Name.: Suryansh Sandeep Kumar Singh
Roll No.: S1951122
PRN: 72018560C
DIV: B (Batch B2)

314448: LABORATORY PRACTICE – I (ML)

Teaching Scheme: **Examination Scheme:**
Practicals : 4 Hours/Week **Termwork : 25Marks** **Practical : 25MMarks**

CourseObjectives:

1. The objective of this course is to provide students with the fundamental elements of machine learning for classification, regression, clustering.

2. Design and evaluate the performance of a different machine learning models.

CourseOutcomes:

On completion of the course, students will be able to–

1. CO1: Implement different supervised and unsupervised learning algorithms.
2. CO2: Evaluate performance of machine learning algorithmsfor real-world applications.

1. **Data preparation:**

Download heart dataset from following link.

https://www.kaggle.com/zhaoyingzhu/heartcsv

Perform following operation on given dataset. a) Find Shape of Data b) Find Missing Values c) Find data type of each column d) Finding out Zero's e) Find Mean age of patients f) Now extract only Age, Sex, ChestPain, RestBP, Chol. Randomly divide dataset in training (75%) and testing (25%).

Through the diagnosis test I predicted 100 report as COVID positive, but only 45 of those were actually positive. Total 50 people in my sample were actually COVID positive. I have total 500 samples. Create confusion matrix based on above data and find I. Accuracy II. Precision III. Recall IV. F-1 score

2. **Assignment on Regression technique**

Download temperature data from below link.
https://www.kaggle.com/venky73/temperaturesof-india?select=temperatures.csv

This data consists of temperatures of INDIA averaging the temperatures of all places month wise. Temperatures values are recorded in CELSIUS

a. Apply Linear Regression using suitable library function and predict the Month-wistemperature.

b. Assess the performance of regression models using MSE, MAE and R-Square metrics

c. Visualize simple regression model.

3. **Assignment on Classification technique**

Every year many students give the GRE exam to get admission in foreign Universities. The data set contains GRE Scores (out of 340), TOEFL Scores (out of 120), University Rating (out of 5), Statement of Purpose strength (out of 5), Letter of Recommendation strength (out of 5), Undergraduate GPA (out of 10), Research Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is the target variable.

Data Set Available on kaggle (The last column of the dataset needs to be changed to 0 or

Data Set : https://www.kaggle.com/mohansacharya/graduate-admissions

The counselor of the firm is supposed check whether the student will get an admission or not based on his/her GRE score and Academic Score. So to help the counselor to take appropriate decisions build a machine learning model classifier using Decision tree to predict whether a student will get admission or not.

Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary. Perform data-preparation (Train-Test Split)

C. Apply Machine Learning Algorithm

D. Evaluate Model.

4. Download the following customer dataset from below link:

Data Set: https://www.kaggle.com/shwetabh123/mall-customers
This dataset givesthe data of Income and money spent by the customers visiting a Shopping Mall. The data set contains Customer ID, Gender, Age, Annual Income, Spending Score. Therefore, as a mall owner you need to find the group of people who are the profitable customers for the mall owner. Apply at least two clustering algorithms (based on Spending Score) to find the group of customers.
a. Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary.
b. Perform data-preparation(Train-Test Split)
c. Apply Machine Learning Algorithm
d. Evaluate Model.
e. Apply Cross-Validation and Evaluate Model

5. Assignment on Association Rule Learning Download Market Basket Optimization dataset from below link.

Data Set:

https://www.kaggle.com/hemanthkumar05/market-basket-optimization

This dataset comprises the list of transactions of a retail company over the period of one week. It contains a total of 7501 transaction records where each record consists of the list of items sold in one transaction. Using this record of transactions and items in each transaction, find the association rules between items. There is no header in the dataset and the first row contains the first transaction, so mentioned header = None here while loading dataset.
a. Follow following steps:
b. Data Preprocessing
c. Generate the list of transactions from the dataset
d. Train Apriori algorithm on the dataset
e. Visualize the list of rules
F. Generated rules depend on the values of hyper parameters. By increasing the minimum confidence value and find the rules accordingly

Assignment No. 1

AIM: To Write a python program to perform data preparation operation and perform exploratory data analysis.

THEORY:

Data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data.

Data preparation helps:

- Fix errors quickly — Data preparation helps catch errors before processing. After data has been removed from its original source, these errors become more difficult to understand and correct.
- Produce top-quality data — Cleaning and reformatting datasets ensures that all data used in analysis will be high quality.
- Make better business decisions — Higher quality data that can be processed and analyzed more quickly and efficiently leads to more timely, efficient and high-quality business decisions.

Loading data, cleaning data (removing unnecessary data or erroneous data), transforming data formats, and rearranging data are the various steps involved in the data preparation step

Pandas

Pandas is a software library written for Python. It is very famous in the data science community because it offers powerful, expressive, and flexible data structures that make data manipulation, analysis easy

To use the pandas library, first import it

import pandas as pd

Loading data

load the dataset in the system using the .read_csv() function in pandas.

data = pd.read_csv('path_to_file.csv')

Missing Data

Handling Missing Data

For numerical data, pandas uses a floating point value **NaN** (Not a Number) to represent missing data. It is a unique value defined under the library **Numpy**

isnull() which returns a boolean true or false value. True, when the data at that particular index is actually missing or NaN. The opposite of this is the **notnull()** function.

To check if and what index in the dataset contains null value

data.isnull()

To check where the dataset does not contain null value -opposite of isnull()

data.notnull()

dropna() function - to filter out missing data and to remove the null (missing) value and see only the non-null values.

data.dropna()

Filling in Missing Data

To replace or rather "fill in" the null data, you can use the fillna() function

Data Transformation

To replace a non-null value with a different value replace() function can be used.

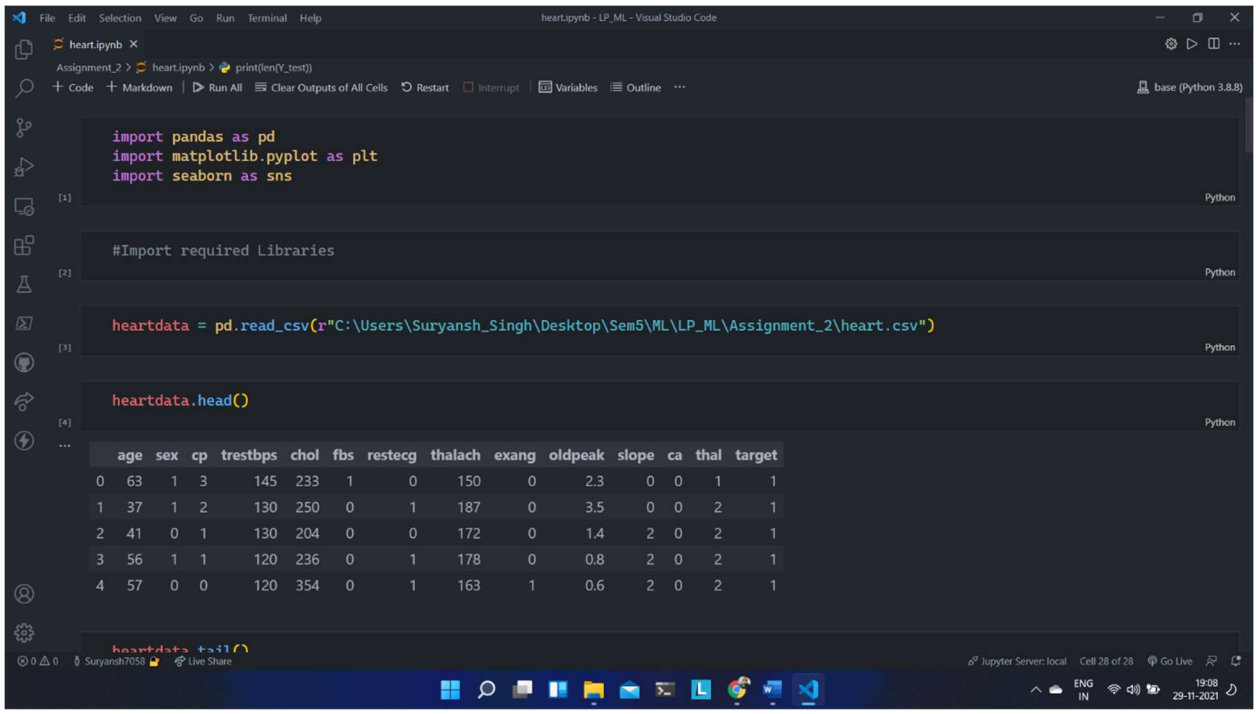
data = pd.Series([1,2,-99,4,5,-99,7,8,-99])

Replace the placeholder -99 as NaN

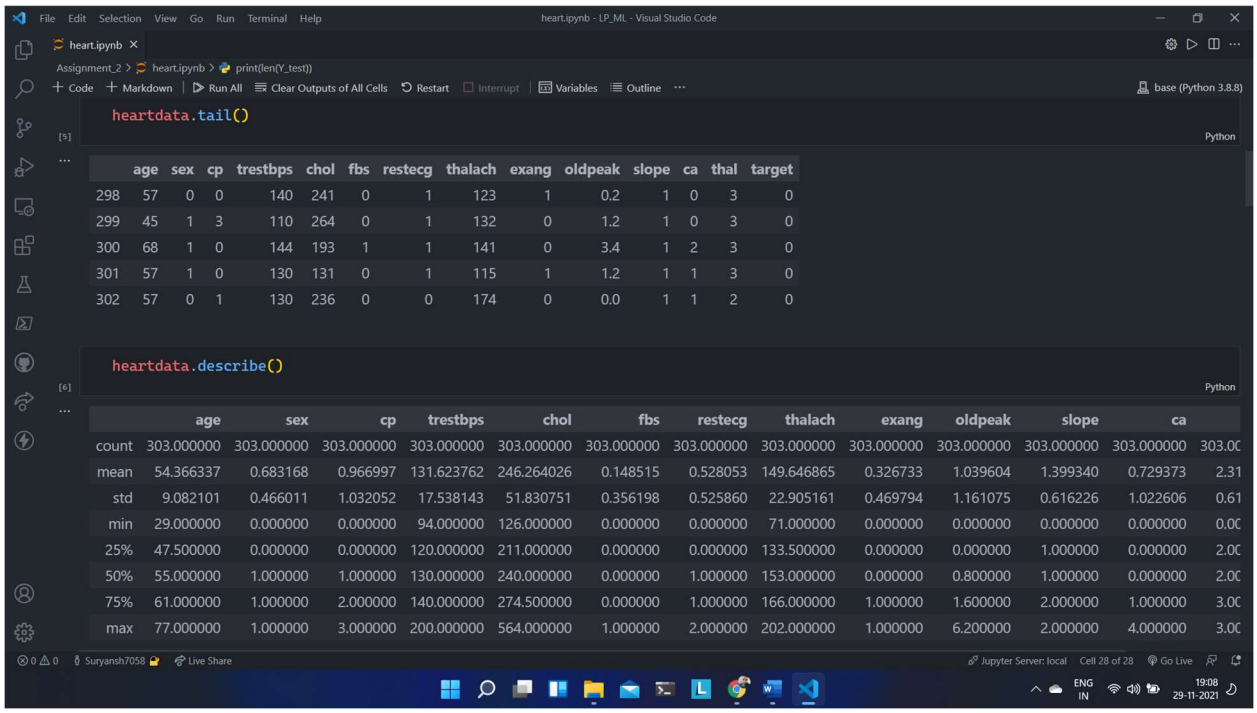
data.replace(-99, np.nan)

Concatenating Pandas Series

To do this, we can use the concat() function in pandas
 # Create a new Series
new_data = pd.Series([-100, 11, 12, 13])
combined_series = pd.concat([data, new_data], ignore_index = True)

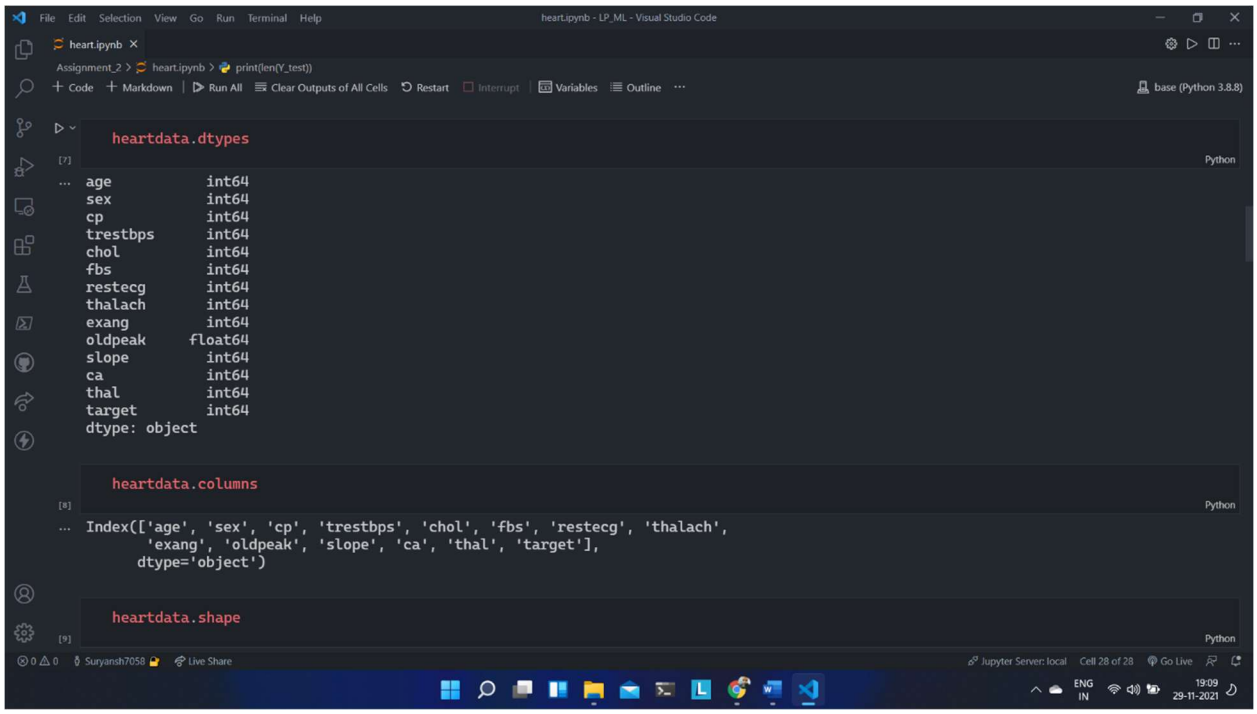


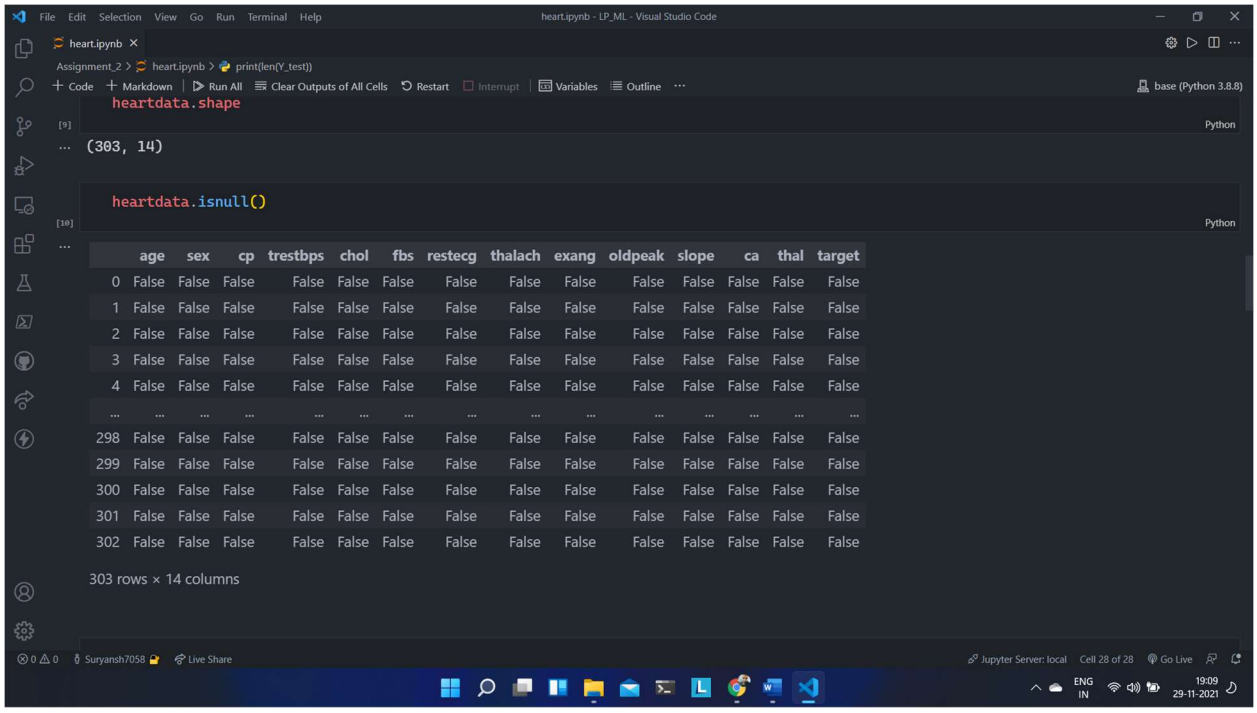
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

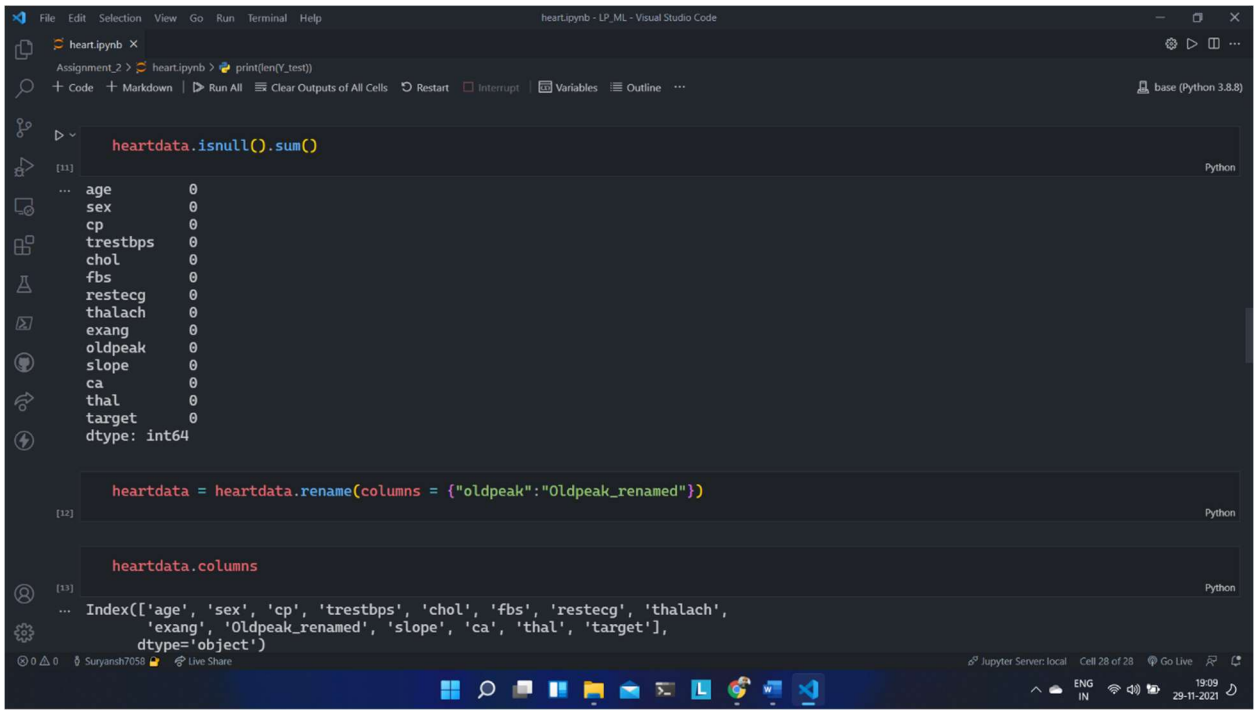


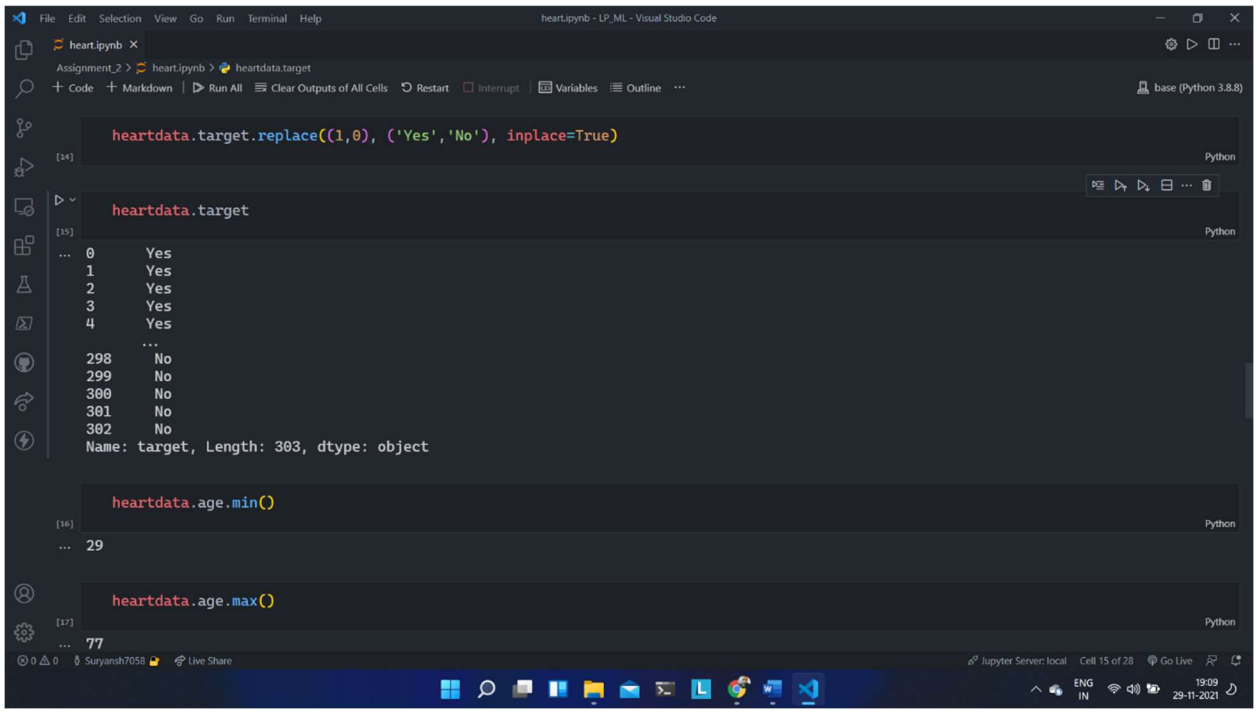
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

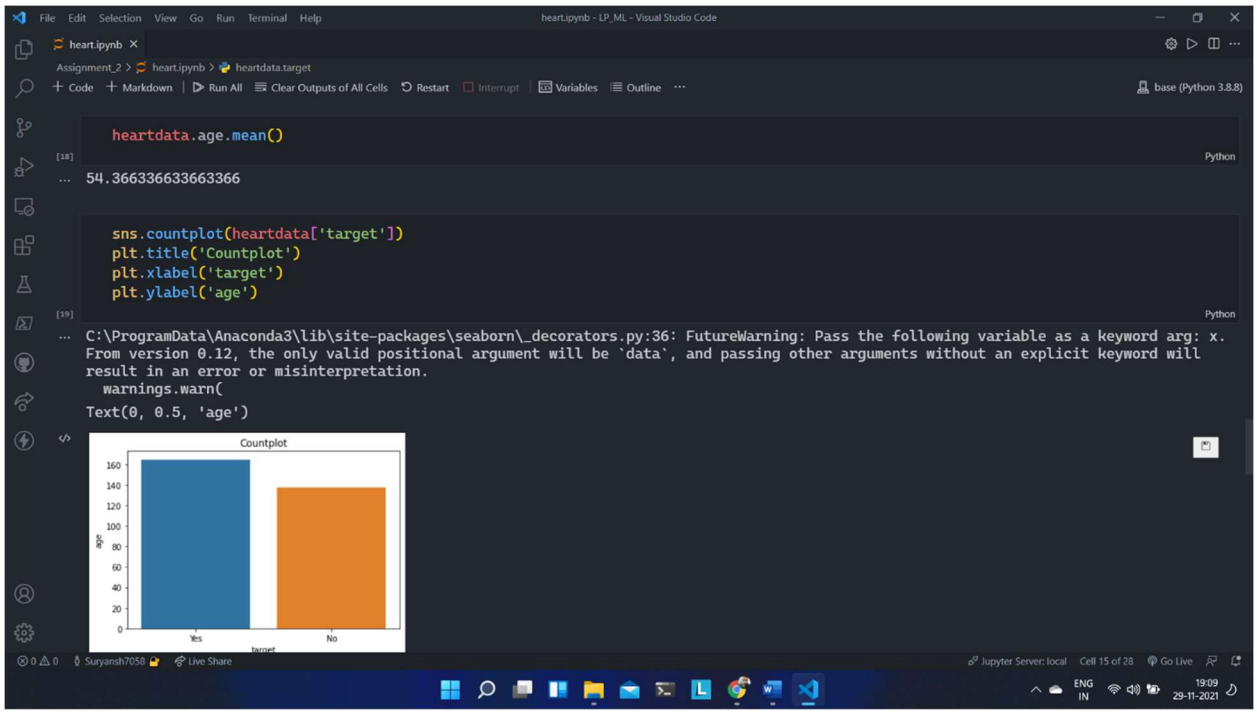
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.31
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0

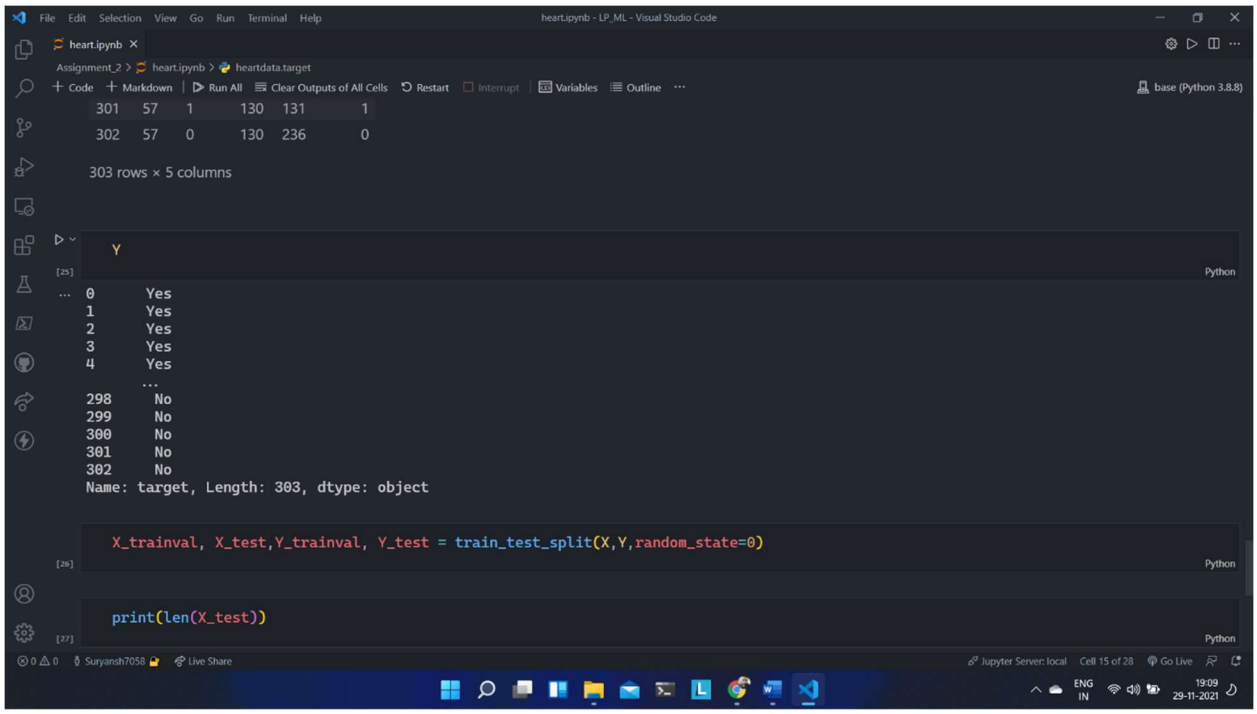

heart.ipynb - LP_ML - Visual Studio Code
heartdata.dtypes
age int64
sex int64
cp int64
trestbps int64
chol int64
fbs int64
restecg int64
thalach int64
exang int64
oldpeak float64
slope int64
ca int64
thal int64
target int64
dtype: object
heartdata.columns
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
dtype='object')
heartdata.shape

