VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by:
Anoshor B. Paul
(1BM21CS024)

Under the Guidance of
Sheetal V A
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
March 2024 - June 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Machine Learning**" carried out by **Anoshor B. Paul (1BM21CS024),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of **Machine Learning - (22CS6PCMAL)** work prescribed for the said degree.

Prof. Sheeta VA
Associate Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table Of Contents

1. Course Outcomes

CO1: Apply machine learning techniques in computing systems.

CO2: Evaluate the model using metrics.

CO3: Design a model using machine learning to solve a problem.

CO4: Conduct experiments to solve real-world problems using appropriate machine learning techniques

2. Experiments

2.1 Experiment - 1

2.1.1 Question:

Write a python program to import and export data using Pandas library functions.

2.1.2 Code with Output:

```
import pandas as pd
import numpy as np

california_housing_train_data = pd.read_csv("/content/sample_data/california_housing_train.csv")

# View the first 5 rows

california_housing_train_data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

```
import pandas as pd
import numpy as np

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Define the column names

col_names = ["sepal_length_in_cm",
"sepal_width_in_cm",
"petal_length_in_cm",
"petal_width_in_cm",
"class"]

# Read data from URL

iris_data = pd.read_csv(url, names=col_names)
iris_data.head()
```

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
[3] iris_data.to_csv("cleaned_iris_data.csv")
```

2.2 Experiment - 2

2.2.1 Question:

End-to-end ML Project.

2.2.2 Code with Output:

Download the Data

In [1]:
```
import os
import tarfile
import urllib
```

In [2]:
```
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("data", "01")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
```

In [3]:
```
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(name=housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(url=housing_url, filename=tgz_path)
    housing_tgz = tarfile.open(name=tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

Download the data:

In [4]:
```
fetch_housing_data()
```

Load the data using pandas:

In [5]:
```
import pandas as pd
```

In [6]:
```
def load_housing_data(housing_path=HOUSING_PATH):
    data_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(data_path)
```

Data Structure

In [7]:
```
housing = load_housing_data()
```

In [8]:
```
housing.head()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

In [9]:
```
housing.info()
```

```
<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
```

There exist $20,640$ instances (rows) in the dataset. Which means that it is fairly small data sample by machine learning standards.

207 districts are missing the `total_bedrooms` attribute, we will need to take care of this later.

On the other hand, all attributes are numerical, except `ocean_proximity`
Since we noticed repeated `ocean_proximity` values for the top 5 rows, we suspect that it is a categorical column, let's check it out:

In [10]:
```
housing['ocean_proximity'].value_counts()
```

Out[10]:
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

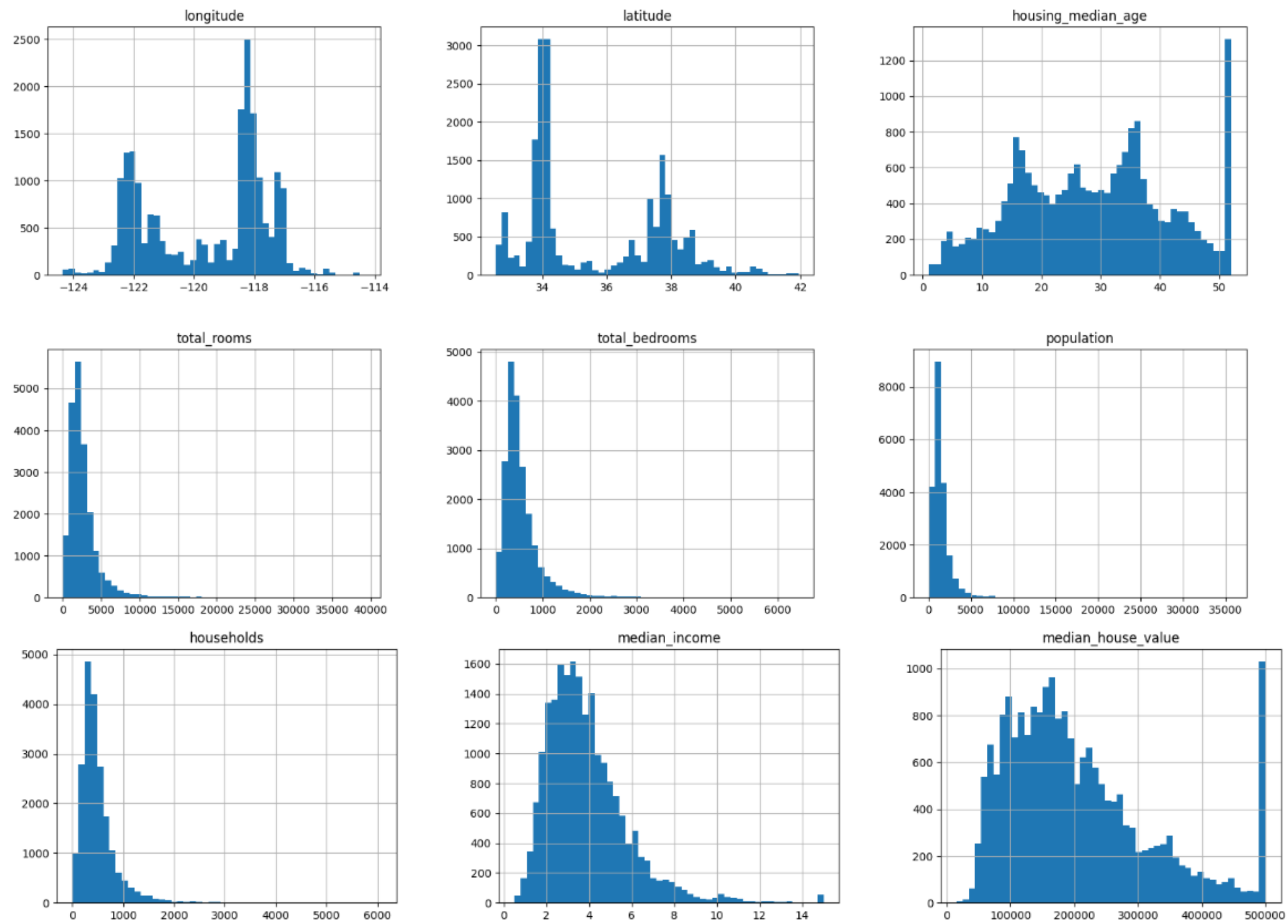
In [11]:
```
housing.describe()
```

Out[11]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	me
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	

```
In [12]: import matplotlib.pyplot as plt
         import seaborn as sns
```

```
In [13]: housing.hist(bins=50, figsize=(20,15))
         plt.show()
```



Create a Test Set

```
In [14]: import numpy as np
```

```
In [15]: def split_train_test(data, test_ratio=0.2):
             shuffled_indices = np.random.permutation(len(data))
             test_set_size = int(len(data) * test_ratio)
             test_indices = shuffled_indices[:test_set_size]
             train_indices = shuffled_indices[test_set_size:]
             return data.iloc[train_indices], data.iloc[test_indices]
```

```
In [16]: # you can then use the function like this
         train_set, test_set = split_train_test(data=housing)
         len(train_set), len(test_set)

Out[16]: (16512, 4128)
```

In [17]:

```
from zlib import crc32
```

In [18]:

```
def test_set_check(identifier, test_ratio=.2):
    total_size = 2**32
    hex_repr = crc32(np.int64(identifier)) & 0xffffffff
    in_test = hex_repr < (test_ratio * total_size)
    return in_test
```

In [19]:

```
[test_set_check(i) for i in range(10)]
```

Out[19]:

```
[False, False, True, False, False, True, False, False, False, False]
```

In [20]:

```
def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

Unfortunately, the housing dataset does not have an identifier, column. We will use the row index as an identifier:

In [21]:

```
housing_with_id = housing.reset_index()
```

In [22]:

```
train_set, test_set = split_train_test_by_id(data=housing_with_id, test_ratio=0.2, id_column="index")
train_set.shape, test_set.shape
```

Out[22]:

```
((16512, 11), (4128, 11))
```

In [23]:

```
def from_Z_to_N(z):
    if z >= 0:
        n = 2 * z
    else:
        n = -2 * z - 1
    return n
```

In [24]:

```
def cantor_pairing(n1, n2):
    n = ( ( (n1 + n2) * (n1 + n2 + 1) ) / 2 ) + n2
    return n
```

In [25]:

```
def lat_lon_to_index(lat, lon):
    lat, lon = int(lat*100), int(lon*100)
    lat, lon = from_Z_to_N(lat), from_Z_to_N(lon)
    index = cantor_pairing(lat, lon)
    return np.int64(index)
```

In [26]:

```
housing['id'] = housing.apply(lambda row: lat_lon_to_index(row['latitude'], row['longitude']), axis=1)
```

In [27]:
```python
housing['id'].value_counts()
```

Out[27]:
```
id
513289261    24
513481522    20
513417431    18
513353344    18
463609694    14
             ..
513032709     1
513417159     1
519523778     1
519459311     1
515855387     1
Name: count, Length: 11573, dtype: int64
```

We still get duplicate indexes, and at the same time, we have duplicate (lat,lon) tuples as follows:

In [28]:
```python
housing.groupby(by=['longitude', 'latitude']).count()['total_rooms'].sort_values()
```

Out[28]:
```
longitude  latitude
-124.35    40.54        1
-118.90    34.41        1
           35.26        1
           35.41        1
-118.89    34.22        1
                       ..
-122.41    37.75       10
-122.42    37.75       10
-122.44    37.78       11
-122.42    37.80       11
-122.41    37.80       15
Name: total_rooms, Length: 12590, dtype: int64
```

In [29]:
```python
del(housing['id'])
```

In [30]:
```python
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
```

In [31]:
```python
train_set, test_set = split_train_test_by_id(data=housing_with_id, test_ratio=0.2, id_column='id')
train_set.shape, test_set.shape
```

Out[31]:
```
((16322, 12), (4318, 12))
```

Split the dataset

In [32]:
```python
from sklearn.model_selection import train_test_split
```

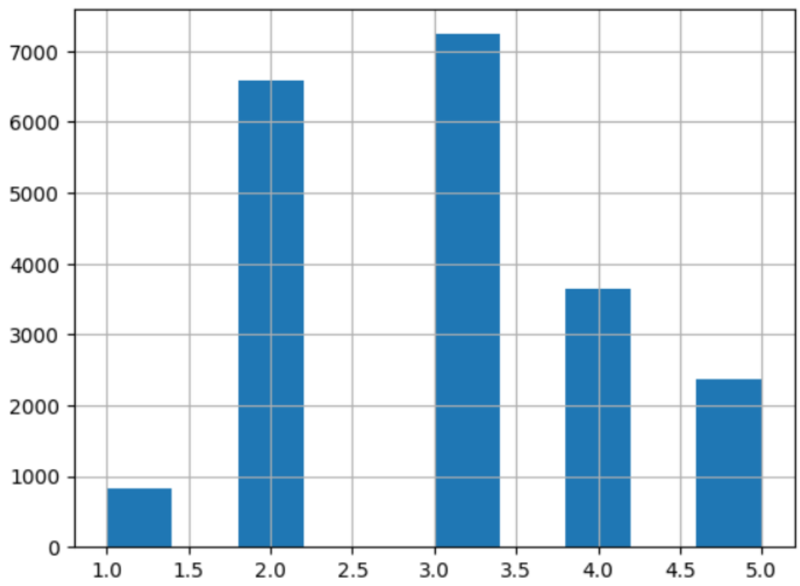
In [33]:
```python
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
train_set.shape, test_set.shape
```

Out[33]:
```
((16512, 10), (4128, 10))
```

In [34]:
```python
housing['income_cat'] = pd.cut(x=housing['median_income'], bins=[0, 1.5, 3, 4.5, 6, np.inf], labels=[1, 2, 3, 4, 5])
```

In [35]:
```python
# visualize the categories
housing['income_cat'].hist()
```

Out[35]: `<Axes: >`



Now we are ready to do stratified sampling based on income category:

In [36]:
```python
from sklearn.model_selection import StratifiedShuffleSplit
```

In [37]:
```python
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
```

In [38]:
```python
for train_index, test_index in split.split(X=housing, y=housing['income_cat']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

checking the proportions of income categories in the test set:

In [39]:
```python
strat_test_set['income_cat'].value_counts() / len(strat_test_set)
```

Out[39]:
```
income_cat
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: count, dtype: float64
```

Now that we have a test set that is representative of `income_cat`'s distribution, it's time to remove it:

In [40]:
```python
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
```

3.Discover & Visualize the Data to Gain Insights

Exploring the training set:

In [41]:
```
strat_train_set.shape, strat_test_set.shape
```

Out[41]: ((16512, 10), (4128, 10))

In [43]:
```
strat_test_set.reset_index().to_feather(fname='data/01/strat_test_set.f')
```

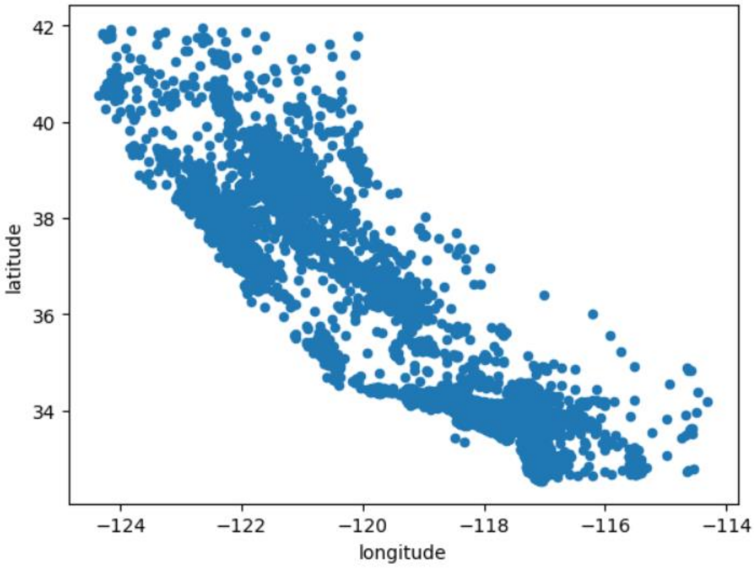
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-044385fea95e> in <cell line: 1>()
----> 1 strat_test_set.reset_index().to_feather(fname='data/01/strat_test_set.f')

TypeError: DataFrame.to_feather() missing 1 required positional argument: 'path'
```

Let's create a copy of the training set to test without harming the original one:

In [44]:
```
housing = strat_train_set.copy(); housing.shape
```

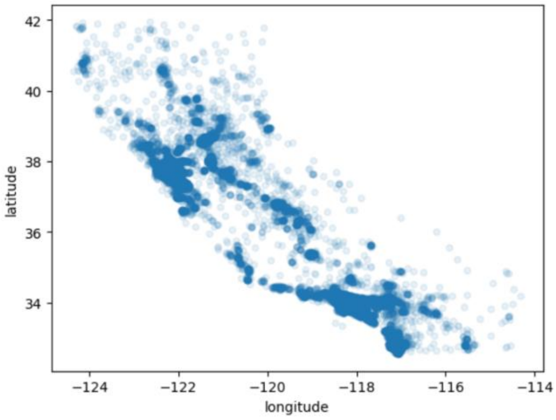
Out[44]: (16512, 10)

In [45]:
```
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()
```



This looks like california, but other than that, we can't really see any other pattern. Setting the alpha to 0.1 makes it much easier to estimate densities:

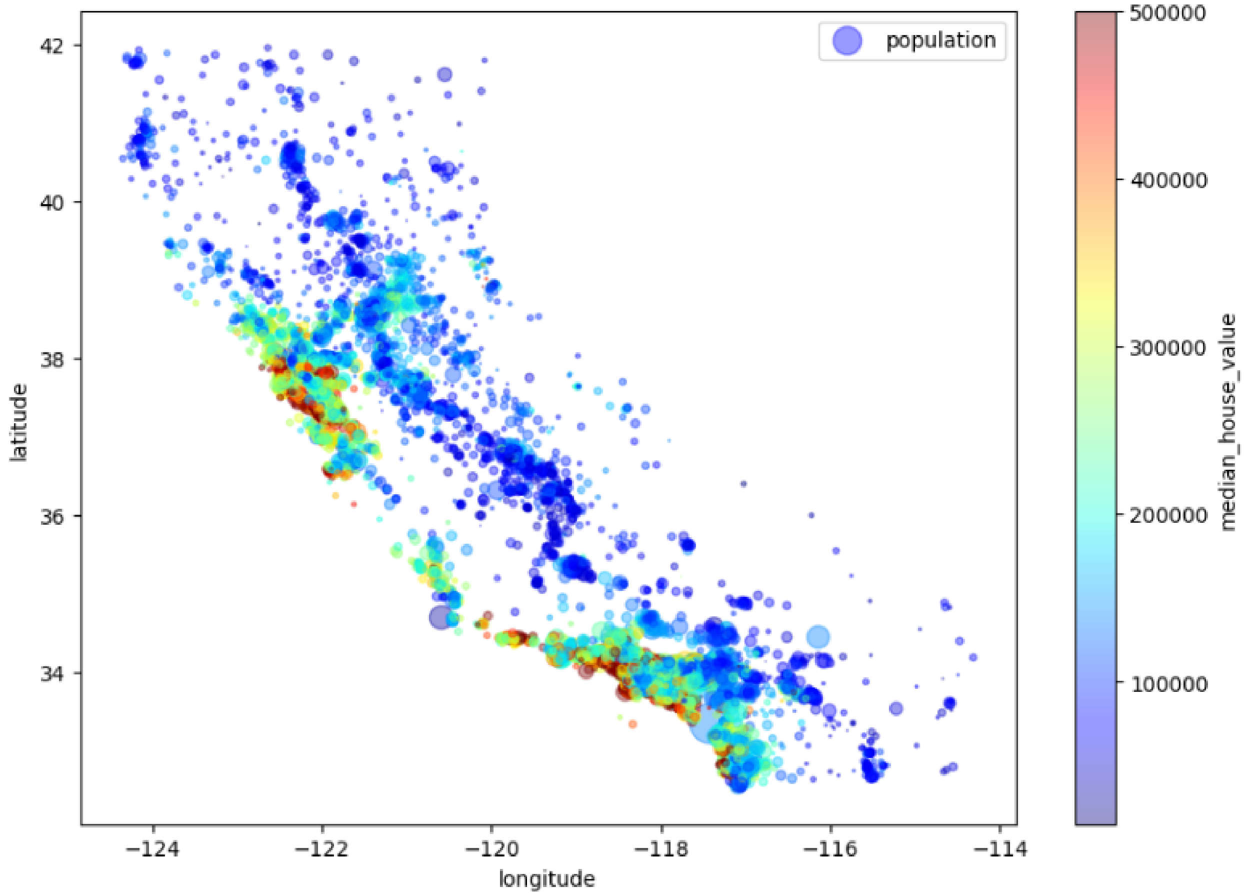
In [46]:
```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)
plt.show()
```



In the following figure, the radius of each circle represents the district's population (option `s`). The color represents the price (option `c`). We will also use a pre-defined color map called **jet** (option `cmap`) which ranges from `blue` (low levels) to `red` (high level).

In [47]:
```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=.4, s=housing['population']/100.,
             label='population', figsize=(10, 7), c='median_house_value', cmap=plt.get_cmap(name='jet'), colorbar=True
plt.legend()
```

Out[47]: <matplotlib.legend.Legend at 0x7a8306385630>



Experimenting with Attribute Combinations

We may want to transform tail heavy distributions using the logarithm function ($log(.)$).

In [56]:
```
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms']/housing['total_rooms']
housing['population_per_household'] = housing['population']/housing['households']
```

Look at the correlation matrix again:

In [57]:
```
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

We notice that `bedrooms_per_room` is much more correlated with `median_house_value` . meaning that the more expensive the house, the less the `bedrooms per room` ratio. `rooms_per_household` have a moderate positive correlation with `median_house_value` , the more expensive a house is, the more rooms it will have.

4. Prepare the Data for Machine Learning Algorithms

In [58]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
housing.shape, housing_labels.shape
```

Out[58]:
```
((16512, 9), (16512,))
```

Data Cleaning

We saw earlier that `total_bedrooms` have missing values, we have 3 options:

1. Get rid of the corresponding districts
   - `housing.dropna(subset='total_bedrooms')`
2. Get rid of the whole attribute (feature)
   - `housing.drop('total_bedrooms', axis=1)`
3. Set the missing values to some value (zero, mean, median, regressor preds,...)
   - `median = housing['total_bedrooms'].median()`
   - `housing['total_bedrooms'].fillna(median, inplace=True)`

We can also use `scikit-learn` 's `SimpleImputer` :

In [59]:
```
from sklearn.impute import SimpleImputer
```

In [60]:
```
imputer = SimpleImputer(strategy='median')
```

Since the imputer can only work on numerical attributes, we need to create a copy of the dataFrame without the `OCEAN_PROXIMITY` text attribute:

In [61]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
```

In [61]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
```


Now we can just fit the imputer to the dataframe:

In [62]:
```
imputer.fit(housing_num)
```

Out[62]:
```
SimpleImputer(strategy='median')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

The imputer has calculated the median of all attributes and stored them in `.statistics_` .

In [63]:
```
imputer.statistics_
```

Out[63]:
```
array([-118.51    ,   34.26    ,   29.     , 2119.      ,  433.      ,
        1164.    ,  408.      ,    3.54155])
```

In [64]:
```
housing_num.median().values
```

Out[64]:
```
array([-118.51    ,   34.26    ,   29.     , 2119.      ,  433.      ,
        1164.    ,  408.      ,    3.54155])
```

Now we can use the "trained or fitted" imputer to transform the numerical attributes by replacing missing values with their corresponding medians:

In [65]:
```
X = imputer.transform(housing_num)
X.shape
```

Out[65]:
```
(16512, 8)
```

The result is a numpy array containing the transformed features. If we want to put it back into a Pandas DataFrame, it's simple:

In [66]:
```
housing_tr = pd.DataFrame(data=X, index=housing_num.index, columns=housing_num.columns)
housing_tr.head()
```

Out[66]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	2.1736
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	6.3373
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	2.8750
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	2.2264
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	580.0	4.4964

Handling Text & Categorical Attributes

In [67]:
```
housing_cat = housing[['ocean_proximity']]
housing_cat.head(10)
```

Out[67]:

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

In [68]:
```
housing_cat['ocean_proximity'].value_counts()
```

Out[68]:
```
ocean_proximity
<1H OCEAN     7277
INLAND        5262
NEAR OCEAN    2124
NEAR BAY      1847
ISLAND           2
Name: count, dtype: int64
```

Most ML algorithms prefer to work with numbers, so let's convert the text into **ordinal categorical** numbers:

In [69]:
```
from sklearn.preprocessing import OrdinalEncoder
```

In [70]:
```
ordinal_encoder = OrdinalEncoder()
```

In [71]:
```
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat.values)
housing_cat_encoded.shape
```

Out[71]:
```
(16512, 1)
```

In [72]:
```
housing_cat_encoded[:10]
```

Out[72]:
```
array([[1.],
       [4.],
       [1.],
       [4.],
       [0.],
       [3.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

We can get the list of categories using the `categories_` attribute of the `OrdinalEncoder`:

In [73]:
```
ordinal_encoder.categories_
```

Out[73]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

One issue with this representation is that the encoder will assume that two nearby categories are more similar than distant ones, but this is not the case for us (ex. categories 0 and 4 are clearly more similar than 0 and 1). To fix this issue, we create one binary attribute per category:

- One attribute is equal to `1` if the category is equal to `<1H OCEAN` and `0` otherwise.
- One attribute is equal to `1` if the category is equal to `INLAND` and `0` otherwise.
- ...

This is called 1-hot encoding because, for any row, only one binary attribute will be equal to 1 (hot), while the others are 0s (cold).

The new attributes are sometimes called dummy attributes, let's create them:

In [75]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [76]:
```
one_hot_encoder = OneHotEncoder()
```

In [77]:
```
housing_cat_1hot = one_hot_encoder.fit_transform(housing_cat.values)
housing_cat_1hot
```

Out[77]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

The output is a sparse scipy matrix instead of a numpy array. If we use numpy, we have to store all of the zeros in memory, comprising of most of the array. Instead, we store the information as a Scipy sparse matrix which only stores the locations of the non-zeros (which is more efficient).

We can mostly use it as a normal `2D` array, but if we want to convert it into a dense numpy array:

In [78]:
```
housing_cat_1hot.toarray()
```

Out[78]:
```
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

In [79]:
```
one_hot_encoder.categories_
```

Out[79]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Custom Transformers

Although `scikit-learn` provide many useful transformers, we will need to write our own for custom tasks such as data cleanup or feature engineering. We'll want our transformer to easily work with other scikit-learn functionalities (such as Pipelines).

