Credit Score Prediction

Features

- **ID:** Represents a unique identification of an entry
- **Customer_ID:** Represents a unique identification of a person
- **Month:** Represents the month of the year
- **Name:** Represents the name of a person
- **Age:** Represents the age of the person
- **SSN:** Represents the social security number of a person
- **Occupation:** Represents the occupation of the person
- **Annual Income:** Represents the annual income of the person
- **Monthly Inhand Salary:** Represents the monthly base salary of a person
- **Num Bank Accounts:** Represents the number of bank accounts a person holds
- **Num Credit Card:** Represents the number of other credit cards held by a person
- **Interest_Rate:** Represents the interest rate on credit card
- **Num of Loan:** Represents the number of loans taken from the bank
- **Type of Loan:** Represents the types of loan taken by a person
- **Delay from due date:** Represents the average number of days delayed from the payment date
- **Num of Delayed Payment:** Represents the average number of payments delayed by a person
- **Changed Credit Limit:** Represents the percentage change in credit card limit
- **Num Credit Inquiries:** Represents the number of credit card inquiries
- **Credit Mix:** Represents the classification of the mix of credits
- **Outstanding Debt:** Represents the remaining debt to be paid (in USD)
- **Credit Utilization Ratio:** Represents the utilization ratio of credit card
- **Credit History Age:** Represents the age of credit history of the person
- **Payment of Min Amount:** Represents whether only the minimum amount was paid by the person
- **Total EMI per month:** Represents the monthly EMI payments (in USD)
- **Amount invested monthly:** Represents the monthly amount invested by the customer (in USD)
- **Payment Behaviour:** Represents the payment behavior of the customer (in USD)
- **Monthly_Balance:** Represents the monthly balance amount of the customer (in USD)
- **Credit Score:** Represents the bracket of credit score (Poor, Standard, Good)

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,ExtraTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import (
    RandomForestClassifier,
    GradientBoostingClassifier,
    VotingClassifier,
)
from sklearn.feature_selection import SelectKBest, RFE, chi2, mutual_info_cl
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    accuracy_score,
    roc_curve,
    auc,
    confusion_matrix
)
from sklearn.decomposition import PCA
from sklearn.ensemble import VotingClassifier, StackingClassifier

warnings.filterwarnings("ignore")
pd.set_option("display.max_columns", None)
```

In [3]:
```
data = pd.read_csv("train.csv")
data.shape
```

Out[3]:
```
(100000, 28)
```

In [4]:
```
dict = {"January" : 1,"February" : 2,"March" : 3,"April" : 4,"May" : 5,"June
data["Month"] = data["Month"].map(dict)
data.head()
```

Out[4]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthl
0	0x1602	CUS_0xd40	1	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
1	0x1603	CUS_0xd40	2	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
2	0x1604	CUS_0xd40	3	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	
3	0x1605	CUS_0xd40	4	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
4	0x1606	CUS_0xd40	5	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	

In [5]:

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

Out[5]:
```
ID                             0
Customer_ID                    0
Month                          0
Name                        9985
Age                            0
SSN                            0
Occupation                     0
Annual_Income                  0
Monthly_Inhand_Salary      15002
Num_Bank_Accounts              0
Num_Credit_Card                0
Interest_Rate                  0
Num_of_Loan                    0
Type_of_Loan               11408
Delay_from_due_date            0
Num_of_Delayed_Payment      7002
Changed_Credit_Limit           0
Num_Credit_Inquiries        1965
Credit_Mix                     0
Outstanding_Debt               0
Credit_Utilization_Ratio       0
Credit_History_Age          9030
Payment_of_Min_Amount          0
Total_EMI_per_month            0
Amount_invested_monthly     4479
Payment_Behaviour              0
Monthly_Balance             1200
Credit_Score                   0
dtype: int64
```

In [6]:
```
data.describe()
```

Out[6]:

	Month	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Int
count	100000.000000	84998.000000	100000.000000	100000.00000	1000
mean	4.500000	4194.170850	17.091280	22.47443	
std	2.291299	3183.686167	117.404834	129.05741	4
min	1.000000	303.645417	-1.000000	0.00000	
25%	2.750000	1625.568229	3.000000	4.00000	
50%	4.500000	3093.745000	6.000000	5.00000	
75%	6.250000	5957.448333	7.000000	7.00000	
max	8.000000	15204.633333	1798.000000	1499.00000	57

In [7]:
```
columns_with_underscore = [col for col in data.columns if any("_" in str(val
columns_with_underscore
```

Out[7]:
```
['Customer_ID',
 'Age',
 'Occupation',
 'Annual_Income',
 'Num_of_Loan',
 'Num_of_Delayed_Payment',
 'Changed_Credit_Limit',
 'Credit_Mix',
 'Outstanding_Debt',
 'Amount_invested_monthly',
 'Payment_Behaviour',
 'Monthly_Balance']
```

Data Cleaning

In [8]:
```
def remove_underscore(col):
    data[col] = data[col].apply(lambda x: str(x).replace("_", "") if str(x)
    data[col] = pd.to_numeric(data[col], errors="coerce")

data["Num_of_Loan"].fillna("-100")
data["Num_of_Delayed_Payment"].fillna("-1")

remove_underscore("Age")
remove_underscore("Num_of_Delayed_Payment")
remove_underscore("Changed_Credit_Limit")
remove_underscore("Outstanding_Debt")
remove_underscore("Amount_invested_monthly")
remove_underscore("Monthly_Balance")
```

In [9]:
```
dict = {
    'High_spent_Small_value_payments' : 0,
    'Low_spent_Large_value_payments' : 1,
    'Low_spent_Medium_value_payments' : 2,
    'Low_spent_Small_value_payments' : 3,
    'High_spent_Medium_value_payments' : 4,
    'High_spent_Large_value_payments': 5,
    '!@9#%8' : np.nan
}

data['Payment_Behaviour'] = data['Payment_Behaviour'].map(dict)
```

Finding mean, mode and filling the missing values for a person

In [10]:

```
def find_mean(i, col):
    mean = 0
    j = i
    while j != i + 8:
        value = data.at[j, col]
        if pd.notna(value) and (np.issubdtype(type(value), np.floating)or np
            mean += float(value)
        j += 1
    return mean / 8

def find_mode(i, col):
    mode = {}
    j = i

    while j != i + 8:
        value = data.at[j, col]
        if pd.notna(value) and (np.issubdtype(type(value), np.floating) or (
            if data.at[j, col] in mode:
                mode[value] += 1
            else:
                mode[value] = 1
        j += 1

    return max(mode, key=mode.get)

def date_to_int(value):
    year = []
    month = []
    i = 0
    flag = 0

    for char in value:
        if char.isnumeric() and not flag:
            year.append(char)
        else:
            flag = 1

        if char.isnumeric() and flag:
            month.append(char)

    result = result = int(''.join(map(str, year))) * 12 +  int(''.join(map(s

    return result
```

In [11]:
```
def fill_missing(i,col,condition):
     index = []
     j = i
     valid = ''
     while (j != i + 8):
         if condition(j,col):
             index.append(j)
         else:
             valid = data.at[j,col]
         j+=1
     for k in index:
         data.at[k,col] = valid

def fill_with_mean(i,col,condition):
    mean = find_mean(i,col)
    j = i
    while (j != i + 8):
         if condition(j,col):
            data.at[j,col] = mean
         j+=1

def fill_with_mode(i,col,condition):
        mode = find_mode(i,col)
        j = i
        while (j != i + 8):
             if condition(j,col):
                data.at[j,col] = mode
             j+=1

def transform_dates(i):
    j = i
    while(j != i + 8):
        data.at[j, "Credit_History_Age"] = date_to_int(data.at[j, "Credit_Hi
        j += 1
```

In [12]:
```
def find_missing():
    for i, _ in data.iterrows():
        if i % 8 == 0:
            fill_missing(i, "Name", lambda j, col: pd.isna(data.at[j, col]))
            fill_missing(i, "Occupation", lambda j, col: "__" in data.at[j,
            fill_missing(i, "Credit_Mix", lambda j, col: "_" in data.at[j, c
            fill_missing(i, "Annual_Income", lambda j, col: "_" in data.at[j
            fill_missing(i, "Type_of_Loan", lambda j, col: pd.isna(data.at[j
            fill_missing(i, "Num_of_Loan", lambda j, col: "-" in data.at[j,
            fill_missing(i, "SSN", lambda j, col: "#" in data.at[j, col])
            fill_missing(i, "Credit_History_Age", lambda j, col: pd.isna(dat
            fill_with_mean(i, "Changed_Credit_Limit", lambda j, col: pd.isna
            fill_with_mean(i, "Monthly_Inhand_Salary", lambda j, col: pd.isn
            fill_with_mean(i, "Delay_from_due_date", lambda j, col: data.at[
            fill_with_mean(i, "Num_of_Delayed_Payment", lambda j, col: pd.is
            fill_with_mean(i, "Num_of_Delayed_Payment", lambda j, col: pd.is
            fill_with_mean(i, "Amount_invested_monthly", lambda j, col: data
            fill_with_mean(i, "Monthly_Balance", lambda j, col: pd.isna(data
            fill_with_mean(i,"Num_Credit_Inquiries",lambda j, col: pd.isna(d
            fill_with_mean(i, "Payment_Behaviour", lambda  j, col: pd.isna(d
            fill_with_mode(i, "Age", lambda j, col: "-" in str(data.at[j, co
            transform_dates(i)

find_missing()
```

In [13]:
```
remove_underscore("Num_of_Loan")
remove_underscore("Annual_Income")
```

In [14]:
```
columns_with_underscore = [col for col in data.columns if any("_" in str(val
columns_with_underscore
```

Out[14]:
```
['Customer_ID', 'Occupation']
```

In [15]:
```
label_encoder = LabelEncoder()

data["Occupation"] = label_encoder.fit_transform(data["Occupation"])
data["Credit_Mix"] = label_encoder.fit_transform(data["Credit_Mix"])
data["Payment_of_Min_Amount"] = label_encoder.fit_transform(data["Payment_of
data["Credit_Score"]=data["Credit_Score"].map({"Standard":0,"Good":1,"Poor":

data.drop("ID", axis=1, inplace=True)
data.drop("Name", axis=1, inplace=True)
data.drop("Customer_ID", axis=1, inplace=True)
data.drop("SSN", axis=1, inplace=True)
data.drop("Type_of_Loan", axis=1, inplace=True)
```

In [16]:
```
data.shape
```

Out[16]:
```
(100000, 23)
```

In [17]:
```
data.head(8)
```

Out[17]:

	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts
0	1	23	12	19114.12	1824.843333	3
1	2	23	12	19114.12	912.421667	3
2	3	23	12	19114.12	912.421667	3
3	4	23	12	19114.12	912.421667	3
4	5	23	12	19114.12	1824.843333	3
5	6	23	12	19114.12	912.421667	3
6	7	23	12	19114.12	1824.843333	3
7	8	23	12	19114.12	1824.843333	3

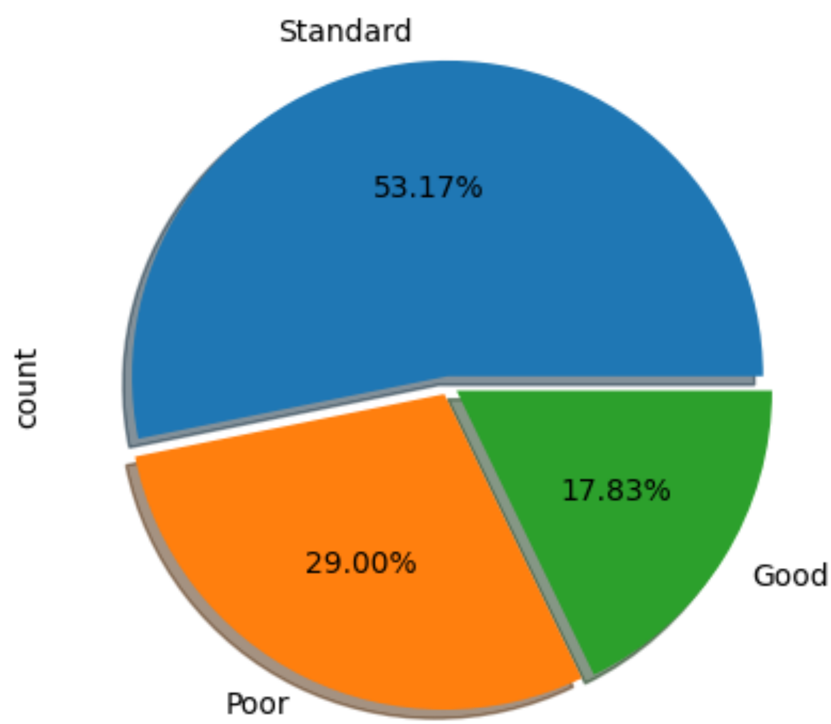
In [18]:
```
data.describe()
```

Out[18]:

	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Sala
count	100000.000000	100000.000000	100000.000000	1.000000e+05	100000.0000(
mean	4.500000	115.373310	6.949840	1.789199e+05	4034.0750:
std	2.291299	683.856594	4.309542	1.441853e+06	3107.5466
min	1.000000	14.000000	0.000000	7.005930e+03	243.5604
25%	2.750000	25.000000	3.000000	1.945751e+04	1571.4425(
50%	4.500000	33.000000	7.000000	3.757975e+04	2990.3175(
75%	6.250000	42.000000	11.000000	7.281702e+04	5746.5616(
max	8.000000	8698.000000	14.000000	2.419806e+07	15204.6333:

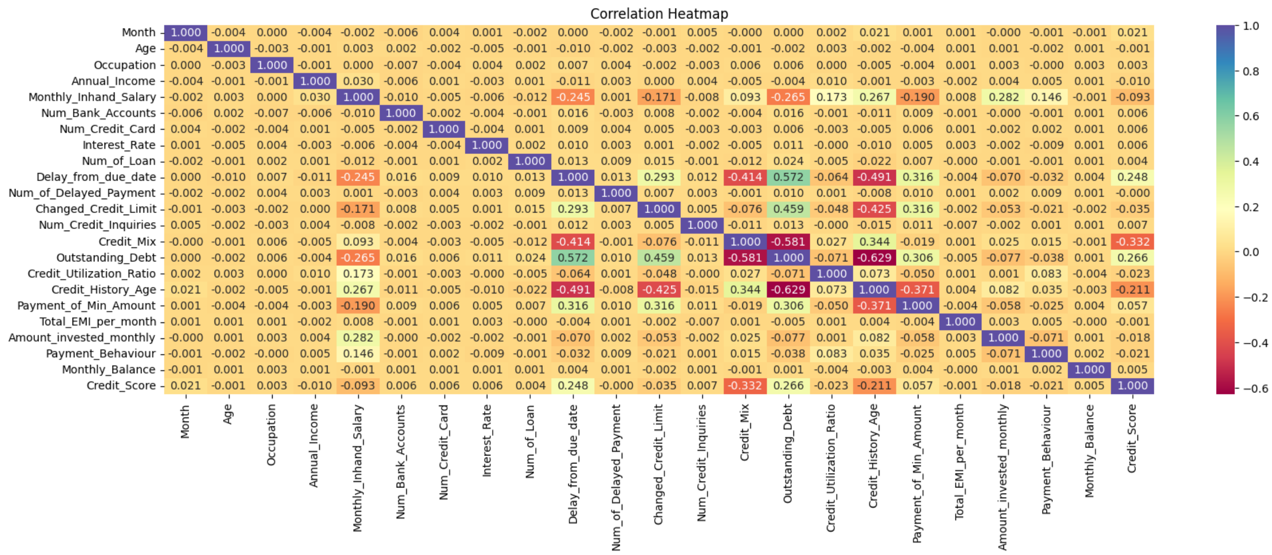
In [19]:
```
data["Credit_Score"].value_counts().plot.pie(explode = [0.03,0.03,0.03], aut
```

Out[19]:
```
<Axes: ylabel='count'>
```



Correlation map without removing outliers

In [20]:
```
def show_heat_map(data):
    correlation_matrix = data.corr()
    plt.figure(figsize=(20, 6))
    sns.heatmap(
        correlation_matrix,
        annot=True,
        cmap="Spectral",
        fmt=".3f",
    )
    plt.title("Correlation Heatmap")