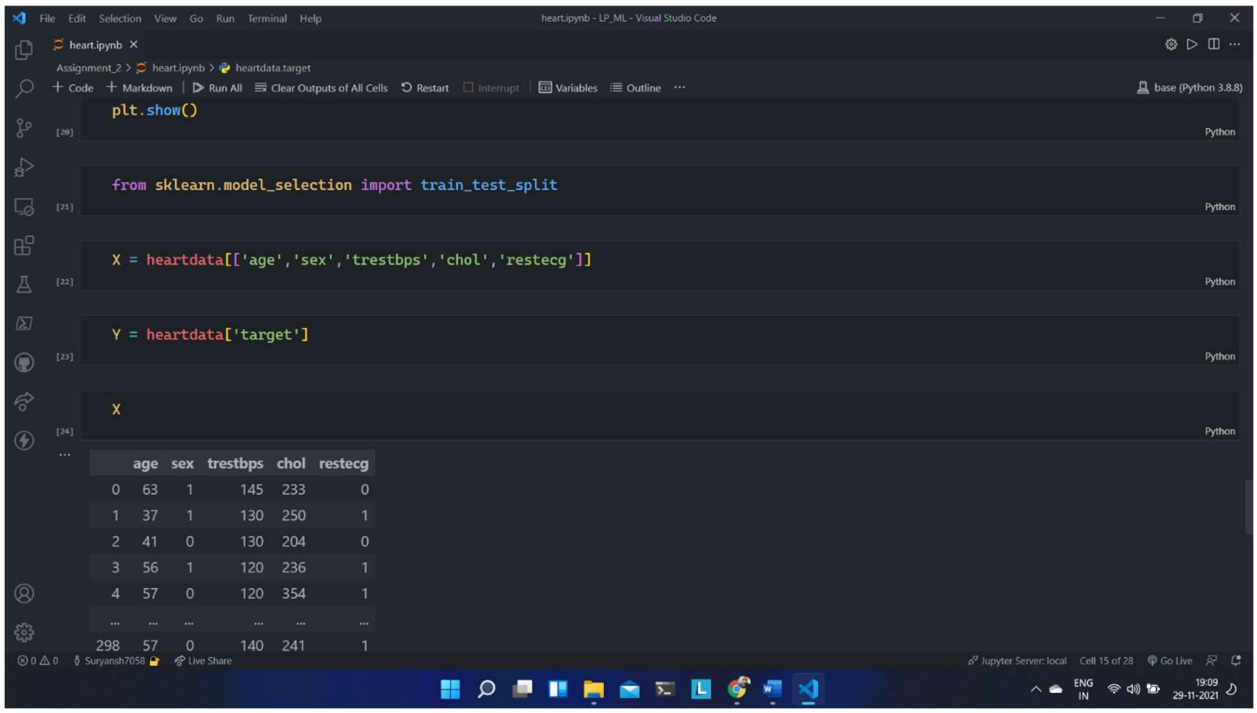

heartdata.shape
(303, 14)
heartdata.isnull()
303 rows × 14 columns

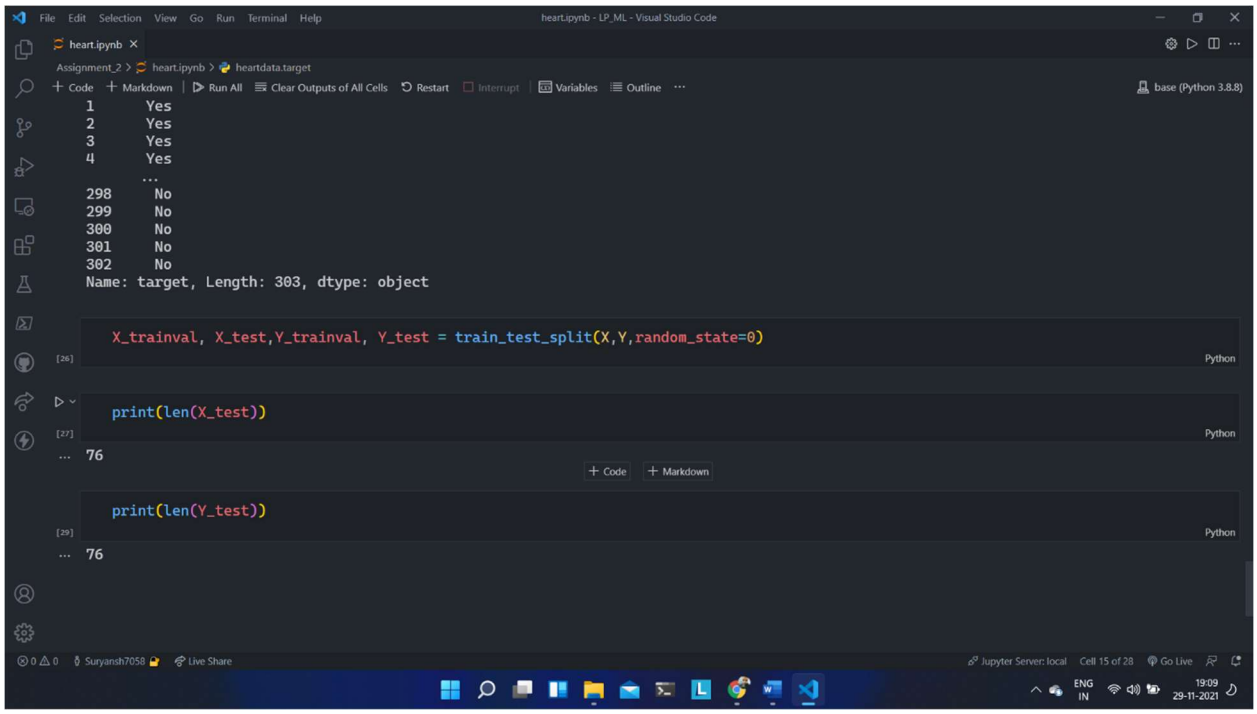

heart.ipynb - LP_ML - Visual Studio Code
heartdata.isnull().sum()
age 0
sex 0
cp 0
trestbps 0
chol 0
fbs 0
restecg 0
thalach 0
exang 0
oldpeak 0
slope 0
ca 0
thal 0
target 0
dtype: int64
heartdata = heartdata.rename(columns = {"oldpeak":"Oldpeak_renamed"})
heartdata.columns
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
'exang', 'Oldpeak_renamed', 'slope', 'ca', 'thal', 'target'],
dtype='object')


heart.ipynb - LP_ML - Visual Studio Code
heartdata.target.replace((1,0), ('Yes','No'), inplace=True)
heartdata.target
0 Yes
1 Yes
2 Yes
3 Yes
4 Yes
298 No
299 No
300 No
301 No
302 No
Name: target, Length: 303, dtype: object
heartdata.age.min()
29
heartdata.age.max()
77


heart.ipynb - LP_ML - Visual Studio Code
heartdata.age.mean()
54.366336633663366
sns.countplot(heartdata['target'])
plt.title('Countplot')
plt.xlabel('target')
plt.ylabel('age')
C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
warnings.warn(
Text(0, 0.5, 'age')
Countplot
age
target
Yes
No


heart.ipynb - LP_ML - Visual Studio Code
301 57 1 130 131 1
302 57 0 130 236 0
303 rows × 5 columns
Y
0 Yes
1 Yes
2 Yes
3 Yes
4 Yes
298 No
299 No
300 No
301 No
302 No
Name: target, Length: 303, dtype: object
X_trainval, X_test,Y_trainval, Y_test = train_test_split(X,Y,random_state=0)
print(len(X_test))


heart.ipynb - LP_ML - Visual Studio Code
plt.show()
from sklearn.model_selection import train_test_split
X = heartdata[['age','sex','trestbps','chol','restecg']]
Y = heartdata['target']
X
age sex trestbps chol restecg
0 63 1 145 233 0
1 37 1 130 250 1
2 41 0 130 204 0
3 56 1 120 236 1
4 57 0 120 354 1
298 57 0 140 241 1



Conclusion:

Here we have learned and implemented various data cleaning and preparation techniques in python.

Assignment No.2

AIM: Write a program to Apply Linear Regression using suitable library function and predict the Month-wise temperature and evaluate the performance

THEORY:

Mean Squared Error: General steps to calculate MSE the from a set of X and Y values:

1. Find the regression line.

2. Insert your X values into the linear regression equation to find the new Y values (Y').

3. Subtract the new Y value from the original to get the error.

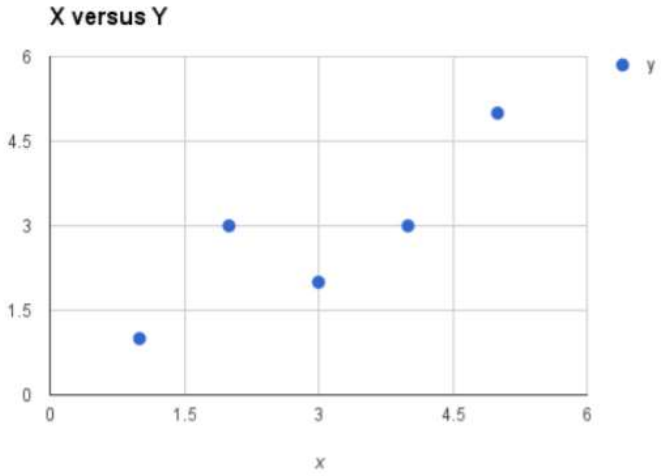
4. Square the errors.

5. Add up the errors.

6. Find the mean

Tutorial Data Set

The data set we are using is completely made up. Below is the raw data. The attribute x is the input variable and y is the output variable that we are trying to predict. If we got more data, we would only have x values and we would be interested in predicting y values. Below is a simple scatter plot of x versus y



x	y
1	1
2	3
4	3
3	2
5	5

Plot of the Dataset for Simple Linear Regression

We can see the relationship between x and y looks kind of linear. As in, we could probably draw a line somewhere diagonally from the bottom left of the plot to the top right to generally describe the relationship between the data.

This is a good indication that using linear regression might be appropriate for this little dataset.

Simple Linear Regression

When we have a single input attribute (x) and we want to use linear regression, this is called simple linear regression. If we had multiple input attributes (e.g. x1, x2, x3, etc.) This would be called multiple linear regression. The procedure for linear regression is different and simpler than that for multiple linear regression, so it is a good place to start. In this section we are going to create a simple linear regression model from our training data, then make predictions for our training data to get an idea of how well the model learned the relationship in the data.

With simple linear regression we want to model our data as follows:

$$y = B0 + B1 * x$$

This is a line where y is the output variable we want to predict, x is the input variable we know and B0 and B1 are coefficients that we need to estimate that move the line around. Technically, B0 is called the intercept because it determines where the line intercepts the y-axis. In machine learning we can call this the bias, because it is added to offset all predictions that we make. The B1 term is called the slope because it defines the slope of the line or how x translates into a y value before we add our bias.

The goal is to find the best estimates for the coefficients to minimize the errors in predicting y from x. Simple regression is great, because rather than having to search for values by trial and

error or calculate them analytically using more advanced linear algebra, we can estimate them directly from our data. We can start off by estimating the value for B1 as:

$$B1 = \frac{\sum (Xi - \bar{X}) * (Yi - \bar{Y})}{\sum (Xi - \bar{X})^2}$$

Where mean() is the average value for the variable in our dataset. The xi and yi refer to the fact that we need to repeat these calculations across all values in our dataset and i refers to the i'th value of x or y. We can calculate B0 using B1 and some statistics from our dataset, as follows:

$$B0 = \bar{Y} - (B1 * \bar{X})$$

Estimating Slope (B1) Let's start with the top part of the equation, the numerator. First we need to calculate the mean value of x and y. The mean is calculated as: sum(x) / n Where n is the number of values (5 in this case). Let's calculate the mean value of our x and y variables:

$\bar{X} = 3$, $\bar{Y} = 2.8$ We now have the parts for calculating the numerator. All we need to do is multiple the error for each x with the error for each y and calculate the sum of these multiplications.

	x - mean(x)	y - mean(y)	Multiplication
2	-2	-1.8	3.6
3	-1	0.2	-0.2
4	1	0.2	0.2
5	0	-0.8	0
6	2	2.2	4.4

Summing the final column we have calculated our numerator as 8.

Now we need to calculate the bottom part of the equation for calculating B1, or the denominator. This is calculated as the sum of the squared differences of each x value from the mean. We have already calculated the difference of each x value from the mean, all we need to do is square each value and calculate the sum. Calculating the sum of these squared values gives us up denominator of 10 Now we can calculate the value of our slope

B1 = 8 / 10 so further B1 = 0.8

Estimating Intercept (B0)

This is much easier as we already know the values of all of the terms involved. $B0 = \bar{Y} - (B1 * \bar{X})$ or B0 = 2.8 – 0.8 * 3 , or further B0 = 0.4

Making Predictions

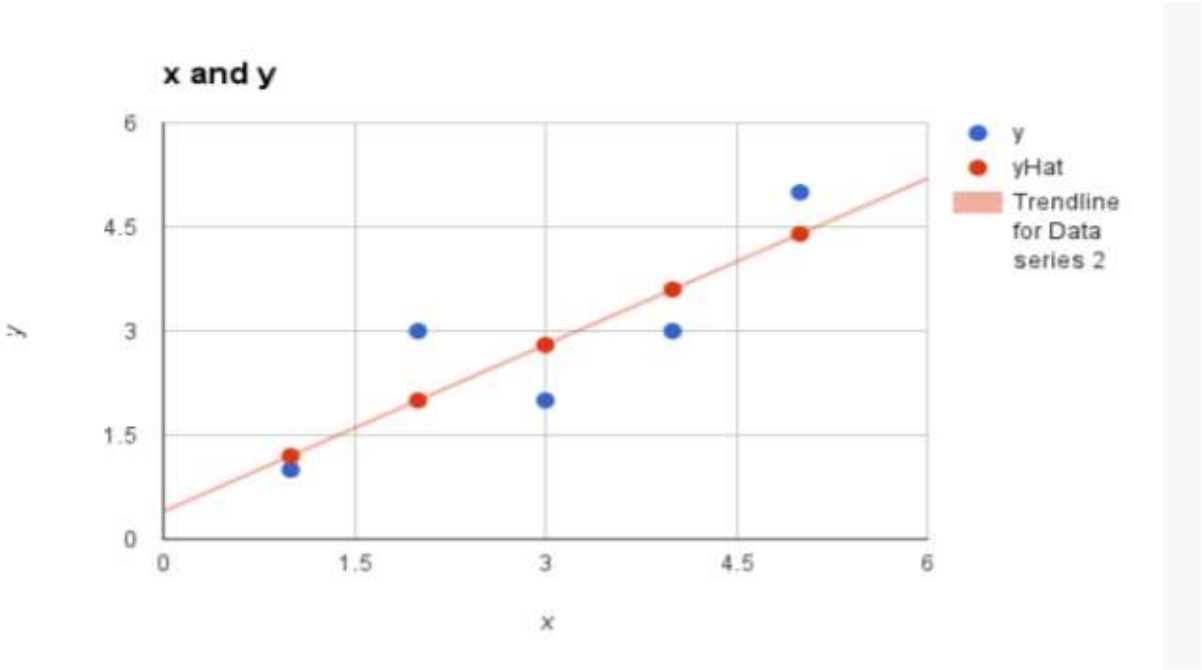
We now have the coefficients for our simple linear regression equation.

y = B0 + B1 * x or y = 0.4 + 0.8 * x

Let's try out the model by making predictions for our training data.

	x	y	predicted y
2	1	1	1.2
3	2	3	2
4	4	3	3.6
5	3	2	2.8
6	5	5	4.4

We can plot these predictions as a line with our data. This gives us a visual idea of how well the line models our data.



Estimating Error We can calculate a error for our predictions called the Root Mean Squared Error or RMSE.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

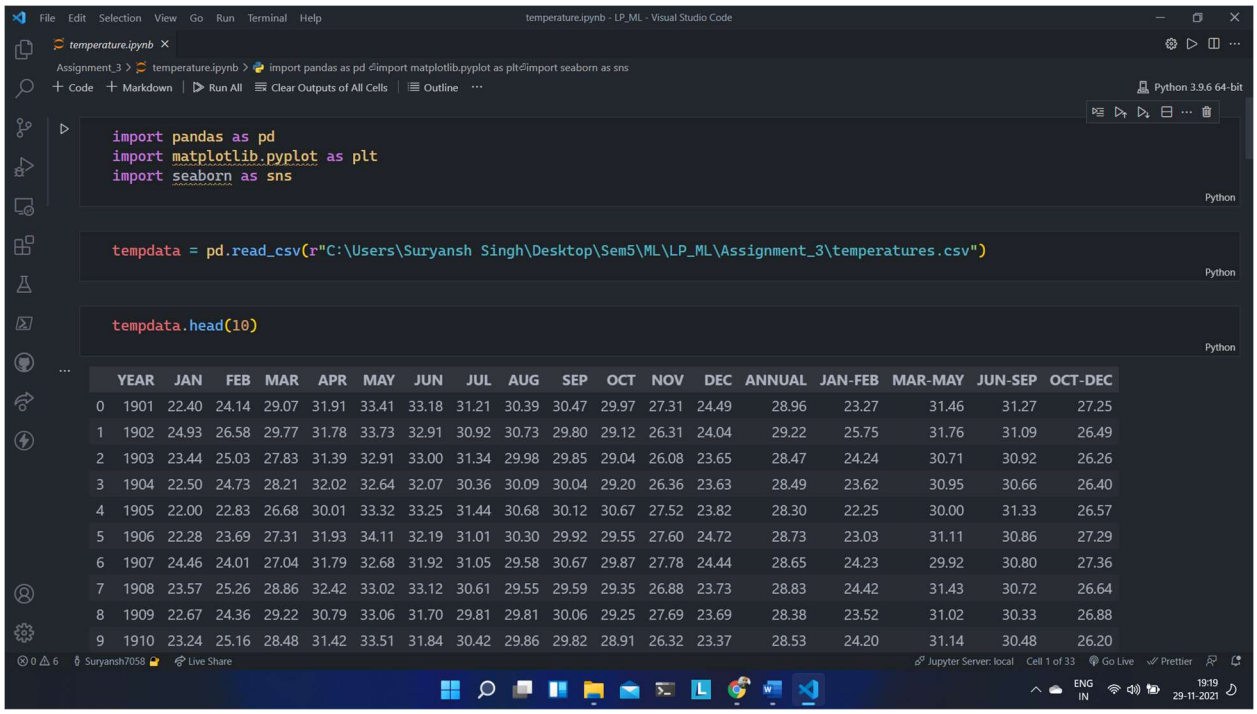
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

Where sqrt() is the square root function, p is the predicted value and y is the actual value, i is the index for a specific instance, n is the number of predictions, because we must calculate the error across all predicted values.

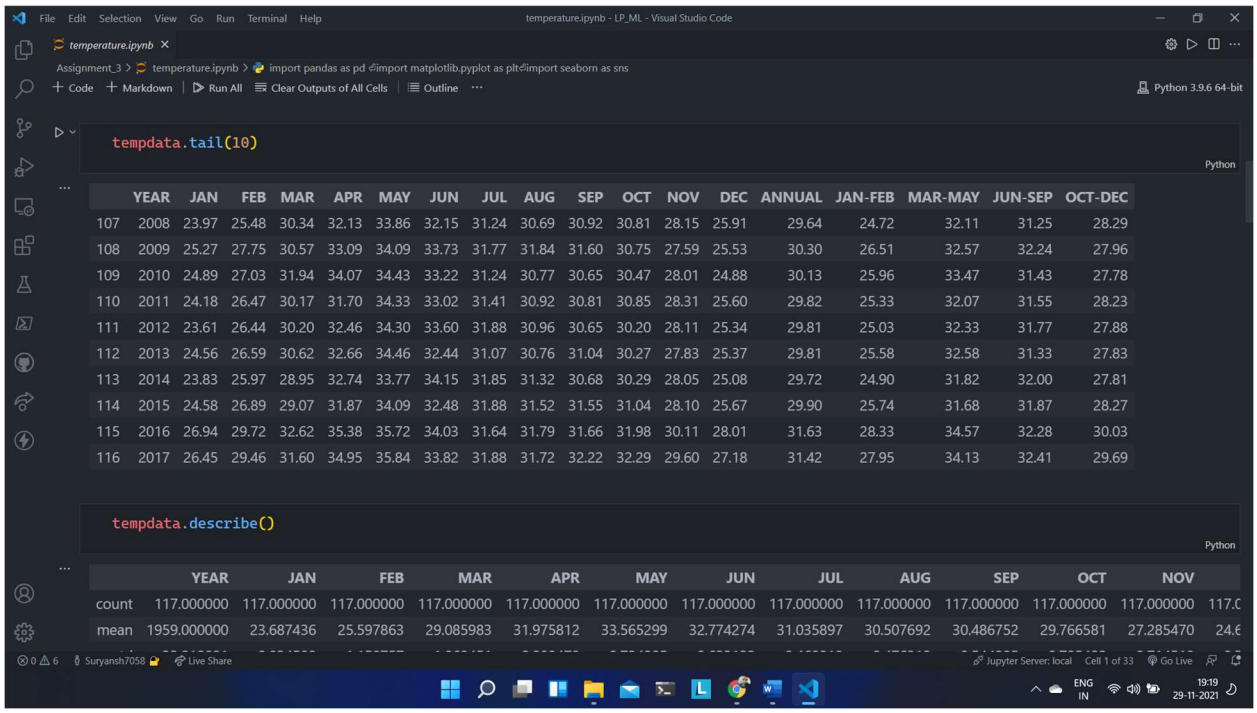
First we must calculate the difference between each model prediction and the actual y values. We can easily calculate the square of each of these error values (error*error or error^2).

1	pred-y	y	error	Squared error
2	1.2	1	0.2	0.04
3	2	3	-1	1
4	3.6	3	0.6	0.36
5	2.8	2	0.8	0.64
6	4.4	5	-0.6	0.36

The sum of these errors is 2.4 units, dividing by n and taking the square root gives us: RMSE = 0.692 Or, each prediction is on average wrong by about 0.692 units

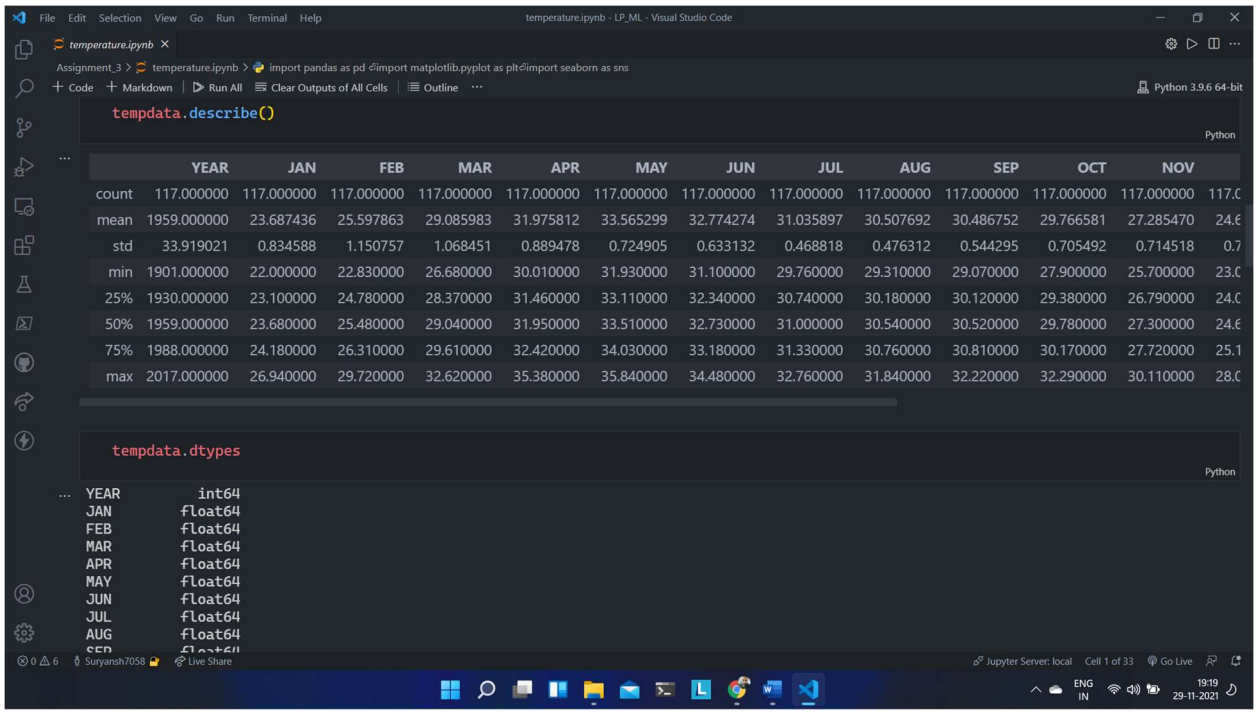


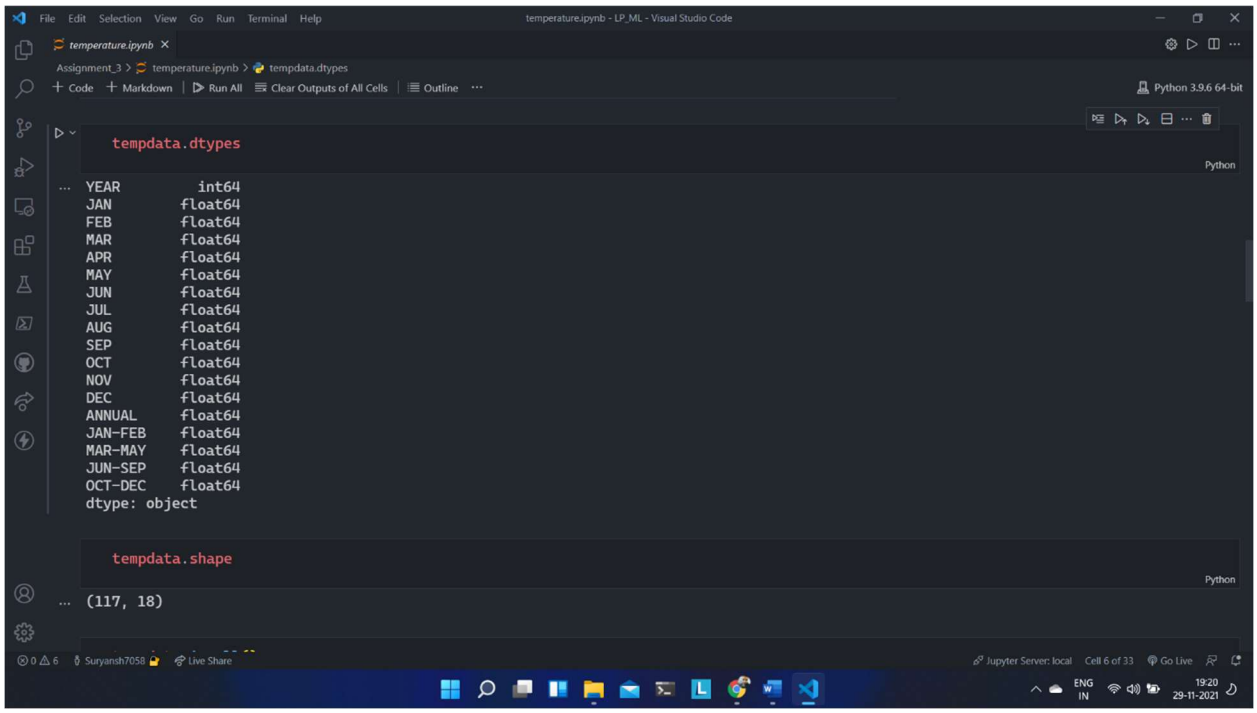
	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49	28.96	23.27	31.46	31.27	27.25
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04	29.22	25.75	31.76	31.09	26.49
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65	28.47	24.24	30.71	30.92	26.26
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63	28.49	23.62	30.95	30.66	26.40
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.82	28.30	22.25	30.00	31.33	26.57
5	1906	22.28	23.69	27.31	31.93	34.11	32.19	31.01	30.30	29.92	29.55	27.60	24.72	28.73	23.03	31.11	30.86	27.29
6	1907	24.46	24.01	27.04	31.79	32.68	31.92	31.05	29.58	30.67	29.87	27.78	24.44	28.65	24.23	29.92	30.80	27.36
7	1908	23.57	25.26	28.86	32.42	33.02	33.12	30.61	29.55	29.59	29.35	26.88	23.73	28.83	24.42	31.43	30.72	26.64
8	1909	22.67	24.36	29.22	30.79	33.06	31.70	29.81	29.81	30.06	29.25	27.69	23.69	28.38	23.52	31.02	30.33	26.88
9	1910	23.24	25.16	28.48	31.42	33.51	31.84	30.42	29.86	29.82	28.91	26.32	23.37	28.53	24.20	31.14	30.48	26.20

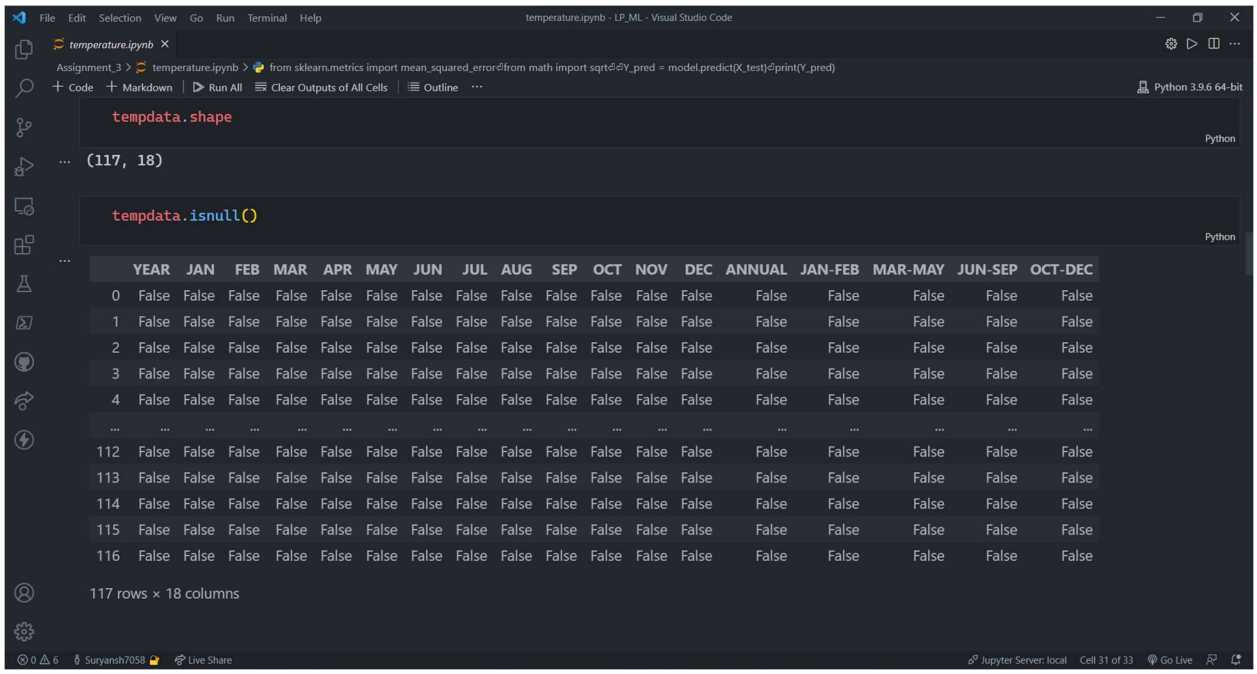


	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
107	2008	23.97	25.48	30.34	32.13	33.86	32.15	31.24	30.69	30.92	30.81	28.15	25.91	29.64	24.72	32.11	31.25	28.29
108	2009	25.27	27.75	30.57	33.09	34.09	33.73	31.77	31.84	31.60	30.75	27.59	25.53	30.30	26.51	32.57	32.24	27.96
109	2010	24.89	27.03	31.94	34.07	34.43	33.22	31.24	30.77	30.65	30.47	28.01	24.88	30.13	25.96	33.47	31.43	27.78
110	2011	24.18	26.47	30.17	31.70	34.33	33.02	31.41	30.92	30.81	30.85	28.31	25.60	29.82	25.33	32.07	31.55	28.23
111	2012	23.61	26.44	30.20	32.46	34.30	33.60	31.88	30.96	30.65	30.20	28.11	25.34	29.81	25.03	32.33	31.77	27.88
112	2013	24.56	26.59	30.62	32.66	34.46	32.44	31.07	30.76	31.04	30.27	27.83	25.37	29.81	25.58	32.58	31.33	27.83
113	2014	23.83	25.97	28.95	32.74	33.77	34.15	31.85	31.32	30.68	30.29	28.05	25.08	29.72	24.90	31.82	32.00	27.81
114	2015	24.58	26.89	29.07	31.87	34.09	32.48	31.88	31.52	31.55	31.04	28.10	25.67	29.90	25.74	31.68	31.87	28.27
115	2016	26.94	29.72	32.62	35.38	35.72	34.03	31.64	31.79	31.66	31.98	30.11	28.01	31.63	28.33	34.57	32.28	30.03
116	2017	26.45	29.46	31.60	34.95	35.84	33.82	31.88	31.72	32.22	32.29	29.60	27.18	31.42	27.95	34.13	32.41	29.69

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	
count	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.0
mean	1959.000000	23.687436	25.597863	29.085983	31.975812	33.565299	32.774274	31.035897	30.507692	30.486752	29.766581	27.285470	24.6


temperature.ipynb - LP_ML - Visual Studio Code
tempdata.describe()
Python
tempdata.dtypes
YEAR int64
JAN float64
FEB float64
MAR float64
APR float64
MAY float64
JUN float64
JUL float64
AUG float64


temperature.ipynb - LP_ML - Visual Studio Code
tempdata.dtypes
YEAR int64
JAN float64
FEB float64
MAR float64
APR float64
MAY float64
JUN float64
JUL float64
AUG float64
SEP float64
OCT float64
NOV float64
DEC float64
ANNUAL float64
JAN-FEB float64
MAR-MAY float64
JUN-SEP float64
OCT-DEC float64
dtype: object
tempdata.shape
(117, 18)



