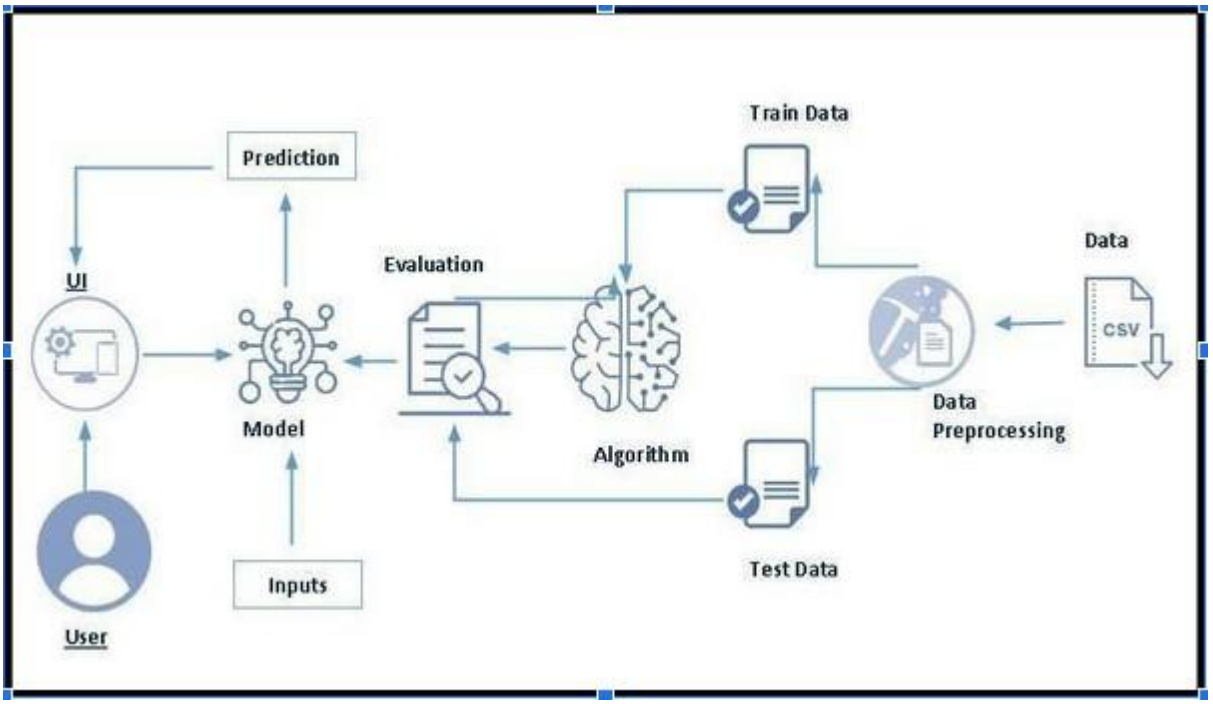
Horology 2.0: Forecasting the Future of Smartwatch Prices

Smartwatches have become increasingly popular in recent years due to their ability to provide a range of functionalities beyond traditional timekeeping, such as fitness tracking, communication, and entertainment. As with any electronic device, the cost of a smartwatch can vary significantly depending on its features, brand, and other factors.

In this project, we aim to develop a machine learning model that predicts the price of a smartwatch based on a given set of features. Our dataset contains information on various attributes of smartwatches, including brand, model, operating system, connectivity, display type and size, resolution, water resistance, battery life, heart rate monitor, GPS, NFC, and price. We will use this dataset to train and evaluate our predictive model, which has the potential to transform the way consumers and manufacturers approach pricing and purchasing decisions.

The development of an accurate smartwatch price prediction model using machine learning can have significant implications in various domains, including e-commerce, marketing, and consumer research.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact

- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation

- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis

- Collect the dataset
  - Training the model in multiple algorithms
  - Testing the model

- Performance Testing & Hyperparameter Tuning
  - Testing model with multiple evaluation metrics
  - Comparing model accuracy before & after applying hyperparameter tuning

- Model Deployment
  - Save the best model
  - Integrate with Web Framework

- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation - Step by step project development procedure

Prior Knowledge:

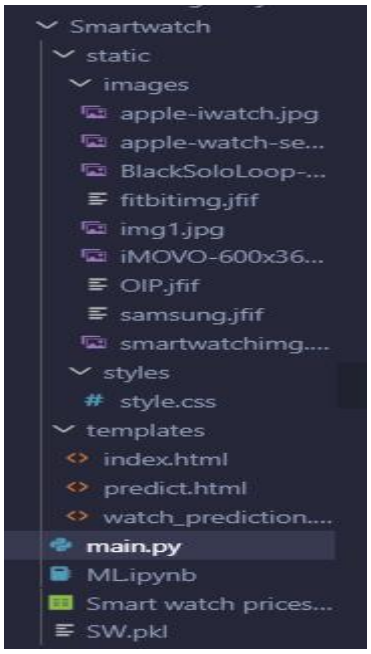
You must have prior knowledge of following topics to complete this project.

- ML Concepts
  - Linear Regression: - https://www.javatpoint.com/linear-regression-in-machine-learning
  - Decision Tree Regressor: - https://www.javatpoint.com/machine-learning-decision-tree-classification algorithm
  - Random Forest Regressor: - https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Gradient Boosting Regressor: - https://www.javatpoint.com/gbm-in-machine-learning
  - Xtreme Gradient Boost Regressor: - https://www.javatpoint.com/xgboost-ml-model-in-python

- Flask Basics: - https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create a smart watch folder which contains files as shown below:



- We are building a flask application which needs HTML pages stored in the templates folder and a python script main.py for scripting.

- SW.pkl is where our machine learning model is saved. Further we will use this model for flask integration.

Static folder contains the main css files and images which will be used to style our web application.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

The development of smartwatches in the watch industry has been fascinating, cost is still a difficult problem. The pricing of smartwatches is influenced by a number of issues and trends, making this an exciting area to research. So we need to build a Model for it.

Activity 2: Business requirements

To ensure that the smart watch price prediction model meets business requirements and can be deployed for public use, it should follow the following rules and requirements:

- **Accuracy:** The model should have a high level of accuracy in predicting smartwatch prices, with a low margin of error. This is crucial to ensure that the predictions are reliable and trustworthy.
- **Privacy and security:** The model should be developed in accordance with privacy and security regulations to protect user data. This includes ensuring that sensitive data is stored securely and implementing proper data access controls.
- **Interpretability:** The model should be interpretable, meaning that the predictions can be explained and understood by the end-users. This is important to build trust in the model and to allow users to make informed decisions based on the predictions.
- **User interface:** The model should have a user-friendly interface that is easy to use and understand. This is important to ensure that the model can be deployed for public use, even by individuals who may not have technical expertise.

Activity 3: Literature Survey

This literature review summarizes current issues and future expectations regarding smartwatch pricing:

Sustainability and Ethical Issues: The importance of sustainability is becoming when setting prices for products. The increasing awareness of ethical manufacturing practices among consumers may impact their inclination to pay higher prices for eco-friendly smartwatches.

Consumer Perception and Expectations: study explores consumer preferences and how much they are willing to pay for particular features on smartwatches. The difficulty is in keeping a competitive price point while matching product offerings to customer expectations.

Competitive Market Dynamics: Examines the competitive environment in the smartwatch industry. Companies need to strategically position their offerings to remain competitive without sacrificing quality, as there are many players in the market offering a wide range of products at different price points.

Costs of Manufacturing and the Supply Chain: Tells about how difficult it is to control manufacturing and supply chain expenses in the smartwatch sector. Pricing strategies are directly impacted by factors such as economies of scale, assembly costs, and component sourcing.

Economic Factors and Pricing Strategies: Analyses highlight how economic factors affect the cost of smartwatches. Pricing strategies are influenced by a number of factors, including global economic conditions, inflation, and currency fluctuations.

Subscription Models and Service Bundling: In order to maintain competitive pricing and create recurring revenue streams, some researchers, propose that smartwatch companies investigate subscription-based models or bundle services with hardware.

Activity 4: Social or Business Impact

Social Impact: From a social perspective, the smartwatch price prediction model can help consumers make informed purchasing decisions and save money on their electronic purchases. This is particularly important for individuals with limited financial resources who may not have the luxury of buying high-priced consumer electronics.

Business Impact: The smartwatch price prediction model can have significant implications for various industries, including e-commerce and retail. For instance, the model can help retailers optimize their pricing strategies, improve customer segmentation, and increase their profitability. Additionally, the model can assist in reducing inventory costs and minimizing losses due to overstocking.

Milestone 2: Data collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible.
So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. E.g., kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/code/ishantgargml/smart-watch-prices-analysis/input

Download the dataset from the above link. Let us understand and analyze the dataset properly using visualization techniques.

Note: There are several approaches for understanding the data. But we have applied some of it here. You can also employ a variety of techniques.

