

**Smart Bridge internship training**  
**Vellore Institute of Technology**  
**School of Computer Science and Engineering**  
**Campus: Vellore** **Date: 30-06-2023**

**Smart Bridge internship program project**

**Project title: Smart watch price prediction**

**Team members:**

| S.no | Team members names: | Reg.no:   | Email address:                        |
|------|---------------------|-----------|---------------------------------------|
| 1    | Suriya Narayanan S  | 20MID0059 | suriyanarayanan.s2020@vitsudent.ac.in |
| 2    | Hariprasath R       | 20MID0197 | hariprasath.r2020@vitstudent.ac.in    |
| 3    | Vijay B             | 20MID0184 | vijay.b2020@vitstudent.ac.in          |
| 4    | Hanush Karthick V   | 20BCE2127 | hanushkarthick.v2020@vitstudent.ac.in |

**Introduction:**

**Overview:**

The initiative to anticipate the price of smartwatches uses past market data and machine learning algorithms to make its predictions. The initiative aims to offer useful information for buyers, manufacturers, and investors in the quickly developing smartwatch market by examining criteria like brand reputation, features, technological breakthroughs, and market trends. The project uses predictive modelling to improve decision-making and help

stakeholders make wise decisions about pricing strategies and investments.

### **Purpose:**

The goal of the smartwatch price prediction project is to forecast future pricing of smartwatches using machine learning techniques and market data analysis. This will help consumers and industry players make strategic decisions.

### **Problem statement and problem understanding (Literature Survey):**

#### **Business problem specification:**

Predicting the price of a smart watch poses several business challenges. Here are some key problems that may arise:

**Market Volatility:** The consumer electronics market, including smart watches, is highly dynamic and subject to rapid changes. New product launches, technological advancements, and competitive pricing strategies can significantly impact the pricing of smartwatches. Predicting prices accurately becomes challenging due to the constant market fluctuations.

**Multiple Variables:** Smartwatch prices are influenced by numerous factors, including brand reputation, features, materials, production costs, marketing expenses, profit margins, and demand-supply dynamics. Incorporating all these variables into a predictive model requires comprehensive data analysis and accurate forecasting techniques.

**Lack of Transparency:** Manufacturers often keep their pricing strategies and cost structures confidential. Obtaining reliable data on production costs, profit margins, and other relevant factors can be

difficult, especially for new or emerging brands. Limited access to this information may hinder the accuracy of price predictions.

### **Business requirements:**

- 1) A model that predicts a smart watch price with good accuracy while testing.
- 2) Users need to understand why this price for the data they give.
- 3) A very good looking user interface with less screen complexity.

### **Literature survey:**

The Smart watch Price Dataset contains information about the features and prices of popular smart watch models from various brands. The dataset includes columns such as Brand, Model, Operating System, Connectivity, Price (USD), Display Type, Display Size (inches), Resolution, Water Resistance (meters), Battery Life (days), Heart Rate Monitor, GPS, and NFC.

### **Social and business impact:**

**Social impact:** Users can understand the cost of each feature the watch holds and they can find the value of the smart watch price with the necessary features they need in the smart watch.

**Business Impact:** Companies can understand customer demands and features they want in their smart watch by surveying and this make the business to manufacture more edible products for the lifestyle of specified customer.

### **Approaches and solutions:**

**Dataset source:** Kaggle

Link: [Smart Watch prices | Kaggle](#)

- 1) **Data exploration:** Start by comprehending the Kaggle-obtained dataset. Examine its contents, paying special attention to the target variable (prices of smart watches), as well as the features (columns) and their descriptions. Note any category variables, missing values, or data pre-processing needs.
- 2) **Data pre-processing:** Cleaning and pre-processing the dataset will make it appropriate for using in machine learning model training. The management of missing values, encoding of categorical variables, normalisation of numerical features, and division of the data into training and test sets could all be part of this process.
- 3) **Feature Selection/Engineering:** Evaluate the features in the dataset to ascertain which ones are important for forecasting the cost of smart watches.
- 4) **Model Selection:** Select a suitable machine learning model for your task of price prediction. Regression models are typically appropriate when attempting to forecast a numerical number (such as the price of a smartwatch). Common options include decision trees, random forests, and linear regression. When choosing the model, take into account the dataset's complexity, quantity, and needs for interpretability.
- 5) **Model Training:** Split the pre-processed data into training and validation sets for the model. Train the chosen machine learning model on the target variable and features using the training set. To improve the performance of the model, alter its hyper parameters, often using grid search or cross-validation methods.
- 6) **Model Evaluation:** Determine the effectiveness of the trained model using suitable evaluation metrics, such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), or R-squared. Assess the model's generalizability and predictive accuracy using the validation set.

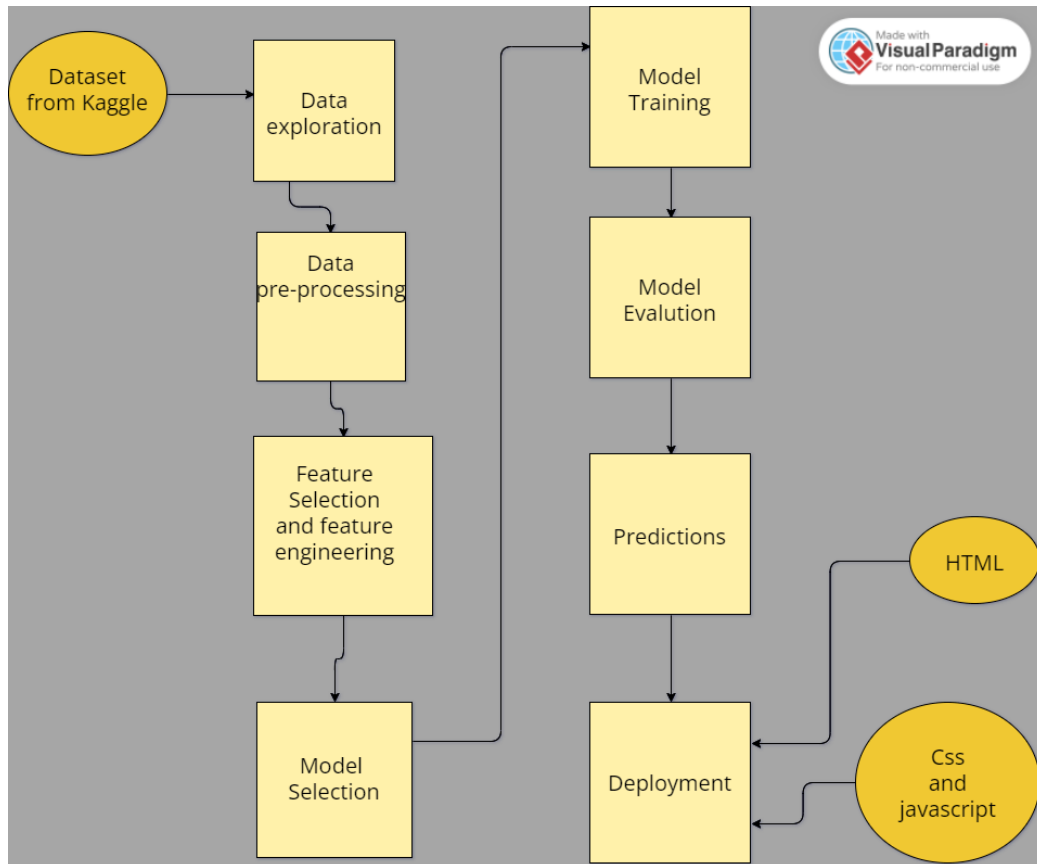
- 7) **Predictions:** Once you are satisfied with the model's performance, use it to make predictions on new, unseen data. Apply the same pre-processing steps used during training to pre-process the new data, and then feed it into the trained model to obtain price predictions for smart watches.
- 8) **Deployment:** Once everything is done go for model deployment using pickle package and flask package in python with necessary html files(minimum two; one to get input from user and another to display output).

### **Solutions:**

- 1) Using pandas package to import dataset from the local directory.
- 2) Using matplotlib package to visualize.
- 3) Data preprocess the dataset
- 4) Select suitable training model (Random Forest, Decision Tree or linear regression)
- 5) Test the model.
- 6) Deploy the model.

## Theoretical Analysis:

### Block Diagram for the project:



### Hardware and software requirements:

#### Hardware Requirements:

- 1) CPU: A contemporary processor that can handle the computational demands of developing machine learning models. For speedier processing, multi-core CPUs are preferred.
- 2) RAM: The dataset and model training procedure will fit in enough RAM. Depending on the size and complexity of the dataset, a different quantity may be needed, although 8GB or more is typically advised.
- 3) Storage: Enough room to keep the dataset, the programme, and any intermediate files produced throughout the project.

## Software requirements:

- 1) Python is a popular programming language for machine learning tasks. Install the most recent version of Python on your machine (for example, Python 3.9 or higher).
- 2) Development environment integrated (IDE): Select a Python IDE based on your personal tastes. Popular choices that offer a convenient environment for code development, execution, and data analysis include PyCharm, Jupyter Notebook, or Anaconda.
- 3) Libraries for machine learning: Install the necessary Python machine learning libraries, including:  
**NumPy**: For manipulating arrays and doing numerical calculations.  
**Pandas**: Used for data analysis and manipulation.  
A thorough machine learning library with a variety of tools and algorithms through **Scikit-learn**.  
To visualise data and produce charts for data analysis, use the data visualisation libraries **Matplotlib** or **Seaborn**.
- 4) Jupyter Notebooks (Optional): If you prefer an interactive environment for developing and documenting your code, install Jupyter Notebooks or JupyterLab.
- 5) Spyder software for running flask application with suitable web browser to run the flask application.

