6th SEMESTER TRAINING PROJECT

-ANSHUL GUPTA (UE178022) IT SECTION-1 (2017-2021)

The dataset used here is originally "Statlog (German Credit Data) Data Set" from UCI Machine learning repository

since the original source contains data in a format that no one can understand , the dataset is imported here from Kaggle website and is in a format which can be understood (not clean or normalised) , only understandable !

- **About the data- This dataset classifies people described by a set of attributes as good or bad credit risks.**
- **AIM - to make a model using the dataset which classifies new entries as good risk or bad risk and thus helping to identify which people should be given credit or not**

In [1]:

```
import pandas as pd
import numpy as np
df_raw1=pd.read_csv(r'C:\Users\Anshul\Desktop\6th sem training\german_credit_data (1).csv')
df_raw1.head(10)
```

Out[1]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Pur
0	0	67	male	2	own	NaN	little	1169	6	rac
1	1	22	female	2	own	little	moderate	5951	48	rac
2	2	49	male	1	own	little	NaN	2096	12	edu
3	3	45	male	2	free	little	little	7882	42	furniture/equip
4	4	53	male	2	free	little	little	4870	24	
5	5	35	male	1	free	NaN	NaN	9055	36	edu
6	6	53	male	2	own	quite rich	NaN	2835	24	furniture/equip
7	7	35	male	3	rent	little	moderate	6948	36	
8	8	61	male	1	own	rich	NaN	3059	12	rac
9	9	28	male	3	own	little	moderate	5234	30	

This is the description of the attributes as provided by the source :-

1. Age (numeric)
2. Sex (text: male, female)

3. Job (numeric: 0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled)
4. Housing (text: own, rent, or free)
5. Saving accounts (text - little, moderate, quite rich, rich)
6. Checking account (text - little, moderate, rich, quite rich)
7. Credit amount (numeric, in DM)
8. Duration (numeric, in month)
9. Purpose (text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others)

IMPORTING THE LIBRARIES

In [2]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
print('done')
```

```
done
```

EXPLORING THE DATASET

In [3]:

```
df_raw1.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 [4]:

```
df_raw1.nunique()
```

Out[4]:

```
Unnamed: 0          1000
Age                   53
Sex                    2
Job                    4
Housing                3
Saving accounts        4
Checking account       3
Credit amount        921
Duration              33
Purpose                8
Risk                   2
dtype: int64
```

In [5]:

```
df_raw1.head(10)
```

Out[5]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Pur
0	0	67	male	2	own	NaN	little	1169	6	rac
1	1	22	female	2	own	little	moderate	5951	48	rac
2	2	49	male	1	own	little	NaN	2096	12	edu
3	3	45	male	2	free	little	little	7882	42	furniture/equip
4	4	53	male	2	free	little	little	4870	24	
5	5	35	male	1	free	NaN	NaN	9055	36	edu
6	6	53	male	2	own	quite rich	NaN	2835	24	furniture/equip
7	7	35	male	3	rent	little	moderate	6948	36	
8	8	61	male	1	own	rich	NaN	3059	12	rac
9	9	28	male	3	own	little	moderate	5234	30	

In [6]:

```
df_raw1.drop(columns=['Unnamed: 0'],inplace=True)
print('Done')
```

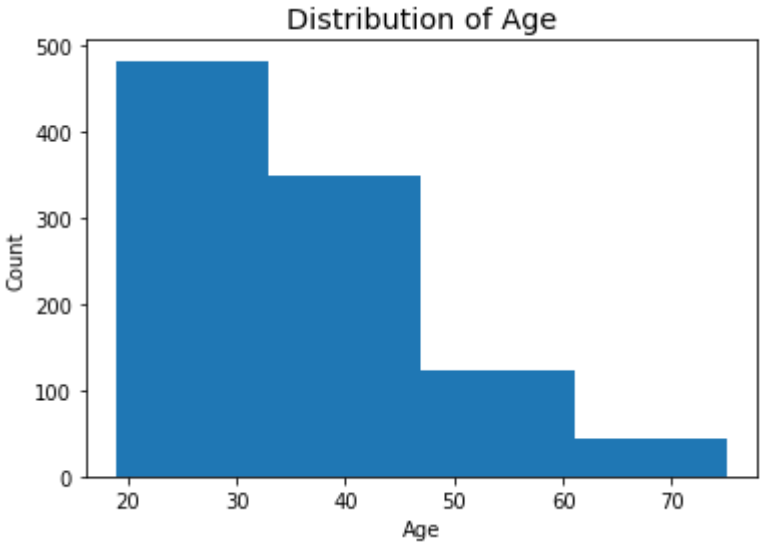
```
Done
```

In [7]:

```
plt.hist(df_raw1.Age,bins=4,label='Age distribution')
plt.ylabel('Count')
plt.xlabel('Age')
sns.set_context("paper", font_scale=1.5)
plt.title('Distribution of Age')
```

Out[7]:

```
Text(0.5, 1.0, 'Distribution of Age')
```



We will define the age groups as follows :-

- 18-30 years = student
- 30-45 years = young adults
- 45-60 years = mature adults
- 60+ years = senior citizen

In [8]:

```
def age_cat(x):
    if x>=18 and x<30:
        return str("student")
    if x>=30 and x<45:
        return str("young adult")
    if x>=45 and x<60:
        return str("mature adult")
    else:
        return str("senior citizen")
```

In [9]:

```
df_raw1['Age']=df_raw1['Age'].apply(age_cat)
```

VALUES FOR ATTRIBUTE 'JOB' ARE NOT CONVENIENT TO READ IN DATAFRAME AS THEY ARE IN NUMBERS Thue changing them into simple understandable language

Attribute description=>

- 0 - unskilled + non-resident
- 1 - unskilled + resident
- 2 - skilled
- 3 - highly skilled

In [10]:

```
def Job_cat(x):
    if x==0:
        return str(" unskilled + non-resident")
    if x==1:
        return str("unskilled + resident")
    if x==2:
        return str("skilled")
    if x==3:
        return str("highly skilled")
```

In [11]:

```
df_raw1['Job']=df_raw1['Job'].apply(Job_cat)
```

In [12]:

```
df_raw1.head(10)
```

Out[12]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
0	senior citizen	male	skilled	own	NaN	little	1169	6	radio/TV
1	student	female	skilled	own	little	moderate	5951	48	radio/TV
2	mature adult	male	unskilled + resident	own	little	NaN	2096	12	education
3	mature adult	male	skilled	free	little	little	7882	42	furniture/equipment
4	mature adult	male	skilled	free	little	little	4870	24	car
5	young adult	male	unskilled + resident	free	NaN	NaN	9055	36	education
6	mature adult	male	skilled	own	quite rich	NaN	2835	24	furniture/equipment
7	young adult	male	highly skilled	rent	little	moderate	6948	36	car
8	senior citizen	male	unskilled + resident	own	rich	NaN	3059	12	radio/TV
9	student	male	highly skilled	own	little	moderate	5234	30	car

In [13]:

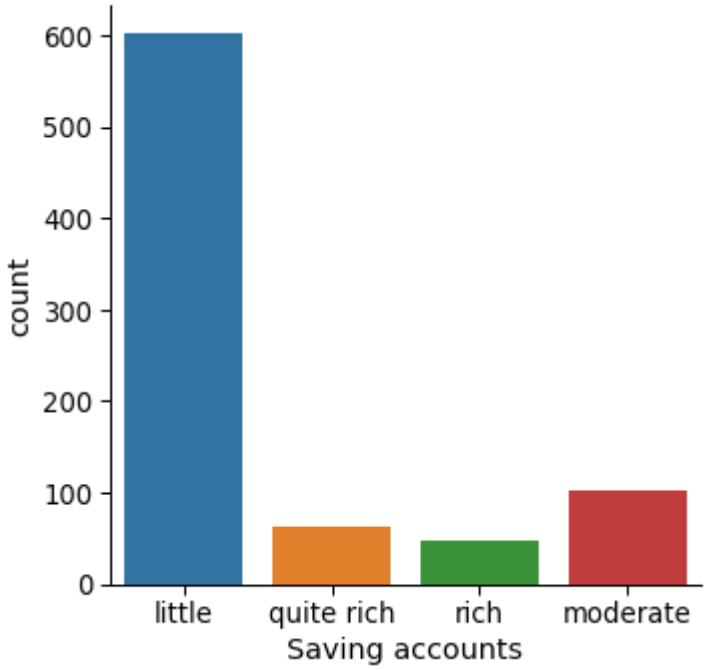
```
df_raw1['Housing'].unique()
```

Out[13]:

```
array(['own', 'free', 'rent'], dtype=object)
```

In [14]:

```
sns.catplot(x='Saving accounts',kind='count',data=df_raw1)
sns.set_context("paper", font_scale=1.5)
```



Its quite evident that most datapoints are having little amount in their savings account and this attribute is very crucial to determine who will be able to pay back credit

Also there are 'Null' values in this attributes

We replace the values by a simple rule

- peope who are bad risk are assumed at very best guess that they must NOT be rich or moderate valued in saving account attribute.
- thus labelling everyone classified as 'bad' risk as having 'little' value in savings account
- Whereas its vice versa might not be true in all cases

The reason for above assumption is that one who is rich or moderate has very good probability to be able to pay back ,whereas one who is at good risk cannot be determined as rich , moderate or little savings as the ability to pay back is driven by number of factors such as credit amount,duration, INTENTION etc.

In [15]:

```
df_raw1['Saving accounts']=df_raw1['Saving accounts'].replace(np.nan,0)
```

In [16]:

```
df_raw1.head()
```

Out[16]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
0	senior citizen	male	skilled	own	0	little	1169	6	radio/TV
1	student	female	skilled	own	little	moderate	5951	48	radio/TV
2	mature adult	male	unskilled + resident	own	little	NaN	2096	12	education
3	mature adult	male	skilled	free	little	little	7882	42	furniture/equipment
4	mature adult	male	skilled	free	little	little	4870	24	car

In [17]:

```
df_raw1['Saving accounts']=df_raw1['Saving accounts'].astype(str)
```

In [18]:

```
def acc_null_handling(vector):
    x=vector[0]
    y=vector[1]
    if y=='bad' and x=='0':
        return str('little')
    if y=='good' and x=='0':
        return 0
    else:
        return x
```

In [19]:

```
df_raw1['Savings acc']=df_raw1[['Saving accounts','Risk']].apply(acc_null_handling,axis=1)
```

In [20]:

```
df_raw1['Savings acc']=df_raw1['Savings acc'].replace('0',0)
```

In [21]:

```
df_raw1[df_raw1['Savings acc']!=0].count()
```

Out[21]:

```
Age                 849
Sex                 849
Job                 849
Housing             849
Saving accounts     849
Checking account    545
Credit amount       849
Duration            849
Purpose             849
Risk                849
Savings acc         849
dtype: int64
```

EARLIER THERE USED TO BE 817 VALUES IN 'SAVING ACCOUNTS' ATTRIBUTE BUT NEW 'SAVINGS ACC' ATTRIBUTE HAS 849 VALUES

In [22]:

```
#dropping the 'savings account' column from the dataframe
df_raw1=df_raw1.drop(columns=['Saving accounts'])
df_raw1.head()
```

Out[22]:

	Age	Sex	Job	Housing	Checking account	Credit amount	Duration	Purpose	Risk	Sa
0	senior citizen	male	skilled	own	little	1169	6	radio/TV	good	
1	student	female	skilled	own	moderate	5951	48	radio/TV	bad	
2	mature adult	male	unskilled + resident	own	NaN	2096	12	education	good	
3	mature adult	male	skilled	free	little	7882	42	furniture/equipment	good	
4	mature adult	male	skilled	free	little	4870	24	car	bad	

In [23]:

```
#dropping the rows with 0 value in Savings account
df_raw1=df_raw1[df_raw1['Savings acc']!=0]
df_raw1.shape
```

Out[23]:

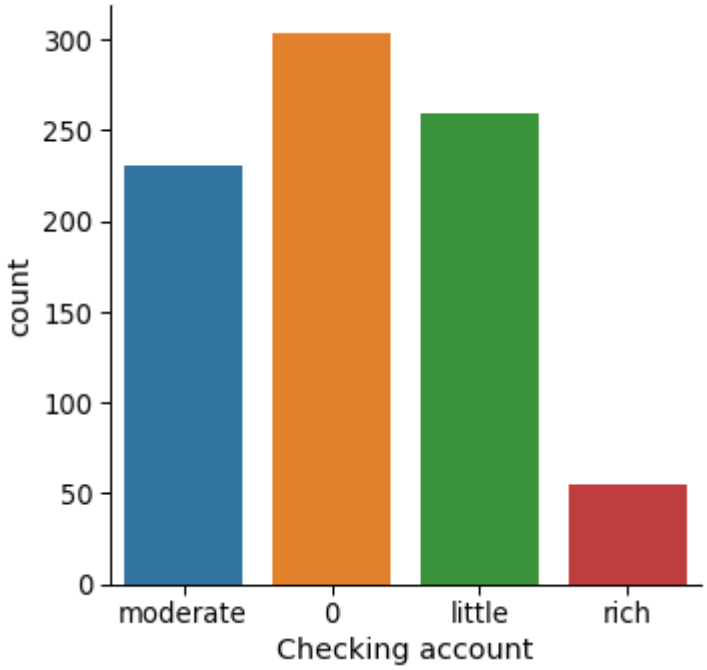
```
(849, 10)
```

In [24]:

```
df_raw1['Checking account']=df_raw1['Checking account'].replace(np.nan,0)
```

In [25]:

```
sns.catplot(x='Checking account',kind='count',data=df_raw1)
sns.set_context("paper", font_scale=1.5)
```



- Checking accounts are better for EVERYDAY TRANSACTIONS such as purchases, bills and ATM withdrawals.
- Savings accounts are better for STORING MONEY and earning interest, and because of that, you have a monthly limit on what you can withdraw

THUS IT IS QUITE REASONABLE TO SAY THAT AMOUNT IN SAVINGS ACCOUNT WILL BE GREATER THAN AMOUNT IN CHECKING ACCOUNT.

SO WE CAN SAY THAT THE STATUS OF SAVINGS ACCOUNT AND CHECKING ACCOUNT WILL BE SIMILAR IN MOST CASES.

Clearly most of the not null values are either 'little' or 'moderate' valued

Replacing null values in the 'Checking account' column by values in 'Savings acc' column

In [26]:

```
def chec_acc_null_handling(vec):
    x=vec[0]
    y=vec[1]
    if y=='little' and x==0:
        return str('little')
    if y=='moderate' and x==0:
        return str('moderate')
    if y=='rich' and x==0:
        return str('rich')
    if y=='quite rich' and x==0:
        return str('quite rich')
    else:
        return x
```

In [27]:

```
df_raw1['Checking Acc']=df_raw1[['Checking account','Savings acc']].apply(chec_acc_null_han
```

In [28]:

```
df_raw1.head(10)
```

Out[28]:

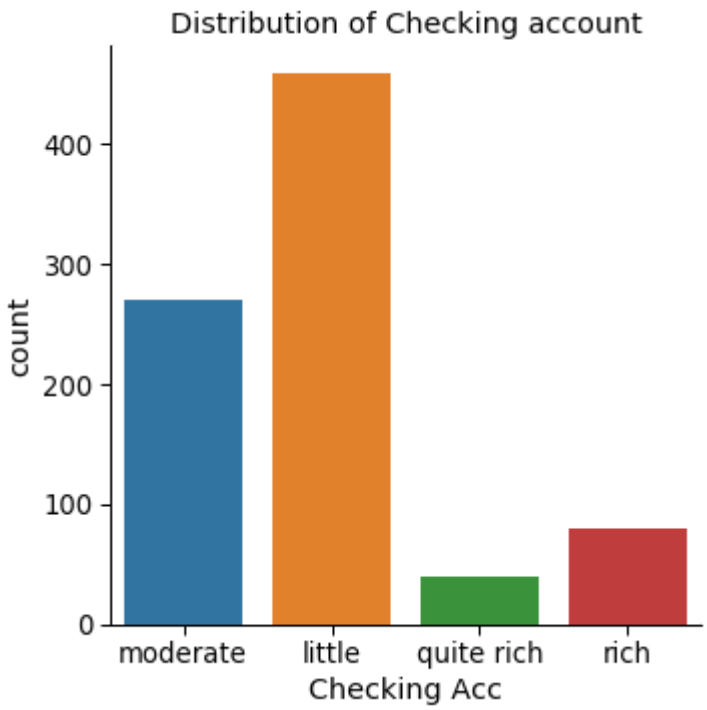
	Age	Sex	Job	Housing	Checking account	Credit amount	Duration	Purpose	Risk	S
1	student	female	skilled	own	moderate	5951	48	radio/TV	bad	
2	mature adult	male	unskilled + resident	own	0	2096	12	education	good	
3	mature adult	male	skilled	free	little	7882	42	furniture/equipment	good	
4	mature adult	male	skilled	free	little	4870	24	car	bad	
6	mature adult	male	skilled	own	0	2835	24	furniture/equipment	good	
7	young adult	male	highly skilled	rent	moderate	6948	36	car	good	
8	senior citizen	male	unskilled + resident	own	0	3059	12	radio/TV	good	
9	student	male	highly skilled	own	moderate	5234	30	car	bad	
10	student	female	skilled	rent	moderate	1295	12	car	bad	
11	student	female	skilled	rent	little	4308	48	business	bad	

In [29]:

```
sns.catplot(x='Checking Acc',kind='count',data=df_raw1)
sns.set_context("paper", font_scale=1.5)
plt.title('Distribution of Checking account')
```

Out[29]:

```
Text(0.5, 1, 'Distribution of Checking account')
```



In [30]:

```
df_raw2=df_raw1.drop(columns=['Checking account'])
df_raw2.head()
```

Out[30]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Che
1	student	female	skilled	own	5951	48	radio/TV	bad	little	mo
2	mature adult	male	unskilled + resident	own	2096	12	education	good	little	
3	mature adult	male	skilled	free	7882	42	furniture/equipment	good	little	
4	mature adult	male	skilled	free	4870	24	car	bad	little	
6	mature adult	male	skilled	own	2835	24	furniture/equipment	good	quite rich	qu

In [31]:

```
df_raw2[df_raw2['Checking Acc']==0].count()
```

Out[31]:

```
Age              0
Sex              0
Job              0
Housing          0
Credit amount    0
Duration         0
Purpose          0
Risk             0
Savings acc      0
Checking Acc     0
dtype: int64
```

In [32]:

```
df_raw2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 849 entries, 1 to 999
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Age            849 non-null    object
 1   Sex            849 non-null    object
 2   Job            849 non-null    object
 3   Housing        849 non-null    object
 4   Credit amount  849 non-null    int64 
 5   Duration       849 non-null    int64 
 6   Purpose        849 non-null    object
 7   Risk           849 non-null    object
 8   Savings acc    849 non-null    object
 9   Checking Acc   849 non-null    object
dtypes: int64(2), object(8)
memory usage: 113.0+ KB
```

In [33]:

```
df_raw2['Credit amount'].dtype
```

Out[33]:

```
dtype('int64')
```

In [34]:

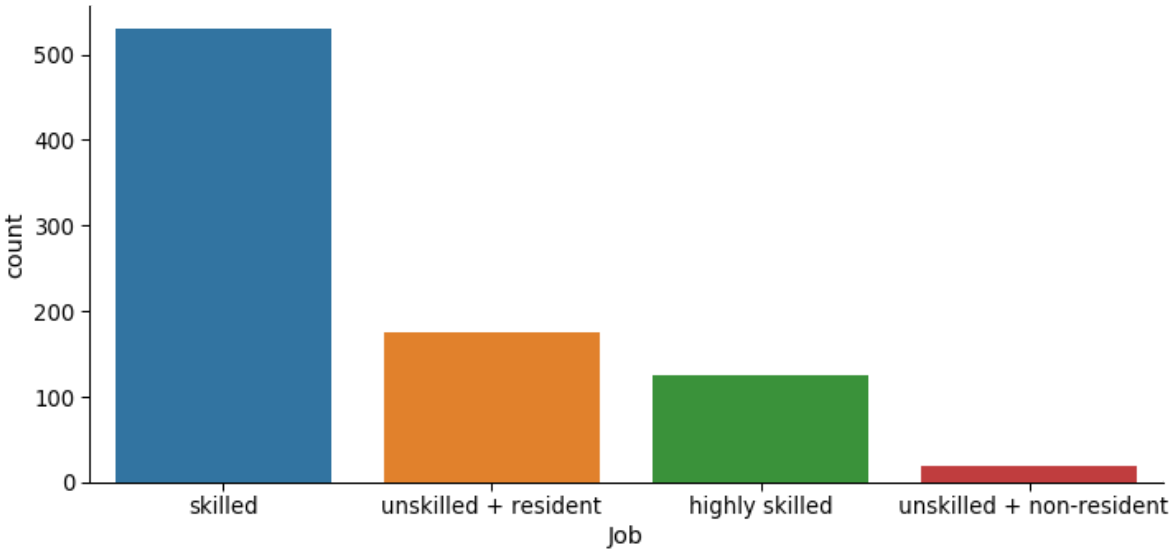
```
df_raw2['Duration'].unique()
```

Out[34]:

```
array([48, 12, 42, 24, 36, 30, 15,  9,  6, 10,  7, 60, 18, 45, 11, 27,  8,
       54, 14, 33, 21, 16,  4, 47, 13, 39, 28,  5, 20, 26, 72, 22, 40],
      dtype=int64)
```

