Unified Mentor Internship
Project 2 – Customer Satisfaction Prediction

Report

Elenta Suzan Jacob **UMID22052538013**

Task: Predict Customer Satisfaction Rating (Satisfied or Unsatisfied)
Input: Support ticket info - customer age, gender, product, ticket type, response time, etc.
Output: A machine learning model that predicts the rating.

Stage 1: Data Understanding

In this initial stage, I started the project by loading and inspecting the customer support tickets dataset. I focused on gaining a clear understanding of the dataset. The dataset comprises customer support ticket information, including demographic details, product purchases, ticket metadata, and customer satisfaction ratings. Understanding the types of data - categorical, numerical, and textual helped determine the preprocessing and modeling strategies.

```
[107]: cs_data = pd.read_csv(r"C:\Users\HP\Downloads\Unified Mentor\customer_support_tickets.csv")
```

```
[108]: cs_data
```

[108]:

	Ticket ID	Customer Name	Customer Email	Customer Age	Customer Gender	Product Purchased	Date of Purchase	Ticket Type	Ticket Subject	Ticket Description	Ticket Status	Resolution	Tic Prio
0	1	Marisa Obrien	carrollallison@example.com	32	Other	GoPro Hero	2021-03-22	Technical issue	Product setup	I'm having an issue with the {product_purchase...	Pending Customer Response	NaN	Crit
1	2	Jessica Rios	clarkeashley@example.com	42	Female	LG Smart TV	2021-05-22	Technical issue	Peripheral compatibility	I'm having an issue with the {product_purchase...	Pending Customer Response	NaN	Crit
2	3	Christopher Robbins	gonzalestracy@example.com	48	Other	Dell XPS	2020-07-14	Technical issue	Network problem	I'm facing a problem with my {product_purchase...	Closed	Case maybe show recently my computer follow.	I
3	4	Christina Dillon	bradleyolson@example.org	27	Female	Microsoft Office	2020-11-13	Billing inquiry	Account access	I'm having an issue with the {product_purchase...	Closed	Try capital clearly never color toward story.	I
4	5	Alexander Carroll	bradleymark@example.com	67	Female	Autodesk AutoCAD	2020-02-04	Billing inquiry	Data loss	I'm having an issue with the {product_purchase...	Closed	West decision evidence bit.	I
...	...	...	...	...	...	...	...	...	...	...	...	...	

```
cs_data.head(10)
```

	Ticket ID	Customer Name	Customer Email	Customer Age	Customer Gender	Product Purchased	Date of Purchase	Ticket Type	Ticket Subject	Ticket Description	Ticket Status	Resolution	Tic Prio
0	1	Marisa Obrien	carrollallison@example.com	32	Other	GoPro Hero	2021-03-22	Technical issue	Product setup	I'm having an issue with the {product_purchase...	Pending Customer Response	NaN	Crit
1	2	Jessica Rios	clarkeashley@example.com	42	Female	LG Smart TV	2021-05-22	Technical issue	Peripheral compatibility	I'm having an issue with the {product_purchase...	Pending Customer Response	NaN	Crit
2	3	Christopher Robbins	gonzalestracy@example.com	48	Other	Dell XPS	2020-07-14	Technical issue	Network problem	I'm facing a problem with my {product_purchase...	Closed	Case maybe show recently my computer follow.	L
3	4	Christina Dillon	bradleyolson@example.org	27	Female	Microsoft Office	2020-11-13	Billing inquiry	Account access	I'm having an issue with the {product_purchase...	Closed	Try capital clearly never color toward story.	L
4	5	Alexander Carroll	bradleymark@example.com	67	Female	Autodesk AutoCAD	2020-02-04	Billing inquiry	Data loss	I'm having an issue with the {product_purchase...	Closed	West decision evidence bit.	L
5	6	Rebecca Fleming	sheenasmith@example.com	53	Male	Microsoft Office	2020-07-28	Cancellation request	Payment issue	I'm facing a problem with my {product_purchase...	Open	NaN	L