## Experimental investigation:

### Code with explanation:

#### Smart Watch price prediction

##### Importing basic required libraries

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Report: Importing libraries pandas, numpy, matplotlib and seaborn for data extraction and preprocessing

### Importing the smart watch price prediction dataset from the kaggle website

```
In [2]: df=pd.read_csv("Smart watch prices.csv")
```

```
In [3]: df
```

Out[3]:

|     | Brand    | Model          | Operating System | Connectivity               | Display Type | Display Size (inches) | Resolution | Water Resistance (meters) | Battery Life (days) |
|-----|----------|----------------|------------------|----------------------------|--------------|-----------------------|------------|---------------------------|---------------------|
| 0   | Apple    | Watch Series 7 | watchOS          | Bluetooth, Wi-Fi, Cellular | Retina       | 1.90                  | 396 x 484  | 50                        |                     |
| 1   | Samsung  | Galaxy Watch 4 | Wear OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.40                  | 450 x 450  | 50                        |                     |
| 2   | Garmin   | Venu 2         | Garmin OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.30                  | 416 x 416  | 50                        |                     |
| 3   | Fitbit   | Versa 3        | Fitbit OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.58                  | 336 x 336  | 50                        |                     |
| 4   | Fossil   | Gen 6          | Wear OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.28                  | 416 x 416  | 30                        |                     |
| ... | ...      | ...            | ...              | ...                        | ...          | ...                   | ...        | ...                       | ...                 |
| 374 | Withings | ScanWatch      | Withings OS      | Bluetooth, Wi-Fi           | PMOLED       | 1.38                  | 348 x 442  | 50                        |                     |
| 375 | Zepp     | Z              | Zepp OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.39                  | 454 x 454  | 50                        |                     |
|     |          | Watch OS       |                  | Bluetooth                  |              |                       |            |                           |                     |

Report: Importing the dataset from the local directory and storing it in the df variable



## Knowing more about the dataset

```
In [5]: df.shape
```

```
Out[5]: (379, 13)
```

```
In [6]: df.dtypes
```

```
Out[6]: Brand                object
Model                object
Operating System      object
Connectivity          object
Display Type          object
Display Size (inches)  float64
Resolution            object
Water Resistance (meters) object
Battery Life (days)   object
Heart Rate Monitor    object
GPS                   object
NFC                   object
Price (USD)           object
dtype: object
```

Report: Knowing the shape and data types of each column in the dataset

## Identify duplicate columns and dropping duplicate columns

```
In [7]: # Identify duplicate columns
duplicate_cols = df.columns.duplicated()

# Drop duplicate columns
df = df.loc[:, ~duplicate_cols]
df
```

```
Out[7]:
```

|     | Brand    | Model          | Operating System | Connectivity               | Display Type | Display Size (inches) | Resolution | Water Resistance (meters) | Battery Life (days) |
|-----|----------|----------------|------------------|----------------------------|--------------|-----------------------|------------|---------------------------|---------------------|
| 0   | Apple    | Watch Series 7 | watchOS          | Bluetooth, Wi-Fi, Cellular | Retina       | 1.90                  | 396 x 484  | 50                        |                     |
| 1   | Samsung  | Galaxy Watch 4 | Wear OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.40                  | 450 x 450  | 50                        |                     |
| 2   | Garmin   | Venu 2         | Garmin OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.30                  | 416 x 416  | 50                        |                     |
| 3   | Fitbit   | Versa 3        | Fitbit OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.58                  | 336 x 336  | 50                        |                     |
| 4   | Fossil   | Gen 6          | Wear OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.28                  | 416 x 416  | 30                        |                     |
| ... | ...      | ...            | ...              | ...                        | ...          | ...                   | ...        | ...                       | ...                 |
| 374 | Withings | ScanWatch      | Withings OS      | Bluetooth, Wi-Fi           | PMOLED       | 1.38                  | 348 x 442  | 50                        |                     |
| 375 | Zepp     | Z              | Zepp OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.39                  | 454 x 454  | 50                        |                     |

Report: Identify the duplicate columns and removing it

### finding the null values in the dataset

```
In [9]: df.isnull().sum()
```

```
Out[9]: Brand          1  
Model          1  
Operating System    3  
Connectivity       1  
Display Type       2  
Display Size (inches) 3  
Resolution         4  
Water Resistance (meters) 1  
Battery Life (days) 1  
Heart Rate Monitor  1  
GPS               1  
NFC               1  
Price (USD)       1  
dtype: int64
```

### finding the unique values in the dataset

```
In [10]: df.nunique()
```

```
Out[10]: Brand          42  
Model          137  
Operating System    35  
Connectivity       5  
Display Type       27  
Display Size (inches) 32  
Resolution         36  
Water Resistance (meters) 7  
Battery Life (days) 30  
Heart Rate Monitor  1  
GPS               2  
NFC               2  
Price (USD)       50  
dtype: int64
```

Report: Identifying the null and unique values in the dataset

**the column model is removed from the dataset**

```
In [11]: df.drop(columns='Model', inplace=True)
df.shape
```

```
Out[11]: (379, 12)
```

Report: The column model is removed

**Using regular expression we removed dollar symbol and "," symbol in the price column and converted the column to numeric**

```
In [12]: df['Price (USD)'] = df['Price (USD)'].replace('[$,]', '', regex=True)

# Convert the Price(USD) column to numeric values
df['Price (USD)'] = pd.to_numeric(df['Price (USD)'])
df.head()
```

```
Out[12]:
```

|   | Brand   | Operating System | Connectivity               | Display Type | Display Size (inches) | Resolution | Water Resistance (meters) | Battery Life (days) | Heart Rate Monitor |
|---|---------|------------------|----------------------------|--------------|-----------------------|------------|---------------------------|---------------------|--------------------|
| 0 | Apple   | watchOS          | Bluetooth, Wi-Fi, Cellular | Retina       | 1.90                  | 396 x 484  | 50                        | 18                  | Yes                |
| 1 | Samsung | Wear OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.40                  | 450 x 450  | 50                        | 40                  | Yes                |
| 2 | Garmin  | Garmin OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.30                  | 416 x 416  | 50                        | 11                  | Yes                |
| 3 | Fitbit  | Fitbit OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.58                  | 336 x 336  | 50                        | 6                   | Yes                |
| 4 | Fossil  | Wear OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.28                  | 416 x 416  | 30                        | 24                  | Yes                |

Report: Using regular expression we removed dollar symbol and "," symbol in the price column and converted the column to numeric

For the null values in the column of battery life we fill the most reoccurring value

```
In [16]: df['Battery Life (days)'] = df['Battery Life (days)'].fillna(df['Battery Life (days)'].mode())
```

```
In [17]: df.isnull().sum()
```

```
Out[17]: Brand                1
Operating System            3
Connectivity                1
Display Type                2
Display Size (inches)       3
Resolution                  4
Water Resistance (meters)    1
Battery Life (days)        5
Heart Rate Monitor          1
GPS                         1
NFC                         1
Price (USD)                 1
dtype: int64
```

similarly we do the same for the column display size and water resistance

```
In [18]: df['Display Size (inches)'] = df['Display Size (inches)'].fillna(df['Display Size (inches)'].mode())
```

```
In [19]: df.isnull().sum()
```

```
Out[19]: Brand                1
Operating System            3
Connectivity                1
Display Type                2
```

```
In [20]: df['Water Resistance (meters)'] = pd.to_numeric(df['Water Resistance (meters)'],errors='coerce')
```

```
In [21]: df.head()
```

```
Out[21]:
```

|   | Brand   | Operating System | Connectivity               | Display Type | Display Size (inches) | Resolution | Water Resistance (meters) | Battery Life (days) | Heart Rate Monitor | GPS | NFC | Price (USD) |
|---|---------|------------------|----------------------------|--------------|-----------------------|------------|---------------------------|---------------------|--------------------|-----|-----|-------------|
| 0 | Apple   | watchOS          | Bluetooth, Wi-Fi, Cellular | Retina       | 1.90                  | 396 x 484  | 50.0                      | 18.0                | Yes                | Yes | Yes | 399.0       |
| 1 | Samsung | Wear OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.40                  | 450 x 450  | 50.0                      | 40.0                | Yes                | Yes | Yes | 249.0       |
| 2 | Garmin  | Garmin OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.30                  | 416 x 416  | 50.0                      | 11.0                | Yes                | Yes | No  | 399.0       |
| 3 | Fitbit  | Fitbit OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.58                  | 336 x 336  | 50.0                      | 6.0                 | Yes                | Yes | Yes | 229.0       |
| 4 | Fossil  | Wear OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.28                  | 416 x 416  | 30.0                      | 24.0                | Yes                | Yes | Yes | 299.0       |

```
In [22]: df['Water Resistance (meters)'] = df['Water Resistance (meters)'].fillna(df['Water Resistance (meters)'].mode())
```

```
In [23]: df.isnull().sum()
```

```
Out[23]: Brand                1
Operating System            3
Connectivity                1
Display Type                2
Display Size (inches)       3
Resolution                  4
Water Resistance (meters)    2
Battery Life (days)        5
Heart Rate Monitor          1
GPS                         1
NFC                         1
Price (USD)                 1
```

