May 26,2023

DYNAMIC PRICING APP: OPTIMIZING PRICING STRATEGIES FOR BUSINESS SUCCESS

Machine Learning Model

Empower business with a dynamic pricing model app that analyzes market trends, competitor prices, and customer behavior to generate optimized pricing recommendations for increased revenue and profitability.

Khushi Gupta

Index

TOPICS	PAGE NO.
Abstract	2
Problem Statement	2
Business need assessment	2-3
Customer need assessment	3
Target specifications and characterization	4
Benchmarking alternate products	5
Areas where the existing apps lack	6-8
Applicable Regulations	8
Applicable Constraints	9-10
Business Idea	10-11
Final Product Prototype	11-12
Schematic Diagram	13
Product Details	14-15
Data Sources	16-17
Models of Dynamic Pricing Algorithm	17-18
Code Implementations	18-25
Conclusion	25
References	26

Abstract

Dynamic pricing is a pricing strategy that adjusts product or service prices in real time based on various factors such as demand, market conditions, and customer preferences. This abstract presents a dynamic pricing model that aims to maximize revenue and optimize customer satisfaction through adaptive pricing strategies. The model incorporates data-driven approaches, machine-learning algorithms, and economic principles to enable businesses to dynamically set prices in response to changing market dynamics.

Problem Statement

Traditional fixed pricing models are inflexible and unable to adapt to changing market conditions, resulting in missed revenue opportunities and dissatisfied customers. The business struggle to effectively utilize the vast amounts of available data to inform its pricing decisions. Implementing real-time pricing updates across multiple sales channels is challenging due to a lack of integration and infrastructure. Personalized pricing strategies that cater to individual customer segments are often overlooked.

To address these challenges, there is a need for a dynamic pricing model that leverages data-driven approaches, machine learning algorithms, and real-time updates to optimize pricing decisions, maximize revenue, and enhance customer satisfaction.

Business Need Assessment

- Maximizing Revenue: Businesses aim to maximize their revenue and profitability. A dynamic pricing model allows businesses to adjust prices based on real-time market conditions, demand fluctuations, and competitor behavior. By optimizing prices, businesses can capture the

maximum value for their products or services and capitalize on revenue opportunities.

- Responding to market dynamics: Markets are dynamic and constantly changing. Fixed pricing models often fail to respond quickly to market shifts, resulting in missed opportunities or overpricing. A dynamic pricing model enables businesses to adapt their prices in real-time, aligning with market demand, supply levels, and competitor pricing. This agility allows businesses to stay competitive and capitalize on market trends.
- Optimizing Inventory Utilization: Effective inventory management is crucial for businesses. By dynamically adjusting prices, businesses can optimize their inventory utilization.

Customer Need Assessment

- Competitive prices: Customers seek competitive prices when making purchasing decisions. If offered competitive prices, businesses can attract these price-sensitive customers and increase their satisfaction.
- Personalized pricing: Customers have different price sensitivities and preferences. A dynamic pricing model enables businesses to implement personalized pricing strategies based on customer segmentation and behavior analysis.
- Fairness and Transparency: Customers value fairness and transparency in pricing. A dynamic pricing model, when implemented correctly, ensures that prices are adjusted based on market dynamics rather than arbitrary factors. This builds trust and confidence among customers.
- Real-Time pricing information: Customers appreciate up-to-date and accurate pricing information. With a dynamic pricing model, businesses can provide real-time pricing updates across various sales channels. This eliminates confusion or frustration caused by outdated pricing information.

Target Specifications and Characterization

1. Real-time Data Integration: It must be able to collect and integrate real-time data from various sources, such as sales transactions, market trends, competitor pricing, and customer behavior. Continuous Updated model with the current information.
2. Advanced Analytics and Machine Learning: It should incorporate advanced analytics and machine learning algorithms to analyze the collected data and generate insights. These algorithms should be capable of identifying patterns, demand elasticity, and customer segmentation.
3. Pricing Strategy Customization: Allow Businesses to customize their pricing strategies based on their specific goals and requirements. It should offer flexibility in setting pricing rules, such as minimum and maximum price limits, discount thresholds, and pricing tiers according to customer segments.
4. Dynamic Price Optimization: It should provide intelligent price optimization capabilities and leverage the collected data and analytics to generate optimal pricing recommendations that maximize revenue and profitability. The optimization algorithms should consider factors such as cost, demand, inventory levels, competitive dynamics, and customer preferences to determine the most effective pricing strategies.
5. Pricing Rule Automation: It should automate the execution of pricing rules and adjustments based on predefined conditions. It should seamlessly integrate with the business's pricing infrastructure and sales channels to ensure real-time updates.It eliminates the need for manual price adjustments and ensures consistency.
6. Performance Monitoring and Reporting: It should provide comprehensive performance monitoring and reporting features. It should track key metrics such as revenue, profit margin, sales volume, and customer satisfaction.
7. User-friendly interface: The app should have a user-friendly interface that is intuitive and easy to navigate. It should provide

clear and actionable information to businesses, allowing them to make informed pricing decisions.

8. Scalability and Integrations: The app should be capable of handling large volumes of data and supports a growing number of users and transactions. The app should have integration capabilities with existing business systems, such as CRM(Customer Relationship Management) and ERP(Enterprise Resource Planning), to streamline data flow and ensure data consistency.

Bench Marking Alternate Products

1. Revionics: Revionics offers a dynamic pricing platform that leverages advanced analytics and machine learning algorithms to optimize pricing strategies. It provides capabilities for competitive pricing analysis, price optimization, and demand forecasting.
2. Prisync: Prisysnc is a pricing optimization and competitor tracking tool. It allows businesses to monitor competitor prices, track product availability, and receive real-time price change notifications.
3. Competera: Competera is a dynamic pricing and assortment optimization platform. It utilizes AI and machine learning algorithms to analyze market data, competitor pricing, and customer demand.
4. Omnia Retail: It offers a dynamic pricing and assortment optimization solution. It combines market data, competitive insights, and demand forecasting to provide pricing recommendations that align with business goals and market conditions.
5. Wiser: Wiser is a pricing intelligence and dynamic pricing solution. It provides real-time competitor price monitoring, pricing analytics, and dynamic repricing capabilities.

Areas in which the already existing apps lack

1. Real-time Data Integration: Improving the speed and efficiency of data integration from various sources can enhance the accuracy and timeliness of pricing recommendations. Streamlining the data collection process and reducing latency can provide more up-to-date insights for pricing decisions.
2. Advanced Analytics and Machine Learning: Continuously refining and enhancing the algorithms used for pricing analytics and machine learning can lead to more accurate demand forecasting, better price optimization, and improved insights into customer behavior. Incorporating more sophisticated techniques and models can enhance the app's analytical capabilities.
3. Customization and Flexibility: Offering greater customization options and flexibility in defining pricing rules and strategies can cater to the diverse needs of businesses. Providing more granular controls, segmentation capabilities, and pricing scenario simulations can enable businesses to align pricing strategies with their specific objectives.
4. User Interface and Experience: Enhancing the user interface and user experience can improve the ease of navigation, intuitiveness, and overall usability of the apps. Clear and visually appealing visualizations, intuitive workflows, and user-friendly interfaces can make it easier for users to understand pricing insights and make informed decisions.
5. Integration with Business Systems: Strengthening integration capabilities with existing business systems, such as CRM or ERP, can streamline data flow, reduce manual efforts, and improve overall efficiency. Seamless integration can enable the app to access relevant customer data, sales data, and inventory information to make more informed pricing recommendations.
6. Pricing Strategy Recommendations: Providing more detailed and actionable pricing recommendations can empower businesses to make informed decisions. Offering specific insights, such as price elasticity analysis, competitor benchmarking, and market trend analysis, can assist businesses in setting optimal prices.
7. Scalability and Performance: Enhancing the scalability and performance of the apps can ensure they can handle large volumes of data, support a

growing user base, and provide real-time updates. Optimizing processing speed, scalability, and system responsiveness can improve overall app performance.