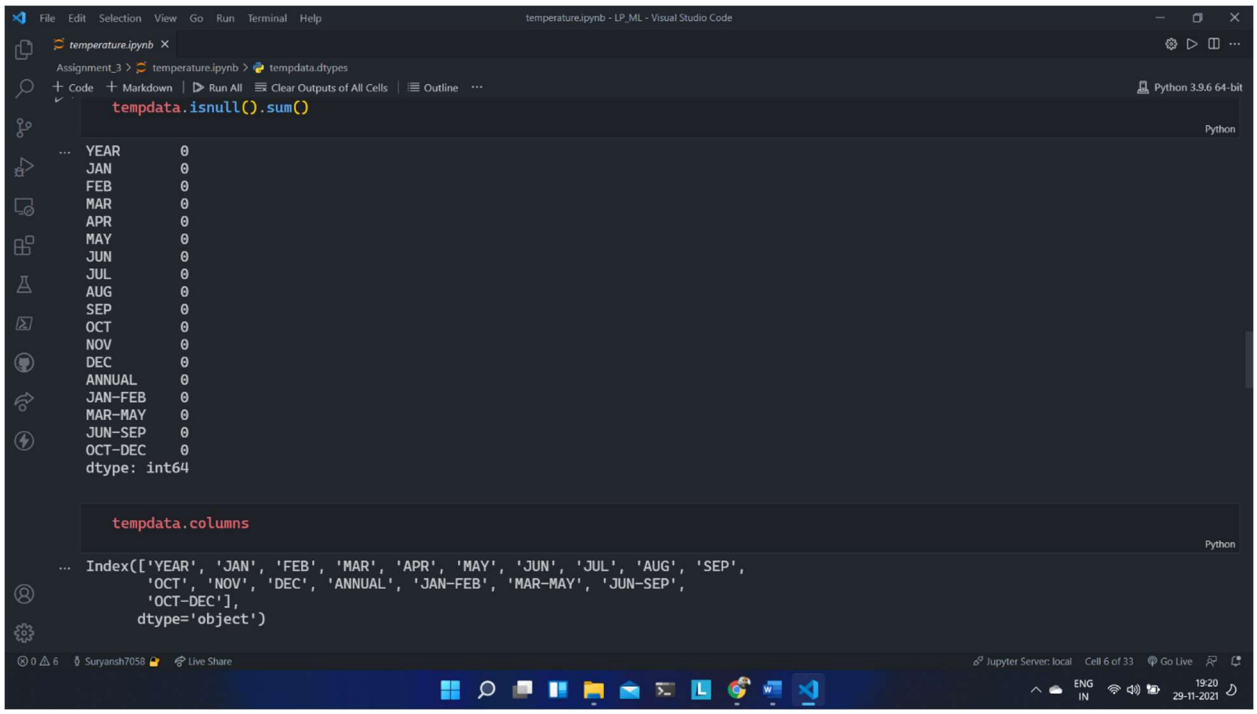
```
tempdata.shape
```

```
(117, 18)
```

```
tempdata.isnull()
```

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
112	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
113	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
114	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
115	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
116	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

117 rows × 18 columns

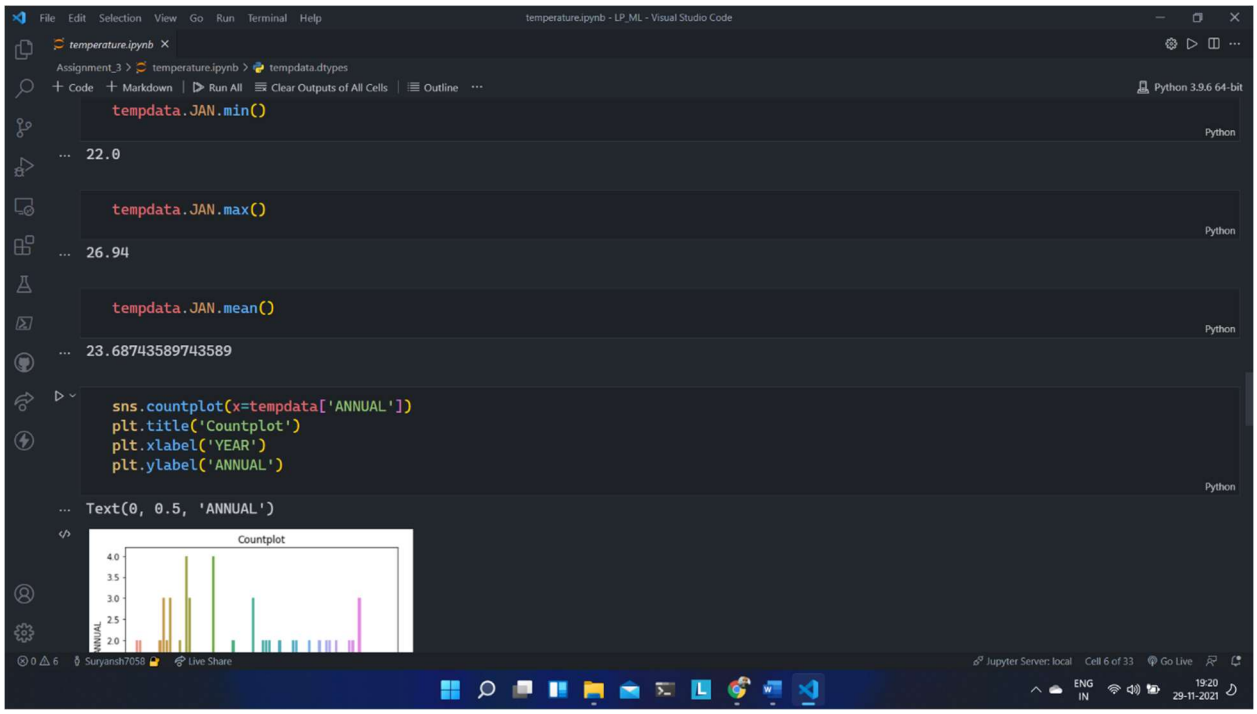


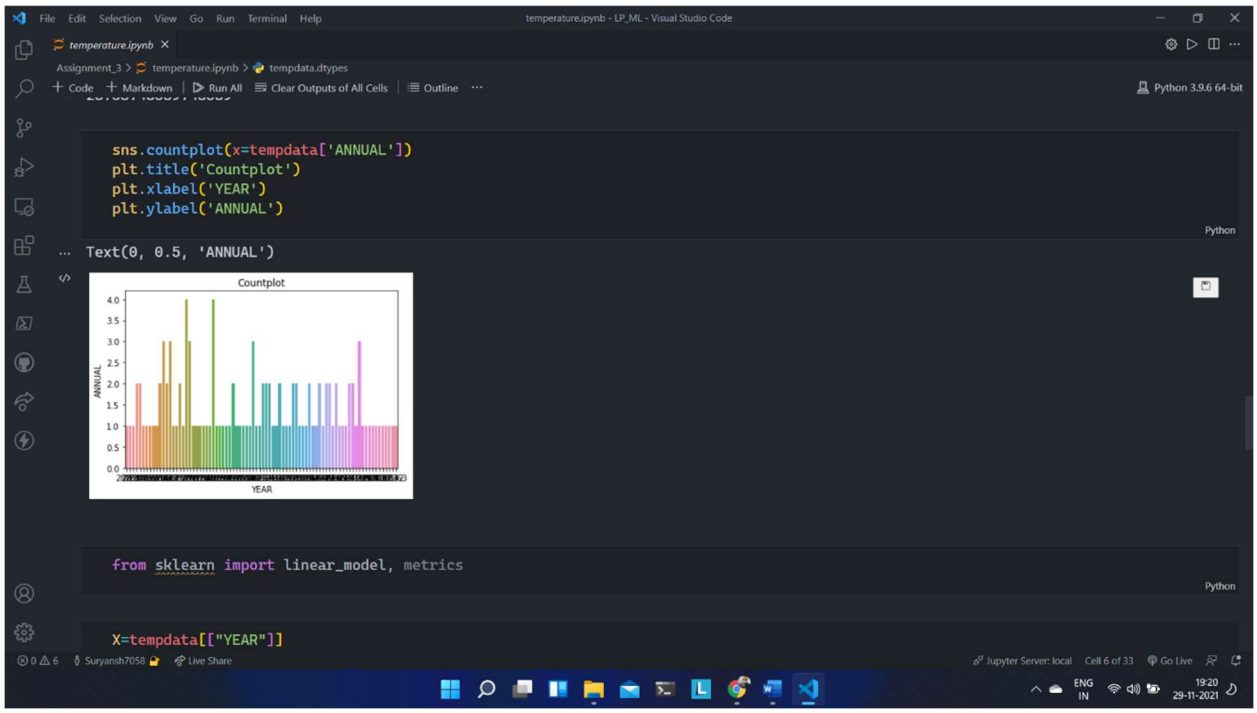
```
tempdata.isnull().sum()
```

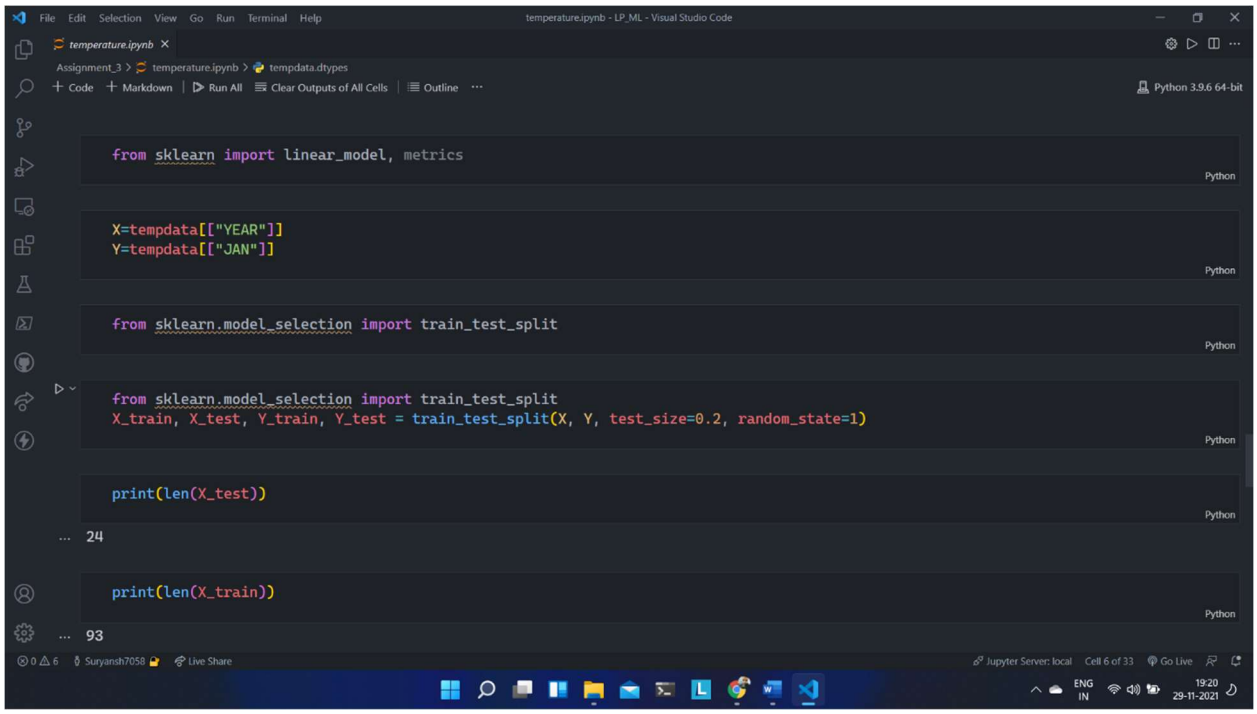
```
YEAR        0
JAN         0
FEB         0
MAR         0
APR         0
MAY         0
JUN         0
JUL         0
AUG         0
SEP         0
OCT         0
NOV         0
DEC         0
ANNUAL      0
JAN-FEB     0
MAR-MAY     0
JUN-SEP     0
OCT-DEC     0
dtype: int64
```

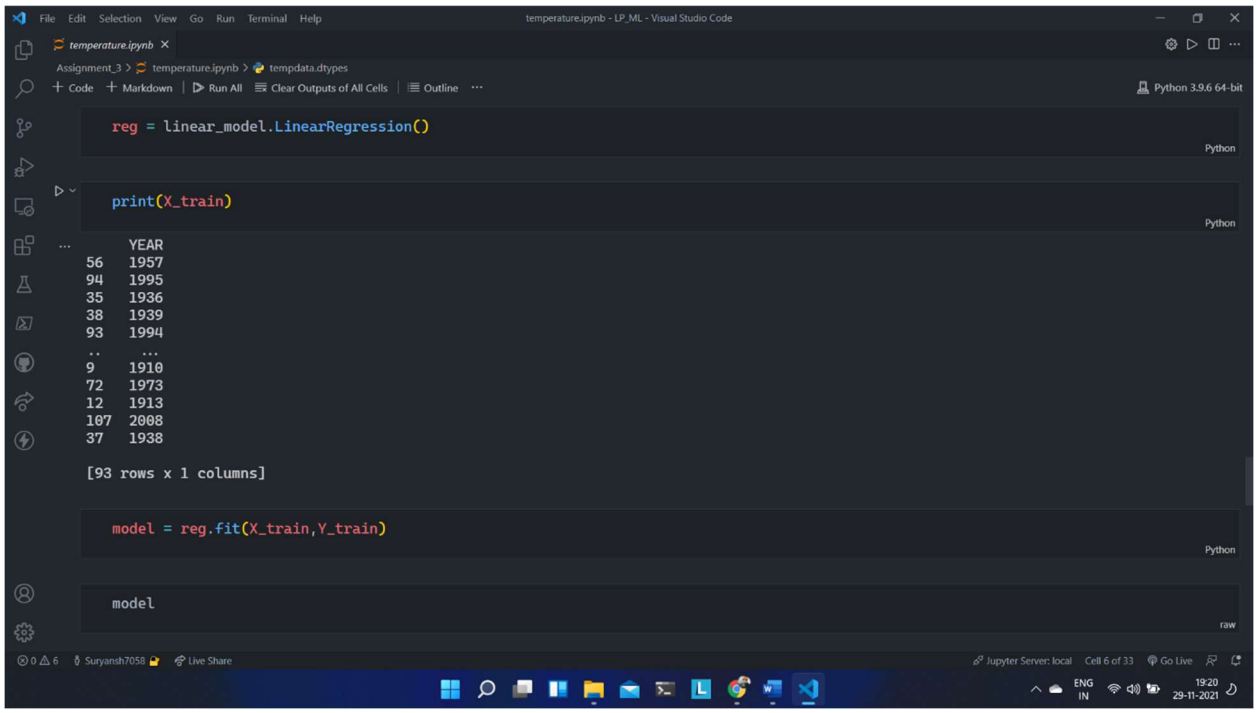
```
tempdata.columns
```

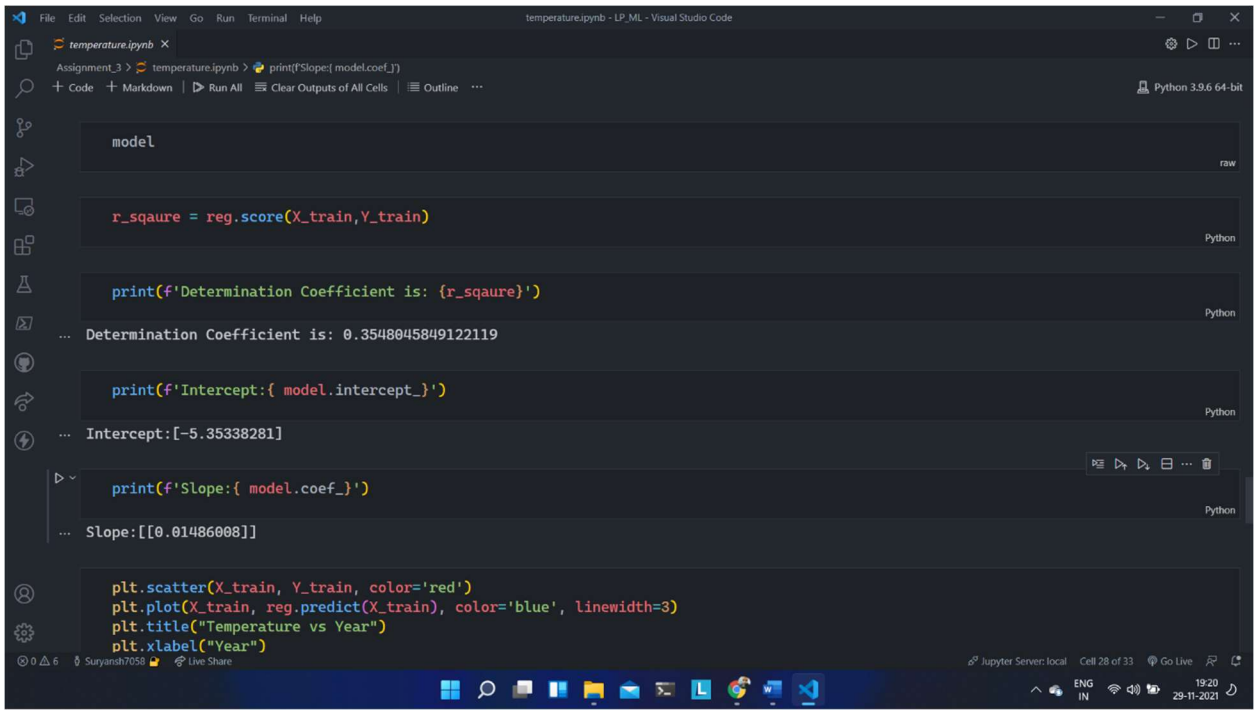
```
Index(['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC', 'ANNUAL', 'JAN-FEB', 'MAR-MAY', 'JUN-SEP',
       'OCT-DEC'],
      dtype='object')
```

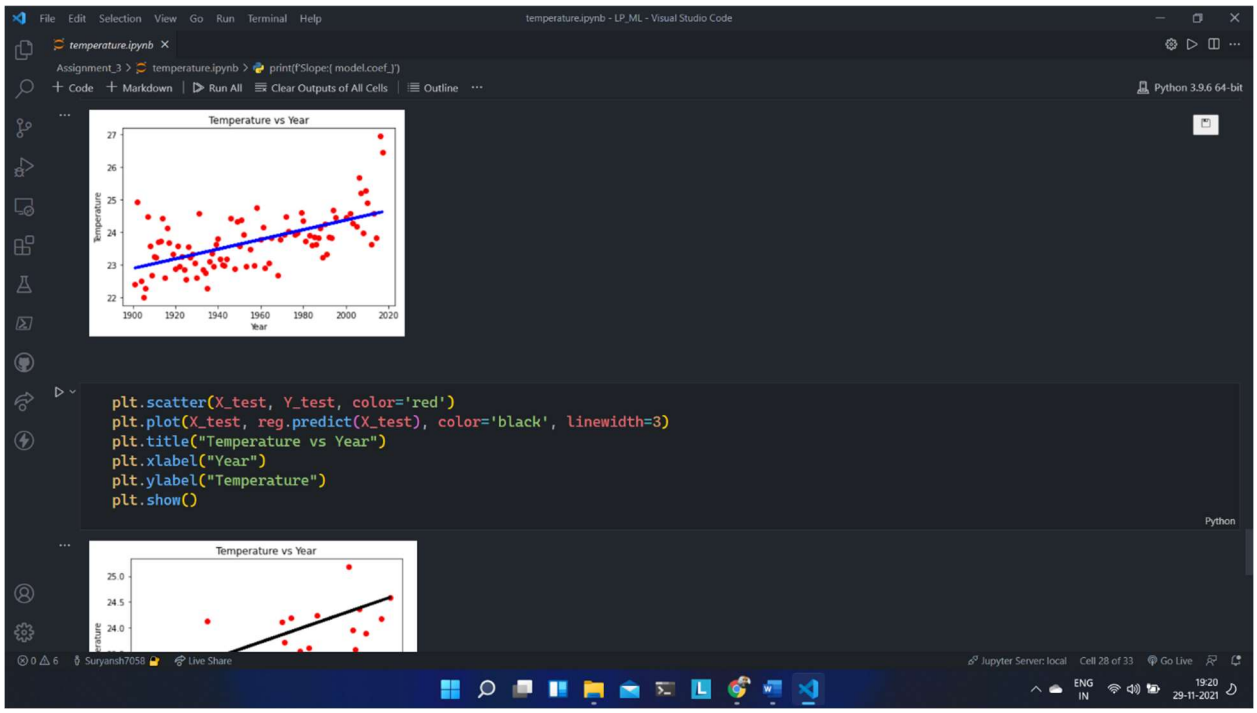

temperature.ipynb - LP_ML - Visual Studio Code
tempdata.JAN.min()
22.0
tempdata.JAN.max()
26.94
tempdata.JAN.mean()
23.68743589743589
sns.countplot(x=tempdata['ANNUAL'])
plt.title('Countplot')
plt.xlabel('YEAR')
plt.ylabel('ANNUAL')
Text(0, 0.5, 'ANNUAL')
Countplot
ANNUAL

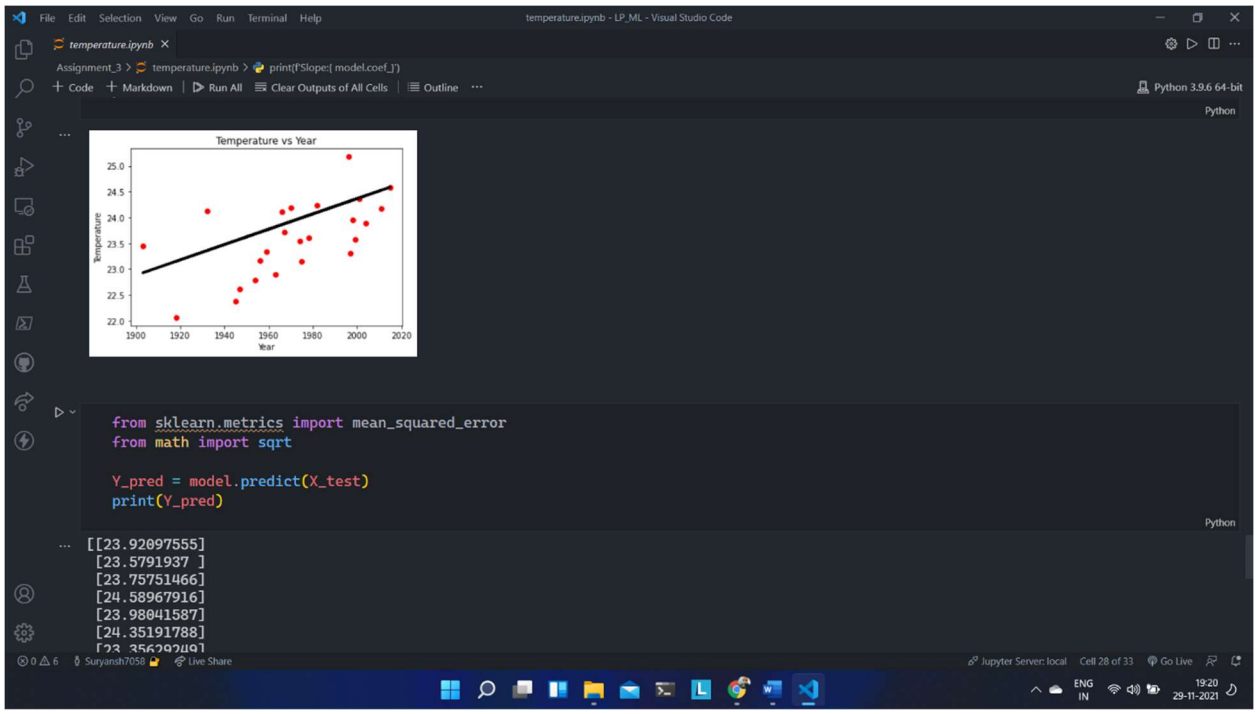

temperature.ipynb - LP_ML - Visual Studio Code
sns.countplot(x=tempdata['ANNUAL'])
plt.title('Countplot')
plt.xlabel('YEAR')
plt.ylabel('ANNUAL')
Text(0, 0.5, 'ANNUAL')
Countplot
ANNUAL
YEAR
from sklearn import linear_model, metrics
X=tempdata[["YEAR"]]

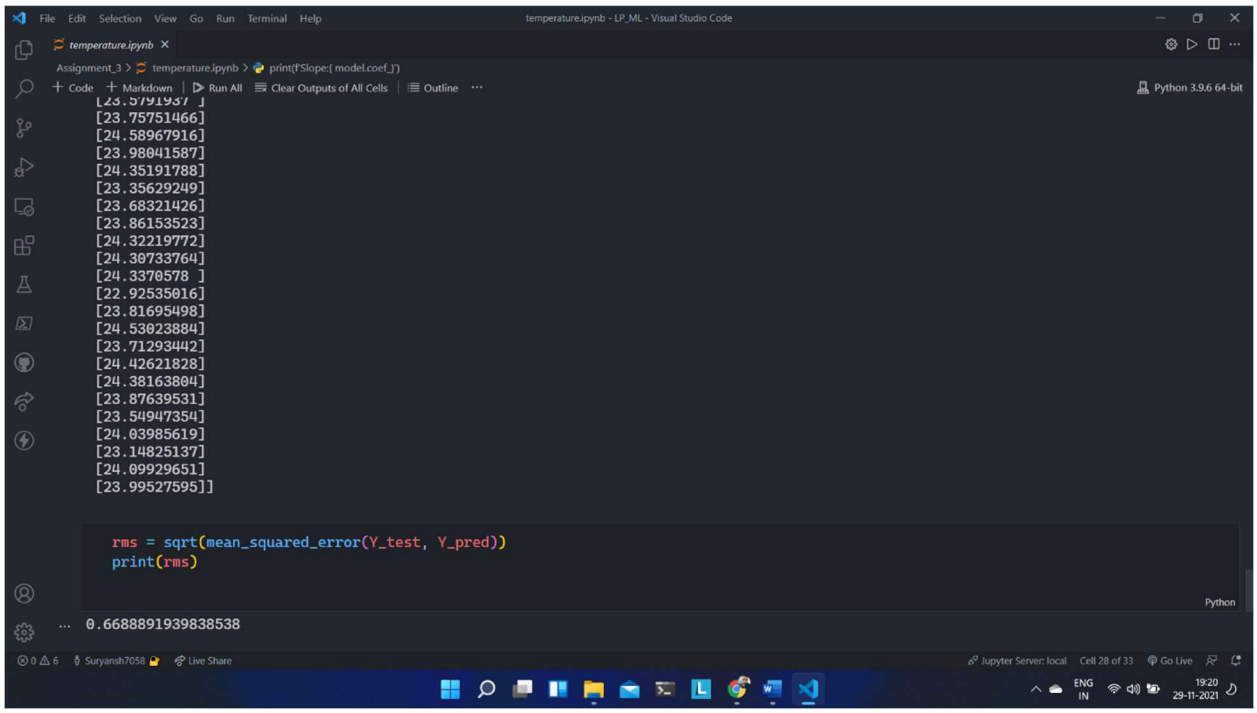

temperature.ipynb - LP_ML - Visual Studio Code
from sklearn import linear_model, metrics
X=tempdata[["YEAR"]]
Y=tempdata[["JAN"]]
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
print(len(X_test))
24
print(len(X_train))
93


temperature.ipynb - LP_ML - Visual Studio Code
reg = linear_model.LinearRegression()
print(X_train)
YEAR
56 1957
94 1995
35 1936
38 1939
93 1994
.. ...
9 1910
72 1973
12 1913
107 2008
37 1938
[93 rows x 1 columns]
model = reg.fit(X_train,Y_train)
model


temperature.ipynb - LP_ML - Visual Studio Code
model
r_sqaure = reg.score(X_train,Y_train)
print(f'Determination Coefficient is: {r_sqaure}')
Determination Coefficient is: 0.3548045849122119
print(f'Intercept:{ model.intercept_}')
Intercept:[-5.35338281]
print(f'Slope:{ model.coef_}')
Slope:[[0.01486008]]
plt.scatter(X_train, Y_train, color='red')
plt.plot(X_train, reg.predict(X_train), color='blue', linewidth=3)
plt.title("Temperature vs Year")
plt.xlabel("Year")


temperature.ipynb - LP_ML - Visual Studio Code
Temperature vs Year
Temperature
Year
plt.scatter(X_test, Y_test, color='red')
plt.plot(X_test, reg.predict(X_test), color='black', linewidth=3)
plt.title("Temperature vs Year")
plt.xlabel("Year")
plt.ylabel("Temperature")
plt.show()
Temperature vs Year





Conclusion:

Thus we have implemented linear regression and evaluated the performance.

Assignment No . 3

AIM: To implement K-Means Clustering and visualize the clusters for customer segmentation

.

THEORY:

Unsupervised learning:

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning.

Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. By contrast with SUPERVISED LEARNING or REINFORCEMENT LEARNING, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output.

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

K-Means Clustering

K-means (MacQueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids shoud be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

where $\left\| x_i^{(j)} - c_j \right\|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j, is an indicator of the distance of the n data points from their respective cluster centres.

The algorithm is composed of the following steps:

1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Advantages

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other.

Disadvantages

1) The learning algorithm requires apriori specification of the number of cluster centers.

2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.

3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get

different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).

4) Euclidean distance measures can unequally weight underlying factors.

5) The learning algorithm provides the local optima of the squared error function.

6) Randomly choosing of the cluster center cannot lead us to the fruitful result.

7) Applicable only when mean is defined i.e. fails for categorical data.

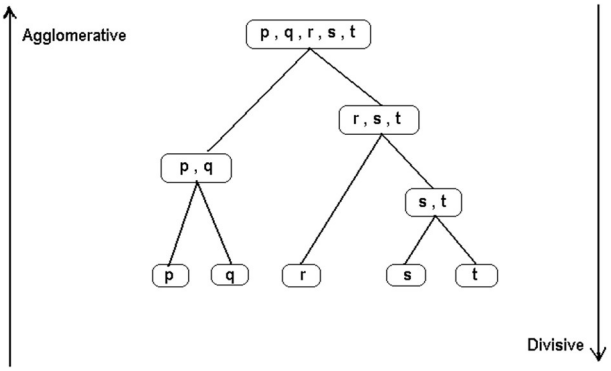
8) Unable to handle noisy data and outliers.

Hierarchical Clustering

Cluster Analysis (data segmentation) has a variety of goals that relate to grouping or segmenting a collection of objects (i.e., observations, individuals, cases, or data rows) into subsets or clusters, such that those within each cluster are more closely related to one another than objects assigned to different clusters. Central to all of the goals of cluster analysis is the notion of degree of similarity

(or dissimilarity) between the individual objects being clustered. There are two major methods of clustering: hierarchical clustering and k-means clustering. For information on k-means clustering, refer to the k-Means Clustering section.

In hierarchical clustering, the data is not partitioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a single cluster containing all objects to n clusters that each contain a single object. Hierarchical Clustering is subdivided into agglomerative methods, which proceed by a series of fusions of the n objects into groups, and divisive methods, which separate n objects successively into finer groupings. Agglomerative techniques are more commonly used, and this is the method implemented in XLMiner. Hierarchical clustering may be represented by a two-dimensional diagram known as a dendrogram, which illustrates the fusions or divisions made at each successive stage of analysis. Following is an example of a dendrogram.



Agglomerative methods

An agglomerative hierarchical clustering procedure produces a series of partitions of the data, Pn, Pn-1, ... , P1. The first Pn consists of n single object clusters, the last P1, consists of single group containing all n cases.

At each particular stage, the method joins together the two clusters that are closest together (most similar). (At the first stage, this amounts to joining together the two objects that are closest together, since at the initial stage each cluster has only one object.)

