Question 1 : What is the difference between AI, ML, DL, and Data Science? Provide a brief explanation of each.

Answer:

- Artificial Intelligence (AI): AI refers to the broad field of research aimed at creating machines that can perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. AI encompasses a range of techniques, including ML, DL, and rule-based systems.

- Machine Learning (ML): ML is a subset of AI that focuses on developing algorithms and statistical models that enable machines to learn from data, make predictions, and improve their performance over time. ML techniques include regression, decision trees, clustering, and neural networks.

- Deep Learning (DL): DL is a subset of ML that uses neural networks with multiple layers to analyze data. DL techniques are particularly effective for image and speech recognition, natural language processing, and other complex tasks.

- Data Science: Data Science is an interdisciplinary field that combines techniques from computer science, statistics, and domain expertise to extract insights and knowledge from data. Data Science encompasses data cleaning, visualization, modeling, and interpretation, often using ML and DL techniques.

Key differences:

- Scope: AI is the broadest field, followed by Data Science, ML, and then DL.

- Techniques: AI includes rule-based systems, ML focuses on learning from data, DL uses neural networks, and Data Science combines various techniques.

- Applications: AI has many applications, ML is used for prediction and classification, DL is used for complex tasks like image recognition, and Data Science is used for extracting insights from data.

Question 2: Explain overfitting and underfitting in ML. How can you detect and prevent them?

Answer:

Overfitting and Underfitting in ML:

Overfitting and underfitting are two common problems in machine learning that can affect the performance of a model.

- Overfitting: Overfitting occurs when a model is too complex and learns the training data too well, capturing noise and outliers rather than the underlying pattern. This results in poor performance on new, unseen data.

- Underfitting: Underfitting occurs when a model is too simple and fails to capture the underlying pattern in the data, resulting in poor performance on both the training and new data.

Bias-Variance Tradeoff:

The bias-variance tradeoff is a fundamental concept in machine learning that helps explain overfitting and underfitting.

- Bias: Bias refers to the error introduced by simplifying a model or making assumptions about the data. High bias leads to underfitting.
- Variance: Variance refers to the error introduced by fitting a model too closely to the training data. High variance leads to overfitting.

Detecting Overfitting and Underfitting:

1. Monitor performance metrics: Track metrics like accuracy, precision, recall, and F1-score on the training and validation sets.

2. Plot learning curves: Plot the training and validation errors as a function of the training size or iterations.

3. Cross-validation: Use techniques like k-fold cross-validation to evaluate the model's performance on unseen data.

Preventing Overfitting and Underfitting:

1. Regularization techniques: L1 and L2 regularization, dropout, and early stopping can help prevent overfitting.

2. Cross-validation: Use cross-validation to evaluate the model's performance on unseen data and prevent overfitting.

3. Data augmentation: Increase the size of the training data by applying transformations or generating new samples.

4. Feature selection: Select relevant features to reduce the dimensionality of the data.

5. Model selection: Choose a model with the right complexity for the problem at hand.

6. Hyperparameter tuning: Tune hyperparameters like learning rate, batch size, and number of hidden layers to optimize the model's performance.

Some popular regularization techniques include:

- L1 regularization (Lasso): Adds a penalty term proportional to the absolute value of the model's weights.
- L2 regularization (Ridge): Adds a penalty term proportional to the square of the model's weights.

- Dropout: Randomly drops out neurons during training to prevent overfitting.

- Early stopping: Stops training when the model's performance on the validation set starts to degrade.

Question 3:How would you handle missing values in a dataset? Explain at least three methods with examples.

Answer:

Handling missing values is a crucial step in data preprocessing. Here are three methods to handle missing values:

- Deletion: Deletion involves removing rows or columns with missing values. There are two types of deletion:

- Listwise deletion: Remove rows with missing values.

- Pairwise deletion: Remove columns with missing values.
Example: If a dataset has 1000 rows and 10 columns, and 50 rows have missing values, listwise deletion would remove those 50 rows, resulting in a dataset with 950 rows.

- Mean/Median Imputation: Impute missing values with the mean or median of the respective feature.

- Mean imputation: Replace missing values with the mean of the feature.

- Median imputation: Replace missing values with the median of the feature.
Example: If a feature 'Age' has missing values, and the mean age is 35, replace missing values with 35.

- Predictive Modeling: Use machine learning models to predict missing values.

- Regression: Use regression models to predict continuous missing values.

- Classification: Use classification models to predict categorical missing values.
Example: If a feature 'Income' has missing values, use a regression model to predict the missing values based on other features like 'Age', 'Education', and 'Occupation'.

Some popular techniques for predictive modeling include:

- K-Nearest Neighbors (KNN) imputation: Impute missing values based on the k-nearest neighbors.

- Multiple Imputation by Chained Equations (MICE): Impute missing values using a series of regression models.

- Random Forest imputation: Impute missing values using a random forest model.

Here's a Python example using mean imputation with scikit-learn:

```
from sklearn.impute import SimpleImputer
import numpy as np

# Create a sample dataset with missing values
data = np.array([[1, 2], [np.nan, 3], [4, np.nan]])

# Create an imputer object with mean imputation strategy
imputer = SimpleImputer(strategy='mean')

# Fit and transform the data
imputed_data = imputer.fit_transform(data)

print(imputed_data)
```

Output:

```
[[1.  2. ]
 [2.5 3. ]
 [4.  2.5]]
```

Question 4:What is an imbalanced dataset? Describe two techniques to handle it (theoretical + practical).

Answer:

Imbalanced Dataset:

An imbalanced dataset is a dataset where one class has a significantly larger number of instances than the other classes. This can lead to biased models that favor the majority class, resulting in poor performance on the minority class.

