1. Write a program for the following

a.To generate an array of random numbers from a normal distribution for the array of a given shape.

```
import numpy as np
# Enter the value of n
n=int(input('Enter no. of values:'))
# Generates n random numbers from Normal Distribution
rand_num = np.random.normal(0,1,n)
print(n, " random numbers from a standard normal distribution:")
print(rand_num)
arr=np.array([rand_num])
# Displays the size of the Array
print(arr.shape)
```

output

```
Enter no. of values:10
10  random numbers from a standard normal distribution:
[-0.34953998  1.60514591 -0.60005696  0.26263808  0.87930153
0.98339437
  0.40472381 -0.73362668 -0.20067116 -0.97191095]
(1, 10)
```

b. Implement Arithmetic operations on two arrays (perform broadcasting also.)

```
#Generates an Array A with 0 to 11 in a 3X4 form
A = np.arange(12).reshape(3,4)
print(A)
#Generates an Array B with 0 to 3 in a 1X3 form
B  = np.arange(4)
print(B)
#Performs the addition between A and B
c=A+B
print(c)
#Similarly perform remaining arithmetic operations (Subtraction,multiplication, division )
```

output

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[0 1 2 3]
[[ 0  2  4  6]
 [ 4  6  8 10]
 [ 8 10 12 14]]
```

c. Find minimum, maximum, mean in a given array. (in both the axes)

```
arr = np.array([[11, 2, 3],[4, 5, 16],[7, 81, 22]])
# finding the maximum and minimum element in the array
max_element = np.max(arr)
min_element = np.min(arr)

# printing the result
```

```
print('maximum element in the array is:', max_element)
print('minimumm element in the array is:', min_element)
# finding the maximum and
# minimum element in the array
max_element_column = np.max(arr, 0)
max_element_row = np.max(arr, 1)

min_element_column = np.amin(arr, 0)
min_element_row = np.amin(arr, 1)

# printing the result
print('maximum elements in the columns of the array is:',max_element_column)

print('maximum elements in the rows of the array is:', max_element_row)

print('minimum elements in the columns of the array is:',min_element_column)

print('minimum elements in the rows of the array is:',min_element_row)

# mean of the flattened array
print("\nmean of arr, axis = None : ", np.mean(arr))

# mean along the axis = 0 (row-wise)
print("\nmean of arr, axis = 0 : ", np.mean(arr, axis = 0))

# mean along the axis = 1 (Column-wise)
print("\nmean of arr, axis = 1 : ", np.mean(arr, axis = 1))
```

output

```
maximum element in the array is: 81
minimumm element in the array is: 2
maximum elements in the columns of the array is: [11 81 22]
maximum elements in the rows of the array is: [11 16 81]
minimum elements in the columns of the array is: [4 2 3]
minimum elements in the rows of the array is: [2 4 7]

mean of arr, axis = None :  16.77777777777778

mean of arr, axis = 0 :  [ 7.33333333 29.33333333 13.66666667]

mean of arr, axis = 1 :  [ 5.33333333  8.33333333 36.66666667]
```

d. Implement np.arange and np.linspace functions.

```
# Prints all numbers from 0 to 9 in steps of 1
arr=np.linspace(start = 0, stop = 10, num = 11,dtype = int)
print(arr)
arr=np.linspace(start = 0, stop = 1, num = 11)
print(arr)
# Prints all numbers from 1 to 2 in steps of 0.1
```

```
print(np.arange(1, 2, 0.1))
```

output

```
[ 0  1  2  3  4  5  6  7  8  9 10]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
```

e. Create a pandas series from a given list.

```
# import pandas lib. as pd
import pandas as pd
# Assume l1 is a list of the following words
l1 = ['ZERO', 'ONE', 'TWO', 'THREE',
                    'FOUR', 'FIVE', 'SIX','SEVEN','EIGHT','NINE','TEN']

# create Pandas Series with define indexes
x = pd.Series(l1)

# print the Series
print(x)
```

output

```
0      ZERO
1       ONE
2       TWO
3     THREE
4      FOUR
5      FIVE
6       SIX
7     SEVEN
8     EIGHT
9      NINE
10      TEN
dtype: object
```

f. Create pandas series with data and index and display the index values.

```
# import pandas lib. as pd
import pandas as pd

# create Pandas Series with define indexes
x = pd.Series([10, 20, 30, 40, 50], index =['a', 'b', 'c', 'd', 'e'])

# print the Series
print(x)
```

output

```
a    10
b    20
c    30
```

```
d    40
e    50
dtype: int64
```

g. Create a data frame with columns at least 5 observations
i. select a particular column from the DataFrame
ii. Summarize the data frame and observe the stats of the DataFrame created
iii. Observe the mean and standard deviation of the data frame and print the values.