All we need to do is create a class with 3 methods: `fit`, `transform`, `fit_transform`. We can get `fit_transform` for free by adding `TransformerMixin` as a base class.

If we add `BaseEstimator` as another base class & avoid the use of *args* and *kwargs*, we get two extra methods (`.get_params()` & `.set_params()`).

In [80]:
```
from sklearn.base import TransformerMixin, BaseEstimator
```

In [81]:
```
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6
```

In [82]:
```
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y=None):
        return self  # We don't have any internal parameters. Only interested in transforming data.

    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

In [83]:
```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
```

In [84]:
```
housing_extra_attribs = attr_adder.transform(housing.values)
```

The `add_bedrooms_per_room` hyper-parameter will easily help us find out whether adding the attributes helps the ML algorithm or not.

We can add hyper-parameters to control any pre-processing step that we're not sure about. The more we automate these data preprocessing steps, the more combinations we get to try out.

Transformation Pipelines

In [85]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [86]:
```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler())
])
```

In [87]:
```
housing_num_tr = num_pipeline.fit_transform(housing_num)
housing_num_tr.shape
```

Out[87]:
```
(16512, 11)
```

So far, we have handeled categorical/continuous columns separately. It would be better if we had a single transformer that is able to transform all columns.

`ColumnTransformer` s to the rescue:

In [88]:
```
from sklearn.compose import ColumnTransformer
```

In [89]:
```
num_attribs = housing_num.columns.tolist()
cat_attribs = ["ocean_proximity"]
```

In [90]:
```
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs)
])
```

In [91]:
```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared.shape
```

5. Select & Train a Model

Training & Evaluating on the Training Set

Train a Linear Regression model:

In [92]:
```
from sklearn.linear_model import LinearRegression
```

In [93]:
```
lin_reg = LinearRegression()
```

In [94]:
```
lin_reg.fit(X=housing_prepared, y=housing_labels)
```

Out[94]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Let's try the model on a few instances from the training set:

In [95]:
```
some_data = housing.iloc[:5]
```

In [96]:
```
some_labels = housing_labels.iloc[:5]
```

In [97]:
```
some_data_prepared = full_pipeline.transform(some_data)
```

In [98]:
```
print("Predictions: ", lin_reg.predict(some_data_prepared))
```

```
Predictions:  [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

In [99]:
```
print("Labels: ", some_labels.tolist())
```

```
Labels:  [72100.0, 279600.0, 82700.0, 112500.0, 238300.0]
```

It works, although the predictions are not exactly accurate.

Let's measure the performance of our model using the RMSE metric.

In [100…
```
from sklearn.metrics import mean_squared_error
```

In [101…
```
housing_predictions = lin_reg.predict(housing_prepared)
```

In [102…
```
lin_mse = mean_squared_error(housing_labels, housing_predictions)
```

In [103…
```
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[103…
```
68627.87390018745
```

Most districts median housing values range between 120K to 265K, so an average error of 68K is not good.

This is an example of a model overfitting the data. When this happens, it can mean two things:

- The features do not provide enough information to make better predictions.
- The model is not powerful enough, meaning its hypothesis space is narrow.

The main ways to tackle underfitting:

- To feed the model better features.
- To select a more powerful model.
- To loosen the model's restrictions.

This model is not regularized, which rules out the last option. We could try to input more features, but let's start by testing a more powerful model.

Let's try out `DecisionTreeRegressor`, this is a powerful model, capable of finding non-linear relationships within the data:

In [104…
```
from sklearn.tree import DecisionTreeRegressor
```

In [105…
```
tree_reg = DecisionTreeRegressor()
```

In [106…
```
tree_reg.fit(X=housing_prepared, y=housing_labels)
```

Out[106…
```
DecisionTreeRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [107…
```
housing_predictions = tree_reg.predict(housing_prepared)
```

In [108…
```
tree_mse = mean_squared_error(y_true=housing_labels, y_pred=housing_predictions)
```

In [109…
```
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[109…
```
0.0
```

Better Evaluation using Cross-Validation

One way to evaluate our model is to use `train_test_split()` again on the training set, extract a validation set and evaluate our iterative models on it.

A great alternative is to use K-fold cross-validation. We randomly split the training data into 10 folds, we iteratively train the model on 9 folds and evaluate on 1, doing this 10 times.

We will endup with `10` metric scores:

In [110…
```
from sklearn.model_selection import cross_val_score
```

In [111…
```
scores = cross_val_score(estimator=tree_reg, X=housing_prepared,
                         y=housing_labels, scoring='neg_mean_squared_error', cv=10)
```

In [112…
```
tree_rmse_scores = np.sqrt(-scores)
```

`scikit-learn`'s cross validation features expect a utility function (the greater the better) rather than a cost function (the lower the better). That's why we used `ned_mean_squared_error` and we negated it at RMSE evaluation.

In [113…
```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard Deviation:", scores.std())
```

In [114…
```python
display_scores(tree_rmse_scores)
```

```
Scores: [73420.18119578 69564.42303171 68891.37403651 71450.13832167
 69371.93163844 77144.32132592 70645.53949428 73310.3218479
 68484.47299548 70726.35627711]
Mean: 71300.90601648012
Standard Deviation: 2528.456433119772
```

The decision tree seems to perform worse than the linear regression model!

We should notice that cross validation allows us to not only get an estimate of the performance of your model (mean), but how precise it is (std). We would not have this estimation if we used only one validation set. However, cross-validation comes at the cost of training the model several times, which is not always possible.

Let's compute the same scores for the linear regression model just to be sure:

In [115…
```python
scores = cross_val_score(estimator=lin_reg, X=housing_prepared,
                         y=housing_labels, scoring='neg_mean_squared_error', cv=10)
```

In [116…
```python
lin_rmse_scores = np.sqrt(-scores)
```

In [117…
```python
display_scores(lin_rmse_scores)
```

```
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard Deviation: 2880.3282098180634
```

That's right! the decision tree model is overfitting so badly that it performs worse than the linear regression model.

Let's try one last model now, the random forest regressor. Random forests work by training many decision trees on random feature subsets then average out their predictions.

Building a model on top of many other models is called **Ensemble Learning**.

In [118…
```python
from sklearn.ensemble import RandomForestRegressor
```

In [119…
```python
forest_reg = RandomForestRegressor()
```

In [120…
```python
forest_reg.fit(X=housing_prepared, y=housing_labels)
```

Out[120… RandomForestRegressor()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [121…
```python
forest_mse = mean_squared_error(y_true=housing_labels, y_pred=forest_reg.predict(X=housing_prepared))
```

In [122…
```python
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[122… 18677.177813034952

In [ ]:
```python
scores = cross_val_score(estimator=forest_reg, X=housing_prepared,
                         y=housing_labels, scoring='neg_mean_squared_error', cv=10)
```

In [ ]:
```python
forest_rmse_scores = np.sqrt(-scores)
```

In [ ]:
```python
display_scores(scores=forest_rmse_scores)
```

This is much better, random forests seem very promising. We should notice, however, that the RMSE on the training set is still much lower then the validation RMSE, meaning the model overfitted, but not as badly as the decision tree model. Possible solutions to overfitting are:

- Getting more training data
- Simplifying the model
- Regularizing the model

We should save any model after training so that we can come back to it at any time you want. We make sure to save both the hyper-parameters and the parameters (weights) of the model. We can easily save scikit-learn models using Python's `joblib`:

In [ ]:
```
import joblib
```

In [ ]:
```
joblib.dump(value=forest_reg, filename='models/01/forest_reg.m')
```

In [ ]:
```
# & later
forest_reg = joblib.load(filename='models/01/forest_reg.m')
```

6. Fine-Tune Your Model

Grid Search

In [ ]:
```
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}
]
```

In [ ]:
```
forest_reg = RandomForestRegressor()
```

In [ ]:
```
grid_search = GridSearchCV(estimator=forest_reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5, return_trai
```

In [ ]:
```
grid_search.fit(X=housing_prepared, y=housing_labels)
```

If we can't guess an initial quality search grids, we can start with powers of 10 then zoom in once we have the best estimate.

The model will first explore 3×4 combinations of hyper-parameters, then jump to the 2nd hyper-parameter space and try $1 \times 2 \times 3$. For each combination, it will train 5 times using the cross validation strategy, all in all: It will train **90** different model variations.

In [ ]:
```
grid_search.best_params_
```

We can also get the best estimator directly:

In [ ]:
```
grid_search.best_estimator_
```

When `GridSearchCV` finds the best estimator, it will retrain it on the whole training set. This can be controlled by the parameter `refit=True` (by default)

In [ ]:
```
cvres = grid_search.cv_results_
```

In [ ]:
```
for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
```

In this example, the best hyper-parameter combination is: `50110.7370892457 {'max_features': 6, 'n_estimators': 30}` with an average RMSE of 50110. The model performs slightly better than a random forest with default hyper-parameters.

Randomized Search

The grid search is fine when you're exploring a few hyper-parameter combinations, but when the search space is big though, it is better to use `RandomizedSearchCV` instead. It works almost in the same way of a grid search, but it try out a limited randomly selected number of hyper-paraemeters for each iteration. This approach has two main benefits:

- If we let this approach run for 1,000 iterations, it will explore 1,000 values for each hyper-parameters, instead of combining each unique value.
- By setting the number of iterations, we can control computing resources much more effectively than doing Grid search.

Ensemble Methods

Another way to fine-tune your model is to combine the models that work best. Usually, the ensemble model will perform better than any part of the model, especially if its models are producing different errors.

Analyze the best models & their errors

In [ ]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
(feature_importances*100).astype(int)
```

In [ ]:
```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
```

In [ ]:
```
cat_encoder = full_pipeline.named_transformers_['cat']
```

In [ ]:
```
cat_one_hot_attributes = cat_encoder.categories_[0].tolist()
```

In [ ]:
```
attributes = num_attribs + extra_attribs + cat_one_hot_attributes
```

In [ ]:
```
# sorted(zip(feature_importances, attributes), reverse=True)
dict(zip(feature_importances, attributes))
```

With this information, we might want to start dropping some of the attributes to simplify the model (ex. only one `ocean_proximity` value is important).

Evaluate your system on the test set

After tweaking the system for a while, we finally have a model that can be evaluated on the test set. There is nothing special about this process, we reproduce the same steps you used with training data to benchmark the model.

However, we should call `transform()`, and not `fit_transform()`.

In [ ]:
```
final_model = grid_search.best_estimator_
```

In [ ]:
```
X_test = strat_test_set.drop(labels='median_house_value', axis=1)
y_test = strat_test_set['median_house_value'].copy()
```

In [ ]:
```
X_test_prepared = full_pipeline.transform(X=X_test)
```

In [ ]:
```
final_predictions = final_model.predict(X=X_test_prepared)
```

In [ ]:
```
final_mse = mean_squared_error(y_true=y_test, y_pred=final_predictions)
```

In [ ]:
```
final_rmse = np.sqrt(final_mse)
final_rmse
```

In some cases, such a point estimate of the generalization error won't be enough for us to launch it in production. We might want to create a confidence interval of 95% around the metric.

In some cases, such a point estimate of the generalization error won't be enough for us to launch it in production. We might want to create a confidence interval of 95% around the metric.

For this, we use the individual predictions for each test set element.

In [ ]:
```
from scipy import stats
```

In [ ]:
```
confidence = .95
```

In [ ]:
```
squared_errors = (y_test - final_predictions) ** 2
```

In [ ]:
```
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1, loc=squared_errors.mean(), scale=stats.sem(squared_errors)))
```

If we do a lot of hyper-parameter fine-tuning, we will endup with a slightly worse performance on the test set because we will sometimes overfit to the changing validation set. This didn't happen now, but when it happens, resist the temptation to go back and do more fine-tuning to have better results for the test set.

In our case with the California dataset, our system didn't actually beat the experts system (with 20% error). But management still decided to launch the service to free some time for its experts to work on other tasks.

7. Launch, Monitor, & Maintain your system

2.3 Experiment - 3

2.3.1 Question:

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

2.3.2 Code with Output:

In [1]:
```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

