Credit Facility Dataset

Question 1

This credit facility dataset to be analyzed comprises records of customers' demographics, amount owed, repayment history/status etc. The data dictionary of this dataset is depicted in Appendix 1.

List the categorical and numeric variables in this dataset.

APPENDIX 1 – DATA DICTIONARY

Variable	**Description**
ID	Customer unique identifier
LIMIT	Customer total limit
BALANCE	Customer current credit balance (snapshot in time)
INCOME	Customer current income
GENDER	Customer gender (0: Male, 1: Female)
EDUCATION	Customer highest education attained (0: Others, 1: Postgraduate, 2: Tertiary, 3: High School)
MARITAL	Customer marital status (0: Others, 1: Single, 2: Married)
AGE	Customer age in years
S(n)	Customer repayment reflected status in nth month. (-1; Prompt payment, 0: Minimum sum payment, x = Delayed payment for x month(s))
B(n)	Customer billable amount in nth month
R(n)	Customer previous repayment amount, paid in nth month
RATING	Customer rating (0: Good, 1: Bad)

Note: n=1 signifies the most recent month, while n=5 signifies the previous 4th month. If n=1 is the month of May 2022, then n=5 is the month of January 2022.

In [18]:

```
import pandas as pd
df = pd.read_csv('ECA_data.csv')
df.head()
```

Out[18]:

	ID	LIMIT	BALANCE	INCOME	RATING	GENDER	EDUCATION	MARITAL	AGE	S1	...
0	1	210000	0.00	235822	1	1	1.0	2.0	30	0	...
1	2	260000	10928.05	278481	0	0	2.0	2.0	31	0	...
2	3	400000	65397.85	431993	0	0	3.0	1.0	51	0	...
3	4	20000	3695.30	22368	0	0	2.0	1.0	58	-1	...
4	5	180000	68.25	166900	0	1	2.0	1.0	42	0	...

5 rows × 24 columns

List the categorical and numeric variables in this dataset.

In [19]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18769 entries, 0 to 18768
Data columns (total 24 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         18769 non-null  int64  
 1   LIMIT      18769 non-null  int64  
 2   BALANCE    18769 non-null  float64
 3   INCOME     18769 non-null  int64  
 4   RATING     18769 non-null  int64  
 5   GENDER     18769 non-null  int64  
 6   EDUCATION  18756 non-null  float64
 7   MARITAL    18731 non-null  float64
 8   AGE        18769 non-null  int64  
 9   S1         18769 non-null  int64  
 10  S2         18769 non-null  int64  
 11  S3         18769 non-null  int64  
 12  S4         18769 non-null  int64  
 13  S5         18769 non-null  int64  
 14  B1         18769 non-null  int64  
 15  B2         18769 non-null  int64  
 16  B3         18769 non-null  int64  
 17  B4         18769 non-null  int64  
 18  B5         18769 non-null  int64  
 19  R1         18769 non-null  int64  
 20  R2         18769 non-null  int64  
 21  R3         18769 non-null  object 
 22  R4         18769 non-null  int64  
 23  R5         18769 non-null  int64  
dtypes: float64(3), int64(20), object(1)
memory usage: 3.4+ MB
```

Question 2

Conduct four (4) data pre-processing tasks for the analysis of the data, explaining results obtained.

first data pre-processing task

In [20]:

```
#first data quality issue
#as we clean the data based on missing value, but the cleaning process is not done yet
#following command will show the null value containing in the column
df.isnull().sum()
```

Out[20]:

```
ID            0
LIMIT         0
BALANCE       0
INCOME        0
RATING        0
GENDER        0
EDUCATION    13
MARITAL      38
AGE           0
S1            0
S2            0
S3            0
S4            0
S5            0
B1            0
B2            0
B3            0
B4            0
B5            0
R1            0
R2            0
R3            0
R4            0
R5            0
dtype: int64
```

In [21]:

```
#solution for first data pre-processing task
# based on above information now we have to remove the null value rows because the dataset
df = df[df.EDUCATION.notnull()]
# now runing the below command while ensure that no null value is present now in dataset,
# dataset is fully qualified based on no null values

df.isnull().sum()
```

Out[21]:

```
ID            0
LIMIT         0
BALANCE       0
INCOME        0
RATING        0
GENDER        0
EDUCATION     0
MARITAL      36
AGE           0
S1            0
S2            0
S3            0
S4            0
S5            0
B1            0
B2            0
B3            0
B4            0
B5            0
R1            0
R2            0
R3            0
R4            0
R5            0
dtype: int64
```

Second pre-processing task

In [22]:

```
#second data quality issue
#as we clean the data based on missing value, but the cleaning process is not done yet
#following command will show the null value containing in the column
df.isnull().sum()
```

Out[22]:

```
ID            0
LIMIT         0
BALANCE       0
INCOME        0
RATING        0
GENDER        0
EDUCATION     0
MARITAL      36
AGE           0
S1            0
S2            0
S3            0
S4            0
S5            0
B1            0
B2            0
B3            0
B4            0
B5            0
R1            0
R2            0
R3            0
R4            0
R5            0
dtype: int64
```

In [23]:

```
#solution for second data pre-processing task
# based on above information now we have to remove the null value rows because the dataset
df = df[df.MARITAL.notnull()]
# now runing the below command while ensure that no null value is present now in dataset,
# dataset is fully qualified based on no null values

df.isnull().sum()
```

Out[23]:

```
ID           0
LIMIT        0
BALANCE      0
INCOME       0
RATING       0
GENDER       0
EDUCATION    0
MARITAL      0
AGE          0
S1           0
S2           0
S3           0
S4           0
S5           0
B1           0
B2           0
B3           0
B4           0
B5           0
R1           0
R2           0
R3           0
R4           0
R5           0
dtype: int64
```

Third pre-processing task

In [24]:

```
df.describe()
```

Out[24]:

	ID	LIMIT	BALANCE	INCOME	RATING	GENDER
count	18720.000000	18720.000000	18720.000000	18720.000000	18720.000000	18720.000000
mean	9381.028419	168307.888889	9135.957564	177815.004006	0.219605	0.618056
std	5419.090246	129496.631582	13057.178823	143200.252119	0.413990	0.485876
min	1.000000	10000.000000	0.000000	10000.000000	0.000000	0.000000
25%	4685.750000	50000.000000	644.612500	56427.750000	0.000000	0.000000
50%	9381.500000	140000.000000	3974.512500	148178.000000	0.000000	1.000000
75%	14076.250000	240000.000000	11986.318750	257206.750000	0.000000	1.000000
max	18766.000000	800000.000000	130692.450000	908846.000000	1.000000	1.000000

8 rows × 23 columns

as we see in above command output AGE should not be smaller than 18 and not greater than 90

In [25]:

```
df = df[df.AGE >= 18]
df = df[df.AGE < 90]
```

In [26]:

```
df.describe()
```

Out[26]:

	ID	LIMIT	BALANCE	INCOME	RATING	GENDER
count	18710.000000	18710.000000	18710.000000	18710.000000	18710.000000	18710.000000
mean	9380.541315	168333.173704	9137.654454	177843.910476	0.219722	0.618172
std	5417.637087	129498.284950	13057.804966	143200.065776	0.414069	0.485848
min	1.000000	10000.000000	0.000000	10000.000000	0.000000	0.000000
25%	4687.250000	50000.000000	644.437500	56466.500000	0.000000	0.000000
50%	9381.500000	140000.000000	3975.912500	148189.000000	0.000000	1.000000
75%	14072.750000	240000.000000	11989.731250	257209.000000	0.000000	1.000000
max	18766.000000	800000.000000	130692.450000	908846.000000	1.000000	1.000000

8 rows × 23 columns

Fourth pre-processing task

In [27]:

```
### given appendix data compare with the dataset values
# as we know some of the column values are fixed
```

GENDER	Customer gender (0: Male, 1: Female)

In [28]:

```
#check Gender column has only 0 and 1 values
df.GENDER.value_counts()
```

Out[28]:

```
1    11566
0     7144
Name: GENDER, dtype: int64
```

EDUCATION	Customer highest education attained (0: Others, 1: Postgraduate, 2: Tertiary, 3: High School)

In [29]:

```
#check EDUCATION column has only 0,1,2,3 values
df.EDUCATION.value_counts()
```

Out[29]:

```
2.0    8865
1.0    6404
3.0    3107
0.0     334
Name: EDUCATION, dtype: int64
```

MARITAL	Customer marital status (0: Others, 1: Single, 2: Married)

In [30]:

```
#check MARITAL column has only 0,1,2 values
df.MARITAL.value_counts()
```

Out[30]:

```
2.0    9822
1.0    8699
0.0     189
Name: MARITAL, dtype: int64
```

S(n)	Customer repayment reflected status in nth month. (-1; Prompt payment, 0: Minimum sum payment, x = Delayed payment for x month(s))

In [31]:

```
#check S(1-5) column has only -1,0,x(any number greater than 0) values
print(df.S1.value_counts())
print(df.S2.value_counts())
print(df.S3.value_counts())
print(df.S4.value_counts())
print(df.S5.value_counts())
```

