Il calcolo della perdita attesa secondo il principio IFRS9

Il principio contabile IFRS 9 “Strumenti Finanziari” ha introdotto significative novità in tema di riduzione di valore delle attività finanziarie, segnando il passaggio da un approccio retrospettivo basato sulle evidenze delle perdite incorse - "incurred loss approach" - , ad uno prospettico - Forword Loocking Approach - finalizzato ad intercettare in anticipo eventuali possibili perdite di valore con lo scopo di rilevarne gli effetti nel conto economico senza dover attendere che le perdite stesse si realizzino. Secondo l’IFRS 9 per la rilevazione delle rettifiche di valore non si deve, quindi, attendere che l’evento di perdita si manifesti quanto piuttosto anticiparlo attraverso una stima della perdita attesa di valore (expected credit loss - ECL). La perdita attesa (ECL) corrisponderà alla perdita che si prevede di subire sull’attività finanziaria, ponderata per la probabilità che l’evento stesso di perdita si verifichi, ovvero:

ECL = (LGD*EAD) PD dove

ECL = Expected Credit Loss LGD*EAD = Loss Given deafult*Exposure at Default PD = Probabilità di default

Ai fini della determinazione della perdita attesa si dovrà fare riferimento: Ad un orizzonte temporale di 12 mesi nel caso in cui il rischio di credito dello strumento finanziario non abbia subito significativi incrementi rispetto alla rilevazione iniziale (perdita attesa ad 1 anno); A tutta la vita dell’attività, nel caso in cui il rischio di credito dello strumento finanziario sia significativamente aumentato (c.d. perdita attesa multiperiodale o life time).

Da tali disposizioni discende ancora una volta la necessità di una stretta comunicazione tra Bilancio e Risk Management in modo tale da garantire la coerenza tra i modelli di rischio e le rettifiche di valore sulle attività finanziarie così come rappresentate in bilancio.

In particolare, ai fini del calcolo della Expected Credit Loss (ECL) si rende necessario determinare l'influenza dei fattori sistemici o macroeconomici in modo tale da poterne misurare l'effetto nei diversi scenari. In tale contesto devono pertanto essere affrontate due questioni principali relative a come:

1) elaborare e definire prospetticamente degli scenari significativi e realistici riguardo l'evoluzione futura dei fattori che caratterizzano il quadro sistemico e quindi i valori dei parametri di rischio;

2) modellizzare la relazione di dipendenza rispetto al quadro sistemico e quindi stimare le sensitivities rispetto ai sopracitati fattori.

in relazione al secondo punto si riportano di seguito i risultati di alcune prove di analisi effettuate al fine di stimare un modello di regressione lineare tra il Credit Worhiness Y implicito nei tassi di decadimento trimestrali Bankit -TD - e un set di fattori macroeconomici quali il PIL, il tasso di disoccupazione, i tassi d'interesse bancari, il commercio estero, indici immobiliari, offerta di moneta M3, credito concesso ad imprese non finanziarie e famiglie consumatrici, inflazione ... Come modello di riferimento si è preso il Credit Portfolio View di Thomas Wilson che definisce la relazione di dipendenza esistente tra probabilità di default e fattori macroeconomici attraverso una funzione logistica. Il Credit Worthiness - CW - è stato estratto dai Tassi Decadimento trimestrali -TD - sulla base della relazione:

1) TD = 1/(1+EXP(-CW)) - Tasso di Decadimento -

2) CW = -log[(1-TD)/TD] - Credit Worthiness -

Le evidenze sembrano mostrare la presenza di una relazione significativa tra il Credit Worthiness - CW - estratto dai Tassi di Decadimento e il Credit To GDP Gap; quest'ultimo indicatore è anche la grandezza di riferimento in base alla quale il Comitato di Basilea prevede che sia determinato (con gli adattamenti previsti dalle singole autorità nazionali) il requisito patrimoniale relativo alla riserva di capitale anticiclica.

I tassi di decadimento sono reperibili sul sito della Banca d'Italia Tassi di decadimento trimestrali dei Residenti al netto delle Istituzioni Finanziarie - Numeri -

In [3]:
```
#se si vuole eseguire in Google Colab occorre
#impostare la variabile colab a True - colab = True per installare le due librerie
colab = True
if colab:
    !pip install XlsxWriter
    !pip install pandasql
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simp
le/
Requirement already satisfied: XlsxWriter in /usr/local/lib/python3.7/dist-packages (3.0.3)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simp
le/
Requirement already satisfied: pandasql in /usr/local/lib/python3.7/dist-packages (0.7.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from pandasql)
(1.3.5)
Requirement already satisfied: sqlalchemy in /usr/local/lib/python3.7/dist-packages (from panda
sql) (1.4.41)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pandasql)
(1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pan
das->pandasql) (2022.4)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages
(from pandas->pandasql) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-
dateutil>=2.7.3->pandas->pandasql) (1.15.0)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.7/dist-packages (from
sqlalchemy->pandasql) (1.1.3.post0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (fr
om sqlalchemy->pandasql) (5.0.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packag
es (from importlib-metadata->sqlalchemy->pandasql) (4.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from import
lib-metadata->sqlalchemy->pandasql) (3.9.0)
```

In [4]:
```
#Importazione delle librerie necessarie
# se non già presenti devono essere installate tramite !pip o conda
%matplotlib inline
import math
import numpy as np
import pandas as pd
import pandasql as pdsql
from pandasql import sqldf

import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})
import datetime
import seaborn as sb

from sklearn.preprocessing import StandardScaler # for standardizing the Data
#from sklearn.decomposition import PCA # for PCA calculation
#from sklearn.decomposition import FactorAnalysis
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
import openpyxl
import os
#import glob
#we set some print and visualization options
np.set_printoptions(precision=2)
pd.options.display.max_columns = None #to show all the columns
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

Utility Functions

In [5]:

```
class CLS_Util(object):

    #we wrapp statsmodels adfullur just to get a nice result object
    @staticmethod
    def GM_ADF(series):
      from statsmodels.tsa.stattools import adfuller
      result = adfuller(series, autolag='AIC')
      out={'ADF Statistic':result[0],
          'P-values':result[1],
          'Lags':result[2], 'Critical Values':result[4]}
      return out

