CREDIT SCORE PREDICTION WITH LINEAR REGRESSION

- Exploratory Analysis and Graphics
- Treatment of missing data
- Treatment of outliers
- OneHotEncoding
- Attribute Engineering
- Data processing
- Data normalization
- Creation, testing and validation of a machine learning model

In [1]:

```
# Python Version
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
```

```
Versão da Linguagem Python Usada Neste Jupyter Notebook: 3.9.7
```

In [2]:

```
import sys
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

```
C:\Users\User\anaconda3\python.exe
3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=9, micro=7, releaselevel='final', serial=0)
```

In [3]:

```
# The first step is to import the packages we're going to use.

# Note: Python packages are "group of resources" available in the tool.

# Pandas: It has numerous functions and commands to import files, analyze and manage data etc.
import pandas as pd

# Matplotlib: It has a series of functions and commands for displaying graphs
import matplotlib.pyplot as plt

# Seaborn: It has a series of functions and commands for displaying graphs (More robust visualizations than Matplotlib)
import seaborn as sns

# Numpy: It has a series of functions and commands to work with numbers in general (formatting, calculations, etc.)
import numpy as np

# Time: It has a number of time functions
import time  as time

# Warnings: It has details about the warnings and alerts that appear, but we can also use it so that alerts for
# future updates and old methods are not displayed
import warnings
warnings.filterwarnings("ignore")


from sklearn.model_selection import train_test_split # Used to separate training and test data
from sklearn.preprocessing import StandardScaler # Used to normalize the data
from sklearn.preprocessing import MinMaxScaler # Used to normalize the data
from sklearn.preprocessing import LabelEncoder # Used to do OneHotEncoding
from sklearn.linear_model import LinearRegression # Linear Regression Algorithm
from sklearn.metrics import r2_score # Used to measure the accuracy of the predictive model


# Command to display all columns on the file
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

In [4]:

```
# Versions of packages used in this jupyter notebook
%reload_ext watermark
%watermark -a "Score Credit" --iversions
```

```
Author: Score Credit

numpy     : 1.22.3
matplotlib: 3.4.3
pandas    : 1.4.2
seaborn   : 0.11.2
sys       : 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
```

In [5]:

```
# the scikit-learn version
import sklearn
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 0.24.2.
```

Importação dos dados, Analise Exploratória e Tratamento de Dados

In [6]:

```
# Command used to load the file and store it as a Pandas DataFrame
# A Pandas DataFrame is like an Excel spreadsheet, where we can manage lines and columns.
# If the excel file is not in the same directory as the script, the path must be specified.
df_dados = pd.read_excel("dados_credito2.xlsx")
```

In [7]:

```
# Command used to check the number of lines and columns in the file
# Columns are also called variables.
df_dados.shape
```

Out[7]:

```
(10476, 17)
```

In [8]:

```
# Command used to check the initial lines of the DataFrame
df_dados.head()
```

Out[8]:

	CODIGO_CLIENTE	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA
0	1	SP	19	Superior Cursando	Solteiro	0	Não	0	0	Não
1	2	MG	23	Superior Completo	Solteiro	1	Não	0	0	Não
2	3	SC	25	Segundo Grau Completo	Casado	0	Sim	1	220000	Não
3	4	PR	27	Superior Cursando	Casado	1	Sim	0	0	Não
4	5	RJ	28	Superior Completo	Divorciado	2	Não	1	370000	Não

In [9]:

```
# Command used to check the final lines of the DataFrame
df_dados.tail()
```

Out[9]:

	CODIGO_CLIENTE	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RE
10471	10472	PR	51	Superior Completo	Solteiro	1	Não	0	0	
10472	10473	SP	48	Segundo Grau Completo	Casado	0	Sim	1	220000	
10473	10474	RJ	51	Superior Cursando	Casado	1	Sim	0	0	
10474	10475	RJ	48	Superior Completo	Divorciado	2	Não	1	370000	
10475	10476	PR	51	Segundo Grau Completo	Divorciado	0	Não	0	0	

In [10]:

```
#Command used to check information about the data (Variable types, Variables, Number of records, etc.)

# The variable CODIGO_CLIENTE can be excluded
# The variables  UF, ESCOLARIDADE, CASA_PROPRIA, OUTRA_RENDA, TRABALHANDO_ATUALMENTE and ESTADO_CIVIL --> OneHotEncoding
# The variable  ULTIMO_SALARIO is as a STRING and needs to be NUMERIC

df_dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10476 entries, 0 to 10475
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   CODIGO_CLIENTE              10476 non-null  int64  
 1   UF                          10476 non-null  object 
 2   IDADE                       10476 non-null  int64  
 3   ESCOLARIDADE                10476 non-null  object 
 4   ESTADO_CIVIL                10476 non-null  object 
 5   QT_FILHOS                   10476 non-null  int64  
 6   CASA_PROPRIA                10476 non-null  object 
 7   QT_IMOVEIS                  10476 non-null  int64  
 8   VL_IMOVEIS                  10476 non-null  int64  
 9   OUTRA_RENDA                 10476 non-null  object 
 10  OUTRA_RENDA_VALOR           10476 non-null  int64  
 11  TEMPO_ULTIMO_EMPREGO_MESES  10476 non-null  int64  
 12  TRABALHANDO_ATUALMENTE      10476 non-null  object 
 13  ULTIMO_SALARIO              10474 non-null  object 
 14  QT_CARROS                   10476 non-null  int64  
 15  VALOR_TABELA_CARROS         10476 non-null  int64  
 16  SCORE                       10476 non-null  float64
dtypes: float64(1), int64(9), object(7)
memory usage: 1.4+ MB
```