8. Customer Support and Training: Offering comprehensive customer support, training resources, and documentation can help businesses effectively utilize the app's features and functionalities. Providing training materials, FAQs, and responsive support channels can enhance the overall customer experience.
9. Complexity: Some dynamic pricing apps may have a steep learning curve and complex setup process, requiring users to invest time and effort in understanding and implementing the app effectively.
10. Data Accuracy: Dynamic pricing relies heavily on accurate and up-to-date data. Inaccurate or incomplete data sources may affect the effectiveness of pricing recommendations, requiring careful data management and validation processes.
11. Integration Challenges: Integrating the app with existing business systems and data sources can pose challenges, particularly if the app doesn't offer seamless integration options or requires extensive customization to fit specific environments.
12. Overreliance on Historical Data: If the app primarily relies on historical data for pricing decisions, it may not account for real-time market dynamics, sudden changes in demand, or emerging trends, limiting its responsiveness to dynamic market conditions.
13. Lack of Customization: Some apps may have limited customization options, making it challenging to tailor pricing strategies to specific business needs or industry requirements. Lack of flexibility in defining pricing rules and segmentation may hinder optimal pricing strategies.
14. Pricing Transparency: Dynamic pricing can sometimes raise concerns about transparency and customer perception. Pricing strategies that are overly complex or appear unfair may negatively impact customer trust and loyalty.
15. Resource Intensive: Implementing and maintaining a dynamic pricing app may require significant resources in terms of time, IT infrastructure, and personnel. Small businesses with limited resources may find it challenging to fully utilize and optimize the app.
16. Scalability Issues: As the number of products, customers, and transactions increases, some apps may face scalability issues, impacting

performance and responsiveness. Ensure the app can handle large data volumes and growing user demands.

17. Cost: The cost of implementing and licensing dynamic pricing apps can vary significantly. High pricing structures may limit accessibility, particularly for smaller businesses with tighter budgets.

Applicable Regulations:

I. Consumer Protection Laws: Ensure compliance with consumer protection laws, which vary by country and region. These laws typically cover areas such as fair pricing practices, transparency in pricing, anti-discrimination, and protection against misleading or deceptive pricing tactics.

II. Data Protection and Privacy Laws: Protect user data and comply with data protection and privacy laws, such as the General Data Protection Regulation (GDPR) in the European Union or the California Consumer Privacy Act (CCPA) in the United States. Implement appropriate data security measures, obtain necessary consents, and handle personal data in accordance with relevant regulations.

III. Competition Laws: Comply with competition laws and regulations to prevent anti-competitive behavior, price fixing, or collusion with competitors. Understand the guidelines and restrictions related to pricing practices, market dominance, and fair competition.

IV. Industry-Specific Regulations: Depending on the industry in which your app operates, there may be specific regulations to consider. For example, if you are in the financial sector, you may need to comply with regulations related to financial services and pricing transparency.

V. Pricing Display Regulations: Some jurisdictions have specific regulations regarding how prices should be displayed to consumers. This may include requirements for clear and accurate price labeling, currency conversion rules, or mandatory disclosure of additional fees or charges.

VI. Contractual Obligations: Ensure that the dynamic pricing model app complies with any contractual agreements or obligations with customers, partners, or third-party vendors.

Applicable constraints

I. Regulatory Constraints: As mentioned earlier, compliance with applicable laws and regulations is a critical constraint. Ensure that the app adheres to consumer protection laws, data protection and privacy regulations, competition laws, and any industry-specific regulations.

II. Data Availability and Quality: The availability and quality of data can be a constraint. The app relies on accurate and timely data from various sources, such as sales transactions, market trends, competitor pricing, and customer behavior. Ensure that data collection mechanisms are in place and that the data is reliable and sufficient for accurate analysis and decision-making.

III. Technology Infrastructure: The app's performance and scalability depend on the underlying technology infrastructure. Consider constraints such as computing power, data storage capacity, processing speed, and network connectivity to ensure that the app can handle the required data volumes and respond in real-time.

IV. Integration with Existing Systems: If the app needs to integrate with existing business systems, constraints related to compatibility, data synchronization, and security protocols may arise. Ensure that the app can seamlessly integrate with relevant systems to enable efficient data exchange and pricing updates across different platforms.

V. Budget and Resources: Consider the constraints of budget and resources, including development costs, ongoing maintenance, and support. Assess the feasibility of building and maintaining the app within the available resources, or consider outsourcing options if necessary.

VI. Time Constraints: Development timelines and deployment schedules can be a constraint. Ensure that the app can be developed and deployed within the desired timeframe, considering factors such as development complexity, testing requirements, and regulatory compliance processes.

VII. User Experience and Acceptance: The app's success relies on user acceptance and adoption. Consider constraints related to user experience, interface design, and ease of use. Aim to create an intuitive and user-friendly app that meets the needs and expectations of its intended users.

VIII. Ethical Considerations: Pricing decisions can have ethical implications. Constraints related to fairness, transparency, and avoiding discriminatory practices should be considered. Ensure that the app's pricing strategies align with ethical standards and promote a positive customer experience.

Business Idea:

There are various monetization ideas for a dynamic pricing app. Here are a few options:

- ✓ Subscription Model: By offering different tiers of subscription plans with varying features and pricing levels. Users can choose a plan that suits their needs, and you can provide additional value-added services or advanced features for higher-tier subscribers.
- ✓ Transaction Fees: Charge a small percentage or fixed fee for each transaction facilitated through the app. This could apply to sales transactions where the app's dynamic pricing recommendations are used.
- ✓ Data Insights and Analytics: Offer premium data insights and analytics as an add-on service. Provide in-depth market analysis, competitor benchmarking, and customized reports that businesses can access for a fee.
- ✓ Consultancy and Support: Provide consultancy and support services related to dynamic pricing strategy development, implementation, and optimization. Businesses can engage your team for personalized assistance and guidance.
- ✓ White-labelling and Licensing: Offer the app as a white-label solution or license the technology to other businesses that want to incorporate dynamic pricing capabilities into their own systems. Generate revenue through licensing fees or revenue-sharing agreements.
- ✓ Performance-based Pricing: Introduce a performance-based pricing model that charges a percentage of the revenue increase generated through the app's dynamic pricing recommendations. This aligns your revenue with the value delivered to businesses using the app.

- ✓ API Access: Providing access and charging a fee for app's APIs (Application Programming Interfaces) for businesses to integrate and utilize the dynamic pricing capabilities within their own systems.

Final Product Prototype:

Frontend:

1. User Interface (UI): The frontend is responsible for presenting the user interface, allowing users to interact with the app. It includes screens, forms, and visual elements that users can see and interact with.

2. User Input and Interactions: The frontend captures user input, such as login credentials, data uploads, customization settings, and pricing adjustments. It handles user interactions, such as button clicks, form submissions, and menu selections.

3. Displaying Data and Results: The frontend receives data from the backend and displays it to the user in a visually appealing and user-friendly manner. This includes presenting pricing insights, reports, charts, and visualizations.

4. Real-Time Updates: The frontend may implement real-time updates to display dynamic pricing changes, notifications, and alerts to users. This ensures users have the latest information without manually refreshing the page.

Backend:

1. Server and Infrastructure: The backend consists of servers, databases, and infrastructure that support the app's operations. It handles requests from the frontend, processes data, and provides responses.

2. Data Management: The backend manages data storage and retrieval. It stores user information, pricing rules, historical data, competitor data, and other relevant data required for pricing analysis.

