1. Data Preprocessing

```
# importing libraries

import numpy as num   # working with arrays.
import matplotlib.pyplot as mpt
import seaborn as sns

import pandas as pd  # used for clean, analyse and manipulate data

# Load the dataset
df = pd.read_csv('email_data.csv')

# Display the first few rows of the dataset
print(df.head())


# Extracting independent variable:  To extract an independent variable, we will use iloc[ ] method of Pandas library. It is used to extract tl

x= df.iloc[:,:-1].values
print(x);
```

```
[[48.  5.  4.  0.]
 [38.  2.  4.  1.]
 [24.  3.  2.  0.]
 [17.  3.  3.  0.]
 [30.  2.  1.  0.]
 [48.  9.  2.  0.]
 [28.  2.  4.  1.]
 [32.  2.  0.  1.]
 [20.  3.  4.  0.]
 [20.  6.  3.  0.]
 [33.  3.  4.  1.]
 [45.  8.  0.  0.]
 [49.  0.  3.  1.]
 [33.  7.  4.  0.]
 [12.  6.  3.  0.]
 [31.  1.  1.  1.]
 [11.  7.  1.  0.]
 [33.  0.  4.  1.]
 [39.  8.  3.  0.]
 [47.  8.  0.  0.]
 [11.  1.  4.  0.]
 [30.  6.  1.  1.]
 [42.  9.  1.  0.]
 [21.  2.  4.  0.]
 [31.  6.  3.  0.]
 [34.  9.  1.  0.]
 [36.  8.  3.  1.]
 [37.  3.  1.  1.]
 [25.  0.  1.  0.]
 [24.  1.  2.  1.]
 [12.  0.  1.  0.]
 [46.  4.  0.  0.]
 [16.  4.  4.  0.]
 [30.  6.  4.  1.]
 [18.  8.  3.  1.]
 [48.  8.  1.  1.]
 [27.  2.  0.  1.]
 [13.  2.  3.  1.]
 [34.  2.  2.  1.]
 [23.  3.  3.  1.]
 [18.  7.  3.  1.]
 [35.  5.  1.  1.]
 [11.  7.  2.  0.]
 [29.  0.  3.  1.]
 [37.  7.  0.  1.]
 [16.  3.  0.  1.]
 [17.  0.  4.  1.]
 [44.  7.  2.  1.]
 [23.  3.  2.  1.]
 [26.  5.  4.  1.]
 [45.  7.  3.  1.]
 [49.  3.  2.  1.]
 [13.  2.  0.  1.]
 [11.  8.  0.  1.]
 [15.  2.  1.  0.]
 [13.  8.  2.  0.]
 [38.  1.  3.  1.]
```

```
 [27.  1.  4.  1.]
```

```
# Here we have taken all the rows with the last column only. It will give the array of dependent variables.

y= df.iloc[:,3].values
print(y);
```

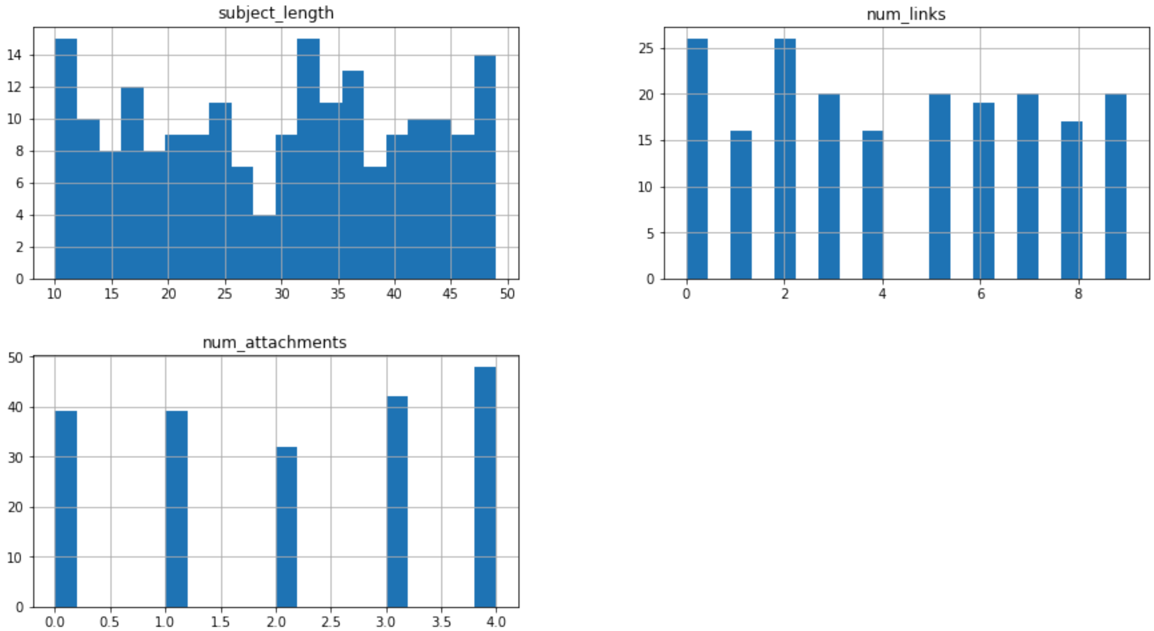
```
[ 0.  1.  0.  0.  0.  0.  1.  1.  0.  0.  1.  0.  1.  0.  0.  1.  0.  1.
  0.  0.  0.  1.  0.  0.  0.  0.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  0.  0.  1.  1.  0.  1.  0.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  1.
  0.  0.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.  1.  1.  0.  0.  0.  0.
  1.  0.  1.  0.  1.  1.  0.  0.  0.  0.  1.  0.  0.  1.  0.  0.  0.  0.
  0.  1.  0.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  0.  1.  0.  0.
  0.  0.  0.  0.  1.  1.  0.  0.  0.  1.  0.  1.  0.  0.  0.  0.  1.  1.
  0.  0.  1.  1.  1.  1.  0.  1.  0.  1.  1.  1.  1.  1.  0.  0.  1.  1.
  0.  1.  1.  0.  1.  1.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  1.
  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  0.  0.  0.  1.  1.
  0. nan nan  1.]
```

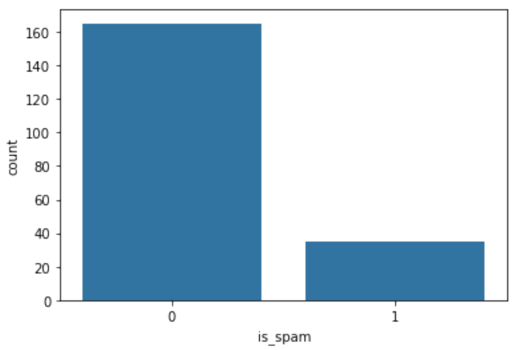
```
# Check for missing values

print(df.isnull().sum()) # sum of missing values in the column


print(df.isnull())  # sum of missing values in the rows

# If there are missing values, handle them. For example, if there are missing values in a column,
# we can choose to fill them with the mean (for numeric columns) or mode (for categorical columns),
# or we can drop rows/columns with missing values.

# Filling missing values with the mean for numeric columns
df.fillna(df.mean(), inplace=True)

# Alternatively, drop rows with missing values
#df.dropna(inplace=True)
```

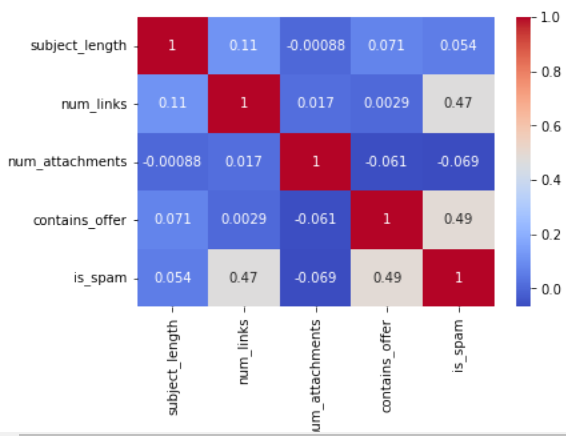
```
subject_length     0
num_links          0
num_attachments    0
contains_offer     0
is_spam            0
dtype: int64
     subject_length  num_links  num_attachments  contains_offer  is_spam
0             False      False            False           False    False
1             False      False            False           False    False
2             False      False            False           False    False
3             False      False            False           False    False
4             False      False            False           False    False
..              ...        ...              ...             ...      ...
195           False      False            False           False    False
196           False      False            False           False    False
197           False      False            False           False    False
198           False      False            False           False    False
201           False      False            False           False    False

[200 rows x 5 columns]
```

```
# Display summary statistics
print(df.describe())

# Plot the distribution of each feature
df[['subject_length', 'num_links', 'num_attachments']].hist(bins=20, figsize=(15, 8))
plt.show()

# Plot the distribution of the target variable
sns.countplot(x='is_spam', data=df)
plt.show()

# Check the correlation matrix
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

	subject_length	num_links	num_attachments	contains_offer	is_spam
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	29.575000	4.310000	2.105000	0.470000	0.175000
std	12.026954	2.933758	1.464479	0.500352	0.380921
min	10.000000	0.000000	0.000000	0.000000	0.000000
25%	18.000000	2.000000	1.000000	0.000000	0.000000
50%	31.000000	4.000000	2.000000	0.000000	0.000000
75%	41.000000	7.000000	3.000000	1.000000	0.000000
max	49.000000	9.000000	4.000000	1.000000	1.000000







```
from sklearn.preprocessing import StandardScaler

# Separate features and target variable
X = df[['subject_length', 'num_links', 'num_attachments', 'contains_offer']]
y = df['is_spam']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert the standardized features back to a DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Display the first few rows of the standardized features
print(X_scaled_df.head())
```

```
   subject_length  num_links  num_attachments  contains_offer
0        1.542518   0.227497         1.305618       -0.946393
1        0.706712  -0.796241         1.305618        1.067210
2       -0.463418  -0.454995        -0.068357       -0.946393
3       -1.048482  -0.454995         0.618630       -0.946393
4        0.038066  -0.796241        -0.755344       -0.946393
```

2. Concept Learning

```
#Find's Algorithm

import numpy as np
import pandas as pd

def find_s_algorithm(df):
    # Separate features and target variable
    X = df[['subject_length', 'num_links', 'num_attachments', 'contains_offer']].values
    y = df['is_spam'].values

    # Get indices of positive examples
    positive_indices = np.where(y == 1)[0]

    # Check if there are any positive examples
    if len(positive_indices) == 0:
        raise ValueError("No positive examples found in the dataset.")

    # Initialize the hypothesis to the most specific hypothesis based on the first positive example
    hypothesis = X[positive_indices[0]].copy()

    # Update the hypothesis for each positive example
    for index in positive_indices:
        example = X[index]
        for i in range(len(hypothesis)):
            if hypothesis[i] != example[i]:
                hypothesis[i] = None  # Generalize feature if it does not match

    return hypothesis

# Load the dataset (assuming it's already preprocessed)
df = pd.read_csv('email_data.csv')

# Apply the Find-S algorithm
most_specific_hypothesis = find_s_algorithm(df)
print("Most Specific Hypothesis:", most_specific_hypothesis)
```

```
Most Specific Hypothesis: [nan nan nan nan]
```

3. Model Training and Evaluation

```
# Check lengths and a few examples for debugging
print(f"Length of y_test: {len(y_test)}")
print(f"Length of y_pred: {len(y_pred)}")

print("y_test sample:", y_test.head())
print("y_pred sample:", y_pred[:10])

# Ensuring hypothesis is correct or not
def apply_hypothesis(hypothesis, X):
    y_pred = []
    for example in X.values:  # Ensure we're iterating over a numpy array
        match = True
        for i in range(len(hypothesis)):
            if hypothesis[i] is not None and hypothesis[i] != example[i]:
                match = False
                break
        y_pred.append(1 if match else 0)
    return np.array(y_pred)
```

```
Length of y_test: 41
Length of y_pred: 4
y_test sample: 95     0
15     0
30     0
159    0
186    1
Name: is_spam, dtype: int64
y_pred sample: [0 0 0 0]
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def find_s_algorithm(df):
    X = df[['subject_length', 'num_links', 'num_attachments', 'contains_offer']].values
    y = df['is_spam'].values

    positive_indices = np.where(y == 1)[0]

    if len(positive_indices) == 0:
        raise ValueError("No positive examples found in the dataset.")

    hypothesis = X[positive_indices[0]].copy()

    for index in positive_indices:
        example = X[index]
        for i in range(len(hypothesis)):
            if hypothesis[i] != example[i]:
                hypothesis[i] = None  # Generalize feature if it does not match

    return hypothesis

def apply_hypothesis(hypothesis, X):
    y_pred = []
    for example in X:
        match = True
        for i in range(len(hypothesis)):
            if hypothesis[i] is not None and hypothesis[i] != example[i]:
                match = False
```

4. Interpretation and Discussion

Interpretation and Discussion of the Find-S Algorithm Concept Learned:

The Find-S algorithm generates the most specific hypothesis that classifies all positive (spam) examples in the training data. For instance, it might produce a hypothesis like `[10, 5, 2, 1]`, meaning spam emails have a subject length of exactly 10, 5 links, 2 attachments, and contain an offer.

Effectiveness:

- **Strengths**: Accurately identifies spam in training data if the training examples are representative.
- **Limitations**: Overfits to training data, struggles with noisy data, and may miss feature interactions.

Limitations:

1. **Overfitting**: Hypothesis may be too specific, performing poorly on new data.
2. **Noisy Data**: May incorporate errors from noisy or mislabeled examples.
3. **Feature Interaction**: Doesn't capture complex interactions between features.

Improvements:

1. **Advanced Algorithms**: Use models like decision trees or random forests for better generalization.
2. **Feature Engineering**: Combine or create new features to capture more complex patterns.
3. **Regularization**: Apply techniques to reduce overfitting.
4. **Data Cleaning**: Address noise and ensure data quality.

In summary, while Find-S provides a specific hypothesis for spam detection, it has limitations in terms of overfitting and handling noisy data. Using more advanced techniques and improving data quality can enhance performance.

```
print(f"F1 Score: {f1:.2f}")
```

Start coding or generate with AI.

```
Unique values in y_pred: [0]
Number of positive predictions: 0
```