Customer_Satisfaction_Project

May 21, 2025

[1]:
```
import pandas as pd  # Step 1: Import required libraries
import numpy as np

df = pd.read_csv("customer_support_tickets.csv")  # Load the dataset

df.head()  # Display the first few rows
```

[1]:

	Ticket ID	Customer Name	Customer Email	Customer Age	Customer Gender	Product Purchased	Date of Purchase	Ticket Type	Ticket Subject	Ticket Description	Ticket Status	Resolution
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
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
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…		
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…		
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…		

```
2                    Closed   Case maybe show recently my computer follow.
3                    Closed  Try capital clearly never color toward story.
4                    Closed                    West decision evidence bit.

  Ticket Priority Ticket Channel  First Response Time   Time to Resolution  \
0        Critical   Social media  2023-06-01 12:15:36                  NaN
1        Critical           Chat  2023-06-01 16:45:38                  NaN
2             Low   Social media  2023-06-01 11:14:38  2023-06-01 18:05:38
3             Low   Social media  2023-06-01 07:29:40  2023-06-01 01:57:40
4             Low          Email  2023-06-01 00:12:42  2023-06-01 19:53:42

   Customer Satisfaction Rating
0                           NaN
1                           NaN
2                           3.0
3                           3.0
4                           1.0
```

[2]:
```
df.drop(columns=[                                                    # Drop␣
 ↪unnecessary columns
    'Ticket ID', 'Customer Name', 'Customer Email',
    'Ticket Subject', 'Ticket Description', 'Resolution'
], inplace=True)

df.shape, df.columns  # Check shape and column names
```

[2]:
```
((8469, 11),
 Index(['Customer Age', 'Customer Gender', 'Product Purchased',
        'Date of Purchase', 'Ticket Type', 'Ticket Status', 'Ticket Priority',
        'Ticket Channel', 'First Response Time', 'Time to Resolution',
        'Customer Satisfaction Rating'],
       dtype='object'))
```

[3]:
```
df.isnull().sum()
```

[3]:
```
Customer Age                       0
Customer Gender                    0
Product Purchased                  0
Date of Purchase                   0
Ticket Type                        0
Ticket Status                      0
Ticket Priority                    0
Ticket Channel                     0
First Response Time             2819
Time to Resolution              5700
Customer Satisfaction Rating    5700
dtype: int64
```

[4]:
```
df = df.dropna(subset=['Customer Satisfaction Rating'])  # Dropingg rows with␣
↪missing target Customer Satisfaction Rating
df = df.dropna(subset=['Time to Resolution'])  # Droping rows where resolution␣
↪time is missing
df.isnull().sum()
df.shape
```

[4]: (2769, 11)

[5]:
```
print(df.columns.tolist())
```

```
['Customer Age', 'Customer Gender', 'Product Purchased', 'Date of Purchase',
'Ticket Type', 'Ticket Status', 'Ticket Priority', 'Ticket Channel', 'First
Response Time', 'Time to Resolution', 'Customer Satisfaction Rating']
```

[6]:
```
df = df.drop(columns=['First Response Time'], errors='ignore')  #got error␣
↪while drop of First Response Time
```

[7]:
```
print(df.columns.tolist())
```

```
['Customer Age', 'Customer Gender', 'Product Purchased', 'Date of Purchase',
'Ticket Type', 'Ticket Status', 'Ticket Priority', 'Ticket Channel', 'Time to
Resolution', 'Customer Satisfaction Rating']
```

[8]:
```
categorical_cols = ['Customer Gender', 'Product Purchased', 'Ticket Type',  #␣
↪Listing of all categorical columns to encode
                    'Ticket Status', 'Ticket Priority', 'Ticket Channel']

df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)  #␣
↪drop the first column to avoid multiple

print(df_encoded.shape)  # Show new shape
df_encoded.head()
```

(2769, 57)

[8]:

	Customer Age	Date of Purchase	Time to Resolution	\
2	48	2020-07-14	2023-06-01 18:05:38	
3	27	2020-11-13	2023-06-01 01:57:40	
4	67	2020-02-04	2023-06-01 19:53:42	
10	48	2021-01-19	2023-05-31 23:51:49	
11	51	2021-10-24	2023-06-01 09:27:51	

	Customer Satisfaction Rating	Customer Gender_Male	Customer Gender_Other	\
2	3.0	False	True	
3	3.0	False	False	
4	1.0	False	False	
10	1.0	True	False	
11	1.0	True	False	

```
    Product Purchased_Amazon Echo  Product Purchased_Amazon Kindle  \
2                           False                            False
3                           False                            False
4                           False                            False
10                          False                            False
11                          False                            False

    Product Purchased_Apple AirPods  Product Purchased_Asus ROG  …  \
2                             False                       False  …
3                             False                       False  …
4                             False                       False  …
10                            False                       False  …
11                            False                       False  …

    Ticket Type_Cancellation request  Ticket Type_Product inquiry  \
2                              False                        False
3                              False                        False
4                              False                        False
10                              True                        False
11                             False                         True

    Ticket Type_Refund request  Ticket Type_Technical issue  \
2                        False                         True
3                        False                        False
4                        False                        False
10                       False                        False
11                       False                        False

    Ticket Priority_High  Ticket Priority_Low  Ticket Priority_Medium  \
2                  False                True                   False
3                  False                True                   False
4                  False                True                   False
10                  True               False                   False
11                  True               False                   False

    Ticket Channel_Email  Ticket Channel_Phone  Ticket Channel_Social media
2                  False                 False                        True
3                  False                 False                        True
4                   True                 False                       False
10                 False                  True                       False
11                 False                 False                       False

[5 rows x 57 columns]
```

[9]:
```
from sklearn.model_selection import train_test_split
```

```
X = df_encoded.drop(columns=['Customer Satisfaction Rating'])  # Features
y = df_encoded['Customer Satisfaction Rating']                 # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[10]:
```
df['Date of Purchase'] = pd.to_datetime(df['Date of Purchase'],␣
 ↪errors='coerce')  # Convert Date of Purchase to datetime and extract year,␣
 ↪month, day
df['Purchase_Year'] = df['Date of Purchase'].dt.year
df['Purchase_Month'] = df['Date of Purchase'].dt.month
df['Purchase_Day'] = df['Date of Purchase'].dt.day
df = df.drop(columns=['Date of Purchase'])  # Drop original column after␣
 ↪extraction

df['Time to Resolution'] = pd.to_datetime(df['Time to Resolution'],␣
 ↪errors='coerce')  # Convert Time to Resolution to datetime and convert to␣
 ↪total seconds from a reference point
df['Resolution_Seconds'] = (df['Time to Resolution'] - df['Time to Resolution'].
 ↪min()).dt.total_seconds()
df = df.drop(columns=['Time to Resolution'])  # Drop original column after␣
 ↪conversion

df[['Purchase_Year', 'Purchase_Month', 'Purchase_Day', 'Resolution_Seconds']].
 ↪head()  # Check the results
```

[10]:

	Purchase_Year	Purchase_Month	Purchase_Day	Resolution_Seconds
2	2020	7	14	72728.0
3	2020	11	13	14650.0
4	2020	2	4	79212.0
10	2021	1	19	7099.0
11	2021	10	24	41661.0

[11]:
```
print(df_encoded.select_dtypes(include=['datetime64[ns]']).columns)
```

```
Index([], dtype='object')
```

[12]:
```
print(X_train.dtypes)
```

```
Customer Age                                  int64
Date of Purchase                             object
Time to Resolution                           object
Customer Gender_Male                           bool
Customer Gender_Other                          bool
Product Purchased_Amazon Echo                  bool
Product Purchased_Amazon Kindle                bool
Product Purchased_Apple AirPods                bool
Product Purchased_Asus ROG                     bool
```

```
Product Purchased_Autodesk AutoCAD                    bool
Product Purchased_Bose QuietComfort                   bool
Product Purchased_Bose SoundLink Speaker              bool
Product Purchased_Canon DSLR Camera                   bool
Product Purchased_Canon EOS                           bool
Product Purchased_Dell XPS                            bool
Product Purchased_Dyson Vacuum Cleaner                bool
Product Purchased_Fitbit Charge                       bool
Product Purchased_Fitbit Versa Smartwatch             bool
Product Purchased_Garmin Forerunner                   bool
Product Purchased_GoPro Action Camera                 bool
Product Purchased_GoPro Hero                          bool
Product Purchased_Google Nest                         bool
Product Purchased_Google Pixel                        bool
Product Purchased_HP Pavilion                         bool
Product Purchased_LG OLED                             bool
Product Purchased_LG Smart TV                         bool
Product Purchased_LG Washing Machine                  bool
Product Purchased_Lenovo ThinkPad                     bool
Product Purchased_MacBook Pro                         bool
Product Purchased_Microsoft Office                    bool
Product Purchased_Microsoft Surface                   bool
Product Purchased_Microsoft Xbox Controller           bool
Product Purchased_Nest Thermostat                     bool
Product Purchased_Nikon D                             bool
Product Purchased_Nintendo Switch                     bool
Product Purchased_Nintendo Switch Pro Controller      bool
Product Purchased_Philips Hue Lights                  bool
Product Purchased_PlayStation                         bool
Product Purchased_Roomba Robot Vacuum                 bool
Product Purchased_Samsung Galaxy                      bool
Product Purchased_Samsung Soundbar                    bool
Product Purchased_Sony 4K HDR TV                      bool
Product Purchased_Sony PlayStation                    bool
Product Purchased_Sony Xperia                         bool
Product Purchased_Xbox                                bool
Product Purchased_iPhone                              bool
Ticket Type_Cancellation request                      bool
Ticket Type_Product inquiry                           bool
Ticket Type_Refund request                            bool
Ticket Type_Technical issue                           bool
Ticket Priority_High                                  bool
Ticket Priority_Low                                   bool
Ticket Priority_Medium                                bool
Ticket Channel_Email                                  bool
Ticket Channel_Phone                                  bool
Ticket Channel_Social media                           bool
dtype: object
```

[13]:
```
non_numeric_cols = X_train.select_dtypes(exclude=['number']).columns.tolist()
print("Non-numeric columns in X_train:", non_numeric_cols)
```

```
Non-numeric columns in X_train: ['Date of Purchase', 'Time to Resolution',
'Customer Gender_Male', 'Customer Gender_Other', 'Product Purchased_Amazon
Echo', 'Product Purchased_Amazon Kindle', 'Product Purchased_Apple AirPods',
'Product Purchased_Asus ROG', 'Product Purchased_Autodesk AutoCAD', 'Product
Purchased_Bose QuietComfort', 'Product Purchased_Bose SoundLink Speaker',
'Product Purchased_Canon DSLR Camera', 'Product Purchased_Canon EOS', 'Product
Purchased_Dell XPS', 'Product Purchased_Dyson Vacuum Cleaner', 'Product
Purchased_Fitbit Charge', 'Product Purchased_Fitbit Versa Smartwatch', 'Product
Purchased_Garmin Forerunner', 'Product Purchased_GoPro Action Camera', 'Product
Purchased_GoPro Hero', 'Product Purchased_Google Nest', 'Product
Purchased_Google Pixel', 'Product Purchased_HP Pavilion', 'Product Purchased_LG
OLED', 'Product Purchased_LG Smart TV', 'Product Purchased_LG Washing Machine',
'Product Purchased_Lenovo ThinkPad', 'Product Purchased_MacBook Pro', 'Product
Purchased_Microsoft Office', 'Product Purchased_Microsoft Surface', 'Product
Purchased_Microsoft Xbox Controller', 'Product Purchased_Nest Thermostat',
'Product Purchased_Nikon D', 'Product Purchased_Nintendo Switch', 'Product
Purchased_Nintendo Switch Pro Controller', 'Product Purchased_Philips Hue
Lights', 'Product Purchased_PlayStation', 'Product Purchased_Roomba Robot
Vacuum', 'Product Purchased_Samsung Galaxy', 'Product Purchased_Samsung
Soundbar', 'Product Purchased_Sony 4K HDR TV', 'Product Purchased_Sony
PlayStation', 'Product Purchased_Sony Xperia', 'Product Purchased_Xbox',
'Product Purchased_iPhone', 'Ticket Type_Cancellation request', 'Ticket
Type_Product inquiry', 'Ticket Type_Refund request', 'Ticket Type_Technical
issue', 'Ticket Priority_High', 'Ticket Priority_Low', 'Ticket Priority_Medium',
'Ticket Channel_Email', 'Ticket Channel_Phone', 'Ticket Channel_Social media']
```

[14]:
```
non_numeric_cols_test = X_test.select_dtypes(exclude=['number']).columns.
↪tolist()
print("Non-numeric columns in X_test:", non_numeric_cols_test)
```

```
Non-numeric columns in X_test: ['Date of Purchase', 'Time to Resolution',
'Customer Gender_Male', 'Customer Gender_Other', 'Product Purchased_Amazon
Echo', 'Product Purchased_Amazon Kindle', 'Product Purchased_Apple AirPods',
'Product Purchased_Asus ROG', 'Product Purchased_Autodesk AutoCAD', 'Product
Purchased_Bose QuietComfort', 'Product Purchased_Bose SoundLink Speaker',
'Product Purchased_Canon DSLR Camera', 'Product Purchased_Canon EOS', 'Product
Purchased_Dell XPS', 'Product Purchased_Dyson Vacuum Cleaner', 'Product
Purchased_Fitbit Charge', 'Product Purchased_Fitbit Versa Smartwatch', 'Product
Purchased_Garmin Forerunner', 'Product Purchased_GoPro Action Camera', 'Product
Purchased_GoPro Hero', 'Product Purchased_Google Nest', 'Product
Purchased_Google Pixel', 'Product Purchased_HP Pavilion', 'Product Purchased_LG
OLED', 'Product Purchased_LG Smart TV', 'Product Purchased_LG Washing Machine',
'Product Purchased_Lenovo ThinkPad', 'Product Purchased_MacBook Pro', 'Product
Purchased_Microsoft Office', 'Product Purchased_Microsoft Surface', 'Product
Purchased_Microsoft Xbox Controller', 'Product Purchased_Nest Thermostat',
```

```
'Product Purchased_Nikon D', 'Product Purchased_Nintendo Switch', 'Product
Purchased_Nintendo Switch Pro Controller', 'Product Purchased_Philips Hue
Lights', 'Product Purchased_PlayStation', 'Product Purchased_Roomba Robot
Vacuum', 'Product Purchased_Samsung Galaxy', 'Product Purchased_Samsung
Soundbar', 'Product Purchased_Sony 4K HDR TV', 'Product Purchased_Sony
PlayStation', 'Product Purchased_Sony Xperia', 'Product Purchased_Xbox',
'Product Purchased_iPhone', 'Ticket Type_Cancellation request', 'Ticket
Type_Product inquiry', 'Ticket Type_Refund request', 'Ticket Type_Technical
issue', 'Ticket Priority_High', 'Ticket Priority_Low', 'Ticket Priority_Medium',
'Ticket Channel_Email', 'Ticket Channel_Phone', 'Ticket Channel_Social media']
```

[15]:
```
X_train = X_train.drop(columns=non_numeric_cols)
X_test = X_test.drop(columns=non_numeric_cols_test)
```

[16]:
```
X_train = X_train.astype(float)
X_test = X_test.astype(float)
```

[17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 1.9732292027883853
R² Score: -0.00035449892385375215
```

