```
 import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('/content/Customer-Churn-Records (1).csv')
```

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   RowNumber           10000 non-null  int64
 1   CustomerId          10000 non-null  int64
 2   Surname             10000 non-null  object
 3   CreditScore         10000 non-null  int64
 4   Geography           10000 non-null  object
 5   Gender              10000 non-null  object
 6   Age                 10000 non-null  int64
 7   Tenure              10000 non-null  int64
 8   Balance             10000 non-null  float64
 9   NumOfProducts       10000 non-null  int64
 10  HasCrCard           10000 non-null  int64
 11  IsActiveMember      10000 non-null  int64
 12  EstimatedSalary     10000 non-null  float64
 13  Exited              10000 non-null  int64
 14  Complain            10000 non-null  int64
 15  Satisfaction Score  10000 non-null  int64
 16  Card Type           10000 non-null  object
 17  Point Earned        10000 non-null  int64
dtypes: float64(2), int64(12), object(4)
memory usage: 1.4+ MB
```

```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000

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

	0
RowNumber	0
CustomerId	0
Surname	0
CreditScore	0
Geography	0
Gender	0
Age	0
Tenure	0
Balance	0
NumOfProducts	0
HasCrCard	0
IsActiveMember	0
EstimatedSalary	0
Exited	0
Complain	0
Satisfaction Score	0
Card Type	0
Point Earned	0

dtype: int64

```
df.drop_duplicates()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

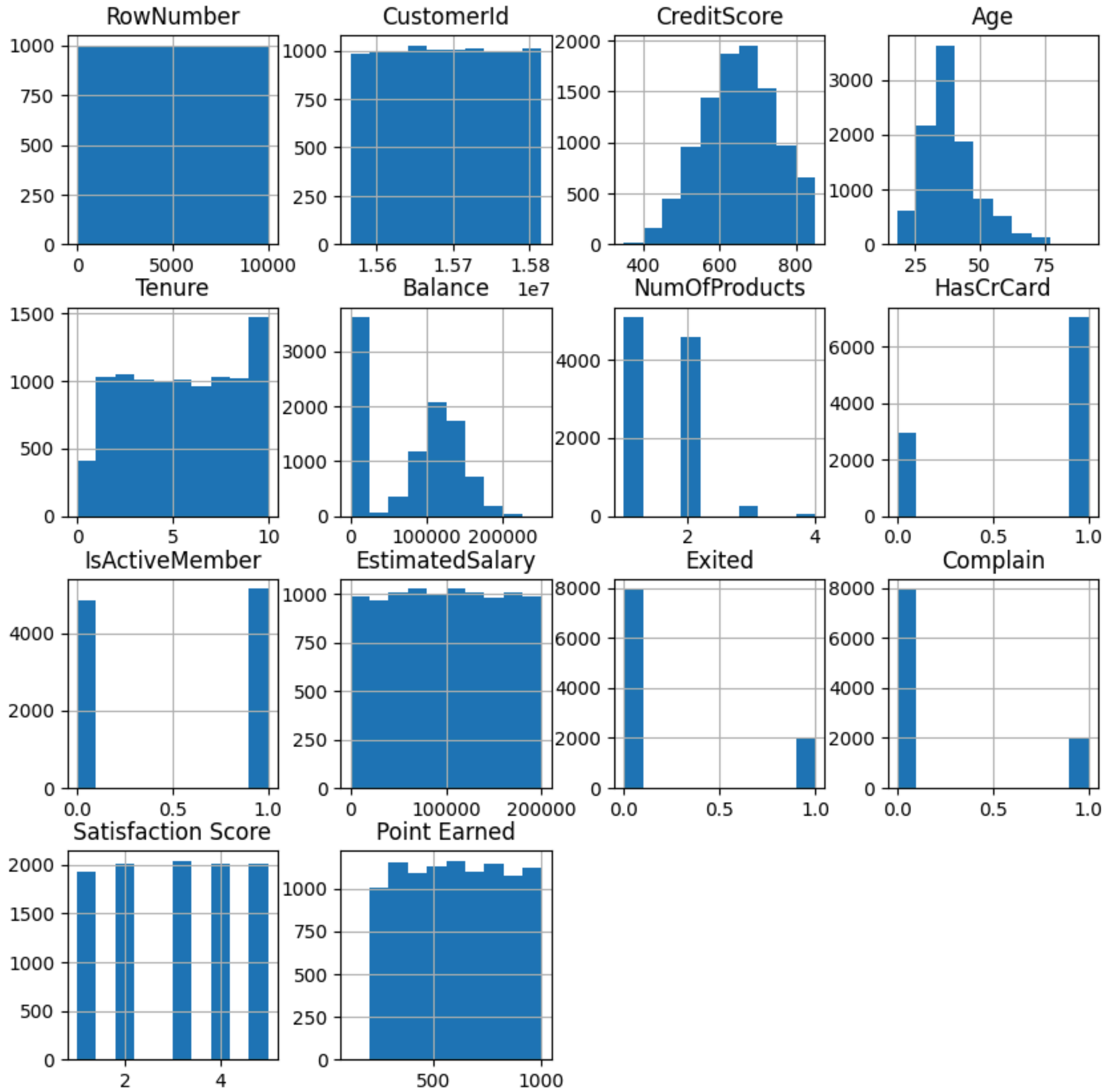
10000 rows × 18 columns

