Credit Score Classification

Banks and credit card companies categorize their customers into three credit score tiers:

- Good
- Standard
- Poor

Individuals possessing a good credit score can secure loans from all banks and financial institutions. It helps financial companies determine if you can repay the loan or credit you are applying for. To accomplish the Credit Score Classification task using Machine Learning, I have selected the below dataset.

Dataset

- ID: Unique ID of the record
- Customer_ID: Unique ID of the customer
- Month: Month of the year
- Name: The name of the person
- Age: The age of the person
- SSN: Social Security Number of the person
- Occupation: The occupation of the person
- Annual_Income: The Annual Income of the person
- Monthly_Inhand_Salary: Monthly in-hand salary of the person
- Num_Bank_Accounts: The number of bank accounts of the person
- Num_Credit_Card: Number of credit cards the person is having
- Interest_Rate: The interest rate on the credit card of the person
- Num_of_Loan: The number of loans taken by the person from the bank
- Type_of_Loan: The types of loans taken by the person from the bank
- Delay_from_due_date: The average number of days delayed by the person from the date of payment
- Num_of_Delayed_Payment: Number of payments delayed by the person
- Changed_Credit_Card: The percentage change in the credit card limit of the person
- Num_Credit_Inquiries: The number of credit card inquiries by the person
- Credit_Mix: Classification of Credit Mix of the customer
- Outstanding_Debt: The outstanding balance of the person

- Credit_Utilization_Ratio: The credit utilization ratio of the credit card of the customer
- Credit_History_Age: The age of the credit history of the person
- Payment_of_Min_Amount: Yes if the person paid the minimum amount to be paid only, otherwise no.
- Total_EMI_per_month: The total EMI per month of the person
- Amount_invested_monthly: The monthly amount invested by the person
- Payment_Behaviour: The payment behaviour of the person
- Monthly_Balance: The monthly balance left in the account of the person
- Credit_Score: The credit score of the perso-n

Importing Required Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score,classification_report,confu
```

In [2]:
```
# Reading the data

dk = pd.read_csv("/Users/harshitha/Downloads/Credit Score Data/train.csv")
```

In [3]:
```
# Checking how our data looks

dk.head()
```

Out[3]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...
0	5634	3392	1	Aaron Maashoh	23.0	821000265.0	Scientist	19114.12	1824.843333	3.0	...
1	5635	3392	2	Aaron Maashoh	23.0	821000265.0	Scientist	19114.12	1824.843333	3.0	...
2	5636	3392	3	Aaron Maashoh	23.0	821000265.0	Scientist	19114.12	1824.843333	3.0	...
3	5637	3392	4	Aaron Maashoh	23.0	821000265.0	Scientist	19114.12	1824.843333	3.0	...
4	5638	3392	5	Aaron Maashoh	23.0	821000265.0	Scientist	19114.12	1824.843333	3.0	...

5 rows × 28 columns

Check for missing values

In [4]:
```
dk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   ID                        100000 non-null  int64  
 1   Customer_ID               100000 non-null  int64  
 2   Month                     100000 non-null  int64  
 3   Name                      100000 non-null  object 
 4   Age                       100000 non-null  float64
 5   SSN                       100000 non-null  float64
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     100000 non-null  float64
 9   Num_Bank_Accounts         100000 non-null  float64
 10  Num_Credit_Card           100000 non-null  float64
 11  Interest_Rate             100000 non-null  float64
 12  Num_of_Loan               100000 non-null  float64
 13  Type_of_Loan              100000 non-null  object 
 14  Delay_from_due_date       100000 non-null  float64
 15  Num_of_Delayed_Payment    100000 non-null  float64
 16  Changed_Credit_Limit      100000 non-null  float64
 17  Num_Credit_Inquiries      100000 non-null  float64
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        100000 non-null  float64
 22  Payment_of_Min_Amount     100000 non-null  object 
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   100000 non-null  float64
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           100000 non-null  float64
 27  Credit_Score              100000 non-null  object 
dtypes: float64(18), int64(3), object(7)
memory usage: 21.4+ MB
```

In [5]:
```
# Check for Null Values

print(dk.isnull().sum())
```

```
ID                          0
Customer_ID                 0
Month                       0
Name                        0
Age                         0
SSN                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Type_of_Loan                0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
dtype: int64
```

OBSERVATIONS:

- Our data consists of 28 columns and 100,000 fields
- There are no missing values in our dataset

Exploratory Data Analysis

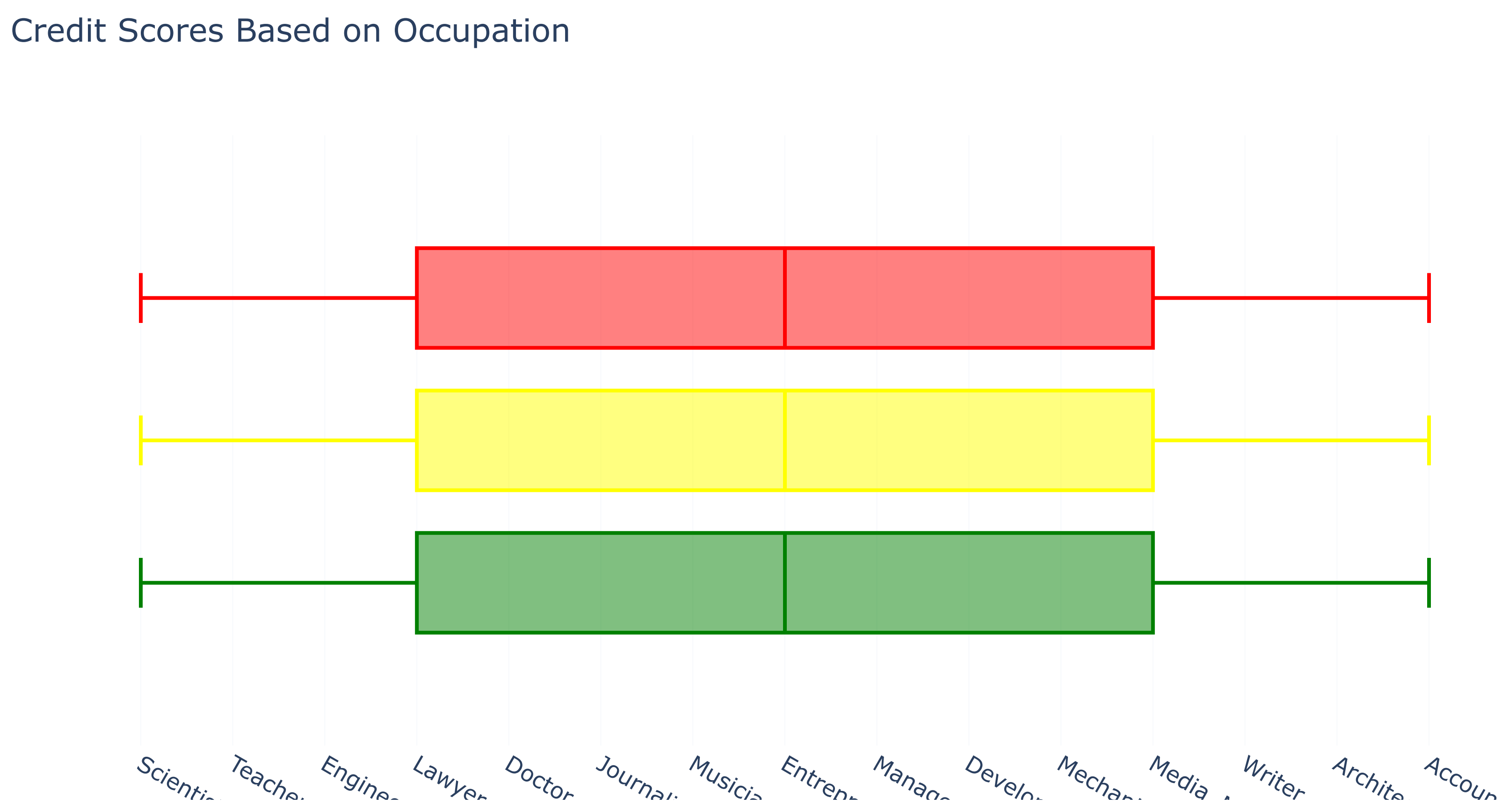
In [6]:
```
# Checking the Credit Score Counts

