Pricing Analytics

By Prof.Vishal Chugh

What is Pricing Analytics ?

Pricing Analytics is the process of collecting and analyzing data related to market dynamics, customer behavior, competitor strategies, and historical sales to determine optimal prices that maximize revenue and profitability

Importance of Pricing Analytics

NPD

At what price the product should be launched in the market

Profitability

By identifying optimal price points, businesses can maximize revenue and profit margins without alienating customers with prices that are too high or losing sales by underpricing.

Strategic Decisions

Data-driven pricing provides critical insights into market dynamics, demand patterns, and the impact of price changes on revenue and volume, supporting more effective business strategies.

Steps in Pricing Analytics

Survey / Feedback

Perform regular surveys or feedbacks from time to time to understand what customer wants. This will help business to stay ahead in the competition

Elasticity of Demand

It measures how sensitive customer demand is when price changes

Competition Analysis

Stay updated with the changes competitors are bringing in the market and how it is impacting consumer behaviour

Price Elasticity of Demand

What is Price Elasticity of Demand (PED)?

- It measures how **sensitive customer demand** is when price changes.
- Formula:

$$PED = \frac{\%\Delta \text{Quantity Demanded}}{\%\Delta \text{Price}}$$

- **Interpretation:**
  - **PED > 1** → Elastic (customers are sensitive to price; lowering price boosts demand significantly).
  - **PED < 1** → Inelastic (customers are not very sensitive; price hikes don't reduce demand much).
  - **PED = 1** → Unit elastic (demand changes proportionally with price).

Loops in Python

By Prof.Vishal Chugh

Loops in Python

“ Loops in Python are control flow statements that allow a block of code to be executed repeatedly. Python primarily offers two types of loops: for loops and while loops.”

for Loops

1. `for` loops:

`for` loops are used to iterate over a sequence (like a list, tuple, string, or range) or other iterable objects. They execute a block of code for each item in the sequence.

Python

```python
# Example of a for loop iterating over a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Example of a for loop using range()
for i in range(5):  # Iterates from 0 to 4
    print(i)
```

while Loops

2. `while` loops:

`while` loops repeatedly execute a block of code as long as a given condition remains true. The condition is checked before each iteration.

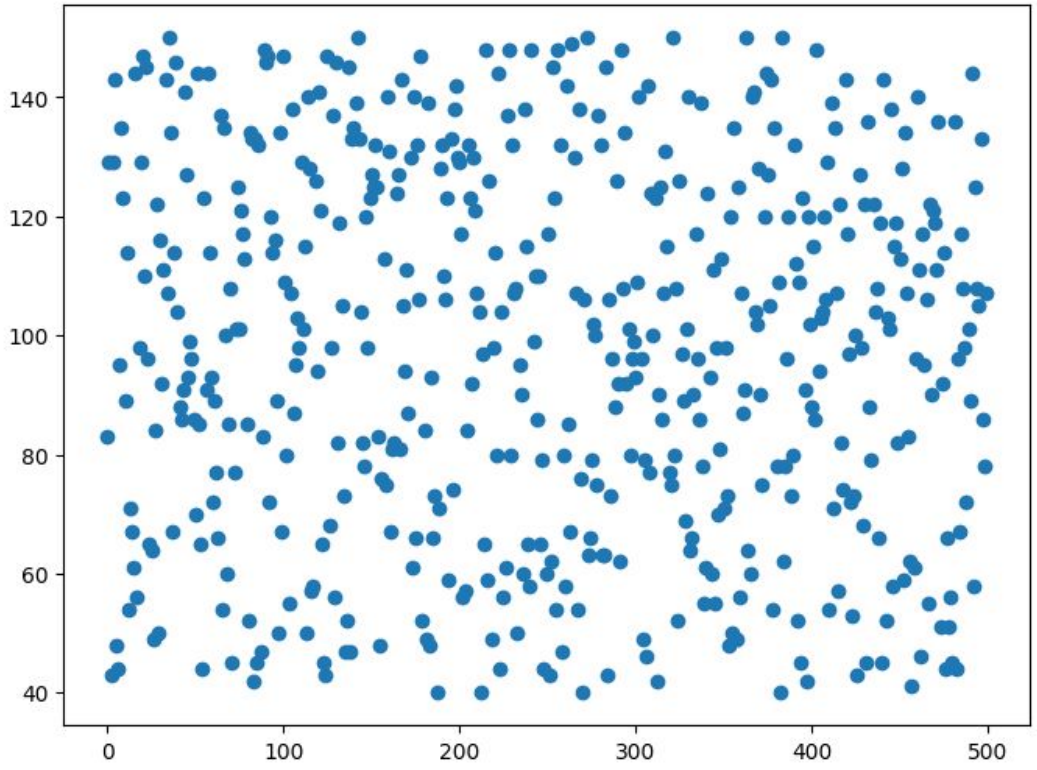
Python

```python
# Example of a while loop
count = 0
while count < 5:
    print(count)
    count += 1
```

Data Preprocessing

By Prof.Vishal Chugh

Lets See the Data in Scatter Plot of Variable that needs to be Predicted



What is Standardization ?

