machine learning all programmes

optimization technique

In [ ]:

```

```

In [3]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset
house_data = pd.read_csv('houseprice1.csv')

# Display the first few rows of the dataset
print(house_data.head())
```

```
   area  bedrooms  old    price
0   400       3.0    5  1050000
1   600       4.0   15   950000
2  3200       NaN   15  6500000
3  3600       3.0   30  5900000
4  4000       5.0    8  7600000
```

In [8]:

```
house_data.isnull().sum()
```

Out[8]:

```
area        0
bedrooms    1
old         0
price       0
dtype: int64
```

In [9]:

```
house_data['bedrooms']=house_data['bedrooms'].fillna(house_data.bedrooms.me
```

In [10]:

```
house_data.isnull().sum()
```

Out[10]:

```
area        0
bedrooms    0
old         0
price       0
dtype: int64
```

In [11]:

```
X = house_data[['bedrooms', 'area', 'old']]
y = house_data['price']
```

In [97]:

```
y
```

Out[97]:

```
0    1050000
1     950000
2    6500000
3    5900000
4    7600000
5    8100000
Name: price, dtype: int64
```

In [12]:

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

In [13]:

```
model = LinearRegression()
```

In [14]:

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

Out[14]:

```
LinearRegression()
```

In [24]:

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

Out[24]:

```
array([4491022.44389027, 3424314.21446383])
```

In [16]:

```
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 8981433145627.158
```

In [19]:

```
model.score(X_test,y_test)
```

Out[19]:

```
-3591.573258250863
```

In [ ]:

In [26]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sco

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = mean_squared_error(y_test, predictions, squared=False)
print("Root Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 2957668.32917705
Root Mean Squared Error: 2996903.926659505
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

LinearRegression using houseprice1 data

In [ ]:

In [86]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [87]:

```
df=pd.read_csv("houseprice1.csv")
```

In [88]:

```
df.head()
```

Out[88]:

	area	bedrooms	old	price
0	400	3.0	5	1050000
1	600	4.0	15	950000
2	3200	NaN	15	6500000
3	3600	3.0	30	5900000
4	4000	5.0	8	7600000

In [89]:

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

Out[89]:

```
area        0
bedrooms    1
old         0
price       0
dtype: int64
```

In [90]:

```
df['bedrooms']=df['bedrooms'].fillna(df.bedrooms.mean())
```

In [94]:

```
x=df[['area','bedrooms','old']]
y=df['price']
```

```
In [96]: y
```

```
Out[96]: 0    1050000
         1     950000
         2    6500000
         3    5900000
         4    7600000
         5    8100000
         Name: price, dtype: int64
```

```
In [98]: x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_st
```

```
In [99]: len(x_train)
```

```
Out[99]: 4
```

```
In [100]: len(x_test)
```

```
Out[100]: 2
```

model=LinearRegression()

```
In [101]: model=LinearRegression()
```

```
In [103]: model.fit(x_train,y_train)
```

```
Out[103]: LinearRegression()
```

```
In [106]: predict=model.predict(x_test)
```

```
In [105]: model.score(x_test,y_test)
```

```
Out[105]: -3591.5732582505257
```

```
In [110]: mse=mean_squared_error(y_test,predict)
          mse
```

```
Out[110]: 8981433145626.314
```

```
In [109]: mae=mean_absolute_error(y_test,predict)
          mae
```

```
Out[109]: 2957668.329176908
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In [ ]:

Logitic Regression using insurance data

In [ ]:

In [ ]:

In [111]:

```
from sklearn.linear_model import LogisticRegression
```

In [118]:

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

In [119]:

```
df.head()
```

Out[119]:

	age	bought_insurance
0	40	1
1	12	0
2	44	1
3	33	0
4	32	0

In [139]:

```
x=df[['age']]
y=df[['bought_insurance']]
```

In [140]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_st
```

In [141]:

```
print(len(x_train))
print(len(y_train))
print(len(x_test))
print(len(y_test))
```

```
24
24
11
11
```

In [142]:

```
model=LogisticRegression()
```

In [143]:

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

```
C:\Users\Nitheesh\anaconda3\lib\site-packages\sklearn\utils\validation.py:
63: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example usi
ng ravel().
  return f(*args, **kwargs)
```

Out[143]:

```
LogisticRegression()
```

In [115]:

```
predict=model.predict(x_test)
```

In [144]:

```
model.score(x_test,y_test)
```

Out[144]:

```
0.5454545454545454
```

In [145]:

```
x_test
```

Out[145]:

	age
26	33
13	46
24	31
21	28
15	48
29	36
19	53
12	45
8	88
16	49
9	39

In [146]:

```
model.predict(x_test)
```

Out[146]:

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Logistic Regression in images

In [ ]:

In [ ]:

In [157]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
```

In [159]:

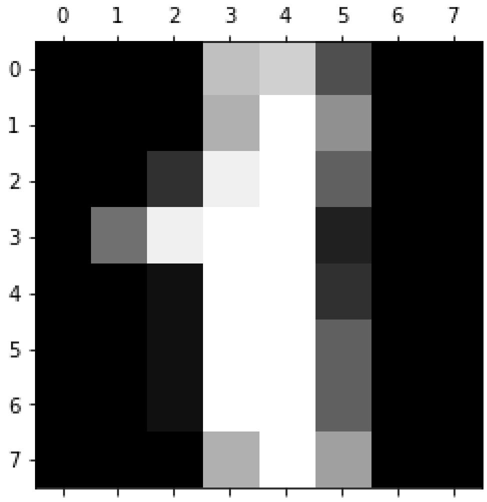
```
df=load_digits()
```

In [163]:

```
plt.matshow(digits.images[1])
```

Out[163]:

```
<matplotlib.image.AxesImage at 0x1b319f3d550>
```

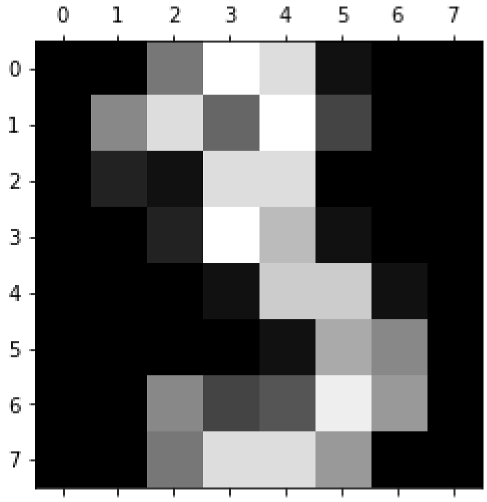


In [166]:

```
plt.matshow(digits.images[3])
```

Out[166]:

```
<matplotlib.image.AxesImage at 0x1b31a0351f0>
```



In [167]:

```
dir(digits)
```

Out[167]:

```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_na
mes']
```

In [171]:
```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,tr
```

In [172]:
```
model=LogisticRegression()
```

In [173]:
```
model.fit(x_train,y_train)
```

```
C:\Users\Nitheesh\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[173]:
```
LogisticRegression()
```

In [188]:
```
model.predict(digits.data[:5])
```

Out[188]:
```
array([0, 1, 2, 3, 4])
```

In [189]:
```
y_predict=model.predict(x_test)
```

In [190]:
```
from sklearn.metrics import confusion_matrix
```

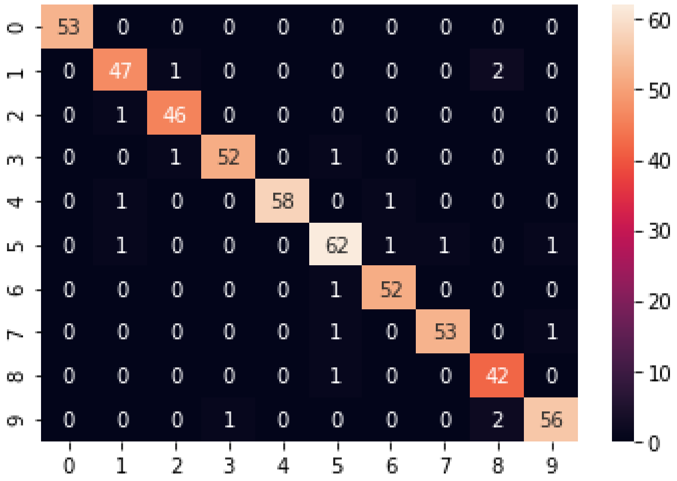
In [191]:
```
cm=confusion_matrix(y_test,y_predict)
```

In [192]:
```
cm
```

Out[192]:
```
array([[53,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 47,  1,  0,  0,  0,  0,  0,  2,  0],
       [ 0,  1, 46,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  1, 52,  0,  1,  0,  0,  0,  0],
       [ 0,  1,  0,  0, 58,  0,  1,  0,  0,  0],
       [ 0,  1,  0,  0,  0, 62,  1,  1,  0,  1],
       [ 0,  0,  0,  0,  0,  1, 52,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  1,  0, 53,  0,  1],
       [ 0,  0,  0,  0,  0,  1,  0,  0, 42,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  2, 56]], dtype=int64)
```