In [22]:
```
df=pd.read_csv('/content/play_tennis.csv')
df = df.drop('day',axis=1)
```

In [23]:
```
df.head(14)
```

Out[23]:

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

In [24]:
```
print(f'Rows: {df.shape[0]}, Columns: {df.shape[1]}')
```

```
Rows: 14, Columns: 5
```

In [25]:
```
print(df.columns)
```

```
Index(['outlook', 'temp', 'humidity', 'wind', 'play'], dtype='object')
```

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   outlook   14 non-null     object
 1   temp      14 non-null     object
 2   humidity  14 non-null     object
 3   wind      14 non-null     object
 4   play      14 non-null     object
dtypes: object(5)
memory usage: 688.0+ bytes
```

In [64]:
```
df.describe()
```

Out[64]:

	outlook	temp	humidity	wind	play
count	14	14	14	14	14
unique	3	3	2	2	2
top	Sunny	Mild	High	Weak	Yes
freq	5	6	7	8	9

In [63]:
```
df.isnull()
```

Out[63]:

	outlook	temp	humidity	wind	play
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	False	False
10	False	False	False	False	False
11	False	False	False	False	False
12	False	False	False	False	False
13	False	False	False	False	False

All values are FALSE for isnull(). Therefore no data cleaning is required.

In [29]:
```
# Entropy

def find_entropy(df):
    #target column
    target = df.keys()[-1]
    entropy = 0
    values = df[target].unique()
    #calc entropy
    for value in values:
        fraction = df[target].value_counts()[value]/len(df[target])
        entropy += -fraction*np.log2(fraction)
    return entropy
```

In [30]:
```
# Average Information

def average_information(df,attribute):
  target = df.keys()[-1]   #target column
  target_variables = df[target].unique()  #This gives all 'Yes' and 'No'
  variables = df[attribute].unique()    #This gives different features in that attribute (like 'Hot','Cold' in Temperature)
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[target] ==target_variable])
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
  return abs(entropy2)
```

In [31]:

```python
# Information Gain

def find_winner(df):
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-average_information(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

In [32]:

```python
def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
```

In [33]:

```python
def buildTree(df,tree=None):
    target = df.keys()[-1]   #target column

    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

    #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable[target],return_counts=True)

        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling the function recursively

    return tree
```

In [34]:

```python
tree = buildTree(df)
```

In [35]:

```python
import pprint
pprint.pprint(tree)
```

```
{'outlook': {'Overcast': 'Yes',
             'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

2.4 Experiment - 4

2.4.1 Question:

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

2.4.2 Code with Output:

Simple Linear Regression

Link - 1 (https://github.com/shuv50/Data-Science/blob/main/simple_Linear_Regression.ipynb)

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression
```

```
# Get dataset
df_sal = pd.read_csv('/content/50_Startups.csv')
df_sal.head()
```

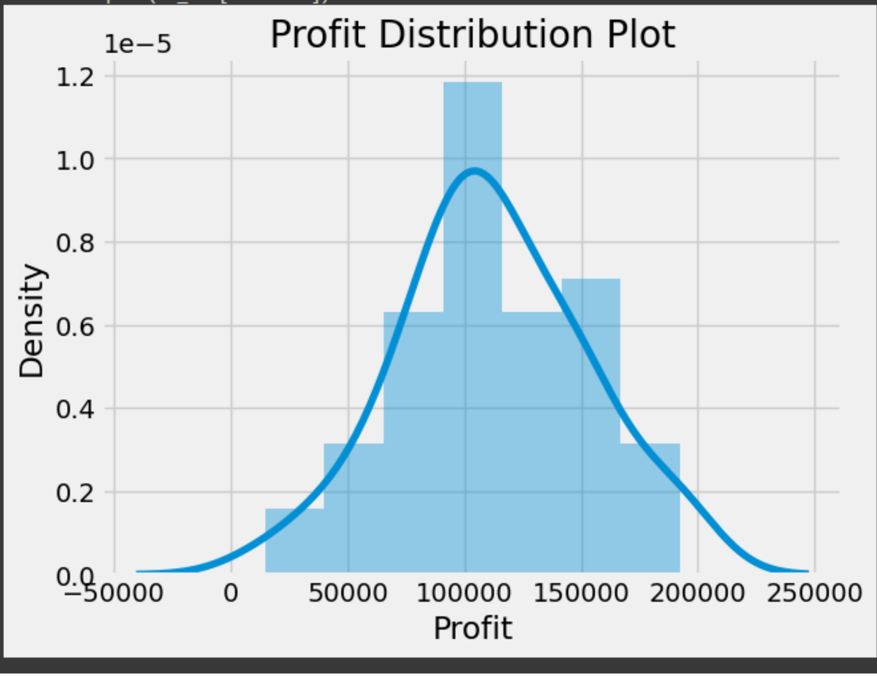
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
# Describe data
df_sal.describe()
```

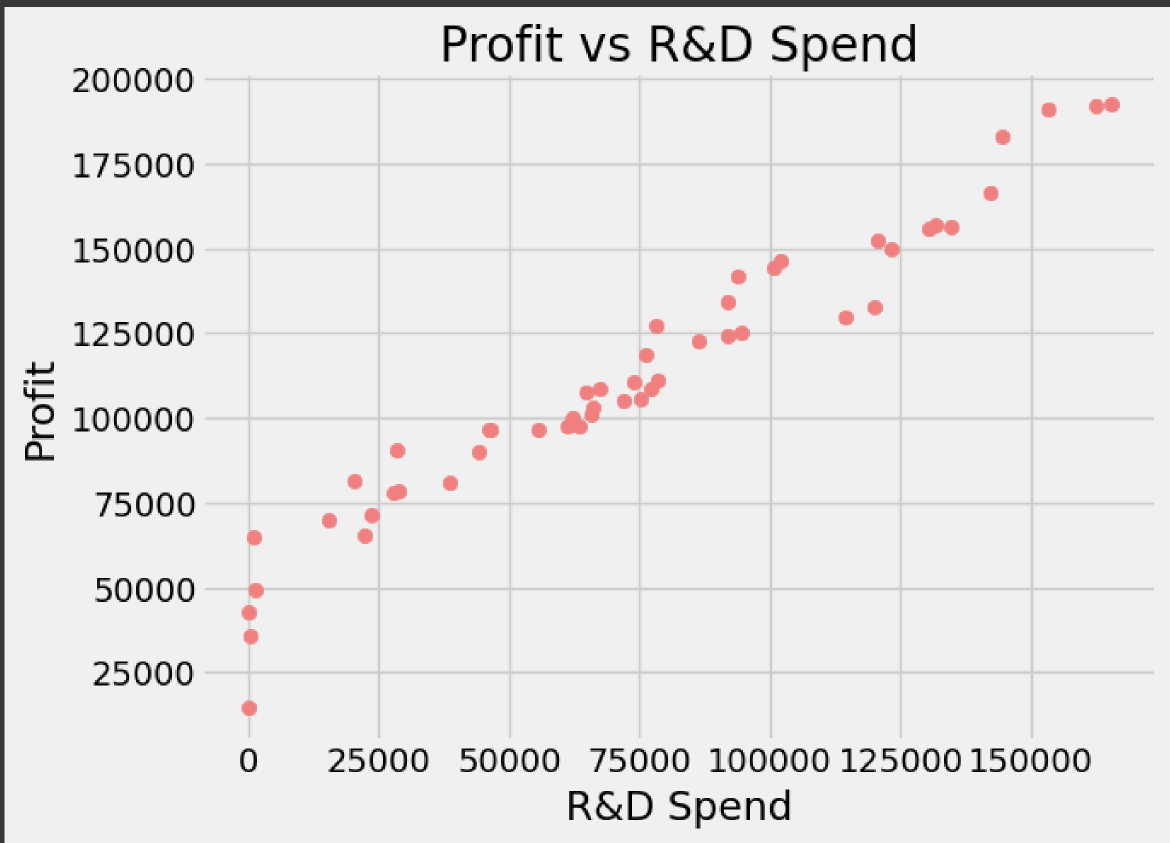
	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

```
# Data distribution
plt.title('Profit Distribution Plot')
sns.distplot(df_sal['Profit'])
plt.show()
```

```
sns.distplot(df_sal['Profit'])
```




```
# Relationship between Salary and Experience
plt.scatter(df_sal['R&D Spend'], df_sal['Profit'], color = 'lightcoral')
plt.title('Profit vs R&D Spend')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.box(False)
plt.show()
```