In [11]:

```
# Let's delete the CODIGO_CLIENTE variable
df_dados.drop('CODIGO_CLIENTE', axis=1, inplace=True)
```

In [12]:

```
# Command used to evaluate if any variable has null value or called values missing or NAN (Not Available)
# The variable ULTIMO_SALARIO has NULL("SEM DADOS") values and we will need to manage it
df_dados.groupby(['ULTIMO_SALARIO']).size()
```

Out[12]:

```
ULTIMO_SALARIO
1800         846
2200         792
3100         792
3900         792
4500         468
4800         792
5300         522
6100         522
6800         611
9000         522
9800         468
11500        790
13000        522
15000        522
17500        522
18300        522
22000        468
SEM DADOS      1
dtype: int64
```

In [13]:

```
# Here we could solve it in two ways.
# The first way would be to delete the entire record, but we would be losing data.
#df_dados.drop(df_dados.loc[df_dados['VALOR']=='SEM VALOR'].index, inplace=True)


# The second way would be to check the mean or median value of this model and substitute the word NO VALUE for an average value
df_dados.loc[df_dados['ULTIMO_SALARIO'] == 'SEM DADOS']
```

Out[13]:

	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA	OUTRA_REND
10459	RJ	45	Superior Cursando	Solteiro	1	Sim	1	185000	Sim	

In [14]:

```
# Now we replace the word NO VALUE with a NULL value
df_dados.replace('SEM DADOS',np.nan, inplace = True)
```

In [15]:

```
# Then convert the field to a float
df_dados['ULTIMO_SALARIO'] = df_dados['ULTIMO_SALARIO'].astype(np.float64)
```

In [16]:

```
# Command used to evaluate if any variable has null value or called values missing or NAN (Not Available)
# The variable ULTIMO_SALARIO has NULL values and we will need to treat them
df_dados.isnull().sum()
```

Out[16]:

```
UF                            0
IDADE                         0
ESCOLARIDADE                  0
ESTADO_CIVIL                  0
QT_FILHOS                     0
CASA_PROPRIA                  0
QT_IMOVEIS                    0
VL_IMOVEIS                    0
OUTRA_RENDA                   0
OUTRA_RENDA_VALOR             0
TEMPO_ULTIMO_EMPREGO_MESES    0
TRABALHANDO_ATUALMENTE        0
ULTIMO_SALARIO                3
QT_CARROS                     0
VALOR_TABELA_CARROS           0
SCORE                         0
dtype: int64
```

In [17]:

```
# Here we update the value according to the median of that model
df_dados['ULTIMO_SALARIO'] = df_dados['ULTIMO_SALARIO'].fillna((df_dados['ULTIMO_SALARIO'].median()))
```

In [18]:

```
# Let's confirm that there are no null values left
df_dados.isnull().sum()
```

Out[18]:

```
UF                            0
IDADE                         0
ESCOLARIDADE                  0
ESTADO_CIVIL                  0
QT_FILHOS                     0
CASA_PROPRIA                  0
QT_IMOVEIS                    0
VL_IMOVEIS                    0
OUTRA_RENDA                   0
OUTRA_RENDA_VALOR             0
TEMPO_ULTIMO_EMPREGO_MESES    0
TRABALHANDO_ATUALMENTE        0
ULTIMO_SALARIO                0
QT_CARROS                     0
VALOR_TABELA_CARROS           0
SCORE                         0
dtype: int64
```

In [19]:

```
# Let's evaluate the types of the variables again
df_dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10476 entries, 0 to 10475
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   UF                          10476 non-null  object
 1   IDADE                       10476 non-null  int64
 2   ESCOLARIDADE                10476 non-null  object
 3   ESTADO_CIVIL                10476 non-null  object
 4   QT_FILHOS                   10476 non-null  int64
 5   CASA_PROPRIA                10476 non-null  object
 6   QT_IMOVEIS                  10476 non-null  int64
 7   VL_IMOVEIS                  10476 non-null  int64
 8   OUTRA_RENDA                 10476 non-null  object
 9   OUTRA_RENDA_VALOR           10476 non-null  int64
 10  TEMPO_ULTIMO_EMPREGO_MESES  10476 non-null  int64
 11  TRABALHANDO_ATUALMENTE      10476 non-null  object
 12  ULTIMO_SALARIO              10476 non-null  float64
 13  QT_CARROS                   10476 non-null  int64
 14  VALOR_TABELA_CARROS         10476 non-null  int64
 15  SCORE                       10476 non-null  float64
dtypes: float64(2), int64(8), object(6)
memory usage: 1.3+ MB
```

In [20]:

```
# Let's evaluate some basic statistical measures
df_dados.describe()
```

Out[20]:

	IDADE	QT_FILHOS	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA_VALOR	TEMPO_ULTIMO_EMPREGO_MESES	ULTIMO_SALAR
count	10476.000000	10476.000000	10476.000000	10476.000000	10476.000000	10476.000000	10476.0000
mean	41.054124	1.122566	0.847079	238453.608247	641.237113	43.070447	8286.5311
std	13.878162	1.113537	0.957374	265843.934416	1295.978195	40.851521	5826.5897
min	19.000000	0.000000	0.000000	0.000000	0.000000	8.000000	1800.0000
25%	28.000000	0.000000	0.000000	0.000000	0.000000	14.000000	3900.0000
50%	42.000000	1.000000	1.000000	185000.000000	0.000000	22.000000	6100.0000
75%	53.000000	2.000000	1.000000	370000.000000	0.000000	75.000000	11500.0000
max	65.000000	42.000000	3.000000	900000.000000	4000.000000	150.000000	22000.0000

