Atma Ram Sanatan Dharma College
University of Delhi

MACHINE LEARNING
Practical

SUBMITTED BY- SURAJ RAI
ROLL NO. - 21/18102
CORSE - B.Sc(H) COMPUTER SCIENC
SUBMITTED TO- Dr. UMA OJHA
DEPARTMENT OF COMPUTER SCIENCE

index

11	Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting, or multiplying two matrices.	
12	Implement a Linear Regression problem.	
13	Perform Linear Regression based on multiple features.	
14	Implement a Classification/Logistic Regression problem.	
15	Use some function for regularization of dataset based on problem 14.	
16	Use some function for neural networks, like Stochastic Gradient Descent or Backpropagation to predict the value of a variable based on the dataset of problem 14.	

PRACTICAL-1

Objective :

Perform elementary mathematical operations like addition, multiplication, division, and exponentiation.

Code and Output

```
1 + 2
```

```
3
```

```
1 * 2
```

```
2
```

```
1 / 2
```

```
0.5
```

```
2 ** 3
```

```
8
```

PRACTICAL-2

Objective :

Perform elementary logical operations (like OR, AND, Checking for Equality, NOT, XOR).

Code and Output

```
True or False
```

```
True
```

```
True and False
```

```
False
```

```
not False
```

```
True
```

```
True ^ False
```

```
True
```

```
True ^ True
```

```
False
```

PRACTICAL-3

Objective :

Create, initialize, and display simple variables and simple strings and use simple formatting for variables.

Code and Output

```
var_a = 5.4
print(f"Value of var_a is {var_a}")
```

```
Value of var_a is 5.4
```

PRACTICAL-4

Objective :

Create/Define single dimension/multidimensional arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

Code and Output

```
import numpy as np
np.ones((4, 4))
```

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

```
np.zeros((4, 4))
```

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

```
np.random.rand(16).reshape(4, 4)
```

```
array([[0.0606613 , 0.62463632, 0.39359752, 0.26024275],
       [0.96898964, 0.47221527, 0.60819777, 0.88253582],
       [0.20901817, 0.19255731, 0.86897655, 0.51461569],
       [0.68598813, 0.14152689, 0.3721947 , 0.74407958]])
```

```
from scipy.linalg import block_diag
block_diag(1, 2, 3, 4)
```

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]], dtype=int32)
```

PRACTICAL-5

Objective :

Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.

Code and Output

```
import numpy as np
mat = np.matrix([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
mat
```

```
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

```
mat.size
```

```
9
```

```
mat.shape
```

```
(3, 3)
```

```
mat[1,].size
```

```
3
```

```
mat[:,2].size
```

```
3
```

```
np.savetxt('mat.txt', mat)
```

```
!type mat.txt
```

```
1.0000000000000000000e+00 2.0000000000000000000e+00
3.0000000000000000000e+00
4.0000000000000000000e+00 5.0000000000000000000e+00
6.0000000000000000000e+00
7.0000000000000000000e+00 8.0000000000000000000e+00
9.0000000000000000000e+00
```

```python
np.loadtxt('mat.txt')
```

```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

```python
print(dir())
```

```
['A', 'B', 'In', 'Out', '_', '_10', '_11', '_12', '_13',
'_17', '_18', '_19', '_2', '_20', '_21', '_22', '_23', '_24',
'_25', '_26', '_27', '_28', '_29', '_3', '_30', '_33', '_34',
'_36', '_39', '_40', '_41', '_42', '_43', '_44', '_45', '_46',
'_47', '_48', '_49', '_5', '_50', '_51', '_53', '_55', '_56',
'_58', '_59', '_6', '_60', '_61', '_67', '_68', '_69', '_7',
'_70', '_71', '_72', '_73', '_74', '_75', '_77', '_78', '_8',
'_81', '_9', '__', '___', '__builtin__', '__builtins__',
'__doc__', '__loader__', '__name__', '__package__',
'__spec__', '_dh', '_exit_code', '_i', '_i1', '_i10', '_i11',
'_i12', '_i13', '_i14', '_i15', '_i16', '_i17', '_i18',
'_i19', '_i2', '_i20', '_i21', '_i22', '_i23', '_i24', '_i25',
'_i26', '_i27', '_i28', '_i29', '_i3', '_i30', '_i31', '_i32',
'_i33', '_i34', '_i35', '_i36', '_i37', '_i38', '_i39', '_i4',
'_i40', '_i41', '_i42', '_i43', '_i44', '_i45', '_i46',
'_i47', '_i48', '_i49', '_i5', '_i50', '_i51', '_i52', '_i53',
'_i54', '_i55', '_i56', '_i57', '_i58', '_i59', '_i6', '_i60',
'_i61', '_i62', '_i63', '_i64', '_i65', '_i66', '_i67',
'_i68', '_i69', '_i7', '_i70', '_i71', '_i72', '_i73', '_i74',
'_i75', '_i76', '_i77', '_i78', '_i79', '_i8', '_i80', '_i81',
'_i82', '_i9', '_ih', '_ii', '_iii', '_oh', 'block_diag',
'exit', 'get_ipython', 'mat', 'np', 'quit', 'var_a']
```