dk["Credit_Score"].value_counts()
```

Out[6]:
```
Credit_Score
Standard    53174
Poor        28998
Good        17828
Name: count, dtype: int64
```

Let us explore various features of this dataset that can be used to train our machine learning model

1. Occupation

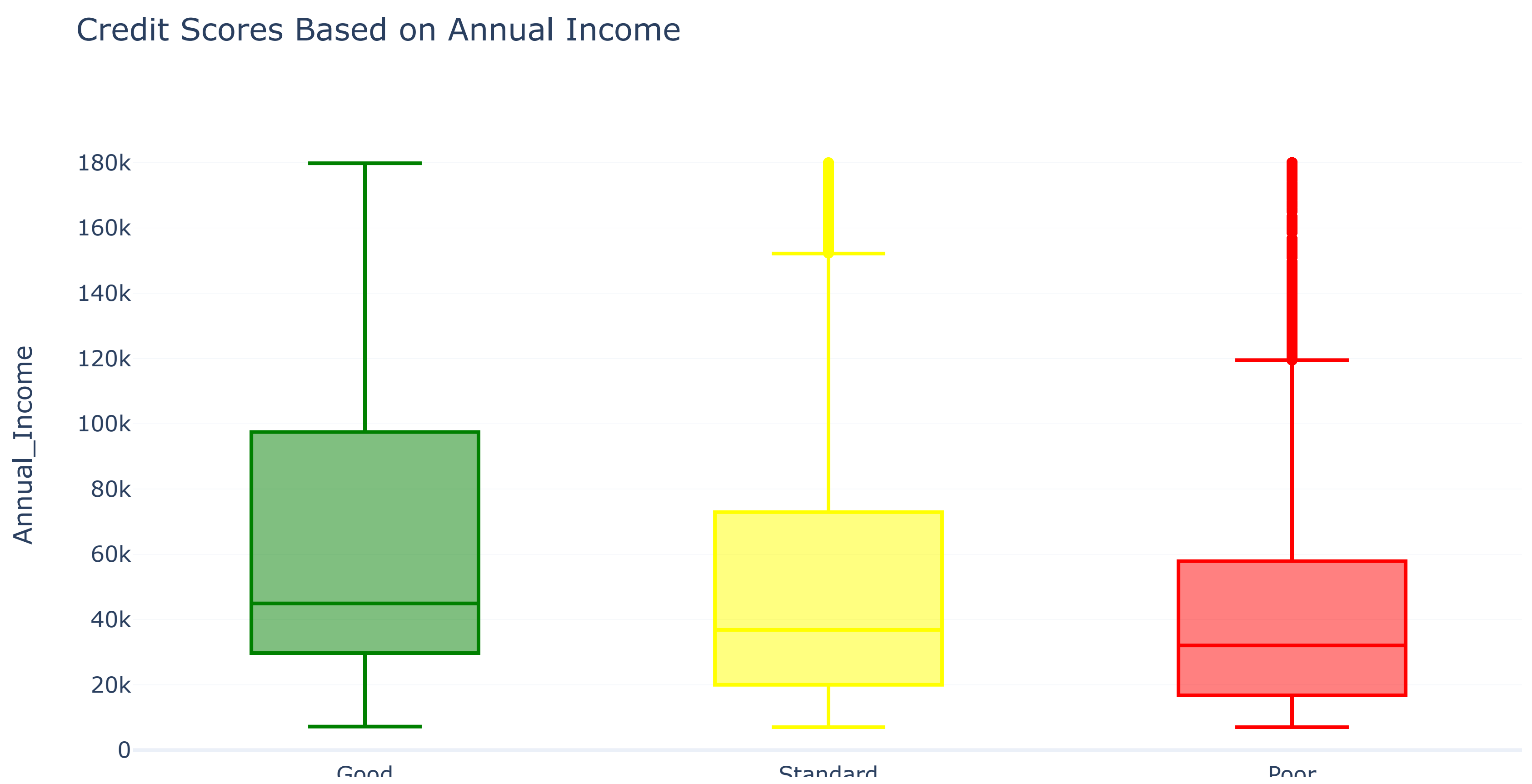
In [7]:
```
fig = px.box(dk,
             x="Occupation",
             color="Credit_Score",
             title="Credit Scores Based on Occupation",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.show()
```



Observation: The credit scores across the various occupations listed in the data show minimal variation.

2. Annual Income

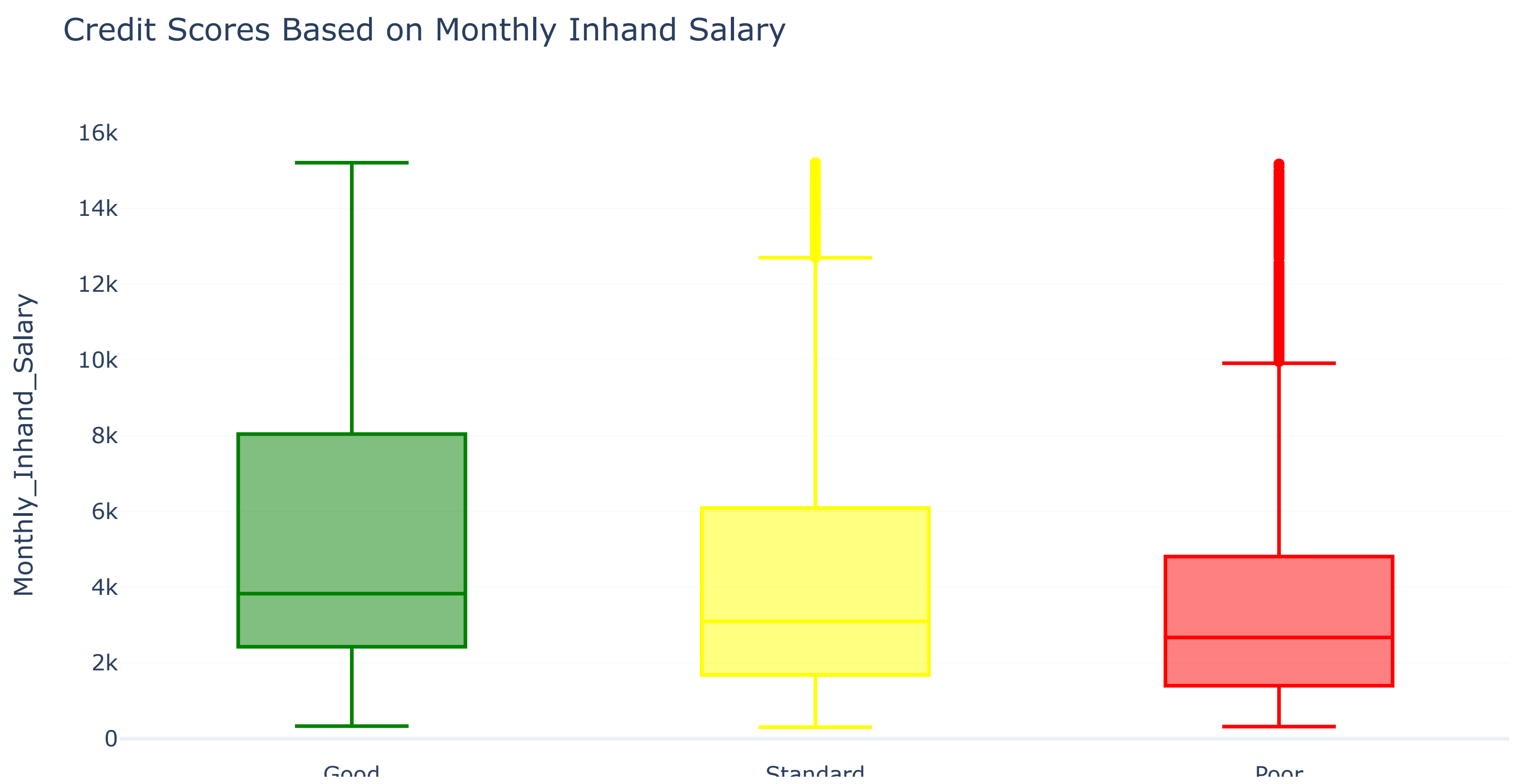
In [8]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Annual_Income",
             color="Credit_Score",
             title="Credit Scores Based on Annual Income",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: The visualization indicates that an individual's credit score improves with an increase in their annual income.

3. Monthly In-hand Salary