In [21]:

```
# Now we will evaluate the outliers of the columns that are numeric
# OUTLIERS are outliers that are far above or far below the other values

# Let's load in a list the variables that are of type INT64 AND FLOAT64
variaveis_numericas = []
for i in df_dados.columns[0:16].tolist():
        if df_dados.dtypes[i] == 'int64' or df_dados.dtypes[i] == 'float64':
            print(i, ':' , df_dados.dtypes[i])
            variaveis_numericas.append(i)
```

```
IDADE : int64
QT_FILHOS : int64
QT_IMOVEIS : int64
VL_IMOVEIS : int64
OUTRA_RENDA_VALOR : int64
TEMPO_ULTIMO_EMPREGO_MESES : int64
ULTIMO_SALARIO : float64
QT_CARROS : int64
VALOR_TABELA_CARROS : int64
SCORE : float64
```

In [22]:

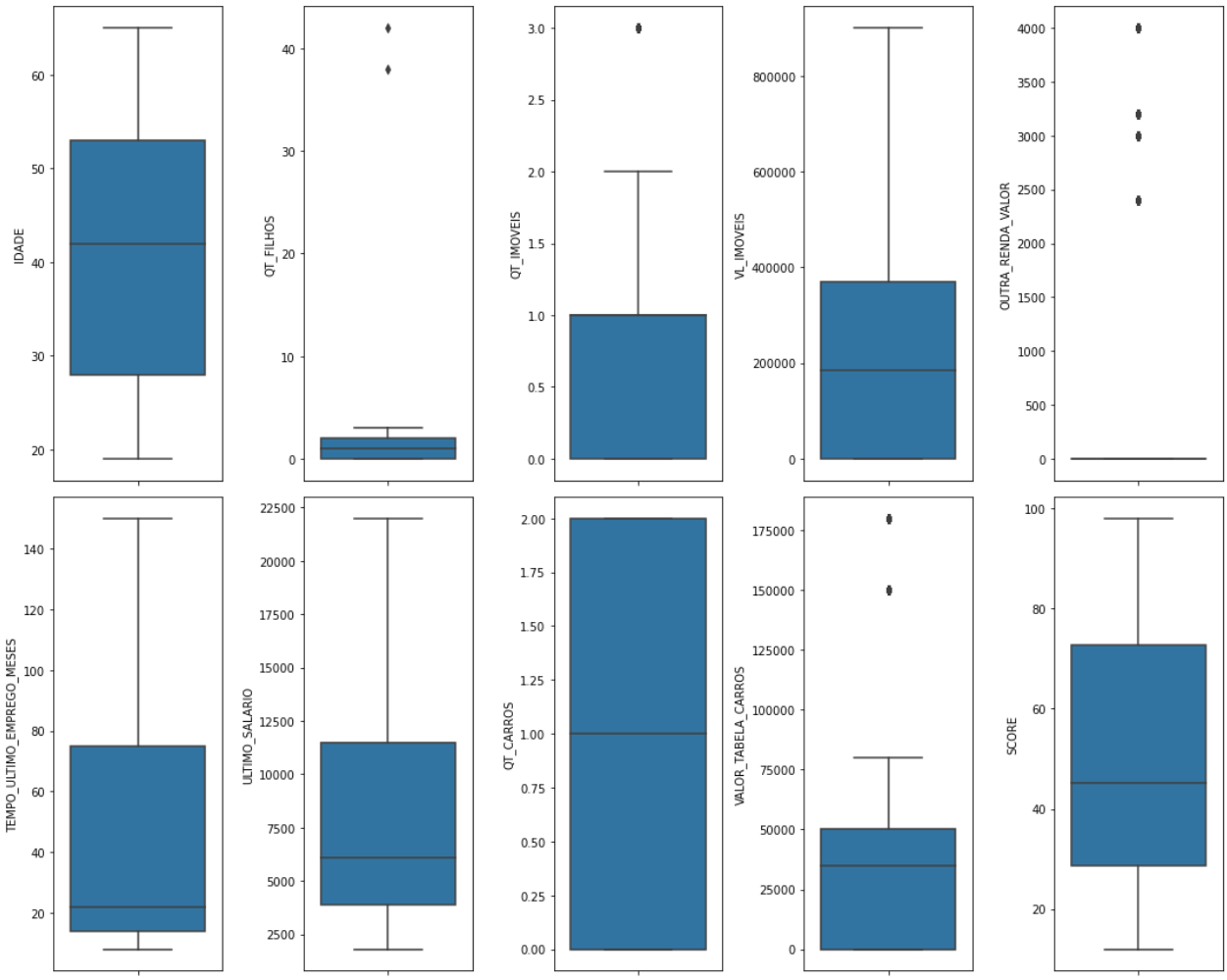
```
# Let's look at the list of variables and evaluate whether these variables have outliers through a boxplot
variaveis_numericas
```

Out[22]:

```
['IDADE',
 'QT_FILHOS',
 'QT_IMOVEIS',
 'VL_IMOVEIS',
 'OUTRA_RENDA_VALOR',
 'TEMPO_ULTIMO_EMPREGO_MESES',
 'ULTIMO_SALARIO',
 'QT_CARROS',
 'VALOR_TABELA_CARROS',
 'SCORE']
```

In [23]:

```
# With this command we will display all graphs of all columns at once to facilitate our analysis.

# Here we define the screen size for displaying the graphs
plt.rcParams["figure.figsize"] = [15.00, 12.00]
plt.rcParams["figure.autolayout"] = True

# Here we define in how many rows and columns we want to display the graphs
f, axes = plt.subplots(2, 5) #2 linhas e 5 colunas

linha = 0
coluna = 0

for i in variaveis_numericas:
    sns.boxplot(data = df_dados, y=i, ax=axes[linha][coluna])
    coluna += 1
    if coluna == 5:
        linha += 1
        coluna = 0

plt.show()
```