```python
print(dir(mat))
```

```
['A', 'A1', 'H', 'I', 'T', '__abs__', '__add__', '__and__',
'__annotations__', '__array__', '__array_finalize__',
'__array_function__', '__array_interface__',
'__array_prepare__', '__array_priority__', '__array_struct__',
'__array_ufunc__', '__array_wrap__', '__bool__', '__class__',
'__class_getitem__', '__complex__', '__contains__',
'__copy__', '__deepcopy__', '__delattr__', '__delitem__',
'__dict__', '__dir__', '__divmod__', '__dlpack__',
'__dlpack_device__', '__doc__', '__eq__', '__float__',
'__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__iand__',
'__ifloordiv__', '__ilshift__', '__imatmul__', '__imod__',
'__imul__', '__index__', '__init__', '__init_subclass__',
'__int__', '__invert__', '__ior__', '__ipow__', '__irshift__',
'__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__',
'__len__', '__lshift__', '__lt__', '__matmul__', '__mod__',
'__module__', '__mul__', '__ne__', '__neg__', '__new__',
'__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
```

```
'__rfloordiv__', '__rlshift__', '__rmatmul__', '__rmod__',
'__rmul__', '__ror__', '__rpow__', '__rrshift__',
'__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__setitem__', '__setstate__', '__sizeof__',
'__str__', '__sub__', '__subclasshook__', '__truediv__',
'__xor__', '_align', '_collapse', '_getitem', 'all', 'any',
'argmax', 'argmin', 'argpartition', 'argsort', 'astype',
'base', 'byteswap', 'choose', 'clip', 'compress', 'conj',
'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum', 'data',
'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags',
'flat', 'flatten', 'getA', 'getA1', 'getH', 'getI', 'getT',
'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max',
'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero',
'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat',
'reshape', 'resize', 'round', 'searchsorted', 'setfield',
'setflags', 'shape', 'size', 'sort', 'squeeze', 'std',
'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile',
'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']
```

PRACTICAL-6

Objective :

Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

Code and Output

```
import numpy as np
```

```
A = np.array([0, 1, -1, 0]).reshape(2, 2)
A
```

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

```
B = np.array([1, 2, 3, 4][::-1]).reshape(2, 2)
B
```

```
array([[4, 3],
       [2, 1]])
```

```
A + B
```

```
array([[4, 4],
       [1, 1]])
```

```
A * B
```

```
array([[ 0,  3],
       [-2,  0]])
```

```
A / B
```

```
array([[ 0.        ,  0.33333333],
       [-0.5       ,  0.        ]])
```

```
A ** B
```

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

```
np.logical_or(A, B)
```

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

```
np.logical_and(A, B)
```

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

```
np.logical_not(A, B)
```

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

```
np.logical_xor(A, B)
```

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

```
np.equal(A, B)
```

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

```
np.equal(A, A)
```

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

```
B[1, ]
```

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

```
A[:,0]
```

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

PRACTICAL-7

Objective :

Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, editing/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

Code and Output

```
import numpy as np
```

```
A = np.array([0, 1, -1, 0]).reshape(2, 2)
A
```

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

```
B = np.array([1, 2, 3, 4][::-1]).reshape(2, 2)
B
```

```
array([[4, 3],
       [2, 1]])
```

```
abs(A)
```

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

```
np.negative(mat)
```

```
matrix([[-1, -2, -3],
        [-4, -5, -6],
        [-7, -8, -9]])
```

```
np.append(A, [[2, 3]], axis=0)
```

```
array([[ 0,  1],
       [-1,  0],
       [ 2,  3]])
```

```
np.append(A, [[2], [3]], axis=1)
```

```
array([[ 0,  1,  2],
       [-1,  0,  3]])
```

```
np.delete(mat, 1, 0)
```

```
matrix([[1, 2, 3],
        [7, 8, 9]])
```

```
np.delete(mat, 0, 1)
```

```
matrix([[2, 3],
        [5, 6],
        [8, 9]])
```

```
np.min(mat)
```

```
1
```

```
np.max(mat)
```

```
9
```

```
np.sum(mat)
```

```
45
```

PRACTICAL-8

Objective :

Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

Code and Output

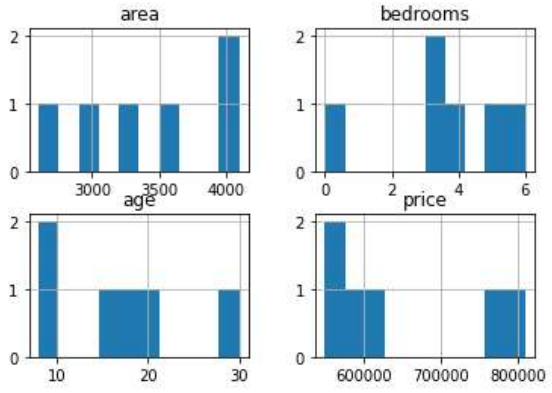
```
import matplotlib.pyplot as plt
```

```
import pandas as pd
df = pd.read_csv('homeprices.csv')
df.head()
```

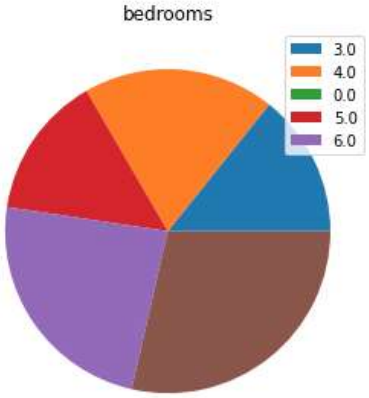
	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000