show_heat_map(data)
```



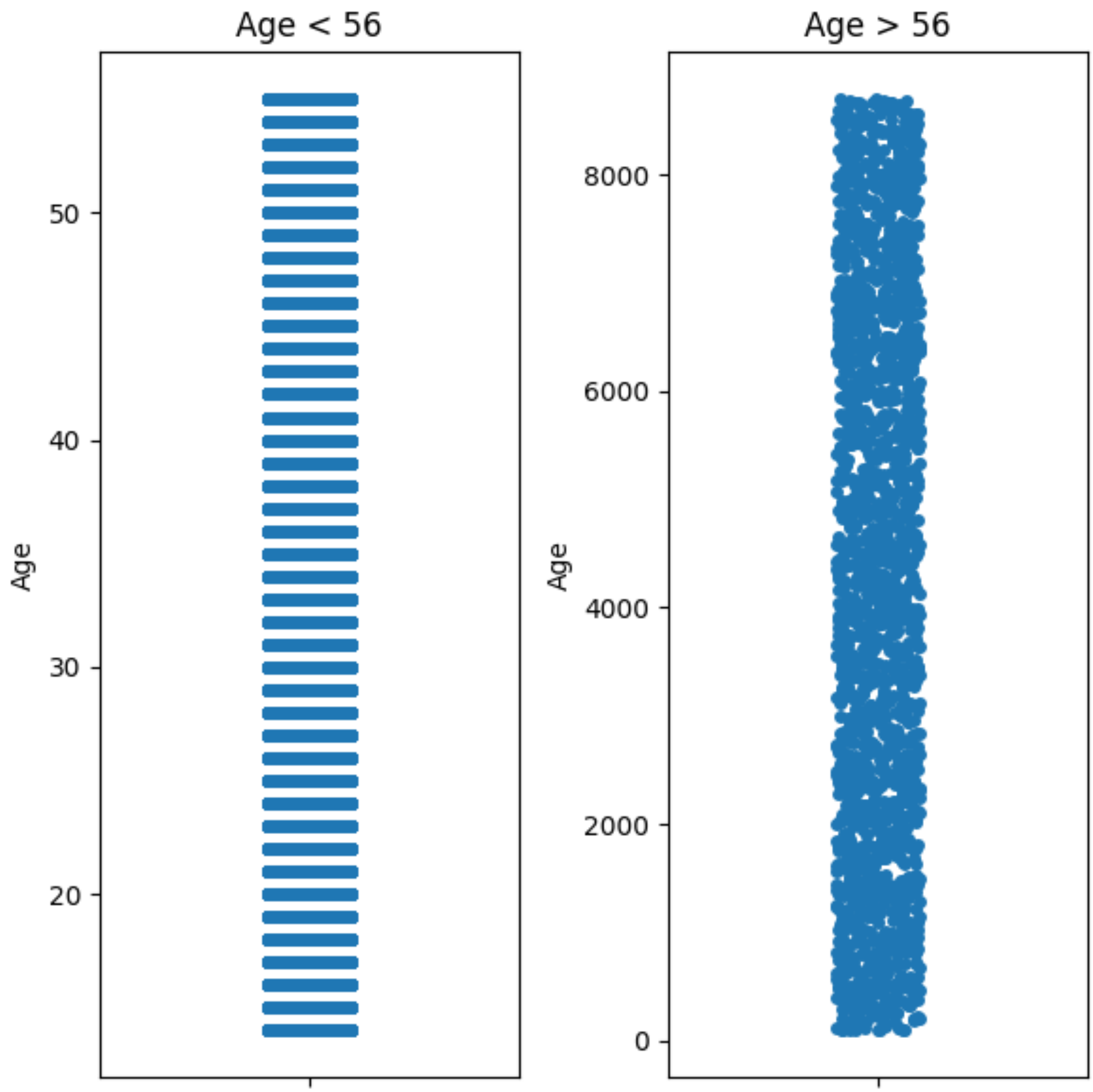
Visualizing the Outliers

In [21]:
```
def show_outliers(data, col, edge):
    fig, axs = plt.subplots(1, 2, figsize=(6, 6))
    sns.stripplot(y=col, data=data[data[col] < edge], ax=axs[0])
    axs[0].set_title(f"{col} < {edge}")
    sns.stripplot(y=col, data=data[data[col] > edge], ax=axs[1])
    axs[1].set_title(f"{col} > {edge}")
    plt.tight_layout()
    plt.show()
```

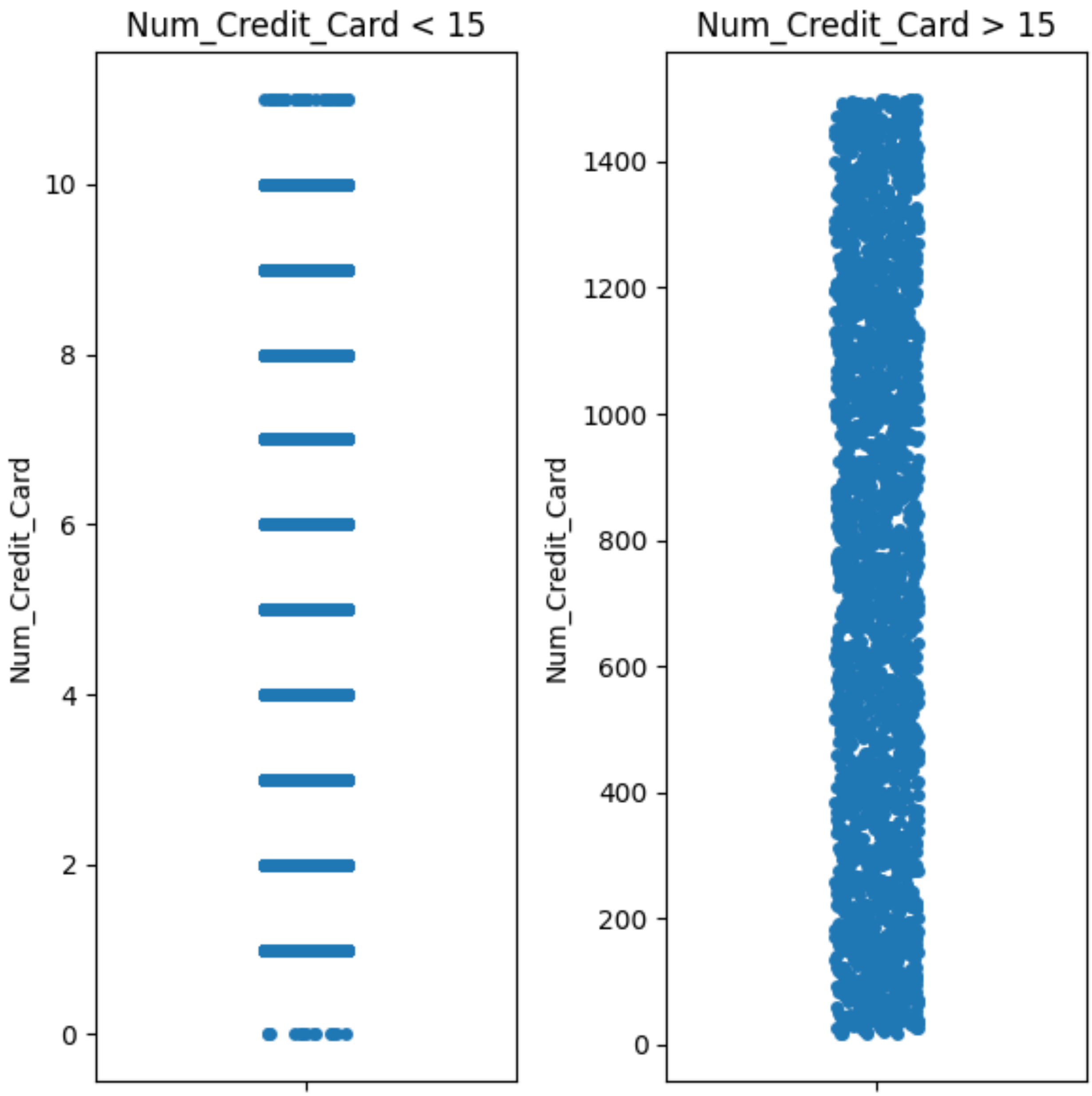
In [22]:
```
def outlier_one(data):
    show_outliers(data,"Age", 56)
    show_outliers(data,"Num_Bank_Accounts", 11)
    show_outliers(data,"Num_Credit_Card", 15)
    show_outliers(data,"Interest_Rate", 35)
```

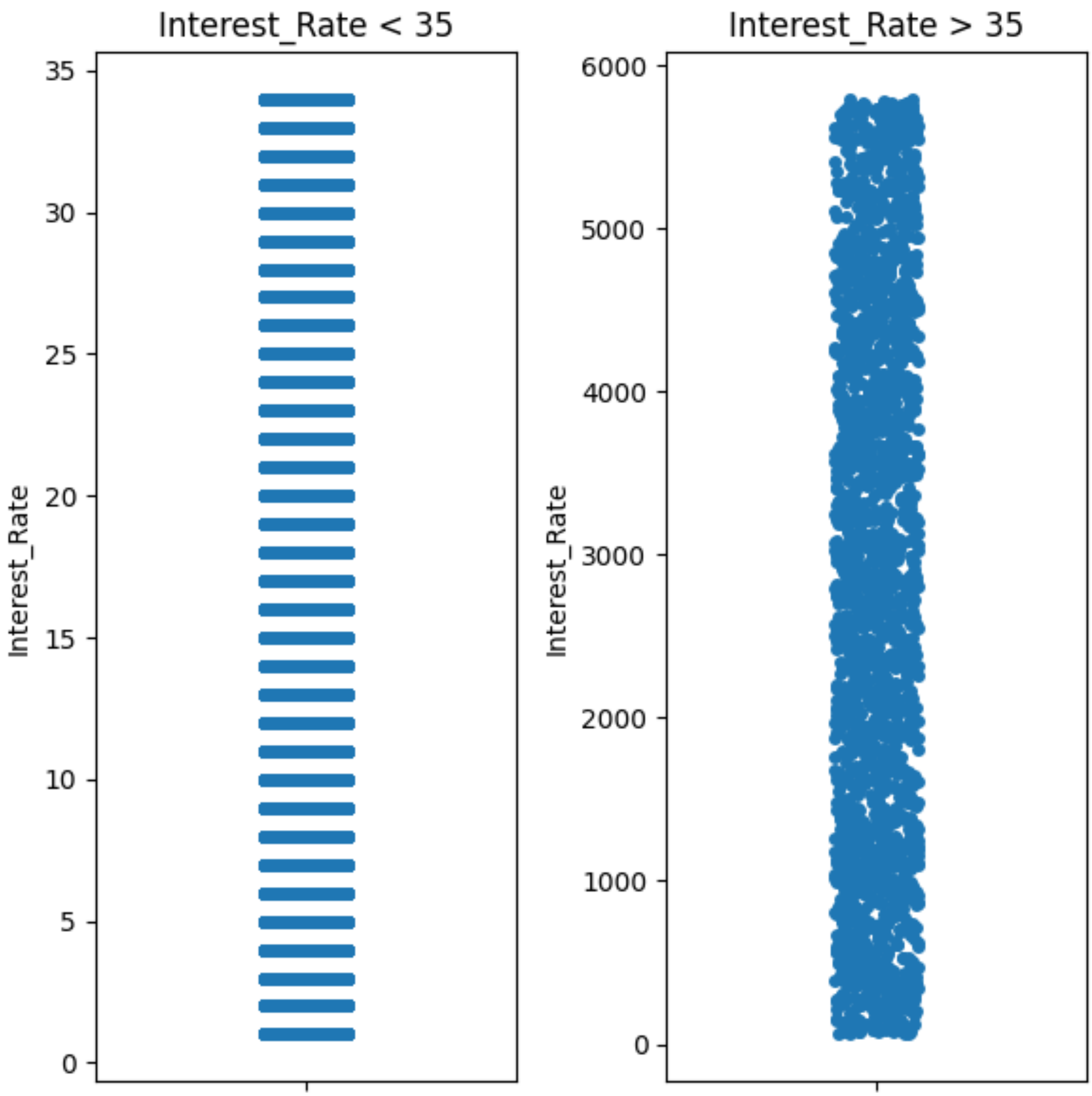
In [23]:
```
def outlier_two(data):
    show_outliers(data,"Num_of_Loan", 10)
    show_outliers(data,"Num_of_Delayed_Payment", 30)
    show_outliers(data,"Num_Credit_Inquiries",27)
    show_outliers(data,"Monthly_Balance",1e-10)
```

In [24]:
```
outlier_one(data)
```

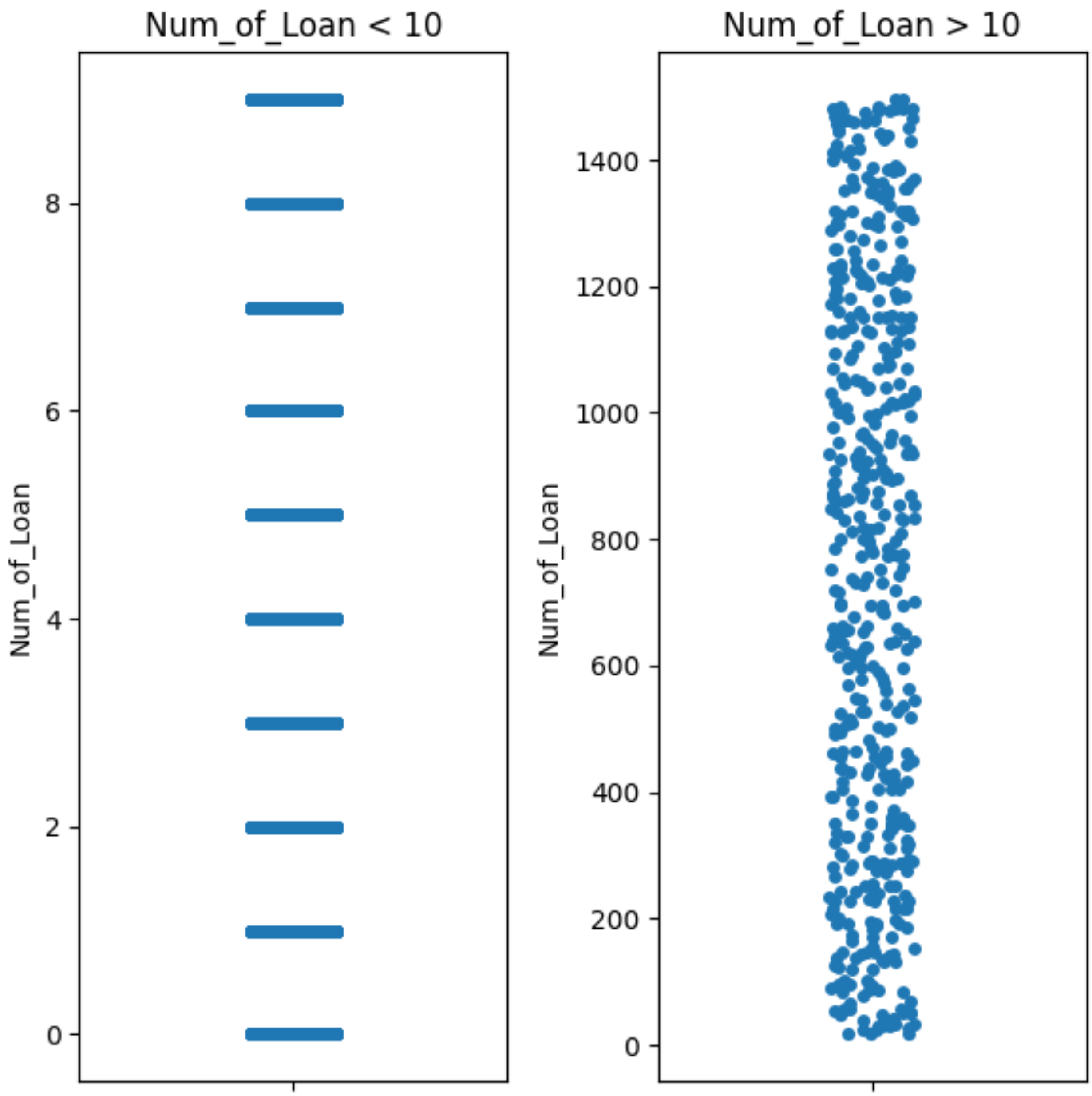

