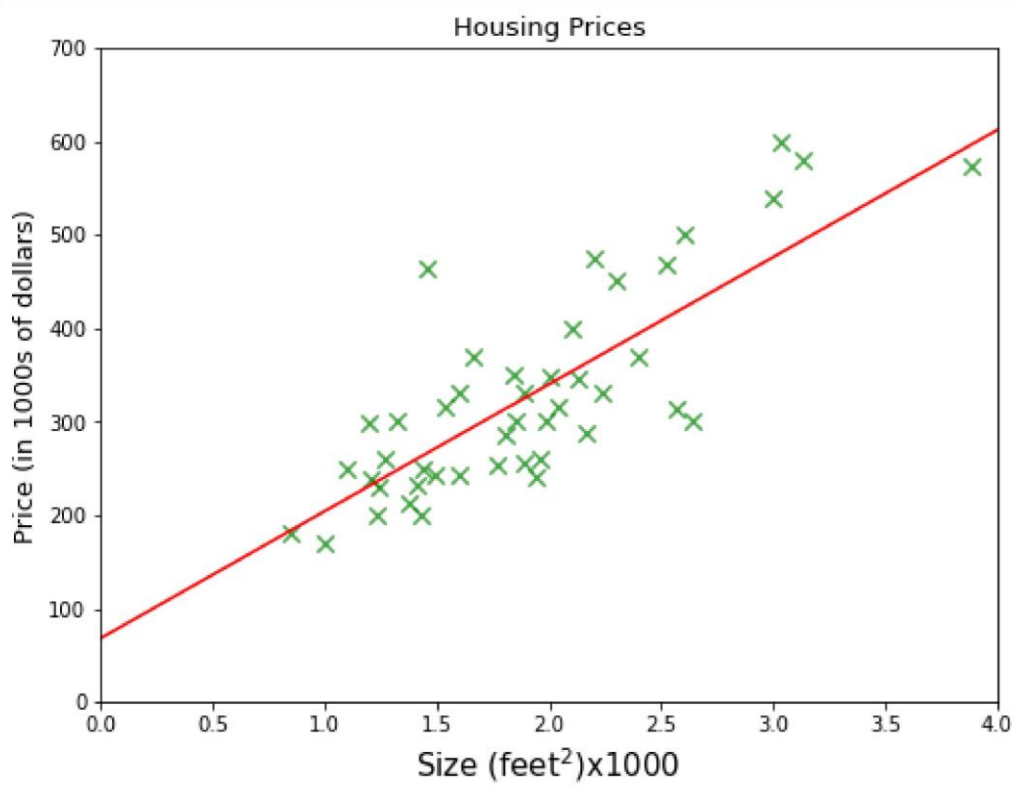
```
intercept_term(theta0): 68.12351315549327
 bias_term(theta1): 135.9217432231492
```

12]:

```
hypothesis = theta0+np.dot(theta1,[0,4])
x=[0,4]
%matplotlib inline
plt.figure(figsize=(8,6))
plt.scatter(housing_data['size']/1000,housing_data['price']/1000, c='g',marker='x',s=70, al
plt.plot(x,hypothesis, c='r')
plt.xlabel('Size (feet$^2$)x1000',size=15)
plt.ylabel('Price (in 1000s of dollars)', size=13)
plt.title('Housing Prices',size=13)
plt.plot()
plt.xlim(0,4)
plt.ylim(0,700)
```

Out[12]:

```
(0.0, 700.0)
```



In [13]:

```
def predict(X):
    return theta0 + theta1*X
predict(3)
```

Out[13]:

```
475.8887428249409
```

Program for implementation of apriori

In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori
```

In [7]:

```
data = pd.read_csv(r"Market_Basket_Optimisation.csv", low_memory=False, header=None)
data.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	10
0	shrimp	nut	lemon	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	lemon	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	iced tea	NaN	NaN	NaN	NaN	NaN	NaN

