LAPTOP PRICE PRIEDICTOR USING ML

Anjan Das

(20103017)

(Department of Computer Science)
Dr. B R Ambedkar national
Institute of technology
Jalandhar, Punjab

Anjand.cs.20@nitj.ac.in

Arav Khandal

(20103028)
(Department of Computer Science)
Dr. B R Ambedkar national
Institute of technology
Jalandhar, Punjab

Aravk.cs.20@nitj.ac.in

Devendra Chouhan

(20103049)
(Department of Computer Science)
Dr. B R Ambedkar national
Institute of technology
Jalandhar, Punjab

Devendra.cs.20@nitj.ac.in

IT industry is in the expansion phase, more and more people are continuously shifting towards IT sector. In this high-tech era, laptop has turned into a necessity. Gone are the days when laptop was considered a luxury item.

However even though people know the speciation and the purpose of purchasing a laptop but budget has ways been a contributing factor. People tend to buy the best item in the lowest price possible of course with due consideration to the purpose of purchase.

This is why we decided to make a project of predicting the price of laptop based on user's input of its need and specification.

I. INTRODUCTION AND MOTIVATION

n our daily lives, the laptop has become one of the most important and frequently used items. However, I'm sure you've hard to find a personal laptop that meets your requirements? With so many specifications and brand names on the market, it's tough for laptop manufacturers to sell their products and for users to choose which laptop to buy.

Why should we be concerned about this issue? In 2019, it is estimated that 166 million laptops were sold and shipped (International Data Corporation). The average selling price of a laptop is approximately $629 USD. According to Grand View Research, the global laptop market is estimated to be worth over $100 billion, with a forecasted market value of 108.9 billion by 2025. Laptops will continue to be in high demand due to rising consumer buying power and increased desire for technologically advanced products. People and manufacturers alike will need to be aware and competitive as laptops become more popular around the world.

II. Project Motivation:-

1. Determine which of the three kinds of laptop contributes the most in terms of quantity and quality.

2. Research laptop trends across a variety of brands, models, and performance levels.

3. Estimate laptop pricing based on quantity and quality factors such as memory, storage size and kind, CPU and GPU performance, brand, type, and screen size.

4. The KNN Recommendation System recommends a laptop based on its specifications.

III. AIM :-

Through the proposed project, we aim to create a working machine learning model that streamlines the process of assessing the price of laptops and predict their price .

Machine learning is a very powerful tool when it comes to predicting and analyzing large amounts of data, which can be used to create AI powered systems.

IV. TERMINOLOGY

1. Company:- laptop manufacturing companies manufacture many types and many cost of laptop, Apple, Dell, HP, Acer, Asus, Lenovo etc.

2. Screen Size:- Standard laptop models usually come with a built-in **13-inch to 15-inch** wide-

screen LCD with a 1280 × 800 dot or 1366 × 768 dot resolution. This level of resolution is good enough for basic Windows operation Standard laptop models often include a built-in 13-inch to 15-inch wide-screen LCD with a resolution of 1280 800 dots or 1366 768 dots. Although this resolution is adequate for basic Windows activities, a large, high-resolution computer screen is unquestionably more user-friendly.s, but it goes without saying that a large, high-resolution computer screen is much more user-friendly.

3. **Screen Resolution:-** Horizontal and vertical pixel counts are used to indicate display resolution. 1366-by-768 (also known as HD) and 1920-by-1080 (sometimes known as Full HD) are the most often utilised resolutions on laptops and 2-in-1 PCs currently (Full HD or 1080p). In our opinion, the most ideal screen resolution for laptops is 1920-by-1080.

4. **CPU(Central Processing Unit):-** All kinds of computing devices such as tablets, PCs, or laptops feature a brain-like unit called the central processing unit or CPU. Your computer's CPU calculates and interprets instructions while you're surfing the web, creating documents, playing games, or running software programs. It's a critical component that your PC can't function without.

We'll look at why a CPU is so crucial and how to keep track of your CPU utilisation to ensure optimal system performance in the sections below.

5. **RAM**(**Random Access Memory**):- Random-access memory (RAM) stands for random-access memory, but what exactly does that imply? The RAM in your computer is simply short-term memory that stores data as the processor requires it. This should not be mistaken with long-term data stored on your hard disc, which remains accessible even after your computer has been switched off.

6. **GPU(Graphics processing unit):-** A customised processor that was created to speed up the rendering of graphics. GPUs can process a large amount of data at once, making them ideal for machine learning, video editing, and gaming.

7. **Memory:**- Memory is the part of your computer that allows you to store and retrieve data over an extended period of time. Storage is usually in the form of a solid-state drive (SSD) or a hard disc (HDD).

8. **Operating System**:- The operating system (OS) is the software that controls the software, hardware, and resources of a computer. The operating system is required to organise shared functions and to offer a user interface for interacting with software and hardware.

9. **Weight of Laptop:-** Laptops can be as light as 2 pounds or as heavy as 8 pounds. They are usually classified into one of five groups: Ultrabook /Chromebook (2–3 pounds), Ultraportable (2–5 pounds), Thin and Light (3–6 pounds), Desktop Replacement (4 pounds), and Luggables (8 pounds) are the different types of laptops (8 lbs).

V. PREVIOUS WORK:-

E-commerce websites use a similar type of ML model to sort products based on the user's requirements and availability of products on their website, as well as generate results based on the user's previous search history on the site, providing better results for the user by utilising decision tree algorithms and many other algorithm combinations.

VI. PROPOSED SOLUTION:-

We have developed a model that predicts the price of the laptop based on the requirement of the user i.e ram , processor , graphics card , screen size , resolution audio quality

The feature act as the independent variable for the model that decides the price of the laptop mainly (price may vary depending on the brand)

A. Algorithms used:-

- **ColumnTransformer:-** ColumnTransformer is a scikit-learn Python machine learning library class that allows you to apply data preparation transformations selectively.

It allows you to apply a certain transform or series of transforms to only the numerical columns, and another sequence of transforms to only the category columns, for example.

You must give a list of transformers to utilise the ColumnTransformer.

Each transformer is a three-element tuple that specifies the transformer's name, the transform to be applied, and the column indices to which it should be applied.

- **Pipeline:-** The pipeline is a scikit-learn tool for coordinating machine learning activities written in Python.

Pipelines work by connecting a linear succession of data transformations, resulting in a quantifiable modelling process.

The goal is to ensure that all stages of the pipeline, such as training datasets and each fold in the cross-validation process, are confined to the data available for the evaluation.

- **OneHotEncoder:-** On categorical data, most present machine learning methods are ineffective. Rather, categorical data must be translated to numerical data first. One of the strategies used to make this conversion is one-hot encoding. This strategy is typically utilised when applying deep learning techniques to sequential classification tasks.

The representation of categorical variables as binary vectors is known as one-hot encoding. First, the categorical values are converted to integers. Each integer value is then represented as an all-zero binary vector (except the index of the integer which is marked as 1).

- **LinearRegression:-** The first machine learning method that any data scientist encounters is linear regression. It's a simple model, but it's one that everyone should learn because it's the cornerstone for all other machine learning algorithms.

It is a very effective strategy for determining the elements that determine profitability. It may be used to forecast sales in the following months by looking at previous month's sales data. It may also be used to learn more about how customers behave. By the conclusion of the blog, we will have created a model that looks like the one below, i.e., we will have determined the optimum line to suit the data.

This is the first post in a series on machine learning that I'll be covering. The sheer volume of material regarding machine learning algorithms on the internet might be overwhelming. This site has a dual role for me. It can serve as a starting point for anyone interested in machine learning, as well as a reference for myself.

- **KNeighborsRegressor:-** The K nearest neighbours method stores all available examples and predicts the numerical target using a similarity metric (e.g., distance functions). KNN has been utilised as a non-parametric approach in statistical estimates and pattern recognition since the early 1970s. Algorithm Calculating the average of the numerical goal of the K nearest neighbours is a straightforward implementation of KNN regression. An inverse distance weighted average of the K closest neighbours is another method. The same distance functions are used in KNN regression as in KNN classification**.**

Only continuous variables are eligible for the following three distance measurements. In the case of categorical variables, the Hamming distance must be used, which is a measure of the number of times similar symbols in two strings of equal length differ.

A single neighbour forecast is just the target value of the nearest neighbour.