Divisive method

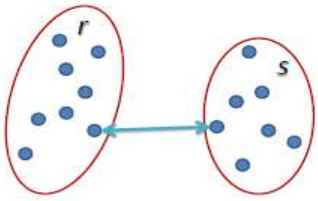
In this method we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. Finally, we proceed recursively on each cluster until there is one cluster for each observation.

Given:
A set X of objects $\{x_1,\ldots,x_n\}$
A distance function $dist(c_1,c_2)$
for $i = 1$ to n
    $c_i = \{x_i\}$
end for
$C = \{c_1,\ldots,c_n\}$
$l = n+1$
while C.size > 1 **do**
- (c_{min1},c_{min2}) = minimum $dist(c_i,c_j)$ for all c_i,c_j in C
- remove c_{min1} and c_{min2} from C
- add $\{c_{min1},c_{min2}\}$ to C
- $l = l + 1$

end while

Single Linkage

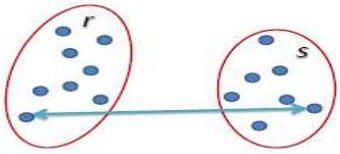
In single linkage hierarchical clustering, the distance between two clusters is defined as the shortest distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two closest points.



$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

Complete Linkage

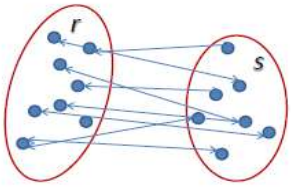
In complete linkage hierarchical clustering, the distance between two clusters is defined as the longest distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two furthest points.



$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

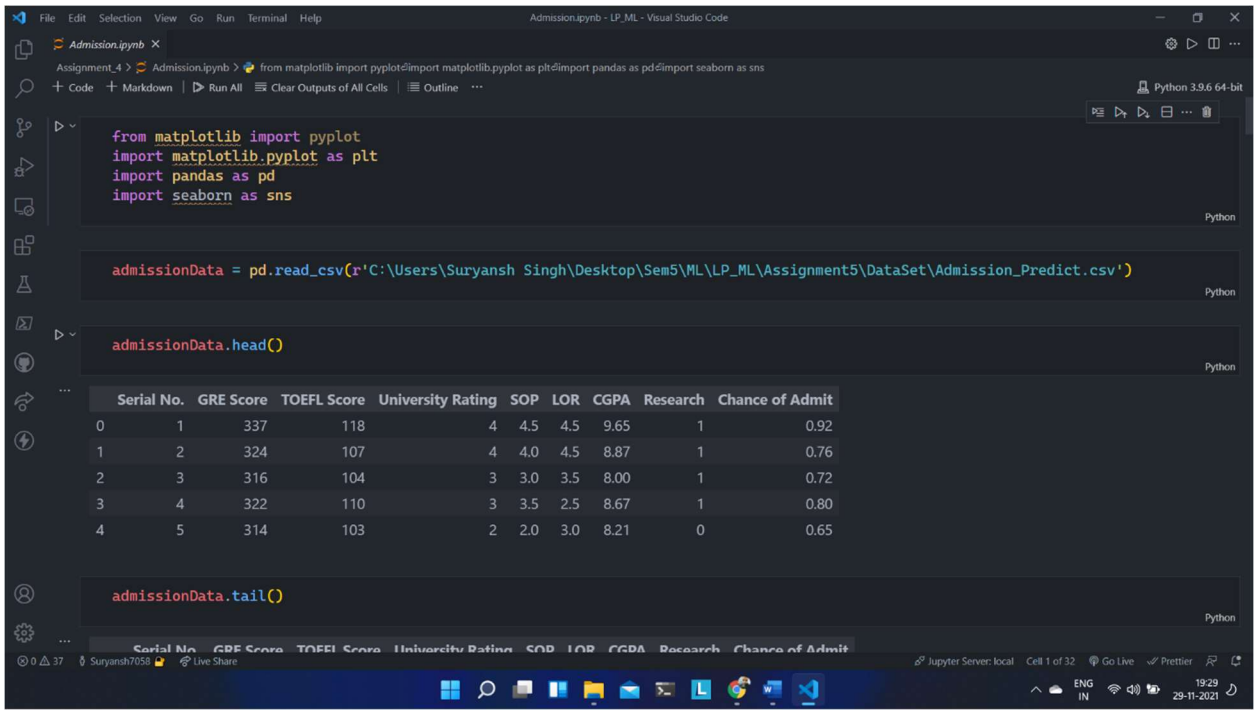
Average Linkage

average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters "r" and "s" to the left is equal to the average length each arrow between connecting the points of one cluster to the other.

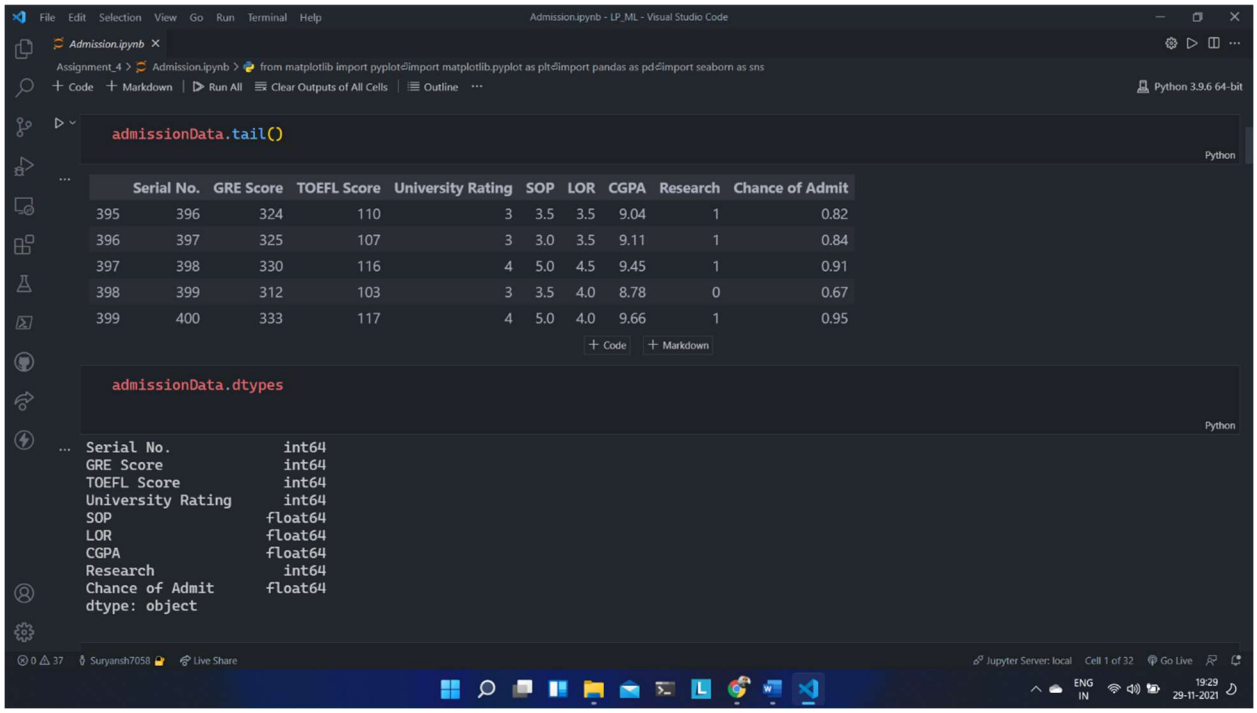


$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

if your data is small then go for Hierarchical Clustering and if it is large then go for K-Means Clustering.

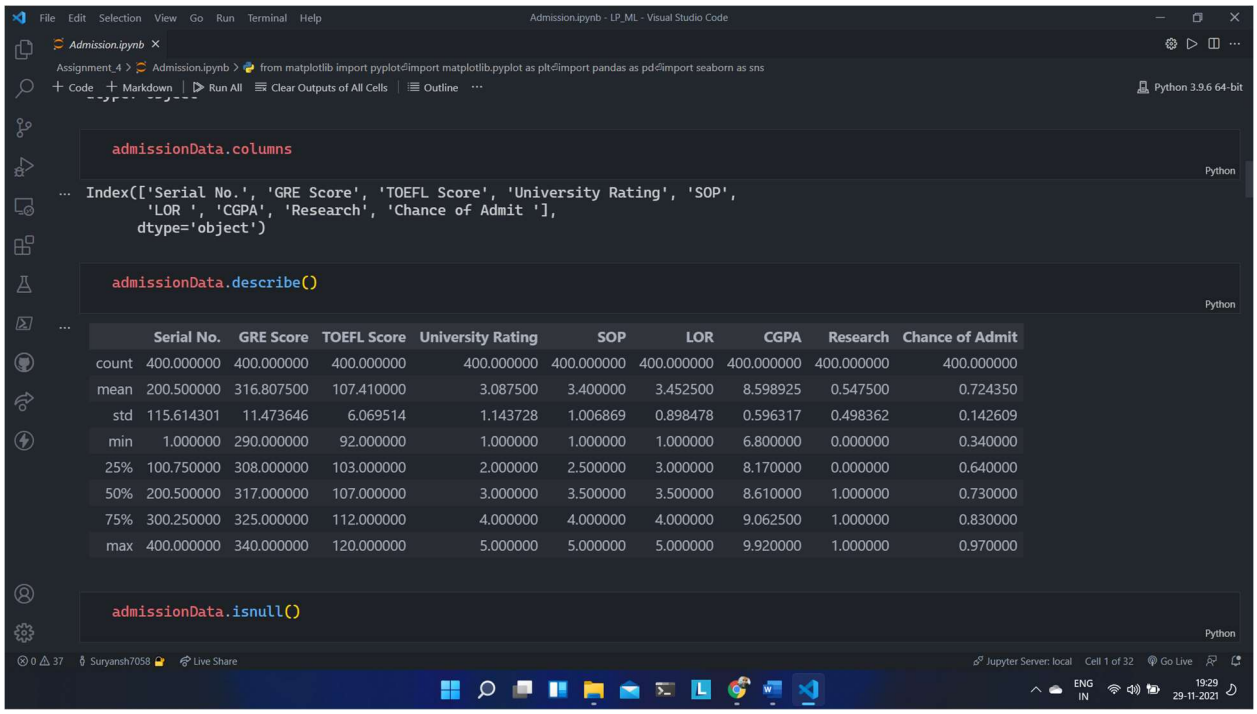


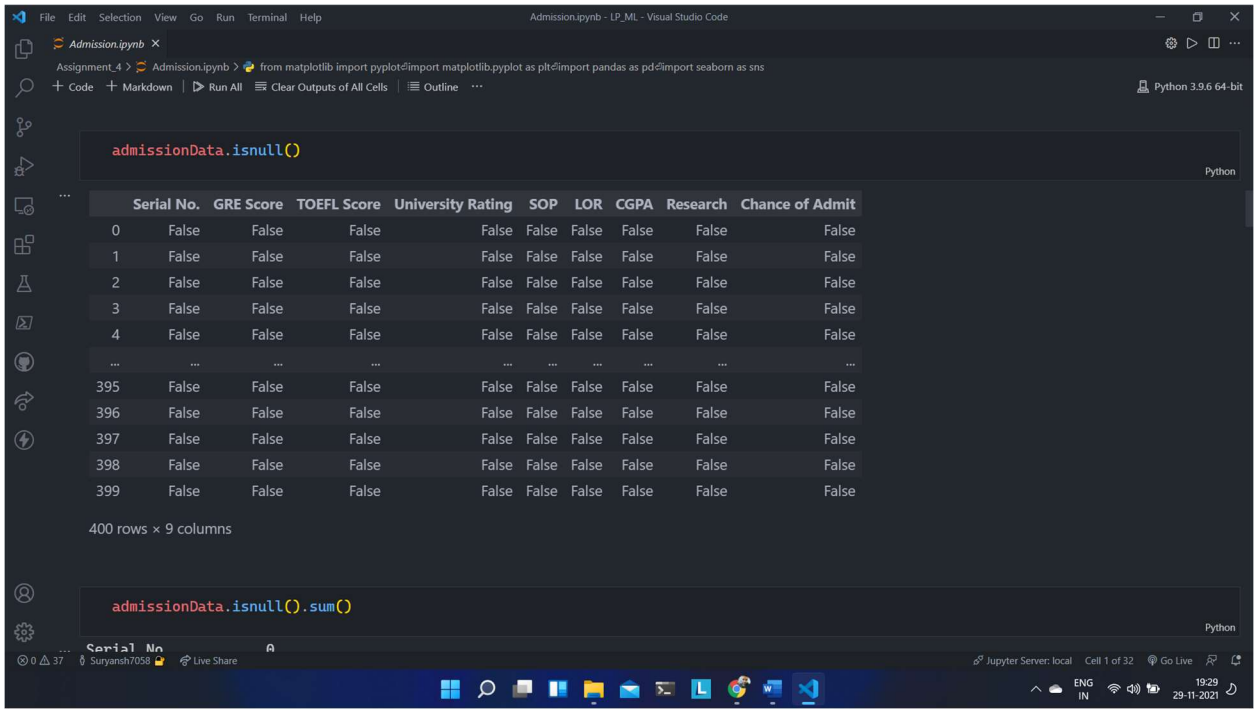
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65



	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
395	396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	333	117	4	5.0	4.0	9.66	1	0.95

```
Serial No.            int64
GRE Score             int64
TOEFL Score           int64
University Rating     int64
SOP                 float64
LOR                 float64
CGPA                float64
Research              int64
Chance of Admit     float64
dtype: object
```


Admission.ipynb - LP_ML - Visual Studio Code
admissionData.columns
Python
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
dtype='object')
admissionData.describe()
Python
Serial No. GRE Score TOEFL Score University Rating SOP LOR CGPA Research Chance of Admit
count 400.000000 400.000000 400.000000 400.000000 400.000000 400.000000 400.000000 400.000000 400.000000
mean 200.500000 316.807500 107.410000 3.087500 3.400000 3.452500 8.598925 0.547500 0.724350
std 115.614301 11.473646 6.069514 1.143728 1.006869 0.898478 0.596317 0.498362 0.142609
min 1.000000 290.000000 92.000000 1.000000 1.000000 1.000000 6.800000 0.000000 0.340000
25% 100.750000 308.000000 103.000000 2.000000 2.500000 3.000000 8.170000 0.000000 0.640000
50% 200.500000 317.000000 107.000000 3.000000 3.500000 3.500000 8.610000 1.000000 0.730000
75% 300.250000 325.000000 112.000000 4.000000 4.000000 4.000000 9.062500 1.000000 0.830000
max 400.000000 340.000000 120.000000 5.000000 5.000000 5.000000 9.920000 1.000000 0.970000
admissionData.isnull()
Python


Admission.ipynb - LP_ML - Visual Studio Code
admissionData.isnull()
Python
Serial No. GRE Score TOEFL Score University Rating SOP LOR CGPA Research Chance of Admit
0 False False False False False False False False False
1 False False False False False False False False False
2 False False False False False False False False False
3 False False False False False False False False False
4 False False False False False False False False False
395 False False False False False False False False False
396 False False False False False False False False False
397 False False False False False False False False False
398 False False False False False False False False False
399 False False False False False False False False False
400 rows × 9 columns
admissionData.isnull().sum()
Python

```
admissionData.isnull().sum()
```

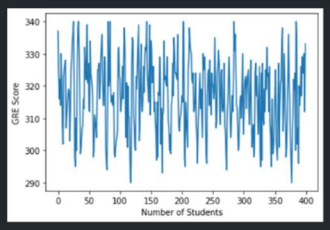
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

```
admissionData = admissionData.drop(['Serial No.'], axis=1)
```

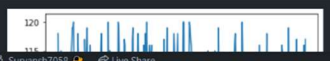
```
admissionData.columns
```

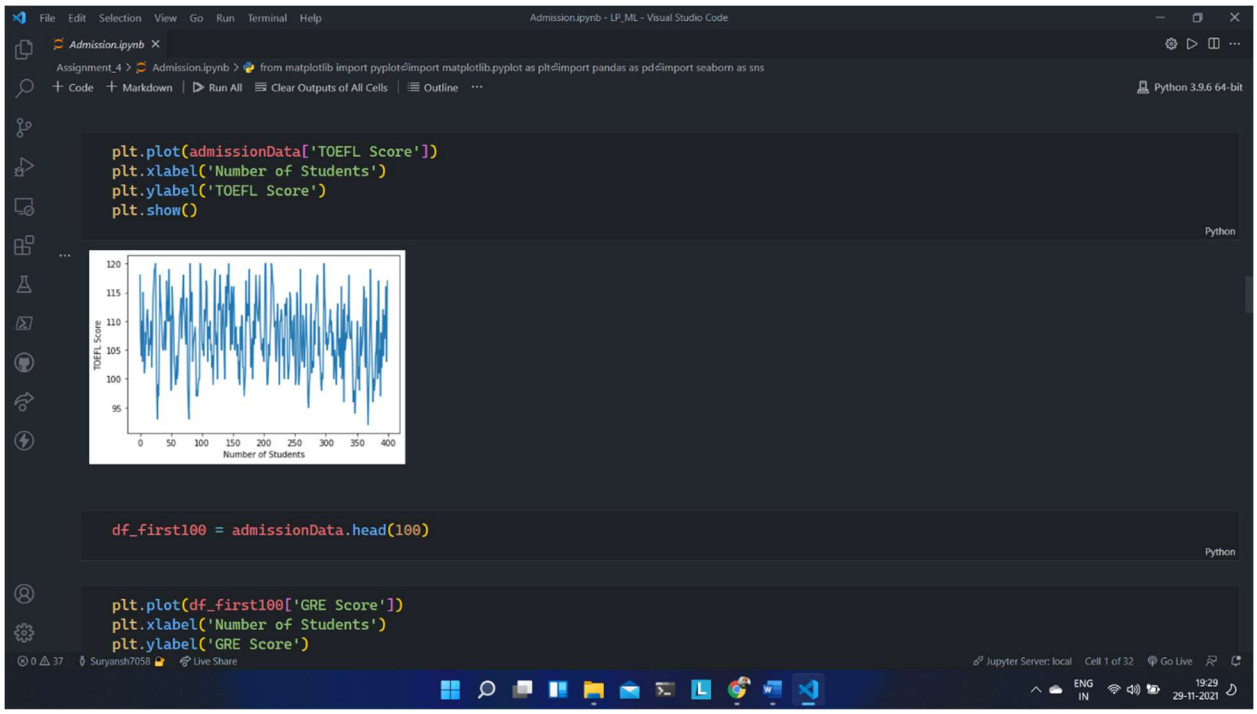
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

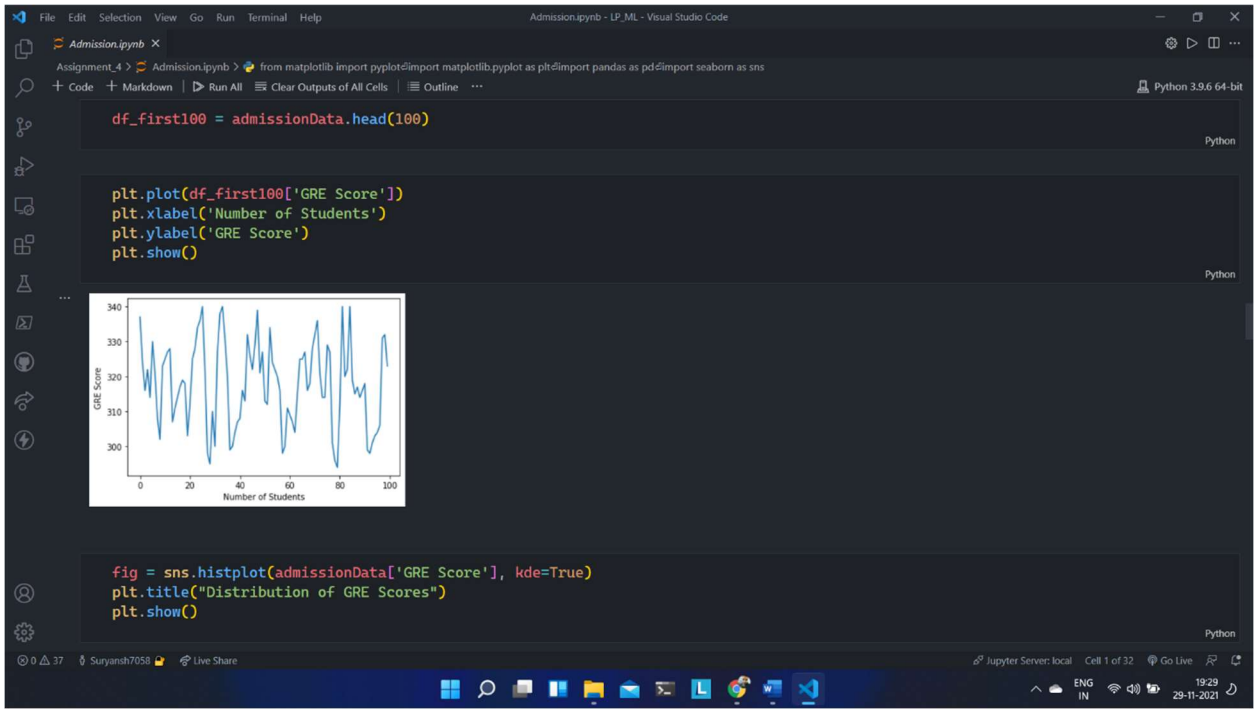
```
plt.plot(admissionData['GRE Score'])
plt.xlabel('Number of Students')
plt.ylabel('GRE Score')
plt.show()
```

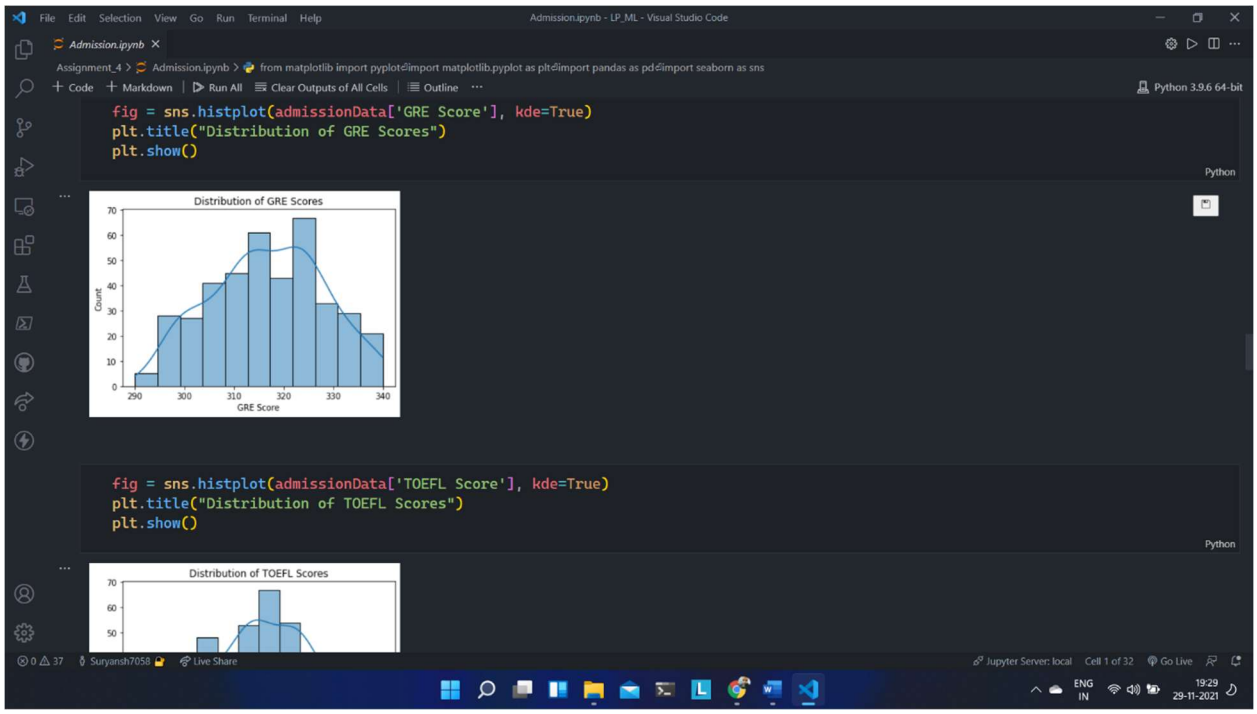


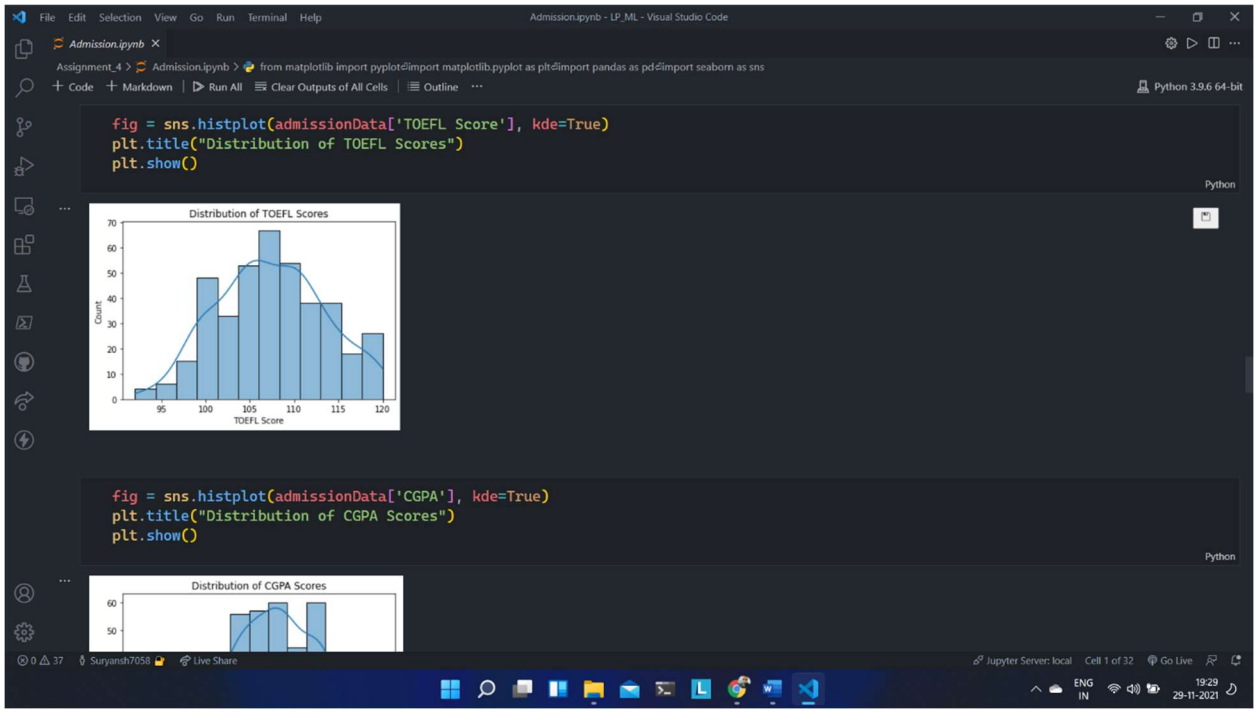

```
plt.plot(admissionData['TOEFL Score'])
plt.xlabel('Number of Students')
plt.ylabel('TOEFL Score')
plt.show()
```

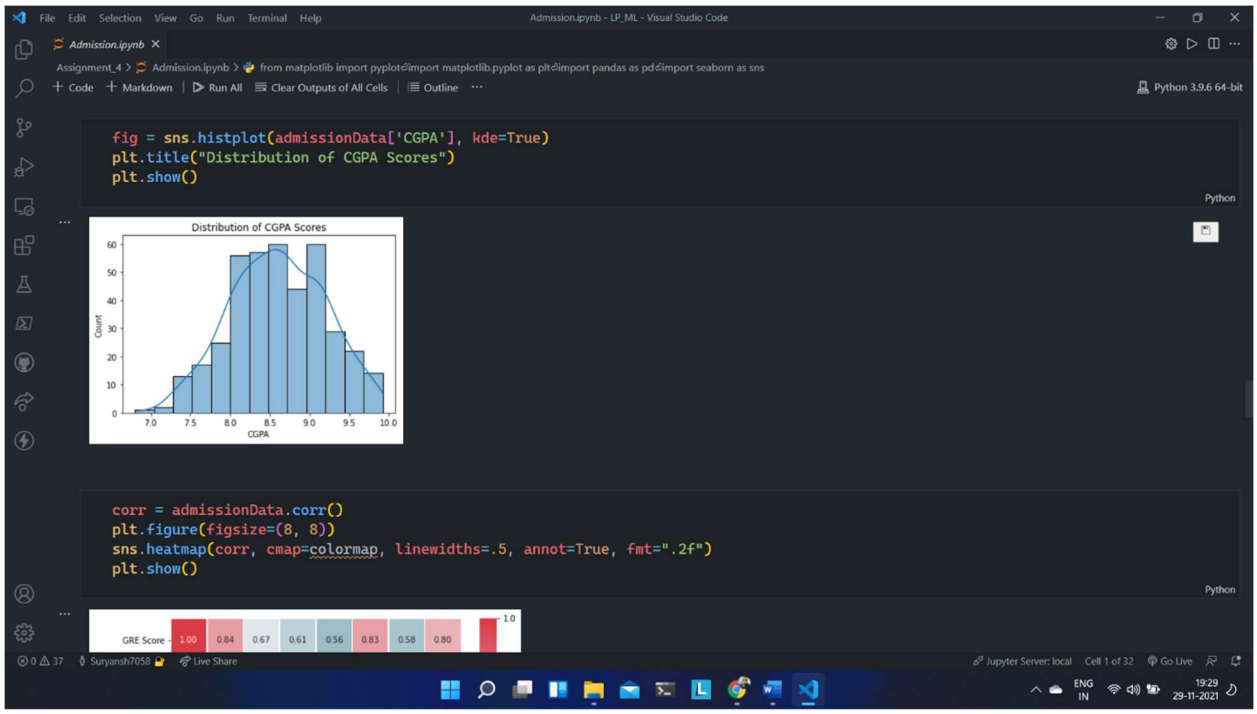


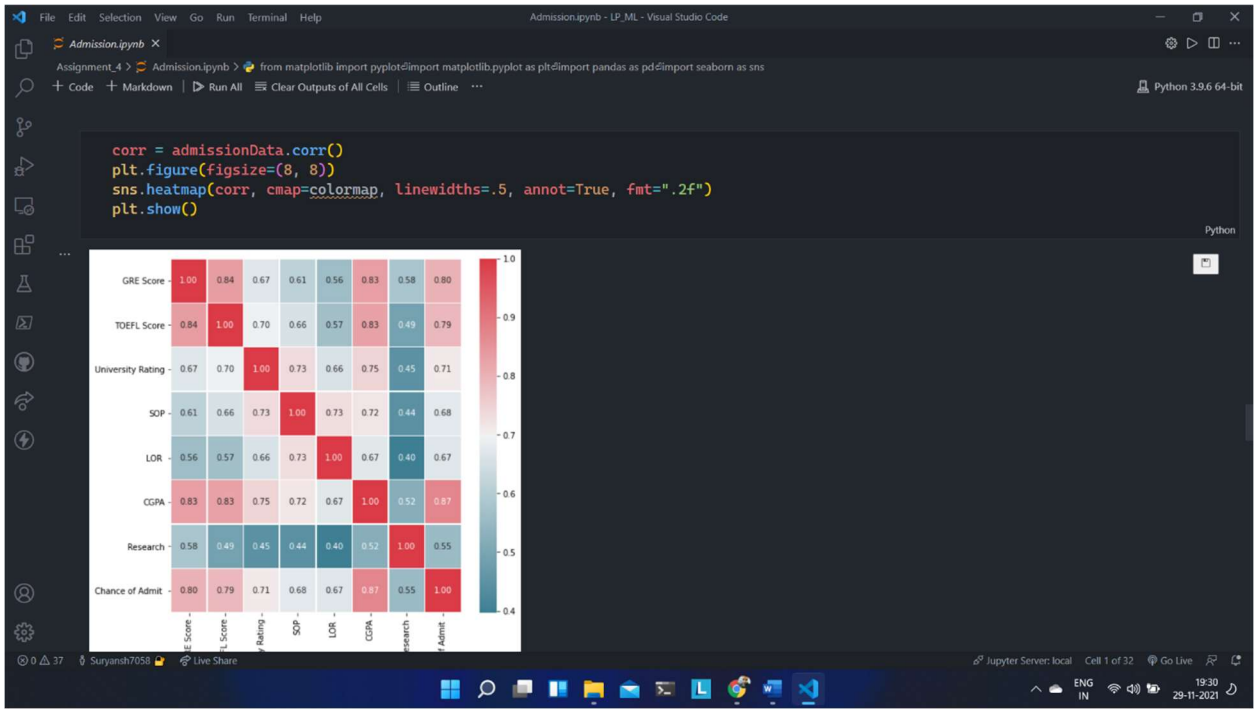

Admission.ipynb - LP_ML - Visual Studio Code
plt.plot(admissionData['TOEFL Score'])
plt.xlabel('Number of Students')
plt.ylabel('TOEFL Score')
plt.show()
TOEFL Score
Number of Students
df_first100 = admissionData.head(100)
plt.plot(df_first100['GRE Score'])
plt.xlabel('Number of Students')
plt.ylabel('GRE Score')