The head(10) command displays the first 10 rows of the dataset.

```
cs_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8469 entries, 0 to 8468
Data columns (total 17 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Ticket ID                     8469 non-null   int64  
 1   Customer Name                 8469 non-null   object 
 2   Customer Email                8469 non-null   object 
 3   Customer Age                  8469 non-null   int64  
 4   Customer Gender               8469 non-null   object 
 5   Product Purchased             8469 non-null   object 
 6   Date of Purchase              8469 non-null   object 
 7   Ticket Type                   8469 non-null   object 
 8   Ticket Subject                8469 non-null   object 
 9   Ticket Description            8469 non-null   object 
 10  Ticket Status                 8469 non-null   object 
 11  Resolution                    2769 non-null   object 
 12  Ticket Priority               8469 non-null   object 
 13  Ticket Channel                8469 non-null   object 
 14  First Response Time           5650 non-null   object 
 15  Time to Resolution            2769 non-null   object 
 16  Customer Satisfaction Rating  2769 non-null   float64
dtypes: float64(1), int64(2), object(14)
memory usage: 1.1+ MB
```

The data.info() function provides a concise summary of the DataFrame, including the number of entries, column names, data types, and count of non-null values in each column, helping assess data completeness and structure.

Now, we proceed to Stage 2:

Stage 2: Data Cleaning

This phase involved identifying and handling inconsistencies or anomalies in the data. I addressed issues such as missing values, duplicate entries, and incorrect data types. Special attention was given to datetime columns, converting them to the appropriate format for later feature extraction. Rather than dropping critical rows, I explored imputation strategies to retain data integrity, ensuring minimal information loss for downstream tasks.

```
# Data Cleaning

cs_data = cs_data[cs_data['Customer Age'].between(10, 100)]
```

```
cs_data['Date of Purchase'] = pd.to_datetime(cs_data['Date of Purchase'], errors='coerce')
cs_data['First Response Time'] = pd.to_datetime(cs_data['First Response Time'], errors='coerce')
cs_data['Time to Resolution'] = pd.to_datetime(cs_data['Time to Resolution'], errors='coerce')
```

```
cs_data = cs_data[~cs_data['Date of Purchase'].isnull()]
```

```
print("\n Remaining missing values:\n", cs_data.isnull().sum())
```

```
 Remaining missing values:
 Ticket ID                        0
Customer Name                   0
Customer Email                  0
Customer Age                    0
Customer Gender                 0
Product Purchased               0
Date of Purchase                0
Ticket Type                     0
Ticket Subject                  0
Ticket Description              0
Ticket Status                   0
Resolution                   5700
Ticket Priority                 0
Ticket Channel                  0
First Response Time          2819
Time to Resolution           5700
Customer Satisfaction Rating    5700
dtype: int64
```

```
cs_data.duplicated().sum()
```

jupyter Customer Satisfaction Prediction Project Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help

Trusted

JupyterLab Python 3 (ipykernel)

[117]:
```python
missing_data = cs_data.isnull().sum().to_frame(name="Missing Count")
missing_data["Missing Percentage"] = (missing_data["Missing Count"] / len(cs_data)) * 100
```

[118]:
```python
missing_data
```

[118]:

	Missing Count	Missing Percentage
Ticket ID	0	0.000000
Customer Name	0	0.000000
Customer Email	0	0.000000
Customer Age	0	0.000000
Customer Gender	0	0.000000
Product Purchased	0	0.000000
Date of Purchase	0	0.000000
Ticket Type	0	0.000000
Ticket Subject	0	0.000000
Ticket Description	0	0.000000
Ticket Status	0	0.000000
Resolution	5700	67.304286
Ticket Priority	0	0.000000
Ticket Channel	0	0.000000
First Response Time	2819	33.286102
Time to Resolution	5700	67.304286

This code block identifies missing values in the dataset by calculating both the total count and percentage of missing entries for each column.