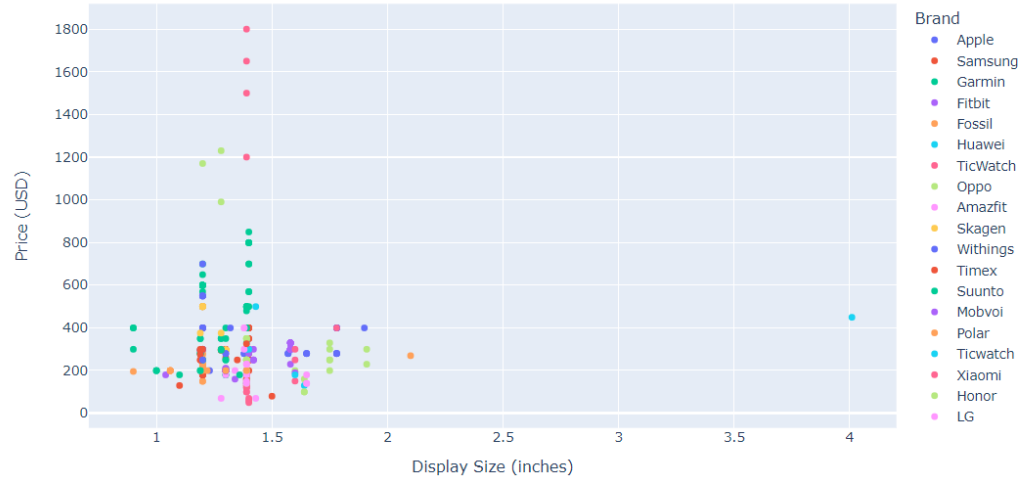
Report: For the null values in the column of battery life we fill the most reoccurring value and similarly we do the same for the column display size and water resistance

**Visualization:**

## Visualization of the dataset

### bi variate analysis

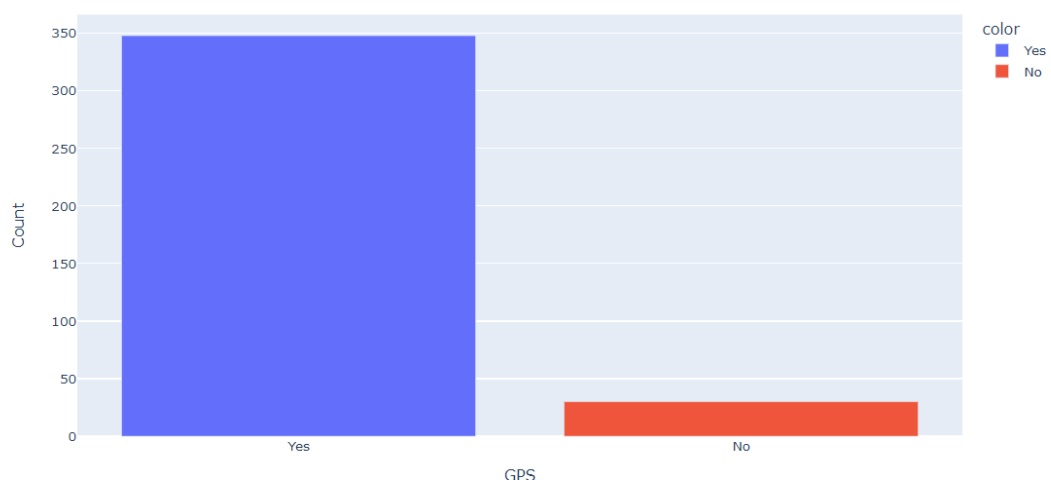
```
In [25]: import plotly.express as px  
fig=px.scatter(df, x='Display Size (inches)', y='Price (USD)', color='Brand')  
fig
```



Report: Scatter plot with watch brand to denote with display size as x axis and price as y axis

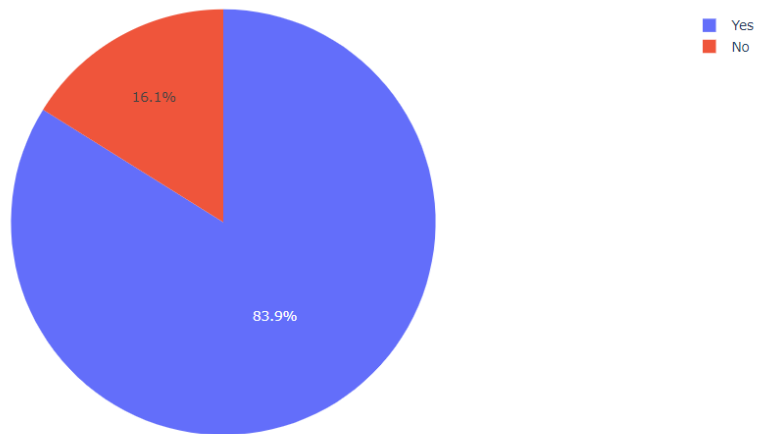
### univariate analysis

```
In [26]: gps_count = df['GPS'].value_counts()  
fig = px.bar(x=gps_count.index, y=gps_count.values, color=gps_count.index,  
            labels={'x': 'GPS', 'y': 'Count'})  
fig.show()
```



Report: Counting which watches as GPS facility using bar graph

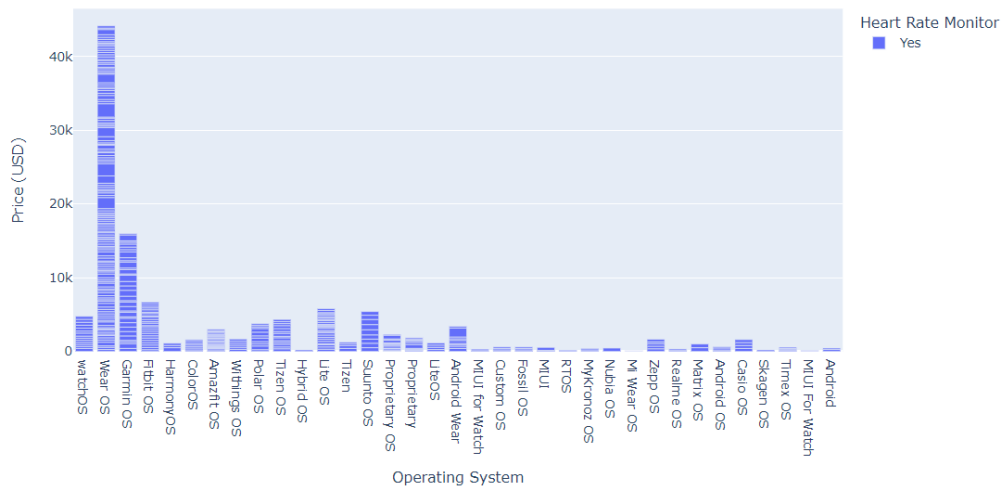
```
In [27]: nfc_count = df['NFC'].value_counts()
fig = px.pie(nfc_count, values=nfc_count.values, names=nfc_count.index)
fig.show()
```



## Report: Counting which watches as NFC facility using PIE chart

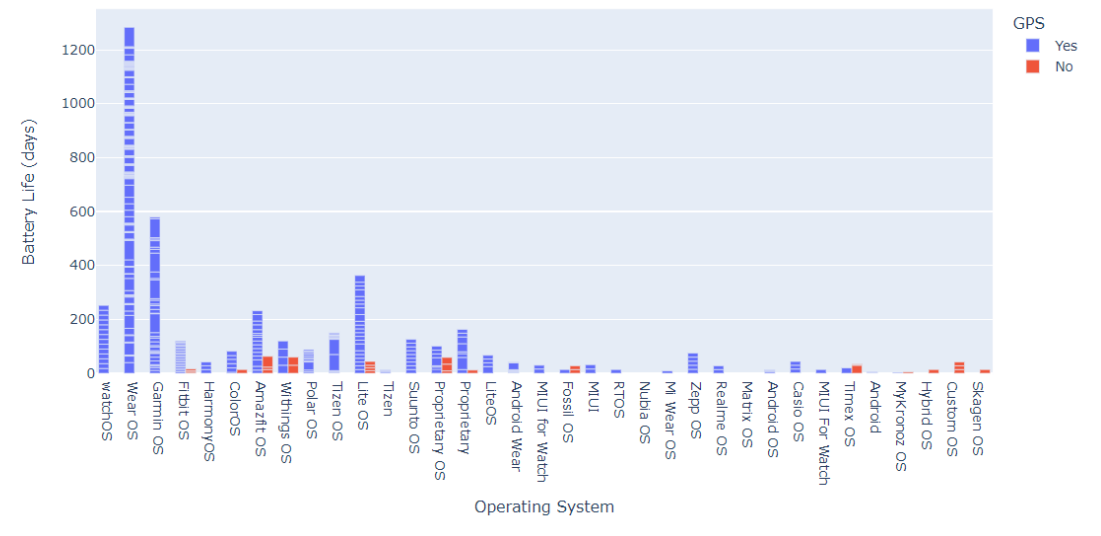
### Multivariate analysis

```
In [28]: fig = px.bar(df, x='Operating System', y='Price (USD)', color='Heart Rate Monitor', barmode='group')
fig.show()
```

