PORT CITY INTERNATIONAL UNIVERSITY

(PCIU)

Department of Computer Science & Engineering

(CSE)

Report

Project name : Mid term report

Course Title : Pattern Recognition Sessional

Course Code : CSE 331

Submitted to : Mr. Shafayet Abir

Lecture of PCIU

Dept.of CSE

Submitted by : Swehlamong Marma

CSE 01806789(C)

Dept.of CSE

Data of Submission : 09.02.2022

Experiment No: 01

Experiment Name: Visualization, mean, mode, standard deviation.

Objective: I want to visualize data and predict mean, median and standard deviation using Melbourne Housing Prices dataset Problem.

Theory: Mean is also known as average of all the numbers in the data set. Median is mid value in this ordered data set. It is a measure of dispersion of observation within dataset relative to their mean. Standard deviation is expressed in the same unit as the values in the dataset so it measure how much observations of the data set differs from its mean.

Description:

Code+Output:

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('MELBOURNE_HOUSE_PRICES_LESS.csv')
```

In [3]:
```
df
```

Out[3]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Postcode	Regionname	Propertycount	Distance	CouncilArea
0	Abbotsford	49 Lithgow St	3	h	1490000.0	S	Jellis	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
1	Abbotsford	59A Turner St	3	h	1220000.0	S	Marshall	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
2	Abbotsford	119B Yarra St	3	h	1420000.0	S	Nelson	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
3	Aberfeldie	68 Vida St	3	h	1515000.0	S	Barry	1/04/2017	3040	Western Metropolitan	1543	7.5	Moonee Valley City Council
4	Airport West	92 Clydesdale Rd	2	h	670000.0	S	Nelson	1/04/2017	3042	Western Metropolitan	3464	10.4	Moonee Valley City Council
...	...	...	...	...	...	...	...	...	...	...	...	...	...
63018	Roxburgh Park	3 Carr Pl	3	h	566000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63019	Roxburgh Park	9 Parker Ct	3	h	500000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63020	Roxburgh Park	5 Parkinson Wy	3	h	545000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63021	Thomastown	3/1 Travers St	3	u	NaN	PI	Barry	31/03/2018	3074	Northern Metropolitan	7955	15.3	Whittlesea City Council

63023 rows × 13 columns

In [4]:
```
df.head(10)
```

Out[4]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Postcode	Regionname	Propertycount	Distance	CouncilArea
0	Abbotsford	49 Lithgow St	3	h	1490000.0	S	Jellis	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
1	Abbotsford	59A Turner St	3	h	1220000.0	S	Marshall	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
2	Abbotsford	119B Yarra St	3	h	1420000.0	S	Nelson	1/04/2017	3067	Northern Metropolitan	4019	3.0	Yarra City Council
3	Aberfeldie	68 Vida St	3	h	1515000.0	S	Barry	1/04/2017	3040	Western Metropolitan	1543	7.5	Moonee Valley City Council
4	Airport West	92 Clydesdale Rd	2	h	670000.0	S	Nelson	1/04/2017	3042	Western Metropolitan	3464	10.4	Moonee Valley City Council
5	Airport West	4/32 Earl St	2	t	530000.0	S	Jellis	1/04/2017	3042	Western Metropolitan	3464	10.4	Moonee Valley City Council
6	Airport West	3/74 Hawker St	2	u	540000.0	S	Barry	1/04/2017	3042	Western Metropolitan	3464	10.4	Moonee Valley City Council
7	Airport West	1/26 Highridge Cr	3	h	715000.0	SP	Nelson	1/04/2017	3042	Western Metropolitan	3464	10.4	Moonee Valley City Council
8	Albanvale	1 Jackson Cct	6	h	NaN	PI	hockingstuart	1/04/2017	3021	Western Metropolitan	1899	14.0	Brimbank City Council
9	Albert Park	18 Mills St	3	h	1925000.0	S	Cayzer	1/04/2017	3206	Southern Metropolitan	3280	3.0	Port Phillip City Council

In [5]:
```
df.tail(10)
```
Out[5]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Postcode	Regionname	Propertycount	Distance	CouncilArea
63013	Frankston	7 Ince Ct	5	h	710000.0	PI	hockingstuart	31/03/2018	3199	South-Eastern Metropolitan	17055	38.0	Frankston City Council
63014	Frankston	1/34 Petrie St	2	u	345000.0	SP	Aquire	31/03/2018	3199	South-Eastern Metropolitan	17055	38.0	Frankston City Council
63015	Frankston	3/34 Petrie St	2	u	340000.0	SP	Aquire	31/03/2018	3199	South-Eastern Metropolitan	17055	38.0	Frankston City Council
63016	Frankston	4/34 Petrie St	2	u	347700.0	SP	Aquire	31/03/2018	3199	South-Eastern Metropolitan	17055	38.0	Frankston City Council
63017	Preston	229 Murray Rd	3	h	808000.0	S	RW	31/03/2018	3072	Northern Metropolitan	14577	8.4	Darebin City Council
63018	Roxburgh Park	3 Carr Pl	3	h	566000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63019	Roxburgh Park	9 Parker Ct	3	h	500000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63020	Roxburgh Park	5 Parkinson Wy	3	h	545000.0	S	Raine	31/03/2018	3064	Northern Metropolitan	5833	20.6	Hume City Council
63021	Thomastown	3/1 Travers St	3	u	NaN	PI	Barry	31/03/2018	3074	Northern Metropolitan	7955	15.3	Whittlesea City Council
63022	Williams Landing	1 Diadem Wy	4	h	NaN	SP	Aussie	31/03/2018	3027	Western Metropolitan	1999	17.6	Wyndham City Council

In [6]:
```
df.size
```
Out[6]:
```
819299
```

In [7]:
```
df.shape
```
Out[7]:
```
(63023, 13)
```

In [8]:
```
df.columns
```
Out[8]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Postcode', 'Regionname', 'Propertycount', 'Distance',
       'CouncilArea'],
      dtype='object')
```

In [9]:
```
df.info
```
Out[9]:
```
<bound method DataFrame.info of                    Suburb           Address  Rooms Type      Price Method  \
0            Abbotsford      49 Lithgow St      3    h  1490000.0      S
1            Abbotsford      59A Turner St      3    h  1220000.0      S
2            Abbotsford      119B Yarra St      3    h  1420000.0      S
3            Aberfeldie         68 Vida St      3    h  1515000.0      S
4          Airport West   92 Clydesdale Rd      2    h   670000.0      S
...                 ...               ...    ...  ...        ...    ...
63018     Roxburgh Park          3 Carr Pl      3    h   566000.0      S
63019     Roxburgh Park        9 Parker Ct      3    h   500000.0      S
63020     Roxburgh Park    5 Parkinson Wy      3    h   545000.0      S
63021        Thomastown     3/1 Travers St      3    u        NaN     PI
63022  Williams Landing        1 Diadem Wy      4    h        NaN     SP

[63023 rows x 13 columns]>
```

In [10]:
```
df.isnull().sum()
```
Out[10]:
```
Suburb               0
Address              0
Rooms                0
Type                 0
Price            14590
Method               0
SellerG              0
Date                 0
Postcode             0
Regionname           0
Propertycount        0
Distance             0
CouncilArea          0
dtype: int64
```

In [11]:
```
df['Price'].mean()
```
Out[11]:
```
997898.2414882415
```

In [12]:
```
df['Price'].median()
```
Out[12]:
```
830000.0
```

In [13]:
```
df['Price'].std()
```
Out[13]:
```
593498.9190372757
```

Conclusion: Here I have found mean, median and standard deviation using a dataset using Pandas Library function.