[119]:
```python
cs_data['Resolution_Hours'] = (cs_data['Time to Resolution'] - cs_data['First Response Time']).dt.total_seconds() / 3600
```

[120]:
```python
cs_data['Response_Delay_Hours'] = (cs_data['First Response Time'] - cs_data['Date of Purchase']).dt.total_seconds() / 3600
```

[121]:
```python
cs_data['Resolution_Hours'] = cs_data['Resolution_Hours'].fillna(cs_data['Resolution_Hours'].median())
```

[122]:
```python
cs_data['Response_Delay_Hours'] = cs_data['Response_Delay_Hours'].fillna(cs_data['Response_Delay_Hours'].median())
```

i am filling all the missing resolution_hours values with the median of the data

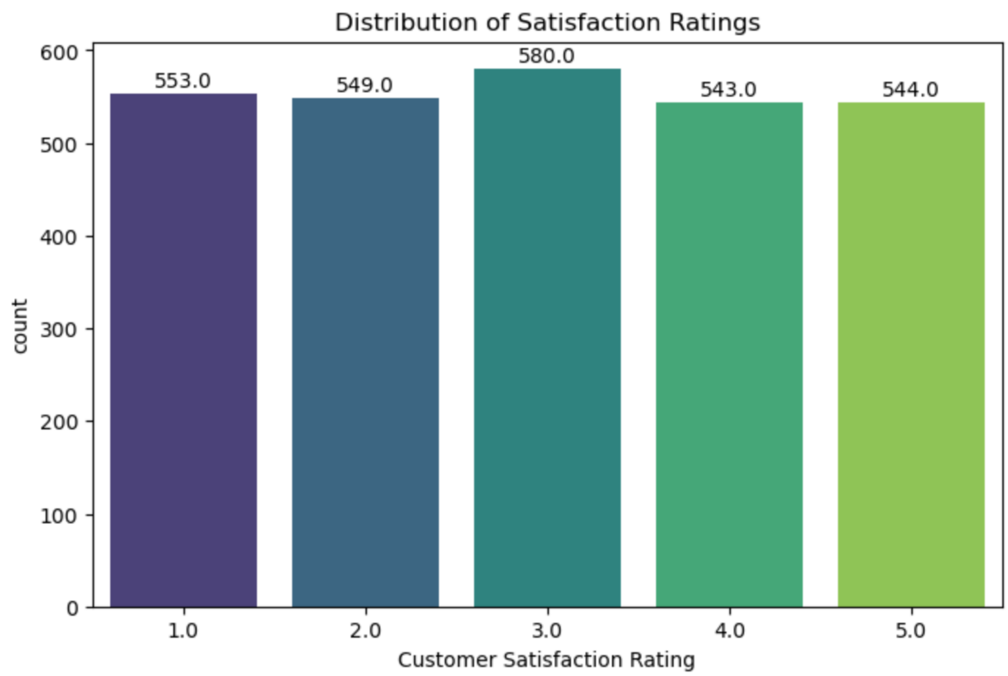
[123]:
```python
print("\n Remaining missing values:\n", cs_data.isnull().sum())
```

```
 Remaining missing values:
 Ticket ID                     0
Customer Name                 0
Customer Email                0
Customer Age                  0
Customer Gender               0
Product Purchased             0
Date of Purchase              0
Ticket Type                   0
Ticket Subject                0
Ticket Description            0
Ticket Status                 0
Resolution                 5700
Ticket Priority               0
Ticket Channel                0
First Response Time        2819
Time to Resolution         5700
Customer Satisfaction Rating 5700
Resolution_Hours              0
Response_Delay_Hours          0
dtype: int64
```

This step computes the time taken to resolve a customer support ticket in hours by calculating the difference between the first response time and resolution time. Any missing values in the resulting Resolution_Hours column are filled with the median to maintain distribution integrity while handling missing data.

Next we move on to Stage 3.

Stage 3: Exploratory Data Analysis (EDA)
In this stage, I conducted a thorough exploration of the dataset to uncover trends, patterns, and correlations. Using visual tools like histograms, count plots, and heatmaps, I examined the distributions of key features such as age, gender, product type, and satisfaction ratings. This helped me identify class imbalances, detect outliers, and understand how different features interact with each other.