```
In [195]: sns.heatmap(cm,annot=True)
```

```
Out[195]: <AxesSubplot:>
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

implement cluster

```
In [ ]:
```

```
In [196]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn.cluster import KMeans
```

```
In [198]: df=pd.read_csv('clus.csv')
```

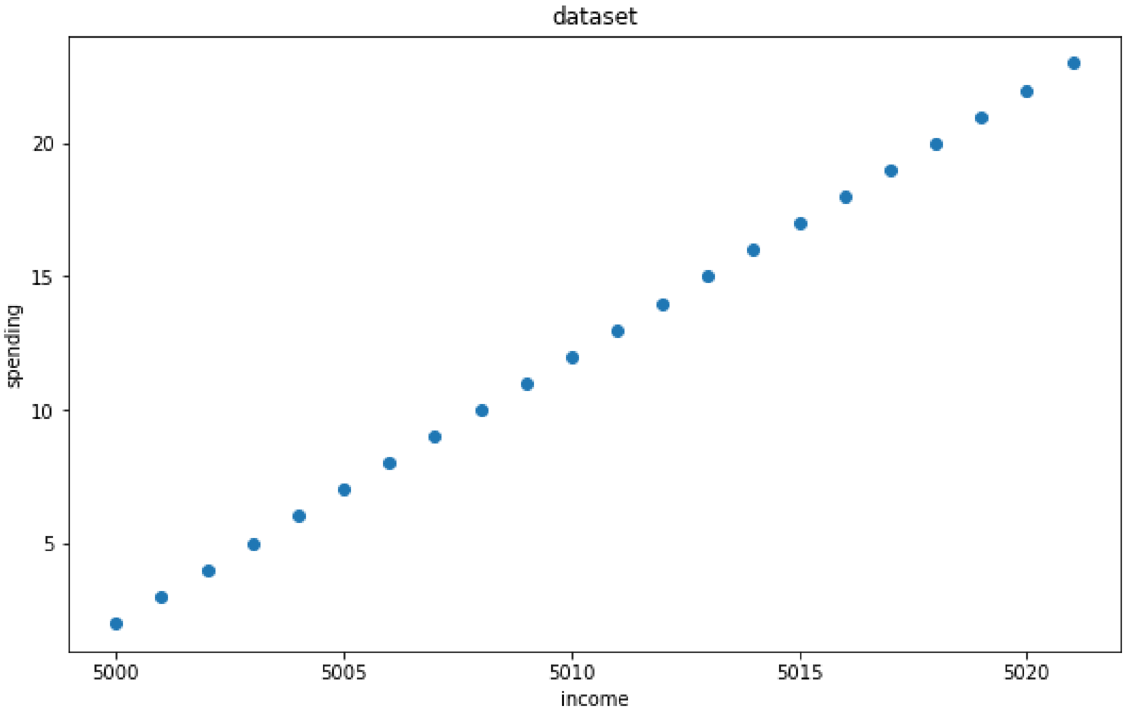
```
In [200]: df.head()
```

Out[200]:

	income	spending
0	5000	2
1	5001	3
2	5002	4
3	5003	5
4	5004	6

In [201]:
```
plt.figure(figsize=(10,6))
plt.scatter(df['income'],df['spending'])
plt.xlabel('income')
plt.ylabel('spending')
plt.title('dataset')
```

Out[201]:
```
Text(0.5, 1.0, 'dataset')
```



In [205]:
```
a=[]
for i in range(1,5):
    kmeans=KMeans(n_clusters=i, init='random',random_state=42)
    kmeans.fit(df)
    a.append(kmeans.inertia_)
```