- **DecisionTreeRegressor:-** A decision tree is a model of decisions and all of their possible consequences, including outcomes, input costs, and utility, that employs a flowchart-like tree structure.

The supervised learning methods include the decision-tree algorithm. It may be used with both continuous and categorical output variables.

Decision tree regression examines an object's characteristics and trains a model in the shape of a tree to forecast future data and create meaningful continuous output. The output/result is not discrete, in the sense that it is not represented solely by a discrete, known set of numbers or values.

A weather prediction model that forecasts whether or not it will rain on a specific day is an example of a discrete output.

A profit prediction model that specifies the likely profit that may be earned from the sale of a product is an example of continuous output.

- **RandomForestRegressor:-** Every decision tree has a significant variance, but when we mix all of them in parallel, the final variance is minimal since each decision tree is completely trained on that specific sample data, and so the outcome is dependent on numerous decision trees rather than one. In the event of a classification problem, the majority voting classifier is used to determine the final result. The ultimate result in a regression problem is the average of all the outputs. Aggregation is the name for this section.

Random Forest is an ensemble approach that uses several decision trees with a technique called Bootstrap and Aggregation, sometimes known as bagging, to solve both regression and classification problems. Instead than depending on individual decision trees, the main idea is to aggregate numerous decision trees to determine the final outcome.

As a fundamental learning model, Random Forest uses several decision trees. Row and feature sampling are done at random from the dataset to create sample datasets for each model. Bootstrap is the name of this section.

- **GradientBoostingRegressor:-** Gradient boosting is a strong machine learning approach for ensembles.

It's commonly utilised in winning solutions to machine learning contests, such as those on Kaggle, for structured predictive modelling issues like classification and regression on tabular data.

There are several gradient boosting implementations available, including conventional SciPy versions and efficient third-party libraries. Each has its own user interface and even names for the algorithm.

This tutorial will show you how to utilise gradient boosting models in Python for classification and regression.

For the four primary implementations of gradient boosting in Python, standardised code samples are supplied, ready for you to copy-paste and apply in your own predictive modelling project.

- **AdaBoostRegressor:-** Boosting is a type of ensemble machine learning technique that combines the predictions of several weak learners.

A weak learner is a model that is relatively basic but has some ability on the dataset. The AdaBoost (adaptive boosting) algorithm was the first effective attempt to the notion.

As weak learners, the AdaBoost method employs extremely small (one-level) decision trees that are introduced progressively to the ensemble. Each succeeding model in the chain seeks to correct the predictions produced by the model preceding it. This is accomplished by balancing the training dataset to focus more on training cases where previous models failed to predict correctly.

- **ExtraTreesRegressor:-** Extra Trees is a machine learning approach that combines the predictions of many decision trees into a single forecast.

It has something in common with the commonly used random forest algorithm. Although it employs a simpler approach to create the decision trees used as members of the ensemble, it may frequently yield similar or better results than the random forest algorithm.

It's also simple to use, as it just has a few important hyperparameters and logical heuristics for tuning them.

- **SVM:-** SVM stands for Support Vector Machine and is one of the most widely used Supervised Learning algorithms for Classification and Regression issues. However, it is mostly utilised in Machine Learning for Classification difficulties.

The SVM algorithm's purpose is to find the optimum line or decision boundary for categorising n-dimensional space into classes so that additional data points may be readily placed in the proper category in the future. A hyperplane is the name for the optimal choice boundary.

The extreme points/vectors that assist create the hyperplane are chosen via SVM. Support vectors are the extreme instances, and the method is called a Support Vector Machine.

- **XGBRegressor:-** Extreme Gradient Boosting (XGBoost) is an open-source toolkit that implements the gradient boosting technique in an efficient and effective manner.

XGBoost became the go-to approach and frequently the primary component in winning solutions for a variety of issues in machine learning contests shortly after its creation and initial release.

Predicting a numerical value, such as a dollar amount or a height, is an issue in regression predictive modelling. For regression predictive modelling, XGBoost may be utilised directly.

-

B. Library used:-

- **numpy:-** NumPy is a Python package that allows you to interact with arrays.

It also provides functions for working with matrices, fourier transforms, and linear algebra.

Travis Oliphant invented NumPy in 2005. It is an open source project that you are free to use.

Numerical Python is referred to as NumPy.

We have lists in Python that act as arrays, however they are sluggish to parse. NumPy intends to deliver a 50-fold quicker array object than ordinary Python lists. The array object in NumPy is named ndarray, and it comes with a number of helper methods that make dealing with it a breeze. In data research, when speed and resources are critical, arrays are widely employed.

- **Pandas:-** Pandas is a Python data analysis package. Wes McKinney created pandas in 2008 to fill a demand for a strong and versatile quantitative analysis tool. It has now evolved to become one of the most used Python tools. It has an incredibly active contributor community.

Pandas is based on two key Python libraries: matplotlib for data visualisation and NumPy for arithmetic operations. Pandas acts as a wrapper around these libraries, enabling you to use fewer lines of code to access many of matplotlib's and NumPy's functions. Pandas'.plot(), for example, integrates numerous matplotlib methods into a single function, allowing you to plot a chart in only a few lines. Most analysts utilised Python for data munging and preprocessing before switching to a more domain specialised language like pandas.

Pandas introduced two new types of data storage objects: Series, which have a list-like structure, and DataFrames, which have a tabular structure, which make analytical jobs easier and eliminate the need to switch tools.

- **Matplotlib and Pyplot:-** Matplotlib is a data visualisation and graphical charting package for Python and its numerical extension NumPy that works across platforms. As a result, it provides an open source alternative to MATLAB. Matplotlib's APIs (Application Programming Interfaces) may also be used to incorporate charts in GUI programmes.

In most cases, a Python matplotlib script is constructed so that only a few lines of code are necessary to produce a visual data plot. Two APIs are overlaid by the matplotlib scripting layer:

Matplotlib is at the top of the pyplot API hierarchy of Python code objects.

pyplot is an OO (Object-Oriented) API collection of objects that can be constructed more easily than pyplot. This API allows you to use Matplotlib's backend layers directly.

- **Seaborn:-** Seaborn is a Python module for creating statistical visuals. It is based on matplotlib and tightly interacts with pandas data structures.

Seaborn assists you in exploring and comprehending your data. Its charting functions work with dataframes and arrays containing whole datasets, doing the necessary semantic mapping and statistical aggregation internally to generate useful graphs. Its dataset-oriented, declarative API allows you to

concentrate on the meaning of your charts rather than the mechanics of drawing them.

- **Sklearn:-** In Python, Scikit-learn (Sklearn) is the most usable and robust machine learning package. It uses a Python consistency interface to deliver a set of fast machine learning and statistical modelling capabilities, such as classification, regression, clustering, and dimensionality reduction. NumPy, SciPy, and Matplotlib are used extensively in this Python-based toolkit.

- **Pickle:-** For serialising and de-serializing Python object structures, the pickle package is utilised. Pickling, serialisation, flattening, or marshalling is the process of converting any type of Python object (list, dict, etc.) into byte streams (0s and 1s). Unpickling is a technique that turns the byte stream (produced by pickling) back into Python objects.

Pickling and unpickling are widely used in real-world scenarios because they allow us to effortlessly move data from one server/system to another and then save it in a file or database.

It's best not to unpickle data from an untrustworthy source since it might compromise your security. The pickle module, on the other hand, has no method of knowing or raising an alarm when pickingle harmful data.