In [8]:

```
data.shape
```

Out[8]:

```
(7501, 20)
```

In [9]:

```
!pip install apyori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py): started
  Building wheel for apyori (setup.py): finished with status 'done'
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5975
sha256=0dea5ca949d2b6684e1cbd4fbe98fbe2a50b2cd4a0a9818886b19707d44197f3
  Stored in directory: c:\users\rajba\appdata\local\pip\cache\wheels\1b\02\6
c\a45230be8603bd95c0a51cd2b289aefdd860c1a100eab73661
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

In [10]:

```
list_of_transactions = []
for i in range(0, 7501):
    list_of_transactions.append([str(data.values[i,j]) for j in range(0, 20)])

list_of_transactions[0]
```

Out[10]:

```
['shrimp',
 'nut',
 'lemon',
 'vegetables mix',
 'green grapes',
 'whole weat flour',
 'yams',
 'cottage cheese',
 'energy drink',
 'tomato juice',
 'low fat yogurt',
 'iced tea',
 'honey',
 'salad',
 'mineral water',
 'salmon',
 'antioxydant juice',
 'frozen smoothie',
 'spinach',
 'olive oil']
```

In [13]:

```
rules = apriori(list_of_transactions, min_support = 0.004, min_confidence = 0.2, min_lift =
results = list(rules)
results[0]
```

Out[13]:

```
RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004532
728969470737, ordered_statistics=[OrderedStatistic(items_base=frozenset({'li
ght cream'}), items_add=frozenset({'chicken'}), confidence=0.290598290598290
57, lift=4.84395061728395)])
```

In [14]:

```
def inspect(results):
    lhs      =  [tuple(result [2] [0] [0]) [0] for result in results]
    rhs      =  [tuple(result [2] [0] [1]) [0] for result in results]
    supports = [result [1] for result in results]
    confidences = [result [2] [0] [2]   for result in results]
    lifts = [result [2] [0] [3]   for result in results]
    return list(zip(lhs,rhs,supports,confidences, lifts))
resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Hand Side', 'Right Hand
resultsinDataFrame.head(3)
```

Out[14]:

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812

In [15]:

```
resultsinDataFrame.nlargest(n=6, columns='Lift')
```

Out[15]:

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
7	light cream	nan	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
12	pasta	escalope	0.005866	0.372881	4.700812
30	pasta	shrimp	0.005066	0.322034	4.515096
6	pasta	shrimp	0.005066	0.322034	4.506672

Artificial Neural Network using the Backpropagation algorithm

In [1]:

```
import numpy as np
from sklearn.model_selection import train_test_split
```

In [2]:

```
db = np.loadtxt("duke-breast-cancer.txt")
print("Database raw shape (%s,%s)" % np.shape(db))
```

```
Database raw shape (86,7130)
```

In [3]:

```
np.random.shuffle(db)
y = db[:, 0]
x = np.delete(db, [0], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
print(np.shape(x_train),np.shape(x_test))
```

```
(77, 7129) (9, 7129)
```

In [4]:

```
hidden_layer = np.zeros(72)
weights = np.random.random((len(x[0]), 72))
output_layer = np.zeros(2)
hidden_weights = np.random.random((72, 2))
```

Sum Function

In [5]:

```
def sum_function(weights, index_locked_col, x):
    result = 0
    for i in range(0, len(x)):
        result += x[i] * weights[i][index_locked_col]
    return result
```

Activation Function

In [6]:

```
def activate_layer(layer, weights, x):
    for i in range(0, len(layer)):
        layer[i] = 1.7159 * np.tanh(2.0 * sum_function(weights, i, x) / 3.0)
```

Soft-Max function

In [8]:

```
def soft_max(layer):
    soft_max_output_layer = np.zeros(len(layer))
    for i in range(0, len(layer)):
        denominator = 0
        for j in range(0, len(layer)):
            denominator += np.exp(layer[j] - np.max(layer))
        soft_max_output_layer[i] = np.exp(layer[i] - np.max(layer)) / denominator
    return soft_max_output_layer
```

Recalculate weights function

In [9]:

```
def recalculate_weights(learning_rate, weights, gradient, activation):
    for i in range(0, len(weights)):
        for j in range(0, len(weights[i])):
            weights[i][j] = (learning_rate * gradient[j] * activation[i]) + weights[i][j]
```

Back-propagation function

In [11]:

```
def back_propagation(hidden_layer, output_layer, one_hot_encoding, learning_rate, x):
    output_derivative = np.zeros(2)
    output_gradient = np.zeros(2)

    for i in range(0, len(output_layer)):
        output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]
    for i in range(0, len(output_layer)):
        output_gradient[i] = output_derivative[i] * (one_hot_encoding[i] - output_layer[i])

    hidden_derivative = np.zeros(72)
    hidden_gradient = np.zeros(72)

    for i in range(0, len(hidden_layer)):
        hidden_derivative[i] = (1.0 - hidden_layer[i]) * (1.0 + hidden_layer[i])
    for i in range(0, len(hidden_layer)):
        sum_ = 0
        for j in range(0, len(output_gradient)):
            sum_ += output_gradient[j] * hidden_weights[i][j]
        hidden_gradient[i] = sum_ * hidden_derivative[i]

    recalculate_weights(learning_rate, hidden_weights, output_gradient, hidden_layer)
    recalculate_weights(learning_rate, weights, hidden_gradient, x)
```

13]:

```
one_hot_encoding = np.zeros((2,2))
for i in range(0, len(one_hot_encoding)):
    one_hot_encoding[i][i] = 1
training_correct_answers = 0

for i in range(0, len(x_train)):
    activate_layer(hidden_layer, weights, x_train[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    training_correct_answers += 1 if y_train[i] == np.argmax(output_layer) else 0
    back_propagation(hidden_layer, output_layer, one_hot_encoding[int(y_train[i])], -1, x_t

print("MLP Correct answers while learning: %s / %s (Accuracy = %s) on %s database." % (trai
                                                                    training_correct_answers/len
```

```
MLP Correct answers while learning: 63 / 77 (Accuracy = 0.8181818181818182)
on Duke breast cancer database.
```

In [14]:

```
testing_correct_answers = 0
for i in range(0, len(x_test)):
    activate_layer(hidden_layer, weights, x_test[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    testing_correct_answers += 1 if y_test[i] == np.argmax(output_layer) else 0

print("MLP Correct answers while testing: %s / %s (Accuracy = %s) on %s database" % (testin
                                                                testing_correct_answers/le
```

```
MLP Correct answers while testing: 4 / 9 (Accuracy = 0.4444444444444444) on
Duke breast cancer database
```

Implementation of Clustering Algorithms

In [28]:

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from itertools import product
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc

warnings.filterwarnings('ignore')
```

K-Means Clustering

In [3]:

```
dataset = pd.read_csv('Mall_Customers.csv',index_col='CustomerID')
dataset.head()
```

Out[3]:

CustomerID	Genre	Age	Annual_Income_(k$)	Spending_Score
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40

5]:

```
dataset.describe()
```

Out[5]:

	Age	Annual_Income_(k$)	Spending_Score
count	200.000000	200.000000	200.000000
mean	38.850000	60.560000	50.200000
std	13.969007	26.264721	25.823522
min	18.000000	15.000000	1.000000
25%	28.750000	41.500000	34.750000
50%	36.000000	61.500000	50.000000
75%	49.000000	78.000000	73.000000
max	70.000000	137.000000	99.000000

In [6]:

```
dataset.isnull().sum()
```

Out[6]:

```
Genre                  0
Age                    0
Annual_Income_(k$)     0
Spending_Score         0
dtype: int64
```

In [10]:

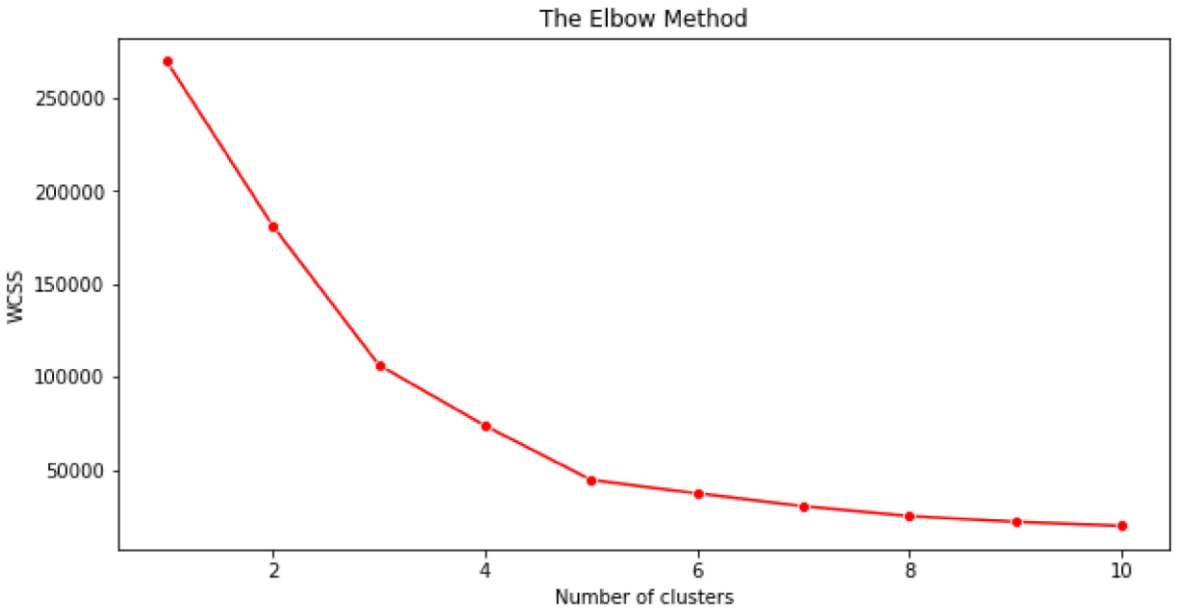
```
dataset.drop_duplicates(inplace=True)
X = dataset.iloc[:, [2, 3]].values
```

In [14]:

```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

15

```
plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



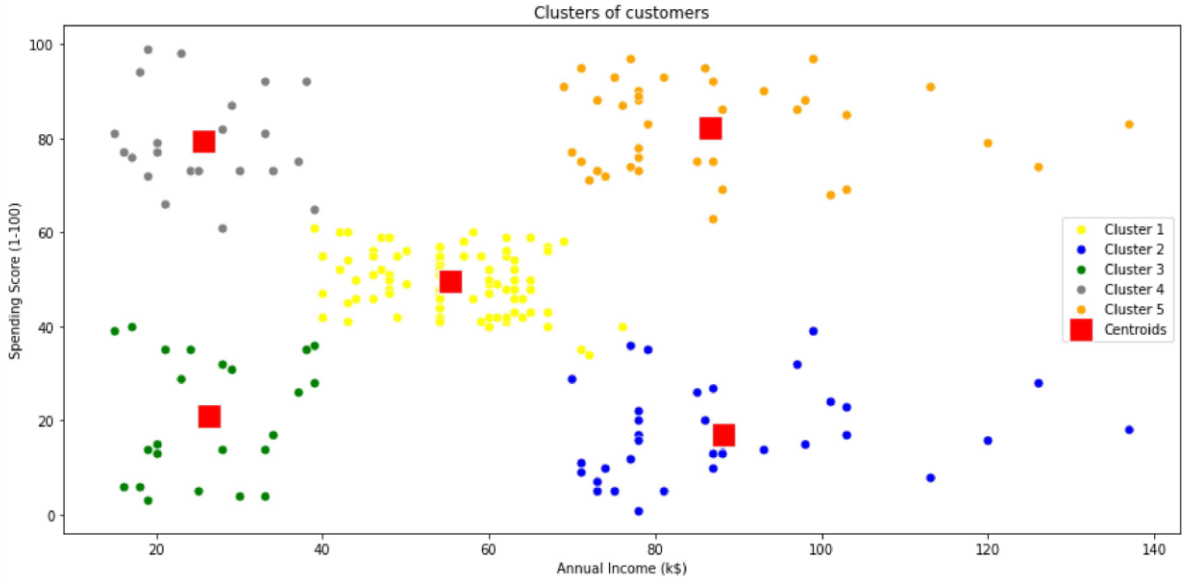
In [16]:

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Hierarchical clustering

18

```
plt.figure(figsize=(15,7))
sns.scatterplot(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], color = 'yellow', label = 'Cluste
sns.scatterplot(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], color = 'blue', label = 'Cluster
sns.scatterplot(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], color = 'green', label = 'Cluster
sns.scatterplot(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], color = 'grey', label = 'Cluster
sns.scatterplot(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], color = 'orange', label = 'Cluste
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color = 'red'
plt.grid(False)
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



1

```
raw_df = pd.read_csv('CC GENERAL.csv')
raw_df = raw_df.drop('CUST_ID', axis = 1)
raw_df.fillna(method ='ffill', inplace = True)
raw_df.head()
```

Out[21]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_F
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [22]:

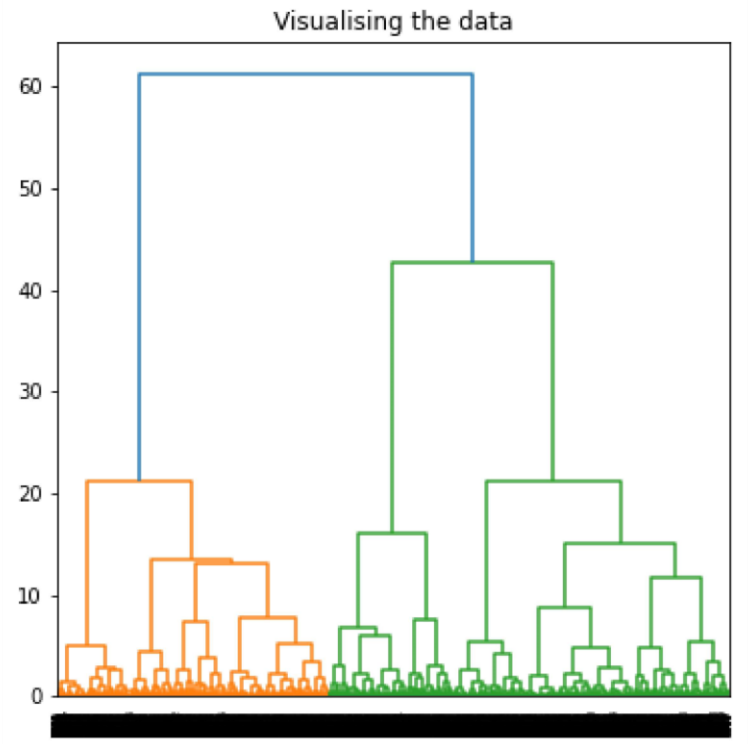
```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(raw_df)

normalized_df = normalize(scaled_df)

normalized_df = pd.DataFrame(normalized_df)

pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

X_principal.head()
```

Out[22]:

	P1	P2
0	-0.489949	-0.679976
1	-0.519099	0.544827
2	0.330633	0.268880
3	-0.481656	-0.097614
4	-0.563512	-0.482506

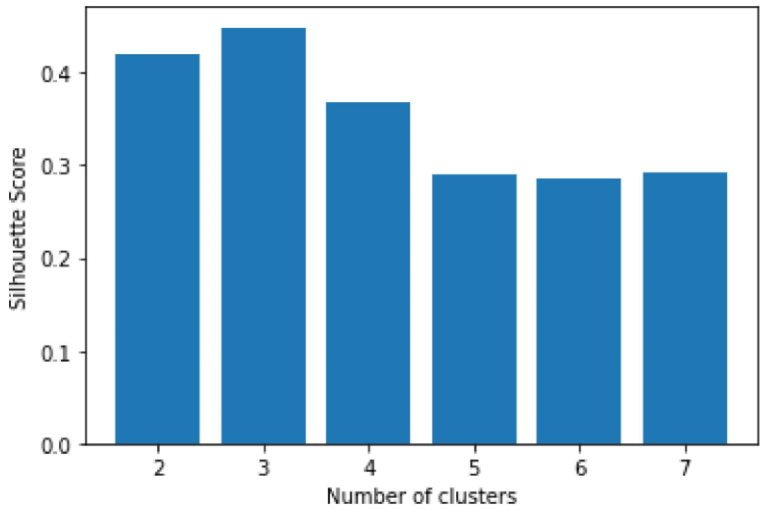
3