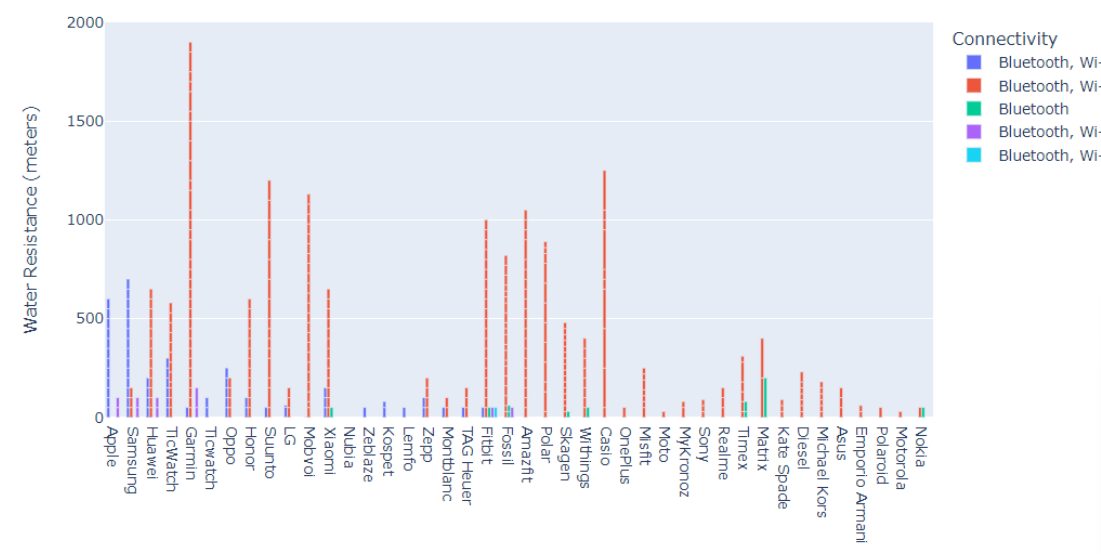


```
In [29]: fig = px.bar(df, x='Operating System', y='Battery Life (days)', color='GPS', barmode='group')
fig.show()
```

```
fig = px.bar(df, x='Operating System', y='Battery Life (days)', color='GPS', barmode='group')
fig.show()
```

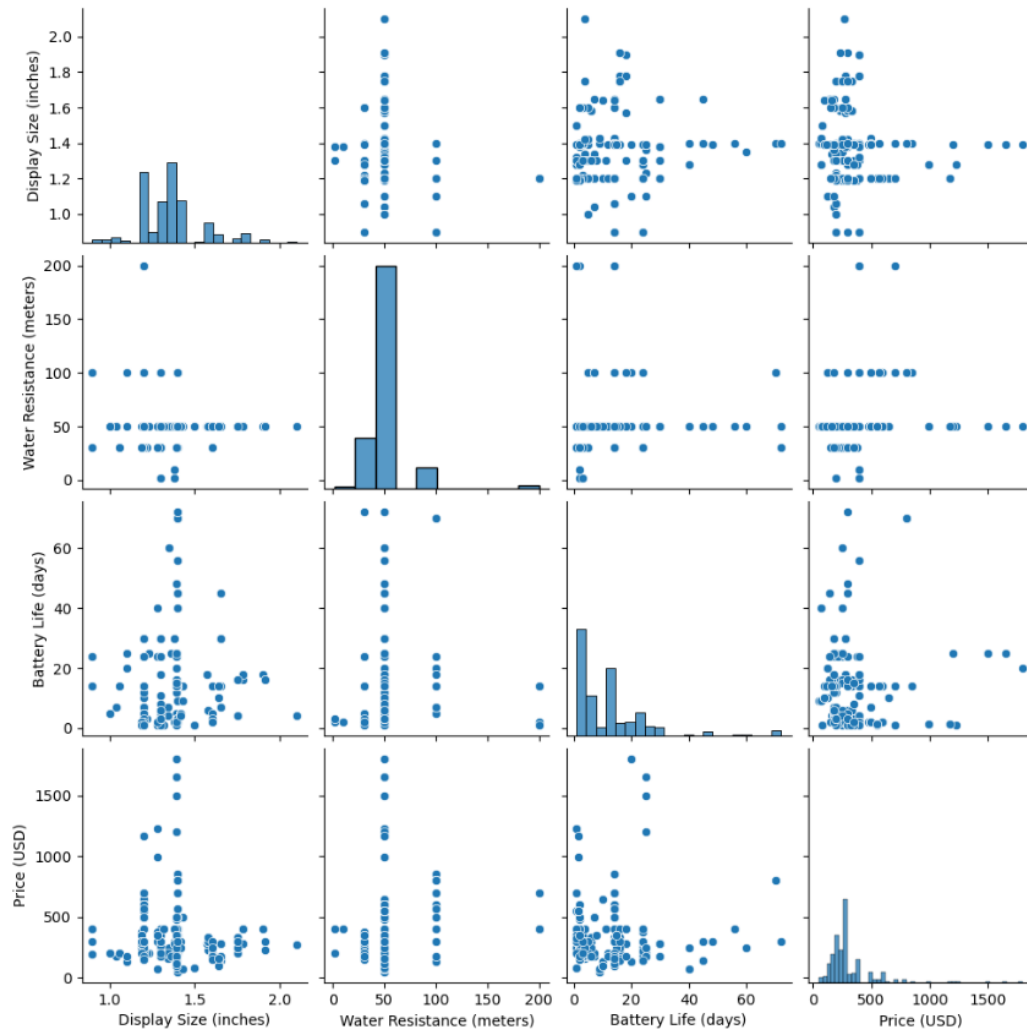


```
fig = px.bar(df, x='Brand', y='Water Resistance (meters)', color='Connectivity', barmode='group')
fig.show()
```



```
In [65]: sns.pairplot(df)
```

```
Out[65]: <seaborn.axisgrid.PairGrid at 0x196216ff910>
```



## Replacing and changing the datatype of the column for the model requirements

```
In [31]: # Replace "Unlimited" with infinity
df['Battery Life (days)'] = df['Battery Life (days)'].replace('Unlimited', df['Battery Life (days)'].max()+100)

# Convert column to numeric type
#df['Battery Life (days)'] = pd.to_numeric(df['Battery Life (days)'])
```

```
In [32]: df.isnull().sum()
```

```
Out[32]: Brand                1
Operating System             3
Connectivity                 1
Display Type                 2
Display Size (inches)        3
Resolution                   4
Water Resistance (meters)     2
Battery Life (days)          5
Heart Rate Monitor           1
GPS                           1
NFC                           1
Price (USD)                   1
dtype: int64
```

Ques: How can we replace the infinity value to particular value?

Report: Replacing the infinity value to particular value



## Constraining more values to suitable less values based on analysis

```
In [33]: def segment_resolution(resolution):  
         if pd.isnull(resolution):  
             return 'Unknown'  
         res = resolution.split(' x '  
         width = int(res[0])  
         height = int(res[1])  
         if width < 200 or height < 200:  
             return 'Low'  
         elif width < 400 or height < 400:  
             return 'Medium'  
         elif width < 800 or height < 800:  
             return 'High'  
         else:  
             return 'Very high'  
  
         df['Resolution'] = df['Resolution'].apply(segment_resolution)
```

```
In [34]: df.head()
```

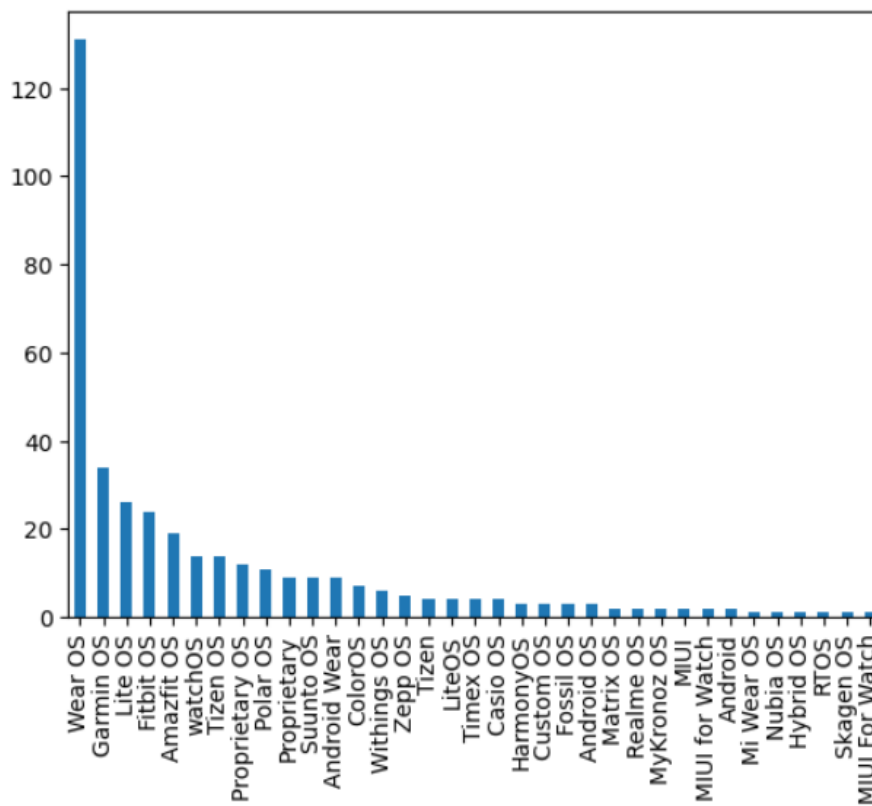
Out[34]:

|   | Brand   | Operating System | Connectivity               | Display Type | Display Size (inches) | Resolution | Water Resistance (meters) | Battery Life (days) | Heart Rate Monitor | GPS | NFC | Price (USD) |
|---|---------|------------------|----------------------------|--------------|-----------------------|------------|---------------------------|---------------------|--------------------|-----|-----|-------------|
| 0 | Apple   | watchOS          | Bluetooth, Wi-Fi, Cellular | Retina       | 1.90                  | Medium     | 50.0                      | 18.0                | Yes                | Yes | Yes | 399.0       |
| 1 | Samsung | Wear OS          | Bluetooth, Wi-Fi, Cellular | AMOLED       | 1.40                  | High       | 50.0                      | 40.0                | Yes                | Yes | Yes | 249.0       |
| 2 | Garmin  | Garmin OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.30                  | High       | 50.0                      | 11.0                | Yes                | Yes | No  | 399.0       |
| 3 | Fitbit  | Fitbit OS        | Bluetooth, Wi-Fi           | AMOLED       | 1.58                  | Medium     | 50.0                      | 6.0                 | Yes                | Yes | Yes | 229.0       |
| 4 | Fossil  | Wear OS          | Bluetooth, Wi-Fi           | AMOLED       | 1.28                  | High       | 30.0                      | 24.0                | Yes                | Yes | Yes | 299.0       |

