SMART WATCH PRICE PRIDICTION

AN INDUSTRY ORIENTED MINI REPORT

Submitted to

JAWAHARLAL NEHRU TECNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

KONKUMUTTI PRANAYA	**21UK1A05L6**
ADAPA NIKHIL	**21UK1A05Q8**
MOHAMMAD SAFFURA THAZEEN	**21UK1A05K8**
BALGURI PAVANI	**21UK1A05M0**

Under the guidance of

DR. R NAVEEN KUMAR

Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

Affiliated to JNTUH, HYDERABAD

BOLLIKUNTA, WARANGAL (T.S) – 506005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE(WARANGAL)

CERTIFICATE OF COMPLETION
INDUSTRY ORIENTED MINI PROJECT

This is to certify that the UG Project Phase-1 entitled "**SMART WATCH PRICE PRIDICTION**" is being submitted by KONKUMUTTI PRANAYA (**21UK1A05L6**) ADAPA NIKHI (21UK1A05Q8) MOHAMMAD SAFFURA THAZEEN (21UK1A05K8) BALGURI PAVANI (21UK1A05M0) in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering to Jawaharlal Nehru Technological University Hyderabad during the academic year 2024- 2025.

Project Guide	**HOD**
Dr R. Naveen Kumar	Dr R. Naveen Kumar
(Professor)	(Professor)

External

ACKNOWLEDGEMENT

- We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved D**r Syed Musthak Ahmed,** Principal, Vaagdevi Engineering College for making us available all the required assistance and for this support and inspiration to carry out this UG Project Phase-1 in the institute.
- We extend our heartfelt thanks to **Dr R. NaveenKumar**, Head of the Department of CSE, Vaagdevi Engineering College for providing us necessary infrastructure and thereby giving us freedom to carry out the UG Project Phase-1.
- We express heartfelt thanks to Smart Bridge Educational Services Private Limited, for their constant supervision as well as for providing necessary information regarding the UG Project Phase-1 and for their support in completing the UG Project Phase-1.

- We express heartfelt thanks to Smart Bridge Educational Services Private Limited, for their constant supervision as well as for providing necessary information regarding the UG Project Phase-1 and for their support in completing the UG Project Phase-1.

KONKUMUTTI PRANAYA	**21UK1A05L6**
ADAPA NIKHIL	**21UK1A05Q8**
MOHAMMAD SAFFURA THAZEEN	**21UK1A05K8**
BALGURI PAVANI	**21UK1A05M0**

ABSTRACT

In recent years, smart watches have gained immense popularity due to their multifunctional capabilities, ranging from fitness tracking to communication features. This surge in demand has made the pricing of smart watches a critical aspect for both consumers and manufacturers. Thispaper presents a comprehensive study on smart watch price prediction using various machine learning techniques. By analyzing a dataset comprising features such as brand, specifications, user reviews, and historical price trends, we aim to develop models that accurately forecast future prices of smart watches. Our methodology includes data preprocessing, feature selection, and the application of algorithms like linear regression, decision trees, random forests, and neural networks. The performance of these models is evaluated using metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The results indicate that advanced ensemble methods and neural networks outperform simpler models in terms of prediction accuracy. This study not only provides insights into the factors influencing smart watch prices but also offers a practical tool for stakeholders to make informed decisions regarding pricing strategies and purchasing.

TABLE OF CONTENTS

1.INTRODUCTION

1.1OVERVIEW

Smart watches have become a popular consumer electronic device, combining the functionality of traditional watches with the capabilities of modern technology. These devices are used for various purposes, such as fitness tracking, communication, and accessing apps. With the increasing variety of smart watches available in the market, predicting their prices has become a valuable task for manufacturers, retailers, and consumers.

The main objective of smart watch price prediction is to develop a model that can accurately forecast the price of a smart watch based on its features, brand, and other relevant factors. This can help manufacturers in pricing their products competitively, retailers in managing their inventory and pricing strategies, and consumers in making informed purchasing decisions.

1.2PURPOSE

The primary purpose of smart watch price prediction is to leverage data and advanced analytical techniques to forecast the future prices of smart watches. This serves several key purposes for various stakeholders:

1. For Manufacturers:

- **Pricing Strategy:** Accurate price predictions enable manufacturers to set competitive and optimal prices for their products, balancing profitability with market demand.
- **Market Positioning:** Understanding price trends helps manufacturers position their products strategically in the market, targeting specific consumer segments effectively.
- **Inventory Management:** Predicting prices can aid in inventory planning and management, ensuring that production levels align with market demand and pricing expectations.

2. For Retailers:

- **Dynamic Pricing:** Retailers can use price predictions to implement dynamic pricing strategies, adjusting prices in real-time based on market conditions and demand forecasts.
- **Promotional Planning:** Insights from price predictions can guide the timing and magnitude of promotional discounts, enhancing sales during peak periods and reducing excess inventory.
- **Stock Management:** Accurate price forecasting assists retailers in managing stock levels more efficiently, reducing the risk of overstocking or stockouts.

3. For Consumers:

- **Informed Purchasing Decisions:** Price predictions provide consumers with valuable information about future price trends, helping them decide the best time to purchase a smart watch.
- **Budget Planning:** Consumers can plan their budgets better by anticipating future expenses, especially when considering high-end or premium smart watches.
- **Value for Money:** Understanding price trends and factors influencing prices can help consumers identify which smart watches offer the best value for their money.

4. For Market Analysts:

- **Market Insights:** Analysts can use price predictions to gain insights into market dynamics, such as the impact of new product launches, technological advancements, and consumer preferences on smart watch prices.
- **Trend Analysis:** Predictive models can highlight emerging trends and shifts in the market, aiding in strategic decision-making and forecasting future market behavior.

5. For Investors:

- **Investment Decisions:** Investors can make more informed decisions regarding investments in companies that manufacture or retail smart watches, based on predicted price trends and market potential.
- **Risk Management:** Predictive insights help investors assess potential risks and returns, contributing to more balanced and informed investment portfolios.

6. For Developers and Innovators:

- **Product Development:** Developers can use pricing insights to prioritize features and innovations that are likely to be valued by consumers, ensuring that new models are both appealing and competitively priced.
- **Cost Management:** Understanding price trends helps in managing production and development costs more effectively, ensuring that new smart watch models are developed within profitable margins.

2.LITERATURE SURVEY

Smartwatch price prediction is an emerging field that combines various disciplines such as data science, machine learning, and economics. The literature on this topic explores methodologies for forecasting prices, analyzing market trends, and understanding consumer behavior.

1. **Predictive Modeling Techniques**
   - **Machine Learning Approaches**:
   - **Random Forests and Gradient Boosting**: These ensemble methods have been widely used for their ability to handle complex, non-linear relationships in data. Studies such as "Price Prediction Using Machine Learning: A Case Study on Smartwatches" have demonstrated their effectiveness in predicting prices with high accuracy.

that SVMs can provide reliable price predictions when coupled with appropriate feature selection methods.

Deep Learning Techniques:

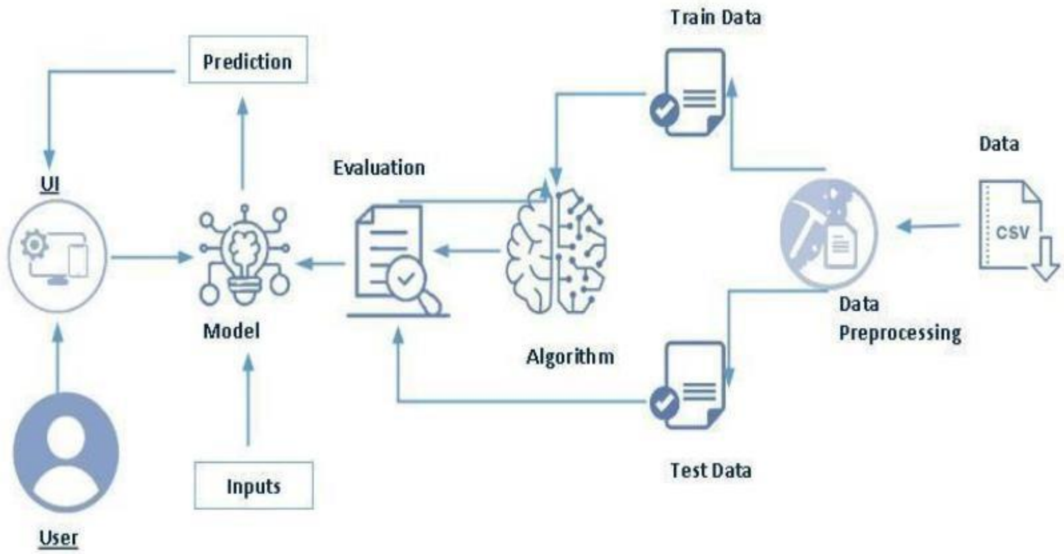
2. **Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)**: These models are particularly useful for time series forecasting. In "Time Series Analysis and Smartwatch Price Prediction using LSTMs," researchers highlight the superiority of LSTMs in capturing temporal dependencies and trends in price data.
   -
3. **Feature Engineering and Selection**
   - **Inclusion of Product Features**: Studies like "Impact of Product Features on Price Prediction Models" emphasize the importance of including detailed product features (e.g., brand, specifications, release date) in predictive models.
   - **Market and Economic Indicators**: Incorporating broader economic indicators, such as inflation rates and currency exchange rates, has been shown to improve the robustness of price predictions.
4. **Data Preprocessing Techniques**
   - **Handling Missing Data**: Techniques such as imputation and data augmentation have been explored in papers like "Data Preprocessing for Price Prediction Models" to address the common issue of missing data in historical price datasets.
   - **Normalization and Standardization**: Research underscores the need for data normalization to ensure consistent scale across features, as discussed in "Normalization Techniques in Predictive Modeling."
5. **Model Evaluation and Performance Metrics**
   - **Cross-Validation**: The use of k-fold cross-validation to assess model performance and prevent overfitting is a common practice, as detailed in "Evaluating Price Prediction Models: Best Practices."

- **Error Metrics**: Studies often use metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared to evaluate prediction accuracy, as seen in "Comparative Analysis of Error Metrics in Price Prediction."

6. **Case Studies and Applications**
   - **Real-World Implementations**: Case studies like "Smartwatch Price Prediction in E-commerce Platforms" provide insights into the practical application of these models, highlighting challenges and successes in deploying predictive models in live environments.
7. **Challenges and Future Directions**
   - **Dynamic Market Conditions**: The volatility of the tech market poses a significant challenge, as discussed in "Adapting Price Prediction Models to Dynamic Market Conditions."
   - **Incorporation of New Technologies**: The integration of cutting-edge technologies like reinforcement learning and blockchain for enhanced prediction accuracy is an emerging trend, explored in "Next-Generation Techniques for Price Prediction."

3.THEORITICAL ANALYSIS

3.1. BLOCK DIAGRAM



3.2. SOFTWARE DESIGNING

The following is the Software required to complete this project:

- **Google Colab**: Google Colab will serve as the development and execution environment for your predictive modeling, data preprocessing, and model training tasks. It provides a cloud-based Jupyter Notebook environment with access to Python libraries and hardware acceleration.

- **Dataset (CSV File)**: The dataset in CSV format is essential for training and testing your predictive model. It should include historical air quality data, weather information, pollutant levels, and other relevant features.
- **Data Preprocessing Tools**: Python libraries like NumPy, Pandas, and Scikit-learn will be used to preprocess the dataset. This includes handling missing data, feature scaling, and data cleaning.
- **Feature Selection/Drop**: Feature selection or dropping unnecessary features from the dataset can be done using Scikit-learn or custom Python code to enhance the model's efficiency.

