```
#libraries importing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data =pd.read_csv("/content/Bank Customer Churn Prediction.csv")
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
data.drop_duplicates(inplace=True)
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
data.columns
```

```
Index(['customer_id', 'credit_score', 'country', 'gender', 'age', 'tenure',
       'balance', 'products_number', 'credit_card', 'active_member',
       'estimated_salary', 'churn'],
      dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       10000 non-null  int64
 1   credit_score      10000 non-null  int64
 2   country           10000 non-null  object
 3   gender            10000 non-null  object
 4   age               10000 non-null  int64
 5   tenure            10000 non-null  int64
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64
 8   credit_card       10000 non-null  int64
 9   active_member     10000 non-null  int64
 10  estimated_salary  10000 non-null  float64
```

```
 11  churn             10000 non-null  int64
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

```
#finding missing Values
data.isnull().sum()
```

	0
customer_id	0
credit_score	0
country	0
gender	0
age	0
tenure	0
balance	0
products_number	0
credit_card	0
active_member	0
estimated_salary	0
churn	0

dtype: int64

```
#dropping missing values
data.dropna()
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
#filling the null values
data["credit_score"].fillna(data["credit_score"].median(), inplace=True)
data["age"].fillna(data["age"].mean(), inplace=True)
data["balance"].fillna(data["balance"].mean(), inplace=True)
```

```
<ipython-input-59-d79f7ec909dd>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead,


  data["credit_score"].fillna(data["credit_score"].median(), inplace=True)
<ipython-input-59-d79f7ec909dd>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead,


  data["age"].fillna(data["age"].mean(), inplace=True)
<ipython-input-59-d79f7ec909dd>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead,


  data["balance"].fillna(data["balance"].mean(), inplace=True)
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
data.isnull().sum()
```

	0
customer_id	0
credit_score	0
country	0
gender	0
age	0
tenure	0
balance	0
products_number	0
credit_card	0
active_member	0
estimated_salary	0
churn	0

dtype: int64

```
#removing duplicates
data.drop_duplicates(inplace=True)
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
data.head()
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       10000 non-null  int64
 1   credit_score      10000 non-null  int64
 2   country           10000 non-null  object
 3   gender            10000 non-null  object
 4   age               10000 non-null  int64
 5   tenure            10000 non-null  int64
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64
 8   credit_card       10000 non-null  int64
 9   active_member     10000 non-null  int64
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

```
data.describe()
```

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

```
data.isnull().sum()
```

	0
customer_id	0
credit_score	0
country	0
gender	0
age	0
tenure	0
balance	0
products_number	0
credit_card	0
active_member	0
estimated_salary	0
churn	0

dtype: int64

```
data.drop_duplicates()
```

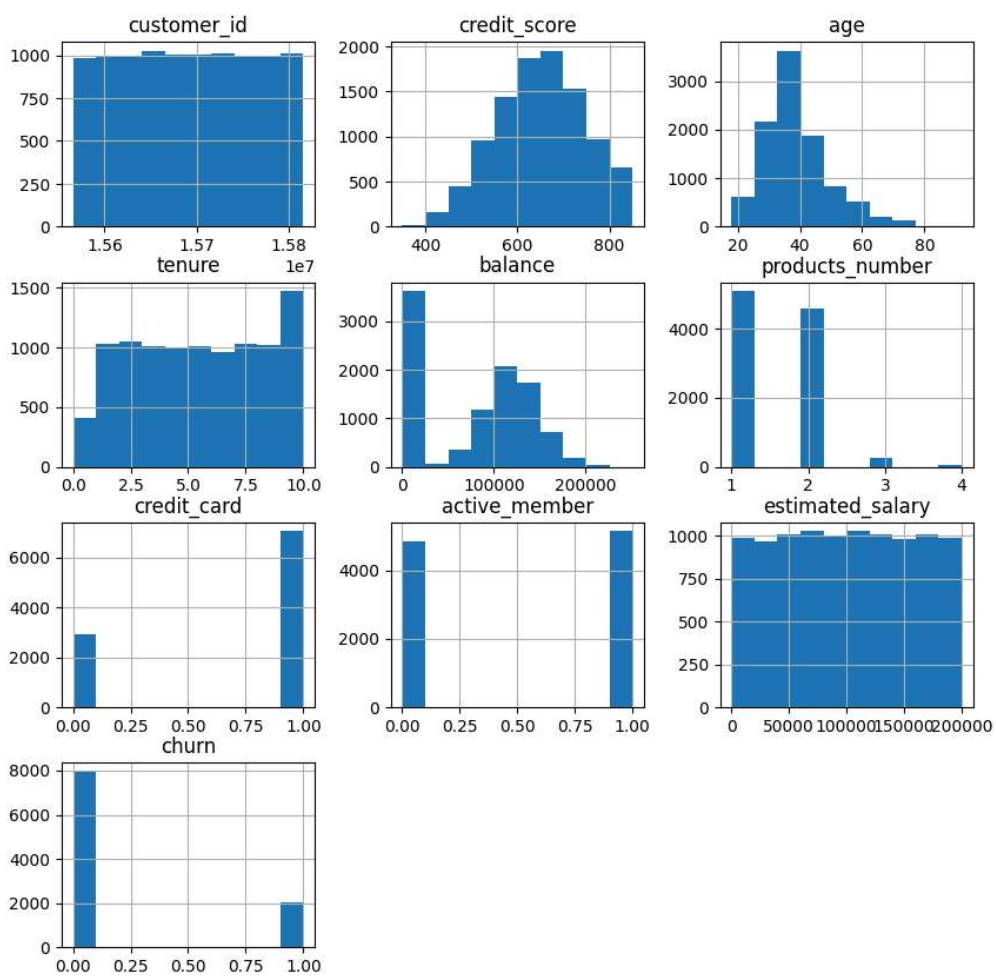
	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

