Grupo 20: Thiago e Gabriele

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from sklearn import preprocessing
```

In [2]:

```
df = pd.read_csv('trabalho6_dados_20.csv')
```

Análise inicial

Como observado na tabela abaixo temos 3 colunas numéricas do tipo float, com as seguintes descrições

- **temperatura**: Temperatura ambiente em graus Celsius;
- **vacuo**: Pressão do vapor de escape medida em cm Hg; e
- **energia**: Quantidade de energia produzida em mega watts.

E um total de 11481 registros disponíveis.

In [3]:

```
df
```

Out[3]:

	temperatura	vacuo	energia
0	23.82	44.89	445.45
1	22.72	69.84	436.70
2	12.11	41.17	475.53
3	30.27	64.05	438.68
4	15.23	37.87	464.02
...	...	...	...
11477	24.26	61.02	442.86
11478	21.67	69.71	440.16
11479	11.43	40.22	477.50
11480	27.60	69.05	436.08
11481	5.97	36.25	487.03

11482 rows × 3 columns

Também pode ser observado que há 3 faixas distintas de máximos e mínimos de cada variável.

In [4]:

```
df.describe()
```

Out[4]:

	temperatura	vacuo	energia
count	11482.000000	11482.000000	11482.000000
mean	19.740275	54.433817	454.117745
std	7.437203	12.682335	17.079184
min	1.810000	25.360000	425.120000
25%	13.670000	41.780000	439.520000
50%	20.550000	52.720000	451.015000
75%	25.760000	66.540000	467.957500
max	37.110000	81.560000	495.760000

E por fim é observado que não há dados faltantes nem dados nulos. E que todas as colunas das variáveis possuem o tipo float64

In [5]:

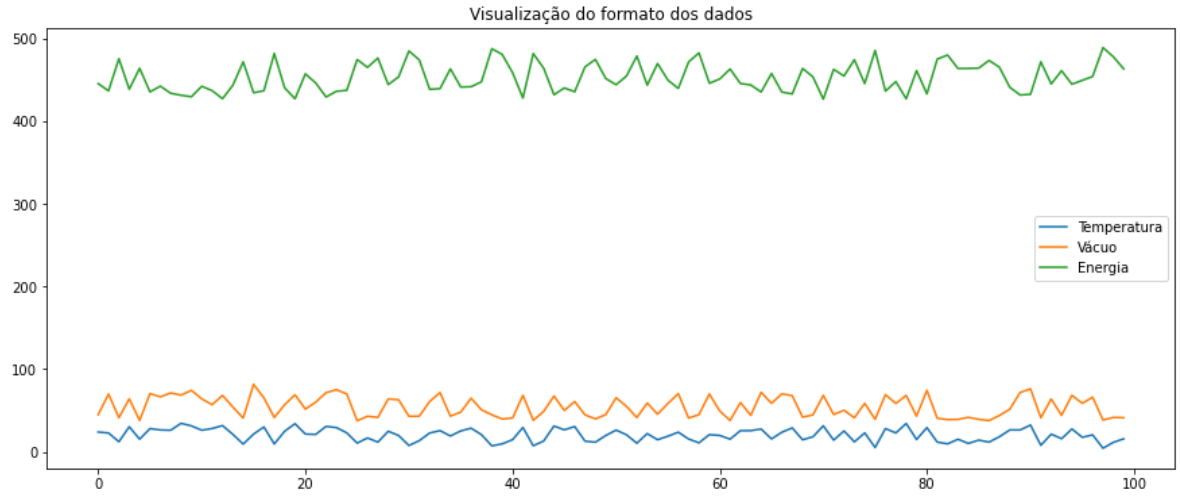
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11482 entries, 0 to 11481
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   temperatura  11482 non-null  float64
 1   vacuo        11482 non-null  float64
 2   energia      11482 non-null  float64
dtypes: float64(3)
memory usage: 269.2 KB
```

Visualização

In [6]:

```
plt.figure(figsize=(15,6))
plt.plot(df.head(100)['temperatura'], label='Temperatura')
plt.plot(df.head(100)['vacuo'], label=u'Vácuo')
plt.plot(df.head(100)['energia'], label='Energia')
plt.title('Visualização do formato dos dados')
plt.legend()
plt.show()
```



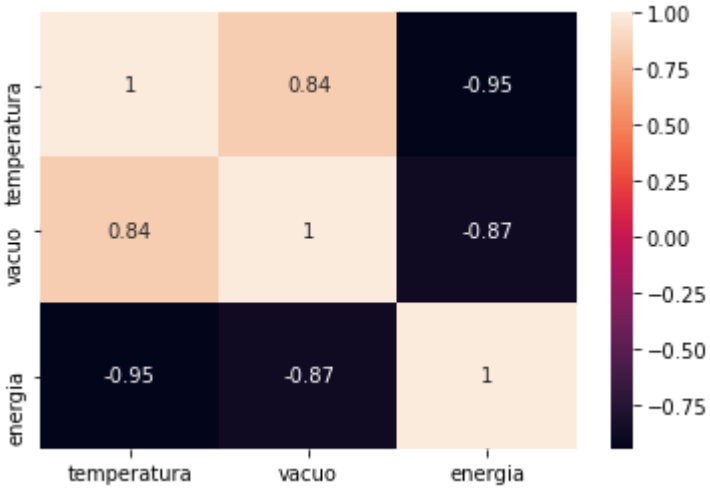
No gráfico acima pode se observar que os dados das colunas Temperatura e Vácuo estão em faixas ligeiramente distintas, enquanto isso a Energia está bem acima das outras duas. Assim, como ambas estão em escalas diferentes é necessária uma normalização para maior eficiência dos algoritmos de regressão.