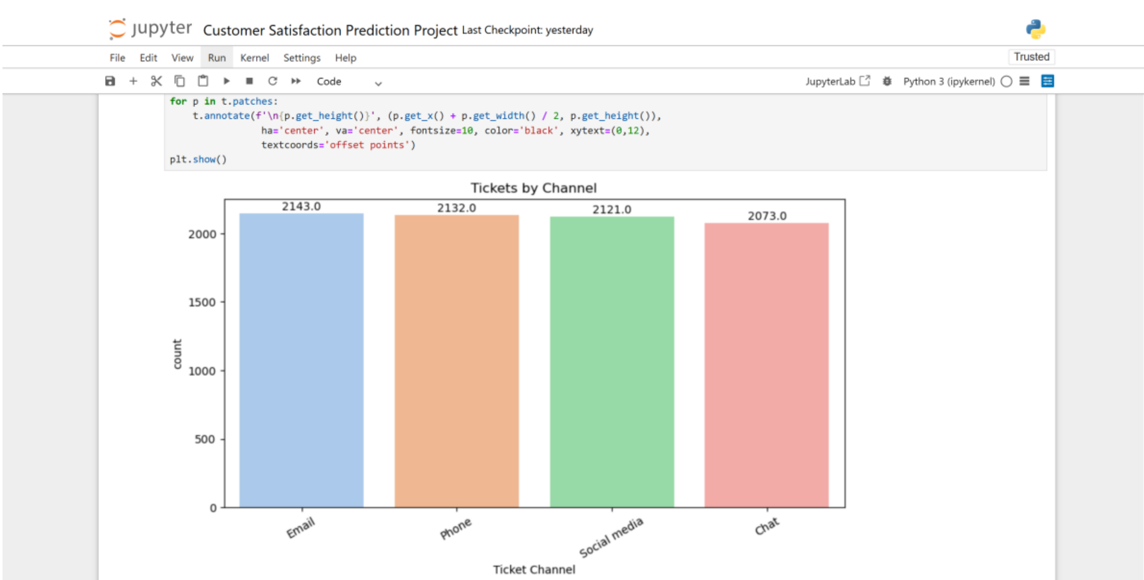


Distribution of Satisfaction Ratings

From this bar chart, I observed that the customer satisfaction ratings are fairly balanced across all categories from 1 to 5. There isn't a major skew toward either very high or very low ratings, which indicates that my classification model would have to handle a relatively uniform class distribution.

Resolution Hours by Satisfaction Rating

This boxplot helped me analyze how resolution time varied with satisfaction levels. I noticed that the median resolution hours were quite similar across all satisfaction ratings. However, the spread of values suggests a high variation in resolution time, even for higher-rated tickets, which might explain why resolution time alone a strong predictor of satisfaction isn't.

