Dynamic pricing Strategy

September 4, 2024

[1]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

data = pd.read_csv("dynamic_pricing.csv")
print(data.head())
```

```
   Number_of_Riders  Number_of_Drivers Location_Category  \
0                90                 45             Urban
1                58                 39          Suburban
2                42                 31             Rural
3                89                 28             Rural
4                78                 22             Rural

  Customer_Loyalty_Status  Number_of_Past_Rides  Average_Ratings  \
0                  Silver                    13             4.47
1                  Silver                    72             4.06
2                  Silver                     0             3.99
3                 Regular                    67             4.31
4                 Regular                    74             3.77

  Time_of_Booking Vehicle_Type  Expected_Ride_Duration  \
0           Night      Premium                      90
1         Evening      Economy                      43
2       Afternoon      Premium                      76
3       Afternoon      Premium                     134
4       Afternoon      Economy                     149

   Historical_Cost_of_Ride
0               284.257273
1               173.874753
2               329.795469
3               470.201232
4               579.681422
```

[2]:
```
print(data.describe())
```

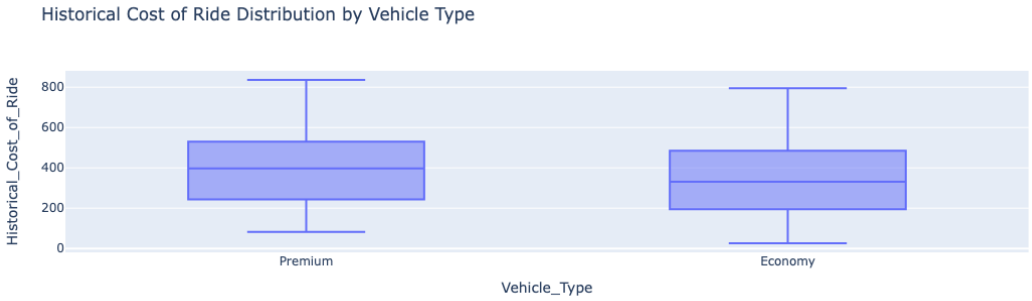
```
       Number_of_Riders  Number_of_Drivers  Number_of_Past_Rides  \
count       1000.000000        1000.000000           1000.000000
```

mean	60.372000	27.076000	50.031000
std	23.701506	19.068346	29.313774
min	20.000000	5.000000	0.000000
25%	40.000000	11.000000	25.000000
50%	60.000000	22.000000	51.000000
75%	81.000000	38.000000	75.000000
max	100.000000	89.000000	100.000000

	Average_Ratings	Expected_Ride_Duration	Historical_Cost_of_Ride
count	1000.000000	1000.00000	1000.000000
mean	4.257220	99.58800	372.502623
std	0.435781	49.16545	187.158756
min	3.500000	10.00000	25.993449
25%	3.870000	59.75000	221.365202
50%	4.270000	102.00000	362.019426
75%	4.632500	143.00000	510.497504
max	5.000000	180.00000	836.116419

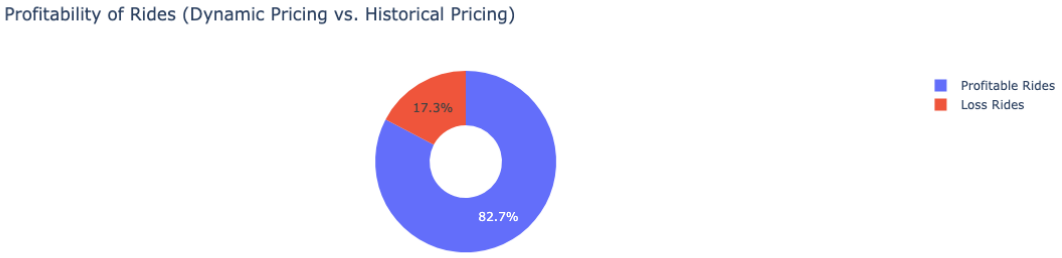
[3]:
```
fig = px.scatter(data, x='Expected_Ride_Duration',
                 y='Historical_Cost_of_Ride',
                 title='Expected Ride Duration vs. Historical Cost of Ride',
                 trendline='ols')
fig.show()
```

[4]:
```
fig = px.box(data, x='Vehicle_Type',
             y='Historical_Cost_of_Ride',
             title='Historical Cost of Ride Distribution by Vehicle Type')
fig.show()
```

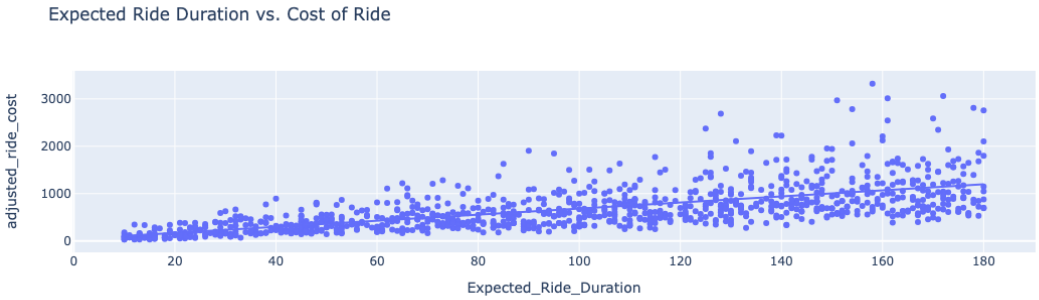


[5]:
```
import numpy as np