```
df = df.fillna(0)
```

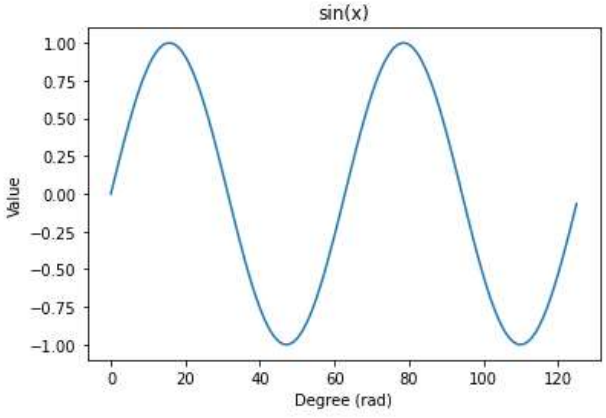
```
df.hist()
plt.show()
```



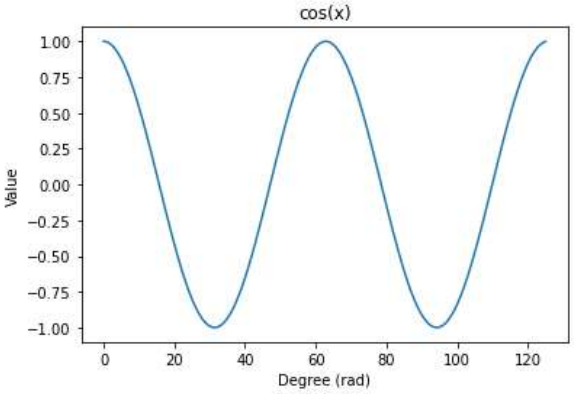
```
patches, _ = plt.pie(df['bedrooms'])
text = pd.unique(df['bedrooms'])
plt.legend(patches, text, loc="best")
plt.tight_layout()
plt.title('bedrooms')
plt.show()
```



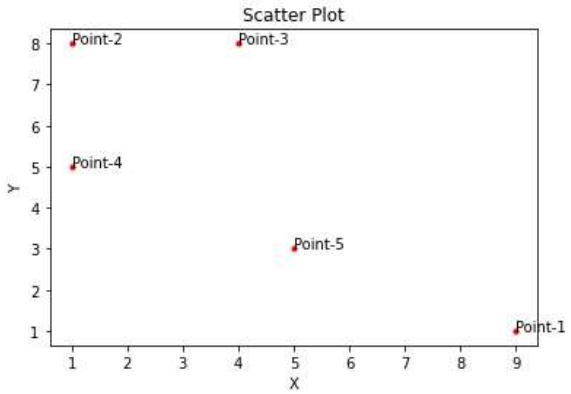

```
import numpy as np
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)
df_sin = pd.DataFrame({'x': x, 'y': y})
plt.plot(df_sin['y'])
plt.title('sin(x)')
plt.ylabel('Value')
plt.xlabel('Degree (rad)')
plt.show()
```



```
import numpy as np
x = np.arange(0, 4 * np.pi, 0.1)
y = np.cos(x)
df_cos = pd.DataFrame({'x': x, 'y': y})
plt.plot(df_cos['y'])
plt.title('cos(x)')
plt.ylabel('Value')
plt.xlabel('Degree (rad)')
plt.show()
```




```
X = np.random.randint(10, size=(5))
Y = np.random.randint(10, size=(5))

annotations=["Point-1",
             "Point-2",
             "Point-3",
             "Point-4",
             "Point-5"]

plt.scatter(X, Y, s=10, color='r')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Scatter Plot")
for i, label in enumerate(annotations):
    plt.annotate(label, (X[i], Y[i]))

plt.show()
```



PRACTICAL-9

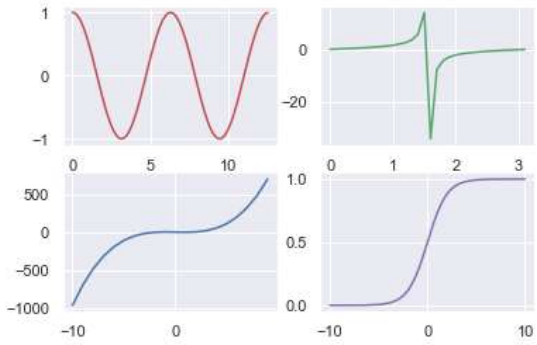
Objective :

Generate different subplots from a given plot and colour plotted data.

Code and Output

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

fig = plt.figure()

plt.subplot(2, 2, 1)
x = np.arange(0, 4 * np.pi, 0.1)
y = np.cos(x)
plt.plot(x, y, c='r')

plt.subplot(2, 2, 2)
x = np.arange(0, 1 * np.pi, 0.1)
y = np.tan(x)
plt.plot(x, y, c='g')

plt.subplot(2, 2, 3)
x = np.arange(-10, 10)
y = x ** 3 - 3 * x + 2
plt.plot(x, y)

plt.subplot(2, 2, 4)
x = np.linspace(-10, 10, 200)
y = 1 / (1 + np.exp(-x))
plt.plot(x, y, c='m')

plt.show()
```

✓ 0.2s



PRACTICAL-10

Objective :

Use conditional statements and different type of loops based on simple example/s.

Code and Output

```
a = 15
b = 4
c = 23

if a < b and a < c:
  print("a is smallest")
elif b < a and b < c:
  print("b is smallest")
else:
  print("c is smallest")
```

✓ 0.2s

```
b is smallest
```

```
for x in range(1, 10):
    if not x % 2:
        print(x)
```

✓ 0.1s

```
2
4
6
8
```

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

✓ 0.2s

```
1
2
3
4
5
6
7
8
9
10
```

PRACTICAL-11

Objective :

Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting, or multiplying two matrices.

Code and Output

```
import numpy as np

mat = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mat
```

✓ 0.8s

```
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

```
mat.T
```

✓ 0.2s

```
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

```
A = np.matrix([[0, 1], [-1, 0]])
A
```

✓ 0.2s

```
matrix([[ 0,  1],
        [-1,  0]])
```

```
B = np.matrix([[4, 3], [2, 1]])
B
```

✓ 0.2s

```
matrix([[4, 3],
        [2, 1]])
```

```
A + B
```

✓ 0.2s

```
matrix([[4, 4],
        [1, 1]])
```

```
A - B
```
✓ 0.3s

```
matrix([[-4, -2],
        [-3, -1]])
```

```
A * B
```
✓ 0.4s

```
matrix([[ 2,  1],
        [-4, -3]])
```

PRACTICAL-12

Objective :

Implement a Linear Regression problem.

Code and Output

Preprocessing

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style='darkgrid')
```

```
df = pd.read_csv('homeprices.csv')
df.head()
```

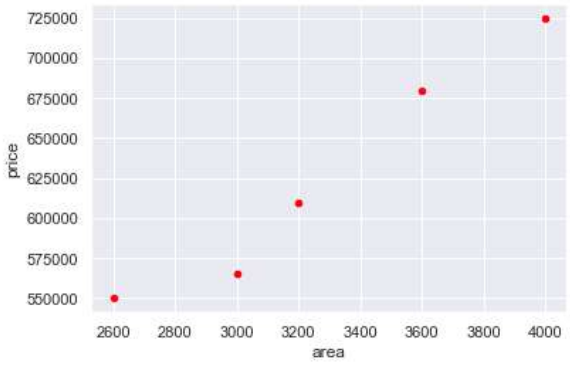
	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
df.describe()
```

	area	price
count	5.000000	5.000000
mean	3280.000000	626000.000000
std	540.370243	74949.983322
min	2600.000000	550000.000000
25%	3000.000000	565000.000000
50%	3200.000000	610000.000000
75%	3600.000000	680000.000000
max	4000.000000	725000.000000

```
X_train = df.drop('price', axis=1)
y_train = df['price']
```

```
df.plot.scatter('area', 'price', color='red')
plt.show()
```



Modelling

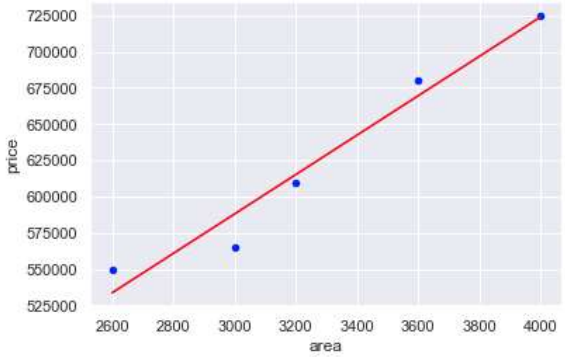
```
from sklearn.linear_model import LinearRegressio
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
model.coef_, model.intercept_
```

```
(array([135.78767123]), 180616.43835616432)
```

```
df.plot.scatter('area', 'price', color='blue')
plt.plot(
    np.array(df['area']),
    model.coef_[0] * np.array(df['area'])
        + model.intercept_,
    color='red'
)
plt.show()
```



Evaluation

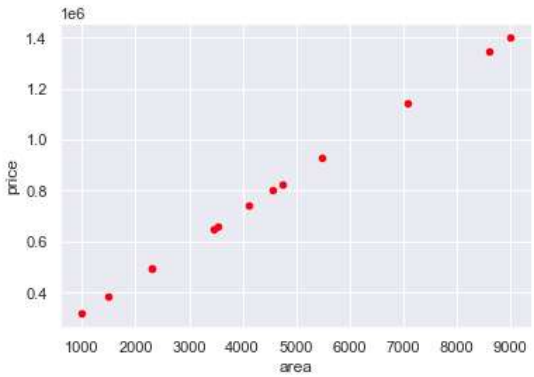
```
test_df = pd.read_csv('areas.csv')
test_df.head()
```

	area
0	1000
1	1500
2	2300
3	3540
4	4120

```
test_df['price'] = model.predict(test_df)
test_df
```

	area	price
0	1000	3.164041e+05
1	1500	3.842979e+05
2	2300	4.929281e+05
3	3540	6.613048e+05
4	4120	7.400616e+05
5	4560	7.998082e+05
6	5490	9.260908e+05
7	3460	6.504418e+05
8	4750	8.256079e+05
9	2300	4.929281e+05
10	9000	1.402705e+06
11	8600	1.348390e+06
12	7100	1.144709e+06

```
test_df.plot.scatter('area', 'price', color='red')
plt.show()
```



PRACTICAL-13

Objective :

Perform Linear Regression based on multiple features.

