Vicepresidencia Banca Personas & Pymes

Gerencia Analítica Personas y Pymes

Por: José Max Barrios Lara

1- Analísis de datos

En esta sección vamos a importar los datos, a analizarlos, y realizar la limpieza de datos necesaria para tener los datos más optimos.

1.1 Importando modulos y datos

In [361]:

```
# Data Manipulation.
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import random
import numpy as np
import time

# Data Visualization.
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning Data process and metrics
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.feature_selection import f_regression

# Linear Regression
from sklearn.linear_model import LinearRegression

# Random forest Regressor.
from sklearn.ensemble import RandomForestRegressor

# Model Save
import joblib

# statistics
import statsmodels.formula.api as sm
import statsmodels.formula.api as smf

# ANN
import keras

from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [362]:

```
# Reading data from github repository.
url_base_entrenamiento =
'https://raw.githubusercontent.com/jmbarrios27/Pymes/main/base_entrenamiento.csv'
url_base_prueba =  'https://raw.githubusercontent.com/jmbarrios27/Pymes/main/base_prueba.csv'


# Converting into pandas dataframe.
train = pd.read_csv(url_base_entrenamiento, sep=',')
test = pd.read_csv(url_base_prueba, sep=',')
```

1.2 Inspección de Datos

En esta sección vamos a identificar el formato de los datos, valores NaN entre otros.

In [363]:

```
train.head()
```

Out[363]:

	llave_cod_cliente	admin_antiguedad_banco	admin_flag_gerenciado	buro_creditos_otros_bancos	buro_score_apc	buro_wallet_share
0	1183	23	0	0	NaN	0.00
1	1361	23	0	1	629.0	0.58
2	1551	23	0	1	661.0	0.02
3	1702	23	0	1	600.0	0.94
4	2897	23	0	1	329.0	0.00

5 rows × 49 columns

In [364]:

```
# revisando tipo de datos y viendo valores nulos por variable.
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 49 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   llave_cod_cliente           5000 non-null   int64  
 1   admin_antiguedad_banco      5000 non-null   int64  
 2   admin_flag_gerenciado       5000 non-null   int64  
 3   buro_creditos_otros_bancos  5000 non-null   int64  
 4   buro_score_apc              4303 non-null   float64
 5   buro_wallet_share           4992 non-null   float64
 6   comp_flag_atm               4996 non-null   float64
 7   comp_flag_bpi               4996 non-null   float64
 8   comp_flag_cnb               4996 non-null   float64
 9   comp_flag_pos               4996 non-null   float64
 10  comp_flag_suc               4996 non-null   float64
 11  comp_perc_atm               5000 non-null   float64
 12  comp_perc_canal_fisico      5000 non-null   float64
 13  comp_perc_cnb               5000 non-null   float64
 14  comp_perc_bpi               5000 non-null   float64
 15  comp_perc_pos               5000 non-null   float64
 16  comp_perc_suc               5000 non-null   float64
 17  comp_score_digital          5000 non-null   int64  
 18  comp_txn_atm                4996 non-null   float64
 19  comp_txn_bpi                4996 non-null   float64
 20  comp_txn_cnb                4996 non-null   float64
 21  comp_txn_pos                5000 non-null   int64  
 22  comp_txn_suc                4996 non-null   float64
 23  comp_usd_atm_prom           4996 non-null   float64
 24  comp_usd_bpi_prom           4996 non-null   float64
 25  comp_usd_cnb_prom           4996 non-null   float64
 26  comp_usd_pos_prom           4996 non-null   float64
 27  comp_usd_suc_prom           4996 non-null   float64
 28  dem_edad                    5000 non-null   int64  
 29  dem_planilla                5000 non-null   int64  
 30  finc_bal_act                5000 non-null   float64
 31  finc_bal_pas                5000 non-null   float64
 32  finc_perc_act_tc            5000 non-null   float64
 33  finc_perc_pas_tc            5000 non-null   float64
 34  finc_sva                    4792 non-null   float64
 35  finc_tamano_comercial       5000 non-null   float64
 36  pdcto_flag_auto             5000 non-null   int64  
 37  pdcto_flag_cta_ahorro       5000 non-null   int64  
 38  pdcto_flag_cta_corriente    5000 non-null   int64  
 39  pdcto_flag_cta_dpf          5000 non-null   int64  
 40  pdcto_flag_financomer       5000 non-null   int64  
 41  pdcto_flag_garantizado      5000 non-null   int64  
 42  pdcto_flag_hipoteca         5000 non-null   int64  
```

```
 43  pdcto_flag_pp               5000 non-null   int64
 44  pdcto_flag_seguros          5000 non-null   int64
 45  pdcto_flag_tiene_tdc        4996 non-null   float64
 46  pdcto_flag_tiene_tdd        4996 non-null   float64
 47  pdcto_ivc_actual            5000 non-null   int64
 48  dem_salario                 5000 non-null   float64
dtypes: float64(31), int64(18)
memory usage: 1.9 MB
```

Todas las variables tienen valores numericos enteros o flotantes

In [365]:

```
# Descripción de datos.
train.describe()
```

Out[365]:

	llave_cod_cliente	admin_antiguedad_banco	admin_flag_gerenciado	buro_creditos_otros_bancos	buro_score_apc	buro_wallet_s
count	5.000000e+03	5000.000000	5000.0	5000.000000	4303.000000	4992.000
mean	6.922316e+07	8.659000	0.0	0.587400	530.963514	0.251
std	4.314458e+07	7.813893	0.0	0.492351	155.179391	0.393
min	1.183000e+03	0.000000	0.0	0.000000	0.000000	0.000
25%	5.462370e+05	3.000000	0.0	0.000000	457.000000	0.000
50%	9.867239e+07	7.000000	0.0	1.000000	544.000000	0.000
75%	9.892435e+07	12.000000	0.0	1.000000	625.500000	0.540
max	9.906340e+07	40.000000	0.0	1.000000	948.000000	1.000

8 rows × 49 columns

In [366]:

```
train.finc_tamano_comercial.describe()
```

Out[366]:

```
count       5000.000000
mean        5893.783930
std        15265.023313
min            0.000000
25%            2.660000
50%          179.685000
75%         5454.827500
max       372671.230000
Name: finc_tamano_comercial, dtype: float64
```

Podemos observar que existen algunas columnas con valores NaN, por lo que vamos a observar con detenimiento cuales son estas variables.

In [367]:

```
# revisando columnas con NAn Values
train.isna().sum()
```

Out[367]:

```
llave_cod_cliente              0
admin_antiguedad_banco         0
admin_flag_gerenciado          0
buro_creditos_otros_bancos     0
buro_score_apc               697
buro_wallet_share              8
comp_flag_atm                  4
comp_flag_bpi                  4
comp_flag_cnb                  4
comp_flag_pos                  4
```

```
comp_flag_suc                  4
comp_perc_atm                  0
comp_perc_canal_fisico         0
comp_perc_cnb                  0
comp_perc_bpi                  0
comp_perc_pos                  0
comp_perc_suc                  0
comp_score_digital             0
comp_txn_atm                   4
comp_txn_bpi                   4
comp_txn_cnb                   4
comp_txn_pos                   0
comp_txn_suc                   4
comp_usd_atm_prom              4
comp_usd_bpi_prom              4
comp_usd_cnb_prom              4
comp_usd_pos_prom              4
comp_usd_suc_prom              4
dem_edad                       0
dem_planilla                   0
finc_bal_act                   0
finc_bal_pas                   0
finc_perc_act_tc               0
finc_perc_pas_tc               0
finc_sva                     208
finc_tamano_comercial          0
pdcto_flag_auto                0
pdcto_flag_cta_ahorro          0
pdcto_flag_cta_corriente       0
pdcto_flag_cta_dpf             0
pdcto_flag_financomer          0
pdcto_flag_garantizado         0
pdcto_flag_hipoteca            0
pdcto_flag_pp                  0
pdcto_flag_seguros             0
pdcto_flag_tiene_tdc           4
pdcto_flag_tiene_tdd           4
pdcto_ivc_actual               0
dem_salario                    0
dtype: int64
```

Tenemos 19 columnas que contienen NaN values, algunas de ellas con pocos pero otras si con bastantes ,por lo que vamos a inspeccionar y analizar cada una de ellas. Vamos a analizar primero las dos columnas que contienen mayor cantidad de valores nulos como:

-**buro_score_apc:** Score de Riesgo.

-**finc_sva:** medida de rentabilidad del banco generada por el cliente con todos sus productos.

In [368]:

```
# Score de riesgo
train['buro_score_apc'].unique()
```

Out[368]:

```
array([ nan, 629., 661., 600., 329., 555., 642., 774., 771., 758., 789.,
       631., 770., 669., 647., 749., 708., 589., 796., 616., 712., 685.,
       674., 578., 547., 559., 861., 619., 571., 640., 741., 524., 577.,
       461., 764., 723., 644., 645., 612., 476., 362., 729., 690., 672.,
       711., 444., 561., 626., 518., 624., 614., 807., 719., 635., 648.,
       576., 855., 696., 517., 604., 520., 511., 622., 772., 439., 548.,
       609., 652., 527., 500., 608., 823., 687., 471., 765., 449., 452.,
       757., 786., 545., 707., 759., 733., 606., 663., 818., 587., 487.,
       534., 795., 544., 793., 798., 666., 670., 637.,   0., 744., 423.,
       535., 570., 522., 519., 671., 692., 540., 474., 568., 562., 508.,
       660., 599., 479., 627., 698., 777., 760., 566., 748., 584., 634.,
       800., 594., 657., 805., 596., 462., 512., 715., 697., 575., 493.,
       470., 694., 446., 597., 412., 756., 665., 514., 392., 794., 567.,
       656., 374., 406., 419., 682., 728., 769., 436., 288., 592., 371.,
       722., 820., 747., 531., 572., 618., 494., 838., 603., 438., 489.,
       581., 367., 537., 613., 579., 380., 482., 664., 521., 676., 456.,
       787., 549., 434., 721., 832., 569., 780., 564., 370., 411., 457.,
       632., 503., 655., 705., 688., 546., 667., 398., 773., 691., 731.,
```

```
       453., 480., 714., 752., 574., 650., 763., 869., 766., 739., 585.,
       825., 621., 740., 605., 699., 641., 416., 307., 557., 309., 743.,
       847., 420., 427., 580., 400., 441., 299., 659., 746., 448., 513.,
       623., 675., 282., 617., 654., 528., 465., 704., 464., 497., 556.,
       375., 410., 495., 491., 504., 702., 342., 678., 643., 628., 425.,
       658., 799., 588., 730., 437., 839., 415., 630., 335., 767., 558.,
       551., 552., 424., 538., 543., 817., 686., 418., 525., 679., 783.,
       428., 481., 346., 713., 539., 703., 523., 598., 590., 607., 396.,
       651., 553., 472., 684., 550., 845., 422., 785., 750., 824., 700.,
       397., 736., 414., 591., 403., 401., 653., 483., 724., 460., 636.,
       417., 639., 573., 486., 734., 755., 782., 633., 435., 541., 753.,
       498., 533., 515., 814., 405., 693., 477., 846., 499., 463., 431.,
       442., 509., 506., 505., 677., 393., 620., 536., 689., 595., 681.,
       727., 333., 516., 583., 761., 501., 781., 732., 317., 507., 625.,
       466., 530., 391., 701., 473., 601., 469., 376., 430., 490., 294.,
       646., 454., 717., 450., 542., 445., 615., 554., 649., 720., 683.,
       948., 377., 668., 404., 390., 484., 237., 735., 413., 776., 941.,
       443., 725., 421., 402., 387., 638., 726., 602., 560., 468., 426.,
       833., 429., 610., 409., 407., 532., 358., 680., 745., 372., 496.,
       455., 529., 440., 737., 386., 408., 354., 792., 451., 510., 565.,
       815., 808., 827., 809., 459., 467., 843., 797., 695., 326., 458.,
       395., 563., 384., 742., 341., 882., 790., 327., 488., 716., 851.,
       709., 738., 897., 379., 357., 816., 399., 364., 762., 806., 826.,
       754., 351., 611., 485., 432., 853., 810., 898., 751., 365., 526.,
       385., 366., 718., 356., 378., 348., 447., 388., 353., 492., 478.,
       325., 394., 308., 811., 710., 706., 314., 433., 381., 363., 475.,
       328., 298., 337., 361., 582., 383., 323., 369., 345., 803., 368.,
       347., 593., 662., 382., 331., 812., 502., 321., 344., 349., 804.,
       673., 287., 360., 339., 312., 292., 334., 291., 304., 319., 332.,
       284., 340., 355., 343., 311., 338., 359., 322., 310., 836., 336.,
       276., 389., 271., 279., 313., 373.])
```

Podemos obsevar que en esta columna los valores van de 0 a 948. el 75% de los datos se encuentran dentro del rango entre 0 y 675, por lo que utilizar el promedio de la columna para rellenar los Nan Values es bastante util.

In [369]:

```
# Vamos a observar los detalles de la columna finc_sva
train.finc_sva.describe()
```

Out[369]:

```
count    4792.000000
mean        9.231594
std        10.121024
min         0.848822
25%         2.607410
50%         4.752980
75%        12.844623
max       134.781764
Name: finc_sva, dtype: float64
```

In [370]:

```
# medida de rentabilidad del banco generada por el cliente.
train['finc_sva'].unique()
```

Out[370]:

```
array([11.237869, 10.145764, 18.786289, ...,  6.184711, 13.754551,
        9.847391])
```

Podemos observar que los valores en esta columna, van de 0.84 a 134.78. Sin embargo el 75% de los datos, esta en el rango de 0.84 a 12.84. Por lo que es bastante practico utilizar el promedio de la columna 9.23 para rellenar los valores faltantes.

Importante: Debido a que estos datos los vamos para utilizar para entrenar un modelo, las démas variables que contienen valores NAn el máximo es de 8, por lo que podemos eliminar estas filas ya que no se pierde mucha información.

1.3 Manejando valores NaN

In [371]:

```
# Remplazando Valores NaN con promedio de columnas.
train['buro_score_apc'] = train.buro_score_apc.fillna(train.buro_score_apc.mean())
train['finc_sva'] = train.finc_sva.fillna(train.finc_sva.mean())

# Quitando valores NaN
train =train.dropna()

# Revisando Forma de datos
train.shape
```

Out[371]:

```
(4988, 49)
```

Contamos con 4988 filas, lo que es bastante data para entrenar nuestro modelo.

In [372]:

```
# Veamos la columna adming_flag_gerenciado
train.admin_flag_gerenciado.value_counts()
```

Out[372]:

```
0    4988
Name: admin_flag_gerenciado, dtype: int64
```

observamos que el único valor para esta columna es 0. Por lo que realmente no agregar ningún valor a los datos. Vamos a eliminarlo.

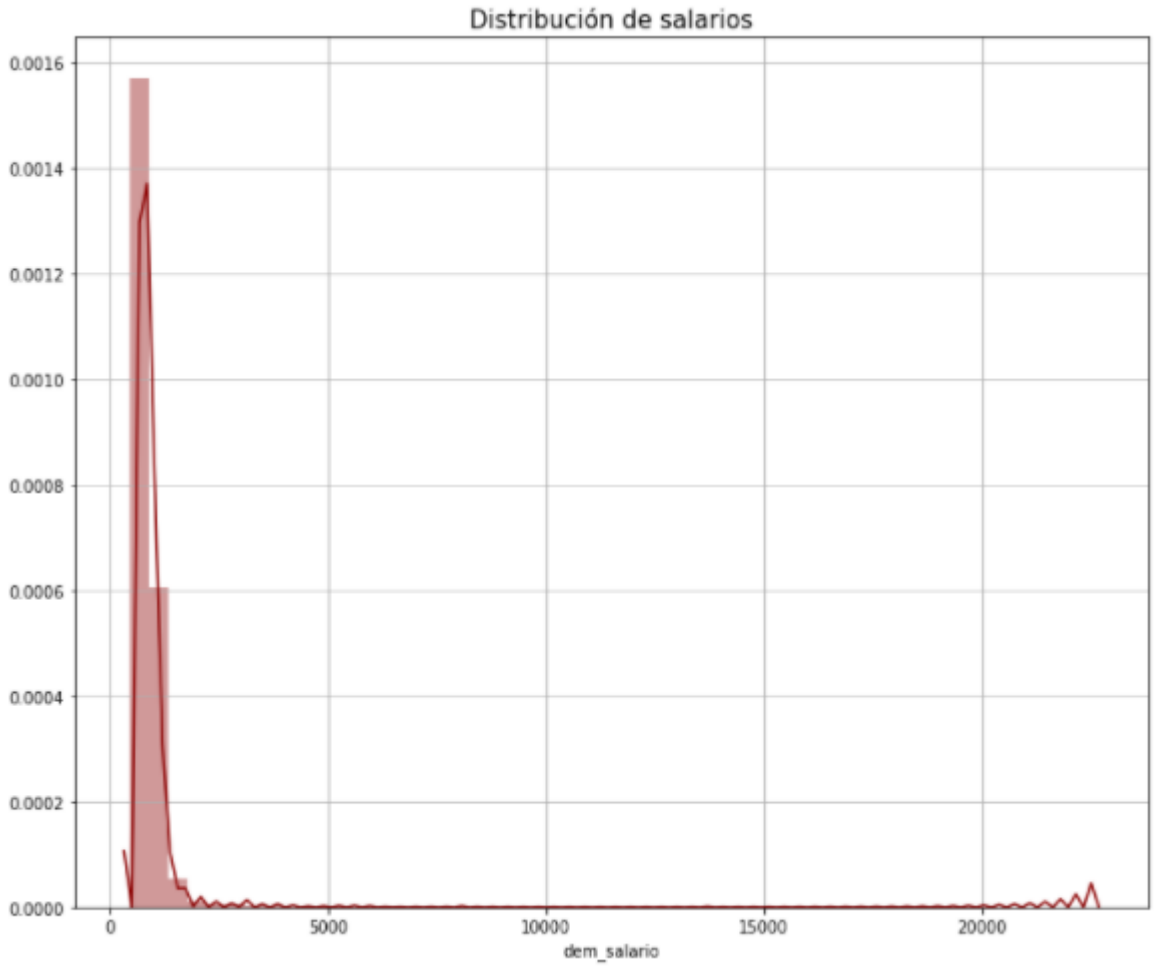
In [373]:

```
# Eliminando columna admin_flag, ya que de esto no depende ningún salario.
train  = train.drop(columns=['admin_flag_gerenciado'])
```

2- Analisis de Variables

En esta sección analizaremos con detenimiento las variables existentes, cúal es su relación con la variable objetivo (dem_salario), y cuales de estas nos ayudan a construir el mejor modelo posible.

Vamos a observar la correlación que existe entre la variable objetivo, y las demás variables.



Desde este momento ya que vemos la distribución de los salarios, se puede anticipar que nuestro modelo será capaz de predecir mejor salarios que esten por debajo de los 1000 dolares. En la celda de detalles estadisticos de igual manera se observa que 75% de los salarios esta por debajo de 937.00 dolares.

In [375]:

```
# Observando correlacion
corr = train.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[375]:

	llave_cod_cliente	admin_antiguedad_banco	buro_creditos_otros_bancos	buro_score_apc	buro_wallet_shar
llave_cod_cliente	1.000000	-0.758254	-0.174291	-0.226514	-0.10068
admin_antiguedad_banco	-0.758254	1.000000	0.209162	0.257553	0.08481
buro_creditos_otros_bancos	-0.174291	0.209162	1.000000	0.190719	-0.14028
buro_score_apc	-0.226514	0.257553	0.190719	1.000000	0.11371
buro_wallet_share	-0.100680	0.084810	-0.140287	0.113717	1.00000
comp_flag_atm	0.138750	-0.192696	-0.055339	-0.141366	-0.28994
comp_flag_bpi	0.060321	-0.077445	-0.069601	-0.014103	-0.07496
comp_flag_cnb	0.026629	-0.035219	-0.056291	-0.059924	-0.00679
comp_flag_pos	0.140629	-0.191609	-0.045547	-0.128065	-0.25509
comp_flag_suc	0.099498	-0.111370	-0.105191	-0.073080	-0.03275

comp_perc_atm	0.093425	-0.136289	-0.005745	-0.117567	-0.24790
comp_perc_canal_fisico	0.006581	-0.017068	-0.044529	-0.046207	-0.01809
comp_perc_cnb	0.020635	-0.009242	-0.029471	-0.015333	0.03822
comp_perc_bpi	0.017952	-0.017846	-0.072618	0.019418	-0.02938
comp_perc_pos	0.128637	-0.175367	-0.072887	-0.093240	-0.21459
comp_perc_suc	0.006630	0.009927	-0.052439	0.037100	0.16081
comp_score_digital	0.031226	-0.044108	-0.052275	-0.007165	-0.04929
comp_txn_atm	0.046648	-0.071765	0.014959	-0.049323	-0.16977
comp_txn_bpi	0.016923	-0.028313	-0.028889	-0.016675	-0.02745
comp_txn_cnb	0.016070	-0.014704	-0.044200	-0.077384	0.00170
comp_txn_pos	0.058464	-0.083888	-0.027355	-0.030330	-0.12387
comp_txn_suc	0.025687	-0.011463	-0.112268	-0.051982	-0.01786
comp_usd_atm_prom	0.028034	-0.044084	0.014034	-0.069825	-0.17064
comp_usd_bpi_prom	0.014458	-0.018785	-0.067479	0.000231	-0.01897

	llave_cod_cliente	admin_antiguedad_banco	buro_creditos_otros_bancos	buro_score_app	buro_wallet_cha
comp_usd_cnb_prom	0.023895	-0.030540	-0.040417	-0.054093	0.0114
comp_usd_pos_prom	0.043077	-0.061655	-0.037305	-0.023753	-0.12030
comp_usd_suc_prom	0.016998	-0.012760	-0.070088	-0.020626	-0.04155
dem_edad	-0.513234	0.568213	0.051861	0.202940	0.12936
dem_planilla	0.056172	-0.086020	0.092854	-0.015156	-0.15383
finc_bal_act	-0.024561	0.025523	0.081947	0.092474	0.44143
finc_bal_pas	-0.002214	0.007150	-0.077002	-0.022194	-0.03774
finc_perc_act_tc	-0.143489	0.117332	0.054907	0.179213	0.69631
finc_perc_pas_tc	0.145151	-0.136585	-0.077358	-0.193323	-0.64141
finc_sva	-0.062186	0.082541	0.039626	0.142747	0.44617
finc_tamano_comercial	-0.024100	0.025926	0.065066	0.085188	0.41905
pdcto_flag_auto	0.032488	-0.024903	0.031512	0.056810	0.08354
pdcto_flag_cta_ahorro	0.028389	0.000487	-0.020894	-0.044334	-0.10484
pdcto_flag_cta_corriente	0.143908	-0.114139	0.057330	-0.067160	-0.16052
pdcto_flag_cta_dpf	-0.022647	0.018747	0.011863	0.005710	-0.00904
pdcto_flag_financomer	-0.005917	-0.009778	0.072033	-0.019296	-0.03883
pdcto_flag_garantizado	-0.018105	0.057285	0.016778	0.006407	-0.00821
pdcto_flag_hipoteca	-0.047958	0.066136	0.086538	0.062046	0.36858
pdcto_flag_pp	-0.108006	0.045082	-0.057889	0.100748	0.58548
pdcto_flag_seguros	-0.048700	0.090615	0.066420	0.168291	0.19025
pdcto_flag_tiene_tdc	-0.057888	0.091496	0.056030	0.170942	0.15085
pdcto_flag_tiene_tdd	0.192912	-0.262945	-0.068724	-0.158426	-0.29334
pdcto_ivc_actual	0.018493	0.007041	0.074994	0.091131	0.38461
dem_salario	-0.131581	0.175915	0.082950	0.161151	-0.02371

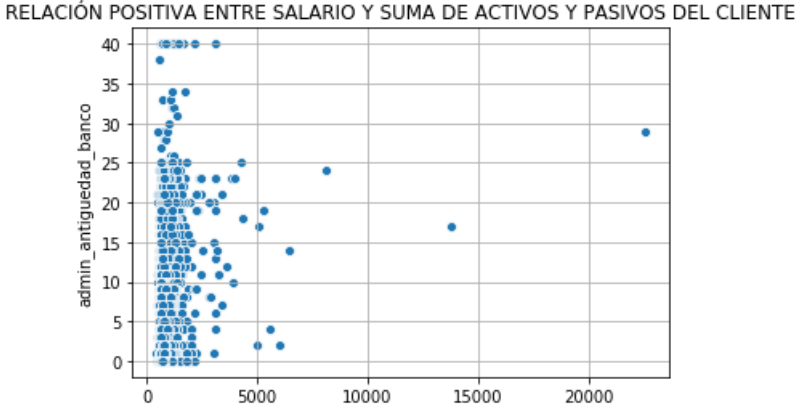
Podemos observar que la mayoria de las variables, si utilizamos este metodo de correlación con respecto a la variable salario, tienen muy poca correlación, la mayoria tiene valores de 0.0 o -0.0.

Vamos a utilizar las variables que tiene valores >= a -0.1 o 0.1 ya que tienen una pequeña correlación positiva o negativa,

Desde este punto podemos observar que linermente es muy dificil realizar un modelo altamente eficaz, ya que ninguna variable tiene una correlación modera o fuerte con el salario.

In [376]:

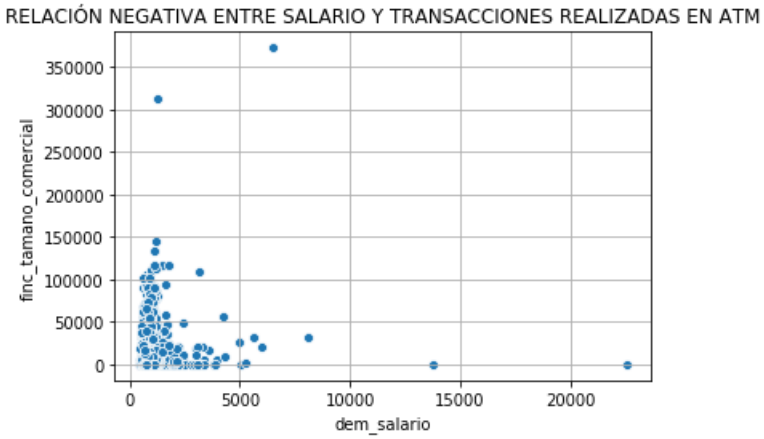
```
# relación entre salario y variable de activos y pasivos del cliente.
sns.scatterplot(data=train, x='dem_salario', y='admin_antiguedad_banco')
plt.title('RELACIÓN POSITIVA ENTRE SALARIO Y SUMA DE ACTIVOS Y PASIVOS DEL CLIENTE')
plt.grid()
plt.show()
```



dem_salario

In [377]:

```
# Relación entre salario y variable de transacciones de atm por el cliente.
sns.scatterplot(data=train, x='dem_salario', y='finc_tamano_comercial')
plt.title('RELACIÓN NEGATIVA ENTRE SALARIO Y TRANSACCIONES REALIZADAS EN ATM')
plt.grid()
plt.show()
```



In [378]:

```
# Variables con correlación mayor.
data = train[['admin_antiguedad_banco',
'buro_score_apc','comp_perc_atm','comp_perc_bpi','comp_score_digital','comp_txn_bpi','comp_usd_bpi_
prom','comp_usd_pos_prom','dem_edad','dem_planilla','finc_bal_act'
              ,'finc_bal_pas','finc_tamano_comercial','pdcto_flag_auto','pdcto_flag_tiene_tdd','dem_
salario']]
```

In [379]:

```
# Valores que tiennen una pequeña correlación ya sea positiva o negativa.
corr_data = data.corr()
corr_data.style.background_gradient(cmap='coolwarm')
```

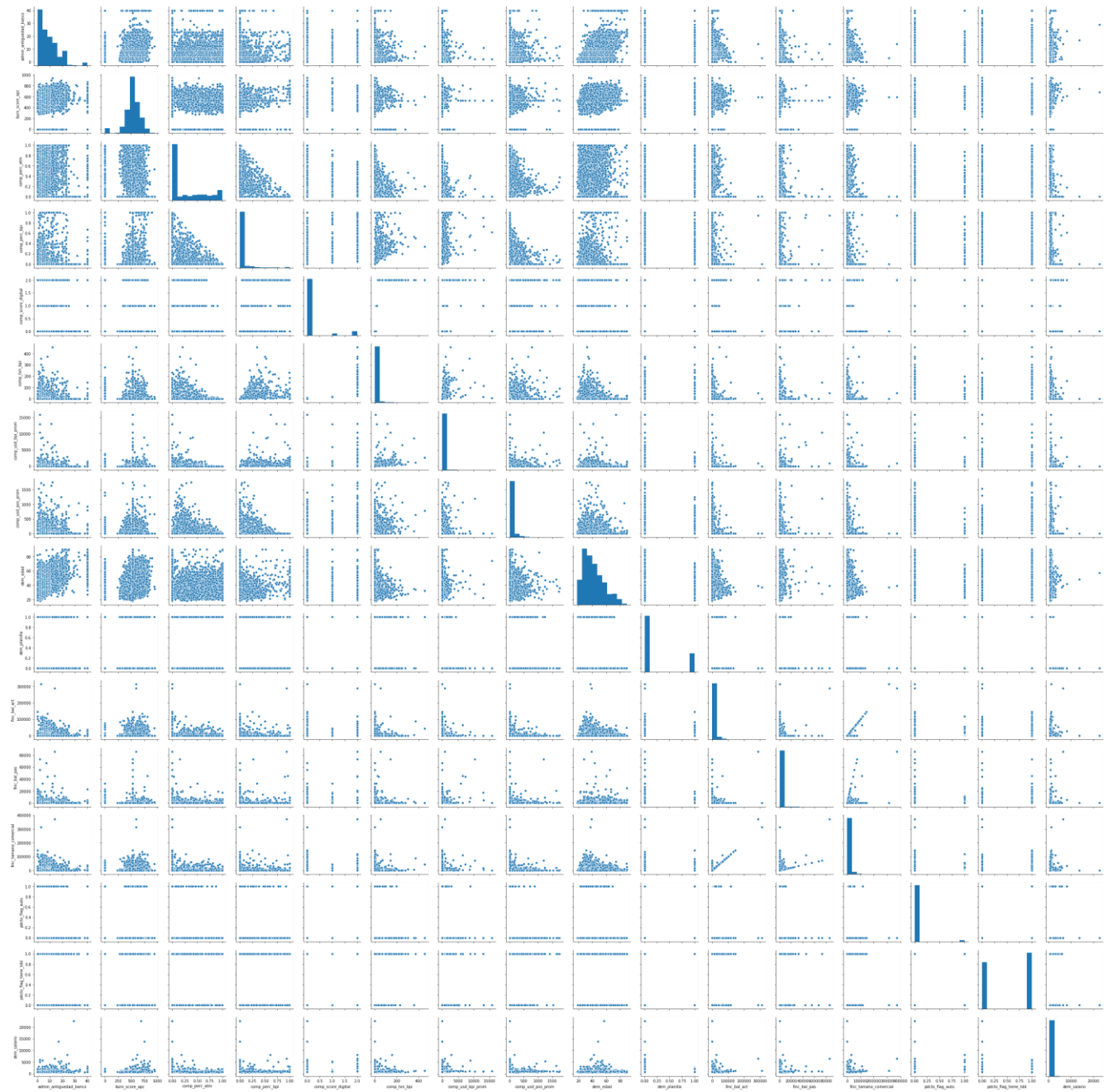
Out[379]:

	admin_antiguedad_banco	buro_score_apc	comp_perc_atm	comp_perc_bpi	comp_score_digital	comp_txn_b
admin_antiguedad_banco	1.000000	0.257553	-0.136289	-0.017846	-0.044108	-0.0283
buro_score_apc	0.257553	1.000000	-0.117567	0.019418	-0.007165	-0.0166
comp_perc_atm	-0.136289	-0.117567	1.000000	-0.096403	-0.040786	-0.0444
comp_perc_bpi	-0.017846	0.019418	-0.096403	1.000000	0.626317	0.5412
comp_score_digital	-0.044108	-0.007165	-0.040786	0.626317	1.000000	0.7706
comp_txn_bpi	-0.028313	-0.016675	-0.044483	0.541287	0.770601	1.0000
comp_usd_bpi_prom	-0.018785	0.000231	-0.052077	0.442866	0.443092	0.4688
comp_usd_pos_prom	-0.061655	-0.023753	0.056246	0.124478	0.438712	0.3965
dem_edad	0.568213	0.202940	-0.149647	-0.049353	-0.113003	-0.0993
dem_planilla	-0.086020	-0.015156	0.593334	0.016089	0.119250	0.0884
finc_bal_act	0.025523	0.092474	-0.188974	0.018053	-0.009551	0.0125
finc_bal_pas	0.007150	-0.022194	-0.035167	0.165151	0.127052	0.0906
finc_tamano_comercial	0.025926	0.085188	-0.188727	0.047448	0.013886	0.0286
pdcto_flag_auto	-0.024903	0.056810	-0.078644	0.047975	0.027859	0.0365
pdcto_flag_tiene_tdd	-0.262945	-0.158426	0.676972	0.055438	0.182614	0.1395
dem_salario	0.175915	0.161151	-0.130686	0.117816	0.109411	0.1021

dem_salario
0.175915
0.161151
-0.130686
0.117816
0.109411
0.102152
0.125597
0.103169
0.122621
-0.115481
0.104222
0.143573
0.126665
0.108696
-0.105596
1.000000

In [380]:

```
# Observando correlación entre variable de salario y predictoras.
sns.pairplot(data)
plt.show()
```



Vamosa eliminar de igual manera variables discretizadas o con contenido binario, ya que no aportan realmente nada al modelo, ni indican alguan tendencia.

In [381]:

```
# Borrando variables que poseen valores binarios o discretizados ya que no aportan un
comportamiento lineal positivo o negativo.
data = data.drop(columns=['comp_score_digital', 'pdcto_flag_tiene_tdd', 'pdcto_flag_auto',
'dem_planilla'])
```

La metrica que vamos a estar utilizando es el **Mean Absolute Percentage error**, para evaluar las predicciones.

MAPE-value	Accuracy of forecast
Less than 10%	Highly Accurate Forecast
11% to 20%	Good Forecast
21% to 50%	Reasonable Forecast
More than 51%	Inaccurate Forecast

Source: Lewis, C.D., 1982.

3- Modelo Predictivos Basado en Correlación

In [382]:

```
# Train _test Split
X = data.drop(columns=['dem_salario'])
y = data['dem_salario']
```

In [383]:

```
# Max Min Scaler
sc = MinMaxScaler()
X = sc.fit_transform(X)
```

In [384]:

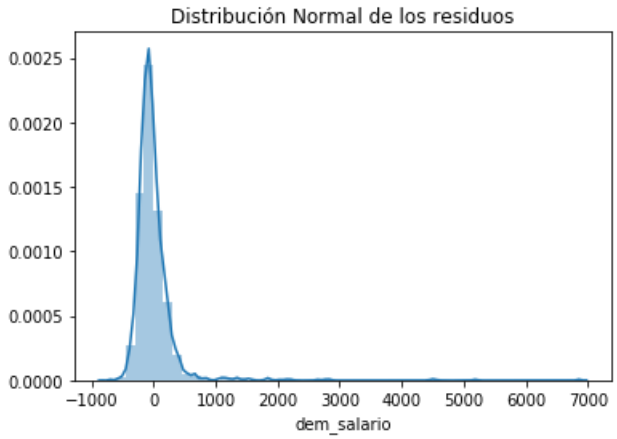
```
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=42)
```

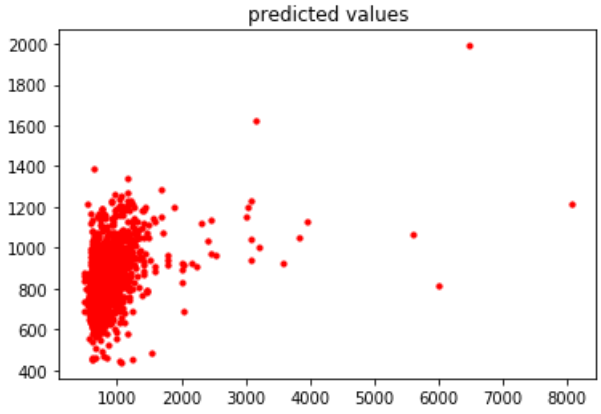
3.1 Regresión lineal

In [385]:

```
# Modelo de Regresión lineal con variables con mayor correlación (aunque baja).

def linear_regression(X_train, X_test, y_train, y_test):
  # Modelo
  lm = LinearRegression()
  lm.fit(X_train, y_train)
  pred = lm.predict(X_test)

  # Metricas
  print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
  print()

  # Durbin-watson
  residual = (y_test - pred)
  print('Durbin-watson:',durbin_watson(residual))

  # Residuos
  sns.distplot(residual)
  plt.title('Distribución Normal de los residuos')
  plt.show()

  # Predicted Values
  plt.scatter(y_test, pred, s=10, color='r')
  plt.title('predicted values')
  plt.show()

  # Dataframe
  prediction = lm.predict(X_test)
  prediction = pd.DataFrame(prediction, columns=['salario_estimado'])


  # Valores actuales.
  y_test_l = pd.DataFrame(y_test)
  y_test_l = y_test_l.reset_index(drop=True)

  # Dataframe predicciones y actuales.
  dataframe = pd.concat([y_test_l, prediction], axis=1)

  print(dataframe.head(15))

  return lm

# Modelo de Regresión linear
lm = linear_regression(X_train, X_test, y_train, y_test)
lm
```

```
MAPE 19.593120758845682 %

Durbin-watson: 2.0634452096991565
```






```
    dem_salario  salario_estimado
0         613.57  715.107923
1        1107.00       1006.382143
2         964.70        959.484878
3         687.40        872.040277
4         661.07        584.284724
5         697.96        681.668672
6        1030.00        717.095022
7         798.72        918.502059
8        1176.09       1059.774046
9        1023.21        742.805155
10       1052.19       1052.731215
11        766.40       1021.150983
12        634.62        661.602410
13        766.40        818.409554
14        670.35        910.341287
```

Out[385]:

```
LinearRegression()
```

Podemos observar que nuestro modelo predice bastante bien, los salarios, con rangos bastantes cercanos, ya que según nuestra metrica tiene alrededor de 20% de error. Errores más comunes son salarios arribas de 1000$ ya que nuestros datos el 75 porciento esta en el rango de 975 dolares o menos, por lo que nuestro modelo le cuesta más predecir salarios altos.

3.2- Redes Neuronales Artificiales

In [386]:

```
# ANN
def ann(X_train, X_test, y_train, y_test):
    model = keras.Sequential()
    model.add(keras.layers.Dense(11, input_shape=(11,)))
    model.add(keras.layers.Dense(1, activation='relu'))
    model.compile(keras.optimizers.Adam(lr=0.1), 'mean_absolute_percentage_error')

    # Compilar modelo
```

```
    history = model.fit(X_train, y_train, batch_size=8, epochs=7)

    # prediction
    ann_prediction = model.predict(X_test)
    ann_prediction = pd.DataFrame(ann_prediction, columns=['salario_estimado'])

    # Valores actuales.
    y_test_ann = pd.DataFrame(y_test)
    y_test_ann = y_test_ann.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    ypred_ann = pd.concat([y_test_ann, ann_prediction], axis=1)

    # Resultado de ANN.
    y_true_ann = ypred_ann['dem_salario']
    y_pred_ann = ypred_ann['salario_estimado']

    print('MAPE:',mean_absolute_percentage_error(y_true=y_true_ann, y_pred=y_pred_ann)*100,'%')
    print()
    print('VALORES ACTUALES VS PREDICCIONES ')
    print(ypred_ann.head(15))

    return model

# llamando función de ANN
ann_model = ann(X_train, X_test, y_train, y_test)
ann_model
```

```
Epoch 1/7
  1/418 [..............................] - ETA: 0s - loss: 99.9832WARNING:tensorflow:Callbacks
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0000s vs
`on_train_batch_end` time: 0.0010s). Check your callbacks.
418/418 [==============================] - 0s 715us/step - loss: 23.4518
Epoch 2/7
418/418 [==============================] - 0s 967us/step - loss: 16.0782
Epoch 3/7
418/418 [==============================] - 0s 721us/step - loss: 16.0065
Epoch 4/7
418/418 [==============================] - 0s 588us/step - loss: 15.9612
Epoch 5/7
418/418 [==============================] - 0s 571us/step - loss: 16.0381
Epoch 6/7
418/418 [==============================] - 0s 641us/step - loss: 15.9748
Epoch 7/7
418/418 [==============================] - 0s 933us/step - loss: 16.0444
MAPE: 16.256078553845775 %

VALORES ACTUALES VS PREDICCIONES
    dem_salario  salario_estimado
0        613.57        713.116760
1       1107.00        813.226868
2        964.70        799.631714
3        687.40        821.487183
4        661.07        697.010071
5        697.96        677.023804
6       1030.00        710.981934
7        798.72        790.316956
8       1176.09        847.746338
9       1023.21        701.526062
10      1052.19        852.047241
11       766.40        827.844666
12       634.62        658.072327
13       766.40        750.766113
14       670.35        804.018005
```

Out[386]:

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x26de307a518>
```

3.3- Random Forest Regressor

In [387]:

```
def random_forest_regressor(X_train, X_test, y_train, y_test):
```

```
    rfr =RandomForestRegressor()
    rfr.fit(X_train, y_train)
    pred = rfr.predict(X_test)

    # Metricas
    print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
    print()

    # Dataframe
    rfr_prediction = rfr.predict(X_test)
    rfr_prediction = pd.DataFrame(rfr_prediction, columns=['salario_estimado'])


    # Valores actuales.
    y_test_rfr = pd.DataFrame(y_test)
    y_test_rfr = y_test_rfr.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    dataframe = pd.concat([y_test_rfr, rfr_prediction], axis=1)

    print(dataframe.head(15))

    return rfr


# LLamando a la función
rfr = random_forest_regressor(X_train, X_test, y_train, y_test)
rfr
```

```
MAPE 18.300383326093428 %

    dem_salario  salario_estimado
0        613.57  744.8485
1       1107.00         1088.0325
2        964.70          978.8074
3        687.40          787.7570
4        661.07          834.3791
5        697.96          681.8382
6       1030.00          750.9257
7        798.72          741.0530
8       1176.09         1491.1040
9       1023.21          707.7917
10      1052.19         1041.2039
11       766.40         1035.4439
12       634.62          653.8232
13       766.40          700.9679
14       670.35          781.1028
```

Out[387]:

```
RandomForestRegressor()
```

4- Modelos Predictivos extrayendo variables con p-valor estadisticamente significativo

In [388]:

```
# Salarios
train.dem_salario.value_counts()
```

Out[388]:

```
618.50     258
626.40     239
766.40     177
687.40     173
791.10     112
          ...
713.69       1
595.54       1
996.60       1
886.25       1
1009.13      1
```

```
Name: dem_salario, Length: 3341, dtype: int64
```

In [389]:

```
def continous_filter(df, low_exclusive = 2, high_inclusive = 15):
    """
    Función que retorna las columnas que tienen valores menores o iguales a las categorias
    """
    list_of_features = []
    for i in df.columns:
        if low_exclusive == high_inclusive:
            if df[i].nunique() <= low_exclusive :
                list_of_features.append(i)
        else:
            if df[i].nunique() <= high_inclusive and df[i].nunique() > low_exclusive :
                list_of_features.append(i)
    return list_of_features
```

In [390]:

```
# elegimos 3 porque no queremos variables binarias ni la de score digital ya que no aportan en nad
a al salario.
remainder_cols = continous_filter(train, 3, len(train))
print('# Variables continuas con más de 3 atributos) = ',len(remainder_cols))
```

```
# Variables continuas con más de 3 atributos) =  29
```

In [391]:

```
n_rows = len(train)

# Atributos que tengan más de 3 clases
train_df_cols = train[remainder_cols]
```

In [392]:

```
# Extrayendo los atributos más importantes.
target_df = train['dem_salario']
# target_df_log = np.log(target_df)
f, p_val = f_regression(train_df_cols,target_df)
```

In [393]:

```
# Extrayendo Valores que tienen el p-valor menor de 0.05 para hacerlo estadisticamente
significativos.
f_reg_df = pd.DataFrame(np.array([f, p_val]).T, index = train_df_cols.columns, columns = ['f-statis
tic', 'p-value'])
continous_stored_features = f_reg_df[f_reg_df['p-value'] < 0.05].sort_values(by = 'f-statistic', as
cending = False)
continous_stored_features
```

Out[393]:

	f-statistic	p-value
dem_salario	2.201465e+17	0.000000e+00
admin_antiguedad_banco	1.592243e+02	5.895625e-36
buro_score_apc	1.329363e+02	2.259270e-30
finc_bal_pas	1.049403e+02	2.192455e-24
llave_cod_cliente	8.784697e+01	1.045706e-20
comp_perc_atm	8.663506e+01	1.909609e-20
finc_tamano_comercial	8.129986e+01	2.713598e-19
comp_usd_bpi_prom	7.991217e+01	5.415967e-19
dem_edad	7.611292e+01	3.598506e-18
comp_perc_bpi	7.018265e+01	6.952672e-17

	f-statistic	p-value
finc_bal_act	5.475344e+01	1.594916e-13
comp_usd_pos_prom	5.364123e+01	2.791977e-13
comp_txn_bpi	5.257788e+01	4.770162e-13
finc_perc_pas_tc	3.038276e+01	3.725131e-08
comp_txn_pos	1.474333e+01	1.247096e-04
finc_sva	1.380410e+01	2.051138e-04
comp_usd_suc_prom	9.424878e+00	2.152131e-03
comp_txn_atm	7.038270e+00	8.003899e-03
comp_txn_suc	4.332707e+00	3.743764e-02
finc_perc_act_tc	4.199956e+00	4.047724e-02

Con este paso hemos encontrado las variables estadisticamente significativas con respecto a la variable objetivo, y que su p valor sea < 0.05

In [394]:

```
# construyendo dataframe con los valores estadisticamente sifnicativos.
p_train = train[['admin_antiguedad_banco', 'buro_score_apc', 'finc_bal_pas', 'comp_perc_atm','finc_
tamano_comercial','comp_usd_bpi_prom','dem_edad','comp_perc_bpi','finc_bal_act','comp_usd_pos_prom'
,'comp_txn_bpi','finc_perc_pas_tc','comp_txn_pos','finc_sva','comp_usd_suc_prom','comp_txn_atm','co
mp_txn_suc','finc_perc_act_tc','dem_salario']]
```

In [395]:

```
# Train _test Split
X_p_train = p_train.drop(columns=['dem_salario'])
y_p_train = p_train['dem_salario']

# Max Min Scaler
sc = MinMaxScaler()
X_p_train = sc.fit_transform(X_p_train)

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X_p_train,y_p_train, test_size=0.33, random_sta
te=42)
```

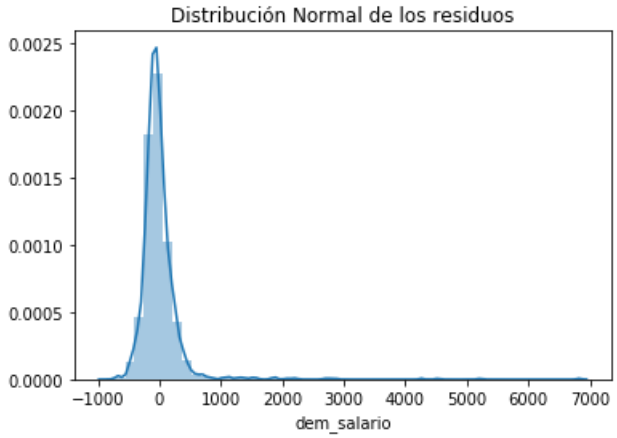
4.1- Regresión lineal

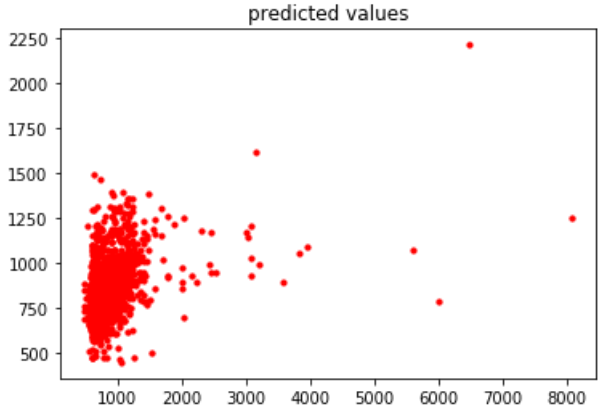
In [396]:

```
# Modelo de Regresión lineal con valores estadisticamente sifnicantes.
def linear_regression_p(X_train, X_test, y_train, y_test):
  # Modelo
  lmp = LinearRegression()
  lmp.fit(X_train, y_train)
  pred = lmp.predict(X_test)

  # Metricas
  print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
  print()

    # Durbin-watson
  residual = (y_test - pred)
  print('Durbin-watson:',durbin_watson(residual))

  # Residuos
  sns.distplot(residual)
  plt.title('Distribución Normal de los residuos')
  plt.show()

  # Predicted Values
  plt.scatter(y_test, pred, s=10, color='r')
  plt.title('predicted values')
  plt.show()
  # Dataframe
  prediction = lmp.predict(X_test)
  prediction = pd.DataFrame(prediction, columns=['salario_estimado'])
```

```
  # Valores actuales.
  y_test_p = pd.DataFrame(y_test)
  y_test_p = y_test_p.reset_index(drop=True)

  # Dataframe predicciones y actuales.
  dataframe = pd.concat([y_test_p, prediction], axis=1)

  print(dataframe.head(15))

  return lm

# Modelo de Regresión linear
lmp = linear_regression_p(X_train, X_test, y_train, y_test)
lmp
```

```
MAPE 19.792274797647185 %

Durbin-watson: 2.067799870062239
```





	dem_salario	salario_estimado
0	613.57	1076.868121
1	1107.00	974.590190
2	964.70	1297.526493
3	687.40	885.286520
4	661.07	572.095569
5	697.96	686.617612
6	1030.00	695.933272
7	798.72	871.320807
8	1176.09	1048.039852
9	1023.21	743.835166
10	1052.19	1040.466386
11	766.40	978.009638
12	634.62	665.544205
13	766.40	778.715956
14	670.35	902.063859

Out[396]:

```
LinearRegression()
```

4.2- ANN

In [397]:

```
# ANN
def ann(X_train, X_test, y_train, y_test):
    model = keras.Sequential()
    model.add(keras.layers.Dense(11, input_shape=(18,)))
    model.add(keras.layers.Dense(1, activation='relu'))
    model.compile(keras.optimizers.Adam(lr=0.1), 'mean_absolute_percentage_error')

    # Compilar modelo
    history = model.fit(X_train, y_train, batch_size=8, epochs=7)

    # prediction
    ann_prediction = model.predict(X_test)
    ann_prediction = pd.DataFrame(ann_prediction, columns=['salario_estimado'])


    # Valores actuales.
    y_test_ann = pd.DataFrame(y_test)
    y_test_ann = y_test_ann.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    ypred_ann = pd.concat([y_test_ann, ann_prediction], axis=1)

    # Resultado de ANN.
    y_true_ann = ypred_ann['dem_salario']
    y_pred_ann = ypred_ann['salario_estimado']

    print('MAPE:',mean_absolute_percentage_error(y_true=y_true_ann, y_pred=y_pred_ann)*100,'%')
    print()
    print('VALORES ACTUALES VS PREDICCIONES ')
    print(ypred_ann.head(15))

    return model

# llamando función de ANN
ann_model_p = ann(X_train, X_test, y_train, y_test)
ann_model_p
```

```
Epoch 1/7
418/418 [==============================] - 0s 839us/step - loss: 22.7030
Epoch 2/7
418/418 [==============================] - 0s 834us/step - loss: 16.5980
Epoch 3/7
418/418 [==============================] - 0s 872us/step - loss: 16.3012
Epoch 4/7
418/418 [==============================] - 0s 996us/step - loss: 16.10120s - loss: 16.10
Epoch 5/7
418/418 [==============================] - 0s 970us/step - loss: 16.1072
Epoch 6/7
418/418 [==============================] - 0s 754us/step - loss: 16.0046
Epoch 7/7
418/418 [==============================] - 0s 750us/step - loss: 15.8659
MAPE: 17.149599423612365 %

VALORES ACTUALES VS PREDICCIONES
    dem_salario  salario_estimado
0         613.57        749.352478
1        1107.00        852.368164
2         964.70        866.640320
3         687.40        863.071899
4         661.07        713.512146
5         697.96        703.451172
6        1030.00        701.069641
7         798.72        816.909729
8        1176.09        929.829895
9        1023.21        731.817627
10       1052.19        933.359924
11        766.40        876.559448
12        634.62        667.387878
```

```
13       766.40         758.182800
14       670.35         867.136719
```

Out[397]:

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x26d83faa048>
```

4.3 Random Forest Regressor

In [398]:

```
def random_forest_regressor(X_train, X_test, y_train, y_test):
     rfr_r = RandomForestRegressor()
     rfr_r.fit(X_train, y_train)
     pred =rfr_r.predict(X_test)

     # Metricas
     print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
     print()

     # Dataframe
     rfr_prediction = rfr_r.predict(X_test)
     rfr_prediction = pd.DataFrame(rfr_prediction, columns=['salario_estimado'])


     # Valores actuales.
     y_test_rfr = pd.DataFrame(y_test)
     y_test_rfr = y_test_rfr.reset_index(drop=True)

     # Dataframe predicciones y actuales.
     dataframe = pd.concat([y_test_rfr, rfr_prediction], axis=1)

     print(dataframe.head(15))

     return rfr_r


# LLamando a la función
rfr_r = random_forest_regressor(X_train, X_test, y_train, y_test)
rfr_r
```

```
MAPE 17.710036707757475 %

    dem_salario  salario_estimado
0        613.57  722.8484
1       1107.00         1099.7028
2        964.70         1026.4403
3        687.40          806.2141
4        661.07          724.8845
5        697.96          680.9762
6       1030.00          745.5852
7        798.72          877.2321
8       1176.09         1704.6529
9       1023.21          678.8778
10      1052.19         1002.3214
11       766.40         1269.7610
12       634.62          680.5500
13       766.40          689.8028
14       670.35          862.4503
```

Out[398]:

```
RandomForestRegressor()
```

5 Modelo utilizando todos los atributos.

In [399]:

```
# Train _test Split
X = train.drop(columns=['dem_salario'])
y = train['dem_salario']
```

