Problem 2

Task 2.1 [8 pts] Drop the 3 columns that contribute the least to the dataset. These would be the columns with the highest number of non-zero 'none' values. Break ties by going left to right in columns. (Your code should be generalizable to drop n columns, but for the rest of the analysis, you can call your code for n=3.)

In [2]:

```
### problem 2

import pandas as pd
import csv
from pandas import DataFrame
import math
from collections import OrderedDict
from matplotlib import pyplot as plt

mpgfile = "GermanCredit.csv"
#mpgfile = "test.csv"
mpgs = pd.read_csv(mpgfile)
maxCity = {}
maxCityCount = {}

for i in range(len(mpgs)):
    checking_status = mpgs.values[i][0]
    duration = mpgs.values[i][1]
    credit_history = mpgs.values[i][2]
    purpose = mpgs.values[i][3]
    credit_amount = mpgs.values[i][4]
    savings_status = mpgs.values[i][5]
    employment = mpgs.values[i][6]
    personal_status = mpgs.values[i][7]
    other_parties = mpgs.values[i][8]
    residence_since = mpgs.values[i][9]
    property_magnitude = mpgs.values[i][10]
    age = mpgs.values[i][11]
    housing = mpgs.values[i][12]
    existing_credits = mpgs.values[i][13]
    job = mpgs.values[i][14]
    num_dependents = mpgs.values[i][15]
    own_telephone = mpgs.values[i][16]
    foreign_worker = mpgs.values[i][17]
    class_name = mpgs.values[i][18]


#Task 2.1
drop_list = []
index = 0
n=3
for c in mpgs:
    drop_list.append((c, (mpgs[c].eq('none').sum())))
max_val = max(drop_list, key = lambda x:x[1])
```

```
#print(max_val)
#drop_list.sort(key=lambda x:x[1], reverse=True)
a=0
while a <n:
    max_val = max(drop_list, key = lambda x:x[1])
    mpgs = mpgs.drop(max_val[0], axis=1)
    print(max_val, " dropped")
    drop_list.remove(max_val)
    a+=1
```

```
('other_parties', 907)  dropped
('other_payment_plans', 814)  dropped
('own_telephone', 596)  dropped
```

Task 2.1.2 [4 pts] Certain values in some of the columns contain unnecessary apostrophes (‘). Remove the apostrophes.

In [3]:

```
row_drop_list = []
for c in mpgs:
    #mpgs[c] = str(mpgs[c]).replace("'", "")
    new_list = []
    for value in mpgs[c]:
        value = str(value).replace("'", "")
        new_list.append(value)
    mpgs[c] = new_list
    new_list = []
```

[5 pts] The checking_status column has values in 4 categories: 'no checking', '<0', '0<=X<200', and '>=200'. Change these to 'No Checking', 'Low', 'Medium', and 'High' respectively.

In [4]:

```
mpgs['checking_status'] = mpgs['checking_status'].str.replace('no checking',
mpgs['checking_status'] = mpgs['checking_status'].str.replace('<0', 'Low')
mpgs['checking_status'] = mpgs['checking_status'].str.replace('0<=X<200', 'Me
mpgs['checking_status'] = mpgs['checking_status'].str.replace('>=200', 'High'
```

[5 pts] The savings_status column has values in 4 categories: 'no known savings', '<100', '100<=X<500', '500<=X<1000', and '>=1000'. Change these to 'No Savings', 'Low', 'Medium', 'High', and 'High' respectively. (Yes, the last two are both 'High').

In [5]:

```
#Task 2.4
mpgs['savings_status'] = mpgs['savings_status'].str.replace('500<=X<1000', 'H
mpgs['savings_status'] = mpgs['savings_status'].str.replace('100<=X<500', 'Me
mpgs['savings_status'] = mpgs['savings_status'].str.replace('<100', 'Low')
mpgs['savings_status'] = mpgs['savings_status'].str.replace('>=1000', 'High')
mpgs['savings_status'] = mpgs['savings_status'].str.replace('no known savings
```

[4 pts] Change class column values from 'good' to '1' and 'bad' to '0'.

In [6]:

```
#Task 2.5
mpgs['class'] = mpgs['class'].str.replace('good', '1')
mpgs['class'] = mpgs['class'].str.replace('bad', '0')
```

[5 pts] Change the employment column value 'unemployed' to 'Unemployed', and for the others, change to 'Amateur', 'Professional', 'Experienced' and 'Expert', depending on year range.