In [7]:

```
correlacao = df.corr()
sns.heatmap(correlacao, annot=True)
```

Out[7]:

```
<AxesSubplot:>
```



Também podemos observar pela matriz de correlação que as variáveis Temperatura e Vácuo tem forte correlação inversa (Maior que 80%) com a coluna Energia.

Normalização

Nessa etapa os dados são normalizados num intervalo de 0 a 1

In [8]:

```
scaler = preprocessing.MinMaxScaler( feature_range=(0, 1) )
df_norm = pd.DataFrame( scaler.fit_transform(df), columns=df.columns )
df_norm
```

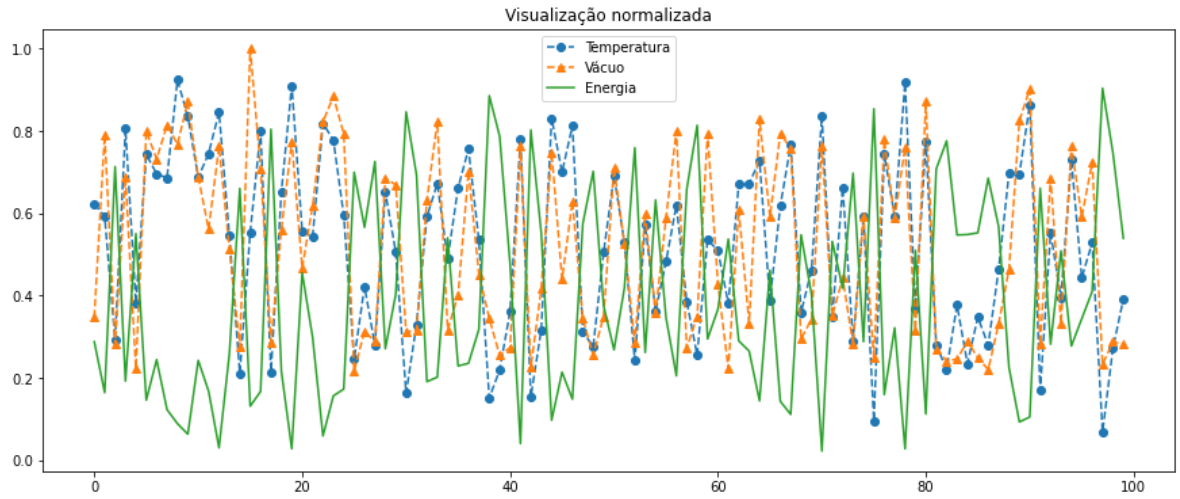
Out[8]:

	temperatura	vacuo	energia
0	0.623513	0.347509	0.287797
1	0.592351	0.791459	0.163930
2	0.291785	0.281317	0.713618
3	0.806232	0.688434	0.191959
4	0.380170	0.222598	0.550680
...	...	...	...
11477	0.635977	0.634520	0.251133
11478	0.562606	0.789146	0.212911
11479	0.272521	0.264413	0.741506
11480	0.730595	0.777402	0.155153
11481	0.117847	0.193772	0.876416

11482 rows × 3 columns

In [9]:

```
plt.figure(figsize=(15,6))
plt.plot(df_norm.head(100)['temperatura'], 'o--', label='Temperatura')
plt.plot(df_norm.head(100)['vacuo'], '^--',label='Vácuo')
plt.plot(df_norm.head(100)['energia'], label='Energia')
plt.title('Visualização normalizada')
plt.legend()
plt.show()
```



Os dados normalizados ficam dentro da mesma escala, facilitando a sua utilização pelos algoritmos de regressão abaixo.