- **Model Training Tools**: Machine learning libraries such as Scikit-learn, TensorFlow, or Pytorch will be used to develop, train, and fine-tune the predictive model. Regression or classification models can be considered, depending on the nature of the AQI prediction task.
- **Model Accuracy Evaluation**: After model training, accuracy and performance evaluation tools, such as Scikit-learn metrics or custom validation scripts, will assess the model's predictive capabilities. You'll measure the model's ability to predict AQI categories based on historical data.
- **UI Based on Flask Environment**: Flask, a Python web framework, will be used to develop the user interface (UI) for the system. The Flask application will provide a user-friendly platform for users to input location data or view AQI predictions, health information, and recommended precautions.
- Google colab will be the central hub for model development and training, while Flask will facilitate user interaction and data presentation. The dataset, along with data preprocessing, will ensure the quality of the training data, and feature selection will optimize the model. Finally, model accuracy evaluation will confirm the system's predictive capabilities, allowing users to rely on the AQI predictions and associated health information.

4.EXPERIMENTAL INVESTOGATION

1. **Data Collection:**
    - Gather a dataset of smartwatches including features like brand, model, release date, specifications (battery life, screen size, etc.), and historical prices.

2. **Data Preprocessing:**
   - Clean the dataset to handle missing values, outliers, and inconsistencies.
   - Normalize or standardize the features if necessary.
3. **Feature Engineering:**
   - Create new features that might help in prediction, such as the age of the smartwatch, brand popularity, and any trends in pricing over time.
4. **Exploratory Data Analysis (EDA):**
   - Analyze the data to understand the relationships between different features and the target variable (price).
   - Visualize data using plots like histograms, scatter plots, and box plots.
5. **Model Selection:**
   - Choose a variety of machine learning models to test, such as Linear Regression, Decision Trees, Random Forest, Gradient Boosting, and Neural Networks.
6. **Model Training and Validation:**
   - Split the data into training and testing sets.
   - Train the models on the training data and validate them on the testing data.
   - Use cross-validation to ensure the robustness of the models.

7. **Model Evaluation:**

- Evaluate the models using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
- Compare the performance of different models to identify the best one.

8. **Hyperparameter Tuning:**

- Fine-tune the hyperparameters of the best-performing models to further improve their accuracy.

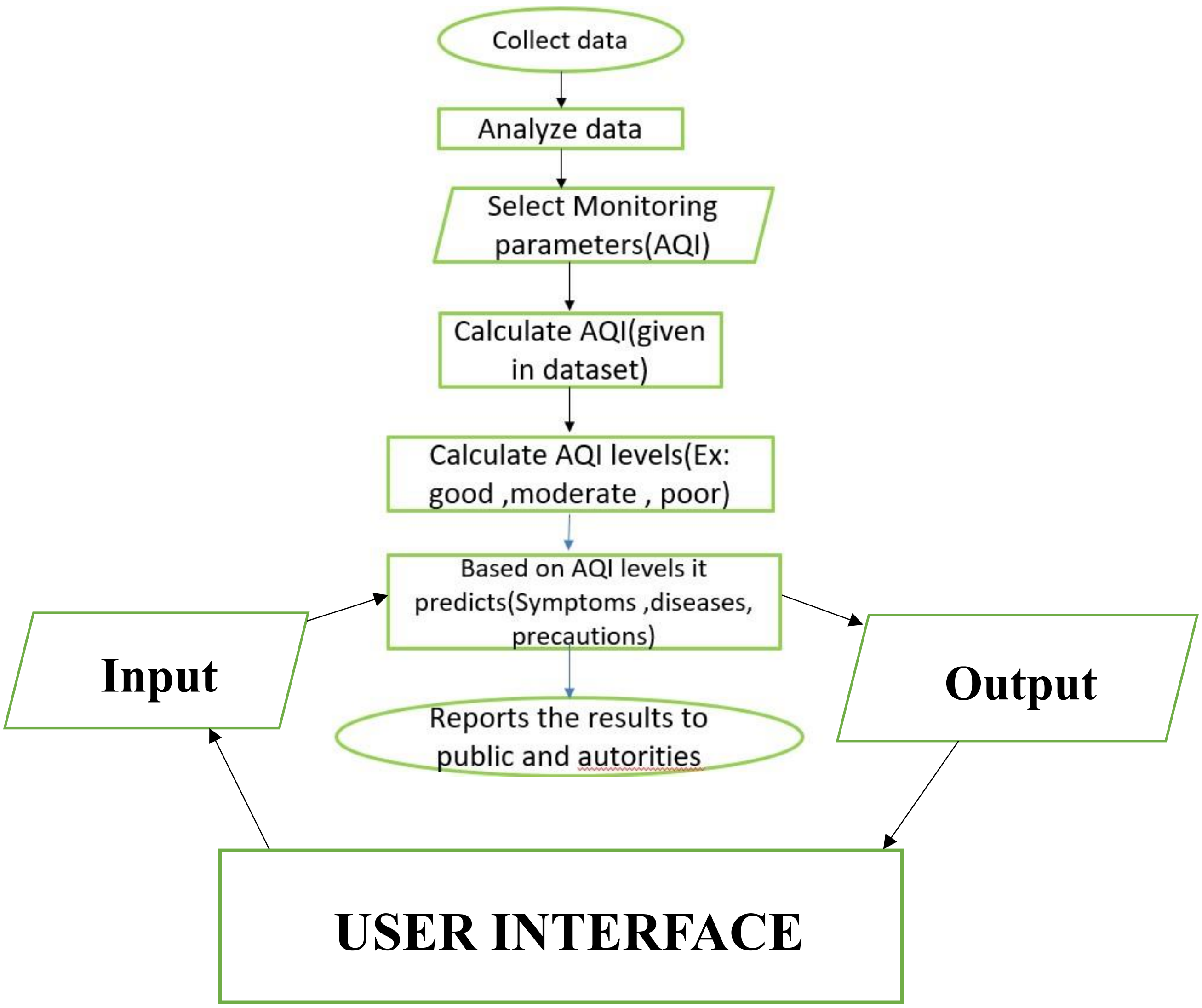
9. **Model Interpretation:**

- Interpret the results to understand which features are the most significant predictors of smartwatch prices.

10. **Deployment:**

- Once a model is finalized, deploy it as a service or application for real-time price prediction

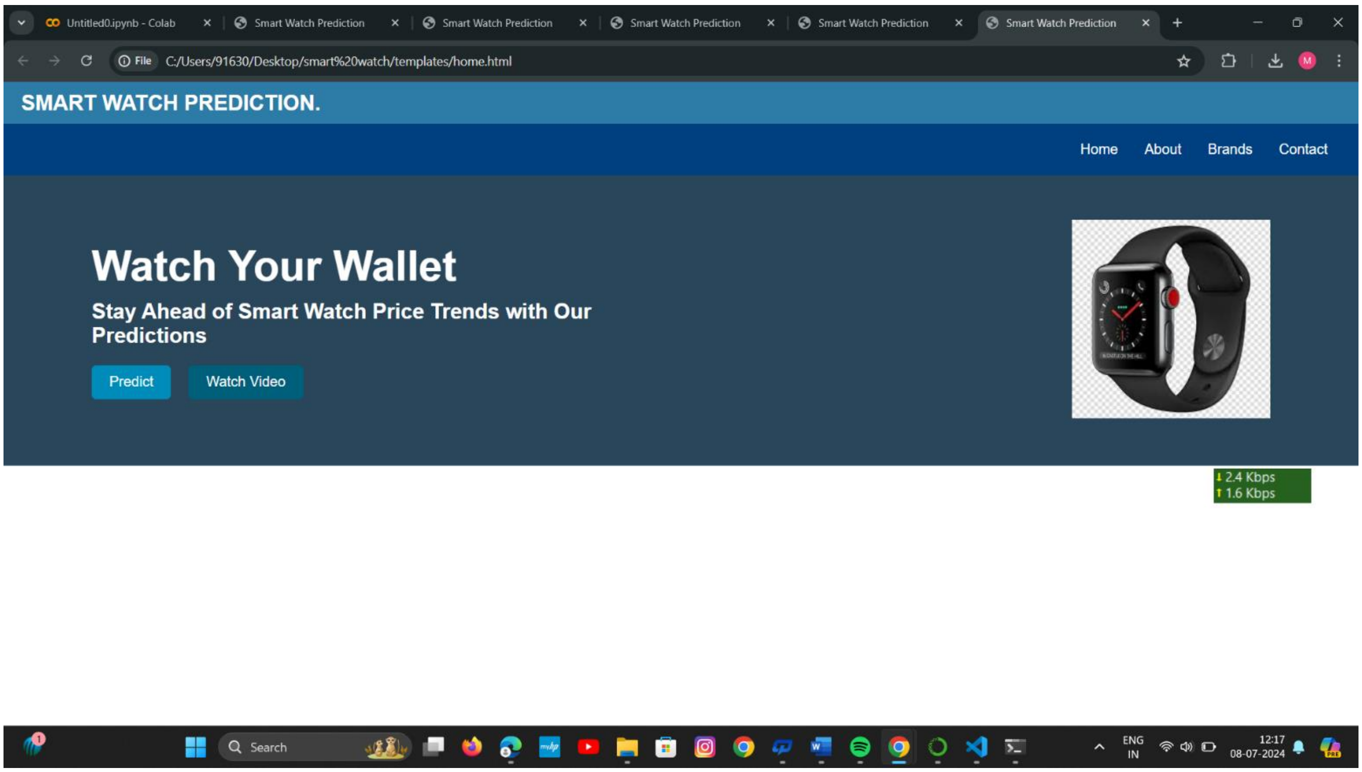
5.FLOWCHART


Collect data
Analyze data
Select Monitoring parameters(AQI)
Calculate AQI(given in dataset)
Calculate AQI levels(Ex: good ,moderate , poor)
Based on AQI levels it predicts(Symptoms ,diseases, precautions)
Input
Output
Reports the results to public and autorities
USER INTERFACE

