Nota 0,4

Regessão Linear - abordagem computacional

Regressão linear com scikit-learn

Etapas básicas para implementar a regressão linear:

- Importar os pacotes e classes necessários
- Definir os dados e fazer as transformações apropriadas (se necessário)
- Criar um modelo de regressão e ajustá-lo aos dados existentes
- Verificar os resultados do ajuste do modelo para saber se o modelo é satisfatório
- Aplicar o modelo para previsões

Importar bibliotecas necessárias

In [ ]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
```

Modelo 1 - Regressão linear simples

O escritório de projetos deseja identificar verificar se o processo de avaliação de esforço está adequado. Para isso precisa avliar se existe precisão nas estimativas de horas (derivadas da avalição de esforço) com relação à quantidade de horas relizadas no desenvolvimento.

Os dados encontram-se nos seguintes arquivos:

- AMP_esforco_sprint.csv
- AMP_modulo_sprint

Os dois arquivos são necessários porque se deseja fazer a avaliação por módulos.

Etapas do procedimento:

- Carregar os arquivos e combinar utilizando innerjoin
- Construir um modelo de regressão linear para prever a quantidade de horas * realizadas usando uma variável: estimativa de horas
- Construir um modelo de regressão linear para prever a quantidade de horas realizadas usando duas variáveis: estimativa de horas e pontos de caso de uso
- Avaliar se os modelos podem ser usados para previsões
- Utilizar os modelos para fazer previsões

Carregar os dados

- Carregar os arquivos AMP_esforco_sprint e AMP_modulo_sprint
- Combinar com innerjoin

In [ ]:
```
ES = pd.read_csv('./AMP_esforco_sprint.csv')
MS = pd.read_csv('./AMP_modulo_sprint.csv')

# sprints = innerjoin(modulo_sprint,esforco_sprint,'Keys',{'ID_SPRINT'})
sprints = pd.merge(MS,ES, on='ID_SPRINT', how='outer')
print(sprints.head(5))
```

```
  ID_MODULO  ID_SPRINT  Estimativa_UCP  Estimativa_Horas  Realizado_Horas  \
0       M01          1         16.7437             91.89             84.7
1       M01          2         41.3325            225.74            194.0
2       M02          3         26.0250            111.65             94.0
3       M02          4         11.4835             49.69             41.0
4       M01          5         44.3278            261.53            217.0

   Entrega_no_Prazo
0                 1
1                 0
2                 1
3                 1
4                 0
```

In [ ]:
```
M = input('Entre o identificador do módulo (M01 M02 M03 M04 M05): ')
sprintsM = sprints.loc[sprints['ID_MODULO'] == M]
print(sprintsM.head(5))
```

```
   ID_MODULO  ID_SPRINT  Estimativa_UCP  Estimativa_Horas  Realizado_Horas  \
11       M05         12         66.5514            284.08           262.00
14       M05         15         61.5668            264.66           245.90
21       M05         22         25.1214            107.99           129.65
25       M05         26         26.6305            114.48           129.80
30       M05         31         43.9759            189.04           150.00

    Entrega_no_Prazo
11                 1
14                 0
21                 1
25                 1
30                 1
```

In [ ]:
```
x = sprintsM['Estimativa_Horas'].to_numpy()
print(x)
y = sprintsM['Realizado_Horas'].to_numpy()
print(y)
```

```
[284.08 264.66 107.99 114.48 189.04  29.3   90.54 116.   277.79  75.01
 111.89 242.71 176.91 130.84  48.53]
[262.   245.9  129.65 129.8  150.    35.    86.3  143.7  238.37  59.67
 109.   244.85 153.   138.4   58.6 ]
```

Ajustar o formato das variáveis

A biblioteca requer que x e y sejam arrays multidimensionais.

In [ ]:
```
x = x.reshape((-1,1));
# print(x)
y = y.reshape((-1,1));
# print(y)
```

Calcular os coeficientes de regressão

In [ ]:
```
modelo1 = LinearRegression().fit(x, y)
print(f"b0: {modelo1.intercept_}")
print(f"b1: {modelo1.coef_}")
```

```
b0: [17.65321106]
b1: [[0.84939699]]
```

Analisar a qualidade do modelo

In [ ]:
```
# Calcular R2
R2 = modelo1.score(x,y)
print('R2: {:.4f}'.format(R2))

if (R2 >= 0.92):
    print('Ajuste do modelo: excelente')
elif (R2 >= 0.90):
    print('Ajuste do modelo: muito bom')
elif (R2 >= 0.88):
    print('Ajuste do modelo: bom')
elif (R2 >= 0.84):
    print('Ajuste do modelo: adequado')
else:
    print('Ajuste do modelo: não adequado')
```

```
R2: 0.9491
Ajuste do modelo: excelente
```

Realizar previsões

O modelo de previsões calculado pela regressão é dado pela seguinte equação:

- $Y = \beta_0 + \beta_1 X_1$
- onde X_1 é a quantidade de horas estimada e Y é a previsão para horas realizadas

A biblioteca fornece o método **predict** para realizar as previsões.

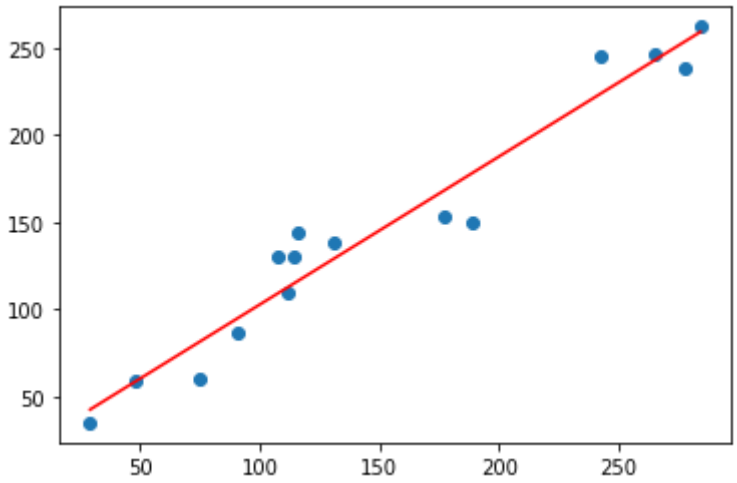
In [ ]:
```
# Realizar previsões
x_novo = np.array([75, 125, 210, 275, 325, 350]).reshape((-1, 1))
print(x_novo)
y_novo = modelo1.predict(x_novo)
print(y_novo)
```

```
[[ 75]
 [125]
 [210]
 [275]
 [325]
 [350]]
[[ 81.35798524]
 [123.8278347 ]
 [196.02657878]
 [251.23738307]
 [293.70723252]
 [314.94215725]]
```

Plotar diagrama de dispersão e reta de regressão

In [ ]:
```
# Plotar diagrama de dispersão e reta de regressão
```

```
fig, ax = plt.subplots()
ax.scatter(x, y);
x_ = np.arange(x.min(), x.max(), (x.max()-x.min())/1000).reshape((-1, 1))
y_ = modelo1.predict(x_)
ax.plot(x_,y_,color='red');
```



Modelo 2 - Regressão linear múltipla

Construir um modelo de previsão do esforço de horas de trabalho utilizando duas variáveis: pontos de caso de uso e esforço previsto em horas.

Para cada módulo:

- Selecionar o módulo
- Calcular coeficientes de regressão e a estatística R2
- Plotar o diagrama de dispersão e a reta de regreção
- Analisar a qualidade do modelo
- Realizar previsões

In [ ]:
```
X1 = sprintsM[['Estimativa_Horas','Estimativa_UCP']].to_numpy()
print(X1)
y = sprintsM['Realizado_Horas'].to_numpy()
```