```
import pandas as pd
import numpy as np

exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily'],'score': [12.5, 9, 16.5, np.nan, 9],'attempts': [1, 3, 2, 3, 2],
        'qualify': ['yes', 'no', 'yes', 'no', 'no']}
labels = ['a', 'b', 'c', 'd', 'e']

df = pd.DataFrame(exam_data , index=labels)
print("Dataset is as follows")
print(df)
print("Summary of the Dataset")
print(df.info())
print("Statistical values of numerical attributes")
print(df.describe())
meanvalue=df.score.mean()
stdvalue=df.score.std()
print('mean value of Score is',meanvalue)
print('Standard deviation of score is',stdvalue)
```

output

```
Dataset is as follows
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
Summary of the Dataset
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, a to e
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      5 non-null      object
 1   score     4 non-null      float64
 2   attempts  5 non-null      int64
 3   qualify   5 non-null      object
dtypes: float64(1), int64(1), object(2)
memory usage: 200.0+ bytes
None
Statistical values of numerical attributes
           score  attempts
count   4.000000   5.00000
mean   11.750000   2.20000
```

```
std      3.570714   0.83666
min      9.000000   1.00000
25%      9.000000   2.00000
50%     10.750000   2.00000
75%     13.500000   3.00000
max     16.500000   3.00000
mean value of Score is 11.75
Standard deviation of score is 3.570714214271425
```

1. Write a Program to determine the following in the Titanic Survival data.

a. Determine the data type of each column.

```
# importing all the necessary libraries
import pandas as pd
import numpy as np

#we need to read the data
data = pd.read_csv("https://raw.githubusercontent.com/naveenjoshii/Intro-to-MachineLearning/master/Titanic/titanic.csv")
#print top 5 rows
print(data.head())
```

output

```
PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0        3  ...   7.2500   NaN         S
1            2         1        1  ...  71.2833   C85         C
2            3         1        3  ...   7.9250   NaN         S
3            4         1        1  ...  53.1000  C123         S
4            5         0        3  ...   8.0500   NaN         S

[5 rows x 12 columns]
```

```
# to get the datatype of all columns we can use Dataframe.dtypes
print(data.dtypes)
```

output

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

b. Find the number of non-null values in each column.

```
# Dataframe.info() gives all information about every column in our dataset
data.info()
```

output

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

c. Find out the unique values in each categorical column and frequency of each unique value.

```
# categorical is nothing but the datatype which is other than numerical datatype (i.e int,float etc).
# to get the all categorical columns, we can use Dataframe.select_dtypes and we have to specify which
#datatype we required.
# In our case it would be "object" datatype
categorical_cols  = data.select_dtypes(include=['object']).columns.tolist()
print("Categorical columns are : ",categorical_cols)
print("printing the results")
for i in categorical_cols:
  print("==========  Column '"+i+"' =============")
  print(data[i].value_counts())
```

output

```
Categorical columns are :  ['Name', 'Sex', 'Ticket', 'Cabin',
'Embarked']
printing the results
==========  Column 'Name' =============
Robert, Mrs. Edward Scott (Elisabeth Walton McMillan)    1
Smith, Mr. Thomas                                        1
Cameron, Miss. Clear Annie                               1
Parkes, Mr. Francis "Frank"                              1
```

```
Panula, Mrs. Juha (Maria Emilia Ojala)                 1
                                                      ..
Walker, Mr. William Anderson                           1
Hassab, Mr. Hammad                                     1
Olsen, Mr. Karl Siegwart Andreas                       1
Reed, Mr. James George                                 1
Wiseman, Mr. Phillippe                                 1
Name: Name, Length: 891, dtype: int64
==========  Column 'Sex' =============
male      577
female    314
Name: Sex, dtype: int64
==========  Column 'Ticket' =============
1601                7
CA. 2343            7
347082              7
3101295             6
CA 2144             6
                   ..
350034              1
19947               1
A/5 21174           1
PC 17474            1
SOTON/OQ 392082     1
Name: Ticket, Length: 681, dtype: int64
==========  Column 'Cabin' =============
G6             4
B96 B98        4
C23 C25 C27    4
F2             3
E101           3
              ..
B38            1
B102           1
E58            1
C101           1
B4             1
Name: Cabin, Length: 147, dtype: int64
==========  Column 'Embarked' =============
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

d. Find the number of rows where age is greater than the mean age of data.

```
# to get mean of age column
age_mean = data['Age'].mean()
print("Mean of Age is : ",age_mean)
print("printing the result")
print(np.sum(data['Age']>age_mean))
```

output

```
Mean of Age is :  29.69911764705882
printing the result
330
```

e. Delete all the rows with missing values.

```
print("length of dataframe before deleting rows with missing values",len(data))
# deletes the rows where at least one element is missing
data.dropna(inplace=True)
print("length of dataframe after the deletion of missing value rows",len(data))
```

output

```
length of dataframe before deleting rows with missing values 891
length of dataframe after the deletion of missing value rows 183
```