“It is the process which transforms independent features to have a mean of 0 and a Standard Deviation of 1”

How do we standardize data ?

$$z = \frac{x - \mu}{\sigma}$$

- x is the original data point,
- μ is the mean of the feature, and
- σ is the standard deviation of the feature.

When do we perform Standardization ?

1. Features have different units or scale (eg: g, kg, ml, lts)

2. The algorithm relies on distance metrics (Eg: One Data Point How Far From Other

3. Principal Component Analysis (It tries to find the directions of maximum variance of data)

When do not perform Standardization ?

1. Tree based algorithms

2. Interpretability

What is One Hot Encoding ?

“One-hot encoding is a technique used to convert categorical data into a binary format where each category is represented by a separate column with a 1 indicating its presence and 0s for all other categories.”

One Hot Encoding

Color
Red
Green
Blue

Color	Red	Green	Blue
Red	1	0	0
Green	0	1	0
Blue	0	0	1

Explore the Data in Python

By Prof.Vishal Chugh

Import Libraries

```
# import libraries
from cryptography.fernet import Fernet
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import copy
import warnings
```

After Decrypting the Data

```
# Loading the data
df = pd.read_csv('nutriboost_data.csv')
df.head()
```

	CustomerID	Age	Gender	Income_Level	Health_Conscious_Score	Current_Beverage_Preference
0	1	46	Male	Medium	8	Energy Drinks
1	2	32	Male	High	6	NaN
2	3	25	Female	Low	5	Water
3	4	38	Other	Low	6	Water
4	5	36	Other	Low	2	Juices

5 rows × 22 columns

Create Copy of the Original Data

```
# Create the copy
df_copy = df.copy(deep = True)
```

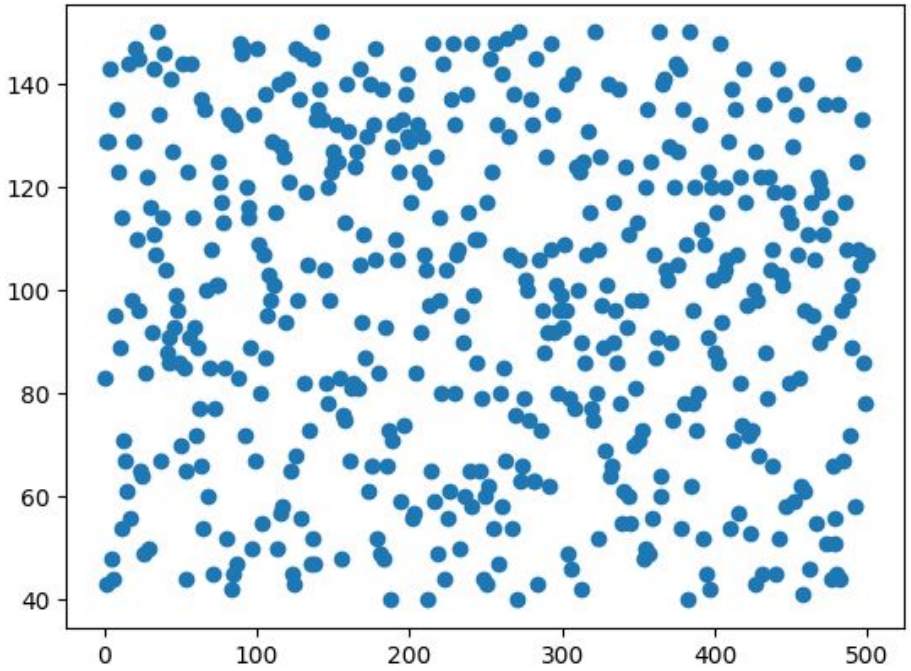
Replace the missing values with mode

```
# Replace the missing values with mode of the data
df_copy['Current_Beverage_Preference'] = df_copy['Current_Beverage_Preference'].fillna(df_copy['Current_Beverage_Preference'].mode()[0])
```