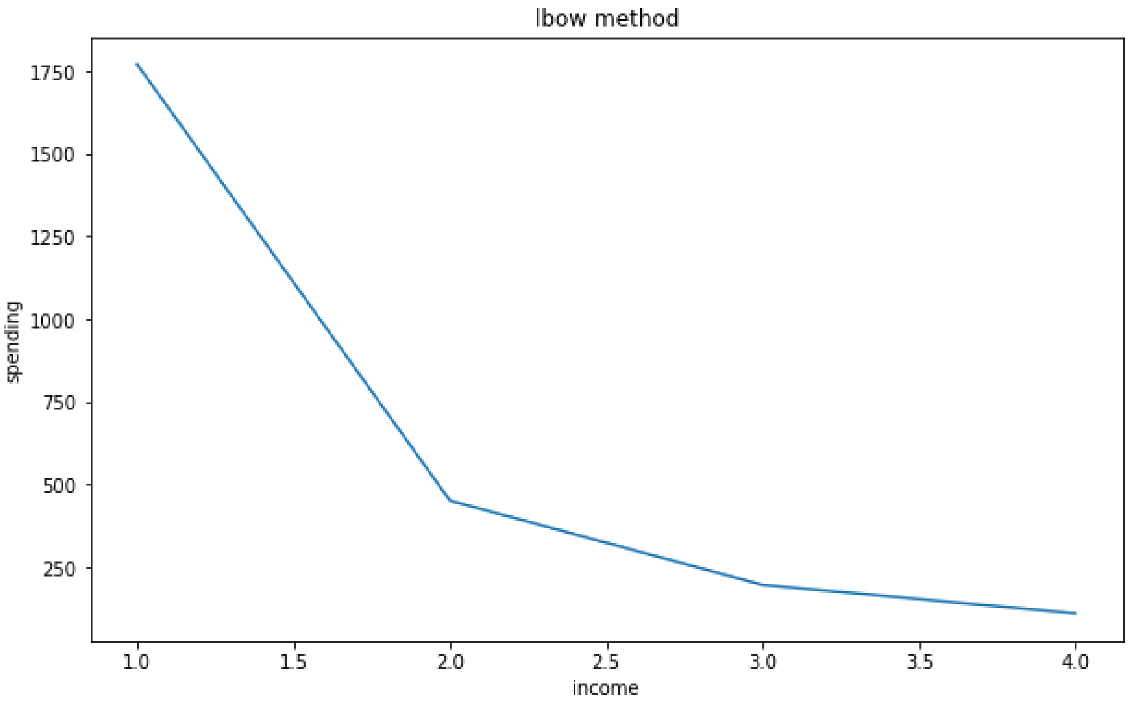
```
C:\Users\Nitheesh\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:8
81: UserWarning: KMeans is known to have a memory leak on Windows with MK
L, when there are less chunks than available threads. You can avoid it by
setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [206]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,5),a)
plt.xlabel('income')
plt.ylabel('spending')
plt.title('lbow method')
```

Out[206]:

```
Text(0.5, 1.0, 'lbow method')
```



In [220]:

```
kmeans=KMeans(n_clusters=2,random_state=42)
kmeans.fit(df)
pred=kmeans.predict(df)
```

In [221]:

```
df['cluster']=pd.DataFrame(pred)
```

In [223]:

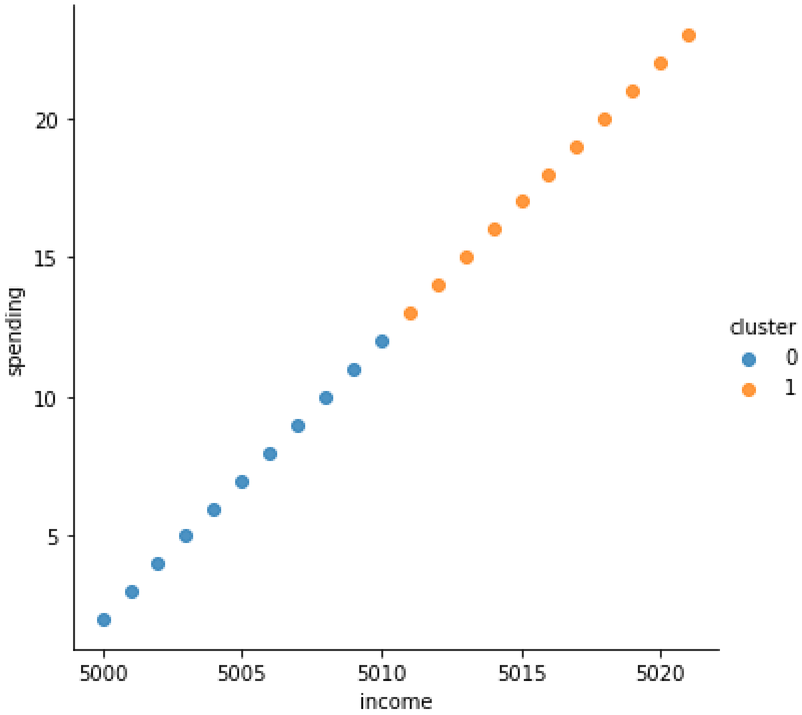
```
df.head()
```

Out[223]:

	income	spending	cluster
0	5000	2	0
1	5001	3	0
2	5002	4	0
3	5003	5	0
4	5004	6	0

In [226]:
```
sns.lmplot(x='income',y='spending',data=df,hue='cluster',fit_reg=False)
```

Out[226]:
```
<seaborn.axisgrid.FacetGrid at 0x1b31d64ca30>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

data cleaning using normal way

In [ ]:

In [257]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [258]:
```
df=pd.read_csv("Data_preprocessing.csv")
```

In [259]:
```
df.head()
```

Out[259]:

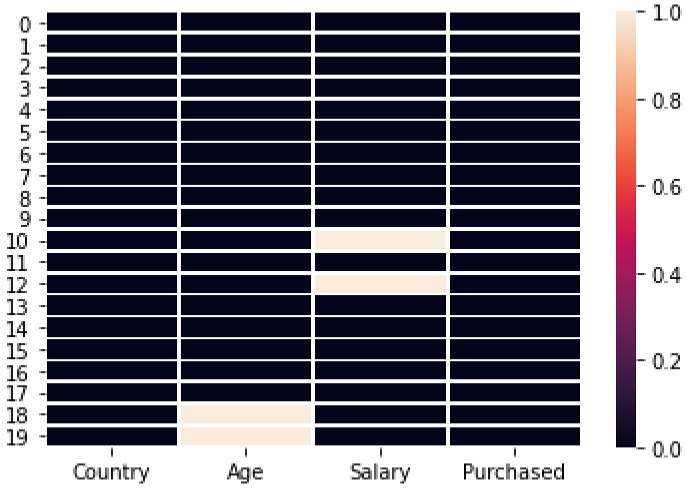
	Country	Age	Salary	Purchased
0	Australia	27.0	52000.0	Yes
1	Russia	25.0	49000.0	Yes
2	India	30.0	56000.0	No
3	Russia	29.0	54000.0	No
4	India	35.0	58000.0	Yes

In [260]:
```
df.isnull().sum()
```

Out[260]:
```
Country      0
Age          2
Salary       2
Purchased    0
dtype: int64
```

In [261]:
```
sns.heatmap(df.isnull(),linewidth=0.5)
```

Out[261]:
```
<AxesSubplot:>
```



In [262]:
```
df['Age']=df['Age'].fillna(df.Age.mean())
```

In [263]:
```
df.isnull().sum()
```

Out[263]:
```
Country      0
Age          0
Salary       2
Purchased    0
dtype: int64
```

In [264]:
```
df.fillna(df.mean(),inplace=True)
```

In [265]:
```
df.isnull().sum()
```

Out[265]:
```
Country      0
Age          0
Salary       0
Purchased    0
dtype: int64
```

In [266]:
```
df.duplicated().sum()
```

Out[266]:
```
0
```

In [267]:
```
df.drop_duplicates()
```

Out[267]:

	Country	Age	Salary	Purchased
0	Australia	27.000000	52000.000000	Yes
1	Russia	25.000000	49000.000000	Yes
2	India	30.000000	56000.000000	No
3	Russia	29.000000	54000.000000	No
4	India	35.000000	58000.000000	Yes
5	Russia	36.000000	60000.000000	Yes
6	India	37.000000	61000.000000	Yes
7	Australia	41.000000	67000.000000	Yes
8	Australia	38.000000	65000.000000	No
9	Russia	39.000000	66000.000000	No
10	Australia	40.000000	65333.333333	Yes
11	India	40.000000	72000.000000	No
12	Russia	40.000000	65333.333333	Yes
13	India	44.000000	72000.000000	No
14	Russia	48.000000	79000.000000	Yes
15	India	47.000000	78000.000000	Yes
16	Australia	50.000000	83000.000000	No
17	Russia	52.000000	85000.000000	No
18	India	38.777778	61000.000000	No
19	Australia	38.777778	58000.000000	No

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

using the imputer to fill the null values

In [ ]:

In [339]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
```