```
plt.figure(figsize =(6, 6))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='ward')))
```



4

```
silhouette_scores = []

for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(X_principal, AgglomerativeClustering(n_clusters = n_cluster).fit_p

k = [2, 3, 4, 5, 6,7]
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```



In [25]:

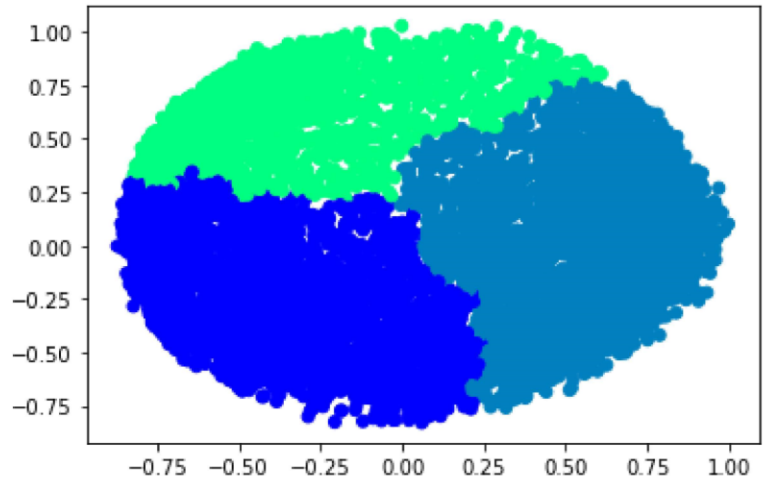
```
agg = AgglomerativeClustering(n_clusters=3)
agg.fit(X_principal)
```

Out[25]:

```
AgglomerativeClustering(n_clusters=3)
```

In [26]:

```
plt.scatter(X_principal['P1'], X_principal['P2'], c = AgglomerativeClustering(n_clusters =
            cmap =plt.cm.winter)
plt.show()
```



Density-based clustering

In [29]:

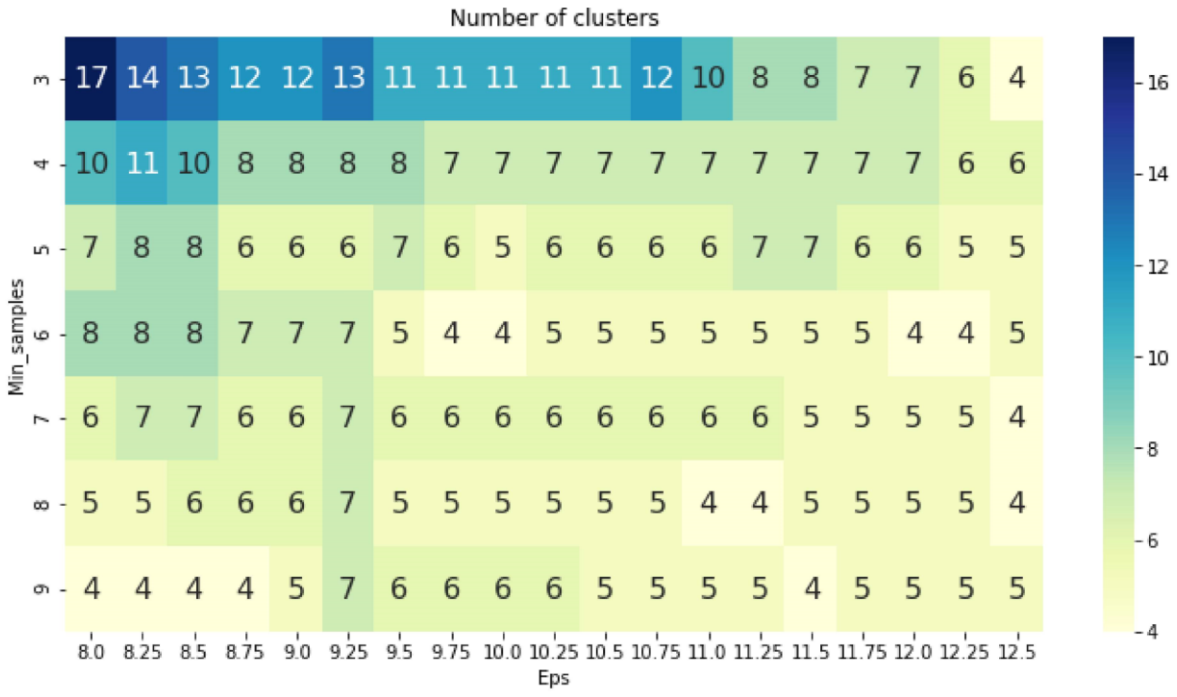
```
eps_values = np.arange(8,12.75,0.25)
min_samples = np.arange(3,10)
DBSCAN_params = list(product(eps_values, min_samples))
```