Techniques to Handle Imbalanced Datasets:

1. SMOTE (Synthetic Minority Oversampling Technique):

- SMOTE is an oversampling technique that generates synthetic samples from the minority class.
- It works by selecting a random sample from the minority class and one of its k-nearest neighbors, then generating a new sample along the line connecting these two samples.
- SMOTE helps to increase the size of the minority class, reducing the imbalance in the dataset.

```
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

Generate a sample imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_repeated=0, n_classes=2, n_clusters_per_class=1, weights=[0.9, 0.1], random_state=42)

Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Apply SMOTE oversampling
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_train, y_train)

print("Original training set shape:", X_train.shape, y_train.shape)
print("Resampled training set shape:", X_res.shape, y_res.shape)
```

2. **Class Weights in Models:**

* Many machine learning models allow you to specify class weights, which can help handle imbalanced datasets.
* Class weights assign a higher penalty to misclassifying the minority class, encouraging the model to focus on the minority class.
* This technique can be used with various models, including logistic regression, decision trees, and support vector machines.

python
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

# Train a logistic regression model with class weights
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print("F1-score:", f1_score(y_test, y_pred, average='macro'))
```

Other techniques to handle imbalanced datasets include:

- Random Under/Oversampling: Randomly remove samples from the majority class (undersampling) or duplicate samples from the minority class (oversampling).

- Random Forest with class weights: Use a random forest model with class weights to handle imbalanced datasets.

- Anomaly detection: Use anomaly detection techniques to identify the minority class as anomalies.

Question 5: Why is feature scaling important in ML? Compare Min-Max scaling and Standardization. Hint: Explain impact on distance-based algorithms (e.g., KNN, SVM) and gradient descent.

Answer:

Feature scaling is a crucial step in machine learning that ensures all features contribute equally to the model. Here's why it's important and how Min-Max scaling and Standardization differ:

Why Feature Scaling Matters:

- Distance-Based Algorithms: KNN, SVM, and clustering algorithms rely on distance calculations, which can be dominated by features with large scales. Scaling ensures fair comparisons.

- Gradient Descent: Feature scaling helps gradient descent converge faster and more efficiently, especially for algorithms like linear regression, logistic regression, and neural networks.

- Model Performance: Scaling improves model performance, reduces bias, and prevents features with large ranges from dominating the model.

Min-Max Scaling (Normalization):

- Scales features to a fixed range, typically [0, 1] or [-1, 1].

- Formula: X_scaled = (X - X_min) / (X_max - X_min)

- Preserves relationships between data points.

- Sensitive to outliers.

Standardization (Z-Score Normalization):

- Centers data around the mean (0) with a standard deviation of 1.

- Formula: X_scaled = (X - mean) / std_dev

- Not sensitive to outliers.

- Suitable for algorithms assuming normal distributions.

Key Differences:

- Range: Min-Max scaling has a fixed range, while standardization doesn't.

- Outlier Sensitivity: Standardization is less sensitive to outliers.

- Algorithm Suitability: Min-Max scaling is suitable for algorithms requiring bounded inputs, while standardization is better for algorithms assuming normal distributions [1] [2] [3].

Question 6: Compare Label Encoding and One-Hot Encoding. When would you prefer one over the other? Hint: Consider categorical variables with ordinal vs. nominal relationships.

Answer:

Label Encoding and One-Hot Encoding are two popular techniques for encoding categorical variables.

Label Encoding:

- Assigns a unique integer value to each category.
- Suitable for categorical variables with an ordinal relationship (e.g., education level: high school, bachelor's, master's, Ph.D.).
- Preserves the order of categories.
- Reduces dimensionality compared to One-Hot Encoding.

One-Hot Encoding:

- Creates a binary column for each category.
- Suitable for categorical variables with no ordinal relationship (e.g., colors: red, green, blue).
- Avoids implying order between categories.
- Increases dimensionality.

When to Prefer Each:

- Label Encoding:

  - Ordinal categorical variables.
  - Limited categories.
  - Tree-based models (e.g., decision trees, random forests).

- One-Hot Encoding:
  - Nominal categorical variables.
  - Many categories (but beware of dimensionality curse).
  - Linear models (e.g., linear regression, logistic regression).

Here's a Python example using scikit-learn:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import pandas as pd

# Sample data
```

```
data = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green']})

# Label Encoding
label_encoder = LabelEncoder()
data['color_label'] = label_encoder.fit_transform(data['color'])
print("Label Encoding:\n", data)

# One-Hot Encoding
onehot_encoder = OneHotEncoder(sparse=False)
data_onehot = onehot_encoder.fit_transform(data[['color']])
print("One-Hot Encoding:\n", data_onehot)
```

Question 7: Google Play Store Dataset a). Analyze the relationship between app categories and ratings. Which categories have the highest/lowest average ratings, and what could be the possible reasons? Dataset: https://github.com/MasteriNeuron/datasets.git (Include your Python code and output in the code box below.)

Answer:

```
import pandas as pd
import numpy as np

# Load dataset (assuming you have downloaded googleplaystore.csv locally)
df = pd.read_csv('/googleplaystore (2).csv')

# Inspect columns
print("Columns:", df.columns.tolist())
print(df.head())

# Clean data: convert rating to numeric, drop invalid
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df_clean = df.dropna(subset=['Category', 'Rating'])

# Compute average rating by category
cat_rating =
df_clean.groupby('Category')['Rating'].agg(['mean','count','std']).sort_values(by='mean',
ascending=False)
print(cat_rating)

# For robustness: only consider categories with at least some minimum number of apps (e.g.
50)
min_n = 50
```

```
cat_rating_min = cat_rating[cat_rating['count'] >= min_n]
print("\nCategories with >= {} apps:".format(min_n))
print(cat_rating_min.sort_values(by='mean', ascending=False))

# Also look at categories with low counts (maybe high variance)
cat_rating_small = cat_rating[cat_rating['count'] < min_n]
print("\nCategories with < {} apps (likely noisy):".format(min_n))
print(cat_rating_small.sort_values(by='mean', ascending=False))
```