Code and Output

Preprocessing

```
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style='darkgrid')
```

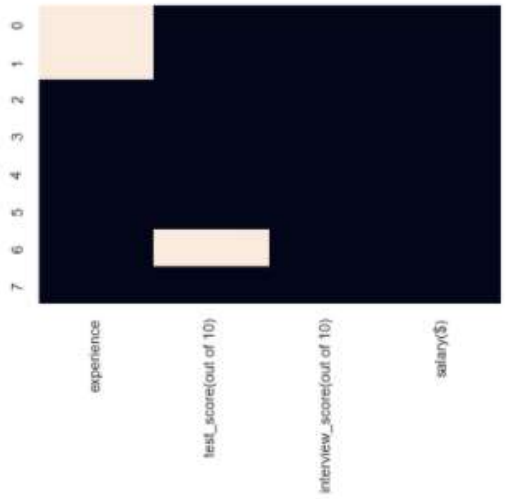
```
df = pd.read_csv('hiring.csv')
df
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	NaN	8.0	9	50000
1	NaN	8.0	6	45000
2	five	6.0	7	60000
3	two	10.0	10	65000
4	seven	9.0	6	70000
5	three	7.0	10	62000
6	ten	NaN	7	72000
7	eleven	7.0	8	80000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 4 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   experience                  6 non-null      object
 1   test_score(out of 10)       7 non-null      float64
 2   interview_score(out of 10)  8 non-null      int64
 3   salary($)                   8 non-null      int64
dtypes: float64(1), int64(2), object(1)
memory usage: 384.0+ bytes
```

```
sns.heatmap(df.isnull(), cbar=False)
plt.show()
```




```
df['experience'].unique()
```

```
array([nan, 'five', 'two', 'seven', 'three', 'ten', 'eleven'],
      dtype=object)
```

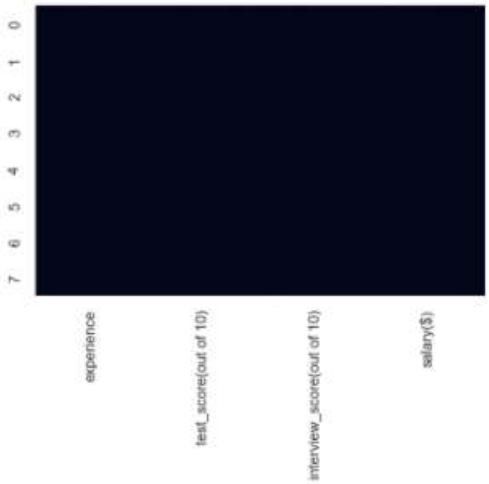
```
df['experience'] = df['experience'].replace({
    'five': 5,
    'two': 2,
    'seven': 7,
    'three': 3,
    'ten': 10,
    'eleven': 11
})
df['experience'] = df['experience'].fillna(0)
df
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	0.0	8.0	9	50000
1	0.0	8.0	6	45000
2	5.0	6.0	7	60000
3	2.0	10.0	10	65000
4	7.0	9.0	6	70000
5	3.0	7.0	10	62000
6	10.0	NaN	7	72000
7	11.0	7.0	8	80000

```
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(
  df['test_score(out of 10)'].median()
)
df
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	0.0	8.0	9	50000
1	0.0	8.0	6	45000
2	5.0	6.0	7	60000
3	2.0	10.0	10	65000
4	7.0	9.0	6	70000
5	3.0	7.0	10	62000
6	10.0	8.0	7	72000
7	11.0	7.0	8	80000

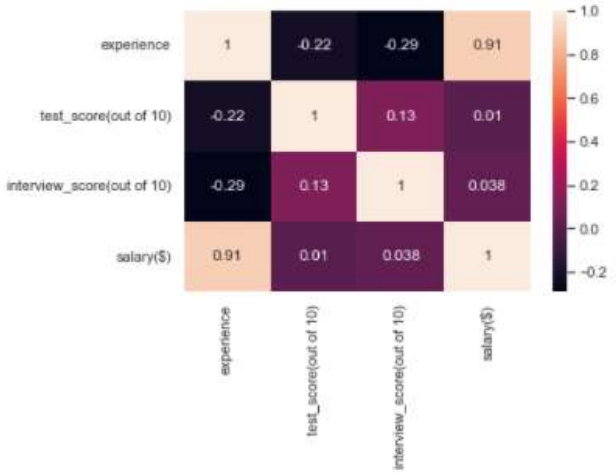
```
sns.heatmap(df.isnull(), cbar=False)
plt.show()
```




```
df.describe()
```