high_demand_percentile = 75
low_demand_percentile = 25

data['demand_multiplier'] = np.where(data['Number_of_Riders'] > np.
 ↪percentile(data['Number_of_Riders'], high_demand_percentile),
                                      data['Number_of_Riders'] / np.
 ↪percentile(data['Number_of_Riders'], high_demand_percentile),
                                      data['Number_of_Riders'] / np.
 ↪percentile(data['Number_of_Riders'], low_demand_percentile))

high_supply_percentile = 75
low_supply_percentile = 25

data['supply_multiplier'] = np.where(data['Number_of_Drivers'] > np.
 ↪percentile(data['Number_of_Drivers'], low_supply_percentile),
                                      np.percentile(data['Number_of_Drivers'],␣
 ↪high_supply_percentile) / data['Number_of_Drivers'],
                                      np.percentile(data['Number_of_Drivers'],␣
 ↪low_supply_percentile) / data['Number_of_Drivers'])

demand_threshold_high = 1.2  # Higher demand threshold
demand_threshold_low = 0.8  # Lower demand threshold
supply_threshold_high = 0.8  # Higher supply threshold
supply_threshold_low = 1.2  # Lower supply threshold

data['adjusted_ride_cost'] = data['Historical_Cost_of_Ride'] * (
    np.maximum(data['demand_multiplier'], demand_threshold_low) *
    np.maximum(data['supply_multiplier'], supply_threshold_high)
)
```

[6]:
```
data['profit_percentage'] = ((data['adjusted_ride_cost'] -␣
↪data['Historical_Cost_of_Ride']) / data['Historical_Cost_of_Ride']) * 100
profitable_rides = data[data['profit_percentage'] > 0]

loss_rides = data[data['profit_percentage'] < 0]


import plotly.graph_objects as go

profitable_count = len(profitable_rides)
loss_count = len(loss_rides)

labels = ['Profitable Rides', 'Loss Rides']
values = [profitable_count, loss_count]

fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=0.4)])
fig.update_layout(title='Profitability of Rides (Dynamic Pricing vs. Historical␣
↪Pricing)')
fig.show()
```



[7]:
```
fig = px.scatter(data,
                 x='Expected_Ride_Duration',
                 y='adjusted_ride_cost',
                 title='Expected Ride Duration vs. Cost of Ride',
                 trendline='ols')
fig.show()
```



[8]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def data_preprocessing_pipeline(data):
    numeric_features = data.select_dtypes(include=['float', 'int']).columns
    categorical_features = data.select_dtypes(include=['object']).columns

    data[numeric_features] = data[numeric_features].
 ↪fillna(data[numeric_features].mean())

    for feature in numeric_features:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)
        data[feature] = np.where((data[feature] < lower_bound) | (data[feature]␣
 ↪> upper_bound),
                                 data[feature].mean(), data[feature])

    data[categorical_features] = data[categorical_features].
 ↪fillna(data[categorical_features].mode().iloc[0])

    return data
```