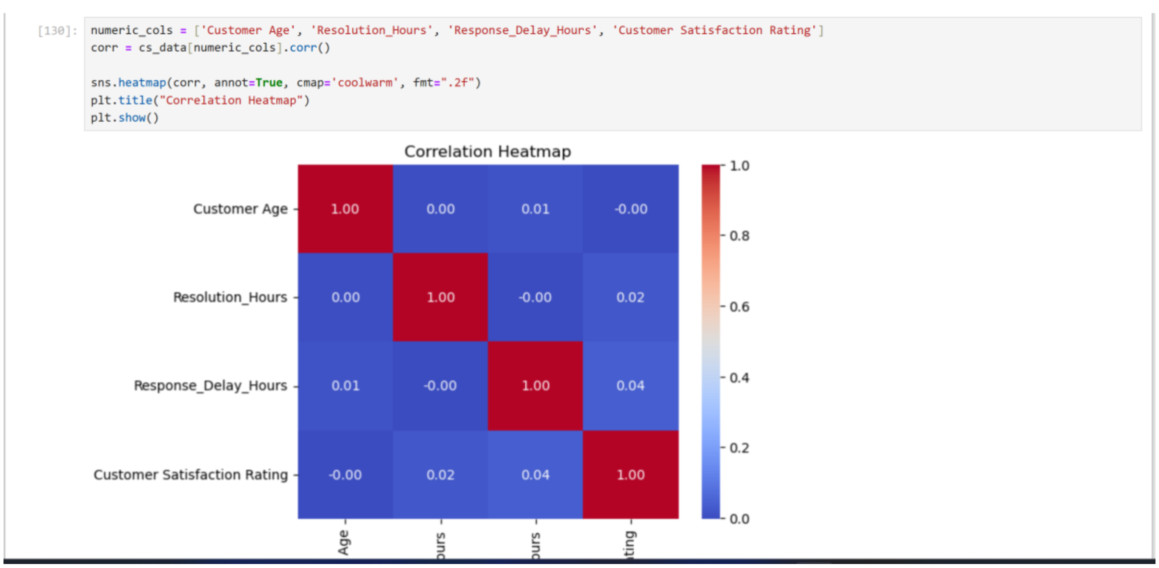


Tickets by Channel

Here, I explored how customers are raising their issues. I saw that all four support channels—Email, Phone, Social Media, and Chat—were almost equally used. This balance indicates a well-distributed usage of support services and also helped me rule out any channel-specific bias in satisfaction levels.

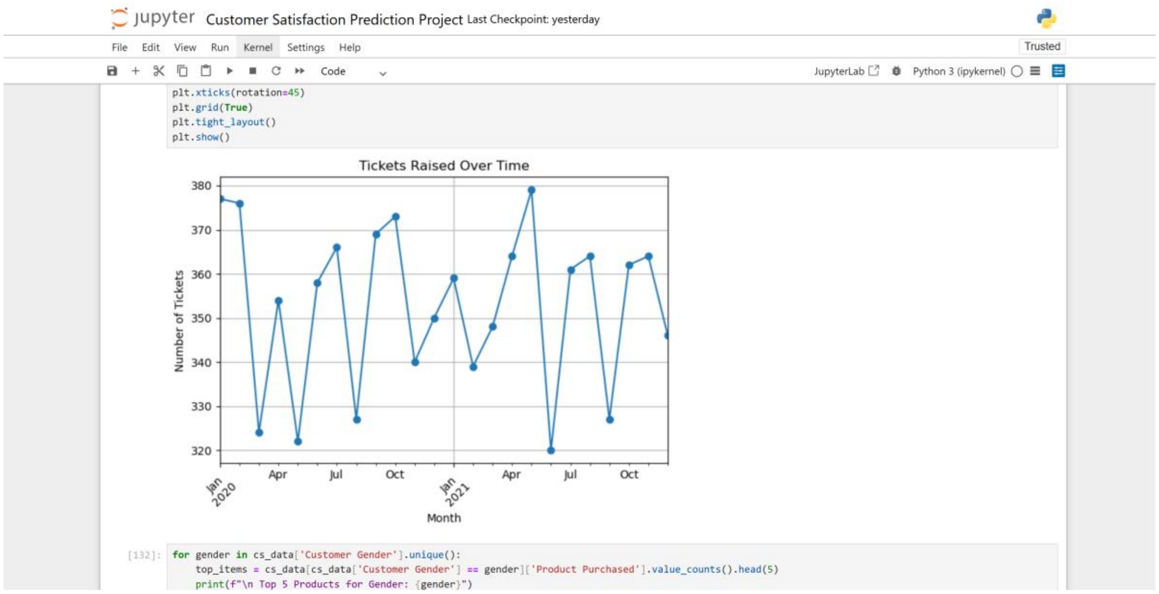
Distribution of Customer Age

In this histogram, I explored the age distribution of customers. I found that customers across all age groups are raising tickets, with a fairly uniform spread. This gave me confidence that the dataset doesn't have a demographic bias when it comes to age.