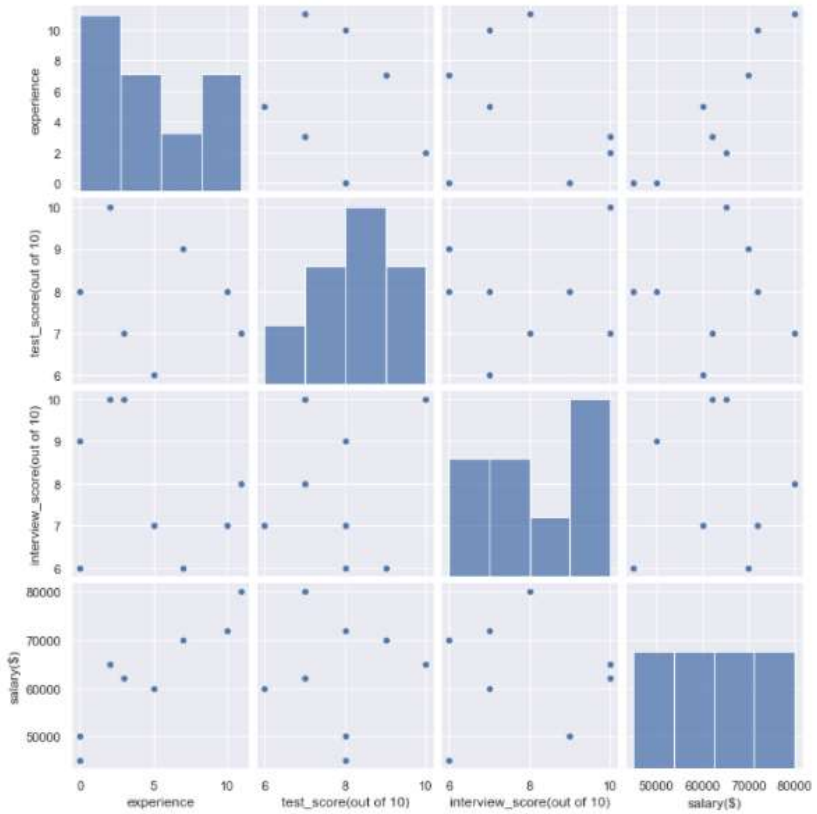
	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
count	8.00000	8.000000	8.000000	8.00000
mean	4.75000	7.875000	7.875000	63000.00000
std	4.26782	1.246423	1.642081	11501.55269
min	0.00000	6.000000	6.000000	45000.00000
25%	1.50000	7.000000	6.750000	57500.00000
50%	4.00000	8.000000	7.500000	63500.00000
75%	7.75000	8.250000	9.250000	70500.00000
max	11.00000	10.000000	10.000000	80000.00000

```
sns.heatmap(df.corr(), annot=True)
plt.show()
```




```
sns.pairplot(df)
plt.show()
```

Python




```
X_train = df.drop('salary($)', axis=1)
y_train = df['salary($)']
```

Python

Modelling

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
model.coef_, model.intercept_
```

```
(array([2812.95487627, 1845.70596798, 2205.24017467]), 17737.263464337666)
```

Evaluation

- 2 yr experience, 9 test score, 6 interview score
- 12 yr experience, 10 test score, 10 interview score

```
eval_df = pd.DataFrame({
    'experience': [2, 12],
    'test_score(out of 10)': [9., 10.],
    'interview_score(out of 10)': [6, 10],
})
eval_df
```

	experience	test_score(out of 10)	interview_score(out of 10)
0	2	9.0	6
1	12	10.0	10

```
eval_df['salary($)'] = model.predict(eval_df)
eval_df
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	2	9.0	6	53205.967977
1	12	10.0	10	92002.183406

PRACTICAL-14

Objective :

Implement a Classification/Logistic Regression problem.

Code and Output

Preprocessing

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style='darkgrid')
```

Python

```python
from sklearn.datasets import load_digits
digits = load_digits()
digits.keys()
```

Python

```
dict_keys(['data', 'target', 'frame', 'feature_names',
'target_names', 'images', 'DESCR'])
```

```python
digits.data
```

Python

```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

```python
digits.target
```

Python

```
array([0, 1, 2, ..., 8, 9, 8])
```

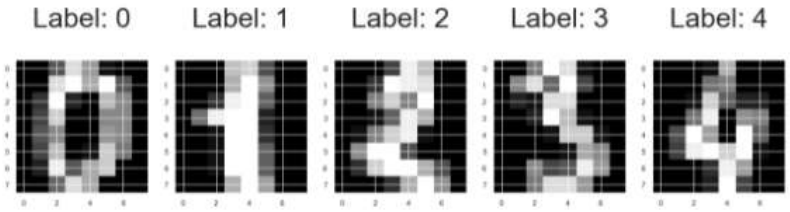
```python
print("Data Shape:", digits.data.shape)
print("Target Shape:", digits.target.shape)
```

Python

```
Data Shape: (1797, 64)
Target Shape: (1797,)
```

```python
plt.figure(figsize=(20, 4))
for index, (image, label) in enumerate(
  zip(digits.data[0:5], digits.target[0:5])
):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8, 8)), cmap=plt.cm.gray)
    plt.title(f'Label: {label}\n', fontsize=40)
```

Python




```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    digits.data,
    digits.target,
    test_size=0.3,
    random_state=0
)
```

Python

Modelling

```python
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1e5)
lr
```

Python

```
LogisticRegression(max_iter=100000.0)
```

```python
lr.fit(x_train, y_train)
```

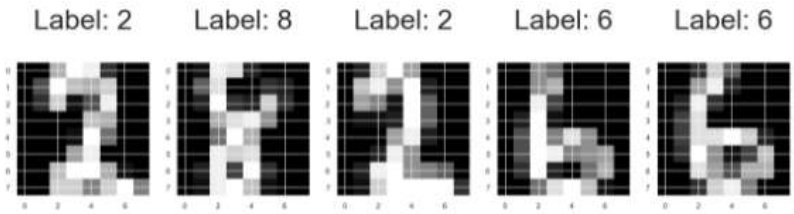
Python

```
LogisticRegression(max_iter=100000.0)
```