C. Solution:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('laptop_data.csv')
df.head()
df.shape
df.info()
df.duplicated().sum()
df.drop(columns=['Unnamed: 0'],inplace=True)
df.head()
df['Ram'] = df['Ram'].str.replace('GB','')
df['Weight'] = df['Weight'].str.replace('kg','')
df.head()
df['Ram'] = df['Ram'].astype('int32')
df['Weight'] = df['Weight'].astype('float32')
df.info()
import seaborn as sns
sns.distplot(df['Price'])
df['Company'].value_counts().plot(kind='bar')
sns.barplot(x=df['Company'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
df['TypeName'].value_counts().plot(kind='bar')
sns.barplot(x=df['TypeName'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
sns.distplot(df['Inches'])
sns.scatterplot(x=df['Inches'],y=df['Price'])
df['ScreenResolution'].value_counts()
df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:1 if 'Touchscreen' in x else 0)
df.sample(5)
df['Touchscreen'].value_counts().plot(kind='bar')
sns.barplot(x=df['Touchscreen'],y=df['Price'])
df['Ips'] = df['ScreenResolution'].apply(lambda x:1 if 'IPS' in x else 0)
df.head()
df['Ips'].value_counts().plot(kind='bar')
sns.barplot(x=df['Ips'],y=df['Price'])
new = df['ScreenResolution'].str.split('x',n=1,expand=True)
df['X_res'] = new[0]
df['Y_res'] = new[1]
df.sample(5)
df['X_res'] = df['X_res'].str.replace(',','').str.findall(r'(\d+\.?\d+)').apply(lambda x:x[0])
df.head()
df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')
df.info()
df.corr()['Price']
df['ppi'] = (((df['X_res']**2) + (df['Y_res']**2))**0.5/df['Inches']).astype('float')
df.corr()['Price']
df.drop(columns=['ScreenResolution'],inplace=True)
```

```
df.head()
df.drop(columns=['Inches','X_res','Y_re
s'],inplace=True)
df.head()
df['Cpu'].value_counts()
df['Cpu Name'] = df['Cpu'].apply(lambda
 x:" ".join(x.split()[0:3]))
df.head()
def fetch_processor(text):
    if text == 'Intel Core i7' or text
== 'Intel Core i5' or text == 'Intel Co
re i3':
        return text
    else:
        if text.split()[0] == 'Intel':
            return 'Other Intel Process
or'
        else:
            return 'AMD Processor'
df['Cpu brand'] = df['Cpu Name'].apply(
fetch_processor)
df.head()
df['Cpu brand'].value_counts().plot(kin
d='bar')
sns.barplot(x=df['Cpu brand'],y=df['Pri
ce'])
plt.xticks(rotation='vertical')
plt.show()
df.drop(columns=['Cpu','Cpu Name'],inpl
ace=True)
df.head()
df['Ram'].value_counts().plot(kind='bar
')
sns.barplot(x=df['Ram'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
df['Memory'].value_counts()
df['Memory'] = df['Memory'].astype(str)
.replace('\.0', '', regex=True)
df["Memory"] = df["Memory"].str.replace
('GB', '')
df["Memory"] = df["Memory"].str.replace
('TB', '000')
new = df["Memory"].str.split("+", n = 1
, expand = True)

df["first"]= new[0]
df["first"]=df["first"].str.strip()

df["second"]= new[1]

df["Layer1HDD"] = df["first"].apply(lam
bda x: 1 if "HDD" in x else 0)
df["Layer1SSD"] = df["first"].apply(lam
bda x: 1 if "SSD" in x else 0)
df["Layer1Hybrid"] = df["first"].apply(
lambda x: 1 if "Hybrid" in x else 0)
df["Layer1Flash_Storage"] = df["first"]
.apply(lambda x: 1 if "Flash Storage" i
n x else 0)

df['first'] = df['first'].str.replace(r
'\D', '')

df["second"].fillna("0", inplace = True
)

df["Layer2HDD"] = df["second"].apply(la
mbda x: 1 if "HDD" in x else 0)
df["Layer2SSD"] = df["second"].apply(la
mbda x: 1 if "SSD" in x else 0)
df["Layer2Hybrid"] = df["second"].apply
(lambda x: 1 if "Hybrid" in x else 0)
df["Layer2Flash_Storage"] = df["second"
].apply(lambda x: 1 if "Flash Storage"
in x else 0)

df['second'] = df['second'].str.replace
(r'\D', '')

df["first"] = df["first"].astype(int)
df["second"] = df["second"].astype(int)

df["HDD"]=(df["first"]*df["Layer1HDD"]+
df["second"]*df["Layer2HDD"])
df["SSD"]=(df["first"]*df["Layer1SSD"]+
df["second"]*df["Layer2SSD"])
df["Hybrid"]=(df["first"]*df["Layer1Hyb
rid"]+df["second"]*df["Layer2Hybrid"])
df["Flash_Storage"]=(df["first"]*df["La
yer1Flash_Storage"]+df["second"]*df["La
yer2Flash_Storage"])

df.drop(columns=['first', 'second', 'La
yer1HDD', 'Layer1SSD', 'Layer1Hybrid',
       'Layer1Flash_Storage', 'Layer2HD
D', 'Layer2SSD', 'Layer2Hybrid',
       'Layer2Flash_Storage'],inplace=T
rue)
df.sample(5)
```

```
df.drop(columns=['Memory'],inplace=True
)
df.head()
df.corr()['Price']
df.drop(columns=['Hybrid','Flash_Storag
e'],inplace=True)
df.head()
df['Gpu'].value_counts()
df['Gpu brand'] = df['Gpu'].apply(lambd
a x:x.split()[0])
df.head()
df['Gpu brand'].value_counts()
df = df[df['Gpu brand'] != 'ARM']
df['Gpu brand'].value_counts()
sns.barplot(x=df['Gpu brand'],y=df['Pri
ce'],estimator=np.median)
plt.xticks(rotation='vertical')
plt.show()
df.drop(columns=['Gpu'],inplace=True)
df.head()
df['OpSys'].value_counts()
sns.barplot(x=df['OpSys'],y=df['Price']
)
plt.xticks(rotation='vertical')
plt.show()
def cat_os(inp):
    if inp == 'Windows 10' or inp == 'W
indows 7' or inp == 'Windows 10 S':
        return 'Windows'
    elif inp == 'macOS' or inp == 'Mac
OS X':
        return 'Mac'
    else:
        return 'Others/No OS/Linux'
df['os'] = df['OpSys'].apply(cat_os)
df.head()
df.drop(columns=['OpSys'],inplace=True)
sns.barplot(x=df['os'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
sns.distplot(df['Weight'])
sns.scatterplot(x=df['Weight'],y=df['Pr
ice'])
df.corr()['Price']
sns.heatmap(df.corr())
sns.distplot(np.log(df['Price']))
X = df.drop(columns=['Price'])
y = np.log(df['Price'])
X
Y
```

```
from sklearn.model_selection import tra
in_test_split
X_train,X_test,y_train,y_test = train_t
est_split(X,y,test_size=0.15,random_sta
te=2)
X_train
from sklearn.compose import ColumnTrans
former
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHo
tEncoder
from sklearn.metrics import r2_score,me
an_absolute_error
from sklearn.linear_model import Linear
Regression,Ridge,Lasso
from sklearn.neighbors import KNeighbor
sRegressor
from sklearn.tree import DecisionTreeRe
gressor
from sklearn.ensemble import RandomFore
stRegressor,GradientBoostingRegressor,A
daBoostRegressor,ExtraTreesRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = LinearRegression()

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
```

```
],remainder='passthrough')

step2 = Ridge(alpha=10)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = Lasso(alpha=0.001)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = KNeighborsRegressor(n_neighbors
=3)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = DecisionTreeRegressor(max_depth
=8)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = RandomForestRegressor(n_estimat
ors=100,
                              random_st
ate=3,
                              max_sampl
es=0.5,
                              max_featu
res=0.75,
                              max_depth
=15)
```

```
pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = ExtraTreesRegressor(n_estimator
s=100,
                            random_st
ate=3,
                            max_sampl
es=0.5,
                            max_featu
res=0.75,
                            max_depth
=15)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = AdaBoostRegressor(n_estimators=
15,learning_rate=1.0)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = GradientBoostingRegressor(n_est
imators=500)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = XGBRegressor(n_estimators=45,ma
x_depth=5,learning_rate=0.5)

pipe = Pipeline([
```

```
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

from sklearn.ensemble import VotingRegr
essor,StackingRegressor

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')


rf = RandomForestRegressor(n_estimators
=350,random_state=3,max_samples=0.5,max
_features=0.75,max_depth=15)
gbdt = GradientBoostingRegressor(n_esti
mators=100,max_features=0.5)
xgb = XGBRegressor(n_estimators=25,lear
ning_rate=0.3,max_depth=5)
et = ExtraTreesRegressor(n_estimators=1
00,random_state=3,max_samples=0.5,max_f
eatures=0.75,max_depth=10)

step2 = VotingRegressor([('rf', rf), ('
gbdt', gbdt), ('xgb',xgb), ('et',et)],w
eights=[5,1,1,1])

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))


step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = AdaBoostRegressor(n_estimators=
15,learning_rate=1.0)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = GradientBoostingRegressor(n_est
imators=500)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))
```