```
[ ] # Splitting variables
    X = df_sal.iloc[:, :1]  # independent
    y = df_sal.iloc[:, -1:]  # dependent
```

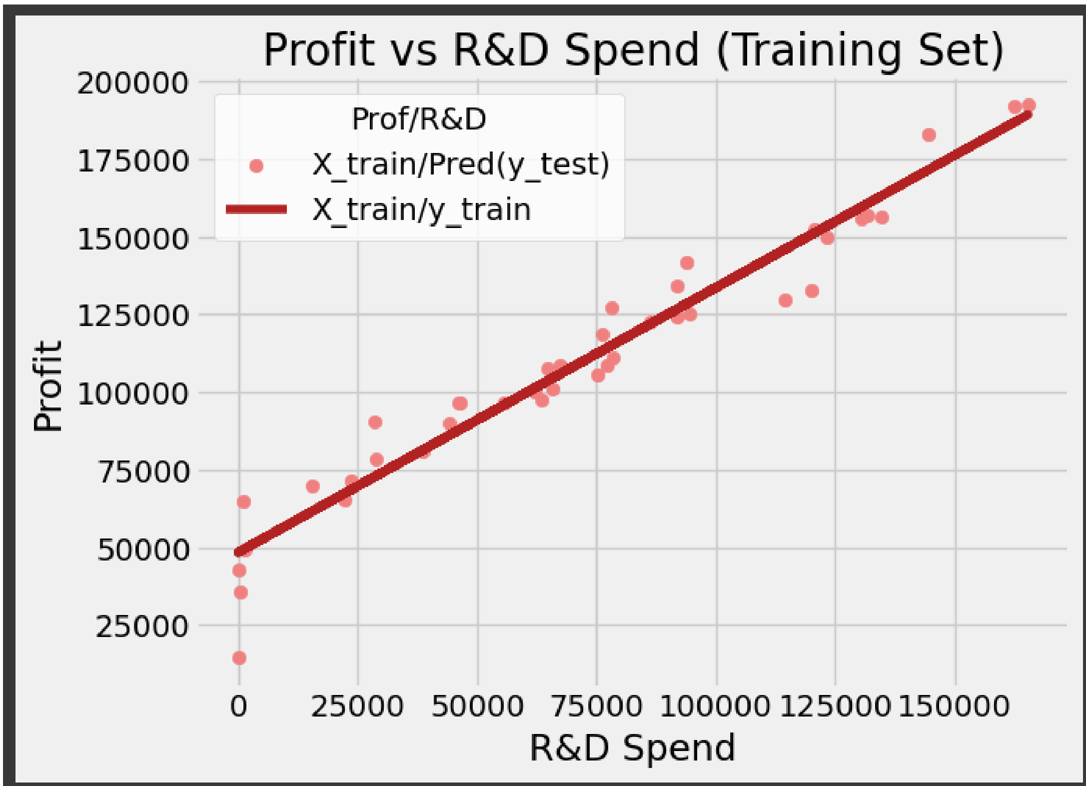
```
[ ] # Splitting dataset into test/train
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
[ ] # Regressor model
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
```

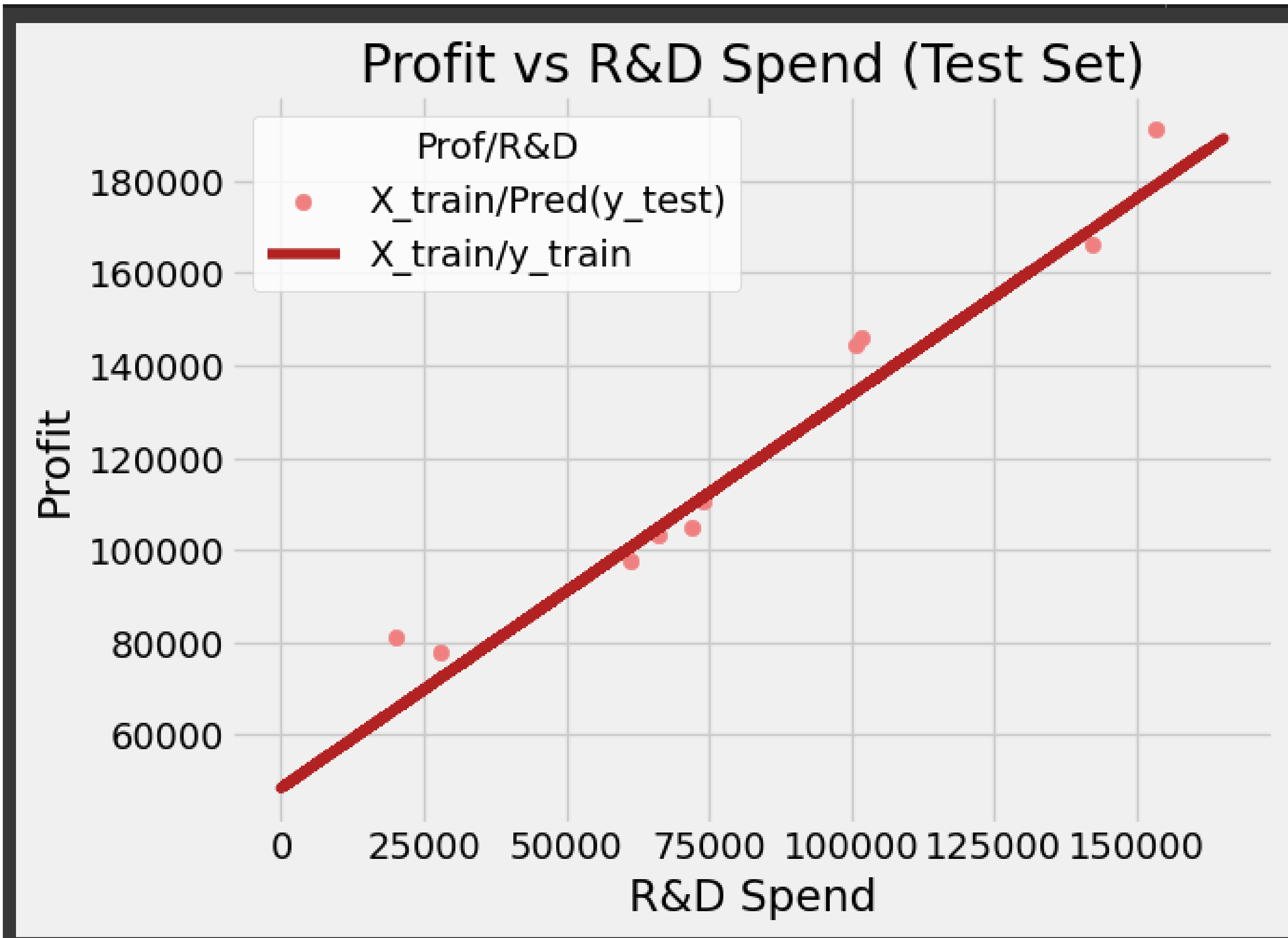
```
▾ LinearRegression
LinearRegression()
```

```
[ ] # Prediction result
    y_pred_test = regressor.predict(X_test)     # predicted value of y_test
    y_pred_train = regressor.predict(X_train)   # predicted value of y_train
```

```
# Prediction on training set
plt.scatter(X_train, y_train, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Profit vs R&D Spend (Training Set)')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Prof/R&D', loc='best', facecolor='white')
plt.box(False)
plt.show()
```




```
# Prediction on test set
plt.scatter(X_test, y_test, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Profit vs R&D Spend (Test Set)')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Prof/R&D', loc='best', facecolor='white')
plt.box(False)
plt.show()
```




```python
# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
```

```
Coefficient: [[0.8516228]]
Intercept: [48416.29766139]
```

Multiple Linear Regression

Link - 2 (https://github.com/shuv50/Data-Science/blob/main/Multiple_Linear_Regression.ipynb)

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
```

```
# Get dataset
df_start = pd.read_csv('/content/50_Startups.csv')
df_start.head()
```

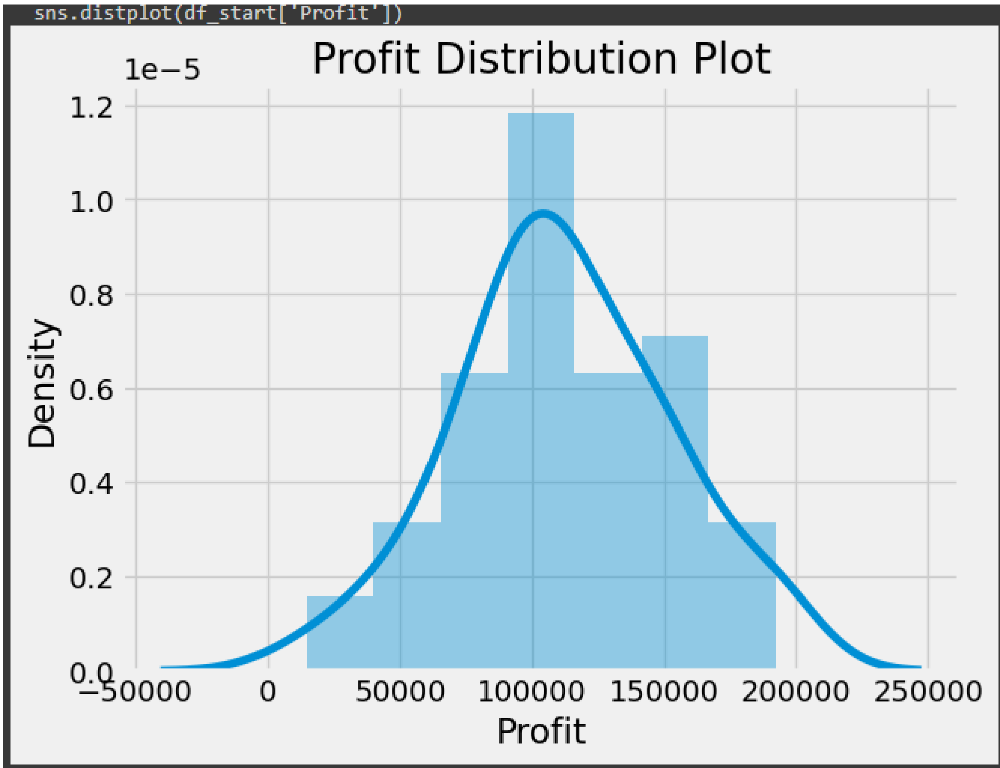
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
# Describe data
df_start.describe()
```

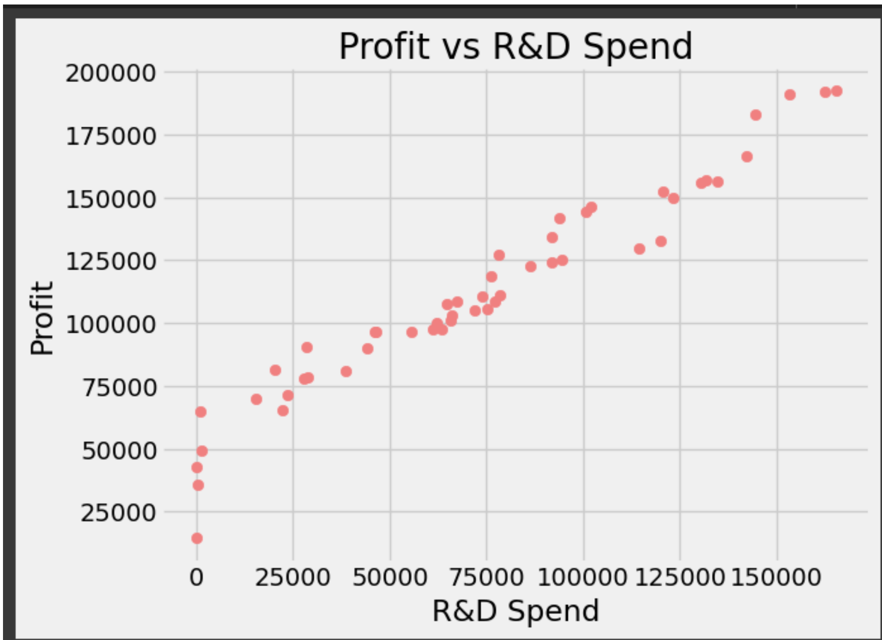
	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

```
# Data distribution
plt.title('Profit Distribution Plot')
sns.distplot(df_start['Profit'])
plt.show()
```

```
sns.distplot(df_start['Profit'])
```




```
# Relationship between Profit and R&D Spend
plt.scatter(df_start['R&D Spend'], df_start['Profit'], color = 'lightcoral')
plt.title('Profit vs R&D Spend')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.box(False)
plt.show()
```




```
# Split dataset in dependent/independent variables
X = df_start.iloc[:, :-1].values
y = df_start.iloc[:, -1].values
```

```
# One-hot encoding of categorical data
ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(), [3])], remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
```

```python
# Split dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```python
# Train multiple regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
# Predict result
y_pred = regressor.predict(X_test)
```

```python
# Compare predicted result with actual value
np.set_printoptions(precision = 2)
result = np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test), 1)), 1)
result
```

```
array([[103015.2 , 103282.38],
       [132582.28, 144259.4 ],
       [132447.74, 146121.95],
       [ 71976.1 ,  77798.83],
       [178537.48, 191050.39],
       [116161.24, 105008.31],
       [ 67851.69,  81229.06],
       [ 98791.73,  97483.56],
       [113969.44, 110352.25],
       [167921.07, 166187.94]])
```

2.5 Experiment - 5

2.5.1 Question:

Build Logistic Regression Model for a given dataset.

2.5.2 Code with Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

```
df_net = pd.read_csv('/content/Social_Network_Ads.csv')
df_net.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
df_net.drop(columns = ['User ID'], inplace=True)
df_net.head()
```

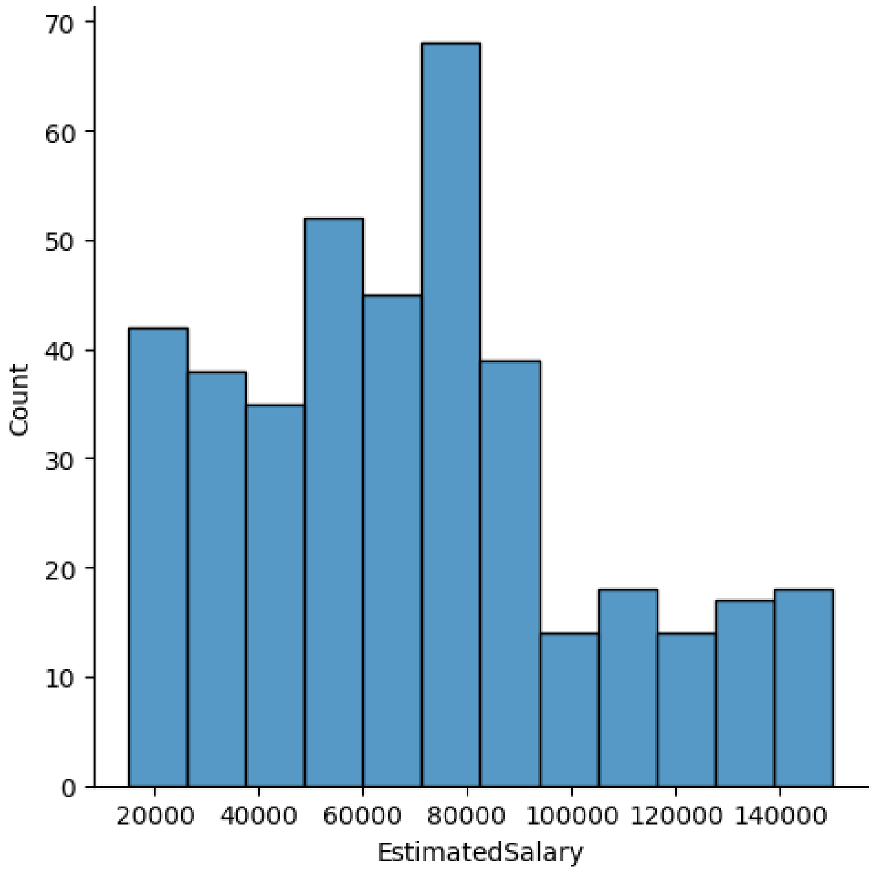
	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