3. Perform Data Analysis on the Titanic Data Set to answer the following.

```
#importing all the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#reading data
data = pd.read_csv("https://raw.githubusercontent.com/naveenjoshii/Intro-to-MachineLearning/master/Titanic/titanic.csv")
print(data.head()
```

output

```
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]
```

a. Information regarding each column of the data

```
#printing the info about all the columns
print(data.info())
```

output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
```

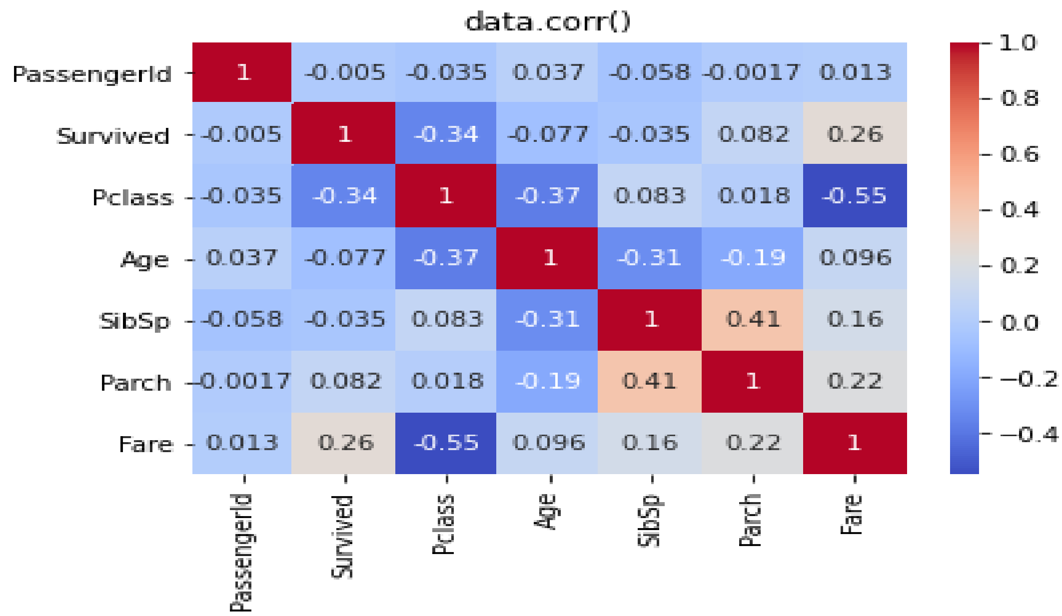
```
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

b. Impact of each column on the label

```
# plotting the correlation using heatmap
sns.heatmap(data.corr(),cmap='coolwarm',xticklabels=True,annot=True)
plt.title('data.corr()')
```

output

```
Text(0.5, 1.0, 'data.corr()')
```

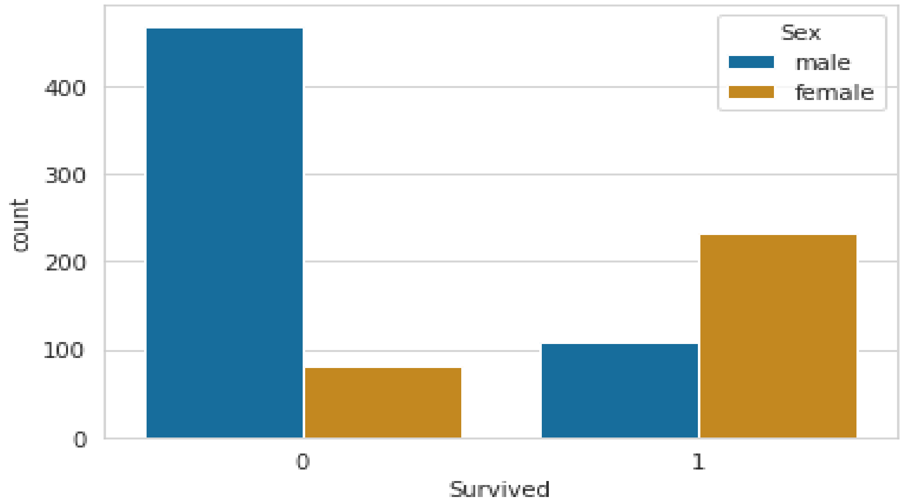


c. Number of survivals in each gender

```
# plotting countplot for Each gender who has survived and not survived
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=data,palette='colorblind')
```

output

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f621a047810>
```

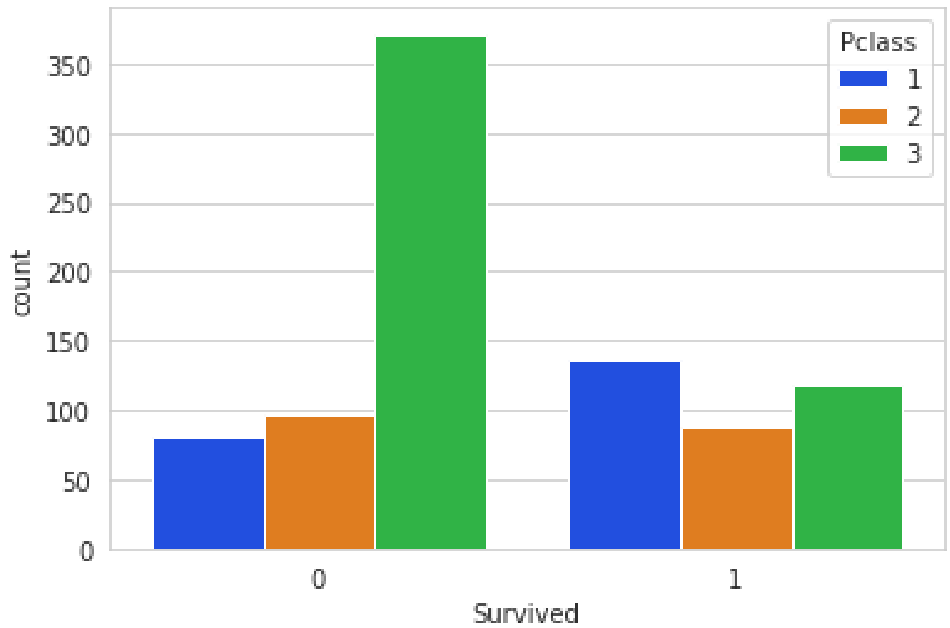


d. Number of survivals in each passenger class

#plotting count plot for no of survivals in each class
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=data,palette='bright')

output

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f621a034510>
```

