Sunkara Rohit
160119733043

ID3 Algorithm Aug 11, 2022

ID3 uses a top-down greedy approach to build a decision tree. Top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

Most generally ID3 is only used for classification problems with nominal features only.

ID3 algorithm selects the best feature at each step while building a Decision tree.

Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gai**n is selected as the **best** one.

Entropy is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset.

In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

Entropy(S) = - $\sum p_i * \log_2(p_i)$ **; i = 1 to n**

IG(S, A) = Entropy(S) - $\sum((|S_v| / |S|) * \text{Entropy}(S_v))$

ID3 Steps

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset S into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

```
'''
1. Create a class Circle and intialize it with radius value.
Make two methods getArea and getCircumference inside thisclass.
And calculate area and Circumference then print
'''
import math

class Circle:
  def_init_(self, radius):
    self.radius = radius

  def getArea(self):
    return round(math.pi * (self.radius ** 2), 2)

  def getCircumference(self):
    return round(math.pi * (2 * self.radius), 2)

c1 = Circle(2)
print("area =  {}".format(c1.getArea()))
print("circumference = {}".format(c1.getCircumference()))
```

```
area = 12.57
circumference = 12.57
```

```
'''
2. Create a Temperature class. Create two methods:
a. convertFahrenheit - It will take celsius and will print it into Fahrenheit.
b. convertCelsius - It will take Fahrenheit and will convert it into Celsius.
'''

class Temperature:
  def convertFahrenheit(self, tempC):
    return round(32 + ((9 / 5) * tempC), 2)

  def convertCelsius(self, tempF):
    return round((5 / 9) * (tempF - 32), 2)

t1 = Temperature()
print("{} celsius = {} farhenheit".format(0, t1.convertFahrenheit(0)))
print("{} celsius = {} farhenheit".format(100, t1.convertFahrenheit(100)))
print("{} farhenheit = {} celsius".format(32, t1.convertCelsius(32)))
print("{} farhenheit = {} celsius".format(212, t1.convertCelsius(212)))
```

```
0 celsius = 32.0 farhenheit
100 celsius = 212.0 farhenheit
32 farhenheit = 0.0 celsius
212 farhenheit = 100.0 celsius
```

```
'''
3. Create a student class and initialize it with name and roll number. Create me
a. Display - It should display all information of the student.
```

```
b. setAge - It should assign age to student
c. setMarks - It should assign marks to the student.
'''

class Student:
  def __init__(self, name, rollnum, age=None, marks=None):
    self.name = name
    self.rollnum = rollnum
    self.age = age
    self.marks = marks

  def Display(self):
    print("Student info:")
    print("name: {}\nroll number: {}\nage: {}\nmarks: {}".format(self.name, self
    print()

  def setAge(self, age):
    self.age = age

  def setMarks(self, marks):
    self.marks = marks

s1 = Student('akshi',1)
s1.Display()
s1.setAge(19)
s1.setMarks(83)
s1.Display()
```

```
Student info:
name: akshi
roll number: 1
age: None
marks: None

Student info:
name: akshi
roll number: 1
age: 19
marks: 83
```

```
'''
4. Create a Time class and initialize it with hours and minutes.
a. Make a method addTime which should take two time object and add them. E.g.- (
b. Make a method displayTime which should print the time.
c. Make a method DisplayMinute which should display the total minutes in the Tim
'''

class Time:
  def __init__(self, hr, min):
    self.hr = hr
    self.min = min

  @staticmethod
```

```
  def addTime  time1,  time2):
    minsum = time1.min + time2.min
    min = minsum % 60
    hr = time1.hr + time2.hr + (minsum // 60)
    print("{} hr and {} min".format(hr, min))

  def displayTime(self):
    print("{} hr and {} min".format(self.hr,  self.min))

  def displayMinute(self):
    min = (self.hr * 60) + self.min
    print("{} minutes".format(min))

t1 = Time(1, 20)
t1.displayTime()

t2 = Time(2, 50)
t2.displayTime()

Time.addTime(t1, t2)
t1.displayMinute()
```

```
1 hr and 20 min
2 hr and 50 min
4 hr and 10 min
80 minutes
```

numpy.*unique*(ar, return_index=False, return_inverse=False, return_counts=False, axis=No ne, *, equal_nan=True)

Parameters

ar: array_like Input array. Unless axis is speci�ed, this will be �attened if it is not already 1-D.

return_index: bool, optional If True, also return the indices of ar (along the speci�ed axis, if provided, or in the �attened array) that result in the unique array.

return_inverse: bool, optional If True, also return the indices of the unique array (for the speci�ed axis, if provided) that can be used to reconstruct ar.

return_counts: bool, optional If True, also return the number of times each unique item appears in ar.

axis: int or None, optional The axis to operate on. If None, ar will be �attened. If an integer, the subarrays indexed by the given axis will be �attened and treated as the elements of a 1-D array with the dimension of the given axis, see the notes for more details. Object arrays or structured arrays that contain objects are not supported if the axis kwarg is used. The default is None.

equal_nan:bool, optional If True, collapses multiple NaN values in the return array into one.

Returns

Unique: ndarray The sorted unique values.

unique_indices: ndarray, optional The indices of the �rst occurrences of the unique values in the original array. Only provided if return_index is True.

unique_inverse: ndarray, optional The indices to reconstruct the original array from the unique array. Only provided if return_inverse is True.

unique_counts: ndarray, optional The number of times each of the unique values comes up in the original array. Only provided if return_counts is True.

```
# unique method
import numpy as np

arr = [1, 1, 5, 10, 2, 7, 3, 9, 2, 2]
print(arr)
uq, inv, ind, counts = np.unique(arr,return_inverse=True, return_index=True, ret
print("unique array = {}".format(uq))
print("return_inverse = {}".format(inv))
print("return_index = {}".format(ind))
print("return_counts = {}".format(counts))
```

```
[1, 1, 5, 10, 2, 7, 3, 9, 2, 2]
 unique array =  [ 1  2  3  5  7  9 10]
return_inverse  = [0 4 6 2 5 7 3]
return_index = [0 0 3 6 1 4 2 5 1 1]
return_counts  = [2 3 1 1 1 1 1]
```

```
# unique  q1
import numpy as np

arr = [1, 1, 5, 10, 2, 7, 3, 9, 2, 2]
print(arr)
uq, ind = np.unique(arr, return_index=True)
print("unique values: {}".format(uq))
print("indices of unique values: {}".format(ind))
print(list(zip(uq,ind)))
```

```
[1, 1, 5, 10, 2, 7, 3, 9, 2, 2]
unique  values: [ 1  2  3  5  7  9  10]
indices of unique  values: [0  4 6 2 5 7 3]
[(1, 0), (2, 4), (3, 6), (5, 2), (7, 5), (9, 7), (10, 3)]
```

```
# unique  q2
import numpy as np

arr = [[1, 2, 1],
       [1, 2,  1],
        [2, 5, 2],
        [4, 5, 4],
         2   6
```

```
      ,  ,
uq_rows = np.unique(arr, axis=0)
print("unique rows:\n {}".format(uq_rows))
uq_cols = np.unique(arr, axis=1)
print("unique columns:\n {}".format(uq_cols))
uq, ind = np.unique(arr, return_index=True)
print(list(zip(uq,ind)))
```

```
unique rows:
 [[1 2 1]
 [2 5 2]
 [2 6 2]
 [4 5 4]]
unique columns:
 [[1 2]
 [1 2]
 [2 5]
 [4 5]
 [2 6]]
[(1, 0), (2, 1), (4, 9), (5, 7), (6, 13)]
```

ID3 algorithm implementation.