Report : Changing unusable values to necessary values for model prediction

```
In [35]: df['Operating System'].unique()
df['Operating System'].value_counts().plot(kind='bar')
```

Out[35]: <Axes: >

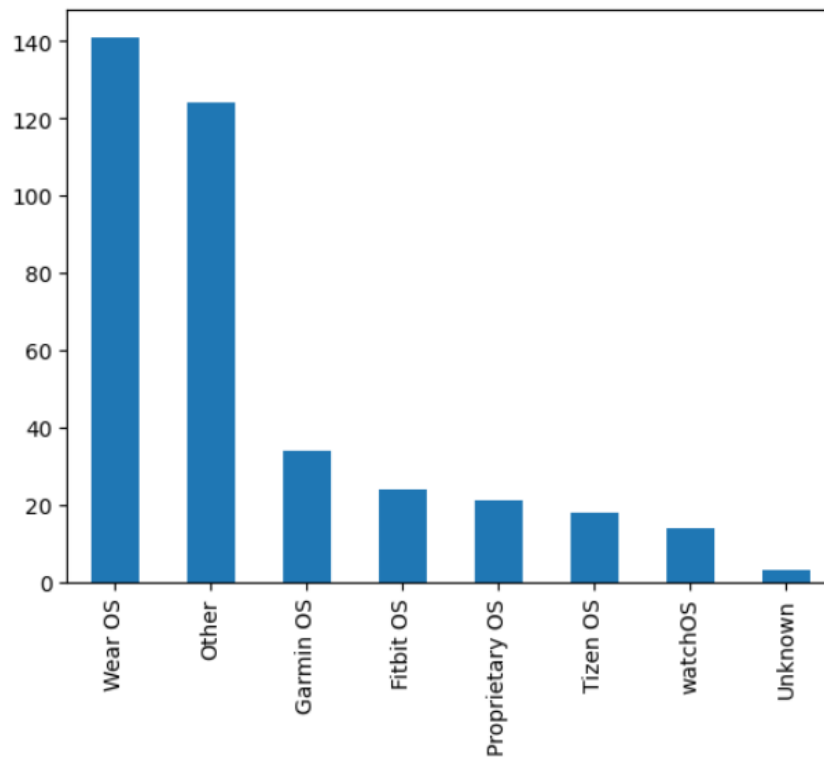


```
In [36]: def segment_os(os):
    if pd.isnull(os):
        return 'Unknown'
    elif 'watchOS' in os:
        return 'watchOS'
    elif 'Wear OS' in os or 'Android Wear' in os:
        return 'Wear OS'
    elif 'Garmin' in os:
        return 'Garmin OS'
    elif 'Fitbit' in os:
        return 'Fitbit OS'
    elif 'Tizen' in os:
        return 'Tizen OS'
    elif 'Proprietary' in os:
        return 'Proprietary OS'
    else:
        return 'Other'

df['Operating System'] = df['Operating System'].apply(segment_os)
```

```
In [37]: df['Operating System'].value_counts().plot(kind='bar')
```

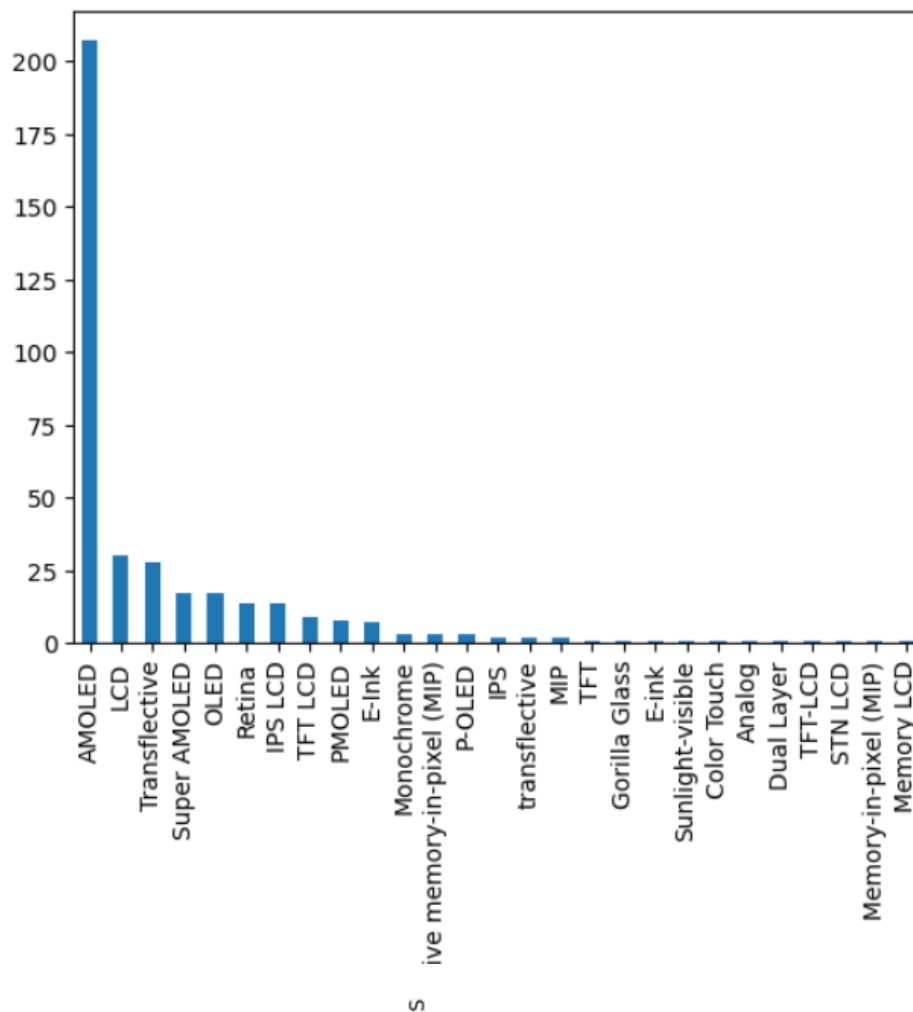
```
Out[37]: <Axes: >
```



Report: Since there are more OS unique values we reduce it to less values

```
In [38]: df['Display Type'].unique()
df['Display Type'].value_counts().plot(kind='bar')
```

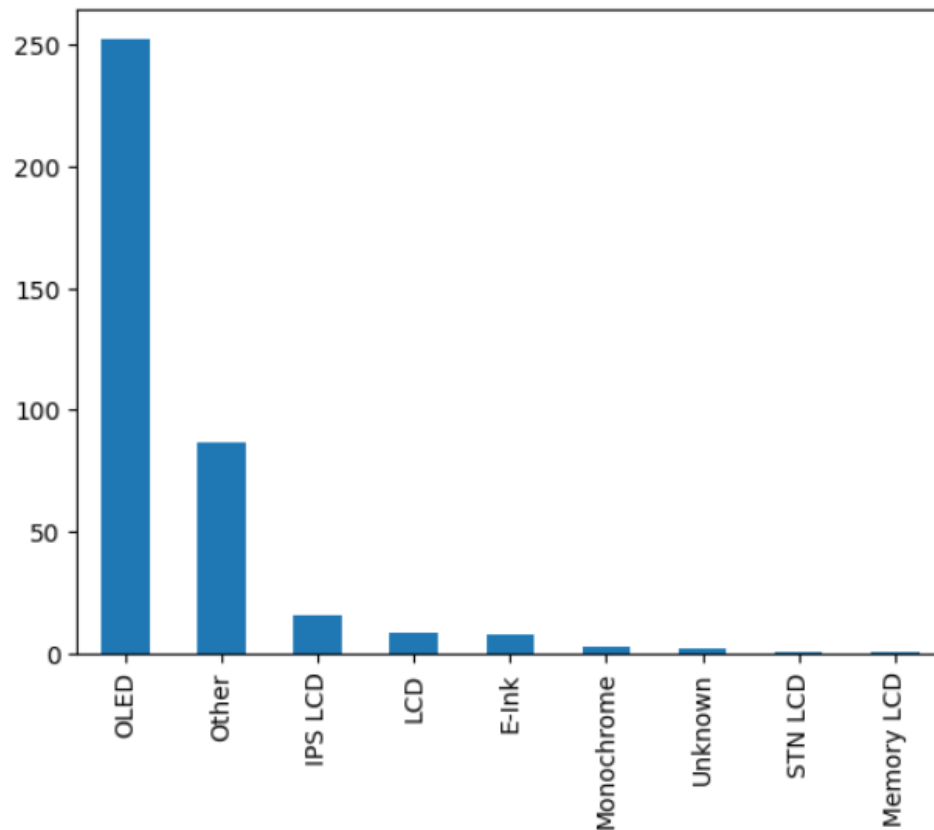
Out[38]: <Axes: >



```
In [39]: def segment_display_type(display_type):
    if pd.isnull(display_type):
        return 'Unknown'
    elif 'PAMOLED' in display_type:
        return 'AMOLED'
    elif 'IPS LCD' in display_type or 'IPS' in display_type:
        return 'IPS LCD'
    elif 'TFT LCD' in display_type:
        return 'LCD'
    elif 'OLED' in display_type:
        return 'OLED'
    elif 'E-Ink' in display_type or 'E-ink' in display_type:
        return 'E-Ink'
    elif 'STN LCD' in display_type:
        return 'STN LCD'
    elif 'Monochrome' in display_type:
        return 'Monochrome'
    elif 'Memory LCD' in display_type:
        return 'Memory LCD'
    else:
        return 'Other'
df['Display Type'] = df['Display Type'].apply(segment_display_type)
```

```
In [40]: df['Display Type'].value_counts().plot(kind='bar')
```

```
Out[40]: <Axes: >
```