Output :

```
Columns: ['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating',
'Genres', 'Last Updated', 'Current Ver', 'Android Ver']
                                                App        Category  Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN     4.1
1                                Coloring book moana  ART_AND_DESIGN     3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN     4.7
3                               Sketch - Draw & Paint  ART_AND_DESIGN     4.5
4                Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN     4.3

  Reviews  Size     Installs  Type Price Content Rating  \
0     159   19M      10,000+  Free     0       Everyone
1     967   14M     500,000+  Free     0       Everyone
2   87510  8.7M   5,000,000+  Free     0       Everyone
3  215644   25M  50,000,000+  Free     0           Teen
4     967  2.8M     100,000+  Free     0       Everyone

                      Genres      Last Updated         Current Ver  \
0               Art & Design   January 7, 2018               1.0.0
1  Art & Design;Pretend Play  January 15, 2018               2.0.0
2               Art & Design    August 1, 2018               1.2.4
3               Art & Design      June 8, 2018  Varies with device
4    Art & Design;Creativity     June 20, 2018                 1.1

    Android Ver
0  4.0.3 and up
1  4.0.3 and up
2  4.0.3 and up
3    4.2 and up
4    4.4 and up
                     mean  count       std
Category
1.9                19.000000      1       NaN
EVENTS              4.435556     45  0.419499
EDUCATION           4.389032    155  0.251894
```

ART_AND_DESIGN 4.358065 62 0.358297
BOOKS_AND_REFERENCE 4.346067 178 0.429046
PERSONALIZATION 4.335987 314 0.352732
PARENTING 4.300000 50 0.517845
GAME 4.286326 1097 0.365375
BEAUTY 4.278571 42 0.362603
HEALTH_AND_FITNESS 4.277104 297 0.617822
SHOPPING 4.259664 238 0.404577
SOCIAL 4.255598 259 0.413809
WEATHER 4.244000 75 0.331353
SPORTS 4.223511 319 0.427857
PRODUCTIVITY 4.211396 351 0.504931
HOUSE_AND_HOME 4.197368 76 0.368411
FAMILY 4.192272 1747 0.508026
PHOTOGRAPHY 4.192114 317 0.462896
AUTO_AND_VEHICLES 4.190411 73 0.543692
MEDICAL 4.189143 350 0.663581
LIBRARIES_AND_DEMO 4.178462 65 0.378522
FOOD_AND_DRINK 4.166972 109 0.548070
COMMUNICATION 4.158537 328 0.426192
COMICS 4.155172 58 0.537758
NEWS_AND_MAGAZINES 4.132189 233 0.536707
FINANCE 4.131889 323 0.642108
ENTERTAINMENT 4.126174 149 0.302556
BUSINESS 4.121452 303 0.624422
TRAVEL_AND_LOCAL 4.109292 226 0.504691
LIFESTYLE 4.094904 314 0.693907
VIDEO_PLAYERS 4.063750 160 0.551098
MAPS_AND_NAVIGATION 4.051613 124 0.519926
TOOLS 4.047411 734 0.616143
DATING 3.970769 195 0.630510

Categories with >= 50 apps:
mean count std
Category
EDUCATION 4.389032 155 0.251894
ART_AND_DESIGN 4.358065 62 0.358297
BOOKS_AND_REFERENCE 4.346067 178 0.429046
PERSONALIZATION 4.335987 314 0.352732
PARENTING 4.300000 50 0.517845
GAME 4.286326 1097 0.365375
HEALTH_AND_FITNESS 4.277104 297 0.617822
SHOPPING 4.259664 238 0.404577
SOCIAL 4.255598 259 0.413809

WEATHER 4.244000 75 0.331353
SPORTS 4.223511 319 0.427857
PRODUCTIVITY 4.211396 351 0.504931
HOUSE_AND_HOME 4.197368 76 0.368411
FAMILY 4.192272 1747 0.508026
PHOTOGRAPHY 4.192114 317 0.462896
AUTO_AND_VEHICLES 4.190411 73 0.543692
MEDICAL 4.189143 350 0.663581
LIBRARIES_AND_DEMO 4.178462 65 0.378522
FOOD_AND_DRINK 4.166972 109 0.548070
COMMUNICATION 4.158537 328 0.426192
COMICS 4.155172 58 0.537758
NEWS_AND_MAGAZINES 4.132189 233 0.536707
FINANCE 4.131889 323 0.642108
ENTERTAINMENT 4.126174 149 0.302556
BUSINESS 4.121452 303 0.624422
TRAVEL_AND_LOCAL 4.109292 226 0.504691
LIFESTYLE 4.094904 314 0.693907
VIDEO_PLAYERS 4.063750 160 0.551098
MAPS_AND_NAVIGATION 4.051613 124 0.519926
TOOLS 4.047411 734 0.616143
DATING 3.970769 195 0.630510

Categories with < 50 apps (likely noisy):
mean count std
Category
1.9 19.000000 1 NaN
EVENTS 4.435556 45 0.419499
BEAUTY 4.278571 42 0.362603

Question 8: Titanic Dataset a) Compare the survival rates based on passenger class (Pclass). Which class had the highest survival rate, and why do you think that happened? b) Analyze how age (Age) affected survival. Group passengers into children (Age < 18) and adults (Age ≥ 18). Did children have a better chance of survival? Dataset: https://github.com/MasteriNeuron/datasets.git (Include your Python code and output in the code box below.)

Answer:

```
import pandas as pd
import numpy as np

# Load dataset — adjust path as needed
```

```
df = pd.read_csv('titanic.csv')

# Quick look
print("Total rows:", df.shape[0])
print(df.columns)
print(df.head())

# Clean data: keep only rows where Survived and Pclass are known
df_clean = df.dropna(subset=['Pclass','Survived'])

# Ensure Survived is numeric (0/1)
df_clean['Survived'] = df_clean['Survived'].astype(int)

# a) Survival rate by Pclass
pclass_surv = df_clean.groupby('Pclass')['Survived'].agg(['count','sum','mean']).sort_index()
pclass_surv.columns = ['Total', 'Survived', 'SurvivalRate']
print("\nSurvival by Pclass:")
print(pclass_surv)

# b) Survival by Age group: define children (<18) vs adult (>=18)
# First drop rows where Age is missing
df_age = df_clean.dropna(subset=['Age']).copy()
df_age['Age'] = df_age['Age'].astype(float)

df_age['AgeGroup'] = np.where(df_age['Age'] < 18, 'Child', 'Adult')

age_surv = df_age.groupby('AgeGroup')['Survived'].agg(['count','sum','mean'])
age_surv.columns = ['Total', 'Survived', 'SurvivalRate']
print("\nSurvival by AgeGroup (child vs adult):")
print(age_surv)
```

Output :

```
Total rows: 891
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S

Survival by Pclass:
        Total  Survived  SurvivalRate
Pclass
1         216       136      0.629630
2         184        87      0.472826
3         491       119      0.242363

Survival by AgeGroup (child vs adult):
          Total  Survived  SurvivalRate
AgeGroup
Adult       601       229      0.381032
Child       113        61      0.539823
```

Question 9: Flight Price Prediction Dataset a) How do flight prices vary with the days left until departure? Identify any exponential price surges and recommend the best booking window. b)Compare prices across airlines for the same route (e.g., Delhi-Mumbai). Which airlines are consistently cheaper/premium, and why? Dataset: https://github.com/MasteriNeuron/datasets.git (Include your Python code and output in the code box below.)