AIM: Implement ID3 algorithm on 'weather.csv' dataset.

DESCRIPTION:

ID3 uses a top-down greedy approach to build a decision tree. Top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

ID3 steps:

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset S into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

```
import  math
import pandas as pd
from operator import itemgetter


class DecisionTree:
  #constructor
    def __init__(self, df, target, positive, parent_val, parent):
        self.data = df      #df meant dataframe
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = ''
#design entropy function
    def _get_entropy(self, data):
        pos = sum(data[self.target]==self.positive)
        neg = data.shape[0] - pos
        pos_ratio = pos/(pos+neg)
        neg_ratio = 1 - pos_ratio
        #calculate entropy
        entropy_p = -pos_ratio*math.log2(pos_ratio) if pos_ratio != 0 else 0
        entropy_n = - neg_ratio*math.log2(neg_ratio) if neg_ratio !=0 else 0
        return entropy_p + entropy_n

    def _get_gain(self  feat):         # feat represent features here
```

```
        avg_info=0
        # get all unique features
        for val in self.data[feat].unique():
          # now calc info gain using entropy value
            avg_info+=self._get_entropy(self.data[self.data[feat] == val])*sum(s
        # return final entropy value
        return self._get_entropy(df) - avg_info

    def _get_splitter(self):
        self.splitter = max(self.gains, key = itemgetter(1))[0]  #for info gain

    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:       #if entropy not zero, then calc gain for each
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features
            self._get_splitter() # for each column
            residual_columns = [k for k in self.data.columns if k != self.splitt
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter]==val][residual_colum
                #temp node creation for storing tree
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val,
                tmp_node.update_nodes()
                self.childs.append(tmp_node) # initially child list empty, now a


def print_tree(n):
    for child in n.childs:
        if child:
            print(child._dict_.get('parent', ''))
            print(child._dict_.get('parent_val', ''), '\n')
            print_tree(child)


df = pd.read_csv('weather.nominal.csv')
df
```

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	norma	False	

10sunny mild normal True yes **11**overcast

mild high True yes **12**overcast

hot normal False yes **13**rainy mild

high True no

```
#def_init_(self, df, target, positive, parent_val, parent):
dt = DecisionTree(df, 'play', 'yes', '', '')
dt.update_nodes()
print_tree(dt)
```

outlook
sunny

humidity
high

humidity
normal

outlook
overcast

outlook
rainy

windy
False

windy
True

0s completed at 12:08 PM

1. Use employees.csv and **perform** operations to deal **with missing** values.

AIM: Given a set of questions:

1. Extract rows with missing values for a specific column, use isnull() for that column.
2. Extract columns that contain at least one missing value.
3. Extract rows that contain at least one missing value, use any() method.
4. Find a list of columns with missing data
5. Find the number of missing values/data per column
6. Find the column with the maximum number of missing data
7. Find the number total of missing values in the DataFrame
8. Find rows with missing data
9. print a list of rows with missing data
10. print the number of missing data per row
11. Find the row with the largest number of missing data
12. Remove rows with missing data

DESCRIPTION:

isnull(): The isnull() method returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False.

nonull(): Detects non-missing values for an array-like object.

any(): The any() method returns one value for each column, True if ANY value in that column is True, otherwise False.

By specifying the column axis (axis='columns'), the all() method returns True if ANY value in that axis is True.

The above functions in python provided by pandas are helpful in order to deal with missing values.

```
#1.Extract rows with missing values foa a specific column, use isnull() for that column.
import pandas as pd
data=pd.read_csv(”employees.csv“)
b=pd.isnull(data[”DEPARTMENT_ID“])
data[b]
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB
15	106	Valli	Pataballa	VPATABAL	590.423.4560	05-Feb-06	IT PF
16	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-Feb-07	IT PF

```
#2.Extract columns that contain at least one missing value.
#To remove rows and columns where all values are missing values.
ans=data.dropna(how='all').dropna(how='all',axis=1)
x=ans.loc[:,ans.isnull().any()]
x
```

18	9000.0	108.0	100.0
19	8200.0	108.0	100.0
20	7700.0	108.0	100.0
21	7800.0	108.0	100.0
22	6900.0	108.0	100.0
23	11000.0	100.0	30.0
24	3100.0	114.0	30.0
25	2900.0	114.0	30.0
26	2800.0	114.0	30.0
27	2600.0	114.0	30.0
28	2500.0	114.0	30.0
29	8000.0	100.0	50.0
30	8200.0	100.0	50.0
31	7900.0	100.0	50.0
32	6500.0	100.0	50.0
33	5800.0	100.0	50.0
34	3200.0	120.0	50.0
35	2700.0	120.0	50.0
36	2400.0	120.0	50.0
37	2200.0	120.0	50.0
38	3300.0	121.0	50.0
39	2800.0	121.0	50.0
40	2500.0	121.0	50.0
41	2100.0	121.0	50.0
42	3300 0	122 0	50 0

#3.Extract rows that contain at least one missing value, use any() method.

ans3=a49[an89g9A11().an$\6 s=1)] 50.0

an S 3

44	2400.0	122.0	50.0
45	2200.0	122.0	50.0
46	3600.0	123.0	50.0
47	3200.0	123.0	50.0
48	2700.0	123.0	50.0
49	NaN	123.0	50.0

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL PHONE_NUMBER HIRE_DATE 31

#4. F1n@ a 11st of fggumns w V s s1ng dat g SKING 515.123.4567 17-Jun-03 AD_I

```
ans4=d a.columns[ a.isnull al ny()].tolist
ans4
```

Pataballa VPATABAL 590.423.4560 05-Feb-06 IT_F

16 107 Diana Lorentz DLORENTZ 590.423.5567 07-Feb-07 T_F

['SALARY' 'MANAGER_ID', 'DEPARTMENT_ID']

49 140 Joshua Pate, JPATEL 650.121.1834 06-Apr-06 ST C

```
#5.Find the number of missing values/data per column
ans5=data.isnull().sum()
ans5
```

```
EMPLOYEE_ID      0
FIRST_NAME       0
LAST_NAME        0
EMAIL            0
PHONE_NUMBER     0
HIRE_DATE        0
JOB_ID           0
SALARY           1
COMMISSION_PCT   0
MANAGER_ID       1
DEPARTMENT_ID    2
dtype: int64
```

```
#6.Find the column with the maximum number of missing data
print(data.count().idxmin())
```

DEPARTMENT_ID

```
#7.Find the number total of missing values in the DataFrame
print(data.isnull().values.sum())
```

4