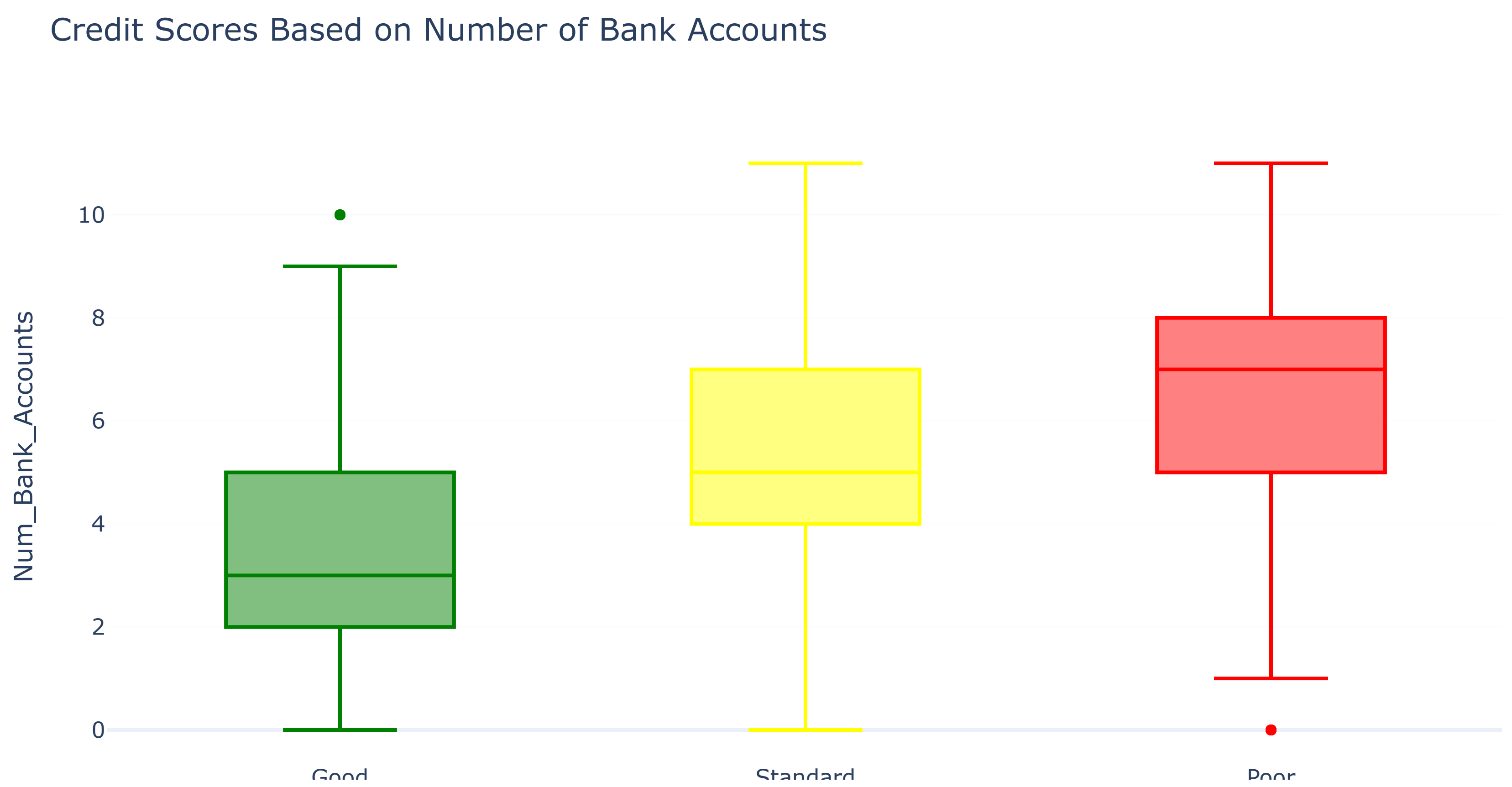
In [9]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Monthly_Inhand_Salary",
             color="Credit_Score",
             title="Credit Scores Based on Monthly Inhand Salary",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Similar to annual income, an increase in your monthly take-home pay contributes to an improvement in your credit score.

4. Number of Bank Accounts

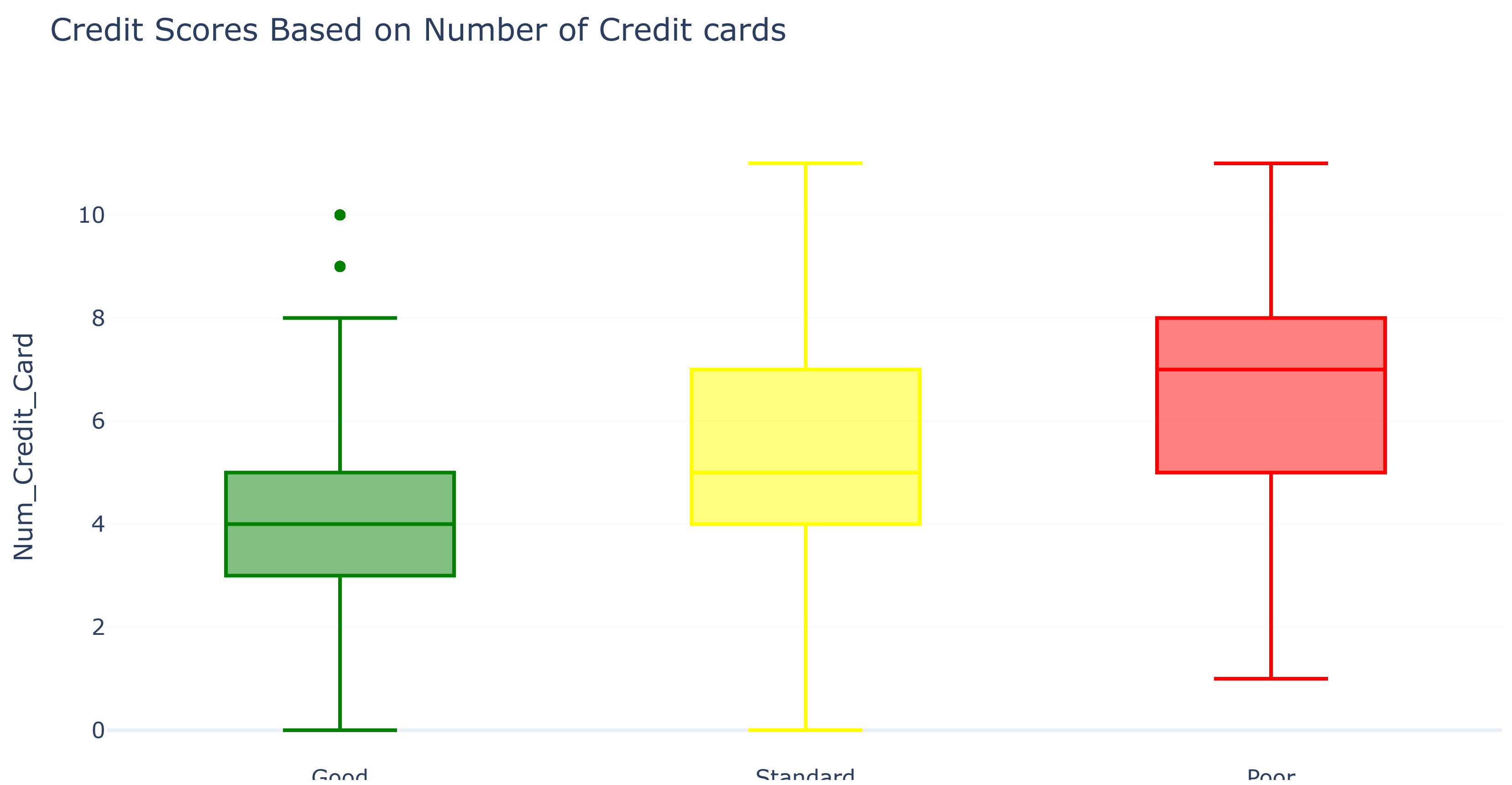
```
In [10]: fig = px.box(dk,
                      x="Credit_Score",
                      y="Num_Bank_Accounts",
                      color="Credit_Score",
                      title="Credit Scores Based on Number of Bank Accounts",
                      color_discrete_map={'Poor':'red',
                                          'Standard':'yellow',
                                          'Good':'green'})
         fig.update_traces(quartilemethod="exclusive")
         fig.show()
```