```
df.duplicated().sum()
```

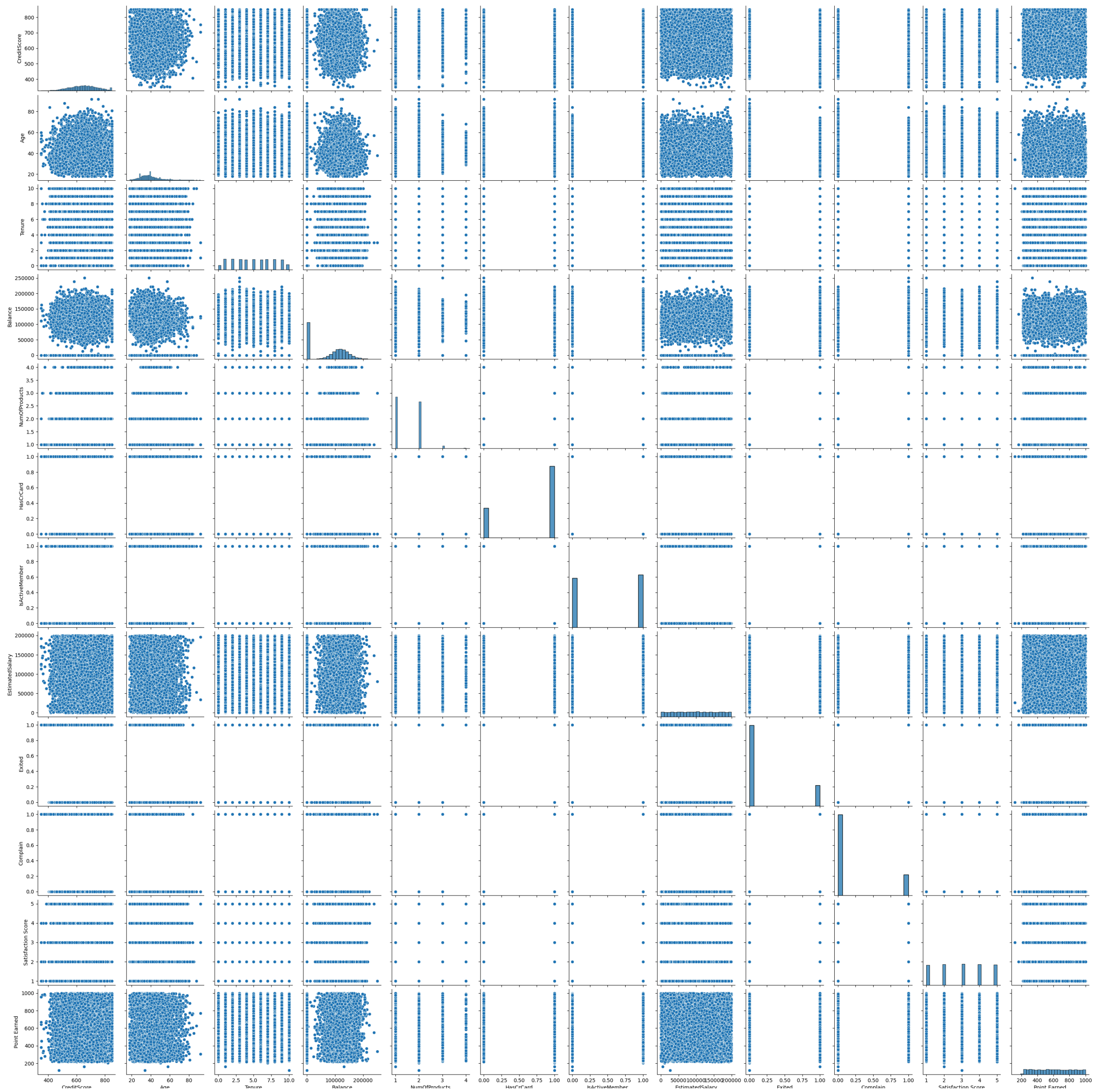
```
np.int64(0)
```

```
df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, inplace=True)
```

```
#histogram chart
df.hist(figsize=(10,10))
plt.show()
```




```
#bivariate analysis
sns.pairplot(df)
plt.show()
```




```
#feature engineering
for col in ['Geography', 'Gender', 'Card Type']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

```
df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Comp
0	619	0	0	42	2	0.00	1	1	1	101348.88	1	
1	608	2	0	41	1	83807.86	1	0	1	112542.58	0	
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1	
3	699	0	0	39	1	0.00	2	0	0	93826.63	0	
4	850	2	0	43	2	125510.82	1	1	1	79084.10	0	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	771	0	1	39	5	0.00	2	1	0	96270.64	0	
9996	516	0	1	35	10	57369.61	1	1	1	101699.77	0	
9997	709	0	0	36	7	0.00	1	0	1	42085.58	1	
9998	772	1	1	42	3	75075.31	2	1	0	92888.52	1	
9999	792	0	0	28	4	130142.79	1	1	0	38190.78	0	

10000 rows × 15 columns

```
#scalar standardization
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Comp
0	619	0	0	42	2	0.00	1	1	1	101348.88	1	
1	608	2	0	41	1	83807.86	1	0	1	112542.58	0	
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1	
3	699	0	0	39	1	0.00	2	0	0	93826.63	0	
4	850	2	0	43	2	125510.82	1	1	1	79084.10	0	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	771	0	1	39	5	0.00	2	1	0	96270.64	0	
9996	516	0	1	35	10	57369.61	1	1	1	101699.77	0	
9997	709	0	0	36	7	0.00	1	0	1	42085.58	1	
9998	772	1	1	42	3	75075.31	2	1	0	92888.52	1	
9999	792	0	0	28	4	130142.79	1	1	0	38190.78	0	

10000 rows × 15 columns

```
#label encoding and onehot encoding
df_encoded = pd.get_dummies(df, columns=['Geography', 'Gender', 'Card Type'])
```

```
df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Comp
0	619	0	0	42	2	0.00	1	1	1	101348.88	1	
1	608	2	0	41	1	83807.86	1	0	1	112542.58	0	
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1	
3	699	0	0	39	1	0.00	2	0	0	93826.63	0	
4	850	2	0	43	2	125510.82	1	1	1	79084.10	0	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	771	0	1	39	5	0.00	2	1	0	96270.64	0	
9996	516	0	1	35	10	57369.61	1	1	1	101699.77	0	
9997	709	0	0	36	7	0.00	1	0	1	42085.58	1	
9998	772	1	1	42	3	75075.31	2	1	0	92888.52	1	
9999	792	0	0	28	4	130142.79	1	1	0	38190.78	0	

10000 rows × 15 columns

```
#model building
X = df.drop('Exited', axis=1)
y = df['Exited']
```

```
#import model
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
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(
```

▾ LogisticRegression

LogisticRegression()

```
#prediction
y_pred = model.predict(x_test)
print("y_prediction", y_pred)
```

```
y_prediction [0 0 0 ... 0 0 0]
```

```
#random forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(x_train, y_train)
y_random_prediction = model.predict(x_test)
print("y_prediction", y_random_prediction)
```

```
y_prediction [0 0 0 ... 1 1 1]
```

```
# Evaluate
y_pred = model.predict(x_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