```
#8.Find rows with missing data
data.loc[data.isnull().any(axis=1)]
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUNBER	HIRE_DATE	J(
9	100	Steven	King	SKING	515.123.4567	17-Jun-03	AD_I
15	106	Valli	Pataballa	VPATABAL	590.423.4560	05-Feb-06	IT_F
16	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-Feb-07	IT_F
49	140	Joshua	Patel	JPATEL	650.121.1834	06-Apr-06	ST C

```
#9.print a list of rows with missing data
data.loc[data.isnull().any(axis=1)]
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	J(
9	100	Steven	King	SKING	515.123.4567	17-Jun-03	AD I
15	106	Valli	Pataballa	VPATABAL	590.423.4560	05-Feb-06	IT F
16	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-Feb-07	IT_F
49	140	Joshua	Patel	JPATEL	650.121.1834	06-Apr-06	ST C

```
#10.print the number of missing data per row
for i in range(len(data.index)) :
    print("Nan in row ", i , "   " ,  data.iloc[i].isnull().sum())
```

```
Nan in row  0  :  0
Nan in row  1  :  0
Nan in row  2  :  0
Nan in row  3  :  0
Nan in row  4  :  0
Nan in row  5  :  0
Nan in row  6  :  0
Nan in row  7  :  0
Nan in row  8  :  0
Nan in row  9  :  1
Nan in row  10  :  0
Nan in row  11  :  0
Nan in row  12  :  0
Nan in row  13  :  0
Nan in row  14  :  0
Nan in row  15  :  1
Nan in row  16  :  1
Nan in row  17  :  0
Nan in row  18  :  0
Nan in row  19  :  0
Nan in row  20  :  0
Nan in row  21  :  0
Nan in row  22  :  0
Nan in row  23  :  0
Nan in row  24  :  0
Nan in row  25  :  0
Nan in row  26  :  0
Nan in row  27  :  0
Nan in row  28  :  0
Nan in row  29  :  0
Nan in row  30  :  0
Nan in row  31  :  0
Nan in row  32  :  0
Nan in row  33  :  0
Nan in row  34  :  0
Nan in row  35  :  0
Nan in row  36  :  0
Nan in row  37  :  0
```

```
Nan in row  38  :  0
Nan in row  39  :  0
Nan in row  40  :  0
Nan in row  41  :  0
Nan in row  42  :  0
Nan in row  43  :  0
Nan in row  44  :  0
Nan in row  45  :  0
Nan in row  46  :  0
Nan in row  47  :  0
Nan in row  48  :  0
Nan in row  49  :  1
```

```
#11.Find the row with the largest number of missing data
answer=[ ]
maxi=0
for i in range(len(data.index)) :
    x=data.iloc[i].isnull().sum()
    if x>maxi:
      index=i
print(index)
```

```
49
```

```
#12.Remove rows with missing data
data.dropna()
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	
0	198	Donald	OConnell	DOCONNEL	650.507.9833	21-Jun-07	SH_
1	199	Douglas	Grant	DGRANT	650.507.9844	13-Jan-08	SH_
2	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-Sep-03	AT
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-Feb-04	M
4	202	Pat	Fay	PFAY	603.123.6666	17-Aug-05	M
5	203	Susan	Mavris	SMAVRIS	515.123.7777	07-Jun-02	H
6	204	Hermann	Baer	HBAER	515.123.8888	07-Jun-02	F
7	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-Jun-02	A
8	206	William	Gietz	WGIETZ	515.123.8181	07-Jun-02	AC_AC
10	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-Sep-05	
11	102	Lex	De Haan	LDEHAAN	515.123.4569	13-Jan-01	
12	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-Jan-06	IT
13	104	Bruce	Ernst	BERNST	590.423.4568	21-May-07	IT
14	105	David	Austin	DAUSTIN	590.423.4569	25-Jun-05	IT
17	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-Aug-02	
18	109	Daniel	Faviet	DFAVIET	515.124.4169	16-Aug-02	FI_AC
19	110	John	Chen	JCHEN	515.124.4269	28-Sep-05	FI AC
20	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-Sep-05	FI_AC
21	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-Mar-06	FI_AC
22	113	Luis	Popp	LPOPP	515.124.4567	07-Dec-07	FI_AC
23	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-Dec-02	P
24	115	Alexander	Khoo	AKHOO	515.127.4562	18-May-03	PU_
25	116	Shelli	Baida	SBAIDA	515.127.4563	24-Dec-05	PU_
26	117	Sigal	Tobias	STOBIAS	515.127.4564	24-Jul-05	PU_
27	118	Guy	Himuro	GHIMURO	515.127.4565	15-Nov-06	PU_
28	119	Karen	Colmenares	KCOLMENA	515.127.4566	10-Aug-07	PU_
29	120	Matthew	Weiss	MWEISS	650.123.1234	18-Jul-04	*
30	121	Adam	Fripp	AFRIPP	650.123.2234	10-Apr-05	fi
31	122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-May-03	*
32	123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-Oct-05	£ ”

2. Implement Linear Regression using Scikit-Learn.

AIM:

Task 1: use the salary data.csv file to implement linear regression in python

Task 2: Perform linear regression on insurance.csv dataset and predict value (y) for 'charges' for age value 45 (say x=45) and check whether the predicted value(y^) is same as actual y from the dataset, also calculate loss for that sample loss for an instance = y - y^.

DESCRIPTION: Simple linear regression is an approach for predicting a response using a single feature.

It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Packages used are *matplotlib, pandas, numpy, sklearn.model selection, sklearn.linear model.*

A scikit-learn linear regression script begins by importing the *LinearRegression* class:

from sklearn.linear model import LinearRegression

sklearn.linear model.LinearRegression()

Linear Regression is carried on 2 datasets. The first data set is salary data on which linear regression is carried out and the 2nd dataset is the insurance dataset on which linear regression is performed along with few other operations.

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

# Importing the dataset
dataset = pd.read_csv('insurance.csv')
X = dataset.iloc[:, : 1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, -1].values #get array of dataset in last column
```

```
X
```

```
array([[19],
       [18],
       [28],
```

```
       [18],
       [21],
       [61]])
```

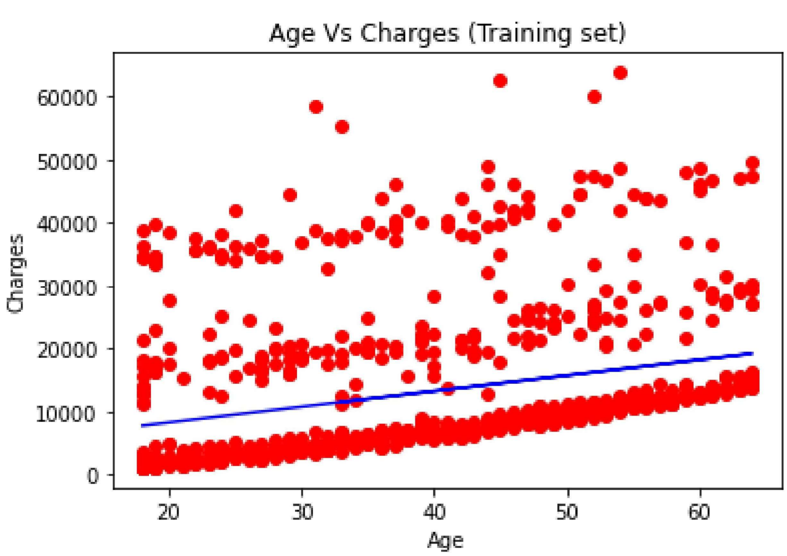
```
y
```