See the Scatter Plot

```
# See the Dependent Variable in Scatter Plot
plt.scatter(df_copy.index, df_copy['Willingness_To_Pay'])
```

```
<matplotlib.collections.PathCollection at 0x78986d2a05f0>
```



Define Independent and Dependent Variable

```
# Independent and Dependent Variables
x = df_copy[['Age', 'Income_Level', 'Health_Conscious_Score', 'Feature_Preference_Taste',
       'Feature_Preference_EnergyBoost', 'Feature_Preference_Packaging',
       'Feature_Preference_Price', 'Feature_Preference_Availability']]
y = df_copy['Willingness_To_Pay']
```

Create Separate Variables for Categorical and Numerical Features

```
# Separate categorical and numerical columns
categorical_col = ['Income_Level']
numerical_col = [col for col in x.columns if col not in categorical_col]
```

Data Preprocessing

```
# Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_col),
        ('cat', OneHotEncoder(), categorical_col)
    ]
  )
```

Building a Pipeline

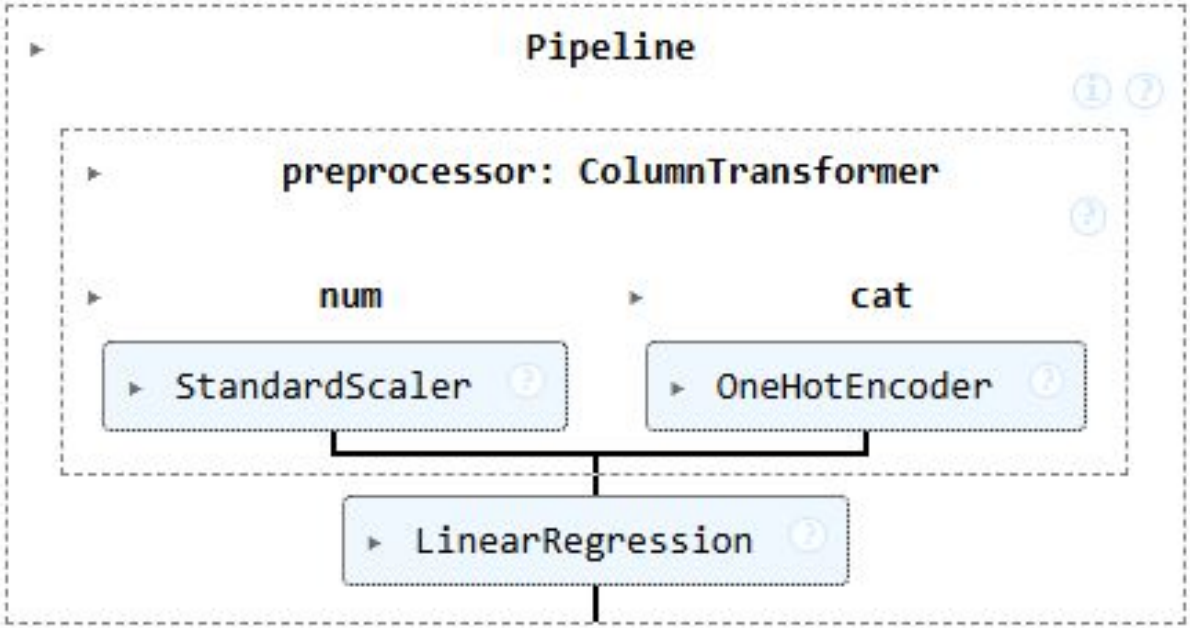
```
# Building a Pipeline
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('regressor', LinearRegression())])
```

Train Test Split

```
# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)
```

Model Fit

```
# Model fit
model.fit(x_train, y_train)
```



Predict Price

```
# Prediction
# ------------------------------
y_pred = model.predict(x_test)
predicted_price_regression = y_pred.mean()
print(f"\nPredicted Price (with Income_Level using OneHotEncoder): {predicted_price_regression:.2f}")
```

```
Predicted Price (with Income_Level using OneHotEncoder): 95.55
```