Observation: Holding over five accounts can negatively affect one's credit score. It is advisable for an individual to limit themselves to having only 2 to 3 bank accounts. Therefore, possessing multiple bank accounts does not have a beneficial effect on credit scores.

5. Number of Credit Cards Owned

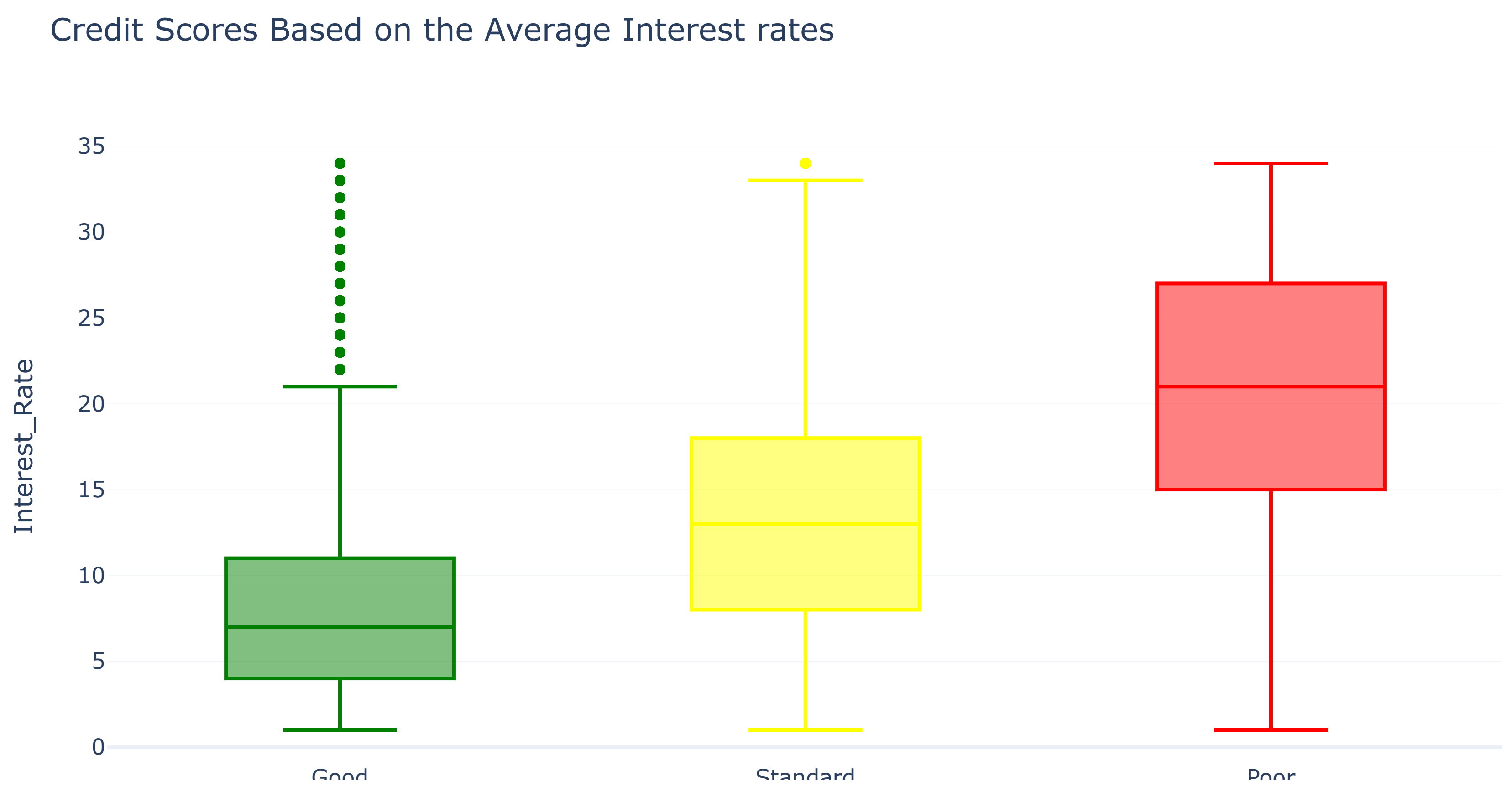
In [11]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Num_Credit_Card",
             color="Credit_Score",
             title="Credit Scores Based on Number of Credit cards",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Similar to the effect of multiple bank accounts, possessing numerous credit cards does not enhance your credit scores. Maintaining between three to five credit cards is beneficial for your credit rating.

6. Average Interest Rates

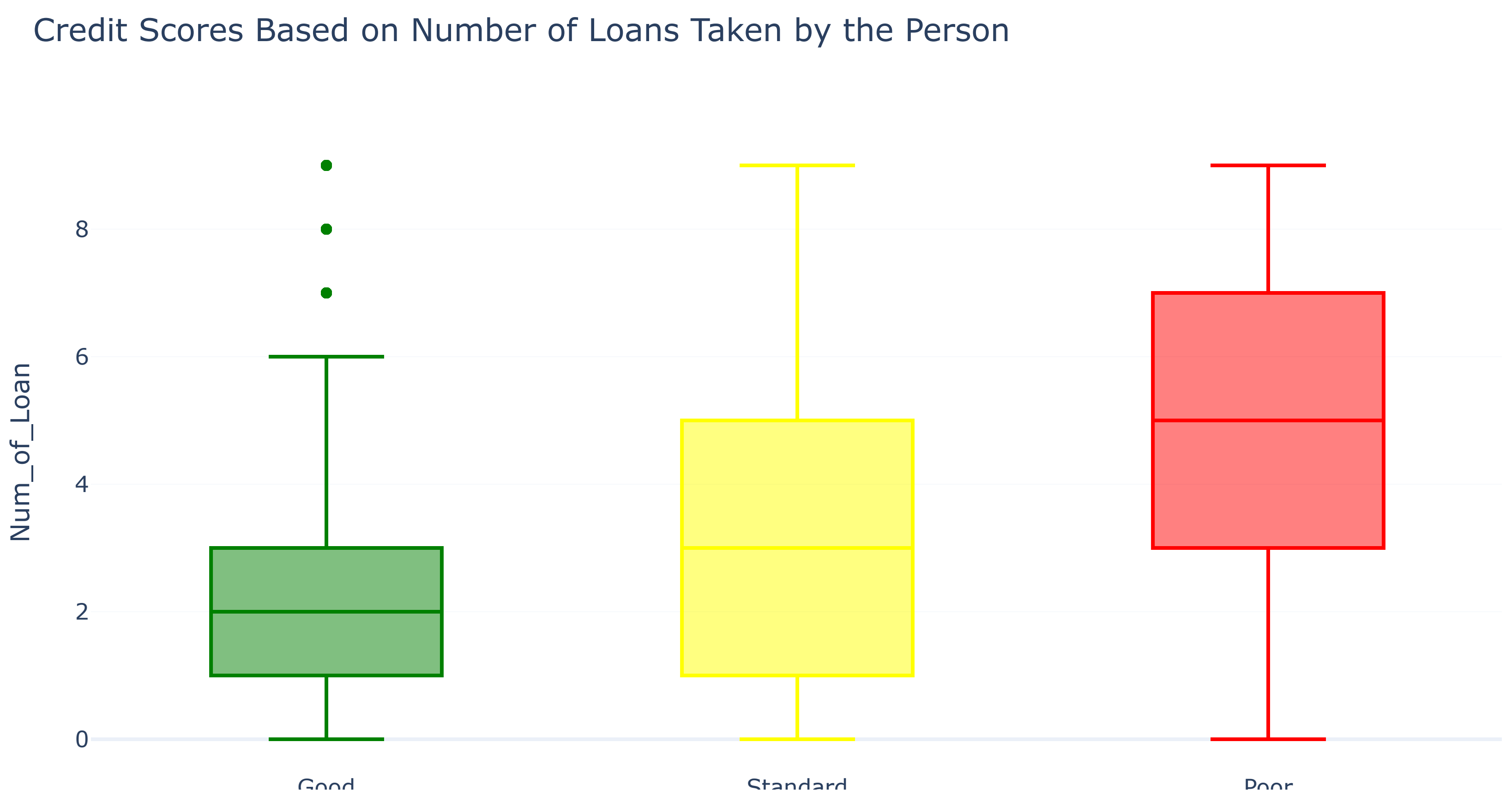
In [12]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Interest_Rate",
             color="Credit_Score",
             title="Credit Scores Based on the Average Interest rates",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: A credit score is considered good if the average interest rate falls between 4% and 11%. Conversely, an average interest rate exceeding 15% negatively impacts your credit scores.