Admission.ipynb - LP_ML - Visual Studio Code
df_first100 = admissionData.head(100)
plt.plot(df_first100['GRE Score'])
plt.xlabel('Number of Students')
plt.ylabel('GRE Score')
plt.show()
GRE Score
Number of Students
fig = sns.histplot(admissionData['GRE Score'], kde=True)
plt.title("Distribution of GRE Scores")
plt.show()

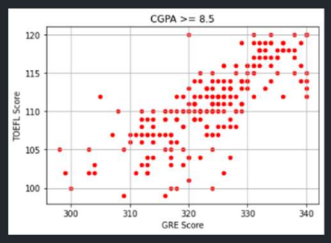

Admission.ipynb - LP_ML - Visual Studio Code
fig = sns.histplot(admissionData['GRE Score'], kde=True)
plt.title("Distribution of GRE Scores")
plt.show()
Distribution of GRE Scores
Count
GRE Score
fig = sns.histplot(admissionData['TOEFL Score'], kde=True)
plt.title("Distribution of TOEFL Scores")
plt.show()
Distribution of TOEFL Scores


Admission.ipynb - LP_ML - Visual Studio Code
fig = sns.histplot(admissionData['TOEFL Score'], kde=True)
plt.title("Distribution of TOEFL Scores")
plt.show()
Distribution of TOEFL Scores
Count
TOEFL Score
fig = sns.histplot(admissionData['CGPA'], kde=True)
plt.title("Distribution of CGPA Scores")
plt.show()
Distribution of CGPA Scores

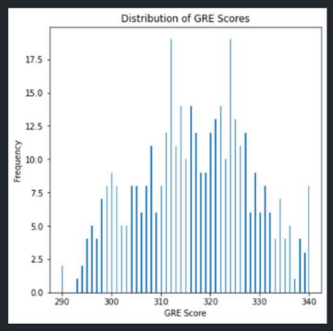

Admission.ipynb - LP_ML - Visual Studio Code
fig = sns.histplot(admissionData['CGPA'], kde=True)
plt.title("Distribution of CGPA Scores")
plt.show()
Distribution of CGPA Scores
Count
CGPA
corr = admissionData.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(corr, cmap=colormap, linewidths=.5, annot=True, fmt=".2f")
plt.show()


Admission.ipynb - LP_ML - Visual Studio Code
corr = admissionData.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(corr, cmap=colormap, linewidths=.5, annot=True, fmt=".2f")
plt.show()
GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
Chance of Admit

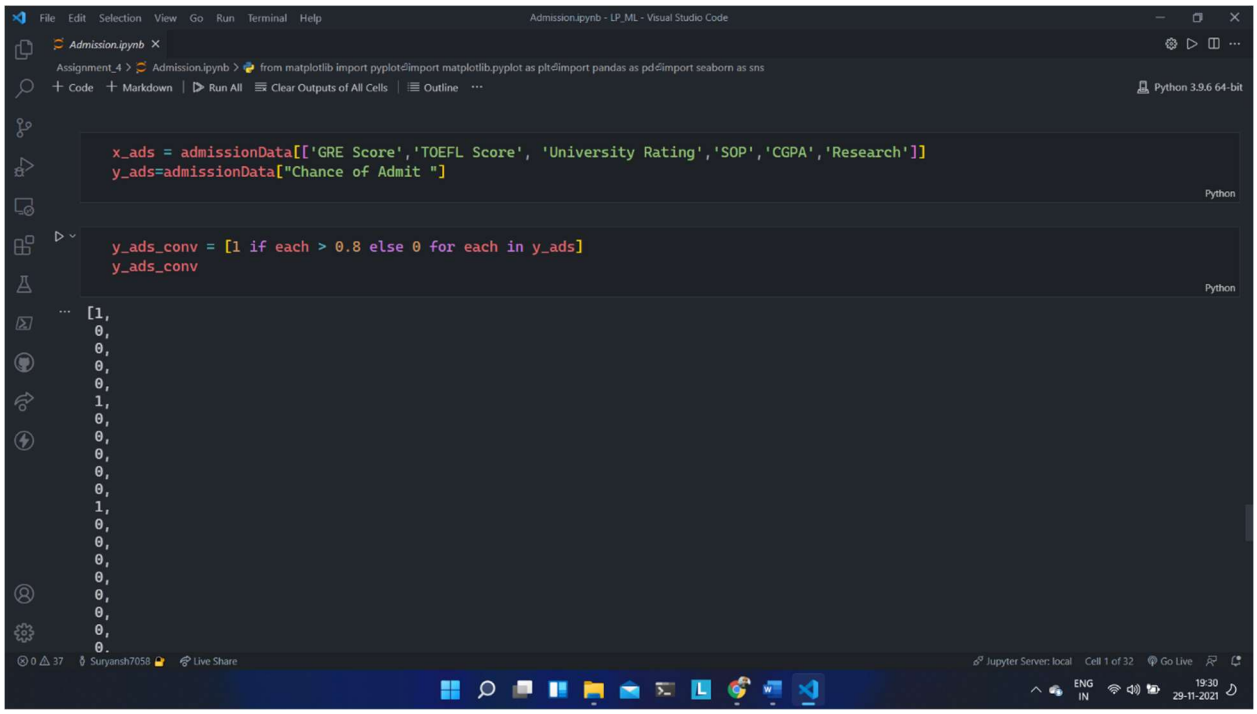
```
admissionData[admissionData.CGPA >= 8.5].plot(kind='scatter',x='GRE Score',y='TOEFL Score',color="red")
plt.xlabel("GRE Score")
plt.ylabel("TOEFL Score")
plt.title("CGPA >= 8.5")
plt.grid(True)
plt.show()
```

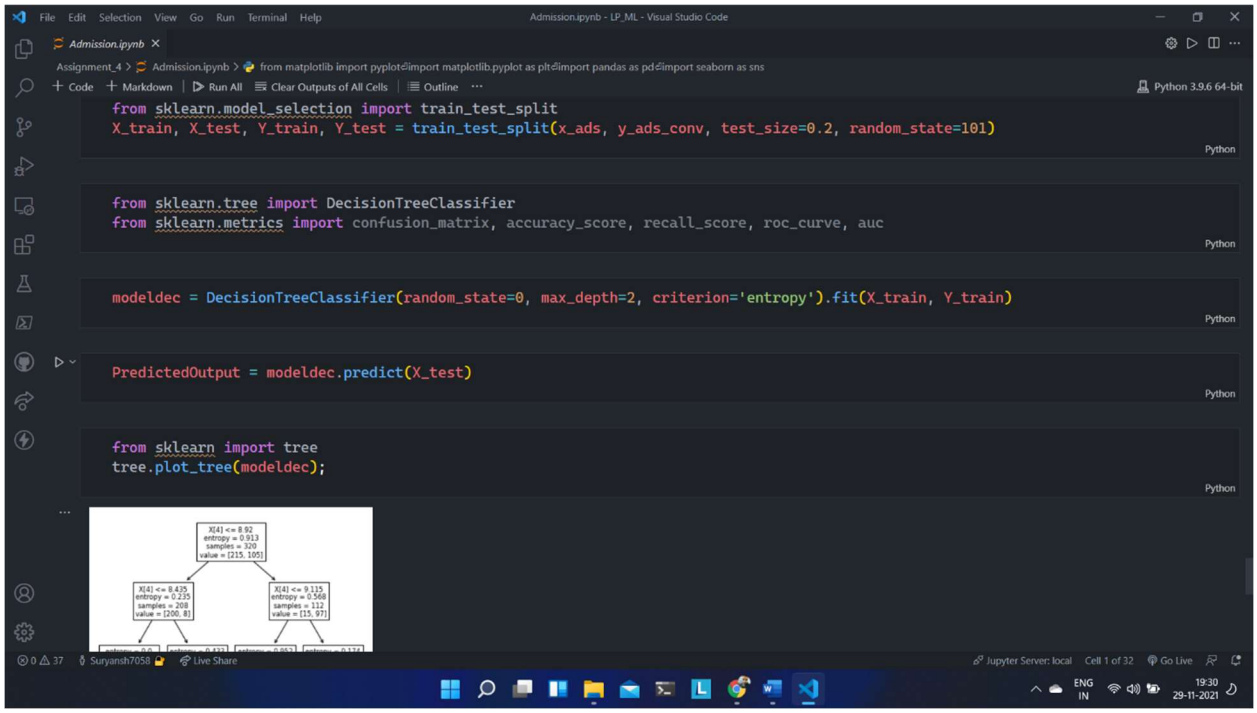


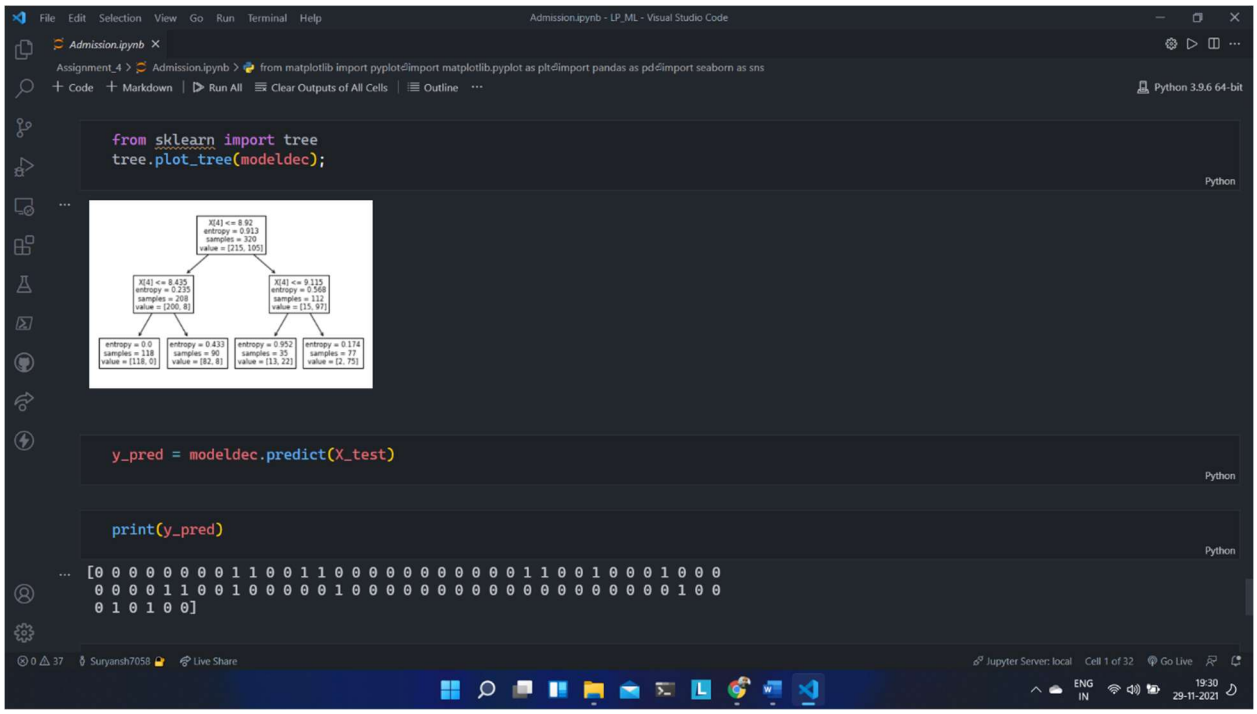

```
admissionData["GRE Score"].plot(kind='hist', bins=200,figsize=(6,6))
plt.title("Distribution of GRE Scores")
plt.xlabel("GRE Score")
plt.ylabel("Frequency")
plt.show()
```

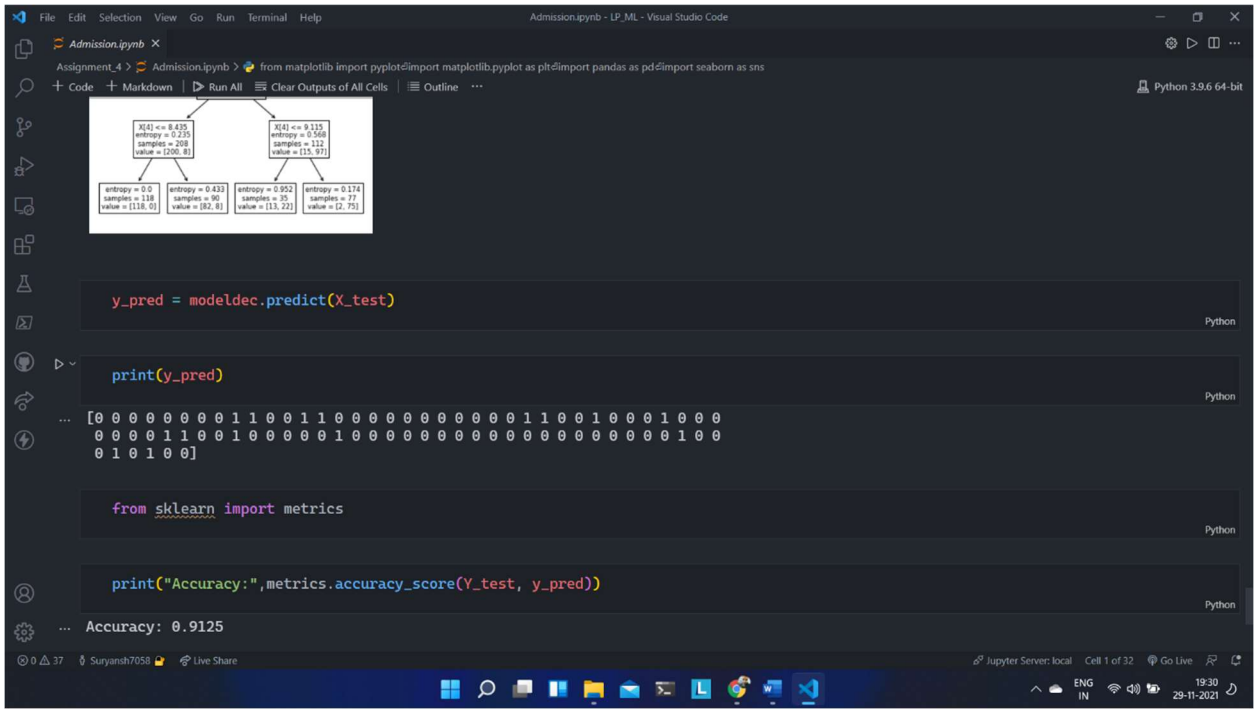



```
x_ads = admissionData[['GRE Score','TOEFL Score', 'University Rating','SOP','CGPA','Research']]
y_ads=admissionData["Chance of Admit "]
```


Admission.ipynb - LP_ML - Visual Studio Code
x_ads = admissionData[['GRE Score','TOEFL Score', 'University Rating','SOP','CGPA','Research']]
y_ads=admissionData["Chance of Admit "]
y_ads_conv = [1 if each > 0.8 else 0 for each in y_ads]
y_ads_conv
[1,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,


Admission.ipynb - LP_ML - Visual Studio Code
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_ads, y_ads_conv, test_size=0.2, random_state=101)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, roc_curve, auc
modeldec = DecisionTreeClassifier(random_state=0, max_depth=2, criterion='entropy').fit(X_train, Y_train)
PredictedOutput = modeldec.predict(X_test)
from sklearn import tree
tree.plot_tree(modeldec);





Conclusion:

Thus we have implemented K means clustering algorithm and visualized the clusters for customer segmentation dataset.

Assignment No . 4

AIM: Create Association Rules for the Market Basket Analysis for the given dataset.

THEORY:

Association Rules

There are many ways to see the similarities between items. These are techniques that fall under the general umbrella of association. The outcome of this type of technique, in simple terms, is a set of rules that can be understood as "if this, then **that"**.

Applications

There are many applications of association:

- Product recommendation – like Amazon's "customers who bought that, also bought this"
- Music recommendations – like Last FM's artist recommendations
- Medical diagnosis – like with diabetes really cool stuff
- Content optimisation – like in magazine websites or blogs

In this post we will focus on the retail application – it is simple, intuitive, and the dataset comes packaged with R making it repeatable.

Apriori Algorithm

Apriori algorithm is used to find patterns of relationships between one or more items in a dataset and this algorithm assumes that any subset of a frequent itemset must be frequent

The importance of an association rules can be determined by three parameters.

Support

It is the percentage of transactions containing particular combination of items relative to the total number of transactions in the database.

Confidence

It is a measure of conditional probability. Given that the item on the left hand side (**antecedent**) is purchased then the item on the right hand side(**consequent**) would also be purchased.
Confidence(**antecedent** => **consequent**)=(Transactions involving both **antecedent** and **consequent**)/(Transactions involving only **antecedent**)

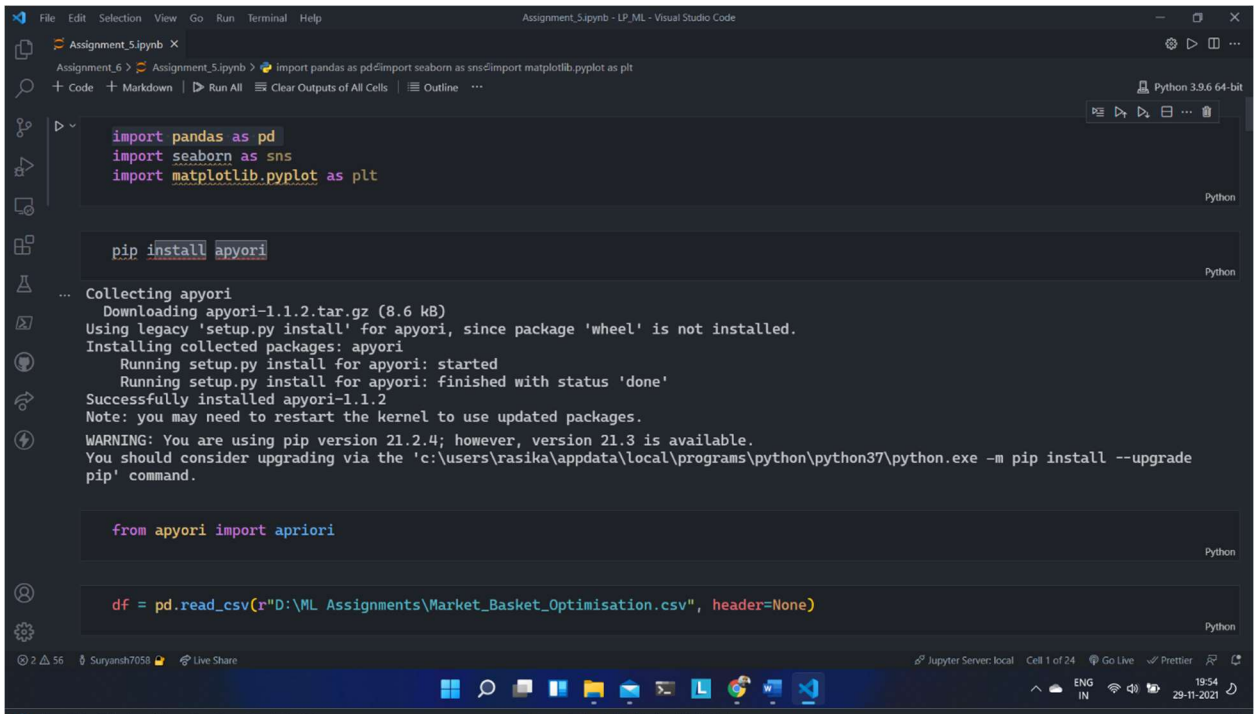
Lift

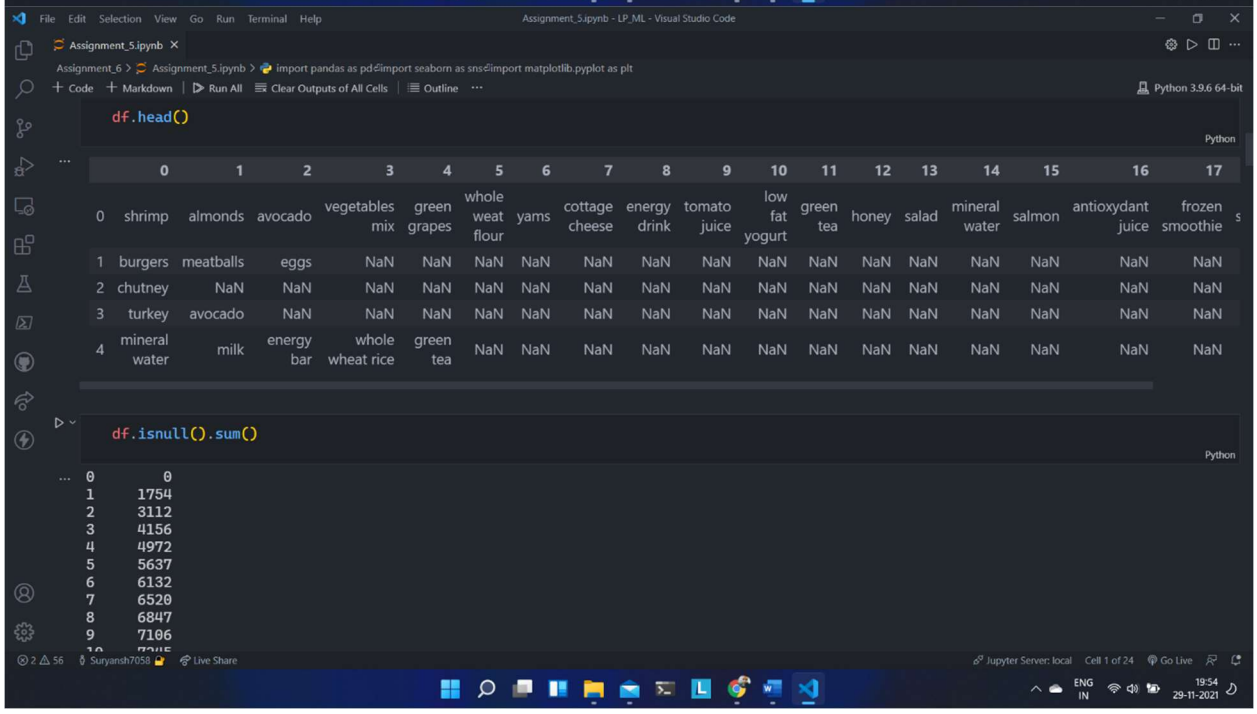
It is the probability of all items occurring together divided by the product of **antecedent** and **consequent** occurring as if they are independent of each other

Lift(antecedent => consequent**) = Confidence(**antecedent**,** consequent**) / Support(**consequent**)**

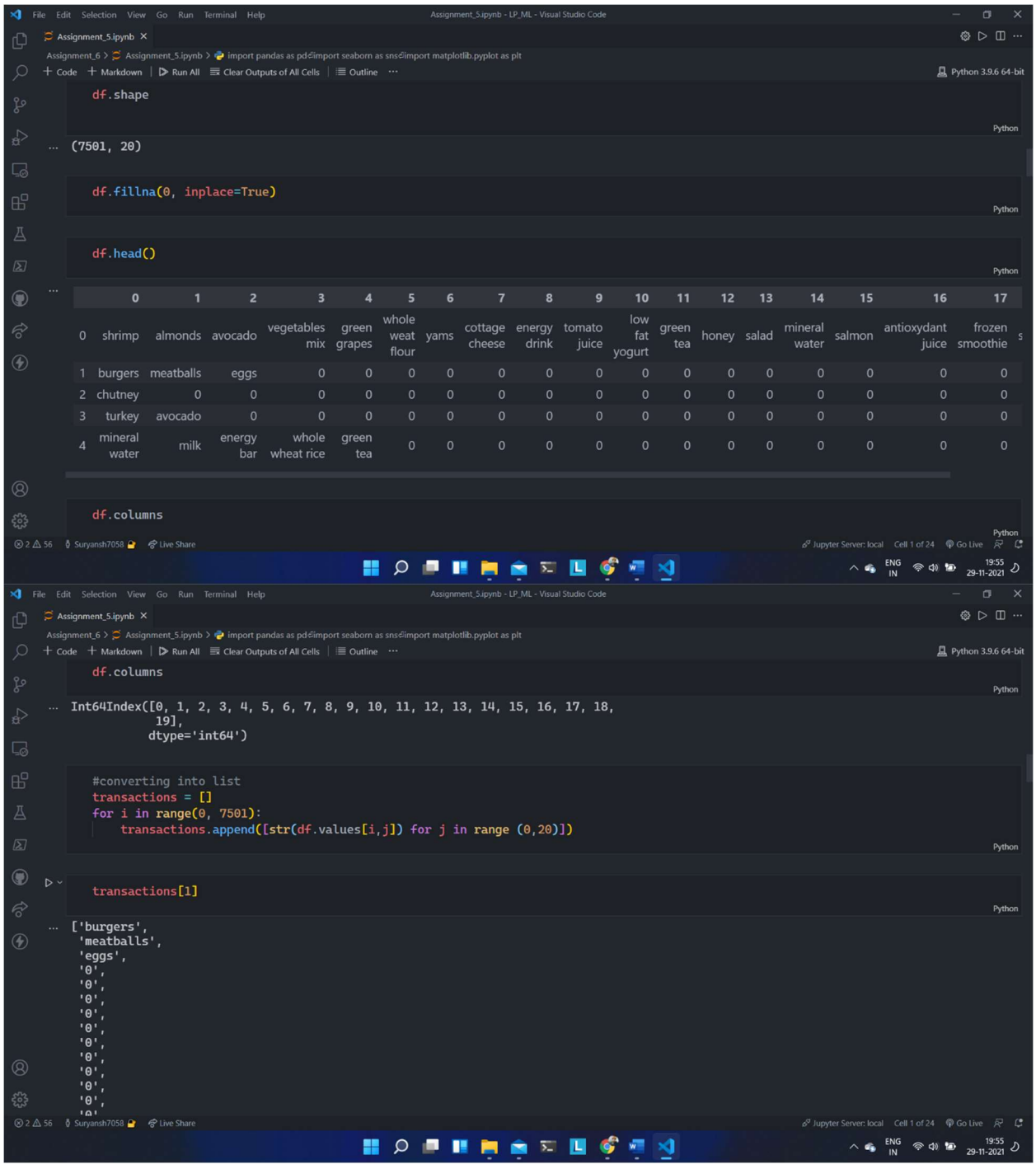
Therefore, likelihood of a customer buying both **antecedent** and **consequent** together is 'lift-value' times more than the chance if purchasing alone.

- **Lift (antecedent => consequent)** = 1 means that there is no correlation within the itemset.
- **Lift (antecedent=> consequent)** > 1 means that there is a positive correlation within the itemset, i.e., products in the itemset, **antecedent**, and **consequent**, are more likely to be bought together.
- **Lift (antecedent=> consequent)** < 1 means that there is a negative correlation within the itemset, i.e., products in itemset, **antecedent**, and **consequent**, are unlikely to be bought together.