```
array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

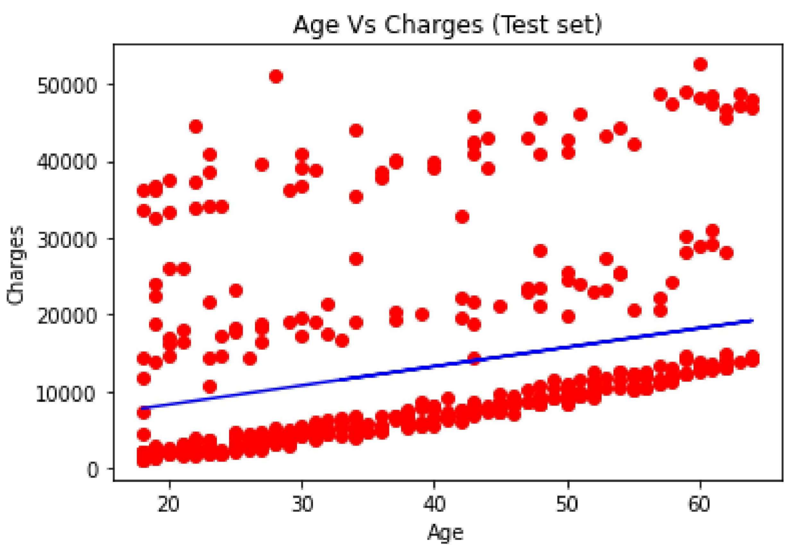
```
from sklearn.model_selection import train_test_split
# X = X.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
# X_train = X_train.reshape(-1, 1)
# X_test = X_test.reshape(-1, 1)
```

```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
# Visualizing the Training set results
v_train = plt
#to visualise scatter plot
v_train.scatter(X_train, y_train, color='red')
# for training datset
v_train.plot(X_train, regressor.predict(X_train), color='blue')
# title for our plot
v_train.title('Age Vs Charges (Training set)')
# define x-axis and y-axis
v_train.xlabel('Age')
v_train.ylabel('Charges')
# display plot for training dataset
v_train.show()
```



```
# Visualizing the Test set results using matplot library methods
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
#title for our plot
viz_test.title('Age Vs Charges (Test set)')
# x-axis
viz_test.xlabel('Age')
#y-axis
viz_test.ylabel('Charges )
#display plot
viz_test.show()
```




```
# Predicting the result of 45 Years Age
y_pred = regressor.predict([[45]])
y_pred
```

```
array([14492.89470003])
```

```
y_pred_all = regressor.predict(X_test)
y_pred_all
```

```
AiiOA.£ £t T¥IOOULg  XA3AO OO3DDdAV¿  X/JOD 33AOOt0N   ULI30T.O/J3OOA
11765.96564363,  13005.47885108, 15236.6026245    9534.84187021,
17715.62903941,  19203.04488836, 17219.82375643, 17715.62903941,
12757.57620959,   9534.84187021,  7799.52337978,  14740.79734152,
17963.5316809 ,   7799.52337978, 11270.16036065, 13501.28413407,
15236.6026245 ,   9286.93922872, 12757.57620959, 16476.11583196,
12757.57620959,   8295.32866276, 13501.28413407, 13997.08941705,
13997.08941705,  16476.11583196,  8543.23130425,  13997.08941705,
 8791.13394574,  13501.28413407, 10774.35507767, 18955.14224687,
 9782.7445117 ,   9534.84187021,  7799.52337978,   7799.52337978,
14492.89470003,   7799.52337978,  9534.84187021,  10030.64715319,
14492.89470003,  15732.40790748, 18707.23960538, 18707.23960538,
```

```
       13501.28413407, 17467.72639792, 10774.35507767, 14740.79734152,
       12509.6735681 ,  7799.52337978,  9039.03658723, 11270.16036065,
       17715.62903941,  9534.84187021, 12261.77092661, 11765.96564363,
       16476.11583196, 14492.89470003,  8295.32866276, 18211.43432239,
       17715.62903941, 11765.96564363, 18211.43432239,  7799.52337978,
       12757.57620959,  9039.03658723, 13005.47885108, 13997.08941705,
        9782.7445117 ,  7799.52337978,  8047.42602127, 18459.33696388,
       15484.50526599, 11022.25771916, 16971.92111494, 16971.92111494,
        9039.03658723, 13253.38149258, 12013.86828512, 15732.40790748,
        8047.42602127,  8295.32866276,  9782.7445117 , 10030.64715319,
       18707.23960538,  9534.84187021,  9534.84187021, 14492.89470003,
       16228.21319047,  7799.52337978, 11270.16036065, 11518.06300214,
        8295.32866276,  8047.42602127, 13253.38149258, 11765.96564363,
       14988.69998301,  7799.52337978, 13997.08941705, 14244.99205854,
        8791.13394574, 10030.64715319,  8047.42602127, 15236.6026245 ,
       10526.45243618, 17715.62903941, 19203.04488836, 10774.35507767,
       15484.50526599, 13997.08941705, 17467.72639792, 10526.45243618,
       15980.31054898,  9039.03658723, 18459.33696388, 11270.16036065,
       18459.33696388,  8047.42602127,  8295.32866276,  9782.7445117 ,
       16724.01847345, 13005.47885108,  9286.93922872,  9039.03658723,
       10774.35507767, 12261.77092661, 18459.33696388,  8047.42602127,
       13997.08941705,  8047.42602127, 15484.50526599,  9782.7445117 ,
       11765.96564363, 10774.35507767, 16476.11583196, 15236.6026245 ,
       11765.96564363,  8791.13394574, 12757.57620959, 13501.28413407,
        8047.42602127, 10030.64715319,  8047.42602127, 15236.6026245 ,
       16724.01847345, 17467.72639792,  8047.42602127, 17963.5316809 ,
        8543.23130425, 14492.89470003, 10526.45243618,  9039.03658723,
        9782.7445117 ,  8047.42602127, 12757.57620959,  7799.52337978,
       12509.6735681 , 16971.92111494, 15732.40790748, 17715.62903941,
       11270.16036065,  9039.03658723,  8295.32866276, 15732.40790748,
       12013.86828512,  7799.52337978, 16228.21319047, 12757.57620959,
       13749.18677556,  9782.7445117 ,  9039.03658723, 10774.35507767,
        8543.23130425, 15732.40790748,  7799.52337978, 13749.18677556,
       17219.82375643, 10278.54979468,  8543.23130425, 16971.92111494,
       19203.04488836, 18707.23960538,  7799.52337978, 13005.47885108,
       16228.21319047, 14244.99205854, 11518.06300214,  8047.42602127,
        8047.42602127, 18459.33696388, 13501.28413407, 18707.23960538,
       14988.69998301, 16724.01847345, 18955.14224687,  8791.13394574,
       14492.89470003,  8543.23130425, 17963.5316809 ,  8543.23130425,
       12757.57620959,  7799.52337978, 17715.62903941, 12509.6735681 ,
       17467.72639792, 15236.6026245 , 12261.77092661,  8791.13394574,
       11518.06300214, 17467.72639792, 14740.79734152,  8791.13394574,
       15236.6026245 , 10526.45243618, 15732.40790748, 13997.08941705,
       10526.45243618, 10526.45243618, 15732.40790748, 13253.38149258,
        9534.84187021,  8295.32866276,  9534.84187021,  8047.42602127,
       10278.54979468, 16228.21319047])
```

```
# MSE mean square error
# MSE is calculated by:
# measuring the distance of the observed y-values from the predicted y-values at each value o
# squaring each of these distances;
# calculating the mean of each of the squared distances.

age_45 = dataset.loc[dataset['age'] == 45, ['charges']]
```

```
age_45
y_act = float(np.mean(age_45))
y_act
error = y_act - y_pred
print(y_act, y_pred, error)
```

```
14830.199856206897 [14492.89470003] [337.30515618]
```

```
from sklearn.metrics import mean_squared_error
df_grouped = dataset.groupby(['age']).mean()
df_grouped.add_suffix('_mean').reset_index()
# type(df_grouped)
y_all = df_grouped.iloc[:, -1]
y_all
print(y_all, y_pred_all)
# mse = mean_squared_error(y_all, y_pred_all)
```

```
 17715.62903941 19203.04488836 17219.82375643 17715.62903941
 12757.57620959  9534.84187021  7799.52337978 14740.79734152
 17963.5316809   7799.52337978 11270.16036065 13501.28413407
 15236.6026245   9286.93922872 12757.57620959 16476.11583196
 12757.57620959  8295.32866276 13501.28413407 13997.08941705
 13997.08941705 16476.11583196  8543.23130425 13997.08941705
  8791.13394574 13501.28413407 10774.35507767 18955.14224687
  9782.7445117   9534.84187021  7799.52337978  7799.52337978
 14492.89470003  7799.52337978  9534.84187021 10030.64715319
 14492.89470003 15732.40790748 18707.23960538 18707.23960538
 13501.28413407 17467.72639792 10774.35507767 14740.79734152
 12509.6735681   7799.52337978  9039.03658723 11270.16036065
 17715.62903941  9534.84187021 12261.77092661 11765.96564363
 16476.11583196 14492.89470003  8295.32866276 18211.43432239
 17715.62903941 11765.96564363 18211.43432239  7799.52337978
 12757.57620959  9039.03658723 13005.47885108 13997.08941705
  9782.7445117   7799.52337978  8047.42602127 18459.33696388
 15484.50526599 11022.25771916 16971.92111494 16971.92111494
  9039.03658723 13253.38149258 12013.86828512 15732.40790748
  8047.42602127  8295.32866276  9782.7445117  10030.64715319
 18707.23960538  9534.84187021  9534.84187021 14492.89470003
 16228.21319047  7799.52337978 11270.16036065 11518.06300214
  8295.32866276  8047.42602127 13253.38149258 11765.96564363
 14988.69998301  7799.52337978 13997.08941705 14244.99205854
  8791.13394574 10030.64715319  8047.42602127 15236.6026245
 10526.45243618 17715.62903941 19203.04488836 10774.35507767
 15484.50526599 13997.08941705 17467.72639792 10526.45243618
 15980.31054898  9039.03658723 18459.33696388 11270.16036065
 18459.33696388  8047.42602127  8295.32866276  9782.7445117
 16724.01847345 13005.47885108  9286.93922872  9039.03658723
 10774.35507767 12261.77092661 18459.33696388  8047.42602127
 13997.08941705  8047.42602127 15484.50526599  9782.7445117
 11765.96564363 10774.35507767 16476.11583196 15236.6026245
 11765.96564363  8791.13394574 12757.57620959 13501.28413407
  8047.42602127 10030.64715319  8047.42602127 15236.6026245
 16724.01847345 17467.72639792  8047.42602127 17963.5316809
  8543.23130425 14492.89470003 10526.45243618  9039.03658723
```

```
 9782.7445117  8047.42602127 12757.57620959  7799.52337978
12509.6735681 16971.92111494 15732.40790748 17715.62903941
11270.16036065  9039.03658723  8295.32866276 15732.40790748
12013.86828512  7799.52337978 16228.21319047 12757.57620959
13749.18677556  9782.7445117  9039.03658723 10774.35507767
 8543.23130425 15732.40790748  7799.52337978 13749.18677556
17219.82375643 10278.54979468  8543.23130425 16971.92111494
19203.04488836 18707.23960538  7799.52337978 13005.47885108
16228.21319047 14244.99205854 11518.06300214  8047.42602127
 8047.42602127 18459.33696388 13501.28413407 18707.23960538
14988.69998301 16724.01847345 18955.14224687  8791.13394574
14492.89470003  8543.23130425 17963.5316809  8543.23130425
12757.57620959  7799.52337978 17715.62903941 12509.6735681
17467.72639792 15236.6026245 12261.77092661  8791.13394574
11518.06300214 17467.72639792 14740.79734152  8791.13394574
15236.6026245 10526.45243618 15732.40790748 13997.08941705
10526.45243618 10526.45243618 15732.40790748 13253.38149258
 9534.84187021  8295.32866276  9534.84187021  8047.42602127
10278.54979468 16228.21319047]
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred_all)
mse
```