In [34]:

```
mall_data = pd.read_csv('Mall_Customers_2.csv')
X_numerics = mall_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
no_of_clusters = []
sil_score = []

for p in DBSCAN_params:
    DBS_clustering = DBSCAN(eps=p[0], min_samples=p[1]).fit(X_numerics)
    no_of_clusters.append(len(np.unique(DBS_clustering.labels_)))
    sil_score.append(silhouette_score(X_numerics, DBS_clustering.labels_))
```

In [37]:

```
tmp = pd.DataFrame.from_records(DBSCAN_params, columns =['Eps', 'Min_samples'])
tmp['No_of_clusters'] = no_of_clusters

pivot_1 = pd.pivot_table(tmp, values='No_of_clusters', index='Min_samples', columns='Eps')

fig, ax = plt.subplots(figsize=(12,6))
sns.heatmap(pivot_1, annot=True,annot_kws={"size": 16}, cmap="YlGnBu", ax=ax)
ax.set_title('Number of clusters')
plt.show()
```



In [39]:

```
DBS_clustering = DBSCAN(eps=12.5, min_samples=4).fit(X_numerics)

DBSCAN_clustered = X_numerics.copy()
DBSCAN_clustered.loc[:,'Cluster'] = DBS_clustering.labels_

DBSCAN_clust_sizes = DBSCAN_clustered.groupby('Cluster').size().to_frame()
DBSCAN_clust_sizes.columns = ["DBSCAN_size"]

DBSCAN_clust_sizes
```

Out[39]:

Cluster	DBSCAN_size
-1	18
0	112
1	8
2	34
3	24
4	4

In [40]:

```
outliers = DBSCAN_clustered[DBSCAN_clustered['Cluster']==-1]

fig2, (axes) = plt.subplots(1,2,figsize=(12,5))


sns.scatterplot('Annual Income (k$)', 'Spending Score (1-100)', data=DBSCAN_clustered[DBSCA
                            hue='Cluster', ax=axes[0], palette='Set1', legend='full', s=45)

sns.scatterplot('Age', 'Spending Score (1-100)', data=DBSCAN_clustered[DBSCAN_clustered['Cl
                            hue='Cluster', palette='Set1', ax=axes[1], legend='full', s=45)

axes[0].scatter(outliers['Annual Income (k$)'], outliers['Spending Score (1-100)'], s=5, la
axes[1].scatter(outliers['Age'], outliers['Spending Score (1-100)'], s=5, label='outliers',
axes[0].legend()
axes[1].legend()

plt.setp(axes[0].get_legend().get_texts(), fontsize='10')
plt.setp(axes[1].get_legend().get_texts(), fontsize='10')

plt.show()
```