Age < 56
Age > 56
Age
Age
50
40
30
20
8000
6000
4000
2000
0

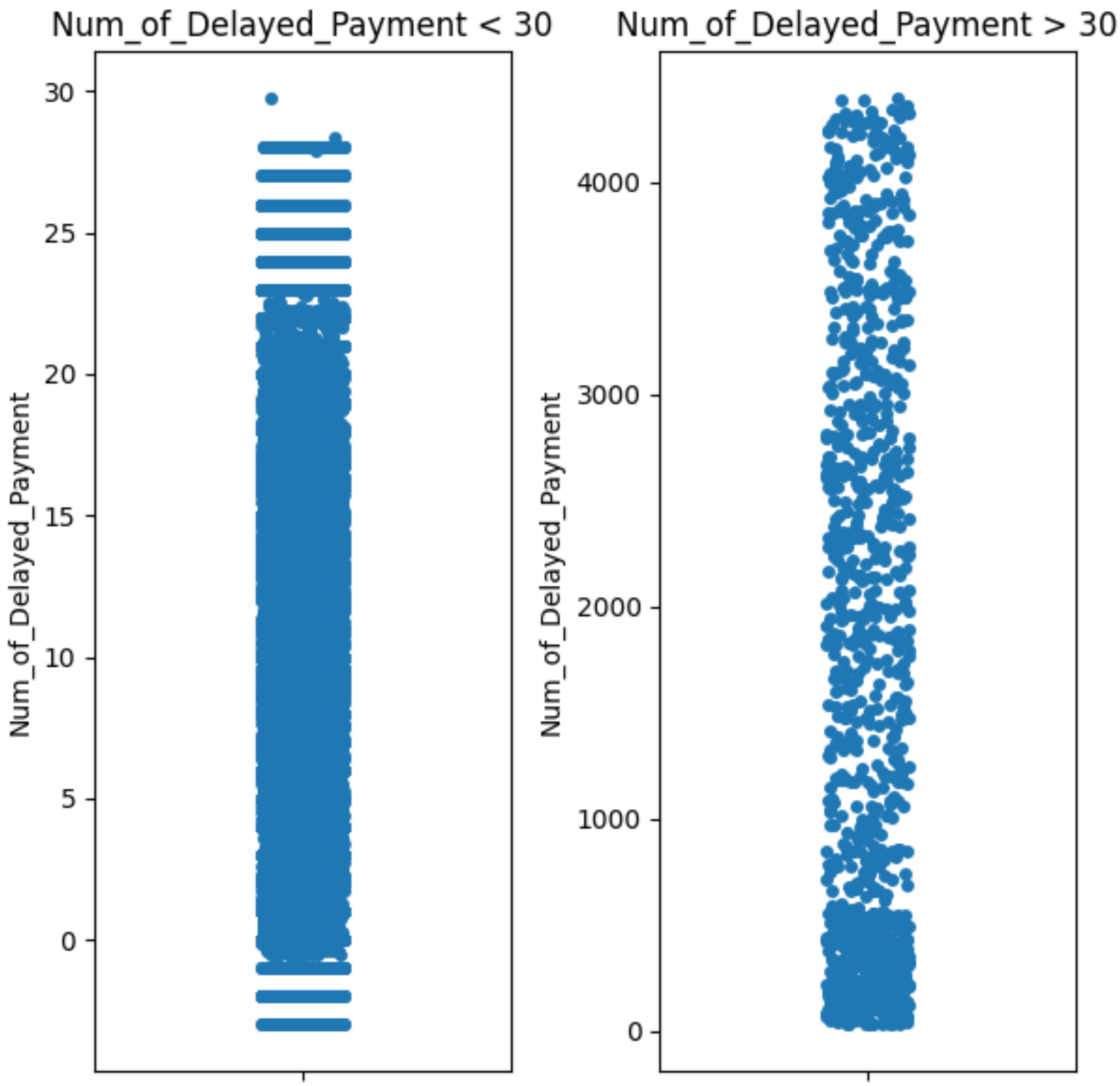

Num_Bank_Accounts < 11
Num_Bank_Accounts > 11
Num_Bank_Accounts
Num_Bank_Accounts
10
8
6
4
2
0
1750
1500
1250
1000
750
500
250
0

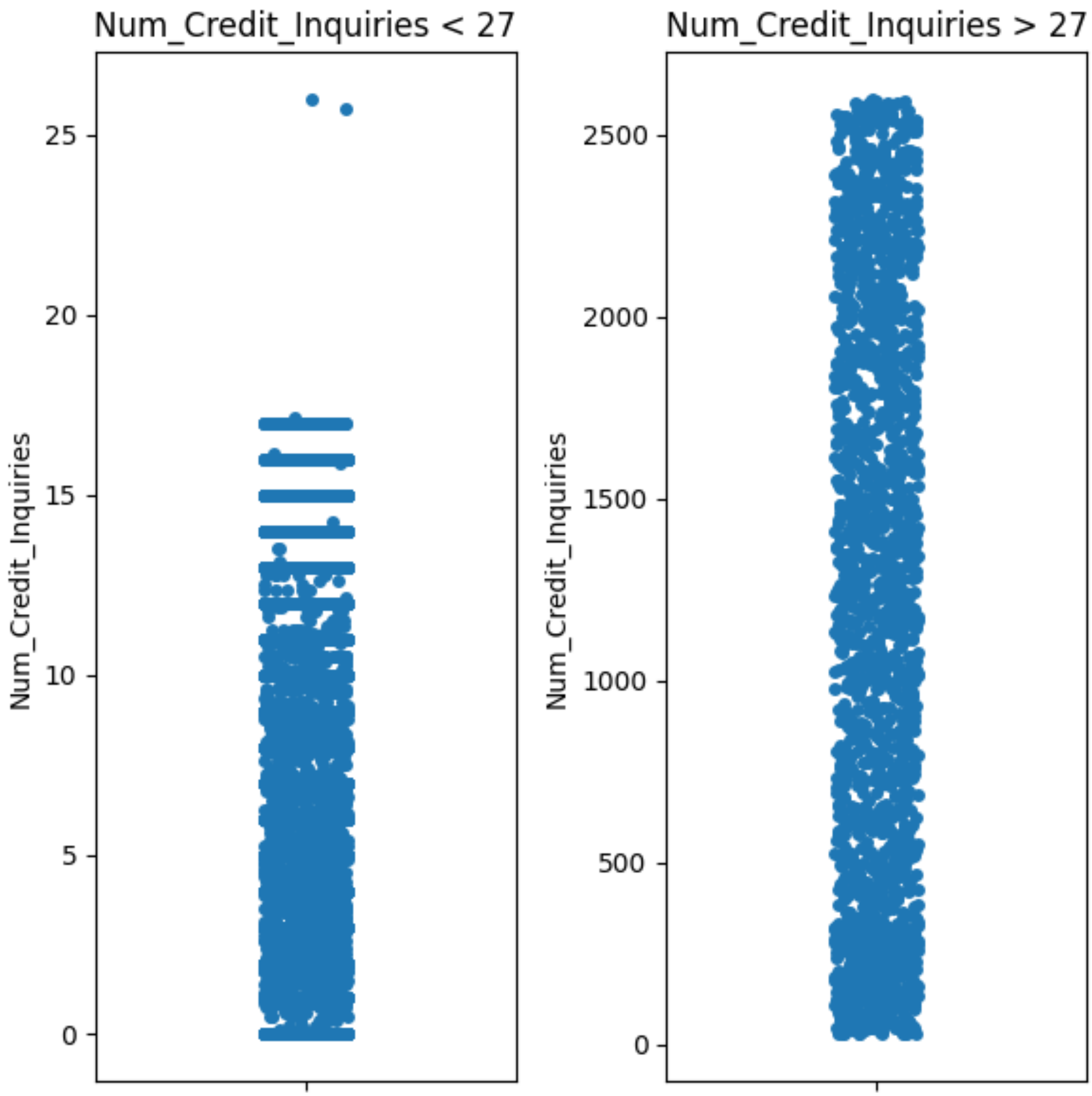

Num_Credit_Card < 15
Num_Credit_Card
0
2
4
6
8
10
Num_Credit_Card > 15
Num_Credit_Card
0
200
400
600
800
1000
1200
1400

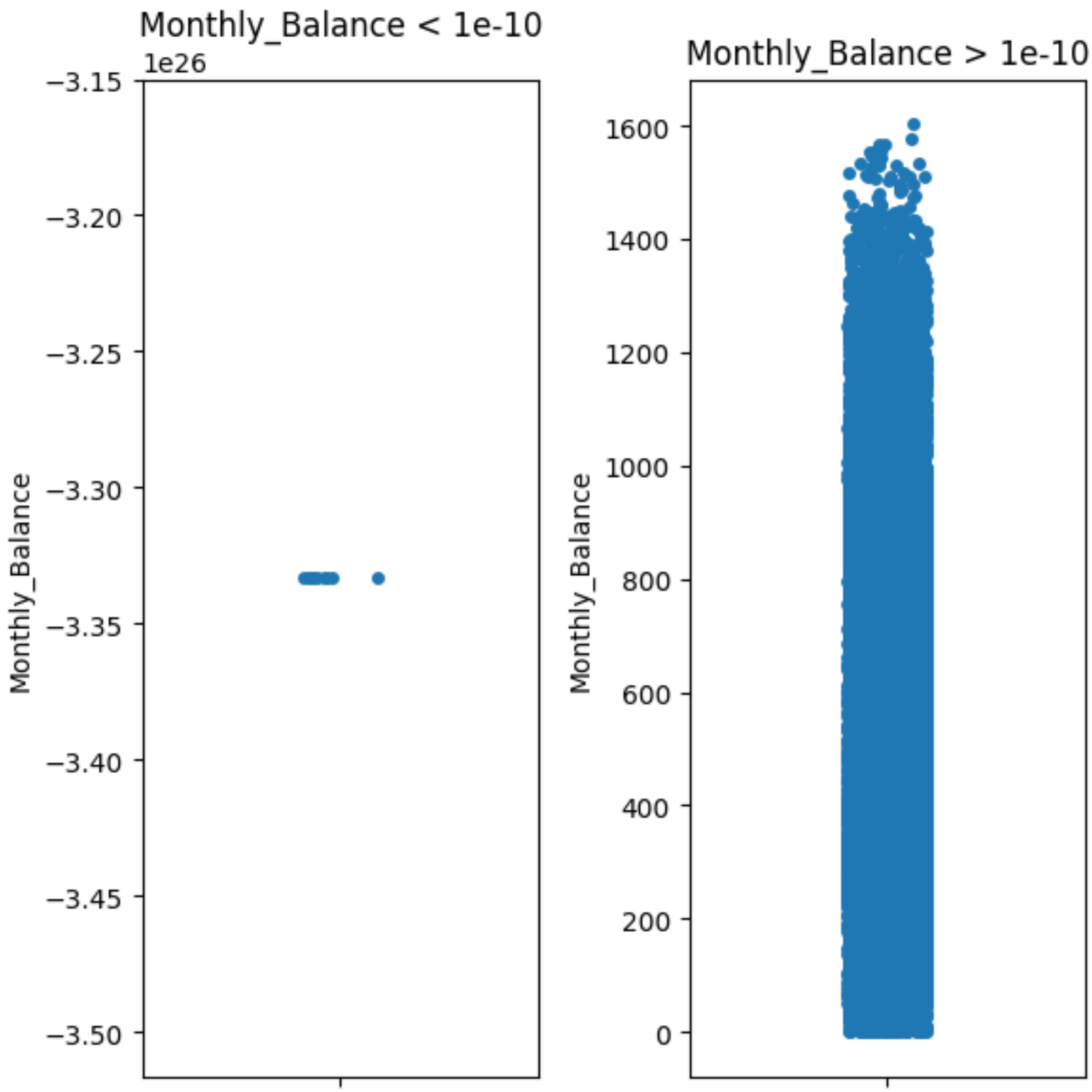


In [25]:
```
outlier_two(data)
```


Num_of_Loan < 10
Num_of_Loan > 10
Num_of_Loan
Num_of_Loan
0
2
4
6
8
0
200
400
600
800
1000
1200
1400


Num_of_Delayed_Payment < 30
Num_of_Delayed_Payment > 30
Num_of_Delayed_Payment
Num_of_Delayed_Payment
30
25
20
15
10
5
0
4000
3000
2000
1000
0


Num_Credit_Inquiries < 27
Num_Credit_Inquiries
0
5
10
15
20
25
Num_Credit_Inquiries > 27
Num_Credit_Inquiries
0
500
1000
1500
2000
2500



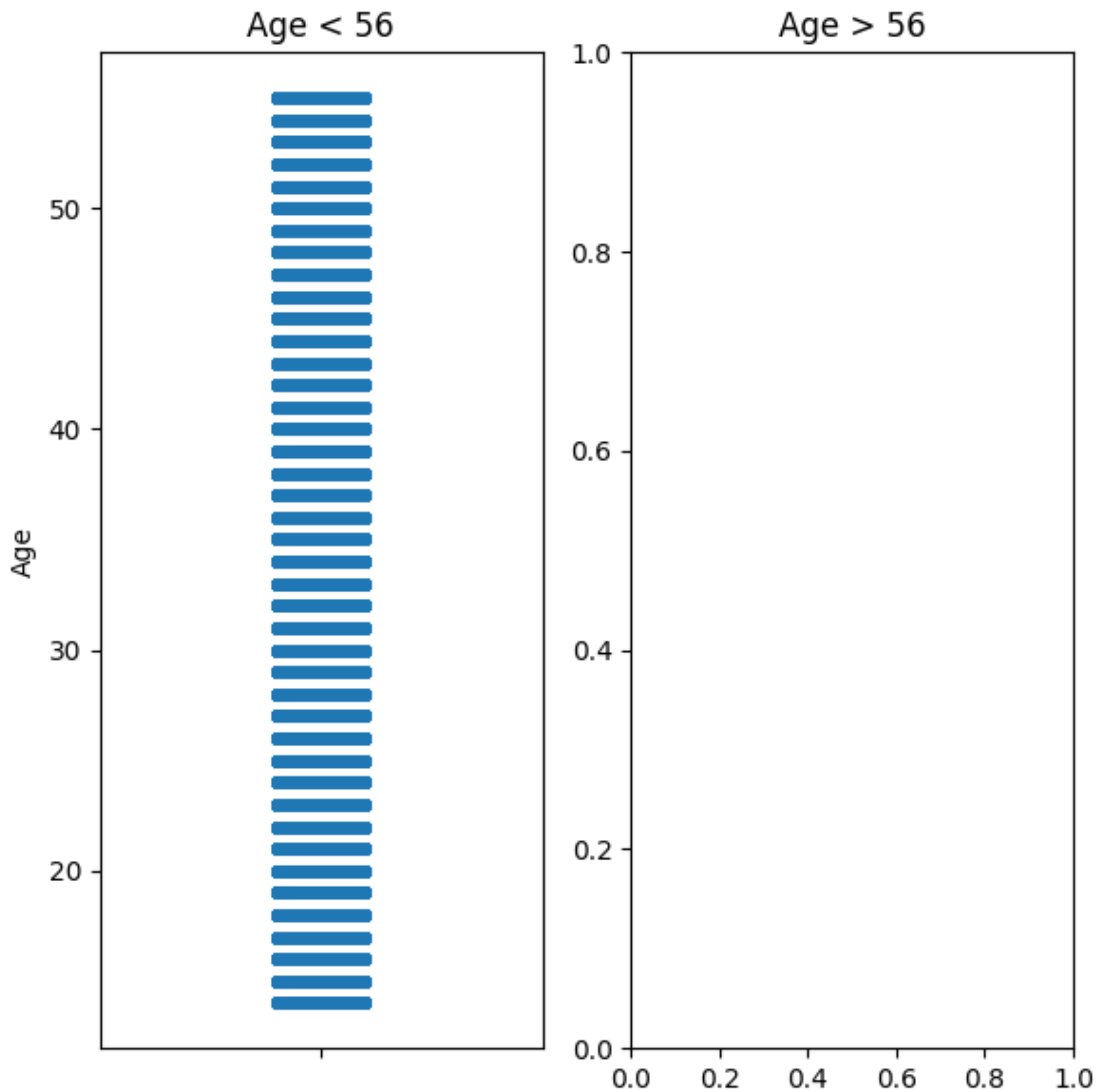
Removing the outliers with IQR

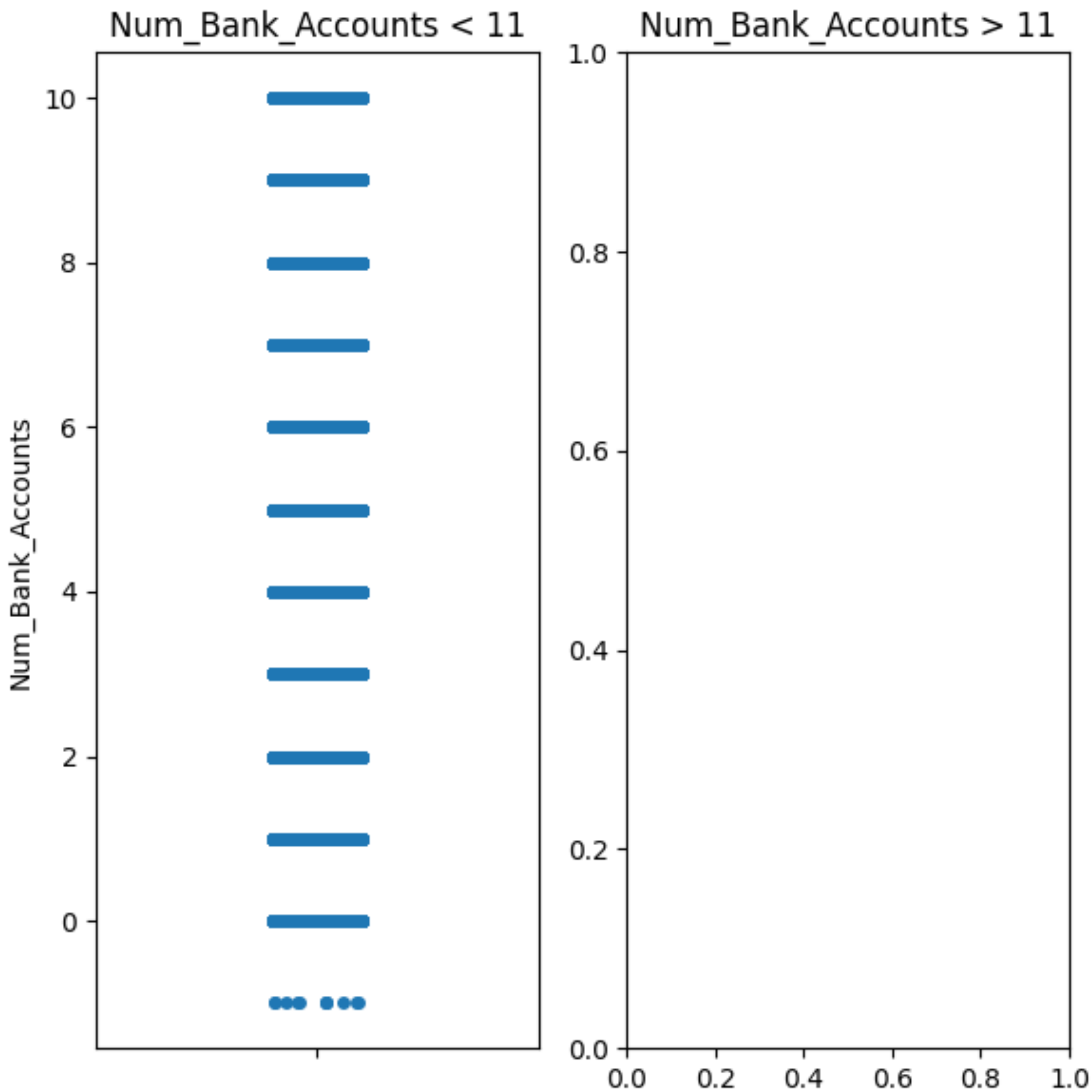
In [26]:
```
new_data = data.copy()