Evaluation

```python
plt.figure(figsize=(20, 4))
for index, (image, label) in enumerate(
    zip(x_test[0:5],y_test[0:5])
):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8, 8)), cmap=plt.cm.gray)
    plt.title(f'Label: {label}\n', fontsize=40)
```

Python




```python
lr.predict(x_test[0:5])
```

Python

```
array([2, 8, 2, 6, 6])
```

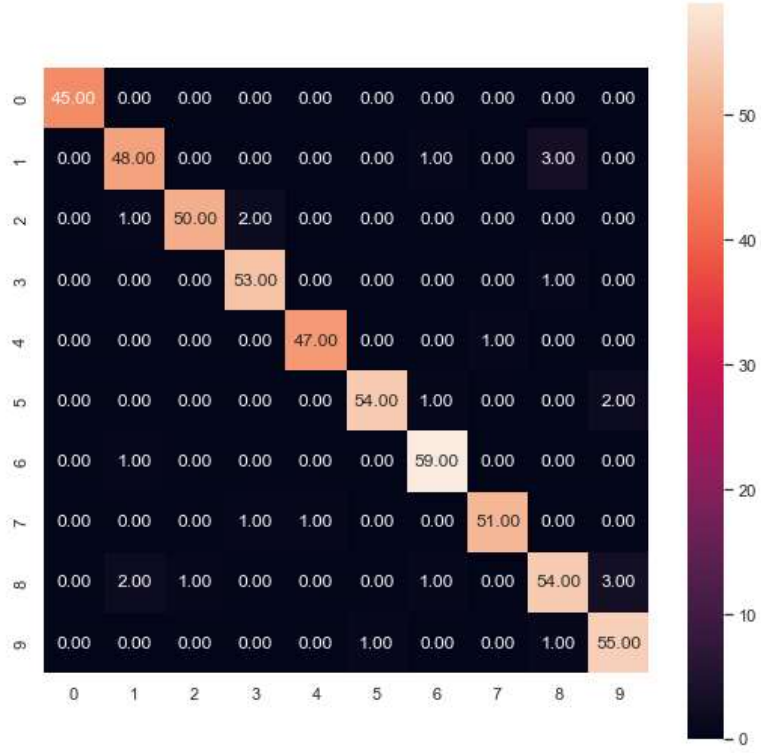
```python
predictions = lr.predict(x_test)
score = lr.score(x_test, y_test)
print(f'Accuracy: {round(score * 100, 5)}%')
```

Python

```
Accuracy: 95.55556%
```

```python
from sklearn import metrics

cm = metrics.confusion_matrix(y_test, predictions)

plt.figure(figsize=(9, 9))
sns.heatmap(cm, annot=True, square=True, fmt='.2f')
plt.show()
```

Python



PRACTICAL-15

Objective :

Use some function for regularization of dataset based on problem 14.

Code and Output

Preprocessing

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style='darkgrid')
```

✓ 1.1s

```
from sklearn.datasets import load_digits
digits = load_digits()
```

✓ 0.1s

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    digits.data,
    digits.target,
    test_size=0.3,
    random_state=0
)
```

✓ 0.3s

Modelling

```
from sklearn.linear_model import RidgeClassifier

lr = None
best_alpha = 0
best_score = 0

for alpha in np.logspace(-3, 3, 10):
    lr = RidgeClassifier(alpha=alpha)
    lr.fit(x_train, y_train)
    score = lr.score(x_test, y_test)
    if score > best_score:
        best_score = score
        best_alpha = alpha
    print(f'alpha {alpha:.3f}\t', end='')
    print(f'score: {score:.5f}')
```

✓ 0.7s

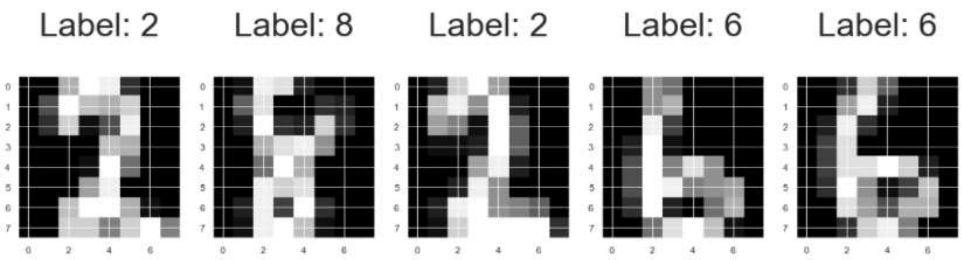
```
alpha 0.001     score: 0.92037
alpha 0.005     score: 0.92037
alpha 0.022     score: 0.92037
alpha 0.100     score: 0.92037
alpha 0.464     score: 0.92037
alpha 2.154     score: 0.92037
alpha 10.000    score: 0.92222
alpha 46.416    score: 0.92222
alpha 215.443   score: 0.92407
alpha 1000.000  score: 0.92037
```

```
lr = RidgeClassifier(alpha=best_alpha)
lr.fit(x_train, y_train)
```
✓ 0.2s

```
RidgeClassifier(alpha=215.44346900318823)
```