In [24]:

```
# Now we know that we have possible OUTLIERS in the variables QT_FILHOS, QT_IMOVEIS, VALOR_TABELA_CARROS and OUTRA_RENDA_VALOR
# Let's look at what these outliers are to evaluate how we will treat them.

# Let's list the number of children greater than 4
# Since we only have 2 records that are really outliers, we will exclude them
df_dados.loc[df_dados['QT_FILHOS'] > 4]
```

Out[24]:

	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA	OUTRA_REND
27	SP	48	Superior Completo	Divorciado	38	Sim	2	600000	Não	
10455	SP	45	Segundo Grau Completo	Casado	42	Sim	1	220000	Não	

In [25]:

```
# In this example we will delete the entire record.
df_dados.drop(df_dados.loc[df_dados['QT_FILHOS'] > 4].index, inplace=True)
```

In [26]:

```
# Let's evaluate the other variables with possible outliers
```

In [27]:

```
# We don't need to change anything
df_dados.groupby(['OUTRA_RENDA_VALOR']).size()
```

Out[27]:

```
OUTRA_RENDA_VALOR
0       8350
2400     468
3000     612
3200     522
4000     522
dtype: int64
```

In [28]:

```
# We don't need to change anything
df_dados.groupby(['VALOR_TABELA_CARROS']).size()
```

Out[28]:

```
VALOR_TABELA_CARROS
0         3762
28000      468
30000      791
35000      792
40000      792
48000      522
50000     1314
70000      521
80000      522
150000     468
180000     522
dtype: int64
```

In [29]:

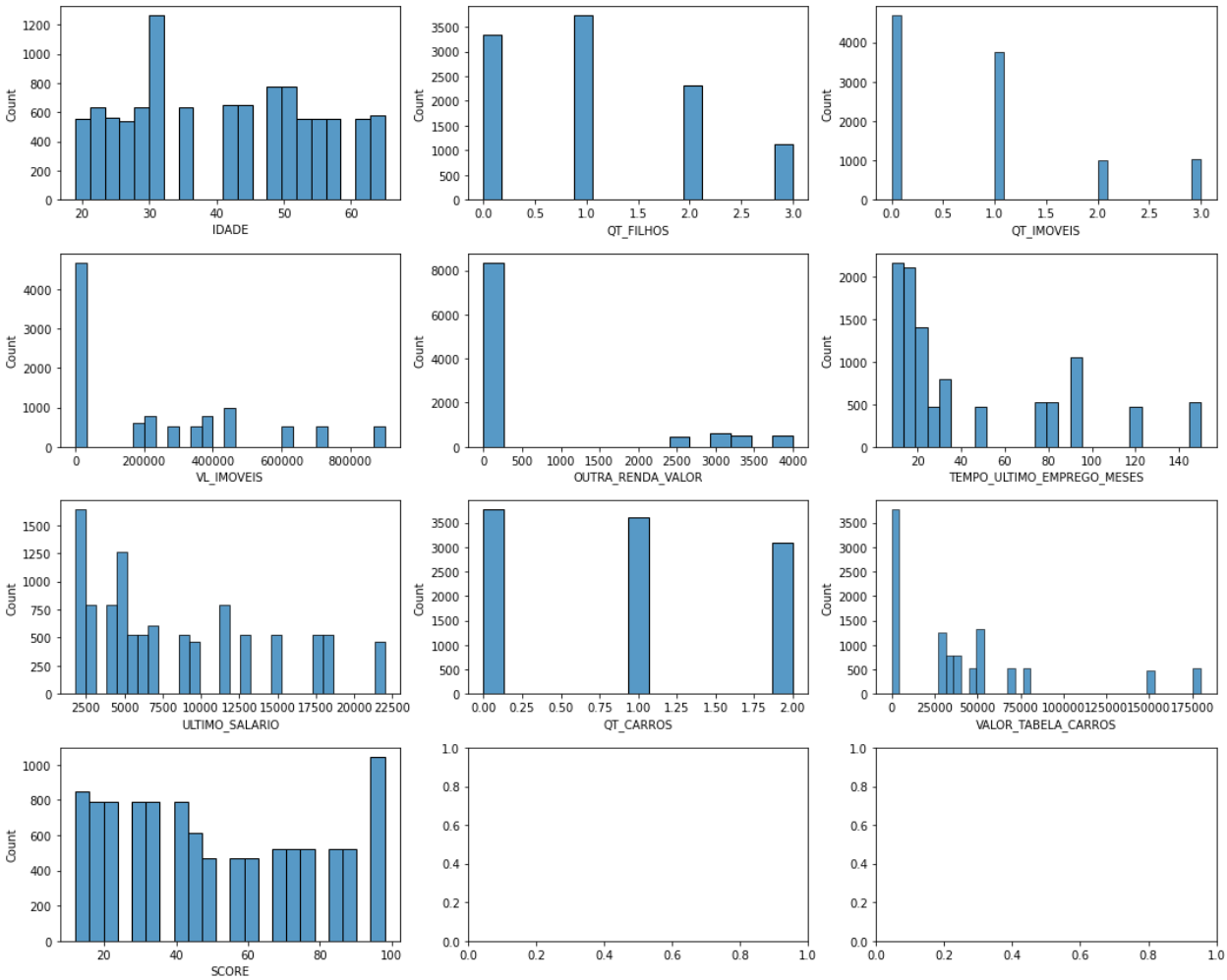
```
# We don't need to change anything
df_dados.groupby(['QT_IMOVEIS']).size()
```

Out[29]:

```
QT_IMOVEIS
0    4680
1    3761
2     989
3    1044
dtype: int64
```

