Flight Fare Prediction

Welcome to our presentation on flight fare prediction. Flight ticket prices often change due to various factors like demand, season, and airline policies, making it hard for travelers to know the best time to book. This is where machine learning comes in. By analyzing past data such as routes, dates, airlines, and booking patterns, Data Science models can predict future prices with good accuracy. This helps travelers save money and make smarter booking decisions. It also benefits airlines by helping them optimize pricing and manage seat availability. In this presentation, we'll explore how this technology works and why it matters.

Domain and Context of Flight Fare Prediction

Airline Industry / Travel Tech

The prediction of flight ticket prices is a critical application within the airline industry and the broader travel technology sector. It supports strategic planning and operational efficiency for all participants.

Dynamic Pricing & Planning

Flight fares are influenced by a multitude of factors including airline, origin, destination, journey date, duration, and number of stops. Accurate predictions empower customers to plan purchases effectively and allow companies to offer competitive, dynamic pricing strategies.

Beneficiaries of Accurate Prediction

Travelers: Identify optimal booking times for cost savings.

Travel Agencies: Implement sophisticated dynamic pricing models.

Airlines: Enhance demand forecasting and revenue management.

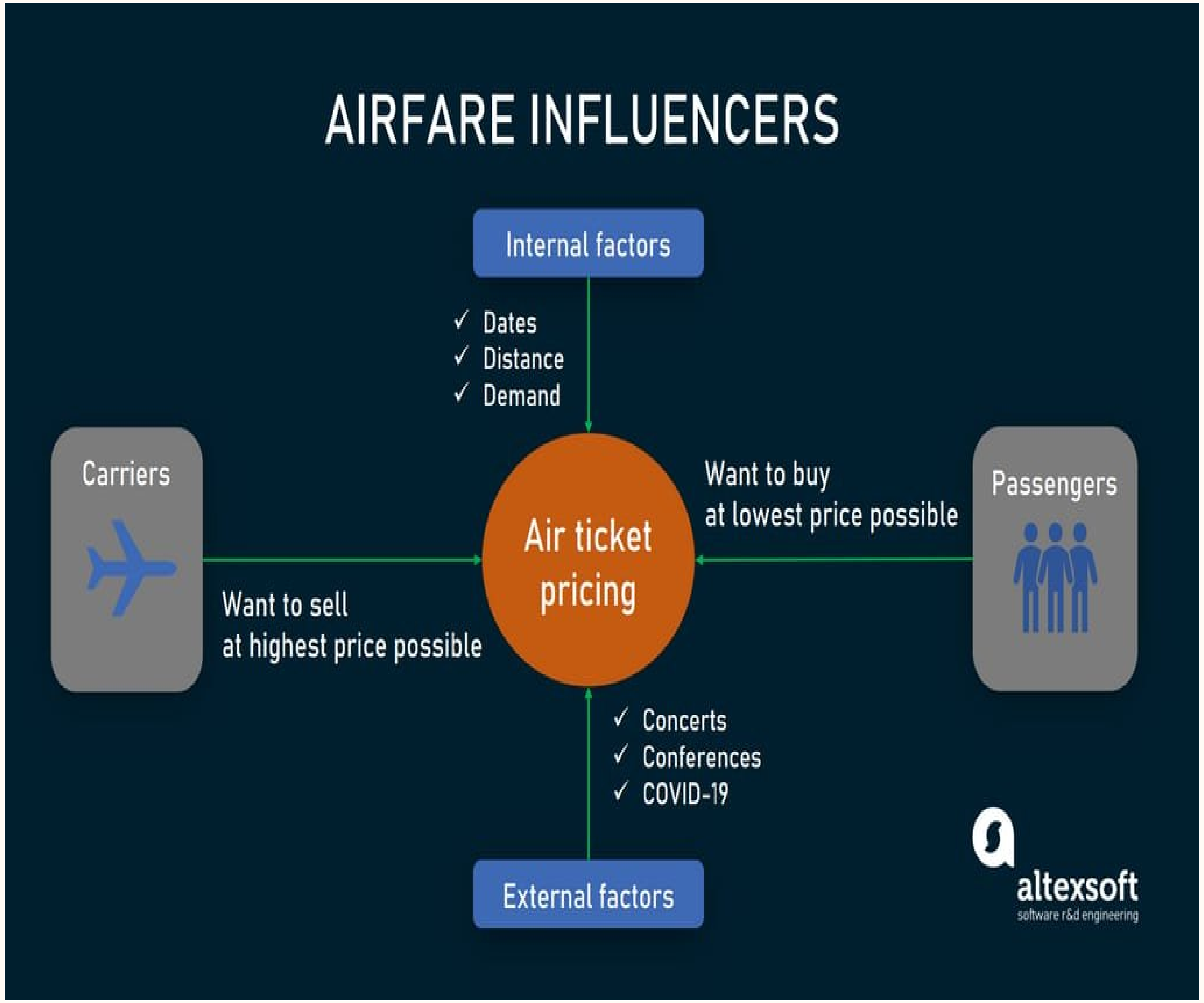
Dataset Overview: Features and Categorization