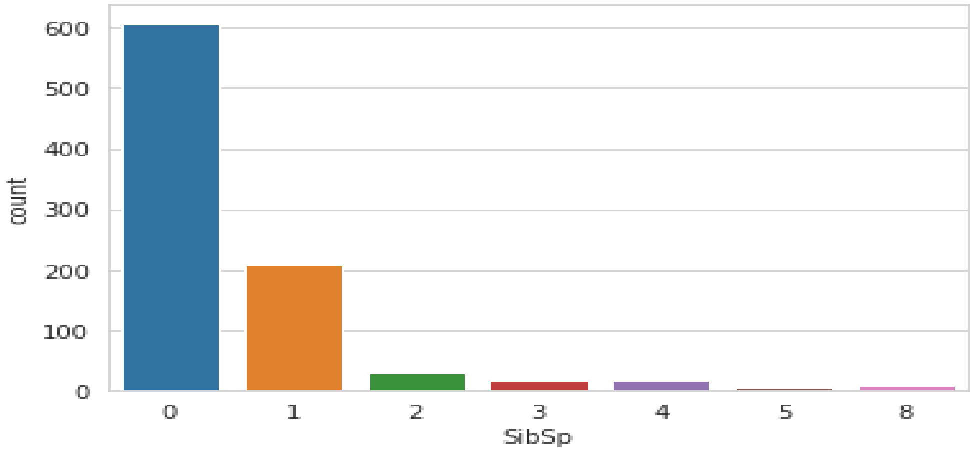


e. The number of people who are not alone.

# count plot for who has siblings/spouse
sns.countplot(x = 'SibSp', data = data,)

output

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6219b36390>
```



4. Perform Data Analysis on the California House Price data to answer the following

```
# importing all the necessary libraries
import pandas as pd
import numpy as np

#we need to read the data
data = pd.read_csv("https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv")
#print top 5 rows
print(data.head())
```

output

```
longitude  latitude  ...  median_house_value  ocean_proximity
0     -122.23     37.88  ...            452600.0         NEAR BAY
1     -122.22     37.86  ...            358500.0         NEAR BAY
2     -122.24     37.85  ...            352100.0         NEAR BAY
3     -122.25     37.85  ...            341300.0         NEAR BAY
4     -122.25     37.85  ...            342200.0         NEAR BAY

[5 rows x 10 columns]
```

a. Data Type of each column and info regarding each column

```
# data information for each column
print(data.info())
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
```

b. The average age of a house in the data set.

```
# printing average age of house
print(data['housing_median_age'].mean())
```

Output

```
28.639486434108527
```

c. Determines top 10 localities with the high difference between income and house value. Also, top 10 localities that have the lowest difference

```
#calculating the difference btw House value and income and adding new column
'diff_income_and_house_value' with difference values
data['diff_income_and_house_value'] = data['median_house_value'] - data['median_income']
# sorting the whole dataframe by the difference value in descending order
data.sort_values(by='diff_income_and_house_value', ascending=False,inplace=True)
#printing the top 10 localities with highest difference
print("the top 10 localities with highest difference")
print(data['ocean_proximity'].head(10))
#printing the top 10 localities with lowest difference
print("the top 10 localities with lowest difference")
print(data['ocean_proximity'].tail(10))
```

Output

```
the top 10 localities with highest difference
4861      <1H OCEAN
6688         INLAND
16642    NEAR OCEAN
15661      NEAR BAY
15652      NEAR BAY
6639      <1H OCEAN
459        NEAR BAY
89         NEAR BAY
10448     <1H OCEAN
17819     <1H OCEAN
Name: ocean_proximity, dtype: object
the top 10 localities with lowest difference
```

```
2779          INLAND
16186         INLAND
14326     NEAR OCEAN
1825        NEAR BAY
13889         INLAND
5887       <1H OCEAN
19802         INLAND
2521          INLAND
2799          INLAND
9188          INLAND
Name: ocean_proximity, dtype: object
```

d. What is the ratio of bedrooms to total rooms in the data

```
# total no of rooms
total_rooms = data['total_rooms'].sum()
# total number of bedrooms
total_bedrooms = data['total_bedrooms'].sum()
#printing the ratio of bedrooms to total rooms
print(total_rooms//total_bedrooms)
```

Output

```
4.0
```

e. Determine the average price of a house for each type of ocean_proximity.

```
# average house price for each ocean_proximity type
data.groupby('ocean_proximity')['median_house_value'].median()
```

Output

```
ocean_proximity
<1H OCEAN     214850.0
INLAND        108500.0
ISLAND        414700.0
NEAR BAY      233800.0
NEAR OCEAN    229450.0
Name: median_house_value, dtype: float64
```

5. Write a program to perform the following tasks

a. Determine the outliers in each non-categorical column of Titanic Data and remove them.