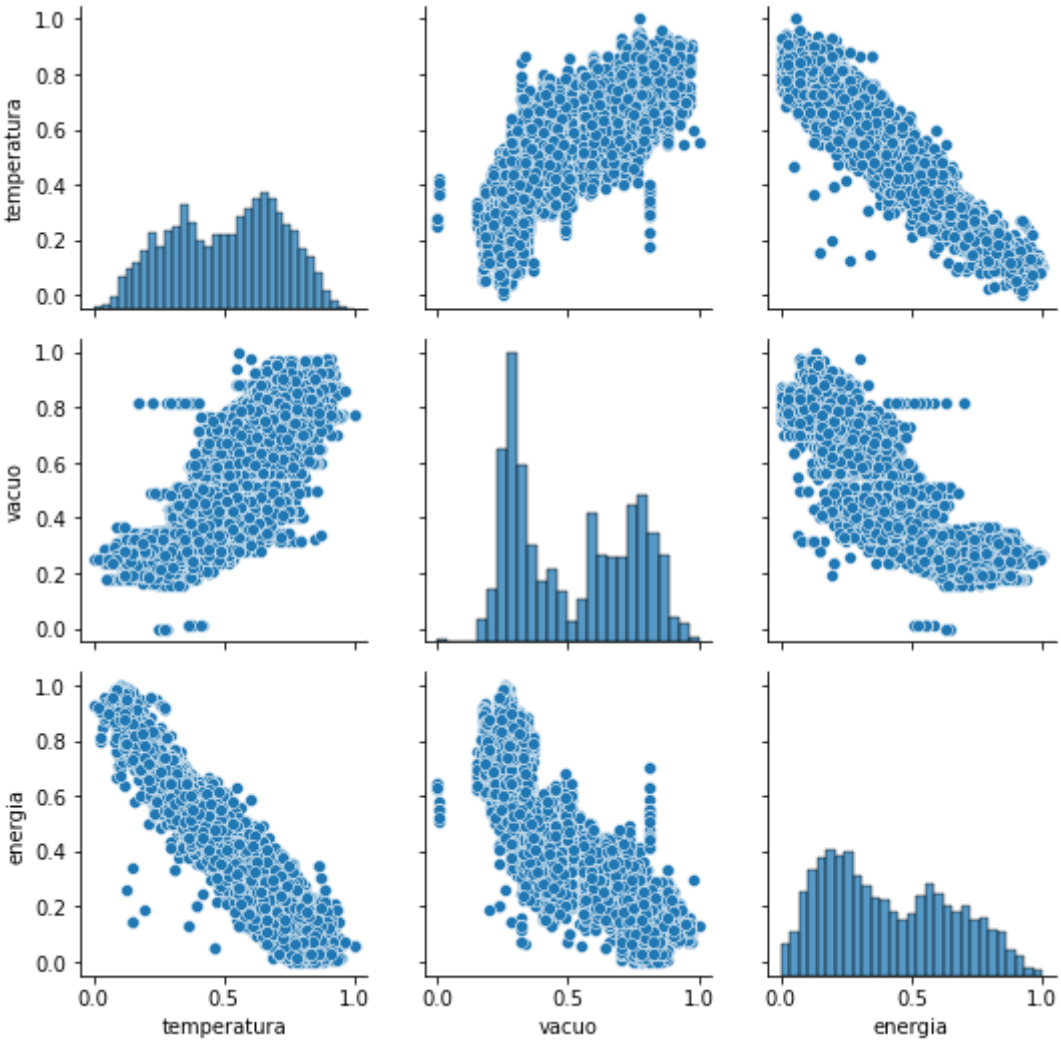
Análise de distribuição

In [10]:

```
g = sns.PairGrid(df_norm)
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend()
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x7f047515f220>
```



Testando Modelos de Regressão

Questões que devem ser levadas em consideração quando escolhemos um modelo para aplicar os dados:

- Tamanho do dataset;
- Acurácia retornada pelo modelo;
- Tempo de processamento;
- Tipo de distribuição;
- Quantidade de variáveis independentes.

Além de escolher o tipo de modelo que se deseja aplicar:

- Não supervisionado;
- Supervisionado;
- Semi-supervisionado.

Separando em conjunto de teste e treinamento

Para este trabalho, escolhemos aplicar métodos supervisionados.

In [33]:

```
from sklearn.model_selection import train_test_split
X = df_norm.drop(columns='energia')
y = df_norm['energia']

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.30,random_state=
```

In [34]:

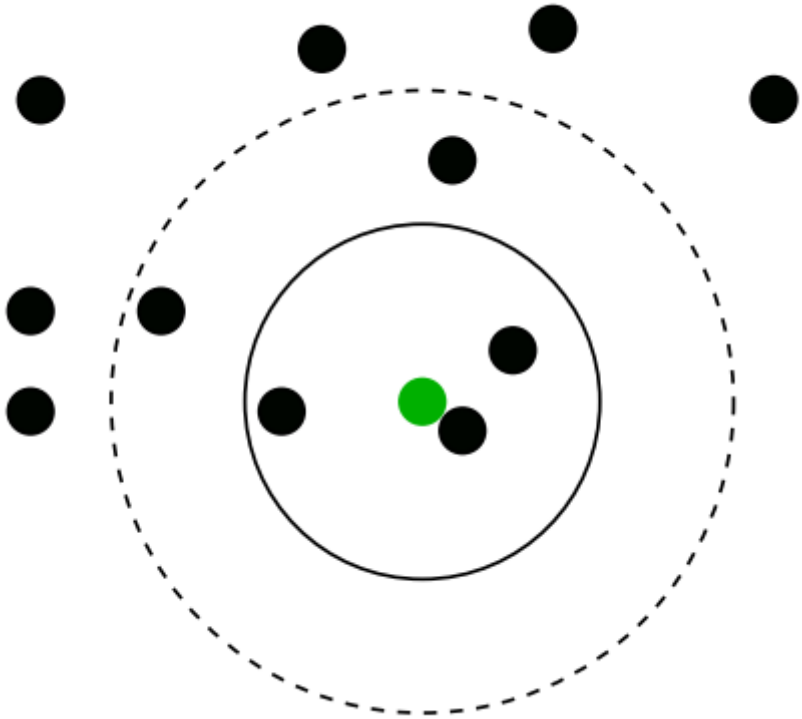
```
#montando o csv
ytest_csv = pd.DataFrame(index=ytest.index,columns=['y_original','RegMult','KNN_tem
ytest_csv['y_original'] = ytest
```

KNN

O algoritmo k-nearest neighbors algorithm é usado em modo regressão e por aprendizado supervisionado.

A ideia básica do funcionamento é que, dado um conjunto de pontos $P(x_i, y_i)$ é montada uma matriz de distâncias entre todos esses pontos. Para avaliação da distância há vários algoritmos disponíveis. No código abaixo é usado algoritmo padrão "minkowski".