cols = [
    "Age",
    "Num_Bank_Accounts",
    "Num_Credit_Card",
    "Interest_Rate",
    "Num_of_Loan",
    "Num_of_Delayed_Payment",
    "Num_Credit_Inquiries",
    "Monthly_Balance",
]

for col in cols:
    q1, q3 = np.percentile(new_data[col], [25,75])
    iqr = q3 - q1
    lower_bound = q1 - (1.5 * iqr)
    upper_bound = q3 + (1.5 * iqr)
    outliers_mask = (new_data[col] < lower_bound) | (new_data[col] > upper_b
    new_data = new_data[~outliers_mask]

new_data.describe()
```

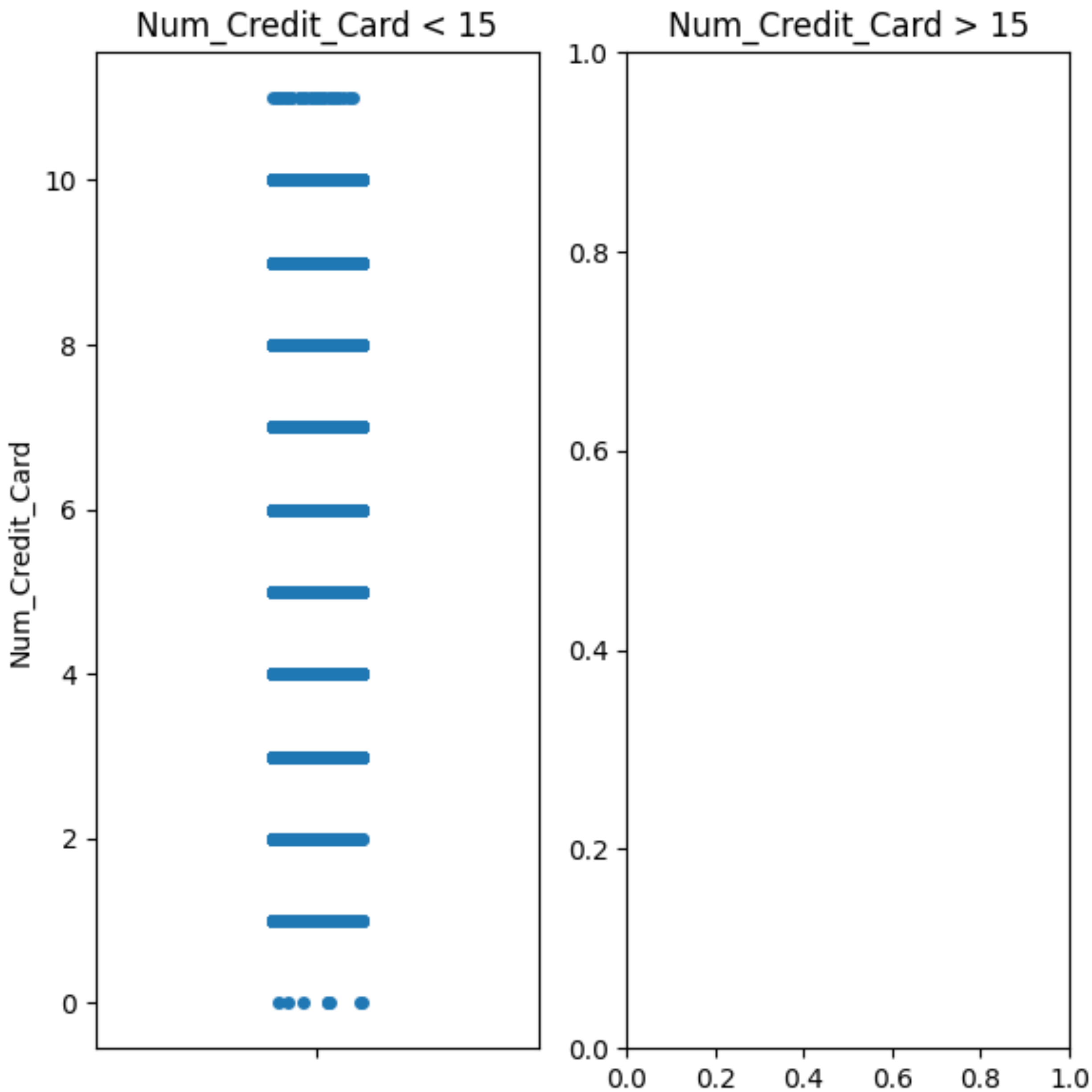
Out[26]:

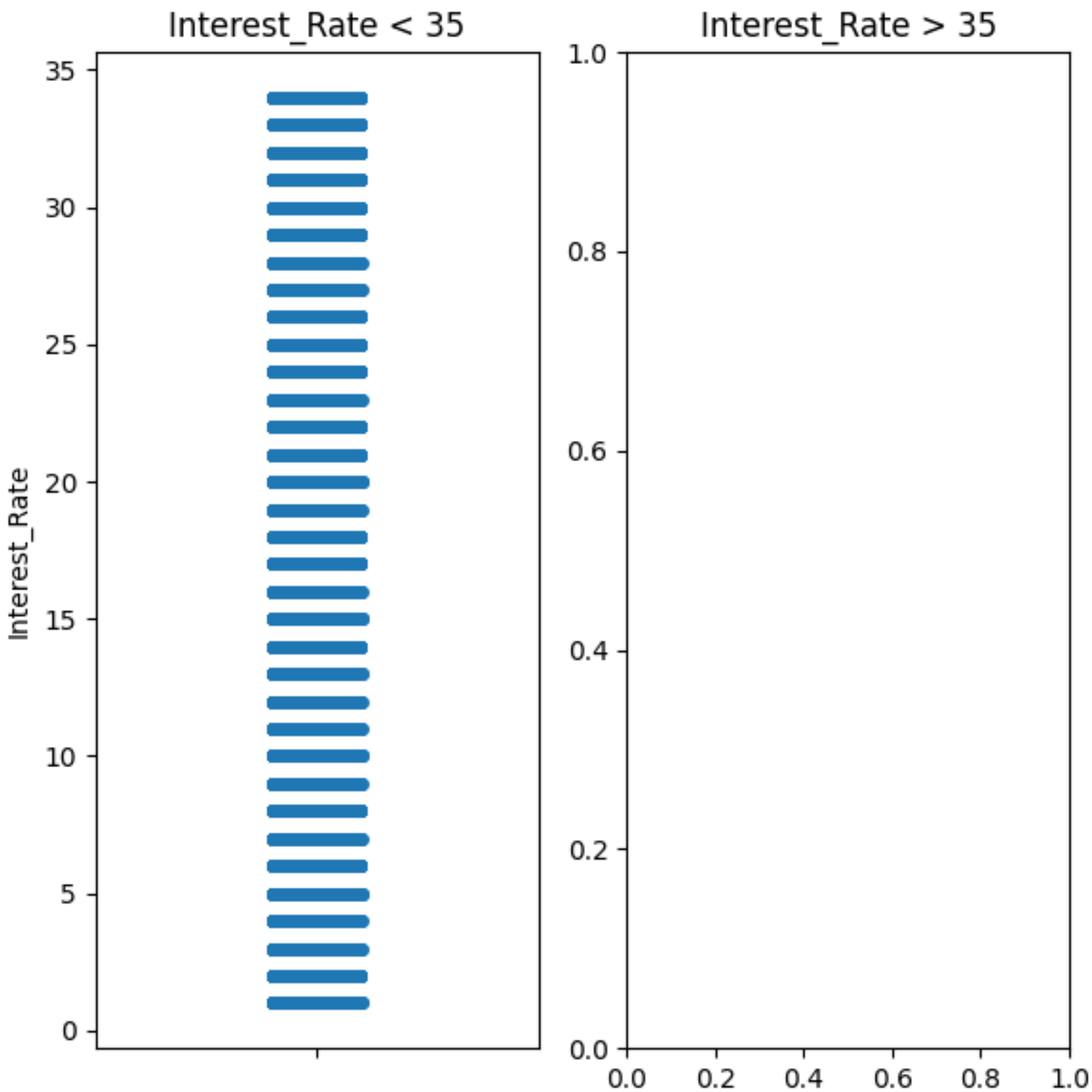
	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary
count	82687.000000	82687.000000	82687.000000	8.268700e+04	82687.000000
mean	4.502497	33.099133	6.966089	1.748802e+05	3543.404766
std	2.291477	10.727333	4.312274	1.454724e+06	2618.491081
min	1.000000	14.000000	0.000000	7.005930e+03	243.560417
25%	3.000000	24.000000	3.000000	1.876069e+04	1506.366667
50%	5.000000	33.000000	7.000000	3.489852e+04	2772.991667
75%	7.000000	41.000000	11.000000	6.353966e+04	5028.165000
max	8.000000	56.000000	14.000000	2.419806e+07	15167.180000

In [27]:
```
outlier_one(new_data)
```