In [35]:

```
sns.catplot(x="Job",kind="count",data=df_raw2,height=5,aspect=2.0)
sns.set_context("paper", font_scale=1.5)
```



In [36]:

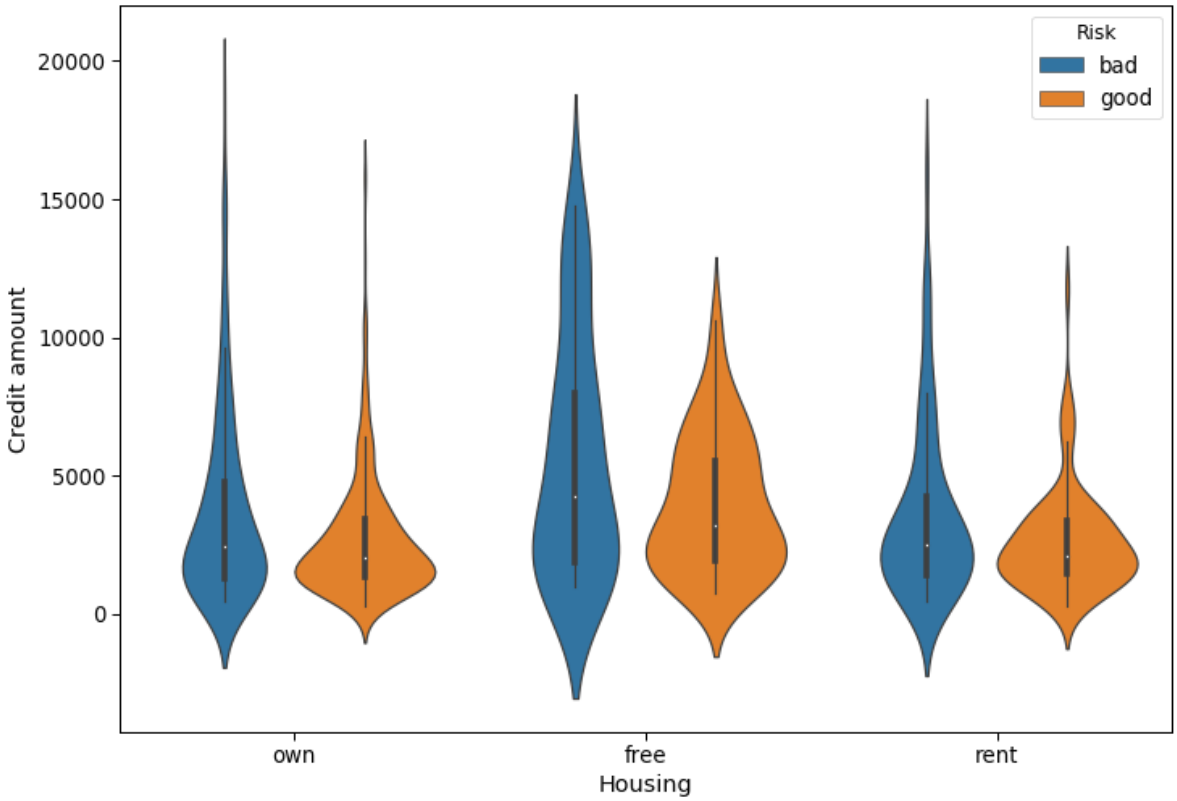
```
df_raw2.head(10)
```

Out[36]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Ch
1	student	female	skilled	own	5951	48	radio/TV	bad	little	m
2	mature adult	male	unskilled + resident	own	2096	12	education	good	little	
3	mature adult	male	skilled	free	7882	42	furniture/equipment	good	little	
4	mature adult	male	skilled	free	4870	24	car	bad	little	
6	mature adult	male	skilled	own	2835	24	furniture/equipment	good	quite rich	q
7	young adult	male	highly skilled	rent	6948	36	car	good	little	m
8	senior citizen	male	unskilled + resident	own	3059	12	radio/TV	good	rich	
9	student	male	highly skilled	own	5234	30	car	bad	little	m
10	student	female	skilled	rent	1295	12	car	bad	little	m
11	student	female	skilled	rent	4308	48	business	bad	little	

In [37]:

```
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.violinplot(x='Housing',y='Credit amount',hue='Risk',data=df_raw2,ax=ax)
sns.set_context("paper", font_scale=1.8)
```



Insights from above visualization:

- People with 'free' category in Housing attribute provides with more of good risk.Median of credit amount in case of good risk 'free' housing is higher but so is in the case of bad risk .
- HIGHEST value of credit amount with good risk is with case of people who 'own' housing.
- More credit in every housing category means more of bad risk.

OUTLIER AND ANAMOLY DETECTION

There could be following anomalies :

1. 'unskilled+non-resident' datapoints having 'quite rich' value in the savings account or checking account attribute and who own house
2. 'highly skilled' datapoints having 'little' value in the savings or checking account attribute and are living on rent.

In [38]:

```
def anom(vector):
    x=vector[0]
    y=vector[1]
    z=vector[2]
    w=vector[3]
    if x=='unskilled + non-resident' and y=='quite rich' and z=='quite rich' and w=='own':
        return 1
    if x=='highly skilled' and y=='little' and w=='rent' and (z in ('little','moderate')):
        return 1
    else:
        return 0
```

making a new column 'drop values' in which rows which have values=1 will be dropped.

In [39]:

```
df_raw2['drop values']=df_raw2[['Job','Savings acc','Checking Acc','Housing']].apply(anom,a
```

In [40]:

```
df_raw2.head()
```

Out[40]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Che
1	student	female	skilled	own	5951	48	radio/TV	bad	little	mo
2	mature adult	male	unskilled + resident	own	2096	12	education	good	little	
3	mature adult	male	skilled	free	7882	42	furniture/equipment	good	little	
4	mature adult	male	skilled	free	4870	24	car	bad	little	
6	mature adult	male	skilled	own	2835	24	furniture/equipment	good	quite rich	qu

In [41]:

```
df_raw2[df_raw2['drop values']==1]
```

Out[41]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Che
7	young adult	male	highly skilled	rent	6948	36	car	good	little	mo
121	young adult	female	highly skilled	rent	3868	24	car	good	little	
178	young adult	male	highly skilled	rent	1963	12	radio/TV	good	little	
291	student	male	highly skilled	rent	9398	36	car	bad	little	mo
335	young adult	male	highly skilled	rent	3384	6	furniture/equipment	bad	little	
462	young adult	female	highly skilled	rent	3017	12	furniture/equipment	good	little	mo
675	student	female	highly skilled	rent	4530	30	radio/TV	good	little	
736	student	female	highly skilled	rent	11560	24	car	bad	little	mo
896	student	female	highly skilled	rent	2606	21	radio/TV	good	little	
905	student	male	highly skilled	rent	1107	12	radio/TV	good	little	
919	young adult	male	highly skilled	rent	3345	24	furniture/equipment	bad	little	
937	young adult	male	highly skilled	rent	2063	6	radio/TV	good	little	mo
967	mature adult	female	highly skilled	rent	3568	15	radio/TV	good	little	
981	young adult	male	highly skilled	rent	4844	48	business	bad	little	

In [42]:

```
#dropping the anomalies
df_raw3=df_raw2[df_raw2['drop values']!=1]
df_raw3.shape
```

Out[42]:

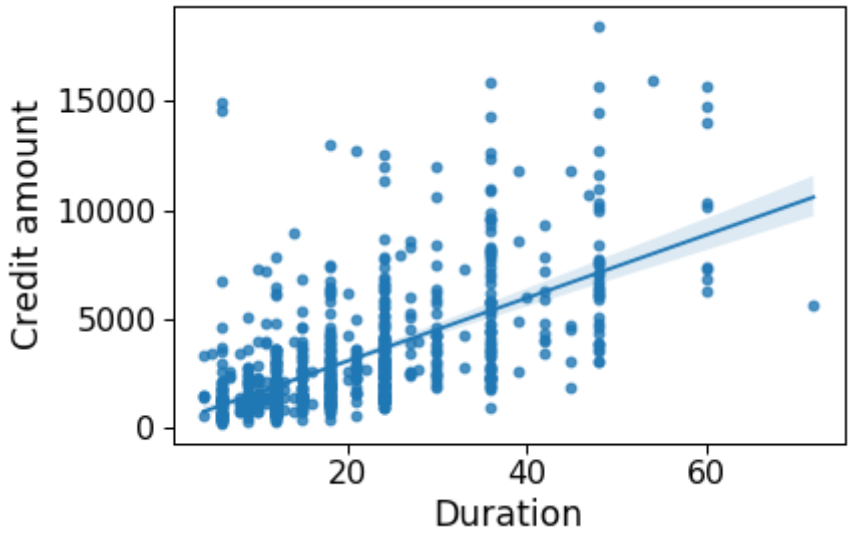
```
(835, 11)
```

In [43]:

```
sns.regplot(x='Duration',y='Credit amount',data=df_raw3,fit_reg=True)
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15dd6451708>
```



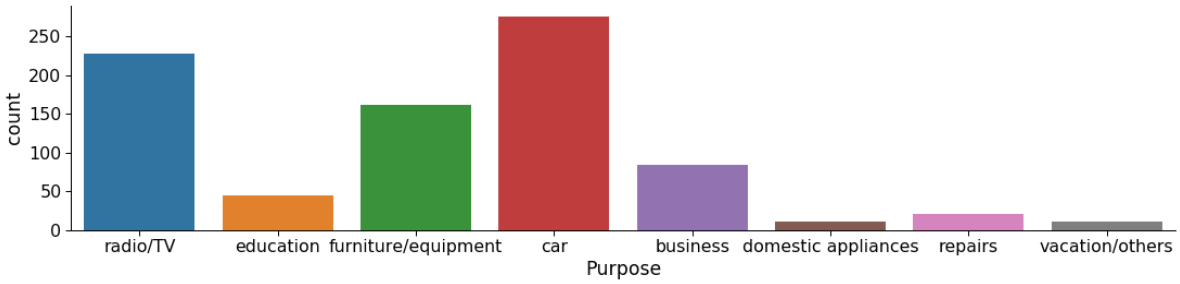
With increase in credit amount , duration of loan is also increasing.

