Marriage Trends in India: Love vs. Arranged

About Dataset

This dataset explores marriage trends in India, comparing love marriages and arranged marriages across various demographic, social, and economic factors. I, capturing key aspects such as age at marriage, caste and religion dynamics, parental approval, dowry exchange, marital satisfaction, divorce rates, income levels, and urban-rural differences.

The dataset aims to provide valuable insights into changing marriage patterns, the role of tradition vs. modernity, and their impact on marital outcomes. Researchers, sociologists, and data analysts can use this dataset to study relationship trends, predict marriage success, and analyze

social influences on marriage in India.


LOVE MARRIAGES

import necessary liberary

In [214…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [215…
```
df=pd.read_csv("marriage_data_india.csv")
```

In [216…
```
df.head()
```

Out[216...

	ID	Marriage_Type	Age_at_Marriage	Gender	Education_Level	Caste_Match	Religion	Parental_Approval	Urban_Rural	Dowry_Exchang
0	1	Love	23	Male	Graduate	Different	Hindu	No	Urban	N
1	2	Love	28	Female	School	Same	Hindu	Yes	Rural	Y
2	3	Arranged	39	Male	Postgraduate	Same	Muslim	Yes	Rural	N
3	4	Arranged	26	Female	School	Different	Hindu	Yes	Urban	Y
4	5	Love	32	Female	Graduate	Same	Hindu	Partial	Rural	Y

In [217...

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   ID                    10000 non-null  int64 
 1   Marriage_Type         10000 non-null  object
 2   Age_at_Marriage       10000 non-null  int64 
 3   Gender                10000 non-null  object
 4   Education_Level       10000 non-null  object
 5   Caste_Match           10000 non-null  object
 6   Religion              10000 non-null  object
 7   Parental_Approval     10000 non-null  object
 8   Urban_Rural           10000 non-null  object
 9   Dowry_Exchanged       10000 non-null  object
 10  Marital_Satisfaction  10000 non-null  object
 11  Divorce_Status        10000 non-null  object
 12  Children_Count        10000 non-null  int64 
 13  Income_Level          10000 non-null  object
 14  Years_Since_Marriage  10000 non-null  int64 
 15  Spouse_Working        10000 non-null  object
 16  Inter-Caste           10000 non-null  object
 17  Inter-Religion        10000 non-null  object
dtypes: int64(4), object(14)
memory usage: 1.4+ MB
```

In [218…

```
df.isnull().sum()
```

Out[218…

```
ID                    0
Marriage_Type         0
Age_at_Marriage       0
Gender                0
Education_Level       0
Caste_Match           0
Religion              0
Parental_Approval     0
Urban_Rural           0
Dowry_Exchanged       0
Marital_Satisfaction  0
Divorce_Status        0
Children_Count        0
Income_Level          0
Years_Since_Marriage  0
Spouse_Working        0
Inter-Caste           0
Inter-Religion        0
dtype: int64
```

In [219…