Evaluation

```
plt.figure(figsize=(20, 4))
for index, (image, label) in enumerate(
    zip(x_test[0:5],y_test[0:5])
):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8, 8)), cmap=plt.cm.gray)
    plt.title(f'Label: {label}\n', fontsize=40)
plt.show()
```
✓ 0.5s




```
lr.predict(x_test[0:5])
```
✓ 0.2s

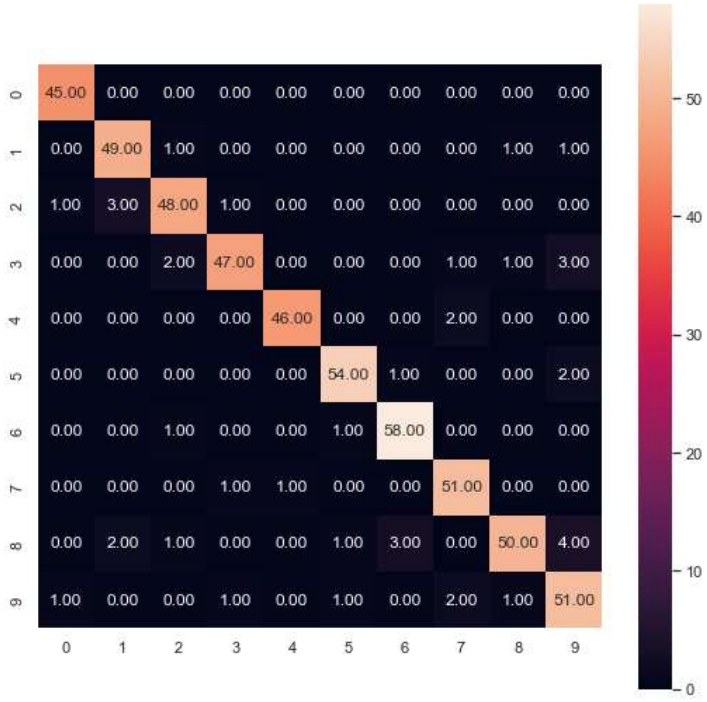
```
array([2, 8, 2, 6, 6])
```

```
predictions = lr.predict(x_test)
score = lr.score(x_test, y_test)
print(f'Accuracy: {round(score * 100, 5)}%')
```
✓ 0.3s

```
Accuracy: 92.40741%
```

```python
from sklearn import metrics

cm = metrics.confusion_matrix(y_test, predictions)

plt.figure(figsize=(9, 9))
sns.heatmap(cm, annot=True, square=True, fmt='.2f')
plt.show()
```

✓ 0.3s



PRACTICAL-16

Objective :

Use some function for neural networks, like Stochastic Gradient Descent or Backpropagation to predict the value of a variable based on the dataset of problem 14.

Code and Output

The Multi-layer Perceptron classifier model optimizes the log-loss function using stochastic gradient descent.

Preprocessing

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style='darkgrid')
```

✓ 1.1s

```
from sklearn.datasets import load_digits
digits = load_digits()
```

✓ 0.1s

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    digits.data,
    digits.target,
    test_size=0.3,
    random_state=0
)

x_train, x_validation, y_train, y_validation = train_test_split(
    x_train,
    y_train,
    test_size=0.2,
    random_state=0
)
```

✓ 0.4s

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_validation = scaler.transform(x_validation)
x_test = scaler.transform(x_test)
```

✓ 0.2s

Modelling

```
import random as r
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

best_model = {'hls': 0,
              'act': '',
              'lr': '',
              'accuracy': 0,
              'model': None}

for i in range(50):
    hls = r.randint(100, 1000)
    act = r.choice(['identity', 'logistic', 'tanh', 'relu'])
    lr = r.choice(['constant', 'invscaling', 'adaptive'])
    model = MLPClassifier(hidden_layer_sizes=hls,
                          activation=act,
                          learning_rate=lr,
                          solver='sgd',
                          max_iter=10000)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_validation)
    accuracy = accuracy_score(y_validation, y_pred)
    if accuracy > best_model['accuracy']:
        best_model['hls'] = hls
        best_model['act'] = act
        best_model['lr'] = lr
        best_model['accuracy'] = accuracy
        best_model['model'] = model

print('best model parameters:'
      + '\n\thidden_layer_size: ' + str(best_model['hls'])
      + '\n\tactivation:  ' + best_model['act']
      + '\n\tlearning_rate: ' + best_model['lr'])

print(f'accuracy: {str(best_model["accuracy"])}')
```

✓ 13m 9.4s

```
best model parameters:
        hidden_layer_size: 198
        activation:  relu
        learning_rate: adaptive
accuracy: 0.9841269841269841
```

```
model = best_model['model']
model
```

✓ 0.4s

```
MLPClassifier(hidden_layer_sizes=198, learning_rate='adaptive', max_iter=10000,
              solver='sgd')
```

Evaluation

```
y_pred = model.predict(x_test)
accuracy_score(y_test, y_pred)
```

✓ 0.6s

```
0.9685185185185186
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(9, 9))
sns.heatmap(cm, annot=True, square=True, fmt='.2f')
plt.show()
```

✓ 0.4s