Experiment No: 02

Experiment Name: Mobile price prediction using simple linear regression.

Objective: I want to predict the Mobile price thru simple linear regression.The Mobile Price dataset was built for regression analysis, linear regression and prediction models. It includes the date of purchase, ram, battery-power, display, dual-sim,camera etc.

Theory: Simple linear regression is used to model the relationship between two continuous variables. Often, the objective is to predict the value of an output variable based on the value of an input variable.

Description:

Code+Output:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
df = pd.read_csv('Mobile_Price.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	thı
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	

5 rows × 21 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
```

In [5]:
```
df.shape
```
Out[5]:
```
(2000, 21)
```

In [6]:
```
df.isnull().sum()
#df.dropna(inplace=True)
#df.drop('date',inplace=True,axis=1)
```
Out[6]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

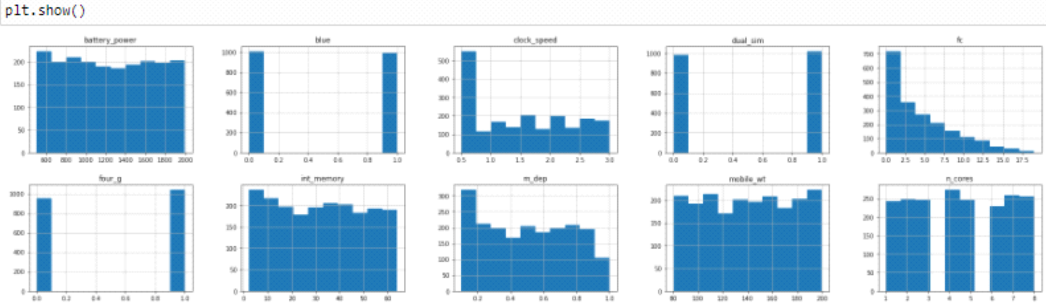
In [7]:
```
df.describe()
```
Out[7]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	12
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	4
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	5
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	8
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	12
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	16
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	19

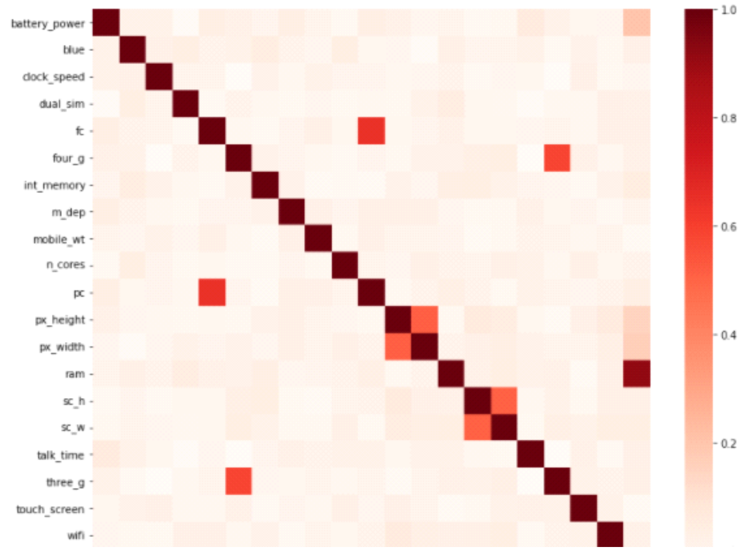
8 rows × 21 columns

In [8]:
```
df.drop(columns=['battery_power']).plot(kind='box',figsize=(50,10))
plt.show()
```

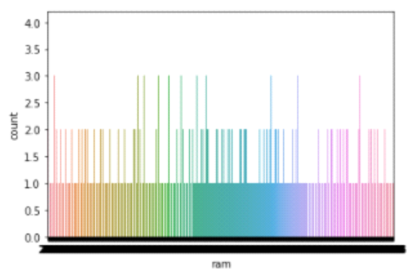
In [9]:
```
df.hist(figsize=(30,20))
plt.show()
```



In [10]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),cmap='Reds')
```
Out[10]: <AxesSubplot:>



In [11]:
```
sns.countplot(x='ram',data=df)
```
Out[11]: <AxesSubplot:xlabel='ram', ylabel='count'>



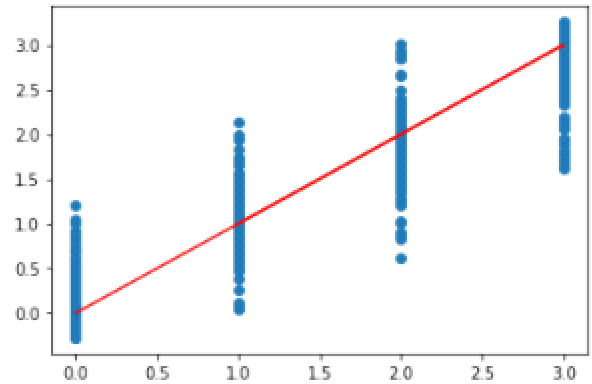
In [12]:
```
X = df[['ram']].values
y = df['price_range'].values
```
In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
```
In [14]:
```
lr = LinearRegression()
```
In [15]:
```
lr.fit(X_train,y_train)
```
Out[15]: LinearRegression()

In [16]:
```
pred_lr = lr.predict(X_test)
```
In [17]:
```
score_lr = lr.score(X_train,y_train)
print(lr.coef_[0])
print(lr.intercept_)
```
```
0.0009458570750908579
-0.519625321286413
```
In [18]:
```
mae_lr = mean_absolute_error(y_test,pred_lr)
mse_lr = mean_squared_error(y_test,pred_lr)
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test,pred_lr)

print('Mae_lr: ',mae_lr)
print('Mse_lr: ',mse_lr)
print('Rmse_lr: ',rmse_lr)
print('R2 score: ',r2_lr)
```
```
Mae_lr:  0.36589292470384405
Mse_lr:  0.22720105113283623
Rmse_lr:  0.4766561141250957
R2 score:  0.818180977006373
```

```
In [19]: plt.scatter(y_test,pred_lr)
         plt.plot(y_test,y_test,color='red')