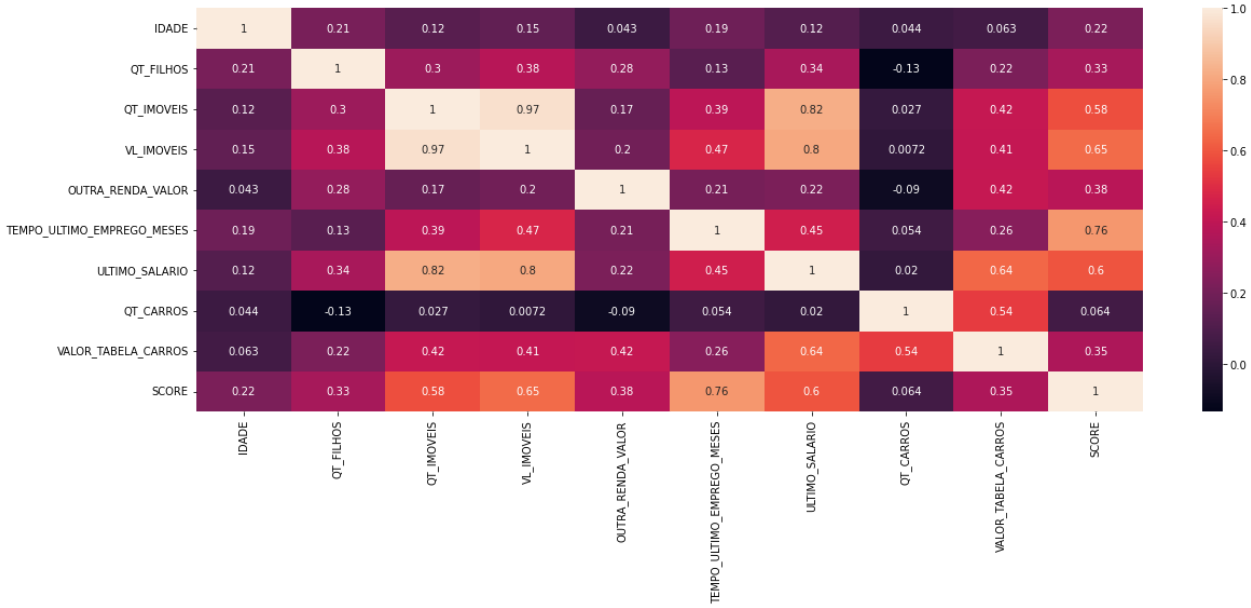
In [30]:

```
# Let's generate a histogram graph to evaluate the data distribution
# We can see that in this case the data is well dispersed

# Here we define the screen size for displaying the graphs
plt.rcParams["figure.figsize"] = [15.00, 12.00]
plt.rcParams["figure.autolayout"] = True

# Here we define in how many rows and columns we want to display the graphs
f, axes = plt.subplots(4, 3) #4 lines e 3 columns

linha = 0
coluna = 0

for i in variaveis_numericas:
    sns.histplot(data = df_dados, x=i, ax=axes[linha][coluna])
    coluna += 1
    if coluna == 3:
        linha += 1
        coluna = 0

plt.show()
```



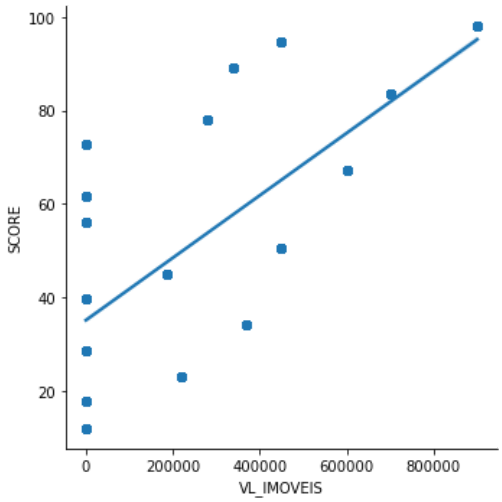
In [31]:

```
# Through the hitmap we can observe the correlation between all variables.
plt.rcParams["figure.figsize"] = (18,8)
ax = sns.heatmap(df_dados.corr(), annot=True)
```



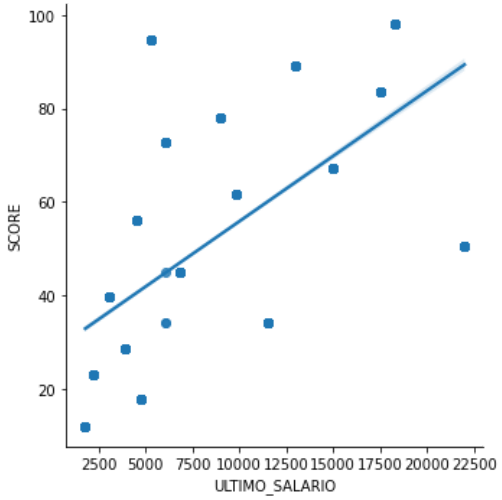
In [32]:

```
# Let's look at a scatter plot to evaluate the correlation of some variables
sns.lmplot(x = "VL_IMOVEIS", y = "SCORE", data = df_dados);
```



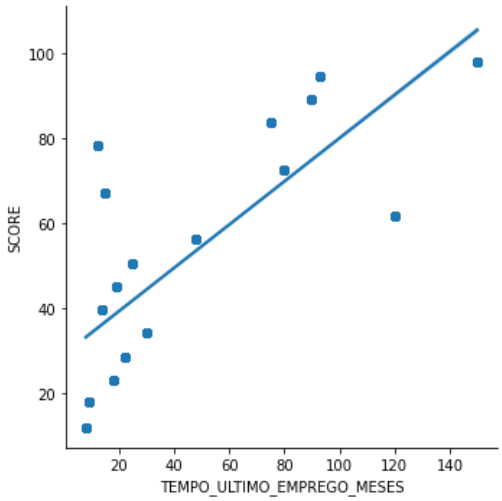
In [33]:

```
# Let's look at a scatter plot to evaluate the correlation of some variables
sns.lmplot(x = "ULTIMO_SALARIO", y = "SCORE", data = df_dados);
```