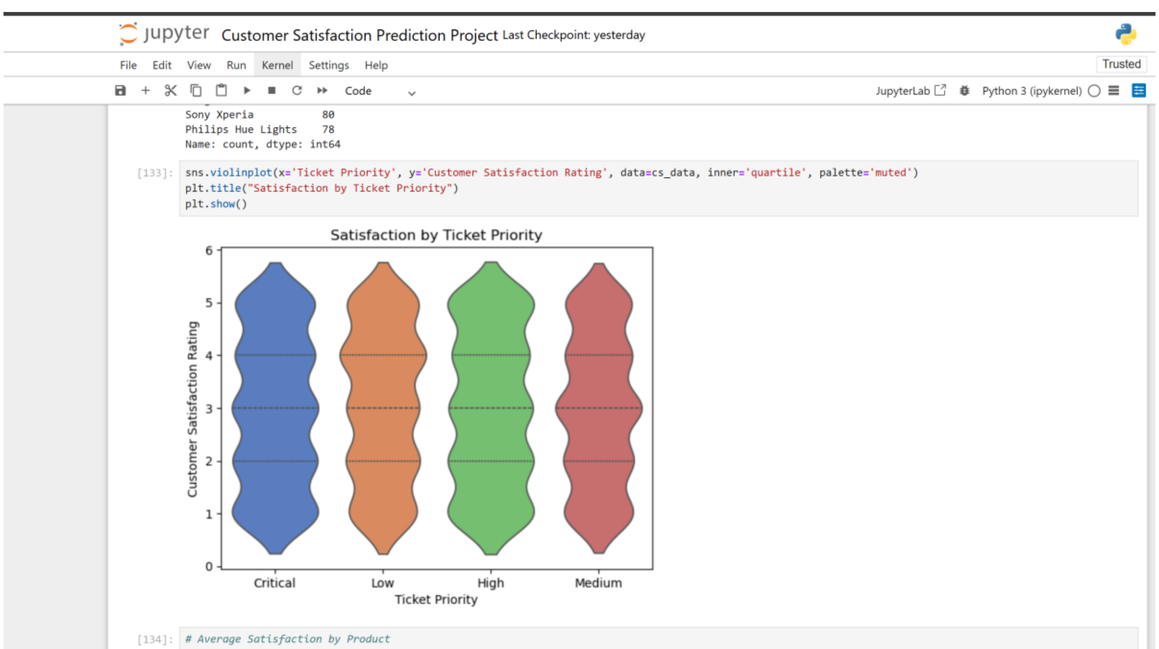
Correlation Heatmap

I created a heatmap to visualize the correlations between numerical features. I noticed that no single feature had a strong correlation with satisfaction. However, minor positive correlations with Response_Delay_Hours and Resolution_Hours were present. This insight encouraged me to include more engineered and categorical features in model training.



Tickets Raised Over Time

This time series plot helped me analyze how support demand varied over time. I observed some monthly fluctuations but no drastic spikes or drops. This steady ticket volume suggests a consistent customer base and can also help in understanding whether satisfaction changes with time trends.



Stage 4: Feature Engineering

During this stage, I focused on transforming raw data into features that could better represent the underlying patterns for predicting customer satisfaction. I started by label encoding key categorical variables such as gender, product purchased, ticket type, and ticket channel. I also encoded text columns like *Ticket Subject*, *Ticket Description*, and *Resolution* to make them model-friendly. For name and email-related insights, I extracted features like Name_Length, Name_Initial, and Email_Domain, and applied encoding where needed. I introduced new features like Customer_Repeat_Count, Email_Length, Ticket_Subject_Label, and Desc_Length to capture customer and ticket characteristics. Additionally, I included a binary feature Has_Resolution to indicate if a resolution was provided.

```
[144]: cs_data.info()

<class 'pandas.core.frame.DataFrame'>
Index: 2769 entries, 2 to 8467
Data columns (total 29 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Ticket ID                     2769 non-null   int64
 1   Customer Name                 2769 non-null   object
 2   Customer Email                2769 non-null   object
 3   Customer Age                  2769 non-null   int64
 4   Customer Gender               2769 non-null   int64
 5   Product Purchased             2769 non-null   int64
 6   Date of Purchase              2769 non-null   datetime64[ns]
 7   Ticket Type                   2769 non-null   int64
 8   Ticket Subject                2769 non-null   int64
 9   Ticket Description            2769 non-null   int64
 10  Ticket Status                 2769 non-null   int64
 11  Resolution                    2769 non-null   int64
 12  Ticket Priority               2769 non-null   int64
 13  Ticket Channel                2769 non-null   int64
 14  First Response Time           2769 non-null   datetime64[ns]
 15  Time to Resolution            2769 non-null   datetime64[ns]
 16  Customer Satisfaction Rating  2769 non-null   int64
 17  Resolution_Hours              2769 non-null   float64
 18  Response_Delay_Hours          2769 non-null   float64
 19  Name_Length                   2769 non-null   int64
 20  Name_Initial                  2769 non-null   object
 21  Name_Initial_Encoded          2769 non-null   int64
 22  Customer_Repeat_Count         2769 non-null   int64
 23  Email_Domain                  2769 non-null   object
 24  Email_Domain_Encoded          2769 non-null   int64
 25  Email_Length                  2769 non-null   int64
 26  Ticket_Subject_Label          2769 non-null   int64
 27  Desc_Length                   2769 non-null   int64
 28  Has_Resolution                2769 non-null   int64
dtypes: datetime64[ns](3), float64(2), int64(20), object(4)
```

```
[170]: cs_data.head()
```