```
df_net.describe()
```

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

```
sns.displot(df_net['EstimatedSalary'])
```

```
<seaborn.axisgrid.FacetGrid at 0x789c32189060>
```




```
le = LabelEncoder()
df_net['Gender']= le.fit_transform(df_net['Gender'])
```

```
# Correlation matrix
df_net.corr()
```

	Gender	Age	EstimatedSalary	Purchased
Gender	1.000000	-0.073741	-0.060435	-0.042469
Age	-0.073741	1.000000	0.155238	0.622454
EstimatedSalary	-0.060435	0.155238	1.000000	0.362083
Purchased	-0.042469	0.622454	0.362083	1.000000

```
sns.heatmap(df_net.corr())
```

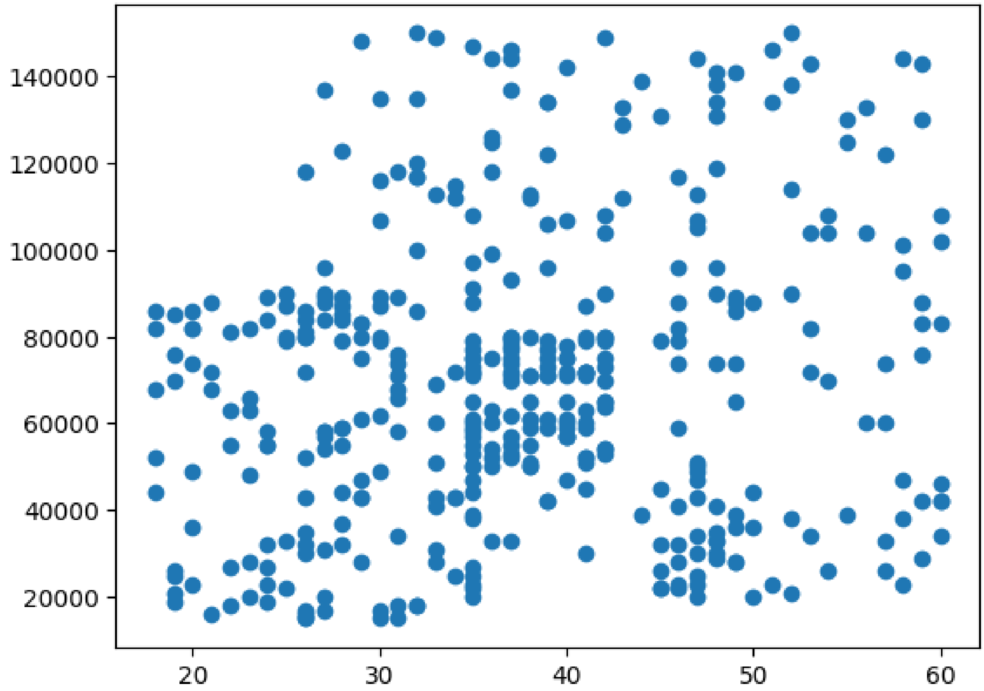
```
<Axes: >
```


```
# Drop Gender column
df_net.drop(columns=['Gender'], inplace=True)
df_net.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
# Relationship between Age and Salary
plt.scatter(df_net['Age'], df_net['EstimatedSalary'])
```

```
<matplotlib.collections.PathCollection at 0x789c2d4e0a90>
```




```
# Split data into dependent/independent variables
X = df_net.iloc[:, :-1].values
y = df_net.iloc[:, -1].values
```

```
# Split data into test/train set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = True)
```

```
# Scale dataset
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classifier
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Prediction
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test), 1)), 1))
```

```
accuracy_score(y_test, y_pred)
```

```
0.83
```

```
# Classification report
print(f'Classification Report: \n{classification_report(y_test, y_pred)}')
```

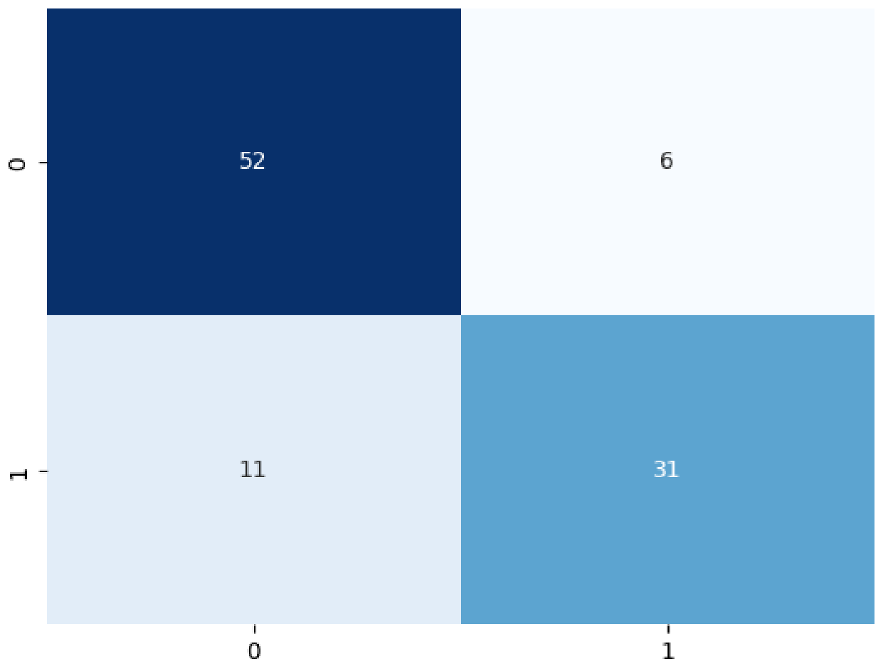
```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.90      0.86        58
           1       0.84      0.74      0.78        42

    accuracy                           0.83       100
   macro avg       0.83      0.82      0.82       100
weighted avg       0.83      0.83      0.83       100
```

```
print(f"F1 Score : {f1_score(y_test, y_pred)}")
```

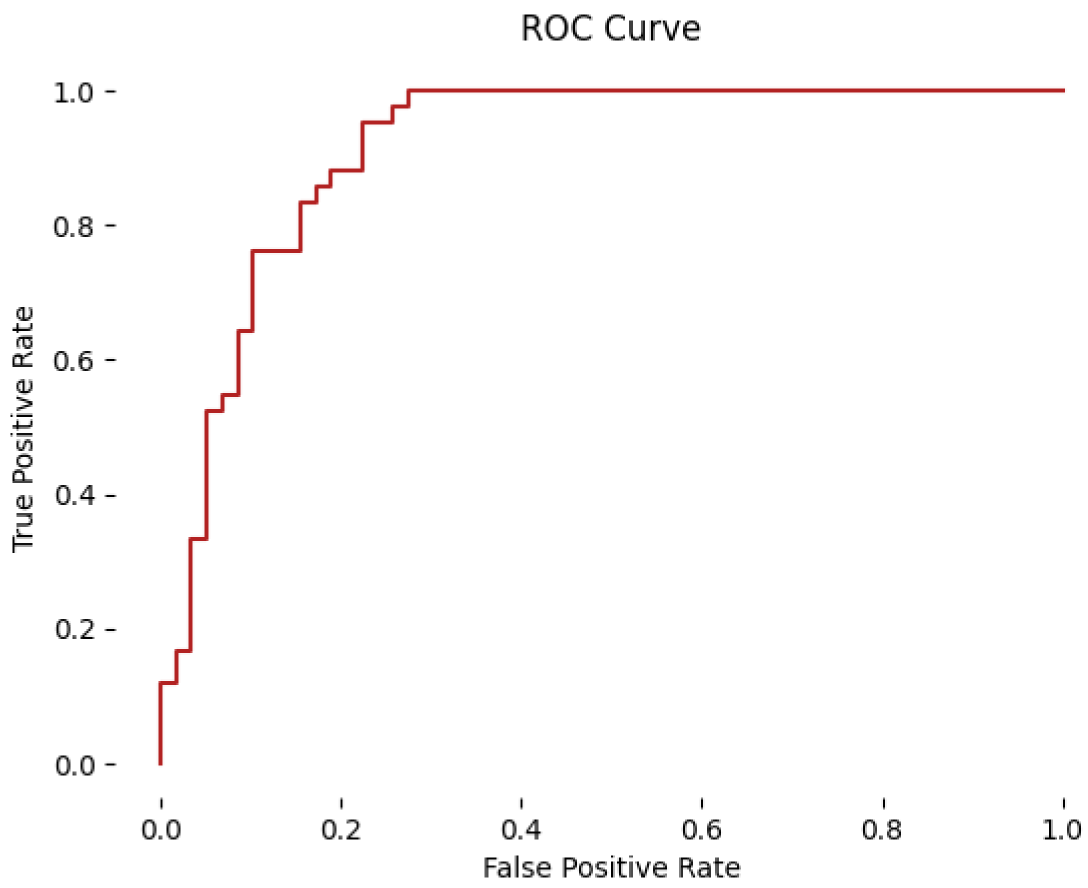
```
F1 Score : 0.7848101265822786
```

```
# Confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
```

<Axes: >




```
# Plot AUC/ROC curve
y_pred_proba = classifier.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred_proba)

plt.plot(fpr, tpr, label='Logistic Regression', color = 'firebrick')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.box(False)
plt.show()
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

2.6 Experiment - 6

2.6.1 Question:

Build KNN Classification model for a given dataset.