7. Number of Loans Acquired

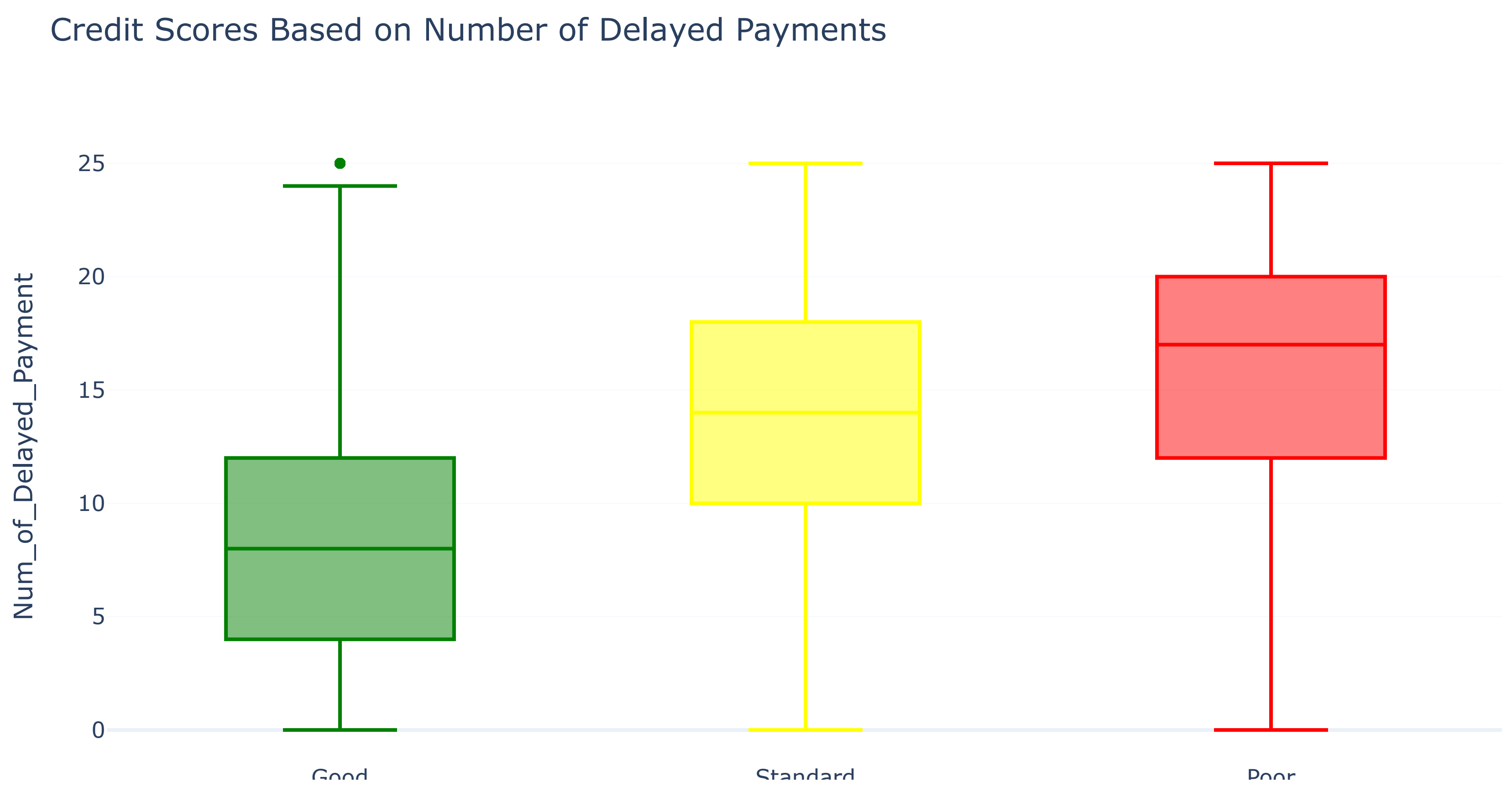
In [13]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Num_of_Loan",
             color="Credit_Score",
             title="Credit Scores Based on Number of Loans Taken by the Person",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Maintaining a good credit score requires limiting oneself to 1 – 3 loans concurrently. Possessing over three loans simultaneously can adversely affect your credit scores.

8. Number of Delayed Payments

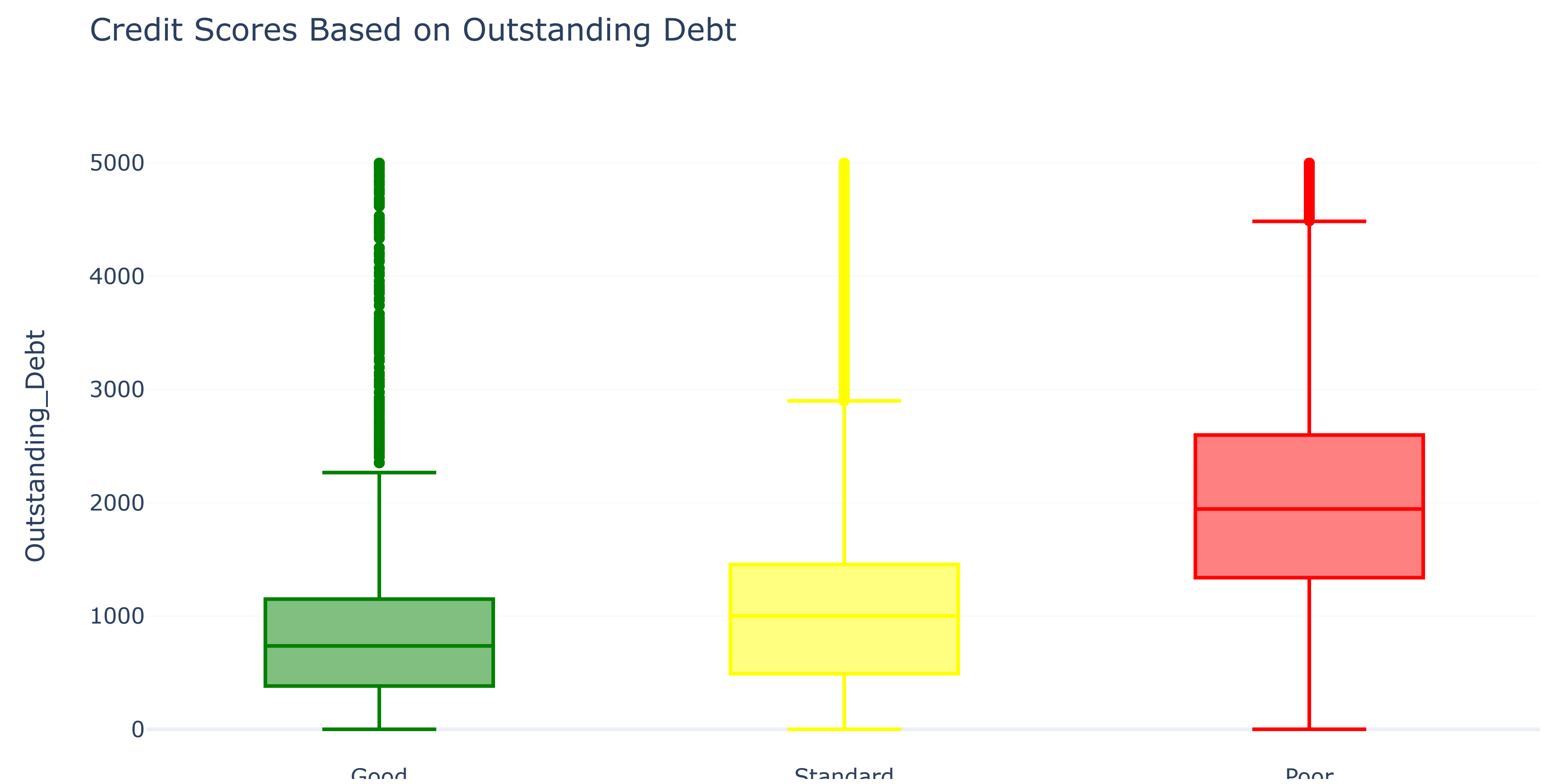
In [14]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Num_of_Delayed_Payment",
             color="Credit_Score",
             title="Credit Scores Based on Number of Delayed Payments",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Postponing payment for 4 to 12 cycles beyond the due date will not impact one's credit scores. However, delaying beyond 12 payment cycles from the due date will have a negative effect on the credit scores.