The dataset used for this case study, often sourced from platforms like Kaggle, comprises

various features crucial for fare prediction.

Airline	Categorical: Carrier of the flight.
Source, Destination	Categorical: Departure and arrival cities.
Date_of_Journey	Date/Time: Specific date of travel.
Duration	Textual/Mixed: Total travel time.
Total_Stops	Categorical: Number of layovers.
Additional_Info	Categorical: Miscellaneous flight details.
Price	Numerical: Target variable (flight fare).

Understanding these features and their types is fundamental for effective data preprocessing and model building.



Problem Statement & Workflow for Flight Fare Prediction

Problem Statement:
Airline ticket prices fluctuate based on various factors such as airline, source and destination cities, date of journey, duration, number of stops, and seasonality.
The goal of this project is to predict flight fares accurately so that:

- Travelers can make cost-effective booking decisions.
- Travel agencies/airlines can optimize their pricing strategies.

Work Done:

1. Collected the flight fare dataset from Kaggle
2. Filled missing values using median/mode as required
3. Removed unnecessary columns from the dataset
4. Converted columns to correct data types
5. Scaled the numerical features for uniformity
6. Visualized relationships between key columns and flight fare

Summary of Preprocessing Steps

Handling Missing Values

Missing entries were imputed using appropriate statistical methods (e.g., **mean, median, mode**) to maintain data integrity.

Duration Transformation

The 'Duration' feature, originally a string, was meticulously **converted into minutes** to allow for numerical analysis and model consumption.

Date/Time Transformation

Granular features like **day, month, and hour** were extracted from 'Date_of_Journey', 'Dep_Time', and 'Arrival_Time' to capture temporal patterns.

Categorical Data Encoding

Categorical features were transformed into numerical formats using **Label Encoding** or **One-Hot Encoding**, depending on their cardinality and potential ordinality.

Outlier Removal

Outliers, particularly in the 'Price' distribution, were **identified and removed** to prevent undue influence on model training and improve prediction accuracy.

Data Readiness

These comprehensive steps collectively ensured a **final cleaned dataset**, optimally prepared and ready for subsequent modelling phases.

Raw Flight Dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13354 entries, 0 to 13353
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          13354 non-null  object
 1   Date_of_Journey  13354 non-null  object
 2   Source           13354 non-null  object
 3   Destination      13354 non-null  object
 4   Dep_Time         13354 non-null  object
 5   Arrival_Time     13354 non-null  object
 6   Duration         13354 non-null  object
 7   Total_Stops      13353 non-null  object
 8   Additional_Info  13354 non-null  object
 9   is_train         13354 non-null  int64
dtypes: int64(1), object(9)
memory usage: 1.0+ MB
```

Preprocessed Dataset with Encoded & Scaled Features