	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



```
df.shape
```

```
(7501, 20)
```

```
df.fillna(0, inplace=True)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie
1	burgers	meatballs	eggs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	chutney	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	turkey	avocado	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	mineral water	milk	energy bar	whole wheat rice	green tea	0	0	0	0	0	0	0	0	0	0	0	0	0

```
df.columns
```

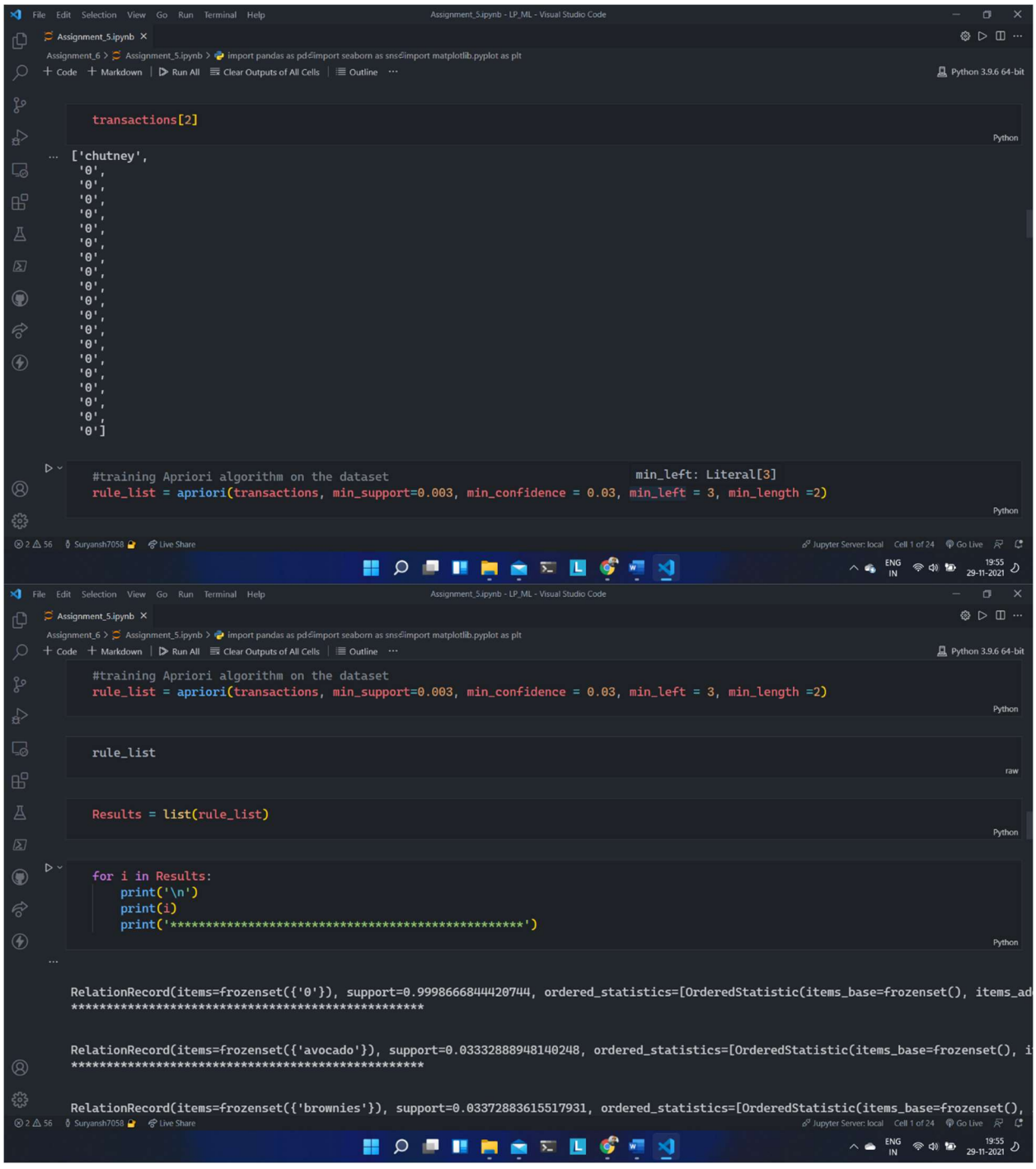
```
df.columns
```

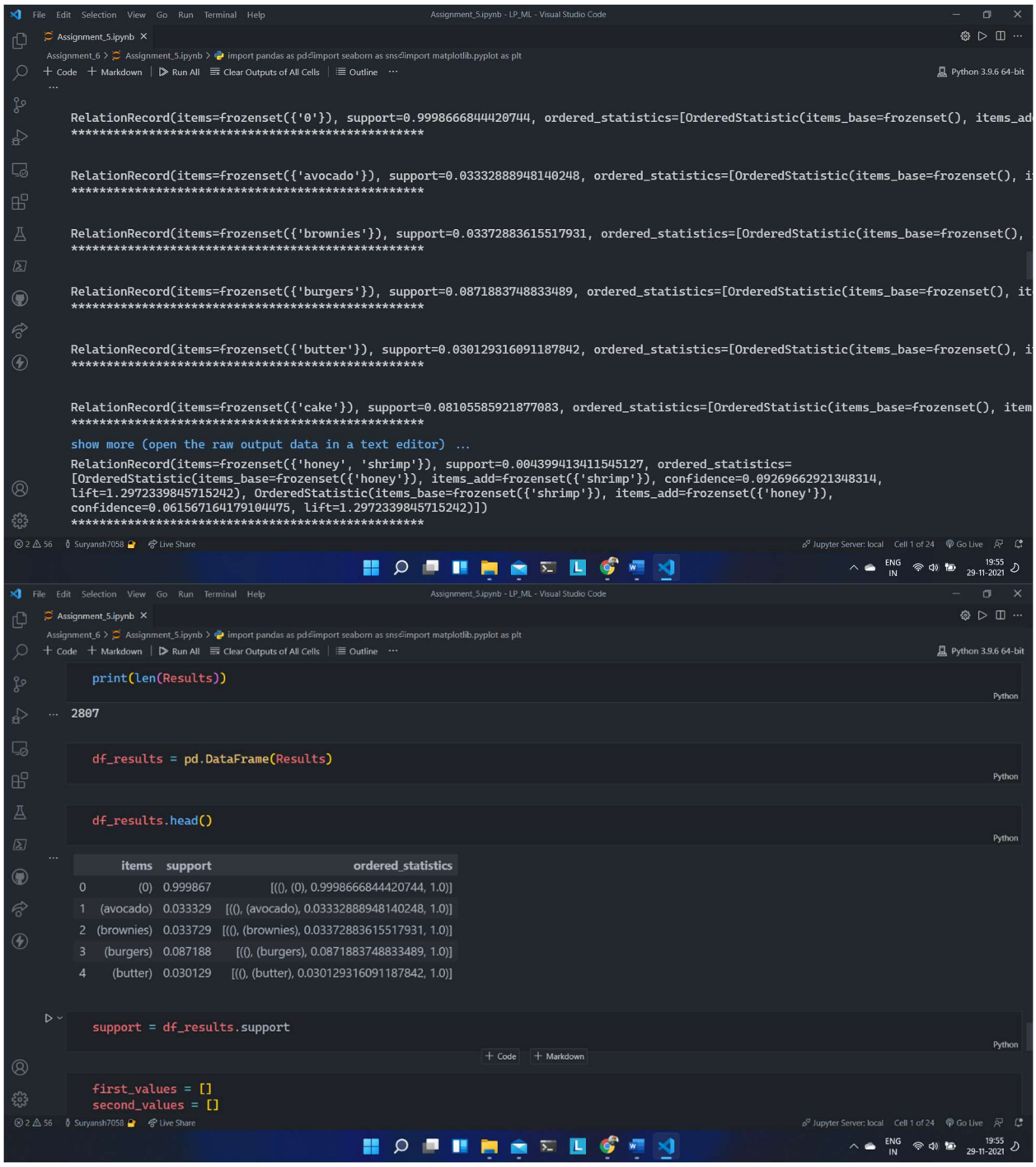
```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
            19],
           dtype='int64')
```

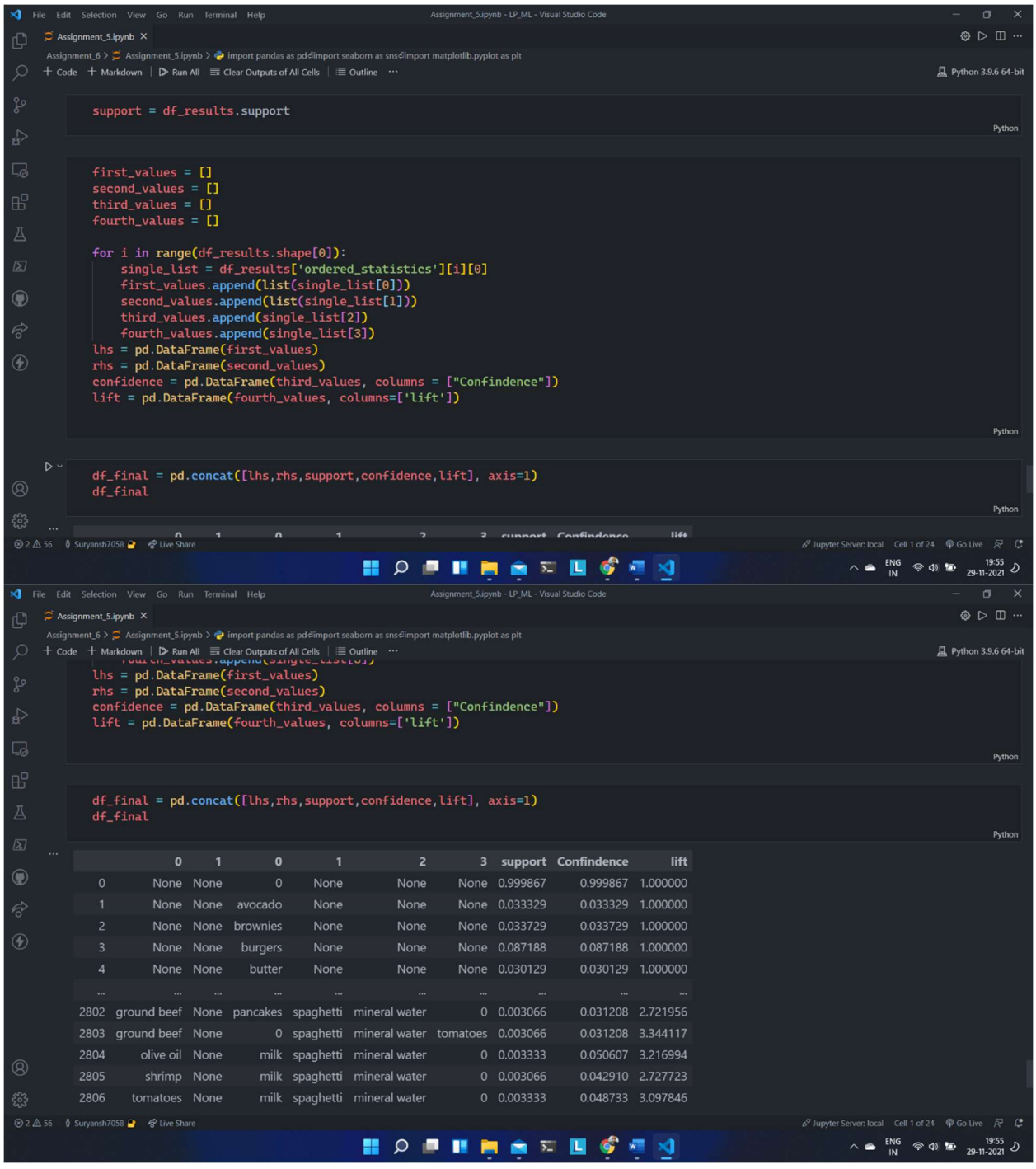
```
#converting into list
transactions = []
for i in range(0, 7501):
    transactions.append([str(df.values[i,j]) for j in range (0,20)])
```

```
transactions[1]
```

```
['burgers',
 'meatballs',
 'eggs',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
```


File Edit Selection View Go Run Terminal Help
Assignment_5.ipynb - LP_ML - Visual Studio Code
Assignment_5.ipynb
Code Markdown Run All Clear Outputs of All Cells Outline
Python 3.9.6 64-bit
transactions[2]
Python
['chutney',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0',
'0']
#training Apriori algorithm on the dataset
rule_list = apriori(transactions, min_support=0.003, min_confidence = 0.03, min_left = 3, min_length =2)
min_left: Literal[3]
Python
Suryansh7058 Live Share
Jupyter Server: local Cell 1 of 24 Go Live
ENG IN 19:55 29-11-2021
File Edit Selection View Go Run Terminal Help
Assignment_5.ipynb - LP_ML - Visual Studio Code
Assignment_5.ipynb
Code Markdown Run All Clear Outputs of All Cells Outline
Python 3.9.6 64-bit
#training Apriori algorithm on the dataset
rule_list = apriori(transactions, min_support=0.003, min_confidence = 0.03, min_left = 3, min_length =2)
Python
rule_list
raw
Results = list(rule_list)
Python
for i in Results:
print('\n')
print(i)
print('**********')
Python
RelationRecord(items=frozenset({'0'}), support=0.9998668444420744, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_ad
****
RelationRecord(items=frozenset({'avocado'}), support=0.03332888948140248, ordered_statistics=[OrderedStatistic(items_base=frozenset(), i
****
RelationRecord(items=frozenset({'brownies'}), support=0.03372883615517931, ordered_statistics=[OrderedStatistic(items_base=frozenset(),
Suryansh7058 Live Share
Jupyter Server: local Cell 1 of 24 Go Live
ENG IN 19:55 29-11-2021


File Edit Selection View Go Run Terminal Help
Assignment_5.ipynb - LP_ML - Visual Studio Code
Assignment_5.ipynb
Code Markdown Run All Clear Outputs of All Cells Outline
Python 3.9.6 64-bit
RelationRecord(items=frozenset({'0'}), support=0.99986668444420744, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_ad
RelationRecord(items=frozenset({'avocado'}), support=0.03332888948140248, ordered_statistics=[OrderedStatistic(items_base=frozenset(), i
RelationRecord(items=frozenset({'brownies'}), support=0.03372883615517931, ordered_statistics=[OrderedStatistic(items_base=frozenset(),
RelationRecord(items=frozenset({'burgers'}), support=0.0871883748833489, ordered_statistics=[OrderedStatistic(items_base=frozenset(), it
RelationRecord(items=frozenset({'butter'}), support=0.030129316091187842, ordered_statistics=[OrderedStatistic(items_base=frozenset(), i
RelationRecord(items=frozenset({'cake'}), support=0.08105585921877083, ordered_statistics=[OrderedStatistic(items_base=frozenset(), item
show more (open the raw output data in a text editor) ...
RelationRecord(items=frozenset({'honey', 'shrimp'}), support=0.004399413411545127, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'honey'}), items_add=frozenset({'shrimp'}), confidence=0.09269662921348314,
lift=1.2972339845715242), OrderedStatistic(items_base=frozenset({'shrimp'}), items_add=frozenset({'honey'}),
confidence=0.061567164179104475, lift=1.2972339845715242)])
Suryansh7058 Live Share
Jupyter Server: local Cell 1 of 24 Go Live
ENG IN 19:55 29-11-2021
print(len(Results))
Python
2807
df_results = pd.DataFrame(Results)
Python
df_results.head()
Python
items support ordered_statistics
0 (0) 0.999867 [(), (0), 0.9998666844420744, 1.0)]
1 (avocado) 0.033329 [(), (avocado), 0.03332888948140248, 1.0)]
2 (brownies) 0.033729 [(), (brownies), 0.03372883615517931, 1.0)]
3 (burgers) 0.087188 [(), (burgers), 0.0871883748833489, 1.0)]
4 (butter) 0.030129 [(), (butter), 0.030129316091187842, 1.0)]
support = df_results.support
Python
Code Markdown
first_values = []
second_values = []



	0	1	0	1	2	3	support	Confindence	lift
0	None	None	0	None	None	None	0.999867	0.999867	1.000000
1	None	None	avocado	None	None	None	0.033329	0.033329	1.000000
2	None	None	brownies	None	None	None	0.033729	0.033729	1.000000
3	None	None	burgers	None	None	None	0.087188	0.087188	1.000000
4	None	None	butter	None	None	None	0.030129	0.030129	1.000000
...	...	...	...	...	...	...	...	...	...
2802	ground beef	None	pancakes	spaghetti	mineral water	0	0.003066	0.031208	2.721956
2803	ground beef	None	0	spaghetti	mineral water	tomatoes	0.003066	0.031208	3.344117
2804	olive oil	None	milk	spaghetti	mineral water	0	0.003333	0.050607	3.216994
2805	shrimp	None	milk	spaghetti	mineral water	0	0.003066	0.042910	2.727723
2806	tomatoes	None	milk	spaghetti	mineral water	0	0.003333	0.048733	3.097846

Conclusion:

Thus we have implemented market basket analysis using Apriori algorithm and mined the Assocation rule.

Assignment No. 5

AIM:
To build a classification model using Decision trees and to Evaluate the performance of the model

THEORY:

Introduction

. Classification is a task of Machine Learning which assigns a label value to a specific class and then can identify a particular type to be of one kind or another. The most basic example can be of the mail spam filtration system where one can classify a mail as either “spam” or “not spam”.

Classification Predictive Modeling in Machine Learning

Classification usually refers to any kind of problem where a specific type of class label is the result to be predicted from the given input field of data. Some types of Classification challenges are :

- Classifying emails as spam or not
- Classify a given handwritten character to be either a known character or not
- Classify recent user behaviour as churn or not

For any model, you will require a training dataset with many examples of inputs and outputs from which the model will train itself. The training data must include all the possible scenarios of the problem and must have sufficient data for each label for the model to be trained correctly. Class labels are often returned as string values and hence needs to be encoded into an integer like either representing 0 for “spam” and 1 for “no-spam”

Generally, the different types of predictive models in machine learning are as follows :

- Binary classification
- Multi-Label Classification
- Multi-Class Classification
- Imbalanced Classification

Binary Classification for Machine Learning

A binary classification refers to those tasks which can give either of any two class labels as the output. Generally, one is considered as the normal state and the other is considered to be the abnormal state. The following examples will help you to understand them better.

- Email Spam detection: Normal State – Not Spam, Abnormal State – Spam
- Conversion prediction: Normal State – Not churned, Abnormal State – Churn
- Conversion Prediction: Normal State – Bought an item, Abnormal State – Not bought an item

The most popular algorithms which are used for binary classification are :

- K-Nearest Neighbours
- Logistic Regression
- Support Vector Machine
- Decision Trees
- Naive Bayes

Multi-Class Classification

- These types of classification problems have no fixed two labels but can have any number of labels. Some popular examples of multi-class classification are :

- Plant Species Classification
- Face Classification
- Optical Character recognition

Here there is no notion of a normal and abnormal outcome but the result will belong to one of many among a range of variables of known classes. There can also be a huge number of labels like predicting a picture as to how closely it might belong to one out of the tens of thousands of the faces of the recognition system.

The most common algorithms which are used for Multi-Class Classification are :

- K-Nearest Neighbours
- Naive Bayes
- Decision trees
- Gradient Boosting
- Random Forest
-

Multi-Label Classification for Machine Learning

In multi-label Classification, we refer to those specific classification tasks where we need to assign two or more specific class labels that could be predicted for each example. A basic example can be photo classification where a single photo can have multiple objects in it like a dog or an apple and etcetera. The main difference is the ability to predict multiple labels and not just one.

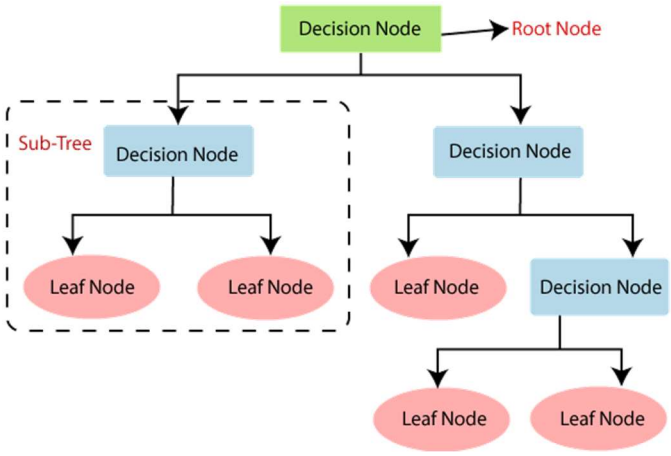
The common algorithms used here are :

- Multi-label Random Forests
- Multi-label Decision trees
- Multi-label Gradient Boosting

Decision Tree Classification Algorithm

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:



Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.
Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Algorithm

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.

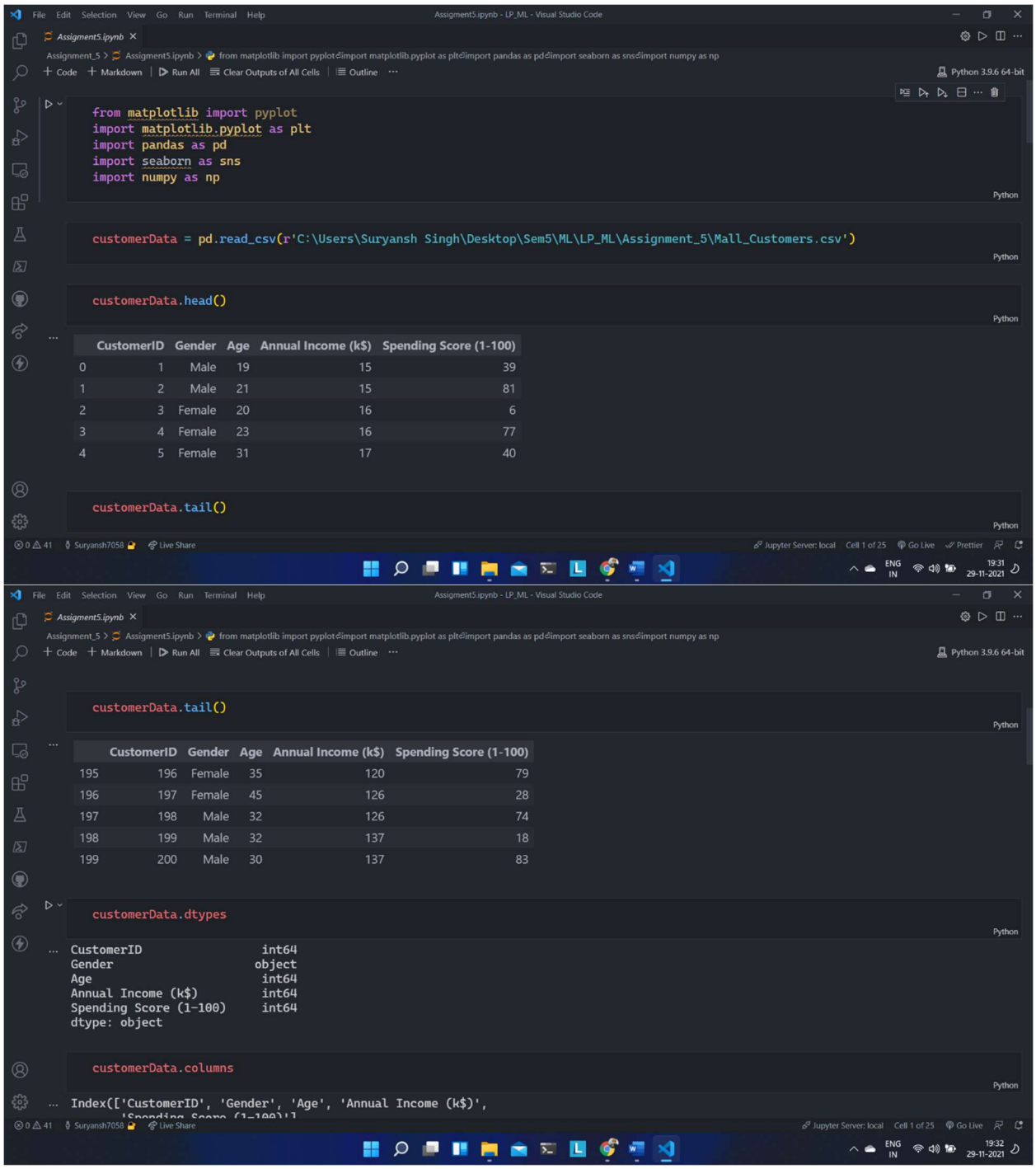
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

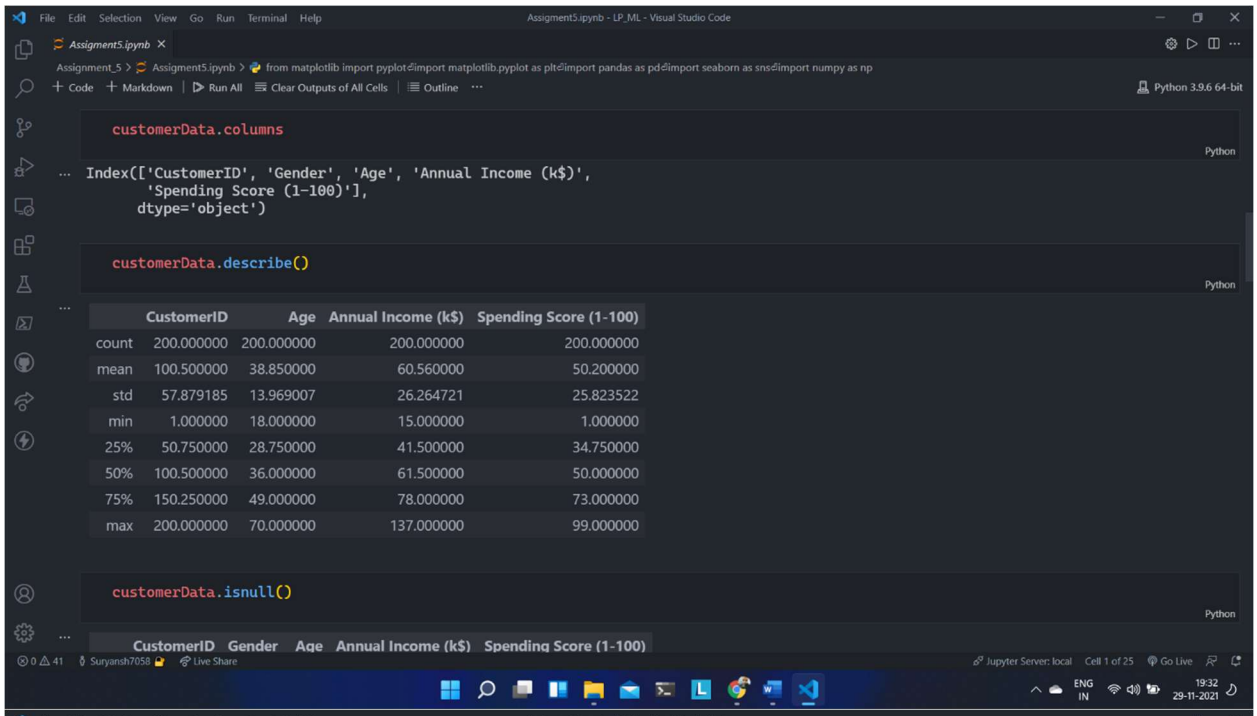
Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increas


Assigment5.ipynb - LP_ML - Visual Studio Code
from matplotlib import pyplot
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
customerData = pd.read_csv(r'C:\Users\Suryansh Singh\Desktop\Sem5\ML\LP_ML\Assignment_5\Mall_Customers.csv')
customerData.head()
customerData.tail()
customerData.tail()
customerData.dtypes
CustomerID int64
Gender object
Age int64
Annual Income (k$) int64
Spending Score (1-100) int64
dtype: object
customerData.columns
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',



```
customerData.columns
```

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

```
customerData.describe()
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

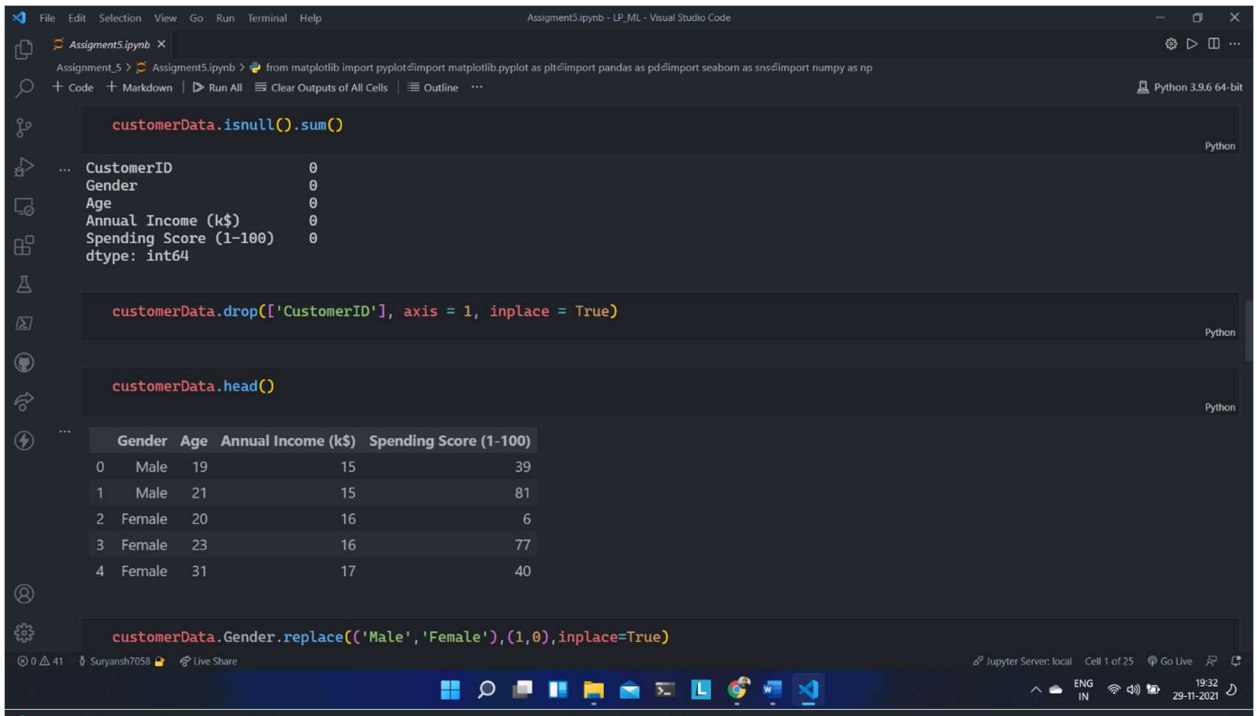
```
customerData.isnull()
```

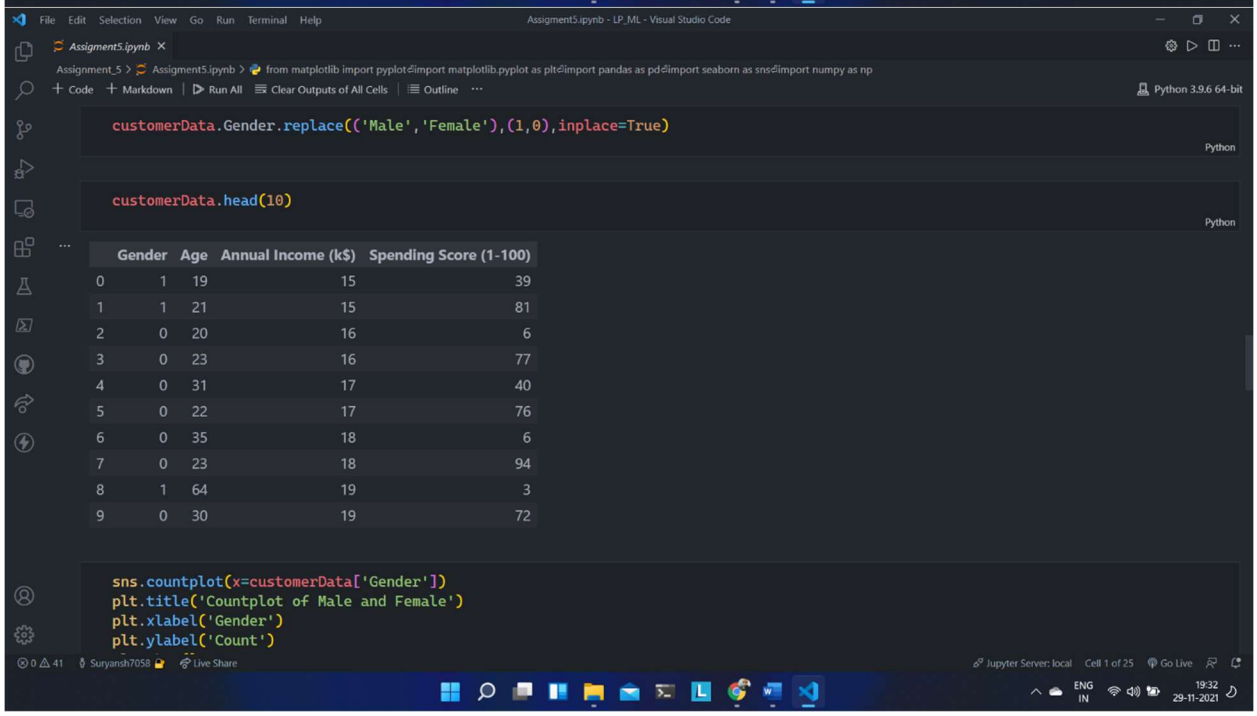
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
195	False	False	False	False	False
196	False	False	False	False	False
197	False	False	False	False	False
198	False	False	False	False	False
199	False	False	False	False	False