[18]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

tree_model = DecisionTreeRegressor(random_state=42)  # Initializing the model

tree_model.fit(X_train, y_train)  # Training

y_pred_tree = tree_model.predict(X_test)  # Predict on test set

mse_tree = mean_squared_error(y_test, y_pred_tree)  # performance
r2_tree = r2_score(y_test, y_pred_tree)

print("Decision Tree - Mean Squared Error:", mse_tree)
print("Decision Tree - R² Score:", r2_tree)
```

```
Decision Tree - Mean Squared Error: 2.06468700160335
Decision Tree - R² Score: -0.046720233009350354
```

[19]:
```
from sklearn.ensemble import RandomForestRegressor                                #␣
 ↪Random Forest didn't improve much
from sklearn.metrics import mean_squared_error, r2_score

rf_model = RandomForestRegressor(random_state=42, n_estimators=100)  #␣
 ↪Initializing the model

rf_model.fit(X_train, y_train)   # Training

y_pred_rf = rf_model.predict(X_test)  # Predict on test data

mse_rf = mean_squared_error(y_test, y_pred_rf)  # Evaluate
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest - Mean Squared Error:", mse_rf)
print("Random Forest - R² Score:", r2_rf)
```

```
Random Forest - Mean Squared Error: 2.0628830931964885
Random Forest - R² Score: -0.04580571791505683
```

[20]:
```
df['Age_Group'] = pd.cut(df['Customer Age'], bins=[0, 30, 50, 100],␣
 ↪labels=['Young', 'Middle', 'Senior'])  # Create Age Group feature

df[['Customer Age', 'Age_Group']].head()  # new column
```

[20]:

	Customer Age	Age_Group
2	48	Middle
3	27	Young
4	67	Senior
10	48	Middle
11	51	Senior

[21]:
```
df['Resolution_Time_Category'] = pd.cut(        # Create Resolution Time␣
 ↪Category
    df['Resolution_Seconds'],
    bins=[-1, 86400, 259200, float('inf')],  # <1 day, 1-3 days, >3 days in␣
 ↪seconds
    labels=['Fast', 'Medium', 'Slow']
)

df[['Resolution_Seconds', 'Resolution_Time_Category']].head()  # Check new␣
 ↪column
```

[21]:

	Resolution_Seconds	Resolution_Time_Category
2	72728.0	Fast
3	14650.0	Fast
4	79212.0	Fast

```
10              7099.0                    Fast
11             41661.0                    Fast
```

[22]:
```
df = pd.get_dummies(df, columns=['Age_Group', 'Resolution_Time_Category'],␣
 ↪drop_first=True)
```

[23]:
```
df = df.drop(columns=['Customer Age', 'Resolution_Seconds'])
```

[24]:
```
X = df.drop(columns=['Customer Satisfaction Rating'])
y = df['Customer Satisfaction Rating']
```

[25]:
```
print(df['Customer Satisfaction Rating'].value_counts())  # checking whether␣
 ↪customers are giving how much rating
```

```
Customer Satisfaction Rating
3.0    580
1.0    553
2.0    549
5.0    544
4.0    543
Name: count, dtype: int64
```

[26]:
```
X = pd.get_dummies(X, drop_first=True)  # One-hot encode all categorical␣
 ↪(non-numeric) columns
```

[27]:
```
from sklearn.model_selection import train_test_split  # spliting

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[28]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

clf = RandomForestClassifier(n_estimators=100, random_state=42)  # Initialize␣
 ↪and train
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)  # Predicting
```

[29]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['Customer Satisfaction Rating'])  # Features and Target
y = df['Customer Satisfaction Rating'].astype(int)  # Convert to int for␣
 ↪classifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)  # Train-test split
```