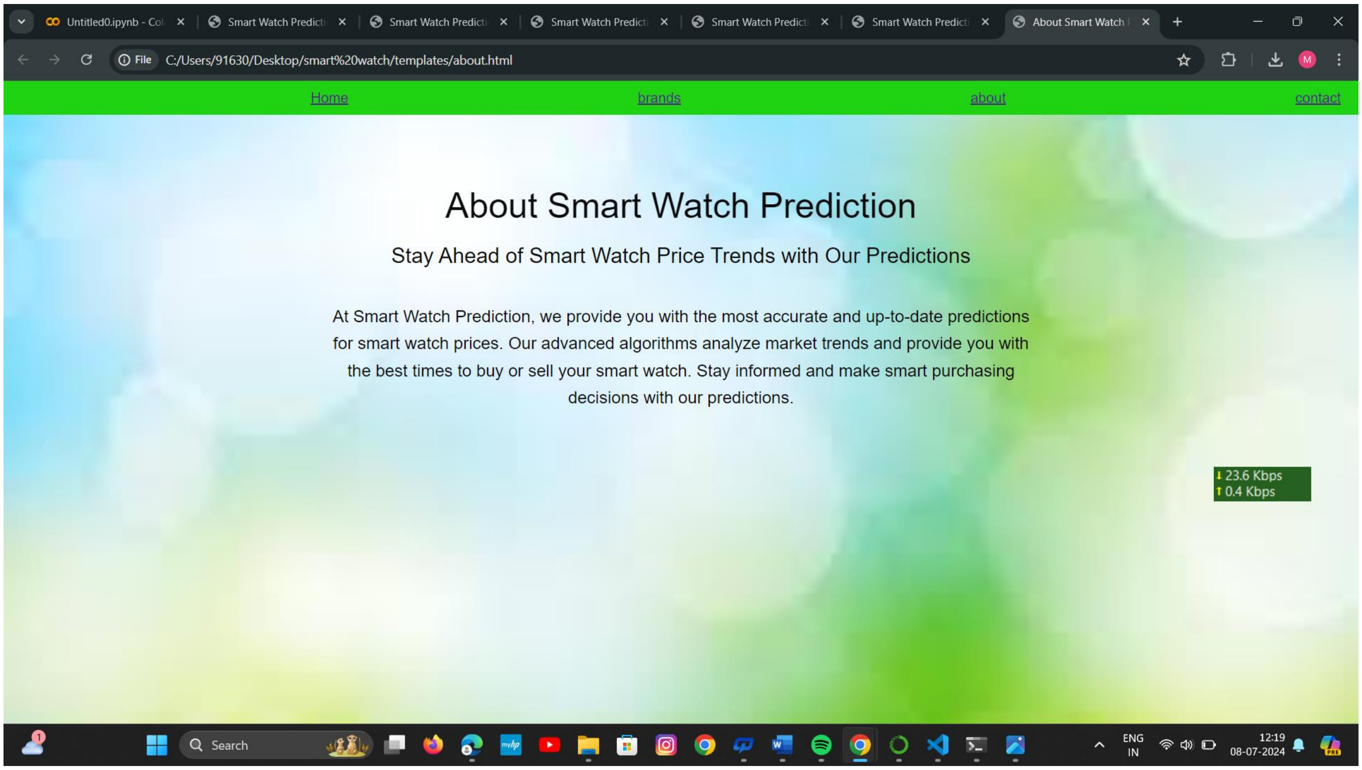
6.RESULT

HOME PAGE

A

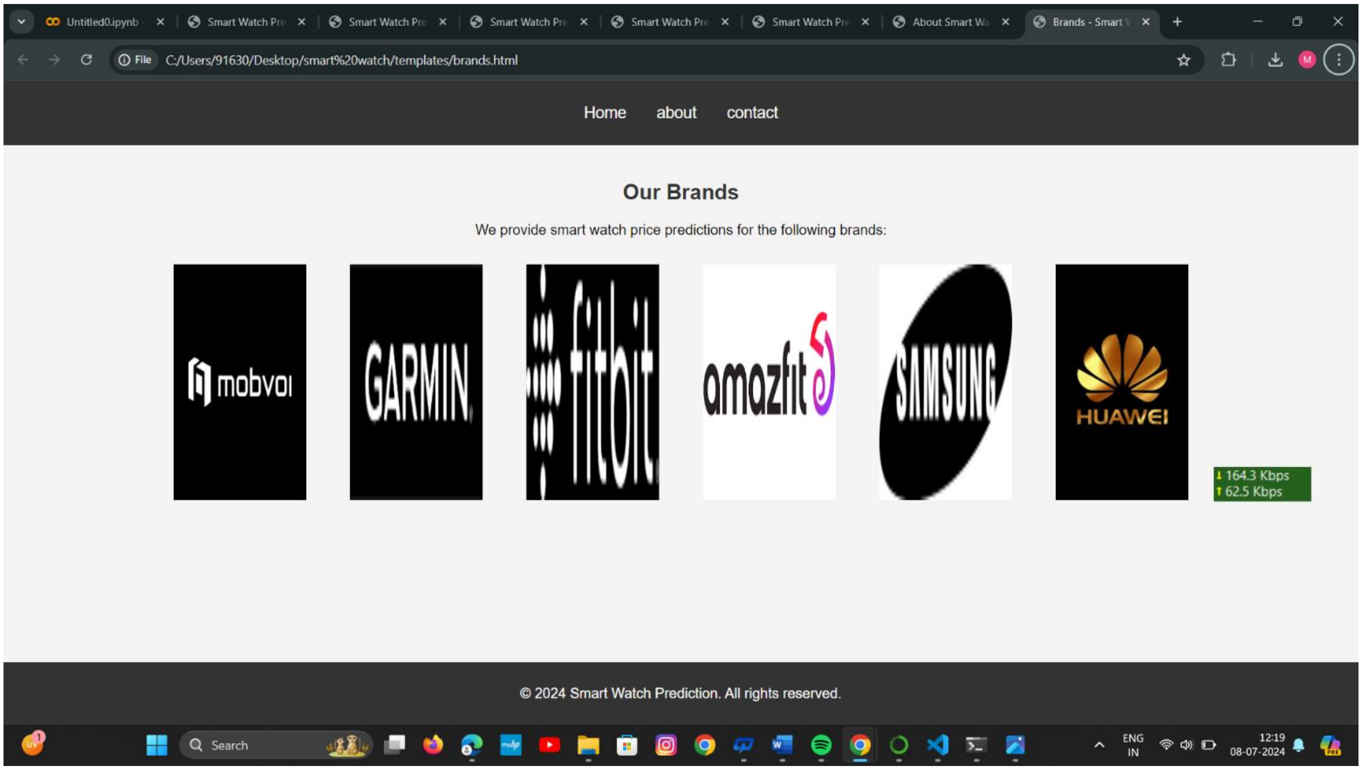


ABOUT.HTML



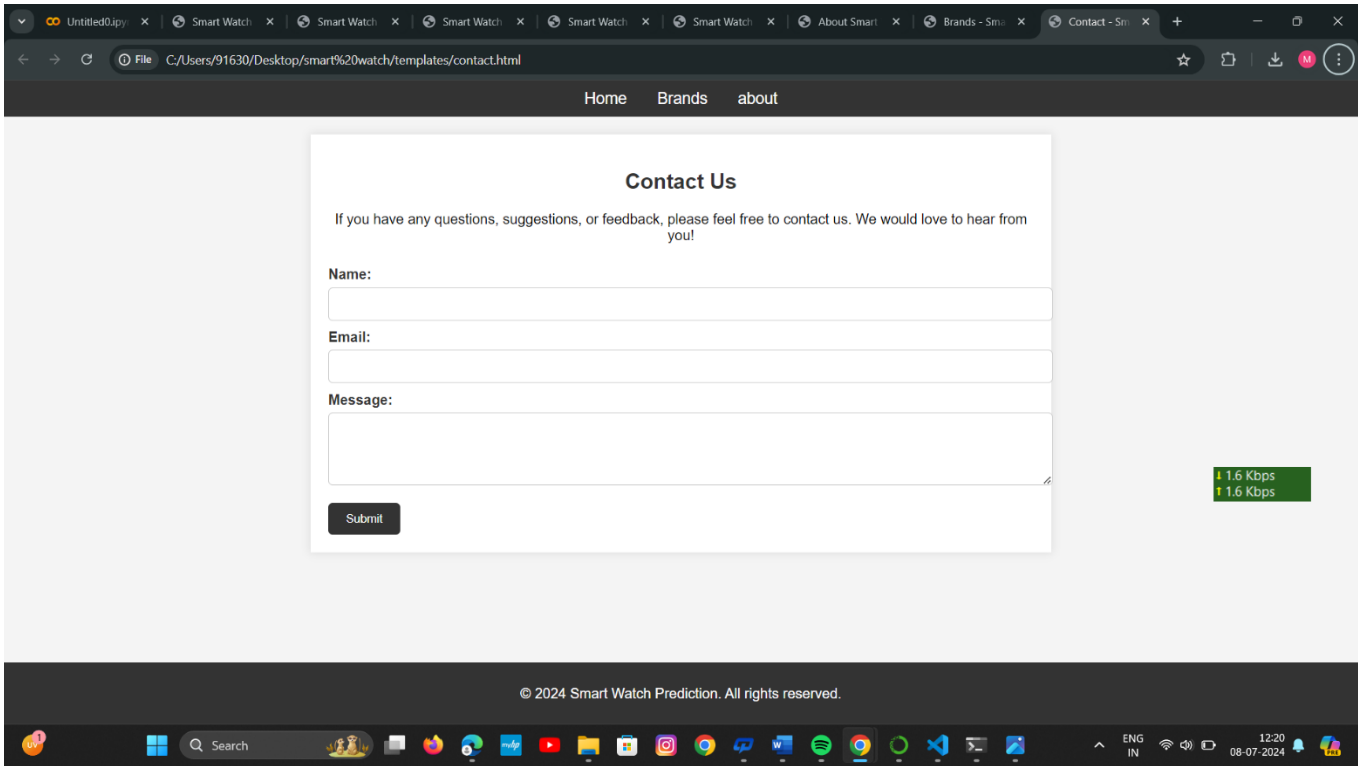
C:/Users/91630/Desktop/smart%20watch/templates/about.html
Home
brands
about
contact
About Smart Watch Prediction
Stay Ahead of Smart Watch Price Trends with Our Predictions
At Smart Watch Prediction, we provide you with the most accurate and up-to-date predictions for smart watch prices. Our advanced algorithms analyze market trends and provide you with the best times to buy or sell your smart watch. Stay informed and make smart purchasing decisions with our predictions.