In [34]:

```
# Let's look at a scatter plot to evaluate the correlation of some variables
sns.lmplot(x = "TEMPO_ULTIMO_EMPREGO_MESES", y = "SCORE", data = df_dados);
```



In [35]:

```
# Let's do an attribute engineering in the AGE field and create a new Age Range field
print('Menor Idade: ', df_dados['IDADE'].min())
print('Maior Idade: ', df_dados['IDADE'].max())
```

```
Menor Idade:  19
Maior Idade:  65
```

In [36]:

```
# Attribute Engineering - We will create a new variable
idade_bins = [0, 30, 40, 50, 60]
idade_categoria = ["Até 30", "31 a 40", "41 a 50", "Maior que 50"]

df_dados["FAIXA_ETARIA"] = pd.cut(df_dados["IDADE"], idade_bins, labels=idade_categoria)

df_dados["FAIXA_ETARIA"].value_counts()
```

Out[36]:

```
Até 30          3552
Maior que 50    2448
41 a 50         2070
31 a 40         1270
Name: FAIXA_ETARIA, dtype: int64
```

In [37]:

```
# Let's evaluate the average score by age group
df_dados.groupby(["FAIXA_ETARIA"]).mean()["SCORE"]
```

Out[37]:

```
FAIXA_ETARIA
Até 30          44.762950
31 a 40         48.883202
41 a 50         51.440177
Maior que 50    56.123775
Name: SCORE, dtype: float64
```

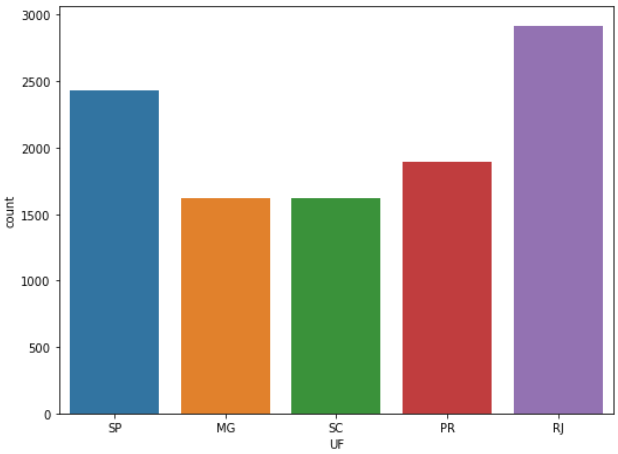
In [38]:

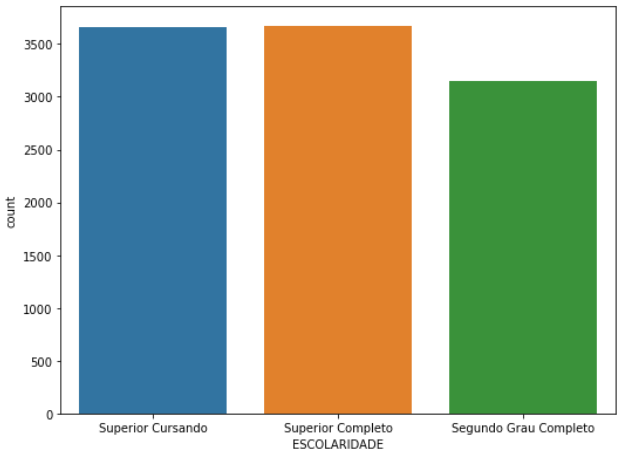
```
variaveis_categoricas = []
for i in df_dados.columns[0:48].tolist():
        if df_dados.dtypes[i] == 'object' or df_dados.dtypes[i] == 'category':
            print(i, ':' , df_dados.dtypes[i])
            variaveis_categoricas.append(i)
```

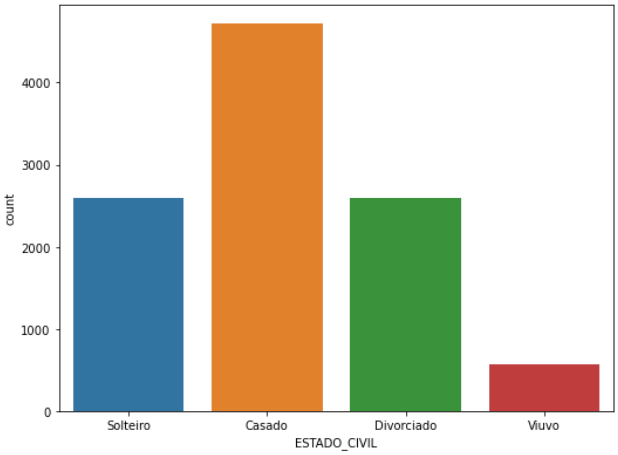
```
UF : object
ESCOLARIDADE : object
ESTADO_CIVIL : object
CASA_PROPRIA : object
OUTRA_RENDA : object
TRABALHANDO_ATUALMENTE : object
FAIXA_ETARIA : category
```