In [44]:

```
sns.catplot(x='Purpose',kind='count',data=df_raw3,height=4,aspect=3.8)
```

Out[44]:

```
<seaborn.axisgrid.FacetGrid at 0x15dd62303c8>
```



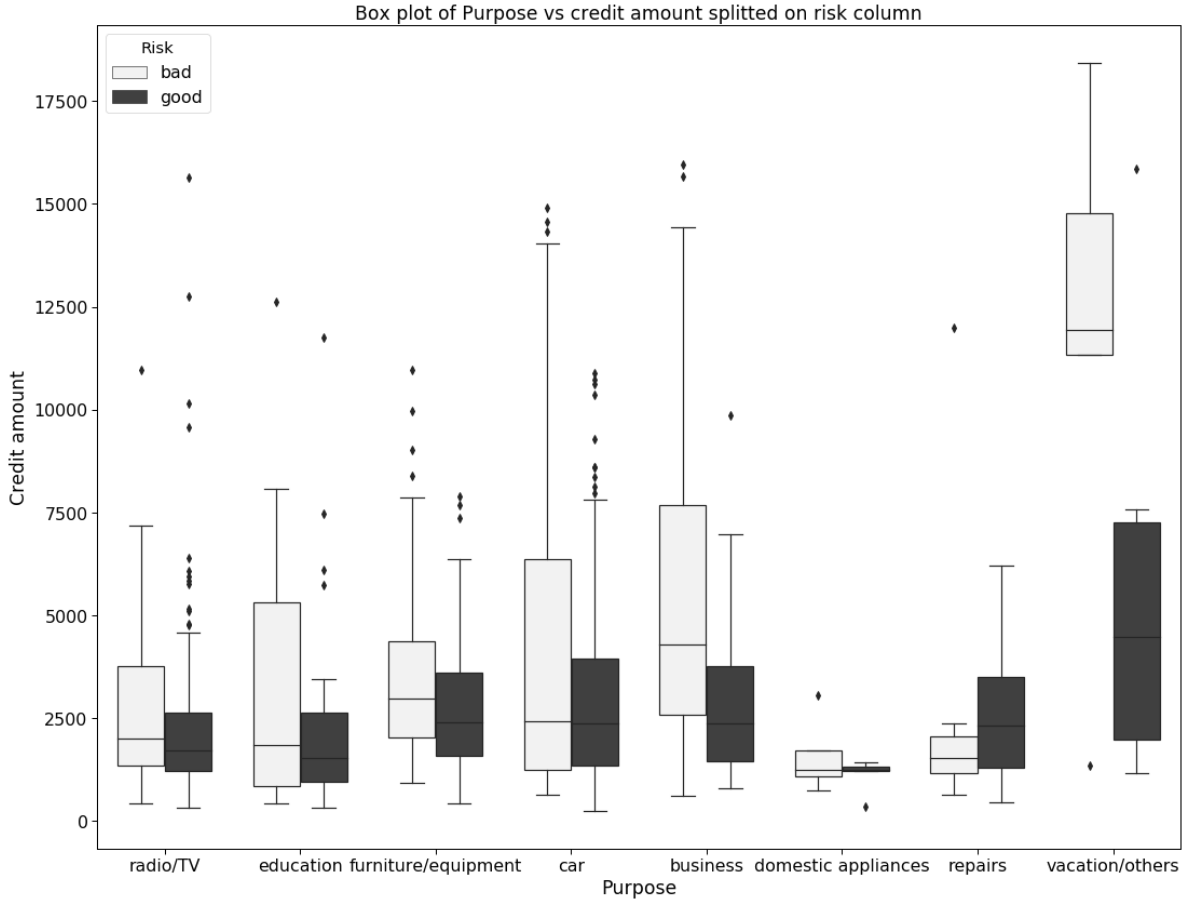
- Most people want loan for car or radio/TV
- Very few people want loan for domestic appliances and vacation.

In [45]:

```
ax1=plt.subplots(figsize=(18,14))
ax1=sns.boxplot(x='Purpose',y='Credit amount',hue='Risk',data=df_raw3, color=".25",width=0.
ax1.set_title('Box plot of Purpose vs credit amount splitted on risk column')
```

Out[45]:

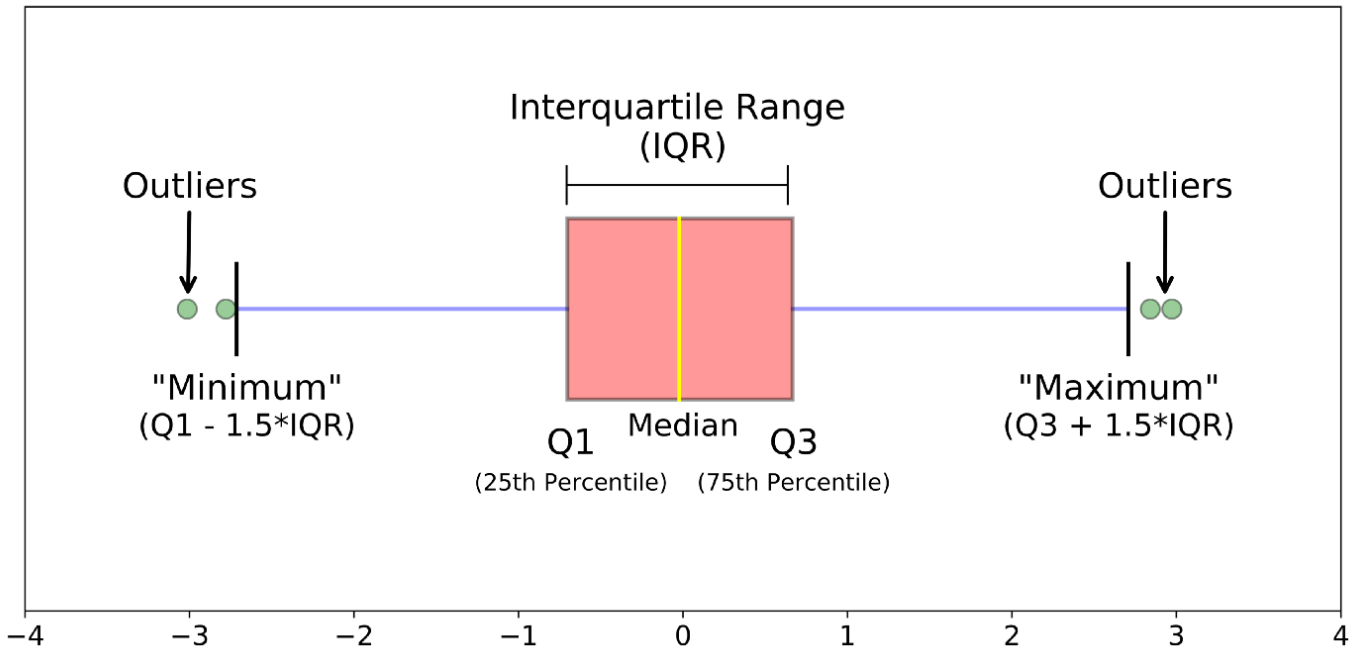
```
Text(0.5, 1.0, 'Box plot of Purpose vs credit amount splitted on risk colum
n')
```



Insights from above plot :

- **outliers** are still present and some need to be removed
- Median of credit amount is higher in case of bad risk in almost all purposes,Thus we can say that **with increase in credit amount the bad risk is more prominent** .

We will be removing outliers using the very basic property of box plot



Removing datapoints which lie beyond 'MINIMUM' and 'MAXIMUM' of boxplot

In [46]:

```
#function to remove outliers based on credit amount with respect to every category in 'Purp
def rem_outlier(df):
    q1=df['Credit amount'].quantile(0.25)
    q3=df['Credit amount'].quantile(0.75)
    iqr=q3-q1
    f1=(q1-1.5*iqr)
    f2=(q3+1.5*iqr)
    filter=(df['Credit amount'] >= f1) & (df['Credit amount'] <= f2)
    return df.loc[filter]
```

In [47]:

```
li=list(df_raw3.Purpose.unique())
li2=list(df_raw3.Risk.unique())
empdf=pd.DataFrame()
for j in li2:
    for i in li:
        dfapp1=rem_outlier(df_raw3[df_raw3['Purpose']==str(i)][df_raw3.Risk==str(j)])
        empdf=empdf.append(dfapp1)
empdf
```

```
C:\Users\Anshul\AppData\Roaming\Python\Python37\site-packages\ipykernel_laun
cher.py:6: UserWarning: Boolean Series key will be reindexed to match DataFr
ame index.
```

Out[47]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Che
1	student	female	skilled	own	5951	48	radio/TV	bad	little	moc
15	young adult	female	unskilled + resident	own	1282	24	radio/TV	bad	moderate	
35	student	male	unskilled + resident	own	4746	45	radio/TV	bad	little	moc
37	young adult	male	skilled	own	2100	18	radio/TV	bad	little	
56	mature adult	male	highly skilled	own	6468	12	radio/TV	bad	little	moc
...	...	...	...	...	...	...	...	...	...	
72	mature adult	male	highly skilled	free	1164	8	vacation/others	good	little	
83	mature adult	female	unskilled + resident	own	1755	24	vacation/others	good	little	
287	young adult	male	highly skilled	free	7582	48	vacation/others	good	moderate	moc
442	student	male	skilled	own	2629	20	vacation/others	good	little	moc
665	student	male	highly skilled	own	6314	24	vacation/others	good	little	

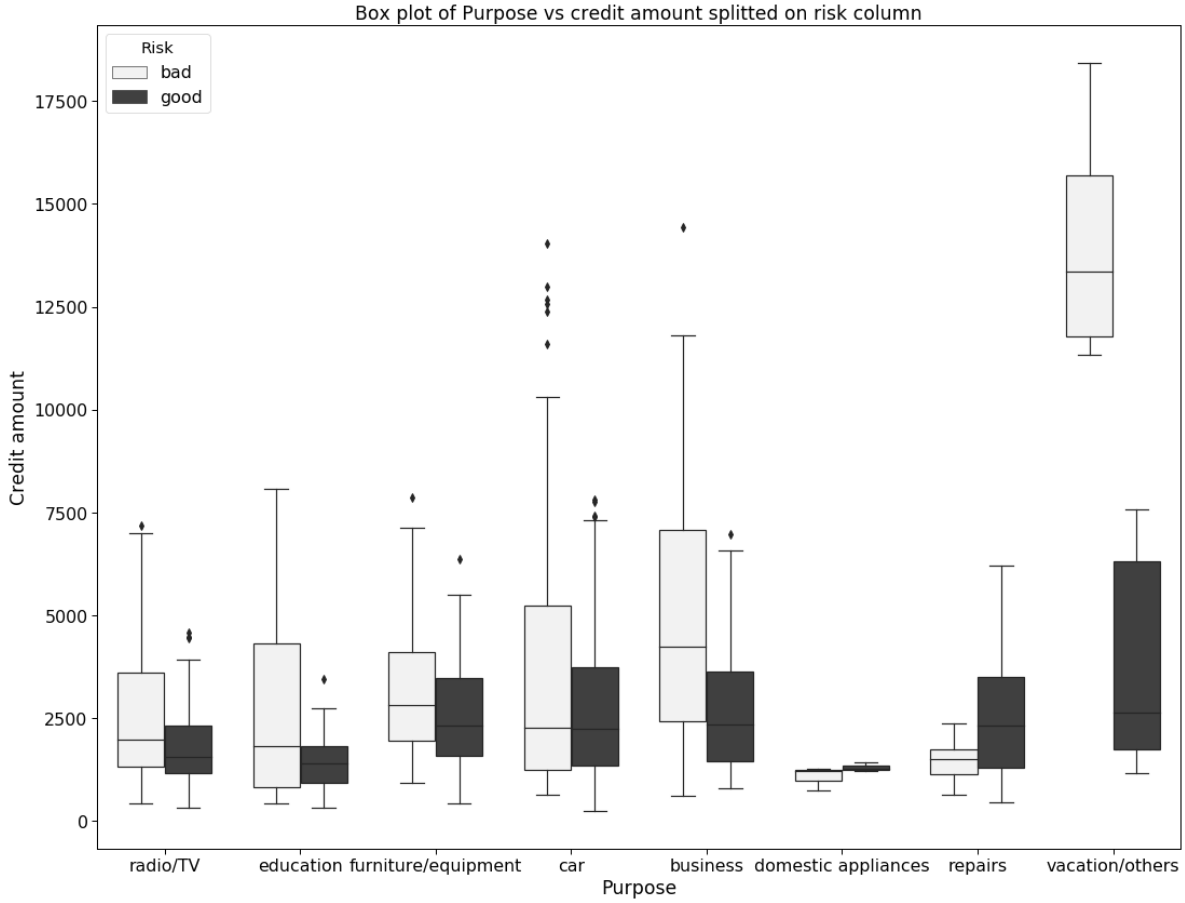
788 rows × 11 columns

In [48]:

```
ax2=plt.subplots(figsize=(18,14))
ax2=sns.boxplot(x='Purpose',y='Credit amount',hue='Risk',data=empdf, color=".25",width=0.7)
ax2.set_title('Box plot of Purpose vs credit amount splitted on risk column')
```

Out[48]:

```
Text(0.5, 1.0, 'Box plot of Purpose vs credit amount splitted on risk colum
n')
```



Number of outliers present has reduced dramatically

- **The outliers removed are not just based on a single column**
- **Outliers which are based on 'Credit amount' column per every category in 'Purpose' column with respect to 'Good' or 'Bad' risk are removed.**

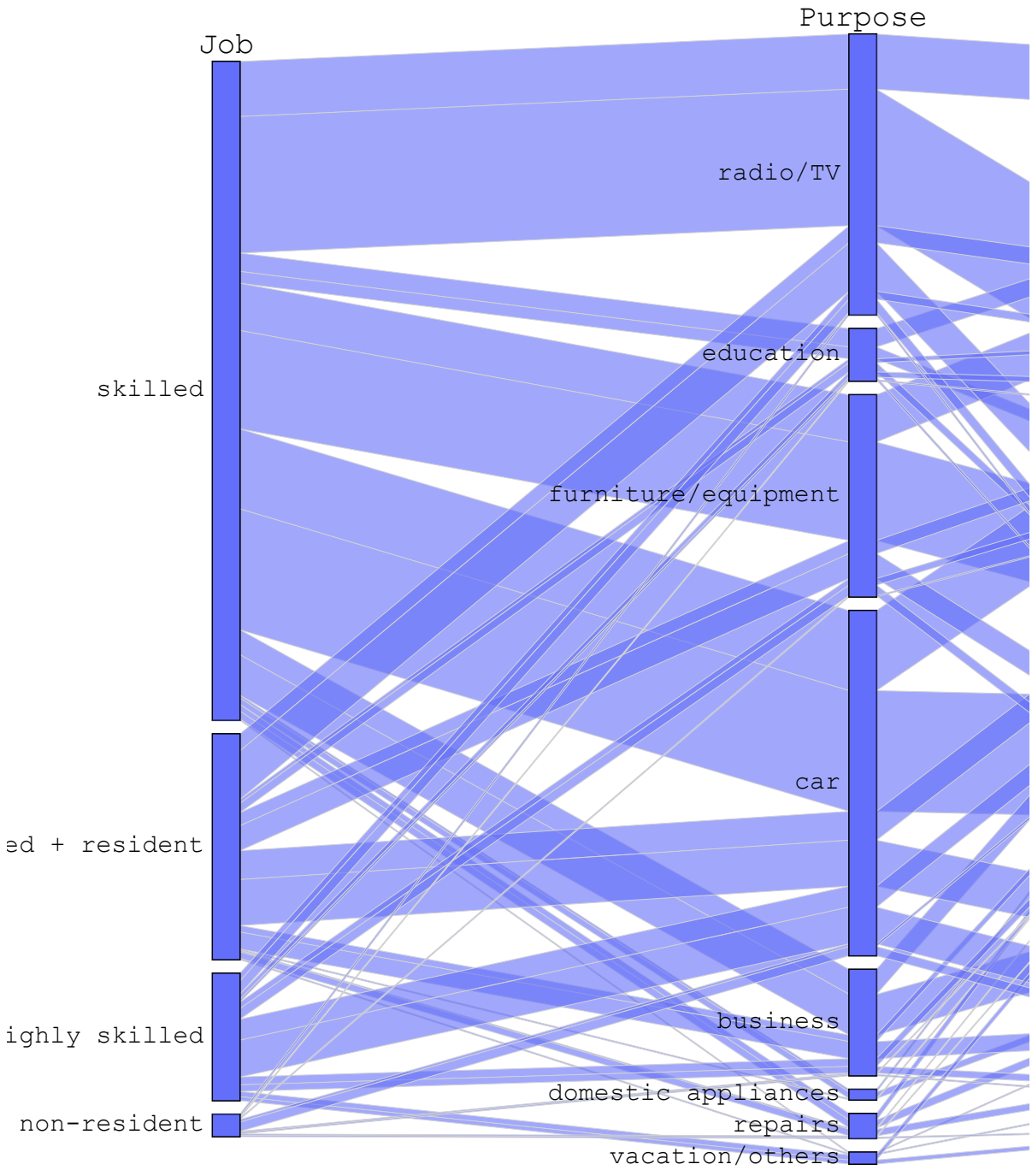
In [49]:

```
import plotly
print('done')
```

```
done
```

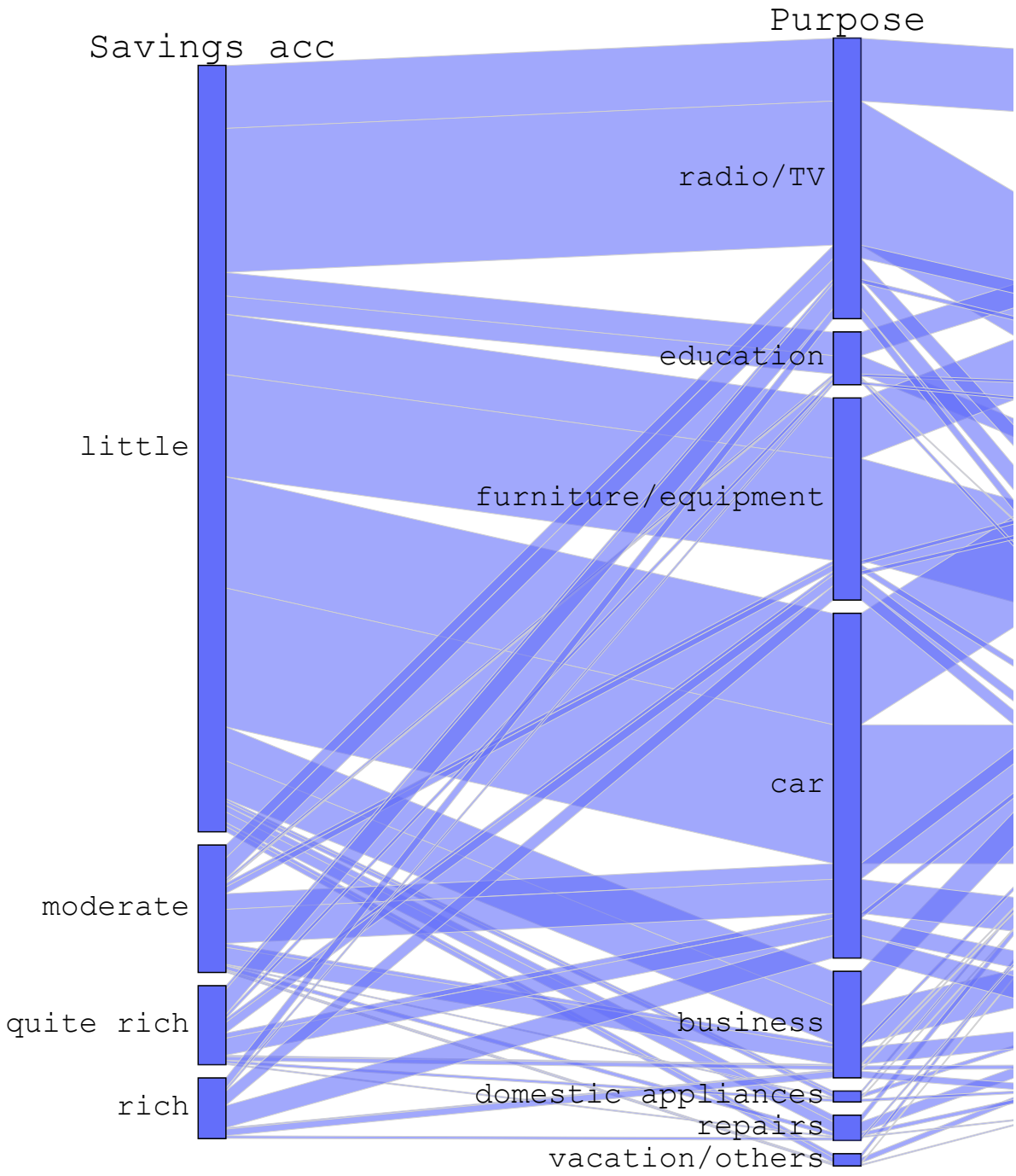
In [50]:

```
import plotly.express as px
fig=px.parallel_categories(empdf,
                           dimensions=['Job','Purpose','Risk'])
fig.update_layout(autosize=True, width=1000, height=800,font=dict(
        family="Courier New, monospace",
        size=18,
        color="Black"
    ))
fig.show()
```



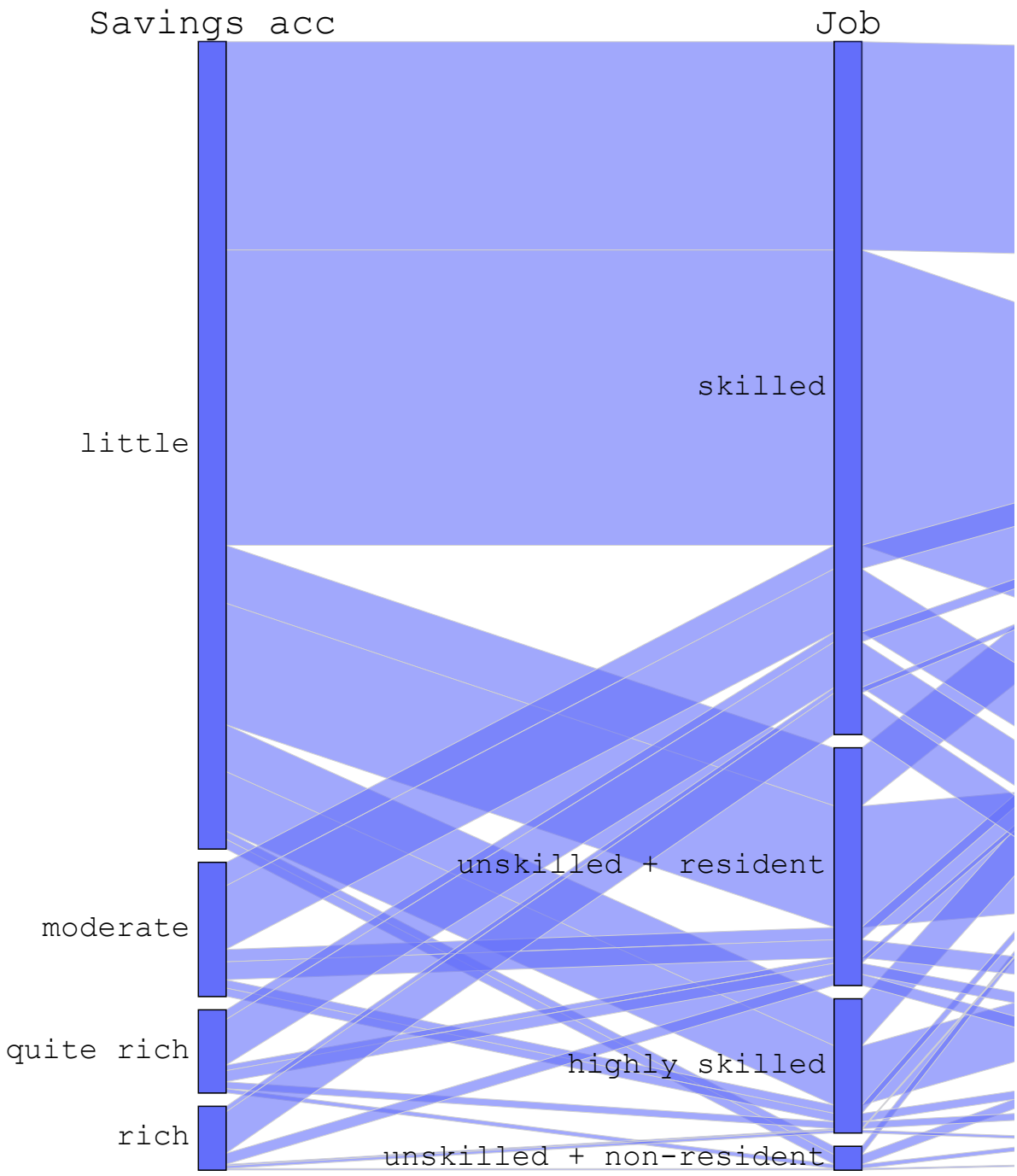
In [51]:

```
fignew=px.parallel_categories(empdf,
                          dimensions=['Savings acc','Purpose','Risk'])
fignew.update_layout(autosize=True, width=1000, height=800,font=dict(
        family="Courier New, monospace",
        size=22,
        color="Black"
    ))
fignew.show()
```



In [52]:

```
figfinal=px.parallel_categories(empdf,
                          dimensions=['Savings acc','Job','Risk'])
figfinal.update_layout(autosize=True, width=1000, height=800,font=dict(
        family="Courier New, monospace",
        size=22,
        color="Black"
    ))
figfinal.show()
```



In [53]:

```
empdf.shape
```

Out[53]:

```
(788, 11)
```

- Final Shape is 788 datapoints and total 11 columns (9 independent attributes and 1 target attribute)
- 'drop values' column is to be ignored , will drop it soon

In [54]:

```
empdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 788 entries, 1 to 665
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Age            788 non-null    object
 1   Sex            788 non-null    object
 2   Job            788 non-null    object
 3   Housing        788 non-null    object
 4   Credit amount  788 non-null    int64 
 5   Duration       788 non-null    int64 
 6   Purpose        788 non-null    object
 7   Risk           788 non-null    object
 8   Savings acc    788 non-null    object
 9   Checking Acc   788 non-null    object
 10  drop values    788 non-null    int64 
dtypes: int64(3), object(8)
memory usage: 93.9+ KB
```

In [55]:

```
empdf.nunique()
```

Out[55]:

```
Age               4
Sex               2
Job               4
Housing           3
Credit amount    731
Duration          30
Purpose           8
Risk              2
Savings acc       4
Checking Acc      4
drop values       1
dtype: int64
```

In [56]:

```
empdf.tail(20)
```

Out[56]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Risk	Savings acc	Che
207	student	male	skilled	own	1424	12	domestic appliances	good	little	moc
350	student	female	skilled	rent	1236	9	domestic appliances	good	little	
816	senior citizen	male	skilled	own	1338	6	domestic appliances	good	quite rich	quit
42	young adult	male	unskilled + resident	own	6204	18	repairs	good	little	moc
156	mature adult	male	skilled	own	1288	9	repairs	good	moderate	
225	student	male	skilled	own	2613	36	repairs	good	little	
433	young adult	male	skilled	own	2058	24	repairs	good	little	
436	student	male	unskilled + resident	rent	660	6	repairs	good	quite rich	quit
438	senior citizen	male	unskilled + non-resident	own	3394	42	repairs	good	little	
520	young adult	male	skilled	free	5507	24	repairs	good	little	
641	young adult	male	unskilled + resident	own	1308	15	repairs	good	little	moc
691	student	female	unskilled + resident	own	2631	15	repairs	good	moderate	moc
779	senior citizen	female	skilled	own	3872	18	repairs	good	little	moc
964	student	male	unskilled + resident	own	454	6	repairs	good	little	moc
970	student	male	skilled	own	1514	15	repairs	good	moderate	moc
72	mature adult	male	highly skilled	free	1164	8	vacation/others	good	little	
83	mature adult	female	unskilled + resident	own	1755	24	vacation/others	good	little	
287	young adult	male	highly skilled	free	7582	48	vacation/others	good	moderate	moc
442	student	male	skilled	own	2629	20	vacation/others	good	little	moc
665	student	male	highly skilled	own	6314	24	vacation/others	good	little	

Above is the provisionary dataframe ready to be modelled

SHAP analysis to get most impactful features

In [57]:

```
from pycaret.classification import *
```

In [58]:

```
expkk=setup(empdf,target='Risk',ignore_features=['drop values'])
```

```
Setup Succesfully Completed!
```

	Description	Value
0	session_id	8057
1	Target Type	Binary
2	Label Encoded	bad: 0, good: 1
3	Original Data	(788, 11)
4	Missing Values	False
5	Numeric Features	2
6	Categorical Features	8
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(788, 11)
11	Transformed Train Set	(551, 30)
12	Transformed Test Set	(237, 30)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	False
16	Normalize Method	None
17	Transformation	False
18	Transformation Method	None
19	PCA	False
20	PCA Method	None
21	PCA Components	None
22	Ignore Low Variance	False
23	Combine Rare Levels	False
24	Rare Level Threshold	None
25	Numeric Binning	False
26	Remove Outliers	False
27	Outliers Threshold	None
28	Remove Multicollinearity	False
29	Multicollinearity Threshold	None
30	Clustering	False
31	Clustering Iteration	None
32	Polynomial Features	False
33	Polynomial Degree	None

	Description	Value
34	Trignometry Features	False
35	Polynomial Threshold	None
36	Group Features	False
37	Feature Selection	False
38	Features Selection Threshold	None
39	Feature Interaction	False
40	Feature Ratio	False
41	Interaction Threshold	None

In [59]:

```
compare_models()
```

Out[59]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	Ada Boost Classifier	0.733300	0.764700	0.873100	0.753400	0.808500	0.376700
1	Gradient Boosting Classifier	0.726000	0.744600	0.879000	0.743500	0.804800	0.353800
2	Extreme Gradient Boosting	0.726000	0.749400	0.881600	0.742200	0.805400	0.351500
3	Ridge Classifier	0.702500	0.000000	0.864400	0.726200	0.788700	0.296400
4	CatBoost Classifier	0.702400	0.734200	0.839800	0.736300	0.783500	0.311800
5	Light Gradient Boosting Machine	0.700700	0.737000	0.814400	0.746100	0.777300	0.321300
6	Linear Discriminant Analysis	0.700600	0.720700	0.856000	0.727600	0.786000	0.296600
7	Logistic Regression	0.697000	0.730000	0.850200	0.726800	0.782500	0.289100
8	Quadratic Discriminant Analysis	0.666000	0.692900	0.844100	0.705200	0.752500	0.205500
9	Extra Trees Classifier	0.658900	0.680500	0.748900	0.731900	0.738300	0.244900
10	Random Forest Classifier	0.655300	0.688700	0.718300	0.740500	0.728400	0.255900
11	K Neighbors Classifier	0.619200	0.618200	0.766100	0.682800	0.719400	0.124700
12	Decision Tree Classifier	0.597100	0.565700	0.672900	0.692300	0.681100	0.131300
13	Naive Bayes	0.595200	0.701200	0.514800	0.781800	0.617100	0.224900
14	SVM - Linear Kernel	0.504800	0.000000	0.500000	0.325500	0.394300	-0.003700

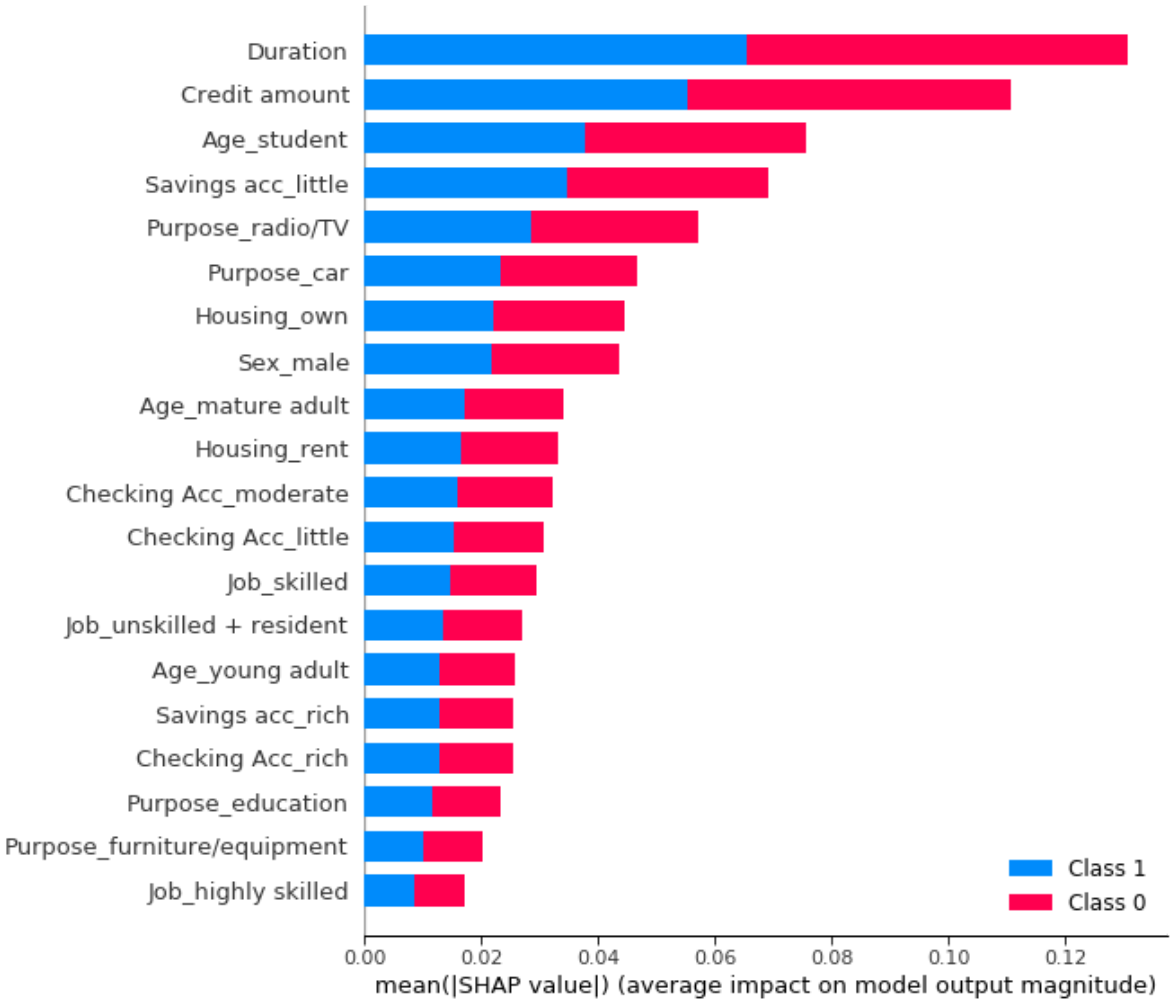
In [60]:

```
rfm=create_model('rf')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.5893	0.6264	0.6667	0.6857	0.6761	0.1154
1	0.6909	0.7149	0.7500	0.7714	0.7606	0.3249
2	0.6545	0.6623	0.7500	0.7297	0.7397	0.2265
3	0.6364	0.6469	0.6944	0.7353	0.7143	0.2154
4	0.7273	0.8085	0.7222	0.8387	0.7761	0.4322
5	0.7455	0.6979	0.8286	0.7838	0.8056	0.4380
6	0.6545	0.7021	0.6857	0.7500	0.7164	0.2768
7	0.6909	0.7457	0.7429	0.7647	0.7536	0.3392
8	0.5818	0.6679	0.6286	0.6875	0.6567	0.1246
9	0.5818	0.6143	0.7143	0.6579	0.6849	0.0664
Mean	0.6553	0.6887	0.7183	0.7405	0.7284	0.2559
SD	0.0560	0.0555	0.0521	0.0509	0.0449	0.1232

In [61]:

```
interpret_model(rfm)
```



WE WILL TRY BUILDING 2 DIFFERENT TYPES OF MACHINE LEARNING MODELS NAMELY -

- **RANDOM FOREST CLASSIFIER**
- **LOGISTIC REGRESSION CLASSIFIER**

- # Model 1 - Random Forest classifier

In [62]:

```
X=empdf.drop(['Risk','drop values'],axis='columns')
X.head()
```

Out[62]:

	Age	Sex	Job	Housing	Credit amount	Duration	Purpose	Savings acc	Checking Acc
1	student	female	skilled	own	5951	48	radio/TV	little	moderate
15	young adult	female	unskilled + resident	own	1282	24	radio/TV	moderate	little
35	student	male	unskilled + resident	own	4746	45	radio/TV	little	moderate
37	young adult	male	skilled	own	2100	18	radio/TV	little	rich
56	mature adult	male	highly skilled	own	6468	12	radio/TV	little	moderate

In [63]:

```
y=empdf.Risk
y.head()
```

Out[63]:

```
1     bad
15    bad
35    bad
37    bad
56    bad
Name: Risk, dtype: object
```

In [64]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=99)
```

In [65]:

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
```

Column transform :applies one-hot encoding on the categorical attributes and leaves the numerical attributes as they were and gives a dataframe upon which machine learning model can be made

In [66]:

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
ct=make_column_transformer((OneHotEncoder(),['Age','Sex','Job','Housing','Purpose','Savings
                    remainder='passthrough')
```

In [67]:

```
xd=ct.fit_transform(X)
```

Creating a pipeline which first applies Column Transform on the dataframe and then applies machine learning algorithm on it

In [68]:

```
from sklearn.pipeline import make_pipeline
pipe=make_pipeline(ct,rfc)
```

In [69]:

```
pipe.fit(X_train,y_train)
```

Out[69]:

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories
='auto',
                                                                drop=None,
                                                                dtype=<class
'numpy.float64'>,
                                                                handle_unkno
wn='error',
                                                                sparse=Tru
e),
                                                  ['Age', 'Sex', 'Job',
                                                   'Housing', 'Purpose',
                                                   'Savings acc',
                                                   'Checking Ac...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=Non
e,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_spli
t=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

In [70]:

```
ypred=pipe.predict(X_test)
```

In [71]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,ypred,normalize=True)
```

Out[71]:

```
0.700507614213198
```

The accuracy is around 68 percent with 75 % as training data using Random Forest Classifier

Accuracy (in case of classification) only tells us the percentage of correctly classified outcomes and does not take into account the probabilities.

Comparison with null accuracy

In [72]:

```
y_test.value_counts()
```

Out[72]:

```
good    129
bad      68
Name: Risk, dtype: int64
```

Null accuracy = 129/(129+68) = 129/197 = 0.65 approx

- This indicates that dumb model which always predicts the most occuring result will be correct 65 % of times.
- since our accuracy(67 %) is not much higher than null accuracy(65 %) , there is surely some kind of error

In [73]:

```
print('true values',y_test.values[0:25])
print('pred values',ypred[0:25])
```

```
true values ['good' 'bad' 'good' 'bad' 'bad' 'good' 'good' 'bad' 'good' 'goo
d' 'bad'
 'good' 'good' 'good' 'bad' 'good' 'good' 'bad' 'bad' 'bad' 'good' 'good'
 'good' 'good' 'good']
pred values ['good' 'bad' 'good' 'bad' 'bad' 'good' 'good' 'good' 'good' 'go
od' 'good'
 'good' 'good' 'good' 'bad' 'bad' 'bad' 'good' 'good' 'good' 'good' 'good'
 'good' 'good' 'good']
```

One thing is quite clear from above values that accuracy of predicting 'good' risk is far greater than accuracy of predicting 'bad' risk

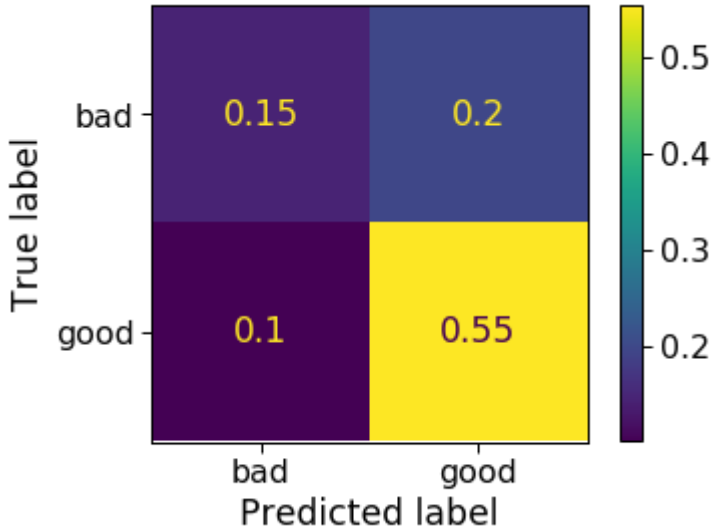
Creating a confusion matrix to better understand the metric calculation

In [74]:

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(pipe,X_test,y_test,normalize='all')
```

Out[74]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15de859e
108>
```



In [75]:

```
from sklearn.metrics import recall_score,precision_score
recall_score(y_test,ypred,average=None)
```

Out[75]:

```
array([0.42647059, 0.84496124])
```

Our model has recall of around 0.39,0.83 (bad, good) - in other words, it correctly identifies ~ 83 % of good risks and ~40% of bad risks

In [76]:

```
precision_score(y_test,ypred,average=None)
```

Out[76]:

```
array([0.59183673, 0.73648649])
```

Our model has precision of around 0.55,0.72 (bad, good) - in other words, when it predicts the risk is good , it is correct ~ 72 % of time and when it predicts the risk is bad, it is correct ~ 55 % of time

Both precision and recall of our model (random forest) are not satisfactory

- ## Tuning hyperparameters of model

In [77]:

```
from sklearn.model_selection import GridSearchCV
param_grid={
    "max_depth": [3,5, 7, 10,None],
    "n_estimators":[3,5,10,25,50,150],
    "max_features": [4,7,15,17,20]
}
clf = GridSearchCV(estimator=rfc,
                   param_grid=param_grid,
                   cv=5)

clf.fit(xd,y)
print('parameters obtained using grid search:',clf.best_params_)
```

```
parameters obtained using grid search: {'max_depth': 10, 'max_features': 15,
'n_estimators': 25}
```

We have got the best parameters for random forest classifier ,so now we'll be making another pipeline so as to compare metrics

In [78]:

```
rfc_tuned=RandomForestClassifier(n_estimators=50,max_features=10,max_depth=7)
```

In [79]:

```
# pipeline for model with parameters obtained using grid search
pipe_tuned=make_pipeline(ct,rfc_tuned)
pipe_tuned.fit(X_train,y_train)
```

Out[79]:

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories
='auto',
                                                                drop=None,
                                                                dtype=<class
'numpy.float64'>,
                                                                handle_unkno
wn='error',
                                                                sparse=Tru
e),
                                                  ['Age', 'Sex', 'Job',
                                                   'Housing', 'Purpose',
                                                   'Savings acc',
                                                   'Checking Ac...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=7, max_features=10,
                                        max_leaf_nodes=None, max_samples=Non
e,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_spli
t=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=50, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

In [80]:

```
ypred_tuned=pipe_tuned.predict(X_test)
```

In [81]:

```
print('accuracy score (grid search):',accuracy_score(y_test,ypred_tuned,normalize=True))
```

```
accuracy score (grid search): 0.700507614213198
```

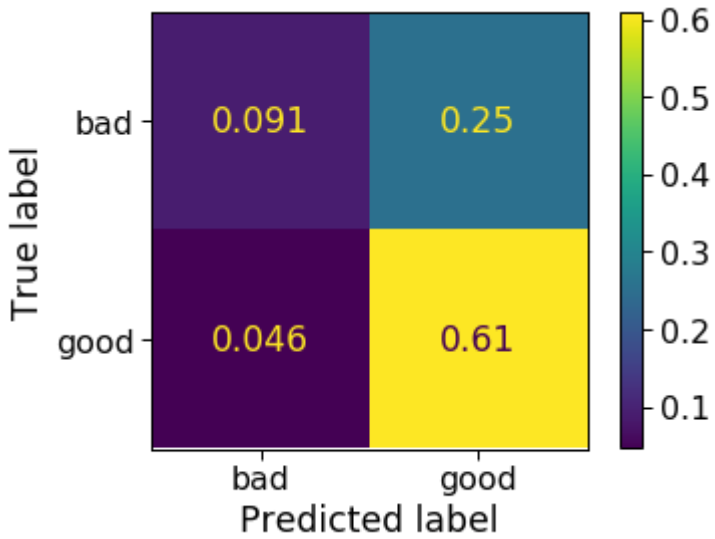
Accuracy increased from ~66 % to ~ 70 % by tuning the hyperparameters

In [82]:

```
plot_confusion_matrix(pipe_tuned,X_test,y_test,normalize='all')
```

Out[82]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15de86b9
188>
```



In [83]:

```
print('tuned recall score:',recall_score(y_test,ypred_tuned,average=None))
print('tuned precision score:',precision_score(y_test,ypred_tuned,average=None))
```

```
tuned recall score: [0.26470588 0.93023256]
tuned precision score: [0.66666667 0.70588235]
```

- ## Model 2 - Logistic Regression classifier

In [84]:

```
from sklearn.linear_model import LogisticRegression
lgr=LogisticRegression()
```

In [85]:

```
pipe2=make_pipeline(ct,lgr)
```

In [86]:

```
pipe2.fit(X_train,y_train)
```

Out[86]:

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories
='auto',
                                                                drop=None,
                                                                dtype=<class
'numpy.float64'>,
                                                                handle_unkno
wn='error',
                                                                sparse=Tru
e),
                                                  ['Age', 'Sex', 'Job',
                                                   'Housing', 'Purpose',
                                                   'Savings acc',
                                                   'Checking Acc'])],
                                   verbose=False)),
                ('logisticregression',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=None,
                                    solver='lbfgs', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

In [87]:

```
ypred2=pipe2.predict(X_test)
```

In [88]:

```
accuracy_score(y_test,ypred2,normalize=True)
```

Out[88]:

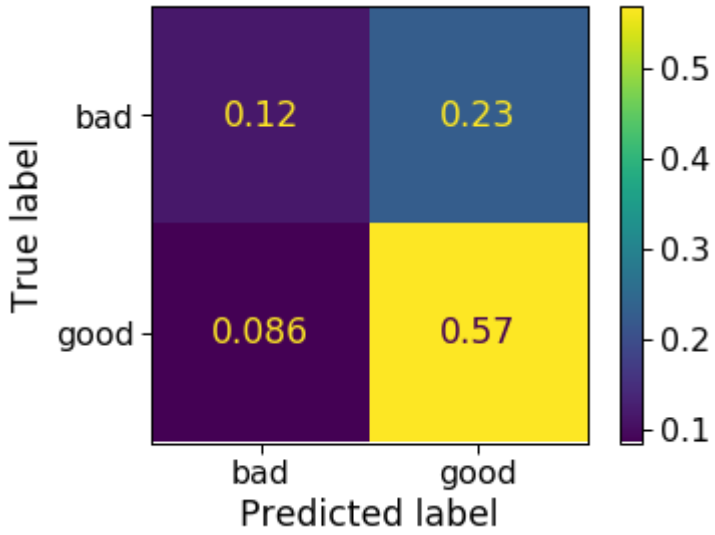
```
0.6852791878172588
```

In [89]:

```
plot_confusion_matrix(pipe2,X_test,y_test,normalize='all')
```

Out[89]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15de86f9
b88>
```



In [90]:

```
recall_score(y_test,ypred2,average=None)
```

Out[90]:

```
array([0.33823529, 0.86821705])
```

In [91]:

```
precision_score(y_test,ypred2,average=None)
```

Out[91]:

```
array([0.575    , 0.7133758])
```

- **Tuning hyperparameters of model 2**

In [92]:

```
param_grid2={
    'C':np.logspace(-3,3,7),
    'penalty': ['l1','l2']

}
clf2 = GridSearchCV(estimator=lgr,
                    param_grid=param_grid2,
                    cv=5)
clf2.fit(xd,y)
print('tuned hyperparameters for logistic reg classifier using grid search:',clf2.best_para
```

```
tuned hyperparameters for logistic reg classifier using grid search: {'C':
0.001, 'penalty': 'l2'}
```

In [93]:

```
lgr_tuned=LogisticRegression(penalty='l2',C=0.001) # l2 ridge, C - inverse of regularizatio
```

In [94]:

```
pipe_lgrt=make_pipeline(ct,lgr_tuned)
```

In [95]:

```
pipe_lgrt.fit(X_train,y_train)
```

Out[95]:

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories
='auto',
                                                                drop=None,
                                                                dtype=<class
'numpy.float64'>,
                                                                handle_unkno
wn='error',
                                                                sparse=Tru
e),
                                                  ['Age', 'Sex', 'Job',
                                                   'Housing', 'Purpose',
                                                   'Savings acc',
                                                   'Checking Acc'])],
                                   verbose=False)),
                ('logisticregression',
                 LogisticRegression(C=0.001, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=None,
                                    solver='lbfgs', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

In [96]:

```
ypredlg_tuned=pipe_lgrt.predict(X_test)
accuracy_score(y_test,ypredlg_tuned,normalize=True)
```

Out[96]:

```
0.7411167512690355
```

Increased accuracy of logistic regression classifier to ~74 %

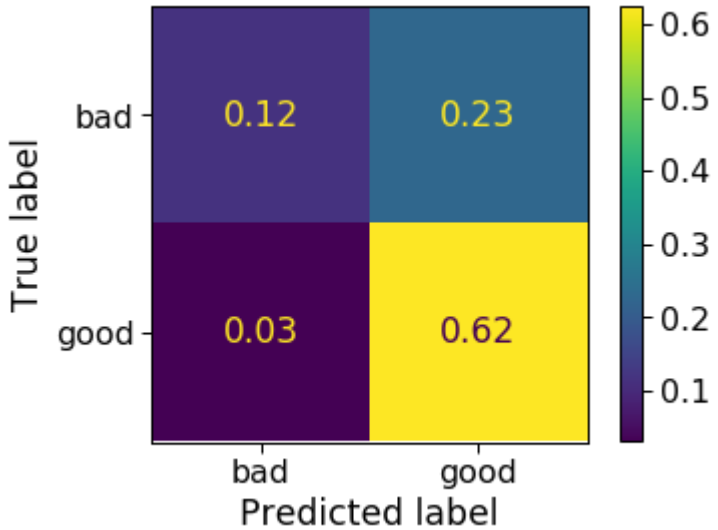
- ***accuracy is increased because by changing the hyperparameter , we have used Ridge Regression (modified version of logistic regression)***
- ***In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.***
- ***This along with low value of C (inverse of strength of regularization) , overfitting is prevented here and thus accuracy is improved.***

In [97]:

```
plot_confusion_matrix(pipe_lgrt,X_test,y_test,normalize='all')
```

Out[97]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15de8979
288>
```



In [98]:

```
recall_score(y_test,ypredlg_tuned,average=None)
```

Out[98]:

```
array([0.33823529, 0.95348837])
```

In [99]:

```
precision_score(y_test,ypredlg_tuned,average=None)
```

Out[99]:

```
array([0.79310345, 0.73214286])
```

In [ ]:

```

```