In [340]:

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

In [341]:

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

Out[341]:

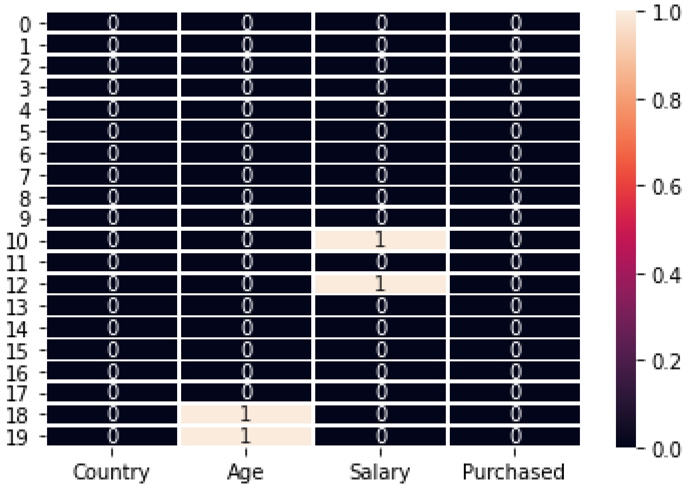
```
Country      0
Age          2
Salary       2
Purchased    0
dtype: int64
```

In [342]:

```
sns.heatmap(df.isnull(),annot=True,linewidth=0.5)
```

Out[342]:

```
<AxesSubplot:>
```



In [343]:

```
imputer=SimpleImputer(strategy='mean')
```

In [344]:

```
x=df.iloc[:,0:3].values
y=df.iloc[:,3:4].values
```

In [345]:

```
imputer.fit(x[:,1:3])
```

Out[345]:

```
SimpleImputer()
```

In [346]:

```
x[:,1:3]=imputer.transform(x[:,1:3])
```

In [347]:

```
x
```

Out[347]:

```
array([['Australia', 27.0, 52000.0],
       ['Russia', 25.0, 49000.0],
       ['India', 30.0, 56000.0],
       ['Russia', 29.0, 54000.0],
       ['India', 35.0, 58000.0],
       ['Russia', 36.0, 60000.0],
       ['India', 37.0, 61000.0],
       ['Australia', 41.0, 67000.0],
       ['Australia', 38.0, 65000.0],
       ['Russia', 39.0, 66000.0],
       ['Australia', 40.0, 65333.333333333336],
       ['India', 40.0, 72000.0],
       ['Russia', 40.0, 65333.333333333336],
       ['India', 44.0, 72000.0],
       ['Russia', 48.0, 79000.0],
       ['India', 47.0, 78000.0],
       ['Australia', 50.0, 83000.0],
       ['Russia', 52.0, 85000.0],
       ['India', 38.77777777777778, 61000.0],
       ['Australia', 38.77777777777778, 58000.0]], dtype=object)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

categorical data cleaning using the LabelEncoder

In [ ]:

In [348]:

```
from sklearn.preprocessing import LabelEncoder
```

In [349]:

```
label=LabelEncoder()
```

In [350]:

```
x[:,0]=label.fit_transform(x[:,0])
```

In [351]:

```
x
```

Out[351]:

```
array([[0, 27.0, 52000.0],
       [2, 25.0, 49000.0],
       [1, 30.0, 56000.0],
       [2, 29.0, 54000.0],
       [1, 35.0, 58000.0],
       [2, 36.0, 60000.0],
       [1, 37.0, 61000.0],
       [0, 41.0, 67000.0],
       [0, 38.0, 65000.0],
       [2, 39.0, 66000.0],
       [0, 40.0, 65333.333333333336],
       [1, 40.0, 72000.0],
       [2, 40.0, 65333.333333333336],
       [1, 44.0, 72000.0],
       [2, 48.0, 79000.0],
       [1, 47.0, 78000.0],
       [0, 50.0, 83000.0],
       [2, 52.0, 85000.0],
       [1, 38.77777777777778, 61000.0],
       [0, 38.77777777777778, 58000.0]], dtype=object)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