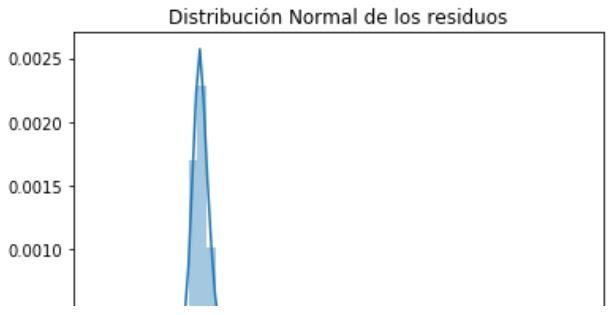
```
# Max Min Scaler
sc = MinMaxScaler()
X = sc.fit_transform(X)

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=42)
```

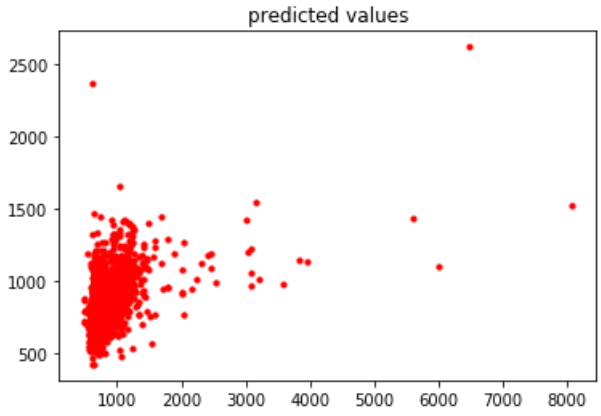
5.1- Linear regression

In [400]:

```
def linear_regression(X_train, X_test, y_train, y_test):
  # Modelo
  lm_all = LinearRegression()
  lm_all.fit(X_train, y_train)
  pred = lm_all.predict(X_test)

  # Metricas
  print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
  print()

   # Durbin-watson
  residual = (y_test - pred)
  print('Durbin-watson:',durbin_watson(residual))

  # Residuos
  sns.distplot(residual)
  plt.title('Distribución Normal de los residuos')
  plt.show()

  # Predicted Values
  plt.scatter(y_test, pred, s=10, color='r')
  plt.title('predicted values')
  plt.show()

  # Dataframe
  prediction = lm_all.predict(X_test)
  prediction = pd.DataFrame(prediction, columns=['salario_estimado'])


  # Valores actuales.
  y_test_l = pd.DataFrame(y_test)
  y_test_l = y_test_l.reset_index(drop=True)

  # Dataframe predicciones y actuales.
  dataframe = pd.concat([y_test_l, prediction], axis=1)

  print(dataframe.head(15))

  return lm_all



# Llammando a regresion lineal con todos los datos
lm_all = linear_regression(X_train, X_test, y_train, y_test)
lm_all
```

```
MAPE 18.794119397621117 %

Durbin-watson: 2.0641758380298327
```





	dem_salario	salario_estimado
0	613.57	1086.204545
1	1107.00	1092.268500
2	964.70	1321.048470
3	687.40	835.983037
4	661.07	519.555992
5	697.96	669.021290
6	1030.00	739.645365
7	798.72	880.732150
8	1176.09	1069.646524
9	1023.21	733.325761
10	1052.19	1045.586163
11	766.40	906.445699
12	634.62	599.180133
13	766.40	792.364030
14	670.35	919.808891

Out[400]:

```
LinearRegression()
```

5.2- Artificial neural network

In [401]:

```
# ANN
def ann(X_train, X_test, y_train, y_test):
    model_all = keras.Sequential()
    model_all.add(keras.layers.Dense(11, input_shape=(47,)))
    model_all.add(keras.layers.Dense(1, activation='relu'))
    model_all.compile(keras.optimizers.Adam(lr=0.1), 'mean_absolute_percentage_error')

    # Compilar modelo
    history = model_all.fit(X_train, y_train, batch_size=8,epochs=7)

    # prediction
    ann_prediction = model_all.predict(X_test)
    ann_prediction = pd.DataFrame(ann_prediction,columns=['salario_estimado'])


    # Valores actuales.
    y_test_ann = pd.DataFrame(y_test)
    y_test_ann = y_test_ann.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    ypred_ann = pd.concat([y_test_ann, ann_prediction], axis=1)

    # Resultado de ANN.
    y_true_ann = ypred_ann['dem_salario']
    y_pred_ann = ypred_ann['salario_estimado']
```

```
    print('MAPE:',mean_absolute_percentage_error(y_true=y_true_ann, y_pred=y_pred_ann)*100,'%')
    print()
    print('VALORES ACTUALES VS PREDICCIONES ')
    print(ypred_ann.head(15))

    return model all


# Modelo con todas las variables.
model_all = ann(X_train, X_test, y_train, y_test)
model_all
```

```
Epoch 1/7
  1/418 [..............................] - ETA: 0s - loss: 99.9274WARNING:tensorflow:Callbacks
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0000s vs
`on_train_batch_end` time: 0.0010s). Check your callbacks.
418/418 [==============================] - 0s 885us/step - loss: 21.9130
Epoch 2/7
418/418 [==============================] - 0s 1ms/step - loss: 16.8018
Epoch 3/7
418/418 [==============================] - 0s 771us/step - loss: 16.3088
Epoch 4/7
418/418 [==============================] - 0s 816us/step - loss: 15.9862
Epoch 5/7
418/418 [==============================] - 0s 721us/step - loss: 15.7805
Epoch 6/7
418/418 [==============================] - 0s 764us/step - loss: 15.6542
Epoch 7/7
418/418 [==============================] - 0s 830us/step - loss: 15.7214
MAPE: 15.735598460097611 %

VALORES ACTUALES VS PREDICCIONES
    dem_salario  salario_estimado
0         613.57        640.565247
1        1107.00        904.208984
2         964.70        781.449097
3         687.40        782.896362
4         661.07        562.036255
5         697.96        592.679565
6        1030.00        690.999084
7         798.72        777.897400
8        1176.09        846.430237
9        1023.21        650.245117
10       1052.19        852.490051
11        766.40        730.880920
12        634.62        557.931702
13        766.40        719.623596
14        670.35        771.098572
```

Out[401]:

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x26dd49b20f0>
```

5.3- random Forest Regressor

In [402]:

```
def random_forest_regressor(X_train, X_test, y_train, y_test):
     rfr_f = RandomForestRegressor()
     rfr_f.fit(X_train, y_train)
     pred =rfr_f.predict(X_test)

     # Metricas
     print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
     print()

     # Dataframe
     rfr_prediction = rfr_f.predict(X_test)
     rfr_prediction = pd.DataFrame(rfr_prediction, columns=['salario_estimado'])


     # Valores actuales.
     y_test_rfr =pd.DataFrame(y_test)
```

```
    y_test_rfr = y_test_rfr.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    dataframe = pd.concat([y_test_rfr, rfr_prediction], axis=1)

    print(dataframe.head(15))

    return rfr f


# LLamando a la función
rfr_f = random_forest_regressor(X_train, X_test, y_train, y_test)
rfr_f
```

```
MAPE 16.39854707901855 %

    dem_salario  salario_estimado
0        613.57  673.6671
1       1107.00         1134.7930
2        964.70          967.1360
3        687.40          777.8081
4        661.07          687.3174
5        697.96          660.9271
6       1030.00          739.0576
7        798.72          910.3383
8       1176.09         1356.4245
9       1023.21          677.6804
10      1052.19         1203.7558
11       766.40         1202.6790
12       634.62          666.3803
13       766.40          732.6373
14       670.35          859.6759
```

Out[402]:

```
RandomForestRegressor()
```

6 analísis Intuitivo

Vamos a utilizar las variables que muestran directamente el valor de activos o pasivos del cliente sin saber cuales estos son, ya que estos afectan direcamente cualquier ingreso que tenga el cliente.

In [403]:

```
intuitivo = train[['finc_bal_pas', 'finc_bal_act', 'finc_tamano_comercial','dem_salario']]
```

In [404]:

```
# Train _test Split
X_intuitivo = intuitivo.drop(columns=['dem_salario'])
y_intuitivo = intuitivo['dem_salario']

# Max Min Scaler
sc = MinMaxScaler()
X_intuitivo = sc.fit_transform(X_intuitivo)

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X_intuitivo,y_intuitivo, test_size=0.33, random
_state=42)
```

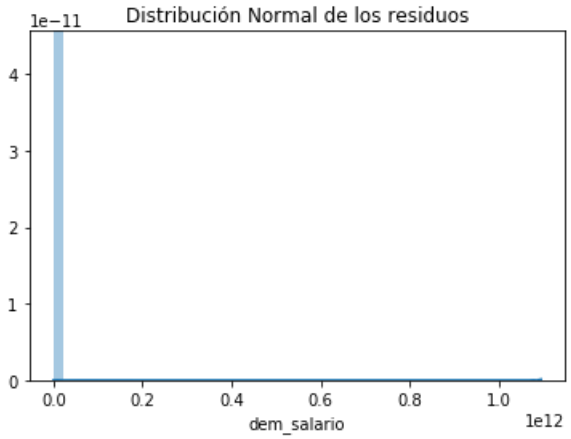
6.1- Regresión lineal

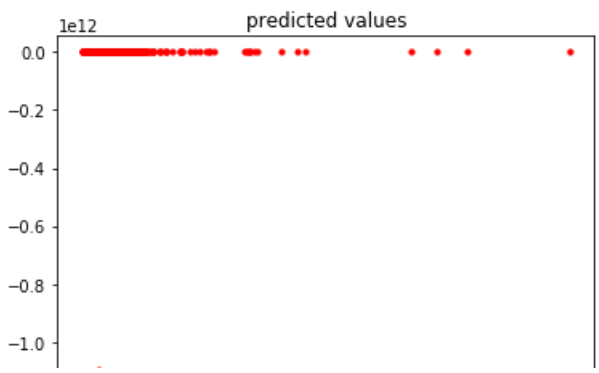
In [405]:

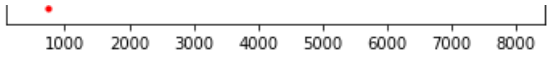
```
def linear_regression(X_train, X_test, y_train, y_test):
  # Modelo
  lm_in = LinearRegression()
  lm_in.fit(X_train, y_train)
  pred = lm_in.predict(X_test)
```

