राष्ट्रीय औद्योगिक प्रशिक्षण संस्थान
National Institute for Industrial Training

One premier Ogranization with Non Profit Status Registered Under Govt.of WB. Certificate of Registration Under Section 60 and Rule No.69 Govt.of West Bengal

Affiliated To

अखिल भारतीय प्रतिष्ठित संकाय परिषद इंजीनियरिंग, प्रबंधन और प्रौद्योगिकी
All India Eminent Faculty Council of Engineering,Management & Technology
Registered & Licensed By Ministry of Corporate Affairs Govt. of India,Section-8

Only Organization in India Conducting Most Authentic Summer Training-Winter Training-Online Internship with its Associates. Eligibility: Any Engineering/BCA/MCA/BSc(Comp. Sc)/ MSc.(Comp. Sc)/Diploma Engg. Students Can Apply.

python

Subject : Credit Risk Prediction

Submitted By : SHOBHIT SHRIVASTAVA

Submitted To : *Soumotanu Mazumdar*

SUBMITTED ON : *JUNE 22, 2021*

Project Content

Student profile

Name: SHOBHIT SHRIVASTAVA

College Name: Geeta Engineering College,Panipat

Stream: Computer Science *Engineering*

***Project** Name: Credit Risk Prediction*

Email Shobhitshrivastava284@gmail.com

Phone ***7206412501***

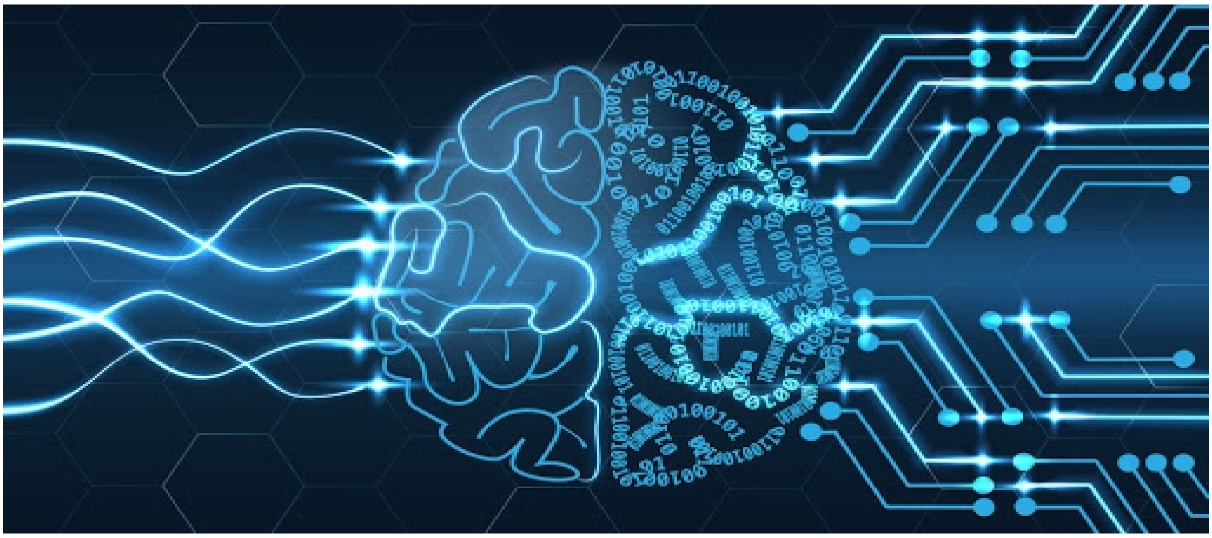
ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher and project guide MR. Soumotanu Majumdar who gave me the golden opportunity to do this wonderful project of **Credit Risk Prediction,** which also helped me doing a lot of Research and I came to know about so many things. Secondly i would also like to thanks my friends who helped me a lot in finalizing this project within the limited time frame. I have put tremendous effort in this project. However, it would not have been possible without the kind support and help of above-mentioned individual and organization. I would like to extend my sincere thanks to all of them.

DATE : *22.6.21* *shobhit*

Introduction

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.



Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate is available now in market.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes.

In this tutorial, we’ll look into the common machine learning methods of supervised and unsupervised learning, and common algorithmic approaches in machine learning, including the k-nearest neighbor algorithm, decision tree learning, and deep learning. We’ll explore which programming languages are most used in machine learning, providing you with some of the positive and negative attributes of each. Additionally, we’ll discuss biases that are perpetuated by machine learning algorithms, and consider what can be kept in mind to prevent these biases when building algorithms.

Hardware Requirements

- Intel core i3 8th generation is used as a processor because it is faster and provide reliable and stable working environment clocked at 2.4GHz.
- A ram size of 4 gb is used as it will provide fast reading and writing capabilities.
- Minimum required Hard Disk space should be 1Tb.

Software Requirements

- Python required version is 3.5 or 3.7.
- Anaconda platform .
- Operating system can be of windows /Linux/Mac

Objective

Credit risk is simply known as the possibility of a loss for a lender due to a borrower's failure to repay a loan. Minimizing the risk of default is a major concern for financial institutions. For this reason, commercial and investment banks, venture capital funds, asset management companies and insurance firms, to name a few, are increasingly relying on technology to predict which clients are more prone to stop honoring their debts.

Credit analysts are typically responsible for assessing this risk by thoroughly analyzing a borrower's capability to repay a loan

But now days **Machine Learning** models have been helping these companies to improve the accuracy of their credit risk analysis, providing a scientific method to identify potential debtors in advance.

Machine learning algorithms have a lot to offer to the world of credit risk assessment due to their unparalleled predictive power and speed. In this article, we will be utilizing machine learning's power to predict whether a borrower will default on a loan or not and to predict their probability of default.

The data sheet we are using has 10 columns and 999 rows.

Future Scope and advantage:

Machine learning are revolutionizing all sectors in the global economy with banking and financial institutions being no exception to this trend. They are adopting latest software and bots to change the face and image of their industry and offer extraordinary services to their existing and future customers.

Banking and financial sectors have been using some form of machine learning to keep track of data but it is usually tedious and manual in nature. With high volume of data, accurate historical records and the quantitative nature of financial institutions, this sector is particularly suited for artificial intelligence.

With help of Credit Risk Prediction the future of banking sector is more accurately secure.

Source code

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Reading Csv file

In [2]:
```
df = pd.read_csv("credit_data.csv")
```

In [3]:
```
df
```

Out[3]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	2	49	male	1	own	little	NaN	2096	12	education	good
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	4	53	male	2	free	little	little	4870	24	car	bad
...	...	...	...	...	...	...	...	...	...	...	...
995	995	31	female	1	own	little	NaN	1736	12	furniture/equipment	good
996	996	40	male	3	own	little	little	3857	30	car	good
997	997	38	male	2	own	little	NaN	804	12	radio/TV	good
998	998	23	male	2	free	little	little	1845	45	radio/TV	bad
999	999	27	male	2	own	moderate	moderate	4576	45	car	good

1000 rows × 11 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Unnamed: 0            0
Age                   0
Sex                   0
Job                   0
Housing               0
Saving accounts     183
Checking account    394
Credit amount         0
Duration              0
Purpose               0
Risk                  0
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Unnamed: 0        1000 non-null   int64 
 1   Age               1000 non-null   int64 
 2   Sex               1000 non-null   object
 3   Job               1000 non-null   int64 
 4   Housing           1000 non-null   object
 5   Saving accounts   817 non-null    object
 6   Checking account  606 non-null    object
 7   Credit amount     1000 non-null   int64 
 8   Duration          1000 non-null   int64 
 9   Purpose           1000 non-null   object
 10  Risk              1000 non-null   object
dtypes: int64(5), object(6)
memory usage: 86.1+ KB
```

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

Out[6]:

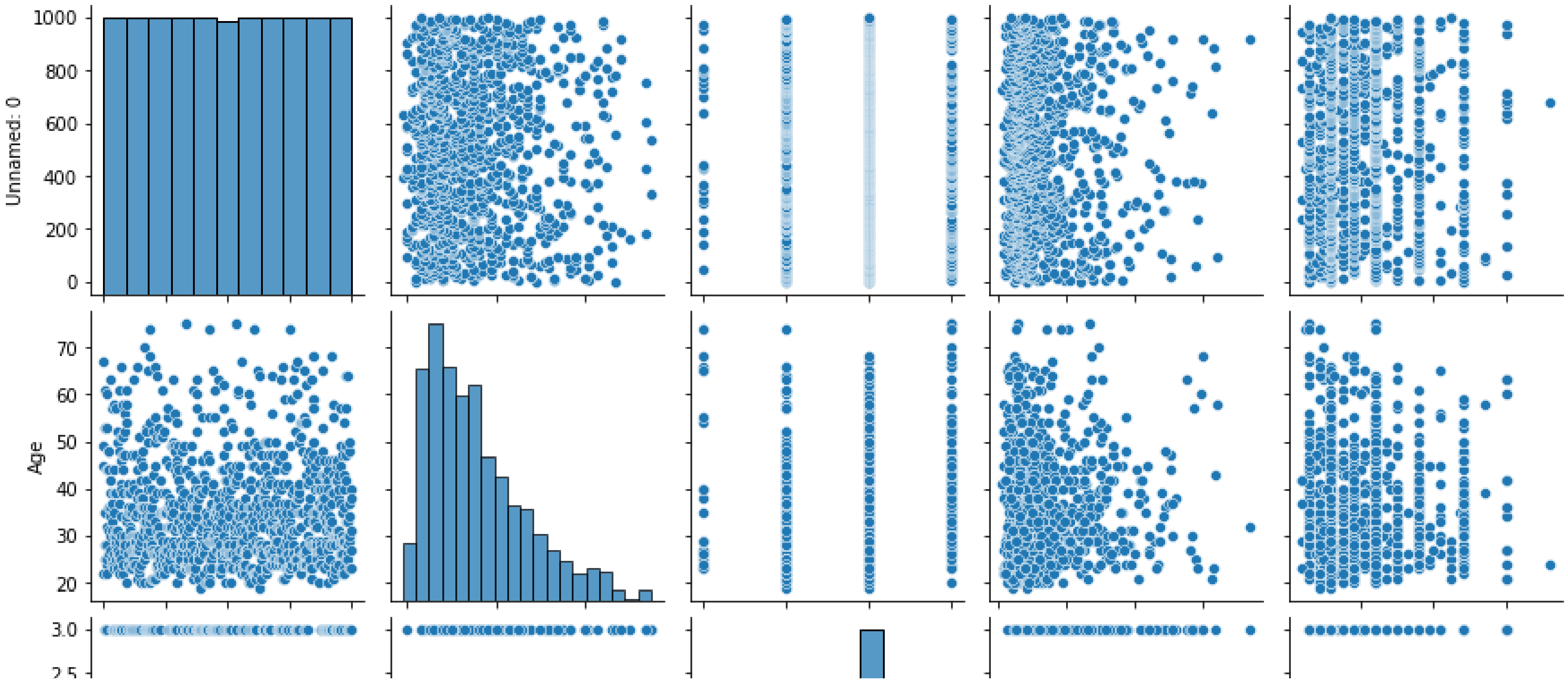
	Unnamed: 0	Age	Job	Credit amount	Duration
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	499.500000	35.546000	1.904000	3271.258000	20.903000
std	288.819436	11.375469	0.653614	2822.736876	12.058814

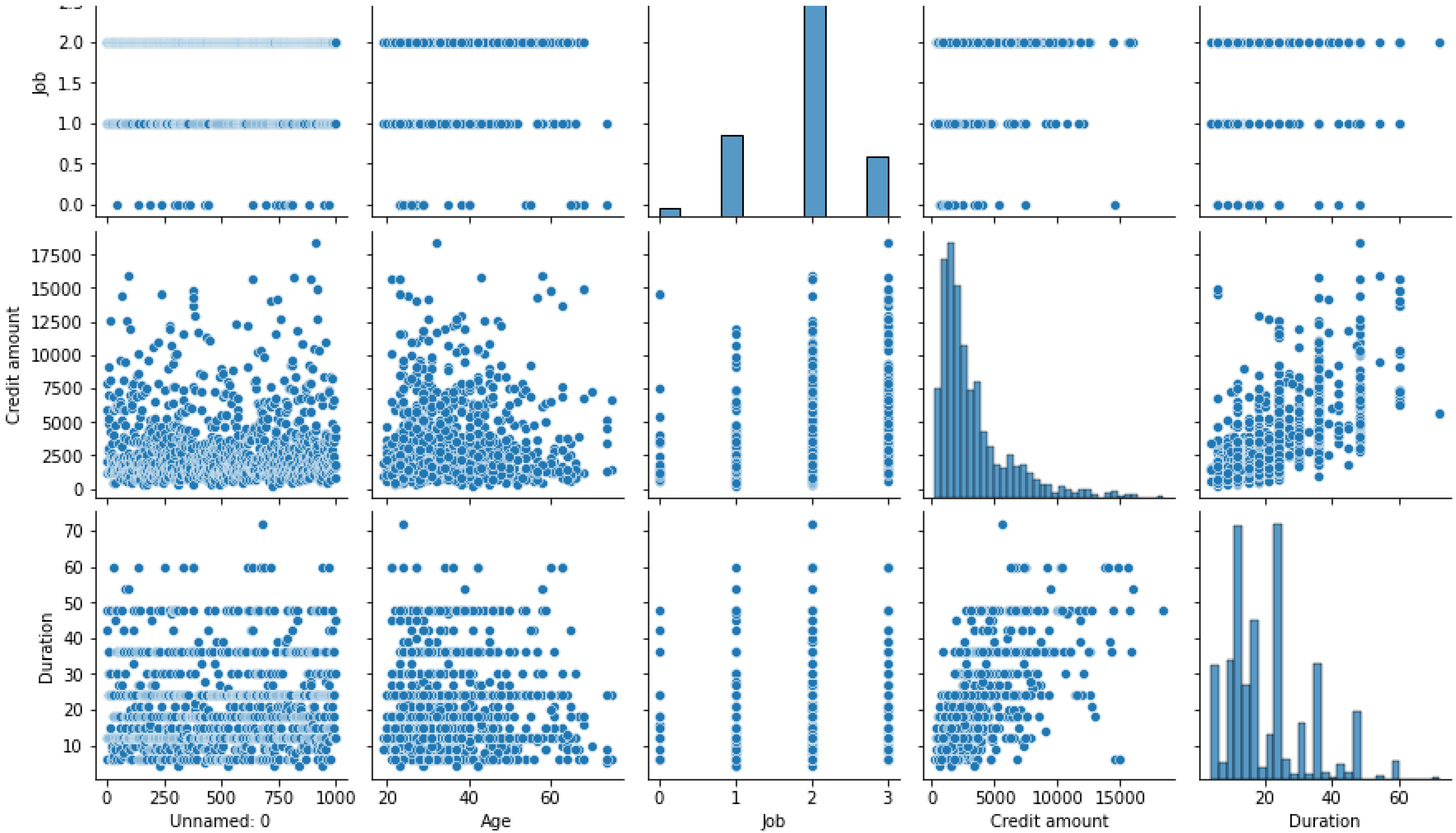
	Unnamed: 0	Age	Job	Credit amount	Duration
min	0.000000	19.000000	0.000000	250.000000	4.000000
25%	249.750000	27.000000	2.000000	1365.500000	12.000000