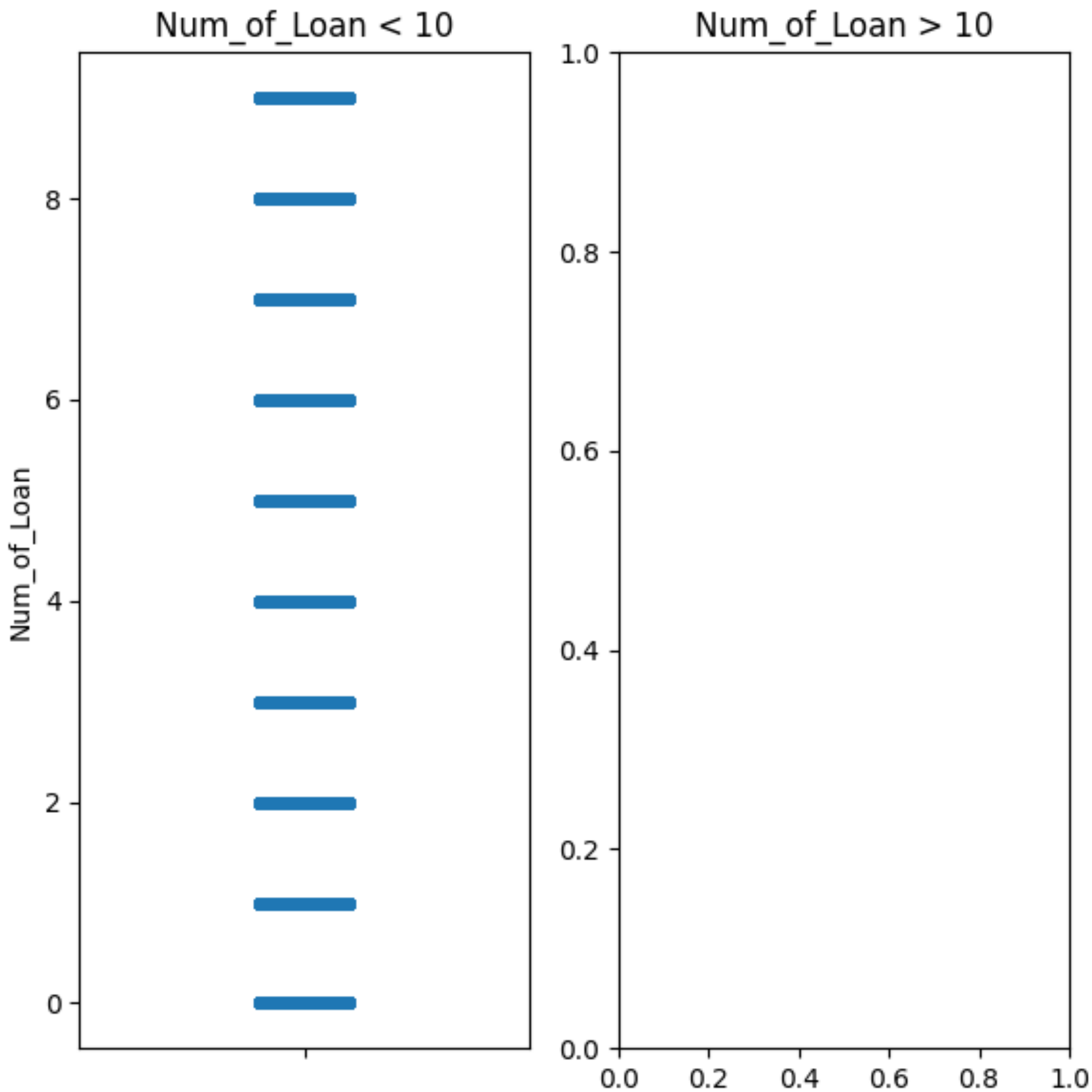

Age < 56
Age > 56
Age
50
40
30
20
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

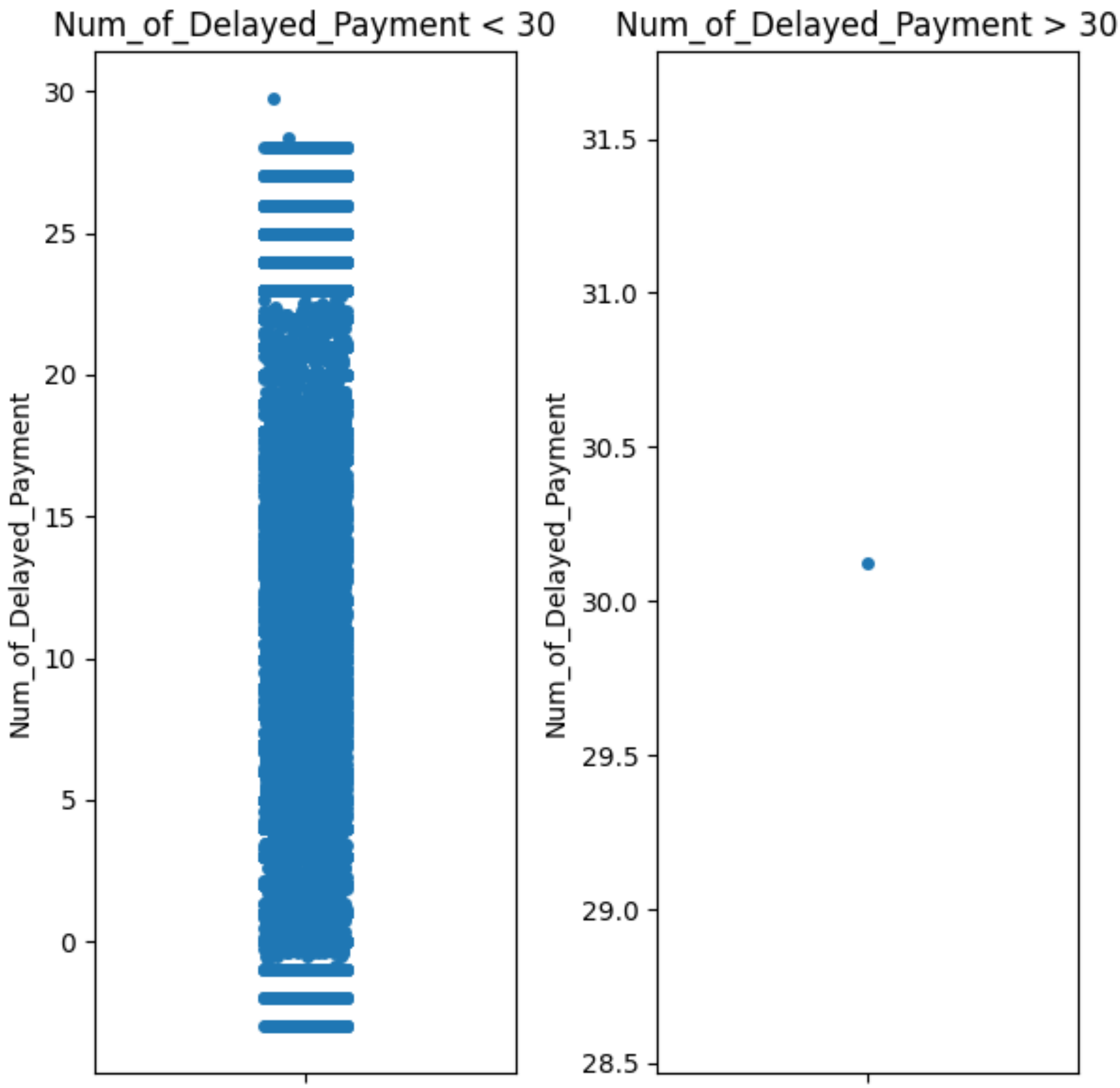

Num_Bank_Accounts < 11
Num_Bank_Accounts > 11
Num_Bank_Accounts
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

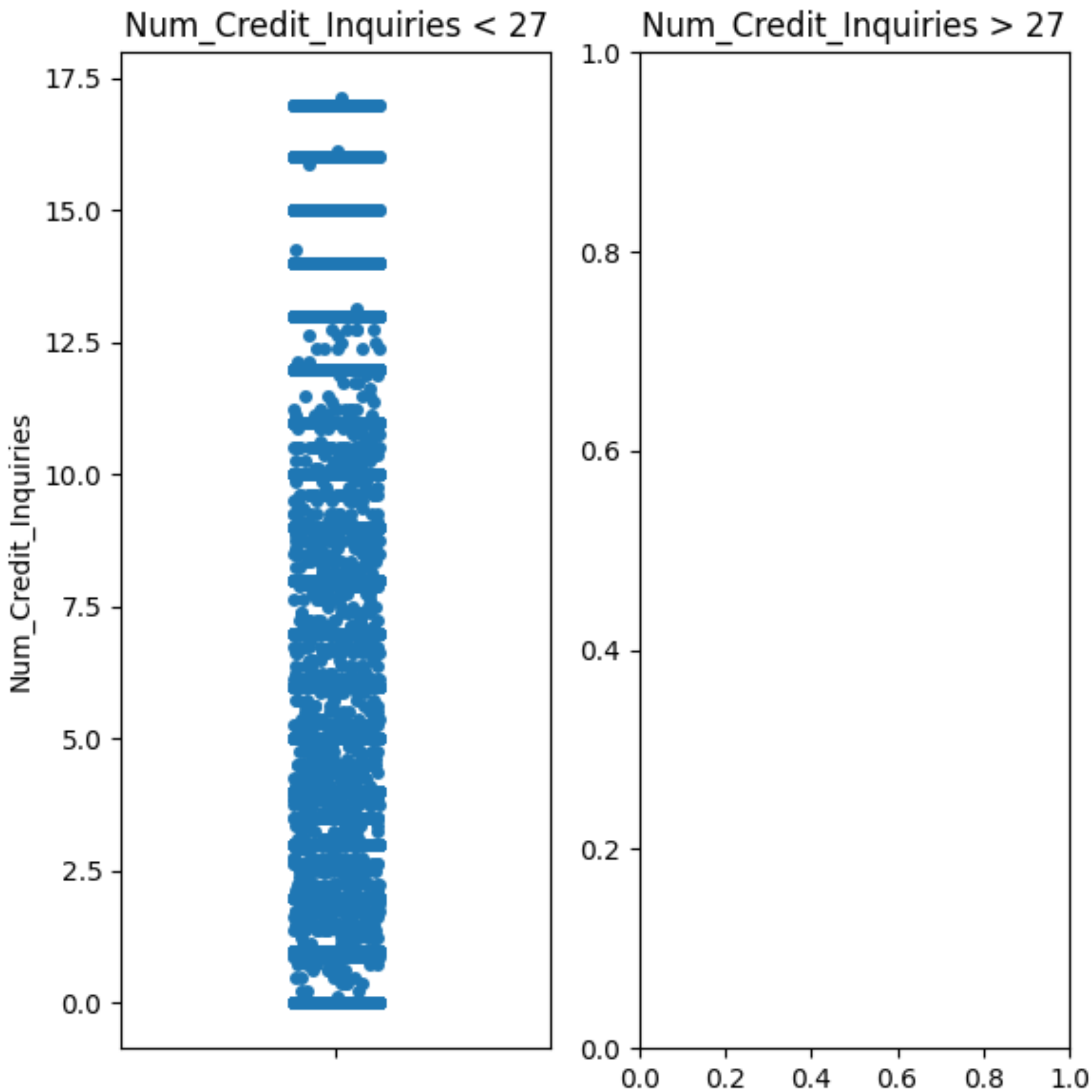

Num_Credit_Card < 15
Num_Credit_Card > 15
Num_Credit_Card
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

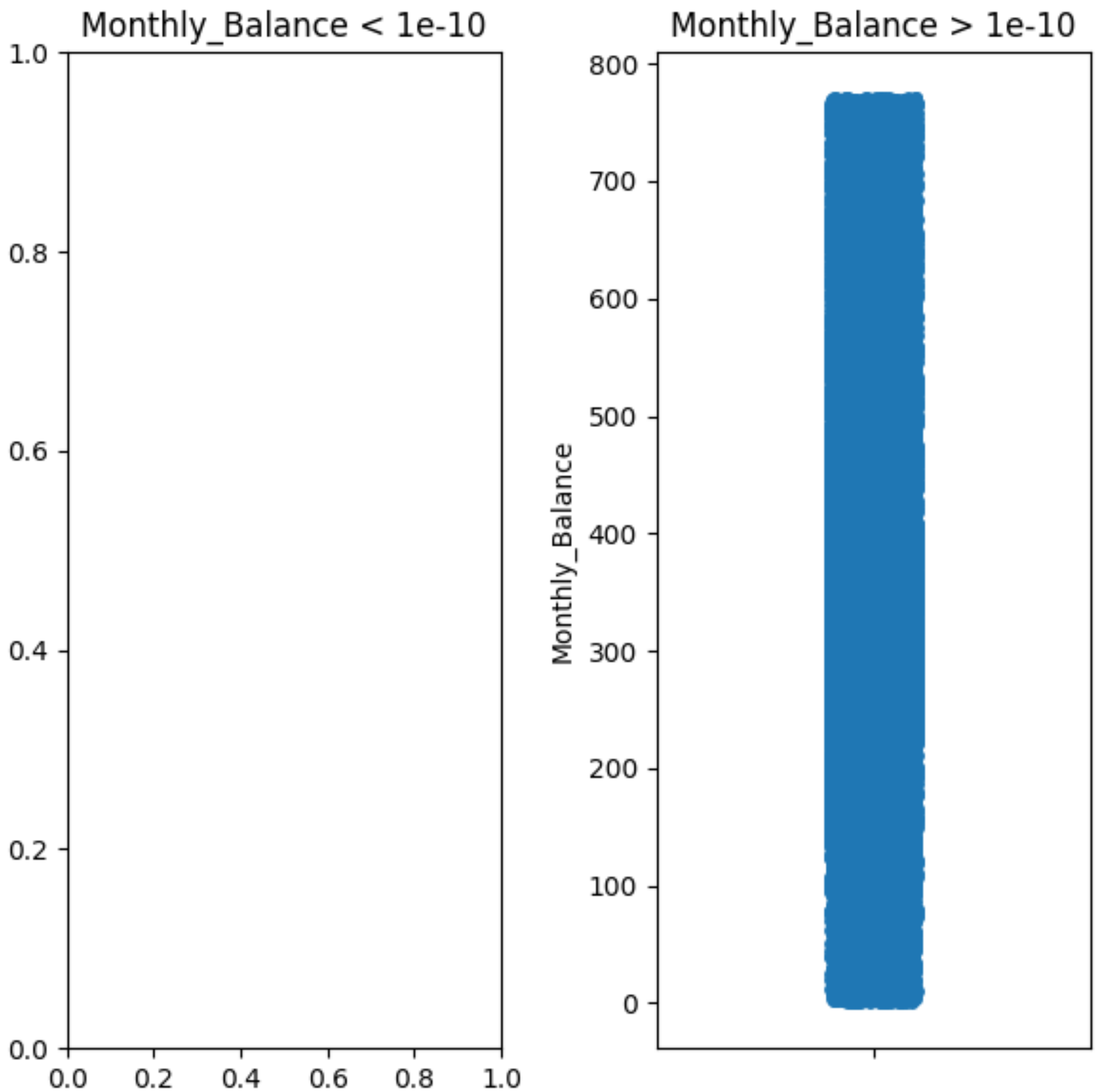


In [28]:
```
outlier_two(new_data)
```


Num_of_Loan < 10
Num_of_Loan > 10
Num_of_Loan
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

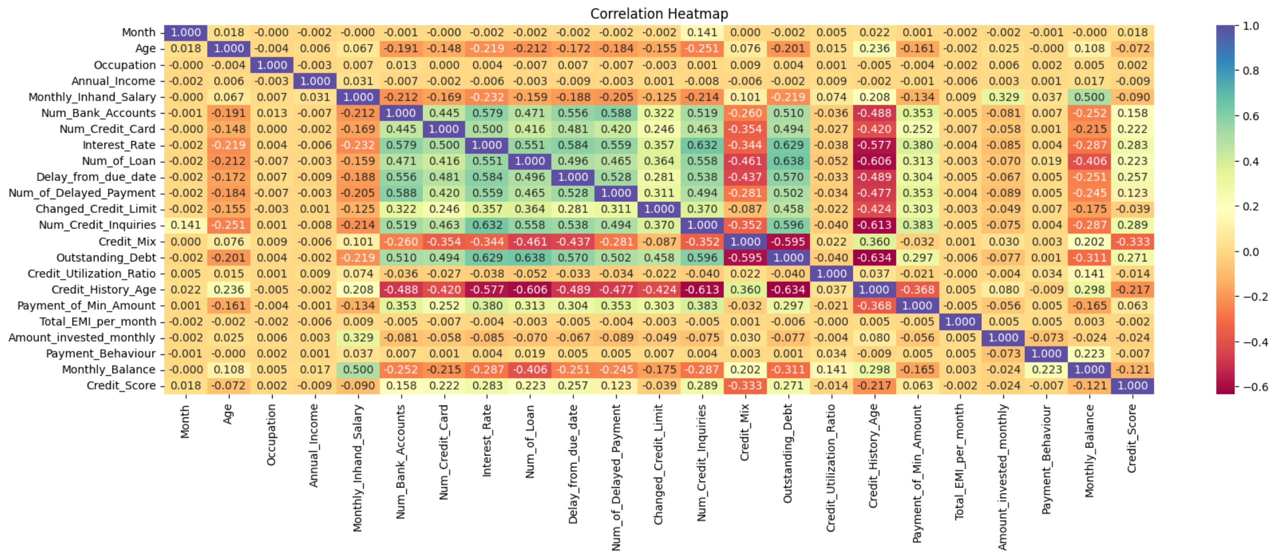

Num_of_Delayed_Payment < 30
Num_of_Delayed_Payment > 30
Num_of_Delayed_Payment
Num_of_Delayed_Payment
30
25
20
15
10
5
0
31.5
31.0
30.5
30.0
29.5
29.0
28.5


Num_Credit_Inquiries < 27
Num_Credit_Inquiries > 27
Num_Credit_Inquiries
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Correlation map after removing outliers

In [29]:
```
show_heat_map(new_data)
```

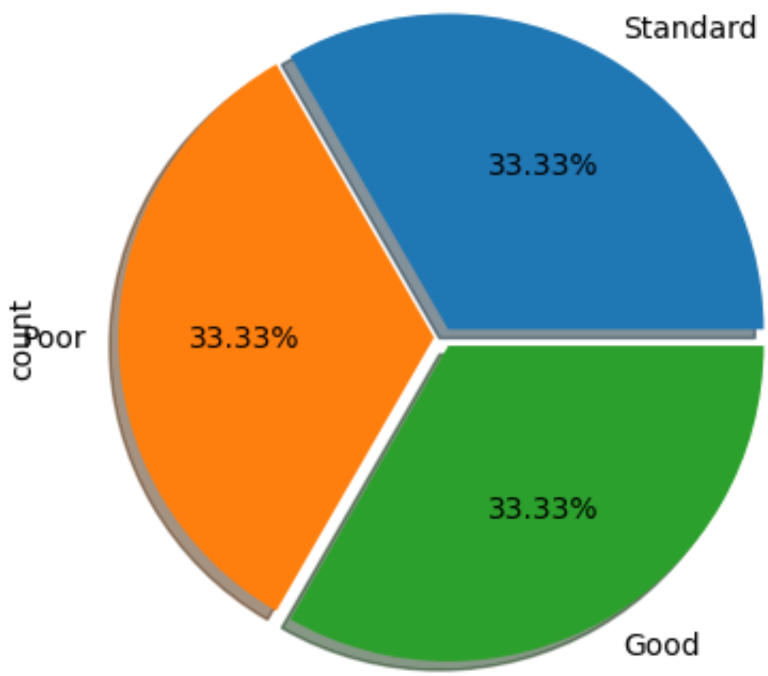


Balancing the data with oversampling

In [30]:
```python
from imblearn.over_sampling import RandomOverSampler