[30]:
```
print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")  # Optional: Confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.2003610108303249

Classification Report:
              precision    recall  f1-score   support

           1       0.24      0.28      0.26       109
           2       0.20      0.19      0.19       117
           3       0.18      0.21      0.20       112
           4       0.19      0.19      0.19       108
           5       0.18      0.14      0.16       108

    accuracy                           0.20       554
   macro avg       0.20      0.20      0.20       554
weighted avg       0.20      0.20      0.20       554


Confusion Matrix:
[[30 17 28 16 18]
 [25 22 27 29 14]
 [30 23 24 18 17]
 [21 22 27 20 18]
 [18 28 26 21 15]]
```

[31]:
```
df['Satisfaction_Class'] = df['Customer Satisfaction Rating'].map({  # Group␣
 ↪ratings into Low (1-2), Neutral (3), and High (4-5)
    1: 'Low',
    2: 'Low',
    3: 'Neutral',
    4: 'High',
    5: 'High'
})
print(df['Satisfaction_Class'].value_counts())
```

```
Satisfaction_Class
Low        1102
High       1087
Neutral     580
```

```
Name: count, dtype: int64
```

[32]:
```
X = df.drop(columns=['Customer Satisfaction Rating', 'Satisfaction_Class'])
y = df['Satisfaction_Class']
```

[33]:
```
X = pd.get_dummies(X, drop_first=True)
```

[34]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[35]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.37364620938628157

Classification Report:
              precision    recall  f1-score   support

        High       0.38      0.43      0.40       216
         Low       0.39      0.49      0.43       226
     Neutral       0.17      0.04      0.07       112

    accuracy                           0.37       554
   macro avg       0.31      0.32      0.30       554
weighted avg       0.34      0.37      0.35       554


Confusion Matrix:
 [[ 92 114  10]
 [101 110  15]
 [ 51  56   5]]
```

[36]:
```
print(df.columns.tolist())
```

```
['Customer Gender', 'Product Purchased', 'Ticket Type', 'Ticket Status', 'Ticket
Priority', 'Ticket Channel', 'Customer Satisfaction Rating', 'Purchase_Year',
'Purchase_Month', 'Purchase_Day', 'Age_Group_Middle', 'Age_Group_Senior',
'Resolution_Time_Category_Medium', 'Resolution_Time_Category_Slow',
```

```
'Satisfaction_Class']
```

[37]:
```
df_full = pd.read_csv("customer_support_tickets.csv")  # or your file name
```

[38]:
```
print(df_full.columns.tolist())
```

```
['Ticket ID', 'Customer Name', 'Customer Email', 'Customer Age', 'Customer
Gender', 'Product Purchased', 'Date of Purchase', 'Ticket Type', 'Ticket
Subject', 'Ticket Description', 'Ticket Status', 'Resolution', 'Ticket
Priority', 'Ticket Channel', 'First Response Time', 'Time to Resolution',
'Customer Satisfaction Rating']
```

[39]:
```
print("Original full df columns:", df_full.columns.tolist())
print("Cleaned df columns:", df.columns.tolist())
```

```
Original full df columns: ['Ticket ID', 'Customer Name', 'Customer Email',
'Customer Age', 'Customer Gender', 'Product Purchased', 'Date of Purchase',
'Ticket Type', 'Ticket Subject', 'Ticket Description', 'Ticket Status',
'Resolution', 'Ticket Priority', 'Ticket Channel', 'First Response Time', 'Time
to Resolution', 'Customer Satisfaction Rating']
Cleaned df columns: ['Customer Gender', 'Product Purchased', 'Ticket Type',
'Ticket Status', 'Ticket Priority', 'Ticket Channel', 'Customer Satisfaction
Rating', 'Purchase_Year', 'Purchase_Month', 'Purchase_Day', 'Age_Group_Middle',
'Age_Group_Senior', 'Resolution_Time_Category_Medium',
'Resolution_Time_Category_Slow', 'Satisfaction_Class']
```

[40]:
```
df = df.reset_index(drop=True)
df_full = df_full.reset_index(drop=True)

df['Ticket Subject'] = df_full['Ticket Subject']  # Adding back
df['Ticket Description'] = df_full['Ticket Description']
```

[41]:
```
df['Description_Length'] = df['Ticket Description'].str.len()
df['Subject_Length']     = df['Ticket Subject'].str.len()

