```python
import pandas as pd
from sklearn.impute import SimpleImputer

# Example dataframe
data = pd.read_csv('email.csv')
```

```python
print(data.head())
```

```
   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  allusty  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]
```

```python
# Define the target column name
target_column = 'Prediction'  # Replace 'Prediction' with the actual target column name if different

# Identify numerical and categorical features
numerical_features = data.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = data.select_dtypes(include=['object']).columns.tolist()

# Remove the target column from the feature lists if it's included
if target_column in numerical_features:
    numerical_features.remove(target_column)
if target_column in categorical_features:
    categorical_features.remove(target_column)

print('Numerical Features:', numerical_features)
print('Categorical Features:', categorical_features)
```

```
Numerical Features: ['the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou', 'in', 'on', 'is', 'this', 'enron', 'i', 'be', 'that', 'will', 'have', 'with', 'your', 'at', 'we', 's', 'are', ...]
```

(large output truncated)

Handle Missing Values

```python
missing_values = data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 Email No.     0
 the           0
 to            0
 ect           0
 and           0
                ..
 military      0
 allowing      0
 ff            0
 dry           0
 Prediction    0
Length: 3002, dtype: int64
```

Encode Categorical Variables

```python
# One-Hot Encoding for nominal features (if any)
data = pd.get_dummies(data, columns=categorical_features, drop_first=True)

print('Categorical features encoded.')
print(data.head())
```

```
Categorical features encoded.
   the  to  ect  and  for  of  a  you  hou ...  Email No._Email 996  \
0   0   0    1    0    0   0  2   0    0  ...    0
1   8  13   24    6    6   2 102  1   27  ...    0
2   0   0    1    0    0   0  8   0    0  ...    0
3   0   5   22    0    5   1 51   2   10  ...    0
4   7   6   17    1    5   2 57   0    9  ...    0

   Email No._Email 991  Email No._Email 992  Email No._Email 993  \
0    0                    0                    0
1    0                    0                    0
2    0                    0                    0
3    0                    0                    0
4    0                    0                    0

   Email No._Email 994  Email No._Email 995  Email No._Email 996  \
0    0                    0                    0
1    0                    0                    0
2    0                    0                    0
3    0                    0                    0
4    0                    0                    0

[5 rows x 8172 columns]
```

Scale Numerical Features

```python
from sklearn.preprocessing import StandardScaler

# Scaling Numerical Features
scaler = StandardScaler()
data[numerical_features] = scaler.fit_transform(data[numerical_features])

print('Numerical features scaled.')
print(data.head())
```

```
Numerical features scaled.
       the        to       ect       and       for  \
0 -0.565449 -0.649881 -0.293895 -0.508752 -0.421725 -0.611168
1  0.115757 0.714509 -1.557237 -0.371935 -0.421725 -0.611168
2 -0.565449 -0.649881 -0.293895 -0.508752 -0.421725 -0.611168
3 -0.565449 -0.649881 -0.293895 -0.508752 -0.421725 -0.611168
4  0.031741 -0.493816 -0.040093 -0.447605 -0.005087 -0.160559 ...
```

(output continues)

Split the Dataset

```python
from sklearn.model_selection import train_test_split

# Splitting data into features and target
X = data.drop(target_column, axis=1)
y = data[target_column]

# 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)

print('Dataset split into training and testing sets.')
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)

Dataset split into training and testing sets.
X_train shape: (4137, 8171)
X_test shape: (1035, 8171)
```

Model Selection and Training

```python
from sklearn.linear_model import LogisticRegression

# Model Selection: Example with Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

print('Logistic Regression model trained.')
```

```python
from sklearn.tree import DecisionTreeClassifier

# Model Selection: Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

print('Decision Tree model trained.')

Decision Tree model trained.
```

```python
from sklearn.svm import SVC

# Model Selection: Support Vector Machine
svm = SVC()
svm.fit(X_train, y_train)

print('Support Vector Machine model trained.')

Support Vector Machine model trained.
```

Model Evaluation

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Function to evaluate model performance
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1

# Evaluate Logistic Regression
log_metrics = evaluate_model(log_reg, X_test, y_test)
print(f'Logistic Regression - Accuracy: {log_metrics[0]}, Precision: {log_metrics[1]}, Recall: {log_metrics[2]}, F1-score: {log_metrics[3]}')

# Evaluate Decision Tree
decision_tree_metrics = evaluate_model(decision_tree, X_test, y_test)
print(f'Decision Tree - Accuracy: {decision_tree_metrics[0]}, Precision: {decision_tree_metrics[1]}, Recall: {decision_tree_metrics[2]}, F1-score: {decision_tree_metrics[3]}')

# Evaluate SVM
svm_metrics = evaluate_model(svm, X_test, y_test)
print(f'Support Vector Machine - Accuracy: {svm_metrics[0]}, Precision: {svm_metrics[1]}, Recall: {svm_metrics[2]}, F1-score: {svm_metrics[3]}')

Logistic Regression - Accuracy: 0.9710144927536232, Precision: 0.940983606557377, Recall: 0.9615384615384615, F1-score: 0.9510818438381939
Decision Tree - Accuracy: 0.9565217391304348, Precision: 0.9056603773584906, Recall: 0.9487179487179487, F1-score: 0.9267015706806283
Support Vector Machine - Accuracy: 0.9468599033816425, Precision: 0.9590361445783133, Recall: 0.8173076923076923, F1-score: 0.8875000000000001
```

Cross-Validation

```python
from sklearn.model_selection import cross_val_score

# Function to perform cross-validation and return the mean accuracy
def cross_validate_model(model, X, y):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    return scores.mean(), scores.std()

# Cross-validate Logistic Regression
log_reg_cv = cross_validate_model(log_reg, X, y)
print(f'Logistic Regression Cross-validation Accuracy: {log_reg_cv[0]}')

# Cross-validate Decision Tree
decision_tree_cv = cross_validate_model(decision_tree, X, y)
print(f'Decision Tree Cross-validation Accuracy: {decision_tree_cv[0]}')

# Cross-validate SVM
svm_cv = cross_validate_model(svm, X, y)
print(f'Support Vector Machine Cross-validation Accuracy: {svm_cv[0]}')
```

Ensemble Methods and Hyperparameter Tuning

```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Ensemble Method: Random Forest and Gradient Boosting
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)

gradient_boosting = GradientBoostingClassifier()
gradient_boosting.fit(X_train, y_train)

print('Random Forest and Gradient Boosting models trained.')

Ensemble methods trained.
```

```python
# Hyperparameter Tuning: Grid Search Example
param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear']}
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print('Best parameters found by Grid Search:', grid_search.best_params_)

Best parameters found by Grid Search: {'C': 0.1, 'solver': 'liblinear'}
```

```python
# RandomSearch Example
param_dist = {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20, 30]}
random_search = RandomizedSearchCV(RandomForestClassifier(), param_distributions=param_dist, n_iter=5, cv=5)
random_search.fit(X_train, y_train)

print('Best parameters found by Random Search:', random_search.best_params_)

Best parameters found by Random Search: {'n_estimators': 100, 'max_depth': None}
```