3. Data Processing and Analysis: The backend performs complex data processing and analysis tasks using algorithms, machine learning models, and pricing rules. It crunches data from various sources, applies pricing algorithms, and generates pricing recommendations.

4. Pricing Engine: The backend houses the pricing engine, which applies business rules, segmentation criteria, and pricing strategies to the data. It calculates optimal prices based on factors like demand, competition, costs, and objectives.

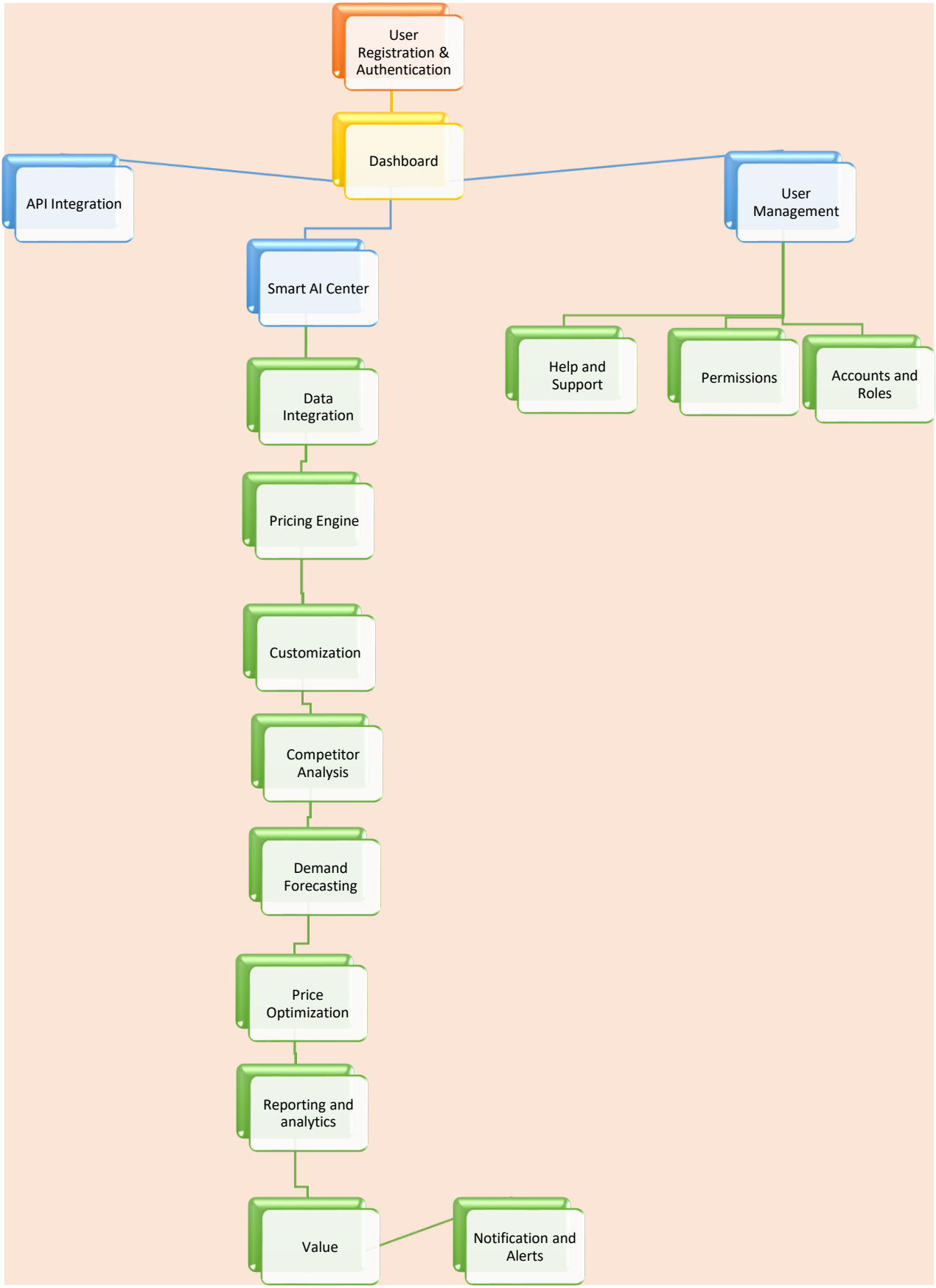
5. Integration with External Systems: The backend handles integrations with external systems, such as CRM, ERP, or data providers, to fetch additional data required for pricing analysis. It ensures seamless data exchange and synchronization.

6. APIs and Data Exchange: The backend exposes APIs (Application Programming Interfaces) that allow the frontend and external systems to communicate with the app. APIs enable data exchange, functionality access, and integration capabilities.

7. Performance Monitoring and Logging: The backend tracks app performance, logs errors, and captures relevant metrics. It helps in identifying bottlenecks, optimizing performance, and ensuring system stability.

8. Security and User Management: The backend implements security measures to protect user data, authenticate user access, and enforce user roles and permissions. It manages user accounts, authentication, and authorization processes.

Schematic Diagram


User Registration & Authentication
Dashboard
API Integration
User Management
Smart AI Center
Help and Support
Permissions
Accounts and Roles
Data Integration
Pricing Engine
Customization
Competitor Analysis
Demand Forecasting
Price Optimization
Reporting and analytics
Value
Notification and Alerts

Product Details

Product Name: FlexiPrice

Working of the application/ User Flow:-

1. User Registration/Login:
    a. Users create an account or log in to the app using their credentials.
    b. If new users, they may need to provide the necessary information and complete the registration process.
2. Dashboard/Homepage:
    a. Upon successful login, users are directed to the dashboard or homepage.
    b. The dashboard overviews key pricing insights, performance metrics, and relevant data.
3. Data Integration:
    a. Users can integrate data sources, such as sales data, competitor prices, and market trends.
    b. They can connect to external systems or upload data files for analysis.
4. Pricing Analysis:
    a. Users access the pricing analysis section to view insights and recommendations.
    b. The app analyzes data, competitor prices, and market trends to generate pricing recommendations.
5. Customization:
    a. Users can customize pricing rules, strategies, and segmentation criteria to align with their business objectives.
    b. They define parameters such as pricing thresholds, pricing tiers, and promotional strategies.
6. Competitor Analysis:
    a. Users can monitor and analyze competitor pricing.
    b. The app provides insights on competitor prices, pricing gaps, and benchmarking against industry competitors.
7. Demand Forecasting:

a. Users can access demand forecasting features to predict customer demand patterns.
b. The app utilizes historical data and market trends to forecast future demand and recommend pricing adjustments.

8. Price Optimization:
    a. Users have the option to optimize prices based on the app's recommendations.
    b. They can simulate pricing scenarios and analyze the potential impact on sales, revenue, and profitability.
9. Reporting and Analytics:
    a. The app generates comprehensive reports, charts, and visualizations to present pricing insights and performance trends.
    b. Users can access these reports to make data-driven pricing decisions.
10. Notifications and Alerts:
    a. Users receive notifications and alerts for significant changes in market conditions, competitor prices, or pricing thresholds.
    b. These alerts enable users to take timely actions to adjust prices accordingly.
11. API Integration:
    a. Users can integrate the app with other business systems, such as CRM or ERP, through provided APIs.
    b. This allows for seamless data synchronization and workflow automation.
12. User Management:
    a. Administrators have access to user management features.
    b. They can manage user accounts, roles, and permissions within the app.
13. Help and Support:
    a. Users can access help resources, FAQs, and contact customer support for assistance with the app's features and functionalities.

Data Sources

1. Internal Sales Data: Historical sales data from your own business provides valuable insights into product performance, demand patterns, and customer behavior. It includes information such as transaction details, purchase history, and customer preferences.