145376685.7679913

Colab paid products - Cancel contracts here

0s completed at 9:00PM

2. Implement Linear Regression using Scikit-Learn on Housing Dataset.

AIM:

Task 1: Perform Feature Engineering on the columns that would help implementing Linear Regression and predict the regression value for lotsize=5000 (choose X as lotsize , Y as price)

Task 2: Also perform linear regression on your training data(ex: X-test) and predict respective price values, then compute loss value for each instance (a record/row in your dataset)

DESCRIPTION: Simple linear regression is an approach for predicting a response using a single feature.

It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Packages used are *matplotlib, pandas, numpy, sklearn.model selection, sklearn.linear model.*

A scikit-learn linear regression script begins by importing the *LinearRegression* class:

from sklearn.linear model import LinearRegression

sklearn.linear model.LinearRegression()

Linear Regression is carried on 2 datasets. The first data set is salary data on which linear regression is carried out and the 2nd dataset is the insurance dataset on which linear regression is performed along with few other operations.

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

# Importing the dataset
dataset = pd.read_csv('Housing.csv')

X = dataset.iloc[:, 2: 3].values
X=X.flatten()
y = dataset.iloc[:, 1: 2].values
y=y.flatten()
df=pd.DataFrame({'LotSize' :X, 'Price' :y})
df
```

	LotSize	Price
0	5850	42000.0
1	4000	38500.0
2	3060	49500.0
3	6650	60500.0
4	6360	61000.0
541	4800	91500.0
542	6000	94000.0
543	6000	103000.0
544	6000	105000.0
545	6000	105000.0

546 rows 2 columns

```
from sklearn.model_selection import train_test_split
X = X.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
# X_train = X_train.reshape(-1, 1)
# X_test = X_test.reshape(-1, 1)
```

```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred_all   regressor.predict(X_test).flatten()
y_pred_all
```