```
 0    12336
-1     3588
 2     2462
 3      208
 4       71
 1       19
 5       13
 7        8
 6        5
Name: S1, dtype: int64
 0    12580
-1     3521
 2     2359
 3      177
 4       42
 6       11
 7        9
 5        8
 8        2
 1        1
Name: S2, dtype: int64
 0    12929
-1     3450
 2     2126
 3      104
 7       41
 4       39
 5       18
 6        2
 1        1
Name: S3, dtype: int64
 0    13581
-1     3317
 2     1599
 3      103
 4       56
 7       40
 5       11
 6        3
Name: S4, dtype: int64
 0    13620
-1     3292
 2     1594
 3      114
 4       36
 7       34
 5       11
 6        9
Name: S5, dtype: int64
```

values of the columns are as per given values set

Fifth pre-processsing task

billable amount should be greater than 0 and should be smaller than the LIMIT

In [32]:

```
df = df[df.B1 >= 0]
df = df[df.B2 >= 0]
df = df[df.B3 >= 0]
df = df[df.B4 >= 0]
df = df[df.B5 >= 0]
df = df[df.B1 <= df.LIMIT]
df = df[df.B2 <= df.LIMIT]
df = df[df.B3 <= df.LIMIT]
df = df[df.B4 <= df.LIMIT]
df = df[df.B5 <= df.LIMIT]
df = df[df.B1 <= df.INCOME]
df = df[df.B2 <= df.INCOME]
df = df[df.B3 <= df.INCOME]
df = df[df.B4 <= df.INCOME]
df = df[df.B5 <= df.INCOME]
```

Question 3

Articulate five (5) relevant insights of the data, with supporting visualization for each insight.

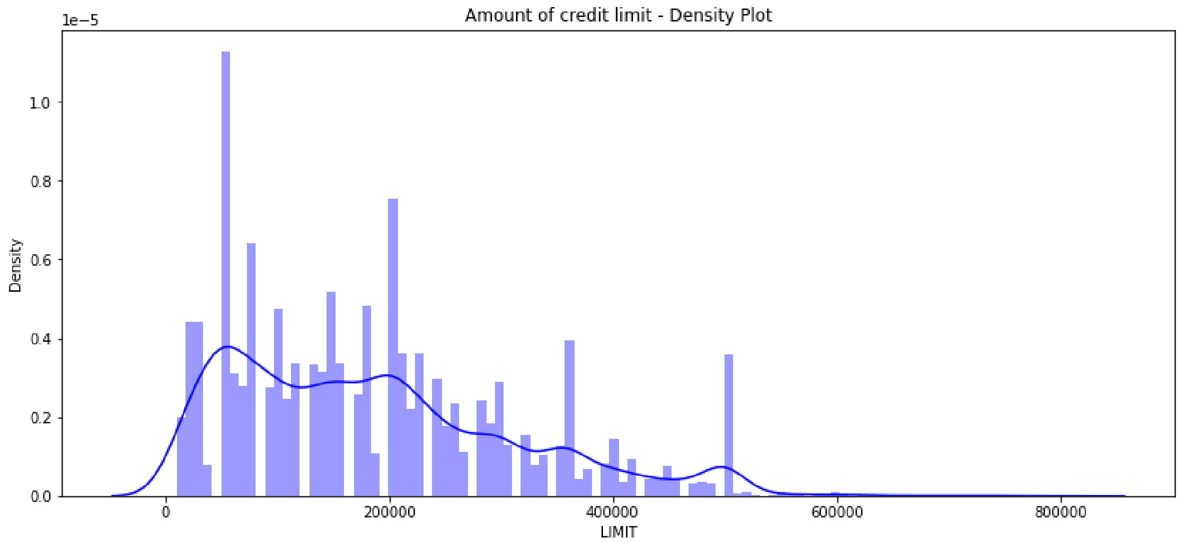
First insights of the data

Amount of credit limit - Density Plot

In [33]:

```
from matplotlib import pyplot as plt
import seaborn as sns
plt.figure(figsize = (14,6))
plt.title('Amount of credit limit - Density Plot')
sns.set_color_codes("pastel")
sns.distplot(df['LIMIT'],kde=True,bins=100, color="blue")
plt.show()
```