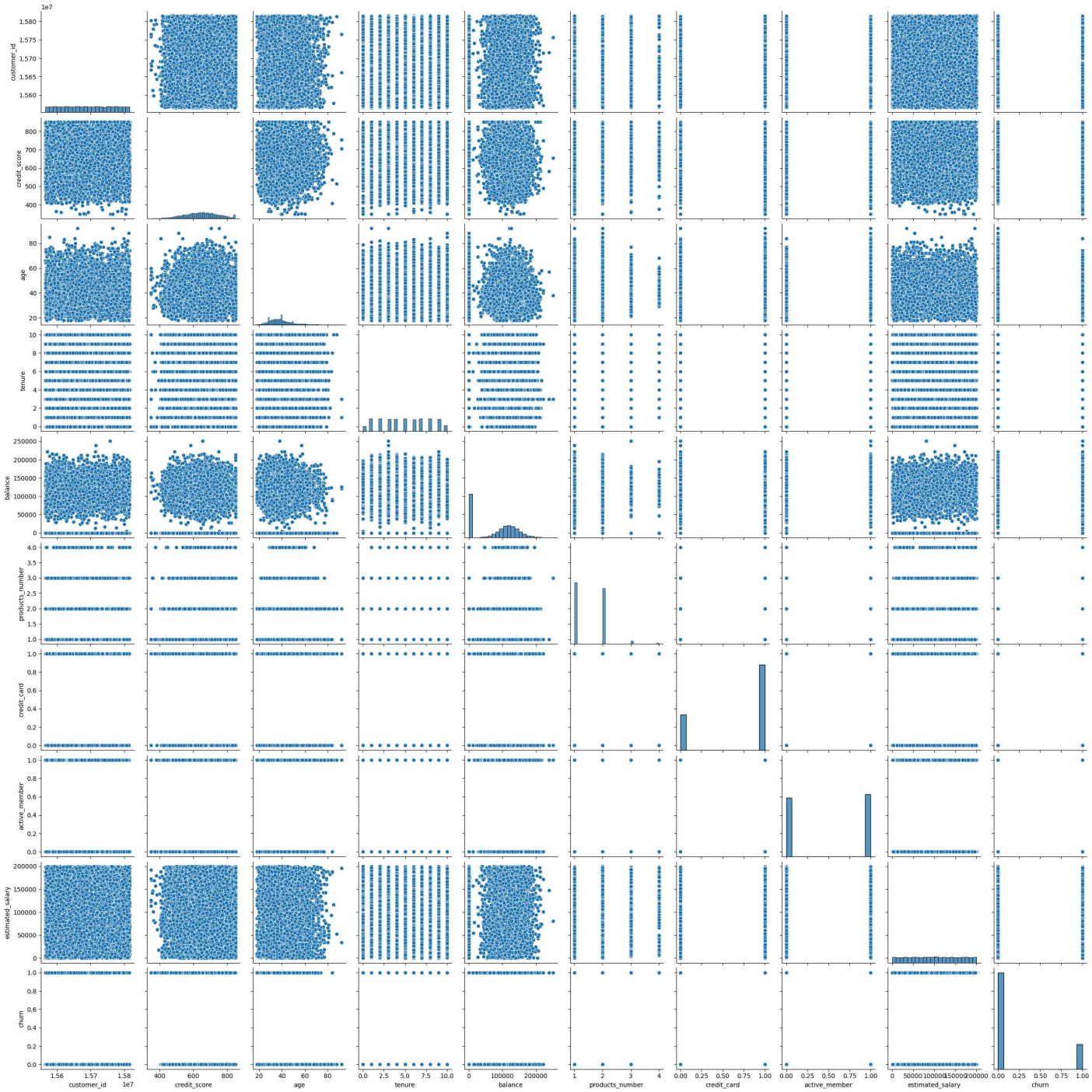
```
data.duplicated().sum()
```

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

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




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


customer_id
credit_score
age
tenure
balance
products_number
credit_card
active_member
estimated_salary
churn
1e7

```
from sklearn.preprocessing import LabelEncoder
```

```
#feature engineering
for col in ['country','gender','churn']:
  le = LabelEncoder()
  data[col] = le.fit_transform(data[col])
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	2	0	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	0	1	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	0	1	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	0	0	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	1	1	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	0	0	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
from sklearn.preprocessing import StandardScaler
```

scalar standardization

scaler = StandardScaler() data_scaled = scaler.fit_transform(data)

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	2	0	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	0	1	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	0	1	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	0	0	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	1	1	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	0	0	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
#label encoding and onehot encoding
data_encoded = pd.get_dummies(data, columns=['country','gender','churn'])
```

```
data