OneHotEncoder using

In [ ]:

In [352]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [354]:

```
label=OneHotEncoder(categories='auto',sparse=False)
x=label.fit_transform(x)
x=x[:,1:]
x
```

Out[354]:

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Vectorization concepts

In [ ]:

In [356]:

```
import numpy as np
import time
```

In [358]:

```
x=np.random.rand(1000000)
y=np.random.rand(1000000)
t=time.time()
z=np.dot(x,y)
duration=(time.time()-t)*1000
print("numpy operatio:",z,"duration:",duration)
z=0
t=time.time()
for i in range(1000000):
    z=z+x[i]*y[i]
duration=(time.time()-t)*1000
print("manual operation:",z,"duration:",duration)
```

```
numpy operatio: 249932.1700324327 duration: 2.0022392272949219
manual operation: 249932.1700324348 duration: 787.0988845825195
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Normalization implement

In [4]:

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
```

In [5]:

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

In [6]:

```
df
```

Out[6]:

	empid	age	salary	job_quit
0	emp1	20	25000	yes
1	emp2	50	50000	no
2	emp3	40	35000	no
3	emp4	30	30000	yes
4	emp5	60	70000	yes
5	emp6	35	30000	no

In [10]:

```
scaler=MinMaxScaler()
```

In [15]:

```
scaler.fit(df[['age','salary']])
```

Out[15]:

```
MinMaxScaler()
```

In [16]:

```
new=scaler.transform(df[['age','salary']])
```

In [17]:

```
new
```

Out[17]:

```
array([[0.        , 0.        ],
       [0.75      , 0.55555556],
       [0.5       , 0.22222222],
       [0.25      , 0.11111111],
       [1.        , 1.        ],
       [0.375     , 0.11111111]])
```

In [18]:

```
df['new_age']=new[:,0]
df['new_salary']=new[:,1]
```

In [19]:

```
df
```

Out[19]:

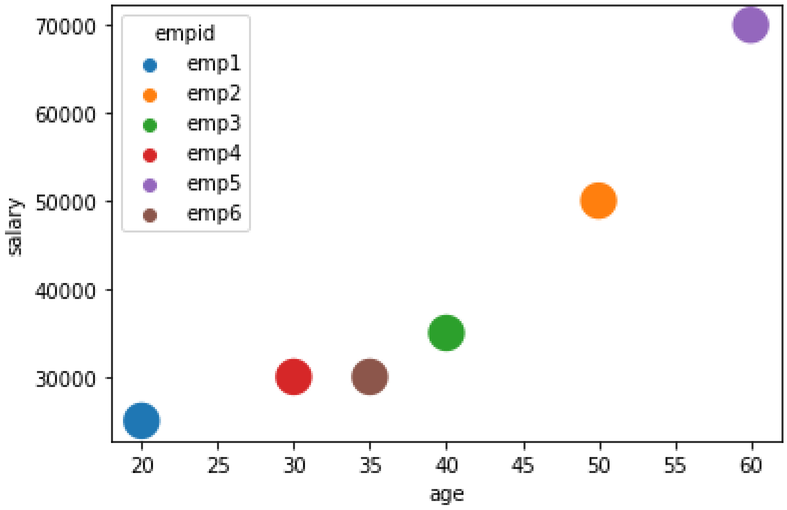
	empid	age	salary	job_quit	new_age	new_salary
0	emp1	20	25000	yes	0.000	0.000000
1	emp2	50	50000	no	0.750	0.555556
2	emp3	40	35000	no	0.500	0.222222
3	emp4	30	30000	yes	0.250	0.111111
4	emp5	60	70000	yes	1.000	1.000000
5	emp6	35	30000	no	0.375	0.111111

In [23]:

```
import seaborn as sns
sns.scatterplot(x="age",y="salary",data=df,hue=df.empid,s=400)
```