```
C:\Users\rao\AppData\Roaming\Python\Python310\site-packages\seaborn\distribu
tions.py:2619: FutureWarning: `distplot` is a deprecated function and will b
e removed in a future version. Please adapt your code to use either `displot
` (a figure-level function with similar flexibility) or `histplot` (an axes-
level function for histograms).
  warnings.warn(msg, FutureWarning)
```

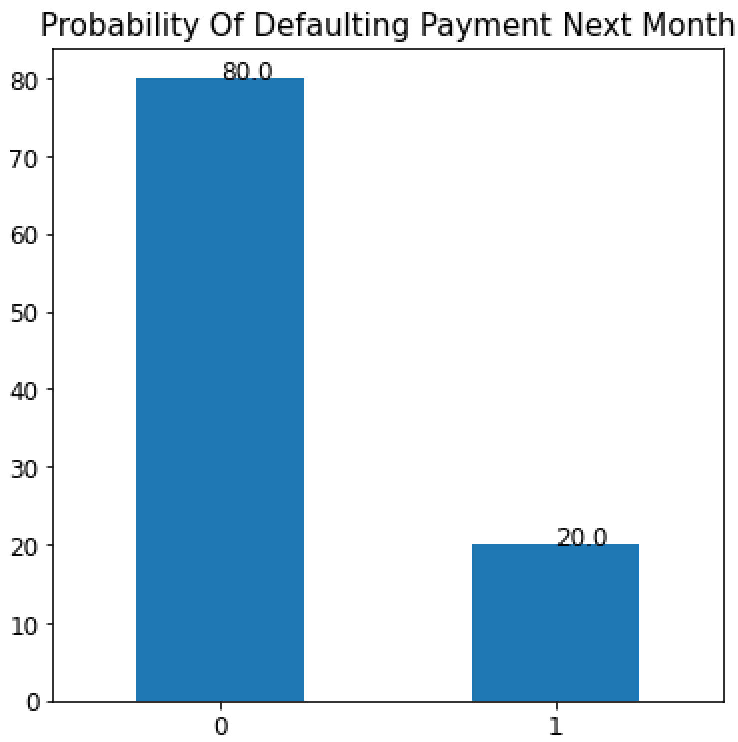


Second insights of the data

Probability Of Defaulting Payment Next Month

In [34]:

```
import seaborn as sns
from matplotlib import pyplot as plt
def_cnt = (df.RATING.value_counts(normalize=True)*100)
def_cnt.plot.bar(figsize=(6,6))
plt.xticks(fontsize=12, rotation=0)
plt.yticks(fontsize=12)
plt.title("Probability Of Defaulting Payment Next Month", fontsize=15)
for x,y in zip([0,1],def_cnt):
    plt.text(x,y,y,fontsize=12)
plt.show()
```



Third insights of the data

Rating based on Gender

In [35]:

```
df.groupby(['GENDER', 'RATING']).size()
#gender 0 is for male and 1 is for female
```

Out[35]:

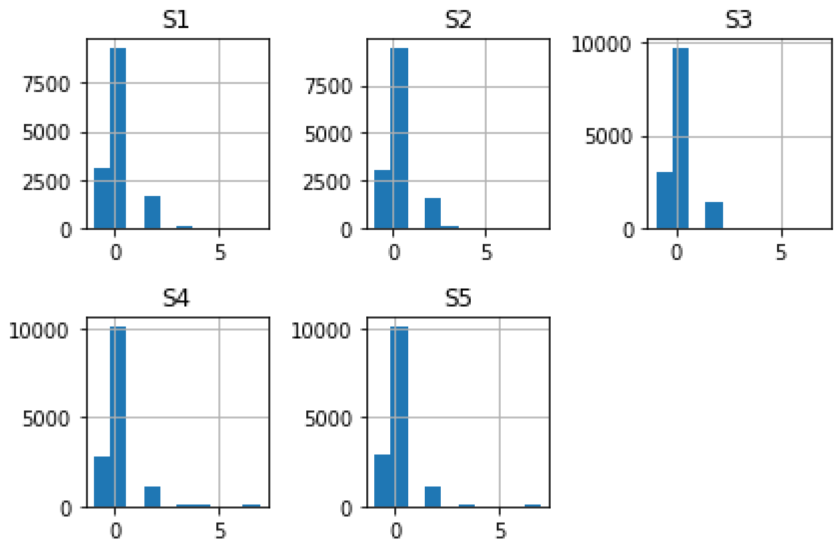
```
GENDER  RATING
0       0         4009
        1         1188
1       0         7415
        1         1668
dtype: int64
```

Fourth insights of the data

Customer repayment status

In [36]:

```
def draw_histograms(df, variables, n_rows, n_cols, n_bins):
    fig=plt.figure()
    for i, var_name in enumerate(variables):
        ax=fig.add_subplot(n_rows,n_cols,i+1)
        df[var_name].hist(bins=n_bins,ax=ax)
        ax.set_title(var_name)
    fig.tight_layout()  # Improves appearance a bit.
    plt.show()

score = df[['S1','S2', 'S3', 'S4', 'S5']]
draw_histograms(score, score.columns, 2, 3, 10)
```

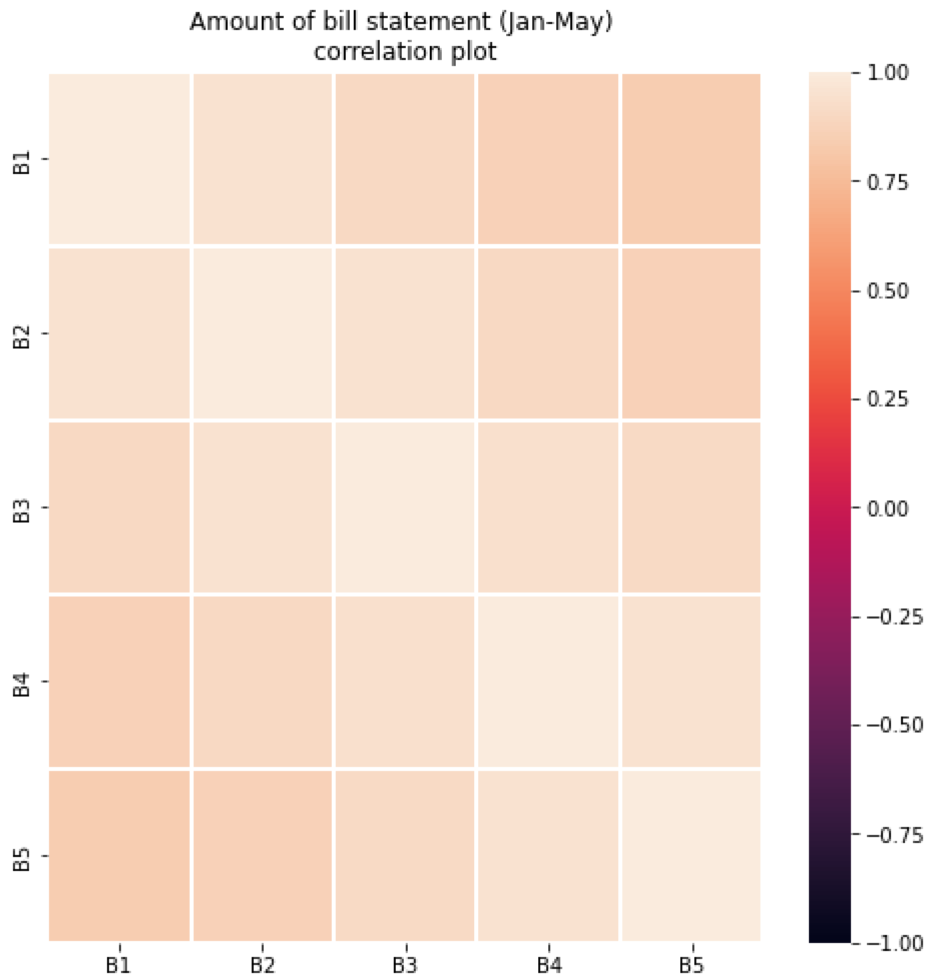


Fifth insights of the data

Correlation between Customer billable amount in n month

In [37]:

```
var = ['B1','B2','B3','B4','B5']

plt.figure(figsize = (8,8))
plt.title('Amount of bill statement (Jan-May) \ncorrelation plot')
corr = df[var].corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,vmin=-1, v
plt.show()
```



Question 4

Perform linear regression modelling to predict the variable, B1, explaining the approach taken, including any further data pre-processing.

In [38]:

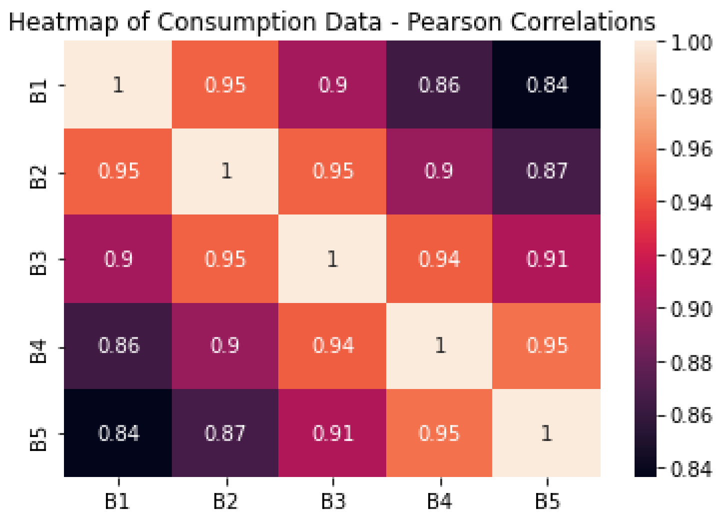
```
var1 = ['B1','B2','B3','B4','B5']
df[var1].describe()
```

Out[38]:

	B1	B2	B3	B4	B5
count	14280.000000	14280.000000	14280.000000	14280.000000	14280.000000
mean	45871.044958	44243.905182	42187.393697	39207.322479	38317.792787
std	67409.605391	65618.882768	62749.192621	59357.195308	58513.838308
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2409.500000	2232.750000	2039.750000	1507.250000	1242.000000
50%	17707.000000	17282.500000	16629.000000	15102.500000	14243.500000
75%	60549.000000	58042.750000	55247.500000	50537.000000	49619.750000
max	577681.000000	577015.000000	565669.000000	530672.000000	499100.000000

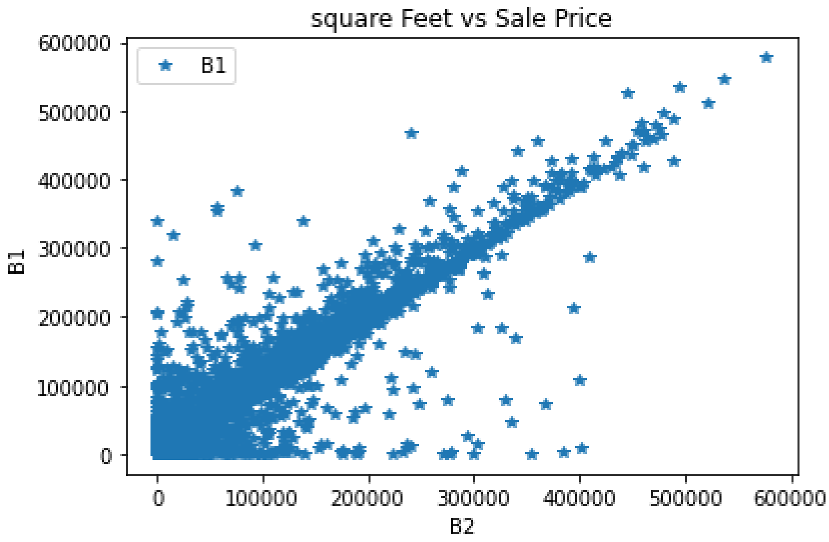
In [39]:

```
var1 = ['B1','B2','B3','B4','B5']
correlations = df[var1].corr()
# annot=True displays the correlation values
sns.heatmap(correlations, annot=True).set(title='Heatmap of Consumption Data - Pearson Corr
```



In [40]:

```
df.plot(x='B2',y='B1',style = '*')
plt.title('square Feet vs Sale Price')
plt.xlabel( 'B2')
plt.ylabel('B1')
plt.show()
```



In [42]:

```
X = df.B2.values.reshape(-1,1)
y = df.B1.values.reshape(-1,1)
```

In [43]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=40)
```

In [44]:

```
df.head(1)
```

Out[44]:

	ID	LIMIT	BALANCE	INCOME	RATING	GENDER	EDUCATION	MARITAL	AGE	S1	...	E
0	1	210000	0.0	235822	1	1	1.0	2.0	30	0	...	

1 rows × 24 columns

In [45]:

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

Out[45]:

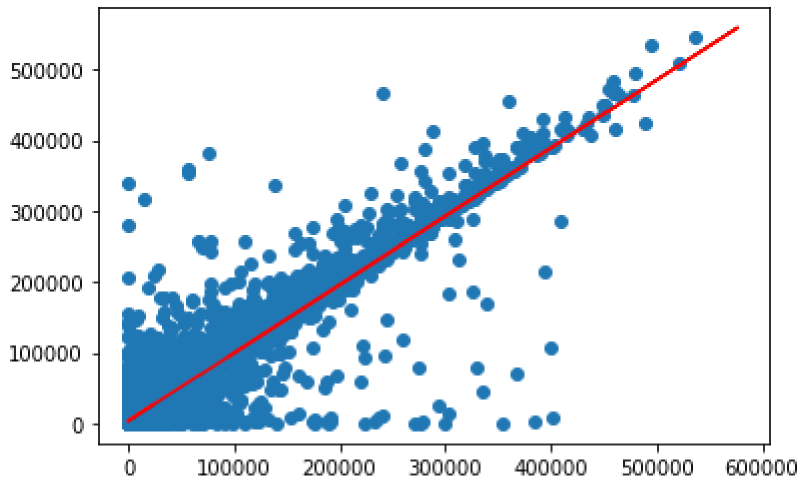
```
▾ LinearRegression
LinearRegression()
```

In [46]:

```
y_pred = regressor.predict(X_test)
```

In [47]:

```
plt.scatter(X_train,y_train)
plt.plot(X_test,y_pred,color = 'red')
plt.show()
```



In [48]:

```
df_preds = pd.DataFrame({'Actual': y_test.squeeze(), 'Predicted': y_pred.squeeze()})
print(df_preds)
```

```
      Actual      Predicted
0     125098   94392.552183
1     164745  165215.821081
2      68544   61597.769001
3          0    7316.549034
4          0    2999.009650
...      ...            ...
2851  125782  117079.644884
2852    2400    5313.527077
2853  135846  134860.925020
2854   12294   15572.625574
2855   36567   26220.370123

[2856 rows x 2 columns]
```

Question 5

State the linear regression equation and explain key insights from the results obtained in Question 4.

The equation that describes any straight line is: $y = a*x+b$

In this equation, y represents the score percentage, x represent the hours studied. b is where the line starts at the Y-axis, also called the Y-axis intercept and a defines if the line is going to be more towards the upper or lower part of the graph (the angle of the line), so it is called the slope of the line.

In [51]:

```
X = df.B2.values.reshape(-1,1)
y = df.B1.values.reshape(-1,1)
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=40)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[51]:

```
▾ LinearRegression
LinearRegression()
```

In [52]:

```
print("intercept(b):",regressor.intercept_)
```

```
intercept(b): [2999.00964951]
```

In [54]:

```
print("slope(a):",regressor.coef_)
```

```
slope(a): [[0.96438226]]
```

Making predictions

In [63]:

```
#equation is y = a * x + b
def calc(slope, intercept, hours):
    return slope*hours+intercept

score = calc(regressor.coef_, regressor.intercept_, 10000)
print(score)
```

```
[[12642.83226386]]
```

In [62]:

```
#our linear regression model
# Passing 10000 in double brackets to have a 2 dimensional array
score = regressor.predict([[10000]])
print(score)
```

```
[[12642.83226386]]
```

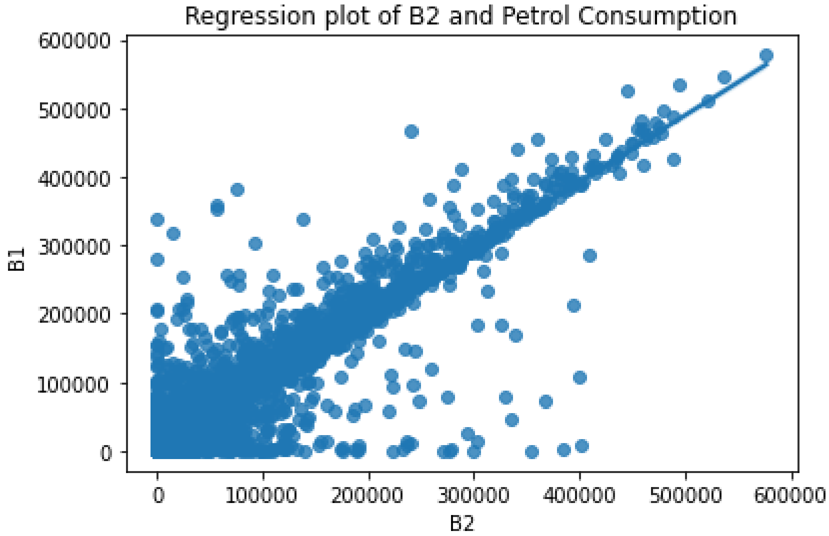
In [65]:

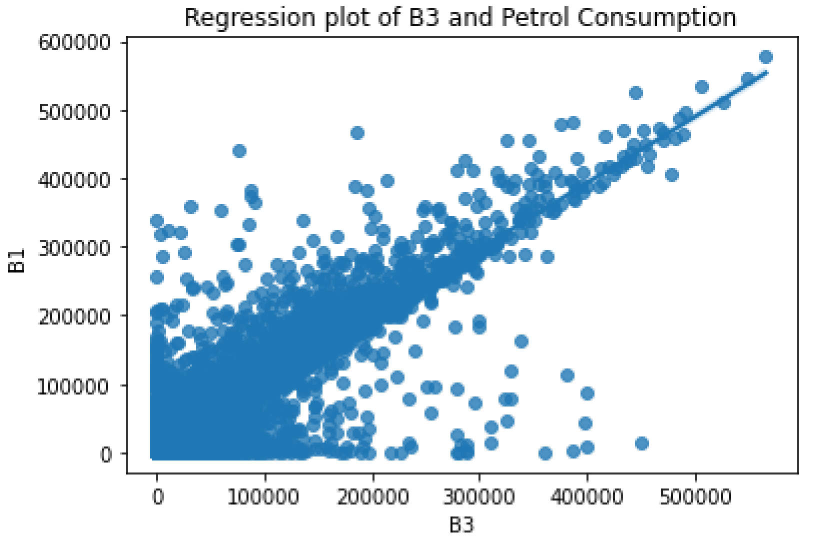
```
y_pred = regressor.predict(X_test)
print(df_preds)
```

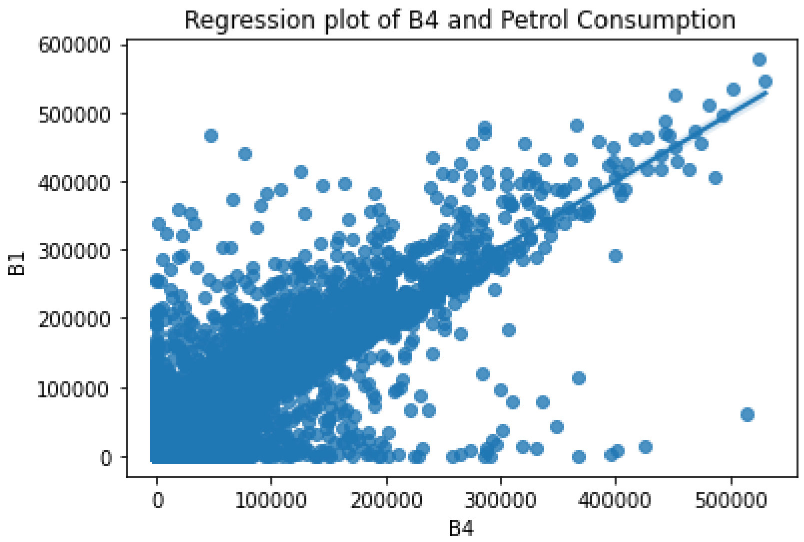
```
      Actual      Predicted
0     125098   94392.552183
1     164745  165215.821081
2      68544   61597.769001
3          0    7316.549034
4          0    2999.009650
...      ...            ...
2851  125782  117079.644884
2852    2400    5313.527077
2853  135846  134860.925020
2854   12294   15572.625574
2855   36567   26220.370123