```
# importing all the necessary libraries
import pandas as pd
import numpy as np

#we need to read the data
data = pd.read_csv("https://raw.githubusercontent.com/naveenjoshii/Intro-to-MachineLearning/master/Titanic/titanic.csv")
#print top 5 rows
print(data.head())
```

Output

```
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]
```

```
# function to calculate the lower and upperbound
def detect_outliers(data,threshold):
  mean = np.mean(data)
  std =np.std(data)
  lb = max(mean - (threshold * std),min(data))
  ub = min(mean + (threshold * std),max(data))
  return lb,ub
```

```
df = data.copy()
lb,ub = detect_outliers(data["Fare"],4)
# removing the rows which are greater than upperbound
df.drop(df[df.Fare > ub].index, inplace=True)
# removing the rows which are less than lowerbound
df.drop(df[df.Fare < lb ].index, inplace=True)
```

```
lb,ub = detect_outliers(data["Age"],5)
# removing the rows which are greater than upperbound
df.drop(df[df.Age > ub].index, inplace=True)
# removing the rows which are less than lowerbound
df.drop(df[df.Age < lb].index, inplace=True)
```

b. Determine missing values in each column of Titanic data. If missing values account for 30% of data, then remove the column.

```
#printing the missing value percentage for every column
df.isnull().mean() * 100
```

Output

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.113636
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
```

```
Cabin          77.954545
Embarked        0.227273
dtype: float64
```

get all the column names in our dataset
df.columns

Output

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

# As we can see cabin column has more than 30% of missing values, so we have to drop that column
df.drop(['Cabin'],inplace=True,axis=1)

after removing the column cabin, printing the columns again. If you observe there is no Cabin in the output
df.columns

Output

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked'],
      dtype='object')
```

c. If missing values are less than 30% of entire data then create a new data frame
i. Missing values in numeric columns are filled with the mean of the corresponding column.

#printing the percentage of missing values in Age before handling
df['Age'].isnull().mean() * 100

Output

```
20.113636363636363
```

Filling the missing values with the mean of respective column
df['Age']=df['Age'].fillna(df['Age'].mean())

#printing the percentage of missing values in Age after handling
df['Age'].isnull().mean() * 100

Output

```
0.0
```

ii. Missing values in categorical columns are filled with the most frequently occurring value.

```
#printing the percentage of missing values in Embarked before handling
df['Embarked'].isnull().mean() * 100
```

Output

```
0.22727272727272727
```

```
# filling with filled with the most frequently occurring value.
df["Embarked"].fillna(df['Embarked'].mode()[0],inplace=True)

#printing the percentage of missing values in Embarked after handling
df['Embarked'].isnull().mean() * 100
```

Output

```
0.0
```

6. Write a program to perform the following tasks

a. Determine the categorical columns in Titanic Dataset. Convert Columns with string data type to numerical data using encoding techniques.

```
#information about data
df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 880 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  880 non-null    int64  
 1   Survived     880 non-null    int64  
 2   Pclass       880 non-null    int64  
 3   Name         880 non-null    object 
 4   Sex          880 non-null    object 
 5   Age          880 non-null    float64
 6   SibSp        880 non-null    int64  
 7   Parch        880 non-null    int64  
 8   Ticket       880 non-null    object 
 9   Fare         880 non-null    float64
 10  Embarked     880 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 122.5+ KB
```

```
print("each unique value and respective counts in Sex column\n",df['Sex'].value_counts())
```

```
#creating another data frame for Sex column
sex_df = pd.get_dummies(df['Sex'],drop_first=3)
sex_df.head()
```

Output

```
each unique value and respective counts in Sex column
 male      572
female    308
Name: Sex, dtype: int64
```

	male
0	1
1	0
2	0
3	0
4	1

```
print("each unique value and respective counts in Sex
column\n",df['Embarked'].value_counts())
# creating dummies for Embarked
embark_df = pd.get_dummies(df['Embarked'],drop_first=True)
embark_df.head()
```

Output

```
each unique value and respective counts in Sex column
 S    642
C    161
Q     77
Name: Embarked, dtype: int64
```

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

```
old_data = df.copy()
# we need to drop the sex and embarked columns and replace them with the newly created
dummies data frames
# as Name and Tickt is not making any impact on the output label, we can drop them also
df.drop(['Sex','PassengerId','Embarked','Name','Ticket'],axis=1,inplace=True)
df.head()
```

Output

	Survived	Pclass	Age	SibSp	Parch	Fare
0	0	3	22.0	1	0	7.2500
1	1	1	38.0	1	0	71.2833

2	1	3	26.0	0	0	7.9250
3	1	1	35.0	1	0	53.1000
4	0	3	35.0	0	0	8.0500

```
# After droping the Sex and Embarked columns, we are replacing them with out new data frames
data = pd.concat([df,sex_df,embark_df],axis=1)
```

b. Convert data in each numerical column so that it lies in the range [0,1]

```
# before scaling the data
data.head()
```

Output

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