50%	499.500000	33.000000	2.000000	2319.500000	18.000000
75%	749.250000	42.000000	2.000000	3972.250000	24.000000
max	999.000000	75.000000	3.000000	18424.000000	72.000000

EDA(Exploratory Data Analysis)

In [7]:
```
sns.pairplot(df)
```

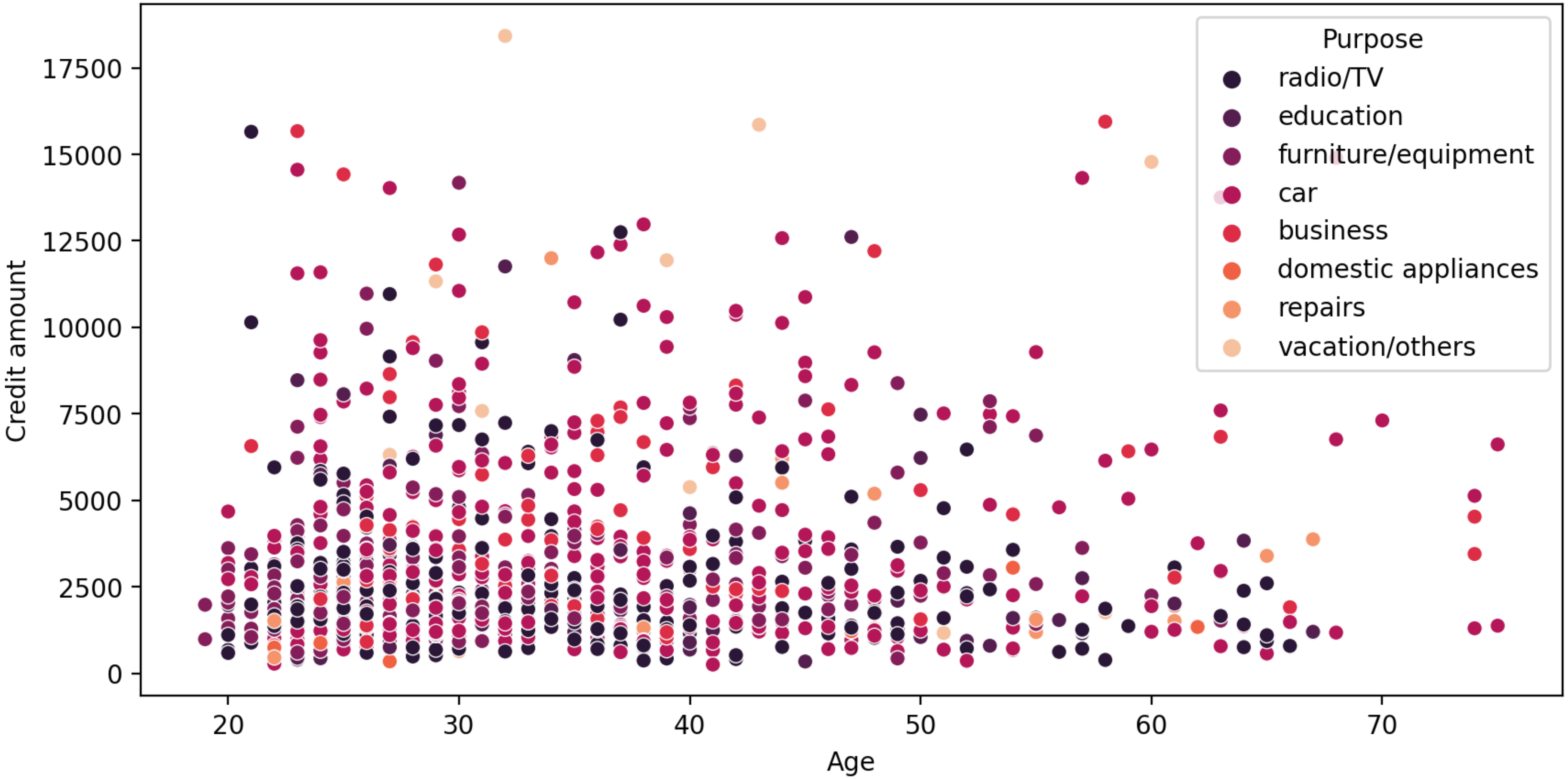
Out[7]: <seaborn.axisgrid.PairGrid at 0x228a8f6cf70>





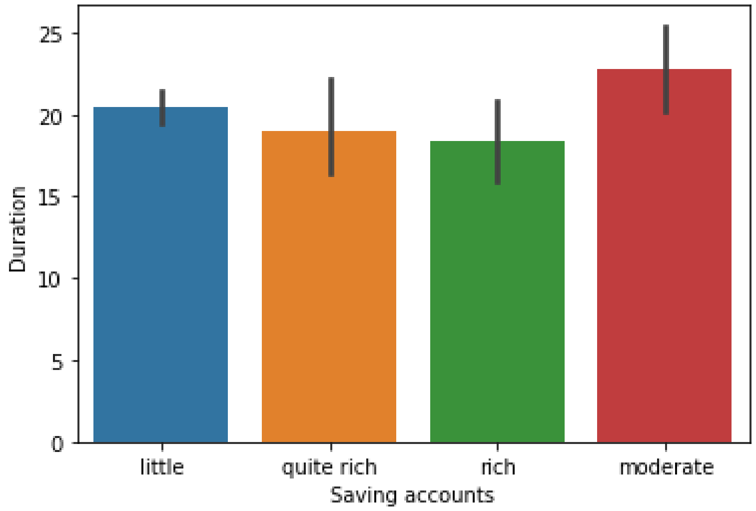
In [8]:
```
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x="Age",y="Credit amount",data=df,hue="Purpose",palette="rocket")
```

Out[8]: <AxesSubplot:xlabel='Age', ylabel='Credit amount'>



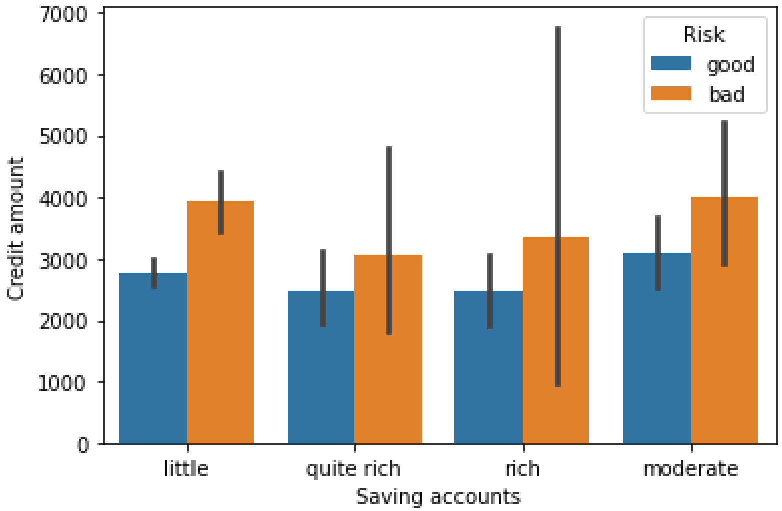
In [9]:
```
sns.barplot(x="Saving accounts",y="Duration",data=df)
```

Out[9]: <AxesSubplot:xlabel='Saving accounts', ylabel='Duration'>



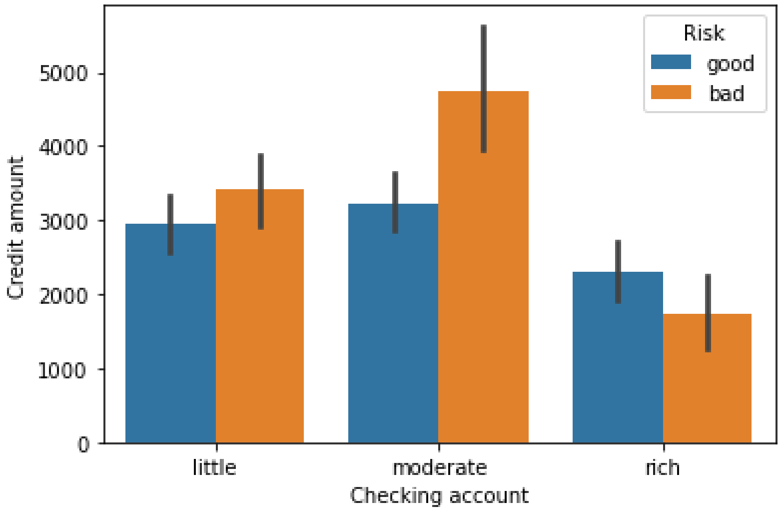
In [10]:
```
sns.barplot(x="Saving accounts",y="Credit amount",data=df,hue="Risk")
```

Out[10]: <AxesSubplot:xlabel='Saving accounts', ylabel='Credit amount'>



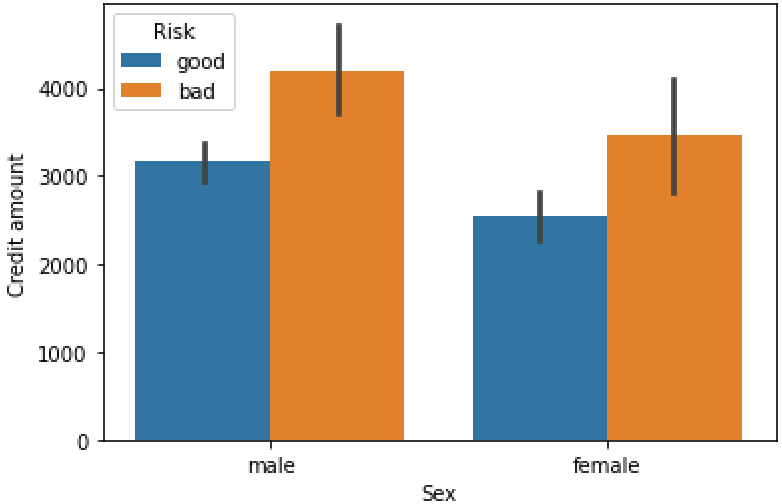
In [11]:
```
sns.barplot(x="Checking account",y="Credit amount",data=df,hue="Risk")
```

Out[11]: <AxesSubplot:xlabel='Checking account', ylabel='Credit amount'>



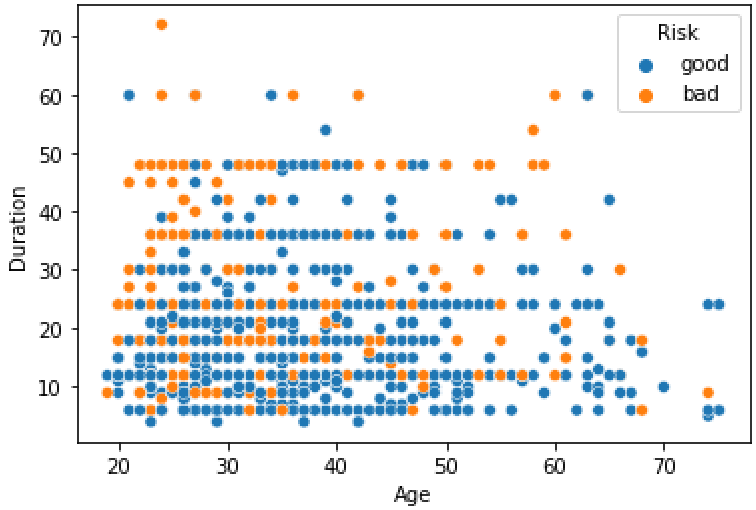
In [12]:
```
sns.barplot(x="Sex",y="Credit amount",data=df,hue="Risk")
```

Out[12]: <AxesSubplot:xlabel='Sex', ylabel='Credit amount'>



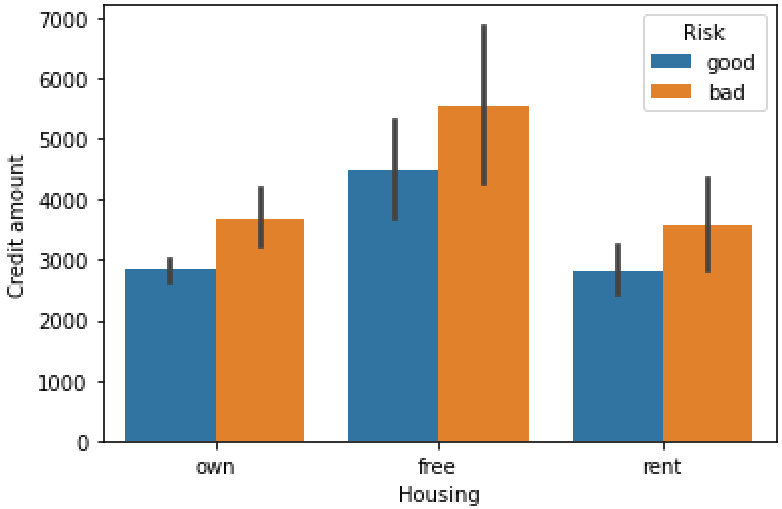
In [13]:
```