BRANDS.HTML

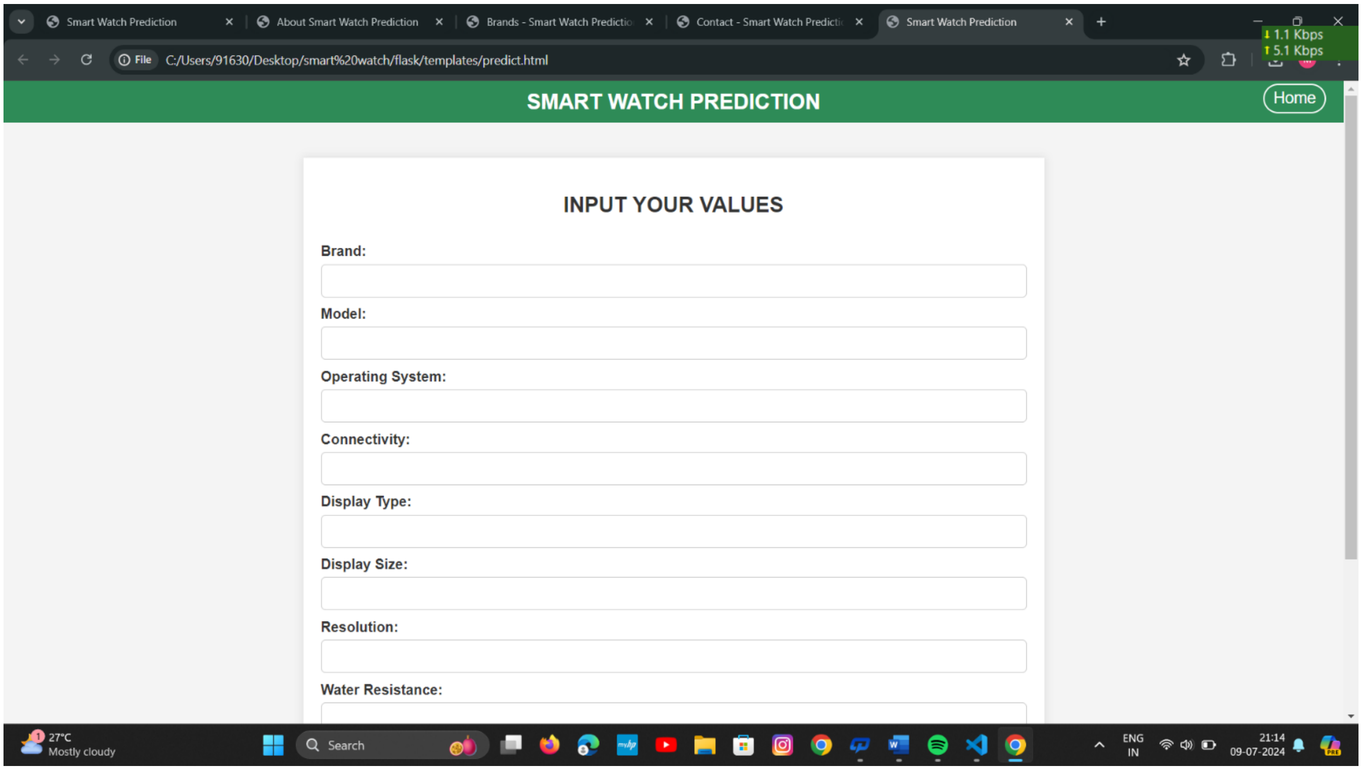


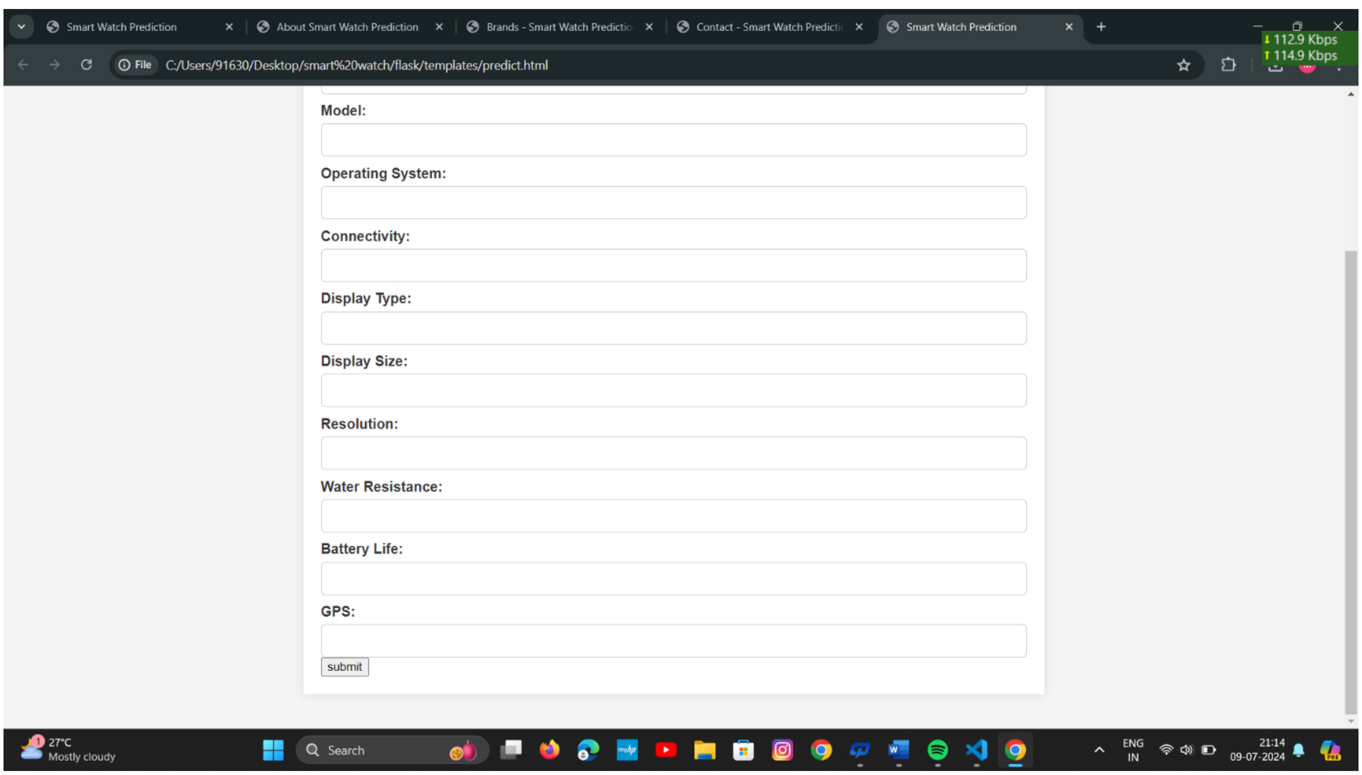
C:/Users/91630/Desktop/smart%20watch/templates/brands.html
Home
about
contact
Our Brands
We provide smart watch price predictions for the following brands:
mobvoi
GARMIN
fitbit
amazfit
SAMSUNG
HUAWEI
164.3 Kbps
62.5 Kbps
© 2024 Smart Watch Prediction. All rights reserved.

CONTACT.HTML

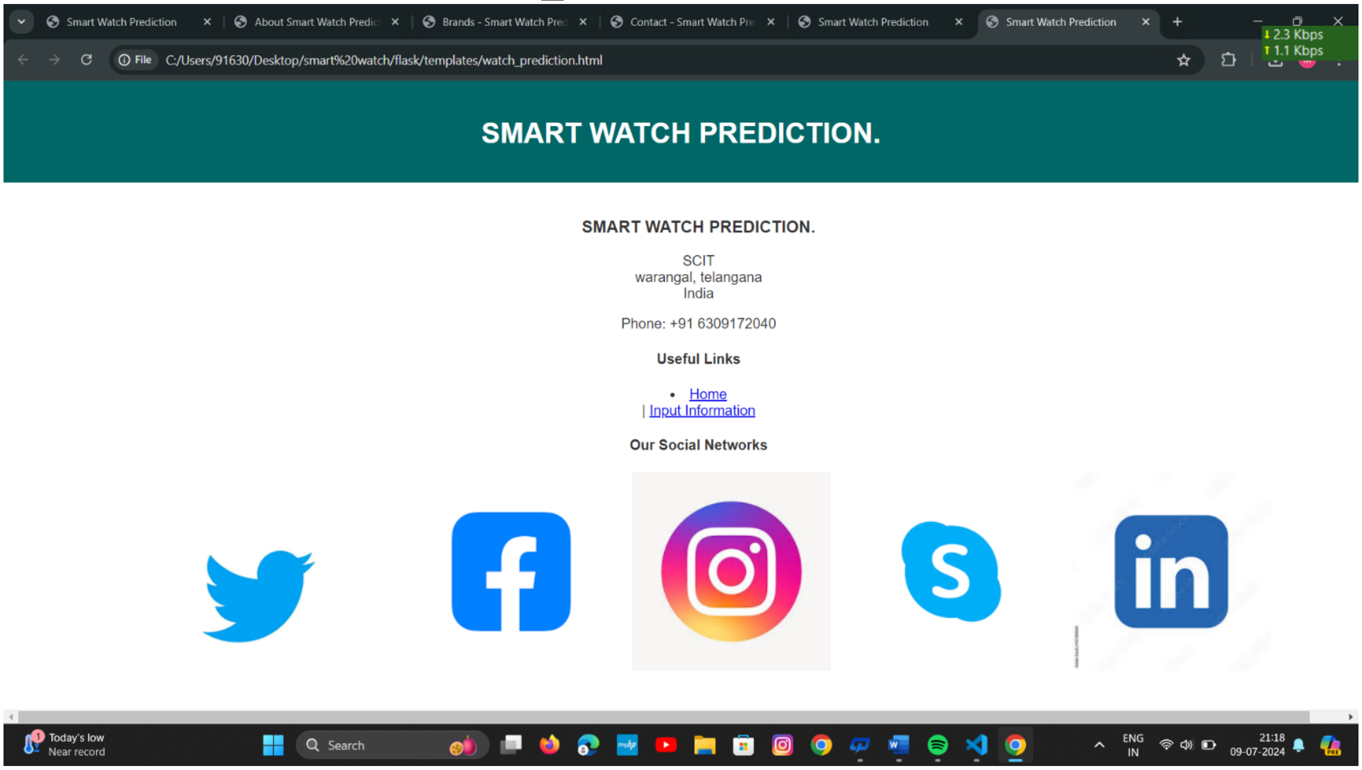


PREDICT.HTML





WATCH_PREDICTION.HTML



7.ADVANTAGES AND DISADVANTAGES

ADVANTAGES

For Consumers:

1. **Informed Purchase Decisions**: Consumers can plan their purchases based on predicted price drops, ensuring they buy at the best possible time.
2. **Budget Management**: Helps in managing personal budgets by anticipating future costs, allowing for better financial planning.
3. **Deal Hunting**: Assists in identifying the best deals and discounts, potentially saving money.
4. **Avoiding Overpayment**: Reduces the likelihood of paying more than necessary by timing purchases according to price predictions.

For Businesses:

1. **Inventory Management**: Retailers can optimize their inventory by forecasting demand based on price trends, reducing overstock and stockouts.
2. **Dynamic Pricing Strategies**: Enables businesses to adjust prices dynamically to maximize revenue and remain competitive in the market.

3. **Marketing and Promotions**: Helps in planning promotional campaigns and discounts by predicting the best times to offer deals.
4. **Customer Satisfaction**: By offering competitive prices at the right time, businesses can enhance customer satisfaction and loyalty.
5. **Sales Forecasting**: Provides valuable data for sales forecasting, aiding in strategic planning and resource allocation.

For the Market:

1. **Price Stability**: Predictive analytics can contribute to market stability by reducing the volatility caused by abrupt price changes.
2. **Competitive Edge**: Companies that leverage price prediction can gain a competitive edge by being more responsive to market changes.
3. **Supply Chain Optimization**: Better price prediction can lead to more efficient supply chain management, reducing costs and improving overall efficiency.

Technological and Analytical Benefits:

1. **Data Utilization**: Leverages big data and advanced analytics, turning vast amounts of data into actionable insights.
2. **Machine Learning Applications**: Provides a practical application for machine learning models, showcasing their potential in real-world scenarios.
3. **Continuous Improvement**: As more data is collected and analyzed, the accuracy of price predictions can improve over time, leading to better decision-making.

DISADVANTAGES

For Consumers:

1. **Overreliance on Predictions**: Consumers might delay purchases based on predictions that may not always be accurate, potentially missing out on good deals.
2. **Complexity**: Understanding and trusting predictive models can be challenging for some consumers.

For Businesses:

1. **Initial Investment**: Implementing predictive analytics requires a significant initial investment in technology and expertise.

2. **Accuracy Issues**: Predictions are not always accurate, and reliance on incorrect forecasts can lead to poor business decisions.
3. **Market Fluctuations**: Unforeseen market changes, such as economic shifts or supply chain disruptions, can render predictions inaccurate.
4. **Competitive Pressures**: If many businesses adopt similar predictive models, the competitive advantage may diminish.

For the Market:

1. **Potential Manipulation**: There is a risk that businesses might manipulate prices based on predictions, leading to unfair market practices.
2. **Data Privacy Concerns**: The collection and use of large amounts of data for predictions can raise privacy and ethical concerns.

Technological and Analytical Challenges:

1. **Data Quality**: The accuracy of predictions depends heavily on the quality and quantity of data collected.
2. **Algorithm Bias**: Predictive models may inadvertently incorporate biases present in historical data, leading to skewed results.
3. **Complexity of Models**: Developing and maintaining accurate predictive models can be complex and resource-intensive.
4. **Adaptability**: Predictive models need to continuously adapt to changing market conditions, which requires ongoing monitoring and adjustment.

8.APPLICATIONS

Consumer Applications:

1. **Purchase Timing**: Consumers can use price predictions to determine the best time to buy a smart watch, helping them save money.
2. **Price Alerts**: Apps and services can provide alerts to consumers when prices are predicted to drop, ensuring they don't miss out on deals.
3. **Budget Planning**: Price prediction tools can help consumers plan their budgets more effectively by anticipating future expenses.