Após a criação dessa matriz, um novo ponto $P'(x'_i, y'_i)$ é criado e avaliado usando a matriz de distâncias. Os pontos são ordenados e, no caso da configuração do código abaixo, são extraidos os 5 pontos mais próximos dele.

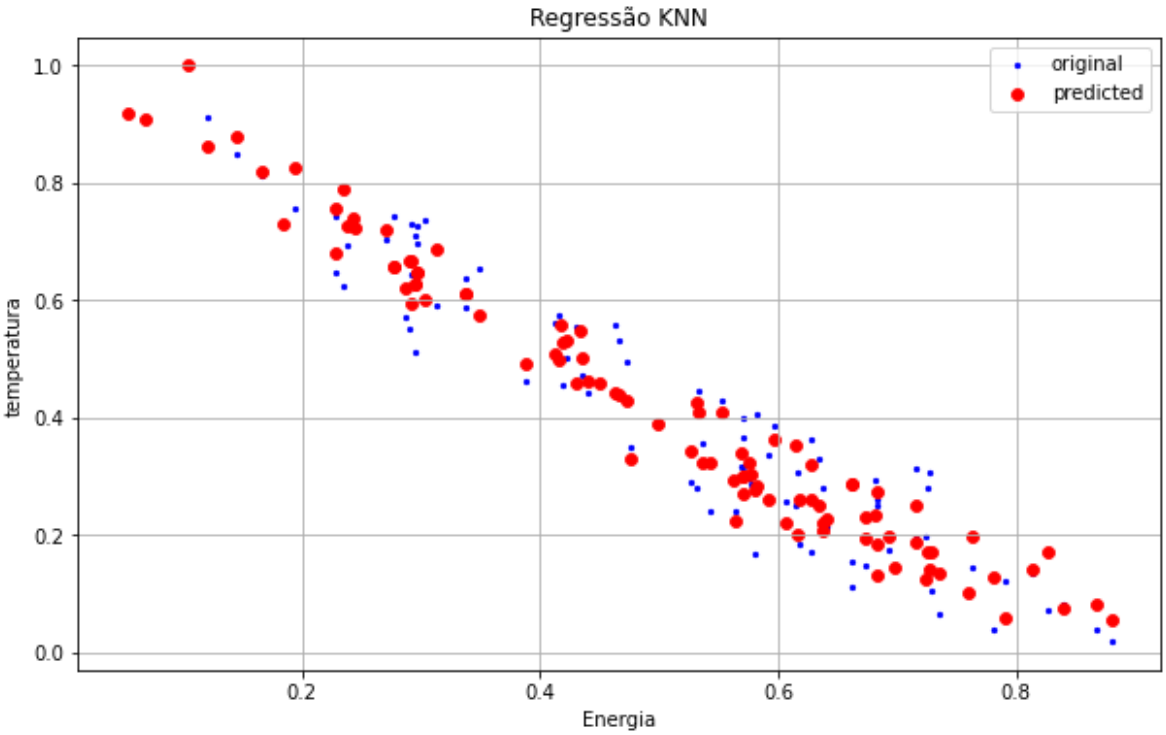


Com os pontos mais próximos do ponto P' definidos, é usada a métrica de distância para calcular o valor aproximado para y.

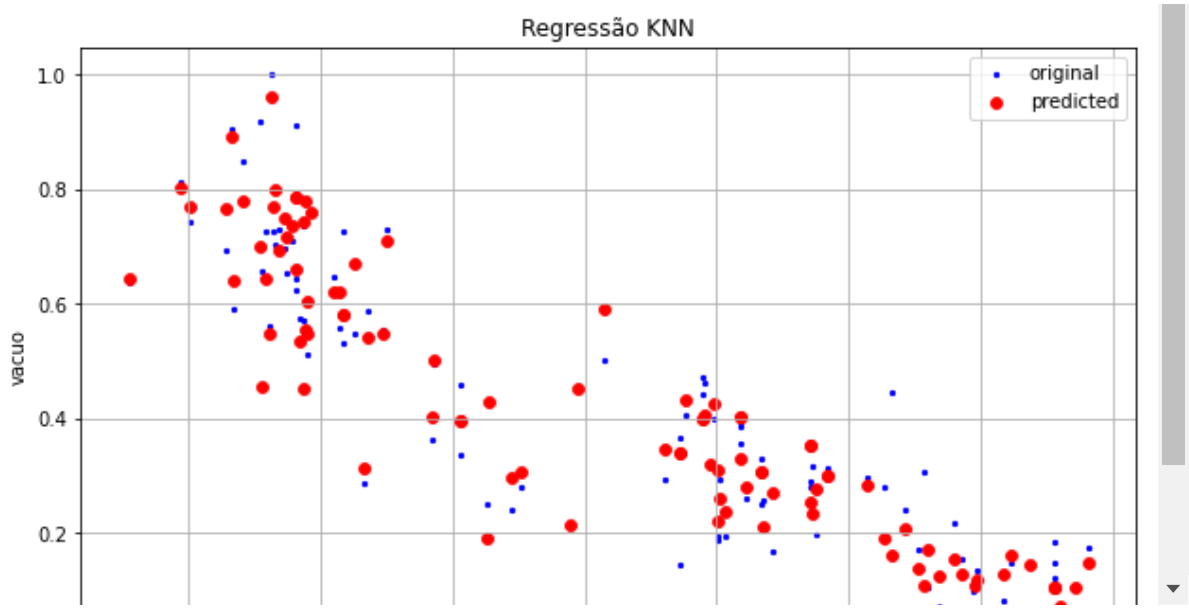
In [35]:

```
from sklearn.neighbors import KNeighborsRegressor

for coluna in xtrain.keys():
    print(coluna+' - energia')
    modelo = KNeighborsRegressor( n_neighbors=5, weights="distance" )

    x = xtrain[coluna].values.reshape(-1, 1)

    modelo.fit( x, ytrain.values.reshape(-1,1))

    print('R2:'+str(modelo.score(xtest[coluna].values.reshape(-1,1),ytest.values.re

    knn_pred = modelo.predict(xtest[coluna].values.reshape(-1,1))

    ytest_csv['KNN_'+coluna] = knn_pred

    plt.figure(figsize=(10,6))
    plt.scatter(xtest[coluna][:100],ytest[:100], s=5, color="blue", label="original
    plt.scatter(xtest[coluna][:100], knn_pred[:100], lw=0.8, color="red", label="pr
    plt.title('Regressão KNN')
    plt.xlabel('Energia')
    plt.ylabel(coluna)
    plt.grid()
    plt.legend()
    plt.show()
```

```
temperatura - energia
R2:0.9051752084430389
```




```
vacuo - energia
R2:0.904716647020458
```