2. Competitor Prices: Accessing competitor pricing data is crucial for effective dynamic pricing. This can be obtained through web scraping, price monitoring tools, or APIs provided by price comparison platforms. It allows you to analyze competitor pricing strategies, identify pricing gaps, and adjust your own prices accordingly.

3. Market Data and Trends: Data sources that offer market insights and trends can be valuable for dynamic pricing. This includes industry reports, market research data, economic indicators, and consumer behavior analysis. It helps in understanding market dynamics, identifying demand patterns, and making informed pricing decisions.

4. Inventory and Supply Chain Data: Integrating data related to inventory levels, production costs, and supply chain information allows for more accurate pricing. It helps in determining optimal pricing thresholds based on product availability, cost fluctuations, and production constraints.

5. Customer Data: Customer data, such as demographics, purchase history, preferences, and loyalty program information, provides valuable insights into individual customer behavior. It helps in implementing personalized pricing strategies and segmenting customers based on their price sensitivity or purchasing patterns.

6. External Data Providers: There are third-party data providers that offer specialized data sets relevant to dynamic pricing. This can include data on competitor prices, market trends, weather conditions, social media sentiment, or other industry-specific data. Integration with these providers can enhance the accuracy and comprehensiveness of your pricing analysis.

7. Real-Time Data Feeds: In certain industries or sectors, real-time data feeds can be crucial for dynamic pricing. For example, in airline ticket pricing, integrating real-time flight availability and demand information allows for dynamic pricing adjustments based on changing market conditions.

Team required to develop:

- Project Manager
- Business Analyst
- UX/UI Designer
- Frontend Developer
- Backend Developer
- Data Scientist
- Database Administrator
- Quality Assurance Engineer
- DevOps Engineer
- Security Specialist

Development Cost: The cost of developing a dynamic pricing app can vary significantly depending on various factors, including the complexity of the app, desired features, technology stack, development approach, team size, and geographic location. It's challenging to provide an exact cost without detailed project requirements and specifications.

Models of dynamic pricing algorithms

---

1. Bayesian model

In a Bayesian model, the user picks a prior value indicating the initial belief about the possible price. Then, whenever a new data point is entered into the algorithm, the initial belief shifts either higher or lower. This type of dynamic pricing model uses historical pricing data as the most important feature to decide on the final price, like a typical pricing algorithm.

2. Reinforcement learning model

Reinforcement learning (RL) is a goal-directed dynamic pricing model which aims to achieve the highest rewards by learning from environmental data. An RL dynamic pricing model analyzes data regarding customers' demand, taking into account seasonality, competitor prices, and the uncertainty of the market, to achieve a revenue optimal price.

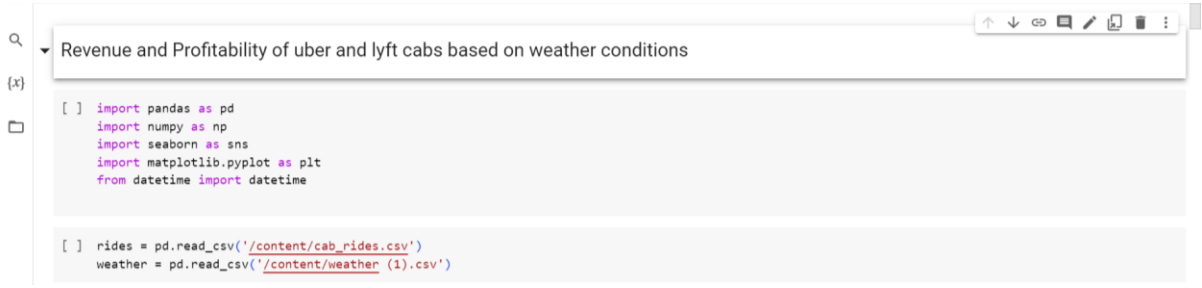
3. Decision tree model

Decision trees are classification machine learning models that output a tree-like model of decisions and their possible consequences, including the possibility of a certain outcome, resource costs, and utility. Decision tree dynamic pricing algorithms help businesses understand which parameters have the most effect on the prices and which of these price ranges predicts the highest revenues, and using this information, the algorithm predicts the best price range for each product.

Code Implementation

Here, I have taken the example of Uber cab fare prices changing according to weather conditions.

#Importing libraries and loading the dataset

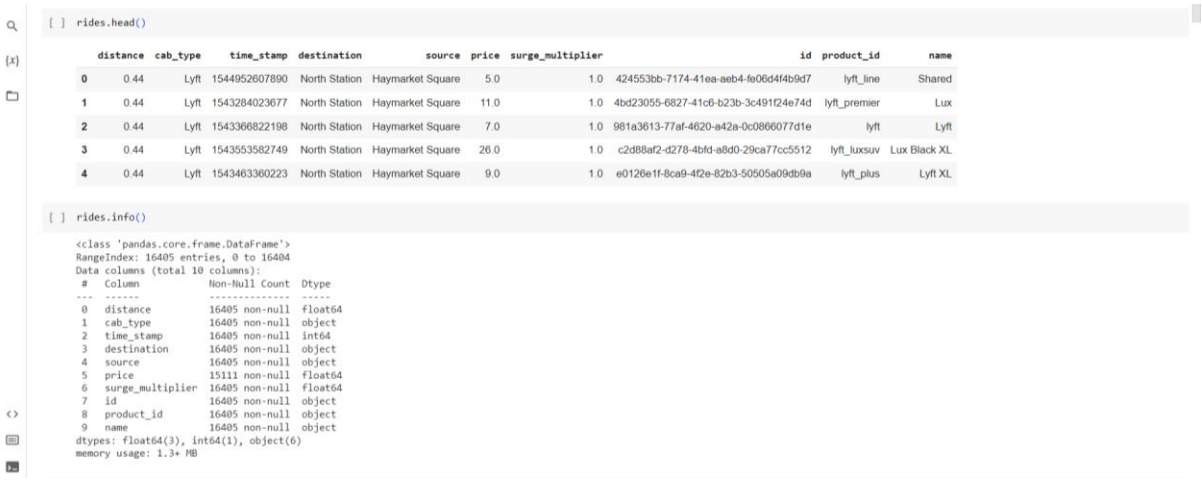


Revenue and Profitability of uber and lyft cabs based on weather conditions

```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from datetime import datetime
```

```
[ ] rides = pd.read_csv('/content/cab_rides.csv')
    weather = pd.read_csv('/content/weather (1).csv')
```

#Exploratory Data Analysis



```
[ ] rides.head()
```

	distance	cab_type	time_stamp	destination	source	price	surge_multiplier	id	product_id	name
0	0.44	Lyft	1544952607890	North Station	Haymarket Square	5.0	1.0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	lyft_line	Shared
1	0.44	Lyft	1543284023677	North Station	Haymarket Square	11.0	1.0	4bd23055-6827-41c6-b23b-3c491f24e74d	lyft_premier	Lux
2	0.44	Lyft	1543366822198	North Station	Haymarket Square	7.0	1.0	981a3613-77af-4620-a42a-0c0866077d1e	lyft	Lyft
3	0.44	Lyft	1543553582749	North Station	Haymarket Square	26.0	1.0	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	lyft_luxsuv	Lux Black XL
4	0.44	Lyft	1543463360223	North Station	Haymarket Square	9.0	1.0	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	lyft_plus	Lyft XL

```
[ ] rides.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16405 entries, 0 to 16404
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   distance          16405 non-null  float64
 1   cab_type          16405 non-null  object
 2   time_stamp        16405 non-null  int64
 3   destination       16405 non-null  object
 4   source            16405 non-null  object
 5   price             15111 non-null  float64
 6   surge_multiplier  16405 non-null  float64
 7   id                16405 non-null  object
 8   product_id        16405 non-null  object
 9   name              16405 non-null  object
dtypes: float64(3), int64(1), object(6)
memory usage: 1.3+ MB
```

```
[ ] weather.head()
```