```
# Scaling the data using minmax scaler so that values should be lies btw [0,1]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['Age','Pclass','Survived','SibSp','Parch','Fare','male','Q','S']] =
scaler.fit_transform(data[['Age','Pclass','Survived','SibSp','Parch','Fare','male','Q','S']])

# after scaling the data
data.head()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0.0	1.0	0.271174	0.125	0.0	0.031865	1.0	0.0	1.0
1	1.0	0.0	0.472229	0.125	0.0	0.313299	0.0	0.0	0.0
2	1.0	1.0	0.321438	0.000	0.0	0.034831	0.0	0.0	1.0
3	1.0	0.0	0.434531	0.125	0.0	0.233381	0.0	0.0	1.0
4	0.0	1.0	0.434531	0.000	0.0	0.035381	1.0	0.0	1.0

7. Implement the following models on Titanic Dataset and determine the values of accuracy, precision, recall, f1 score and confusion matrix for the test data.

```
data.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 880 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  880 non-null    float64
 1   Pclass    880 non-null    float64
 2   Age       880 non-null    float64
 3   SibSp     880 non-null    float64
 4   Parch     880 non-null    float64
 5   Fare      880 non-null    float64
 6   male      880 non-null    float64
 7   Q         880 non-null    float64
 8   S         880 non-null    float64
dtypes: float64(9)
memory usage: 108.8 KB
```

Split the Data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data.drop('Survived',axis=1),
                                    data['Survived'], test_size=0.30,
                                    random_state=101)
```

a. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Build the Model.
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Output

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
```

```
print("Predicting the model on the test set")
predicted =  logmodel.predict(X_test)
```

Output

```
Predicting the model on the test set
```

```
print("predicted result !")
```

predicted

Output

```
predicted result !

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

```
#confusion matrix
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, predicted))
```

Output

```
[[144  24]
 [ 28  68]]
```

```
# Precision Score
from sklearn.metrics import precision_score
print("Precision Score",precision_score(y_test,predicted))
```

Output

```
Precision Score 0.7391304347826086
```

```
# Recall Score
from sklearn.metrics import recall_score
print("recall score",recall_score(y_test,predicted))
```

Output

```
recall score 0.7083333333333334
```

F1 Score

```
from sklearn.metrics import f1_score
print("f1 score",f1_score(y_test,predicted))
```

Output

```
f1 score 0.723404255319149
```

```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test,predicted))
```

Output

```
              precision    recall  f1-score   support

         0.0       0.84      0.86      0.85       168
         1.0       0.74      0.71      0.72        96

    accuracy                           0.80       264
   macro avg       0.79      0.78      0.79       264
weighted avg       0.80      0.80      0.80       264
```

```
# metrics are used to find accuracy or error
from sklearn import metrics
# using metrics module for accuracy calculation
print("ACCURACY of Logistic Regression Model: ", metrics.accuracy_score(y_test,
predicted))
```

Output

```
ACCURACY of Logistic Regression Model:  0.803030303030303
```

b. Random Forest Classifier

```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier

# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred = clf.predict(X_test)

#confusion matrix
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, y_pred))
```

Output

```
[[140  28]
 [ 20  76]]
```

```
# Precision Score
from sklearn.metrics import precision_score
print("Precision Score",precision_score(y_test,y_pred))
```

Output

```
Precision Score 0.7307692307692307
```

```
# Recall Score
from sklearn.metrics import recall_score
print("recall score",recall_score(y_test,y_pred))
```

Output

```
recall score 0.7916666666666666
```

```
# F1 Score
from sklearn.metrics import f1_score
print("f1 score",f1_score(y_test,y_pred))
```

Output

```
f1 score 0.76
```

```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

Output

```
              precision    recall  f1-score   support

         0.0       0.88      0.83      0.85       168
         1.0       0.73      0.79      0.76        96

    accuracy                           0.82       264
   macro avg       0.80      0.81      0.81       264
weighted avg       0.82      0.82      0.82       264
```

```
# metrics are used to find accuracy or error
from sklearn import metrics
# using metrics module for accuracy calculation
```

print("ACCURACY of Random Forest Classifier Model: ", metrics.accuracy_score(y_test, y_pred))

Output

```
ACCURACY of Random Forest Classifier Model:  0.8181818181818182
```

8. Implement the following models on the California House Pricing Dataset and determine the values of R2 score, the area under roc curve and root mean squared error for the test set.
a. Linear Regression with Polynomial Features
b. Random Forest Regressor

Preparing the data

checking for null values
data.isnull().mean() * 100

Output

```
longitude                      0.000000
latitude                       0.000000
housing_median_age             0.000000
total_rooms                    0.000000
total_bedrooms                 1.002907
population                     0.000000
households                     0.000000
median_income                  0.000000
median_house_value             0.000000
ocean_proximity                0.000000
diff_income_and_house_value    0.000000
dtype: float64
```

handling null values in total_bedrooms with the most frequent value in respective column
data["total_bedrooms"].fillna(data['total_bedrooms'].mode()[0],inplace=True)

#checking the null values handled or not
data["total_bedrooms"].isnull().mean() * 100

Output

```
0.0
```

data.info()

Output

```
data.info()
```

data.info()