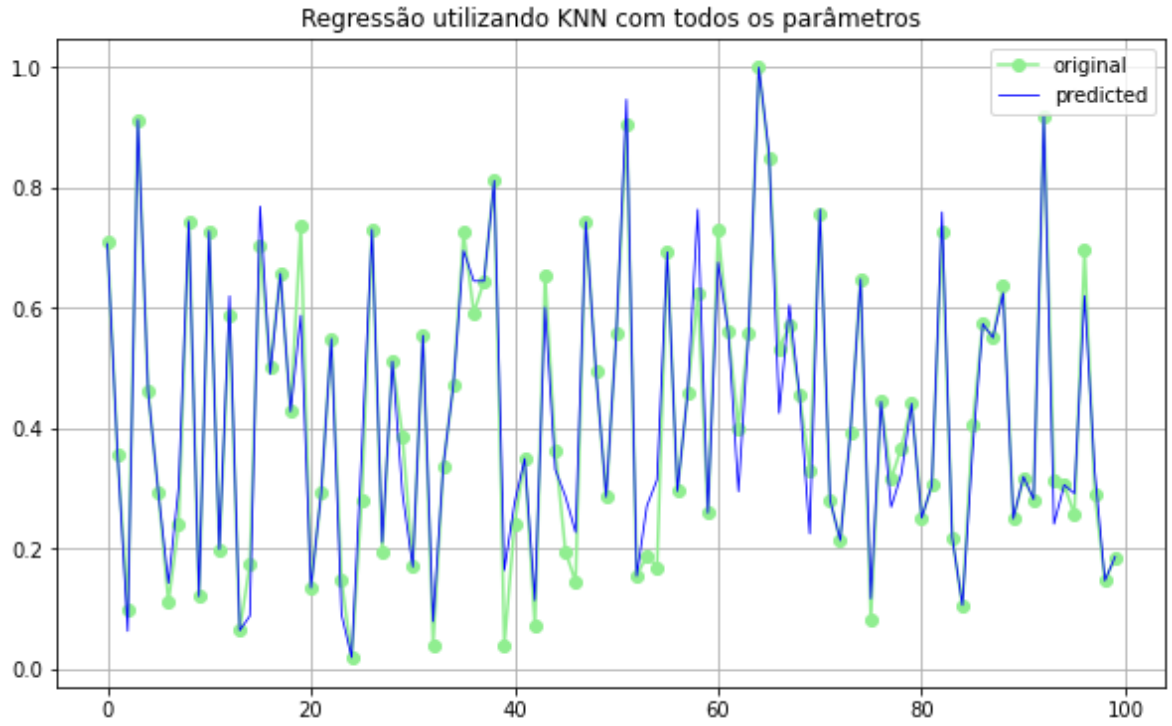


Como foi observado nos gráficos, foi obtido um desempenho semelhante entre as duas colunas e a energia. Sendo que ambas obtiveram um índice R2 com valor superior a 0.9, e como nesse indice quanto mais próximo de 1, melhor, indica que a previsão tem uma qualidade razoável.

In [36]:

```
modelo = KNeighborsRegressor( n_neighbors=5, weights="distance" )

modelo.fit(xtrain, ytrain)

print('R2:'+str(modelo.score(xtest,ytest)))

knn = modelo.predict(xtest)

ytest_csv['KNN'] = knn

plt.figure(figsize=(10,6))
plt.plot(x_ax[:100],ytest[:100], '-o',color="lightgreen", label="original")
plt.plot(x_ax[:100], knn[:100], lw=0.8, color="blue", label="predicted")
plt.title('Regressão utilizando KNN com todos os parâmetros')
plt.grid()
plt.legend()
plt.show()
```

```
R2:0.9666557418822472
```



Podemos observar que aplicando o modelo nas duas variáveis(temperatura e vácuo) obtivemos melhores resultados.