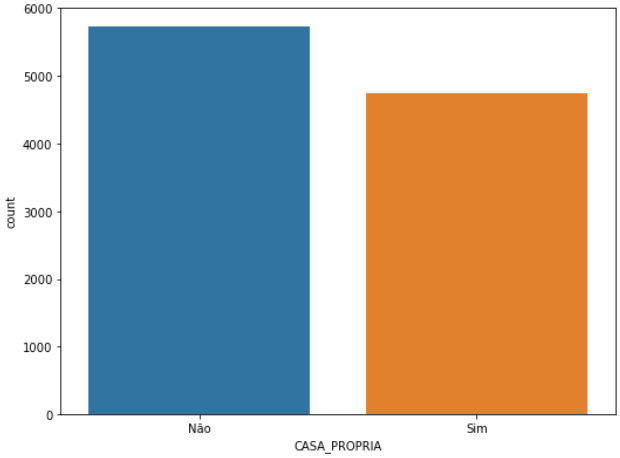
In [39]:

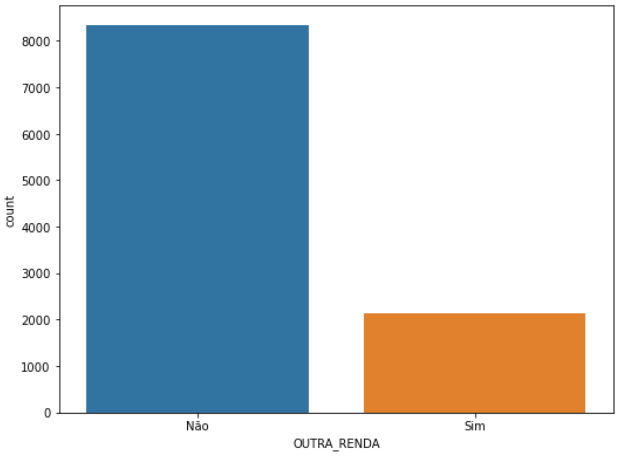
```
# With this command we will display all graphs of all columns at once to facilitate our analysis.

# Here we define the screen size for displaying the graphs
plt.rcParams["figure.figsize"] = [15.00, 22.00]
plt.rcParams["figure.autolayout"] = True

# Here we define in how many rows and columns we want to display the graphs
f, axes = plt.subplots(4, 2) #3 linhas e 2 colunas

linha = 0
coluna = 0

for i in variaveis_categoricas:
    sns.countplot(data = df_dados, x=i, ax=axes[linha][coluna])

    coluna += 1
    if coluna == 2:
        linha += 1
        coluna = 0

plt.show()
```