sns.scatterplot(x="Age",y="Duration",data=df,hue="Risk")
```

Out[13]: <AxesSubplot:xlabel='Age', ylabel='Duration'>



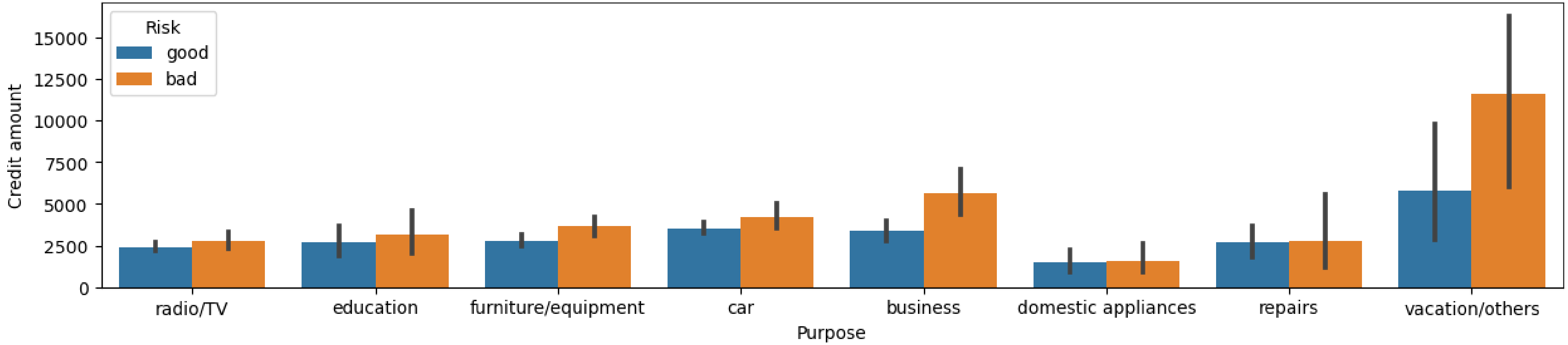
In [14]:
```
sns.barplot(x="Housing",y="Credit amount",data=df,hue="Risk")
```

Out[14]: <AxesSubplot:xlabel='Housing', ylabel='Credit amount'>



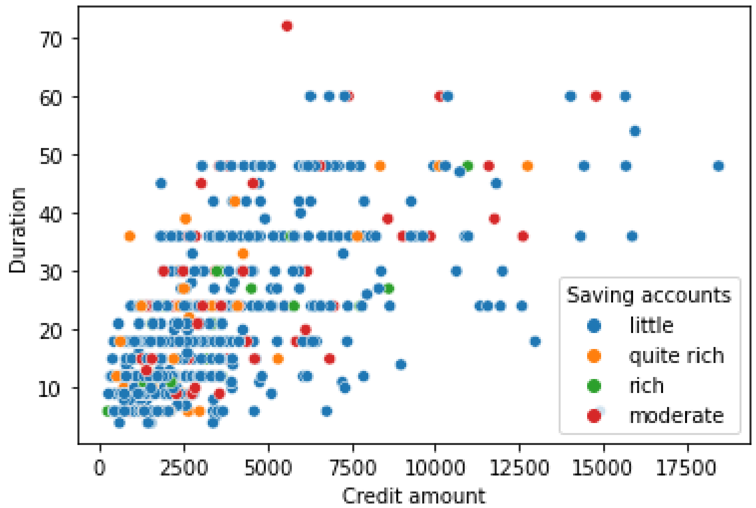
In [15]:
```
plt.figure(figsize=(15,3),dpi=100)
sns.barplot(x="Purpose",y="Credit amount",data=df,hue="Risk")
```

Out[15]: <AxesSubplot:xlabel='Purpose', ylabel='Credit amount'>



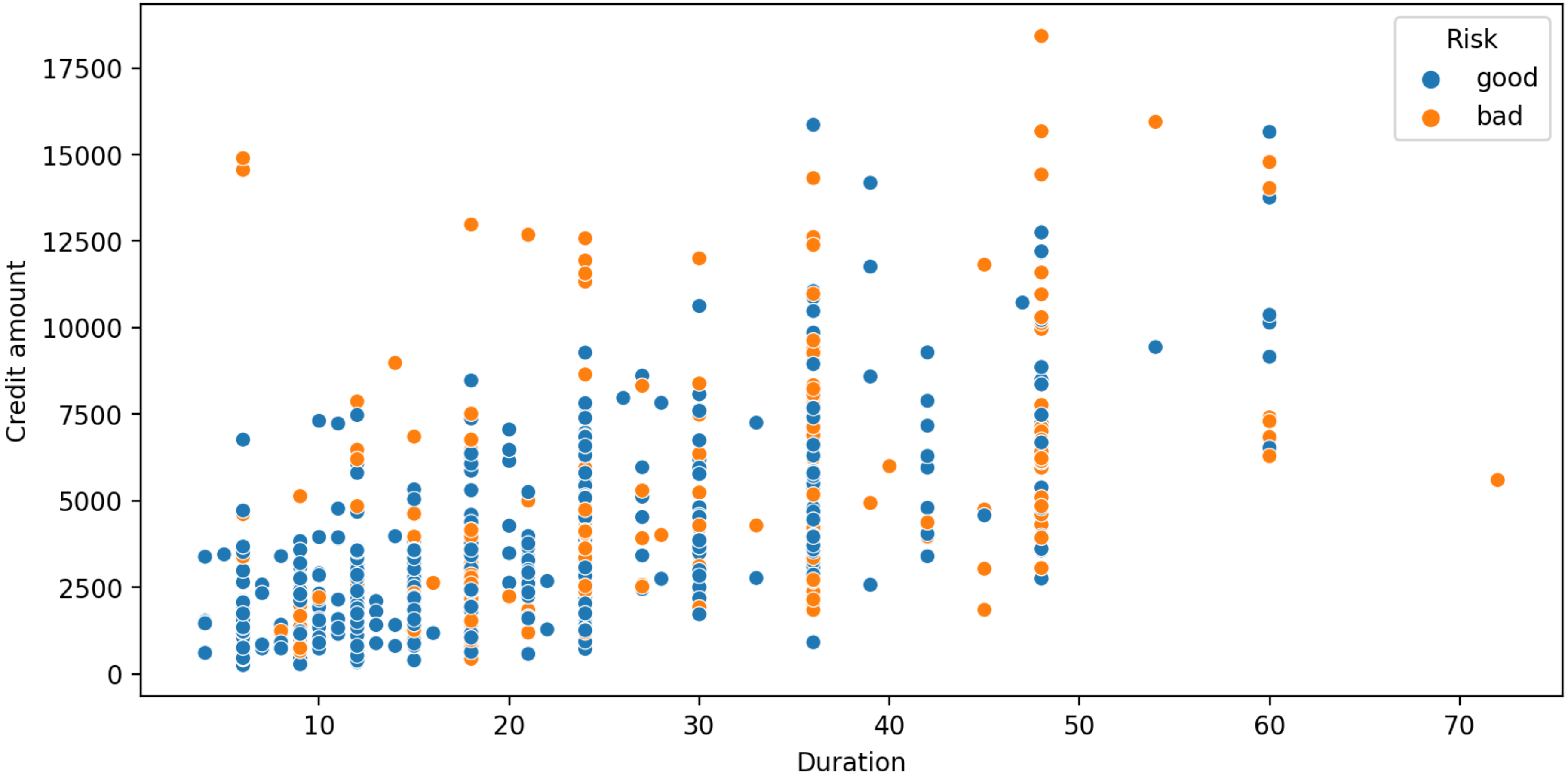
In [16]:
```
sns.scatterplot(x="Credit amount",y="Duration",data=df,hue="Saving accounts")
```

Out[16]: <AxesSubplot:xlabel='Credit amount', ylabel='Duration'>



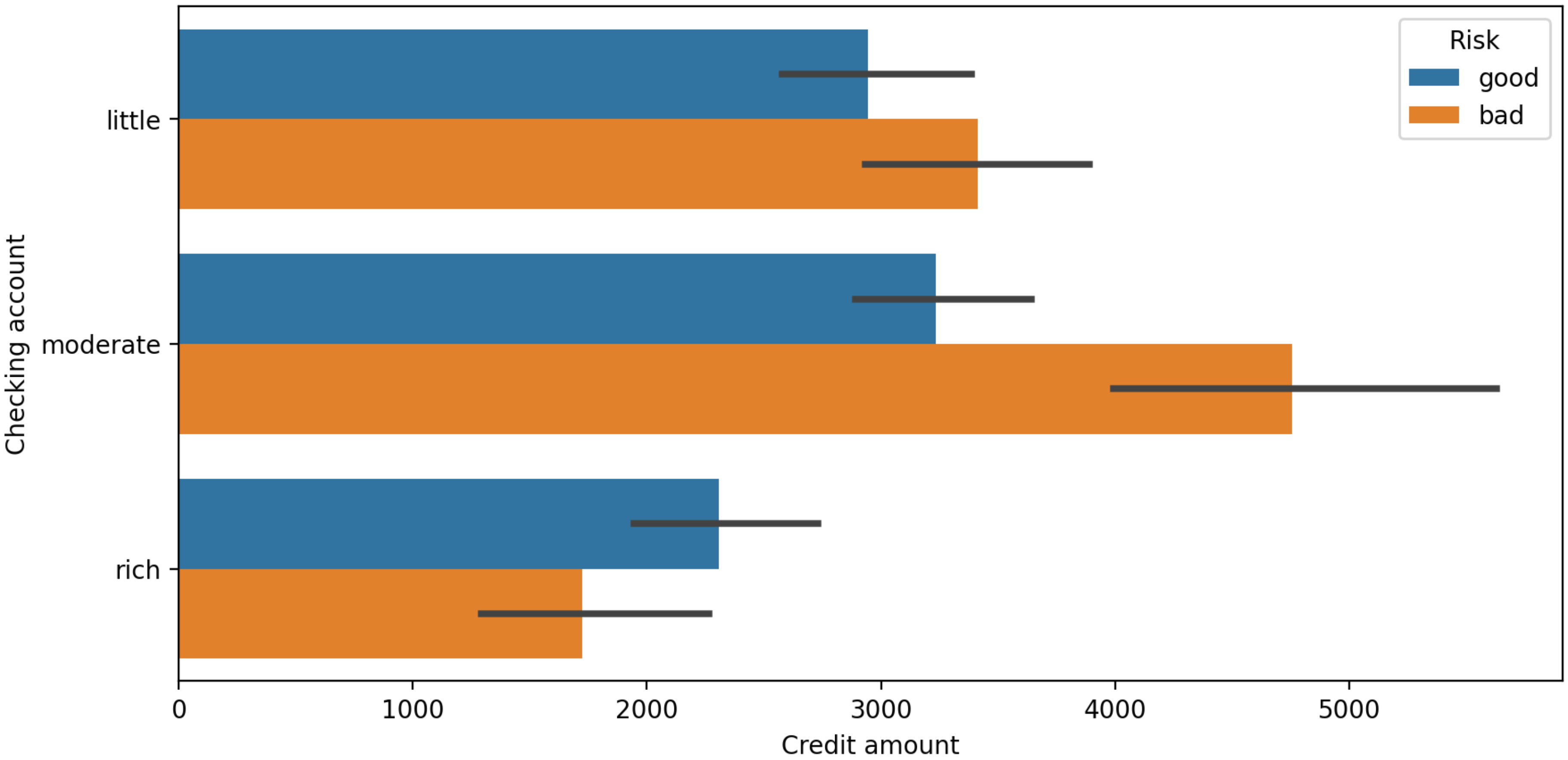
In [17]:
```
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x="Duration",y="Credit amount",data=df,hue="Risk")
```

Out[17]: <AxesSubplot:xlabel='Duration', ylabel='Credit amount'>



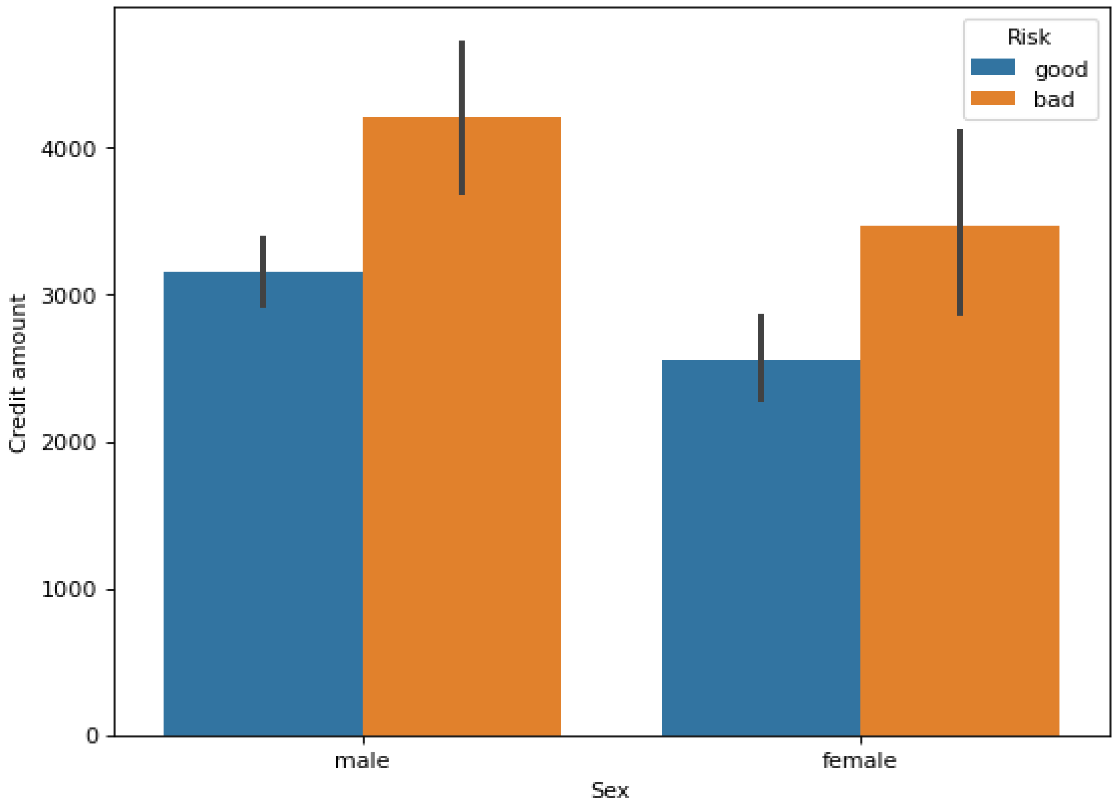
In [18]:
```
plt.figure(figsize=(10,5),dpi=200)
sns.barplot(x="Credit amount",y="Checking account",data=df,hue="Risk")
```

Out[18]: <AxesSubplot:xlabel='Credit amount', ylabel='Checking account'>



In [19]:
```
plt.figure(figsize=(8,6),dpi=80)
sns.barplot(x="Sex",y="Credit amount",hue="Risk",data=df)
```

Out[19]:
```
<AxesSubplot:xlabel='Sex', ylabel='Credit amount'>
```

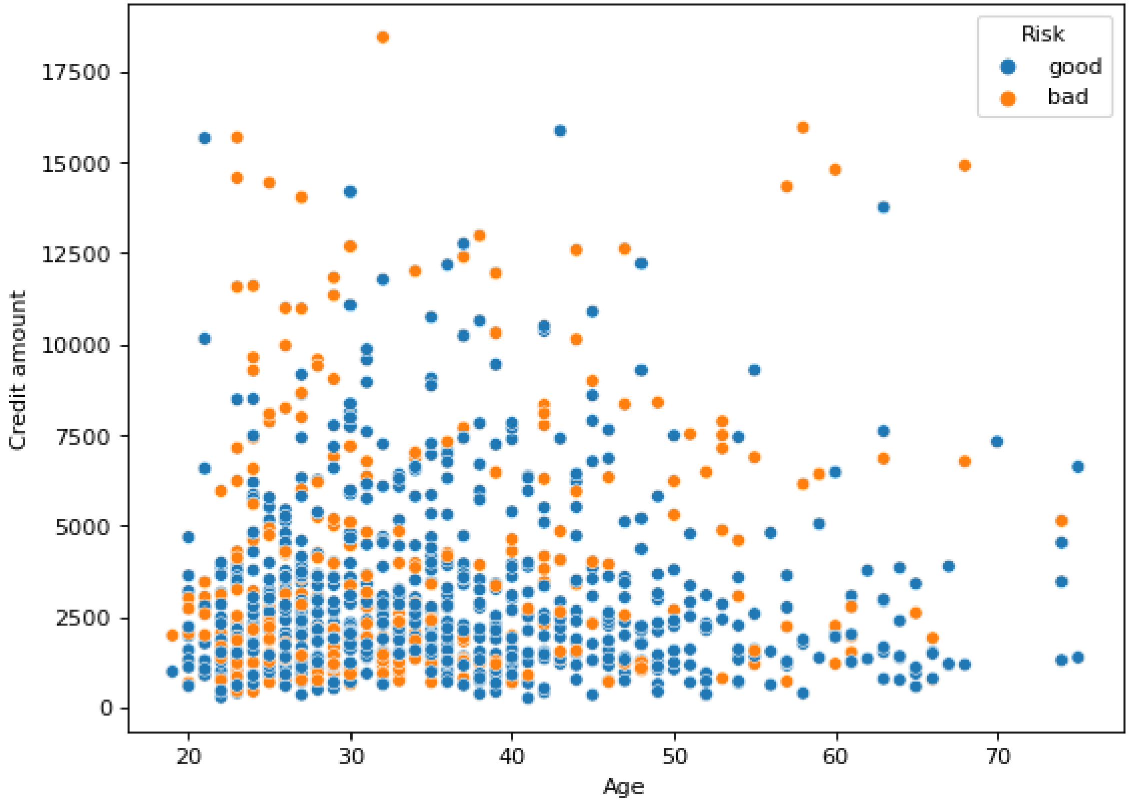


In [20]:

```
plt.figure(figsize=(8,6),dpi=80)
sns.scatterplot(x="Age",y="Credit amount",hue="Risk",data=df)
```

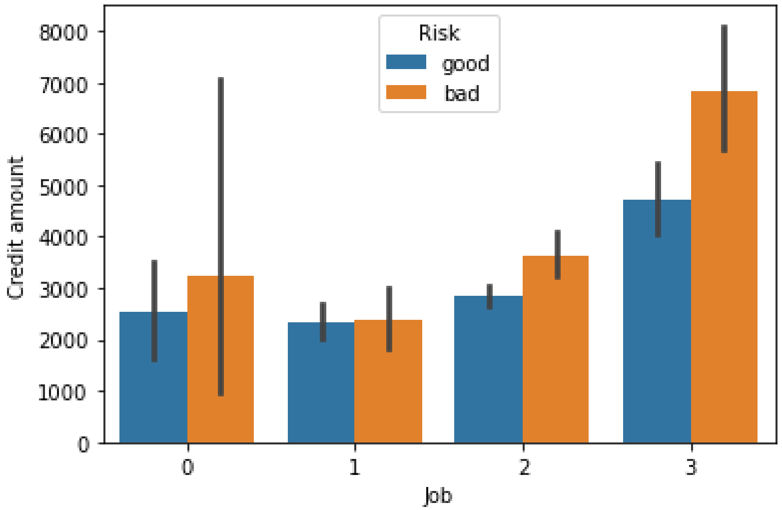
Out[20]:

```
<AxesSubplot:xlabel='Age', ylabel='Credit amount'>
```



In [21]:
```
sns.barplot(x="Job",y="Credit amount",hue="Risk",data=df)
```

Out[21]:
```
<AxesSubplot:xlabel='Job', ylabel='Credit amount'>
```



Preprocessing

In [22]:
```
df.index  = np.arange(1,len(df)+1)
```

In [23]:
```
df.drop(["Unnamed: 0","Purpose"],axis=1,inplace=True)
```

In [24]:
```
df
```

Out[24]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Risk
1	67	male	2	own	NaN	little	1169	6	good
2	22	female	2	own	little	moderate	5951	48	bad
3	49	male	1	own	little	NaN	2096	12	good
4	45	male	2	free	little	little	7882	42	good
5	53	male	2	free	little	little	4870	24	bad
...	...	...	...	...	...	...	...	...	...
996	31	female	1	own	little	NaN	1736	12	good

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Risk
997	40	male	3	own	little	little	3857	30	good
998	38	male	2	own	little	NaN	804	12	good
999	23	male	2	free	little	little	1845	45	bad
1000	27	male	2	own	moderate	moderate	4576	45	good

1000 rows × 9 columns

In [25]:
```
df["Job"].unique()
```

Out[25]:
```
array([2, 1, 3, 0], dtype=int64)
```

In [26]:
```
df["Job"] = df["Job"].replace(0,np.nan)
```

In [27]:
```
df.dropna(subset=["Job"],inplace=True)
```

In [28]:
```
risk = []
for i in df["Duration"]:
    if i>50:
      print(df['Risk'][i],end="=")
      print(i)
```