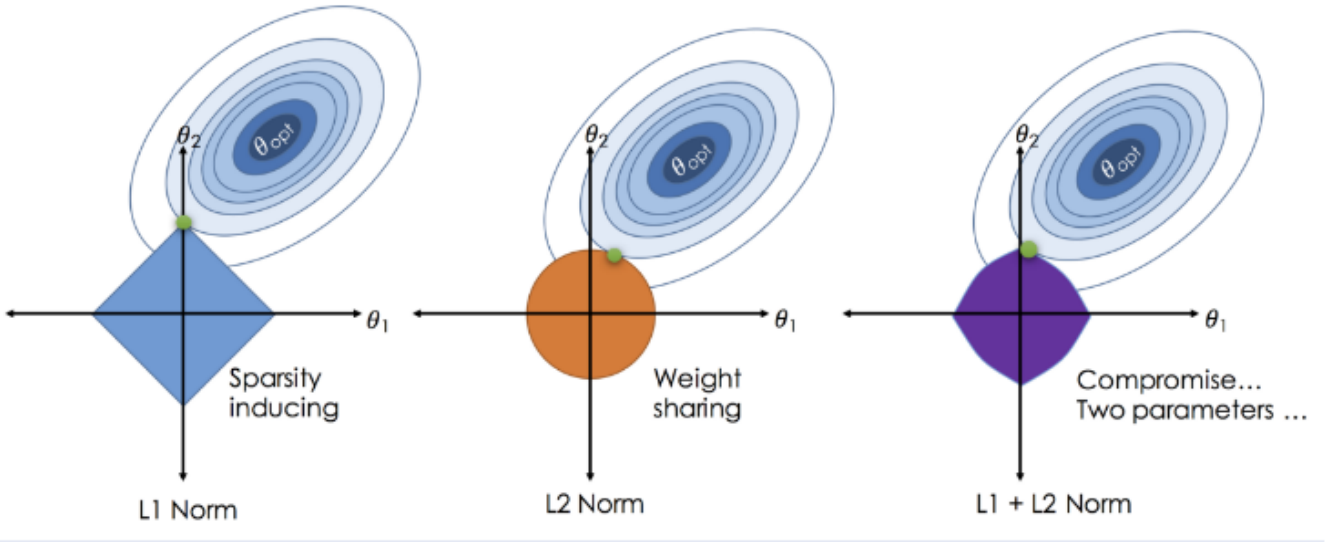
ElasticNet

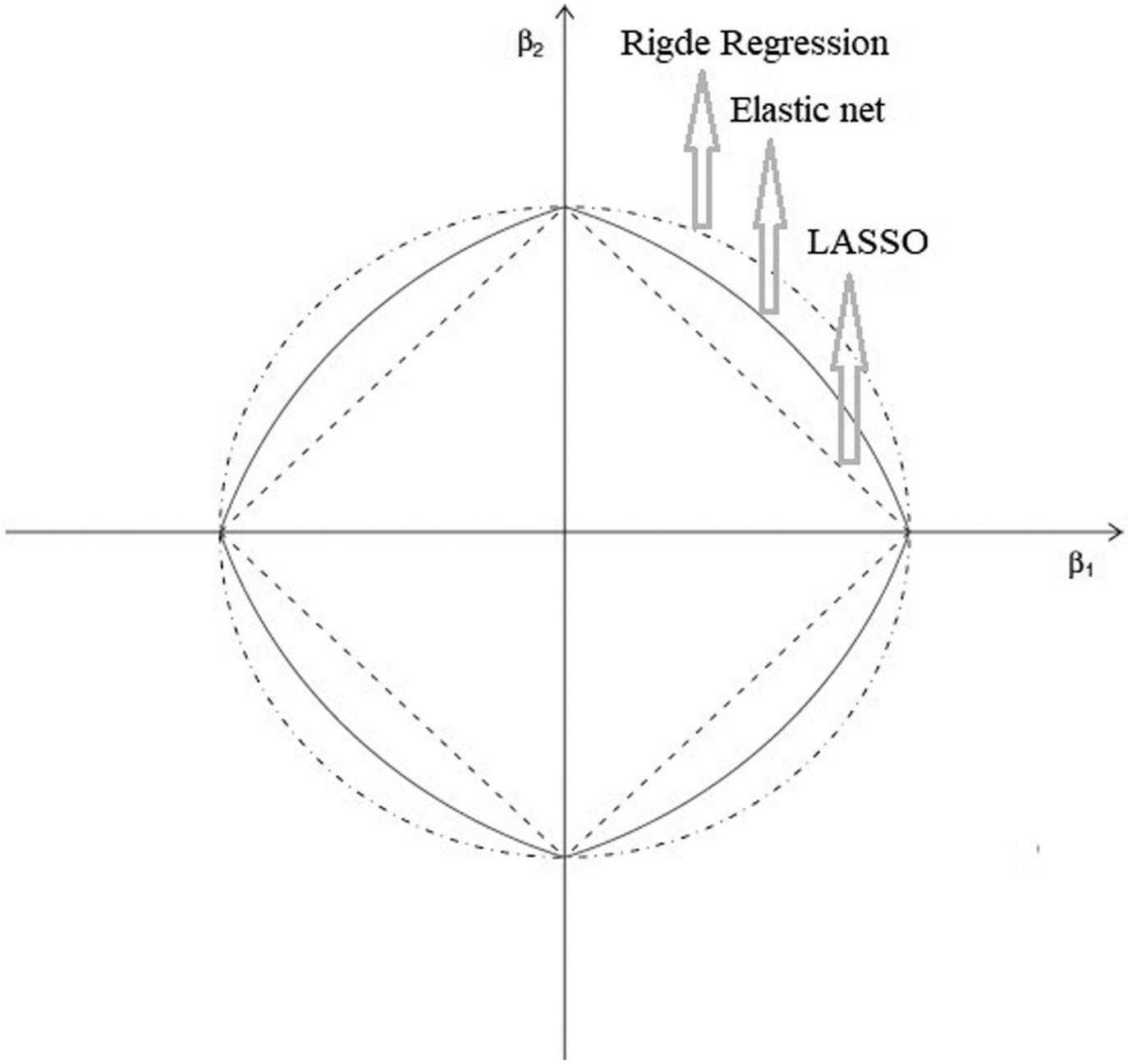
A regressão linear é o algoritmo padrão para regressão que assume uma relação linear entre as entradas(variáevis independentes) e a variável de destino(variável dependente). Uma extensão da regressão linear seria adicionar penalidades à função de perda durante o treinamento, criando modelos mais simples que possuem valores de coeficiente menores do que aqueles achados pela regressão linear. Essas extensões são chamadas de regressão linear regularizada ou regressão linear penalizada.

O ElasticNet é um tipo popular de regressão linear regularizada que combina duas penalidades populares,

especificamente as funções de penalidade L1(LASSO) e L2(RIDGE).

Este método é muito utilizado quando se tem muitos dados e muitas variáveis independentes, não tendo como saber como é a relação dessas com a varável dependente.





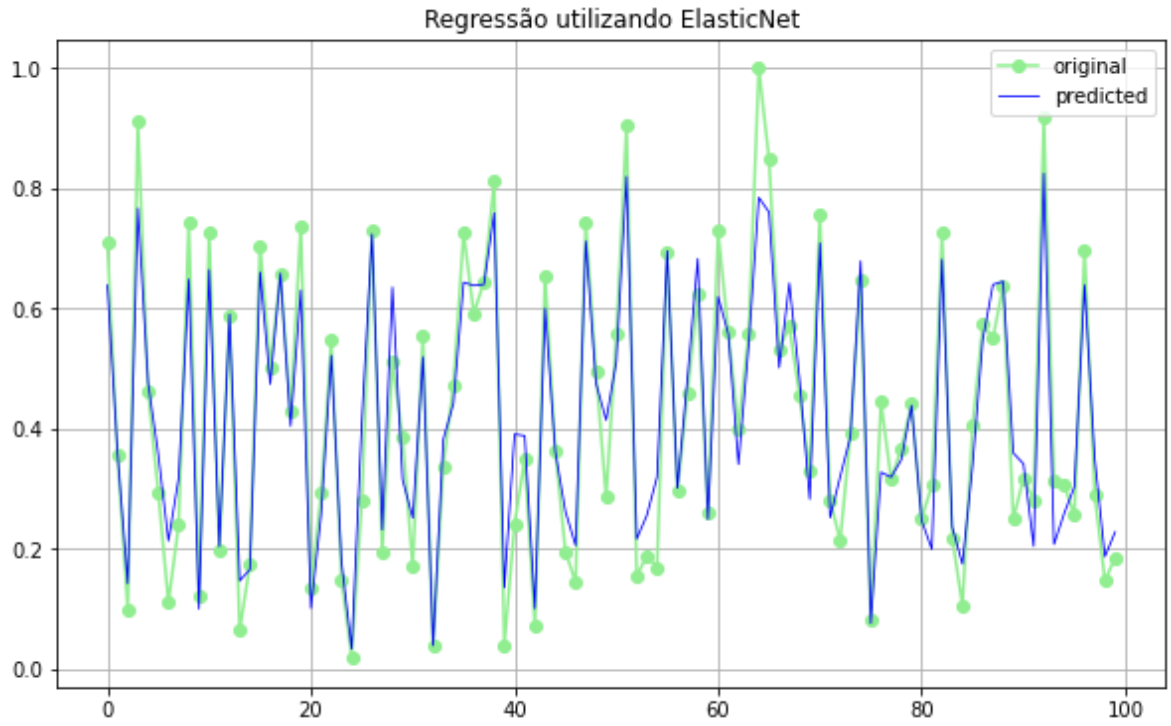
Vantagens:

- Não precisa conhecer a distribuição dos dados;
- Compensações de desvio de variância (tenta chegar em um equilíbrio);
- Consegue lidar muito bem com dados esparsos;
- Lida muito bem com multicolinearidade - alta correlação entre as variáveis independentes;
- Minimiza overfitting nos dados de treinamento.

In [37]:

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_predict

elastic=ElasticNet(alpha= 0.004).fit(xtrain, ytrain)
print('R2:'+str(elastic.score(xtest,ytest)))

y_elastic = cross_val_predict(elastic,xtest,ytest,cv=5)

ytest_csv['ElasticNet'] = y_elastic

plt.figure(figsize=(10,6))
plt.plot(x_ax[:100],ytest[:100], '-o',color="lightgreen", label="original")
plt.plot(x_ax[:100], y_elastic[:100], lw=0.8, color="blue", label="predicted")
plt.title('Regressão utilizando ElasticNet')
plt.grid()
plt.legend()
plt.show()
```

```
R2:0.9104754933372313
```



Regressão Linear Múltipla

Deve se assumir para aplicar o modelo de regressão múltipla OLS:

- Não possui alta correlação entre cada duas variáveis independentes do modelo;
- O valor do erro médio dos termos independentes deve ser zero;
- A amostra obtida para o modelo de regressão OLS deve ser retirada aleatoriamente da população;
- Todos os termos de erro na regressão devem ter a mesma variância entre as variáveis independentes.

In [38]:

```
import statsmodels.api as sm
#A Regressão linear só funciona quando temos uma variável dependente e outra indepe
#como temos mais de uma variável independente, precisamos utilizar o um modelo de r

# é necessário adicionar uma constante a matriz X
X_sm = sm.add_constant(X)
# OLS vem de Ordinary Least Squares e o método fit irá treinar o modelo
results = sm.OLS(y, X_sm,end=True).fit()
# mostrando as estatísticas do modelo
print(results.summary())
# mostrando as previsões para o mesmo conjunto passado
y_res = results.predict(X_sm)
ytest_csv['RegMult'] = y_res
```

```
                            OLS Regression Results
==============================================================================
==========
Dep. Variable:                 energia   R-squared:
0.916
Model:                             OLS   Adj. R-squared:
0.916
Method:                  Least Squares   F-statistic:
6.236e+04
Date:                 Wed, 10 Mar 2021   Prob (F-statistic):
0.00
Time:                         17:35:44   Log-Likelihood:
14211.
No. Observations:               11482   AIC:
-2.842e+04
Df Residuals:                   11479   BIC:
-2.839e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
===========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const          0.9783      0.002    562.546      0.000       0.975
0.982
temperatura   -0.8558      0.006   -148.191      0.000      -0.867
-0.844
vacuo         -0.2573      0.005    -47.730      0.000      -0.268
-0.247
==============================================================================
==========
Omnibus:                      667.048   Durbin-Watson:
2.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2510.801
Skew:                          -0.171   Prob(JB):
0.00
Kurtosis:                       5.265   Cond. No.
14.5
==============================================================================
==========
```

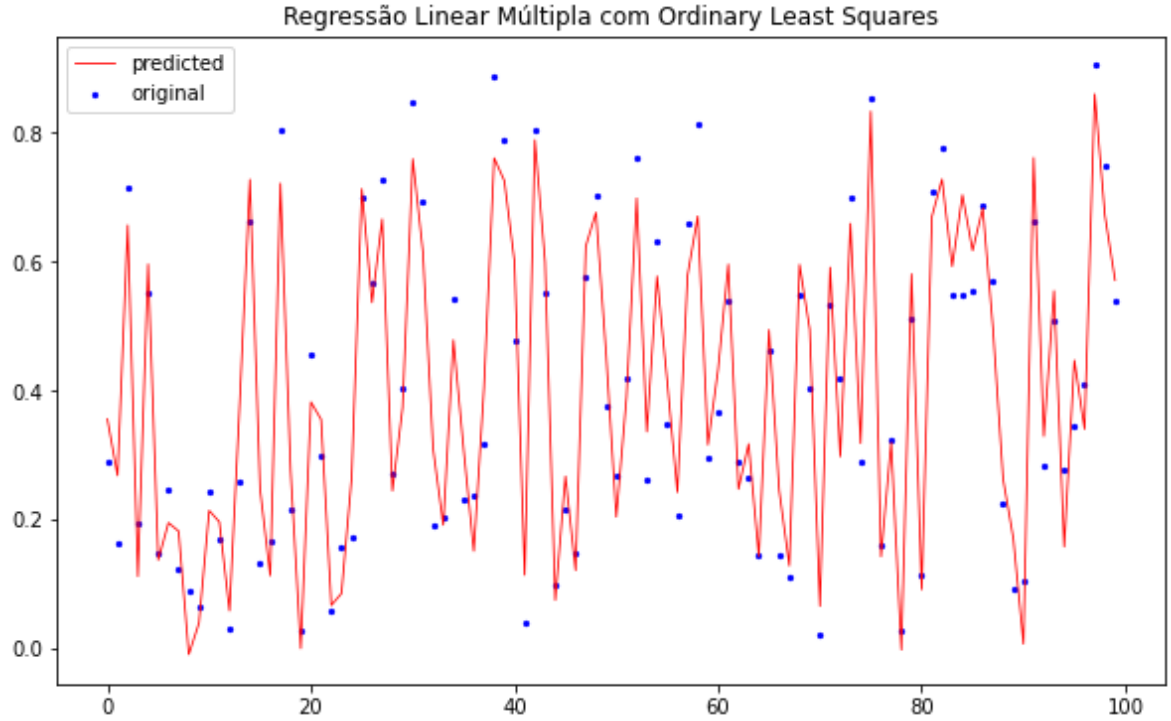
```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
```

Interpretando o resultado:

- **$R-quadrado$**: ou coeficiente de determinação, é uma medida estatística de quão bem a linha de regressão se ajusta aos dados;
- **$R-quadradoajustado$**: ajusta a estatística, levando em consideração a quantidade de variáveis independentes presentes;
- **$estatística-t$**: é a razão de desvio do valor estimado de um parâmetro de seu valor hipotético para seu erro padrão;
- **$estatística-F$:** é calculada como a razão entre o erro quadrático médio do modelo e o erro quadrático médio dos resíduos.

In [39]:

```
plt.figure(figsize=(10,6))
x_ax = range(len(X))
plt.scatter(x_ax[:100],y[:100], s=5, color="blue", label="original")
plt.plot(x_ax[:100], y_res[:100], lw=0.8, color="red", label="predicted")
plt.title('Regressão Linear Múltipla com Ordinary Least Squares')
plt.legend()
plt.show()
```



In [41]:

```
# ytest_csv.to_csv('trabalho6_teste.csv')
```