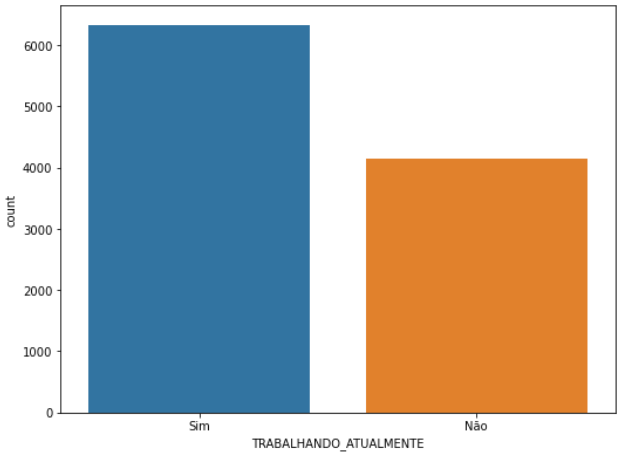

count
UF
SP
MG
SC
PR
RJ

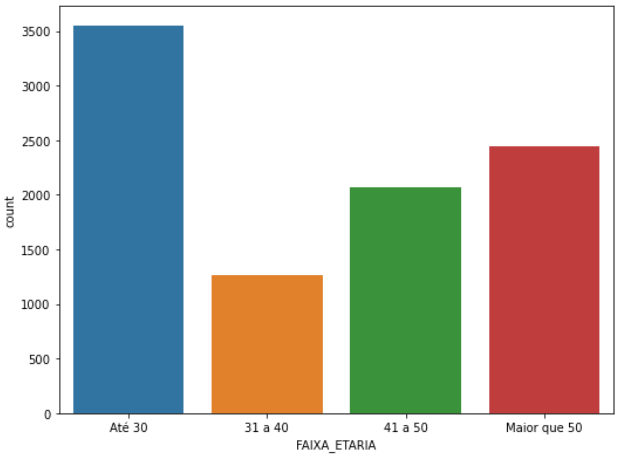

count
ESCOLARIDADE
Superior Cursando
Superior Completo
Segundo Grau Completo

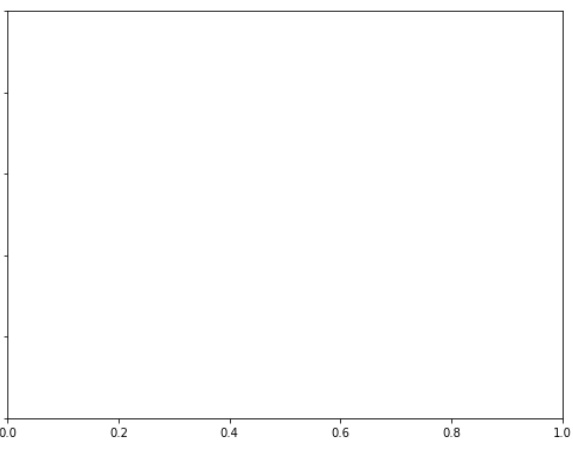

count
ESTADO_CIVIL
Solteiro
Casado
Divorciado
Viuvo


count
CASA_PROPRIA
Não
Sim


count
OUTRA_RENDA
Não
Sim


count
TRABALHANDO_ATUALMENTE
Sim
Não


count
FAIXA_ETARIA
Até 30
31 a 40
41 a 50
Maior que 50



Data Pre-Processing

In [40]:

```
# Create the encoder
lb = LabelEncoder()

# Apply the encoder to the variables that are with string
df_dados['FAIXA_ETARIA'] = lb.fit_transform(df_dados['FAIXA_ETARIA'])
df_dados['OUTRA_RENDA'] = lb.fit_transform(df_dados['OUTRA_RENDA'])
df_dados['TRABALHANDO_ATUALMENTE'] = lb.fit_transform(df_dados['TRABALHANDO_ATUALMENTE'])
df_dados['ESTADO_CIVIL'] = lb.fit_transform(df_dados['ESTADO_CIVIL'])
df_dados['CASA_PROPRIA'] = lb.fit_transform(df_dados['CASA_PROPRIA'])
df_dados['ESCOLARIDADE'] = lb.fit_transform(df_dados['ESCOLARIDADE'])
df_dados['UF'] = lb.fit_transform(df_dados['UF'])

# Remove missing values eventually generated
df_dados.dropna(inplace = True)
```

In [41]:

```
df_dados.head(5)
```

Out[41]:

	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA	OUTRA_RENDA_VA
0	4	19	2	2	0	0	0	0	0	
1	0	23	1	2	1	0	0	0	0	
2	3	25	0	0	0	1	1	220000	0	
3	1	27	2	0	1	1	0	0	0	
4	2	28	1	1	2	0	1	370000	0	

In [42]:

```
# Now we can see that we already have all numerical variables
df_dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10474 entries, 0 to 10475
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   UF                          10474 non-null  int32
 1   IDADE                       10474 non-null  int64
 2   ESCOLARIDADE                10474 non-null  int32
 3   ESTADO_CIVIL                10474 non-null  int32
 4   QT_FILHOS                   10474 non-null  int64
 5   CASA_PROPRIA                10474 non-null  int32
 6   QT_IMOVEIS                  10474 non-null  int64
 7   VL_IMOVEIS                  10474 non-null  int64
 8   OUTRA_RENDA                 10474 non-null  int32
 9   OUTRA_RENDA_VALOR           10474 non-null  int64
 10  TEMPO_ULTIMO_EMPREGO_MESES  10474 non-null  int64
 11  TRABALHANDO_ATUALMENTE      10474 non-null  int32
 12  ULTIMO_SALARIO              10474 non-null  float64
 13  QT_CARROS                   10474 non-null  int64
 14  VALOR_TABELA_CARROS         10474 non-null  int64
 15  SCORE                       10474 non-null  float64
 16  FAIXA_ETARIA                10474 non-null  int32
dtypes: float64(2), int32(7), int64(8)
memory usage: 1.4 MB
```

In [43]:

```
# Separating the target variable
target = df_dados.iloc[:,15:16]
```

In [44]:

```
# Separating the predictor variables

preditoras = df_dados.copy() # Making a copy of the dataframe

del preditoras['SCORE'] # Excluding the target variable, as we already separated it in the previous step

preditoras.head()# Visualizing the predictor variables
```

Out[44]:

	UF	IDADE	ESCOLARIDADE	ESTADO_CIVIL	QT_FILHOS	CASA_PROPRIA	QT_IMOVEIS	VL_IMOVEIS	OUTRA_RENDA	OUTRA_RENDA_VA
0	4	19	2	2	0	0	0	0	0	
1	0	23	1	2	1	0	0	0	0	
2	3	25	0	0	0	1	1	220000	0	
3	1	27	2	0	1	1	0	0	0	
4	2	28	1	1	2	0	1	370000	0	

In [45]:

```
# Divisão em Dados de Treino e Teste.
X_treino, X_teste, y_treino, y_teste = train_test_split(preditoras, target, test_size = 0.3, random_state = 40)
```

In [46]:

```
# Let's apply the normalization in training and testing
# Standardization
sc = MinMaxScaler()
X_treino_normalizados = sc.fit_transform(X_treino)
X_teste_normalizados = sc.transform(X_teste)
```

Create, evaluate and test our predictive model

In [47]:

```
# Train the model
modelo = LinearRegression(normalize = True, fit_intercept = True)

modelo = modelo.fit(X_treino_normalizados, y_treino)
```

In [48]:

```
r2_score(y_teste, modelo.fit(X_treino_normalizados, y_treino).predict(X_teste_normalizados))
```

Out[48]:

```
0.7984013631162862
```

In [49]:

```
UF = 2
IDADE = 42
ESCOLARIDADE = 1
ESTADO_CIVIL = 2
QT_FILHOS = 1
CASA_PROPRIA = 1
QT_IMOVEIS = 1
VL_IMOVEIS = 300000
OUTRA_RENDA = 1
OUTRA_RENDA_VALOR = 2000
TEMPO_ULTIMO_EMPREGO_MESES = 18
TRABALHANDO_ATUALMENTE = 1
ULTIMO_SALARIO = 5400.0
QT_CARROS = 4
VALOR_TABELA_CARROS = 70000
FAIXA_ETARIA = 3

novos_dados = [UF, IDADE, ESCOLARIDADE, ESTADO_CIVIL, QT_FILHOS,CASA_PROPRIA,QT_IMOVEIS,VL_IMOVEIS,OUTRA_RENDA,
               OUTRA_RENDA_VALOR,TEMPO_ULTIMO_EMPREGO_MESES,TRABALHANDO_ATUALMENTE,ULTIMO_SALARIO,QT_CARROS,
               VALOR_TABELA_CARROS, FAIXA_ETARIA]


# Reshape
X = np.array(novos_dados).reshape(1, -1)
X = sc.transform(X)

# Prevision
print("Score de crédito previsto para esse cliente:", modelo.predict(X))
```

```
Score de crédito previsto para esse cliente: [[43.18575662]]
```

In [50]:

```
# by:https://www.youtube.com/@nerddosdados
```