```
step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = XGBRegressor(n_estimators=45,ma
x_depth=5,learning_rate=0.5)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

from sklearn.ensemble import VotingRegr
essor,StackingRegressor

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')


rf = RandomForestRegressor(n_estimators
=350,random_state=3,max_samples=0.5,max
_features=0.75,max_depth=15)
gbdt = GradientBoostingRegressor(n_esti
mators=100,max_features=0.5)
xgb = XGBRegressor(n_estimators=25,lear
ning_rate=0.3,max_depth=5)
et = ExtraTreesRegressor(n_estimators=1
00,random_state=3,max_samples=0.5,max_f
eatures=0.75,max_depth=10)

step2 = VotingRegressor([('rf', rf), ('
gbdt', gbdt), ('xgb',xgb), ('et',et)],w
eights=[5,1,1,1])
```

```
pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
print('MAE',mean_absolute_error(y_test,
y_pred))

from sklearn.ensemble import VotingRegr
essor,StackingRegressor

step1 = ColumnTransformer(transformers=
[
    ('col_tnf',OneHotEncoder(sparse=Fal
se,drop='first'),[0,1,7,10,11])
],remainder='passthrough')


estimators = [
    ('rf', RandomForestRegressor(n_esti
mators=350,random_state=3,max_samples=0
.5,max_features=0.75,max_depth=15)),
    ('gbdt',GradientBoostingRegressor(n
_estimators=100,max_features=0.5)),
    ('xgb', XGBRegressor(n_estimators=2
5,learning_rate=0.3,max_depth=5))
]

step2 = StackingRegressor(estimators=es
timators, final_estimator=Ridge(alpha=1
00))

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred
))
```

```
print('MAE',mean_absolute_error(y_test,
y_pred))