Out[23]:

```
<AxesSubplot:xlabel='age', ylabel='salary'>
```

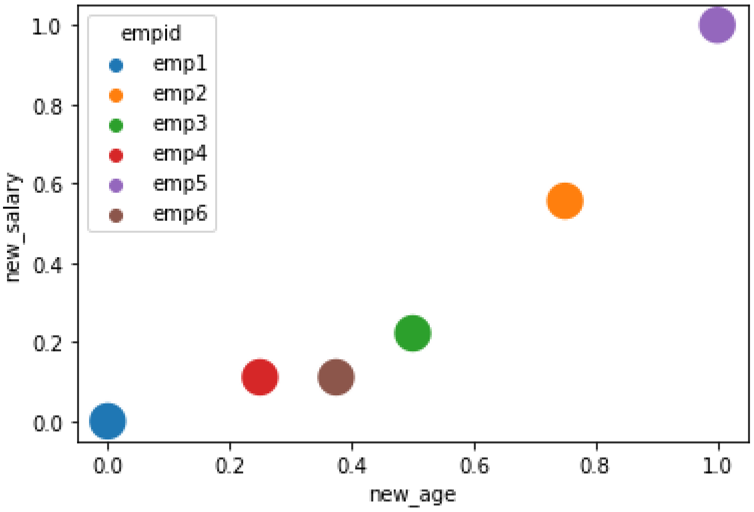


In [26]:

```
sns.scatterplot(x="new_age",y="new_salary",data=df,hue='empid',s=400)
```

Out[26]:

```
<AxesSubplot:xlabel='new_age', ylabel='new_salary'>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

standardization implement

```
In [ ]:
```

```
In [29]: from sklearn.preprocessing import StandardScaler
```

```
In [30]: df=pd.read_csv("employee.csv")
```

```
In [31]: model=StandardScaler()
```

```
In [38]: new=model.fit_transform(df[['age','salary']])
```

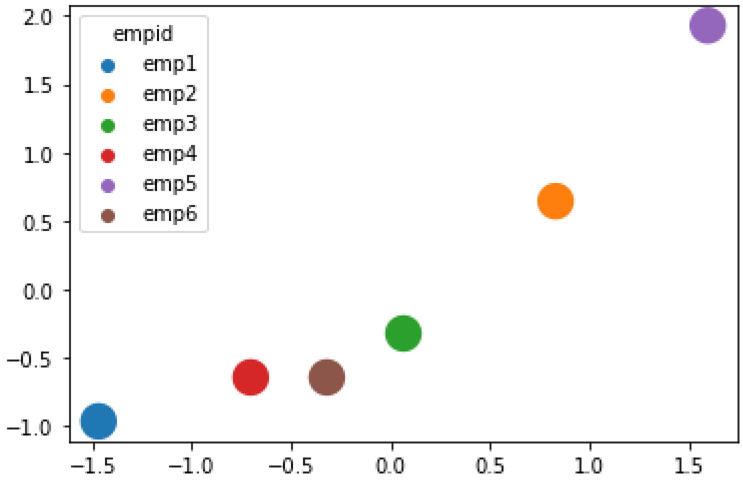
```
In [39]: new
```

```
Out[39]: array([[-1.4694161 , -0.96490128],
                [ 0.83053953,  0.64326752],
                [ 0.06388766, -0.32163376],
                [-0.70276422, -0.64326752],
                [ 1.59719141,  1.92980256],
                [-0.31943828, -0.64326752]])
```

```
In [42]: sns.scatterplot(new[:,0],new[:,1], hue="empid",data=df,s=400)
```

```
C:\Users\Nitheesh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: F
utureWarning: Pass the following variables as keyword args: x, y. From ver
sion 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or mis
interpretation.
  warnings.warn(
```

```
Out[42]: <AxesSubplot:>
```



```
In [45]: df['new_age']=new[:,0]
         df['new_salary']=new[:,1]
```

In [46]:
```
df
```

Out[46]:

	empid	age	salary	job_quit	new_age	new_salary
0	emp1	20	25000	yes	-1.469416	-0.964901
1	emp2	50	50000	no	0.830540	0.643268
2	emp3	40	35000	no	0.063888	-0.321634
3	emp4	30	30000	yes	-0.702764	-0.643268
4	emp5	60	70000	yes	1.597191	1.929803
5	emp6	35	30000	no	-0.319438	-0.643268

In [ ]:
```

```