```
[[284.08    66.5514]
 [264.66    61.5668]
 [107.99    25.1214]
 [114.48    26.6305]
 [189.04    43.9759]
 [ 29.3      6.8153]
 [ 90.54    21.0622]
 [116.      26.9838]
 [277.79    64.6212]
 [ 75.01    17.4482]
 [111.89    26.0273]
 [242.71    56.4599]
 [176.91    41.1542]
 [130.84    30.4373]
 [ 48.53    11.289 ]]
```

In [ ]:
```
modelo2 = LinearRegression().fit(X1, y)
print(f"b0: {modelo2.intercept_}")
print(f"Coeficientes: {modelo2.coef_}")
```

```
b0: 18.208582899538385
Coeficientes: [-1.14580902  8.55349996]
```

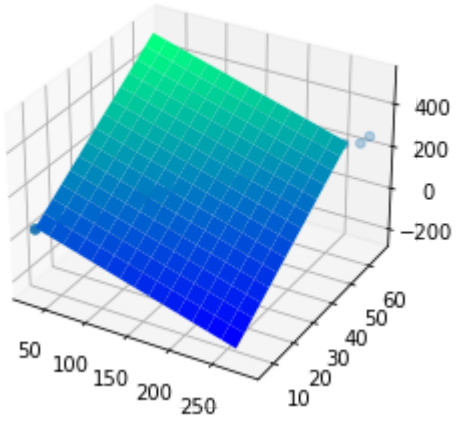
In [ ]:
```
# Valores para previsão
horas_estimadas = np.array([75, 125, 210, 275, 325, 350]).reshape((-1, 1))
pontos_casos_uso = np.array([19, 31, 53, 275, 70, 90]).reshape((-1, 1));
x_novo = np.concatenate((horas_estimadas,pontos_casos_uso), axis=1)
print('Valores novos para previsão')
print(x_novo)

# Primeira maneira de fazer as previsões (usando função predict)
y_novo = modelo2.predict(x_novo)
print('\n Valores previstos')
print(y_novo)

# Segunda maneira de fazer as previsões (usando a equação de regessão)
y_novo = modelo2.intercept_ + np.sum(modelo2.coef_ * x_novo, axis=1)
print('\n Valores previstos')
print(y_novo)
```

```
Valores novos para previsão
[[ 75  19]
 [125  31]
 [210  53]
 [275 275]
 [325  70]
 [350  90]]

 Valores previstos
[  94.78940569  140.14095424  230.92418672 2055.32359244  244.56564876
  386.99042253]

 Valores previstos
[  94.78940569  140.14095424  230.92418672 2055.32359244  244.56564876
  386.99042253]
```

Plotar diagrama de dispersão e plano de regressão

In [ ]:
```
# Plotar diagrama de dispersão
fig = plt.figure()
ax = plt.axes(projection="3d")
x1 = sprintsM['Estimativa_Horas'].to_numpy()
x2 = sprintsM['Estimativa_UCP'].to_numpy()
ax.scatter(x1,x2,y);

# Plotar plano de regressão
# Gera array para mesh grid
pmesh = 15 # quantidade de pontos em cada eixo de mesh grig
x1_ = np.arange(x1.min(), x1.max(), (x1.max()-x1.min())/pmesh)
x2_ = np.arange(x2.min(), x2.max(), (x2.max()-x2.min())/pmesh)
# Calcula mesh grig
X1, X2 = np.meshgrid(x1_, x2_)

# Gera Array para previsão
X = np.concatenate((X1.reshape((-1,1)),X2.reshape((-1,1))), axis=1)

# Faz previsão
Y = modelo2.predict(X)

# Plota gráfico de dispersão e plano de rregressão
ax.plot_surface(X1, X2, Y.reshape(np.size(x2_),np.size(x1_)), rstride=1, cstride=1
```

```
                cmap='winter', edgecolor='none')
plt.show()
```



Formativa