import pickle

pickle.dump(df,open('df.pkl','wb'))
pickle.dump(pipe,open('pipe.pkl','wb'))
df
X_train
```

VII. Experimental Setup:-

For implementation of machine learning we have using python.

- **Python:-** Python is a popular high-level programming language for general-purpose applications. Guido van Rossum designed it in 1991, and the Python Software Foundation developed it. Its syntax lets programmers to express concepts in fewer lines of code, and it was primarily built with code readability in mind.
- **Why :-** Python's extensive library ecosystem allows it to be used in almost every IT field, and it makes it easier for programmers to design things.

  Python was created with readability in mind, making it straightforward to learn for beginners.

Its adaptability allows it to be used with other languages.

Python may run on a variety of operating systems (Windows, Mac, Linux, Raspberry Pi, etc).

Python has a syntax that is quite close to English.

Python's syntax allows programmers to build programmes in less lines than other programming languages.

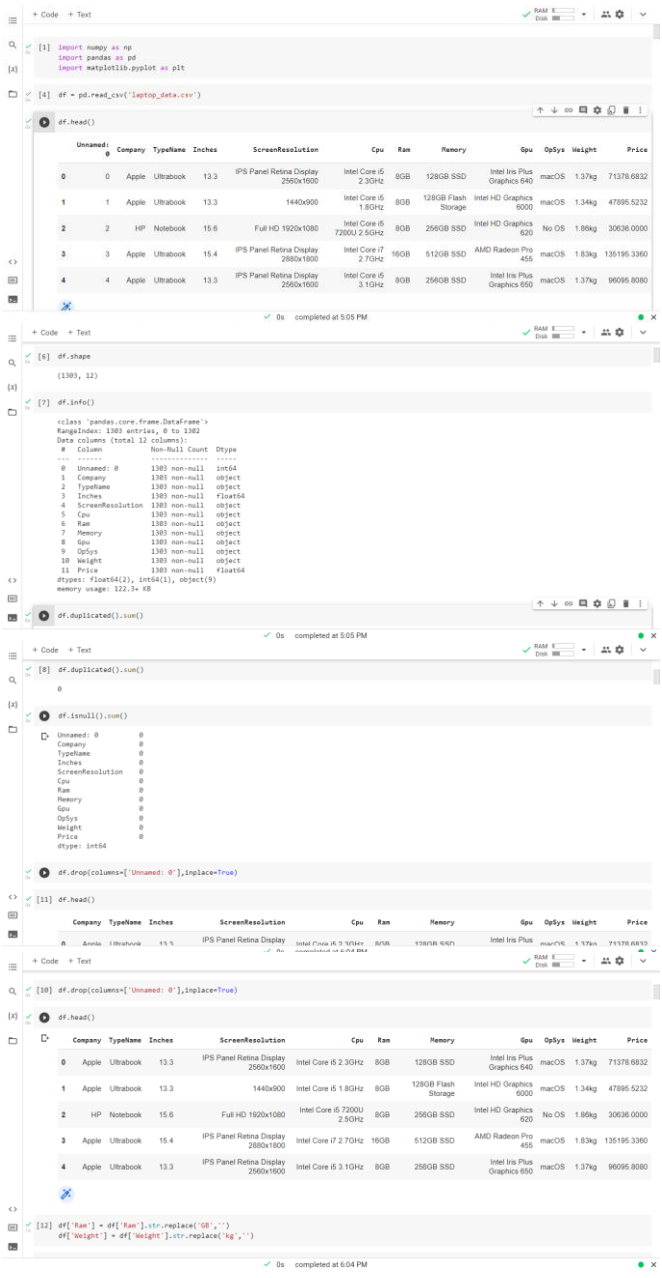
Python is an interpreter language, meaning that code may be run immediately after it is written.

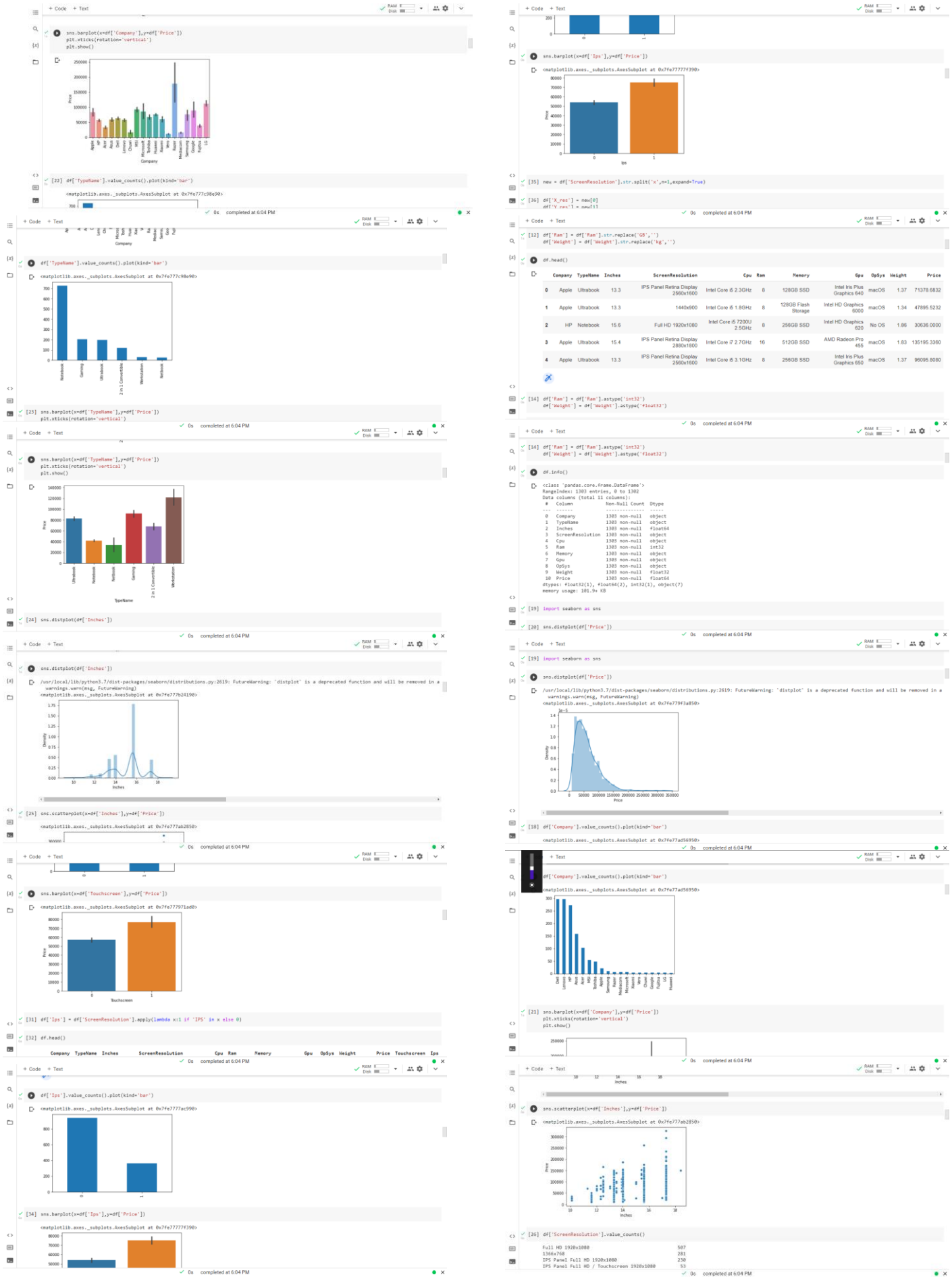
Prototyping may thus be completed quite rapidly.

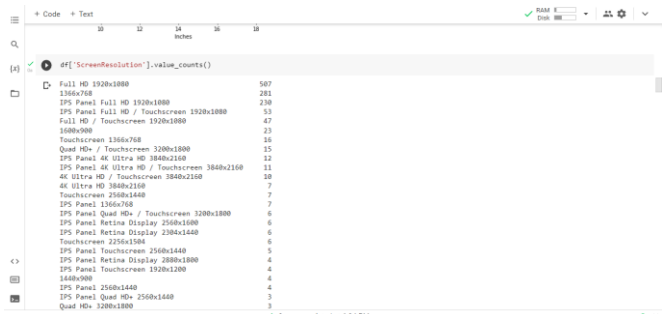
Python can be approached in three ways: procedural, object-oriented, and functional.

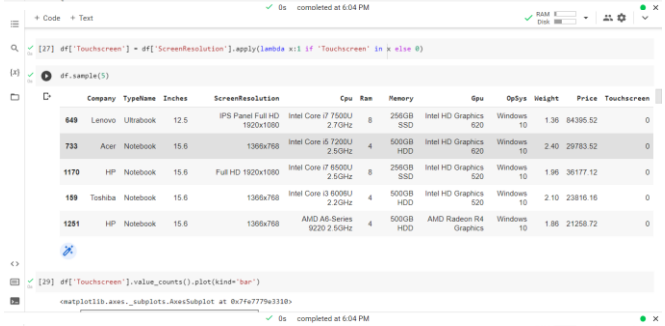
- **Google colab:-** Google Colab is a Google-provided freemium tool. For deep learning technologies, Google Colab is an excellent tool. Google Colab provides a jupyter notebook that runs entirely in the cloud. Most importantly, there is no need to set up the environment, and the notebook you create can be edited simultaneously. Many major machine learning libraries are also available in Google Colab.
- **Why:-** No installation is required: one of the benefits of using Google Colab is that you do not need to install it on your local machine; it runs entirely on the cloud, making it accessible anywhere there is an internet connection. Because it runs on a cloud-based local system, it has no effect on the Google Colab's performance. We only require an internet connection and a browser. Google Colab also has a collaboration function that allows numerous developers to work on the same project at the same time.

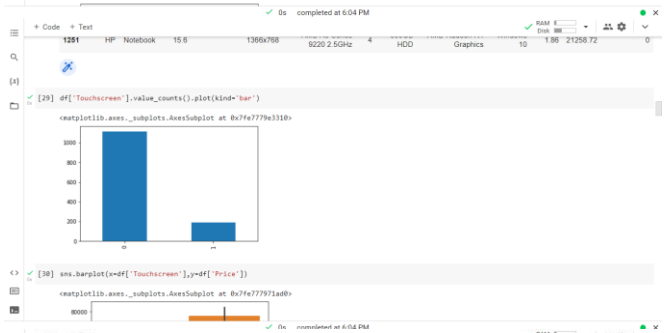
VIII. Result:-

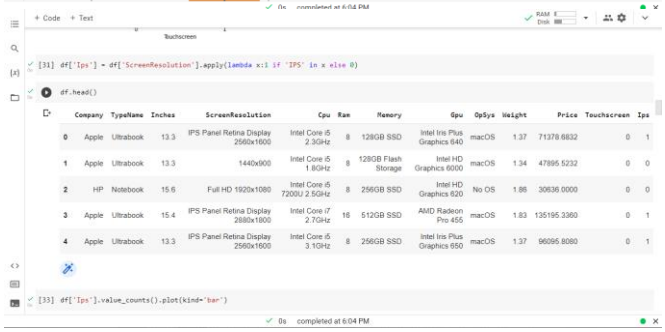


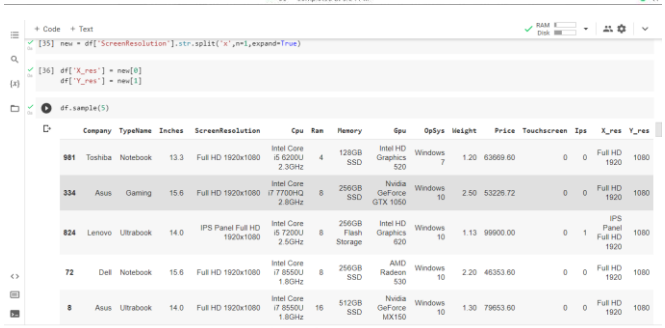


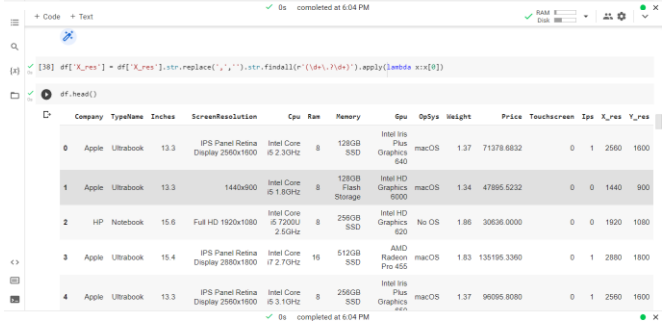

df['ScreenResolution'].value_counts()