	temp	location	clouds	pressure	rain	time_stamp	humidity	wind
0	42.42	Back Bay	1.0	1012.14	0.1228	1545003901	0.77	11.25
1	42.43	Beacon Hill	1.0	1012.15	0.1846	1545003901	0.76	11.32
2	42.50	Boston University	1.0	1012.15	0.1089	1545003901	0.76	11.07
3	42.11	Fenway	1.0	1012.13	0.0969	1545003901	0.77	11.09
4	43.13	Financial District	1.0	1012.14	0.1786	1545003901	0.75	11.49

```
[ ] weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6276 entries, 0 to 6275
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   temp        6276 non-null   float64
 1   location    6276 non-null   object 
 2   clouds      6276 non-null   float64
 3   pressure    6276 non-null   float64
 4   rain        894 non-null    float64
 5   time_stamp  6276 non-null   int64  
 6   humidity    6276 non-null   float64
 7   wind        6276 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 392.4+ KB
```

#Converting timestamp into weekdays and hours

```
[ ] rides['time_stamp']= pd.to_datetime(rides['time_stamp']/1000,unit='s')
    rides['date'] = rides['time_stamp'].dt.date
    rides['time'] = rides['time_stamp'].dt.hour
    rides['weekday'] = rides['time_stamp'].dt.weekday
```

```
[ ] rides['dollars per mile'] = rides['price']/rides['distance']
```

```
[ ] weather['time_stamp']=pd.to_datetime(weather['time_stamp'],unit='s')
    weather['date'] = weather['time_stamp'].dt.date
    weather['time'] = weather['time_stamp'].dt.hour
```

```
[ ] print(rides['name'].unique())
```

```
['Shared' 'Lux' 'Lyft' 'Lux Black XL' 'Lyft XL' 'Lux Black' 'UberXL'
 'Black' 'UberX' 'WAV' 'Black SUV' 'UberPool' 'Taxi']
```

```
[ ] # Creating a new dataframe with just the desired services
    rides = rides[(rides['name']=='Lyft')|(rides['name']=='UberX')]
    rides.reset_index(drop=True,inplace=True)

    # Confirming that only UberX and Lyft remain
    print(rides['name'].unique())
```

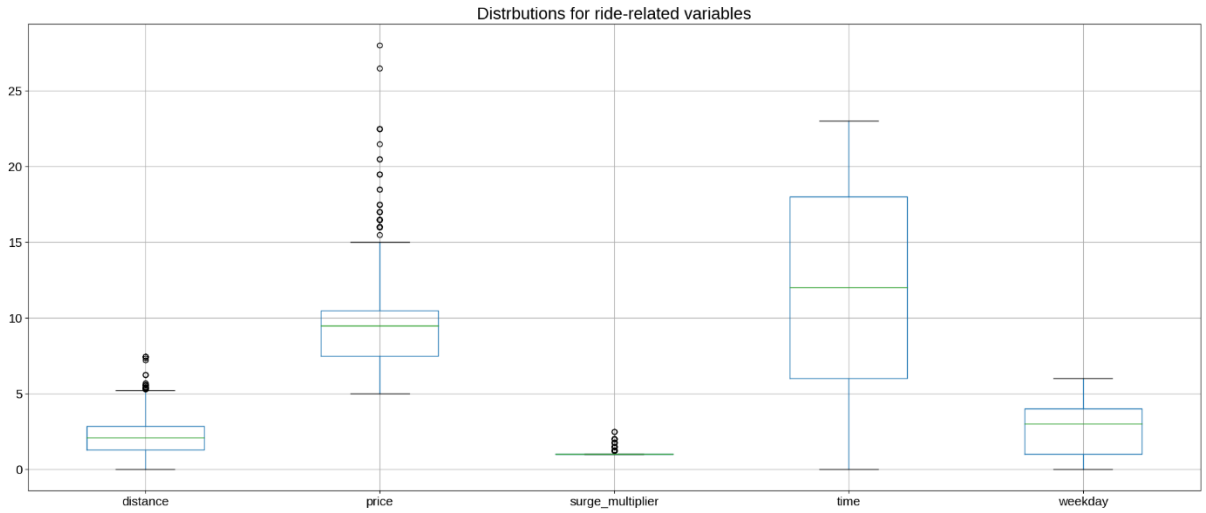
```
['Lyft' 'UberX']
```

```
[ ] cols = ['distance','price','surge_multiplier','time','weekday']
```

#Checking the outliers

```
[ ]
    # Observing characteristics of the distribution of the numerical variables for the rides dataframe
    print(rides[cols].describe())

    # Creating a function to return the necessary information to remove the outliers
    def iqr(df):
        q1 = df.quantile(.25)
        q3 = df.quantile(.75)
        iqr = q3 - q1
        upper_limit = q3 + 1.5*iqr
        lower_limit = q1 - 1.5*iqr
        print(' ')
        print('The IQRs are:')
        print(iqr)
        print(' ')
        print('The upper limits for outliers are:')
        print(upper_limit)
        print(' ')
        print('The lower limits for outliers are:')
        print(lower_limit)
        print(' ')

    # Calculating IQR, as well as upper and lower limits for outliers in case it becomes necessary to remove them
    iqr(rides)

    # Visualizing the distribution of the same variables
    rides[cols].boxplot(figsize=((25,10)),fontsize=15)
    plt.title('Distrbutions for ride-related variables',fontsize=20)
    plt.style.use('seaborn')

    plt.show()
```

```
[ ]          distance        price  surge_multiplier         time      weekday
count  2509.000000  2509.000000       2509.00000  2509.000000  2509.000000
mean      2.153041     9.605420          1.01943    11.668792     2.731367
std       1.116358     2.338897          0.11026     6.973799     2.013604
min       0.020000     5.000000          1.00000     0.000000     0.000000
25%       1.270000     7.500000          1.00000     6.000000     1.000000
50%       2.090000     9.500000          1.00000    12.000000     3.000000
75%       2.860000    10.500000          1.00000    18.000000     4.000000
max       7.460000    28.000000          2.50000    23.000000     6.000000

The IQRs are:
distance             1.590000
price                3.000000
surge_multiplier     0.000000
time                12.000000
weekday              3.000000
dollars per mile     2.742947
dtype: float64

The upper limits for outliers are:
distance             5.245000
price               15.000000
surge_multiplier     1.000000
time                36.000000
weekday              8.500000
dollars per mile    10.478056
dtype: float64

The lower limits for outliers are:
distance            -1.11500
price                3.00000
surge_multiplier     1.00000
time               -12.00000
weekday             -3.50000
dollars per mile    -0.49373
dtype: float64
```



No need to remove the outliers as they are contributing to the change in fare prices.

```
[ ] print('')
    print('Description of the variable "dollars per mile", a proxy for ride profitability:')
    print('')
    print(rides['dollars per mile'].describe())

    # Plotting the distribution of the proxy for profitability of each ride: dollars per mile
    plt.figure(figsize=((25,10)))
    plt.title('Distribution of profitability',fontsize=20)
    plt.boxplot(rides['dollars per mile'])
    plt.xticks([])
    plt.yticks(fontsize=15)
    plt.ylabel('Dollars per mile',fontsize=15)

    plt.style.use('seaborn')
    plt.show()

    Description of the variable "dollars per mile", a proxy for ride profitability:

    count    2509.000000
    mean        5.711502
    std         7.928246
    min         2.076677
    25%         3.620690
    50%         4.681648
    75%         6.363636
    max       375.000000
    Name: dollars per mile, dtype: float64
    <ipython-input-19-2fb67634bb1b>:14: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seabo
      plt.style.use('seaborn')
```