```
In [86]: X_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 33 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   Total_Stops                                10683 non-null  float64
 1   Additional_Info                            10683 non-null  float64
 2   Date                                       10683 non-null  float64
 3   Month                                      10683 non-null  float64
 4   Year                                       10683 non-null  float64
 5   Dep_hour                                   10683 non-null  float64
 6   Dep_minute                                 10683 non-null  float64
 7   Arrival_hour                               10683 non-null  float64
 8   Arrival_minute                             10683 non-null  float64
 9   Duration_minutes                           10683 non-null  float64
 10  Airline_Air Asia                           10683 non-null  float64
 11  Airline_Air India                          10683 non-null  float64
 12  Airline_GoAir                              10683 non-null  float64
 13  Airline_IndiGo                             10683 non-null  float64
 14  Airline_Jet Airways                        10683 non-null  float64
 15  Airline_Jet Airways Business               10683 non-null  float64
 16  Airline_Multiple carriers                  10683 non-null  float64
 17  Airline_Multiple carriers Premium economy  10683 non-null  float64
 18  Airline_SpiceJet                           10683 non-null  float64
 19  Airline_Trujet                             10683 non-null  float64
 20  Airline_Vistara                            10683 non-null  float64
 21  Airline_Vistara Premium economy            10683 non-null  float64
 22  Source_Banglore                            10683 non-null  float64
 23  Source_Chennai                             10683 non-null  float64
 24  Source_Delhi                               10683 non-null  float64
 25  Source_Kolkata                             10683 non-null  float64
 26  Source_Mumbai                              10683 non-null  float64
 27  Destination_Banglore                       10683 non-null  float64
 28  Destination_Cochin                         10683 non-null  float64
 29  Destination_Delhi                          10683 non-null  float64
 30  Destination_Hyderabad                      10683 non-null  float64
 31  Destination_Kolkata                        10683 non-null  float64
 32  Destination_New Delhi                      10683 non-null  float64
dtypes: float64(33)
memory usage: 2.7 MB
```

Feature Engineering

```
#feature engineering
df['Date'] = df['Date_of_Journey'].str.split('/').str[0]
df['Month'] = df['Date_of_Journey'].str.split('/').str[1]
df['Year'] = df['Date_of_Journey'].str.split('/').str[2]
df.drop('Date_of_Journey',axis=1,inplace=True)
df.head()
```

```
df['Dep_hour'] = df['Dep_Time'].str.split(':').str[0]
df['Dep_minute'] = df['Dep_Time'].str.split(':').str[1]
df.drop('Dep_Time', axis=1, inplace=True)

df['Arrival_hour'] = df['Arrival_Time'].str.split(' ').str[0].str.split(':').str[0]
df['Arrival_minute'] = df['Arrival_Time'].str.split(' ').str[0].str.split(':').str[1]
df.drop('Arrival_Time', axis=1, inplace=True)

df.head()
```

```
#accessing unique values in total_stops column
df['Total_Stops'].unique()
```

```
array(['non-stop', '2 stops', '1 stop', '3 stops', nan, '4 stops'],
      dtype=object)
```

```
#converting text data to numerical values
df['Total_Stops'] = df['Total_Stops'].map({'non-stop':0, '2 stops':2, '1 stop':1, '3 stops':3, '4 stops':4})
df.head()
```

```
#splitting duration to hours and minutes
def extract_duration_hours(duration_str):
    try:
        if 'h' in duration_str:
            return int(duration_str.split('h')[0])
        else:
            return 0
    except:
        return 0

def extract_duration_minutes(duration_str):
    try:
        if 'm' in duration_str:
            parts = duration_str.split(' ')
            for part in parts:
                if 'm' in part:
                    return int(part.replace('m', ''))
            return 0
        else:
            return 0
    except:
        return 0

# Apply the functions
df['Duration_hour'] = df['Duration'].apply(extract_duration_hours)
df['Duration_minute'] = df['Duration'].apply(extract_duration_minutes)
df.drop('Duration', axis=1, inplace=True)

#converting hours to minutes to redcue the features
df['Duration_minutes'] = df['Duration_hour']*60 + df['Duration_minute']
df.drop(['Duration_hour','Duration_minute'], axis=1, inplace=True)

df.head()
```

Original Dataset

First few rows of combined dataset:

Out[31]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	is_train
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	1
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	1
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	1
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	1
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	1

```
#getting count of variuos data in 'additional info'
df['Additional_Info'].value_counts()
```

```
Additional_Info
No info                         10493
In-flight meal not included      2426
No check-in baggage included      396
1 Long layover                     20
Change airports                     8
Business class                      5
No Info                             3
1 Short layover                     1
Red-eye flight                      1
2 Long layover                      1
Name: count, dtype: int64
```

```
#dividing it to two categories
df['Additional_Info'] = df['Additional_Info'].apply(lambda x: 'No info' if x == 'No info' else 'Others')
df['Additional_Info'].value_counts()
```

```
Additional_Info
No info     10493
Others       2861
Name: count, dtype: int64
```

Encoding & Scaling

```
#encoding categorical columns
%pip install scikit-learn
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

le = LabelEncoder()
df['Additional_Info'] = le.fit_transform(df['Additional_Info'])

cols_to_encode = ['Airline', 'Source', 'Destination']

ohe = OneHotEncoder(sparse_output=False, drop=None)
encoded = ohe.fit_transform(df[cols_to_encode])
encoded_cols = ohe.get_feature_names_out(cols_to_encode)
encoded_df = pd.DataFrame(encoded, columns=encoded_cols, index=df.index)
df = pd.concat([df.drop(cols_to_encode, axis=1), encoded_df], axis=1)
```