```
array([ 77468.37574046,  87260.72196868,  90599.02181921,  70161.20828986,
        57290.20775505,  67750.21395337,  61444.53645793,  74352.6292133 ,
        79248.80232741,  56548.36334382,  51652.19022971,  56103.25669708,
        66785.81621877,  95272.64160995,  48313.89037918,  82475.82551625,
        58773.8965775 ,  84590.08208826,  86518.87755745,  69827.37830481,
        77023.26909372,  85035.18873499,  73521.76347273,  58773.8965775 ,
        64263.54522059,  70198.30051042,  56325.81002045,  89486.25520236,
        50984.5302596 ,  62779.85639814,  65116.66629351,  66451.98623372,
        56399.99446157,  65598.86516081,  60999.42981119,  50168.50140725,
        64782.83630845,  56466.76045858,  53024.60239048,  48758.99702592,
        96162.85490342,  79471.35565078,  67972.76727674,  59664.10987098,
        75791.80737109, 105576.86048191,  79842.27785639,  45420.69717539,
       125094.78694134, 127320.32017502,  69456.45609919,  54767.93675687,
```

```
        78358.58903394,  89857.17740798,  66118.15624867,  57957.86772515,
        69159.7183347 ,  54990.49008024, 114041.30521403,  77802.20572551,
        53432.61681666,  59656.69142686,  51652.19022971,  52319.85019981,
        87260.72196868,  51948.9279942 ,  61444.53645793,  51652.19022971,
        62854.04083926,  56696.73222606,  75465.39583014,  55658.15005034,
        55361.41228585,  52764.95684655,  52987.51016992,  70198.30051042,
        52987.51016992,  82512.91773682,  77023.26909372,  59441.55654761,
        59070.63434199,  90599.02181921,  69456.45609919,  56325.81002045,
        56325.81002045,  73907.52256657,  55435.59672697,  72631.55017925,
        77987.66682832,  81325.96667885,  49775.3238693   58833.2441304
        76133.05580025,  71681.98933288,  59441.55654761,  56103.25669708,
        56103.25669708, 102542.71683999,  59070.63434199,  47423.67708571,
        42008.21288374,  65747.23404305, 100762.29025304,  88596.04190889,
        91711.78843605,  59070.63434199,  50984.5302596 ,  41637.29067812,
        56474.17890269,  51652.19022971,  61296.16757568,  84367.52876489,
        73907.52256657,  55213.04340361,  59627.01765042,  91504.07200091,
        74278.44477218,  56399.99446157,  73907.52256657,  57586.94551954,
        95235.54938939,  77987.66682832,  82809.65550131,  78358.58903394,
        66489.07845428,  59812.47875322,  70198.30051042,  59367.37210649,
        73907.52256657,  70198.30051042,  78358.58903394,  73610.78480207,
       125421.19848228,  73907.52256657,  78469.86569562,  52987.51016992,
        78358.58903394,  52987.51016992,  60257.58539996,  92602.00172952,
        91711.78843605,  70940.14492165,  86415.01933988,  50072.06163379,
        53877.72346339,  90599.02181921, 127320.32017502,  56325.81002045,
        55361.41228585, 145124.58604451,  54248.64566901,  62779.85639814,
        55213.04340361, 103284.56125122,  67972.76727674,  83039.62726879,
        47201.12376234,  62779.85639814,  59960.84763547,  58180.42104852,
        65005.38963182,  61296.16757568,  57957.86772515, 114412.22741964,
        43862.82391181,  46644.74045392,  69456.45609919,  84471.38698246,
        93047.10837626,  58328.78993077,  62854.04083926,  53610.65947535,
        55509.7811681 ,  48758.99702592,  59960.84763547,  57216.02331392,
        67601.84507112,  47201.12376234,  80806.67559099,  50583.93427754,
        74055.89144881,  65005.38963182])
```

```
y_pred = regressor.predict([[5000]])
y_pred
```

```
array([66489.07845428])
```

```
y_test
```

```
array([ 57000., 132000.,  53900.,  44700.,  65000.,  67000.,  67000.,
        47000.,  70000.,  42000.,  56000.,  58550.,  60000.,  99000.,
        65000., 124000.,  40000., 106000.,  35000.,  52900.,  87500.,
       155000.,  70500.,  57000.,  80000.,  83900.,  57500., 141000.,
        25000.,  51000.,  69000.,  69900.,  46200., 118500.,  75000.,
        70800.,  70000.,  27000.,  37900.,  60000., 103500.,  45000.,
        40750.,  59500.,  85000.,  75000.,  86000.,  44000.,  70000.,
        99000.,  79500.,  54000.,  87000.,  71500.,  62600.,  49000.,
        65000.,  45000.,  73000.,  85000.,  87250.,  60000.,  73000.,
        44000.,  76000.,  41000., 125000.,  30000.,  65900.,  58500.,
        78000.,  72000., 132000.,  51000.,  36000.,  75000.,  55000.,
        84000.,  95000.,  57000.,  97000.,  93000.,  57500.,  67000.,
        43000.,  89500.,  64500.,  83000., 112500.,  68500.,  60000.,
```

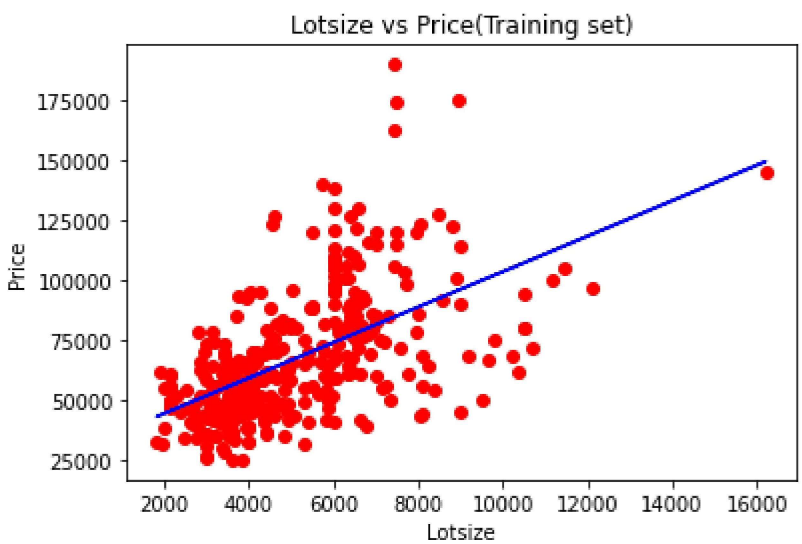
```
        63000.,   70000.,   95000.,   53000.,   42000.,   54000.,   64500.,
        38500.,   38000.,   27000.,   49900.,   86900.,   73500.,   75000.,
        51500.,   37200.,   45000.,   79000.,   70000.,  106500.,   92500.,
       103000.,   50000.,   62000.,   87000.,   74900.,   50000.,  113000.,
        75000.,  120000.,  133000.,   90000.,   95500.,   92000.,   90000.,
        80000.,   47000.,   94000.,   60000.,   80000.,  117000.,   50000.,
       105000.,   87000.,   52000.,   89000.,   33000.,   69000.,   75000.,
        65000.,   58500.,   71000.,   60500.,   79000.,   59900.,  140000.,
        38000.,   45000.,   84900.,   54800.,   57000.,   53500.,  175000.,
        83000.,   85000.,   25245.,   57250.,   48000.,   66000.,   83800.,
        86900.,   55000.,   83900.,   49000.,   60000.,   82000.,   50000.,
       145000.,   62500.,   64000.,   34000.,   60000.,   49000.,   70000.,
        48000.,  105000.,   30000.,   53900.,   54000.,   53000.,   91500.])
```