[2856 rows x 2 columns]
```

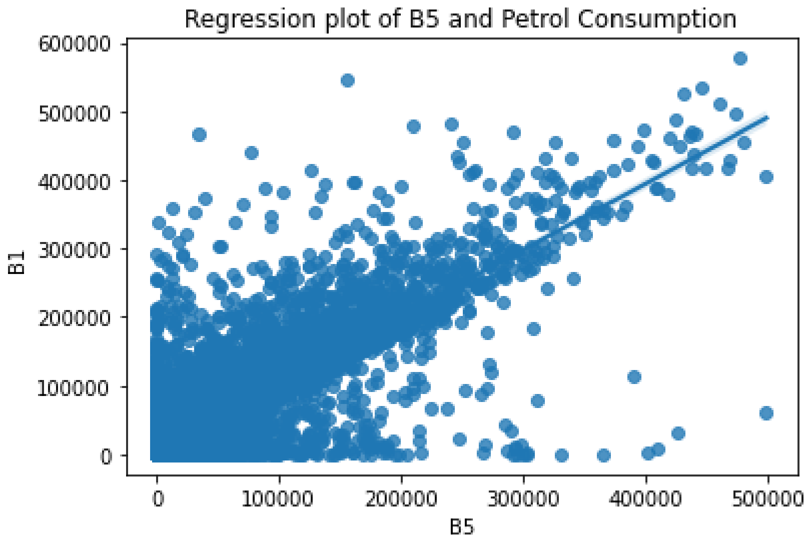
In [66]:

```
import seaborn as sns # Convention alias for Seaborn

variables = ['B2','B3','B4','B5']

for var in variables:
    plt.figure() # Creating a rectangle (figure) for each plot
    # Regression Plot also by default includes
    # best-fitting regression line
    # which can be turned off via `fit_reg=False`
    sns.regplot(x=var, y='B1', data=df).set(title=f'Regression plot of {var} and Petrol Con
```






Regression plot of B4 and Petrol Consumption
B1
B4
0
100000
200000
300000
400000
500000
600000


Regression plot of B5 and Petrol Consumption
B1
B5
0
100000
200000
300000
400000
500000
600000