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 20640 entries, 4861 to 9188
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   longitude                    20640 non-null  float64
 1   latitude                     20640 non-null  float64
 2   housing_median_age           20640 non-null  float64
 3   total_rooms                  20640 non-null  float64
 4   total_bedrooms               20640 non-null  float64
 5   population                   20640 non-null  float64
 6   households                   20640 non-null  float64
 7   median_income                20640 non-null  float64
 8   median_house_value           20640 non-null  float64
 9   ocean_proximity              20640 non-null  object
 10  diff_income_and_house_value  20640 non-null  float64
dtypes: float64(10), object(1)
memory usage: 1.9+ MB
```

```
data['ocean_proximity'].unique()
```

Output

```
array(['<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'NEAR BAY', 'ISLAND'],
      dtype=object)
```

```
#we need to convert categorical values by label encoding
# there are more than two categories, we have to use onehot encoding
data['ocean_proximity'].value_counts()
ocean_prox_df = pd.get_dummies(data['ocean_proximity'],drop_first=True)
ocean_prox_df.head()
```

Output

	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
4861	0	0	0	0
6688	1	0	0	0
16642	0	0	0	1
15661	0	0	1	0
15652	0	0	1	0

```
old_data = data.copy()
```

```
data.drop(['ocean_proximity','longitude','latitude','diff_income_and_house_value'],axis=1,inplace=True)
data.head()
```

Output

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
4861	29.0	515.0	229.0	2690.0	217.0	0.4999	500001.0
6688	28.0	238.0	58.0	142.0	31.0	0.4999	500001.0
16642	19.0	1540.0	715.0	1799.0	635.0	0.7025	500001.0
15661	27.0	1728.0	884.0	1211.0	752.0	0.8543	500001.0
15652	52.0	3260.0	1535.0	3260.0	1457.0	0.9000	500001.0

```
data = pd.concat([data,ocean_prox_df],axis=1)

data.head()
```

Output

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
4861	29.0	515.0	229.0	2690.0	217.0	0.4999	500001.0	0	0	0	0
6688	28.0	238.0	58.0	142.0	31.0	0.4999	500001.0	1	0	0	0
16642	19.0	1540.0	715.0	1799.0	635.0	0.7025	500001.0	0	0	0	1
15661	27.0	1728.0	884.0	1211.0	752.0	0.8543	500001.0	0	0	1	0
15652	52.0	3260.0	1535.0	3260.0	1457.0	0.9000	500001.0	0	0	1	0

Split the data

```
from sklearn.model_selection import train_test_split
# split the data for training and testing
```

```
X_train, X_test, y_train, y_test = train_test_split(data.drop('median_house_value',axis=1),
                                                    data['median_house_value'], test_size=0.30,
                                                    random_state=101)
```

a. Linear Regression with Polynomial Features

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

#model initialization
model = LinearRegression()

# initializing polynomial featuers
poly = PolynomialFeatures(degree=3)
#converting features into polyfeatures
X_ = poly.fit_transform(X_train)
Y_ = poly.fit_transform(y_train.values.reshape(-1,1))
# training the model
model.fit(X_,Y_)
```

Output

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

```
#preparing test data for predictions
testX = poly.fit_transform(X_test)
# predicting the output for test data
predicted = model.predict(testX)

# expected output for test data
expected = poly.fit_transform(y_test.values.reshape(-1,1))

from sklearn.metrics import r2_score
r2 = r2_score(expected, predicted)
print('r2 score is', r2)
```

Output

```
r2 score is 0.590661764648472
```

```
# example of calculate the root mean squared error
from sklearn.metrics import mean_squared_error
# calculate errors
errors = mean_squared_error(expected, predicted, squared=False)
# report error
print("root mean square error is :",errors)
```

Output

```
root mean square error is : 1.1921996852169048e+16
```

b. Random Forest Regressor

```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

 # create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 101)

# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

Output

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto',
max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2,
min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=101, verbose=0, warm_start=False)
```

```
# test the output by changing values
predicted = regressor.predict(X_test)

expected = y_test

from sklearn.metrics import r2_score
r2 = r2_score(expected, predicted)
print('r2 score is', r2)
```

Output

```
r2 score is 0.7091234171276952
```

```
# example of calculate the root mean squared error
from sklearn.metrics import mean_squared_error
# calculate errors
errors =mean_squared_error(expected, predicted,squared=False)
# report error
print("root mean square error is :",errors)
```

Output

```
root mean square error is : 62360.02542136252
```

1. Implement a single neural network and test for different logic gates.

```
#0r gate
import numpy as np
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)
# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test4)))
```

Output

```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

```
# And gate
import numpy as np
# define Unit Step Function
def unitStep(v):
	if v >= 0:
		return 1
	else:
		return 0

# design Perceptron Model
def perceptronModel(x, w, b):
	v = np.dot(w, x) + b
	y = unitStep(v)
	return y

# AND  Logic Function
# w1 = 1, w2 = 1, b = -1.5
def AND_logicFunction(x):
	w = np.array([1, 1])
	b = -1.5
	return perceptronModel(x, w, b)
# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```