```
from sklearn.linear_model import LinearRegression
from sklearn import tree
  'decision_tree=tree.DecisionTreeRegressor()
decision_tree=decision_tree.fit(X_train.reshape(-1,1),y_train)
linreg=regressor.predict(X_test.reshape(-1,1))
d_tree=decision_tree.predict(X_test.reshape(-1,1)) ' ' '

X_test=X_test.flatten()
y_test=y_test.flatten()
y_pred_all=y_pred_all.flatten()
df_predict=pd.DataFrame({'LotSize' :X_test, 'Price' :y_test, 'Predicted_price' :y_pred_all, 'Loss' :
df_predict
```

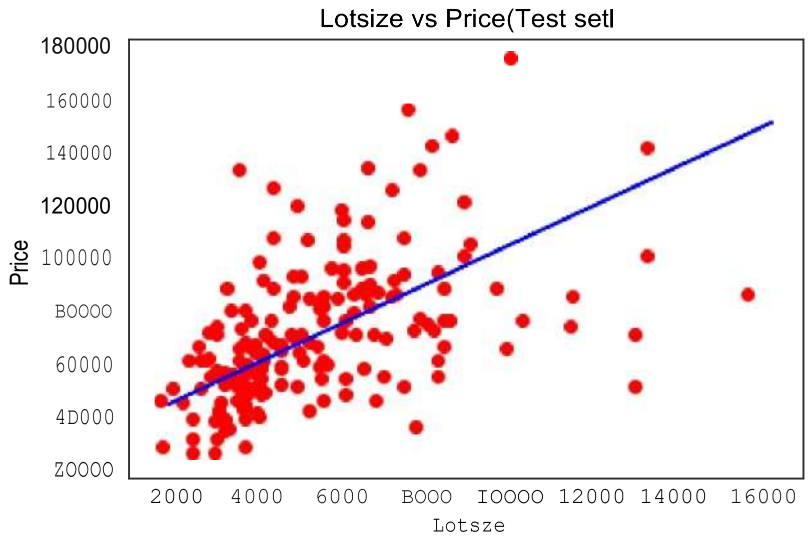
	LotSize	Price	Predicted_price	**Loss**
0	6480	57000.0	77468.375740	-20468.375740
1	7800	132000.0	87260.721969	44739.278031
2	8250	53900.0	90599.021819	-36699.021819
3	5495	44700.0	70161.208290	-25461.208290
4	3760	65000.0	57290.207755	7709.792245
177	2400	30000.0	47201.123762	-17201.123762
178	6930	53900.0	80806.675591	-26906.675591
179	2856	54000.0	50583.934278	3416.065722
180	6020	53000.0	74055.891449	-21055.891449
181	4800	91500.0	65005.389632	26494.610368

182 rows 4 columns

```
# Visualizing the Training set results
v_train = plt
#to visualise scatter plot
v_train.scatter(X_train, y_train, color='red')
# for training datset
v_train.plot(X_train, regressor.predict(X_train), color='blue')
# title for our plot
v_train.title('Lotsize vs Price(Training set)')
# define x-axis and y-axis
v_train.xlabel('Lotsize')
v_train.ylabel('Price')
# display plot for training dataset
v_train.show()
```




```
# Visualizing the Test set results using matplot library methods
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
#title for our plot
viz_test.title('Lotsize vs Price(Test set)')
# x-axis
viz_test.xlabel('Lotsize')
#y-axis
viz_test.ylabel('Price')
#display plot
viz_test.show()
```



Colab paid products Cancel contracts here

0s completed at 9:14 PM

Sunkara Rohit
160119733043

Gradient Descent Algorithm Sep 22, 2022

a. Gradient Descent:

Gradient Descent Algorithm is an iterative optimization algorithm which is used to find the local minimum of a function. In machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters.

The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.

Cost function: The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.

Learning Rate: It is defined as the step size taken to reach the minimum or lowest point.

b. Gradient Descent Algorithm:

1. Choose a starting point (initialization)
2. Calculate gradient at this point
3. Make a scaled step in the opposite direction to the gradient (objective: minimize)
4. Repeat points 2 and 3 until one of the criteria is met:
   - maximum number of iterations reached
   - Step size is smaller than the tolerance (due to scaling or a small gradient).

c. Parameter setting:

Learning Rate: We have the direction we want to move in, now we must decide the size of the step we must take.

It must be chosen carefully to end up with local minima.

If the learning rate is too high, we might OVERSHOOT the minima and keep bouncing, without reaching the minima

If the learning rate is too small, the training might turn out to be too long

Number of iterations: The process is repeated until our loss function is a very small value or ideally 0

Sunkara Rohit
160119733043

d. Implementation of Gradient Descent (code):

```
def gradientdescent(X,Y):
      l=0.08
      m=0
      c=0
      L = 0.0001
      iterations = 1000
      n = float(len(X))
      for i in range(iterations):
            Y_pred = m*X + c
            D_m = (-2/n) * sum(X * (Y - Y_pred))
            D_c = (-2/n) * sum(Y - Y_pred)
            m = m - L * D_m
            c = c - L * D_c
      print(m,c)
```

Output:

1.4796491688889395 0.10148121494753726

e. Results:

The gradient descent function takes X and Y as an input where X is the input variable and Y is the output variable.The number of iterations taken is 1000.The values obtained m and c can be taken for prediction.

f. Conclusion:

Gradient Descent Algorithm helps to find the minimum of the cost function.The values m and c obtained can be used for prediction.

AIM: Construct tree-based model for classification of samples given in diabetes.csv dataset. use ID3 algorithm. Classify the given test data and predict its label.

test data = 3 75 55 32 88 31 0.246 27

Write all possible rules (each branch in the tree represent a rule)

DESCRIPTION:

ID3 uses a top-down greedy approach to build a decision tree. Top-down a

```
#Libraries required
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#Loading the dataset
df = pd.read_csv("diabetes.csv")
df.head()
```

```
#Replacing 0 with NaN wherever required
df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = df[['Glucose', 'BloodPr

#Finding median of all columns
df.groupby(['Outcome']).median()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed3
Outcome							
0	2.0	107.0	70.0	27.0	102.5	30.1	
1	4.0	140.0	74.5	32.0	169.5	34.3	

```
#Replacing NaN values with median of the dataset
def median_target(cols):
  for var in cols:
    df[var].fillna(df.groupby(['Outcome'])[var].transform('median'), inplace=True)

list_cols = df.columns.values.tolist()
median_target(list_cols)
```

```
#Correlation matrix to find feature columns
df.corr()
```