Activity 1.1: Importing the libraries

Import the libraries required for this machine learning project, as shown in the image below.

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

Activity 1.2: Read the Dataset

Our dataset format could be in .csv, excel files,.txt, .json, and so on. With the help of pandas, we can read the dataset.

Since our dataset is a csv file, we use read_csv() which is pandas function to read the dataset. As a parameter we have to give the directory of the csv file.

df.head() will display first 5 rows of the dataset.

```
df=pd.read_csv("Smart watch prices.csv")
```

```
df.head()
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size (inches)	Resolution	Water Resistance (meters)	Battery Life	Heart Rate Monitor	GPS	NFC	Price (USD)
0	Apple	Watch Series 7	watchOS	Bluetooth, Wi-Fi, Cellular	Retina	1.90	396 x 484	50	18	Yes	Yes	Yes	$399
1	Samsung	Galaxy Watch 4	Wear OS	Bluetooth, Wi-Fi, Cellular	AMOLED	1.40	450 x 450	50	40	Yes	Yes	Yes	$249
2	Garmin	Venu 2	Garmin OS	Bluetooth, Wi-Fi	AMOLED	1.30	416 x 416	50	11	Yes	Yes	No	$399
3	Fitbit	Versa 3	Fitbit OS	Bluetooth, Wi-Fi	AMOLED	1.58	336 x 336	50	6	Yes	Yes	Yes	$229
4	Fossil	Gen 6	Wear OS	Bluetooth, Wi-Fi	AMOLED	1.28	416 x 416	30	24	Yes	Yes	Yes	$299

Activity 2: Data Preparation

Data preparation, also known as data preprocessing, is the process of cleaning,transforming, and organizing raw data before it can be used in a data analysis or machine learning model.

The activity include following steps:

- missing values
- handling outliers
- encoding categorical variables
- normalizing data

Note: These are general steps to take in pre-processing before feeding data to machine learning for training. The pre-processing steps differ depending on the dataset. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling Missing Values

Let's first figure out what kind of data is in our columns by using df.info(). We may deduce from this that our columns include data of the types "object" and "float64".

```
# Getting the information of data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 379 entries, 0 to 378
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Brand                      378 non-null    object
 1   Model                      378 non-null    object
 2   Operating System           376 non-null    object
 3   Connectivity               378 non-null    object
 4   Display Type               377 non-null    object
 5   Display Size (inches)      376 non-null    float64
 6   Resolution                 375 non-null    object
 7   Water Resistance (meters)  378 non-null    object
 8   Battery Life               378 non-null    object
 9   Heart Rate Monitor         378 non-null    object
 10  GPS                        378 non-null    object
 11  NFC                        378 non-null    object
 12  Price (USD)                378 non-null    object
dtypes: float64(1), object(12)
memory usage: 38.6+ KB
```

```
# Checking for the null values
df.isnull().any()
```

```
Brand                      True
Model                      True
Operating System           True
Connectivity               True
Display Type               True
Display Size (inches)      True
Resolution                 True
Water Resistance (meters)  True
Battery Life               True
Heart Rate Monitor         True
GPS                        True
NFC                        True
Price (USD)                True
dtype: bool
```

```
df.isnull().sum()
```

```
Brand                      1
Model                      1
Operating System           3
Connectivity               1
Display Type               2
Display Size (inches)      3
Resolution                 4
Water Resistance (meters)  1
Battery Life               1
Heart Rate Monitor         1
GPS                        1
NFC                        1
Price (USD)                1
dtype: int64
```

This line of code is used to count the number of missing or null values in a pandas DataFrame. It returns a list of the total number of missing values in each column of the DataFrame.

```
# Replacing Null values in Cateogrical column with mode.
object_columns=df.select_dtypes(include=["object"]).columns
for col in object_columns:
    mode_value=df[col].mode()[0]
    df[col]=df[col].fillna(mode_value)
```

```
# Replacing Null values in float columns with mean
float_columns= df.select_dtypes(include=["float"]).columns
for col in float_columns:
    mean_value=df[col].mean()
    df[col]=df[col].fillna(mean_value)
```

```
df.isnull().any()
```

```
Brand                        False
Model                        False
Operating System             False
Connectivity                 False
Display Type                 False
Display Size (inches)        False
Resolution                   False
Water Resistance (meters)    False
Battery Life                 False
Heart Rate Monitor           False
GPS                          False
NFC                          False
Price (USD)                  False
dtype: bool
```

```
# Checking for the null values
df.isnull().any()
```

```
Brand                        True
Model                        True
Operating System             True
Connectivity                 True
Display Type                 True
Display Size (inches)        True
Resolution                   True
Water Resistance (meters)    True
Battery Life                 True
Heart Rate Monitor           True
GPS                          True
NFC                          True
Price (USD)                  True
dtype: bool
```

```
df.isnull().sum()
```

```
Brand                        1
Model                        1
Operating System             3
Connectivity                 1
Display Type                 2
Display Size (inches)        3
Resolution                   4
Water Resistance (meters)    1
Battery Life                 1
Heart Rate Monitor           1
GPS                          1
NFC                          1
Price (USD)                  1
dtype: int64
```

This line of code is used to count the number of missing or null values in a pandas DataFrame. It returns a list of the total number of missing values in each column of the DataFrame.

```
# Replacing Null values in Cateogrical column with mode.
object_columns=df.select_dtypes(include=["object"]).columns
for col in object_columns:
    mode_value=df[col].mode()[0]
    df[col]=df[col].fillna(mode_value)
```

```
# Replacing Null values in float columns with mean
float_columns= df.select_dtypes(include=["float"]).columns
for col in float_columns:
    mean_value=df[col].mean()
    df[col]=df[col].fillna(mean_value)
```

```
df.isnull().any()
```

```
Brand                        False
Model                        False
Operating System             False
Connectivity                 False
Display Type                 False
Display Size (inches)        False
Resolution                   False
Water Resistance (meters)    False
Battery Life                 False
Heart Rate Monitor           False
GPS                          False
NFC                          False
Price (USD)                  False
dtype: bool
```

The first code selects all columns in a DataFrame that have data type 'object' (usually text) and fills in any missing values (NaN) in those columns with the most frequent value in each column.

The second code selects all columns in the DataFrame that have data type 'float64' (usually decimal numbers) and fills in any missing values (NaN) in those columns with the mean (average) value of the column.

Activity 2.2: Handling Independent Columns

```
# Handling Independent columns
```

```
# We are renaming the columns for easy coding.
df=df.rename(columns={'Display Size (inches)':'Display Size','Water Resistance (meters)':'Water Resistance','Battery Life (days)':'Battery Life','Price (USD)':'Price'})
```

```
df.head()
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	Heart Rate Monitor	GPS	NFC	Price
0	Apple	Watch Series 7	watchOS	Bluetooth, Wi-Fi, Cellular	Retina	1.90	396 x 484	50	18	Yes	Yes	Yes	$399
1	Samsung	Galaxy Watch 4	Wear OS	Bluetooth, Wi-Fi, Cellular	AMOLED	1.40	450 x 450	50	40	Yes	Yes	Yes	$249
2	Garmin	Venu 2	Garmin OS	Bluetooth, Wi-Fi	AMOLED	1.30	416 x 416	50	11	Yes	Yes	No	$399
3	Fitbit	Versa 3	Fitbit OS	Bluetooth, Wi-Fi	AMOLED	1.58	336 x 336	50	6	Yes	Yes	Yes	$229
4	Fossil	Gen 6	Wear OS	Bluetooth, Wi-Fi	AMOLED	1.28	416 x 416	30	24	Yes	Yes	Yes	$299

We can use the df.rename() function to rename specific columns in the dataset for simplicity.

```
df['Water Resistance'].unique()
```

```
array(['50', '30', '100', '1.5', 'Not specified', '200', '10'],
      dtype=object)
```

```
df['Water Resistance'].describe()
```

```
count     379
unique      7
top        50
freq      276
Name: Water Resistance, dtype: object
```

```
# We are replacing the not specified values in water resistance column
df['Water Resistance']=df['Water Resistance'].replace({'Not specified':'50'})
```

```
df['Display Size'].unique()
```

```
array([1.9       , 1.4       , 1.3       , 1.58      , 1.28      ,
       1.43      , 1.75      , 1.39      , 1.36316489, 1.65      ,
       1.2       , 1.57      , 1.        , 1.78      , 1.91      ,
       1.38      , 1.06      , 1.35      , 1.34      , 0.9       ,
       1.04      , 1.64      , 1.19      , 4.01      , 1.6       ,
       1.42      , 2.1       , 1.23      , 1.1       , 1.22      ,
       1.5       , 1.36      , 1.32      ])
```