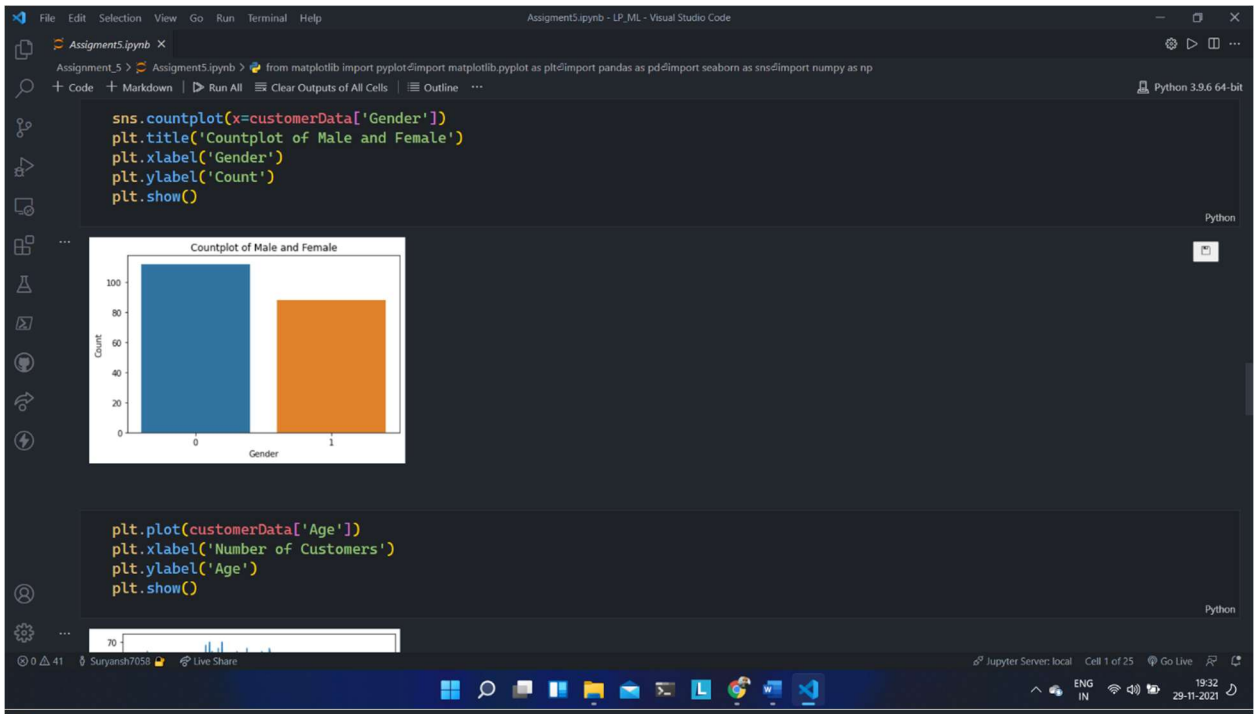
200 rows × 5 columns

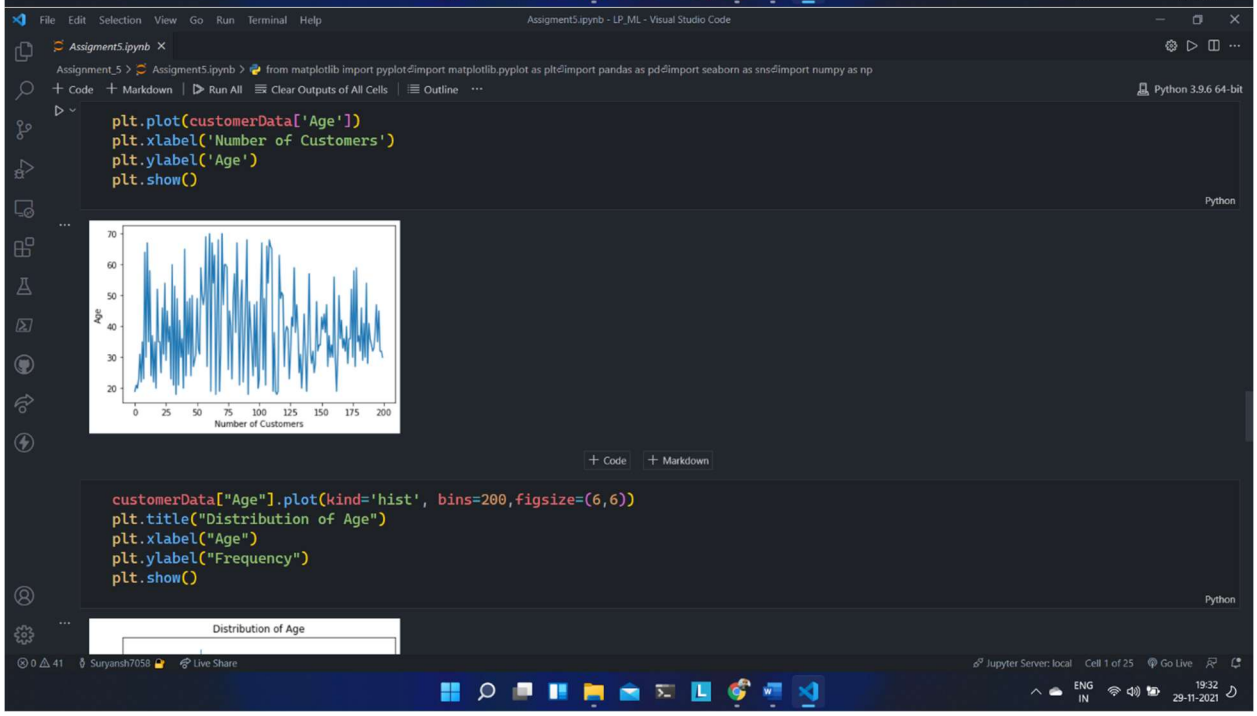
```
customerData.isnull().sum()
```

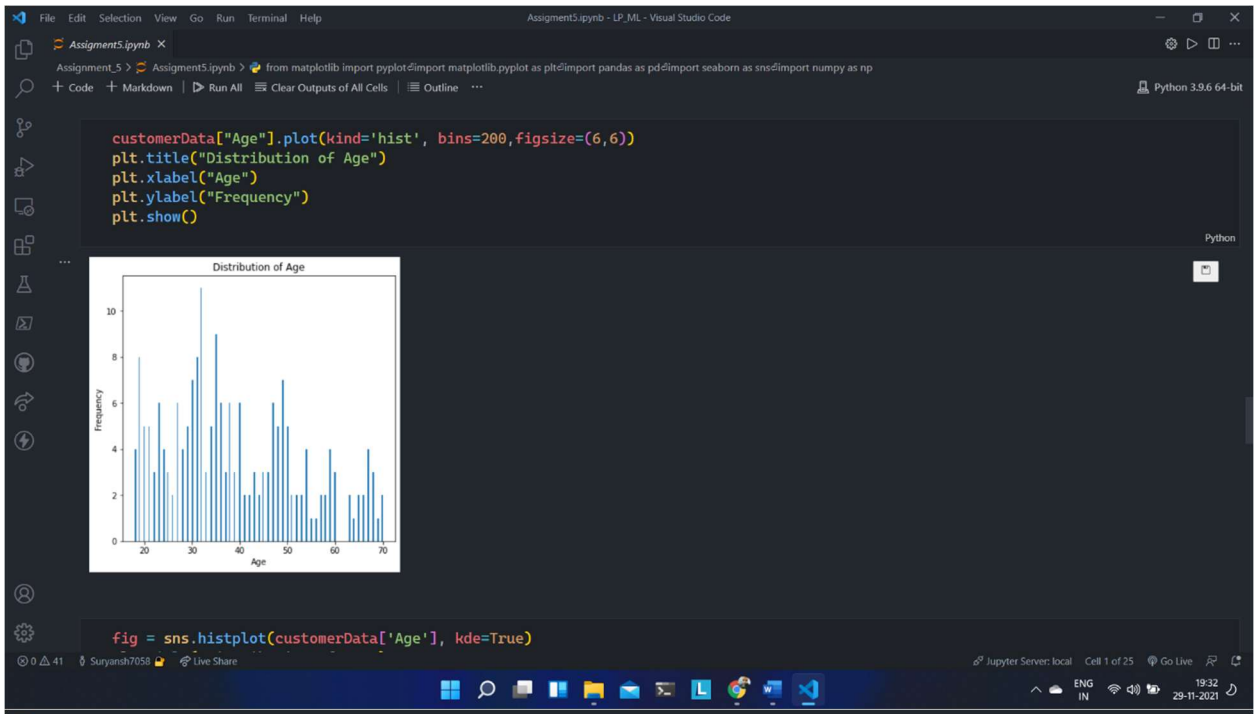
```
CustomerID              0
Gender                  0
```

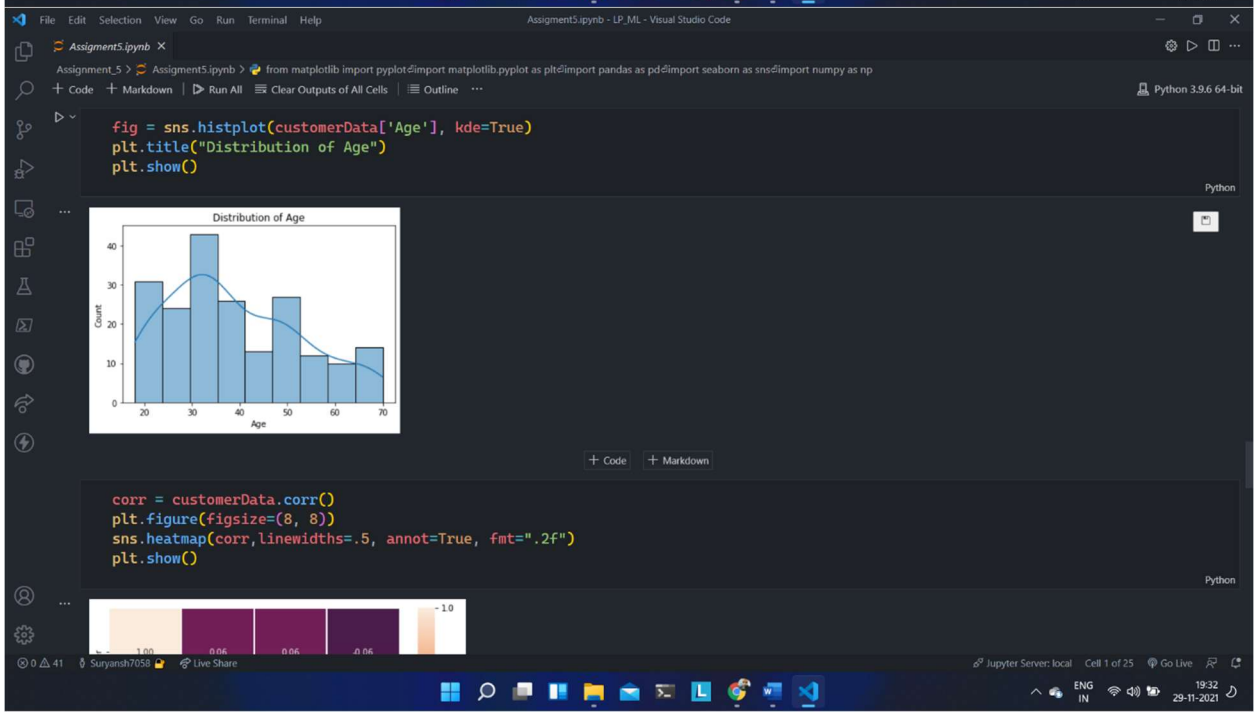

customerData.isnull().sum()
CustomerID 0
Gender 0
Age 0
Annual Income (k$) 0
Spending Score (1-100) 0
dtype: int64
customerData.drop(['CustomerID'], axis = 1, inplace = True)
customerData.head()
Gender Age Annual Income (k$) Spending Score (1-100)
0 Male 19 15 39
1 Male 21 15 81
2 Female 20 16 6
3 Female 23 16 77
4 Female 31 17 40
customerData.Gender.replace(('Male','Female'),(1,0),inplace=True)


customerData.Gender.replace(('Male','Female'),(1,0),inplace=True)
customerData.head(10)
Gender Age Annual Income (k$) Spending Score (1-100)
0 1 19 15 39
1 1 21 15 81
2 0 20 16 6
3 0 23 16 77
4 0 31 17 40
5 0 22 17 76
6 0 35 18 6
7 0 23 18 94
8 1 64 19 3
9 0 30 19 72
sns.countplot(x=customerData['Gender'])
plt.title('Countplot of Male and Female')
plt.xlabel('Gender')
plt.ylabel('Count')

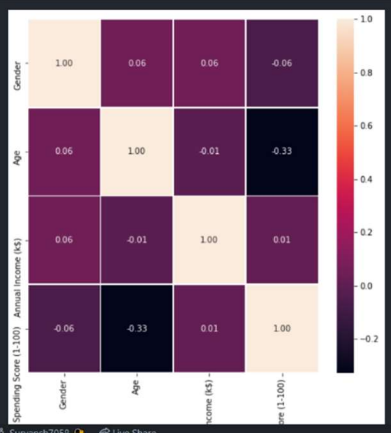

Assigment5.ipynb - LP_ML - Visual Studio Code
sns.countplot(x=customerData['Gender'])
plt.title('Countplot of Male and Female')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
Countplot of Male and Female
Count
Gender
plt.plot(customerData['Age'])
plt.xlabel('Number of Customers')
plt.ylabel('Age')
plt.show()


Assigment5.ipynb - LP_ML - Visual Studio Code
plt.plot(customerData['Age'])
plt.xlabel('Number of Customers')
plt.ylabel('Age')
plt.show()
Age
Number of Customers
customerData["Age"].plot(kind='hist', bins=200,figsize=(6,6))
plt.title("Distribution of Age")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
Distribution of Age

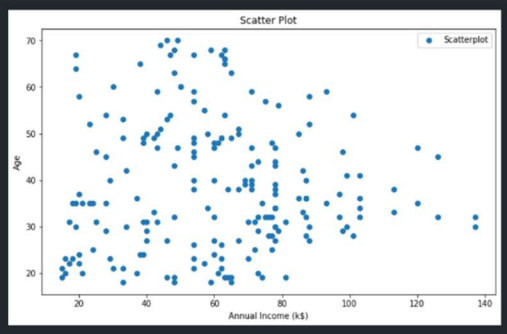

File Edit Selection View Go Run Terminal Help
Assigment5.ipynb - LP_ML - Visual Studio Code
Assigment5.ipynb
Assignment_5 > Assigment5.ipynb > from matplotlib import pyplot import matplotlib.pyplot as plt import pandas as pd import seaborn as sns import numpy as np
Code Markdown Run All Clear Outputs of All Cells Outline
Python 3.9.6 64-bit
customerData["Age"].plot(kind='hist', bins=200,figsize=(6,6))
plt.title("Distribution of Age")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
Python
Distribution of Age
Frequency
Age
fig = sns.histplot(customerData['Age'], kde=True)
Suryansh7058 Live Share
Jupyter Server: local Cell 1 of 25 Go Live
ENG IN 19:32 29-11-2021


File Edit Selection View Go Run Terminal Help
Assigment5.ipynb - LP_ML - Visual Studio Code
Assigment5.ipynb
Assignment_5 > Assigment5.ipynb > from matplotlib import pyplot import matplotlib.pyplot as plt import pandas as pd import seaborn as sns import numpy as np
Code Markdown Run All Clear Outputs of All Cells Outline
Python 3.9.6 64-bit
fig = sns.histplot(customerData['Age'], kde=True)
plt.title("Distribution of Age")
plt.show()
Python
Distribution of Age
Count
Age
Code Markdown
corr = customerData.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(corr,linewidths=.5, annot=True, fmt=".2f")
plt.show()
Python
Suryansh7058 Live Share
Jupyter Server: local Cell 1 of 25 Go Live
ENG IN 19:32 29-11-2021

```
corr = customerData.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(corr,linewidths=.5, annot=True, fmt=".2f")
plt.show()
```




```
plt.figure(figsize=(10,6))

plt.scatter(x=customerData["Annual Income (k$)"], y=customerData['Age'], label = "Scatterplot" )

plt.xlabel('Annual Income (k$)')
plt.ylabel('Age')

plt.title('Scatter Plot')
plt.legend()
plt.show()
```

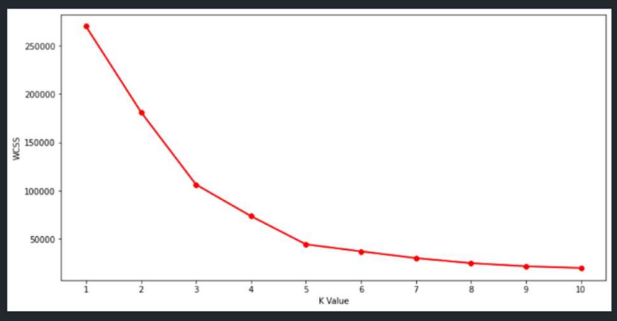


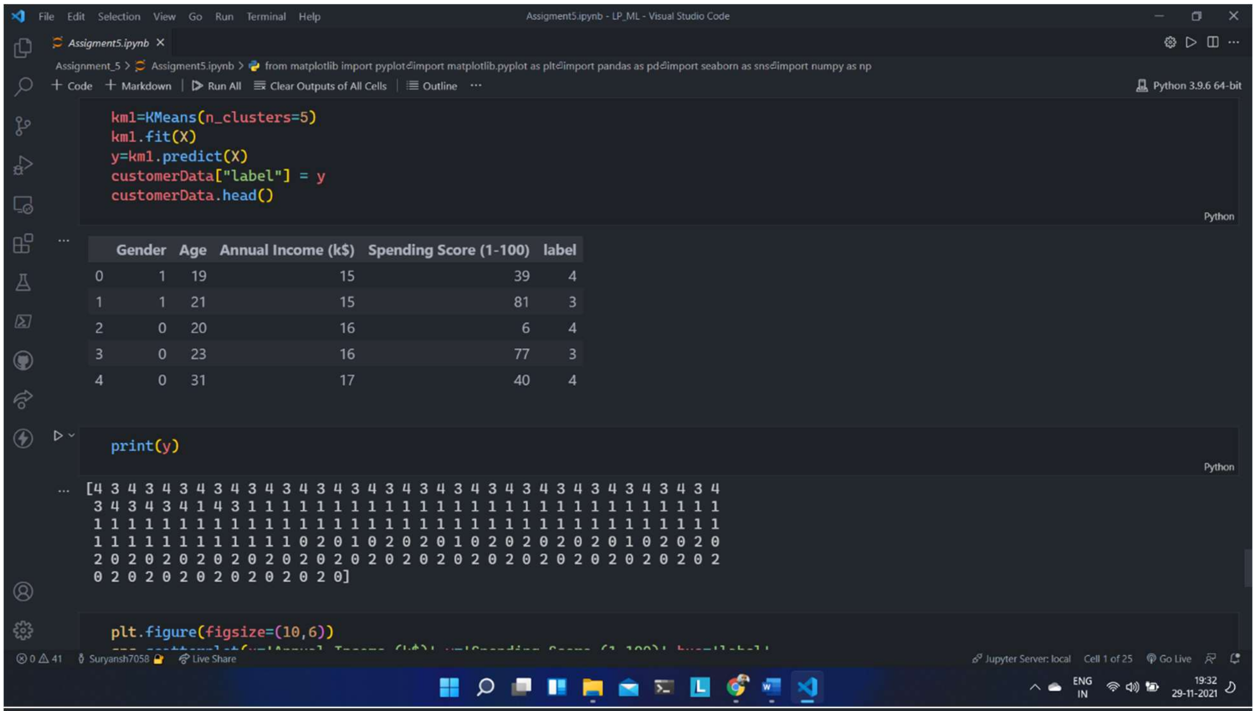
```
from sklearn.cluster import KMeans
X = customerData[["Annual Income (k$)","Spending Score (1-100)"]]
```

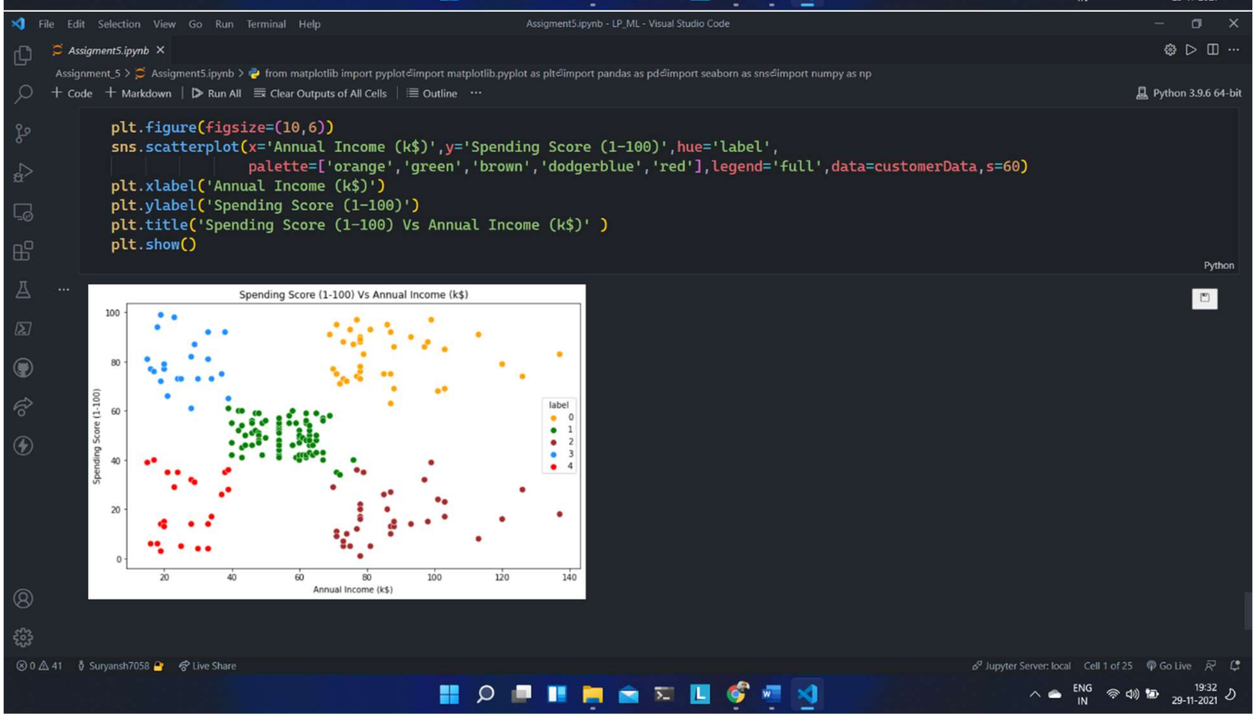
```
wcss=[]
for i in range(1,11):
    km=KMeans(n_clusters=i)
    km.fit(X)
    wcss.append(km.inertia_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

```
plt.figure(figsize =(12,6))
plt.plot(range(1,11),wcss)
plt.plot(range(1,11),wcss,linewidth=2,color="red",marker="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()
```







○

Conclusion:

Thus we have created a decision tree model and evaluated the performance for the given dataset.

Name.: Suryansh Sandeep Kumar Singh
Roll No.: S1951122
PRN: 72018560C
DIV: B (Batch B2)

Assignment No : 1

Title : Fractional knapsack using Greedy algorithm and 0/1 knapsack using dynamic programming

Problem Statement: Write a program to implement Fractional knapsack using Greedy algorithm and 0/1 knapsack using dynamic programming. Show that Greedy strategy does not necessarily yield an optimal solution over a dynamic programming approach.

Objective: Greedy strategy does not necessarily yield an optimal solution over a dynamic programming approach.

Theory:

1) Fractional knapsack using Greedy algorithm

Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

In **Fractional Knapsack**, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

A **brute-force solution** would be to try all possible subset with all different fraction but that will be too much time taking.

An **efficient solution** is to use Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.
A simple code with our own comparison function can be written as follows, please see sort function more closely, the third argument to sort function is our comparison function which sorts the item according to value/weight ratio in non-decreasing order.
After sorting we need to loop over these items and add them in our knapsack satisfying above-mentioned criteria.

Input :
Items as (value, weight) pairs
arr[] = {{60, 10}, {100, 20}, {120, 30}}
Knapsack Capacity, W = 50;
Output :
Maximum possible value = 240
By taking full items of 10 kg, 20 kg and
2/3rd of last item of 30 kg

Below is the implementation of the above idea:

Code :

```
#include <stdio.h>

void main()

{

   int capacity = 50;

   int value[] = {60, 100, 120};

   int weight[] = {10, 20, 30};

   int no_items = 3, cur_weight, item;

   int used[10];

   float total_profit;

   int i;

   for (i = 0; i < no_items; ++i)

      used[i] = 0;

   cur_weight = capacity;

   while (cur_weight > 0)

   {

      item = -1;

      for (i = 0; i < no_items; ++i)
         if ((used[i] == 0) && ((item == -1) || ((float)value[i] / weight[i] > (float)value[item] /
weight[item])))

            item = i;

      used[item] = 1;

      cur_weight -= weight[item];

      total_profit += value[item];

      if (cur_weight >= 0)

         printf("Added object %d (%d Rs., %dKg) completely in the bag. Space left: %d.\n", item
+ 1, value[item], weight[item], cur_weight);

      else
```

```
        {
            int item_percent = (int)((1 + (float)cur_weight / weight[item]) * 100);
            printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n",
item_percent,value[item], weight[item], item + 1);
            total_profit -= value[item];
            total_profit += (1 + (float)cur_weight / weight[item]) * value[item];
        }
    }
    printf("Filled the bag with objects worth %.2f Rs.\n", total_profit);
}
```

Theory :

2) 0/1 knapsack using dynamic programming

In the Dynamic programming we will work considering the same cases as mentioned in the recursive approach. In a DP[][] table let's consider all the possible weights from '1' to 'W' as the columns and weights that can be kept as the rows.
The state DP[i][j] will denote maximum value of 'j-weight' considering all values from '1 to ith'. So if we consider 'wi' (weight in 'ith' row) we can fill it in all columns which have 'weight values > wi'. Now two possibilities can take place:

- Fill 'wi' in the given column.
- Do not fill 'wi' in the given column.

Now we have to take a maximum of these two possibilities, formally if we do not fill 'ith' weight in 'jth' column then DP[i][j] state will be same as DP[i-1][j] but if we fill the weight, DP[i][j] will be equal to the value of 'wi'+ value of the column weighing 'j-wi' in the previous row. So we take the maximum of these two possibilities to fill the current state. This visualization will make the concept clear:

Let weight elements = {1, 2, 3}
Let weight values = {10, 15, 40}
Capacity=6

0 1 2 3 4 5 6

0 0 0 0 0 0 0 0

1 0 10 10 10 10 10 10

2 0 10 15 25 25 25 25

3 0

Explanation:

For filling 'weight = 2' we come
across 'j = 3' in which
we take maximum of
(10, 15 + DP[1][3-2]) = 25
| |
'2' '2 filled'
not filled

0 1 2 3 4 5 6

0 0 0 0 0 0 0 0

1 0 10 10 10 10 10 10

2 0 10 15 25 25 25 25

3 0 10 15 40 50 55 65

Explanation:

For filling 'weight=3',
we come across 'j=4' in which
we take maximum of (25, 40 + DP[2][4-3])
= 50

For filling 'weight=3'
we come across 'j=5' in which
we take maximum of (25, 40 + DP[2][5-3])
= 55

For filling 'weight=3'
we come across 'j=6' in which
we take maximum of (25, 40 + DP[2][6-3])
= 65

Complexity Analysis:

- **Time Complexity:** O(N*W).
  where 'N' is the number of weight element and 'W' is capacity. As for every weight element we traverse through all weight capacities 1<=w<=W.
- **Auxiliary Space:** O(N*W).
  The use of 2-D array of size 'N*W'.

Code :

/* A Naive recursive implementation

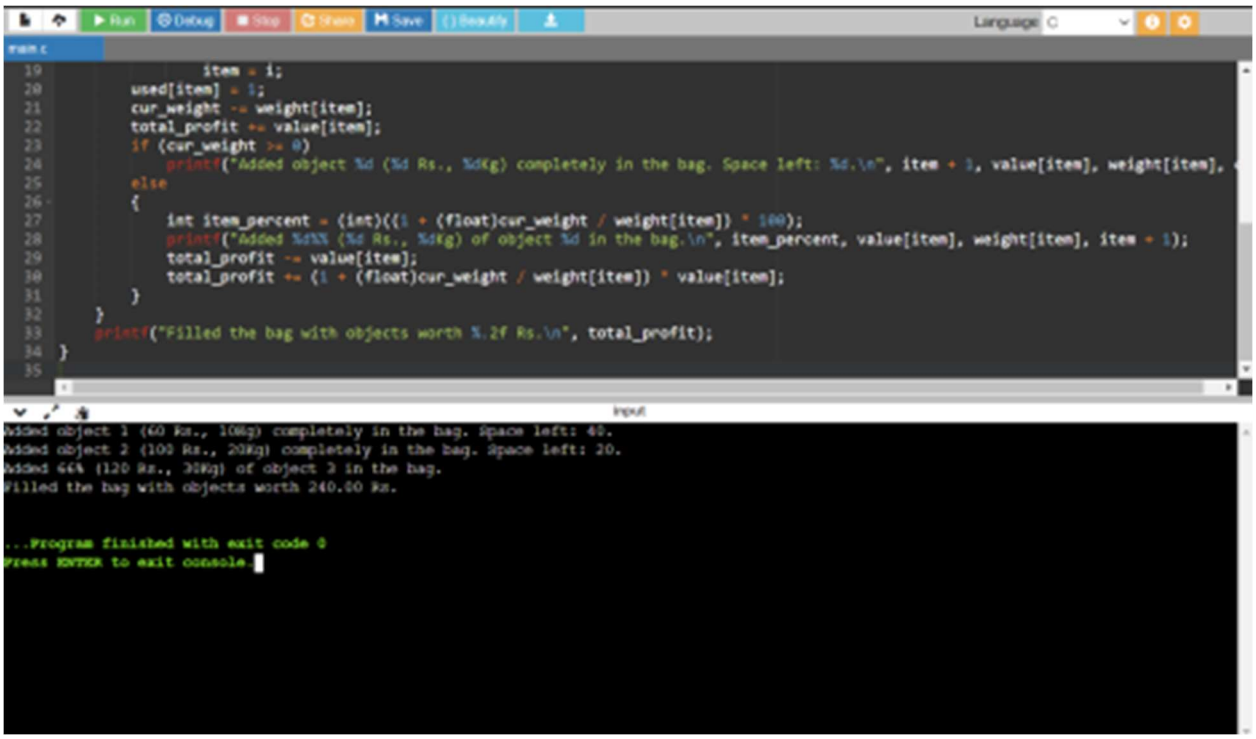
of 0-1 Knapsack problem */

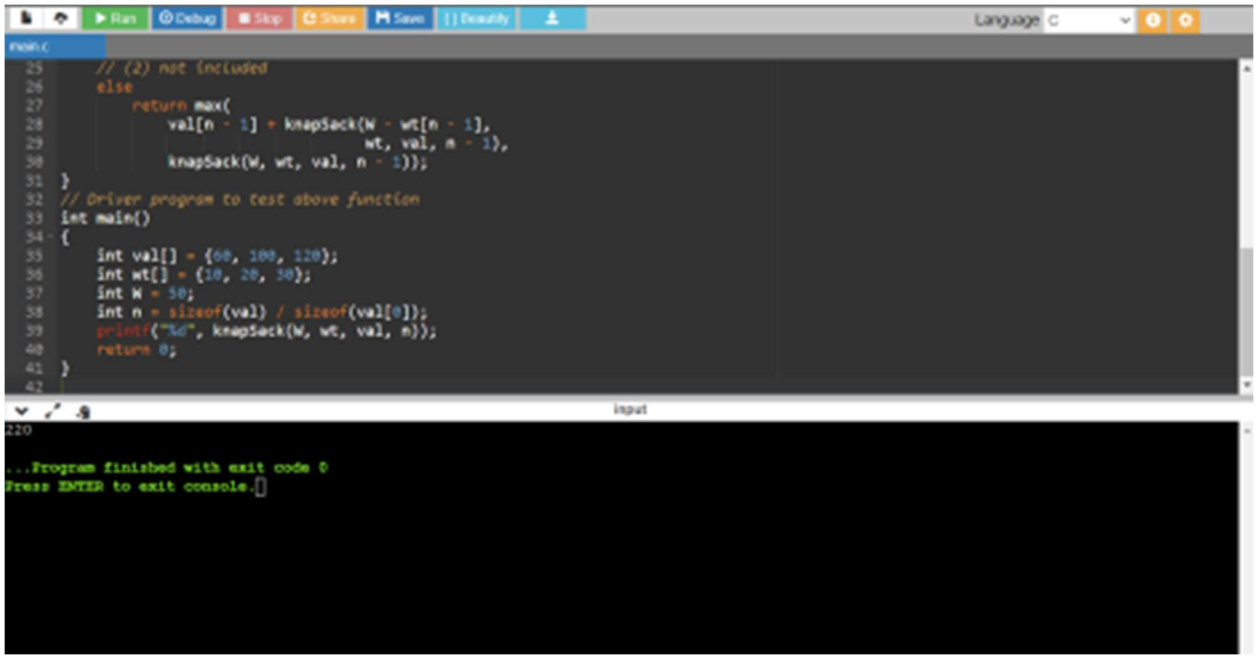
#include <stdio.h>