sampler = RandomOverSampler()

y = new_data["Credit_Score"]
X = new_data.drop("Credit_Score", axis=1)
X, y = sampler.fit_resample(X, y)

new_data = pd.concat([pd.DataFrame(X), pd.DataFrame(y)], axis=1)
```

In [31]:
```python
new_data["Credit_Score"].value_counts().plot.pie(
    explode=[0.03, 0.03, 0.03],
    autopct="%1.2f%%",
    shadow=True,
    labels=["Standard", "Poor", "Good"],
)
```

Out[31]:
```
<Axes: ylabel='count'>
```



In [32]:
```
new_data.describe()
```

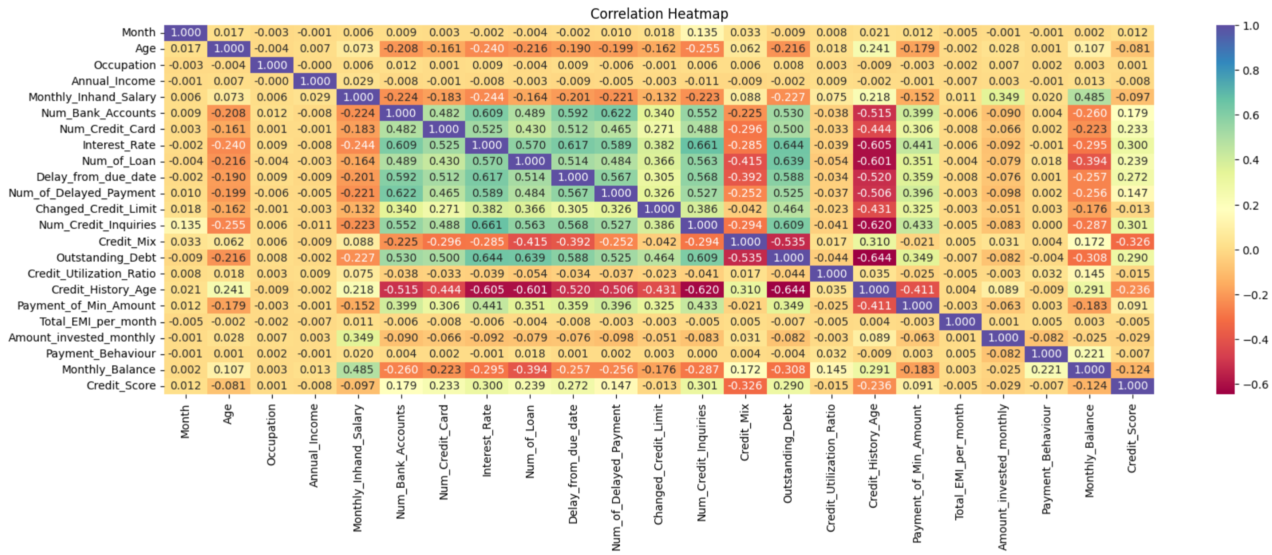
Out[32]:

	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salar
count	131943.000000	131943.000000	131943.000000	1.319430e+05	131943.00000
mean	4.553481	33.562258	6.947250	1.755283e+05	3650.39104
std	2.282699	10.815746	4.299185	1.461136e+06	2710.45050
min	1.000000	14.000000	0.000000	7.005930e+03	243.56041
25%	3.000000	25.000000	3.000000	1.918497e+04	1541.02250
50%	5.000000	33.000000	7.000000	3.573346e+04	2853.36250
75%	7.000000	42.000000	11.000000	6.566878e+04	5125.89750
max	8.000000	56.000000	14.000000	2.419806e+07	15167.18000

In [33]:
```
new_data.to_csv("hw1.csv", index=False)
```

Correlation map after balancing the data

In [34]:
```
show_heat_map(new_data)
```



Feature Selection and Machine Learning Algorithms

Feature Selection Algorithms

- Lasso
- chi2
- MIC
- Ridge
- RFE
- PCA

Machine Learning Algorithms

- Random Forest
- Decision Tree
- Gradient Boosting

In [35]:
```
result_list = {"Lasso" : [], "chi2" : [],"MIC" : [],"Ridge" : [],"RFE" : [],

def calculate(y_test, y_pred, y_proba, y_bin, method_name, num_of_features,

    dictionary = {
        "accuracy" : accuracy_score(y_test, y_pred),
        "f1" : f1_score(y_test, y_pred, average='weighted'),
        "recall" : recall_score(y_test, y_pred, average='weighted'),
        "precision" :  precision_score(y_test, y_pred, average='weighted'),
        "roc_auc" : roc_auc_score(y_test, y_proba, multi_class="ovr"),
        "number_of_features" : num_of_features,
        "model_name" : model_name,
        "confusion_matrix" :  confusion_matrix(y_test, y_pred),
        "roc_curve" : None
    }

    fpr = {}
    tpr = {}
    roc_auc = {}

    for i in range(y_bin.shape[1]):
        fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    dictionary["roc_curve"] = [fpr, tpr, roc_auc]
    result_list[method_name].append(dictionary)
```

Scaling Data

In [36]:
```
def min_max_scale(X):
    scaler = MinMaxScaler()
    X_min_max = pd.DataFrame(scaler.fit_transform(X))
    return X_min_max

def standard_scaler(X):
    scaler = StandardScaler()
    X_standard = pd.DataFrame(scaler.fit_transform(X))
    return X_standard
```

In [37]:
```
X = new_data.drop("Credit_Score", axis=1)
y = new_data["Credit_Score"]

X_standard = standard_scaler(X)
X_min_max = min_max_scale(X)
```

In [38]:
```
def fit_model(X_selected, y,method_name ,model, model_name):
    X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)
    y_bin = label_binarize(y_test, classes=[0,1,2])
    calculate(y_test, y_pred, y_proba, y_bin, method_name,X_selected.shape[1
```

Lasso

In [39]:
```
def lasso(X,y):
    lasso = Lasso(alpha=0.042)
    lasso.fit(X, y)
    selected = X.columns[lasso.coef_ != 0]
    return X[selected]
```

In [40]:
```
X_selected = lasso(X_standard,y)

fit_model(X_selected,y,"Lasso",RandomForestClassifier(n_jobs=-1), "RandomFor
fit_model(X_selected,y,"Lasso",DecisionTreeClassifier(), "DecisionTreeClassi
```

Chi2 and Mutual Info

In [41]:
```
def select_kbest(X, y, method, k):
    k_best = SelectKBest(score_func=method, k=k)
    selected = k_best.fit_transform(X, y)
    return selected

for i in range(15,X.shape[1]+1):
    fit_model(select_kbest(X_min_max,y,chi2,i),y,"chi2",RandomForestClassifi
    fit_model(select_kbest(X_min_max,y,chi2,i),y,"chi2",DecisionTreeClassifi
    fit_model(select_kbest(X_min_max,y,mutual_info_classif,i),y,"MIC",Random
    fit_model(select_kbest(X_min_max,y,mutual_info_classif,i),y,"MIC",Decisi
```

Ridge

In [42]:
```
def ridge(X, y, k):
    ridge = Ridge(alpha=1)
    ridge.fit(X, y)
    feature_importance = np.abs(ridge.coef_)
    selected_feature_indices = np.argsort(feature_importance)[::-1][:k]
    X_selected = X.iloc[:, selected_feature_indices]
    return X_selected

for i in range (15,X.shape[1]+1):
    fit_model(ridge(X_min_max,y,i), y,"Ridge",RandomForestClassifier(n_jobs=
    fit_model(ridge(X_min_max,y,i), y,"Ridge",DecisionTreeClassifier(),"Deci
```

RFE

In [43]:
```
def RFE_feature_selection(X, y, model, k):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
    rfe = RFE(model,n_features_to_select=k)
    rfe.fit(X_train, y_train)
    selected_features = rfe.support_
    selected_columns = X_train.columns[selected_features]
    return  X_train[selected_columns], X_test[selected_columns], y_train,y_t


def RFE_predict(X,y,model,model_name,k):
    X_train, X_test, y_train, y_test = RFE_feature_selection(X,y,model,k)
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)
    y_bin = label_binarize(y_test, classes=[0,1,2])
    calculate(y_test,y_pred,y_proba,y_bin,"RFE",X_train.shape[1],model_name)