The first line of code provides a list of all unique values in the 'Water Resistance' column of the dataframe. The second line of code provides descriptive statistics for the 'Water Resistance' column in order to determine the most frequently observed value. The third line of code changes all instances of 'Not specified' in the 'Water Resistance' column with '50'. When dealing with missing or confusing data, this is useful.

```
df['Display Size'].unique()
```

```
array([1.9       , 1.4       , 1.3       , 1.58      , 1.28      ,
       1.43      , 1.75      , 1.39      , 1.36316489, 1.65      ,
       1.2       , 1.57      , 1.        , 1.78      , 1.91      ,
       1.38      , 1.06      , 1.35      , 1.34      , 0.9       ,
       1.04      , 1.64      , 1.19      , 4.01      , 1.6       ,
       1.42      , 2.1       , 1.23      , 1.1       , 1.22      ,
       1.5       , 1.36      , 1.32      ])
```

```
# Rounding the above float numbers to 1 decimal point.
df['Display Size']=df['Display Size'].round(1)
```

The first line of code retrieves all the unique values in the "Display Size" column of the dataframe. The second line of code rounds the values in the "Display Size" column to one decimal place. This is done to simplify and make the data more consistent.

```
df['Battery Life'].unique()
```

```
array(['18', '40', '11', '6', '24', '14', '2', '4', '12', '30', '3', '45',
       '5', '10', '48', '7', '16', '9', '25', '72', '60', '56', '70', '1',
       '48 hours', '15', 'Unlimited', '1.5', '20', '8'], dtype=object)
```

```
df['Battery Life'].describe()
```

```
count     379
unique     30
top        14
freq       84
Name: Battery Life, dtype: object
```

The first line of code provides a list of all unique values in the 'Battery Life' column of the dataframe. The second line of code provides descriptive statistics for the 'Battery Life' column in order to determine the most frequently observed value. The third line of code changes all instances of '48 hours' and ‘Unlimited’ in the 'Battery Life' column with '14'. When dealing with missing or confusing data, this is useful.

```
# We replace all the 48 hours and the Unlimited with top value of battery life describe to deal with missing and confusing data.
df['Battery Life']= df['Battery Life'].replace({'48 hours':'14','Unlimited':'14'})
```

```
# It removes the dollar symbol before the price.
df['Price']=df['Price'].str[1:]
```

```
df['Water Resistance']=df['Water Resistance'].astype(float)
df['Battery Life']=df['Battery Life'].astype(float)
df['Price'] = df['Price'].str.replace(',', '').astype(float)
```

This line of code removes the dollar symbol from the values in the ‘Price (USD)’ column.

These codes convert the data type of columns 'Water Resistance', 'Battery Life', and 'Price' in dataframe 'df' from object to float.

Activity 2.3: Handling Categorical Values

```
# Label Encoding
```

```
from sklearn.preprocessing import LabelEncoder
lb= LabelEncoder()
```

```
categorical_cols= df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col]=lb.fit_transform(df[col])
```

In this code, the LabelEncoder object is first created. Then, the categorical columns in the dataset are identified and stored in a list. The for loop is then used to iterate through each categorical column, and the fit_transform method of the LabelEncoder object is applied to encode the categorical values in each column with unique integer values.

```
df.head()
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	Heart Rate Monitor	GPS	NFC	Price
0	1	127	34	2	17	1.9	27	50.0	18.0	0	1	1	399.0
1	30	36	31	2	0	1.4	31	50.0	40.0	0	1	1	249.0
2	8	105	9	1	0	1.3	30	50.0	11.0	0	1	0	399.0
3	6	109	7	1	0	1.6	19	50.0	6.0	0	1	1	229.0
4	7	43	31	1	0	1.3	30	30.0	24.0	0	1	1	299.0

```
df=df.drop('Heart Rate Monitor', axis=1)
```

```
df.head()
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	GPS	NFC	Price
0	1	127	34	2	17	1.9	27	50.0	18.0	1	1	399.0
1	30	36	31	2	0	1.4	31	50.0	40.0	1	1	249.0
2	8	105	9	1	0	1.3	30	50.0	11.0	1	0	399.0
3	6	109	7	1	0	1.6	19	50.0	6.0	1	1	229.0
4	7	43	31	1	0	1.3	30	30.0	24.0	1	1	299.0

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

The purpose of descriptive analysis is to analyze the basic features of data using a statistical technique. In this case, Pandas has a useful function called describe. We can understand the unique, top, and frequent values of categorical features with this describe function. We can also get the count, mean, standard deviation, minimum, maximum, and percentile values of continuous features.

```
df.describe(include='all')
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	GPS	NFC	Price
count	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.00000	379.000000
mean	18.168865	68.606860	20.778364	1.203166	6.941953	1.368074	22.139842	52.804749	12.208443	0.920844	0.83905	312.910290
std	13.040757	38.933753	11.407946	0.532927	8.978918	0.219087	9.080415	26.939235	12.326042	0.270338	0.36797	202.163738
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.900000	0.000000	1.500000	1.000000	0.000000	0.00000	49.000000
25%	7.000000	33.500000	9.000000	1.000000	0.000000	1.200000	17.500000	50.000000	3.000000	1.000000	1.00000	199.000000
50%	16.000000	71.000000	27.000000	1.000000	0.000000	1.400000	23.000000	50.000000	11.000000	1.000000	1.00000	279.000000
75%	31.000000	102.000000	31.000000	1.000000	14.000000	1.400000	32.000000	50.000000	15.000000	1.000000	1.00000	329.000000
max	41.000000	136.000000	34.000000	4.000000	26.000000	4.000000	35.000000	200.000000	72.000000	1.000000	1.00000	1800.000000

Activity 2: Visual analysis

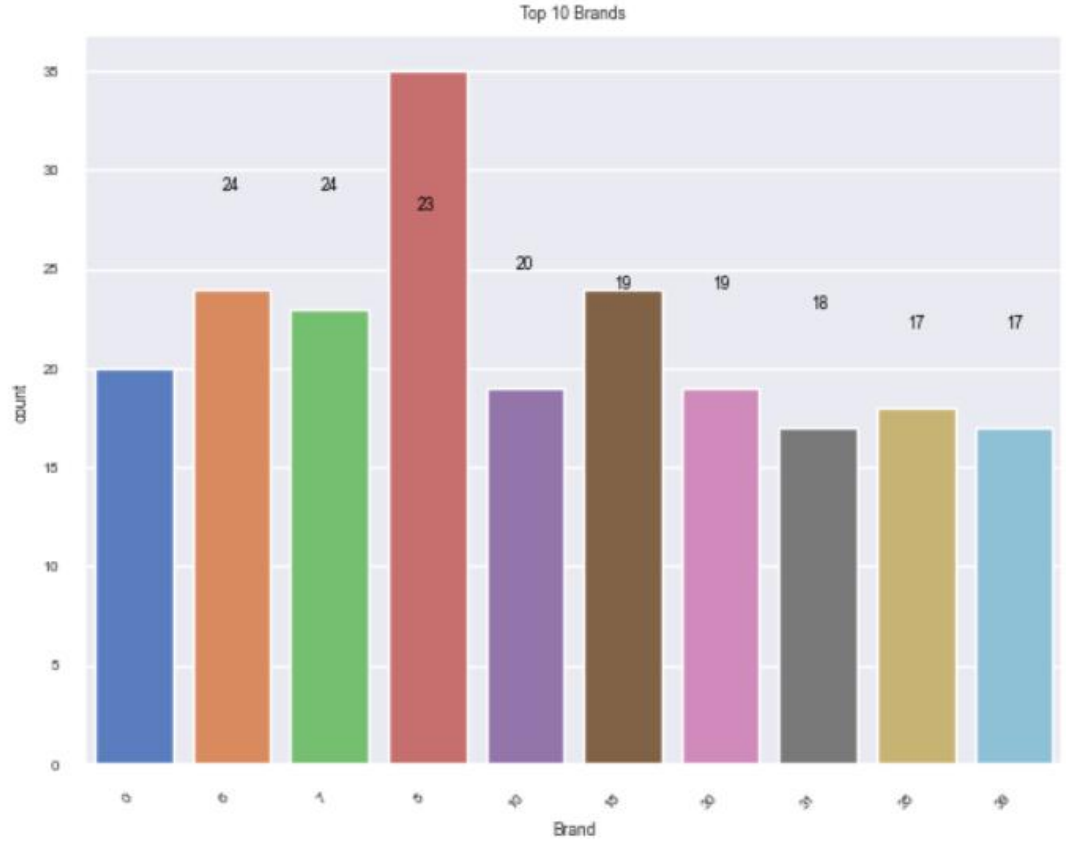
Visual analysis is the process of examining and understanding data via the use of visual representations such as charts, plots, and graphs. It is a method for quickly identifying patterns, trends, and outliers in data, which can aid in gaining insights and making sound decisions.

Activity 2.1: Univariate analysis

Univariate analysis is a statistical method used to analyse a single variable in a dataset. This analysis focuses on understanding the distribution, central tendency, and dispersion of a single variable.

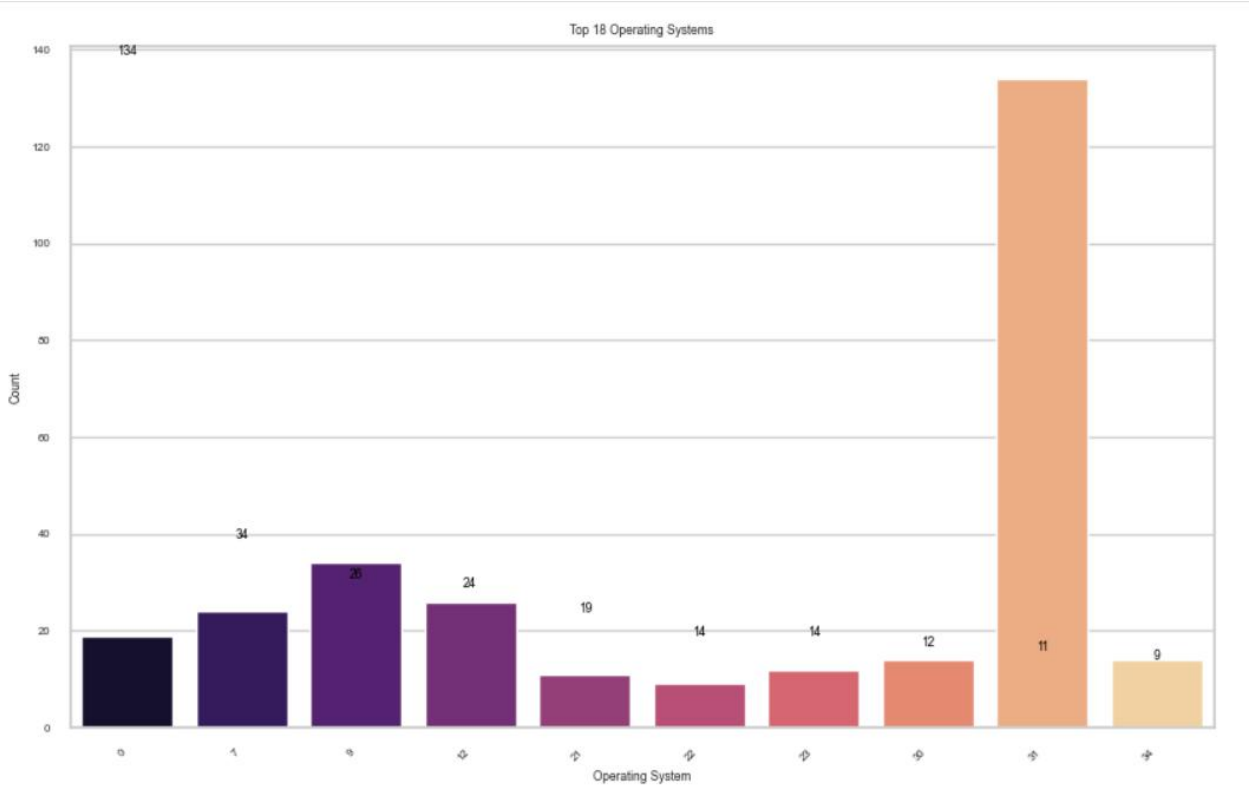
The code below creates a bar plot using the Seaborn library to show the top 10 brands in a dataset, along with their respective counts. It first gets the top 10 brands and their counts using the value_counts() method, then sets the plot style, creates the bar plot using the barplot() method, and rotates the x-tick labels for better visibility. Value labels are added on the bars using the text() method, and the axis labels and title are set using the set() method. Finally, the plot is displayed using the show() method.