In [196…

```
mpgs['employment'] = mpgs['employment'].str.replace('unemployed', 'Unemployed
mpgs['employment'] = mpgs['employment'].str.replace('<1', 'Amateur')
mpgs['employment'] = mpgs['employment'].str.replace('1<=X<4', 'Professional')
mpgs['employment'] = mpgs['employment'].str.replace('4<=X<7', 'Experienced')
mpgs['employment'] = mpgs['employment'].str.replace('>=7', 'Expert')


display(mpgs)
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_statu
0	Low	6	critical/other existing credit	radio/tv	1169	No Saving
1	Medium	48	existing paid	radio/tv	5951	Lo
2	No Checking	12	critical/other existing credit	education	2096	Lo
3	Low	42	existing paid	furniture/equipment	7882	Lo
4	Low	24	delayed previously	new car	4870	Lo
...	...	...	...	...	...	
995	No Checking	12	existing paid	furniture/equipment	1736	Lo
996	Low	30	existing paid	used car	3857	Lo
997	No Checking	12	existing paid	radio/tv	804	Lo
998	Low	45	existing paid	radio/tv	1845	Lo
999	Medium	45	critical/other existing credit	used car	4576	Mediu

1000 rows × 18 columns

Analysis

Task 2.2.1 [5 pts] Often we need to find correlations between categorical attributes, i.e. attributes that have values that fall in one of several categories, such as "yes"/"no" for attr1, or "low","medium","high" for attr2. One such correlation is to find counts in combinations of categorial values across attributes, as in how many instances are "yes" for attr1 and "low" for attr2. A good way to find such counts is to use the Pandas crosstab function. Do this for the following two counts.

[3 pts] Get the count of each category of foreign workers (yes and no) for each class of credit (good and bad). [2 pts] Similarly, get the count of each category of employment for each category of saving_status.

In [197…

```
## analysis
pd.crosstab(mpgs['foreign_worker'], mpgs['class'])
```

Out[197…

class	0	1
foreign_worker		
no	4	33
yes	296	667

Task 2.2.1 #2

In [198…

```
pd.crosstab(mpgs['employment'], mpgs['savings_status'])
```

Out[198…

savings_status	High	Low	Medium	No Savings
employment				
Amateur	12	120	17	23
Experienced	18	100	24	32
Expert	34	133	22	64
Professional	44	210	33	52
Unemployed	3	40	7	12

Task 2.2.2 [4 pts] Find the average credit_amount of single males that have 4<=X<7 years of employment. You can leave the raw result as is, no need for rounding.

In [199…

```
## analysis

# 2.2.1
pd.crosstab(mpgs['foreign_worker'], mpgs['class'])

#2.2.2
## still a work in progress


#mpgs['temperature'] = mpgs['temperature'].fillna(mpgs.groupby(['EU','coastli

mpgs['credit_amount'] = mpgs['credit_amount'].astype(float)



#credit_df = mpgs.groupby(['personal_status', 'employment'])['credit_amount']
credit = mpgs['credit_amount'].where(mpgs['personal_status'] == 'male single'
credit = credit.mean()
print(credit)


#display(mpgs)
```

```
4142.592592592592
```

Task 2.2.3 [4 pts] Find the average credit duration for each of the job types. You can leave the raw result as is, no need for rounding.

In [200…

```
## 2.2.3
mpgs['duration'] = mpgs['duration'].astype(float)
job_mean_dict = {}
for j in mpgs['job']:
    if j not in job_mean_dict:
        job_mean_dict[j] = 0
    else:
        pass
for j in job_mean_dict.keys():
    temp = mpgs['duration'].where(mpgs['job'] == j)
    temp = temp.mean()
    job_mean_dict[j] = temp


#print(job_mean_dict)
for key,val in job_mean_dict.items():
    print(key,' : ', val)
```