Full HD 1920x1080 507
1366x768 281
IPS Panel Full HD 1920x1080 230
IPS Panel Full HD / Touchscreen 1920x1080 53
Full HD / Touchscreen 1920x1080 47
1600x900 23
Touchscreen 1366x768 16
Quad HD+ / Touchscreen 3200x1800 15
IPS Panel 4K Ultra HD 3840x2160 12
IPS Panel 4K Ultra HD / Touchscreen 3840x2160 11
4K Ultra HD / Touchscreen 3840x2160 10
4K Ultra HD 3840x2160 7
Touchscreen 2560x1440 7
IPS Panel 1366x768 7
IPS Panel Quad HD+ / Touchscreen 3200x1800 6
IPS Panel Retina Display 2560x1600 6
IPS Panel Retina Display 2304x1440 6
Touchscreen 2256x1504 6
IPS Panel Touchscreen 2560x1440 5
IPS Panel Retina Display 2880x1800 4
IPS Panel Touchscreen 1920x1200 4
1440x900 4
IPS Panel 2560x1440 4
IPS Panel Quad HD+ 2560x1440 3
Quad HD+ 3200x1800 3

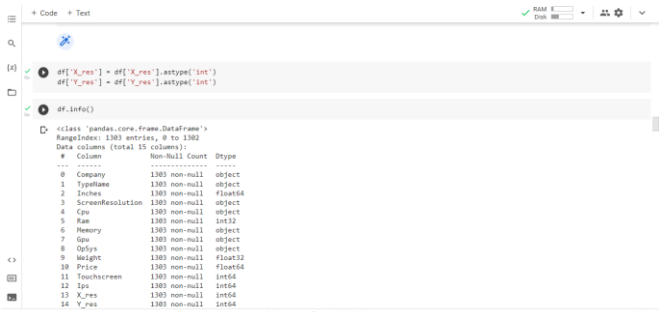

df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:1 if 'Touchscreen' in x else 0)
df.sample(5)
df['Touchscreen'].value_counts().plot(kind='bar')


df['Touchscreen'].value_counts().plot(kind='bar')
sns.barplot(x=df['Touchscreen'],y=df['Price'])

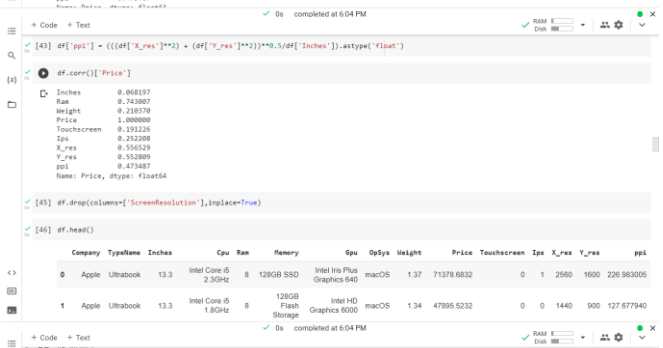

df['Ips'] = df['ScreenResolution'].apply(lambda x:1 if 'IPS' in x else 0)
df.head()
df['Ips'].value_counts().plot(kind='bar')

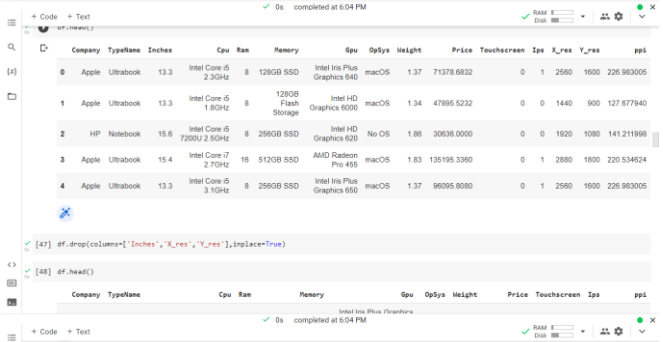

new = df['ScreenResolution'].str.split('x',n=1,expand=True)
df['X_res'] = new[0]
df['Y_res'] = new[1]
df.sample(5)

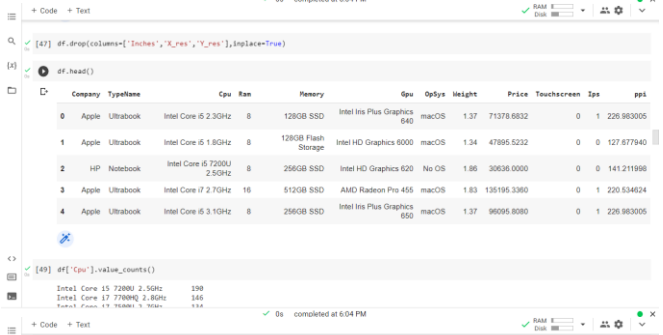

df['X_res'] = df['X_res'].str.replace(',','').str.findall(r'(\d+\.?\d+)').apply(lambda x:x[0])
df.head()

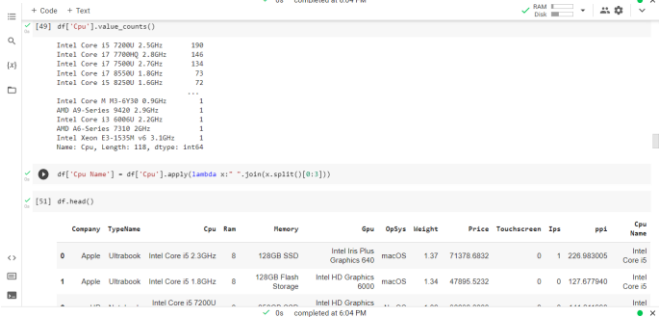

df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')
df.info()
memory usage: 142.6+ KB

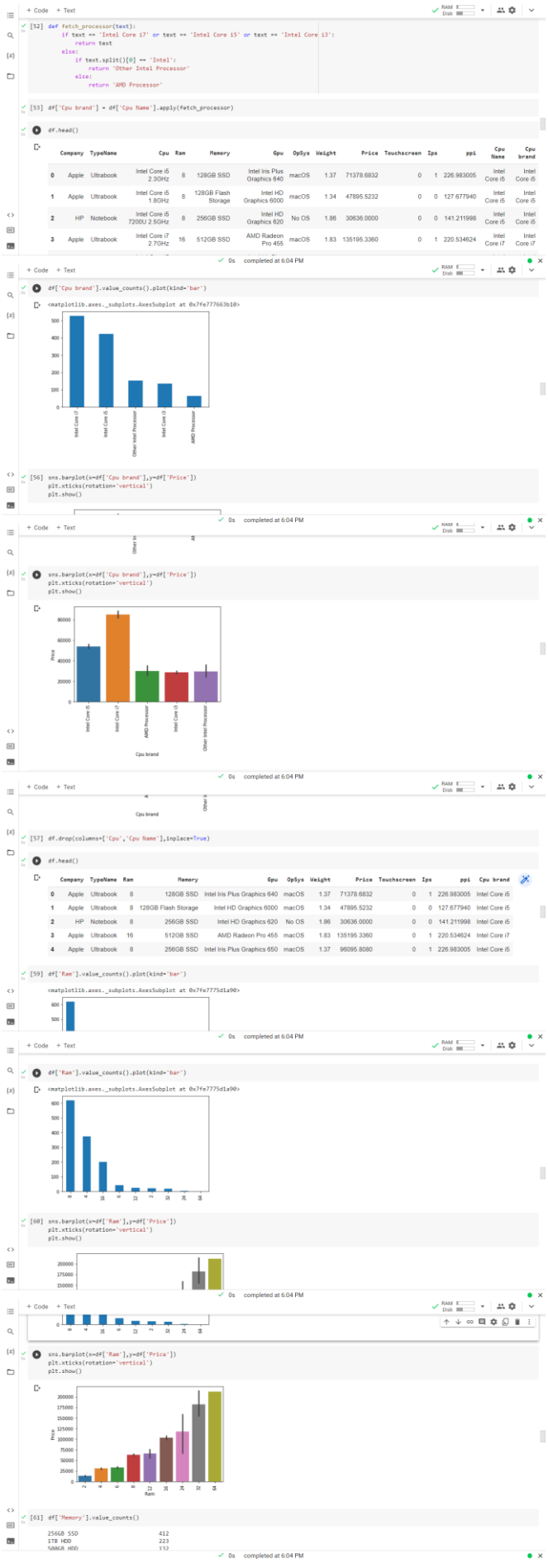

df['ppi'] = (((df['X_res']**2) + (df['Y_res']**2))**0.5/df['Inches']).astype('float')
df.corr()['Price']

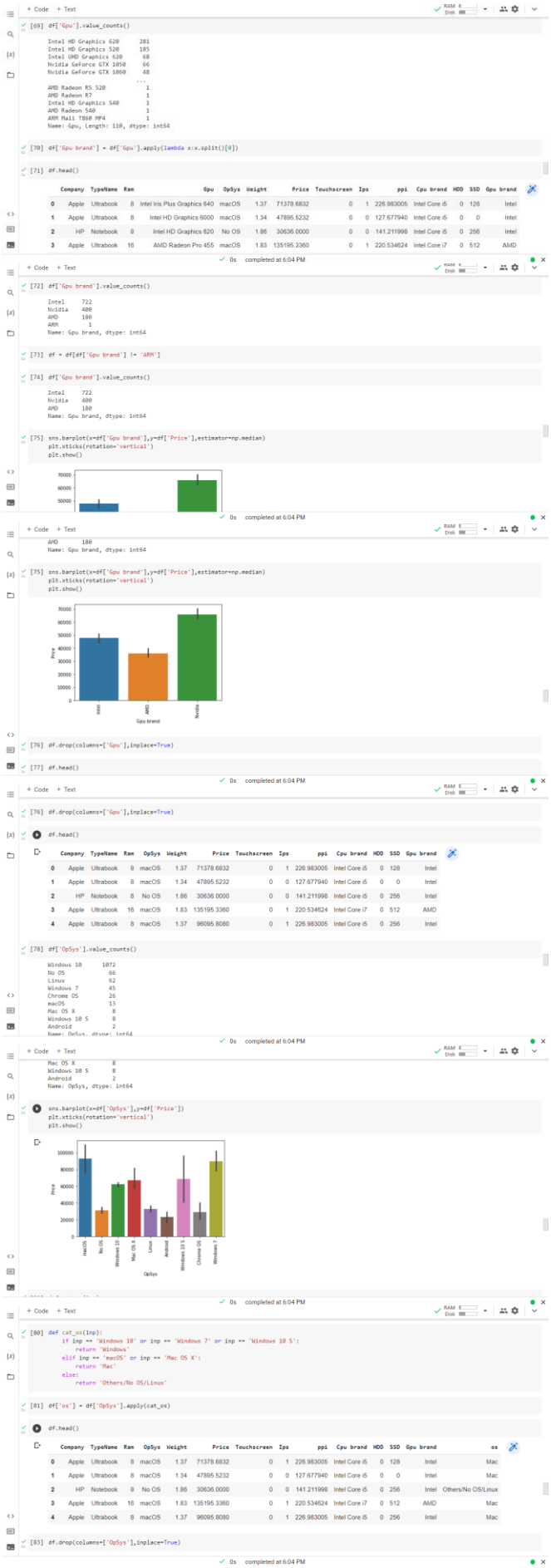

df.drop(columns=['ScreenResolution'],inplace=True)
df.head()

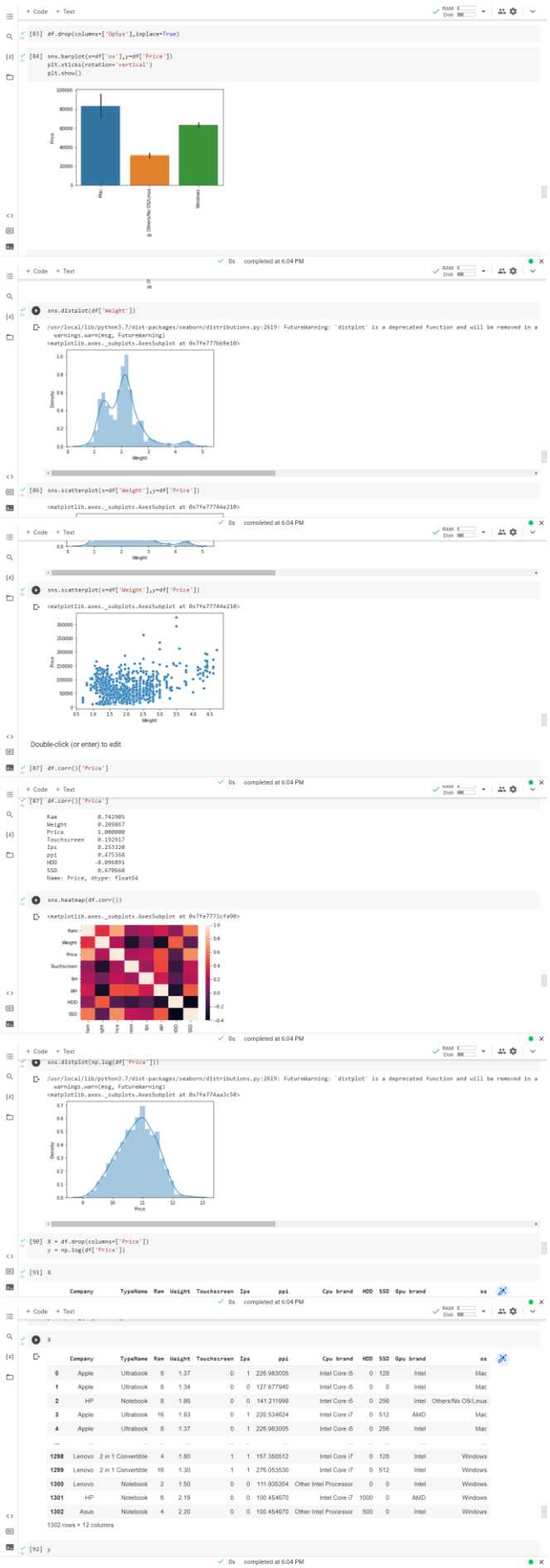

df.drop(columns=['Inches','X_res','Y_res'],inplace=True)
df.head()

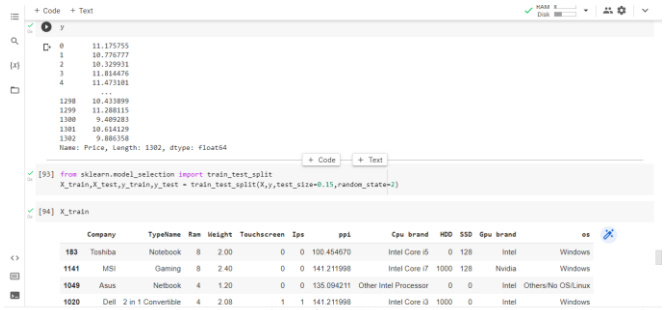

df['Cpu'].value_counts()
Intel Core i5 7200U 2.5GHz 190
Intel Core i7 7700HQ 2.8GHz 146
Intel Core i7 7500U 2.7GHz 134
Intel Core i7 8550U 1.8GHz 73
Intel Core i5 8250U 1.6GHz 72
Intel Core M M3-6Y30 0.9GHz 1
AMD A9-Series 9420 2.9GHz 1
Intel Core i3 6006U 2.2GHz 1
AMD A6-Series 7310 2GHz 1
Intel Xeon E3-1535M v6 3.1GHz 1
Name: Cpu, Length: 118, dtype: int64
df['Cpu Name'] = df['Cpu'].apply(lambda x:" ".join(x.split()[0:3]))


df.drop(columns=['Inches','X_res','Y_res'],inplace=True)
df.head()
df['Cpu'].value_counts()
df.head()







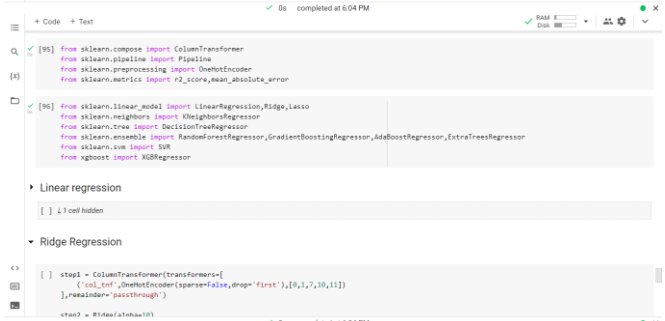


```
0       11.175755
1       10.776777
2       10.329931
3       11.814476
4       11.473101
          ...
1298    10.433899
1299    11.288115
1300     9.409283
1301    10.614129
1302     9.886358
Name: Price, Length: 1302, dtype: float64
```

```
[93] from sklearn.model_selection import train_test_split
     X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15,random_state=2)
```

```
[94] X_train
```

	Company	TypeName	Ram	Weight	Touchscreen	Ips	ppi	Cpu brand	HDD	SSD	Gpu brand	os
183	Toshiba	Notebook	8	2.00	0	0	100.454670	Intel Core i5	0	128	Intel	Windows
1141	MSI	Gaming	8	2.40	0	0	141.211998	Intel Core i7	1000	128	Nvidia	Windows
1049	Asus	Netbook	4	1.20	0	0	135.094211	Other Intel Processor	0	0	Intel	Others/No OS/Linux
1020	Dell	2 in 1 Convertible	4	2.08	1	1	141.211998	Intel Core i3	1000	0	Intel	Windows



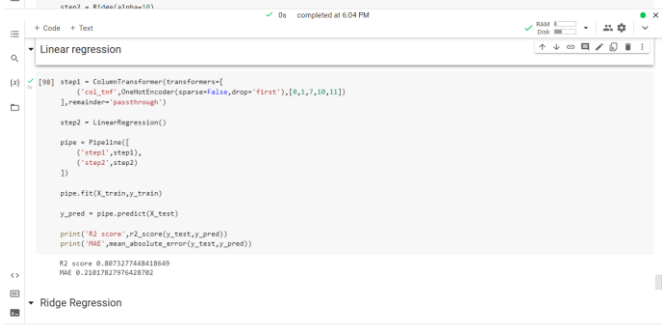
```
[95] from sklearn.compose import ColumnTransformer
     from sklearn.pipeline import Pipeline
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.metrics import r2_score,mean_absolute_error

[96] from sklearn.linear_model import LinearRegression,Ridge,Lasso
     from sklearn.neighbors import KNeighborsRegressor
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegressor,ExtraTreesRegressor
     from sklearn.svm import SVR
     from xgboost import XGBRegressor
```

Linear regression

```
[ ] ↳ 1 cell hidden
```

Ridge Regression

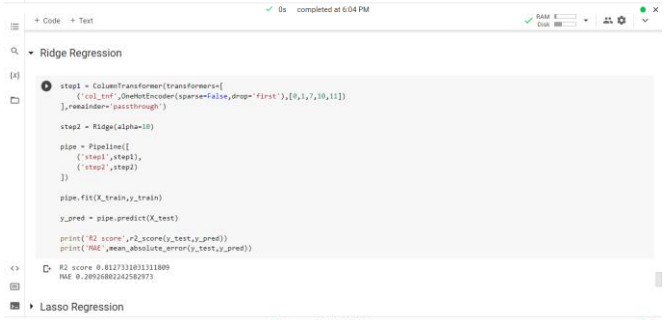
```
[ ] step1 = ColumnTransformer(transformers=[
        ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
    ],remainder='passthrough')
```



Linear regression

```
[98] step1 = ColumnTransformer(transformers=[
        ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
    ],remainder='passthrough')

     step2 = LinearRegression()

     pipe = Pipeline([
        ('step1',step1),
        ('step2',step2)
     ])

     pipe.fit(X_train,y_train)

     y_pred = pipe.predict(X_test)

     print('R2 score',r2_score(y_test,y_pred))
     print('MAE',mean_absolute_error(y_test,y_pred))

     R2 score 0.8073277448418649
     MAE 0.21017827976428702
```

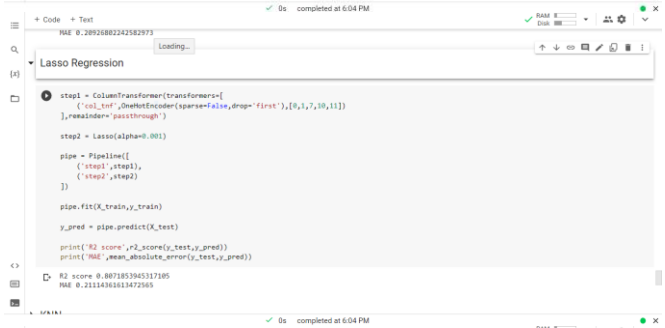
Ridge Regression



Ridge Regression

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = Ridge(alpha=10)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8127331031311809
MAE 0.20926802242582973
```

Lasso Regression



```
MAE 0.20926802242582973
```

Loading...

Lasso Regression

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = Lasso(alpha=0.001)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8071853945317105
MAE 0.21114361613472565
```

KNN

```
R2 score 0.8071853945317105
MAE 0.21114361613472565
```

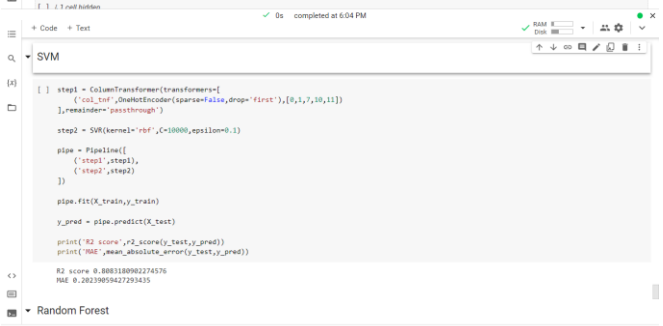
KNN

```
[ ] step1 = ColumnTransformer(transformers=[
        ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
    ],remainder='passthrough')

    step2 = KNeighborsRegressor(n_neighbors=3)

    pipe = Pipeline([
        ('step1',step1),
        ('step2',step2)
    ])

    pipe.fit(X_train,y_train)

    y_pred = pipe.predict(X_test)

    print('R2 score',r2_score(y_test,y_pred))
    print('MAE',mean_absolute_error(y_test,y_pred))

    R2 score 0.8021984680448553
    MAE 0.19319716721521116
```

Decision Tree

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = DecisionTreeRegressor(max_depth=8)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.8420668662998522
MAE 0.18085641910288333
```

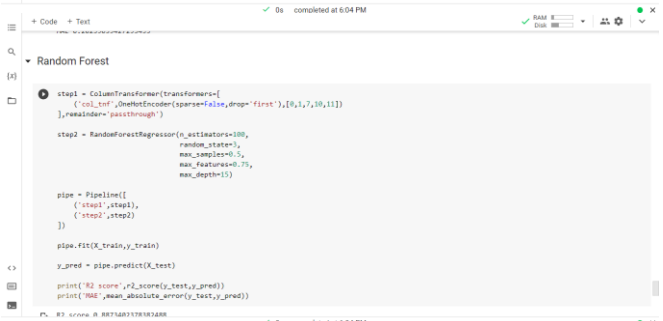
SVM



SVM

```
[ ] step1 = ColumnTransformer(transformers=[
        ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
    ],remainder='passthrough')

    step2 = SVR(kernel='rbf',C=10000,epsilon=0.1)

    pipe = Pipeline([
        ('step1',step1),
        ('step2',step2)
    ])

    pipe.fit(X_train,y_train)

    y_pred = pipe.predict(X_test)

    print('R2 score',r2_score(y_test,y_pred))
    print('MAE',mean_absolute_error(y_test,y_pred))

    R2 score 0.8083180902274576
    MAE 0.20239059427293435
```

Random Forest



Random Forest

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = RandomForestRegressor(n_estimators=100,
                              random_state=3,
                              max_samples=0.5,
                              max_features=0.75,
                              max_depth=15)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
```

```
[ ] ↳ 1 cell hidden
```

AdaBoost

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = AdaBoostRegressor(n_estimators=15,learning_rate=1.0)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

R2 score 0.7862062278529015
MAE 0.23471912900481796
```

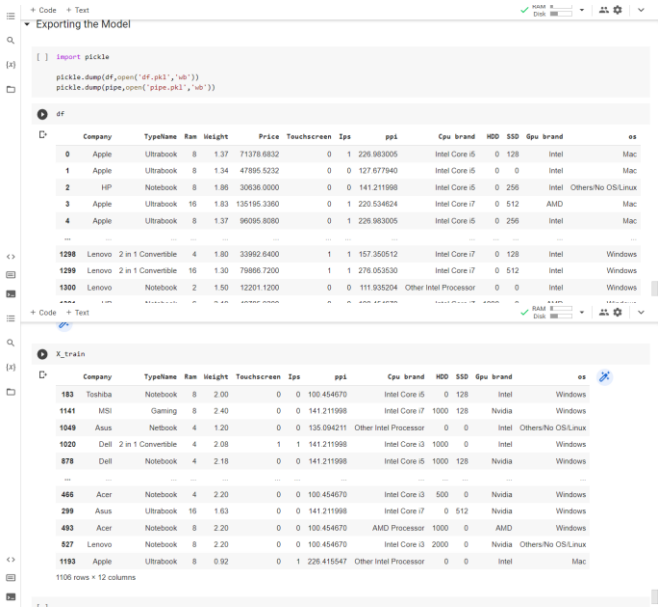
XgBoost

```
step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')

step2 = XGBRegressor(n_estimators=45,max_depth=5,learning_rate=0.5)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)

y_pred = pipe.predict(X_test)

print('R2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))

[09:26:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
R2 score 0.8811773435850243
MAE 0.1649620351260097
```

Voting Regressor

[redacted]king

```
from sklearn.ensemble import VotingRegressor,StackingRegressor

step1 = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,10,11])
],remainder='passthrough')


estimators = [
    ('rf', RandomForestRegressor(n_estimators=350,random_state=3,max_samples=0.5,max_features=0.75,max_depth=15)),
    ('gbdt',GradientBoostingRegressor(n_estimators=100,max_features=0.5)),
    ('xgb', XGBRegressor(n_estimators=25,learning_rate=0.3,max_depth=5))
]

step2 = StackingRegressor(estimators=estimators, final_estimator=Ridge(alpha=100))

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(X_train,y_train)
```



IX. Conclusion:- We have successfully **Predict Price of laptop.**

X. REFERENCES:-

https://github.com/devendra200/laptop_data

https://www.bajajfinserv.in/laptop-buying-guide

https://en.wikipedia.org/wiki/Linear_regression#:~:text=In%20statistics%2C%20linear%20regression%20is,as%20dependent%20and%20independent%20variables).

- **Prediction:-**