```
top_brands= df['Brand'].value_counts().index[:10]
counts = df['Brand'].value_counts().values[:10]
sns.set_style("darkgrid")
ax = sns.barplot(x=top_brands, y=counts, palette="muted")
ax.set_xticklabels (ax.get_xticklabels (), rotation=45, ha='right')
for i, v in enumerate(counts):
    ax.text(i, v+5, str(v), color='black', ha='center')
ax.set(xlabel='Brand', ylabel='count', title='Top 10 Brands')
plt.show()
```




```
sns.set_style('whitegrid')
top_os=df['Operating System'].value_counts().index[:10]
os_counts = df['Operating System'].value_counts().values[:10]
fig, ax = plt.subplots (figsize=(10, 6))
ax = sns.barplot(x=top_os, y=os_counts, palette='magma')
ax.set(xlabel='Operating System', ylabel='Count', title='Top 18 Operating Systems')
plt.xticks(rotation=45, ha='right')
for i, v in enumerate(os_counts):
       ax.text(i, v+5, str(v), color='black', ha='center')
plt.show()
```

This code generates a bar plot showing the top 10 operating systems used by customers of a particular product. The data is obtained from a dataframe called 'data', and the plot is generated using the Seaborn library. The x-axis shows the name of the operating systems and the y-axis shows the count of each operating system. The plot also includes labels on the bars to indicate the exact count of each operating system. The plot is displayed using the matplotlib library.

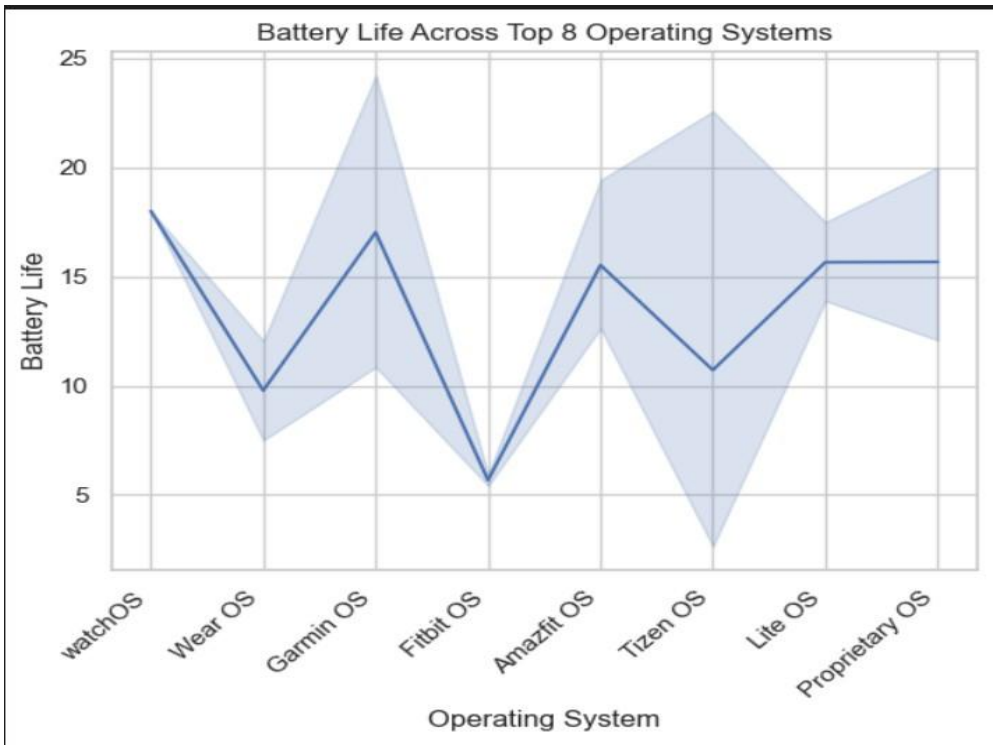


Activity 2.2: Bivariate analysis

Bivariate analysis is a statistical method used to analyse the relationship between two variables in a dataset. This analysis focuses on examining how changes in one variable are related to changes in another variable.

```
import seaborn as sns
import matplotlib.pyplot as plt

top_operating_systems = df['Operating System'].value_counts().head(8).index.tolist()
data_top_operating_systems = df[df['Operating System'].isin(top_operating_systems)]

sns.set_style('whitegrid')
sns.lineplot(x='Operating System', y='Battery Life', data=data_top_operating_systems)
plt.xlabel('Operating System')
plt.ylabel('Battery Life')
plt.title('Battery Life Across Top 8 Operating Systems')
plt.xticks(rotation=45, ha='right')
plt.show()
```

This code is creating a line plot to show the battery life of the top 8 operating systems. It first filters the data to include only the top 8 operating systems by frequency count. Then, it creates a line plot where the x-axis represents the operating system and the y-axis represents the battery life. Finally, it sets the axis labels, title, and rotates the x-axis labels for better readability.



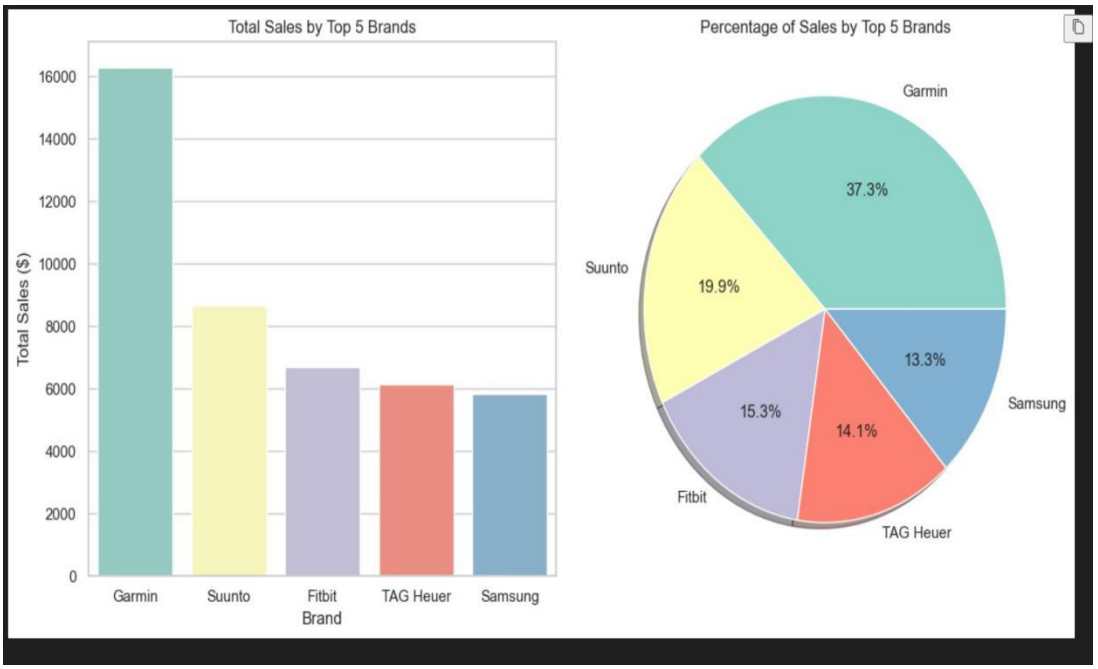
Activity 2.3: Multivariate analysis

Multivariate analysis is a statistical technique used to analyse data that involves more than two variables. It aims to understand the relationships between multiple variables in a dataset by examining how they are related to each other and how they contribute to a particular outcome or phenomenon.

```
total_sales = df.groupby('Brand')['Price'].sum().reset_index()
top_brands = total_sales.sort_values('Price', ascending=False).head(5)
top_brands['Percent'] = (top_brands['Price'] / top_brands['Price'].sum()) * 100
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
sns.barplot(x='Brand', y='Price', data=top_brands, palette='Set3', ax=axes[0])
axes[0].set_xlabel('Brand')
axes[0].set_ylabel('Total sales ($)')
axes[0].set_title('Total Sales by Top 5 Brands')
colors = sns.color_palette('Set3', top_brands.shape[0]).as_hex()
axes[1].pie(top_brands['Percent'], labels=top_brands['Brand'], colors=colors, autopct='%1.1f%%', shadow=True)
axes[1].set_title('Percentage of Sales by Top 5 Brands')
fig.tight_layout()
plt.show()
```

This code generates a bar chart and a pie chart to display the total sales and percentage of sales for the top 5 brands in a dataset.

First, the code calculates the total sales for each brand and sorts the brands by total sales in descending order. Then, it calculates the percentage of sales for each of the top 5 brands. Next, it creates a grid with two subplots, one for the bar chart and the other for the pie chart. The bar chart shows the total sales for each of the top 5 brands, while the pie chart shows the percentage of sales for each brand. Finally, it adjusts the spacing between subplots and displays the plot.



Splitting data into train and test

Now let us split the Dataset into train and test sets. First split the dataset into X and y and then split the data set. The 'X' corresponds to independent features and 'y' corresponds to the target variable.

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

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	GPS	NFC
0	1	127	34	2	17	1.9	27	50.0	18.0	1	1
1	30	36	31	2	0	1.4	31	50.0	40.0	1	1
2	8	105	9	1	0	1.3	30	50.0	11.0	1	0
3	6	109	7	1	0	1.6	19	50.0	6.0	1	1
4	7	43	31	1	0	1.3	30	30.0	24.0	1	1
...	...	...	...	...	...	...	...	...	...	...	...
374	38	79	32	1	16	1.4	21	50.0	30.0	0	1
375	41	132	33	2	0	1.4	32	50.0	15.0	1	1
376	9	119	12	1	0	1.4	32	50.0	25.0	1	1
377	26	118	5	1	0	1.6	17	50.0	14.0	0	1
378	35	71	31	2	0	1.4	32	50.0	72.0	1	1

379 rows × 11 columns

```
y=df['Price']
y
```

```
0      399.0
1      249.0
2      399.0
3      229.0
4      299.0
       ...
374    279.0
375    349.0
376    249.0
377    159.0
378    299.0
Name: Price, Length: 379, dtype: float64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=25)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(303, 11)
(303,)
(76, 11)
(76,)
```

For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying five regression algorithms. The best model is saved based on its performance.

Activity 1.1: Linear Regression Model

1. Linear Regression ¶

```
lr= LinearRegression()
lr.fit(X_train,y_train)
```

This code performs linear regression modeling using the scikit-learn library.
It creates a LinearRegression object named "lr". The "fit" method is then called on the training data X_train and y_train. This method trains the model by finding the coefficients for the linear equation that best fits the data.

Activity 1.2: Decision Tree Regressor Model

2. Decision Tree Regression

```
dtr=DecisionTreeRegressor(max_depth=2,min_samples_split=6,min_samples_leaf=5)
dtr.fit(X_train,y_train)
```

```
DecisionTreeRegressor(max_depth=2, min_samples_leaf=5, min_samples_split=6)
```

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

This code creates a Decision Tree Regression model using the scikit-learn library. It creates a DecisionTreeRegressor object named "dtr" with a maximum depth of 2 levels, a minimum number of samples required to split an internal node, and a minimum number of samples required to be at a leaf node. The "fit" method is then called on the training data X_train and y_train to train the model.

Activity 1.3: Random Forest Regressor Model

3. Random Forest Regression

```
rfr=RandomForestRegressor(n_estimators=50,max_depth=8,min_weight_fraction_leaf=0.05,max_features=0.8,random_state=42)
rfr.fit(X_train,y_train)
```

```
RandomForestRegressor(max_depth=8, max_features=0.8,
                      min_weight_fraction_leaf=0.05, n_estimators=50,
                      random_state=42)
```

This code creates a Random Forest Regression model using the scikit-learn library. It creates a RandomForestRegressor object named "rfr". The n_estimators parameter sets the number of decision trees to create, max_depth parameter sets the maximum depth of each decision tree, min_weight_fraction_leaf parameter sets the minimum fraction of samples required to be at a leaf node, max_features parameter sets the maximum fraction of features used at each split. The random_state parameter sets the seed for the random number generator.

The "fit" method is then called on the training data X_train and y_train to train the model.

Activity 1.4: Gradient Boosting Regressor Model

4. Gradient Boosting Regressor

```
gbr=GradientBoostingRegressor(n_estimators=100,learning_rate=0.1,max_depth=1,random_state=42)
gbr.fit(X_train,y_train)
```

```
GradientBoostingRegressor(max_depth=1, random_state=42)
```

This code creates a Gradient Boosting Regression model with 100 decision trees, a learning rate of 0.1, and a maximum depth of 1 level. The random state is set to 42 to ensure reproducibility of the results.

The "fit" method is then called on the training data X_train and y_train to train the model. The n_estimators parameter sets the number of decision trees to create, while the learning_rate parameter controls the contribution of each decision tree to the final prediction. The max_depth parameter limits the maximum depth of each decision tree, preventing overfitting of the model.

Activity 1.5: Extreme Gradient Boost Regressor Model

```
import xgboost as xgb
xgb=xgb.XGBRegressor(n_estimators=1000,learning_rate=0.06,max_depth=2,sunsample=0.7,colsample_bytree=0.4,colsample_bylevel=0.5,
                     max_leaves=3,random_state=1)
xgb.fit(X_train,y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=0.5, colsample_bynode=None, colsample_bytree=0.4,
             device=None, early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.06, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=2,
             max_leaves=3, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1000,
             n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

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

This code performs XGBoost regression modeling using the XGBoost library. It creates an XGBRegressor object named "xgb". The "fit" method is then called on the training data X_train and y_train.

The n_estimators parameter sets the number of decision trees to create, learning_rate parameter sets the contribution of each decision tree to the final prediction, max_depth parameter limits the maximum depth of each decision tree, subsample parameter controls the fraction of samples used for each tree, colsample_bytree and colsample_bylevel parameters control the fraction of features used at each split in the trees, max_leaves parameter limits the maximum number of leaves in each decision tree.

The random_state parameter sets the seed for the random number generator.

Milestone 5: Performance Testing

Activity 1: Check model performance on test and train data for each model

To check the model performance on test and train data, we can use the "score" method to calculate the coefficient of determination (R-squared) between the predicted and actual values and also calculate the RMSE score. Comparing the
R-squared scores for both the test and train data can help us determine if the model is overfitting or underfitting. The RMSE score tells us how far off the predictions of the model are, on average, from the actual values.

Activity 1.1: Linear Regression Model

```
# training score

error_score_lr_train= r2_score(y_train,predict_train)
print("R2 error is:",error_score_lr_train)
mse=mean_squared_error(y_train,predict_train)
rmse_lr_train=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_lr_train)
```

```
R2 error is: 0.2295546632471358
Root Mean Squared Error: 179.89481395578446
```

```
# testing score

error_score_lr_test=r2_score(y_test,predict_test)
print("R2 error is:",error_score_lr_test)
mse=mean_squared_error(y_test,predict_test)
rmse_lr_test=np.sqrt(mse)
print("Root Mean Squared Error:",rmse_lr_test)
```

```
R2 error is: 0.16590308669836784
Root Mean Squared Error: 172.2507837673408
```

Activity 1.2: Decision Tree Regressor Model

```
# training Score

error_score_dtr_train=r2_score(y_train,predict_train_dtr)
print("R2 erroor is:",error_score_dtr_train)
mse=mean_squared_error(y_train,predict_train_dtr)
rmse_dtr_train=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_dtr_train)
```

```
R2 erroor is: 0.34296149275185 23
Root Mean Squared Error: 166.12811592656647
```

```
#testing score

error_score_dtr_test=r2_score(y_test,predict_test_dtr)
print("R2 erroor is:",error_score_dtr_test)
mse=mean_squared_error(y_test,predict_test_dtr)
rmse_dtr_test=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_dtr_test)
```

```
R2 erroor is: 0.18085636154493812
Root Mean Squared Error: 170.69978774403583
```

Activity 1.3: Random Forest Regressor Model

```
# training Score

error_score_rfr_train=r2_score(y_train,predict_train_rfr)
print("R2 erroor is:",error_score_rfr_train)
mse=mean_squared_error(y_train,predict_train_rfr)
rmse_rfr_train=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_rfr_train)
```

```
R2 erroor is: 0.49758319869121215
Root Mean Squared Error: 145.2712940837865
```

```
#testing score

error_score_rfr_test=r2_score(y_test,predict_test_rfr)
print("R2 erroor is:",error_score_rfr_test)
mse=mean_squared_error(y_test,predict_test_rfr)
rmse_rfr_test=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_rfr_test)
```

```
R2 erroor is: 0.4261481054255444
Root Mean Squared Error: 142.87388678752063
```

Activity 1.4: Gradient Boosting Regressor Model

```
# training Score

error_score_gbr_train=r2_score(y_train,predict_train_gbr)
print("R2 erroor is:",error_score_gbr_train)
mse=mean_squared_error(y_train,predict_train_gbr)
rmse_gbr_train=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_gbr_train)
```

```
R2 erroor is: 0.41649950162493654
Root Mean Squared Error: 156.55550514239548
```

```
#testing score

error_score_gbr_test=r2_score(y_test,predict_test_gbr)
print("R2 erroor is:",error_score_gbr_test)
mse=mean_squared_error(y_test,predict_test_gbr)
rmse_gbr_test=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_gbr_test)
```

```
R2 erroor is: 0.4141301569073099
Root Mean Squared Error: 144.36220988703522
```

Activity 1.5: Extreme Gradient Boost Regressor Model

```
# training Score

error_score_xgb_train=r2_score(y_train,predict_train_xgb)
print("R2 erroor is:",error_score_xgb_train)
mse=mean_squared_error(y_train,predict_train_xgb)
rmse_xgb_train=np.sqrt(mse)
print('Root Mean Squared Error:',rmse_xgb_train)
```

```
R2 erroor is: 0.9219250785587862
Root Mean Squared Error: 57.26687782255483
```

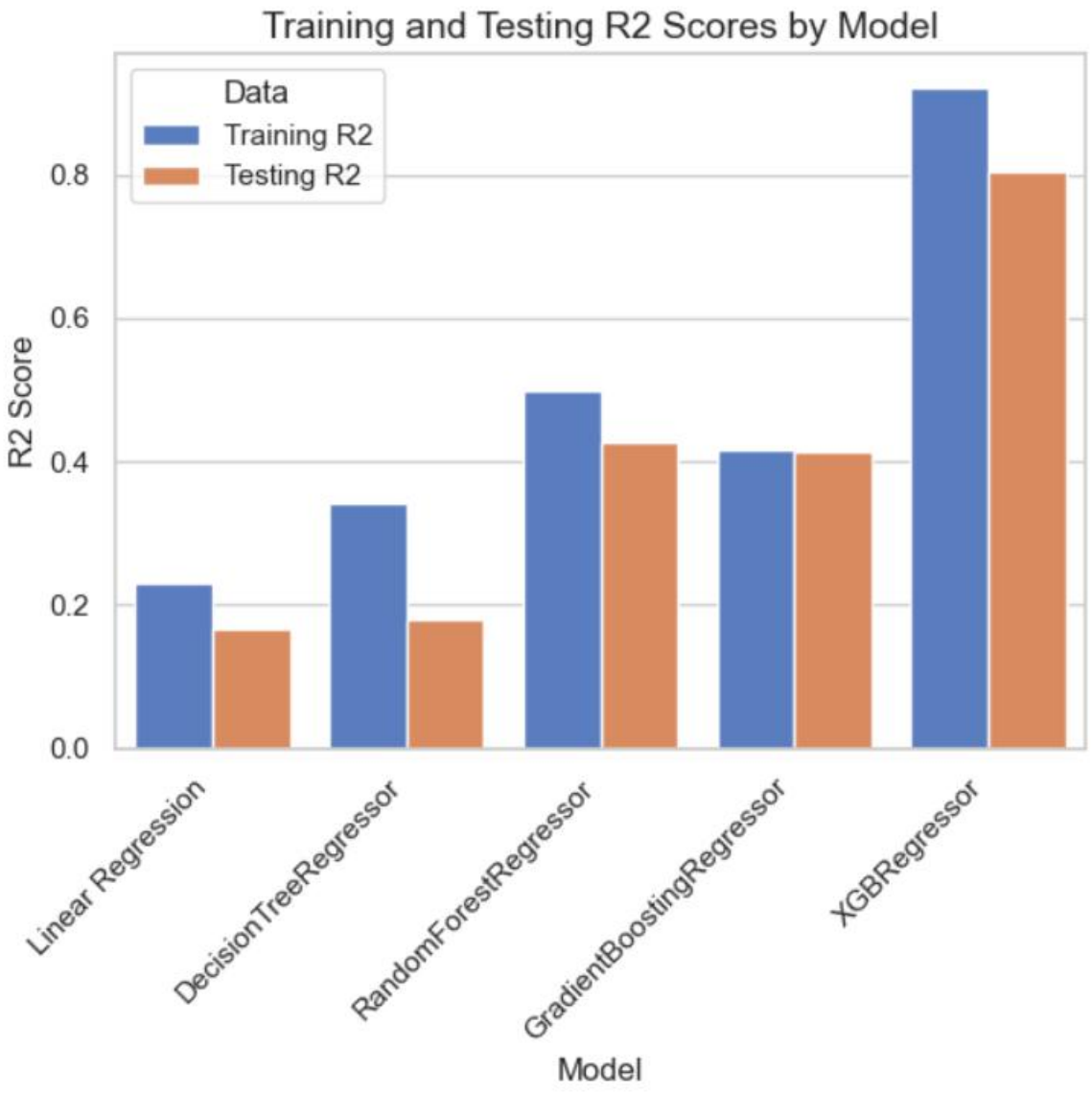
Activity 2: Comparing models

```
results=pd.DataFrame(columns=['Model','Training R2','Testing R2','Traing RMSE','Testing RMSE'])
results.loc[0]=['Linear Regression',error_score_lr_train,error_score_lr_test,rmse_lr_train,rmse_lr_test]
results.loc[1]=['DecisionTreeRegressor',error_score_dtr_train,error_score_dtr_test,rmse_dtr_train,rmse_dtr_test]
results.loc[2]=['RandomForestRegressor',error_score_rfr_train,error_score_rfr_test,rmse_rfr_train,rmse_rfr_test]
results.loc[3]=['GradientBoostingRegressor',error_score_gbr_train,error_score_gbr_test,rmse_gbr_train,rmse_gbr_test]
results.loc[4]=['XG Boost Regressor',error_score_xgb_train,error_score_xgb_test,rmse_xgb_train,rmse_xgb_test]
print(results)
```

```
                        Model  Training R2  Testing R2  Traing RMSE  \
0           Linear Regression     0.229555    0.165903   179.894814
1       DecisionTreeRegressor     0.342961    0.180856   166.128116
2       RandomForestRegressor     0.497583    0.426148   145.271294
3   GradientBoostingRegressor     0.416500    0.414130   156.555505
4          XG Boost Regressor     0.921925    0.805145    57.266878

    Testing RMSE
0     172.250784
1     170.699788
2     142.873887
3     144.362210
4      83.254640
```

This code creates a Pandas Data Frame named "results" that contains the model names, R-squared scores, and root mean squared errors (RMSE) for both the training and testing data for each of the five regression models: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, and XG Boost Regressor.

```
data = {
    'Model': ['Linear Regression', 'DecisionTreeRegressor', 'RandomForestRegressor', 'GradientBoostingRegressor', 'XGBRegressor'],
    'Training R2': [error_score_lr_train,error_score_dtr_train,error_score_rfr_train,error_score_gbr_train,error_score_xgb_train],
    'Testing R2':[error_score_lr_test,error_score_dtr_test,error_score_rfr_test,error_score_gbr_test,error_score_xgb_test]
}

# Create a DataFrame
results = pd.DataFrame(data)

# Melt the DataFrame to long format for easier plotting
results_long = pd.melt(results, id_vars='Model', value_vars=['Training R2', 'Testing R2'], var_name='Data', value_name='R2 Score')

# Set the style of the plot
sns.set(style="whitegrid")

# Create a bar plot using seaborn
plt.figure(figsize=(6, 6))
ax = sns.barplot(x='Model', y='R2 Score', hue='Data', data=results_long, palette='muted')

# Set axis labels and title
plt.xlabel('Model', fontsize=12)
plt.ylabel('R2 Score', fontsize=12)
plt.title('Training and Testing R2 Scores by Model', fontsize=14)

# Rotate x-axis tick labels for better visibility
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.tight_layout()
plt.show()
```



To determine which model is best, we should look for the model with the highest R-squared score and lowest RMSE value on the test data. This suggests that the model is the most accurate and generalizes well to new data. In this out of the five models selected XG Boost Regressor satisfies the conditions and hence selected.

Milestone 6: Model Deployment

Activity 1: Save the best model

```
# dumping the selection model
pickle.dump(xgb,open('SW.pkl','wb'))
```

This code uses the "pickle" library in Python to save the trained XG Boost Regressor model named "xgb" as a file named "SW.pkl".

The "dump" method from the pickle library is used to save the model object in a serialized form that can be used again later. The "wb" parameter indicates that the file should be opened in binary mode to write data to it.

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that would help us integrate the machine learning model we have built and trained.

A user interface is provided for the users to enter the values for predictions. The entered values are fed into the saved model, and the prediction is displayed on the UI.

The section has following task:

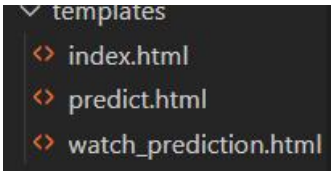
- Building HTML pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project we create three html files:

- index.html
- predict.html
- watch_prediction.html

and save these html files in the templates folder.



Activity 2.2: Build Python code:

Importing the libraries

```
import pickle
from flask import Flask, render_template, request
import pandas as pd
import numpy as np
```

This code first loads the saved XG Boost Regressor model from the "SW.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
model1 = pickle.load(open('SW.pkl', 'rb'))
app=Flask(__name__)
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render_template()" method is used to render an HTML template named "index.html". The “index.html” is the home page.

```
@app.route('/')
def home():
    return render_template('index.html')
```

The route in this case is "/predict". When a user accesses the "/predict" route of the website, this function “index()” is called.

The "render_template()" method is used to render an HTML template named "predict.html".

```
@app.route('/predict') # rendering the html template
def index() :
    return render_template('predict.html')
```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/data_predict", and the method is set to GET and POST.

The function "predict()" is then associated with this route.

This function first receives the user inputs for various smart watch specifications such as brand, model, operating system, connectivity, display type, display size etc., using "request.form['...']".

The form includes several fields, such as "Brand," "Model," "Operating System," and "Connectivity," among others. The function uses a series of conditional statements to convert the values of these fields into numerical values that can be used to make predictions.

For example, the function checks the value of the "Brand" field, and if it matches a certain string (such as "Garmin" or "Mobvoi"), it assigns a numerical value to that brand (such as 8 or 18). Similarly, the function checks the value of the "Model" field, and assigns a numerical value based on the string value (such as 81 or 22). The same goes for the "Operating System" and "Connectivity" fields.

Once all of these values have been assigned, the function then uses the loaded XG Boost Regressor model to predict the smart watch price based on the user inputs.

```
if model == 'Hybrid HR':
    model = 44
if model == 'Venu Sq':
    model = 106
if model == 'MagicWatch 2':
    model = 56
if model == 'TicWatch Pro 3':
    model = 97
if model == 'Vapor X':
    model = 104
if model == 'Z':
    model = 132

os = request.form['Operating System']
if os == 'Wear OS':
    os = 31
if os == 'Garmin OS':
    os = 9
if os == 'Lite OS':
    os = 12
```

```
connect = request.form['Connectivity']
if connect == 'Bluetooth, Wi-Fi':
    connect = 1
if connect == 'Bluetooth, Wi-Fi, Cellular':
    connect = 2
if connect == 'Bluetooth':
    connect = 0
if connect == 'Bluetooth, Wi-Fi, GPS':
    connect = 3
if connect == 'Bluetooth, Wi-Fi, NFC':
    connect = 4

display_type = request.form['Display Type']
if display_type == 'AMOLED':
    display_type = 0
if display_type == 'LCD':
    display_type = 9
```

```
prediction = model1.predict(pd.DataFrame([[brand,model,os,connect,display_type,display_size,resolution,water,battery,gps,nfc]],
                                         columns= ['Brand', 'Model', 'Operating System', 'Connectivity',
                                                   'Display Type', 'Display Size', 'Resolution',
                                                    'Water Resistance','Battery Life','GPS','NFC']))


prediction = np.round(prediction,2)


return render_template('watch_prediction.html', prediction_text ="is {}".format(prediction))
```

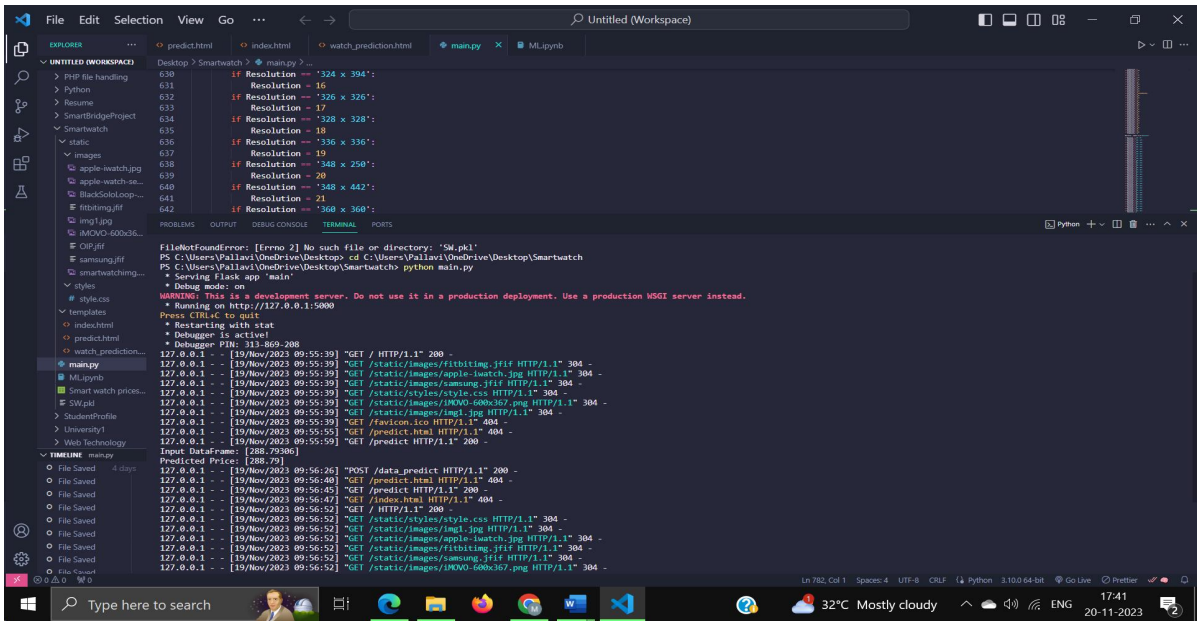
Main Function:

This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.