Report: Since there are more Display type unique values we reduce it to less values

**we did maximum to fill the null valued rows and now we drop the rows as further preprocessing won't work from here**

```
In [41]: df.dropna(inplace=True)
```

```
In [42]: df.isnull().sum()
```

```
Out[42]: Brand                0
Operating System             0
Connectivity                 0
Display Type                 0
Display Size (inches)        0
Resolution                   0
Water Resistance (meters)     0
Battery Life (days)          0
Heart Rate Monitor            0
GPS                           0
NFC                           0
Price (USD)                   0
dtype: int64
```

Report: we did maximum to fill the null valued rows and now we drop the rows as further preprocessing won't work from here

## columns of object type

```
In [44]: col=[x for x in df.columns if df[x].dtypes=='O' ]
```

```
In [45]: col
```

```
Out[45]: ['Brand',  
          'Operating System',  
          'Connectivity',  
          'Display Type',  
          'Resolution',  
          'Heart Rate Monitor',  
          'GPS',  
          'NFC']
```

```
In [46]: df.shape
```

```
Out[46]: (372, 12)
```

## Report: Columns of object type

### splitting the dataset here as independent variable and dependent variable

```
In [47]: X = df.drop(['Price (USD)'], axis=1)## independent variable  
        y = df['Price (USD)'] ##dependent variable
```

## Report: splitting the dataset here as independent variable and dependent variable

## importing necessary libraries to train the model

```
In [48]: from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import pickle as pl
```

## One hot encoding is performed

```
In [49]: enc = OneHotEncoder(handle_unknown='ignore')
q=enc.fit(X)
refreshed_df = q.transform(X).toarray()
```

```
In [50]: refreshed_df
```

```
Out[50]: array([[0., 1., 0., ..., 1., 0., 1.],
 [0., 0., 0., ..., 1., 0., 1.],
 [0., 0., 0., ..., 1., 1., 0.],
 ...,
 [0., 0., 0., ..., 1., 0., 1.],
 [0., 0., 0., ..., 0., 0., 1.],
 [0., 0., 0., ..., 1., 0., 1.]])
```

Report: Performing one hot encoding to the categorical columns

## splitting the dataset into training and testing

```
In [51]: X_train,X_test,y_train,y_test=train_test_split(refreshed_df,y,test_size=0.32)
```

## Random Forest regression is used here..

### Training the model

```
In [52]: dtree = RandomForestRegressor()
dtree.fit(X_train, y_train)
```

```
Out[52]: ▼ RandomForestRegressor
RandomForestRegressor()
```

Report: Performing train test split and training the random forest regression model

Random forest is used here because Random Forest reduces overfitting by averaging multiple decision trees and is less sensitive

to noise and outliers in the data so we get good accuracy and robustness

## **predicting the result**

```
In [53]: #X_test=np.array(X_test).reshape(1, -1)
y_pred = dtree.predict(X_test)
y_pred
```

```
Out[53]: array([ 268.79619048, 298.          , 243.74619048, 291.3          ,
 295.58666667, 223.08657143, 288.26380952, 179.20809524,
 237.66888889, 190.7025          , 278.74166667, 199.          ,
 457.22761905, 268.79619048, 298.8          , 277.71          ,
 543.18690476, 385.93333333, 648.01857143, 374.95510823,
 217.61047619, 221.81848901, 310.41          , 201.48222222,
 243.74619048, 295.          , 464.17333333, 288.78857143,
 233.42347222, 264.52          , 351.85047619, 252.00566667,
 243.09833333, 294.92666667, 187.91166667, 209.95142857,
 234.87666667, 219.61416667, 464.41          , 464.17333333,
 493.44666667, 781.5          , 253.33333333, 301.46666667,
 295.          , 479.7947619 , 590.6          , 178.69666667,
 301.46666667, 320.60952381, 512.64690476, 295.          ,
 221.81848901, 209.95142857, 436.56095238, 132.335          ,
 506.11190476, 222.32          , 245.62285714, 266.38813131,
 930.05          , 174.87357143, 374.95510823, 209.95142857,
 185.28680952, 217.61047619, 303.34666475, 264.52          ,
 197.97333333, 496.33333333, 295.06733333, 235.075          ,
 265.11666667, 300.04166667, 262.05          , 303.34666475,
 277.02          , 374.95510823, 281.7047619 , 468.08666667,
 209.95142857, 194.          , 245.72366545, 448.08857143,
 193.56666667, 184.87          , 303.34666475, 281.7047619 ,
 222.32          , 255.40590909, 338.37857143, 543.18690476,
 209.95142857, 297.40666667, 145.63333333, 1319.06          ,
 205.12181818, 221.81848901, 337.4452381 , 171.07457431,
```

Report: predicting the result of x\_test

```
300.11190476, 200.50952381, 220.11190476, 200.11190476, 17
```

## **finding the accuracy score**

```
In [54]: mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Random Forest Regression MSE: {:.2f}, R-squared: {:.2f}'.format(mse, r2))
```

```
Random Forest Regression MSE: 5690.16, R-squared: 0.83
```

**Here we get the accuracy score 82-84 which is pretty good**



### finding the accuracy score

```
In [122]: mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Random Forest Regression MSE: {:.2f}, R-squared: {:.2f}'.format(mse, r2))

Random Forest Regression MSE: 5690.16, R-squared: 0.83
```

```
In [123]: def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:.2f}%'.format(accuracy))
    return accuracy
base_accuracy = evaluate(dtree, X_test, y_test)

Model Performance
Average Error: 48.3426 degrees.
Accuracy = 81.92%.
```

Here we get the accuracy score 80-84 which is pretty good

Report: we get 83% accuracy for this.

Hyper parameter tuning:

from here we use other algorithms and hyper paramter tuning for more better prediction

```
In [126]: from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
lr = LinearRegression()
lr.fit(X_train, y_train)

# Train the decision tree regression model
dtc = DecisionTreeRegressor(random_state=42)
dtc.fit(X_train, y_train)

# Evaluate the performance of the models
print('Linear Regression R-squared:', lr.score(X_test, y_test))
print('Decision Tree Regression R-squared:', dtc.score(X_test, y_test))
print('Decision Tree Regression accuracy:', evaluate(dtree, X_test, y_test))
print('Linear Regression accuracy:', evaluate(lr, X_test, y_test))

Linear Regression R-squared: -6.651000324480807e+24
Decision Tree Regression R-squared: 0.8174988772476782
Model Performance
Average Error: 48.3426 degrees.
Accuracy = 81.92%.
Decision Tree Regression accuracy: 81.91950941853196
Model Performance
Average Error: 148941848655498.3438 degrees.
Accuracy = -95454635116331.20%.
Linear Regression accuracy: -95454635116331.2
```

Report: Linear regression has less accuracy and decsion tree is good but Random forest performs well

```
In [54]: gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=42)
gbr.fit(X_train,y_train)
```

```
Out[54]: GradientBoostingRegressor
GradientBoostingRegressor(max_depth=1, random_state=42)
```

```
In [55]: xgb=XGBRegressor(estimators=1800, learning_rate=0.06, max_depth=2, subsample=0.7, colsample_bytree=0.4, colsample_bylevel = 0.5,
max_leaves=3, random_state=1)
xgb.fit(X_train, y_train)
```

[14:33:13] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:  
Parameters: { "estimators" } are not used.

```
Out[55]: XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
colsample_bylevel=0.5, colsample_bynode=None, colsample_bytree=0.4,
early_stopping_rounds=None, enable_categorical=False,
estimators=1800, eval_metric=None, feature_types=None, gamma=None,
gpu_id=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,
n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

```
In [56]: print('gradient descent Regression accuracy:', evaluate(gbr, X_test, y_test))
print('xgb Regression accuracy:', evaluate(xgb, X_test, y_test))
```

Model Performance  
Average Error: 74.9756 degrees.  
Accuracy = 69.78%.  
gradient descent Regression accuracy: 69.7819212577608  
Model Performance  
Average Error: 78.1147 degrees.  
Accuracy = 69.98%.  
xgb Regression accuracy: 69.9836575833441

For gradient boosting and xgboost we get accuracy of 70% so we neglect that and go for random forest

**we use hyper parameter tuning on random forest regressor as gives better accuracy and r2 score**

```
In [140]: from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [200]
# Number of features to consider at every split
max_features = [25]
# Maximum number of levels in tree
max_depth = [100]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [25]
# Minimum number of samples required at each leaf node
min_samples_leaf = [24]
# Method of selecting samples for training each tree
bootstrap = [False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

print(random_grid)