```
skilled  :  21.411111111111111
unskilled resident  :  16.535
high qualif/self emp/mgmt  :  25.16891891891892
unemp/unskilled non res  :  17.363636363636363
```

Task 2.2.4 [4 pts] For the purpose 'education', what is the most common checking_status and savings_status? Your code should print: Most common checking status: ... Most common savings status: ...

In [201…

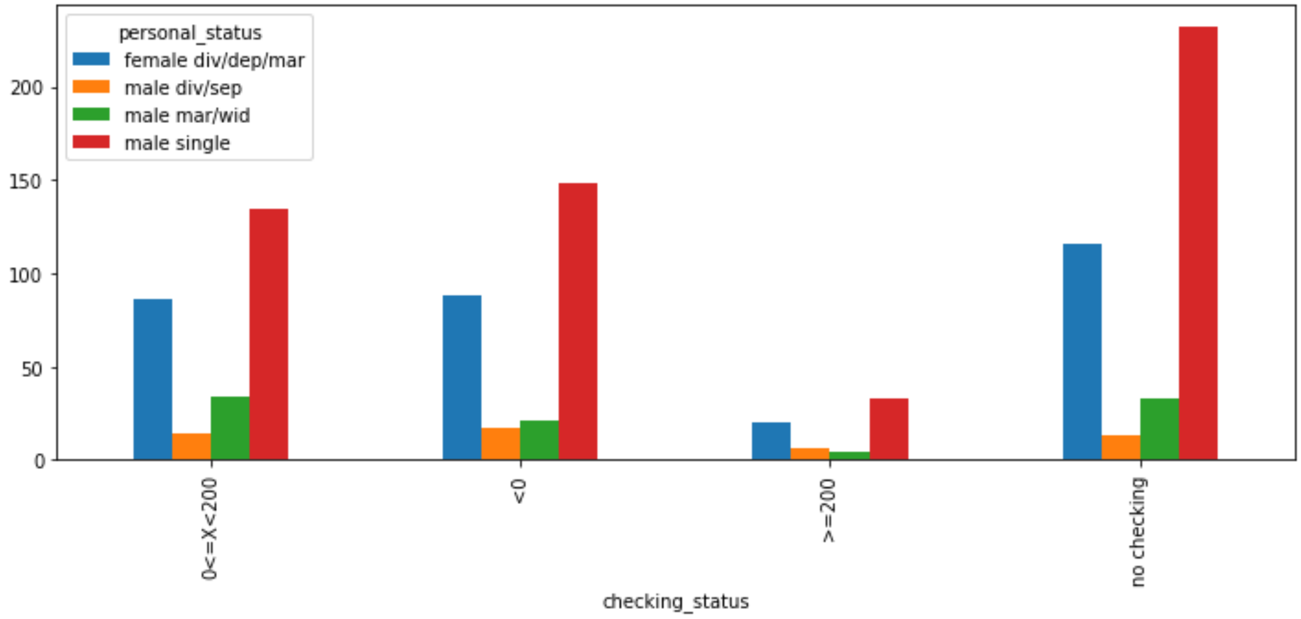
```
edu_checking = mpgs['checking_status'].where(mpgs['purpose'] == 'education')
edu_checking = edu_checking.mode()
edu_checking = edu_checking[0]
edu_saving = mpgs['savings_status'].where(mpgs['purpose'] == 'education')
edu_saving = edu_saving.mode()
edu_saving = edu_saving[0]

print("Most common checking status: " + edu_checking)
print("Most common savings status: " + edu_saving)
```

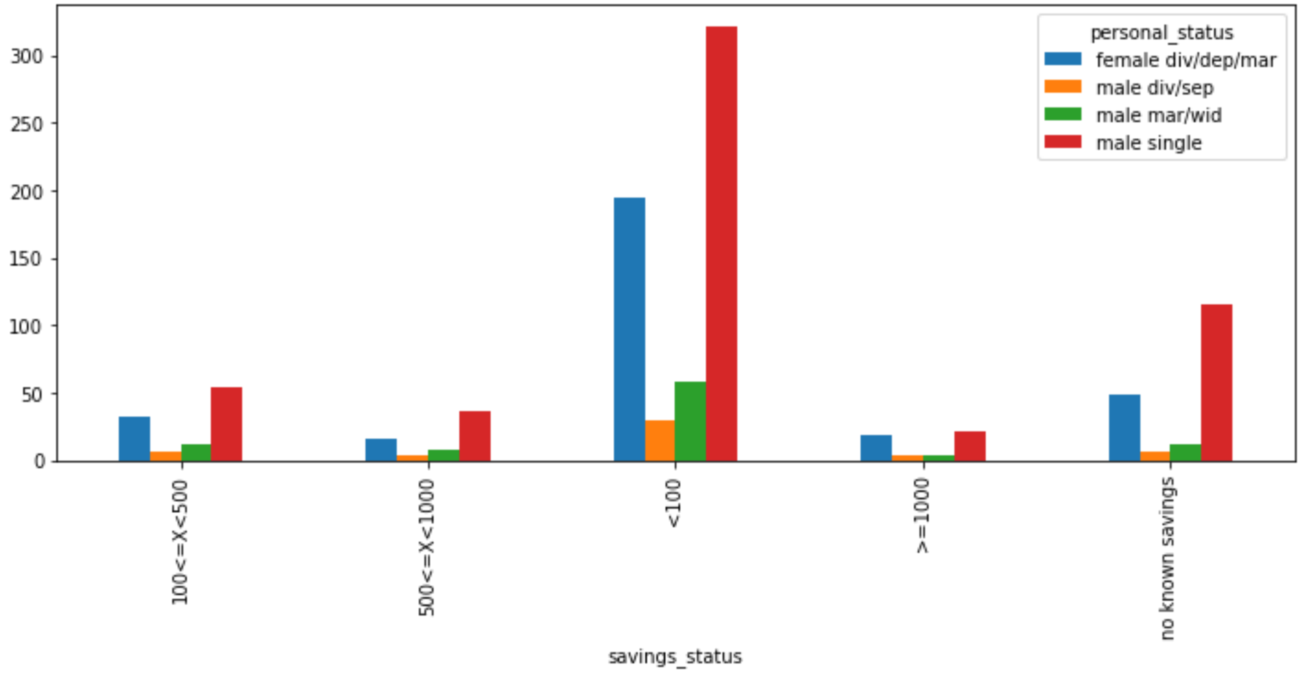
```
Most common checking status: No Checking
Most common savings status: Low
```

[9 pts] Plot subplots of two histograms: one with savings_status on the x-axis and personal_status as different colors, and another with checking_status on the x-axis and personal_status as different colors.

In [188…

```
#2.3.1
#[9 pts] Plot subplots of two histograms: one with savings_status on the x-ax
#personal_status as different colors,
#and another with checking_status on the x-axis and personal_status as differ
personal_status_list = []
checking_status_list = []
savings_status_list = []

#savings_status_list= mpgs['savings_status'].tolist()
plt.figure(figsize= (10,10))
axis = plt.subplot(211)
pd.crosstab(mpgs['checking_status'], mpgs['personal_status']).plot(kind='bar'
ax1 = plt.subplot(212)
pd.crosstab(mpgs['savings_status'], mpgs['personal_status']).plot(kind='bar',
#cross2 = pd.crosstab(mpgs['checking_status'], mpgs['personal_status']).plot(
#plt.hist(cross)
plt.tight_layout()
plt.show()

#plt.subplots(1,2)
#plt.hist(saving_status_list,bins = 10, color="navy", edgecolor="black")
#plt.xticks(range(0, int(max(population_list)+1), int(max(population_list)/5)
#plt.xlabel('Population')
#plt.ylabel('Number of Countries')
```

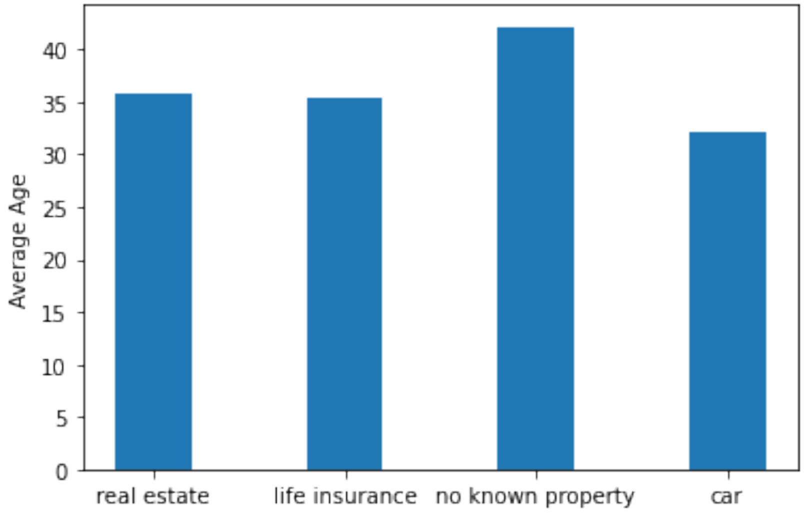




[9 pts] For people having credit_amount more than 4000, plot a bar graph which maps property_magnitude (x-axis) to the average customer age for that magnitude (y-axis).

In [187…

```
#[9 pts] For people having credit_amount more than 4000,
#plot a bar graph which maps property_magnitude (x-axis) to the average custo

df_32 = mpgs
df_32['credit_amount'] = df_32['credit_amount'].astype(int)

df_32_new = df_32.where(df_32['credit_amount'] > 4000)

bar_chart_dict = {}

for row_val in df_32_new.index:
    a = df_32_new['property_magnitude'][row_val]
    b = df_32_new['age'][row_val]
    if str(a)=='nan' or str(b)=='nan':
        continue
    if a not in bar_chart_dict.keys():
        bar_chart_dict[a] = [float(b)]
    else:
        bar_chart_dict[a].append(float(b))


for barkey,barval in bar_chart_dict.items():
    temp = float(sum(barval)) / float(len(barval))
    bar_chart_dict[barkey] = temp
plt.bar(bar_chart_dict.keys(), bar_chart_dict.values(), width = .4)
plt.ylabel('Average Age')
plt.show()
```



Task 2.3.3 [6 pts] For people with a "High" savings_status and age above 40, use subplots to plot the following pie charts:

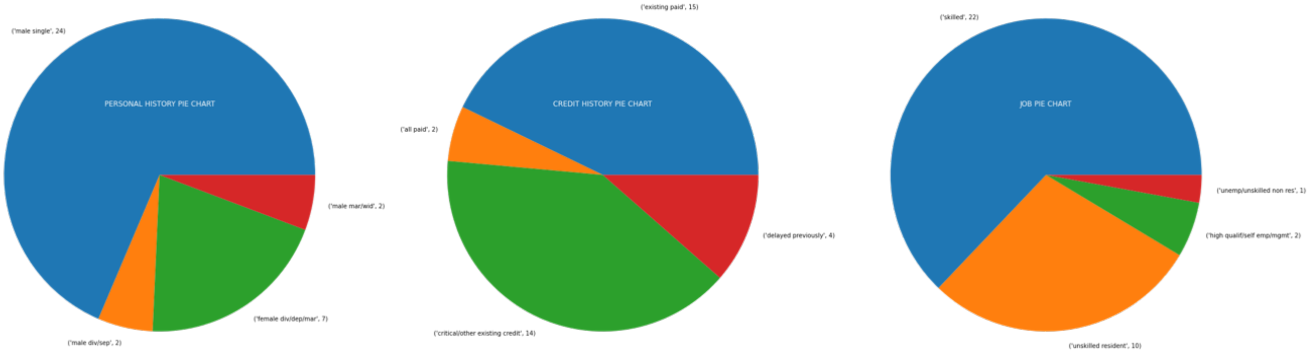
- Personal status
- Credit history
- Job

In [209…

```
#2.3.3
fig, axis = plt.subplots(1,3)

mpgs['age'] = mpgs['age'].astype(float)
p_dict = {}
df_33 = mpgs.where(mpgs['savings_status'] == 'High').where(mpgs['age'] >40)
for row_val in df_33['personal_status']:
    if str(row_val)=='nan':
        continue
    #print(row_val)
    if row_val not in p_dict:
        p_dict[row_val] = 1
    else:
        p_dict[row_val]+=1

c_dict = {}
df_33b = mpgs.where(mpgs['savings_status'] == 'High').where(mpgs['age'] >40)
for row_val in df_33b['credit_history']:
    if str(row_val)=='nan':
        continue
    #print(row_val)
    if row_val not in c_dict:
        c_dict[row_val] = 1
    else:
        c_dict[row_val]+=1


j_dict = {}
df_33c = mpgs.where(mpgs['savings_status'] == 'High').where(mpgs['age'] >40)
for row_val in df_33c['job']:
    if str(row_val)=='nan':
        continue
    #print(row_val)
    if row_val not in j_dict:
        j_dict[row_val] = 1
    else:
        j_dict[row_val]+=1

dataval_dict = {}

labelp = p_dict.items()
labelc = c_dict.items()
labelj = j_dict.items()
dataval_dict[0] = p_dict
dataval_dict[1] = c_dict
dataval_dict[2] = j_dict
axis[0].pie(dataval_dict[0].values(), radius = 3, labels=labelp)
axis[0].title.set_text('PERSONAL HISTORY PIE CHART')
axis[0].title.set_color('white')
axis[1].pie(dataval_dict[1].values(), radius = 3, labels=labelc)
axis[1].title.set_text('CREDIT HISTORY PIE CHART')
axis[1].title.set_color('white')
axis[2].pie(dataval_dict[2].values(), radius = 3, labels=labelj)
```

```
axis[2].title.set_text('JOB PIE CHART')
axis[2].title.set_color('white')


print(dataval_dict)
plt.subplots_adjust(right = 5)
plt.show()
```

```
{0: {'male single': 24, 'male div/sep': 2, 'female div/dep/mar': 7, 'male mar/
wid': 2}, 1: {'existing paid': 15, 'all paid': 2, 'critical/other existing cre
dit': 14, 'delayed previously': 4}, 2: {'skilled': 22, 'unskilled resident': 1
0, 'high qualif/self emp/mgmt': 2, 'unemp/unskilled non res': 1}}
```



```
In [ ]:
```