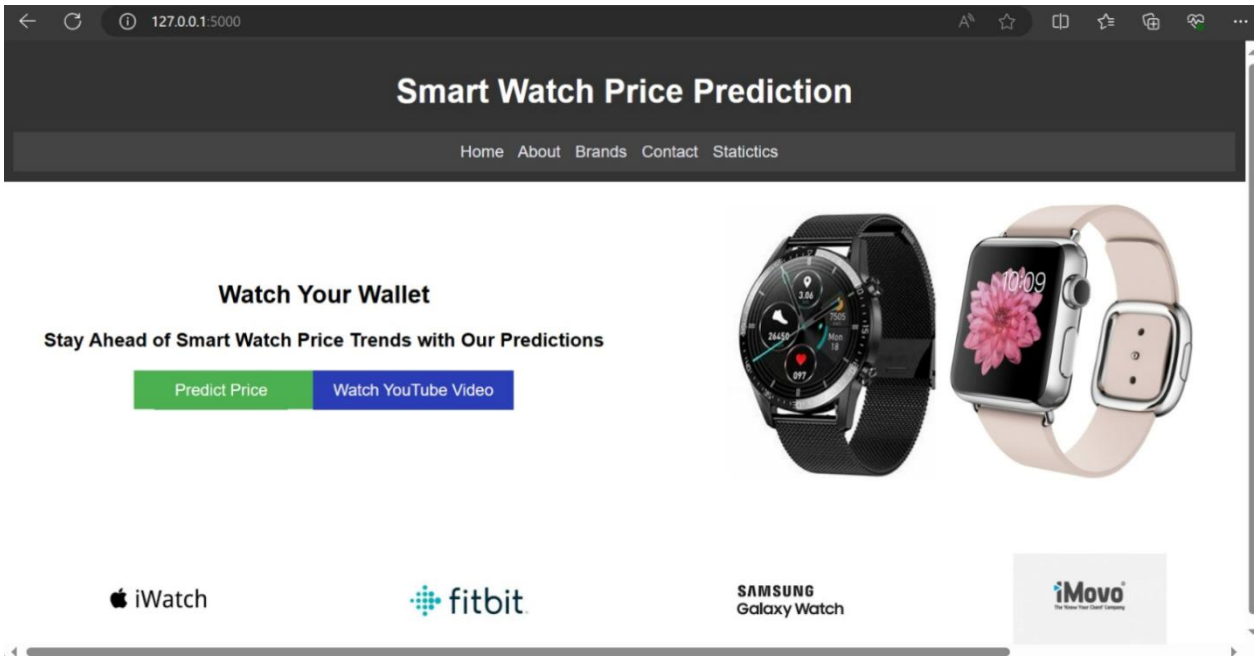
```
if __name__ == '__main__':
    app.run()
```

Activity 2.3: Run the web application

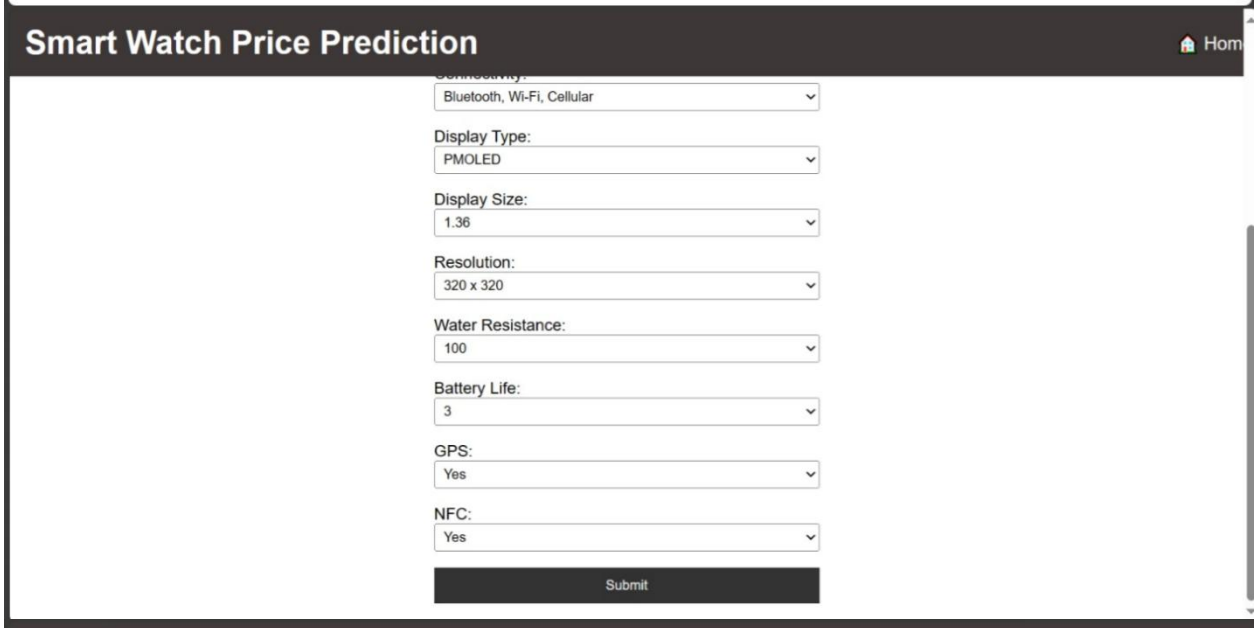
When you run the “main.py” file this window will open in the output terminal. Copy the http://127.0.0.1:5000 and paste this link in your browser.

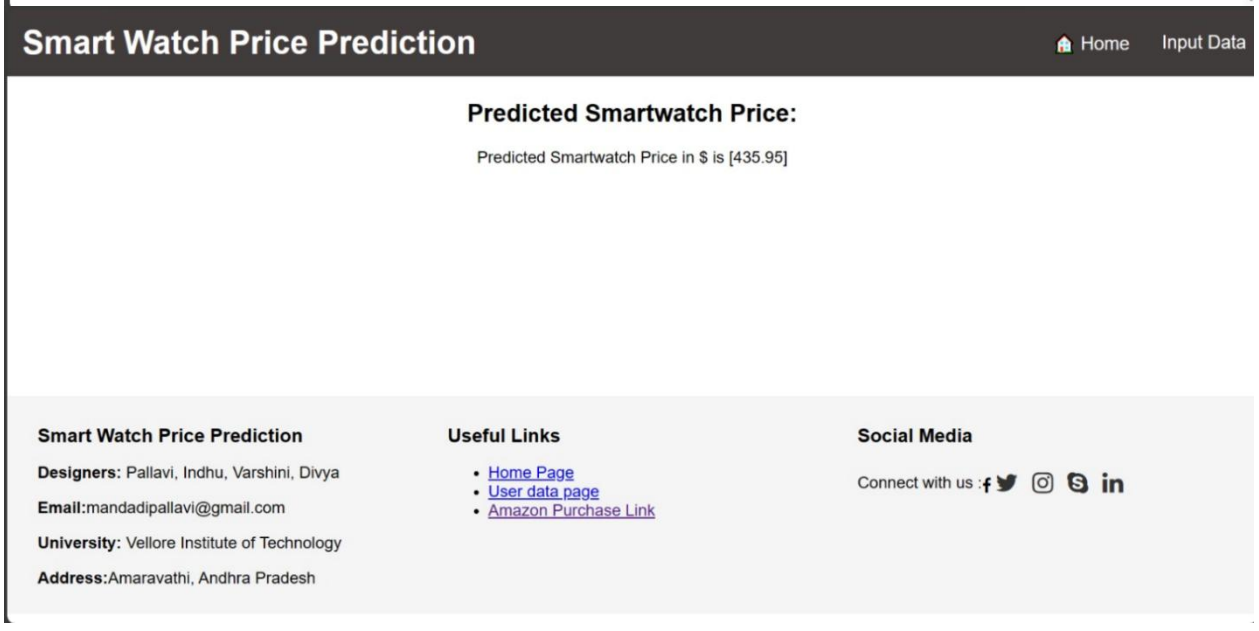


This is the "index.html" file that appears when we paste the URL into the browser. To proceed to the next page, click the 'Predict' button.




Smart Watch Price Prediction
Hom
Fill the features of Smart Watch
Watch Brand:
TicWatch
Watch Model:
Access Runway
Operating System:
Android Wear
Connectivity:
Bluetooth, Wi-Fi, Cellular
Display Type:
PMOLED
Display Size:
1.36
Resolution:
320 x 320


Smart Watch Price Prediction
Hom
Bluetooth, Wi-Fi, Cellular
Display Type:
PMOLED
Display Size:
1.36
Resolution:
320 x 320
Water Resistance:
100
Battery Life:
3
GPS:
Yes
NFC:
Yes
Submit


Smart Watch Price Prediction
Home
Input Data
Predicted Smartwatch Price:
Predicted Smartwatch Price in $ is [435.95]
Smart Watch Price Prediction
Designers: Pallavi, Indhu, Varshini, Divya
Email:mandadipallavi@gmail.com
University: Vellore Institute of Technology
Address:Amaravathi, Andhra Pradesh
Useful Links
Home Page
User data page
Amazon Purchase Link
Social Media
Connect with us :

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution.

Demo Link

https://drive.google.com/file/d/1fXVJUeoJjQdwQLATJiRjwBFix2SyPbo8/view?usp=sharing

Activity 2:- Project Documentation-Step by step project development procedure.

GitHub Link

https://github.com/smartinternz02/SI-GuidedProject-612118-1698753989