All steps to reproduce the Process of Hierarchical Regression Analysis

In [2]:

```
%%capture
%run 'main.py'

# importando o pyplot
from matplotlib import pyplot
# importando o stats
from scipy import stats
```

1. Dataset

In [56]:

```
# 1. Tabela de Métricas
df_all_metrics.head(3)
```

Out[56]:

	uniqueID	ID	location	maturity	totalDevelopers	complexityPoints	start	end	leadTime	technicalDebt	t
0	14187	t1	India	4.0	13.0	60.0	2014-08-11 00:00:00	2015-02-06 00:00:00	179.0	796.0	(
1	15448	b1b3	Virtual	4.0	25.0	170.0	2015-01-19 00:00:00	2015-06-05 00:00:00	137.0	2474.0	(
2	13350	tl1	India	4.0	7.0	35.0	2015-02-09 00:00:00	2015-04-02 00:00:00	52.0	202.0	(

1.1 Normalized Data

In [87]:

```
import pandas as pd
from sklearn import preprocessing

# Dictionary with set of variables
my_dict = {0:'technicalDebt', 1:'leadTime', 2:'complexityPoints',
           3:'totalDevelopers',4:'taskScaling', 5:'maturity', 6:'taskGlobalDistance'}

# Independent variable
y = df_all_metrics.technicalDebt

# Dependents variables (original)
X = df_all_metrics[[my_dict[1], my_dict[2], my_dict[3], my_dict[4], my_dict[5], my_dict[6]]]

# 1.1 X Features Normalized
x_values = X.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
```

```
x_scaled = min_max_scaler.fit_transform(x_values)
df_X_normalized = pd.DataFrame(x_scaled)
# Guardar os valores das features Xi normalizadas
X_normalized = df_X_normalized

df_X_n = df_X_normalized

df_X_n.rename(columns={0:my_dict[1], 1:my_dict[2], 2:my_dict[3], 3:my_dict[4], 4:my_dict[5], 5
:my_dict[6]},inplace = True)
df_X_n['technicalDebt'] = y
df_X_n['location'] = df_all_metrics['location']
df_all_metrics_normalized = df_X_n

df_all_metrics_normalized.head(3)
```

Out[87]:

	leadTime	complexityPoints	totalDevelopers	taskScaling	maturity	taskGlobalDistance	technicalDebt	location
0	1.00000	0.084746	0.478261	0.448529	1.0	0.200168	796.0	India
1	0.73913	0.271186	1.000000	0.294983	1.0	1.000000	2474.0	Virtual
2	0.21118	0.042373	0.217391	0.411765	1.0	0.554705	202.0	India

In [92]:

```
X_normalized = X_normalized[['leadTime', 'complexityPoints', 'totalDevelopers', 'taskScaling']
]
```

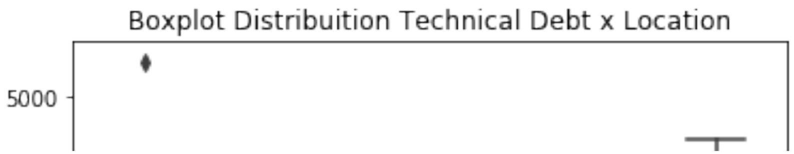
2. Box-plot about TD values group by located

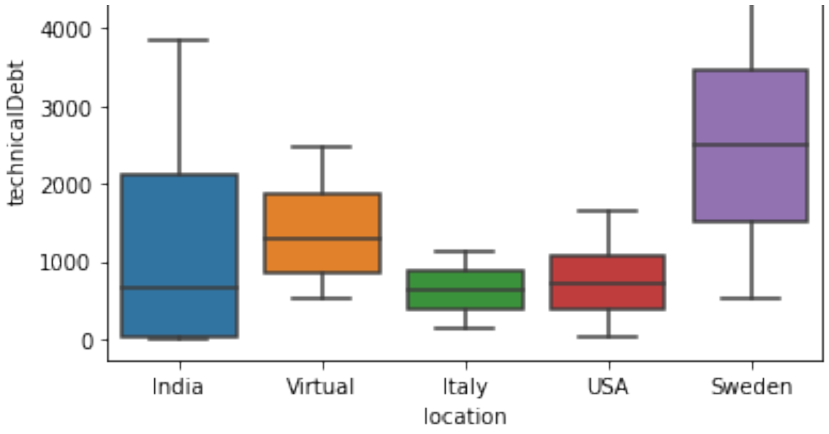
In [93]:

```
sns.boxplot(x='location', y='technicalDebt', data=df_all_metrics_normalized).set_title('Boxplo
t Distribuition Technical Debt x Location')
```

Out[93]:

```
Text(0.5,1,'Boxplot Distribuition Technical Debt x Location')
```





3. Check correlations among factors and TD

Correlation Matrix

In [94]:
```
df_all_metrics_normalized.corr().round(4)
```

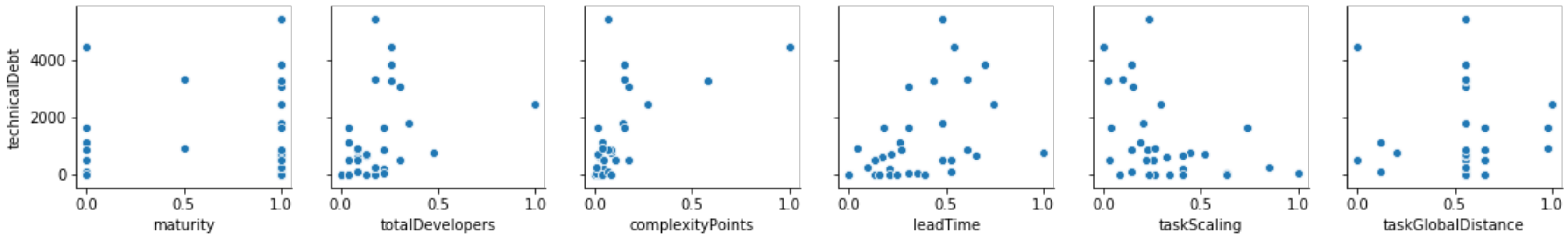
Out[94]:

	leadTime	complexityPoints	totalDevelopers	taskScaling	maturity	taskGlobalDistance	technica
leadTime	1.0000	0.3300	0.5913	-0.2425	0.0119	-0.1703	0.4088
complexityPoints	0.3300	1.0000	0.2966	-0.4695	-0.2918	-0.2742	0.6088
totalDevelopers	0.5913	0.2966	1.0000	0.0765	0.3162	0.2219	0.3433
taskScaling	-0.2425	-0.4695	0.0765	1.0000	0.5259	0.2017	-0.4190
maturity	0.0119	-0.2918	0.3162	0.5259	1.0000	0.2134	-0.0476
taskGlobalDistance	-0.1703	-0.2742	0.2219	0.2017	0.2134	1.0000	-0.0258
technicalDebt	0.4088	0.6088	0.3433	-0.4190	-0.0476	-0.0258	1.0000

Testing Linearity: Scatter Plot

In [95]:
```
ax = sns.pairplot(df_all_metrics_normalized, y_vars='technicalDebt', x_vars=['maturity', 'tota
lDevelopers', 'complexityPoints',
                                                                  'leadTime', 'taskScaling', '
taskGlobalDistance'])
ax.fig.suptitle('', fontsize=20, y=1.05)
ax
```

Out[95]: <seaborn.axisgrid.PairGrid at 0x119290588>



Spearman's Coeficient

In [96]:
```
arrayTechnicalDebt = df_all_metrics_normalized.technicalDebt.values

arrayLeadTime = df_all_metrics_normalized.leadTime.values
print("Lead Time x Technical Debt")
print(stats.spearmanr(arrayLeadTime, arrayTechnicalDebt))
print("")

arrayComplexityPoints = df_all_metrics_normalized.complexityPoints.values
print("Task Complexity x Technical Debt")
print(stats.spearmanr(arrayComplexityPoints, arrayTechnicalDebt))
print("")

arrayTotalDevelopers = df_all_metrics_normalized.totalDevelopers.values
print("Total Developers x Technical Debt")
print(stats.spearmanr(arrayTotalDevelopers, arrayTechnicalDebt))
print("")
```

```
arrayTaskScaling = df_all_metrics_normalized.taskScaling.values
print("Task Scaling x Technical Debt")
print(stats.spearmanr(arrayTaskScaling, arrayTechnicalDebt))
print("")

arrayTeamMaturity = df_all_metrics_normalized.maturity.values
print("Team Maturity x Technical Debt")
print(stats.spearmanr(arrayTeamMaturity, arrayTechnicalDebt))
print("")

arrayTaskGlobalDistance = df_all_metrics_normalized.taskGlobalDistance.values
print("Task GlobalDistance x Technical Debt")
print(stats.spearmanr(arrayTaskGlobalDistance, arrayTechnicalDebt))
```

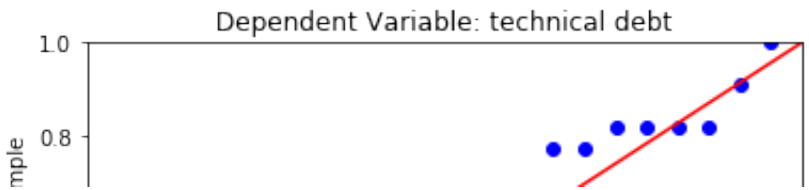
```
Lead Time x Technical Debt
SpearmanrResult(correlation=0.48587185407612227, pvalue=0.004814195878751078)

Task Complexity x Technical Debt
SpearmanrResult(correlation=0.6498748363481108, pvalue=5.689214652268194e-05)

Total Developers x Technical Debt
SpearmanrResult(correlation=0.5049990019693433, pvalue=0.0032000488729230416)

Task Scaling x Technical Debt
SpearmanrResult(correlation=-0.43922598272476504, pvalue=0.011900668916364508)

Team Maturity x Technical Debt
SpearmanrResult(correlation=-0.134695711600555, pvalue=0.462340404155135)

Task GlobalDistance x Technical Debt
SpearmanrResult(correlation=0.03356045187955763, pvalue=0.8553123790027143)
```

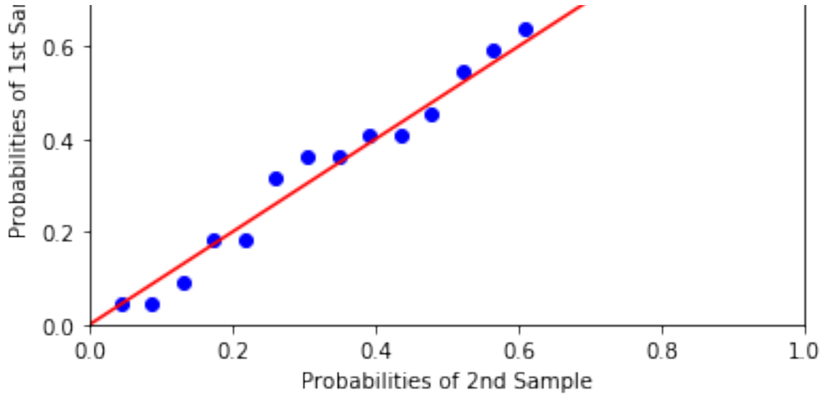
4. Testing Normality

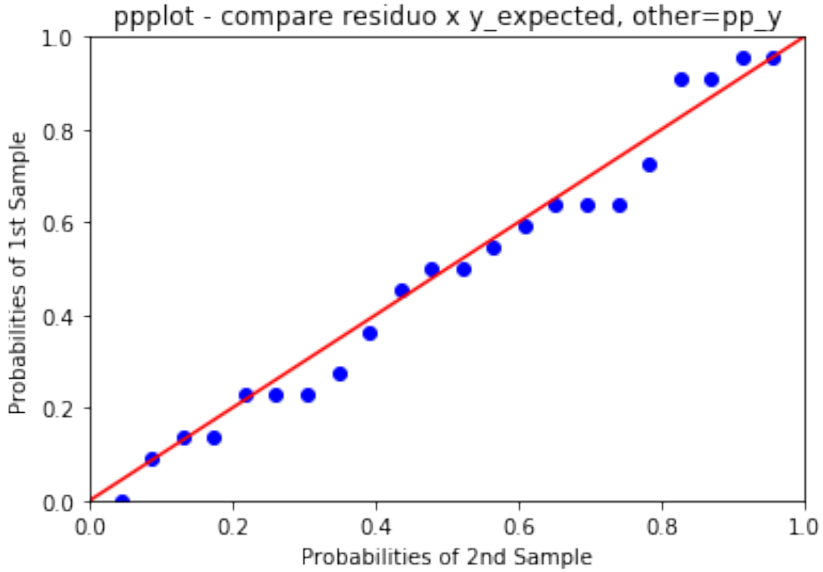
The normal distribution of residuals is tested by visually checking the normal P-P plot. The points on the plot remain close to the diagonal line, which means residuals are normally distributed. So, we do not violate the assumption of normality.

In [97]:
```
import statsmodels.api as sm
```

```python
In [97]: import statsmodels.api as sm
from matplotlib import pyplot as plt
import scipy.stats as stats

from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split

# X normalized
X = X_normalized

# print("Creating the dataset of train and test")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1000)

#Linear regression
modelo = LinearRegression()
modelo.fit(X_train, y_train)

y_previsto_train = modelo.predict(X_train)

residuo = y_train - y_previsto_train

tn_x = residuo.values
tn_y = y_previsto_train

pp_x = sm.ProbPlot(tn_x, fit=True)
pp_y = sm.ProbPlot(tn_y, fit=True)

fig = pp_y.ppplot(line='45', other=pp_x)
h = plt.title('Dependent Variable: technical debt')

fig = pp_x.ppplot(line='45', other=pp_y)
h = plt.title('ppplot - compare residuo x y_expected, other=pp_y')
plt.show()
```







5. No auto-correlation

Durbin_Watson

In [98]:
```
from statsmodels.regression.linear_model import OLS
from statsmodels.stats.stattools import durbin_watson

import numpy as np
```

```
def dw(data):
    ols_res = OLS(data, np.ones(len(data))).fit()
    return durbin_watson(ols_res.resid)

#print("dw of range=%f technicalDebt" % dw(df_all_metrics.technicalDebt.values))

for each in range(0,5):
    print("dw of " + my_dict[each] + " is", dw(df_all_metrics_normalized[my_dict[each]].values
).round(3))

    my_array_dw = np.array([
    [2.041],
        [1.614],
        [2.155],
        [1.23],
        [1.727]
        ])

my_index_dw = ['technical debt', 'lead time', 'task complexity', 'total Developers', 'task Sca
ling']
my_columns_dw = ['value']

df_my_dw = pd.DataFrame(data=my_array_dw , index=my_index_dw , columns=my_columns_dw)
df_my_dw
```

```
dw of technicalDebt is 2.041
dw of leadTime is 1.614
dw of complexityPoints is 2.155
dw of totalDevelopers is 1.23
dw of taskScaling is 1.727
```

Out[98]:

	value
technical debt	2.041
lead time	1.614
task complexity	2.155
total Developers	1.230

task Scaling	1.727

6. Testing Homescedascity

In [104]:

```
# Breusch-Pagan Test

# import smf to process regression model
import statsmodels.formula.api as smf

# 1. Data (y, x1, x2, x3, x4)
my_df_tm_modelo = df_all_metrics_normalized[[my_dict[0], my_dict[1], my_dict[2], my_dict[3],
my_dict[4]]]

df_bp = my_df_tm_modelo[['technicalDebt', 'leadTime', 'complexityPoints', 'totalDevelopers',
'taskScaling']]

# 2. fit regression model
fit = smf.ols('technicalDebt ~ leadTime+complexityPoints+totalDevelopers+taskScaling', data=d
f_bp).fit()

#fit.summary()

print("Perform a Breusch-Pagan test.")

# import lzip and sms
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

names = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f p-value']

# 3. perform Bresuch-Pagan test
test = sms.het_breuschpagan(fit.resid, fit.model.exog)

lzip(names, test)
```

```
Perform a Breusch-Pagan test.
```

Out[104]: [('Lagrange multiplier statistic', 2.326154298737759),
 ('p-value', 0.6760114026390495),
 ('f-value', 0.5291373984536142),
 ('f p-value', 0.7152904901671325)]

A Breusch-Pagan test uses the following null and alternative hypotheses:

The null hypothesis (H0): Homoscedasticity is present. The alternative hypothesis: (Ha): Homoscedasticity is not present (i.e. heteroscedasticity exists) In this dataset, the Lagrange multiplier statistic for the test is 2.326 and the corresponding p-value is 0.676. Because this p-value is not less than 0.05, we fail to reject the null hypothesis.

7. Testing Multicolinearity

In [105]:

```
import statsmodels.formula.api as smf

# import warnings
# warnings.simplefilter(action='ignore', category=FutureWarning)
from sklearn.linear_model import LinearRegression

def sklearn_vif(exogs, data):

    # initialize dictionaries
    vif_dict, tolerance_dict = {}, {}

    # form input data for each exogenous variable
    for exog in exogs:
        not_exog = [i for i in exogs if i != exog]
        X, y = data[not_exog], data[exog]

        # extract r-squared from the fit
        r_squared = LinearRegression().fit(X, y).score(X, y)

        # calculate VIF
        vif = 1/(1 - r_squared)
        vif_dict[exog] = vif
```

```
        # calculate tolerance
        tolerance = 1 - r_squared
        tolerance_dict[exog] = tolerance

    # return VIF DataFrame
    df_vif = pd.DataFrame({'VIF': vif_dict, 'Tolerance': tolerance_dict})

    return df_vif

my_df_tm_modelo4 = df_all_metrics_normalized[[my_dict[0], my_dict[1], my_dict[2], my_dict[3],
my_dict[4]]]
exogs_modelo4 = ['leadTime', 'complexityPoints', 'totalDevelopers', 'taskScaling']

df_vif_modelo4 = sklearn_vif(exogs=exogs_modelo4, data=my_df_tm_modelo4)
df_vif_modelo4
```

Out[105]:

	VIF	Tolerance
complexityPoints	1.496794	0.668094
leadTime	1.764050	0.566877
taskScaling	1.511341	0.661664
totalDevelopers	1.823810	0.548303