2.6.2 Code with Output:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

df = pd.read_csv("prostate.csv")
df.head()

scaler = StandardScaler()
scaler.fit(df.drop('Target', axis=1))
scaled_features = scaler.transform(df.drop('Target', axis=1))

df_feat = pd.DataFrame(scaled_features, columns=df.columns[:-1])

X_train, X_test, y_train, y_test = train_test_split(scaled_features, df['Target'], test_size=0.30)

# FIRST A QUICK COMPARISON TO OUR ORIGINAL K = 1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)

print('WITH K = 1')
print('Confusion Matrix')
print(confusion_matrix(y_test, pred))
print('Classification Report')
print(classification_report(y_test, pred))

# NOW WITH K = 10
knn = KNeighborsClassifier(n_neighbors=10)

knn.fit(X_train, y_train)
pred = knn.predict(X_test)

print('WITH K = 10')
print('Confusion Matrix')
print(confusion_matrix(y_test, pred))
print('Classification Report')
print(classification_report(y_test, pred))
```

```
WITH K = 1
Confusion Matrix
[[22  5]
 [ 1  2]]
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.81      0.88        27
           1       0.29      0.67      0.40         3

    accuracy                           0.80        30
   macro avg       0.62      0.74      0.64        30
weighted avg       0.89      0.80      0.83        30

WITH K = 10
Confusion Matrix
[[24  3]
 [ 1  2]]
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.89      0.92        27
           1       0.40      0.67      0.50         3

    accuracy                           0.87        30
   macro avg       0.68      0.78      0.71        30
weighted avg       0.90      0.87      0.88        30
```

2.7 Experiment - 7

2.7.1 Question:

Build Support vector machine model for a given dataset.

2.7.2 Code with Output:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

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

```
data = '/content/pulsar_stars.csv'
df = pd.read_csv(data)
```

```
df.shape
```

```
(17898, 9)
```

```
df.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

```
col_names = df.columns
col_names
```

```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

```
df.columns = df.columns.str.strip()
```

```
# view column names again
df.columns
```

```
Index(['Mean of the integrated profile',
       'Standard deviation of the integrated profile',
       'Excess kurtosis of the integrated profile',
       'Skewness of the integrated profile', 'Mean of the DM-SNR curve',
       'Standard deviation of the DM-SNR curve',
       'Excess kurtosis of the DM-SNR curve', 'Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

```
df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness', 'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skew
```

```
df.columns
```

```
Index(['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness', 'DM-SNR Mean',
       'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class'],
      dtype='object')
```

```
df['target_class'].value_counts()
```

```
target_class
0    16259
1     1639
Name: count, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   IP Mean          17898 non-null  float64
 1   IP Sd            17898 non-null  float64
 2   IP Kurtosis      17898 non-null  float64
 3   IP Skewness      17898 non-null  float64
 4   DM-SNR Mean      17898 non-null  float64
 5   DM-SNR Sd        17898 non-null  float64
 6   DM-SNR Kurtosis  17898 non-null  float64
 7   DM-SNR Skewness  17898 non-null  float64
 8   target_class     17898 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

```
# check for missing values in variables
df.isnull().sum()
```

```
IP Mean            0
IP Sd              0
IP Kurtosis        0
IP Skewness        0
DM-SNR Mean        0
DM-SNR Sd          0
DM-SNR Kurtosis    0
DM-SNR Skewness    0
target_class       0
dtype: int64
```

```
# view summary statistics in numerical variables
round(df.describe(),2)
```

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness	target_class
count	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00
mean	111.08	46.55	0.48	1.77	12.61	26.33	8.30	104.86	0.09
std	25.65	6.84	1.06	6.17	29.47	19.47	4.51	106.51	0.29
min	5.81	24.77	-1.88	-1.79	0.21	7.37	-3.14	-1.98	0.00
25%	100.93	42.38	0.03	-0.19	1.92	14.44	5.78	34.96	0.00
50%	115.08	46.95	0.22	0.20	2.80	18.46	8.43	83.06	0.00
75%	127.09	51.02	0.47	0.93	5.46	28.43	10.70	139.31	0.00
max	192.62	98.78	8.07	68.10	223.39	110.64	34.54	1191.00	1.00

```
X = df.drop(['target_class'], axis=1)
y = df['target_class']
```

```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
X_train.shape, X_test.shape
```

```
((14318, 8), (3580, 8))
```

```
cols = X_train.columns
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

```
X_train.describe()
```

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness
count	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04
mean	1.908113e-16	-6.550610e-16	1.042143e-17	3.870815e-17	-8.734147e-17	-1.617802e-16	-1.513588e-17	1.122785e-16
std	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00
min	-4.035499e+00	-3.181033e+00	-2.185946e+00	-5.744051e-01	-4.239001e-01	-9.733707e-01	-2.455649e+00	-1.003411e+00
25%	-3.896291e-01	-6.069473e-01	-4.256221e-01	-3.188054e-01	-3.664918e-01	-6.125457e-01	-5.641035e-01	-6.627590e-01
50%	1.587461e-01	5.846646e-02	-2.453172e-01	-2.578142e-01	-3.372294e-01	-4.067482e-01	3.170446e-02	-2.059136e-01
75%	6.267059e-01	6.501017e-01	-1.001238e-02	-1.419621e-01	-2.463724e-01	1.078934e-01	5.362759e-01	3.256217e-01
max	3.151882e+00	7.621116e+00	7.008906e+00	1.054430e+01	7.025568e+00	4.292181e+00	5.818557e+00	1.024613e+01

SVM with default hyperparameterst

```
# Default hyperparameter means C=1.0, kernel=rbf and gamma=auto among other parameters
# import SVC classifier
from sklearn.svm import SVC

# import metrics to compute accuracy
from sklearn.metrics import accuracy_score

# instantiate classifier with default hyperparameters
svc=SVC()

# fit classifier to training set
svc.fit(X_train,y_train)

# make predictions on test set
y_pred=svc.predict(X_test)

# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.9827
```

```python
# SVM with rbf kernel and C=100.0
# instantiate classifier with rbf kernel and C=100
svc=SVC(C=100.0)

# fit classifier to training set
svc.fit(X_train,y_train)

# make predictions on test set
y_pred=svc.predict(X_test)

# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.9832
```

```python
# SVM with rbf kernel and C=1000.0
# instantiate classifier with rbf kernel and C=1000
svc=SVC(C=1000.0)


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=1000.0 : 0.9816
```

SVM with linear kernel

```python
# Run SVM with linear kernel and C=1.0
# instantiate classifier with linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0)


# fit classifier to training set
linear_svc.fit(X_train,y_train)


# make predictions on test set
y_pred_test=linear_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score with linear kernel and C=1.0 : 0.9830
```

```python
# Run SVM with linear kernel and C=100.0
# instantiate classifier with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)


# fit classifier to training set
linear_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernel and C=100.0 : 0.9832
```

```python
# Run SVM with linear kernel and C=1000.0
# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.9832
```

Compare the train-set and test-set accuracy

```python
y_pred_train = linear_svc.predict(X_train)
y_pred_train
```

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

```python
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9783
```

Check for overfitting and underfitting

```python
# print the scores on training and test set
print('Training set score: {:.4f}'.format(linear_svc.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(linear_svc.score(X_test, y_test)))
```

```
Training set score: 0.9783
Test set score: 0.9830
```

Classification metrices

```python
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[3289   17]
 [  44  230]]
```

```
True Positives(TP) =  3289

True Negatives(TN) =  230

False Positives(FP) =  17

False Negatives(FN) =  44
```

```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.93      0.84      0.88       274

    accuracy                           0.98      3580
   macro avg       0.96      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

```python
# Classification accuracy
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

# print classification accuracy
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9830
```

```python
# Classification error
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0170
```

```python
# Precision score
precision = TP / float(TP + FP)
print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9949
```

```python
# Recall
recall = TP / float(TP + FN)
print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9868
```

2.8 Experiment - 8

2.8.1 Question:

a) Implement Random forest ensemble method on a given dataset.
b) Implement Boosting ensemble method on a given dataset.

2.8.2 Code with Output:

a) Random Forest:

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

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

```
df = pd.read_csv("/content/diabetes.csv")
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
pd.set_option('display.float_format', '{:.2f}'.format)
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24	0.35
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00	0.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

```python
categorical_val = []
continous_val = []
for column in df.columns:
#     print('==============================')
#     print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```python
# How many missing zeros are mising in each feature
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 111
============================================
Glucose ==> Missing zeros : 5
============================================
BloodPressure ==> Missing zeros : 35
============================================
SkinThickness ==> Missing zeros : 227
============================================
Insulin ==> Missing zeros : 374
============================================
BMI ==> Missing zeros : 11
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

```python
from sklearn.impute import SimpleImputer


fill_values = SimpleImputer(missing_values=0, strategy="mean", copy=False)
df[feature_columns] = fill_values.fit_transform(df[feature_columns])

for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 0
============================================
Glucose ==> Missing zeros : 0
============================================
BloodPressure ==> Missing zeros : 0
============================================
SkinThickness ==> Missing zeros : 0
============================================
Insulin ==> Missing zeros : 0
============================================
BMI ==> Missing zeros : 0
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

```python
from sklearn.model_selection import train_test_split


X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report


def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```python
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42, n_estimators=1000)
rf_clf.fit(X_train, y_train)
evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[349   0]
 [  0 188]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
                0      1  accuracy  macro avg  weighted avg
precision    1.00   1.00      1.00       1.00          1.00
recall       1.00   1.00      1.00       1.00          1.00
f1-score     1.00   1.00      1.00       1.00          1.00
support    349.00 188.00      1.00     537.00        537.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[123  28]
 [ 29  51]]
ACCURACY SCORE:
0.7532
CLASSIFICATION REPORT:
                0     1  accuracy  macro avg  weighted avg
precision    0.81  0.65      0.75       0.73          0.75
recall       0.81  0.64      0.75       0.73          0.75
f1-score     0.81  0.64      0.75       0.73          0.75
support    151.00 80.00      0.75     231.00        231.00
```

b) Boosting Ensemble:

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

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

```
df = pd.read_csv("/content/diabetes.csv")
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
pd.set_option('display.float_format', '{:.2f}'.format)
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24	0.35
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00	0.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

```python
categorical_val = []
continous_val = []
for column in df.columns:
#     print('==============================')
#     print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```python
# How many missing zeros are mising in each feature
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 111
============================================
Glucose ==> Missing zeros : 5
============================================
BloodPressure ==> Missing zeros : 35
============================================
SkinThickness ==> Missing zeros : 227
============================================
Insulin ==> Missing zeros : 374
============================================
BMI ==> Missing zeros : 11
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

```python
from sklearn.impute import SimpleImputer


fill_values = SimpleImputer(missing_values=0, strategy="mean", copy=False)
df[feature_columns] = fill_values.fit_transform(df[feature_columns])

for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 0
============================================
Glucose ==> Missing zeros : 0
============================================
BloodPressure ==> Missing zeros : 0
============================================
SkinThickness ==> Missing zeros : 0
============================================
Insulin ==> Missing zeros : 0
============================================
BMI ==> Missing zeros : 0
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

```python
from sklearn.model_selection import train_test_split


X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report


def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```python
from sklearn.ensemble import AdaBoostClassifier

ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(X_train, y_train)
evaluate(ada_boost_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[310  39]
 [ 51 137]]
ACCURACY SCORE:
0.8324
CLASSIFICATION REPORT:
               0       1  accuracy  macro avg  weighted avg
precision   0.86    0.78      0.83       0.82          0.83
recall      0.89    0.73      0.83       0.81          0.83
f1-score    0.87    0.75      0.83       0.81          0.83
support   349.00  188.00      0.83     537.00        537.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[123  28]
 [ 27  53]]
ACCURACY SCORE:
0.7619
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.82   0.65      0.76       0.74          0.76
recall      0.81   0.66      0.76       0.74          0.76
f1-score    0.82   0.66      0.76       0.74          0.76
support   151.00  80.00      0.76     231.00        231.00
```

2.9 Experiment - 9

2.9.1 Question:

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

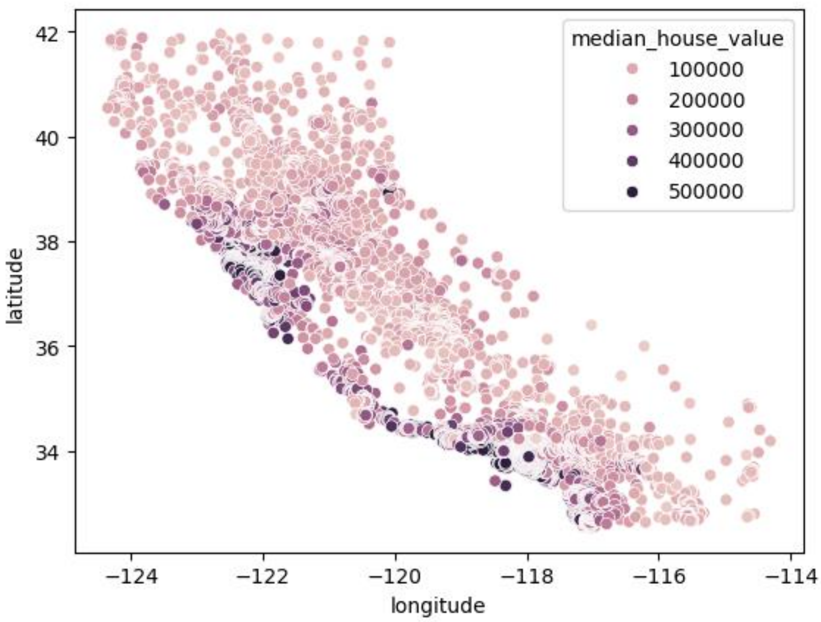
2.9.2 Code with Output:

```
import pandas as pd

home_data = pd.read_csv('/content/housing.csv', usecols = ['longitude', 'latitude', 'median_house_value'])
home_data.head()
```

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

```
import seaborn as sns

sns.scatterplot(data = home_data, x = 'longitude', y = 'latitude', hue = 'median_house_value')
```

<Axes: xlabel='longitude', ylabel='latitude'>




```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(home_data[['latitude', 'longitude']], home_data[['median_house_value']].
```

```
from sklearn import preprocessing

X_train_norm = preprocessing.normalize(X_train)
X_test_norm = preprocessing.normalize(X_test)
```

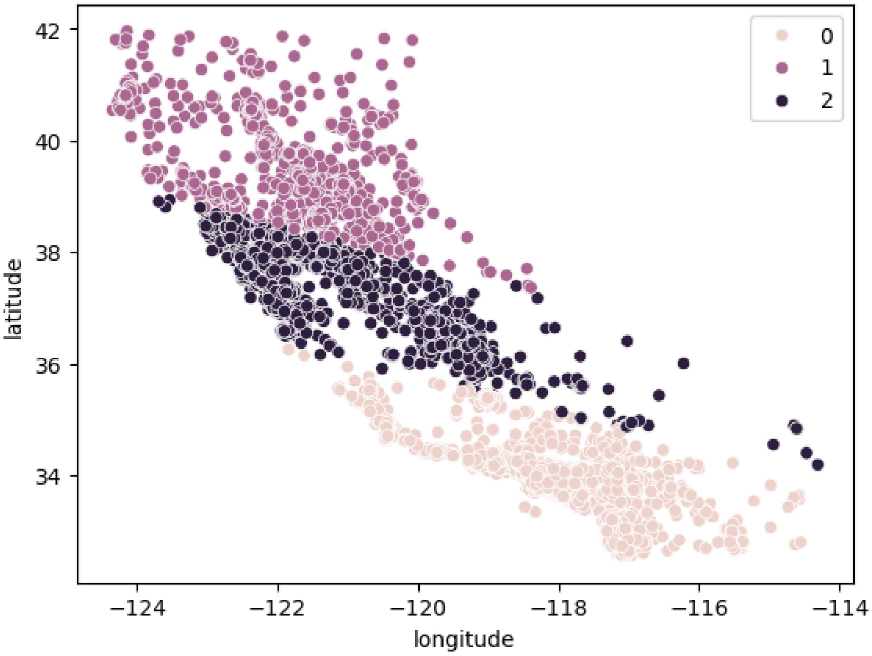
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
kmeans.fit(X_train_norm)
```

```
KMeans(n_clusters=3, n_init='auto', random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```python
sns.scatterplot(data = X_train, x = 'longitude', y = 'latitude', hue = kmeans.labels_)
```

```
<Axes: xlabel='longitude', ylabel='latitude'>
```



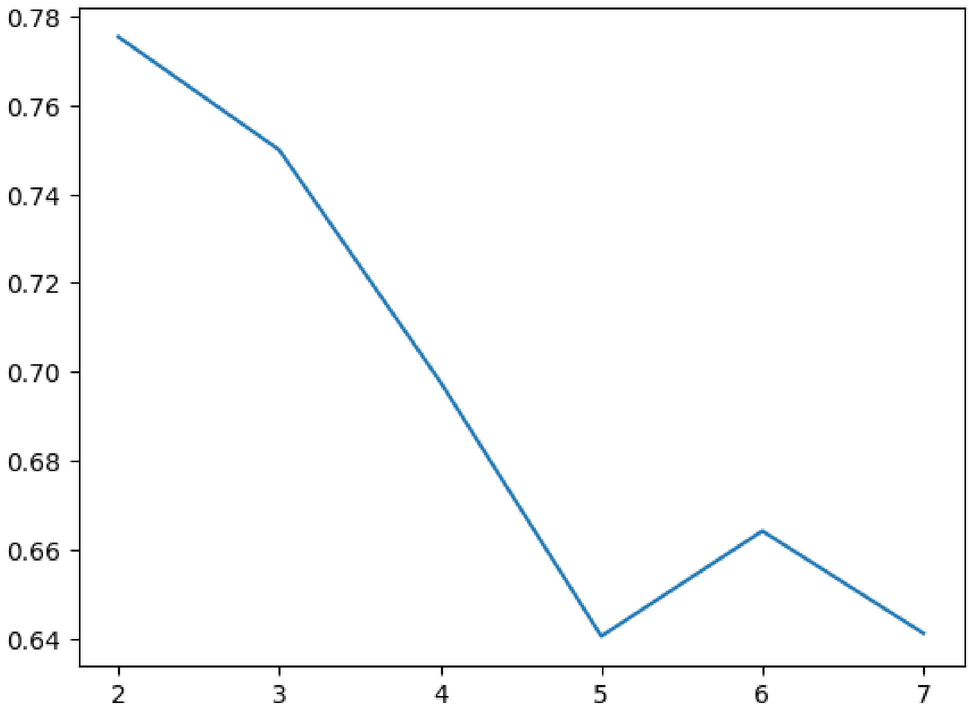

```python
from sklearn.metrics import silhouette_score
silhouette_score(X_train_norm, kmeans.labels_, metric='euclidean')
```

```
0.7499371920703546
```

```python
K = range(2, 8)
fits = []
score = []


for k in K:
    # train the model for current value of k on training data
    model = KMeans(n_clusters = k, random_state = 0, n_init='auto').fit(X_train_norm)

    # append the model to fits
    fits.append(model)

    # Append the silhouette score to scores
    score.append(silhouette_score(X_train_norm, model.labels_, metric='euclidean'))
```

```python
sns.lineplot(x = K, y = score)
```

```
&lt;Axes: &gt;
```



2.10 Experiment - 10

2.10.1 Question:

Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

2.10.2 Code with Output:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pandas as pd
import seaborn as sns
from sklearn import datasets
```

```
iris = datasets.load_iris()
df = pd.DataFrame(iris['data'], columns= iris['feature_names'])
df.head()
```

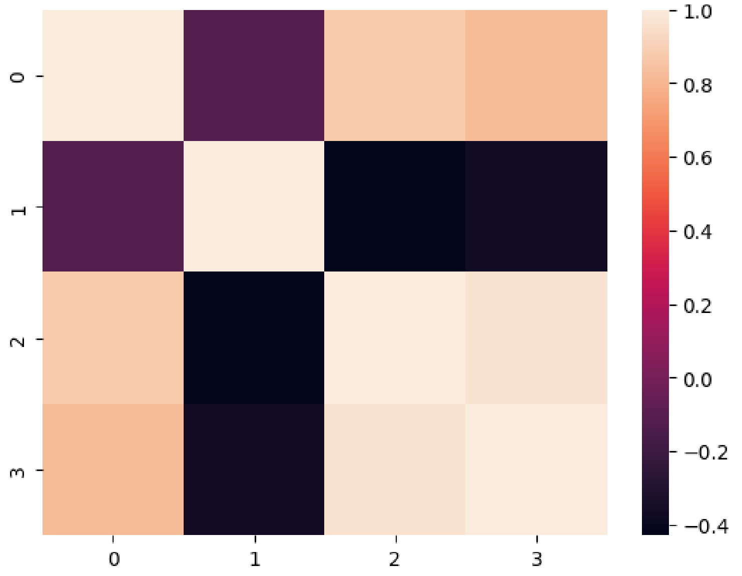
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
scaler = StandardScaler()
scaled_data = pd.DataFrame(scaler.fit_transform(df))
scaled_data.head()
```

	0	1	2	3
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

```
sns.heatmap(scaled_data.corr())
```

```
<Axes: >
```

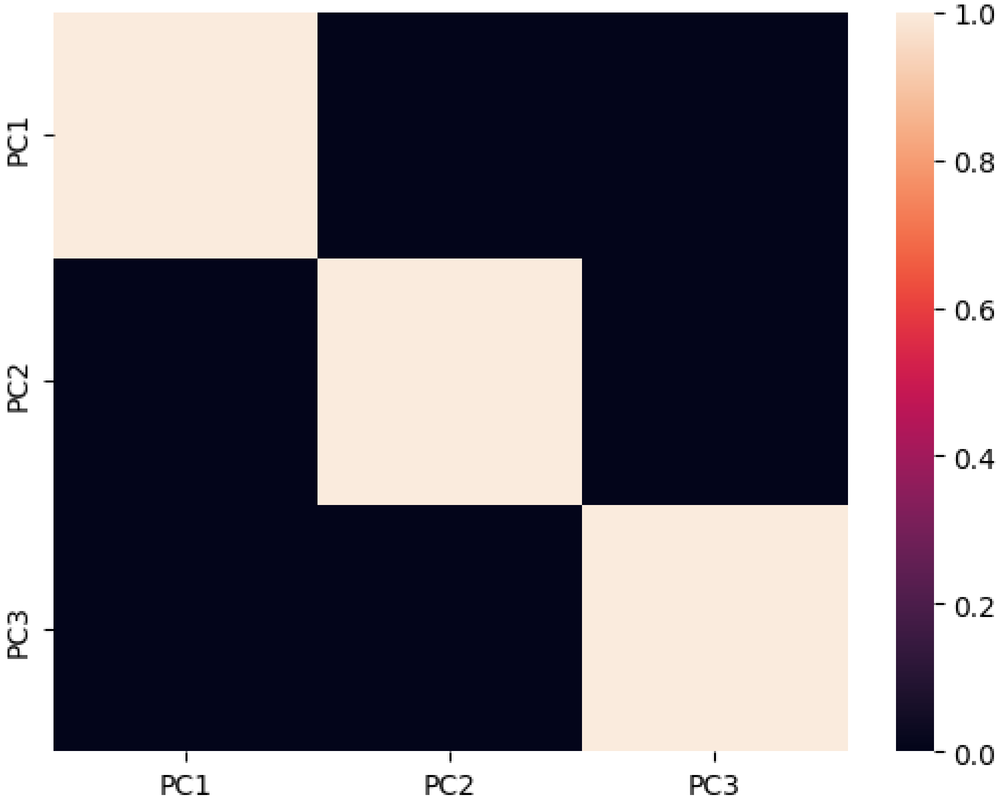


```
data_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2', 'PC3'])
data_pca.head()
```

	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

```
sns.heatmap(data_pca.corr())
```

<Axes: >



2.11 Experiment - 11

2.11.1 Question:

Build Artificial Neural Network model with back propagation on a given dataset.

2.11.2 Code with Output:

```
import numpy as np

# Define input (X) and output (Y) arrays
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep, study]
Y = np.array(([92], [86], [89]), dtype=float) # one output (Expected & in Exams)

# Normalize the data
X = X / np.amax(X, axis=0)  # maximum of X array longitudinally
Y = Y / 100  # max test score is 100

# Set parameters
epoch = 5000
lr = 0.1
inputlayer_neurons = X.shape[1]  # number of features in data set
hiddenlayer_neurons = 3  # number of hidden layer neurons
output_neurons = 1  # number of neurons at output layer

# Weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))  # weights for the input layer to hidden layer
bh = np.random.uniform(size=(1, hiddenlayer_neurons))  # bias for the hidden layer
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))  # weights for the hidden layer to output layer
bout = np.random.uniform(size=(1, output_neurons))  # bias for the output layer
```

```
# Activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of sigmoid function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Training algorithm
for i in range(epoch):
    # Forward Propagation
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = Y - output  # error at output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)  # error at hidden layer
    hiddengrad = derivatives_sigmoid(hlayer_act)  # derivative of sigmoid function
    d_hiddenlayer = EH * hiddengrad
```

```python
    # Updating weights and biases
    wout += hlayer_act.T.dot(d_output) * lr
    bout += np.sum(d_output, axis=0, keepdims=True) * lr
    wh += X.T.dot(d_hiddenlayer) * lr
    bh += np.sum(d_hiddenlayer, axis=0, keepdims=True) * lr

# Output after training
print("Input: \n" + str(X))
print("Actual Output: \n" + str(Y))
print("Predicted Output: \n", output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89526104]
 [0.87867405]
 [0.89490822]]
```