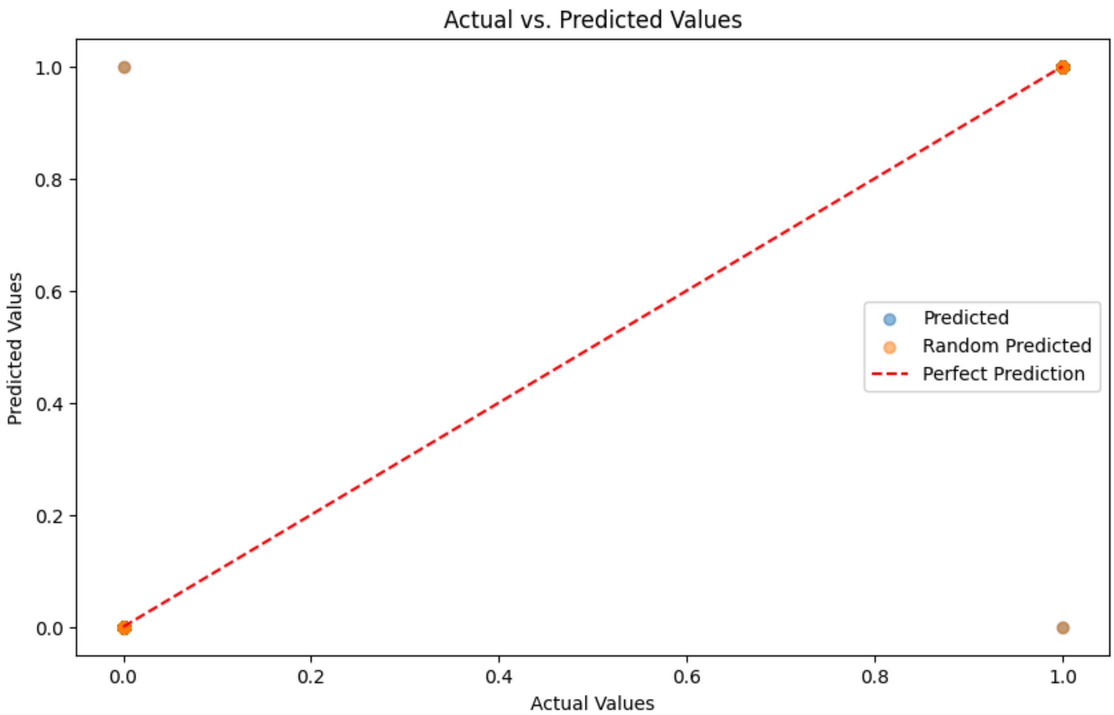
```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1607
           1       1.00      1.00      1.00       393

    accuracy                           1.00      2000
```

```
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000

Confusion Matrix:
 [[1606    1]
 [   1  392]]
```

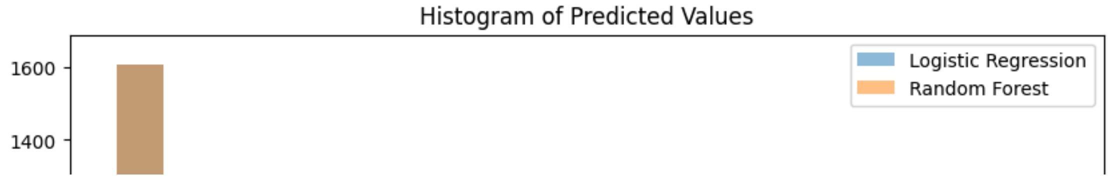
```
# Evaluate
y_random_prediction = model.predict(x_test)
print("Classification Report:\n", classification_report(y_test, y_random_prediction))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_random_prediction))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1607
           1       1.00      1.00      1.00       393

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000

Confusion Matrix:
 [[1606    1]
 [   1  392]]
```

```
#visualize prediction and actual value
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5, label='Predicted')
plt.scatter(y_test, y_random_prediction, alpha=0.5, label='Random Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red', label='Perfect Prediction')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```




```
#histogram chart random forest and logistic regression
plt.figure(figsize=(10, 6))
plt.hist(y_pred, bins=20, alpha=0.5, label='Logistic Regression')
plt.hist(y_random_prediction, bins=20, alpha=0.5, label='Random Forest')
plt.xlabel('Predicted Values')
plt.ylabel('Frequency')
plt.title('Histogram of Predicted Values')
plt.legend()
plt.show()
```


Histogram of Predicted Values
1600
1400
Logistic Regression
Random Forest