9. Outstanding Debt

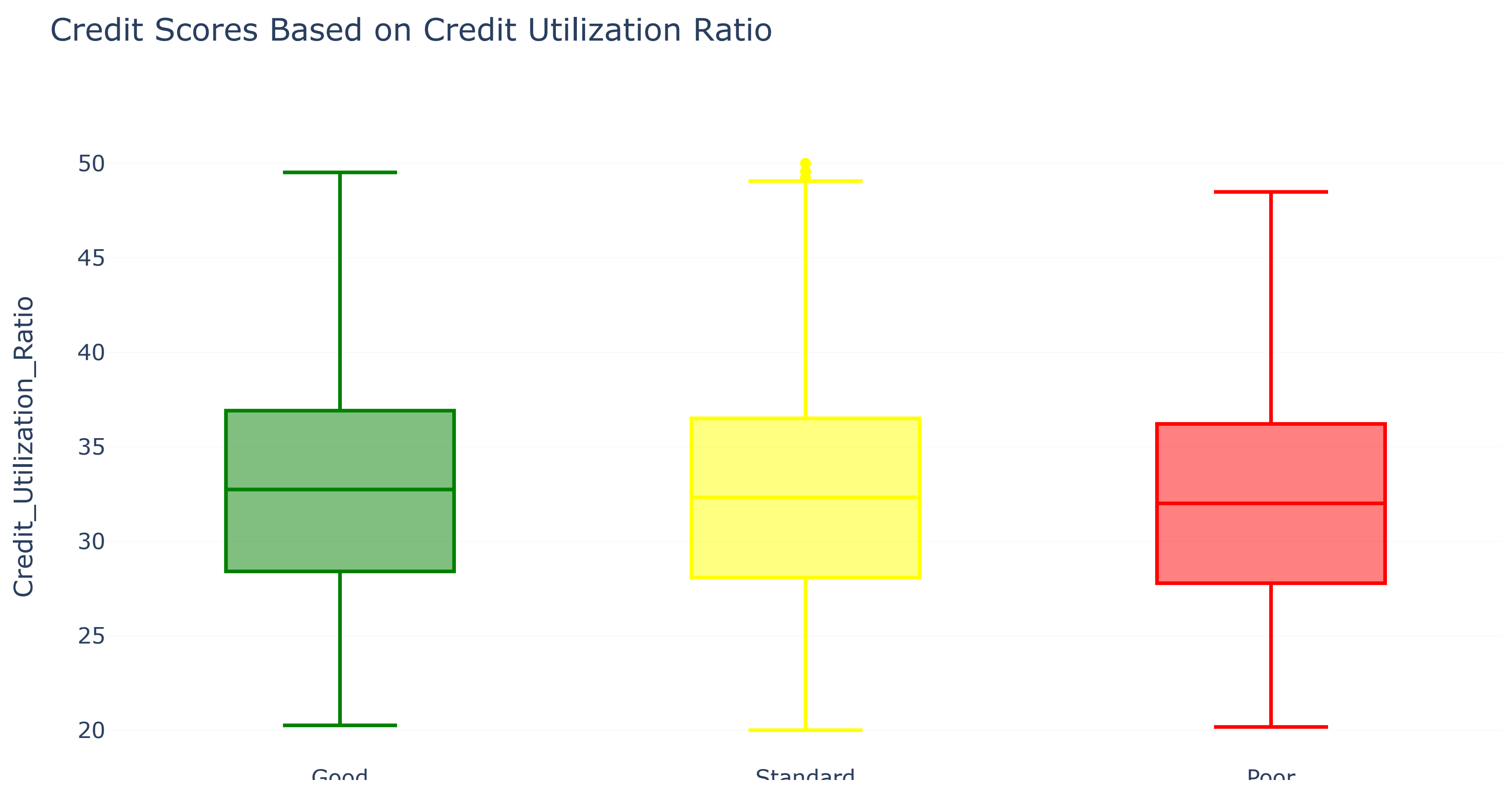
In [15]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Outstanding_Debt",
             color="Credit_Score",
             title="Credit Scores Based on Outstanding Debt",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Having a debt ranging from 380 to 1150 dollars won't impact your credit scores, while consistently carrying a debt exceeding 1338 dollars will have a detrimental effect on your credit standing.

10. Credit Utilization Ratio

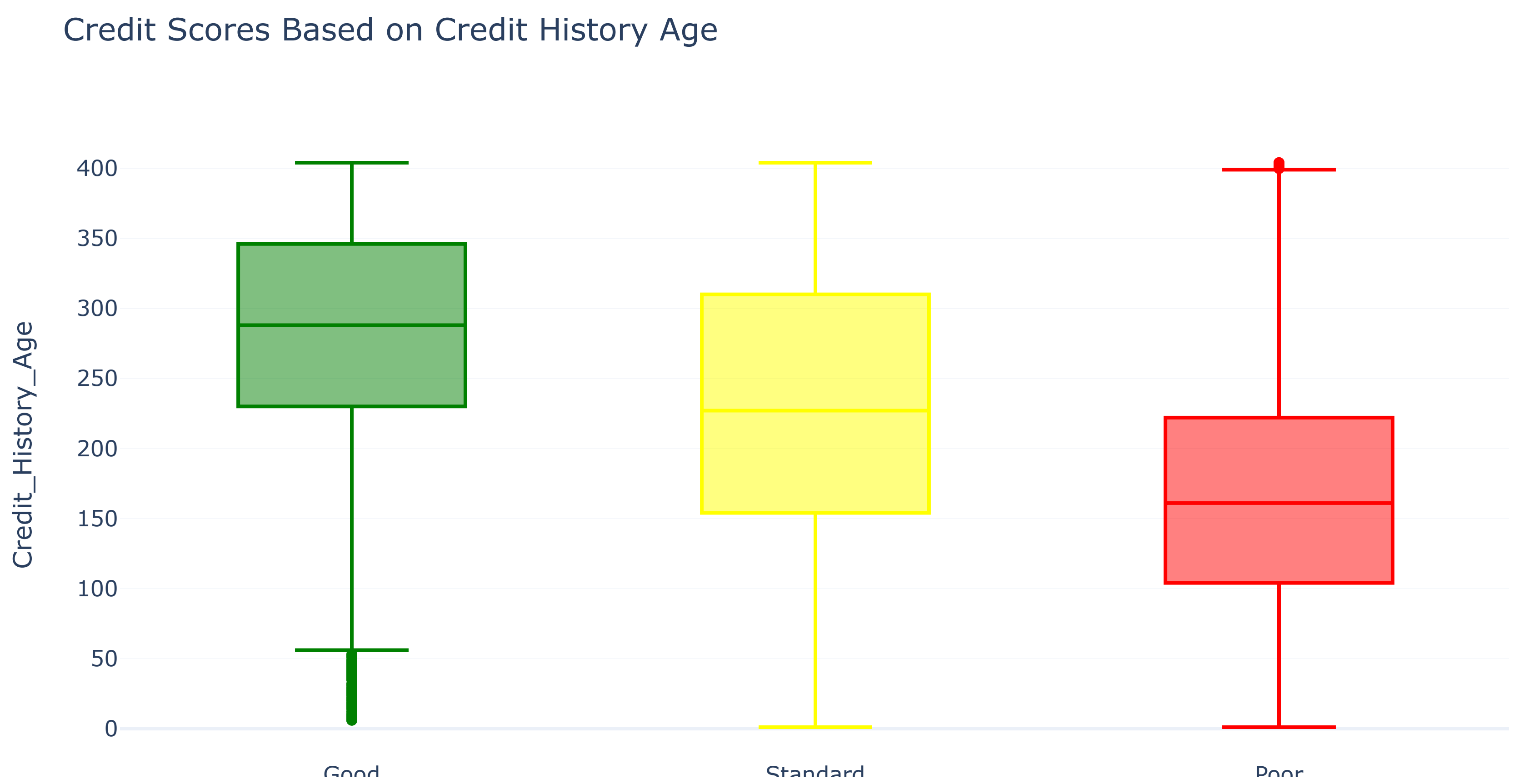
In [16]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Credit_Utilization_Ratio",
             color="Credit_Score",
             title="Credit Scores Based on Credit Utilization Ratio",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: The credit utilization ratio is calculated by dividing your total debt by your total available credit. Based on the data presented, it appears that your credit utilization ratio does not impact your credit scores.

11. Credit History