    #we iterate through all the variable in the input Data Frame
    @staticmethod
    def GM_ADF_ALL( df, filter = -1, export_file='' ):
      DictADF = dict()
      for ColName in df.columns:
        series = df[[str(ColName)]]
        DictADF[ColName]= CLS_Util.GM_ADF(series)
        OUT = pd.DataFrame(DictADF).T

      if (filter != -1) :
          OUT = OUT[OUT['P-values']>filter]
      OUT=OUT.sort_values(by=['P-values'], ascending=False)

      if export_file !='' :
          OUT.to_excel(export_file)
      return OUT

    @staticmethod
    def do_lag(df,N = 12 ):#we get a matrix with the original variable and the corresponding 8
      # df = input DataFrame to transform
      # N = number_of_lag
      # str(i) we put in each variable indicate the lag order
      Dates = df.index.values
      df_with_lag = df
      for i in np.arange(1,N +1):
        df_lag = pd.DataFrame(df.values[:-i,:], index = Dates[i:], columns = ["{}_".format(x) +
        df_with_lag = pd.concat([df_with_lag, df_lag], axis=1, join="inner")
      return df_with_lag


    @staticmethod
    def do_delta_perc(df,i = 12 ):#we get a matrix with the original variable and the correspor
      # df = input DataFrame to transform
      # N = number_of_lag
      # str(i) we put in each variable indicate the lag order
      Dates = df.index.values
      df_perc = pd.DataFrame((df.values[i:,:]-df.values[:-i,:])/df.values[:-i,:], index = Dates
                             columns = ["Perc_{}_".format(x) + str(i) for x in df.columns])
      #df_with_lag = pd.concat([df_with_lag, df_lag], axis=1, join="inner")
      return df_perc

    @staticmethod
    def TransformData(dt_in, TransformType= 0, normalize = 0):
      dt_out = np.empty_like(dt_in)

      if TransformType == 1:
        dt_out = dt_in.diff()
      elif TransformType == 2:
        dt_out = dt_in.pct_change()
      else:
        dt_out =  dt_in
```

```
        if normalize == 1:
          return (dt_out-dt_out.mean())/dt_out.std()
        else:
          return dt_out
        #sc = StandardScaler()
        #std_data = sc.fit_transform(data) #Numpy ndarray

    @staticmethod
    def is_locked(filepath):
       import os
       locked = None
       file_object = None
       if os.path.exists(filepath):
        try:
         buffer_size = 8
         file_object = open(filepath, 'a', buffer_size)
         if file_object:
          locked = False
        except IOError as message:
         locked = True
        finally:
         if file_object:
          file_object.close()
        return locked

    @staticmethod
    def wait_for_file(filepath):
       wait_time = 1
       while CLS_Util.is_locked(filepath):
         time.sleep(wait_time)

    @staticmethod
    def ALL_CHARTS(df, yvar ='CW'):
     for c in df.columns:
      if c not in ['CW', 'TD']:
        FileName = "{}.png".format(c)
        #FileName = "image.png"
        df[[yvar, c]].plot()
        plt.savefig(FileName , dpi = 120)
        plt.cla
     return

    @staticmethod
    def ALL_CHARTS_2(df, excelfile):
        wb = openpyxl.Workbook()
        wb.save(excelfile)
        wb = openpyxl.load_workbook(excelfile)
        ws = wb.active
        n=  1
        for c in df.columns:
          if c not in ['CW', 'TD']:
            FileName = "{}.png".format(c)
            img = openpyxl.drawing.image.Image(FileName)
            if n == 1 :
              ws.add_image(img,'A{}'.format(n) )
            else:
              ws.add_image(img,'A{}'.format((n-1)*25) )
            n += 1
        wb.save(excelfile)

    @staticmethod
    def Delete_Charts():
      import os
      import glob
      files = glob.glob('*.png')
      for f in files:
          os.remove(f)

    @staticmethod
```

```
    def ALL_CHARTS_TOT(df, excelfile):
      CLS_Util.ALL_CHARTS(df)
      CLS_Util.ALL_CHARTS_2(df, excelfile)
```

In [6]:

```
def Unireg_all(df, label):
  DictResults={}
  for var in df.columns :
   if var[:2] not in ['CW','TD']:
     YVAR = df.loc[:,['CW']]
     XVAR = df.loc[:,[var]]
     XVAR['CONST']= 1
     model = sm.OLS(YVAR, XVAR)
     results = model.fit()
     item = {'Data':label, 'Variable':var,'R':results.rsquared, 'PVAL':results.pvalues[0], 'PAF
     DictResults[var]=item

  uniregResults =pd.DataFrame(DictResults).T
  uniregResults = uniregResults.sort_values(by=['R'], ascending=False)
  uniregResults.to_excel('UNIREG_'+ label+'.xlsx')
  return uniregResults
```

Data Import

In [7]:

```
#Importo i tassi di decadimento
path = r'https://github.com/GMISSAGLIA/GM_PyLab/blob/Main/TD_Bankit_regression_input_data_all.x

DF_CW = pd.read_excel(path, 'DF_CW', index_col=[0], parse_dates=[0])
DF_CW['TD']=DF_CW['TD']/100
DF_CW['CW']=-np.log((1-DF_CW['TD'])/DF_CW['TD'])

DF_CW_ALL= DF_CW.copy()
DF_CW_ALL['TD_1Y'] = (1- (1-DF_CW_ALL['TD']).rolling(4).apply(np.prod, raw = True))
DF_CW_ALL['CW_1Y']=-np.log((1-DF_CW_ALL['TD_1Y'])/DF_CW_ALL['TD_1Y'])

DF_CW_ALL['TD_AVG']=(1-DF_CW_ALL['TD']).rolling(4).apply(np.prod, raw = True).map(lambda x: 1-x
DF_CW_ALL=DF_CW_ALL.iloc[3:,:]
DF_CW_ALL['CW_AVG'] =-np.log((1-DF_CW_ALL['TD_AVG'])/DF_CW_ALL['TD_AVG'])

DF_CW_1Y = (1- (1-DF_CW).rolling(4).apply(np.prod, raw = True)).iloc[3:,:]
DF_CW_1Y['CW']=-np.log((1-DF_CW_1Y['TD'])/DF_CW_1Y['TD'])

DF_CW_AVG = pd.DataFrame((1-DF_CW['TD']).rolling(4).apply(np.prod, raw = True).map(lambda x: 1-
DF_CW_AVG['CW']=-np.log((1-DF_CW_AVG['TD'])/DF_CW_AVG['TD'])

TD = DF_CW_ALL[['TD','TD_1Y','TD_AVG']]
CW = DF_CW_ALL[['CW','CW_1Y','CW_AVG']]
```

In [8]:

```
DF_CW_ALL.head()
```

Out[8]:

	TD	CW	TD_1Y	CW_1Y	TD_AVG	CW_AVG
DT						
1996-12-31	0.008	-4.877	0.028	-3.538	0.007	-4.935
1997-03-31	0.006	-5.047	0.028	-3.550	0.007	-4.947
1997-06-30	0.007	-4.955	0.027	-3.597	0.007	-4.993
1997-09-30	0.005	-5.199	0.026	-3.616	0.007	-5.012
1997-12-31	0.008	-4.763	0.027	-3.582	0.007	-4.978

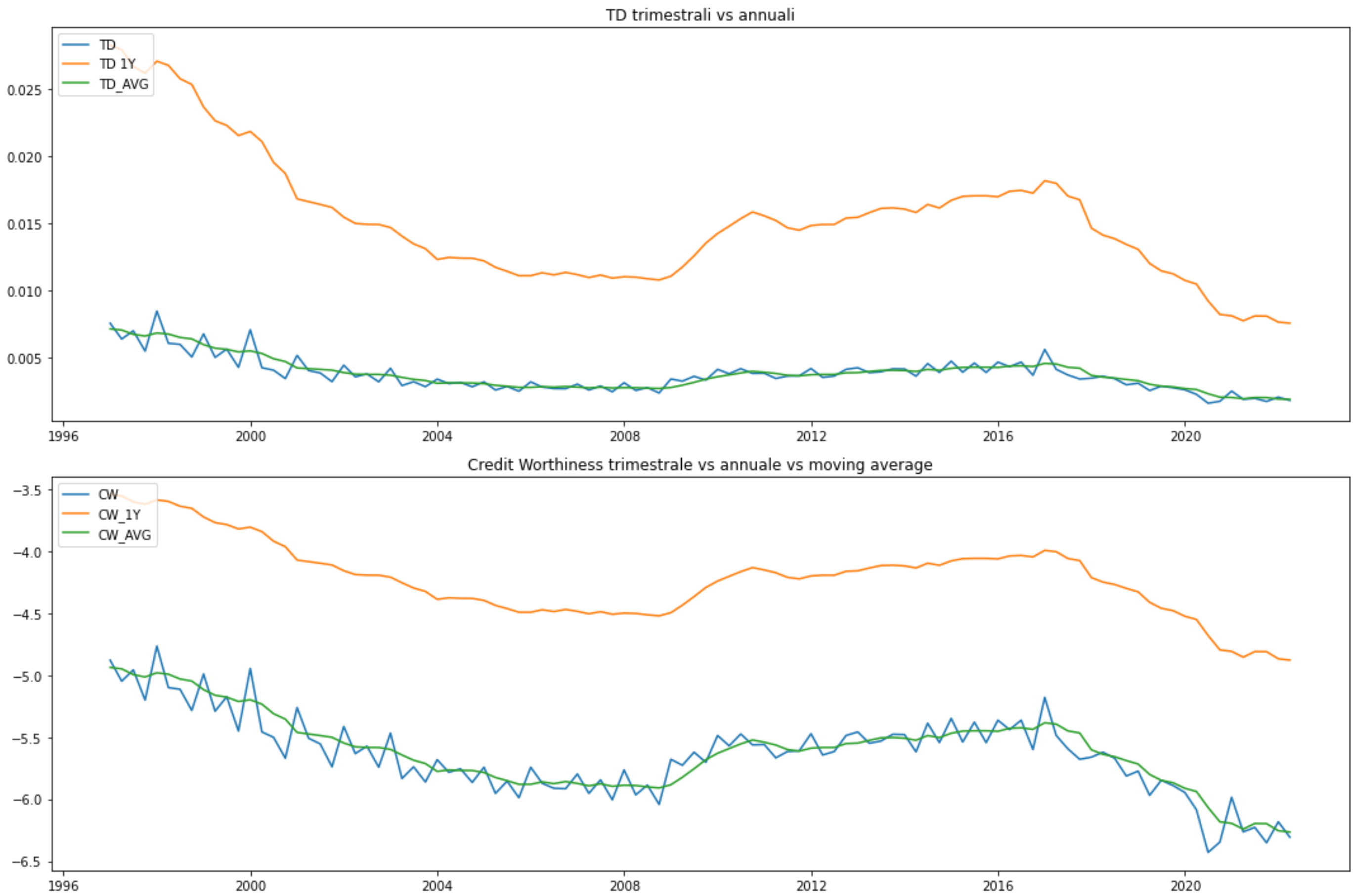
In [18]:

```python
fig, axs = plt.subplots(2,1,figsize=(15, 10))

axs[0].plot(DF_CW_ALL['TD'], label =  'TD')
axs[0].plot(DF_CW_ALL['TD_1Y'], label =  'TD 1Y')
axs[0].plot(DF_CW_ALL['TD_AVG'], label =  'TD_AVG')
axs[0].set_title('TD trimestrali vs annuali')
axs[0].legend(loc='upper left')

axs[1].plot(DF_CW_ALL[['CW']], label = 'CW')
axs[1].plot(DF_CW_ALL[['CW_1Y']], label = 'CW_1Y')
axs[1].plot(DF_CW_ALL[['CW_AVG']], label = 'CW_AVG')
axs[1].set_title('Credit Worthiness trimestrale vs annuale vs moving average')
axs[1].legend(loc='upper left')

fig.tight_layout()
```



In [22]:

```python
DF_YoY = pd.read_excel(path, 'DF_YoY', index_col=[0], parse_dates=[0])
DF_DECO = pd.read_excel(path, 'T_Deco_Var', index_col=[1])
DF_DECO
```

Out[22]:

	ID	Descrizione
Variable		
X_1	1	Retail Sales (Real/Volume)
X_2	2	Harmonised Index of Consumer Prices
X_3	3	Producer Prices (Output Prices)
X_4	4	Disposable Personal Income Real
X_5	5	Eurostat Industrial Production Italy Wages & S...
X_6	6	House Prices SWDA
X_7	7	Property Price - Non-Residential Buildings
X_8	8	Property Price - Offices

Variable	ID	Descrizione
X_9	9	Property Price - Residential Buildings
X_10	10	Property Price Commercial
X_11	11	Property Price - Industrial
X_12	12	Italy Real Effective Exchange Rate Broad
X_13	13	Italy Foreign Currency Reserve
X_14	14	Export NSA
X_15	15	Import NSA
X_16	16	Government Debt
X_17	17	Non Performing Loans
X_18	18	Italy Deposits of Resident Consumer Households
X_19	19	Deposits of Non-financial Corporations
X_20	20	Deposits of Producer Households
X_21	21	Italy Loans to Residents Non Financial Corpora...
X_22	22	Consumer Credit
X_23	23	Italy Loans to Residents > 5Y
X_24	24	Unemployement Rate
X_25	25	Capacity Utilization
X_26	26	EMMI EURO OverNight Index Aver
X_27	27	Italy Bank Interest Rates on Outstanding Euro ...
X_28	28	Minimum Rate on Short Term Loans to Non Financ...
X_29	29	Average Rate on Bonds - Outstanding Amounts
X_30	30	Mortgage Interest Rate NSA
X_31	31	Italy Bank Interest Rates on Mortgage
X_32	32	BIS Italy Credit to Private Non Financial Sect...
X_33	33	Real GDP (swda, yoy%)
X_34	34	EU Italy GDP Deflator (yoy %, sa)
X_35	35	Producer Price Index (yoy %)
X_36	36	Large Industry Employment (yoy %)
X_37	37	Unit Labor Costs (yoy %)
X_38	38	EU Italy Nominal Labour Costs (yoy %, wda)
X_39	39	Industrial Production (yoy %, wda)
X_40	40	Industrial Sales (yoy %)
X_41	41	New Car Registrations (yoy %)
X_42	42	OECD Italy Leading Indicator (yoy %)
X_43	43	Retail Sales (yoy %)
X_44	44	ECB M3 Money Supply (yoy %, sa)

In [23]:

```python
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#DF_YoY_STD = scaler.fit_transform(DF_YoY)
DF_CW_STD = (DF_CW - DF_CW.mean())/DF_CW.std()
DF_CW_1Y_STD = (DF_CW_1Y - DF_CW_1Y.mean())/DF_CW_1Y.std()
DF_CW_AVG_STD = (DF_CW_AVG - DF_CW_AVG.mean())/DF_CW_AVG.std()

DF_YoY_LAG = CLS_Util.do_lag(DF_YoY)
DF_YoY_STD = (DF_YoY - DF_YoY.mean())/DF_YoY.std()
DF_YoY_STD_LAG = CLS_Util.do_lag(DF_YoY_STD)


pd.DataFrame(DF_YoY_STD).describe()
```

Out[23]:

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_10	X_11	X_12	X_13	X_1
count	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.000	77.00
mean	-0.000	-0.000	0.000	-0.000	-0.000	-0.000	0.000	-0.000	-0.000	0.000	0.000	-0.000	0.000	0.00
std	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.00
min	-4.525	-2.064	-1.917	-2.785	-4.466	-1.171	-1.203	-1.271	-1.195	-1.584	-1.192	-2.775	-2.529	-3.09
25%	-0.480	-0.859	-0.776	-0.456	-0.146	-0.614	-0.760	-0.727	-0.573	-0.723	-0.680	-0.655	-0.580	-0.40
50%	-0.023	-0.125	0.094	0.219	0.101	-0.255	-0.438	-0.471	-0.235	-0.298	-0.371	0.084	-0.046	0.01
75%	0.365	0.693	0.441	0.608	0.362	-0.085	0.951	0.679	-0.058	0.744	0.507	0.620	0.603	0.59
max	5.196	2.013	5.150	3.165	4.600	3.074	2.375	2.606	3.067	3.054	3.482	2.580	3.209	1.99

In [24]:

```python
ADF_YoY = CLS_Util.GM_ADF_ALL(DF_YoY)
ADF_YoY_STD = CLS_Util.GM_ADF_ALL(DF_YoY_STD)

ADF_CW = CLS_Util.GM_ADF_ALL(DF_CW)
ADF_CW_1Y = CLS_Util.GM_ADF_ALL(DF_CW_1Y)
ADF_CW_AVG = CLS_Util.GM_ADF_ALL(DF_CW_AVG)

ADF_CW_STD = CLS_Util.GM_ADF_ALL(DF_CW_STD)
ADF_CW_1Y_STD = CLS_Util.GM_ADF_ALL(DF_CW_1Y_STD)
ADF_CW_AVG_STD = CLS_Util.GM_ADF_ALL(DF_CW_AVG_STD)

with pd.ExcelWriter('00_Regression_Analysis_Input_data.xlsx') as writer:
 DF_CW.to_excel(writer, sheet_name='DF_CW')
 DF_CW_STD.to_excel(writer, sheet_name='DF_CW_STD')


 DF_CW_1Y.to_excel(writer, sheet_name='DF_CW_1Y')
 DF_CW_1Y_STD.to_excel(writer, sheet_name='DF_CW_1Y_STD')

 DF_CW_AVG.to_excel(writer, sheet_name='DF_CW_AVG')
 DF_CW_AVG_STD.to_excel(writer, sheet_name='DF_CW_AVG_STD')

 DF_YoY.to_excel(writer, sheet_name='DF_YoY')
 DF_YoY_LAG.to_excel(writer, sheet_name='DF_YoY_LAG')

 DF_YoY_STD.to_excel(writer, sheet_name='DF_YoY_STD')
 DF_YoY_STD_LAG.to_excel(writer, sheet_name='DF_YoY_STD_LAG')

 ADF_YoY.to_excel(writer, sheet_name='ADF_YoY')
 ADF_YoY_STD.to_excel(writer, sheet_name='ADF_YoY_STD')
 ADF_CW.to_excel(writer, sheet_name='ADF_CW')
 ADF_CW_STD.to_excel(writer, sheet_name='ADF_CW_STD')
 ADF_CW_1Y.to_excel(writer, sheet_name='ADF_CW_1Y')
 ADF_CW_1Y_STD.to_excel(writer, sheet_name='ADF_CW_1Y_STD')
```

```
    ADF_CW_AVG.to_excel(writer, sheet_name='ADF_CW_AVG')
    ADF_CW_AVG_STD.to_excel(writer, sheet_name='ADF_CW_AVG_STD')

    DF_DECO.to_excel(writer, sheet_name='DF_DECO')
```

Credith Wortiness Univariate OLS Regression

In [25]:

```
#we get the total Dataframe containing the variable to Explain and all the possible explicative
df_all = pd.concat([DF_CW_1Y, DF_YoY_LAG], axis= 1, join='inner')
df_all_STD= pd.concat([DF_CW_1Y_STD, DF_YoY_STD_LAG], axis= 1, join='inner')
```

In [ ]:

```
CLS_Util.Delete_Charts()
CLS_Util.ALL_CHARTS_TOT(df_all_STD, 'ALL_CHARTS_YoY_STD.XLSX')
CLS_Util.Delete_Charts()
```

In [28]:

```
Unireg_all(df_all, 'CW Reg YoY')
```

Out[28]:

	Data	Variable	R	PVAL	PARAMS
X_32_3	CW Reg YoY	X_32_3	0.741	0.000	-0.049
X_32_2	CW Reg YoY	X_32_2	0.733	0.000	-0.048
X_32_4	CW Reg YoY	X_32_4	0.673	0.000	-0.049
X_32_1	CW Reg YoY	X_32_1	0.664	0.000	-0.046
X_17_9	CW Reg YoY	X_17_9	0.618	0.000	0.900
...	...	...	...	...	...
X_25_8	CW Reg YoY	X_25_8	0.000	0.971	-0.000
X_30_9	CW Reg YoY	X_30_9	0.000	0.992	0.001
X_37_8	CW Reg YoY	X_37_8	0.000	0.994	-0.000
X_31_10	CW Reg YoY	X_31_10	0.000	0.996	-0.000
X_18_6	CW Reg YoY	X_18_6	0.000	0.997	0.001

572 rows × 5 columns

In [69]:

```
# il modello stimato é:
# CW = b0 + b1*CreditToGDP_Gap_2 + b2*CreditToGDP_Gap_6+ b3*NPL_3+ b4*EXPORT_3+ e

df= df_all.loc[:,['CW','X_32_2','X_17_3','X_14_3']]
df.rename(columns={'X_32_2':'CreditToGDP_Gap_2',
                   'X_17_3':'NPL_3','X_14_3':'EXPORT_3'}, inplace=True)

YVAR = df.loc[:,'CW']
XVAR = df.loc[:,['CreditToGDP_Gap_2', 'NPL_3','EXPORT_3']]

CONST = pd.DataFrame(index=df.index)
CONST['CONST']= 1
XVAR_TOT = pd.concat([CONST,XVAR], axis=1, join='inner')
model = sm.OLS(YVAR, XVAR_TOT)
results = model.fit()
```

In [70]:

```
#le variabili esplicative risultano significative e con segno coerente con l'aspettattiva teori
results.summary()
```

Out[70]:

OLS Regression Results

Dep. Variable:	CW	**R-squared:**	0.871
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	137.0
Date:	Sat, 15 Oct 2022	**Prob (F-statistic):**	4.62e-27
Time:	19:05:07	**Log-Likelihood:**	67.469
No. Observations:	65	**AIC:**	-126.9
Df Residuals:	61	**BIC:**	-118.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
CONST	-4.3349	0.012	-367.911	0.000	-4.358	-4.311
CreditToGDP_Gap_2	-0.0470	0.004	-13.115	0.000	-0.054	-0.040
NPL_3	0.2334	0.063	3.693	0.000	0.107	0.360
EXPORT_3	-0.8923	0.129	-6.893	0.000	-1.151	-0.633

Omnibus:	1.020	**Durbin-Watson:**	0.784
Prob(Omnibus):	0.600	**Jarque-Bera (JB):**	0.427
Skew:	-0.079	**Prob(JB):**	0.808
Kurtosis:	3.365	**Cond. No.**	51.0

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

Matrice di correlazione e Condition Number

In [71]:

```
CORR_MAT = XVAR.corr()
CORR_MAT
```

Out[71]:

	CreditToGDP_Gap_2	NPL_3	EXPORT_3
CreditToGDP_Gap_2	1.000	-0.644	-0.349
NPL_3	-0.644	1.000	0.174
EXPORT_3	-0.349	0.174	1.000

In [72]:

```
result =  np.linalg.cond(CORR_MAT)
result
```

Out[72]:

```
5.515205789156765
```

Previsti vs Effettivi

In [73]:
```
def test(X, Y, results):
  CW_Predicted = results.predict(X)
  DF_CW_Predicted = pd.DataFrame(CW_Predicted,index=X.index)
  DF_CW_Predicted.columns = ['CW_Predicted']

  DF_CW_Compare = pd.concat([DF_CW_Predicted,Y, DF_CW_1Y['TD']],axis=1, join= 'inner')
  DF_CW_Compare['TD_Predicted']=DF_CW_Compare['CW_Predicted'].map(lambda x: 1/(1+math.exp(-x)))
  DF_CW_Compare['CW_Residuals']= DF_CW_Compare['CW']-DF_CW_Compare['CW_Predicted']
  DF_CW_Compare['TD_Residuals']= DF_CW_Compare['TD']-DF_CW_Compare['TD_Predicted']

  DF_CW_Compare[['TD_Residuals']].plot.kde()
  DF_CW_Compare[['CW_Residuals']].plot.kde()
  fig, axs = plt.subplots(5,1,figsize=(15, 10))

  axs[0].plot(DF_CW_Compare[['TD']], label = 'TD')
  axs[0].plot(DF_CW_Compare[['TD_Predicted']], label =  'TD_Predicted')
  axs[0].set_title('TD effettivi vs Previsti')
  axs[0].legend(loc='upper left')

  axs[1].plot(DF_CW_Compare[['CW']], label = 'CW')
  axs[1].plot(DF_CW_Compare[['CW_Predicted']], label = 'CW Predicted')
  axs[1].set_title('Credit Worthiness effettivo vs Previsto')
  axs[1].legend(loc='upper left')

  axs[2].plot(df_all_STD[['CW']], label = 'CW')
  axs[2].plot(df_all_STD[['X_32_2']], label =  'Credit To GDP Lag 2')
  axs[2].set_title("Credit to GDP GAP - LAG 2")
  axs[2].legend(loc='upper left')

  axs[3].plot(df_all_STD[['CW','X_17_3']], label = 'CW')
  axs[3].plot(df_all_STD[['X_17_3']], label = 'NPL Lag 3')
  axs[3].set_title("Crediti Non Performing - LAG 3")
  axs[3].legend(loc='upper left')

  axs[4].plot(df_all_STD[['CW']], label = 'CW')
  axs[4].plot(df_all_STD[['X_14_3']], label =  'EXPORT Lag 3')
  axs[4].set_title("Esporatazioni LAG 3")
  axs[4].legend(loc='upper left')

  fig.tight_layout()

  return DF_CW_Compare.describe()
```

In [59]:
```
def Test_evaluation_metrics_func(y_true, y_pred):
    from sklearn import metrics
    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    def sqrt_mean_squared_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.sqrt(metrics.mean_squared_error(y_true, y_pred))
    Forecast_Performance={}
    #Lista di tuple con descrizione e funzione di misurazione delle performance
    Performance_Functions = [('MSE:Mean Square Error', metrics.mean_squared_error),
                             ('MAE:Mean Absolute Error', metrics.mean_absolute_error),
                             ('RMSE:Mean Square Error',sqrt_mean_squared_error),
                            ('MAPE:Mean Square Error', mean_absolute_percentage_error),
                             ('R2: R2 score',metrics.r2_score)]
    print('Evaluation metric results:-')
    for pair in Performance_Functions:
      value =  pair[1](y_true, y_pred)
      Forecast_Performance[pair[0]]=value
      print("{} : {:,.2f}".format( pair[0], value))

    print(end='\n\n')

    result = pd.DataFrame(Forecast_Performance,index=[0])
```

```
    #result.columns=[['MSE', 'MAE', 'RMSE', 'MAPE', 'R2']]
    return result
```

In [84]:

```
X = XVAR_TOT
Y = YVAR
Predicted = results.predict(X)
Test_evaluation_metrics_func(Y.values, Predicted.values)
```