#df = df.drop(columns=['Ticket Description','Ticket Subject'])
```

[42]:
```
print(df_full.columns)
```

```
Index(['Ticket ID', 'Customer Name', 'Customer Email', 'Customer Age',
       'Customer Gender', 'Product Purchased', 'Date of Purchase',
       'Ticket Type', 'Ticket Subject', 'Ticket Description', 'Ticket Status',
       'Resolution', 'Ticket Priority', 'Ticket Channel',
       'First Response Time', 'Time to Resolution',
       'Customer Satisfaction Rating'],
      dtype='object')
```

[43]:
```
df['Description_Length'] = df_full['Ticket Description'].str.len()
df['Subject_Length'] = df_full['Ticket Subject'].str.len()
```

```
# df = df.drop(columns=['Ticket Subject', 'Ticket Description'],␣
 ↪errors='ignore')
```

[44]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=100, stop_words='english')  # adjust max␣
 ↪features as needed
tfidf_matrix = tfidf.fit_transform(df_full['Ticket Description'].fillna(''))
```

[45]:
```
import pandas as pd

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.
 ↪get_feature_names_out())
df = pd.concat([df.reset_index(drop=True), tfidf_df.reset_index(drop=True)],␣
 ↪axis=1)
```

[46]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['Customer Satisfaction Rating', 'Satisfaction_Class',␣
 ↪'Ticket Description', 'Ticket Subject'], errors='ignore')
y = df['Satisfaction_Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[47]:
```
X = df.drop(columns=['Customer Satisfaction Rating', 'Satisfaction_Class',␣
 ↪'Ticket Subject', 'Ticket Description'], errors='ignore')
y = df['Satisfaction_Class']
X_encoded = pd.get_dummies(X, drop_first=True)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.
 ↪2, random_state=42)
```

[48]:
```
print("NaNs in X_train:", X_train.isnull().sum().sum())
print("NaNs in y_train:", y_train.isnull().sum())
```

```
NaNs in X_train: 22845
NaNs in y_train: 4569
```

[49]:
```
train_combined = pd.concat([X_train, y_train], axis=1)  # Combining X and y␣
 ↪temporarily to drop rows with any NaN

train_combined_clean = train_combined.dropna()  # Drop all rows

X_train_clean = train_combined_clean.drop(columns=y_train.name)  # Split
y_train_clean = train_combined_clean[y_train.name]
```

[50]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
import pandas as pd

train_combined = pd.concat([X_train, y_train], axis=1)

train_clean = train_combined.dropna()

X_train_clean = train_clean.drop(columns=y_train.name)
y_train_clean = train_clean[y_train.name]

test_combined = pd.concat([X_test, y_test], axis=1)
test_clean = test_combined.dropna()

X_test_clean = test_clean.drop(columns=y_test.name)
y_test_clean = test_clean[y_test.name]

clf = RandomForestClassifier(n_estimators=200, random_state=42)  # train the␣
 ↪Random Forest model
clf.fit(X_train_clean, y_train_clean)

y_pred = clf.predict(X_test_clean)  # predictions and evaluate

print(" Accuracy:", accuracy_score(y_test_clean, y_pred))
print("\n Classification Report:\n", classification_report(y_test_clean,␣
 ↪y_pred))
print("\n Confusion Matrix:\n", confusion_matrix(y_test_clean, y_pred))
```