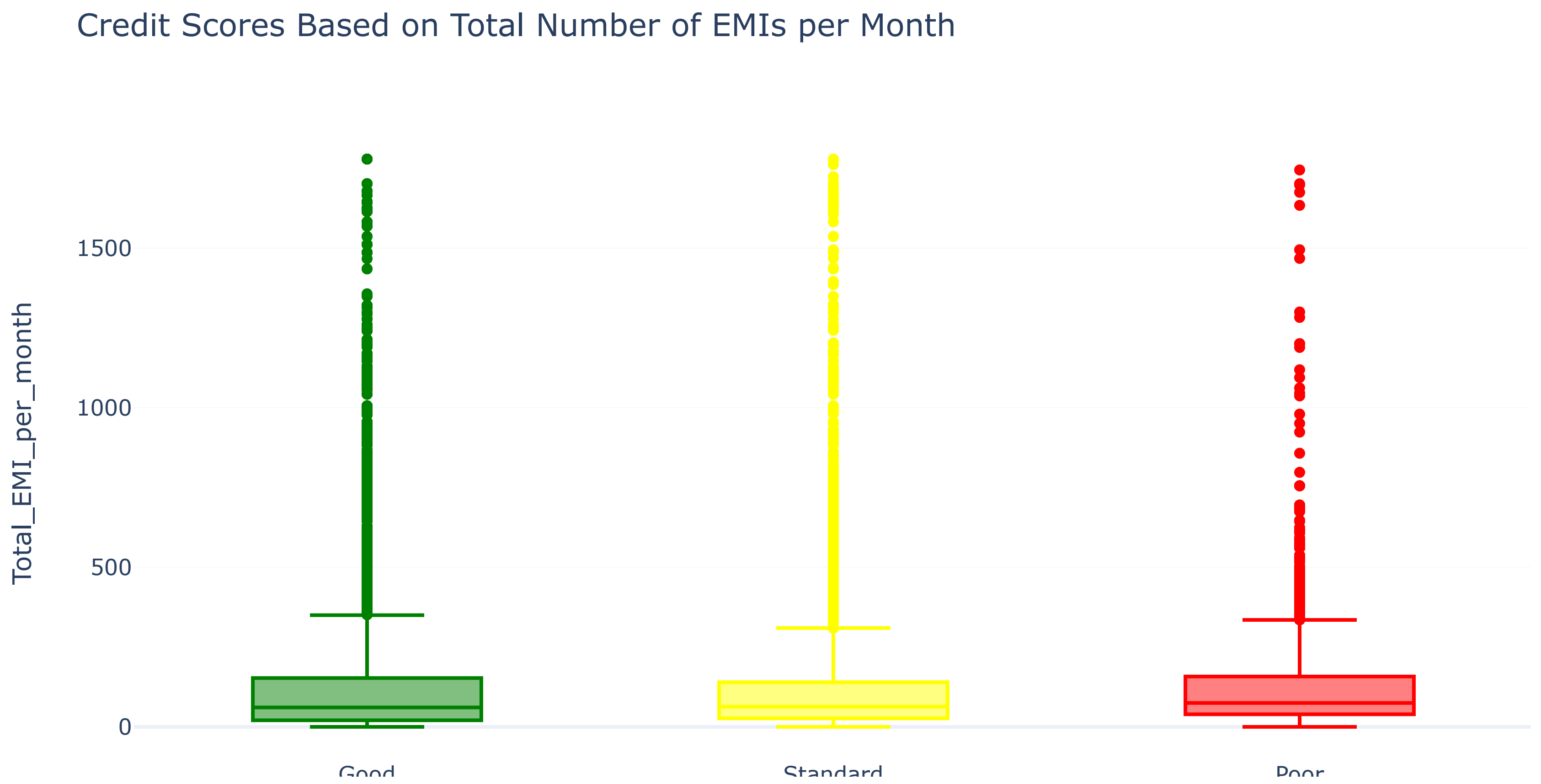
In [17]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Credit_History_Age",
             color="Credit_Score",
             title="Credit Scores Based on Credit History Age",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: Thus, maintaining a lengthy credit history leads to higher credit scores.

12. Number of EMI's

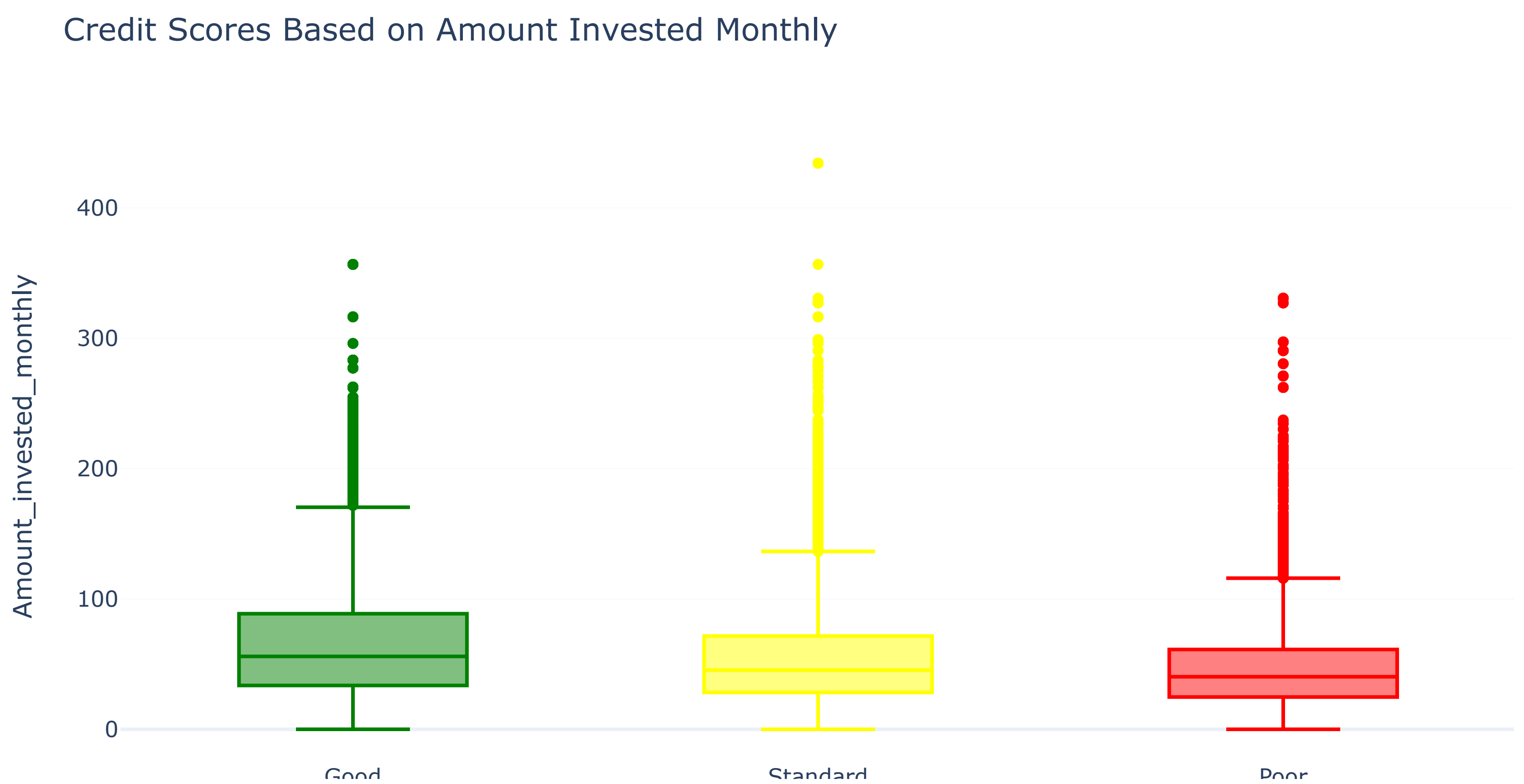
In [18]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Total_EMI_per_month",
             color="Credit_Score",
             title="Credit Scores Based on Total Number of EMIs per Month",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: The number of EMIs you are paying in a month doesn't affect much on credit scores.

13. Amount Invested

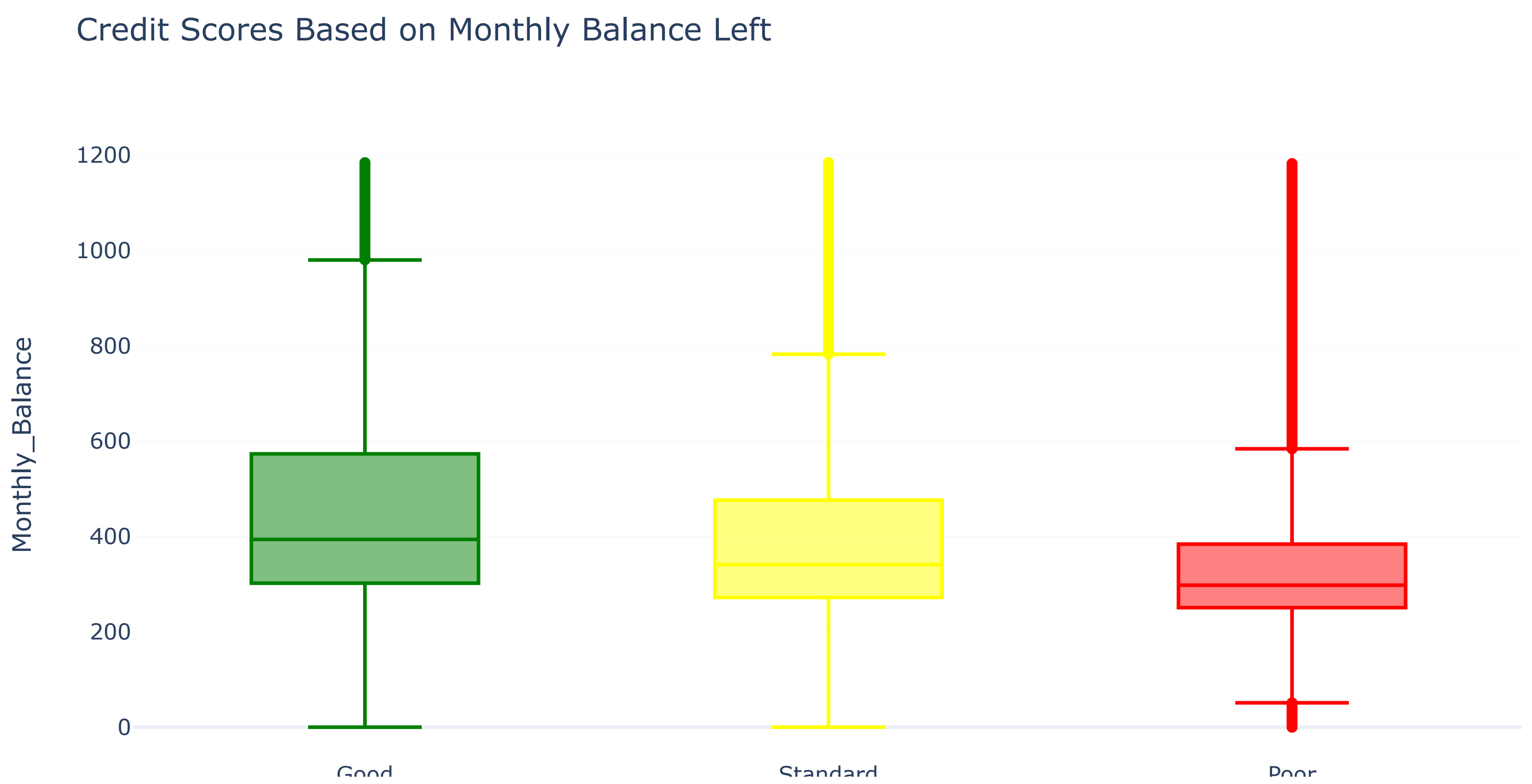
In [19]:
```
fig = px.box(dk,
             x="Credit_Score",
             y="Amount_invested_monthly",
             color="Credit_Score",
             title="Credit Scores Based on Amount Invested Monthly",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