{'n_estimators': [200], 'max_features': [25], 'max_depth': [100, None], 'min_samples_split': [25], 'min_samples_leaf': [24], 'bootstrap': [False]}
```

```
In [141]: rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=
# Fit the random search model
rf_random.fit(X_train, y_train)
```

C:\Users\SURIYA\AppData\Local\anaconda3\lib\site-packages\sklearn\model\_selection\\_search.py:305: UserWarning:  
The total space of parameters 2 is smaller than n\_iter=100. Running 2 iterations. For exhaustive searches, use GridSearchCV.

Fitting 3 folds for each of 2 candidates, totalling 6 fits

```
Out[141]: RandomizedSearchCV
  estimator: RandomForestRegressor
    RandomForestRegressor
```

```
In [142]: base_model = RandomForestRegressor(n_estimators = 200, random_state = 42)
base_model.fit(X_train, y_train)
pred = base_model.predict(X_test)
r2 = r2_score(y_test, pred)
print("R2-score:", r2)
print("The accuracy after hyper parameter:", evaluate(base_model, X_test, y_test))

R2-score: 0.8241066654824716
Model Performance
Average Error: 49.1503 degrees.
Accuracy = 81.36%.
The accuracy after hyper parameter: 81.35733098790294
```

Report: Here we perform hyper parameter tuning and after that regression score is same.

**Now we take a random row out of the dataset to test it with pickle folder**

```
In [55]: try1=df.iloc[5]
try1=try1.drop(['Price (USD)'])
try1
```

```
Out[55]: Brand      Huawei
Operating System    Other
Connectivity        Bluetooth, Wi-Fi, Cellular
Display Type        OLED
Display Size (inches)  1.43
Resolution          High
Water Resistance (meters)  50.0
Battery Life (days)    14.0
Heart Rate Monitor     Yes
GPS                   Yes
NFC                   Yes
Name: 5, dtype: object
```

```
In [56]: try1=pd.DataFrame(try1.T)
```

```
In [57]: try2=try1.T
try2
```

### **dumping the one hot encoder as model1.pkl and dtree as model2.pkl**

```
In [58]: pl.dump(q,open("model1.pkl","wb"))
```

```
In [59]: pl.dump(dtrees,open("model2.pkl","wb"))
```

### **Loading the model 1**

#### **Transforming the try2 row and encoding it**

```
In [60]: m=pl.load(open("model1.pkl","rb"))
```

```
In [61]: f=m.transform(try2)
```

```
In [62]: f
```

```
Out[62]: <1x135 sparse matrix of type '<class 'numpy.float64'>'
          with 11 stored elements in Compressed Sparse Row format>
```

### **Loading the model 2**

#### **predicting what value we get for try2**

```
In [63]: m2=pl.load(open("model2.pkl","rb"))
```

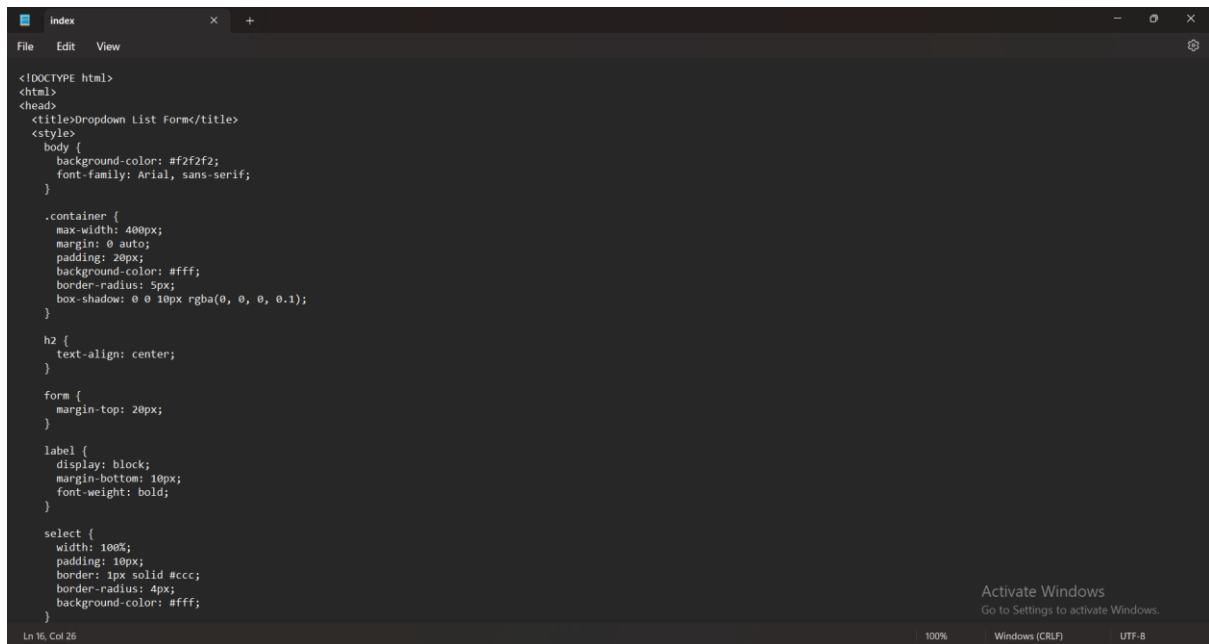
```
In [64]: m2.predict(f)
```

```
Out[64]: array([285.19833333])
```

Report: Model deployment using pickle

Html code:

Index.html



```
<!DOCTYPE html>
<html>
<head>
  <title>Dropdown List Form</title>
  <style>
    body {
      background-color: #f2f2f2;
      font-family: Arial, sans-serif;
    }

    .container {
      max-width: 400px;
      margin: 0 auto;
      padding: 20px;
      background-color: #fff;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
      text-align: center;
    }

    form {
      margin-top: 20px;
    }

    label {
      display: block;
      margin-bottom: 10px;
      font-weight: bold;
    }

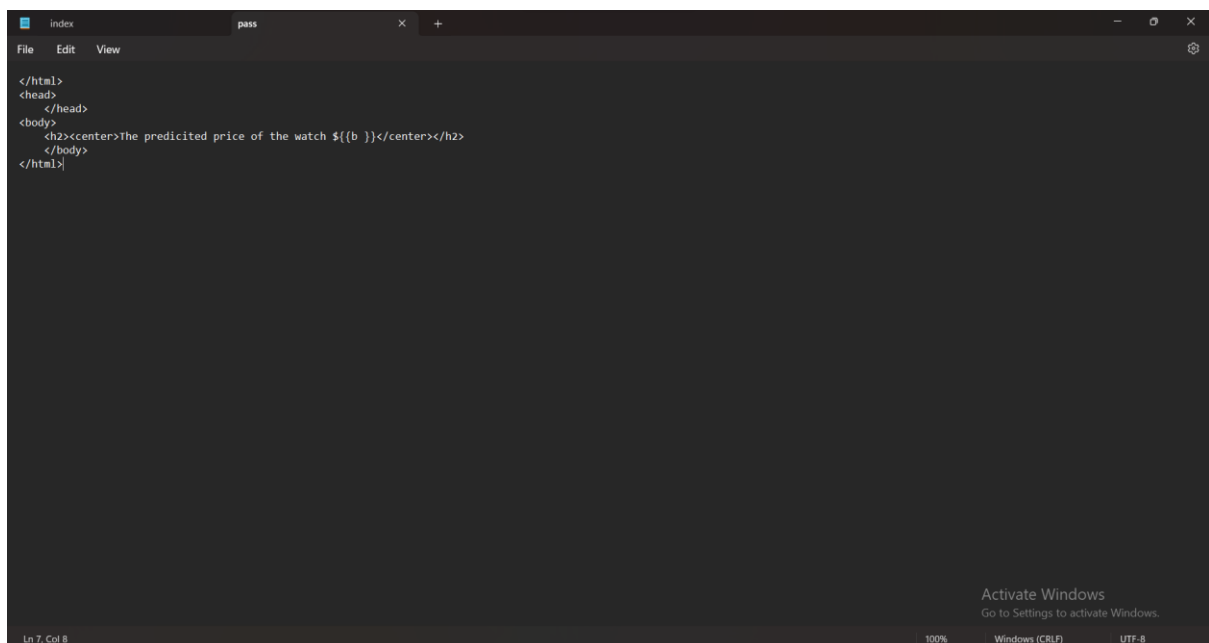
    select {
      width: 100%;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 4px;
      background-color: #fff;
    }
  </style>
</head>
</html>
```

Ln 16, Col 26

100% Windows (CRLF) UTF-8

Activate Windows  
Go to Settings to activate Windows.

## Pass.html



```
</html>
<head>
</head>
<body>
  <h2><center>The predicted price of the watch ${{b }}</center></h2>
</body>
</html>
```

Ln 7, Col 8

100% Windows (CRLF) UTF-8

Activate Windows  
Go to Settings to activate Windows.

## Flask code:

from flask import Flask, render\_template, request, redirect, url\_for

```
import pickle as pl

import pandas as pd

app=Flask(__name__,template_folder='template')


@app.route('/')

def index():

    return render_template('index.html')