```
// A utility function that returns

// maximum of two integers

int max(int a, int b) { return (a > b) ? a : b; }


// Returns the maximum value that can be

// put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int

n) {

        // Base Case

        if (n == 0 || W == 0)

                return 0;


        // If weight of the nth item is more than
        // Knapsack capacity W, then this item

        cannot // be included in the optimal solution

        if (wt[n - 1] > W)

                return knapSack(W, wt, val, n - 1);


        // Return the maximum of two cases:

        // (1) nth item included

        // (2) not included

        else

                return max(

                        val[n - 1]

                                + knapSack(W - wt[n - 1],

                                                wt, val, n - 1),

                        knapSack(W, wt, val, n - 1));
```

```
}

// Driver program to test above function

int main()

{

        int val[] = { 60, 100, 120 };

        int wt[] = { 10, 20, 30 };

        int W = 50;

        int n = sizeof(val) / sizeof(val[0]);
        printf("%d", knapSack(W, wt, val, n));

        return 0;

}
```

Conclusion : We studied the Greedy strategy does not necessarily yield an optimal solution over a dynamic programming approach.





Assignment No : 2

Title : Bellman Ford Algorithm using Dynamic Programming

Problem Statement: Write a program to implement Bellman-Ford Algorithm using Dynamic Programming and verify the time complexity.

Objective: To understand the use of Bellman-Ford algorithm, for finding shortest path and implement.

Theory:

Bellman-Ford Algorithm:

Dijkstra and Bellman-Ford Algorithms used to find out single source shortest paths. i.e. there is a source node, from that node we have to find shortest distance to every other node. Dijkstra algorithm fails when graph has negative weight cycle. But Bellman-Ford Algorithm won't fail even, the graph has negative edge cycle. If there any negative edge cycle it will detect and say there is negative edge cycle. If not it will give answer to given problem.

Bellman-Ford Algorithm will work on logic that, **if graph has n nodes, then shortest path never contain more than n-1 edges**. This is exactly what Bellman-Ford do. It is enough to relax each edge (v-1) times to find shortest path. But to find whether there is negative cycle or not we again do one more relaxation. If we get less distance in nth relaxation we can say that there is negative edge cycle. Reason for this is negative value added and distance get reduced. **Relaxing edge**
In algorithm and code below we use this term Relaxing edge.

Relaxing edge is an operation performed on an edge (u, v) . when,

$d(u) > d(v) + Cost(u,v)$

Here d(u) means distance of u. If already known distance to "u" is greater than the path from "s"

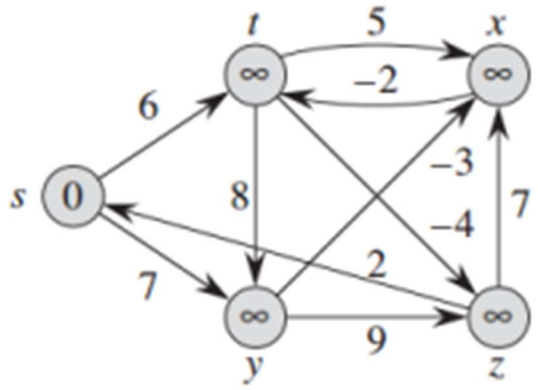
to “v” and “v” to “u” we update that d(u) value with d(v) + cost(u,v).

Algorithm and Time Complexity

```
Bellman-Ford (G,w,S){ //G is graph given, W is weight matrix, S is source vertex (starting
vertex)
1
    Initialize single source (G,S) //means initially distance to every node is infinity except to
2
source. Source is 0 (zero). This will take O(v) time
3
4
  For i=1 to |G.V| -1 //Runs (v-1) times
5
      For each edge (G,V)€ G.E // E times
6
          Relax(u,v,w) //O(1) time
7
8
  For each edge (G,V) € G.E
9
          If (v.d > u.d + w(u,v)) //The idea behind this is we are relaxing edges nth time if we found
10
more shortest path than (n-1)th level, we can say that graph is having negative edge cycle and
11
detected.
12
          Return False
13
  return true
}
```

Finally time complexity is (v-1) (E) O(1) = O(VE)

Example Problem



This is the given directed graph.

(s,t) = 6 (y,x) = -3

(s,y)= 7 (y,z) = 9

(t,y) = 8 (x,t) = -2

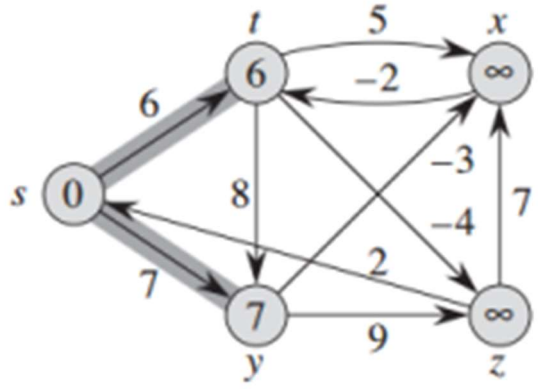
(t,z) = -4 (z,x) = 7

(t,x) = 5 (z,s) = 2

Above we can see using vertex “S” as source (Setting distance as 0), we initialize all other distance as infinity.

S T X Y Z distance 0 ∞ ∞ ∞ ∞ **Path** – – – – –

Table and Image explanation: This table, 2nd row shows distance from source to that particular node ad 3rd row shows to reach that node what is the node we visited recently. This path we can see in the image also.
Note: In each iteration, iteration “n” means it contains the path at most “n” edges. And while we are doing iteration “n” we must follow the graph which we obtained in iteration “n-1”. **Iteration 1:** edge (s,t) and (z,y) relaxed and updating the distances to t and y.

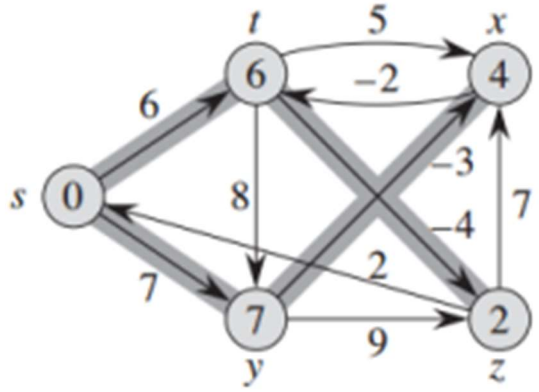


S T X Y Z

distance 0 6 ∞ 7 ∞

Path – S – S –

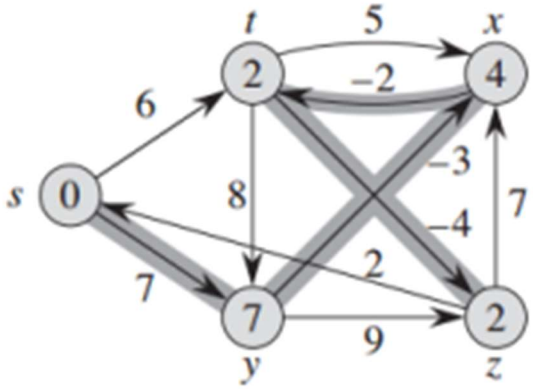
Iteration 2 : edge (t,z) and (y,x) relaxed and x and z values are updated.



S T X Y Z
distance 0 6 4 7 2

Path – S Y S T

Iteration 3: Value of t updated by relaxing (x,t)

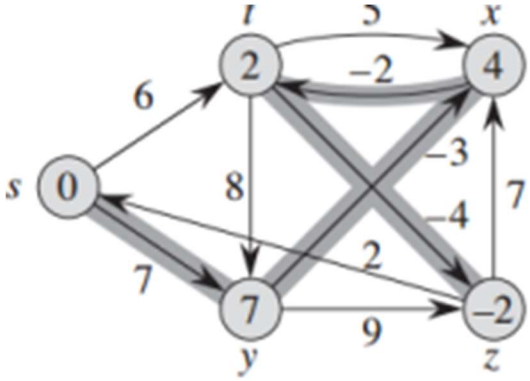


S T X Y Z

distance 0 2 4 7 2

Path – X Y S T

Iteration 4: Value of z updated by relaxing edge (t,z)



Until now 4 iterations completed and shortest path found to every node form source node. Now we have to do one more iteration to find whether there exists negative edge cycle or not. When we do this nth (5th here) relaxation if we found less distance to any vertex from any other path we can say that there is negative edge cycle. Here we can relax any edge to graph which obtained
in iteration 4and we can observe that there is no chance to change those values. So we can confirm that there is no negative edge cycle in this graph.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

struct Edge
{
        int source, destination, weight;
};

struct Graph
{
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
     struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));

    graph->V = V;
    graph->E = E;

     graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

void FinalSolution(int dist[], int n)
{
    printf("\nVertex\tDistance from Source Vertex\n");
    int i;
```

```
    for (i = 0; i < n; ++i){
        printf("%d \t\t %d\n", i, dist[i]);
    }
}

void BellmanFord(struct Graph* graph, int source)
{
    int V = graph->V;

    int E = graph->E;

    int StoreDistance[V];

    int i,j;


    for (i = 0; i < V; i++)
        StoreDistance[i] = INT_MAX;

    StoreDistance[source] = 0;

    for (i = 1; i <= V-1; i++)
    {
        for (j = 0; j < E; j++)
        {
            int u = graph->edge[j].source;

            int v = graph->edge[j].destination;

            int weight = graph->edge[j].weight;

            if (StoreDistance[u] + weight < StoreDistance[v])
                StoreDistance[v] = StoreDistance[u] + weight;
        }
    }



    for (i = 0; i < E; i++)
    {
        int u = graph->edge[i].source;

        int v = graph->edge[i].destination;

        int weight = graph->edge[i].weight;

        if (StoreDistance[u] + weight < StoreDistance[v])
    printf("This graph contains negative edge cycle\n"); }

    FinalSolution(StoreDistance, V);

    return;
}
int main()
{
    int V,E,S;
```

```
    printf("Enter number of vertices in graph\n");
    scanf("%d",&V);

    printf("Enter number of edges in graph\n");
    scanf("%d",&E);

    printf("Enter your source vertex number\n");
    scanf("%d",&S);

    struct Graph* graph = createGraph(V, E);
    int i;
    for(i=0;i<E;i++){
        printf("\nEnter edge %d properties Source, destination, weight respectively\n",i+1);
        scanf("%d",&graph->edge[i].source);
        scanf("%d",&graph->edge[i].destination);
        scanf("%d",&graph->edge[i].weight);
    }

    BellmanFord(graph, S);

    return 0;
}
```

```
input
Enter number of vertices in graph
5
Enter number of edges in graph
7
Enter your source vertex number
0

Enter edge 1 properties Source, destination, weight respectively
0 1 5

Enter edge 2 properties Source, destination, weight respectively
1 2 8

Enter edge 3 properties Source, destination, weight respectively
2 3 2

Enter edge 4 properties Source, destination, weight respectively
3 4 6

Enter edge 5 properties Source, destination, weight respectively
0 4 7

Enter edge 6 properties Source, destination, weight respectively
0 2 4

Enter edge 7 properties Source, destination, weight respectively
1 4 9

Vertex  Distance from Source Vertex
0               0
1               5
2               4
3               6
4               7
```

CONCLUSION: We studied the use and implementation of bellman ford successfully.

Assignment No : 3

Problem Statement: - Write a recursive program to find the solution of placing n queens on chessboard so that no two queens attack each other using Backtracking

Objective: - In chess, a queen can move as far as she pleases, horizontally, vertically, or diagonally. A chess board has 8 rows and 8 columns. The standard 8 by 8 Queen's problem asks how to place 8 queens on an ordinary chess board so that none of them can hit any other in one move.

Theory: -

The **eight queens puzzle** is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general ***n*-queens problem** of placing *n* queens on an *n*×*n* chessboard, where solutions exist for all natural numbers *n* with the exception of *n*=2 and *n*=3. The eight queens puzzle has 92 **distinct** solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 **fundamental** solutions.
Here we are solving it for N queens in NxN chess board.

(Explain 5 queens problem with Diagram)

Approach:

- Create a solution matrix of the same structure as chess board.
- Whenever place a queen in the chess board, mark that particular cell in solution matrix.
- At the end print the solution matrix, the marked cells will show the positions of the queens in the chess board.

Algorithm

Place the queens column wise, start from the left most column

1. If all queens are placed.
    1. return true and print the solution matrix.
2. Else
    1. Try all the rows in the current column.
    2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
    3. If placing the queen in above step leads to the solution return true.
    4. If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
3. If all the rows are tried and nothing worked, return false and print NO SOLUTION.

Better Solution: If you notice in solution matrix, at every row we have only one entry as 1 and rest of the entries are 0. Solution matrix takes $O(N^2)$ space. We can reduce it to O(N). We will solve it by taking one dimensional array and consider solution[1] = 2 as "Queen at 1st row is placed at 2nd column

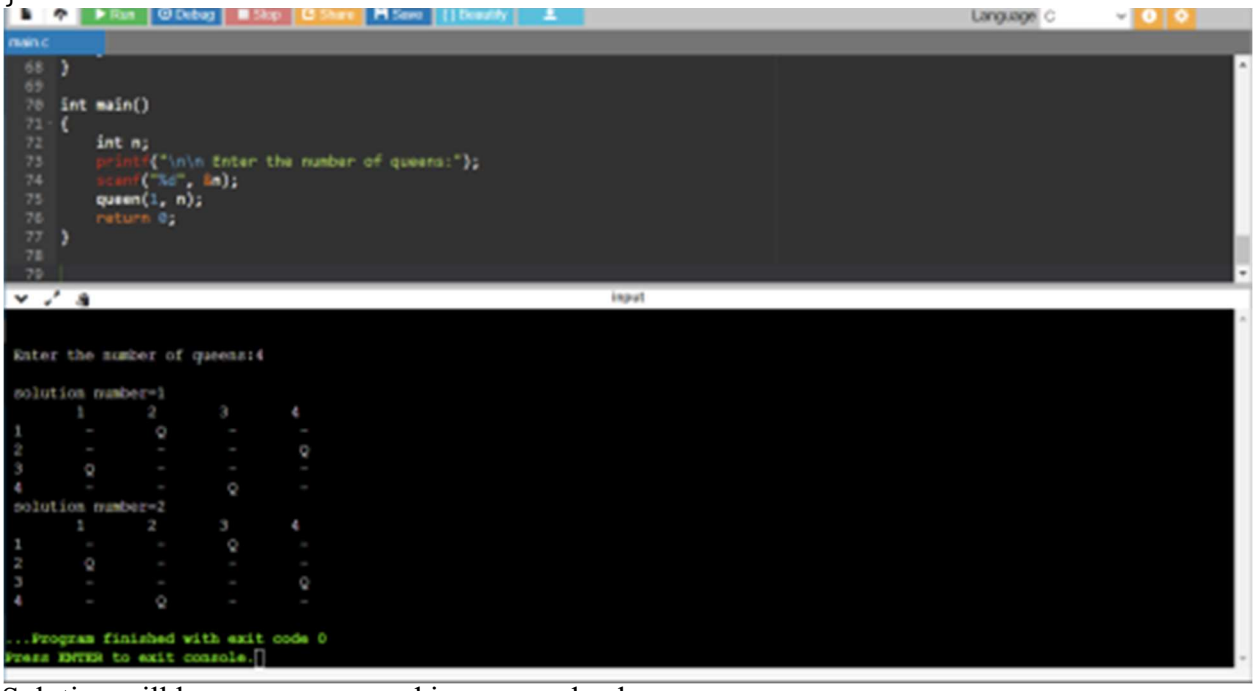
Input:
Enter the number of queens.

Code :

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>

int a[30], count;


int place(int i, int j)
{
    int k;
    for (k = 1; k <= i - 1; k++)
    {
        if (a[k] == j)
        {
            return 0;
        }
        else if (abs(a[k] - j) == abs(k - i))
        {
            return 0;
        }
    }
    return 1;
}

void print_func(int n)
{
    int i, j, k;
    printf("\n solution number=%d\n", ++count);
    for (i = 1; i <= n; i++)
    {
        printf("\t%d", i);
    }
    for (i = 1; i <= n; i++)
    {
        printf("\n %d", i);
        for (j = 1; j <= n; j++)
        {
            if (a[i] == j)
            {
                printf("\t Q");
            }

            else
            {
                printf("\t -");
            }
        }
    }
}

void queen(int i, int n)
{
    int j, k;
    for (j = 1; j <= n; j++)
    {
        if (place(i, j))
        {
            a[i] = j;
            if (i == n)
            {
```

```
                print_func(n);
            }
            else
            {
                queen(i + 1, n);
            }
        }
    }
}


int main()
{
    int n;
    printf("\n\n Enter the number of queens:");
    scanf("%d", &n);
    queen(1, n);
    return 0;
}
```



Solution will be queens arranged in rows and column

Assignment No : 4

***Problem Statement*:** Write a program to solve the travelling salesman problem and to print the path and the cost using Branch and Bound

Objective: To understand the concept of Branch and bound used in Travel Salesman Problem.

Theory:

Branch and Bound Solution

In Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node.
Note that the cost through a node includes two costs.
1) Cost of reaching the node from the root (When we reach a node, we have this cost computed)

2) Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore subtree with this node or not).

- In cases of a **maximization problem**, an upper bound tells us the maximum possible solution if we follow the given node. For example in 0/1 knapsack we used Greedy approach to find an upper bound.
- In cases of a **minimization problem**, a lower bound tells us the minimum possible solution if we follow the given node. For example, in Job Assignment Problem, we get a lower bound by assigning least cost job to a worker.

In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Traveling salesman problem.

Cost of any tour can be written as below.

Cost of a tour T = (1/2) * ∑ (Sum of cost of two edges

adjacent to u and in the

tour T)

where u ∈ V

For every vertex u, if we consider two edges through it in T,

and sum their costs. The overall sum for all vertices would

be twice of cost of tour T (We have considered every edge

twice.)

(Sum of two tour edges adjacent to u) >= (sum of minimum weight

two edges adjacent to

u)

Cost of any tour >= 1/2) * ∑ (Sum of cost of two minimum

weight edges adjacent to u)

where u ∈ V

For example, consider the above shown graph. Below are minimum cost two edges adjacent to every node.

Node Least cost edges Total cost

0 (0, 1), (0, 2) 25

1 (0, 1), (1, 3) 35

2 (0, 2), (2, 3) 45

3 (0, 3), (1, 3) 45

Thus a lower bound on the cost of any tour =

1/2(25 + 35 + 45 + 45)

= 75

Refer this for one more example.

Now we have an idea about computation of lower bound. Let us see how to how to apply it state space search tree. We start enumerating all possible nodes (preferably in lexicographical order)

1. The Root Node: Without loss of generality, we assume we start at vertex "0" for which the lower bound has been calculated above.
Dealing with Level 2: The next level enumerates all possible vertices we can go to (keeping in mind that in any path a vertex has to occur only once) which are, 1, 2, 3… n (Note that the graph is complete). Consider we are calculating for vertex 1, Since we moved from 0 to 1, our tour has now included the edge 0-1. This allows us to make necessary changes in the lower bound of the root.

Lower Bound for vertex 1 =
Old lower bound - ((minimum edge cost of 0 +

minimum edge cost of 1) / 2)

+ (edge cost 0-1)

How does it work? To include edge 0-1, we add the edge cost of 0-1, and subtract an edge weight such that the lower bound remains as tight as possible which would be the sum of the minimum edges of 0 and 1 divided by 2. Clearly, the edge subtracted can't be smaller than this.

Dealing with other levels: As we move on to the next level, we again enumerate all possible vertices. For the above case going further after 1, we check out for 2, 3, 4, …n. Consider lower bound for 2 as we moved from 1 to 1, we include the edge 1-2 to the tour and alter the new lower bound for this node.

Lower bound(2) =

Old lower bound - ((second minimum edge cost of 1 +

minimum edge cost of 2)/2)

+ edge cost 1-2)

Note: The only change in the formula is that this time we have included second minimum edge cost for 1, because the minimum edge cost has already been subtracted in previous level

Time Complexity: The worst case complexity of Branch and Bound remains same as that of the Brute Force clearly because in worst case, we may never get a chance to prune a node. Whereas, in practice it performs very well depending on the different instance of the TSP. The complexity also depends on the choice of the bounding function as they are the ones deciding how many nodes to be pruned.

Code :

```
#include <stdio.h>
#include <conio.h>
int main()
{

    int
        cost[20][20],
        min, l, m, sr[20], sc[20], flag[20][20], i, j, k, rf[20], cf[20], n;

    int
        nrz[20],
        ncz[20], cn, a, noz, nrz1[20], ncz1[20], counter = 0;

    printf(
        "\n\nEnter the total number of assignments:");
    scanf(
        "%d", &n);

    /* Enter the cost matrix*/

    printf(
        "\nEnter the cost matrix\n");

    for (i = 0; i < n; i++)

    {

        printf(
            "\n");

        for (j = 0; j < n; j++)

        {

            printf(
                "cost[%d][%d] = ", i, j);

            scanf(
                "%d", &cost[i][j]);
        }
    }

    printf(
        "\n\n");
```

```
    /* Display the entered cost matrix*/

    printf(
        "Cost matrix:\n");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < n; j++)

            printf(
                "\t%d\t", cost[i][j]);

        printf(
            "\n");
    }
    /* operation on rows*/

    for (i = 0; i < n; i++)

    {

        min = cost[i][0];

        /* find the minmum element in each row*/

        for (j = 0; j < n; j++)

        {

            if (min > cost[i][j])

                min = cost[i][j];
        }

        /*subtract the minimum element from each element of the respective
rows*/

        for (j = 0; j < n; j++)

            cost[i][j] = cost[i][j] - min;
    }

    /* operation on colums*/

    for (i = 0; i < n; i++)

    {

        min = cost[0][i];

        /* find the minimum element in each column*/

        for (j = 0; j < n; j++)
```

```
        {

            if (min > cost[j][i])

                min = cost[j][i];
        }
        /*subtract the minimum element from each element of the respective
columns*/

        for (j = 0; j < n; j++)

            cost[j][i] = cost[j][i] - min;
    }

    printf(
        "\n\n");

    printf(
        "Cost matrix after row & column operation:\n");

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)

            printf(
                "\t%d\t", cost[i][j]);

        printf(
            "\n");
    }

repeatx:;

    /*Draw minimum number of horizontal and vertical lines to cover all zeros
in

resulting matrix*/

    a = 0;
    noz = 0, min = 1000;

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)

            flag[i][j] = 0;
    }

    for (i = 0; i < n; i++)

    {
        cn = 0;
        for (j = 0; j < n; j++)
```

```
    {
        if (cost[i][j] == 0)

        {
            cn++;

            flag[i][j] = 1;
        }
    }

    nrz[i] = cn;

    noz = noz + cn;
}

for (i = 0; i < n; i++)

{
    cn = 0;

    for (j = 0; j < n; j++)

    {
        if (cost[j][i] == 0)

        {
            cn++;

            flag[j][i] = 1;
        }
    }

    ncz[i] = cn;

    noz = noz + cn;
}

for (i = 0; i < n; i++)

{
    nrz1[i] = nrz[i];

    ncz1[i] = ncz[i];
}
k = 0;

while (nrz[k] != 0 || ncz[k] != 0)

{

    for (i = 0; i < n; i++)

    {
        cn = 0;
```

```
    for (j = 0; j < n; j++)

    {
        if (flag[i][j] == 1)

            cn++;

        nrz[i] = cn;
    }

    if (nrz[i] == 1)

    {
        for (j = 0; j < n; j++)

        {
            if (flag[i][j] == 1)

            {
                flag[i][j] = 2;

                for (k = 0; k < n; k++)

                {
                    if (flag[k][j] == 1)

                        flag[k][j] = 0;
                }
            }
        }
    }
}

for (i = 0; i < n; i++)

{
    cn = 0;
    for (j = 0; j < n; j++)

    {
        if (flag[j][i] == 1)

            cn++;

        ncz[i] = cn;
    }

    if (ncz[i] == 1)

    {
        for (j = 0; j < n; j++)

        {
            if (flag[j][i] == 1)

            {
```

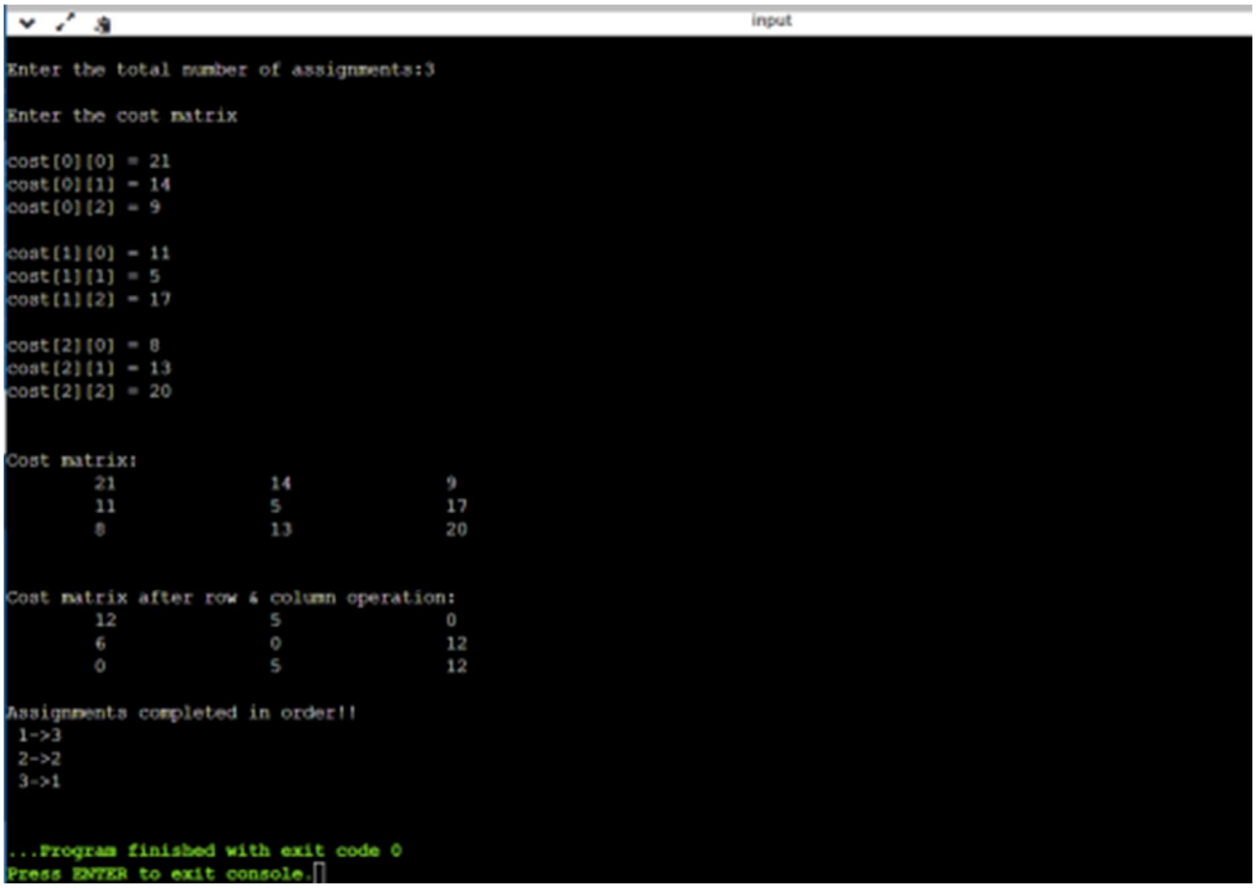
```
                        flag[j][i] = 2;

                        for (k = 0; k < n; k++)

                        {
                            if (flag[j][k] == 1)

                                flag[j][k] = 0;
                        }
                    }
                }
            }
        }

        k++;
    }

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)

        {
            if (flag[i][j] == 2)

                a++;
        }
    }
    /* If minimum number of lines, a is equal to the order of the matrix n
then

assignment can be optimally completed.*/

    if (a == n)

    {

        printf(
            "\nAssignments completed in order!!\n");

        /* Display the order in which assignments will be completed*/

        for (i = 0; i < n; i++)

        {
            for (j = 0; j < n; j++)

            {
                if (flag[i][j] == 2)

                    printf(
                        " %d->%d ", i + 1, j + 1);
            }

            printf(
```

```
                "\n");
        }

        getch();

        exit(0);
    }

    /* if order of matrix and number of lines is not same then its difficult
to

find the optimal solution.

Now determine the smallest uncovered element i.e. element not covered by
lines

and then subtract this minimum element from all uncovered elements and add
the

same elements at the intersection of horizontal and vertical lines.*/
    else

    {
        for (i = 0; i < n; i++)

        {
            rf[i] = 0, sr[i] = 0;

            cf[i] = 0, sc[i] = 0;
        }

        for (k = n; (k > 0 && noz != 0); k--)

        {
            for (i = 0; i < n; i++)

            {
                m = 0;

                for (j = 0; j < n; j++)

                {
                    if ((flag[i][j] == 4) && (cost[i][j] == 0))

                        m++;
                }

                sr[i] = m;
            }

            for (i = 0; i < n; i++)

            {
                if (nrz1[i] == k && nrz1[i] != sr[i])

                {
```

```
            rf[i] = 1;

            for (j = 0; j < n; j++)

            {
                if (cost[i][j] == 0)

                    flag[i][j] = 4;
            }

            noz = noz - k;
        }
    }

    for (i = 0; i < n; i++)

    {

        l = 0;

        for (j = 0; j < n; j++)

        {
            if ((flag[j][i] == 4) && (cost[j][i] == 0))

                l++;
        }

        sc[i] = l;
    }

    for (i = 0; i < n; i++)

    {
        if (ncz1[i] == k && ncz1[i] != sc[i])

        {
            cf[i] = 1;

            for (j = 0; j < n; j++)

            {
                if (cost[j][i] == 0)

                    flag[j][i] = 4;
            }

            noz = noz - k;
        }
    }

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)
```

```
            {
                if (flag[i][j] != 3)
                {
                    if (rf[i] == 1 && cf[j] == 1)

                    {
                        flag[i][j] = 3;

                        if (cost[i][j] == 0)

                            noz = noz + 1;
                    }
                }
            }
        }
    }

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)

        {
            if (rf[i] != 1 && cf[j] != 1)

            {
                if (min > cost[i][j])

                    min = cost[i][j];
            }
        }
    }

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)

        {
            if (rf[i] != 1 && cf[j] != 1)

                cost[i][j] = cost[i][j] - min;
        }
    }

    for (i = 0; i < n; i++)

    {
        for (j = 0; j < n; j++)
        {
            if (flag[i][j] == 3)

                cost[i][j] = cost[i][j] + min;
        }
    }
}
```

```
    printf(
        "\n\n");

    if (counter < 10)

    {

        counter = counter + 1;

        printf(
            "\n\nIntermediate Matrix: \n");

        for (i = 0; i < n; i++)

        {

            for (j = 0; j < n; j++)

                printf(
                    "\t%d\t", cost[i][j]);

            printf(
                "\n");
        }
    }

    else

    {

        printf(
            "\n\nOptimal solution to given problem is not possible");

        getch();

        return 0;
    }

    goto repeatx;
}
```



Conclusion: We successfully studied the use of Branch and Bound for Travelling Salesman Problem.