1. Import Libraries

In [13]:
```
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.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.utils import resample
from sklearn import metrics
from tqdm import tqdm
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")
```

2. Load Dataset

In [14]:
```
df = pd.read_csv("emails.csv")
print(df.shape)
print(df.head())
```

```
(5172, 3002)
  Email No.  the  to  ect  and  for  of    a  you  hou  ...  connevey  jay  \
0   Email 1    0   0    1    0    0   0    2    0    0  ...         0    0
1   Email 2    8  13   24    6    6   2  102    1   27  ...         0    0
2   Email 3    0   0    1    0    0   0    8    0    0  ...         0    0
3   Email 4    0   5   22    0    5   1   51    2   10  ...         0    0
4   Email 5    7   6   17    1    5   2   57    0    9  ...         0    0

   valued  lay  infrastructure  military  allowing  ff  dry  Prediction
0       0    0               0         0         0   0    0           0
1       0    0               0         0         0   1    0           0
2       0    0               0         0         0   0    0           0
3       0    0               0         0         0   0    0           0
4       0    0               0         0         0   1    0           0

[5 rows x 3002 columns]
```

3. Basic Information

```
In [15]: print(df.info())
         print(df.describe().T)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
None
             count      mean        std  min  25%  50%  75%    max
the         5172.0  6.640565  11.745009  0.0  0.0  3.0  8.0  210.0
to          5172.0  6.188128   9.534576  0.0  1.0  3.0  7.0  132.0
ect         5172.0  5.143852  14.101142  1.0  1.0  1.0  4.0  344.0
and         5172.0  3.075599   6.045970  0.0  0.0  1.0  3.0   89.0
for         5172.0  3.124710   4.680522  0.0  1.0  2.0  4.0   47.0
...            ...       ...        ...  ...  ...  ...  ...    ...
military    5172.0  0.006574   0.138908  0.0  0.0  0.0  0.0    4.0
allowing    5172.0  0.004060   0.072145  0.0  0.0  0.0  0.0    3.0
ff          5172.0  0.914733   2.780203  0.0  0.0  0.0  1.0  114.0
dry         5172.0  0.006961   0.098086  0.0  0.0  0.0  0.0    4.0
Prediction  5172.0  0.290023   0.453817  0.0  0.0  0.0  1.0    1.0

[3001 rows x 8 columns]
```

4. Data Preprocessing

```
In [16]: # Drop unnecessary columns
         df = df.drop("Email No.", axis=1)

         # Check for missing values
         print(df.isna().sum())
```

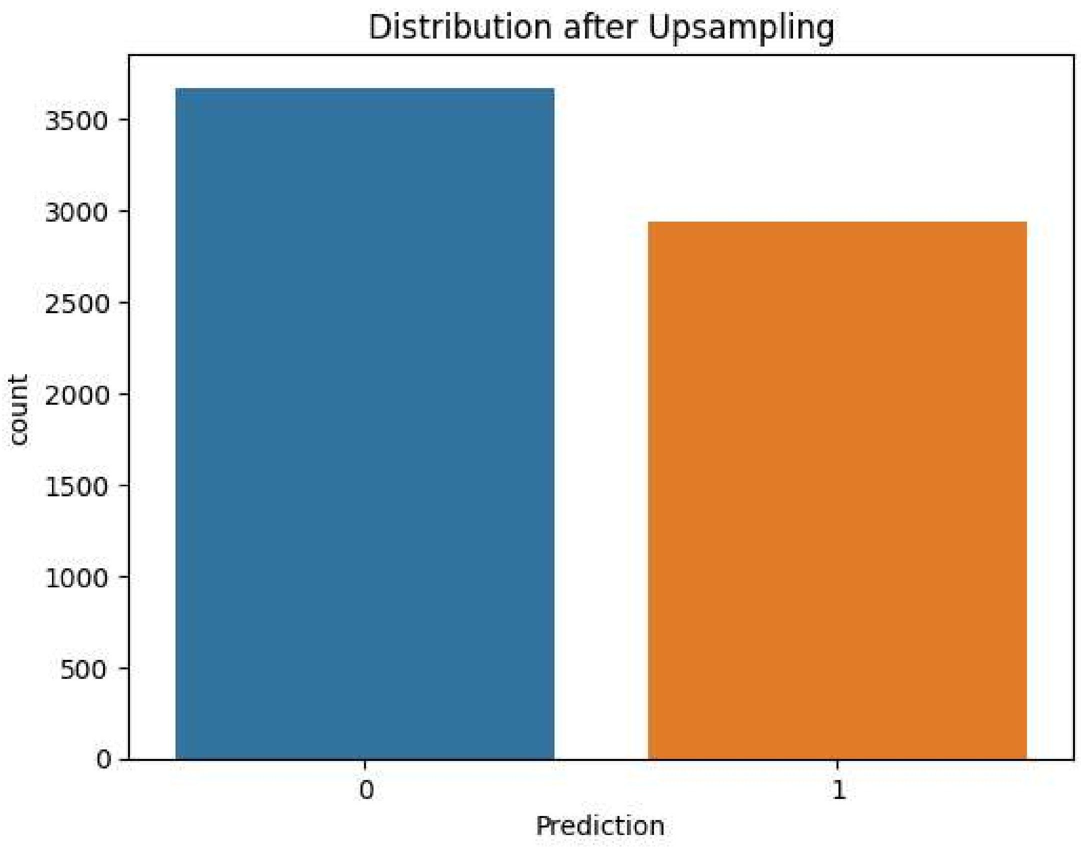
```
the           0
to            0
ect           0
and           0
for           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3001, dtype: int64
```