[9]:
```
data["Vehicle_Type"] = data["Vehicle_Type"].map({"Premium": 1, "Economy": 0})
```

[10]:
```
from sklearn.model_selection import train_test_split
x = np.array(data[["Number_of_Riders", "Number_of_Drivers", "Vehicle_Type",␣
 ↪"Expected_Ride_Duration"]])
y = np.array(data[["adjusted_ride_cost"]])
```

```
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)

# Reshape y to 1D array
y_train = y_train.ravel()
y_test = y_test.ravel()

# Training a random forest regression model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train, y_train)
```

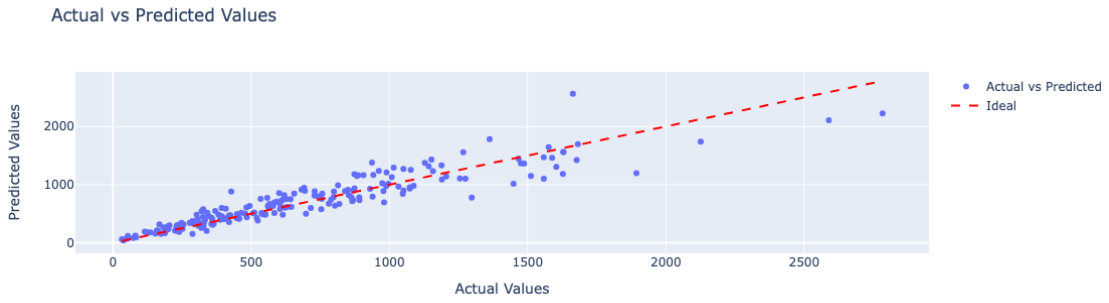
[10]:
```
RandomForestRegressor()
```

[11]:
```
def get_vehicle_type_numeric(vehicle_type):
    vehicle_type_mapping = {
        "Premium": 1,
        "Economy": 0
    }
    vehicle_type_numeric = vehicle_type_mapping.get(vehicle_type)
    return vehicle_type_numeric

def predict_price(number_of_riders, number_of_drivers, vehicle_type,␣
 ↪Expected_Ride_Duration):
    vehicle_type_numeric = get_vehicle_type_numeric(vehicle_type)
    if vehicle_type_numeric is None:
        raise ValueError("Invalid vehicle type")

    input_data = np.array([[number_of_riders, number_of_drivers,␣
 ↪vehicle_type_numeric, Expected_Ride_Duration]])
    predicted_price = model.predict(input_data)
    return predicted_price

user_number_of_riders = 50
user_number_of_drivers = 25
user_vehicle_type = "Economy"
Expected_Ride_Duration = 30
predicted_price = predict_price(user_number_of_riders, user_number_of_drivers,␣
 ↪user_vehicle_type, Expected_Ride_Duration)
print("Predicted price:", predicted_price)
```

```
Predicted price: [238.46040825]
```

[12]:
```
import plotly.graph_objects as go
```

```
y_pred = model.predict(x_test)

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=y_test.flatten(),
    y=y_pred,
    mode='markers',
    name='Actual vs Predicted'
))

fig.add_trace(go.Scatter(
    x=[min(y_test.flatten()), max(y_test.flatten())],
    y=[min(y_test.flatten()), max(y_test.flatten())],
    mode='lines',
    name='Ideal',
    line=dict(color='red', dash='dash')
))

fig.update_layout(
    title='Actual vs Predicted Values',
    xaxis_title='Actual Values',
    yaxis_title='Predicted Values',
    showlegend=True,
)

fig.show()
```



[ ]: