```
import pandas as pd

# Load the dataset
file_path = 'water_potability.csv'
data = pd.read_csv(file_path)

# Display basic information and the first few rows of the dataset
data_info = data.info()
data_head = data.head()

data_info, data_head
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-10-84665b777c8e> in <cell line: 1>()
----> 1 import pandas as pd
      2
      3 # Load the dataset
      4 file_path = 'water_potability.csv'
      5 data = pd.read_csv(file_path)

/usr/local/lib/python3.10/dist-packages/pandas/__init__.py in <module>
    109 )
    110
--> 111 from pandas.core.dtypes.dtypes import SparseDtype
    112
    113 from pandas.tseries.api import infer_freq

ImportError: cannot import name 'SparseDtype' from 'pandas.core.dtypes.dtypes'
(/usr/local/lib/python3.10/dist-packages/pandas/core/dtypes/dtypes.py)

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES 在 STACK OVERFLOW 中搜索

```
# Check for missing values in each column
missing_values = data.isnull().sum()
missing_values_percentage = (missing_values / len(data)) * 100

missing_data = pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_values_percentage})
missing_data[missing_data['Missing Values'] > 0]
```

	Missing Values	Percentage
ph	491	14.987790
Sulfate	781	23.840049
Trihalomethanes	162	4.945055

```
# Split the dataset into two groups based on 'Potability'
potable_water = data[data['Potability'] == 1]
non_potable_water = data[data['Potability'] == 0]

# Checking the split datasets
split_summary = {
    "Potable Water Dataset Size": potable_water.shape,
    "Non-Potable Water Dataset Size": non_potable_water.shape
}

split_summary
```

```
{'Potable Water Dataset Size': (1278, 10),
 'Non-Potable Water Dataset Size': (1998, 10)}
```

```
# Filling missing values with the median for each group
potable_water_filled = potable_water.fillna(potable_water.median())
non_potable_water_filled = non_potable_water.fillna(non_potable_water.median())

# Check if missing values are filled
potable_missing_after = potable_water_filled.isnull().sum().sum()
non_potable_missing_after = non_potable_water_filled.isnull().sum().sum()

filling_summary = {
        "Potable Water Missing Values After Filling": potable_missing_after,
        "Non-Potable Water Missing Values After Filling": non_potable_missing_after
}

filling_summary
```

```
{'Potable Water Missing Values After Filling': 0,
 'Non-Potable Water Missing Values After Filling': 0}
```

```
from scipy.spatial.distance import mahalanobis
import scipy as sp

def calculate_mahalanobis_distance(df):
        # Calculate the inverse of the covariance matrix
        covariance_matrix = df.cov()
        inv_cov_matrix = sp.linalg.inv(covariance_matrix)

        # Calculate the mean of the data
        mean = df.mean()

        # Calculate the Mahalanobis distance for each observation
        df['Mahalanobis'] = df.apply(lambda row: mahalanobis(row, mean, inv_cov_matrix), axis=1)
        return df

# Calculate Mahalanobis distance for potable water dataset
potable_water_mahalanobis = calculate_mahalanobis_distance(potable_water_filled.drop('Potability', axis=1))
potable_water_mahalanobis.head()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Mah
250	9.445130	145.805402	13168.52916	9.444471	310.583374	592.659021	8.606397	77.577460	3.875165	
251	9.024845	128.096691	19859.67648	8.016423	300.150377	451.143481	14.770863	73.778026	3.985251	
252	7.036752	169.974849	23403.63730	8.519730	331.838167	475.573562	12.924107	50.861913	2.747313	
253	6.800119	242.008082	39143.40333	9.501695	187.170714	376.456593	11.432466	73.777275	3.854940	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

```
from scipy.stats import chi2

# Calculate the 95% confidence interval threshold for Mahalanobis distance
confidence_level = 0.95
degrees_of_freedom = potable_water_filled.drop('Potability', axis=1).shape[1]  # Number of features
threshold = chi2.ppf(confidence_level, degrees_of_freedom)

# Identifying outliers in the potable water dataset
potable_outliers = potable_water_mahalanobis[potable_water_mahalanobis['Mahalanobis'] > threshold]
num_potable_outliers = potable_outliers.shape[0]

threshold, num_potable_outliers, potable_outliers.head()
```

```
(16.918977604620448,
 0,
 Empty DataFrame
```

```
 Columns: [ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity, Mahalanobis]
 Index: [])
```

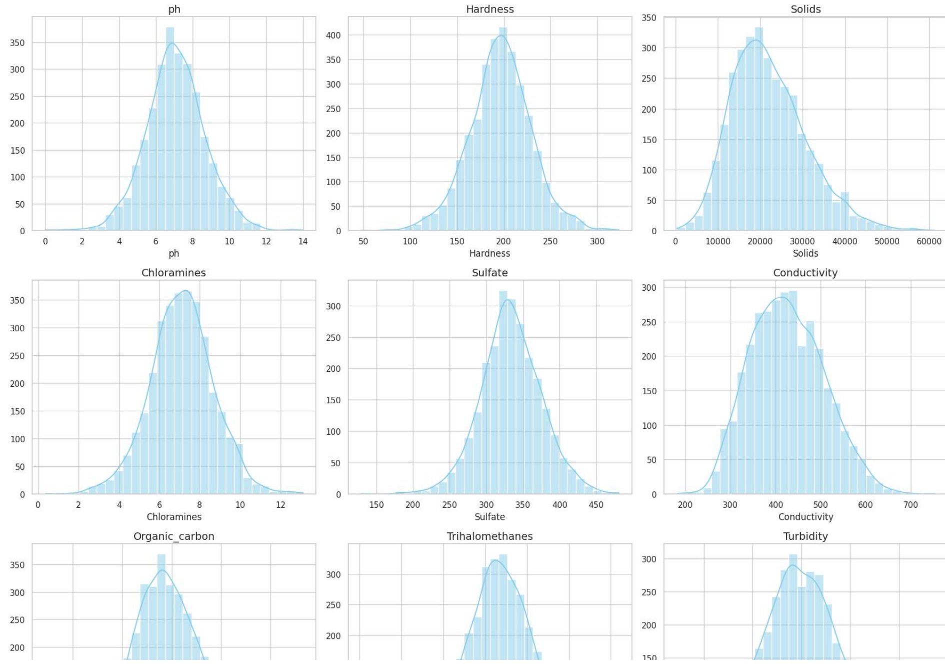
```
# Calculate Mahalanobis distance for non-potable water dataset
non_potable_water_mahalanobis = calculate_mahalanobis_distance(non_potable_water_filled.drop('Potability', axis=1))

# Identifying outliers in the non-potable water dataset
non_potable_outliers = non_potable_water_mahalanobis[non_potable_water_mahalanobis['Mahalanobis'] > threshold]
num_non_potable_outliers = non_potable_outliers.shape[0]

num_non_potable_outliers, non_potable_outliers.head()
```

```
(0,
 Empty DataFrame
 Columns: [ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity, Mahalanobis]
 Index: [])
```

```
# Calculate the 99% confidence interval threshold for Mahalanobis distance
confidence_level_99 = 0.99
threshold_99 = chi2.ppf(confidence_level_99, degrees_of_freedom)

# Identifying outliers for potable and non-potable water datasets at 99% confidence level
potable_outliers_99 = potable_water_mahalanobis[potable_water_mahalanobis['Mahalanobis'] > threshold_99]
non_potable_outliers_99 = non_potable_water_mahalanobis[non_potable_water_mahalanobis['Mahalanobis'] > threshold_99]

num_potable_outliers_99 = potable_outliers_99.shape[0]
num_non_potable_outliers_99 = non_potable_outliers_99.shape[0]

threshold_99, num_potable_outliers_99, num_non_potable_outliers_99, potable_outliers_99.head(), non_potable_outliers_99.head()
```

```
(21.665994333461924,
 0,
 0,
 Empty DataFrame
 Columns: [ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity, Mahalanobis]
 Index: [],
 Empty DataFrame
 Columns: [ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity, Mahalanobis]
 Index: [])
```

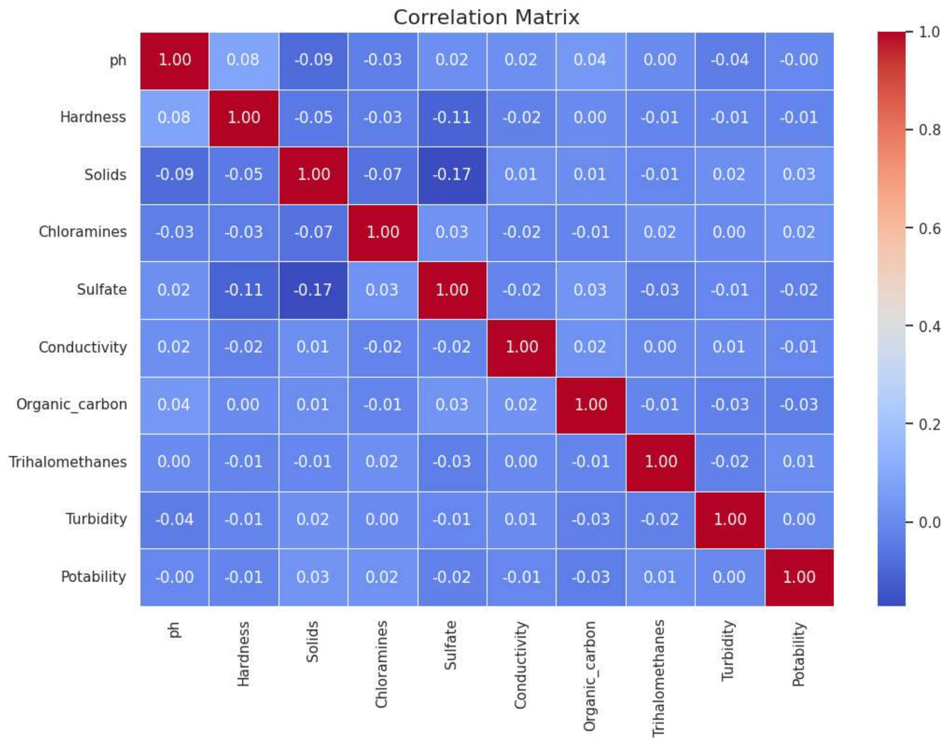
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style of seaborn
sns.set(style="whitegrid")

# Creating a figure for multiple subplots
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(18, 15))

# Flattening the axes array for easy indexing
axes = axes.flatten()

# Plotting histograms for each numeric feature
for i, col in enumerate(data.columns[:-1]):  # Exclude 'Potability' from the plots
    sns.histplot(data[col], ax=axes[i], kde=True, bins=30, color='skyblue')
    axes[i].set_title(col, fontsize=14)
    axes[i].set_ylabel('')

# Adjust layout
plt.tight_layout()
plt.show()
```




```
# Creating a figure for multiple subplots
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(18, 15))

# Flattening the axes array for easy indexing
axes = axes.flatten()

# Plotting boxplots for each numeric feature
for i, col in enumerate(data.columns[:-1]):   # Exclude 'Potability' from the plots
        sns.boxplot(x='Potability', y=col, data=data, ax=axes[i])
        axes[i].set_title(col, fontsize=14)

# Adjust layout
plt.tight_layout()
plt.show()
```

ph Hardness Solids

```
# Calculate the correlation matrix
corr_matrix = data.corr()

# Plotting the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix", fontsize=16)
plt.show()
```



	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.00	0.08	-0.09	-0.03	0.02	0.02	0.04	0.00	-0.04	-0.00
Hardness	0.08	1.00	-0.05	-0.03	-0.11	-0.02	0.00	-0.01	-0.01	-0.01
Solids	-0.09	-0.05	1.00	-0.07	-0.17	0.01	0.01	-0.01	0.02	0.03
Chloramines	-0.03	-0.03	-0.07	1.00	0.03	-0.02	-0.01	0.02	0.00	0.02
Sulfate	0.02	-0.11	-0.17	0.03	1.00	-0.02	0.03	-0.03	-0.01	-0.02
Conductivity	0.02	-0.02	0.01	-0.02	-0.02	1.00	0.02	0.00	0.01	-0.01
Organic_carbon	0.04	0.00	0.01	-0.01	0.03	0.02	1.00	-0.01	-0.03	-0.03
Trihalomethanes	0.00	-0.01	-0.01	0.02	-0.03	0.00	-0.01	1.00	-0.02	0.01
Turbidity	-0.04	-0.01	0.02	0.00	-0.01	0.01	-0.03	-0.02	1.00	0.00
Potability	-0.00	-0.01	0.03	0.02	-0.02	-0.01	-0.03	0.01	0.00	1.00

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Filling missing values in the entire dataset with the median
data_filled = data.fillna(data.median())

# Separating the features and the target variable after filling missing values
X_filled = data_filled.drop('Potability', axis=1)
y_filled = data_filled['Potability']

# Splitting the dataset into training and testing sets
X_train_filled, X_test_filled, y_train_filled, y_test_filled = train_test_split(
        X_filled, y_filled, test_size=0.3, random_state=42)

# Fitting the model again
rf.fit(X_train_filled, y_train_filled)

# Getting feature importances again
feature_importances_filled = rf.feature_importances_

# Creating a DataFrame for feature importances
features_filled = pd.DataFrame({'Feature': X_filled.columns, 'Importance': feature_importances_filled})
features_filled.sort_values(by='Importance', ascending=False, inplace=True)

features_filled
```

	Feature	Importance
0	ph	0.124111
1	Hardness	0.122985
4	Sulfate	0.117781
3	Chloramines	0.116736
2	Solids	0.116070
6	Organic_carbon	0.103027
8	Turbidity	0.101586
5	Conductivity	0.100784
7	Trihalomethanes	0.096920

特征工程：从这些得分可以看出，所有特征的重要性都相对平均，没有特别突出的特征。这意味着每个特征对模型的贡献都是有价值的，我们可能不需要从这个模型的角度进行特征删除。

```
# Defining bins for pH
bins = [0, 6.5, 8.5, 14]
labels = ['Acidic', 'Neutral', 'Alkaline']
data_filled['pH_category'] = pd.cut(data_filled['ph'], bins=bins, labels=labels, include_lowest=True)

# One-hot encoding of the pH categories
pH_dummies = pd.get_dummies(data_filled['pH_category'], prefix='pH')

# Adding the new features to the dataset
data_fe = pd.concat([data_filled, pH_dummies], axis=1)

# Dropping the original and binned pH column
data_fe.drop(['ph', 'pH_category'], axis=1, inplace=True)

# Display the first few rows of the updated dataset
data_fe.head()
```

	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Tri
0	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	
1	129.422921	18630.05786	6.635246	333.073546	592.885359	15.180013	
2	224.236259	19909.54173	9.275884	333.073546	418.606213	16.868637	
3	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

# Log transformation of the 'Solids' feature
data_fe['Solids_log'] = np.log(data_fe['Solids'])

# Creating an interaction feature between 'Organic_carbon' and 'Trihalomethanes'
data_fe['Organic_Trihalomethanes'] = data_fe['Organic_carbon'] * data_fe['Trihalomethanes']

# Selecting features for polynomial transformation
features_for_poly = ['Hardness', 'Chloramines', 'Sulfate']

# Initializing the PolynomialFeatures object
poly = PolynomialFeatures(degree=2, include_bias=False)

# Fitting and transforming the selected features
poly_features = poly.fit_transform(data_fe[features_for_poly])

# Creating a DataFrame for the polynomial features
poly_feature_names = [f'poly_{i}' for i in range(poly_features.shape[1])]
poly_features_df = pd.DataFrame(poly_features, columns=poly_feature_names)

# Merging the polynomial features with the main dataset
data_fe = pd.concat([data_fe, poly_features_df], axis=1)

# Dropping the original features used for polynomial transformation
data_fe.drop(features_for_poly, axis=1, inplace=True)

# Display the first few rows of the updated dataset
data_fe.head()
```

	Solids	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	20791.31898	564.308654	10.379783	86.990970	2.963135	0
1	18630.05786	592.885359	15.180013	56.329076	4.500656	0
2	19909.54173	418.606213	16.868637	66.420093	3.055934	0
3	22018.41744	363.266516	18.436525	100.341674	4.628771	0
4	17978.98634	398.410813	11.558279	31.997993	4.075075	0

对pH值进行分箱：将pH值分为酸性、中性和碱性三类，并进行独热编码。 对Solids进行对数变换：以减少其右偏分布。 创建交互特征：有机碳（Organic_carbon）和三卤甲烷（Trihalomethanes）的交互项。 多项式特征：对pH、硬度（Hardness）、氯胺（Chloramines）和硫酸盐（Sulfate）这四个特征进行2次多项式变换。

数据预处理：

```
from sklearn.preprocessing import StandardScaler

# Separating the features and the target variable for preprocessing
X_fe = data_fe.drop('Potability', axis=1)
y_fe = data_fe['Potability']

# Initializing the StandardScaler
scaler = StandardScaler()

# Fitting the scaler to the features and transforming them
X_fe_scaled = scaler.fit_transform(X_fe)

# Checking the class distribution of the target variable
class_distribution = y_fe.value_counts()

class_distribution
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
from imblearn.over_sampling import SMOTE

# Initializing SMOTE
smote = SMOTE(random_state=42)

# Fitting SMOTE and resampling the dataset
X_fe_resampled, y_fe_resampled = smote.fit_resample(X_fe_scaled, y_fe)

# Checking the new class distribution
new_class_distribution = pd.Series(y_fe_resampled).value_counts()

new_class_distribution
```

```
0    1998
1    1998
Name: Potability, dtype: int64
```

```
from sklearn.utils import resample

# Separating the majority and minority classes
majority_class = data_fe[y_fe == 0]
minority_class = data_fe[y_fe == 1]

# Upsampling the minority class
minority_upsampled = resample(minority_class,
                              replace=True,                     # sample with replacement
                              n_samples=len(majority_class),  # to match majority class
                              random_state=42)                # reproducible results

# Combining majority class with upsampled minority class
upsampled_data = pd.concat([majority_class, minority_upsampled])

# Checking the new class distribution
upsampled_class_distribution = upsampled_data['Potability'].value_counts()

upsampled_class_distribution
```

```
0    1998
1    1998
Name: Potability, dtype: int64
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Splitting the upsampled data into features and target
X_upsampled = upsampled_data.drop('Potability', axis=1)
y_upsampled = upsampled_data['Potability']

# Splitting the upsampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
        X_upsampled, y_upsampled, test_size=0.3, random_state=42)

# Initializing the RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)

# Fitting the model to the training data
rf_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluating the model
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_classification_report = classification_report(y_test, y_pred_rf)

rf_accuracy, rf_classification_report
```

```
(0.8215179316096747,
 '              precision    recall  f1-score   support\n\n           0       0.82      0.84      0.83       617\n           1       0.82      0.80      0.81       582\n\n    accuracy                           0.82      1199\n   macro avg       0.82      0.82      0.82      1199\nweighted avg       0.82      0.82      0.82      1199\n')
```

```
from sklearn.linear_model import LogisticRegression

# Initializing the Logistic Regression model
lr_model = LogisticRegression(random_state=42)

# Fitting the model to the training data
lr_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_lr = lr_model.predict(X_test)

# Evaluating the model
lr_accuracy = accuracy_score(y_test, y_pred_lr)
lr_classification_report = classification_report(y_test, y_pred_lr)

lr_accuracy, lr_classification_report
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
(0.53628023352794,
 '              precision    recall  f1-score   support\n\n           0       0.54      0.62      0.58       617\n           1       0.53      0.45      0.48       582\n\n    accuracy                           0.54      1199\n   macro avg       0.53      0.53      0.53      1199\nweighted avg       0.54      0.54      0.53      1199\n')
```

```
from sklearn.neural_network import MLPClassifier

# Initializing the ANN model
ann_model = MLPClassifier(random_state=42)

# Fitting the model to the training data
ann_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_ann = ann_model.predict(X_test)

# Evaluating the model
ann_accuracy = accuracy_score(y_test, y_pred_ann)
ann_classification_report = classification_report(y_test, y_pred_ann)
ann_accuracy, ann_classification_report
```

```
(0.5437864887406172,
 '              precision    recall  f1-score   support\n\n           0       0.54      0.81      0.65       617\n           1       0.56      0.26      0.36       582\n\n    accuracy                           0.54      1199\n   macro avg       0.55      0.54      0.50      1199\nweighted avg       0.55      0.54      0.51      1199\n')
```

```
from sklearn.neighbors import KNeighborsClassifier

# Initializing the KNN model
knn_model = KNeighborsClassifier()

# Fitting the model to the training data
knn_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_knn = knn_model.predict(X_test)

# Evaluating the model
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_classification_report = classification_report(y_test, y_pred_knn)
knn_accuracy, knn_classification_report
```

```
(0.6263552960800667,
 '              precision    recall  f1-score   support\n\n           0       0.66      0.58      0.61       617\n           1       0.60      0.68      0.64       582\n\n    accuracy                           0.63      1199\n   macro avg       0.63      0.63      0.63      1199\nweighted avg       0.63      0.63      0.63      1199\n')
```

```
from sklearn.svm import SVC

# Initializing the Support Vector Machine model
svm_model = SVC(random_state=42)

# Fitting the model to the training data
svm_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_svm = svm_model.predict(X_test)

# Evaluating the model
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_classification_report = classification_report(y_test, y_pred_svm)
svm_accuracy, svm_classification_report
```

```
(0.5629691409507923,
 '              precision    recall  f1-score   support\n\n           0       0.55      0.87      0.67       617\n           1       0.63      0.24      0.35       582\n\n    accuracy                           0.56      1199\n   macro avg       0.59      0.55      0.51      1199\nweighted avg       0.59      0.56      0.51      1199\n')
```

```
from sklearn.ensemble import GradientBoostingClassifier

# Initializing the Gradient Boosting Classifier
gbm_model = GradientBoostingClassifier(random_state=42)

# Fitting the model to the training data
gbm_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred_gbm = gbm_model.predict(X_test)

# Evaluating the model
gbm_accuracy = accuracy_score(y_test, y_pred_gbm)
gbm_classification_report = classification_report(y_test, y_pred_gbm)
gbm_accuracy, gbm_classification_report
```

```
(0.6688907422852377,
 '              precision    recall  f1-score   support\n\n           0       0.65      0.76      0.70       617\n           1       0.69      0.58      0.63       582\n\n    accuracy                           0.67      1199\n   macro avg       0.67      0.67      0.66      1199\nweighted avg       0.67      0.67      0.67      1199\n')
```

过拟合:

```
# 随机森林模型在训练集上的表现
rf_train_predictions = rf_model.predict(X_train)
rf_train_accuracy = accuracy_score(y_train, rf_train_predictions)

# 随机森林模型在测试集上的表现
rf_test_predictions = rf_model.predict(X_test)
rf_test_accuracy = accuracy_score(y_test, rf_test_predictions)

print("Random Forest Training Accuracy:", rf_train_accuracy)
print("Random Forest Testing Accuracy:", rf_test_accuracy)
```

```
Random Forest Training Accuracy: 1.0
Random Forest Testing Accuracy: 0.8215179316096747
```

处理过拟合:

```
from sklearn.ensemble import RandomForestClassifier

# 调整的参数包括树的数量（n_estimators），树的最大深度（max_depth）
# 以及节点划分所需的最小样本数（min_samples_split）
rf_model_tuned = RandomForestClassifier(
        n_estimators=100,           # 试着减少树的数量
        max_depth=10,                  # 限制树的最大深度
        min_samples_split=4,      # 增加节点划分所需的最小样本数
        random_state=42
)

# 用训练集数据训练模型
rf_model_tuned.fit(X_train, y_train)

# 在训练集和测试集上评估模型性能
rf_train_accuracy_tuned = rf_model_tuned.score(X_train, y_train)
rf_test_accuracy_tuned = rf_model_tuned.score(X_test, y_test)

print("Tuned Random Forest Training Accuracy:", rf_train_accuracy_tuned)
print("Tuned Random Forest Testing Accuracy:", rf_test_accuracy_tuned)
```

```
Tuned Random Forest Training Accuracy: 0.918484090096532
Tuned Random Forest Testing Accuracy: 0.7648040033361134
```

```
from sklearn.ensemble import GradientBoostingClassifier
```