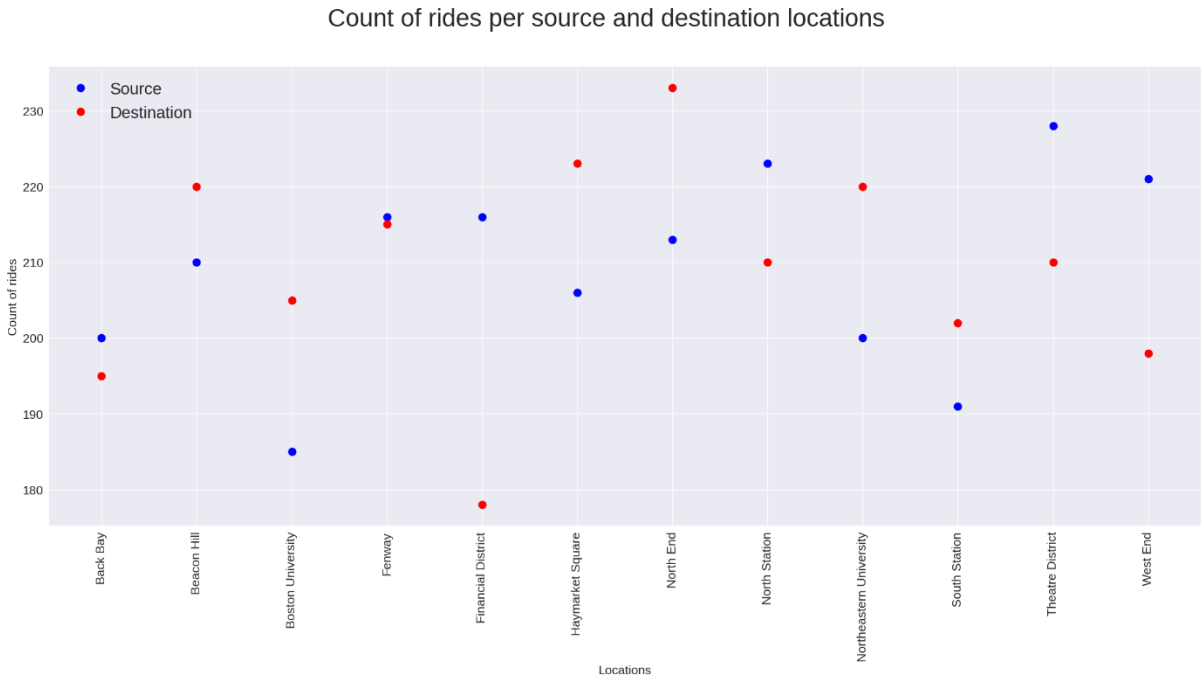
```
[ ] # creating one variable for the rides with profitability above 50$/mile
    profit_outliers = rides[rides['dollars per mile']>50]

    # Description of the distribution of the outtliers
    print(profit_outliers.describe())
```

```
       distance  price  surge_multiplier  time  weekday  dollars per mile
count      1.00    1.0               1.0   1.0      1.0               1.0
mean       0.02    7.5               1.0  23.0      0.0             375.0
std         NaN    NaN               NaN   NaN      NaN               NaN
min        0.02    7.5               1.0  23.0      0.0             375.0
25%        0.02    7.5               1.0  23.0      0.0             375.0
50%        0.02    7.5               1.0  23.0      0.0             375.0
75%        0.02    7.5               1.0  23.0      0.0             375.0
max        0.02    7.5               1.0  23.0      0.0             375.0
```

#Count of rides per source and destination

```
[ ] fig,ax = plt.subplots(figsize=((25,10)))
    plt.plot(source_count,'blue',label='Source',marker='.',linestyle='none',markersize=20)
    plt.plot(destination_count,'red',label='Destination',marker='.',linestyle='none',markersize=20)

    fig.suptitle('Count of rides per source and destination locations',fontsize=30)
    plt.ylabel('Count of rides',fontsize=15)
    plt.xlabel('Locations',fontsize=15)
    plt.yticks(fontsize=15)
    plt.xticks(fontsize=15,rotation=90)
    plt.legend(fancybox=True,fontsize=20)
    plt.style.use('seaborn')

    plt.show()
```

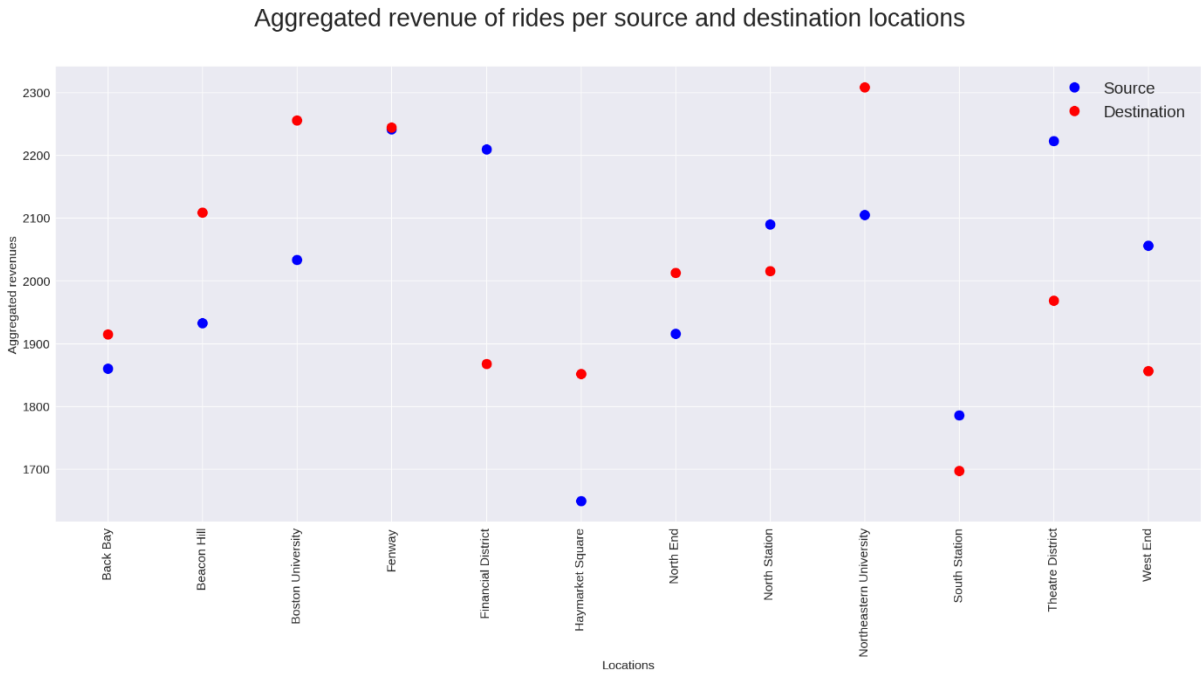
```
<ipython-input-27-e9882b951a46>:11: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seabo
  plt.style.use('seaborn')
```



#Aggregated revenue of rides per source and destination

```
[ ] fig,ax = plt.subplots(figsize=((25,10)))
    plt.plot(source_sum,'blue',label='Source',marker='.',linestyle='none',markersize=25)
    plt.plot(destination_sum,'red',label='Destination',marker='.',linestyle='none',markersize=25)

    fig.suptitle('Aggregated revenue of rides per source and destination locations',fontsize=30)
    plt.ylabel('Aggregated revenues',fontsize=15)
    plt.xlabel('Locations',fontsize=15)
    plt.legend(fancybox=True,fontsize=20)
    plt.style.use('seaborn')
    plt.yticks(fontsize=15)
    plt.xticks(fontsize=15,rotation=90)

    plt.show()
```

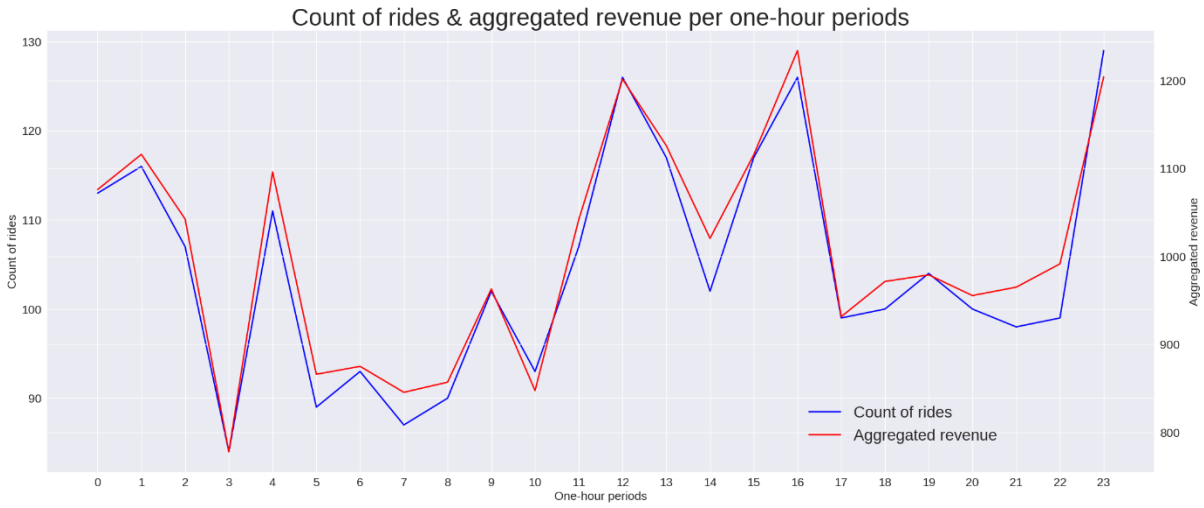
```
<ipython-input-30-d7225b6e16a2>:9: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seabor
  plt.style.use('seaborn')
```



#Count of rides & aggregated revenue per one -hour periods

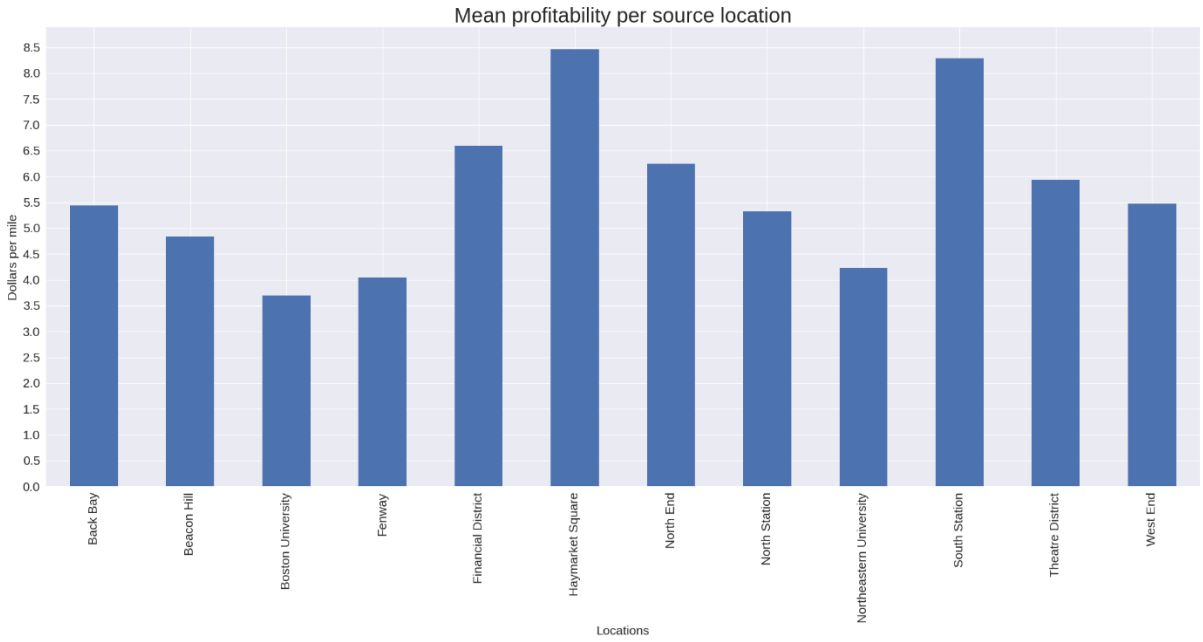
```
[ ] time_count = rides.groupby('time')['id'].count()
    time_sum = rides.groupby('time')['price'].sum()
    fig,ax1 = plt.subplots(figsize=((25,10)))
    plt.plot(time_count,'blue',label='Count of rides')
    plt.title('Count of rides & aggregated revenue per one-hour periods',fontsize=30)
    plt.ylabel('Count of rides',fontsize=15)
    plt.xlabel('One-hour periods',fontsize=15)
    plt.yticks(fontsize=15)
    plt.xticks(np.arange(0,24,1),fontsize=15)

    ax2 = ax1.twinx()
    plt.plot(time_sum,'red',label='Aggregated revenue')
    plt.ylabel('Aggregated revenue',fontsize=15)
    plt.style.use('seaborn')
    plt.yticks(fontsize=15)
    plt.xticks(fontsize=15)

    fig.legend(bbox_to_anchor=(0.8,0.25),fancybox=True,fontsize=20)
    plt.show()
```

```
<ipython-input-31-73cd74cefb4d>:14: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seabo
  plt.style.use('seaborn')
```



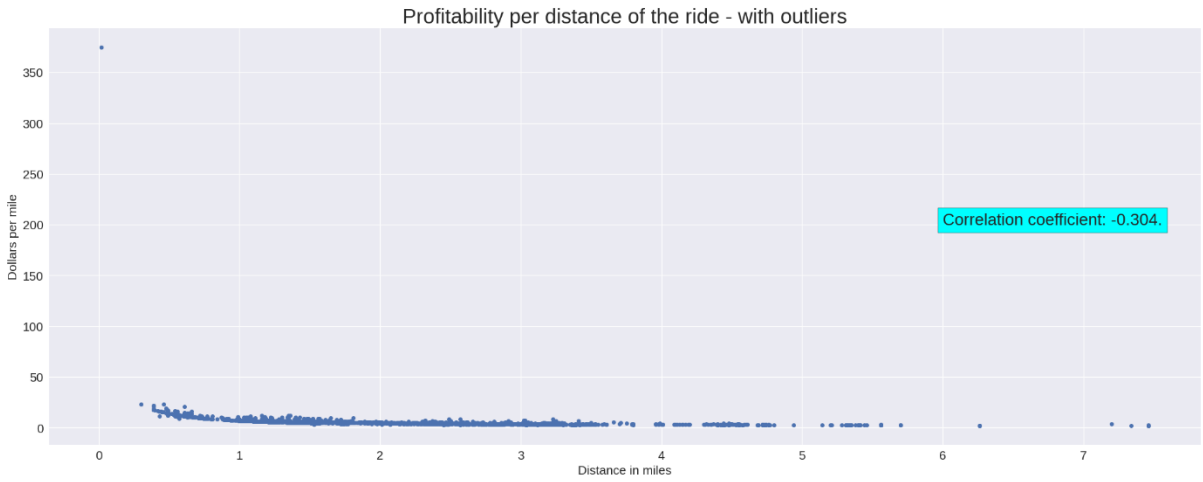
#Mean profitability per source location

```
[ ] # Grouping profitability by source location of the ride
    source_profitability = rides.groupby('source')['dollars per mile'].mean()

    #Plotting profitability per source location
    plt.figure(figsize=((25,10)))
    source_profitability.plot.bar()
    plt.title('Mean profitability per source location',fontsize=25)
    plt.ylabel('Dollars per mile',fontsize=15)
    plt.xlabel('Locations',fontsize=15)
    plt.style.use('seaborn')
    plt.yticks(np.arange(0,9,0.5),fontsize=15)
    plt.xticks(fontsize=15,rotation=90)

    plt.show()

    <ipython-input-36-c835e5ae6eb2>:10: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seabo
      plt.style.use('seaborn')
```



#Profitability per distance with or without outliers

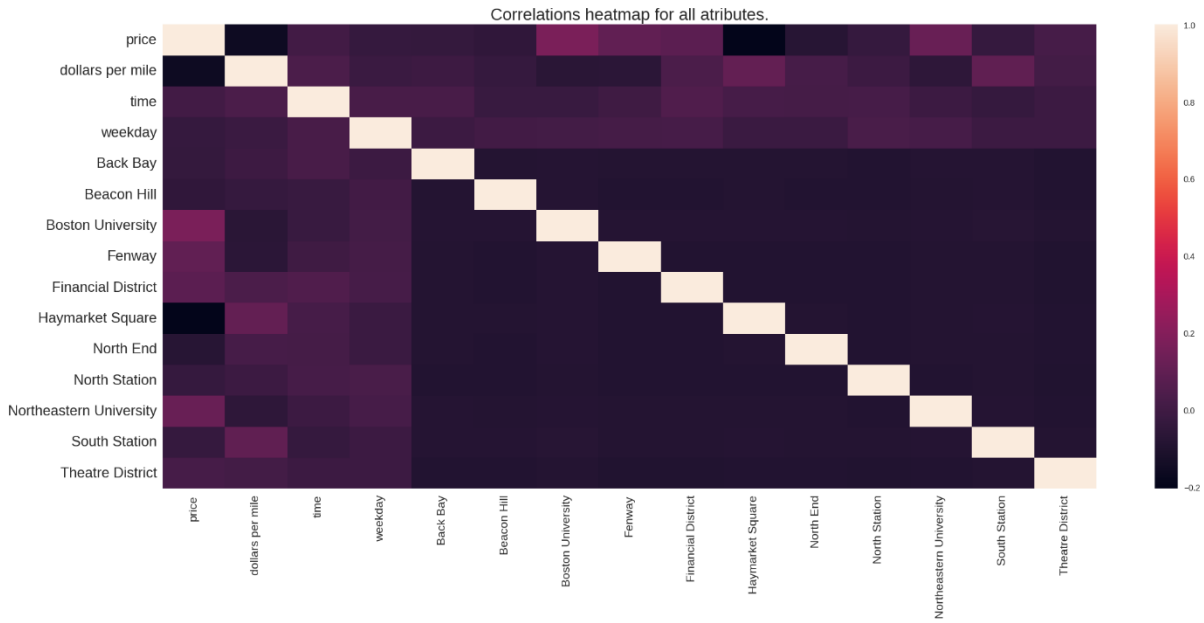
```
[ ] # Calculation correlation between profitability with the outliers and distance of the rides
    corr_a = rides['distance'].corr(rides['dollars per mile']).round(3)

    # Calculation correlation between profitability without the outliers and distance of the rides
    corr_b = rides_profit['distance'].corr(rides_profit['dollars per mile']).round(3)

    plt.figure(figsize=((25,20)))

    # Plotting profitability with the outliers against the distance of the rides
    plt.subplot(2,1,1)
    plt.plot(rides['distance'],rides['dollars per mile'],marker='.',linestyle='none',markersize=10)
    plt.title('Profitability per distance of the ride - with outliers',fontsize=25)
    plt.ylabel('Dollars per mile',fontsize=15)
    plt.xlabel('Distance in miles',fontsize=15)
    plt.yticks(fontsize=15)
    plt.xticks(fontsize=15)
    plt.text(6,200,'Correlation coefficient: '+str(corr_a)+'.',fontsize=20,bbox=dict(boxstyle="square,pad=0.3",fc="cyan"))

    # Plotting profitability without the outliers against the distance of the rides
    plt.subplot(2,1,2)
    plt.plot(rides_profit['distance'],rides_profit['dollars per mile'],marker='.',linestyle='none',markersize=10)
    plt.title('Profitability per distance of the ride - without outliers',fontsize=25)
    plt.ylabel('Dollars per mile',fontsize=15)
    plt.xlabel('Distance in miles',fontsize=15)
    plt.yticks(fontsize=15)
    plt.xticks(fontsize=15)
    plt.text(6,8,'Correlation coefficient: '+str(corr_b)+'.',fontsize=20,bbox=dict(boxstyle="square,pad=0.3",fc="cyan"))

    plt.show()
```



This shows that the profitability decreases as the distance increases which is clearly shown by negative correlation with or without outliers.

#Correlation heatmap of all attributes

```
[ ] # Creating an object with the dummy variables for each source location
    dummies = pd.get_dummies(rides['source'])
    # Concatenating the rides dataframe object and the dummy object into a new one to be used for the OLS analysis only
    rides_ols = pd.concat([rides,dummies], axis=1)
    # Dropping one dummy column to avoid the dummy variable trap
    rides_ols.drop(['West End'], inplace=True, axis=1)

    # Columns of interest in the regression and correlation analysis
    filt1 = ['price','dollars per mile','time','weekday','Back Bay', 'Beacon Hill',
           'Boston University', 'Fenway', 'Financial District', 'Haymarket Square',
           'North End', 'North Station', 'Northeastern University',
           'South Station', 'Theatre District']
    rides_ols_corr = rides_ols[filt1].corr()

    # Plotting the correlations heatmap
    plt.figure(figsize=((25,10)))
    ax = sns.heatmap(rides_ols_corr)
    plt.yticks(fontsize=17)
    plt.xticks(fontsize=14)
    plt.title('Correlations heatmap for all atributes.',fontsize=20)

    plt.show()
```



Dataset link:

https://www.kaggle.com/datasets/mayankvashisht/uber-cab-fare-price-analysis

Here is the link for the google collaboratory notebook:

https://colab.research.google.com/drive/19T8w1ZZaCP9W00LHwPktRq4SMTJZyGYy?usp=sharing

GitHub Link:

https://github.com/Khhushhiiii/Uber-lyft-cab_pricing-

Conclusion

A dynamic pricing app holds significant potential to optimize pricing strategies, enhance competitiveness, and drive business revenue growth. By leveraging real-time data, market insights, and advanced pricing algorithms, this app will enable businesses to dynamically adjust prices based on factors such as demand fluctuations, customer behavior, environmental factors, and competitor pricing.

By embracing this app business can unlock the potential for increased profitability and growth.

References

1) https://medium.com/total-data-science/how-machine-learning-is-helping-in-providing-dynamic-pricing-7efdb8af9083
2) https://www.width.ai/post/dynamic-pricing-price-optimization
3) https://tryolabs.com/blog/price-optimization-machine-learning
4) https://www.oreilly.com/library/view/the-new-frontier/53863MIT59102/.
5) https://hal.science/hal-01942038v2/preview/ArticleNoFormat.pdf
6) https://www.researchgate.net/publication/323202093_Machine_Learning_Methods_to_Perform_Pricing_Optimization_A_Comparison_with_Standard_GLMs