Observation: The amount of money you invest monthly doesn’t affect your credit scores a lot.

14. Monthly Balance Left

```
In [20]: fig = px.box(dk,
                      x="Credit_Score",
                      y="Monthly_Balance",
                      color="Credit_Score",
                      title="Credit Scores Based on Monthly Balance Left",
                      color_discrete_map={'Poor':'red',
                                          'Standard':'yellow',
                                          'Good':'green'})
         fig.update_traces(quartilemethod="exclusive")
         fig.show()
```



Observation: Therefore, maintaining a substantial monthly balance in your account by the end of each month positively impacts your credit scores. Conversely, having a monthly balance below $250 adversely affects your credit scores.

15. Credit Mix

The credit mix characteristic is also an important feature that indicates the variety of credits and loans an individual has obtained. Since the Credit_Mix column is in categorical form, I plan to convert it into a numerical feature. This will enable us to utilize it in training

In [21]:
```
dk["Credit_Mix"] = dk["Credit_Mix"].map({"Standard": 1,
                                "Good": 2,
                                "Bad": 0})
```

Classification Model

In [22]:
```
#Drop columns that are not required

print("Size of Dataset before dropping columns : ",dk.shape)
drop_columns = ['ID','Customer_ID','Name','SSN']
dk.drop(drop_columns,axis=1,inplace=True)
print("Size of Dataset after dropping columns : ",dk.shape)
```

```
Size of Dataset before dropping columns :  (100000, 28)
Size of Dataset after dropping columns :  (100000, 24)
```

In [23]:
```
#Label Encoding
from sklearn.preprocessing import LabelEncoder

categorical_columns = ['Occupation','Type_of_Loan','Credit_Mix','Payment_of_Min_Amount','Payment_Beha

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Loop through each column and apply label encoding
for column in categorical_columns:
    dk[column] = label_encoder.fit_transform(dk[column])
```

In [24]:
```
#Splitting Input & Output Data

X = dk.drop('Credit_Score',axis=1)
y = dk['Credit_Score']
print(X.shape)
print(y.shape)
```

```
(100000, 23)
(100000,)
```

In [25]:
```
#Normalizing the Data

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

In [26]:
```
#Split Data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=17,stratify=y)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(80000, 23)
(20000, 23)
(80000,)
(20000,)
```

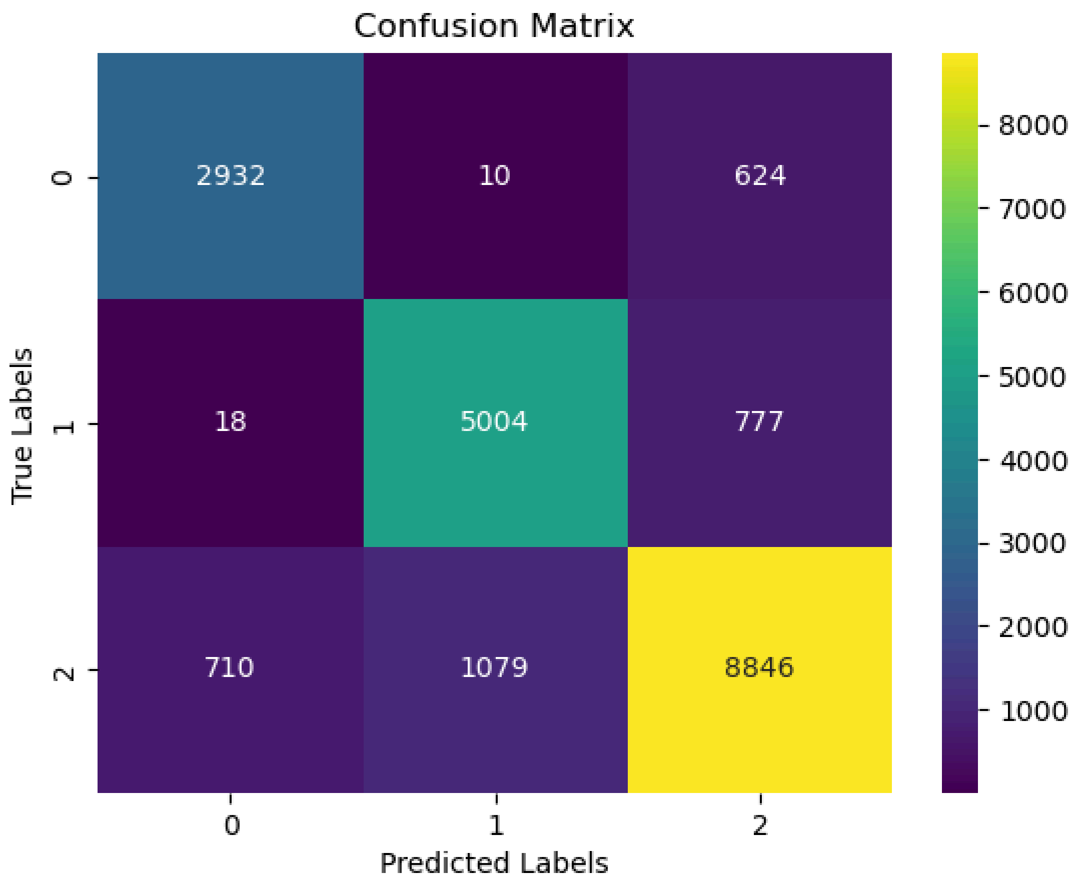
In [27]:
```
#Method to evaluate the performance of the model

def evaluate_model(y_test,y_pred):
    print("Classification Report")
    print(classification_report(y_test, y_pred))

    print("\n---------------------------------------------\n")

    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix using Seaborn
    sns.heatmap(cm, annot=True, cmap='viridis',fmt='.0f')

    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')

    plt.show()
```

In [28]:
```python
# List of classifiers to test
classifiers = [
    ('Decision Tree', DecisionTreeClassifier()),
    ('Random Forest', RandomForestClassifier()),
    ('KNN', KNeighborsClassifier(n_neighbors=5)),
    ('Gaussion NB',GaussianNB()),
    ('XGB',xgb.XGBClassifier())
]

# Iterate over each classifier and evaluate performance
for clf_name, clf in classifiers:
    # Perform cross-validation
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')

    # Calculate average performance metrics
    avg_accuracy = scores.mean()
    avg_precision = cross_val_score(clf, X_train, y_train, cv=5, scoring='precision_macro').mean()
    avg_recall = cross_val_score(clf, X_train, y_train, cv=5, scoring='recall_macro').mean()

    # Print the performance metrics
    print(f'Classifier: {clf_name}')
    print(f'Average Accuracy: {avg_accuracy:.4f}')
    print(f'Average Precision: {avg_precision:.4f}')
    print(f'Average Recall: {avg_recall:.4f}')
    print('-----------------------')
```

```
Classifier: Decision Tree
Average Accuracy: 0.7340
Average Precision: 0.7186
Average Recall: 0.7173
-----------------------
Classifier: Random Forest
Average Accuracy: 0.8233
Average Precision: 0.8151
Average Recall: 0.8195
-----------------------
Classifier: KNN
Average Accuracy: 0.7108
Average Precision: 0.6849
Average Recall: 0.6943
-----------------------
Classifier: Gaussion NB
Average Accuracy: 0.6350
Average Precision: 0.6339
Average Recall: 0.6915
-----------------------
Classifier: XGB
Average Accuracy: 0.7811
Average Precision: 0.7674
Average Recall: 0.7715
-----------------------
```

In [29]:
```
# Creating the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the classifier
rf_classifier.fit(X_train, y_train)

# Making predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluating the model
evaluate_model(y_test, y_pred)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.82      0.81      3566
           1       0.82      0.86      0.84      5799
           2       0.86      0.83      0.85     10635

    accuracy                           0.84     20000
   macro avg       0.83      0.84      0.83     20000
weighted avg       0.84      0.84      0.84     20000


---------------------------------------------
```


Confusion Matrix
True Labels
Predicted Labels
0
1
2
0
1
2
2932
10
624
18
5004
777
710
1079
8846
8000
7000
6000
5000
4000
3000
2000
1000

```
In [ ]:
```