```
  # Metricas
  print('MAPE',mean_absolute_percentage_error(y_true=y_test, y_pred=pred)*100,'%')
  print()

   # Durbin-watson
  residual = (y_test - pred)
  print('Durbin-watson:',durbin_watson(residual))

  # Residuos
  sns.distplot(residual)
  plt.title('Distribución Normal de los residuos')
  plt.show()

  # Predicted Values
  plt.scatter(y_test, pred, s=10, color='r')
  plt.title('predicted values')
  plt.show()

  # Dataframe
  prediction = lm_in.predict(X_test)
  prediction = pd.DataFrame(prediction, columns=['salario_estimado'])


  # Valores actuales.
  y_test_l = pd.DataFrame(y_test)
  y_test_l = y_test_l.reset_index(drop=True)

  # Dataframe predicciones y actuales.
  dataframe = pd.concat([y_test_l, prediction], axis=1)

  print(dataframe.head(15))

  return lm_in



# Llammando a regresion lineal con todos los datos
lm_in = linear_regression(X_train, X_test, y_train, y_test)
lm_in
```

```
MAPE 88673370.93072397 %

Durbin-watson: 2.0000000004650826
```







	dem_salario	salario_estimado
0	613.57	831.673875
1	1107.00	873.673875
2	964.70	831.673875
3	687.40	937.173875
4	661.07	831.815599
5	697.96	833.931688
6	1030.00	851.298875
7	798.72	841.517625
8	1176.09	831.674088
9	1023.21	834.726610
10	1052.19	853.861375
11	766.40	841.173875
12	634.62	832.695360
13	766.40	843.673875
14	670.35	831.881761

Out[405]:

```
LinearRegression()
```

6.2 Redes neuronales

In [406]:

```
# ANN
def ann(X_train, X_test, y_train, y_test):
    model_in = keras.Sequential()
    model_in.add(keras.layers.Dense(3, input_shape=(3,)))
    model_in.add(keras.layers.Dense(1, activation='relu'))
    model_in.compile(keras.optimizers.Adam(lr=0.1), 'mean_absolute_percentage_error')

    # Compilar modelo
    history = model_in.fit(X_train, y_train, batch_size=8, epochs=7)

    # prediction
    ann_prediction = model_in.predict(X_test)
    ann_prediction = pd.DataFrame(ann_prediction, columns=['salario_estimado'])


    # Valores actuales.
    y_test_ann = pd.DataFrame(y_test)
    y_test_ann = y_test_ann.reset_index(drop=True)

    # Dataframe predicciones y actuales.
    ypred_ann = pd.concat([y_test_ann, ann_prediction], axis=1)

    # Resultado de ANN.
    y_true_ann = ypred_ann['dem_salario']
    y_pred_ann = ypred_ann['salario_estimado']

    print('MAPE:',mean_absolute_percentage_error(y_true=y_true_ann, y_pred=y_pred_ann)*100,'%')
    print()
    print('VALORES ACTUALES VS PREDICCIONES ')
    print(ypred_ann.head(15))

    return model_in


# Modelo con todas las variables.
model_in = ann(X_train, X_test, y_train, y_test)
model_in
```

```
Epoch 1/7
418/418 [==============================] - 0s 816us/step - loss: 31.7250
Epoch 2/7
418/418 [==============================] - 0s 587us/step - loss: 17.3468
Epoch 3/7
418/418 [==============================] - 0s 670us/step - loss: 17.4343
Epoch 4/7
```

```
418/418 [==============================] - 0s 625us/step - loss: 17.3472
Epoch 5/7
418/418 [==============================] - 0s 625us/step - loss: 17.3598
Epoch 6/7
418/418 [==============================] - ETA: 0s - loss: 17.39 - 0s 539us/step - loss: 17.3617
Epoch 7/7
418/418 [==============================] - 0s 528us/step - loss: 17.3597
MAPE: 17.38825516470161 %

VALORES ACTUALES VS PREDICCIONES
    dem_salario  salario_estimado
0        613.57        691.386108
1       1107.00        732.491150
2        964.70        691.386108
3        687.40        794.416138
4        661.07        691.552979
5        697.96        694.043152
6       1030.00        710.622437
7        798.72        701.030334
8       1176.09        691.386353
9       1023.21        694.979614
10      1052.19        717.517700
11       766.40        700.661682
12       634.62        692.588684
13       766.40        703.134766
14       670.35        691.630798
```

Out[406]:

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x26de2c6c2b0>
```

Los modelos creados con estas variables estan muy sesgados, por lo que las predicciones son bastante malas.

Base Prueba

En esta sección vamos a utilizar el modelo predictivo creado para, predecir el salario de la base de datos de prueba ó test (base_prueba).

In [407]:

```
# Creando Test para predecir con variables
test_data = test[['admin_antiguedad_banco',
'buro_score_apc','comp_perc_atm','comp_perc_bpi','comp_txn_bpi','comp_usd_bpi_prom','comp_usd_pos_p
rom','dem_edad','finc_bal_act'
            ,'finc_bal_pas','finc_tamano_comercial']]

# Guardadno Key del cliente.
costumer_key = test[['llave_cod_cliente']]
```

In [408]:

```
# Replacing NaN Values with mean
test_data['buro_score_apc'] = test_data.buro_score_apc.fillna(test_data.buro_score_apc.mean())
test_data['comp_txn_bpi'] = test_data.comp_txn_bpi.fillna(test_data.comp_txn_bpi.mean())
test_data['comp_usd_bpi_prom'] = test_data.comp_usd_bpi_prom.fillna(test_data.comp_usd_bpi_prom.me
an())
test_data['comp_usd_pos_prom'] = test_data.comp_usd_pos_prom.fillna(test_data.comp_usd_pos_prom.me
an())
```

In [409]:

```
# Normalizando los valores de las columnas elegidas para mejorar el accuracy del modelo.
test_data = sc.fit_transform(test_data)
```

In [410]:

```
# Creando data alternativa para trabajar.
df = test_data
```

In [411]:

```
# Aplicando modelo de regresión lineal.
test_predictions = lm.predict(df)

# Resultado de predicciones de salarios.
salarios = pd.DataFrame(test_predictions)
salarios.columns = ['dem_salario']

# Archivo final
base_prueba_evaluado = costumer_key.join(salarios)
```

In [415]:

```
# Vista previa de archivo evaluado. LR
base_prueba_evaluado.head(10)
```

Out[415]:

	llave_cod_cliente	dem_salario
0	484	975.149764
1	622	1036.510460
2	845	879.863613
3	884	881.502082
4	961	918.361135
5	1247	1000.352854
6	1382	1125.415205
7	1391	875.253912
8	1410	958.739611
9	1425	1005.197805

In [422]:

```
# A CSV de jupyter Notebook a maquína local.
base_prueba_evaluado.to_csv(r'C:\\Users\\Asus\\Desktop\\BANISTMO\\base_prueba_evaluado.csv',  index
=False, sep=';')
```

Al abrir archivo CSV el separador debe ser **" ; "** , ya que con "," se mezclaba con la coma decimal del monto.

6- Conclusiones

La mayoria de nuestros modelos han entregado un MAPE entre 15% - 20% de error, el cual es bastante bueno. Sin embargo, los datos suministrados no son buenos para predecir la variable del salario, ya que no tienen ninguna correlación tan fuerte con la variable objetivo.

En esta ocasión, no ha sido por modelos, sino por baja calidad de datos, que nuestro modelo no se puede mejorar.

Existen algunas técnicas de transformación como logaritmicas entre otras que siven para mejorar la correlación, pero en este caso al ser tan baja no contribuia o mejoraba demasiado el modelo.

Hemos elegido el modelo de regresión lineal con selección de atributos, ya que este contiene menor información de variables, y es casi tan bueno como los demás modelos realizados, sin variables que no aportan nada al modelo, como pudiese ser en el caso del modelo con todas las variables. O en caso de las redes neuronales ya que necesitan un poco más de poder computacional y su tiempo de entrenamiento es un poco más lento. Además que hemos realizadoe el test de normalidad de los residuos y el test de Durbin watson, y los residuos estan normalmente distribuidos y el resultado del test ha sido de 2.06, lo que demuestra que no hay problemas de multicolinealidad (valores buenos entre 1.5 y 2.5).

Nuestro modelo es bastante bueno para predecir salarios debajo de los 1000 dolares, ya que la mayoria de los datos (75 porciento) de los datos de entrenamiento estaban por debajo de 975 dolares.

El modelo intuitivo realmente no es tan bueno, ya que a pesar de que los resultados son parecidos a los demás, estaban muy sesgados a predecir valores bajos de salarios.

La cantidad de datos tampoco era suficiente como para realizar un modelo más robusto-