```
Accuracy: 0.39431616341030196

Classification Report:
              precision    recall  f1-score   support

        High       0.38      0.52      0.43       209
         Low       0.43      0.45      0.44       249
     Neutral       0.14      0.02      0.03       105

    accuracy                           0.39       563
   macro avg       0.32      0.33      0.30       563
weighted avg       0.36      0.39      0.36       563


Confusion Matrix:
[[108  92   9]
[134 112   3]
[ 46  57   2]]
```

[51]:
```
# Drop rows where Customer Satisfaction Rating is missing
df_model = df_full.dropna(subset=['Customer Satisfaction Rating'])

# Prepare structured data
X_struct = df_model.drop(columns=['Ticket Description', 'Ticket Subject',␣
 ↪'Customer Satisfaction Rating'], errors='ignore')
X_struct = pd.get_dummies(X_struct)
X_struct = X_struct.fillna(0)

# Target as integers
y = df_model['Customer Satisfaction Rating'].astype(int)

# Split and train
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X_struct, y, test_size=0.2,␣
 ↪random_state=42)

clf = RandomForestClassifier(n_estimators=200, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.2003610108303249

Classification Report:
              precision    recall  f1-score   support

           1       0.17      0.19      0.18       109
           2       0.19      0.15      0.16       117
           3       0.23      0.36      0.28       112
           4       0.17      0.13      0.15       108
           5       0.22      0.18      0.20       108

    accuracy                           0.20       554
   macro avg       0.20      0.20      0.19       554
weighted avg       0.20      0.20      0.19       554


Confusion Matrix:
```

```
[[21 20 30 18 20]
 [24 17 46 21  9]
 [27 18 40 10 17]
 [23 20 31 14 20]
 [27 15 30 17 19]]
```

[52]:
```python
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix

# Final evaluation on test set
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.2003610108303249

Classification Report:
              precision    recall  f1-score   support

           1       0.17      0.19      0.18       109
           2       0.19      0.15      0.16       117
           3       0.23      0.36      0.28       112
           4       0.17      0.13      0.15       108
           5       0.22      0.18      0.20       108

    accuracy                           0.20       554
   macro avg       0.20      0.20      0.19       554
weighted avg       0.20      0.20      0.19       554


Confusion Matrix:
[[21 20 30 18 20]
 [24 17 46 21  9]
 [27 18 40 10 17]
 [23 20 31 14 20]
 [27 15 30 17 19]]
```

[53]:
```python
# ================================
# Project Summary: Customer Satisfaction Prediction
# ================================

# Objective:
# Predict customer satisfaction rating (1-5 scale) from support ticket data.
```

```
# Dataset:
# - Source: customer_support_tickets.csv
# - Final cleaned rows: ~2700
# - Target: 'Customer Satisfaction Rating'

# Preprocessing:
# - Dropped rows with missing 'Customer Satisfaction Rating'
# - Converted 'Date of Purchase' to Purchase_Year, Month, Day
# - Converted 'Time to Resolution' into Resolution_Seconds
# - Feature engineered:
#   - Age group dummies (e.g., Age_Group_Middle, Senior)
#   - Resolution speed category dummies
#   - One-hot encoded categorical columns
# - Attempted text features using TF-IDF on Ticket Subject + Description␣
 ↪(optional step)

# Models Tried:
# - Linear Regression (not suitable for classification)
# - Decision Tree Classifier: ~20% accuracy
# - Random Forest Classifier: ~39.4% accuracy (best)

# Final Model:
# - RandomForestClassifier(n_estimators=200, random_state=42)
# - Accuracy: 0.3943
# - Confusion mainly between nearby ratings (3 vs 2 or 4)

# Observations:
# - Model performs better on 'High' and 'Low' classes
# - 'Neutral' class was hardest to predict (lowest recall)
# - Accuracy may improve with:
#   - More advanced NLP methods (like embeddings)
#   - Dimensionality reduction
#   - Balancing classes

# Status:
# - Data cleaning
# - Feature engineering
# - Model training
# - Results analysis
```