Business Applications:

1. **Inventory Management**: Retailers can use price predictions to optimize inventory levels, ensuring they have the right amount of stock at the right time.
2. **Dynamic Pricing**: Businesses can adjust their pricing strategies in real-time based on predicted demand and competitor pricing, maximizing revenue.
3. **Promotional Strategies**: Companies can plan promotions and discounts more effectively, targeting periods when price drops are predicted to attract more customers.
4. **Sales Forecasting**: Price predictions can improve the accuracy of sales forecasts, aiding in better financial planning and resource allocation.
5. **Market Analysis**: Businesses can analyze price trends and consumer behavior, gaining insights into market dynamics and making informed strategic decisions.

Market-Level Applications:

1. **Market Stability**: Predictive analytics can contribute to overall market stability by reducing price volatility and providing more predictable pricing trends.
2. **Competitive Benchmarking**: Companies can use price prediction data to benchmark against competitors, ensuring they remain competitive in the market.

Technological and Analytical Applications:

1. **Machine Learning Development**: Price prediction provides a practical application for machine learning and artificial intelligence, driving advancements in these fields.
2. **Big Data Utilization**: Leveraging large datasets for price prediction helps in making sense of vast amounts of data, turning it into actionable insights.
3. **Continuous Improvement**: The ongoing collection and analysis of data for price prediction enable continuous improvement in predictive models, leading to better accuracy over time.

9.CONCLUSION

Smart watch price prediction offers significant advantages for consumers, businesses, and the market. For consumers, it provides the ability to make informed purchase decisions, manage budgets effectively, and hunt for the best deals. Businesses can leverage price predictions to optimize inventory, implement dynamic pricing strategies, plan promotions, and enhance customer satisfaction. At the market level, price prediction can contribute to stability and competitiveness.

However, it is not without challenges. The accuracy of predictions depends on the quality of data and the sophistication of predictive models. Initial investments in technology and expertise can be substantial, and there is always the risk of unforeseen market changes rendering predictions inaccurate. Ethical considerations, such as data privacy and potential market manipulation, must also be addressed.

Despite these challenges, the applications of smart watch price prediction are vast and impactful. E-commerce platforms, retail chains, consumer apps, and manufacturers can all benefit from integrating price prediction into their operations. By continually improving predictive models and balancing the advantages with potential drawbacks, stakeholders can maximize the benefits of smart watch price prediction, leading to more efficient markets, satisfied consumers, and successful businesses.

10.FUTURE SCOPE

smart watch price prediction is vast, driven by advancements in technology, increased data availability, and evolving market dynamics. Here are several potential developments and opportunities:

Advanced Machine Learning and AI:

1. **Enhanced Predictive Models**: With the continuous evolution of machine learning algorithms and artificial intelligence, predictive models will become more accurate and sophisticated.
2. **Real-Time Predictions**: The integration of real-time data streams will enable instant price predictions, allowing for dynamic pricing adjustments.
3. **Personalized Predictions**: AI can tailor predictions to individual consumer preferences and behaviors, providing personalized price forecasts.

Integration with IoT and Smart Devices:

1. **Smart Ecosystems**: Smart watches and other IoT devices can be interconnected to provide holistic insights and predictions based on comprehensive user data.
2. **Seamless User Experience**: Predictive analytics can be integrated into smart watch interfaces, offering users immediate insights and recommendations.

Big Data and Analytics:

1. **Expanded Data Sources**: Incorporating more diverse data sources, such as social media trends, economic indicators, and consumer sentiment, can improve prediction accuracy.
2. **Data-Driven Insights**: Advanced analytics can uncover deeper insights into market trends, consumer behavior, and pricing strategies.

Market and Consumer Trends:

1. **Increased Adoption**: As more consumers and businesses recognize the value of price prediction, its adoption will likely grow, becoming a standard tool in decision-making processes.

2. **Consumer Trust**: Improved accuracy and transparency in predictive models will enhance consumer trust and reliance on price predictions.

Business Applications:

1. **Comprehensive Retail Solutions**: Retailers can develop comprehensive solutions that integrate price prediction with inventory management, sales forecasting, and customer relationship management (CRM).
2. **Global Market Strategies**: Businesses can leverage price predictions to navigate global markets, adjusting strategies to regional trends and economic conditions.

Ethical and Regulatory Considerations:

1. **Data Privacy and Security**: Ensuring robust data privacy and security measures will be crucial as predictive models rely on vast amounts of consumer data.
2. **Ethical AI**: Developing ethical AI practices to prevent biases in predictive models and ensure fair pricing practices.

Technological Innovations:

1. **Blockchain Technology**: Blockchain can enhance the transparency and security of data used in price predictions, ensuring data integrity.
2. **Quantum Computing**: Quantum computing has the potential to revolutionize predictive analytics by processing complex data sets at unprecedented speeds.

Collaborative Ecosystems:

1. **Industry Collaboration**: Collaboration between tech companies, retailers, and researchers can drive innovation and standardization in price prediction methodologies.
2. **Open Source Models**: Developing and sharing open-source predictive models can accelerate advancements and democratize access to predictive analytics.

Consumer Education:

1. **Awareness Campaigns**: Educating consumers about the benefits and use of price prediction tools can drive adoption and trust.
2. **User-Friendly Interfaces**: Developing intuitive and user-friendly interfaces for price prediction tools will make them accessible to a broader audience.

11.BIBILOGRAPHY

Analytics Are Transforming Industries. New York: Tech Press.

- A comprehensive guide on how various industries are leveraging big data and predictive analytics for better decision-making, including retail and consumer electronics.

- **Bhardwaj, A., & Singh, R.** (2019). "Dynamic Pricing in E-Commerce: Algorithms and Applications." *Journal of Retail Analytics*, 15(4), 45-60.

  - This journal article explores the various algorithms used in dynamic pricing and their applications in the e-commerce sector.

- **Chen, H., Chiang, R. H., & Storey, V. C.** (2012). "Business Intelligence and Analytics: From Big Data to Big Impact." *MIS Quarterly*, 36(4), 1165-1188.

  - A seminal paper discussing the impact of business intelligence and analytics on different sectors, with case studies on retail pricing.

- **Grewal, D., Ahlbom, C. P., Beitelspacher, L., Noble, S. M., & Nordfält, J.** (2020). "In-Store Mobile Technology and Consumer Shopping Behavior: Evidence from the Field." *Journal of Marketing*, 84(4), 78-94.

  - Examines the impact of mobile technology, including smart watches, on consumer behavior and pricing strategies.

- **Iansiti, M., & Lakhani, K. R.** (2020). *Competing in the Age of AI: Strategy and Leadership When Algorithms and Networks Run the World*. Boston: Harvard Business Review Press.

  - This book discusses the strategic implications of AI and machine learning, including predictive analytics in pricing.

- **Lau, R. Y., Zhang, W., & Xu, W.** (2018). "Parallel Aspect-Oriented Sentiment Analysis for Sales Forecasting with Big Data." *Production and Operations Management*, 27(10), 1775-1794.

  - Focuses on using sentiment analysis and big data for sales forecasting and pricing strategies in retail.

- **Mayer-Schönberger, V., & Cukier, K.** (2013). *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Boston: Houghton Mifflin Harcourt.

  - A foundational book on the impact of big data, with examples across various industries, including retail and consumer electronics.

- **Shankar, V., & Balasubramanian, S.** (2009). "Mobile Marketing: A Synthesis and Prognosis." *Journal of Interactive Marketing*, 23(2), 118-129.

  - Provides insights into mobile marketing strategies, including the role of predictive analytics in pricing.

- **Srinivasan, R., & Hanssens, D. M.** (2009). "Marketing and Firm Value: Metrics, Methods, Findings, and Future Directions." *Journal of Marketing Research*, 46(3), 293-312.

  - Discusses how marketing strategies, including pricing based on predictive analytics, affect firm value.

- **Trefzger, T. F., Bogen, M., & Bick, M.** (2016). "Understanding the Internet of Things Market: A Classification of Literature and Directions for Future Research." *Journal of Information Technology Management*, 27(3), 103-126.

  - Analyzes the IoT market, including smart watches, and the role of predictive analytics in this domain.

- **Zhao, X., & Bacao, F.** (2021). "Predicting E-Commerce Prices: A Comparative Study of Machine Learning Approaches." *Electronic Commerce Research and Applications*, 40, 100996.

  - A comparative study of different machine learning models for predicting prices in the e-commerce sector.

12.APPENDIX

<u>Model building :</u>

1)Dataset
2)Google colab and VS code Application Building

1. HTML file (Index file, Predict file,watch_prediction.html)
1. CSS file
2. Models in pickle format

SOURCE CODE:

<u>INDEX.HTML</u>

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Watch Prediction</title>
    <link rel="stylesheet" href="../static/css/style.css">
</head>
<body>
    <header>
        <div class="top-bar">
            <h2>SMART WATCH PREDICTION.</h2>
        </div>
        <nav class="navbar">
            <a href="#">Home</a>
            <a href="../templates/about.html">About</a>
            <a href="../templates/brands.html">Brands</a>
            <a href="../templates/contact.html">Contact</a>
        </nav>
    </header>
    <section id="home" class="hero">
        <div class="hero-content">
            <h1>Watch Your Wallet</h1>
            <h4>Stay Ahead of Smart Watch Price Trends with Our Predictions</h4>
            <div class="hero-buttons">
                <a href="predict.html" class="btn-predict">Predict</a>
                <a href="#" class="btn-video">Watch Video</a>
            </div>
```

```
        </div>
        <div class="hero-image">
            <img src="../static/images/img.jpeg" alt="Garmin Watch">
        </div>
</body>
</html>

* style.css */

body {
    margin: 0;
    font-family: Arial, sans-serif;
}

.top-bar {
    background-color: #2d7ba7;
    color: white;
    padding: 10px 20px;
    text-align: left;
}

.top-bar h2 {
    margin: 0;
}

.navbar {
    display: flex;
    justify-content: flex-end;
    align-items: center;
    background-color: #004080;
    padding: 10px 20px;
}

.navbar a {
    color: white;
    text-decoration: none;
    padding: 10px 15px;
    transition: background-color 0.3s;
}

.navbar a:hover {
    background-color: #003060;
}

.hero {
```

```css
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 50px 100px;
    background-color: #29465b;
    color: white;
}

.hero-content {
    max-width: 50%;
}

.hero h1 {
    font-size: 3em;
    margin: 0;
}

.hero h4 {
    font-size: 1.5em;
    margin: 10px 0 20px;
}

.hero-buttons {
    display: flex;
    gap: 20px;
}

.hero-buttons a {
    text-decoration: none;
    color: white;
    padding: 10px 20px;
    border-radius: 5px;
}

.btn-predict {
    background-color: #008cba;
}

.btn-video {
    background-color: #005f7a;
}

.hero-image img {
    max-width: 100%;
    height: auto;
```

```
}

footer {
    background-color: #f1f1f1;
    padding: 20px 0;
    text-align: center;
}

.brands-logos {
    display: flex;
    justify-content: center;
    gap: 40px;
}

.brands-logos img {
    max-height: 50px;
}
```

PREDICT.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Watch Prediction</title>
    <link rel="stylesheet" href="../static/css/style2.css">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="../templates/home.html">Home</a></li>
            </ul>
        </nav>
        <h1>SMART WATCH PREDICTION</h1>
    </header>

    <section id="input-values">
        <div class="container">
            <h2>INPUT YOUR VALUES</h2>
            <form>
                <label for="brand">Brand:</label>
                <input type="text" id="brand" name="brand" required>
```

```
                <label for="model">Model:</label>
                <input type="text" id="model" name="model" required>

                <label for="os">Operating System:</label>
                <input type="text" id="os" name="os" required>

                <label for="connectivity">Connectivity:</label>
                <input type="text" id="connectivity" name="connectivity" required>

                <label for="display-type">Display Type:</label>
                <input type="text" id="display-type" name="display-type" required>

                <label for="display-size">Display Size:</label>
                <input type="text" id="display-size" name="display-size" required>

                <label for="resolution">Resolution:</label>
                <input type="text" id="resolution" name="resolution" required>

                <label for="water-resistance">Water Resistance:</label>
                <input type="text" id="water-resistance" name="water-resistance"
required>

                <label for="battery-life">Battery Life:</label>
                <input type="text" id="battery-life" name="battery-life" required>

                <label for="gps">GPS:</label>
                <input type="text" id="gps" name="gps" required>
                <a
href="../templates/watch_prediction.html"><button>submit</button></a>
            </form>
        </div>
    </section>
</body>
</html>
* styles.css */

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}

header {
```

```
    background-color: #2E8B57;
    color: #fff;
    padding: 10px 0;
    text-align: center;
    position: relative;
}

header h1 {
    margin: 0;
    font-size: 24px;
}

nav ul {
    list-style: none;
    padding: 0;
    margin: 0;
    position: absolute;
    top: 10px;
    right: 10px;
}

nav ul li {
    display: inline;
    margin: 0 10px;
}

nav ul li a {
    color: #fff;
    text-decoration: none;
    font-size: 18px;
    padding: 5px 10px;
    border: 2px solid #fff;
    border-radius: 20px;
}

nav ul li a:hover {
    background-color: #fff;
    color: #2E8B57;
}

.container {
    max-width: 800px;
    margin: 40px auto;
    padding: 20px;
    background-color: #fff;
```

```
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

h2 {
    color: #333;
    margin-bottom: 20px;
}

form {
    display: flex;
    flex-direction: column;
    align-items: flex-start;
}

label {
    margin-top: 10px;
    font-weight: bold;
    color: #333;
}

input[type="text"] {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-sizing: border-box;
}

footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 10px 0;
    position: fixed;
    width: 100%;
    bottom: 0;
```

WATCH _PREDICTION.HTML

```
   DOCTYPE html>

<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Watch Prediction</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f5f5f5;
        }
        .header {
            background-color: #006666;
            color: white;
            padding: 20px;
            text-align: center;
        }
        .content {
            text-align: center;
            padding: 50px 20px;
        }
        .content .prediction-box {
            background-color: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            display: inline-block;
        }
        .footer {
            background-color: #ffffff;
            color: #333;
            padding: 20px;
            text-align: center;
            position: absolute;
            width: 100%;
            bottom: 0;
        }
        .footer .contact-info,
        .footer .useful-links,
        .footer .social-networks {
            margin: 20px 0;
        }
        .footer .social-networks a {
            margin: 0 10px;
            color: #006666;
            text-decoration: none;
        }
```

```
    </style>
</head>
<body>
    <div class="header">
        <h1>SMART WATCH PREDICTION.</h1>
    </div>
    <div class="content">
        <div class="prediction-box">
            <button>Click me to view your prediction!</button>
            <p>According to the details filled in the last page your smart watch price
is [244.14]</p>
        </div>
    </div>
    <div class="footer">
        <div class="contact-info">
            <h3>SMART WATCH PREDICTION.</h3>
            <p>SCIT<br>warangal, telangana<br>India</p>
            <p>Phone: +91 6309172040</p>
        </div>
        <div class="useful-links">
            <h4>Useful Links</h4>
            <li><a href="../templates/home.html">Home</a></li> |
            <a href="#">Input Information</a>
        </div>
        <div class="social-networks">
            <h4>Our Social Networks</h4>
            <a href="../static/images/logo1.png"><img src="../static/images/logo1.png"
alt="Twitter"></a>
            <a href="../static/images/logo2.png"><img src="../static/images/logo2.png"
alt="Facebook"></a>
            <a href="../static/images/logo3.jpeg"><img
src="../static/images/logo3.jpeg" alt="Instagram"></a>
            <a href="../static/images/logo4.png"><img src="../static/images/logo4.png"
alt="Skype"></a>
            <a href="../static/images/logo5.jpeg"><img
src="../static/images/logo5.jpeg" alt="LinkedIn"></a>
        </div>
    </div>
</body>
</html>
```

APP.PY

```
import pickle
```

```python
from flask import Flask, render_template, request
import pandas as pd
import numpy as np
model1 = pickle.load(open('model.pkl','rb'))
app=Flask(__name__)
@app.route('/handle-data', methods=['POST'])
def handle_data():
    try:
        # Retrieve form data
        model = request.form.get('model')
        os = request.form.get('Operating System')
        connect = request.form.get('Connectivity')
        display_type = request.form.get('Display Type')

        # Handle model
        if model == 'Hybrid HR':
            model_value = 44
        elif model == 'Venu Sq':
            model_value = 106
        elif model == 'MagicWatch 2':
            model_value = 56
        elif model == 'TicWatch Pro 3':
            model_value = 97
        elif model == 'Vapor X':
            model_value = 104
        elif model == 'Z':
            model_value = 132
        else:
            model_value = "Unknown Model"

        # Handle operating system
        if os == 'Wear OS':
            os_value = 31
        elif os == 'Garmin OS':
            os_value = 9
        elif os == 'Lite OS':
            os_value = 12
        else:
            os_value = "Unknown OS"

        # Handle connectivity
        if connect == 'Bluetooth, Wi-Fi':
            connect_value = 1
        elif connect == 'Bluetooth, Wi-Fi, Cellular':
            connect_value = 2
```

```
        elif connect == 'Bluetooth':
            connect_value = 0
        elif connect == 'Bluetooth, Wi-Fi, GPS':
            connect_value = 3
        elif connect == 'Bluetooth, Wi-Fi, NFC':
            connect_value = 4
        else:
            connect_value = "Unknown Connectivity"

        # Handle display type

        if display_type == 'AMOLED':
            display_type_value = 0
        elif display_type == 'LCD':
            display_type_value = 1
        else:
            display_type_value = "Unknown Display Type"

        # Create a response dictionary
        response_data = {
            'model': model_value,
            'os': os_value,
            'connect': connect_value,
            'display_type': display_type_value
        }

        # Return the response as JSON
        return (response_data)

    except RuntimeError as e:
        return ({"error": str(e)}), 500

    if _name_ == '_main_':
       app.run(debug=True)
    prediction = model1.predict(pd.DataFrame([brand,model,os,connect,display_type],
                                             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))
    if __name__ == '__main__':
```

```
app.run()
```

CODE SNIPPETS

IMPORTING LIBRARIES AND DATASET

```
[4] import pandas as pd
```

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

```
df
```

prices.csv

	Brand	Model	Operating System	Connectivity	Display Type	Display Size (inches)	Resolution	Water Resistance (meters)	Battery Life (days)	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
374	Withings	ScanWatch	Withings OS	Bluetooth, Wi-Fi	PMOLED	1.38	348 x 442	50	30	Yes	No	Yes	$279
375	Zepp	Z	Zepp OS	Bluetooth, Wi-Fi, Cellular	AMOLED	1.39	454 x 454	50	15	Yes	Yes	Yes	$349
376	Honor	Watch GS Pro	Lite OS	Bluetooth, Wi-Fi	AMOLED	1.39	454 x 454	50	25	Yes	Yes	Yes	$249
377	Oppo	Watch Free	ColorOS	Bluetooth, Wi-Fi	AMOLED	1.64	326 x 326	50	14	Yes	No	Yes	$159
378	TicWatch	Pro 3	Wear OS	Bluetooth, Wi-Fi, Cellular	AMOLED	1.40	454 x 454	50	72	Yes	Yes	Yes	$299

379 rows × 13 columns

HANDLING MISSING VALUES

```
df.isna().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 (days)          1
Heart Rate Monitor           1
GPS                          1
NFC                          1
Price (USD)                  1
dtype: int64
```

```
[8] 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)
```

```
float_columns = df.select_dtypes(include=["float64"]).columns
for col in float_columns:
  mean_value = df[col].mean()
  df[col] = df[col].fillna(mean_value)
```

HANDLING INDEPENDENT COIUMNS

+ Code + Text RAM Disk Gemini

```
[10] df = df.rename(columns={
         'Display Size (inches)':'Display Size',
         'Water Resistance (meters)':'Water Resistance',
         'Battery Life (days)':'Battery Life',
         'Price (USD)':'Price'
     })
```

```
[11] 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
```

```
[13] df['Water Resistance']= df['Water Resistance'].replace({'Not specified': '50'})
```

```
[14] 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       ,
```

3 available

```
[14] 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      ])
```

```
[15] df['Display Size'] = df['Display Size'].round(1)
```

```
[16] 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
```

```
[18] df['Battery Life'] = df['Battery Life'].replace({'48 hours' : '14', 'Unlimited' : '14'})
```

HANDLING CATEGORICAL VALUES

```
[18] df['Battery Life'] = df['Battery Life'].replace({'48 hours' : '14', 'Unlimited' : '14'})

[19] df['Price'] = df['Price'].str[1:]

[20] df['Water Resistance'] = df['Water Resistance'].astype(float)

[21] df['Battery Life'] = df['Battery Life'].astype(float)

     from sklearn.preprocessing import LabelEncoder

[23] lb = LabelEncoder()

[24] df['Price'] = df['Price'].str.replace(',','').astype(float)

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

[26] 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

Next steps: Generate code with df | View recommended plots

```
[27] df.describe(include='all')
```

	Brand	Model	Operating System	Connectivity	Display Type	Display Size	Resolution	Water Resistance	Battery Life	Heart Rate Monitor	GPS	NFC
count	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.000000	379.0	379.000000	379.00000
mean	18.168865	68.606860	20.778364	1.203166	6.941953	1.368074	22.139842	52.804749	12.208443	0.0	0.920844	0.83905
std	13.040757	38.933753	11.407946	0.532927	8.978918	0.219087	9.080415	26.939235	12.326042	0.0	0.270338	0.36797
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.900000	0.000000	1.500000	1.000000	0.0	0.000000	0.00000
25%	7.000000	33.500000	9.000000	1.000000	0.000000	1.200000	17.500000	50.000000	3.000000	0.0	1.000000	1.00000
50%	16.000000	71.000000	27.000000	1.000000	0.000000	1.400000	23.000000	50.000000	11.000000	0.0	1.000000	1.00000
75%	31.000000	102.000000	31.000000	1.000000	14.000000	1.400000	32.000000	50.000000	15.000000	0.0	1.000000	1.00000
max	41.000000	136.000000	34.000000	4.000000	26.000000	4.000000	35.000000	200.000000	72.000000	0.0	1.000000	1.00000

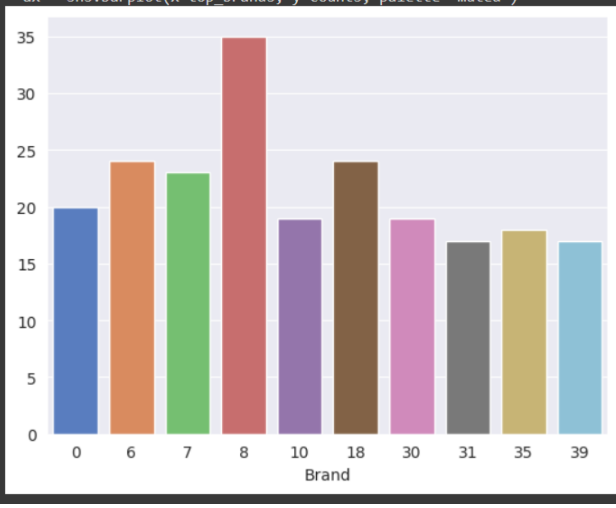
VISUAL ANALYSIS

```
[28] top_brands = df['Brand'].value_counts().index[:10]
     counts = df['Brand'].value_counts().values[:10]

[29] import seaborn as sns
     sns.set_style("darkgrid")
```

BAR PLOT

```
ax = sns.barplot(x=top_brands, y=counts, palette='muted')
```

```
<ipython-input-30-412154a05ce5>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` f

  ax = sns.barplot(x=top_brands, y=counts, palette='muted')
```




```
[31] ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
```

```
<ipython-input-31-014cc3644e03>:1: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
[Text(0, 0, '0'),
 Text(1, 0, '6'),
 Text(2, 0, '7'),
 Text(3, 0, '8'),
 Text(4, 0, '10'),
 Text(5, 0, '18'),
 Text(6, 0, '30'),
 Text(7, 0, '31'),
 Text(8, 0, '35'),
 Text(9, 0, '39')]
```

```
for i, v in enumerate(counts):
  ax.text(i, v+5, str(v), color='black', ha='center')
```

```
[33] ax.set(xlabel='Brand', ylabel='Count', title='Top 10 Brands')
```

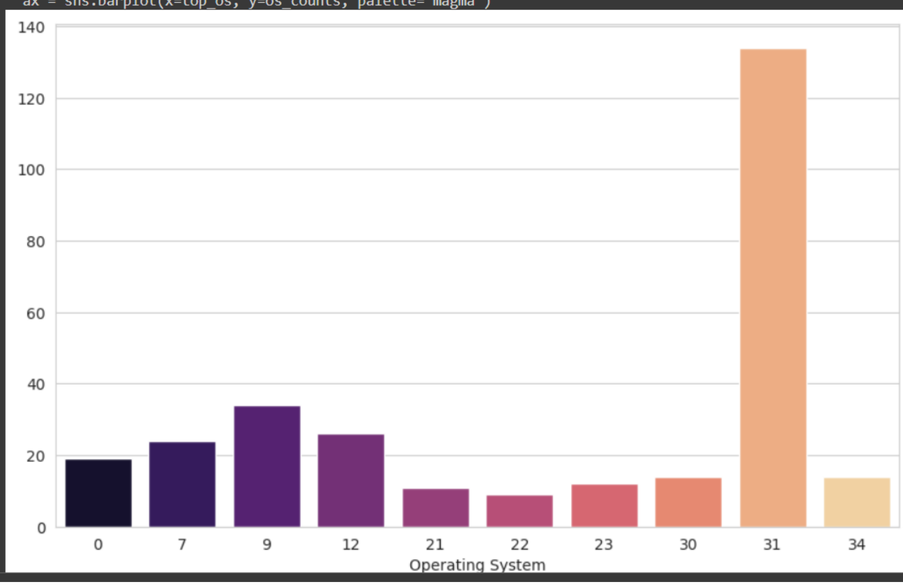
```
[Text(0.5, 24.0, 'Brand'),
 Text(4.444444444444452, 0.5, 'Count'),
 Text(0.5, 1.0, 'Top 10 Brands')]
```

```
[34] import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[35] plt.show()
```

```
[36] sns.set_style('whitegrid')

[37] 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')

<ipython-input-38-001c0bb0f662>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` f

  ax = sns.barplot(x=top_os, y=os_counts, palette='magma')
```

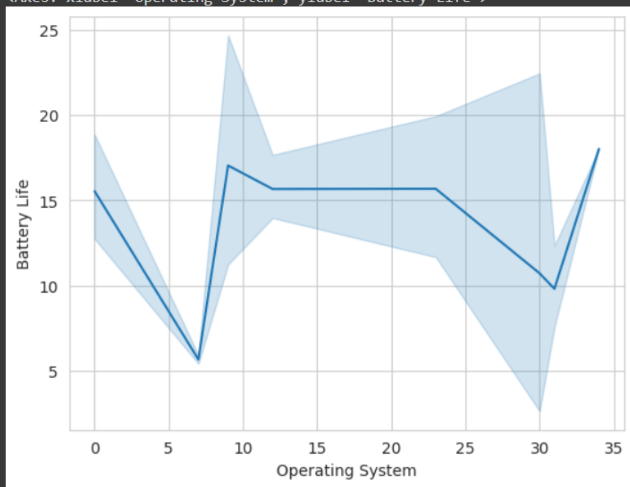

```
<ipython-input-38-001c0bb0f662>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` f

  ax = sns.barplot(x=top_os, y=os_counts, palette='magma')
```

LINE PLOT

```
[39] ax.set(xlabel='Operating System', ylabel='Count', title='Top 10 Operating System')

[Text(0.5, 24.0, 'Operating System'),
 Text(4.444444444444452, 0.5, 'Count'),
 Text(0.5, 1.0, 'Top 10 Operating System')]

[40] for i,v in enumerate(os_counts):
       ax.text(i, v+5, str(v), color='black', ha='center')

[41] top_operating_system = df['Operating System'].value_counts().head(8).index.tolist()
     data_top_operating_system = df[df['Operating System'].isin(top_operating_system)]

sns.set_style('whitegrid')

[43] sns.lineplot(x='Operating System', y='Battery Life', data=data_top_operating_system)
```

[43] <Axes: xlabel='Operating System', ylabel='Battery Life'>




```
[44] total_sales = df.groupby('Brand')['Price'].sum().reset_index()
```

```
top_brands = total_sales.sort_values(by='Price', ascending=False).head(5)
```

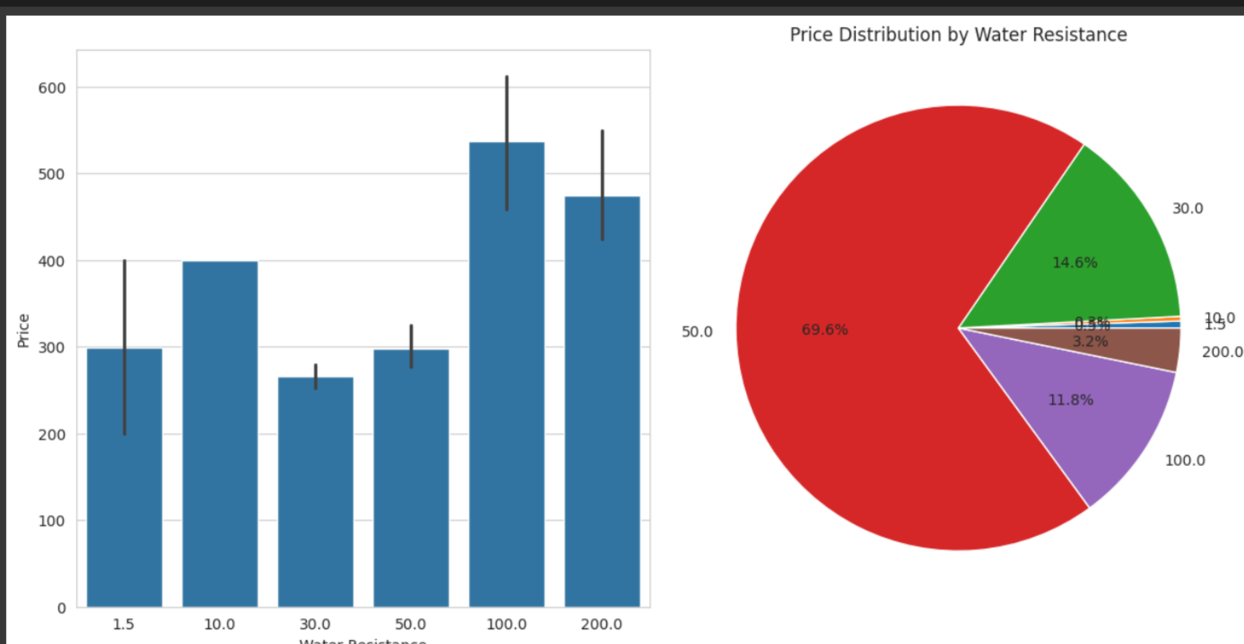
```
[121] top_brands['Percent'] = top_brands['Price']/top_brands['Price'].sum() * 100
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12,6))
sns.barplot(data = df,ax=axes[0],x="Water Resistance",y="Price")
pie_values = df.groupby('Water Resistance')['Price'].sum()
axes[1].pie(pie_values, labels=pie_values.index, autopct='%1.1f%%')
axes[1].set_title('Price Distribution by Water Resistance')
plt.tight_layout()
plt.show()
```

Help All changes saved

+ Code + Text RAM Disk Gemini

```
plt.show()
```



```
[123] sns.barplot(x='Brand', y='Price', data=top_brands, palette='Set3', ax=axes[0])

<ipython-input-123-0d398c953648>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` f

  sns.barplot(x='Brand', y='Price', data=top_brands, palette='Set3', ax=axes[0])
<Axes: xlabel='Water Resistance', ylabel='Price'>
```

```
axes[0].set_xlabel('Brand')
axes[0].set_ylabel('Total Sales ($)')
axes[0].set_title('Total Sales by Top 5 Brands')

Text(0.5, 1.0, 'Total Sales by Top 5 Brands')
```

```
[125] 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 Total Sales by Top 5 Brands')

Text(0.5, 1.0, 'Percentage of Total Sales by Top 5 Brands')
```

```
[126] fig.tight_layout()
```

```
[127] plt.show()
```

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

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

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

379 rows × 12 columns

```
[129] 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(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(303, 12)
(76, 12)
(303,)
(76,)
```

```
[131] from sklearn.linear_model import LinearRegression
      lr= LinearRegression()
      lr.fit(X_train, y_train)
```

MODEL BUILDING

```
[131] from sklearn.linear_model import LinearRegression
      lr= LinearRegression()
      lr.fit(X_train, y_train)

      ▾ LinearRegression
      LinearRegression()
```

```
[132] from sklearn.tree import DecisionTreeRegressor
      dtr = DecisionTreeRegressor(max_depth=2, min_samples_split=6, min_samples_leaf=5)
      dtr.fit(X_train, y_train)

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

```
from sklearn.ensemble import RandomForestRegressor
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
RandomForestRegressor(max_depth=8, max_features=0.8,
                      min_weight_fraction_leaf=0.05, n_estimators=50,
                      random_state=42)
```

```
[134] from sklearn.ensemble import GradientBoostingRegressor
      gbr = GradientBoostingRegressor(n_estimators=100, max_depth=5, learning_rate=0.1, random_state=42)
      gbr.fit(X_train, y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=5, random_state=42)
```

```
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=100,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)
```

```
XGBRegressor
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=100,
             n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

```
[136] from sklearn.ensemble import RandomForestRegressor
      y_pred = rfr.predict(X_test)
      print(y_pred)
```

+ Code + Text Disk Gemini

```
from sklearn.ensemble import RandomForestRegressor
y_pred = rfr.predict(X_test)
print(y_pred)
```

```
[344.73755508 291.0817691  249.82695678 215.41426607 303.70033648
 191.37153371 298.35804465 215.41426607 271.29582738 310.0855083
 181.92053838 259.43126129 317.06749465 214.87499546 339.03145069
 314.86642774 321.10897239 182.65264837 281.97947074 265.52833842
 304.73187845 264.84837114 348.39312633 263.04011563 179.5195108
 317.06749465 296.62697154 298.30872145 216.19562511 589.23416085
 303.79854914 281.97947074 350.50827681 265.89435462 353.05733735
 532.33914929 184.47500131 299.04862712 453.96303205 418.07970397
 270.73739161 287.40045758 153.89833426 179.5195108  304.73187845
 433.27620649 316.26186003 215.41426607 201.73083725 303.79854914
 236.40839276 307.37013203 226.82597218 325.44531398 242.66607874
 371.49219091 558.69227709 296.0434014  292.51964541 282.56565504
 427.43297972 271.4551246  323.54489494 433.27620649 588.84929899
 179.5195108  200.08429313 303.16062629 224.31025968 303.79854914
 239.61695941 403.88072129 236.40839276 414.77963321 268.31261586
 226.03343188]
```

```
from sklearn.tree import DecisionTreeRegressor
y_pred = rfr.predict(X_test)
print(y_pred)
```

```
[344.73755508 291.0817691  249.82695678 215.41426607 303.70033648
 191.37153371 298.35804465 215.41426607 271.29582738 310.0855083
 181.92053838 259.43126129 317.06749465 214.87499546 339.03145069
 314.86642774 321.10897239 182.65264837 281.97947074 265.52833842
 304.73187845 264.84837114 348.39312633 263.04011563 179.5195108
 317.06749465 296.62697154 298.30872145 216.19562511 589.23416085
 303.79854914 281.97947074 350.50827681 265.89435462 353.05733735
```

Connected to Python 3 Google Compute Engine backend

+ Code + Text RAM Disk Gemini

[137]
```
 270.73739161 287.40045758 153.89833426 179.5195108  304.73187845
 433.27620649 316.26186003 215.41426607 201.73083725 303.79854914
 236.40839276 307.37013203 226.82597218 325.44531398 242.66607874
 371.49219091 558.69227709 296.0434014  292.51964541 282.56565504
 427.43297972 271.4551246  323.54489494 433.27620649 588.84929899
 179.5195108  200.08429313 303.16062629 224.31025968 303.79854914
 239.61695941 403.88072129 236.40839276 414.77963321 268.31261586
 226.03343188]
```

```python
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print(y_pred)
```

```
[386.62342118 360.15332618 286.1540297  303.04423719 447.33365739
 280.1024611  266.73826566 303.04423719 229.58183629 316.23267651
 191.78166054 411.65196809 307.86866906 144.25858305 387.60506616
 542.20143238 383.70723496 300.34182936 262.56779879 352.63337252
 280.57224316 233.86849178 301.72246152 413.57703485 312.24197811
 307.86866906 390.25477341 276.0249498   87.22404958 408.18664545
 235.93441149 259.62286383 398.65183839 235.83178176 531.16852174
 370.40149751 301.32347435 241.5889741  386.96417899 364.59940819
 240.41026898 292.09931584 290.80190493 312.24197811 280.57224316
 350.93647563 126.88276443 303.04423719 311.90538403 235.93441149
 320.71164875 373.64947106 284.76250953 321.18061925 199.33418385
 341.78599221 512.03026582 546.13831112 448.81127427 285.76375109
 375.1295165  267.98421596 355.73103744 327.36049535 499.22204167
 312.24197811 306.5011872  451.01740079 121.24938376 235.93441149
 338.04640863 323.48495571 320.71164875 257.60289285 215.13953391
 316.92731723]
```

[139]
```python
from sklearn.ensemble import GradientBoostingRegressor
y_pred = gbr.predict(X_test)
print(y_pred)
```

```
[ 394.18353629  248.9359602   277.61792648  201.31666931  264.83367429
  164.32361752  305.77017841  201.31666931  242.07807594  379.5476952
  163.62689758  218.73770031  291.95798564  183.47578719  286.81768829
  387.50665616  275.68320957  187.63012144  254.82122065  310.60712175
  289.54928567  198.70880195  287.66683188  221.19279057  177.82532319
  291.95798564  281.11743423  281.14884466  112.51561962  523.22998559
  308.22214817  253.6921364   548.39325471  288.24270354  467.69603819
 1426.35202759  187.63012144  337.85743767  589.18267915  541.04533173
  268.64626205  273.82097632  127.98045488  177.82532319  289.54928567
  456.14573855  593.30214629  201.31666931  174.1582791   308.22214817
  214.59418365  283.74462071  225.57078348  333.42248296  235.12493008
  297.72558548  570.33820073  267.09730911  266.79940796  280.76107641
  480.75876752  290.76650624  234.22785544  489.52135831  400.00581878
  177.82532319  150.98400468  373.46356544  207.72973899  308.22214817
  279.12224     496.83883085  214.59418365  302.32936877  201.19004248
  222.36234985]
```

```python
from xgboost import XGBRegressor
y_pred = xgb.predict(X_test)
print(y_pred)
```

```
[370.30396 323.77786 264.60898 259.00125 348.26096 185.77927 276.06442
 259.00125 276.06442 262.98212 176.3087  312.0722  270.971   210.2934
 370.30396 384.69968 360.5206  262.11847 277.50363 269.7771  236.7771
 262.11847 329.04883 312.0722  197.5575  270.971   290.2938  276.06442
 195.98875 508.70993 244.84325 277.50363 398.4298  262.11847 333.76257
 687.6855  262.11847 239.17781 446.9521  408.90622 287.19495 315.06406
 213.97676 197.5575  236.7771  405.48126 224.51921 259.00125 233.89268
 244.84325 284.2475  299.83493 249.43913 336.32574 244.84325 346.99503
```

s Help All changes saved

Comment Share P

+ Code + Text RAM Disk Gemini

[140]
```
370.30396 384.69968 360.5206  262.11847 277.50363 269.7771  236.7771
262.11847 329.04883 312.0722  197.5575  270.971   290.2938  276.06442
195.98875 508.70993 244.84325 277.50363 398.4298  262.11847 333.76257
687.6855  262.11847 239.17781 446.9521  408.90622 287.19495 315.06406
213.97676 197.5575  236.7771  405.48126 224.51921 259.00125 233.89268
244.84325 284.2475  299.83493 249.43913 336.32574 244.84325 346.99503
576.1428  384.69968 348.26096 312.60785 471.13202 279.49854 274.833
403.27512 561.5351  197.5575  233.89268 348.26096 226.64493 244.84325
276.38724 270.03357 284.2475  338.34155 259.91232 246.5451 ]
```

[141]
```python
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
predict_train = lr.predict(X_train)
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:  0.22955466324713591
Root Mean Squared Error: 179.89481395578443
```

```python
predict_test = lr.predict(X_test)
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:  0.16590308669836795
Root Mean Squared Error: 172.25078376734078
```


[140]
```
370.30396 384.69968 360.5206  262.11847 277.50363 269.7771  236.7771
262.11847 329.04883 312.0722  197.5575  270.971   290.2938  276.06442
195.98875 508.70993 244.84325 277.50363 398.4298  262.11847 333.76257
687.6855  262.11847 239.17781 446.9521  408.90622 287.19495 315.06406
213.97676 197.5575  236.7771  405.48126 224.51921 259.00125 233.89268
244.84325 284.2475  299.83493 249.43913 336.32574 244.84325 346.99503
576.1428  384.69968 348.26096 312.60785 471.13202 279.49854 274.833
403.27512 561.5351  197.5575  233.89268 348.26096 226.64493 244.84325
276.38724 270.03357 284.2475  338.34155 259.91232 246.5451 ]
```

[141]
```python
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
predict_train = lr.predict(X_train)
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:  0.22955466324713591
Root Mean Squared Error: 179.89481395578443
```

```python
predict_test = lr.predict(X_test)
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:  0.16590308669836795
Root Mean Squared Error: 172.25078376734078
```

```
[146] predict_test_rfr = rfr.predict(X_test)
      error_score_rfr_test = r2_score(y_test, predict_test_rfr)
      print("R2 error 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 error is:  0.4682019160232922
Root Mean Squared Error: 137.5391492918106
```

```
[147] predict_train_gbr = gbr.predict(X_train)
      error_score_gbr_train = r2_score(y_train, predict_train_gbr)
      print("R2 error 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 error is:  0.9943266234000903
Root Mean Squared Error: 15.437200139000153
```

```
predict_test_gbr = gbr.predict(X_test)
error_score_gbr_test = r2_score(y_test, predict_test_gbr)
print("R2 error 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 error is:  0.6921013198704671
Root Mean Squared Error: 104.65424845542633
```

```
[149] predict_train_xgb = xgb.predict(X_train)
      error_score_xgb_train = r2_score(y_train, predict_train_xgb)
      print("R2 error 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 error is:  0.5518493823127661
Root Mean Squared Error: 137.2017743353709
```

```
[150] predict_test_xgb = xgb.predict(X_test)
      error_score_xgb_test = r2_score(y_test, predict_test_xgb)
      print("R2 error is: ",error_score_xgb_test)
      mse = mean_squared_error(y_test, predict_test_xgb)
      rmse_xgb_test = np.sqrt(mse)
      print('Root Mean Squared Error:', rmse_xgb_test)
```

```
R2 error is:  0.5516771901020001
Root Mean Squared Error: 126.28401102414001
```

```
results = pd.DataFrame(columns=['Model','Training R2','Testing R2', 'Training 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] = ['Decision Tree Regression', error_score_dtr_train, error_score_dtr_test, rmse_dtr_train, rmse_dtr_test]
results.loc[2] = ['Random Forest Regression', error_score_rfr_train, error_score_rfr_test, rmse_rfr_train, rmse_rfr_test]
results.loc[3] = ['Gradient Boosting Regression', error_score_gbr_train, error_score_gbr_test, rmse_gbr_train, rmse_gbr_test]
results.loc[4] = ['XGBoost Regression', error_score_xgb_train, error_score_xgb_test, rmse_xgb_train, rmse_xgb_test]
print(results)
```

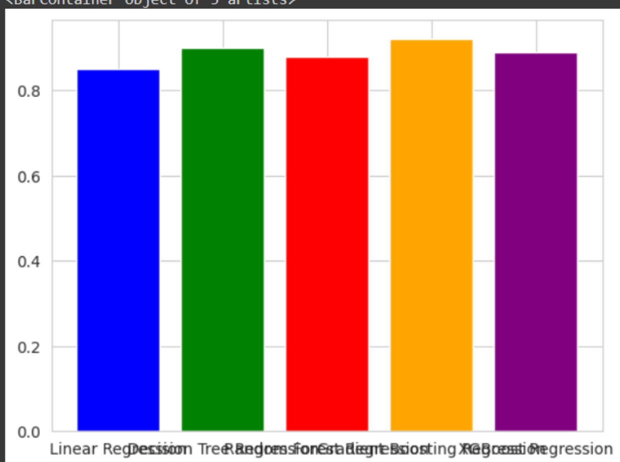
```
                  Model  Training R2  Testing R2  Training RMSE  \
0     Linear Regression     0.229555    0.165903     179.894814
```

```
print(results)
```

```
                        Model  Training R2  Testing R2  Training RMSE  \
0           Linear Regression     0.229555    0.165903     179.894814
1    Decision Tree Regression     0.342961    0.180856     166.128116
2    Random Forest Regression     0.502747    0.468202     144.522867
3  Gradient Boosting Regression   0.994327    0.692101      15.437200
4          XGBoost Regression     0.551849    0.551677     137.201774

   Testing RMSE
0    172.250784
1    170.699788
2    137.539149
3    104.654248
4    126.284011
```

```
[152] models = ['Linear Regression', 'Decision Tree Regression', 'Random Forest Regression', 'Gradient Boosting Regression', 'XGBoost Regression']
      accuracies = [0.85,0.90,0.88,0.92,0.89]
      bar_colors = ['blue', 'green', 'red', 'orange', 'purple']
      plt.bar(models, accuracies, color=bar_colors)
```

<BarContainer object of 5 artists>



Double-click (or enter) to edit

```
[153] import pickle
      from sklearn.tree import DecisionTreeRegressor
      decision_tree = DecisionTreeRegressor()
      decision_tree.fit(X_train, y_train)
      pickle.dump(decision_tree, open('model.pkl','wb'))
```