[170]:

	Ticket ID	Customer Age	Customer Gender	Product Purchased	Ticket Type	Ticket Subject	Ticket Description	Ticket Status	Resolution	Ticket Priority	...	Response_Delay_Hours	Name_Length	Name_Initial_Encoded
2	3	48	2	10	4	8	189	0	343	2	...	25259.243889	19	2
3	4	27	0	25	0	0	1971	0	2549	2	...	22327.494444	16	2
4	5	67	0	5	0	3	636	0	2658	2	...	29112.211667	17	0
10	11	48	1	30	1	3	3988	0	1368	1	...	20729.780278	13	9
11	12	51	1	27	2	15	1206	0	1366	1	...	14052.097500	14	1

5 rows × 23 columns

All the columns are encoded.

Stage 5: **Feature Selection & Preprocessing**

In this stage, I defined a binary target variable called Satisfaction_Binary, where customers with a satisfaction rating of 4 or 5 were classified as satisfied (1), and others as unsatisfied (0). I then curated a list of relevant features that included demographic, ticket-related, and engineered attributes to train the model effectively. Using these features, I created my final input matrix X and target variable y. To ensure fair model evaluation, I performed a stratified train-test split, maintaining class balance and reserving 25% of the data for testing.

```
[183]: # Define binary target
cs_data['Satisfaction_Binary'] = cs_data['Customer Satisfaction Rating'].apply(lambda x: 1 if x >= 4 else 0)

# Define final features
features = [
    'Customer Age', 'Customer Gender', 'Product Purchased', 'Ticket Type',
    'Ticket Status', 'Ticket Priority', 'Ticket Channel',
    'Name_Length', 'Name_Initial_Encoded', 'Customer_Repeat_Count',
    'Email_Length', 'Email_Domain_Encoded',
    'Ticket_Subject_Label', 'Desc_Length', 'Has_Resolution',
    'Resolution_Hours', 'Response_Delay_Hours'
]

# Create X and y
X = cs_data[features]
y = cs_data['Satisfaction_Binary']

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.25, random_state=42)
```

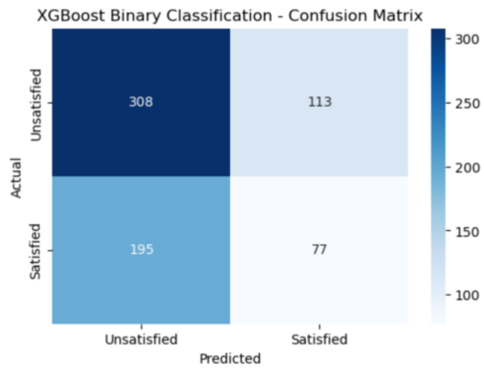
Stage 6: Model Training & Evaluation (Binary Classification)