```
Evaluation metric results:-
MSE:Mean Square Error : 0.01
MAE:Mean Absolute Error : 0.06
RMSE:Mean Square Error : 0.07
MAPE:Mean Square Error : 1.36
R2: R2 score : 0.81
```

Out[84]:

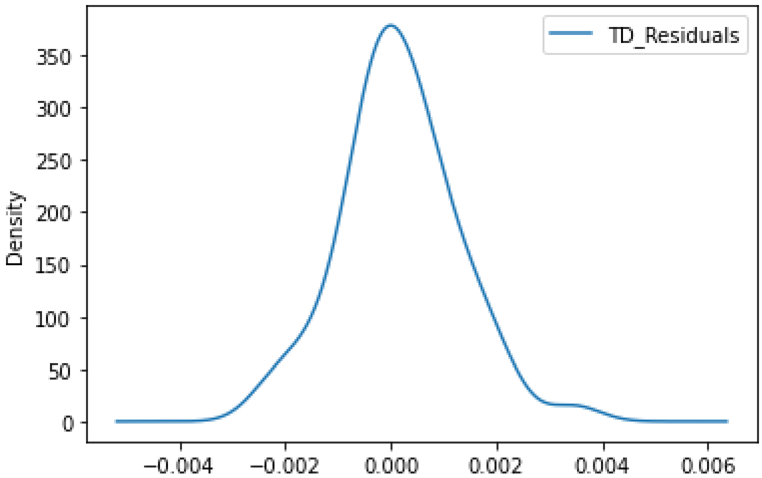
	MSE:Mean Square Error	MAE:Mean Absolute Error	RMSE:Mean Square Error	MAPE:Mean Square Error	R2: R2 score
0	0.005	0.057	0.074	1.359	0.814

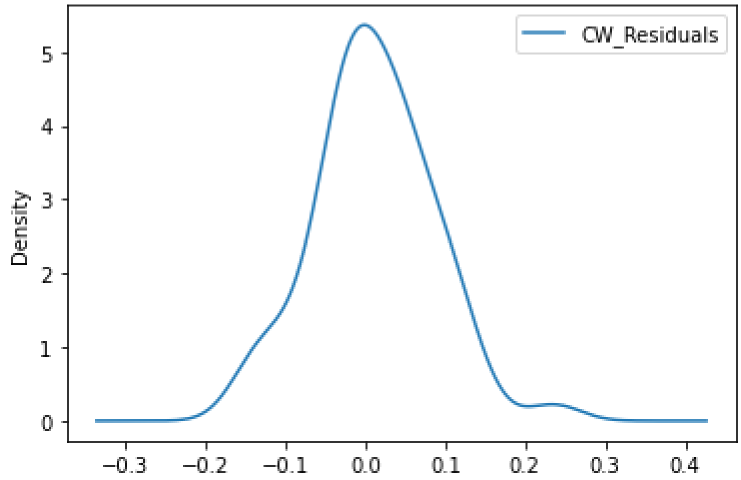
In [85]:

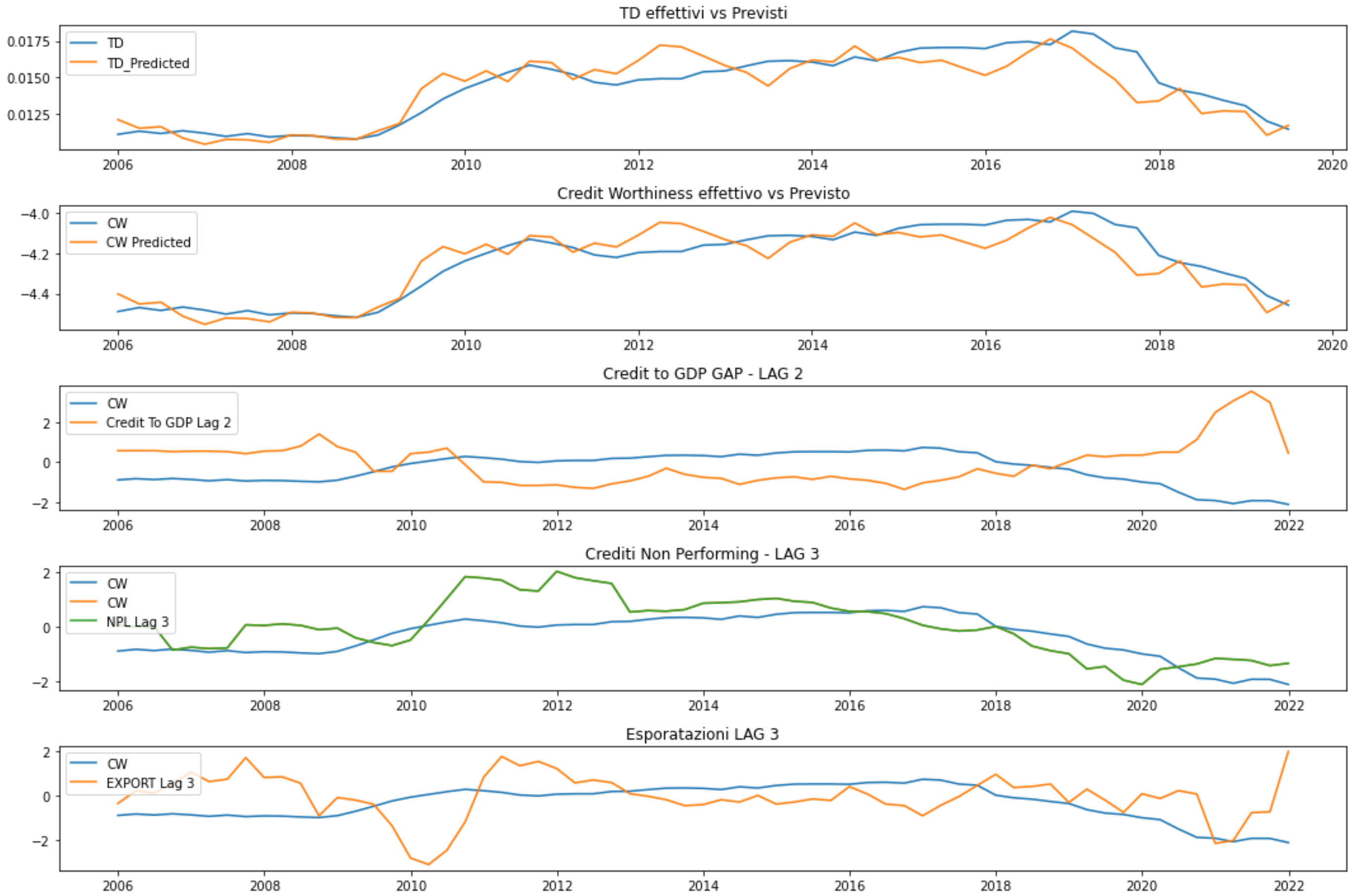
```
test(X,Y, results)
```

Out[85]:

	CW_Predicted	CW	TD	TD_Predicted	CW_Residuals	TD_Residuals
count	55.000	55.000	55.000	55.000	55.000	55.000
mean	-4.254	-4.243	0.014	0.014	0.011	0.000
std	0.169	0.173	0.002	0.002	0.074	0.001
min	-4.553	-4.519	0.011	0.010	-0.145	-0.002
25%	-4.430	-4.445	0.012	0.012	-0.028	-0.000
50%	-4.193	-4.196	0.015	0.015	0.001	0.000
75%	-4.116	-4.110	0.016	0.016	0.054	0.001
max	-4.020	-3.989	0.018	0.018	0.236	0.003







Test

In [60]:

```
NTEST = 10
Train, Test = df.iloc[0:-NTEST],df.iloc[-NTEST:]
X_train,X_test,Y_train, Y_test = Train.iloc[:,1:],Test.iloc[:,1:],Train.iloc[:,0],Test.iloc[:,0
X_test
```

Out[60]:

	CreditToGDP_Gap_2	NPL_3	EXPORT_3
2019-09-30	0.600	-0.405	-0.030
2019-12-31	0.600	-0.439	0.046
2020-03-31	1.200	-0.317	0.027
2020-06-30	1.200	-0.296	0.059
2020-09-30	3.700	-0.274	0.045
2020-12-31	9.100	-0.229	-0.158
2021-03-31	11.400	-0.236	-0.146

	CreditToGDP_Gap_2	NPL_3	EXPORT_3
2021-06-30	13.300	-0.245	-0.031
2021-09-30	11.100	-0.287	-0.028
2021-12-31	1.000	-0.268	0.220

In [76]:

```
YVAR = Y_train
XVAR = X_train
CONST = pd.DataFrame(index=X_train.index)

CONST['CONST']= 1
XVAR_TOT = pd.concat([CONST,XVAR], axis=1, join='inner')
model = sm.OLS(YVAR, XVAR_TOT)
Train_results = model.fit()
```

In [77]:

```
Train_results.summary()
```

Out[77]:

OLS Regression Results

Dep. Variable:	CW	**R-squared:**	0.839
Model:	OLS	**Adj. R-squared:**	0.829
Method:	Least Squares	**F-statistic:**	88.50
Date:	Sat, 15 Oct 2022	**Prob (F-statistic):**	3.23e-20
Time:	19:06:26	**Log-Likelihood:**	69.260
No. Observations:	55	**AIC:**	-130.5
Df Residuals:	51	**BIC:**	-122.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
CONST	-4.3202	0.012	-353.657	0.000	-4.345	-4.296
CreditToGDP_Gap_2	-0.0507	0.004	-11.984	0.000	-0.059	-0.042
NPL_3	0.0809	0.063	1.293	0.202	-0.045	0.206
EXPORT_3	-0.8193	0.113	-7.240	0.000	-1.046	-0.592

Omnibus:	2.130	**Durbin-Watson:**	0.766
Prob(Omnibus):	0.345	**Jarque-Bera (JB):**	1.383
Skew:	0.094	**Prob(JB):**	0.501
Kurtosis:	3.754	**Cond. No.**	41.9

Notes:

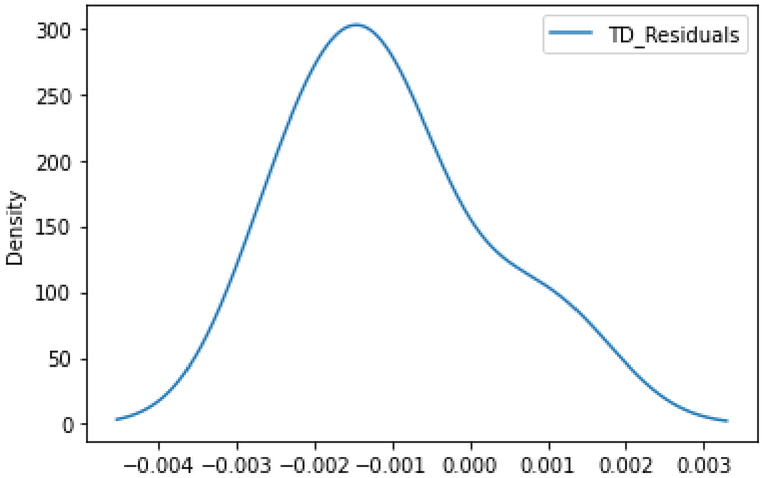
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

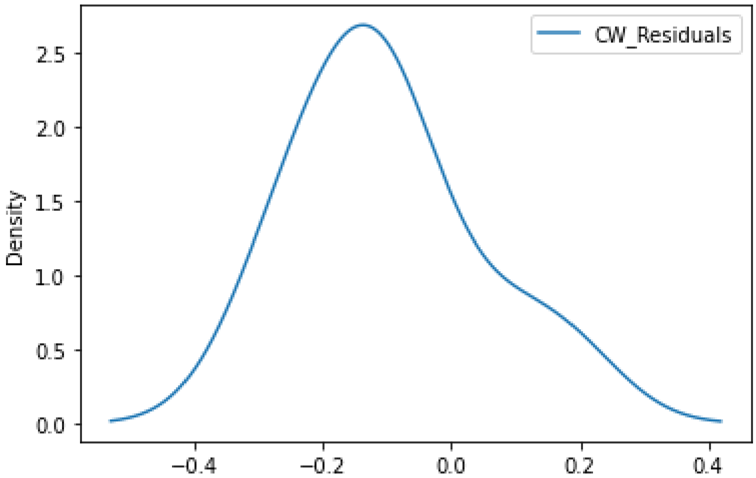
In [78]:

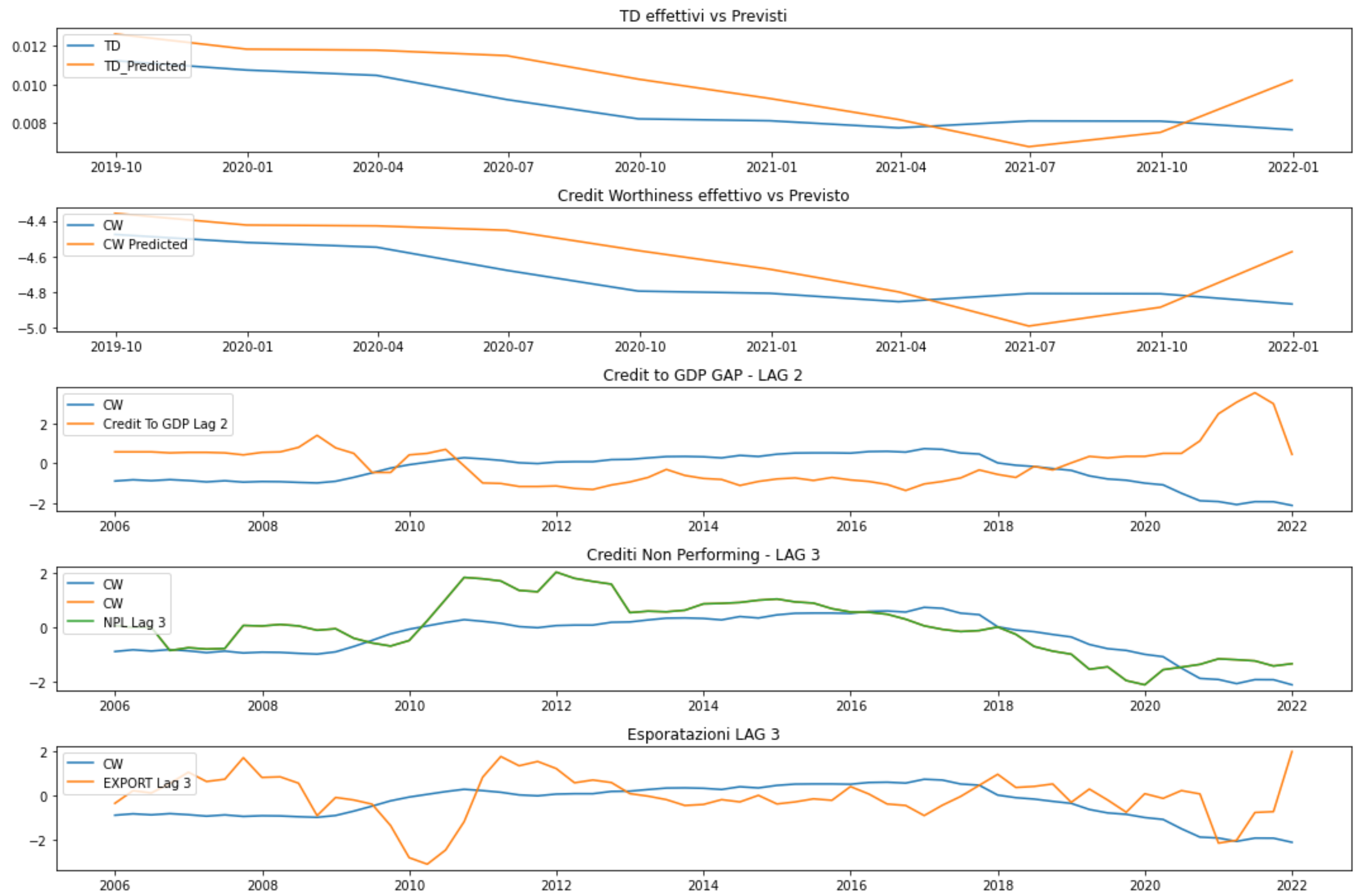
```
CONST = pd.DataFrame(index=X_test.index)
CONST['CONST']= 1
X = pd.concat([CONST,X_test], axis=1, join='inner')
Y = Y_test
test(X,Y, Train_results)
```

Out[78]:

	CW_Predicted	CW	TD	TD_Predicted	CW_Residuals	TD_Residuals
count	10.000	10.000	10.000	10.000	10.000	10.000
mean	-4.615	-4.716	0.009	0.010	-0.101	-0.001
std	0.215	0.148	0.001	0.002	0.142	0.001
min	-4.989	-4.866	0.008	0.007	-0.293	-0.003
25%	-4.766	-4.808	0.008	0.008	-0.202	-0.002
50%	-4.570	-4.800	0.008	0.010	-0.119	-0.001
75%	-4.435	-4.580	0.010	0.012	-0.065	-0.001
max	-4.359	-4.477	0.011	0.013	0.182	0.001







Test usando SKLEARN

In [80]:

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression(fit_intercept=True)
```