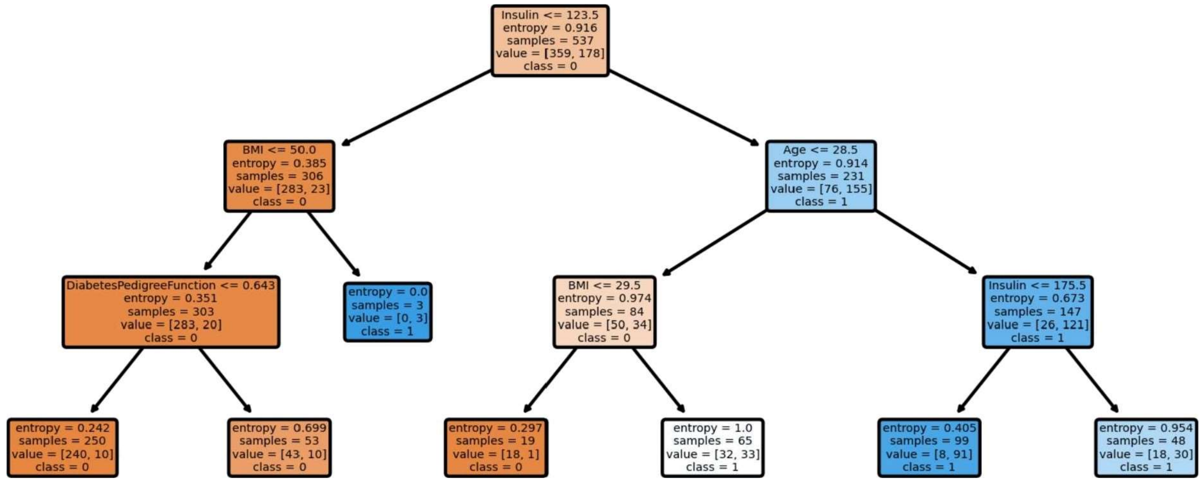
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.130155	0.209151	0.089028	0.058767
Glucose	0.130155	1.000000	0.225141	0.229289	0.490015
BloodPressure	0.209151	0.225141	1.000000	0.199349	0.070128
SkinThickness	0.089028	0.229289	0.199349	1.000000	0.200129
Insulin	0.058767	0.490015	0.070128	0.200129	1.000000
BMI	0.023890	0.236171	0.286399	0.566086	0.238443
DiabetesPedigreeFunction	-0.033523	0.138353	-0.001443	0.106280	0.146878
Age	0.544341	0.268910	0.325135	0.129537	0.123629
Outcome	0.221898	0.495990	0.174469	0.295138	0.377081

#Sp11tt1ng dataset 1nto test and traln

```
feature_cols = ['Insulin', 'BMI', 'Age', 'Glucose', 'BloodPressure', 'DiabetesPedigreeFunctio
X = df[feature_cols]
y = df.0utcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=44)


#Creating a Decision Tree Classifier and training it
clf   DecisionTreeClassifier(criterion='entropy', max_depth=3)
clf = clf.fit(X_train, y_train)
y pred = clf.predict(X test)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("F1 score: ", metrics.f1_score(y_test, y_pred))
```

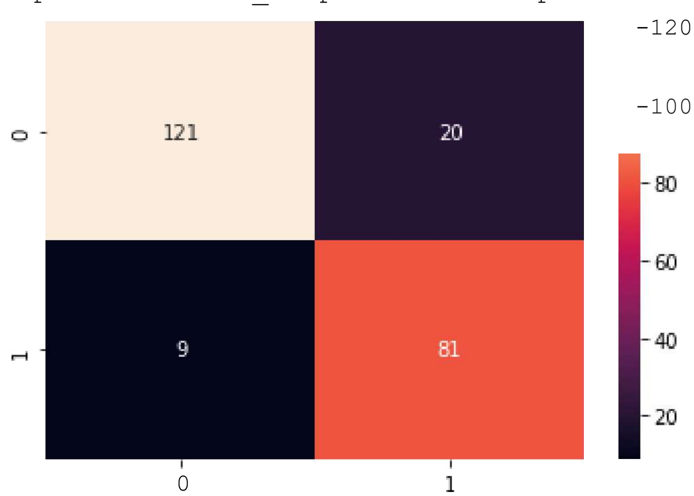
```
    Accuracy:  0.8744588744588745
    F1 score:  0.8481675392670157
```

```
#Plotting the Decision Tree
plt.figure(figsize=(7, 3), dpi=300)
tree.plot_tree(clf, feature_names=feature_cols, class_names=['0', '1'], filled=True, rounded=
plt.savefig('diabetes.png')
```




```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot=True, fmt='d')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efee0f78c10>
```




```
#Test:ing mode1 on g1ven test data
test data = [ 88, 31, 27, 75, 55, 0.246]
test_data = np . array(test_data)
pn1nt ( test_dat a )
y_pred = clf.predict(test_data.reshape(1, -1))
pr1nt (y_pred )
```

```
[88.       31.       27.       75.       55.        0.246]
[0]
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```

We can also follow the decision tree and reach the node with insulin <= 95.5 (here insulin is 88 so it results in class 0/negative)

AIM: Implementation of Naive Bayes Classifier

Predict class label for given test sample = {Sunny, Hot, Normal, False}

DESCRIPTION: Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

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

df = pd.read_csv("weather.nominal.csv") outlook
= df.iloc[:, [0]].values.flatten() temp = df.iloc[:,
[1]].values.flatten() humidity = df.iloc[:,
[2]].values.flatten() windy = df.iloc[:,
[3]].values.flatten() play = df.iloc[:,
[4]].values.flatten()

#print(outlook, temp, humidity, windy, play)


# Import LabelEncoder
from sklearn import preprocessing #
Creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers()
outlook_encoded = le.fit_transform(outlook)
temp_encoded = le.fit_transform(temp)
humidity_encoded = le.fit_transform(humidity)
windy_encoded = le.fit_transform(windy)

print(outlook_encoded, temp_encoded, humidity_encoded, windy_encoded, sep='\n')
```

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
[1 1 1 2 0 0 0 2 0 2 2 2 1 2]
[0 0 0 0 1 1 1 0 1 1 1 0 1 0]
[0 1 0 0 0 1 1 0 0 0 1 1 0 1]
```

```
# Converting string labels into numbers label
= le.fit_transform(play) print("Play:", label)
```

```
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
#Combinig weather and temp into single listof tuples import
pandas as pd
#features=zip(weather_encoded,temp_encoded)
```

```
d1 = pd.DataFrame(outlook_encoded)
d2 = pd.DataFrame(temp_encoded)
d3 = pd.DataFrame(humidity_encoded)
d4 = pd.DataFrame(windy_encoded)

#print(d1, d2, d3, d4)
concate = pd.concat([d1,d2,d3,d4],axis=1)
#cols=d1.join(d2)
#features=pd.DataFrame(cols) print(concate)
#print(features)
```

```
    0  0  0  0
0   2  1  0  0
1   2  1  0  1
2   0  1  0  0
3   1  2  0  0
4   1  0  1  0
5   1  0  1  1
6   0  0  1  1
7   2  2  0  0
8   2  0  1  0
9   1  2  1  0
10  2  2  1  1
11  0  2  0  1
12  0  1  1  0
13  1  2  0  1
```

Generating Model

Generate a model using naive bayes classifier in the following steps:

1. Create naive bayes classifier
2. Fit the dataset on classifier
3. Perform prediction

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(concate,label)

sample =[2, 1, 1, 0]

#Predict Output
predicted = model.predict([sample])
print("Predicted Value:", predicted)
```

```
Predicted Value: [1]
```