```
df.describe()
```

Out[219…

	ID	Age_at_Marriage	Children_Count	Years_Since_Marriage
count	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	28.503800	2.508800	24.973800
std	2886.89568	6.279564	1.695467	14.054838
min	1.00000	18.000000	0.000000	1.000000
25%	2500.75000	23.000000	1.000000	13.000000
50%	5000.50000	29.000000	3.000000	25.000000
75%	7500.25000	34.000000	4.000000	37.000000
max	10000.00000	39.000000	5.000000	49.000000

In [220…
```
df.dropna(inplace=True)
```

In [221…
```
# Unique Values Count
print("\n 🔹 Unique Values per Column:\n", df.nunique())
```

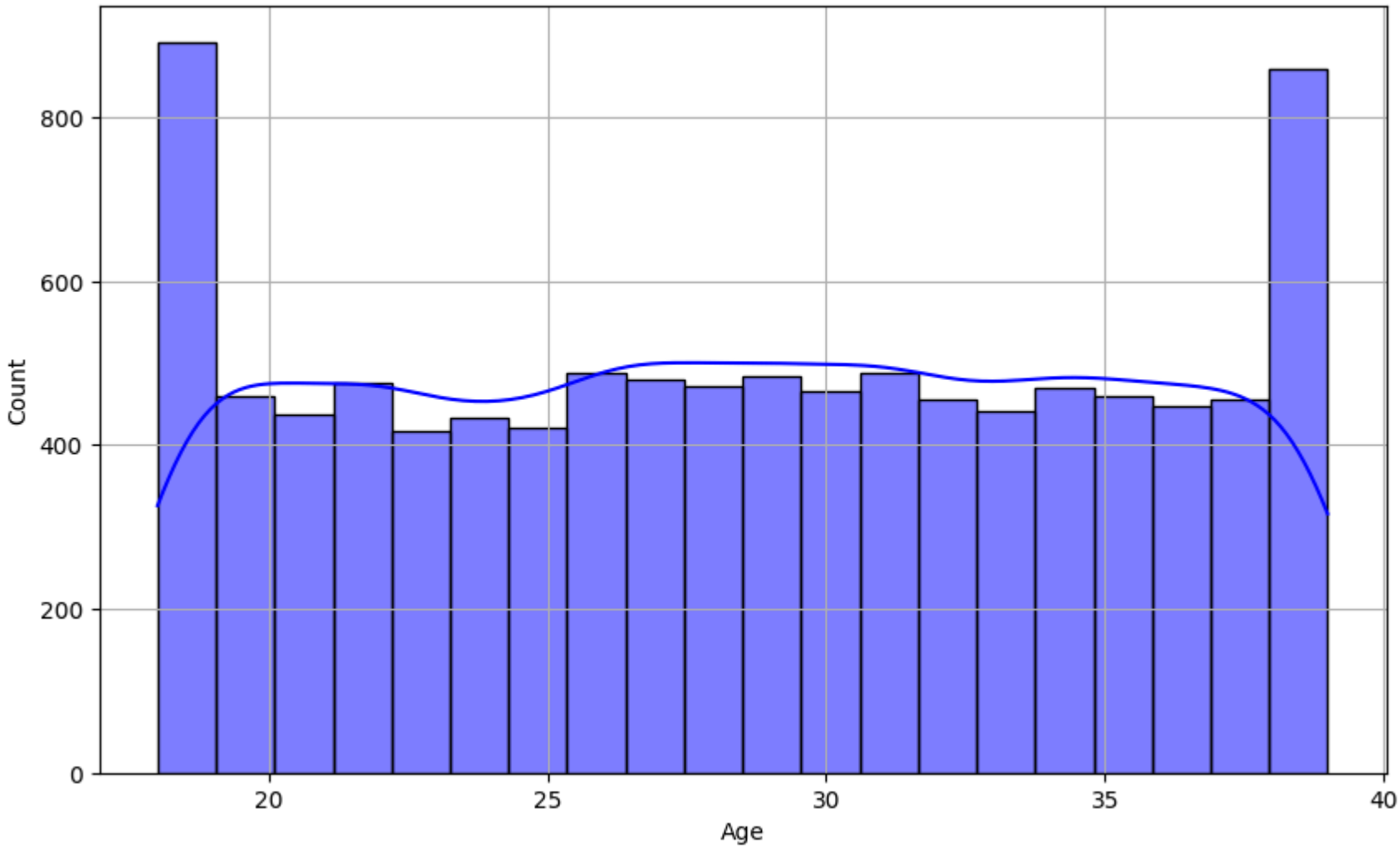
```
 🔹 Unique Values per Column:
 ID                     10000
Marriage_Type               2
Age_at_Marriage            22
Gender                      2