```
bad=60
good=54
good=54
bad=60
bad=60
bad=60
bad=60
bad=60
bad=60
bad=60
bad=60
good=72
bad=60
bad=60
bad=60
bad=60
```

In [29]:
```
for i in df['Duration']:
    if i>55:
        df["Duration"]=df['Duration'].replace(i,np.nan)
```

In [30]:
```
df.dropna(subset=["Duration"],inplace=True)
```

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 964 entries, 1 to 1000
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Age               964 non-null    int64  
 1   Sex               964 non-null    object 
 2   Job               964 non-null    float64
 3   Housing           964 non-null    object 
 4   Saving accounts   790 non-null    object 
 5   Checking account  578 non-null    object 
 6   Credit amount     964 non-null    int64  
 7   Duration          964 non-null    float64
 8   Risk              964 non-null    object 
dtypes: float64(2), int64(2), object(5)
memory usage: 75.3+ KB
```

In [32]:
```
df["Saving accounts"].unique()
```

Out[32]:
```
array([nan, 'little', 'quite rich', 'rich', 'moderate'], dtype=object)
```

In [33]:
```
from sklearn.preprocessing import LabelEncoder
```

In [34]:
```
label_encoder = LabelEncoder()
```

In [35]:
```
df["Housing"] = label_encoder.fit_transform(df["Housing"])
```

In [36]:
```
df = pd.get_dummies(df,columns=["Saving accounts"])
```

In [37]:
```
df = pd.get_dummies(df,columns=["Checking account"])
```

In [38]:
```
df["Sex"] = label_encoder.fit_transform(df["Sex"])
```

In [39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 964 entries, 1 to 1000
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Age                           964 non-null    int64  
 1   Sex                           964 non-null    int32  
 2   Job                           964 non-null    float64
 3   Housing                       964 non-null    int32  
 4   Credit amount                 964 non-null    int64  
 5   Duration                      964 non-null    float64
 6   Risk                          964 non-null    object 
 7   Saving accounts_little        964 non-null    uint8  
 8   Saving accounts_moderate      964 non-null    uint8  
 9   Saving accounts_quite rich    964 non-null    uint8  
 10  Saving accounts_rich          964 non-null    uint8  
 11  Checking account_little       964 non-null    uint8  
 12  Checking account_moderate     964 non-null    uint8  
 13  Checking account_rich         964 non-null    uint8  
dtypes: float64(2), int32(2), int64(2), object(1), uint8(7)
memory usage: 59.3+ KB
```

In [40]:
```
df["Age"].unique()
```

Out[40]:
```
array([67, 22, 49, 45, 53, 35, 61, 28, 25, 24, 60, 32, 44, 31, 48, 26, 36,
       39, 42, 34, 27, 30, 57, 33, 37, 58, 29, 52, 23, 50, 46, 51, 41, 40,
       66, 47, 56, 54, 20, 63, 38, 70, 65, 74, 21, 43, 55, 64, 75, 19, 62,
       59, 68], dtype=int64)
```

In [41]:
```
df.head()
```

Out[41]:

	Age	Sex	Job	Housing	Credit amount	Duration	Risk	Saving accounts_little	Saving accounts_moderate	Saving accounts_quite rich	Saving accounts_rich	Checking account_little	Ch account_mo
1	67	1	2.0	1	1169	6.0	good	0	0	0	0	1	
2	22	0	2.0	1	5951	48.0	bad	1	0	0	0	0	
3	49	1	1.0	1	2096	12.0	good	1	0	0	0	0	

	Age	Sex	Job	Housing	Credit amount	Duration	Risk	Saving accounts_little	Saving accounts_moderate	Saving accounts_quite rich	Saving accounts_rich	Checking account_little	Ch account_mo
4	45	1	2.0	0	7882	42.0	good	1	0	0	0	1	
5	53	1	2.0	0	4870	24.0	bad	1	0	0	0	1	

Feature Selection

In [42]:
```
X = df.drop("Risk",axis=1)
```

In [43]:
```
y=df["Risk"]
```

In [44]:
```
X.head()
```

Out[44]:

	Age	Sex	Job	Housing	Credit amount	Duration	Saving accounts_little	Saving accounts_moderate	Saving accounts_quite rich	Saving accounts_rich	Checking account_little	Checking account_moderate
1	67	1	2.0	1	1169	6.0	0	0	0	0	1	0
2	22	0	2.0	1	5951	48.0	1	0	0	0	0	1
3	49	1	1.0	1	2096	12.0	1	0	0	0	0	0
4	45	1	2.0	0	7882	42.0	1	0	0	0	1	0
5	53	1	2.0	0	4870	24.0	1	0	0	0	1	0

In [45]:
```
y.head()
```

Out[45]:
```
1     good
2      bad
3     good
4     good
5      bad
Name: Risk, dtype: object
```

Train test split

In [46]:
```
from sklearn.model_selection import train_test_split
```

In [47]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=20)
```

In [48]:
```
X_train[:5]
```

Out[48]:

	Age	Sex	Job	Housing	Credit amount	Duration	Saving accounts_little	Saving accounts_moderate	Saving accounts_quite rich	Saving accounts_rich	Checking account_little	Checking account_modera
231	26	1	2.0	1	4210	36.0	1	0	0	0	0	
635	25	0	1.0	1	1355	24.0	1	0	0	0	0	
858	28	1	2.0	0	3343	15.0	1	0	0	0	0	
94	20	1	2.0	2	3617	24.0	0	0	0	0	0	
17	53	1	2.0	1	2424	24.0	0	0	0	0	0	

In [49]:
```
y_train[:5]
```

Out[49]:
```
231      bad
635      bad
858     good
94      good
17      good
Name: Risk, dtype: object
```

scaling

In [50]:
```
from sklearn.preprocessing import MinMaxScaler,StandardScaler
```

In [51]:
```
standard = StandardScaler()
```

In [52]:
```
standard.fit(X_train)
```

Out[52]: StandardScaler()

In [53]:
```
X_train_scaled = standard.transform(X_train)
```

In [54]:
```
X_test_scaled = standard.transform(X_test)
```

In [55]:
```
min_max = MinMaxScaler()
```

In [56]:
```
min_max.fit(X_train)
```

Out[56]: MinMaxScaler()

In [57]:
```
X_train_scaled = min_max.transform(X_train)
```

In [58]:
```
X_test_scaled = min_max.transform(X_test)
```

In [59]:
```
X_train.head()
```

Out[59]:

	Age	Sex	Job	Housing	Credit amount	Duration	Saving accounts_little	Saving accounts_moderate	Saving accounts_quite rich	Saving accounts_rich	Checking account_little	Checkii account_modera
231	26	1	2.0	1	4210	36.0	1	0	0	0	0	
635	25	0	1.0	1	1355	24.0	1	0	0	0	0	
858	28	1	2.0	0	3343	15.0	1	0	0	0	0	
94	20	1	2.0	2	3617	24.0	0	0	0	0	0	
17	53	1	2.0	1	2424	24.0	0	0	0	0	0	

Logistic Regression

In [60]:
```
from sklearn.linear_model import LogisticRegression
```

In [61]:
```
model = LogisticRegression(max_iter=500)
```

In [62]:
```
model.fit(X_train,y_train)
```

Out[62]:
```
LogisticRegression(max_iter=500)
```

In [63]:
```
model_pred = model.predict(X_test)
```

In [64]:
```
#Performance Evaluation
```

In [65]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [66]:
```
print(confusion_matrix(y_test,model_pred))
```

```
[[ 25  65]
 [  8 192]]
```

In [67]:
```
print(classification_report(y_test,model_pred))
```

```
              precision    recall  f1-score   support

         bad       0.76      0.28      0.41        90
        good       0.75      0.96      0.84       200

    accuracy                           0.75       290
   macro avg       0.75      0.62      0.62       290
weighted avg       0.75      0.75      0.71       290
```

Decision Tree

In [68]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [69]:
```
dtree_model = DecisionTreeClassifier()
```