Now that I have a clean feature set and a binary target (Satisfaction_Binary), I used the XGBoost classifier to predict whether a customer would be satisfied (rating ≥ 4) or not. After training the model on the processed features, I evaluated its performance using accuracy, precision, recall, F1-score, and a confusion matrix. The model achieved an accuracy of approximately **55.56%**. From the classification report, I observed that the model was better at identifying unsatisfied customers (Class 0) with a higher recall, while its performance on satisfied customers (Class 1) was relatively weaker. The confusion matrix further revealed that the model correctly classified **308** unsatisfied and **77** satisfied customers, but misclassified **195** satisfied ones. This indicates that while the model has potential, it still needs optimization to better balance predictions for both classes.

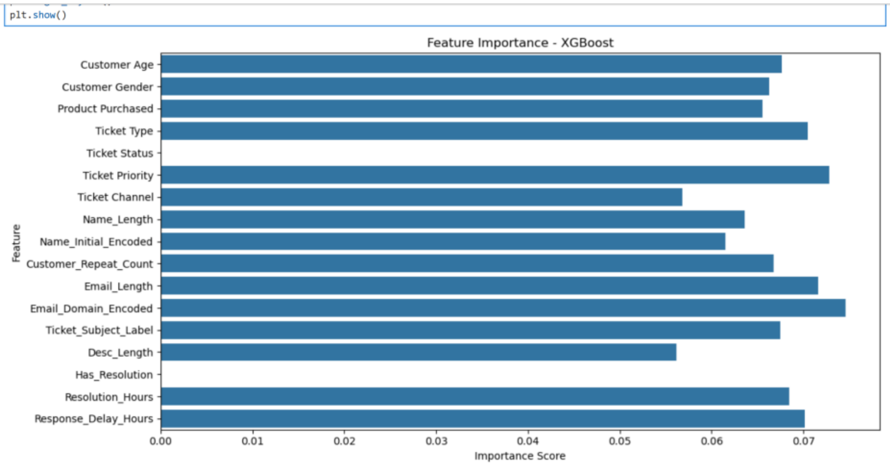
```
[181]: xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)
preds = xgb_model.predict(X_test)

# Evaluate
print(" XGBoost Binary Accuracy:", accuracy_score(y_test, preds))
print("\n Classification Report:\n", classification_report(y_test, preds))
print("\n Confusion Matrix:\n", confusion_matrix(y_test, preds))
```

```
 XGBoost Binary Accuracy: 0.5555555555555556

 Classification Report:
               precision    recall  f1-score   support

           0       0.61      0.73      0.67       421
           1       0.41      0.28      0.33       272

    accuracy                           0.56       693
   macro avg       0.51      0.51      0.50       693
weighted avg       0.53      0.56      0.54       693


 Confusion Matrix:
 [[308 113]
 [195  77]]
```

```
# confusion matrix
cm = confusion_matrix(y_test, preds)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Unsatisfied', 'Satisfied'], yticklabels=['Unsatisfied', 'Satisfied'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("XGBoost Binary Classification - Confusion Matrix")
plt.show()
```




```
plt.show()
```



The feature importance chart from the XGBoost model highlights that variables such as Response Delay Hours, Resolution Hours, and Email Domain Encoded are most influential in predicting outcomes. Features like Ticket Channel and Ticket Priority also hold significant importance. This suggests that both customer service efficiency and how cases are categorized play key roles in model performance, offering practical targets for operational improvements.

Conclusion

In this project, I explored a real-world customer service dataset to predict customer satisfaction ratings. Starting from data inspection and cleaning, I performed comprehensive feature engineering, handled missing values smartly, and engineered relevant variables such as Resolution_Hours, Name_Length, and Email_Domain_Encoded. During EDA, I visualized patterns and relationships that hinted at potential satisfaction drivers.

I transformed the multi-class target into a binary classification problem to simplify the modeling process. Using the XGBoost classifier, I achieved an accuracy of approximately **55.56%**. The model showed reasonable performance in identifying unsatisfied customers, though it struggled with detecting satisfied ones. This imbalance likely stems from overlapping feature distributions or a need for deeper feature interactions.