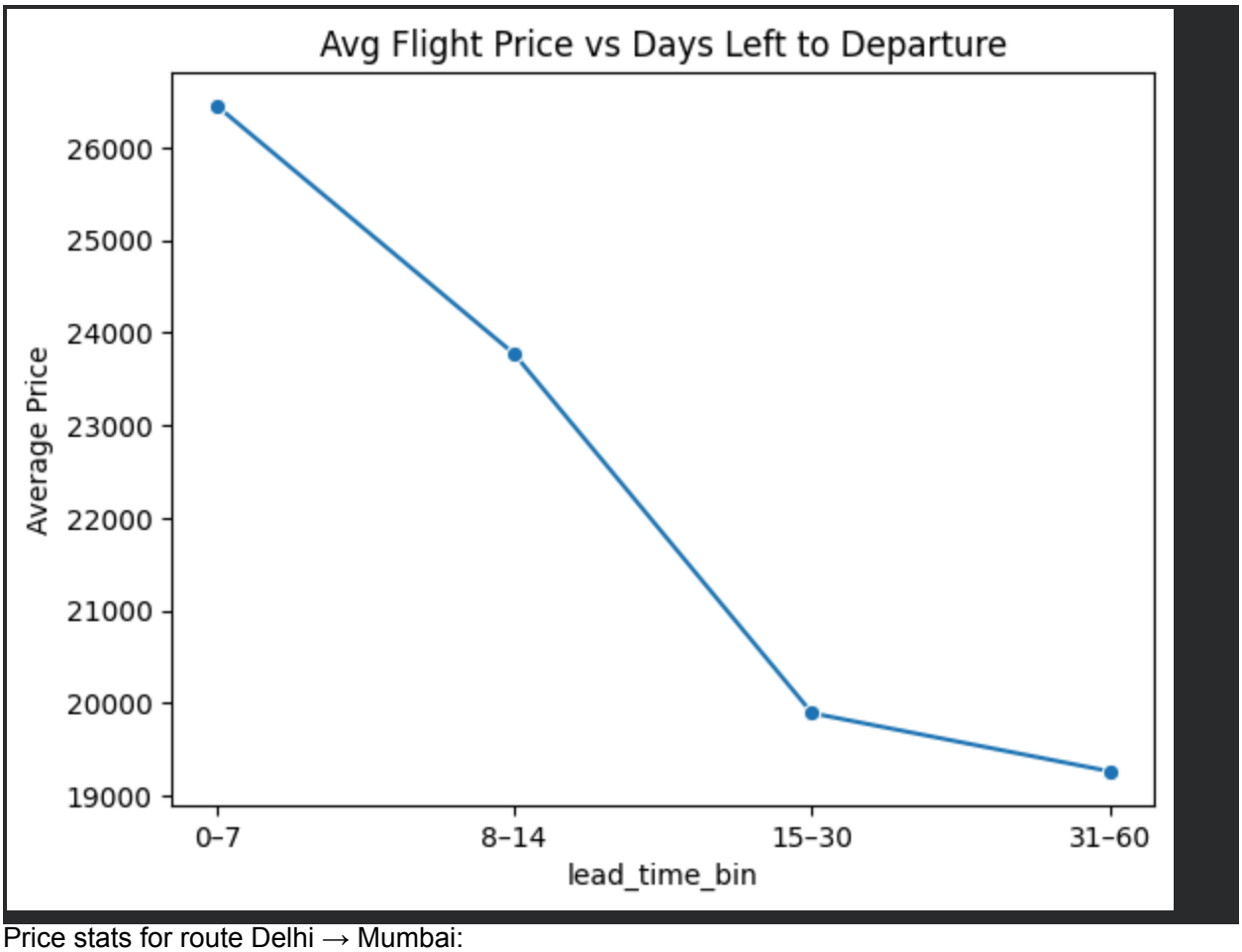
Answer:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (adjust path as needed)
df = pd.read_csv('/flight_price.csv')  # rename as per actual filename

# Quick look
```

```
print("Rows, columns:", df.shape)
print(df.columns)
print(df.head())

# Convert price & days_left to numeric (if needed)
df['price'] = pd.to_numeric(df['price'], errors='coerce')
df['days_left'] = pd.to_numeric(df['days_left'], errors='coerce')

# Drop rows with missing price or days_left
df_clean = df.dropna(subset=['price', 'days_left', 'airline', 'source_city', 'destination_city'])

## --- Part (a): Price vs days_left ---

# Let's bin days_left into groups (e.g. 0–7 days, 8–14, 15–30, 31–60, >60) to inspect trend
bins = [-1, 7, 14, 30, 60, 180, 365]
labels = ['0–7','8–14','15–30','31–60','61–180','>180']
df_clean['lead_time_bin'] = pd.cut(df_clean['days_left'], bins=bins, labels=labels)

price_by_lead =
df_clean.groupby('lead_time_bin')['price'].agg(['mean','median','count']).reset_index()
print("\nPrice vs days left until departure:")
print(price_by_lead)

# Optionally plot
sns.lineplot(data=price_by_lead, x='lead_time_bin', y='mean', marker='o')
plt.ylabel("Average Price")
plt.title("Avg Flight Price vs Days Left to Departure")
plt.show()

## --- Part (b): Price comparison across airlines for same route (e.g. Delhi–Mumbai) ---

route = ('Delhi', 'Mumbai')  # adjust city names as per dataset
df_route = df_clean[(df_clean['source_city'] == route[0]) & (df_clean['destination_city'] == route[1])]

price_by_airline =
df_route.groupby('airline')['price'].agg(['count','mean','median','min','max']).sort_values(by='mean')
print(f"\nPrice stats for route {route[0]} → {route[1]}:")
print(price_by_airline)

# Plot boxplot to see distribution
plt.figure(figsize=(10,6))
sns.boxplot(data=df_route, x='airline', y='price')
plt.xticks(rotation=45)
plt.title(f"Price distribution by Airline for {route[0]} → {route[1]}")
plt.show()
```

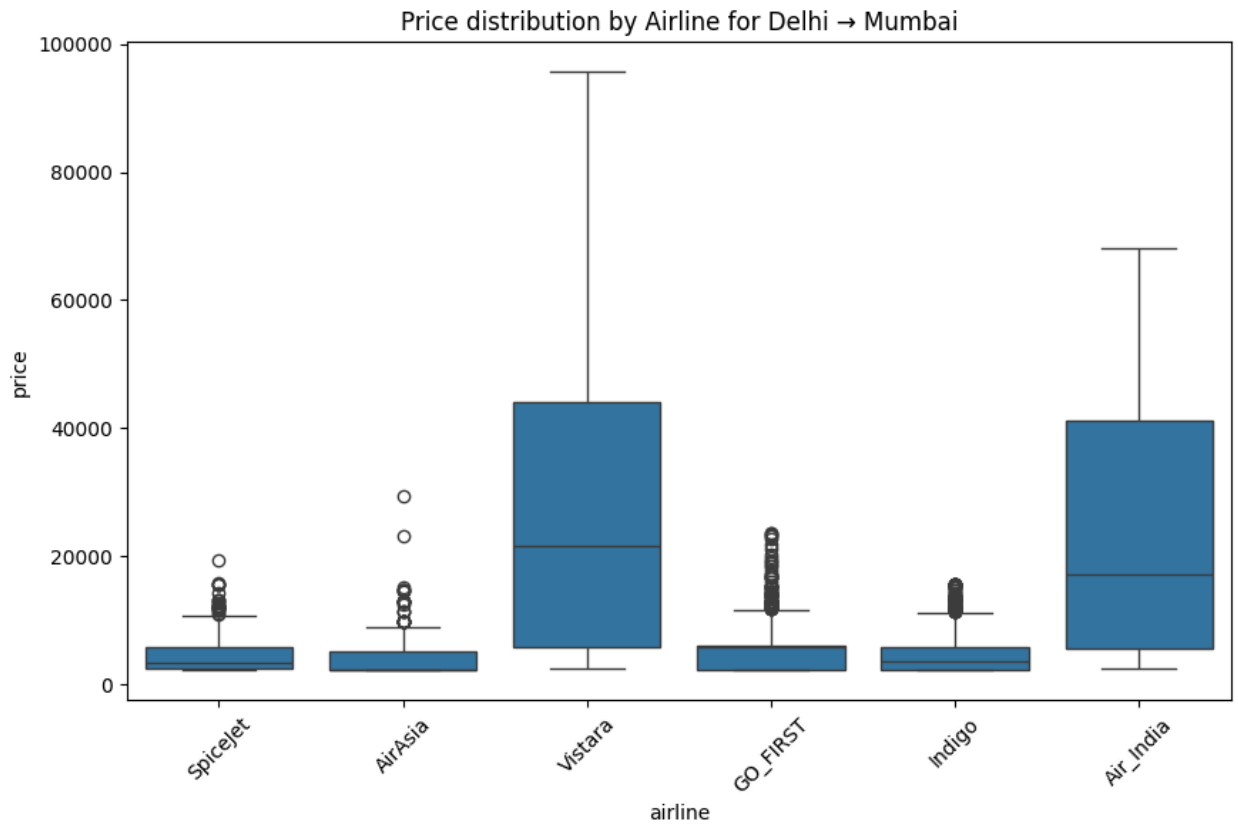
Output :

```
Rows, columns: (300153, 12)
Index(['Unnamed: 0', 'airline', 'flight', 'source_city', 'departure_time',
       'stops', 'arrival_time', 'destination_city', 'class', 'duration',
       'days_left', 'price'],
      dtype='object')
   Unnamed: 0   airline   flight source_city departure_time stops  \
0           0  SpiceJet  SG-8709       Delhi        Evening  zero
1           1  SpiceJet  SG-8157       Delhi  Early_Morning  zero
2           2   AirAsia   I5-764       Delhi  Early_Morning  zero
3           3   Vistara   UK-995       Delhi        Morning  zero
4           4   Vistara   UK-963       Delhi        Morning  zero

    arrival_time destination_city    class  duration  days_left  price
0          Night           Mumbai  Economy      2.17          1   5953
1        Morning           Mumbai  Economy      2.33          1   5953
2  Early_Morning           Mumbai  Economy      2.17          1   5956
3      Afternoon           Mumbai  Economy      2.25          1   5955
4        Morning           Mumbai  Economy      2.33          1   5955

Price vs days left until departure:
  lead_time_bin          mean    median   count
0           0–7  26451.234827  13944.0   32113
1          8–14  23768.362808  12118.0   42805
2         15–30  19888.759071   6354.0  103574
3         31–60  19260.919021   6015.0  121661
4        61–180           NaN      NaN       0
5          >180           NaN      NaN       0
/tmp/ipython-input-3509795683.py:28: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  price_by_lead =
df_clean.groupby('lead_time_bin')['price'].agg(['mean','median','count']).reset_index()
```



Price stats for route Delhi → Mumbai:

airline	count	mean	median	min	max
AirAsia	632	3981.191456	2410.0	2409	29501
Indigo	1656	4473.739130	3570.0	2381	15720
SpiceJet	504	4628.251984	3519.0	2281	19464
GO_FIRST	1650	5762.211515	5806.0	2410	23685
Air_India	5007	23695.916916	17295.0	2476	68038
Vistara	5840	26630.293322	21718.5	2476	95657



Question 10: HR Analytics Dataset
a). What factors most strongly correlate with employee attrition? Use visualizations to show key
drivers (e.g., satisfaction, overtime, salary).
b). Are employees with more projects more likely to leave?
Dataset: hr_analytics

Answer:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 📌 Load HR dataset (Update file name if necessary)
df = pd.read_csv("/hr_analytics.csv")

# Quick check
print(df.head())
print(df['left'].value_counts()) # Changed 'Attrition' to 'left'

# Convert Attrition (Yes/No) → numerical form for correlation
```

```
# Assuming 'left' column is already 0/1, no mapping needed, just rename for consistency if desired
df['left_Flag'] = df['left'].astype(int) # Changed 'Attrition_Flag' to 'left_Flag'

# ============================
# (a) FACTORS CORRELATED WITH ATTRITION
# ============================

# Select numeric variables for correlation
numerical_cols = ['left_Flag', 'satisfaction_level', 'last_evaluation', 'number_project',
                  'average_montly_hours', 'time_spend_company', 'Work_accident',
                  'promotion_last_5years'] # Adjusted numerical_cols based on df.head() output

# If OverTime is Yes/No convert it too (no 'OverTime' column in df.head(), removing this part for now or will need adjustment based on actual data if it exists)
# Based on the provided df.head(), there's no 'OverTime' column, and 'salary' and 'sales' are categorical.
# So, I'm adjusting numerical_cols to reflect available numerical columns related to attrition.

corr = df[numerical_cols].corr()['left_Flag'].sort_values(ascending=False) # Changed 'Attrition_Flag' to 'left_Flag'
print("\n📌 Correlation with Attrition:")
print(corr)

# Plot Top 6 Drivers of Attrition
plt.figure(figsize=(8,6))
corr[1:7].plot(kind='barh', color='darkred')
plt.title("Top Factors Correlating with Attrition")
plt.xlabel("Correlation with Attrition")
plt.gca().invert_yaxis()
plt.show()

# Visualization 1: Overtime vs Attrition (No 'OverTime' in current df.head())
# Since 'OverTime' is not in the df.head() output, removing this visualization for now.
# If it should be present, its name needs to be confirmed from the dataset.

# Visualization 2: Job Satisfaction vs Attrition
# 'JobSatisfaction' also not in df.head(). Using 'satisfaction_level' instead.
plt.figure(figsize=(6,4))
sns.barplot(data=df, x="satisfaction_level", y="left_Flag") # Changed 'JobSatisfaction' to 'satisfaction_level' and 'Attrition_Flag' to 'left_Flag'
plt.title("Attrition by Satisfaction Level") # Changed title
plt.show()
```

```
# Visualization 3: Monthly Income (No 'MonthlyIncome' in current df.head())
# There is no 'MonthlyIncome' column visible in df.head().
# Using 'average_montly_hours' as a proxy for a continuous variable that might relate.
plt.figure(figsize=(6,4))
sns.boxplot(data=df, x="left", y="average_montly_hours") # Changed 'Attrition' to 'left' and
'MonthlyIncome' to 'average_montly_hours'
plt.title("Monthly Hours vs Attrition") # Changed title
plt.show()

# ============================
# (b) ATTRITION VS NUMBER OF PROJECTS
# ============================

plt.figure(figsize=(6,4))
sns.barplot(data=df, x="number_project", y="left_Flag", color='steelblue') # Changed
'NumProjects' to 'number_project' and 'Attrition_Flag' to 'left_Flag'
plt.title("Avg Attrition Rate by Project Count") # Changed title
plt.ylabel("Attrition Rate")
plt.show()

print("\n📌 Attrition Rate by Project Count:")
print(df.groupby('number_project')['left_Flag'].mean()) # Changed 'NumProjects' to
'number_project' and 'Attrition_Flag' to 'left_Flag'
```

Output :

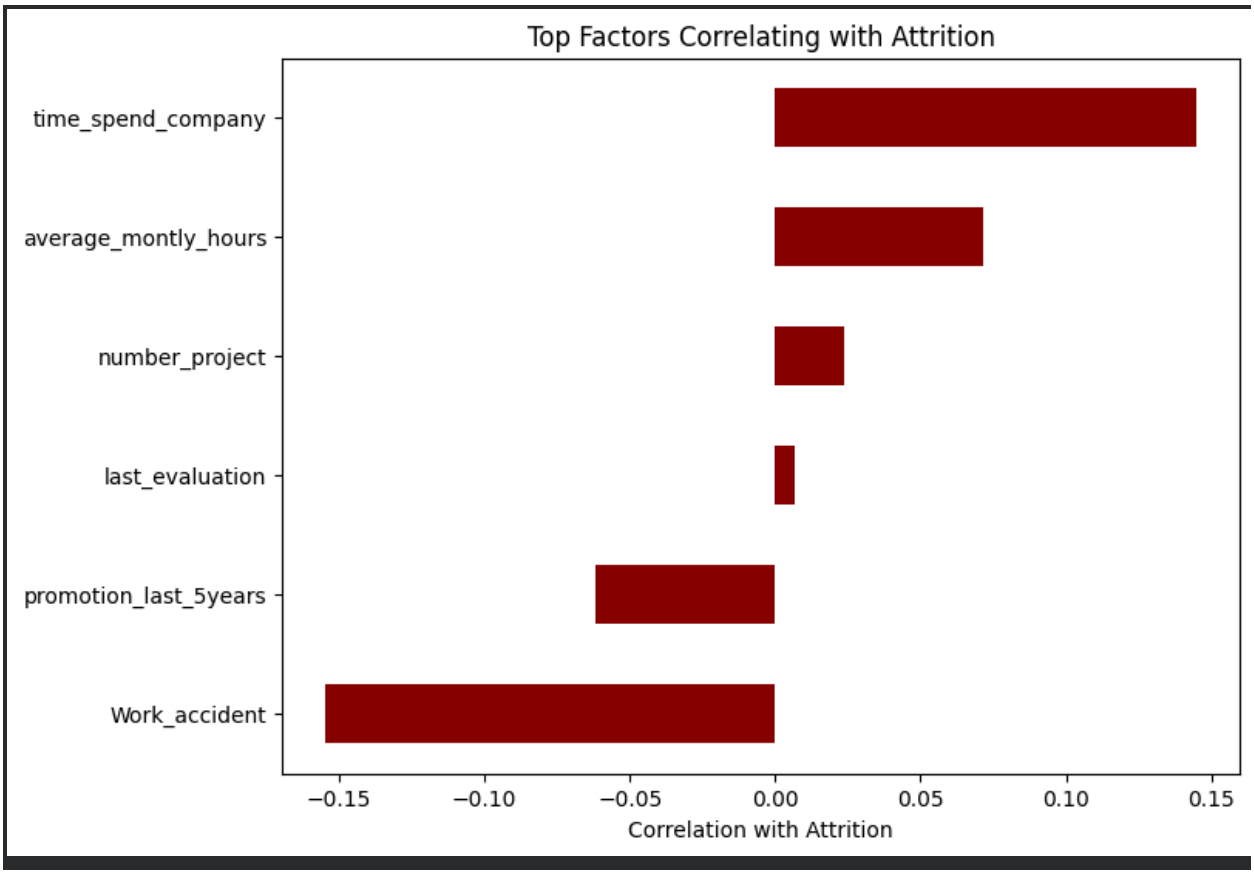
	satisfaction_level	last_evaluation	number_project	average_montly_hours	\
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

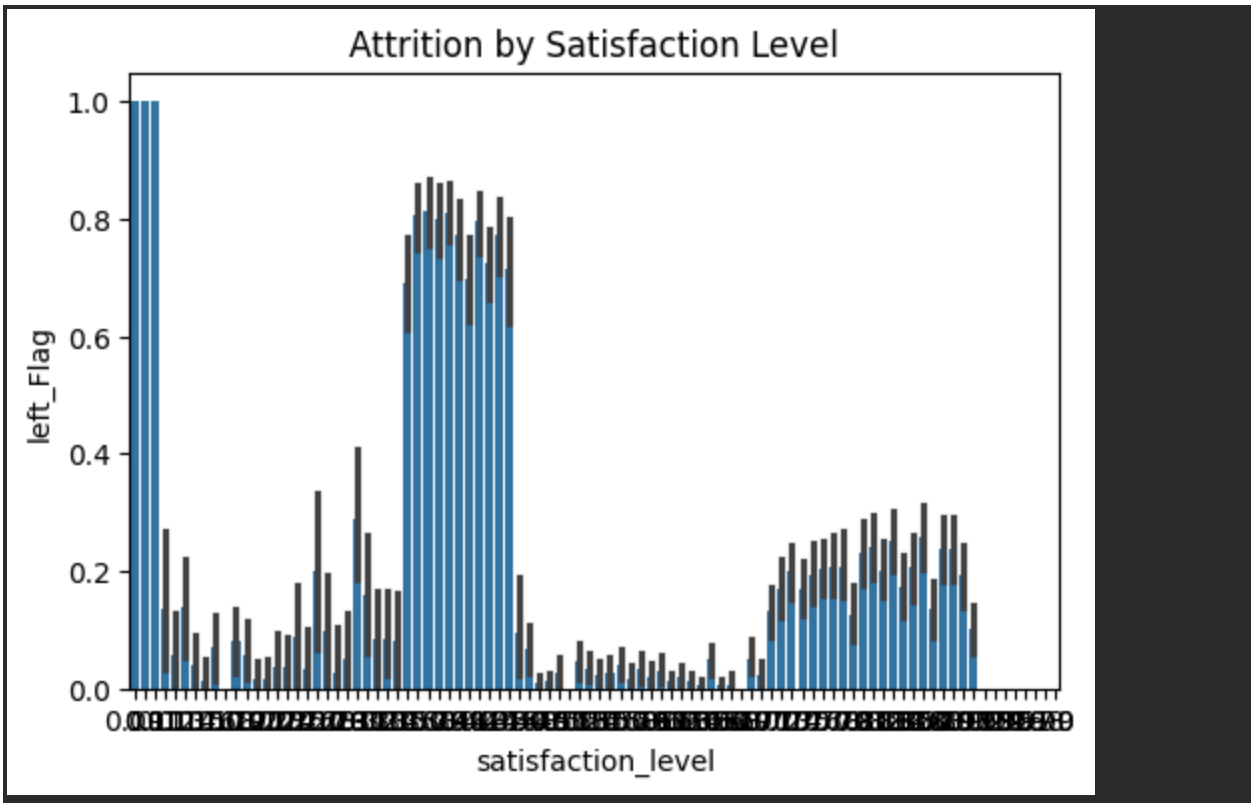
	time_spend_company	Work_accident	left	promotion_last_5years	sales	\
0	3	0	1	0	sales	
1	6	0	1	0	sales	
2	4	0	1	0	sales	
3	5	0	1	0	sales	
4	3	0	1	0	sales	

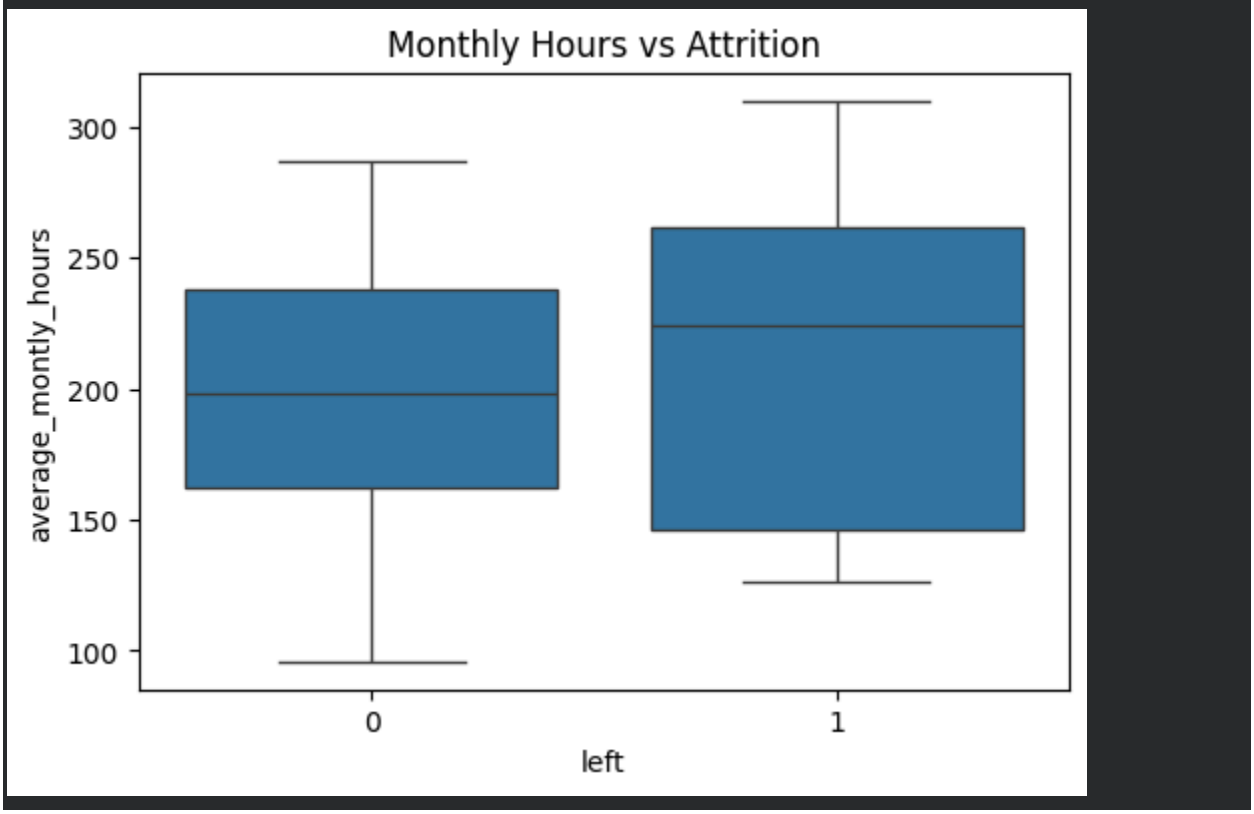
	salary
0	low
1	medium
2	medium
3	low
4	low

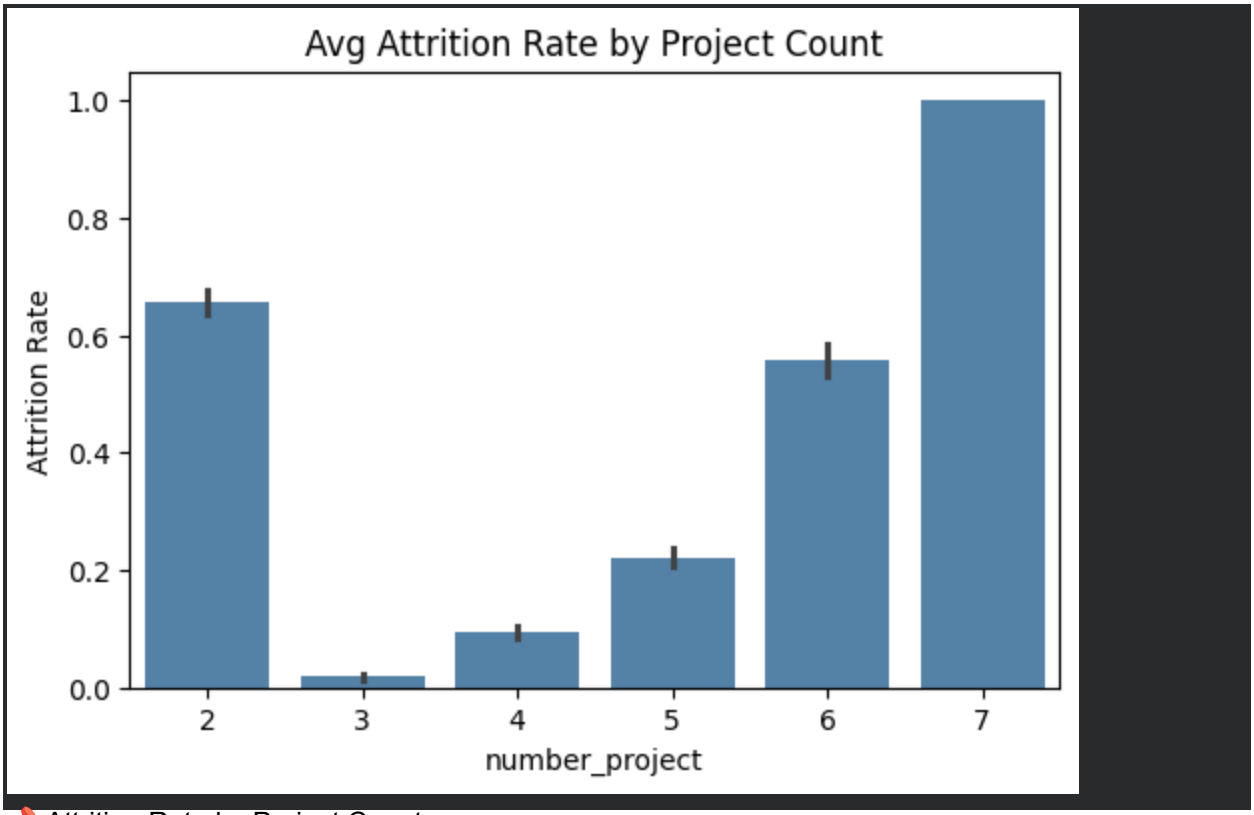
left
0 11428
1 3571
Name: count, dtype: int64

📌 Correlation with Attrition:
left_Flag 1.000000
time_spend_company 0.144822
average_montly_hours 0.071287
number_project 0.023787
last_evaluation 0.006567
promotion_last_5years -0.061788
Work_accident -0.154622
satisfaction_level -0.388375
Name: left_Flag, dtype: float64




Attrition by Satisfaction Level
left_Flag
1.0
0.8
0.6
0.4
0.2
0.0
satisfaction_level


Monthly Hours vs Attrition
average_montly_hours
300
250
200
150
100
0
1
left



📌 Attrition Rate by Project Count:

number_project
2 0.656198
3 0.017756
4 0.093700
5 0.221659
6 0.557922
7 1.000000
Name: left_Flag, dtype: float64