5. Upsampling to Handle Class Imbalance

In [17]:
```
spam_data = df[df["Prediction"] == 1]
ham_data = df[df["Prediction"] == 0]

spam_upsample = resample(spam_data,
                         replace=True,
                         n_samples=int(0.8*len(ham_data)),
                         random_state=42)

df_balanced = pd.concat([ham_data, spam_upsample]).sample(frac=1).reset_index(drop=True
sns.countplot(x=df_balanced["Prediction"])
plt.title("Distribution after Upsampling")
plt.show()
```



6. Split Data into Training and Testing Sets

In [18]:
```
X = df_balanced.drop("Prediction", axis=1)
y = df_balanced["Prediction"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
```

7. K-Nearest Neighbors (KNN) Classification

```
In [19]: k_values = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
         accuracy_values = []

         for k in tqdm(k_values):
             knn = KNeighborsClassifier(n_neighbors=k)
             knn.fit(X_train, y_train)
             y_pred_knn = knn.predict(X_test)
             accuracy_values.append(metrics.accuracy_score(y_test, y_pred_knn))

         # Plot K vs Accuracy
         fig = px.line(x=k_values, y=accuracy_values, title="K Value vs Accuracy (KNN)")
         fig.update_layout(xaxis_title="K Values", yaxis_title="Accuracy")
         fig.show()

         # Choose optimal K
         optimal_k = k_values[np.argmax(accuracy_values)]
         print(f"Optimal K: {optimal_k}")

         # Train KNN with optimal K
         knn_model = KNeighborsClassifier(n_neighbors=optimal_k)
         knn_model.fit(X_train, y_train)
         y_pred_knn = knn_model.predict(X_test)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
████| 15/15 [00:23<00:00,  1.54s/it]
```

```
Optimal K: 1
```

8. Support Vector Machine (SVM) Classification

In [20]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
```

9. Model Evaluation and Performance Analysis

In [21]:
```
print("===== KNN Metrics =====")
print(metrics.classification_report(y_test, y_pred_knn))

print("===== SVM Metrics =====")
print(metrics.classification_report(y_test, y_pred_svm))
```

```
===== KNN Metrics =====
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      1113
           1       0.85      0.96      0.90       870

    accuracy                           0.91      1983
   macro avg       0.91      0.91      0.91      1983
weighted avg       0.92      0.91      0.91      1983

===== SVM Metrics =====
              precision    recall  f1-score   support

           0       0.78      0.93      0.85      1113
           1       0.88      0.66      0.76       870

    accuracy                           0.81      1983
   macro avg       0.83      0.80      0.80      1983
weighted avg       0.82      0.81      0.81      1983
```