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	2	0	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	15606229	771	0	1	39	5	0.00	2	1	0	96270.64	0
9996	15569892	516	0	1	35	10	57369.61	1	1	1	101699.77	0
9997	15584532	709	0	0	36	7	0.00	1	0	1	42085.58	1
9998	15682355	772	1	1	42	3	75075.31	2	1	0	92888.52	1
9999	15628319	792	0	0	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

```
#model building
x = data.drop('credit_card', axis=1)
y = data['credit_card']
```

```
#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)
```

```
LogisticRegression
LogisticRegression()
```

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

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

```
#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 [1 1 1 ... 1 1 1]
```

```
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       0.25      0.03      0.05       573
               1       0.71      0.97      0.82      1427

        accuracy                           0.70      2000
       macro avg       0.48      0.50      0.43      2000
    weighted avg       0.58      0.70      0.60      2000

    Confusion Matrix:
     [[  15  558]
     [  44 1383]]
```

```
# 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))
```

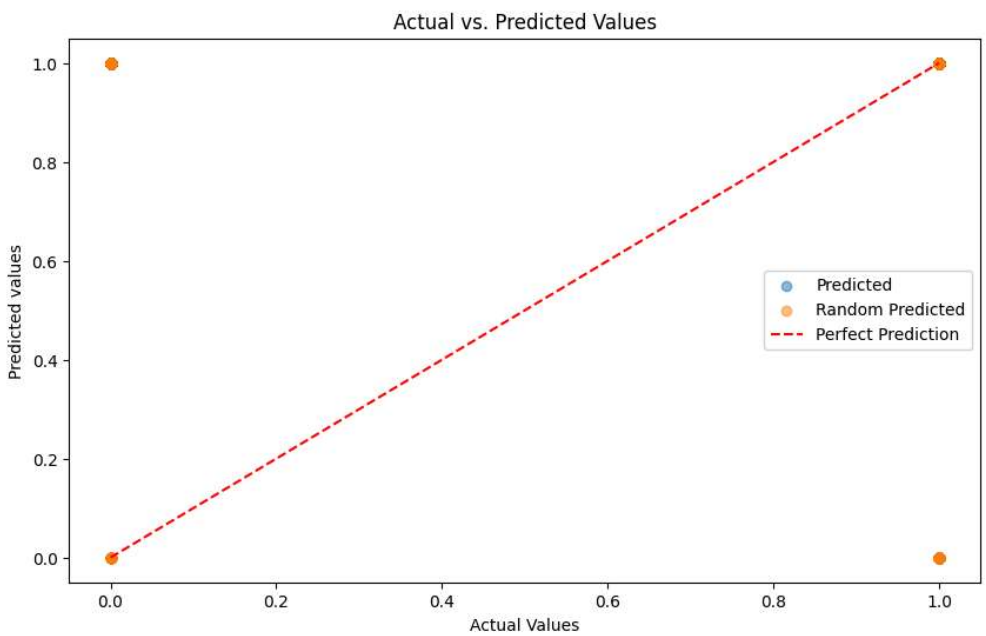
```
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.03      0.05       573
           1       0.71      0.97      0.82      1427

    accuracy                           0.70      2000
   macro avg       0.48      0.50      0.43      2000
weighted avg       0.58      0.70      0.60      2000

Confusion Matrix:
 [[  15  558]
 [  44 1383]]
```

```
#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()
```