Out[19]: [<matplotlib.lines.Line2D at 0x1ef3fdff1c0>]
```



Conclusion: Regression gives us a statistical model that enables us to predict a response at different values of the predictor, including values of the predictor not included in the original data.

Experiment No: 03

Experiment Name: Mobile price prediction using multiple linear regression.

Objective: I want to predict the Mobile price thru Multiple linear regression.The Mobile Price dataset was built for regression analysis, linear regression and prediction models. It includes the date of purchase, ram, battery-power, display, dual-sim,camera etc.

Theory: Multiple linear regression also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory variables and response variables. In essence, multiple regression is the extension of ordinary least-squares regression because it involves more than one explanatory variable.

Description:

Code + Output:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,mean_squared_error,mean_absolute_error,r2_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
```

In [2]:
```
df = pd.read_csv('Mobile_Price.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	thı
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	

5 rows × 21 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

In [5]:
```
df.shape
```

Out[5]:
```
(2000, 21)
```

In [6]:
```
df.isnull().sum()
#df.dropna(inplace=True)
#df.drop('date',inplace=True,axis=1)
```

Out[6]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
```

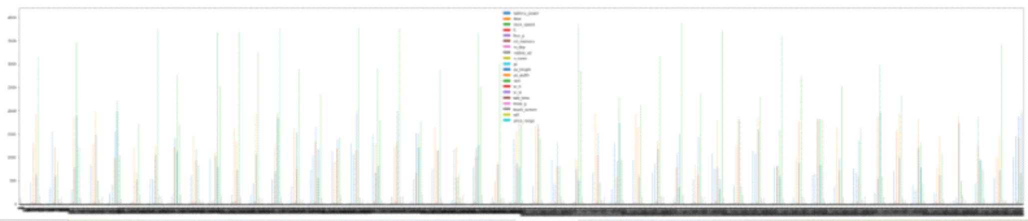
In [7]:
```
df.describe()
```

Out[7]:

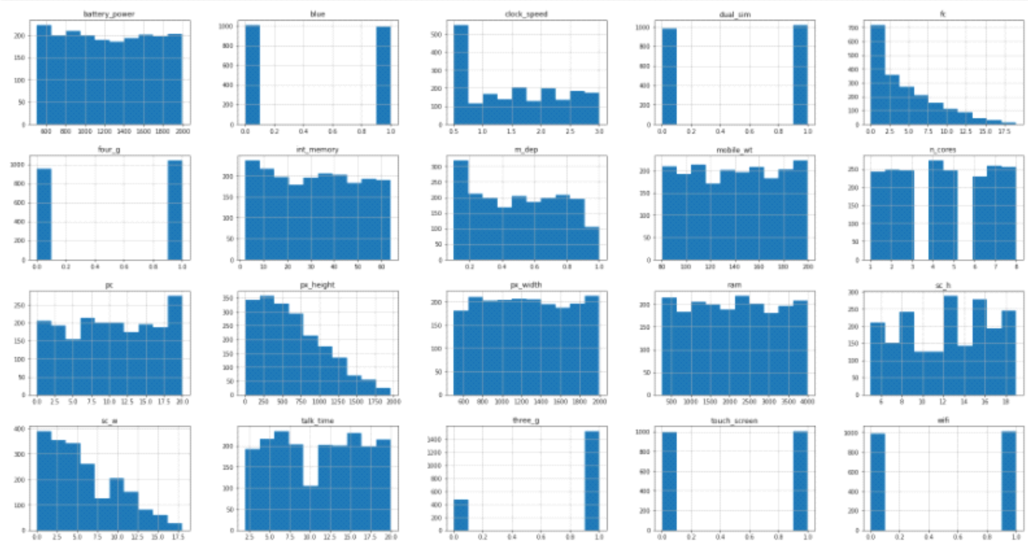
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	12
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	4
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	5
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	8
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	12
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	16
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	19

8 rows × 21 columns

In [8]:
```
df.drop(columns=['dual_sim']).plot(kind='bar',figsize=(50,10))
plt.show()
```

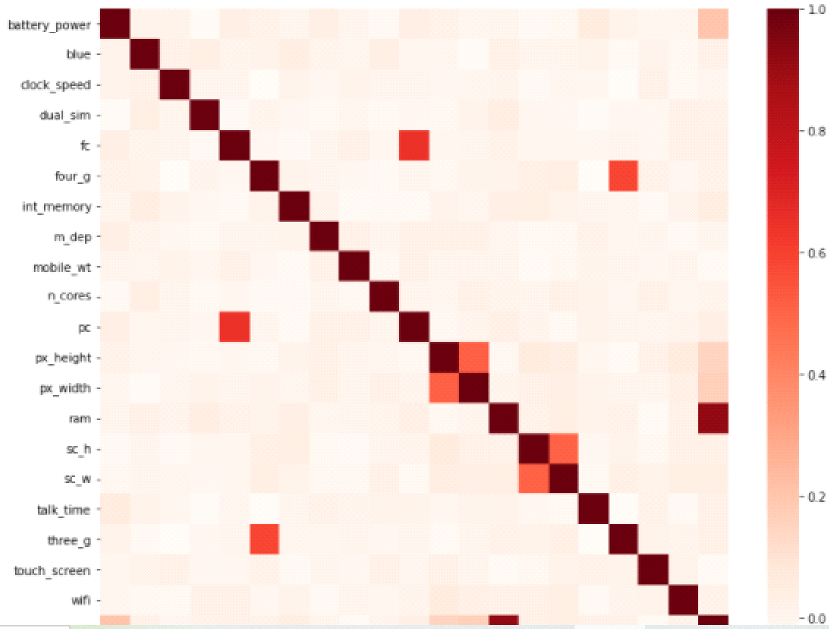


In [9]:
```
df.hist(figsize=(30,20))
plt.show()
```



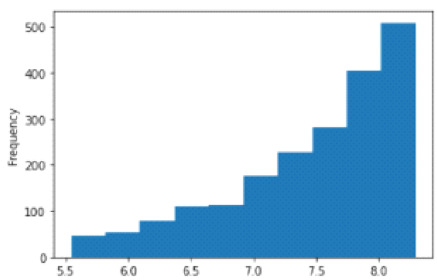
In [10]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),cmap='Reds')
```

Out[10]: <AxesSubplot:>



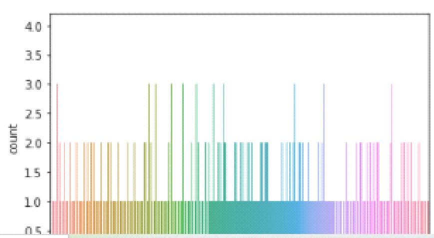
In [11]:
```
priceTransform=np.log(df.ram)
priceTransform.plot(kind='hist')
```

Out[11]: <AxesSubplot:ylabel='Frequency'>



In [12]:
```
sns.countplot(x='ram',data=df)
```

Out[12]: <AxesSubplot:xlabel='ram', ylabel='count'>



In [13]:
```python
X = df[['battery_power','blue','clock_speed','dual_sim','fc','four_g','int_memory','m_dep','mobile_wt',
       'ram',]].values
y = df['price_range'].values
```

In [14]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
```

In [15]:
```python
std = StandardScaler()
X = std.fit_transform(X)
```

In [16]:
```python
lr = LinearRegression()
```

In [17]:
```python
lr.fit(X_train,y_train)
```

Out[17]:
```
LinearRegression()
```

In [18]:
```python
pred_lr = lr.predict(X_test)
```

In [19]:
```python
score_lr = lr.score(X_train,y_train)
print(lr.coef_[0])
print(lr.intercept_)
```

```
0.0004931464786987822
-0.9885015845796081
```

In [20]:
```python
mae_lr = mean_absolute_error(y_test,pred_lr)
mse_lr = mean_squared_error(y_test,pred_lr)
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test,pred_lr)

print('Mae_lr: ',mae_lr)
print('Mse_lr: ',mse_lr)
print('Rmse_lr: ',rmse_lr)
print('Re score: ',r2_lr)
```

```
Mae_lr:  0.3111431534292541
Mse_lr:  0.15961083992103314
Rmse_lr:  0.39951325374890023
Re score:  0.8722704546086483
```

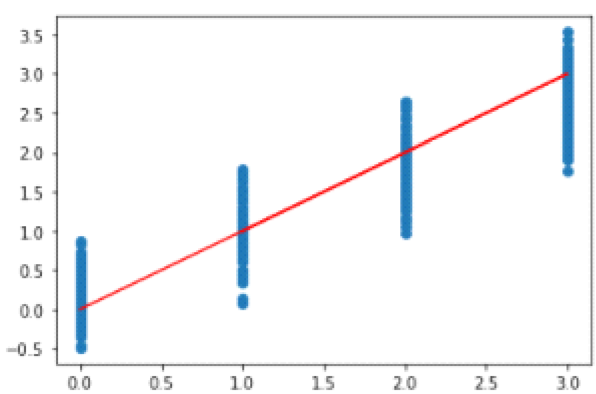
In [22]:
```python
preco_lr = df['price_range']
predict_lr = pd.Series(pred_lr)
error_lr = preco_lr-predict_lr
data_lr = {'price_range':preco_lr,'Predictions':predict_lr,'Error':error_lr}
data_prev_lr = pd.DataFrame(data_lr)
data_prev_lr.head()
```

Out[22]:

	price_range	Predictions	Error
0	1	-0.054638	1.054638
1	2	2.298339	-0.298339
2	2	0.005317	1.994683
3	2	3.001660	-1.001660
4	1	1.307006	-0.307006

In [23]:
```python
plt.scatter(y_test,pred_lr)
plt.plot(y_test,y_test,color='red')
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x2a99b0a7730>]
```



Conclusion: Multiple linear regression models is that we might need to estimate many coefficients. Although modern statistical software can easily fit these models, it is not always straightforward to identify important predictors and interpret the model coefficients.

Experiment No: 04

Experiment Name: Heart Attack Multinomial naive bayes.

Objective: The Naive Bayes method is a strong tool for analyzing text input and solving problems with numerous classes. Here find out shape, head, describe, vertical bar plot of heart dataset, horizontal bar plot of heart dataset, density plot of the dataset, histogram etc.

Theory: Naive Bayes classifier for multinomial models.The multinomial Naive Bayes classifier is suitable for classification with discrete features. The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts.

Description:

Code + Output:

In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score


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

In [2]:
```python
dataset = pd.read_csv('heart.csv')
dataset.shape
```

Out[2]: (303, 14)

In [3]:
```python
dataset = dataset.dropna()
dataset.shape
```

Out[3]: (303, 14)

In [4]:
```python
dataset.head()
```

Out[4]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [5]:
```python
dataset.describe()
```

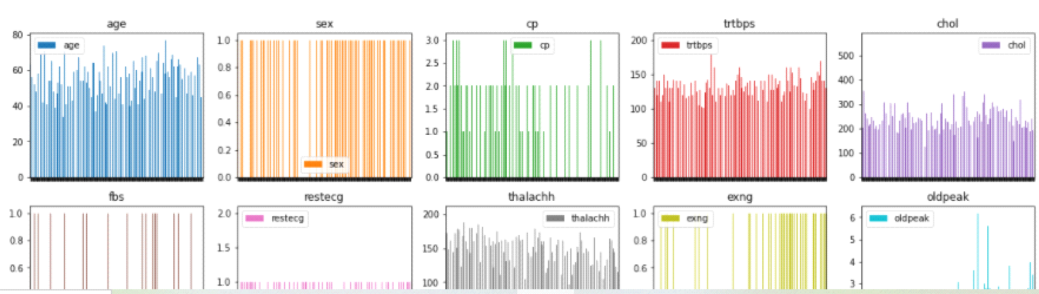
Out[5]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.31
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.00

In [6]:
```python
plt.figure()
dataset.plot(kind='bar', subplots=True, layout=(3,5),figsize=(20,10))
plt.xlabel('rows')
plt.ylabel('data')
plt.legend()
plt.title('vertical bar plot of heart dataset')
plt.show()
```

```
No handles with labels found to put in legend.
```

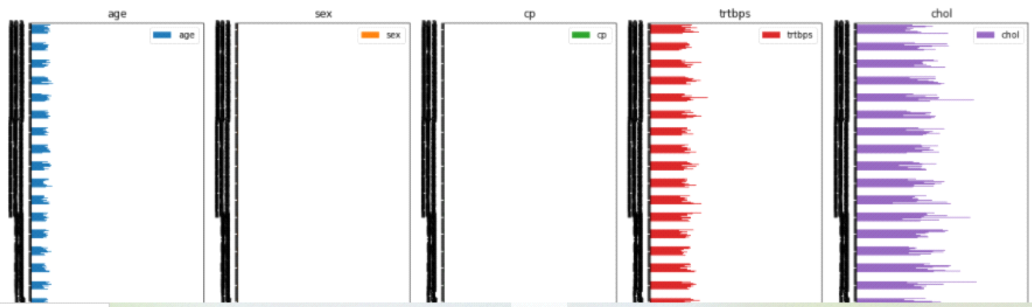
```
<Figure size 432x288 with 0 Axes>
```



In [7]:
```
plt.figure()
dataset.plot(kind='barh', subplots=True, layout=(3,5),figsize=(20,20))
plt.xlabel('rows')
plt.ylabel('data')
plt.legend()
plt.title('horizontal bar plot of heart dataset')
plt.show()
```

```
No handles with labels found to put in legend.
```

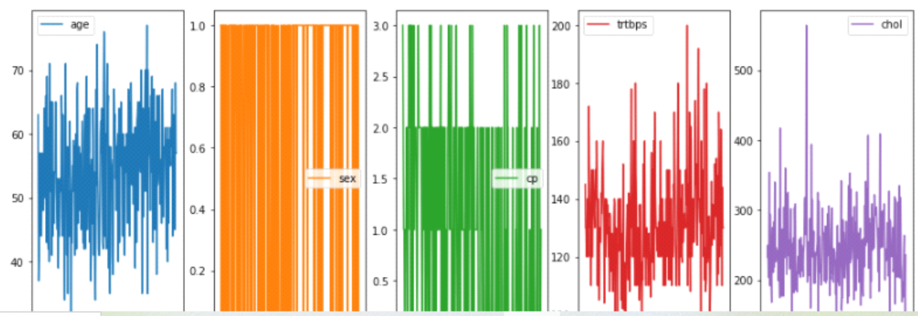
```
<Figure size 432x288 with 0 Axes>
```