In [81]:

```
lm.fit(X_train, Y_train)
DF_CW_Predicted = pd.DataFrame(lm.predict(X_test),index=X_test.index)
DF_CW_Predicted.columns = ['CW_Predicted']
DF_CW_Predicted.describe()
```

Out[81]:

	CW_Predicted
count	10.000
mean	-4.615
std	0.215
min	-4.989
25%	-4.766
50%	-4.570
75%	-4.435
max	-4.359

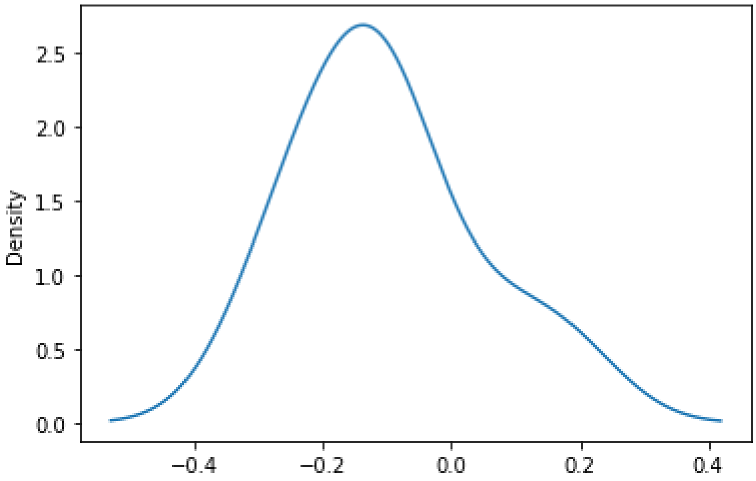
In [ ]:

```
DF_CW_Predicted
```

In [82]:

```
df_test_compare = pd.concat([DF_CW_Predicted,Y_test],axis=1, join= 'inner')
df_test_compare['residuals'] = df_test_compare['CW']-df_test_compare['CW_Predicted']
df_test_compare['residuals'].plot.kde()
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x7f322c4cf610>



In [83]:

```
Test_evaluation_metrics_func(Y_test.values, DF_CW_Predicted.values)
```

```
Evaluation metric results:-
MSE:Mean Square Error : 0.03
MAE:Mean Absolute Error : 0.15
RMSE:Mean Square Error : 0.17
MAPE:Mean Square Error : 4.74
R2: R2 score : -0.45
```

Out[83]:

	MSE:Mean Square Error	MAE:Mean Absolute Error	RMSE:Mean Square Error	MAPE:Mean Square Error	R2: R2 score
0	0.028	0.152	0.169	4.745	-0.446