Output

```
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
```

2. Write a program to train and test a Convolutional Neural Network to determine the number, given an image of a handwritten digit. Determine the training and validation accuracies of your model. (Train your model for 5 epochs).

```
from keras.datasets import mnist

# loading the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# let's print the shape of the dataset
```

Output

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)
```

Output

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
```

```
# keras imports for the dataset and building our neural network
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D
from keras.utils import np_utils


# Flattening the images from the 28x28 pixels to 1D 787 pixels
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalizing the data to help with the training
X_train /= 255
X_test /= 255

# one-hot encoding using keras' numpy-related utilities
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
```

```
# building a linear stack of layers with the sequential model
model = Sequential()
# hidden layer
model.add(Dense(100, input_shape=(784,), activation='relu'))
# output layer
model.add(Dense(10, activation='softmax'))

# looking at the model summary
model.summary()
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
```

```
Shape before one-hot encoding:  (60000,)
Shape after one-hot encoding:  (60000, 10)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               78500

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 3s 5ms/step - loss: 0.3805 -
accuracy: 0.8950 - val_loss: 0.2060 - val_accuracy: 0.9409
Epoch 2/10
469/469 [==============================] - 2s 5ms/step - loss: 0.1812 -
accuracy: 0.9477 - val_loss: 0.1493 - val_accuracy: 0.9566
Epoch 3/10
469/469 [==============================] - 2s 5ms/step - loss: 0.1334 -
accuracy: 0.9613 - val_loss: 0.1223 - val_accuracy: 0.9644
Epoch 4/10
469/469 [==============================] - 2s 5ms/step - loss: 0.1055 -
accuracy: 0.9699 - val_loss: 0.1059 - val_accuracy: 0.9693
Epoch 5/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0863 -
accuracy: 0.9753 - val_loss: 0.1025 - val_accuracy: 0.9697
Epoch 6/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0718 -
accuracy: 0.9796 - val_loss: 0.0951 - val_accuracy: 0.9721
Epoch 7/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0615 -
accuracy: 0.9822 - val_loss: 0.0865 - val_accuracy: 0.9735
Epoch 8/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0535 -
accuracy: 0.9851 - val_loss: 0.0800 - val_accuracy: 0.9761
Epoch 9/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0457 -
accuracy: 0.9868 - val_loss: 0.0829 - val_accuracy: 0.9754
```

```
Epoch 10/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0391 - accuracy: 0.9888 - val_loss: 0.0784 - val_accuracy: 0.9757
```

Output

```
<keras.callbacks.History at 0x7f6bd453df10>
```

```
# keras imports for the dataset and building our neural network
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
from keras.utils import np_utils

# to calculate accuracy
from sklearn.metrics import accuracy_score

# loading the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# building the input vector from the 28x28 pixels
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalizing the data to help with the training
X_train /= 255
X_test /= 255

# one-hot encoding using keras' numpy-related utilities
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)

# building a linear stack of layers with the sequential model
model = Sequential()
# convolutional layer
model.add(Conv2D(25, kernel_size=(3,3), strides=(1,1), padding='valid', activation='relu',
input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(1,1)))
# flatten output of conv
model.add(Flatten())
# hidden layer
model.add(Dense(100, activation='relu'))
# output layer
model.add(Dense(10, activation='softmax'))

# compiling the sequential model
```

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
```

```
Shape before one-hot encoding:  (60000,)
Shape after one-hot encoding:  (60000, 10)
Epoch 1/10
469/469 [==============================] - 41s 86ms/step - loss: 0.2190 - accuracy: 0.9367 - val_loss: 0.0841 - val_accuracy: 0.9768
Epoch 2/10
469/469 [==============================] - 42s 90ms/step - loss: 0.0659 - accuracy: 0.9804 - val_loss: 0.0538 - val_accuracy: 0.9820
Epoch 3/10
469/469 [==============================] - 40s 84ms/step - loss: 0.0376 - accuracy: 0.9891 - val_loss: 0.0527 - val_accuracy: 0.9827
Epoch 4/10
469/469 [==============================] - 40s 86ms/step - loss: 0.0243 - accuracy: 0.9926 - val_loss: 0.0563 - val_accuracy: 0.9806
Epoch 5/10
469/469 [==============================] - 40s 84ms/step - loss: 0.0152 - accuracy: 0.9956 - val_loss: 0.0598 - val_accuracy: 0.9834
Epoch 6/10
469/469 [==============================] - 40s 85ms/step - loss: 0.0104 - accuracy: 0.9968 - val_loss: 0.0579 - val_accuracy: 0.9826
Epoch 7/10
469/469 [==============================] - 40s 85ms/step - loss: 0.0070 - accuracy: 0.9983 - val_loss: 0.0661 - val_accuracy: 0.9828
Epoch 8/10
469/469 [==============================] - 40s 85ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.0542 - val_accuracy: 0.9842
Epoch 9/10
469/469 [==============================] - 40s 85ms/step - loss: 0.0046 - accuracy: 0.9989 - val_loss: 0.0674 - val_accuracy: 0.9833
Epoch 10/10
469/469 [==============================] - 40s 85ms/step - loss: 0.0052 - accuracy: 0.9985 - val_loss: 0.0720 - val_accuracy: 0.9818
```

Output

```
<keras.callbacks.History at 0x7f6bcfde47d0>
```