In [70]:
```
dtree_model.fit(X_train,y_train)
```

Out[70]:
```
DecisionTreeClassifier()
```

In [71]:
```
dtree_pred = dtree_model.predict(X_test)
```

In [72]:
```
print(confusion_matrix(y_test,dtree_pred))
```

```
[[ 39  51]
 [ 52 148]]
```

In [73]:
```
print(classification_report(y_test,dtree_pred))
```

```
              precision    recall  f1-score   support

         bad       0.43      0.43      0.43        90
        good       0.74      0.74      0.74       200

    accuracy                           0.64       290
   macro avg       0.59      0.59      0.59       290
weighted avg       0.65      0.64      0.65       290
```

Random Forest

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [75]:
```
rfc_model = RandomForestClassifier()
```

In [76]:
```
rfc_model.fit(X_train,y_train)
```

Out[76]:
```
RandomForestClassifier()
```

In [77]:
```
rfc_pred = rfc_model.predict(X_test)
```

In [78]:
```
print(confusion_matrix(y_test,rfc_pred))
```

```
[[ 31  59]
 [ 21 179]]
```

In [79]:
```
print(classification_report(y_test,rfc_pred))
```

```
              precision    recall  f1-score   support
```

```
         bad       0.60      0.34      0.44        90
        good       0.75      0.90      0.82       200

    accuracy                           0.72       290
   macro avg       0.67      0.62      0.63       290
weighted avg       0.70      0.72      0.70       290
```

In [80]:
```
from sklearn.metrics import accuracy_score
```

In [81]:
```
print(accuracy_score(y_test,rfc_pred))
```

```
0.7241379310344828
```

K Neareast Neighbours

In [82]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [83]:
```
knn_model = KNeighborsClassifier()
```

In [84]:
```
knn_model.fit(X_train,y_train)
```

Out[84]:
```
KNeighborsClassifier()
```

In [85]:
```
knn_pred = knn_model.predict(X_test)
```

In [86]:
```
print(confusion_matrix(y_test,knn_pred))
```

```
[[ 18  72]
 [ 26 174]]
```

In [87]:
```
print(classification_report(y_test,knn_pred))
```

```
              precision    recall  f1-score   support

         bad       0.41      0.20      0.27        90
        good       0.71      0.87      0.78       200

    accuracy                           0.66       290
```

```
   macro avg       0.56      0.54      0.52       290
weighted avg       0.61      0.66      0.62       290
```

In [88]:

```python
print(list(knn_pred != y_test))
```

```
[True, False, False, False, True, False, True, True, False, False, True, True, False, False, True, False, True, True,
False, False, False, False, True, False, False, False, False, False, True, True, False, False, False, False, True, Tr
ue, True, False, True, False, False, False, True, True, False, False, True, False, True, False, False, False, False,
False, False, False, False, True, False, True, False, False, False, False, True, True, True, False, False, True, Fals
e, True, True, False, False, False, False, True, False, False, True, True, False, False, False, False, False, False,
False, False, False, True, False, True, True, False, True, True, True, False, False, False, True, False, False, Fals
e, False, False, False, True, False, False, False, False, False, True, False, False, False, False, True, False, Fals
e, False, True, False, False, False, False, False, False, False, False, False, True, True, False, False, True, False,
False, True, False, True, False, False, True, False, True, False, False, False, True, False, True, False, False, Fals
e, False, False, False, True, False, False, False, False, False, True, True, True, False, False, True, False, False,
False, True, False, True, False, True, True, False, False, True, False, False, True, True, False, False, True, True,
False, False, False, True, True, True, False, False, False, True, True, True, True, False, True, False, False, False,
True, False, False, True, True, False, False, False, True, True, True, False, True, False, False, False, False, Fals
e, False, False, False, False, False, False, True, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, True, False, False, False, False, True, True, False, True, False, True, False, True, False,
False, False, False, True, False, False, True, True, False, True, True, False, False, True, True, True, False, False,
False, True, True, False, False, False, True, False, False]
```

In [89]:

```python
np.mean(knn_pred != y_test)
```

Out[89]:

```
0.33793103448275863
```

In [90]:

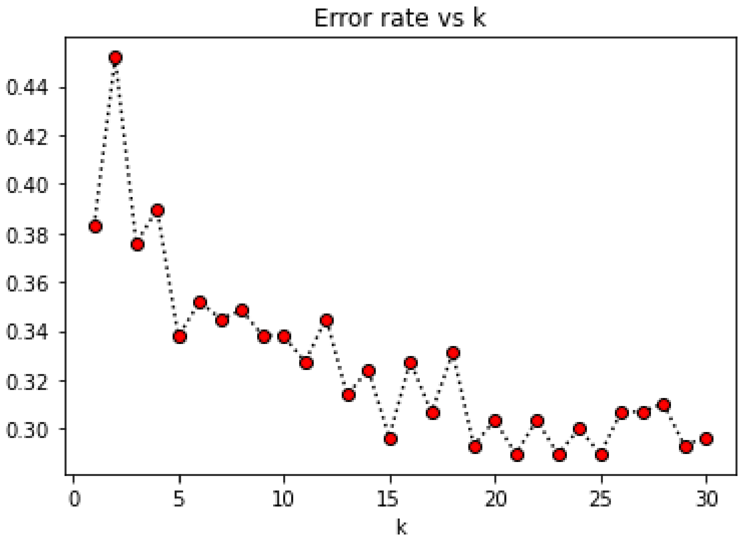
```python
error = []
for i in range(1,31):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train,y_train)
    pred = model.predict(X_test)
    error.append(np.mean(pred!=y_test))
```

In [91]:

```python
plt.plot(range(1,31),error,color='k',ls=':',marker='o',markerfacecolor='r')
plt.title("Error rate vs k")
plt.xlabel('k')
```

Out[91]:

```
Text(0.5, 0, 'k')
```



In [92]:
```
knn_model = KNeighborsClassifier(n_neighbors=15)
```

In [93]:
```
knn_model.fit(X_train,y_train)
```

Out[93]:
```
KNeighborsClassifier(n_neighbors=15)
```

In [94]:
```
knn_pred = knn_model.predict(X_test)
```

In [95]:
```
print(confusion_matrix(y_test,knn_pred))
```

```
[[ 11  79]
 [  7 193]]
```

In [96]:
```
print(classification_report(y_test,knn_pred))
```

```
              precision    recall  f1-score   support

         bad       0.61      0.12      0.20        90
        good       0.71      0.96      0.82       200

    accuracy                           0.70       290
   macro avg       0.66      0.54      0.51       290
```

```
weighted avg       0.68      0.70      0.63       290
```

Support Vector Classifier

In [97]:
```
from sklearn.svm import SVC
```

In [98]:
```
svc_model = SVC()
```

In [99]:
```
svc_model.fit(X_train,y_train)
```

Out[99]:
```
SVC()
```

In [100…
```
svc_pred = svc_model.predict(X_test)
```

In [101…
```
print(confusion_matrix(y_test,svc_pred))
```

```
[[  6  84]
 [  1 199]]
```

In [129…
```
print(classification_report(y_test,svc_pred))
```

```
              precision    recall  f1-score   support

         bad       0.86      0.07      0.12        90
        good       0.70      0.99      0.82       200

    accuracy                           0.71       290
   macro avg       0.78      0.53      0.47       290
weighted avg       0.75      0.71      0.61       290
```

Final Model

In [136…
```
final_model = LogisticRegression(max_iter=500)
```

In [137…
```
final_model.fit(X,y)
```

Out[137… LogisticRegression(max_iter=500)

In [138…
```
final_pred = final_model.predict(X)
```

In [139…
```
print(confusion_matrix(y,final_pred))
```

```
[[109 177]
 [ 75 603]]
```

In [140…
```
print(classification_report(y,final_pred))
```

```
              precision    recall  f1-score   support

         bad       0.59      0.38      0.46       286
        good       0.77      0.89      0.83       678

    accuracy                           0.74       964
   macro avg       0.68      0.64      0.65       964
weighted avg       0.72      0.74      0.72       964
```

In [ ]:
```

```

In [ ]:
```

```

Bibliography

The contents have been gathered from:

1. Google search(http://www.google.com/)
2. TensorFlow(http://www.tensorflow.org/tutorials/)
3. OpenCV(http://www.opencv.org/courses)
4. Kaggle(http://kaggle.com/uciml/german-credit)
5. Youtube Tutorials(http://m.youtube.com)