@app.route('/',methods=['POST'])

def getvalue():

    brand=request.form['brand']

    os=request.form['os']

    conn=request.form['conn']

    displayType=request.form['dt']

    displaySize=request.form['ds']

    res=request.form['res']

    waRes=request.form['wres']

    bL=request.form['bl']

    rMo=request.form['hr']

    gps=request.form['gps']

    nfc=request.form['nfc']
```

```
df2=pd.DataFrame({"Brand":brand,
                  "Operating System":os,
                  "Connectivity":conn,
                  "Display Type":displayType,
                  "Display Size (inches)":displaySize,
                  "Resolution":res,
                  "Water Resistance (meters)":waRes,
                  "Battery Life (days)":bL,
                  "Heart Rate Monitor":rMo,
                  "GPS":gps,
                  "NFC":nfc},index=[0])

mod1=pl.load(open("model1.pkl","rb"))
mod2=pl.load(open("model2.pkl","rb"))

f=mod1.transform(df2)

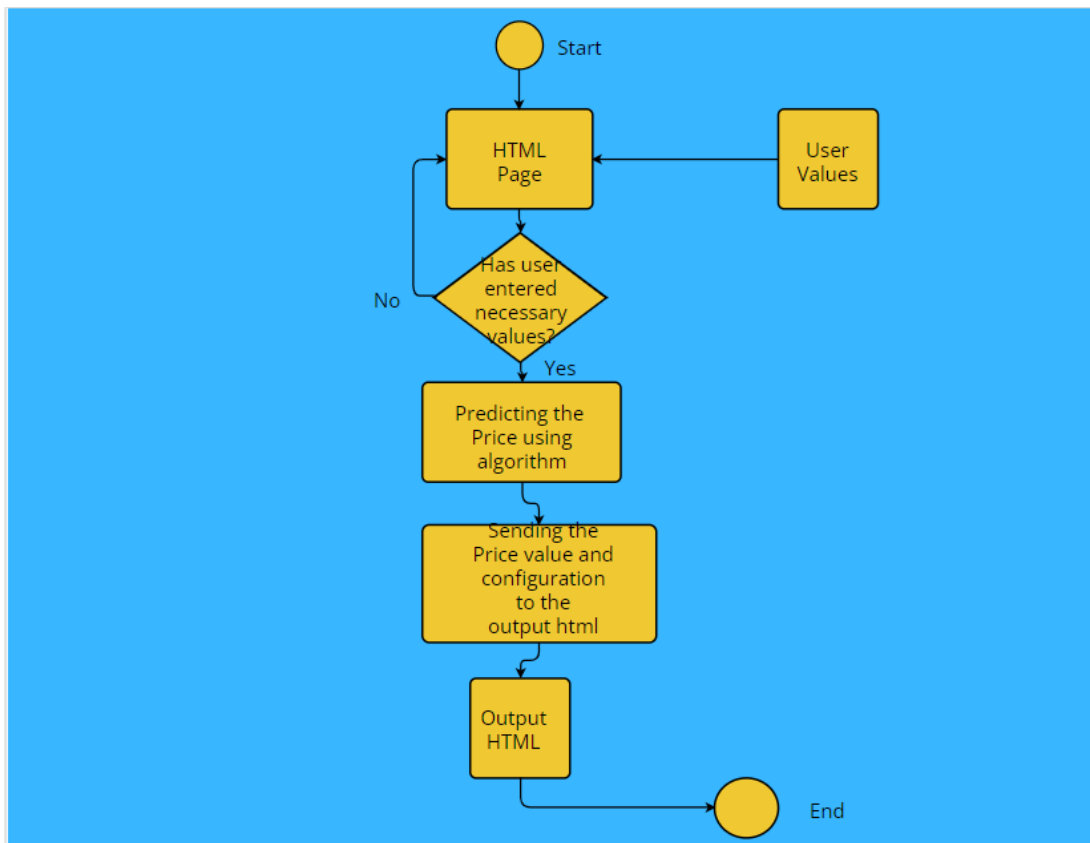
out=mod2.predict(f)

return render_template('pass.html',
                       bran=brand,
                       o=os,
                       con=conn,
                       displayTyp=displayType,
                       displaySiz=displaySize,
```

```
re=res,  
waRe=waRes,  
bLk=bL,  
rM=rMo,  
gp=gps,  
nf=nfc,  
b=round(out[0]))
```

```
if __name__=="__main__":  
    app.run(debug=True)
```

### Flowchart for solution:





**Result:**

**Output-1:**

**Smart Watch Price Predictor**

**Know your Watch Price**

Your Brand:

Your OS:

Connectivity:

Display Type:

Display Size (inches):

Resolution:

Water Resistance (meters):

Battery Life (days):

Heart Rate Monitor:

GPS:

NFC:

**Submit**

Activate Windows  
Go to Settings to activate Windows

**Output-2:**

**Brand:**

**Operating System:Wear OS**

**Connectivity:Bluetooth, Wi-Fi**

**Display Type:OLED**

**Display Size (inche):**

**Resolution:Medium**

**Water Resistance (meters):50.0**

**Battery Life (days):20**

**Heart Rate Monitor:Yes**

**GPS:Yes**

**NFC:Yes**

**The predicted price of the watch**  
**\$292**

Activate Windows  
Go to Settings to activate Windows

The price(output) of the smart watch is predicted and delivered in the final page.

## **Advantages and Disadvantages:**

### **Advantages of Random Forest Regressor model in this dataset:**

- 1) Robustness to Outliers: When compared to some other regression algorithms, the Random Forest Regressor is less susceptible to outliers. As a result of using numerous decision trees, the overall forecast is less affected by outliers.
- 2) Non-linearity Handling: The method is capable of effectively capturing non-linear correlations between the target variable and the features. It is capable of simulating intricate data relationships and patterns without the need for explicit feature engineering.
- 3) The Random Forest Regressor offers a measure of feature importance that enables you to comprehend the relative value of each feature in predicting the price of smartwatches. The selection and engineering of features can both benefit from this information.
- 4) Multiple decision trees are combined to create predictions using the ensemble learning technique known as the Random Forest Regressor. To lessen overfitting, several trees are averaged.

### **Disadvantages of Random Forest Regressor model in this dataset:**

- 1) Interpretability: Although the Random Forest Regressor offers strong predictive accuracy, its interpretability may be inferior to that of more straightforward linear regression models. It can

be difficult to pinpoint the precise reasoning that goes into creating each prediction.

- 2) Overfitting: The Random Forest Regressor might still experience overfitting if there are too many trees in the forest or if the hyperparameters are not properly calibrated, despite the fact that it helps to minimise overfitting to some extent.
- 3) Costly to Run: Running a Random Forest Regressor with a lot of decision trees during training can be costly and time-consuming, especially for big datasets. When compared to simpler models, inference and prediction time can also be somewhat slower.
- 4) Memory Usage: The Random Forest Regressor can use a lot of memory, particularly when working with big datasets or lots of trees.

#### **Advantages of Decision Tree Regressor model in this dataset:**

- 1) Non-linearity Handling: Decision trees can capture non-linear relationships between features and the target variable. They are capable of modeling complex interactions and patterns in the data without requiring explicit feature engineering.

#### **Disadvantages of Decision Tree Regressor model in this dataset:**

- 1) Lack of Global Optimality: Decision trees make locally optimal decisions at each node, which may not necessarily lead to the best overall split. Consequently, they may not always find the globally optimal solution.
- 2) High Variance: Decision trees can be highly sensitive to small changes in the training data, resulting in high variance. This can make the model unstable and lead to different predictions with slight variations in the dataset.

### **Disadvantages of Linear Regressor model in this dataset:**

- 1) A linear relationship between the input features and the target variable is assumed by linear regression. If the relationship is non-linear, the model might not effectively represent the underlying patterns, producing predictions that are less than ideal.
- 2) Limited Flexibility: When it comes to capturing complex correlations between variables, linear regression models are only partially flexible. Intricate interactions, non-linearities, or higher-order effects in the data may be difficult for them to detect.
- 3) Sensitivity to Irrelevant Features: Linear Regression considers all input features as contributing to the prediction. If there are irrelevant or redundant features in the dataset, they can introduce noise and lead to overfitting or suboptimal performance.
- 4) Lack of Robustness to Non-Normality: Linear Regression assumes that the residuals (the differences between the predicted and actual values) follow a normal distribution. If the residuals deviate significantly from normality, it can impact the reliability of the model's predictions.

### **Overall advantages and disadvantages:**

- 1) Here omitted the model column which may would have affected the model.
- 2) Here I reduced unnecessary row values which is of greater convenience for data analysis and predicting.
- 3) Dragged mode values instead of null values in the categorical column.

## **Applications:**

- 1) Pricing Advice: Manufacturers or sellers of smartwatches can use the model to determine the right prices for upcoming or current smartwatch models. For pricing advice, it can take into account factors like specifications, brand, design, and market trends.
- 2) Market analysis: The model can help with market analysis by forecasting smartwatch pricing. It can assist in identifying price patterns, comprehending the effects of various features on cost, and determining how competitively priced the market's offerings for smartwatches are.
- 3) Portfolio Management: The predictions of the model can aid in the management of a smartwatch portfolio. Prior to their release, it can help determine the viability and possible commercial success of new smartwatch models.
- 4) Evaluation of price Strategies: The model in use makes it possible to assess various price plans and their potential effects on market share, sales, and profitability. In order to evaluate the effects of various pricing decisions, it may simulate a variety of situations.

## **Conclusion:**

Now users can predict their own smart watch prices using the model deployed in the respective browsers.

## **Futures scope:**

- 1) E-commerce Pricing: Using the concept, online sellers can establish wristwatch prices that are based on the current state of the market, rivals' asking prices, and consumer demand. It can aid in pricing optimisation to draw clients and boost sales.

2) Evaluation of the Secondary Market: The model's projections can help determine the secondary market worth of reconditioned or used smartwatches. It can determine fair prices by taking into account elements like age, condition, and pertinent attributes.