```
#scale numeric features to mean 0 and std dev 1
from sklearn.preprocessing import StandardScaler
import pandas as pd

#splitting dataset back to training and testing using flag
train_clean = df[df['is_train'] == 1].drop('is_train', axis=1)
test_clean = df[df['is_train'] == 0].drop('is_train', axis=1)

scaler = StandardScaler()
X_train = scaler.fit_transform(train_clean)
X_test = scaler.transform(test_clean)

X_train_df = pd.DataFrame(X_train, columns=train_clean.columns)
X_test_df = pd.DataFrame(X_test, columns=test_clean.columns)
```

```
X_train_df.head()
```

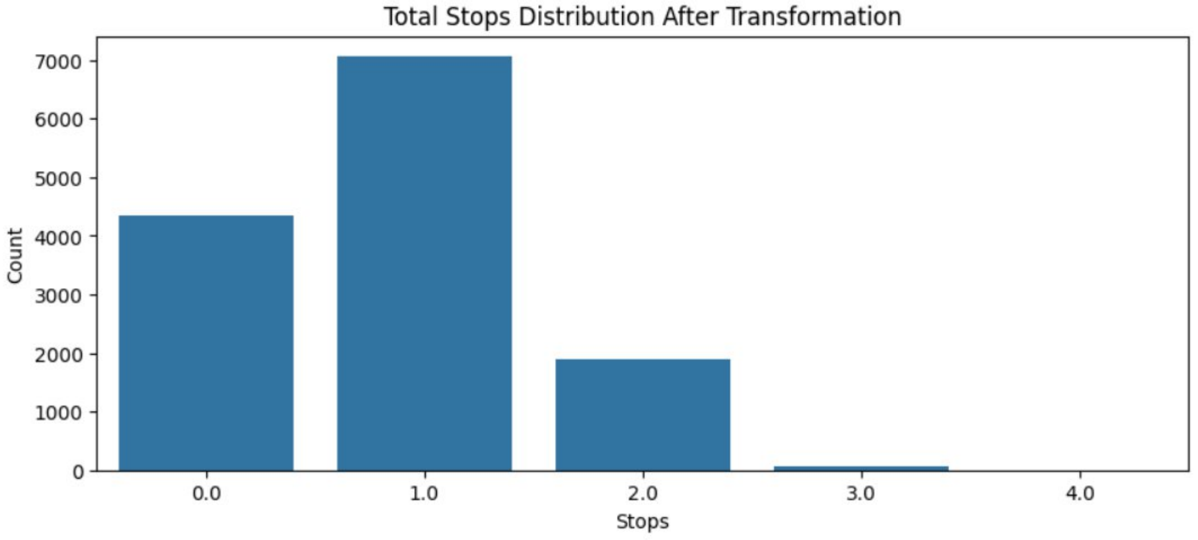
	Total_Stops	Additional_Info	Date	Month	Year	Dep_hour	Dep_minute	Arrival_hour	Arrival_minute	Duration_minutes	...
0	-1.220744	-0.529309	1.237383	-1.467490	0.0	1.654259	-0.235050	-1.800427	-0.890057	-0.931583	...
1	1.741483	-0.529309	-1.475239	0.250276	0.0	-1.303095	1.363492	-0.050851	-0.587124	-0.390072	...
2	1.741483	-0.529309	-0.531719	1.109160	0.0	-0.607247	0.031373	-1.363033	0.018744	0.978475	...
3	0.260370	-0.529309	-0.177898	0.250276	0.0	0.958411	-1.034321	1.407129	0.321677	-0.626367	...
4	0.260370	-0.529309	-1.475239	-1.467490	0.0	0.610487	1.363492	1.115533	0.624611	-0.705132	...

5 rows × 33 columns

Visualization of Applied Transformation

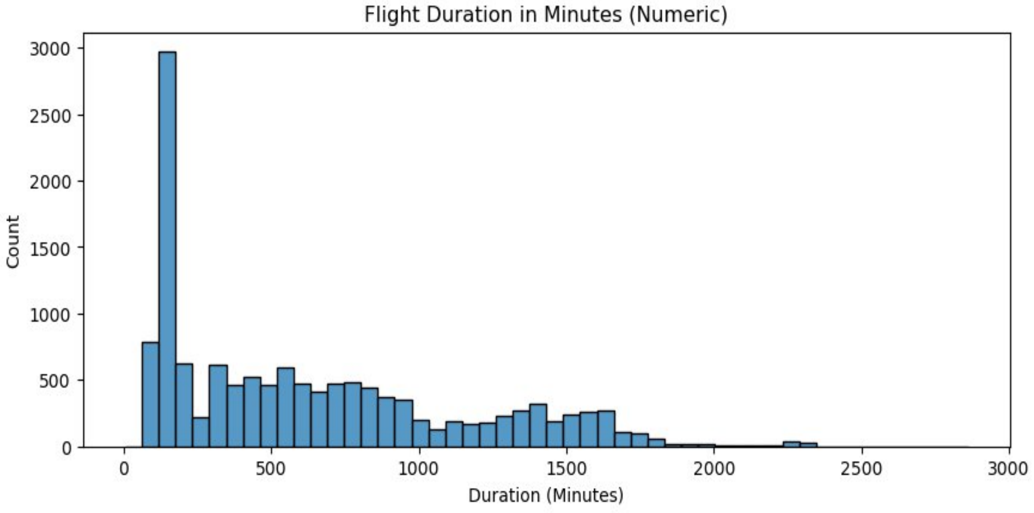
Plot for Total Stops Distribution after Encoding

In [59]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='Total_Stops', data=df)
plt.title("Total Stops Distribution After Transformation")
plt.xlabel("Stops")
plt.ylabel("Count")
plt.show()
```



Plot for Flight Distribution After Conversion to Minutes

[127]:
```
plt.figure(figsize=(10,4))
sns.histplot(df['Duration_minutes'], bins=50)
plt.title("Flight Duration in Minutes (Numeric)")
plt.xlabel("Duration (Minutes)")
plt.ylabel("Count")
plt.show()
```



Model Implementation: K-Nearest Neighbours (KNN) Classifier

- **Objective:** To classify flight fares into three categories: 'Low', 'Medium', and 'High’.

```
# Convert the target variable 'Price' into classes
  price_bins = [0, 6000, 12000, float('inf’)]
  price_labels = ['Low', 'Medium', 'High’]
  y_class = pd.cut(y, bins=price_bins, labels=price_labels, right=False)
# Import the classifier and split the data for validation
  from sklearn.neighbors import KNeighborsClassifier
  from sklearn.model_selection import train_test_split
# Split the preprocessed training data for training and validation
  X_train, X_val, y_train_class, y_val_class = train_test_split(
  X_train_df, y_class, test_size=0.2, random_state=42, stratify=y_class)
# Initialize and train the KNN Classifier with K=5
  knn_classifier = KNeighborsClassifier(n_neighbors=5)
  knn_classifier.fit(X_train, y_train_class)
```

Target Variable Distribution:

- Medium: 4620
- Low: 3167
- High: 2896

Model Evaluation

- 1. The trained KNN model was evaluated on the validation set to assess its performance.

- **2. Accuracy Score:** The model achieved an accuracy of **86%** on the validation data.

- **3. Confusion Matrix:**

$$\begin{bmatrix} 485 & 7 & 87 \\ 2 & 564 & 68 \\ 71 & 66 & 787 \end{bmatrix}$$

Classification Report Structure

	precision	recall	f1-score	support
Class 1 (e.g., High)	0.87	0.84	0.85	579
Class 2 (e.g., Low)	0.89	0.89	0.89	634
Class 3 (e.g., Medium)	0.84	0.85	0.84	924
accuracy			0.86	2137
macro avg	0.86	0.86	0.86	2137
weighted avg	0.86	0.86	0.86	2137

Conclusion & Insights

- ***Summary:***
- This project successfully developed a machine learning model to predict flight fare categories with a high degree of accuracy.
- Extensive data preprocessing and feature engineering were performed to prepare the dataset for modeling. This included handling missing values, converting data types, and encoding categorical variables.
- A K-Nearest Neighbors (KNN) classification model was implemented and trained on the cleaned dataset.
- ***Key Insights:***
- The model achieved an accuracy of 86%, demonstrating its effectiveness in classifying flight prices.
- The classification report shows strong precision and recall across all price categories, indicating a well-balanced model.
- Hyperparameter tuning helped in identifying the optimal K value, which is crucial for the performance of the KNN algorithm.