In [8]:
```
plt.figure()
dataset.plot(kind='line', subplots=True, layout=(3,5),figsize=(15,20))
plt.xlabel('rows')
plt.ylabel('data')
plt.legend()
plt.title('line plot of heart dataset')
plt.show()
```

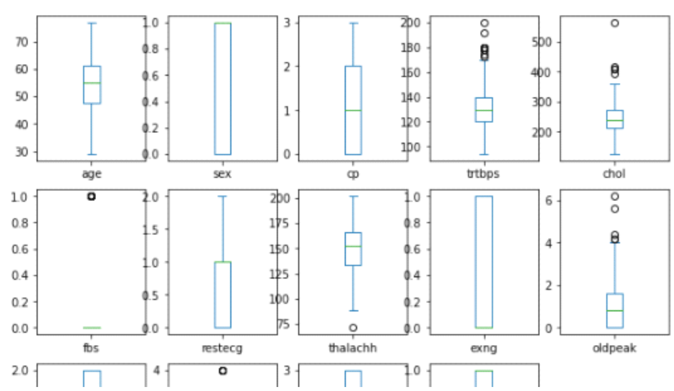
```
No handles with labels found to put in legend.
```

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



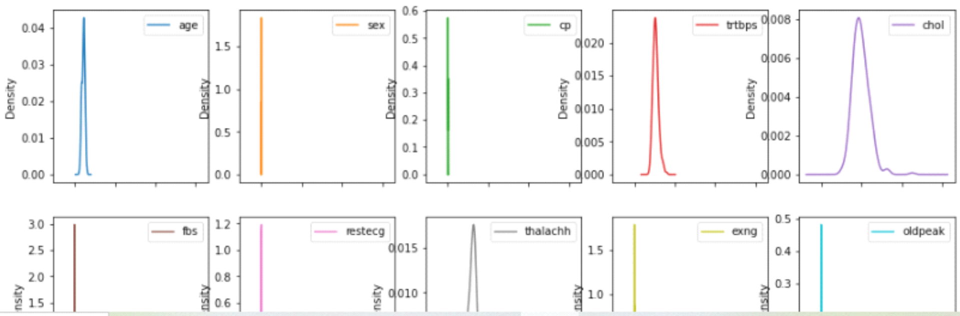
In [9]:
```
plt.figure()
dataset.plot(kind='box', subplots=True, layout=(3,5),figsize=(10,8))
plt.xlabel('rows')
plt.ylabel('data')
plt.title('box plot of heart dataset')
plt.show()
```

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



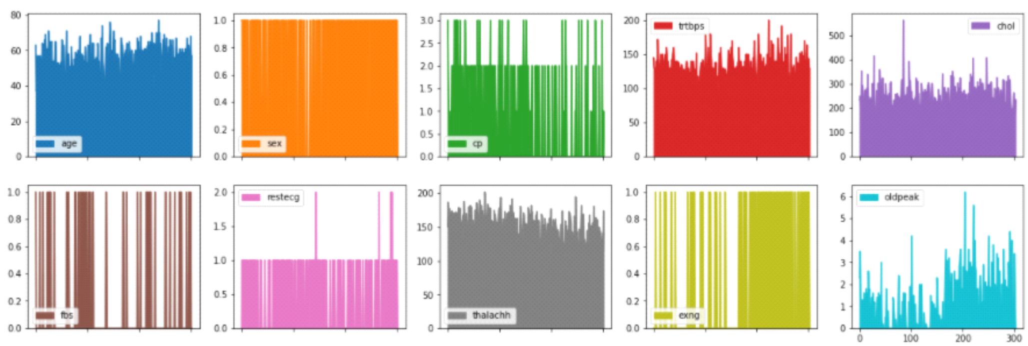
In [10]:
```python
plt.figure()
dataset.plot(kind='density', subplots=True, layout=(3,5),figsize=(15,10))
plt.xlabel('rows')
plt.ylabel('data')
plt.title('density plot of the dataset')
plt.show()
```

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

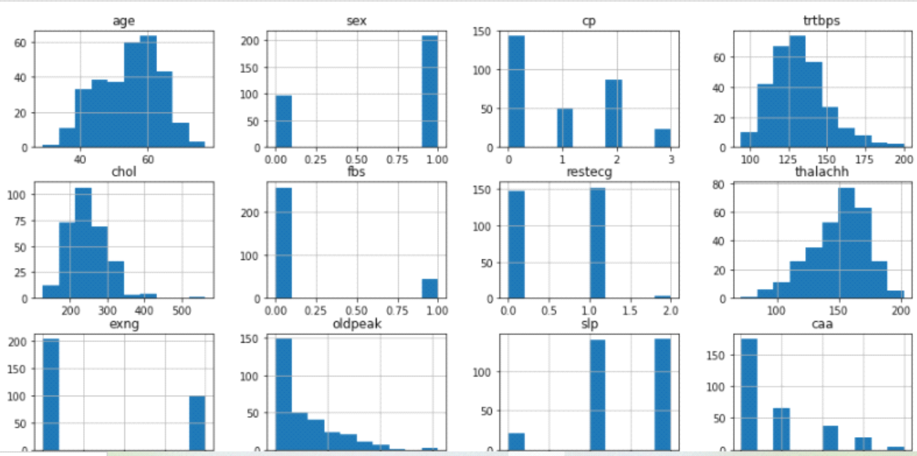


In [11]:
```python
plt.figure()
dataset.plot(kind='area', subplots=True, layout=(3,5),figsize=(20,10))
plt.xlabel('rows')
plt.ylabel('data')
plt.show()
```

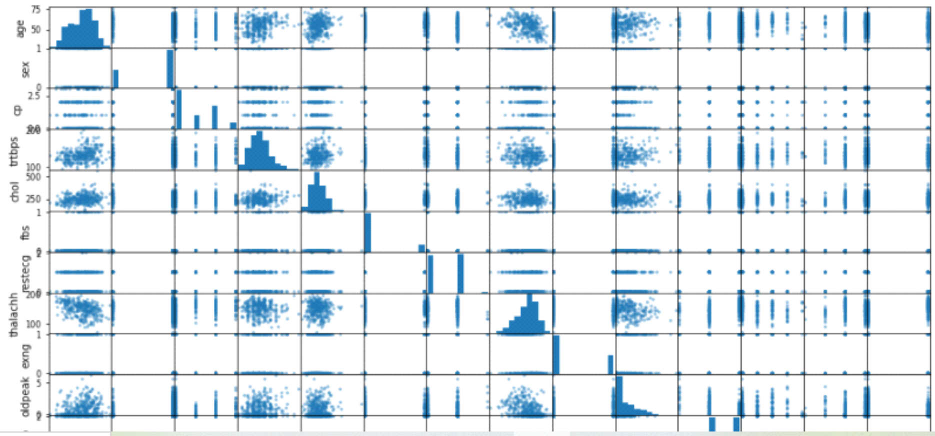
```
<Figure size 432x288 with 0 Axes>
```



In [12]:
```python
dataset.hist(figsize=(15,10))
plt.title('histogram')
plt.show()
```



```
In [13]: pd.plotting.scatter_matrix(dataset,figsize=(15,10))
         plt.title('scatter plot')
         plt.show()
```




```
In [14]: y=dataset['output']
         X=dataset.drop(['output'], axis=1)
```

```
In [15]: X.shape
```

```
Out[15]: (303, 13)
```

```
In [16]: X.dtypes
```

```
Out[16]: age           int64
         sex           int64
         cp            int64
         trtbps        int64
         chol          int64
         fbs           int64
         restecg       int64
         thalachh      int64
         exng          int64
         oldpeak     float64
         slp           int64
         caa           int64
         thall         int64
         dtype: object
```

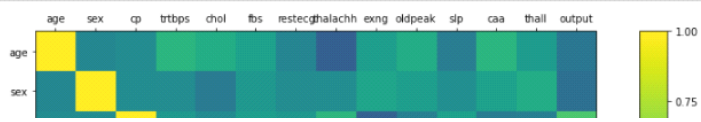
In [17]:
```
X.isna().sum()
```

Out[17]:
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
dtype: int64
```

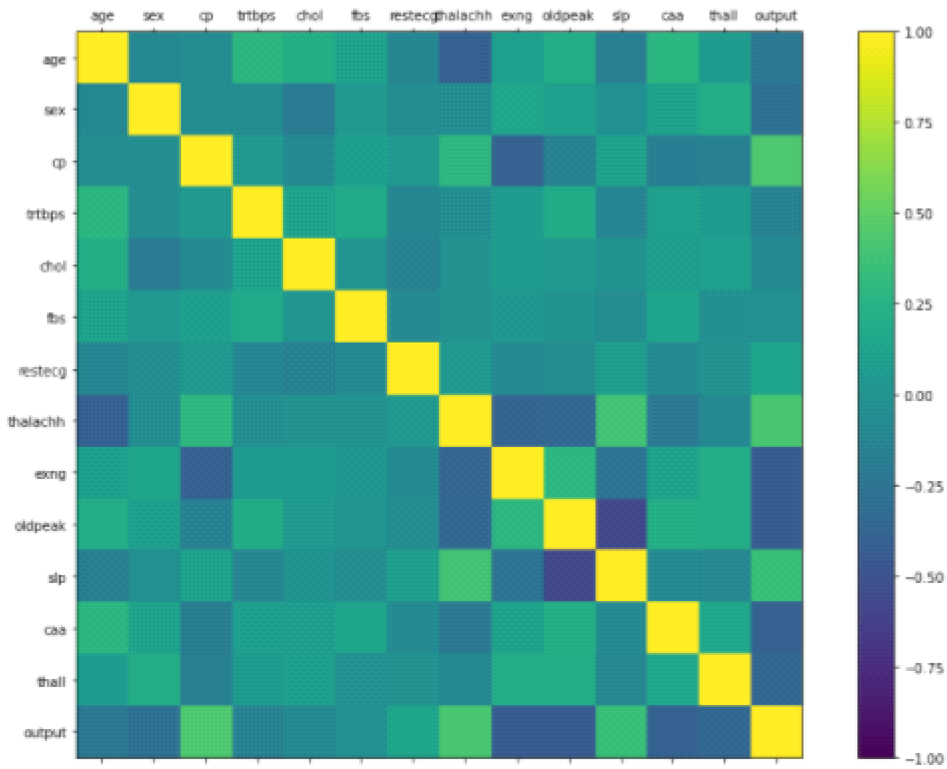
In [18]:
```
y.value_counts()
```

Out[18]:
```
1    165
0    138
Name: output, dtype: int64
```

In [21]:
```
correlations = dataset.corr(method='pearson')
names = ['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh','exng','oldpeak','slp','caa','thall','output']
fig = plt.figure(figsize=(15,10))
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,14,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```



```python
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```



In [22]:
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0, stratify=y)
```

In [23]:
```python
multinb_model = MultinomialNB()
multinb_model.fit(x_train,y_train)
y_pred_multinb = multinb_model.predict(x_test)
```

In [24]:
```python
precision = precision_score(y_test, y_pred_multinb, average='binary')
recall = recall_score(y_test, y_pred_multinb, average='binary')
f1score = f1_score(y_test, y_pred_multinb, average='binary')
print(precision)
print(recall)
print(f1score)
```

```
0.7878787878787878
0.7878787878787878
0.7878787878787878
```

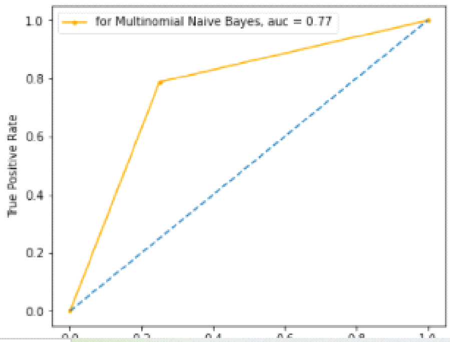
In [25]:
```python
print(classification_report(y_test,y_pred_multinb))
```

```
              precision    recall  f1-score   support

           0       0.75      0.75      0.75        28
           1       0.79      0.79      0.79        33

    accuracy                           0.77        61
   macro avg       0.77      0.77      0.77        61
weighted avg       0.77      0.77      0.77        61
```

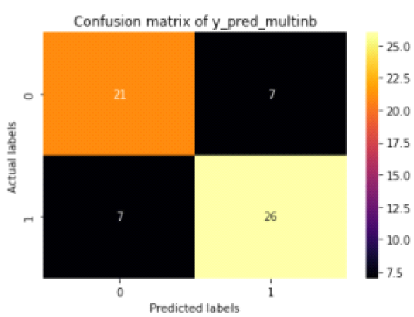
In [26]:
```
#plotting the roc curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,y_pred_multinb)
fpr, tpr, _ = roc_curve(y_test,y_pred_multinb)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, marker='.',color='orange',label="for Multinomial Naive Bayes, auc = %.2f"% roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [27]:
```
#plt.figure(figsize=(6, 4))
import seaborn as sns
sns.heatmap(confusion_matrix(y_test,y_pred_multinb) , annot = True,fmt='d',cmap="inferno")
print(confusion_matrix(y_test,y_pred_multinb))
plt.title('Confusion matrix of y_pred_multinb')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
#plt.savefig('confusion_matrix_dataset1_svm.png')
```

```
[[21  7]
 [ 7 26]]
```

Out[27]: `Text(33.0, 0.5, 'Actual labels')`



In [28]:
```
clf = svm.SVC(kernel='linear')
t = clf.fit(x_train, y_train)
y_pred_svm = t.predict(x_test)
```

In [29]:
```
precision_svm = precision_score(y_test, y_pred_svm, average='binary')
recall_svm = recall_score(y_test, y_pred_svm, average='binary')
f1score_svm = f1_score(y_test, y_pred_svm, average='binary')
print(precision)
print(recall)
print(f1score)
```

```
0.7878787878787878
0.7878787878787878
0.7878787878787878
```

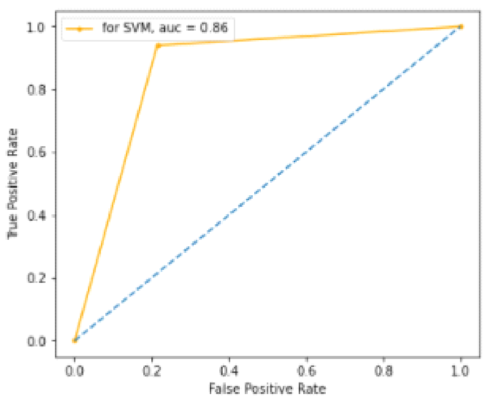
In [30]:
```
print(classification_report(y_test,y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        28
           1       0.84      0.94      0.89        33

    accuracy                           0.87        61
   macro avg       0.88      0.86      0.87        61
weighted avg       0.87      0.87      0.87        61
```

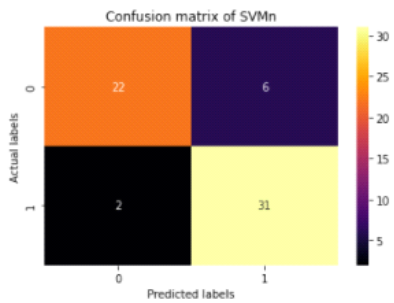
In [31]:
```
#plotting the roc curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,y_pred_svm)
fpr, tpr, _ = roc_curve(y_test,y_pred_svm)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, marker='.',color='orange',label="for SVM, auc = %.2f"% roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



```
In [32]: #plt.figure(figsize=(6, 4))
         import seaborn as sns
         sns.heatmap(confusion_matrix(y_test,y_pred_svm) , annot = True,fmt='d',cmap="inferno")
         print(confusion_matrix(y_test,y_pred_svm))
         plt.title('Confusion matrix of SVMn')
         plt.xlabel('Predicted labels')
         plt.ylabel('Actual labels')
         #plt.savefig('confusion_matrix_dataset1_svm.png')

         [[22  6]
          [ 2 31]]

Out[32]: Text(33.0, 0.5, 'Actual labels')
```



Conclusion: Despite the fact that the far-reaching independence assumptions are often inaccurate, the naive Bayes classifier has several properties that make it surprisingly useful in practice. In particular, the decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This helps alleviate problems stemming from the curse of dimensionality.

Experiment No: 05

Experiment Name: SVM using mushrooms dataset.

Objective: By SVM algorithm I find out the dataset of Mushroom. Here, I classify the dataset of value count, precision, recall etc.

Theory: SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs. Simply put, it does some extremely complex data transformations, then figures out how to seperate your data based on the labels or outputs you've defined.

Description:

Code + Output:

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np

        from sklearn.naive_bayes import MultinomialNB
        from sklearn import svm
        from sklearn.naive_bayes import GaussianNB

        from sklearn.metrics import precision_score
        from sklearn.metrics import recall_score
        from sklearn.metrics import f1_score


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

```
In [2]: dataset = pd.read_csv('mushrooms.csv')
        dataset.shape
```

```
Out[2]: (8124, 23)
```

```
In [23]: dataset.head(2)
```

Out[23]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	1	5	2	4	1	5	1	0	1	4	...	2	7	7	0	2	1	4	2	3
1	0	5	2	9	1	5	1	0	0	4	...	2	7	7	0	2	1	4	3	2

2 rows × 23 columns

```
In [4]: dataset.dtypes
```

```
Out[4]: class                       object
        cap-shape                   object
        cap-surface                 object
        cap-color                   object
        bruises                     object
        odor                        object
        gill-attachment             object
        gill-spacing                object
        gill-size                   object
        gill-color                  object
        stalk-shape                 object
        stalk-root                  object
        stalk-surface-above-ring    object
        stalk-surface-below-ring    object
        stalk-color-above-ring      object
        stalk-color-below-ring      object
        veil-type                   object
        veil-color                  object
        ring-number                 object
        ring-type                   object
        spore-print-color           object
        population                  object
        habitat                     object
        dtype: object
```

In [5]:
```
dataset.isnull().sum()
```

Out[5]:
```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

In [6]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

In [24]:
```
dataset['class'] = le.fit_transform(dataset['class'])
dataset['cap-shape'] = le.fit_transform(dataset['cap-shape'])
dataset['cap-surface'] = le.fit_transform(dataset['cap-surface'])
dataset['cap-color'] = le.fit_transform(dataset['cap-color'])
dataset['bruises'] = le.fit_transform(dataset['bruises'])
dataset['odor'] = le.fit_transform(dataset['cap-shape'])
dataset['gill-attachment'] = le.fit_transform(dataset['gill-attachment'])
dataset['gill-spacing'] = le.fit_transform(dataset['gill-spacing'])
dataset['gill-size'] = le.fit_transform(dataset['gill-size'])
dataset['gill-color'] = le.fit_transform(dataset['gill-color'])
dataset['stalk-shape'] = le.fit_transform(dataset['stalk-shape'])
dataset['stalk-root'] = le.fit_transform(dataset['stalk-root'])
dataset['stalk-surface-above-ring'] = le.fit_transform(dataset['stalk-surface-above-ring'])
dataset['stalk-surface-below-ring'] = le.fit_transform(dataset['stalk-surface-below-ring'])
dataset['stalk-color-above-ring'] = le.fit_transform(dataset['stalk-color-above-ring'])
dataset['stalk-color-below-ring'] = le.fit_transform(dataset['stalk-color-below-ring'])
dataset['veil-type'] = le.fit_transform(dataset['veil-type'])
dataset['veil-color'] = le.fit_transform(dataset['veil-color'])
dataset['ring-number'] = le.fit_transform(dataset['ring-number'])
dataset['ring-type'] = le.fit_transform(dataset['ring-type'])
dataset['spore-print-color'] = le.fit_transform(dataset['spore-print-color'])
dataset['population'] = le.fit_transform(dataset['population'])
dataset['habitat'] = le.fit_transform(dataset['habitat'])
dataset.head(2)
```

Out[24]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	1	5	2	4	1	5	1	0	1	4	...	2	7	7	0	2	1	4	2	3
1	0	5	2	9	1	5	1	0	0	4	...	2	7	7	0	2	1	4	3	2

2 rows × 23 columns

In [8]:
```
dataset.dtypes
```

Out[8]:
```
class                       int32
cap-shape                   int32
cap-surface                 int32
cap-color                   int32
bruises                     int32
odor                        int64
gill-attachment             int32
gill-spacing                int32
gill-size                   int32
gill-color                  int32
stalk-shape                 int32
stalk-root                  int32
stalk-surface-above-ring    int32
stalk-surface-below-ring    int32
stalk-color-above-ring      int32
```

```
stalk-surface-above-ring    int32
stalk-surface-below-ring    int32
stalk-color-above-ring      int32
```

```
stalk-color-below-ring      int32
veil-type                   int32
veil-color                  int32
ring-number                 int32
ring-type                   int32
spore-print-color           int32
population                  int32
habitat                     int32
dtype: object
```

In [9]:
```
y=dataset['class']
X=dataset.drop(['class'], axis=1)
```

In [10]:
```
y.value_counts()
```

Out[10]:
```
0    4208
1    3916
Name: class, dtype: int64
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0, stratify=y)
```

In [12]:
```
multinb_model = MultinomialNB()
multinb_model.fit(x_train,y_train)
y_pred_multinb = multinb_model.predict(x_test)
```

In [13]:
```
precision = precision_score(y_test, y_pred_multinb, average='binary')
recall = recall_score(y_test, y_pred_multinb, average='binary')
f1score = f1_score(y_test, y_pred_multinb, average='binary')
print(precision)
print(recall)
print(f1score)
```

```
0.9044368600682594
0.6768837803320562
0.7742878013148283
```

In [14]:
```
print(classification_report(y_test,y_pred_multinb))
```

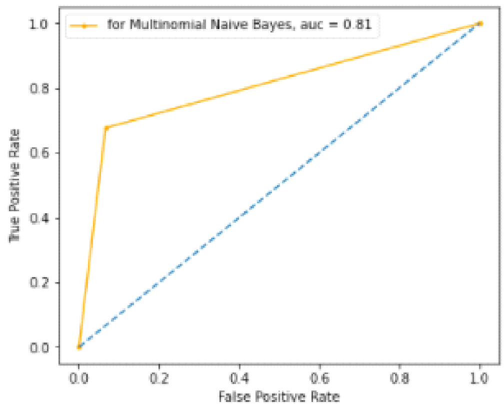
```
              precision    recall  f1-score   support

           0       0.76      0.93      0.84       842
           1       0.90      0.68      0.77       783

    accuracy                           0.81      1625
   macro avg       0.83      0.81      0.81      1625
weighted avg       0.83      0.81      0.81      1625
```

In [15]:
```
#plotting the roc curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,y_pred_multinb)
fpr, tpr, _ = roc_curve(y_test,y_pred_multinb)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, marker='.',color='orange',label="for Multinomial Naive Bayes, auc = %.2f"% roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

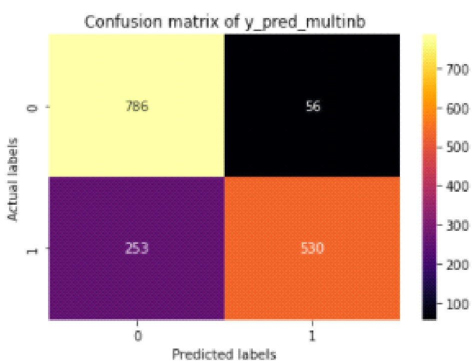
In [15]:
```python
#plotting the roc curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,y_pred_multinb)
fpr, tpr, _ = roc_curve(y_test,y_pred_multinb)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, marker='.',color='orange',label="for Multinomial Naive Bayes, auc = %.2f"% roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [16]:
```python
#plt.figure(figsize=(6, 4))
import seaborn as sns
sns.heatmap(confusion_matrix(y_test,y_pred_multinb) , annot = True,fmt='d',cmap="inferno")
print(confusion_matrix(y_test,y_pred_multinb))
plt.title('Confusion matrix of y_pred_multinb')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
#plt.savefig('confusion_matrix_dataset1_svm.png')
```

```
[[786  56]
 [253 530]]
```

Out[16]: `Text(33.0, 0.5, 'Actual labels')`



In [17]:
```
clf = svm.SVC(kernel='linear')
t = clf.fit(x_train, y_train)
y_pred_svm = t.predict(x_test)
```

In [18]:
```
precision_svm = precision_score(y_test, y_pred_svm, average='binary')
recall_svm = recall_score(y_test, y_pred_svm, average='binary')
f1score_svm = f1_score(y_test, y_pred_svm, average='binary')
print(precision)
print(recall)
print(f1score)
```

```
0.9044368600682594
0.6768837803320562
0.7742878013148283
```

In [19]:
```
print(classification_report(y_test,y_pred_svm))
```

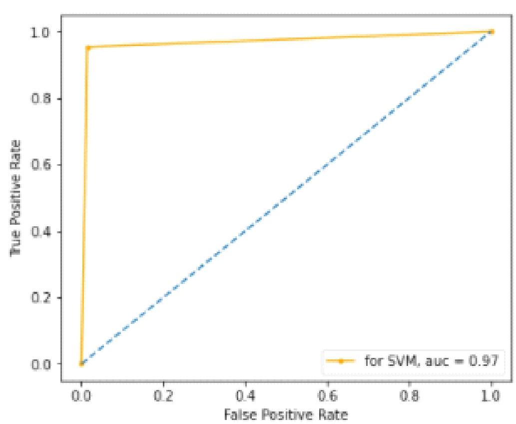
```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       842
           1       0.98      0.95      0.97       783

    accuracy                           0.97      1625
   macro avg       0.97      0.97      0.97      1625
weighted avg       0.97      0.97      0.97      1625
```

In [20]:
```
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,y_pred_svm)
fpr, tpr, _ = roc_curve(y_test,y_pred_svm)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, marker='.',color='orange',label="for SVM, auc = %.2f"% roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x1ed93612160>



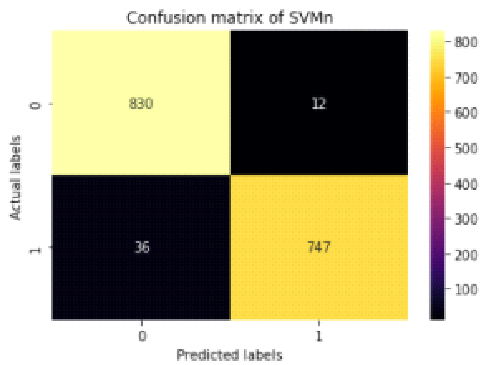
In [21]:

```
#plt.figure(figsize=(6, 4))
import seaborn as sns
sns.heatmap(confusion_matrix(y_test,y_pred_svm) , annot = True,fmt='d',cmap="inferno")
print(confusion_matrix(y_test,y_pred_svm))
plt.title('Confusion matrix of SVMn')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
#plt.savefig('confusion_matrix_dataset1_svm.png')
```

```
[[830  12]
 [ 36 747]]
```

Out[21]:

```
Text(33.0, 0.5, 'Actual labels')
```



Conclusion: A support vector machine is a supervised learning algorithm that sorts data into two categories. It is trained with a series of data already classified into two categories, building the model as it is initially trained. The task of an SVM algorithm is to determine which category a new data point belongs in. This makes SVM a kind of non-binary linear classifier.