for i in range (15,X.shape[1]+1):
    RFE_predict(X,y,RandomForestClassifier(n_jobs=-1),"RandomForestClassifie
    RFE_predict(X,y,DecisionTreeClassifier(),"DecisionTreeClassifier",i)
```

PCA

In [45]:
```
def PCA_feature_selection(X, k):
    pca = PCA(n_components = k)
    pca.fit(X)
    data = pca.transform(X)
    return data

for i in range(15,X.shape[1]+1):
    fit_model(PCA_feature_selection(X_min_max,i), y,"PCA",RandomForestClassi
    fit_model(PCA_feature_selection(X_min_max,i), y,"PCA",DecisionTreeClassi
```

In [46]:
```
models_data = []

for model_name, model_data in result_list.items():
    for j in range(len(model_data)):
        model_item = model_data[j]
        model_entry = {
            'method_name' : model_name,
            'model_name': model_item["model_name"],
            'accuracy': model_item["accuracy"],
            'f1': model_item["f1"],
            'recall': model_item["recall"],
            'precision': model_item["precision"],
            'roc_auc' : model_item["roc_auc"],
            'number_of_features': model_item["number_of_features"],
            'confusion_matrix' : model_item["confusion_matrix"],
            "roc_curve" : model_item["roc_curve"]
        }
        models_data.append(model_entry)

score_dataframe = pd.DataFrame(models_data)
best_method = score_dataframe[score_dataframe["accuracy"] == score_dataframe
```

In [47]:
```
score_dataframe.drop(["confusion_matrix", "roc_curve"], axis=1)
```

Out[47]:

	method_name	model_name	accuracy	f1	recall	precision	roc_au
0	Lasso	RandomForestClassifier	0.871613	0.868629	0.871613	0.871406	0.95268
1	Lasso	DecisionTreeClassifier	0.851074	0.848139	0.851074	0.849482	0.91188
2	chi2	RandomForestClassifier	0.906893	0.904644	0.906893	0.908944	0.97642
3	chi2	DecisionTreeClassifier	0.878434	0.875921	0.878434	0.877970	0.90869
4	chi2	RandomForestClassifier	0.905491	0.903295	0.905491	0.907718	0.97663
...	...	...	...	...	...	...	...
77	PCA	DecisionTreeClassifier	0.838645	0.834764	0.838645	0.837218	0.87862
78	PCA	RandomForestClassifier	0.872068	0.868439	0.872068	0.875475	0.97610
79	PCA	DecisionTreeClassifier	0.837735	0.833733	0.837735	0.836278	0.87828
80	PCA	RandomForestClassifier	0.876085	0.872461	0.876085	0.880123	0.97806
81	PCA	DecisionTreeClassifier	0.836523	0.832068	0.836523	0.835904	0.87752

82 rows × 8 columns

In [48]:
```
best_method.drop(["confusion_matrix", "roc_curve"], axis=1)
```

Out[48]:

	method_name	model_name	accuracy	f1	recall	precision	roc_auc
60	RFE	RandomForestClassifier	0.919363	0.917643	0.919363	0.92158	0.984716

In [49]:
```
method_name = best_method["method_name"].to_string().split(" ")[-1]
number_of_features = int(best_method["number_of_features"].to_string().split
```

In [50]:
```
call_best_method = {
    "Lasso": lambda _, X=X_standard, y=y: lasso(X, y),
    "chi2": lambda k, X=X_min_max, y=y: select_kbest(X, y, chi2, k),
    "MIC": lambda k, X=X_min_max, y=y: select_kbest(X, y, mutual_info_classi
    "Ridge": lambda k, X=X_min_max, y=y: ridge(X, y, k),
    "RFE": lambda k, X=X, y=y: RFE_feature_selection(X, y, DecisionTreeClass
    "PCA": lambda k, X=X_min_max, y=y: PCA_feature_selection(X, k),
}

X_selected_features = call_best_method[method_name](number_of_features)
```

In [51]:
```
if type(X_selected_features) == tuple:
    X_train, X_test, y_train, y_test = X_selected_features
else:
    X_train, X_test, y_train, y_test = train_test_split(X_selected_features,
```

Models for max voting and stacking

In [53]:
```
models = [
    ("RandomForestClassifier", RandomForestClassifier(n_jobs=-1, n_estimator
    ("DecisionTreeClassifier", DecisionTreeClassifier()),
    ("KNeighborsClassifier", KNeighborsClassifier(n_jobs=-1, n_neighbors=1))
    ("ExtraTreeClassifier", ExtraTreeClassifier()),
]
```

Max Voting

In [54]:
```
def max_voting():
    max_voting_model = VotingClassifier(models, voting='soft', n_jobs=-1)
    max_voting_model.fit(X_train, y_train)
    prediction = max_voting_model.predict(X_test)
    y_proba = max_voting_model.predict_proba(X_test)
    y_bin = label_binarize(y_test, classes=[0,1,2])

    calculate(y_test, prediction, y_proba,y_bin,  "MaxVoting", number_of_fea

max_voting()
```

Stacking

In [55]:
```
def stacking():
    stacked = StackingClassifier(estimators=models, final_estimator=RandomFo
    stacked.fit(X_train, y_train)
    predictions = stacked.predict(X_test)
    y_proba = stacked.predict_proba(X_test)
    y_bin = label_binarize(y_test, classes=[0,1,2])

    calculate(y_test, predictions, y_proba, y_bin, "Stacking", number_of_fea

stacking()
```

In [56]:
```
def add_to_dataframe(model, dataframe):
    models_data = []

    for j in range(len(result_list[model])):
        model_item = result_list[model][j]
        model_entry = {
            'method_name' : model,
            'model_name': model_item["model_name"],
            'accuracy': model_item["accuracy"],
            'f1': model_item["f1"],
            'recall': model_item["recall"],
            'precision': model_item["precision"],
            'roc_auc' : model_item["roc_auc"],
            'number_of_features': model_item["number_of_features"],
            "confusion_matrix" : model_item["confusion_matrix"],
            "roc_curve" : model_item["roc_curve"]
        }
        models_data.append(model_entry)

    temp_df =  pd.DataFrame(models_data)
    dataframe = pd.concat([dataframe, temp_df], ignore_index=True)
    return dataframe
```

In [57]:
```
score_dataframe = add_to_dataframe("MaxVoting", score_dataframe)
score_dataframe = add_to_dataframe("Stacking", score_dataframe)
```

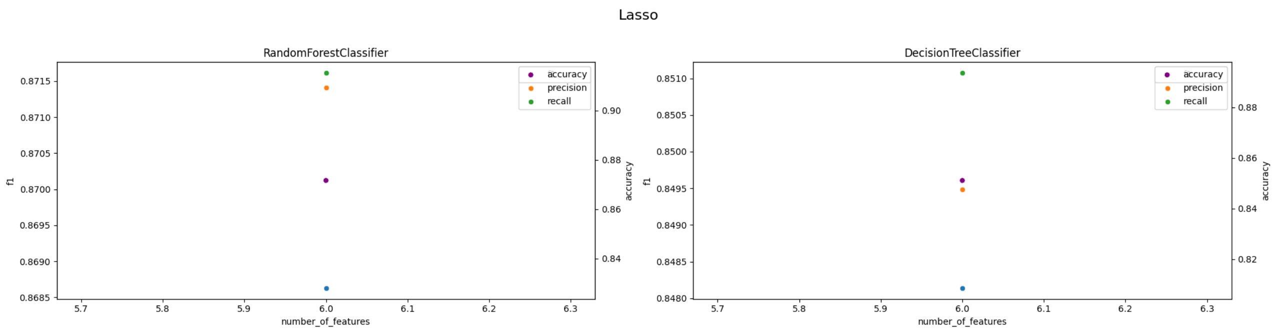
Visualizing the results

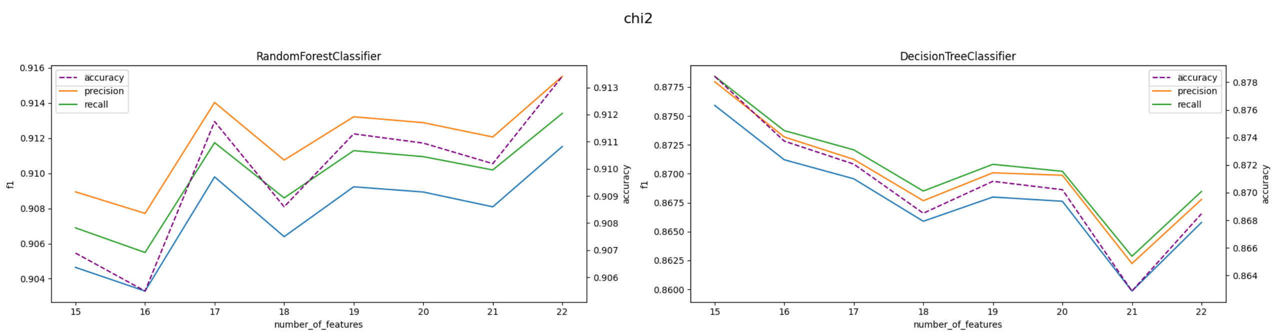
In [59]:

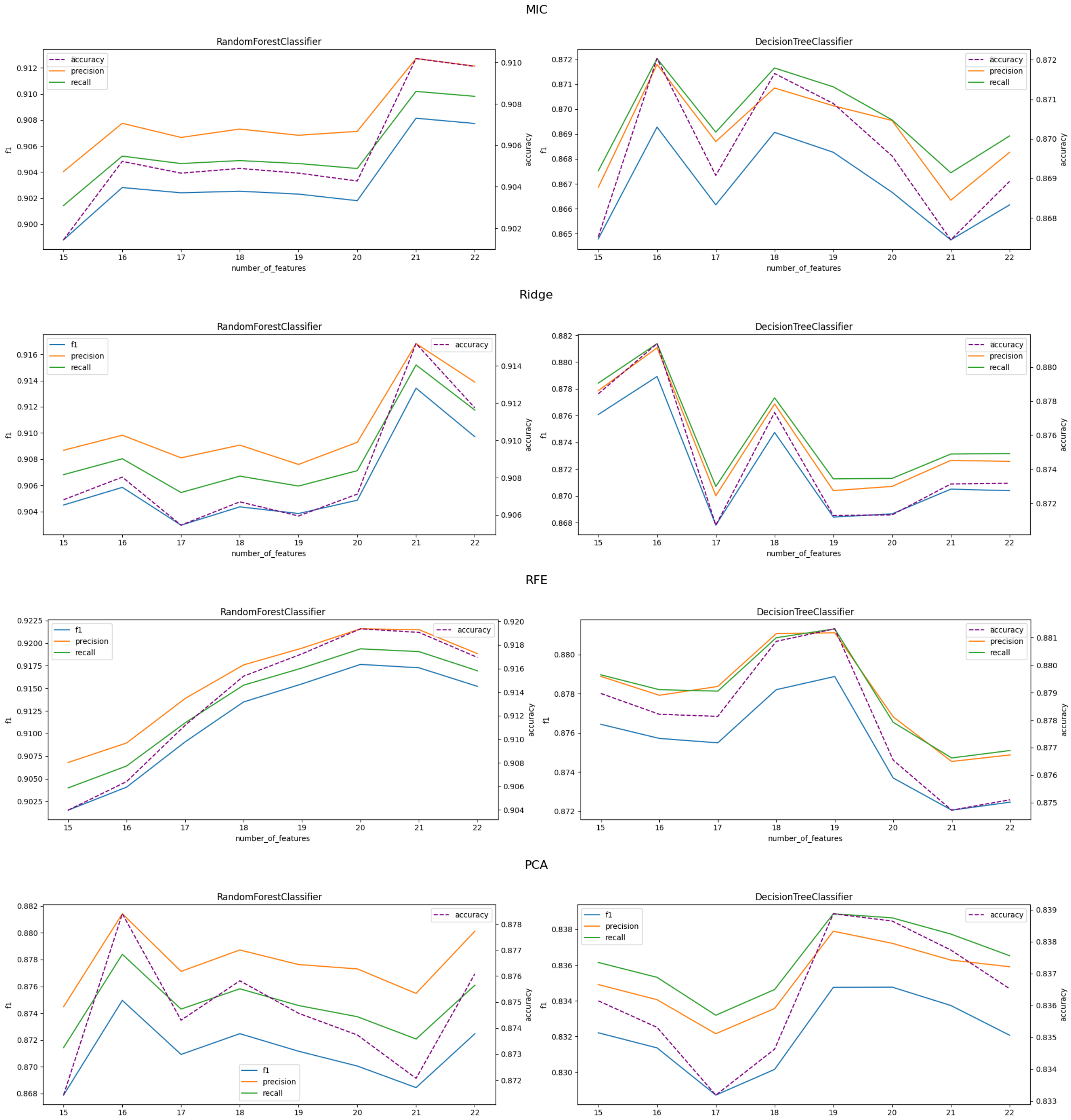
```
scores = score_dataframe.copy()
model_names = [
    "RandomForestClassifier",
    "DecisionTreeClassifier",
]

def show_results(method_name,plt_type = sns.lineplot):
    metrics = ["f1", "precision", "recall"]
    fig, axs = plt.subplots(1, len(model_names), figsize=(20, 5))

    for i, item in enumerate(model_names):
        model_data = scores[
            (scores["method_name"] == method_name) & (scores["model_name"] =
        ]
        for metric in metrics:
            plt_type(x="number_of_features", y=metric, label = metric ,data=

        ax2 = axs[i].twinx()
        plt_type(x="number_of_features", y="accuracy", label = "accuracy",da
        axs[i].set_title(item)

    fig.suptitle(method_name, fontsize=16, y=1.02)
    plt.tight_layout()
    plt.show()
```

In [60]:

```
show_results("Lasso",plt_type=sns.scatterplot)
show_results("chi2")
show_results("MIC")
show_results("Ridge")
show_results("RFE")
show_results("PCA")
```







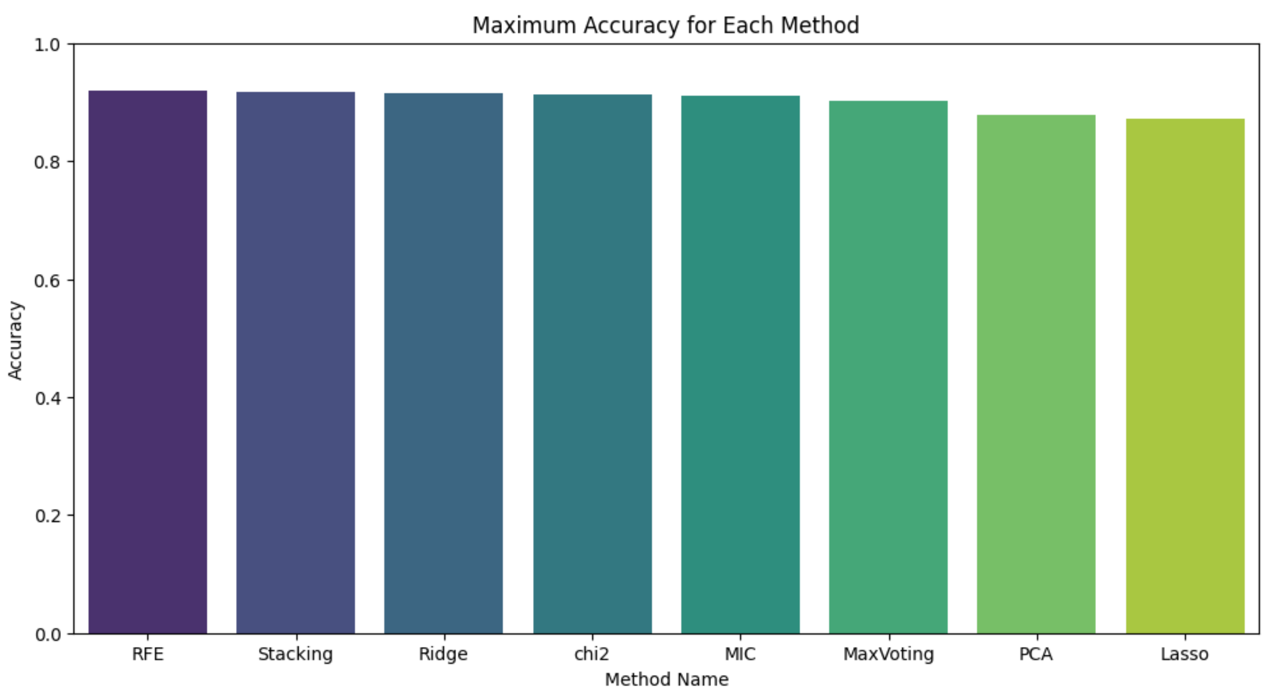
In [61]:
```
def calculate_best_scores():
    best_scores = {"Lasso": None, "chi2": None, "MIC": None, "Ridge": None,

    for i, row in score_dataframe.iterrows():
        current_best = best_scores[row["method_name"]]

        if current_best is None or row["accuracy"] > current_best["accuracy"
            best_scores[row["method_name"]] = row.to_dict()

    return best_scores

best_scores = calculate_best_scores()
```

In [62]:
```
best_scores_df = pd.DataFrame(list(best_scores.values()))
best_scores_df = best_scores_df.sort_values(by="accuracy", ascending=False)
best_scores_df.drop(["confusion_matrix", "roc_curve"], axis=1)
```

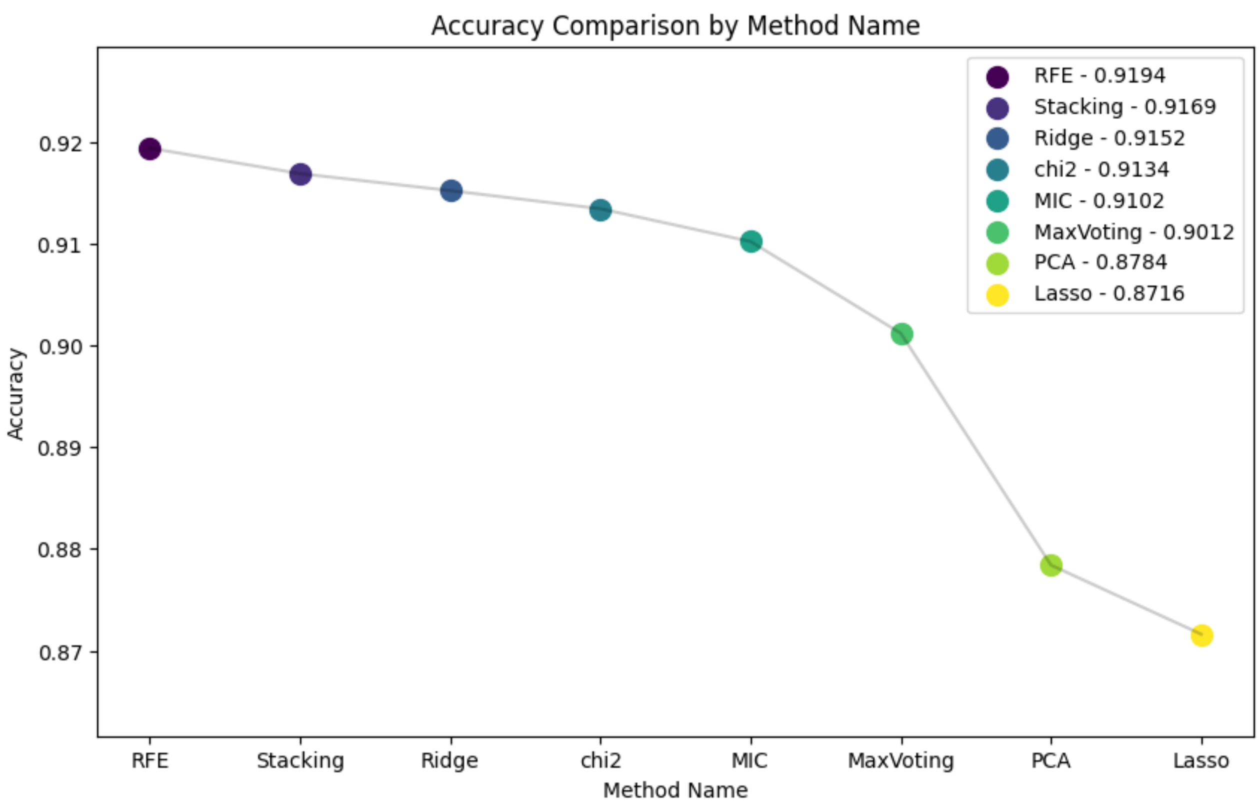
Out[62]:

	method_name	model_name	accuracy	f1	recall	precision	roc_auc
4	RFE	RandomForestClassifier	0.919363	0.917643	0.919363	0.921580	0.984716
7	Stacking	Stacking	0.916862	0.916342	0.916862	0.916919	0.980803
3	Ridge	RandomForestClassifier	0.915192	0.913418	0.915192	0.916825	0.981415
1	chi2	RandomForestClassifier	0.913411	0.911525	0.913411	0.915521	0.982535
2	MIC	RandomForestClassifier	0.910190	0.908132	0.910190	0.912710	0.982051
6	MaxVoting	MaxVoting	0.901175	0.899462	0.901175	0.901278	0.983704
5	PCA	RandomForestClassifier	0.878396	0.874952	0.878396	0.881422	0.976247
0	Lasso	RandomForestClassifier	0.871613	0.868629	0.871613	0.871406	0.952685

In [63]:
```
plt.figure(figsize=(12, 6))
sns.barplot(x='method_name', y='accuracy', data=best_scores_df, palette='vir
plt.title('Maximum Accuracy for Each Method')
plt.xlabel('Method Name')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()
```

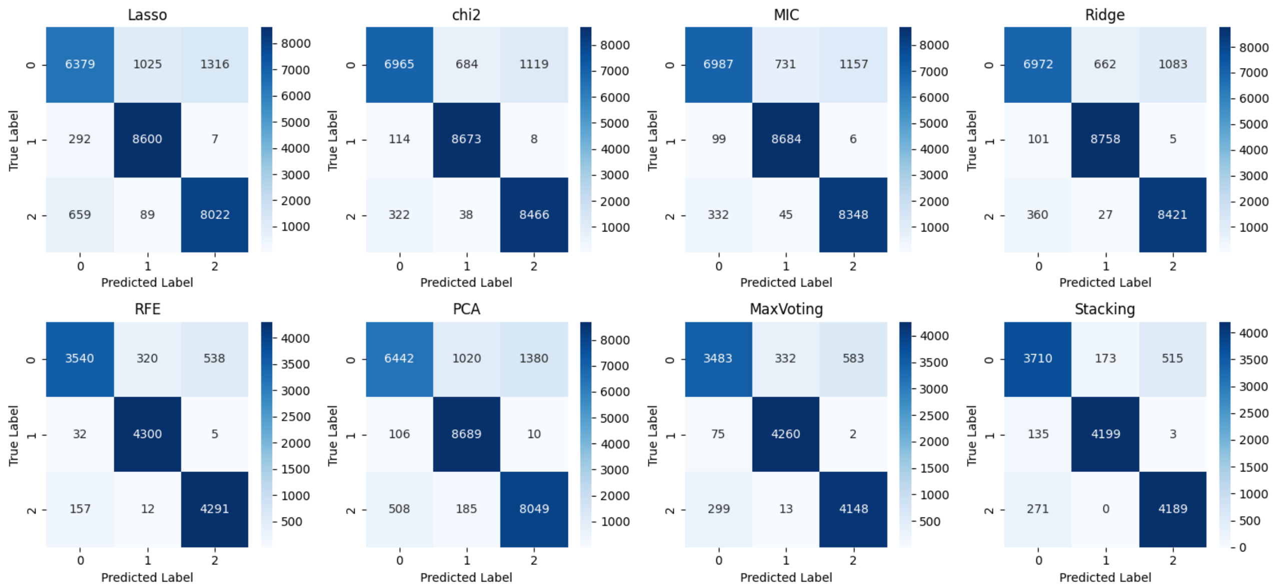


In [64]:
```
method_names = best_scores_df["method_name"].tolist()
accuracies = best_scores_df["accuracy"].tolist()
colors = plt.cm.viridis(np.linspace(0, 1, len(method_names)))

plt.figure(figsize=(10, 6))
for i, (method, accuracy, color) in enumerate(zip(method_names, accuracies,
    plt.scatter(method, accuracy, color=color, label=f"{method} - {accuracy:

plt.plot(method_names, accuracies, linestyle="-", color="black", alpha=0.2)
plt.xlabel("Method Name")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison by Method Name")
plt.ylim([accuracies[-1]-0.01, accuracies[0]+0.01])
plt.legend()
plt.show()
```



Confusion matrix

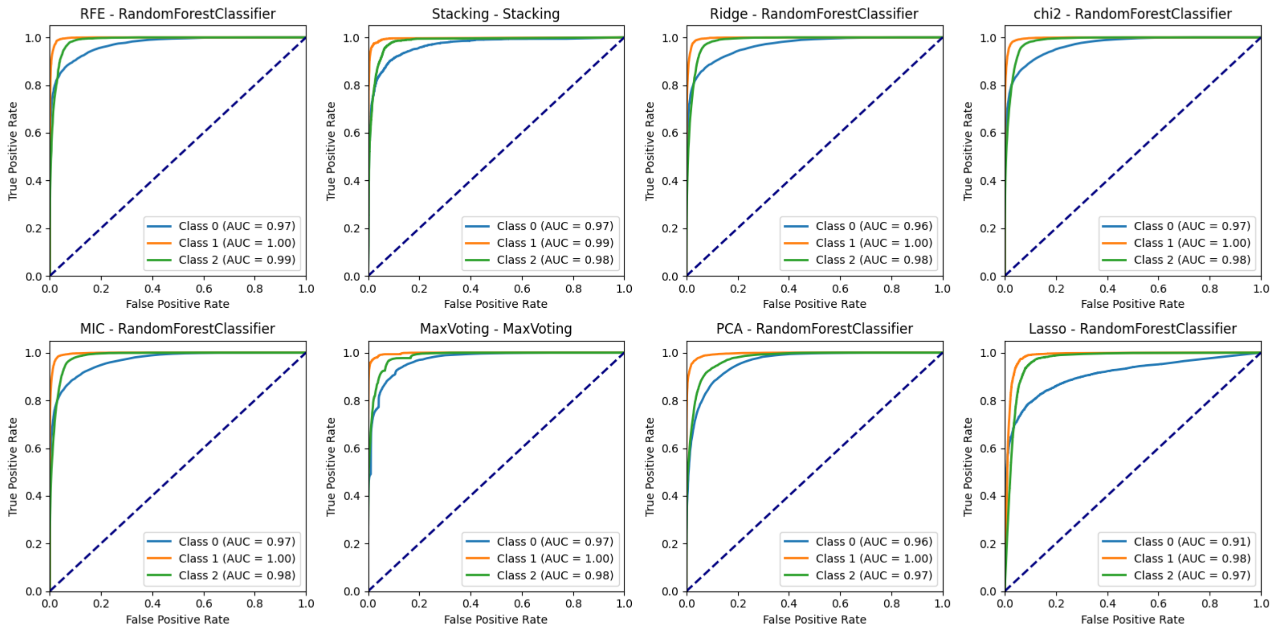
In [65]:
```
fig, axes = plt.subplots(2, 4, figsize=(15, 7))
axes = axes.flatten()
for i in range(len(best_scores_df)):
    sns.heatmap(best_scores_df["confusion_matrix"][i], annot=True, fmt='d',
    axes[i].set_title(best_scores_df['method_name'][i])
    axes[i].set_xlabel('Predicted Label')
    axes[i].set_ylabel('True Label')
plt.tight_layout()
```



ROC curve

In [66]:

```
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))

for index, (i, row) in enumerate(best_scores_df.iterrows()):
    fpr = row["roc_curve"][0]
    tpr = row["roc_curve"][1]
    roc_auc = row["roc_curve"][2]

    row_index = index // 4
    col_index = index % 4

    for j in range(3):
        axes[row_index, col_index].plot(fpr[j], tpr[j], lw=2, label=f'Class

    axes[row_index, col_index].plot([0, 1], [0, 1], color='navy', lw=2, line
    axes[row_index, col_index].set_xlim([0.0, 1.0])
    axes[row_index, col_index].set_ylim([0.0, 1.05])
    axes[row_index, col_index].set_xlabel('False Positive Rate')
    axes[row_index, col_index].set_ylabel('True Positive Rate')
    axes[row_index, col_index].set_title(f'{row["method_name"]} - {row["mode
    axes[row_index, col_index].legend(loc="lower right")

plt.tight_layout()
plt.show()
```



In [67]:
```
X = data.drop("Credit_Score", axis=1)
y = data["Credit_Score"]

def calculate_score_with_outliers(model):
    if method_name == "chi2" or method_name == "MIC":
        selected = call_best_method[method_name](number_of_features,X=min_ma
    else:
        selected = call_best_method[method_name](number_of_features,X=X,y=y)


    if type(selected) == tuple:
        X_train, X_test, y_train, y_test = selected
    else:
        X_train, X_test, y_train, y_test = train_test_split(selected, y, tes

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)

outlier_score = calculate_score_with_outliers(RandomForestClassifier())
best_accuracy = best_scores_df.iloc[0]["accuracy"]
```

In [68]:
```
data = {
    'name': ['Best Accuracy without Outliers', 'Best Accuracy with Outliers'
    'value': [best_accuracy, outlier_score]
}

df_metrics = pd.DataFrame(data)

plt.figure(figsize=(8, 6))
colors = sns.color_palette("viridis", len(df_metrics['name']))
sns.barplot(x='name', y='value', data=df_metrics, palette=colors,width=0.4)
plt.title('Best Accuracy and Outlier Score')
plt.ylabel('Value')
plt.xlabel('Datas')
plt.ylim(0, 1)
plt.show()
```