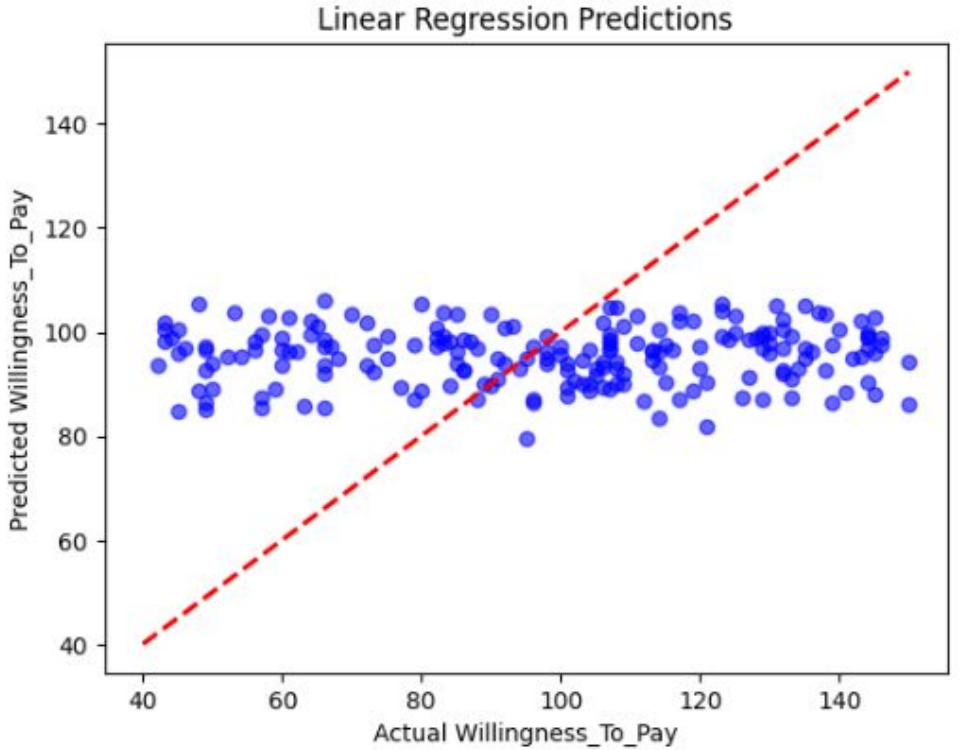
Average Price of Willingness to Pay

```
# Average Willingness to Price
mean_price = df_copy['Willingness_To_Pay'].mean()
round(mean_price,2)
```

```
np.float64(96.96)
```

Linear Regression Line

```
# Linear Regression Line
plt.scatter(y_test, y_pred, alpha=0.6, color="blue")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.xlabel("Actual Willingness_To_Pay")
plt.ylabel("Predicted Willingness_To_Pay")
plt.title("Linear Regression Predictions")
plt.show()
```



Demand

```
# Demand
demand =  df_copy.groupby('Willingness_To_Pay')['Purchase_Intention'].sum().sort_values(ascending = False).reset_index()
demand.head()
```

	Willingness_To_Pay	Purchase_Intention
0	148	7
1	44	6
2	67	6
3	80	6
4	98	6

Create Revenue Column

```
# Create Revenue column
demand['Revenue'] = demand['Willingness_To_Pay'] * demand['Purchase_Intention']
demand.head()
```

	Willingness_To_Pay	Purchase_Intention	Revenue
0	148	7	1036
1	44	6	264
2	67	6	402
3	80	6	480
4	98	6	588

Elasticity

```
# Elasticity
elasticity = []

for i in range(1, len(demand)):
  q1,q2 = demand.loc[i-1, 'Purchase_Intention'], demand.loc[i, 'Purchase_Intention']
  p1,p2 = demand.loc[i-1, 'Willingness_To_Pay'], demand.loc[i, 'Willingness_To_Pay']
  dq = (q2 - q1) / q1 if q1 != 0 else 0
  dp = (p2 - p1) / p1 if p1 != 0 else 0
  Ed = (dq / dp) if dp != 0 else 0

elasticity.append({'Price': p2, 'Elasticity': Ed})

elasticity_df = pd.DataFrame(elasticity)
```

Optimal Price

```
# Optimal price (max revenue)
optimal_price_elasticity = demand.loc[demand["Revenue"].idxmax(), "Willingness_To_Pay"]
optimal_price_elasticity
```

```
np.int64(148)
```

Lowest Pricing

```
# Price
prices = np.arange(min_wtp, max_wtp+1, 1)
revenue = []

for p in prices:
  buyers = df_copy[df_copy['Willingness_To_Pay'] >= p].shape[0]
  revenues = p * buyers
  revenue.append(revenues)

lowest_price = prices[np.argmax(revenue)]

print("To sustain in the competition we can keep the pricing at:", lowest_price)
```

```
To sustain in the competition we can keep the pricing at: 78
```

Conclusion

As we are adopting Market Penetration Strategy to target all Income Levels we will go ahead with Price Predicted by Linear Regression Model only if Competitor pricing is not lower than that Otherwise we will consider Lowest Pricing to Beat Competition