Education_Level             4
Caste_Match                 2
Religion                    5
Parental_Approval           3
Urban_Rural                 2
Dowry_Exchanged             3
Marital_Satisfaction        3
Divorce_Status              2
Children_Count              6
Income_Level                3
Years_Since_Marriage       49
Spouse_Working              2
Inter-Caste                 2
Inter-Religion              2
dtype: int64
```

In [222…
```
df['Religion'].nunique()
```

Out[222… 5

Distribution of Age at Marriage

In [223…
```
plt.figure(figsize=(10,6))
sns.histplot(df['Age_at_Marriage'],bins=20,kde=True,color="blue")
plt.xlabel("Age")
plt.ylabel("Count")
plt.grid(True)
plt.show()
```



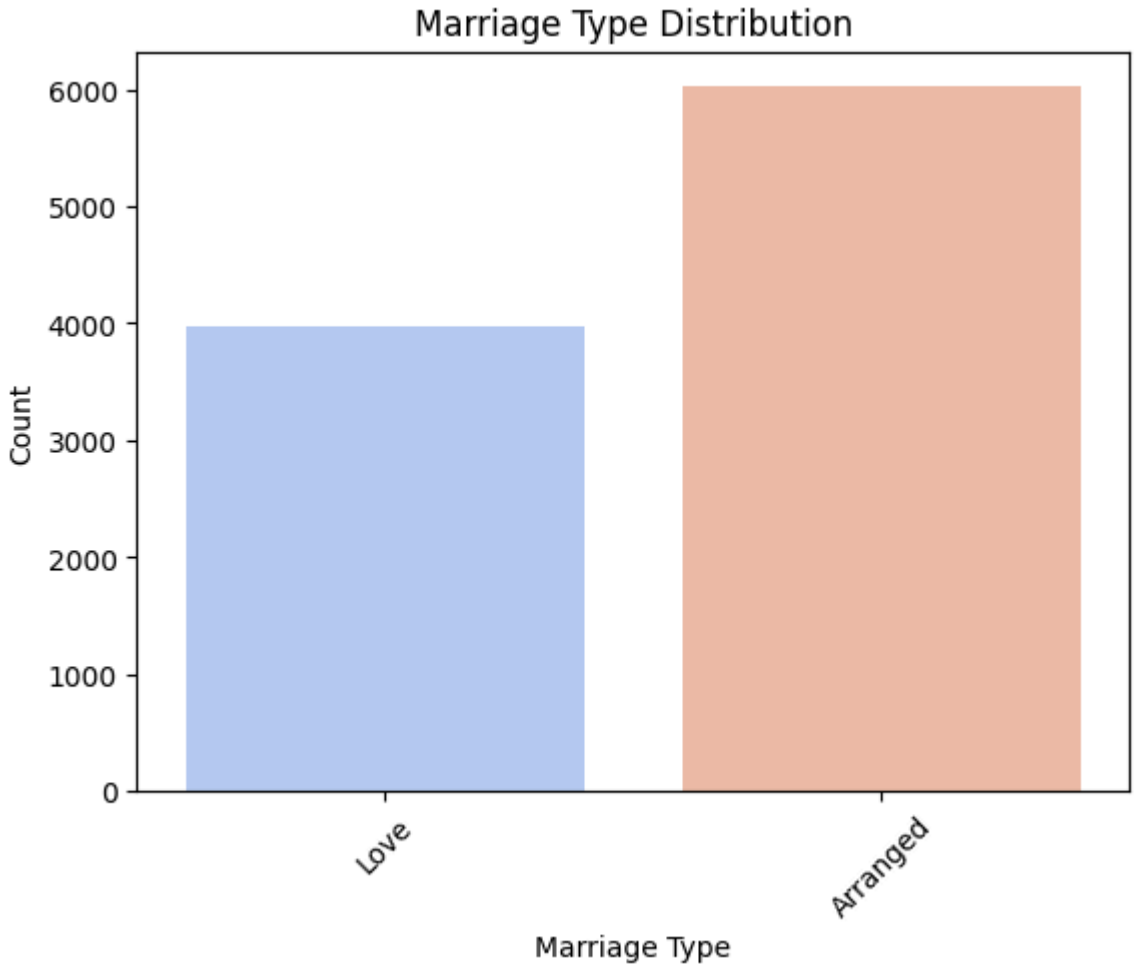
Marriage Type Distribution

In [224…

```
sns.countplot(data=df,x="Marriage_Type",palette="coolwarm")
plt.title("Marriage Type Distribution")
plt.xlabel("Marriage Type")
plt.ylabel("Count")
```

```
plt.xticks(rotation=45)
plt.show()
```

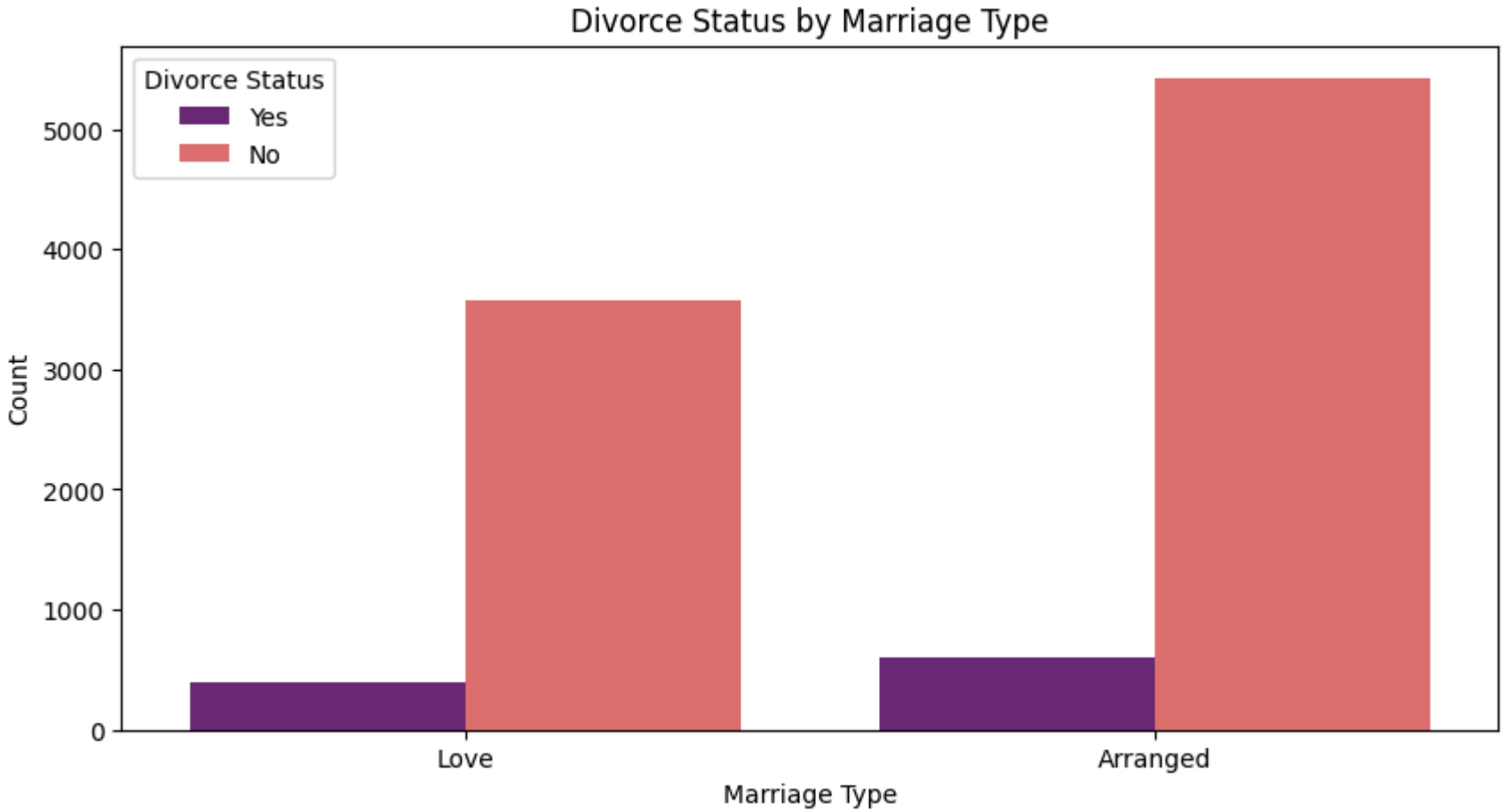
```
C:\Users\hi\AppData\Local\Temp\ipykernel_52488\2557899203.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(data=df,x="Marriage_Type",palette="coolwarm")
```



Divorce Status Count by Marriage Type

In [225…

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x="Marriage_Type", hue="Divorce_Status", palette="magma")
plt.title("Divorce Status by Marriage Type")
plt.xlabel("Marriage Type")
plt.ylabel("Count")
plt.legend(title="Divorce Status")
plt.show()
```

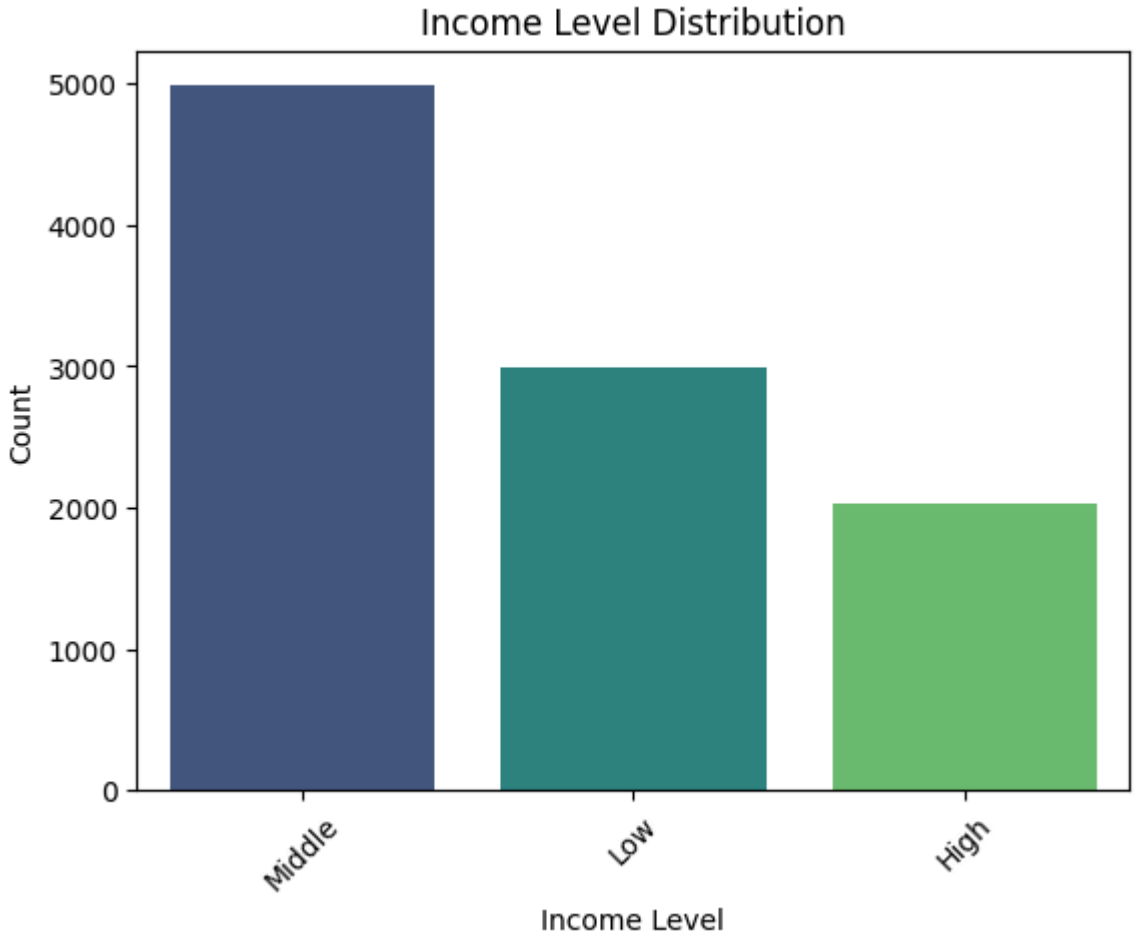


Income Level Distribution

In [226…

```
sns.countplot(data=df,x="Income_Level",palette="viridis",order=df["Income_Level"].value_counts().index)
plt.title("Income Level Distribution")
plt.xlabel("Income Level")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

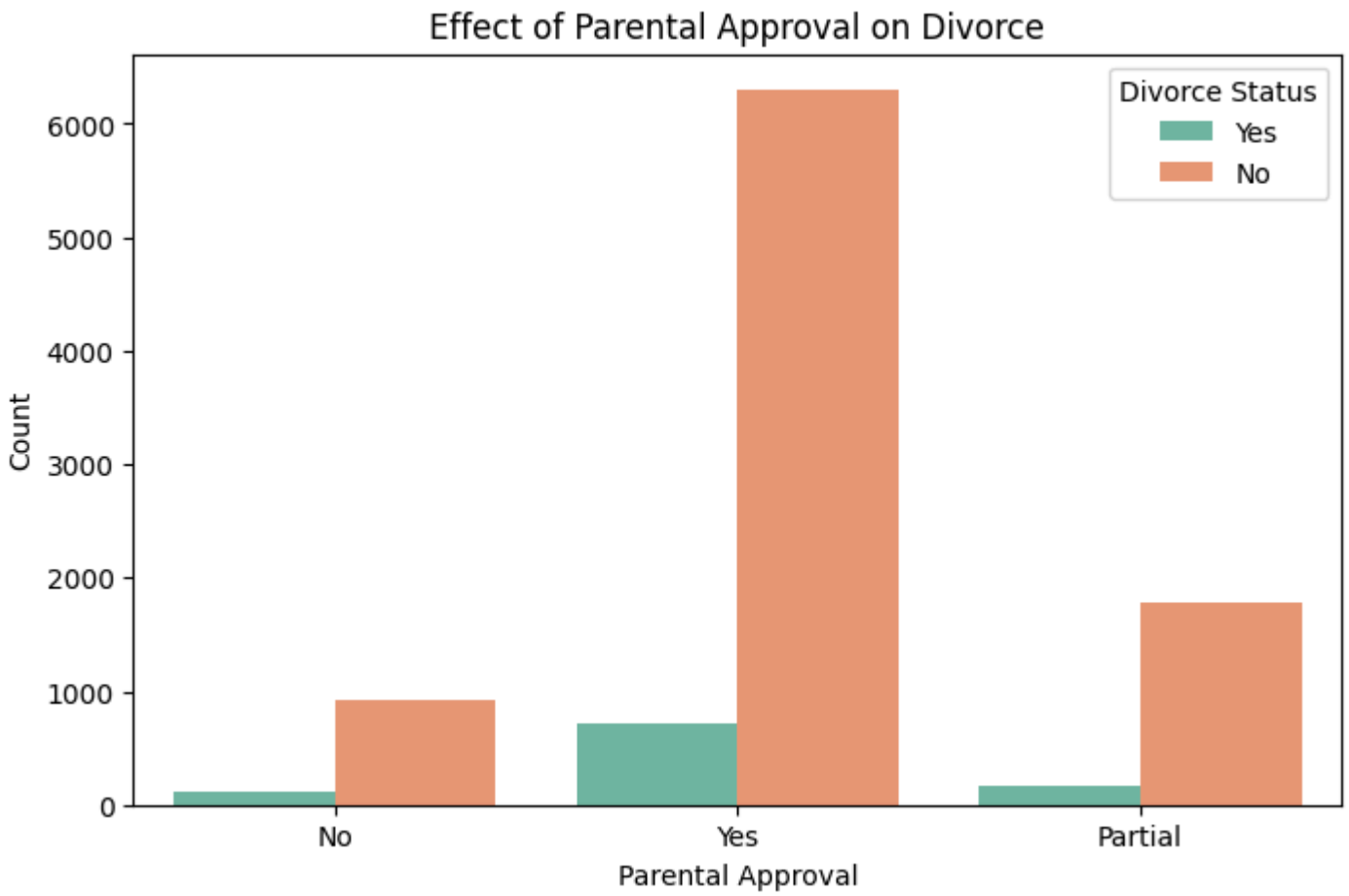
```
C:\Users\hi\AppData\Local\Temp\ipykernel_52488\857970884.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(data=df,x="Income_Level",palette="viridis",order=df["Income_Level"].value_counts().index)
```



Does Parental Approval Affect Divorce?

In [227…

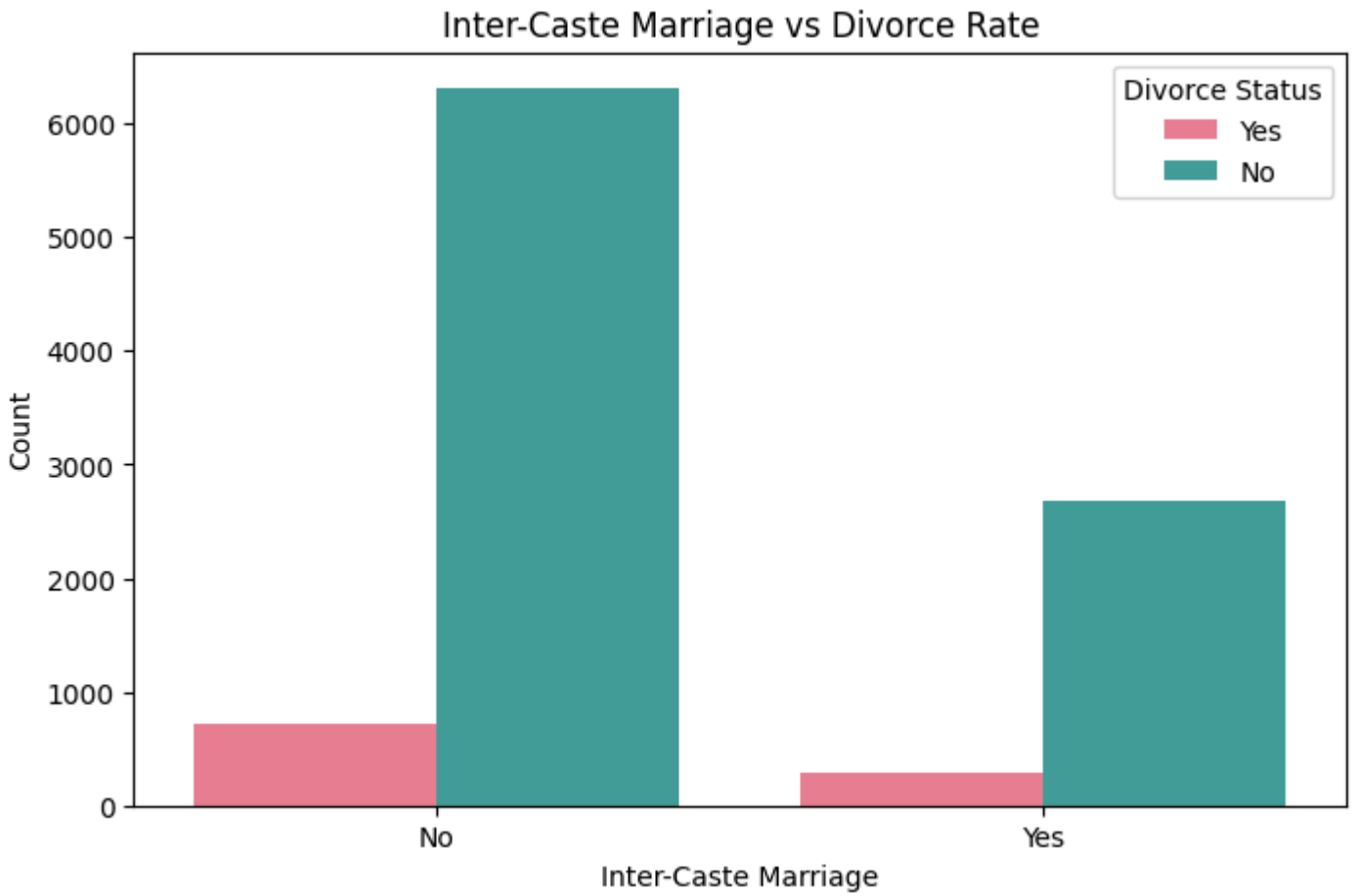
```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x="Parental_Approval", hue="Divorce_Status", palette="Set2")
plt.title("Effect of Parental Approval on Divorce")
plt.xlabel("Parental Approval")
plt.ylabel("Count")
plt.legend(title="Divorce Status")
plt.show()
```



Divorce Rate in Inter-Caste vs Same-Caste Marriages

In [228…

```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x="Inter-Caste", hue="Divorce_Status", palette="husl")
plt.title("Inter-Caste Marriage vs Divorce Rate")
plt.xlabel("Inter-Caste Marriage")
plt.ylabel("Count")
plt.legend(title="Divorce Status")
plt.show()
```

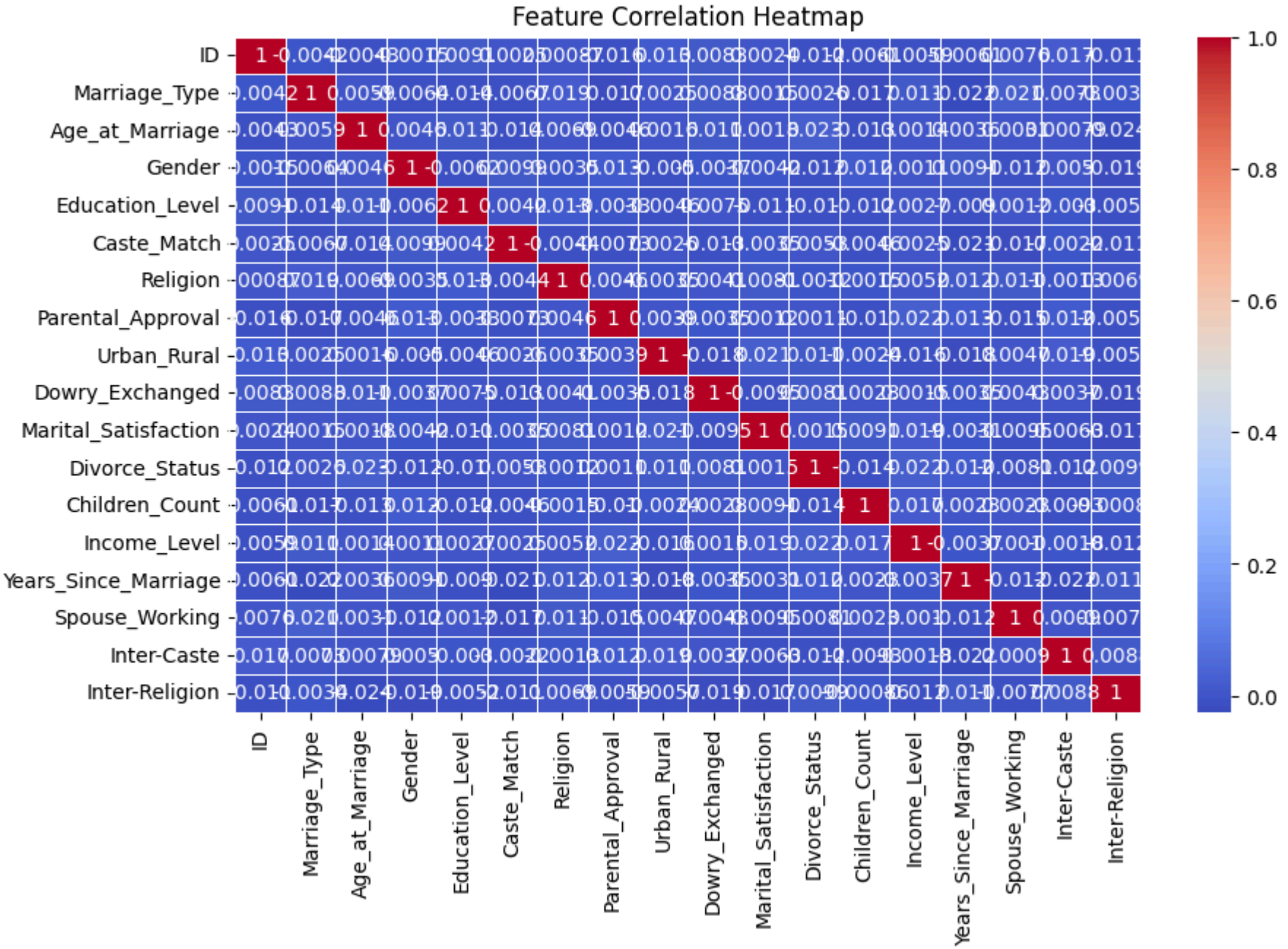


Correlation Heatmap (Numerical Features)

In [229…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load dataset
#df = pd.read_csv("marriage_dataset.csv")  # Replace with your dataset

# Convert categorical columns to numerical values
```

```
label_encoders = {}
for col in df.select_dtypes(include=["object"]).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Plot correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```


Feature Correlation Heatmap
ID
Marriage_Type
Age_at_Marriage
Gender
Education_Level
Caste_Match
Religion
Parental_Approval
Urban_Rural
Dowry_Exchanged
Marital_Satisfaction
Divorce_Status
Children_Count
Income_Level
Years_Since_Marriage
Spouse_Working
Inter-Caste
Inter-Religion
1.0
0.8
0.6
0.4
0.2
0.0

Years Since Marriage vs Marital Satisfaction

In [230…

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=df, x="Marital_Satisfaction", y="Years_Since_Marriage", palette="coolwarm")
plt.title("Years Since Marriage vs Marital Satisfaction")
plt.xlabel("Marital Satisfaction Level")
plt.ylabel("Years Since Marriage")
plt.show()
```

```
C:\Users\hi\AppData\Local\Temp\ipykernel_52488\3792648906.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.boxplot(data=df, x="Marital_Satisfaction", y="Years_Since_Marriage", palette="coolwarm")
```

In [231…

```python
# Feature Engineering: Creating a new column for Age Gap since Marriage
df["Age_Gap_Marriage"] = df["Years_Since_Marriage"] - df["Age_at_Marriage"]

# Feature: High Dowry Exchange Impact
df["High_Dowry_Impact"] = df["Dowry_Exchanged"].apply(lambda x: 1 if x == "Yes" else 0)

# Feature: Is Income High?
df["High_Income"] = df["Income_Level"].apply(lambda x: 1 if x in ["High", "Very High"] else 0)

# Feature: Is the marriage inter-caste & inter-religion?
df["Intercaste_Interreligion"] = ((df["Inter-Caste"] == "Yes") & (df["Inter-Religion"] == "Yes")).astype(int)
```

```
# Feature: Marriage Type Encoding (Categorical to Numeric)
df["Marriage_Type_Num"] = df["Marriage_Type"].map({"Arranged": 0, "Love": 1, "Love-cum-Arranged": 2})

# Display new features
df.head()
```

Out[231…

	ID	Marriage_Type	Age_at_Marriage	Gender	Education_Level	Caste_Match	Religion	Parental_Approval	Urban_Rural	Dowry_Exchang
0	1	1	23	1	0	0	1	0	1	
1	2	1	28	0	3	1	1	2	0	
2	3	0	39	1	2	1	2	2	0	
3	4	0	26	0	3	0	1	2	1	
4	5	1	32	0	0	1	1	1	0	

5 rows × 23 columns

Data Preprocessing

In [232…

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [233…

```
df=df.drop(columns=['ID'])
```

In [234…

```
#encode categorical columns
categorical_cols = df.select_dtypes(include=["object"]).columns
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
    print(df.isnull().sum())
```

In [235…
```
X=df.drop(columns=["Divorce_Status"])
y=df["Divorce_Status"]
```

In [236…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [237…
```
print(df["Divorce_Status"].isnull().sum())  # Count NaN values in target
df = df.dropna(subset=["Divorce_Status"])
df["Divorce_Status"].fillna(df["Divorce_Status"].mode()[0], inplace=True)
```

```
0
```

```
C:\Users\hi\AppData\Local\Temp\ipykernel_52488\4270387805.py:3: FutureWarning: A value is trying to be set on a copy of a DataF
rame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["Divorce_Status"].fillna(df["Divorce_Status"].mode()[0], inplace=True)
```

In [238…
```
X = df.drop(columns=["Divorce_Status"])
y = df["Divorce_Status"]

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

In [239…
```
print(df.columns)  # Check available columns
```

```
Index(['Marriage_Type', 'Age_at_Marriage', 'Gender', 'Education_Level',
       'Caste_Match', 'Religion', 'Parental_Approval', 'Urban_Rural',
       'Dowry_Exchanged', 'Marital_Satisfaction', 'Divorce_Status',
       'Children_Count', 'Income_Level', 'Years_Since_Marriage',
       'Spouse_Working', 'Inter-Caste', 'Inter-Religion', 'Age_Gap_Marriage',
       'High_Dowry_Impact', 'High_Income', 'Intercaste_Interreligion',
       'Marriage_Type_Num'],
      dtype='object')
```

In [240…
```
print(df.shape)  # Check the number of rows and columns
```

```
(10000, 22)
```

In [241…
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')  # You can change 'mean' to 'median' or 'most_frequent'
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

```
C:\Users\hi\anaconda3\Lib\site-packages\sklearn\impute\_base.py:598: UserWarning: Skipping features without any observed value
s: ['Marriage_Type_Num']. At least one non-missing value is needed for imputation with strategy='mean'.
  warnings.warn(
C:\Users\hi\anaconda3\Lib\site-packages\sklearn\impute\_base.py:598: UserWarning: Skipping features without any observed value
s: ['Marriage_Type_Num']. At least one non-missing value is needed for imputation with strategy='mean'.
  warnings.warn(
```

In [242…
```
if "Divorce_Status" in df.columns and not df["Divorce_Status"].empty:
    df["Divorce_Status"].fillna(df["Divorce_Status"].mode()[0], inplace=True)
else:
    print("Error: 'Divorce_Status' column is missing or empty!")
```

In [243…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report
```

In [244…
```
model=RandomForestClassifier(n_estimators=100,random_state=42)
model.fit(X_train,y_train)
```

Out[244…
```
RandomForestClassifier(random_state=42)
```

In [245…
```
#predictions
y_train=model.predict(X_test)
```

In [246…
```
print("Accuracy:",accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
Accuracy: 0.889
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1800
           1       0.19      0.04      0.06       200

    accuracy                           0.89      2000
   macro avg       0.55      0.51      0.50      2000
weighted avg       0.83      0.89      0.85      2000
```

In [247…
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
```

In [248…
```
models={

    "Logistic Regression":LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "svm": SVC(),
     "XGBoost": XGBClassifier()

}
```

In [249…
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')  # You can change 'mean' to 'median' or 'most_frequent'
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

In [255…
```
X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]  # Ensure labels remain aligned
X_test = X_test.dropna()
y_test = y_test.loc[X_test.index]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: