Simulacion Crime & Punishment

April 10, 2018

1 Crime and Punishment: Does it pay to punish?

1.1 Modelo propuesto

Se representa un modelo donde los agentes pueden cometer crímenes según su nivel de honestidad y el beneficio que puede llegar a otorgarles, también teniendo en cuenta la probabilidad que tienen de ser castigados.

```
In [1]: # import de librerías
        import pandas as pd
        import numpy as np
        from math import exp
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        from datetime import datetime, timedelta
        import random

        %matplotlib inline

        pd.options.mode.chained_assignment = None  # default='warn'
```

1.2 Configuración de la simulación

Se definen a continuación los valores necesarios para realizar la simulación deseada: - Cantidad de meses: 240 - Cantidad de agentes: 40 - Agentes libres: Todos - Multa sobre el botín: 25% - Mínimo de agentes corruptos inicialmente: 10

```
In [2]: np.random.seed(0)
        sim_months = 240
        sim_agents = 1000
        free_agents = sim_agents
        fine_factor = 0.25
        honesty_min = 0
        honesty_max = 100
        min_initial_corrupted = 10
```

1.3 Funciones simuladoras

A continuación se especifican las diversas funciones necesarias para realizar la carga inicial de agentes y sus sucesivas simulaciones para cada período a analizar.

1.3.1 Salario (Wage)

Cada agente recibe mensualmente su salario. Se designa aleatoriamente desde una distribución triangular Pw.
W: [1 .. 100]
Wm = Wmin + (Wmax - Wmin) / 3

```
In [3]: def simulate_wages(sim_agents):
            wage_min = 1
            wage_max = 100
            wage_avg = wage_min + ( wage_max - wage_min ) / 3
            wage_mode = wage_avg * 3 - ( wage_min + wage_max )
            return np.random.triangular(wage_min, wage_mode, wage_max, size=sim_agents)
```

1.3.2 Honestidad

El nivel de Honestidad es variable y se modifica cada mes. Depende de los crímemes detectados.

La función de distribución es triangular y se asume que la mayoría de las personas son honestas (moda = max).

Inicialmente, se define una cantidad mínima de agentes con Hmin.

H: [0 .. 100]

```
In [4]: def simulate_honesty(sim_agents):
            honesty_mode = 100
            #honesty_avg = ( honesty_min + honesty_mode + honesty_max ) / 3
            values = np.random.triangular(honesty_min, honesty_mode, honesty_max, size=sim_ager
            return np.append(values, min_initial_corrupted * [ honesty_min ] )
```

La siguiente función permite modificar la Honestidad según el perído simulado. Por lo tanto ajustará los valores dependiendo de los crímenes ocurridos y los detectados.

```
In [5]: def get_adjusted_honesty(agents_honesty, total_crimes, crimes_punished):
            max_change_honesty = 10
            temp = agents_honesty + ( 2 * crimes_punished - total_crimes ) * max_change_honesty
            temp[ temp < honesty_min ] = honesty_min
            temp[ temp > honesty_max ] = honesty_max
            return temp
```

1.3.3 Botín (booty)

El valor adquirido frente a un crimen, se determina aleatoriamente desde una función de probabilidad uniforme.

S: [0 .. 10 * Wm]

```
In [6]: def simulate_booty(wage_avg, sim_agents):
            booty_min = 0
            booty_max = 10 * wage_avg
            return np.random.uniform(booty_min, booty_max, size=sim_agents)
```

1.3.4 Prisión y multas (imprisonment and fines)

Se aplica un tiempo de prisión proporsional al botín al agente que se captura cometiendo un crimen, se le quita el botín y se le aplica una multa proporcional al botín.

```
In [7]: def get_prision_time(booty, wage_avg):
            return 1 + booty / wage_avg
        def get_fine(booty, wealth):
            # fine factor is global
            fine = fine_factor * booty
            #fine[ (fine + booty) > wealth ] = wealth
            return fine
```

1.3.5 Probabilidad de captura

La probabilidad de ser capturado es proporcional al botín del crimen, por lo cual crímenes más importantes son más probables de ser detectados.

Para definir la función de probabilidad (logística) basada en el botín, se utiliza una probabilidad predefinida para crímenes menores (P0) y otra para crímenes mayores (P1).

Existen dos escenarios según cómo se especifiquen P0 y P1: - P0 < P1: Situación razonable donde los crímenes menores se detectan con menor probabilidad que los mayores - P0 > P1: Situación indeseable donde es más probable castigar crímenes menores que los mayore

```
In [8]: # P0
        small_crime_prob = 0.2
        # P1
        major_crime_prob = 0.8
```

```
In [9]: def get_capture_prob(booty,b_avg):
            return major_crime_prob + ( (small_crime_prob - major_crime_prob) / (1 + exp((booty
```

Se agrega también una función que simula la captura de los agentes que cometieron crimen, según la probabilidad previamente calculada.

```
In [10]: def get_captured_sim(capture_prob):
             return np.random.rand(capture_prob.size) < capture_prob
```

1.3.6 Utilidad esperada

La utilidad esperada puede separarse en la utilidad esperada si se comete un crimen o si no se lo hace. Por lo tanto, si la utilidad esperada cuando se comete un crimen es superior a la utilidad esperada cuando no se lo hace, entonces el agente se inclina a comenter el crimen.

U = Ucrimen - Unocrimen

- Si U es positiva => el agente comente crimen

- Si U <= 0 => el agente no comete crimen

```
In [11]: def get_utilities(capture_prob, booty, prision_time, fine, wage, honesty):
             return ( (1 - capture_prob) * (booty + prision_time * wage) - capture_prob * fine
```

2 Simulación del modelo

2.1 Carga inicial

Se simulan los salarios (simulate_wages), la honestidad (simulate_honesty) y el botín (simulate_booty) de cada agente.

Luego se asigna una riqueza inicial de 12 meses de salario a cada agente.

```
In [12]: def initialize_model(df_agents, df_simulated_month):
             df_agents.drop(df_agents.index, inplace=True)
             df_simulated_month.drop(df_simulated_month.index, inplace=True)

             # Wages
             df_agents['WAGE'] = simulate_wages(sim_agents)
             # Honesty
             df_agents['HONESTY'] = simulate_honesty(sim_agents)
             # Wealth
             df_agents['WEALTH'] = df_agents.WAGE * 12
             # Prision_Time_Left = 0 (all free)
             df_agents['PRISION_TIME_LEFT'] = 0
             # Crimes commited
             df_agents['CRIMES'] = 0
             # Times captured
             df_agents['CAPTURED_TIMES'] = 0
```

```
In [13]: # Dataset que contiene la informacion actual de cada agente
         df_agents = pd.DataFrame(columns=['WAGE','WEALTH','HONESTY','PRISION_TIME_LEFT'])
         # Dataset que se utiliza para simular las variables temporales de cada periodo (mes)
         df_simulated_month = pd.DataFrame(columns=['BOOTY','PRISION_TIME','FINE','CAPTURE_PRO

         initialize_model(df_agents, df_simulated_month)
```

2.1.1 Simulación de prueba

```
In [14]: # Simulacion de Booty
         wage_avg = df_agents.WAGE.mean()
         df_simulated_month['BOOTY'] = simulate_booty(wage_avg, sim_agents)
         booty_avg = df_simulated_month.BOOTY.mean()
         # Calculate prision time
         df_simulated_month['PRISION_TIME'] = get_prision_time(df_simulated_month.BOOTY, wage_a
         # Calculate Fine
         df_simulated_month['FINE'] = get_fine(df_simulated_month.BOOTY, df_agents.WEALTH)
         # Calculate Caputure Probability
```

```
df_simulated_month['CAPTURE_PROB'] = df_simulated_month.BOOTY.apply(get_capture_prob,
# Expected Utility
df_simulated_month['EXPECTED_UTILITY'] = get_utilities(df_simulated_month.CAPTURE_PRO
```

2.1.2 Dataset generado según la simulación de prueba

```
In [15]: df_month_crimes = df_agents.join(df_simulated_month)[ df_simulated_month.EXPECTED_UTI
```

2.1.3 Agentes que van a delinquir

```
In [16]: df_month_crimes.describe()
```

Out[16]:

	WAGE	WEALTH	HONESTY	PRISION_TIME_LEFT	CRIMES
count	38.000000	38.000000	38.000000	38.0	38.0
mean	5.720983	68.651799	27.458752	0.0	0.0
std	3.880169	46.562023	15.843498	0.0	0.0
min	1.027029	12.324347	0.000000	0.0	0.0
25%	2.681377	32.176524	15.725658	0.0	0.0
50%	4.900188	58.802252	29.819032	0.0	0.0
75%	7.460410	89.524924	37.852359	0.0	0.0
max	14.889043	178.668516	58.361344	0.0	0.0

	CAPTURED_TIMES	BOOTY	PRISION_TIME	FINE	CAPTURE_PROB
count	38.0	38.000000	38.000000	38.000000	38.000000
mean	0.0	208.022529	7.149519	52.005632	0.531166
std	0.0	71.298173	2.107702	17.824543	0.060197
min	0.0	55.446532	2.639099	13.861633	0.402335
25%	0.0	157.843897	5.666149	39.460974	0.488493
50%	0.0	205.367615	7.071035	51.341904	0.530082
75%	0.0	264.661879	8.823880	66.165470	0.580207
max	0.0	336.043606	10.934052	84.010902	0.634540

	EXPECTED_UTILITY
count	38.000000
mean	16.369539
std	13.619898
min	2.076087
25%	6.946309
50%	12.760408
75%	20.836858
max	60.988727

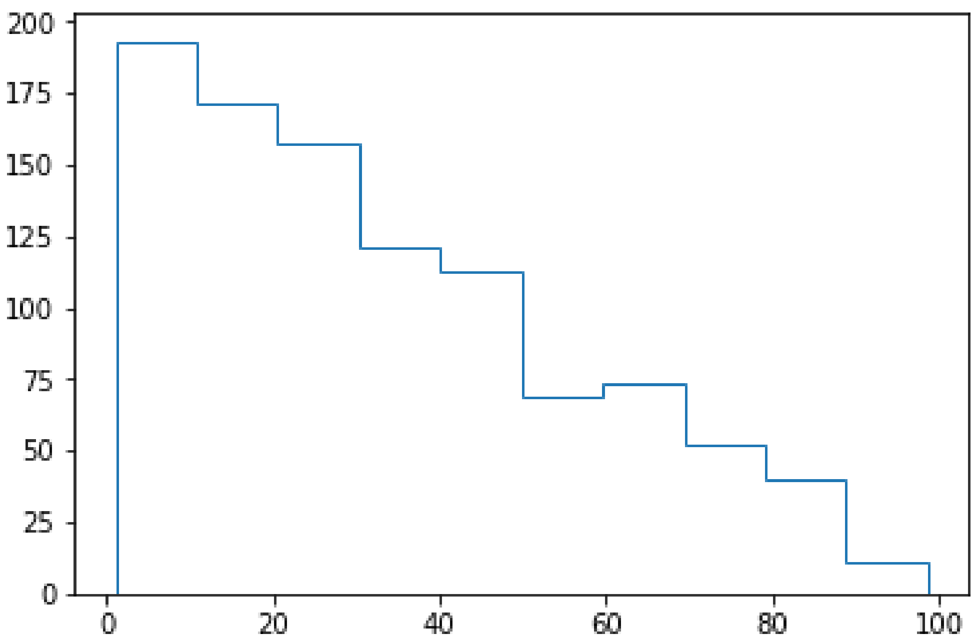
Es interesante notar como primera observación que, según el modelo propuesto, los agentes más propensos a delinquir son quienes perciben un salario menor, aunque no se debe caer en la falacia de afirmar que todo agente con bajos ingresos terminará delinquiendo.

2.1.4 Histogramas de la simulación de prueba

Distribución inicial de ingresos (según la carga simulada)

```
In [17]: plt.hist(df_agents.WAGE, bins=10, range=[df_agents.WAGE.min(), df_agents.WAGE.max()],
```

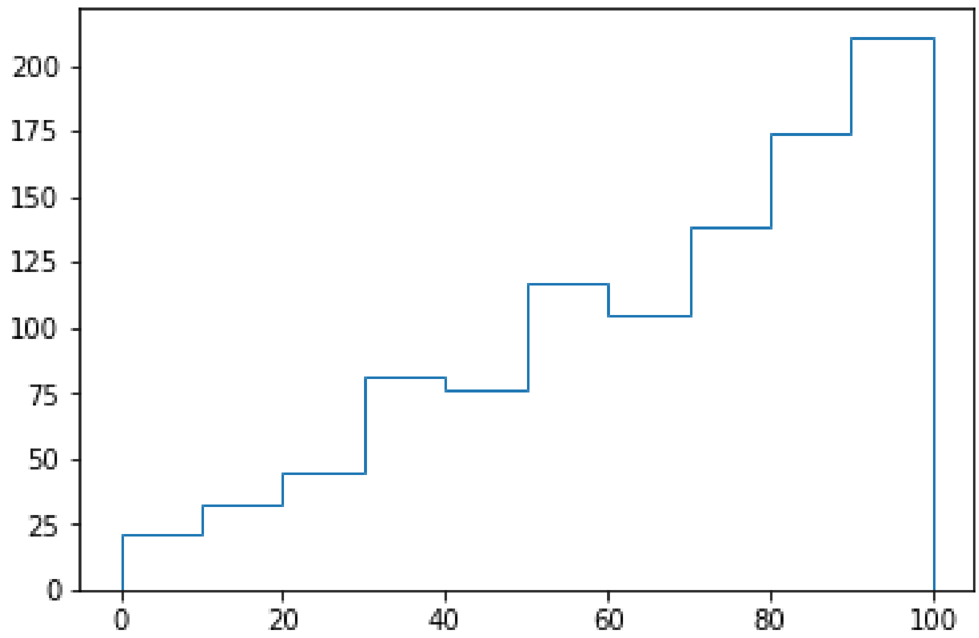
```
Out[17]: (array([ 193.,  171.,  157.,  121.,  113.,   69.,   73.,   52.,   40.,   11.]),
          array([  1.02702895,  10.78735431,  20.54767967,  30.30800503,
                  40.06833039,  49.82865575,  59.58898111,  69.34930648,
                  79.10963184,  88.8699572 ,  98.63028256]),
          <a list of 1 Patch objects>)
```



Distribución de la honestidad entre los agentes según la carga simulada

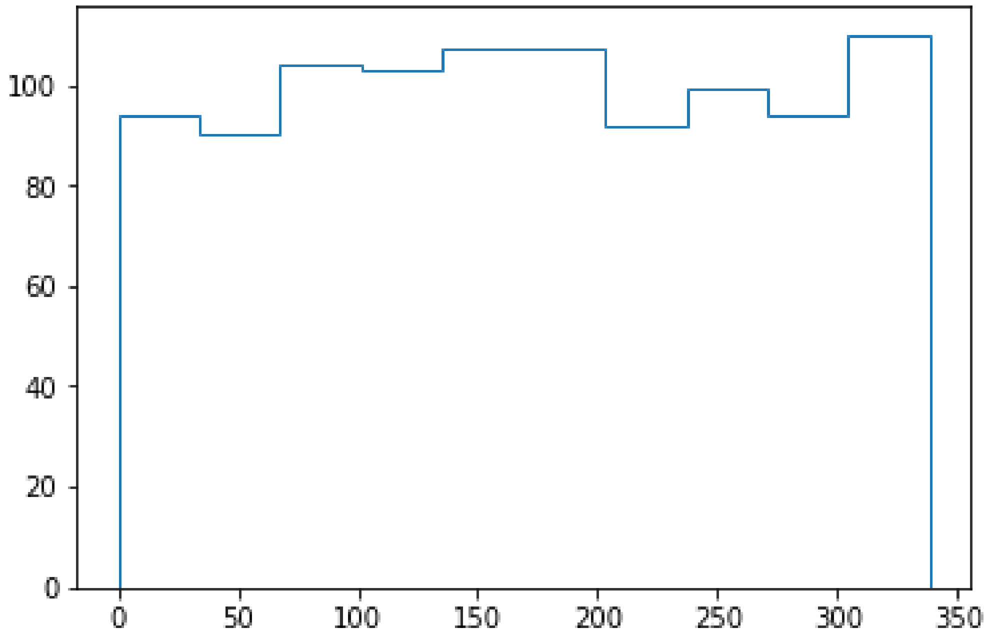
```
In [18]: plt.hist(df_agents.HONESTY, bins=10, range=[df_agents.HONESTY.min(), df_agents.HONEST
```

```
Out[18]: (array([  21.,   32.,   45.,   81.,   76.,  117.,  105.,  138.,  174.,  211.]),
          array([  0.        ,   9.99263017,  19.98526035,  29.97789052,
                  39.97052069,  49.96315086,  59.95578104,  69.94841121,
                  79.94104138,  89.93367156,  99.92630173]),
          <a list of 1 Patch objects>)
```



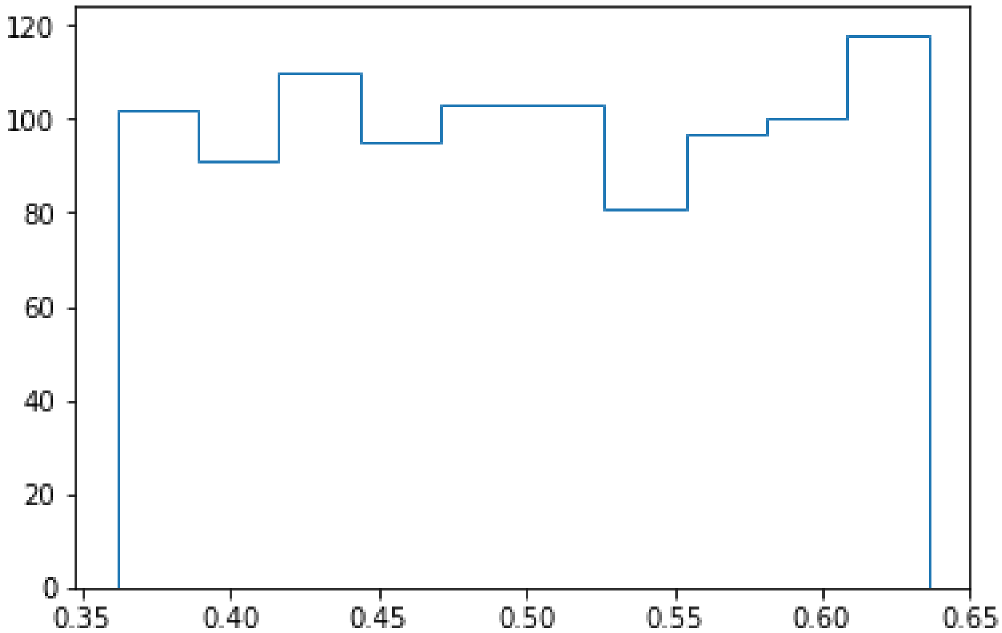
Distribución inicial del botín para cada agente según la carga simulada

```
In [19]: plt.hist(df_simulated_month.BOOTY, bins=10, range=[df_simulated_month.BOOTY.min(), df_

Out[19]: (array([  94.,   90.,  104.,  103.,  107.,  107.,   92.,   99.,   94.,  110.]),
          array([  2.49306353e-02,   3.38486656e+01,   6.76724005e+01,
                   1.01496135e+02,   1.35319870e+02,   1.69143605e+02,
                   2.02967340e+02,   2.36791075e+02,   2.70614810e+02,
                   3.04438545e+02,   3.38262280e+02]),
          <a list of 1 Patch objects>)
```



Distribución del Cálculo de la probabilidad de captura de los agentes. Ya que depende del botín (que se simula de una distribución uniforme), la distribución es también uniforme.

```
In [20]: plt.hist(df_simulated_month.CAPTURE_PROB, bins=10, range=[df_simulated_month.CAPTURE_P

Out[20]: (array([ 102.,   91.,  110.,   95.,  103.,  103.,   81.,   97.,  100.,  118.]),
          array([ 0.36138206,  0.38885295,  0.41632384,  0.44379473,  0.47126562,
                  0.49873651,  0.5262074 ,  0.55367829,  0.58114918,  0.60862007,
                  0.63609096]),
          <a list of 1 Patch objects>)
```



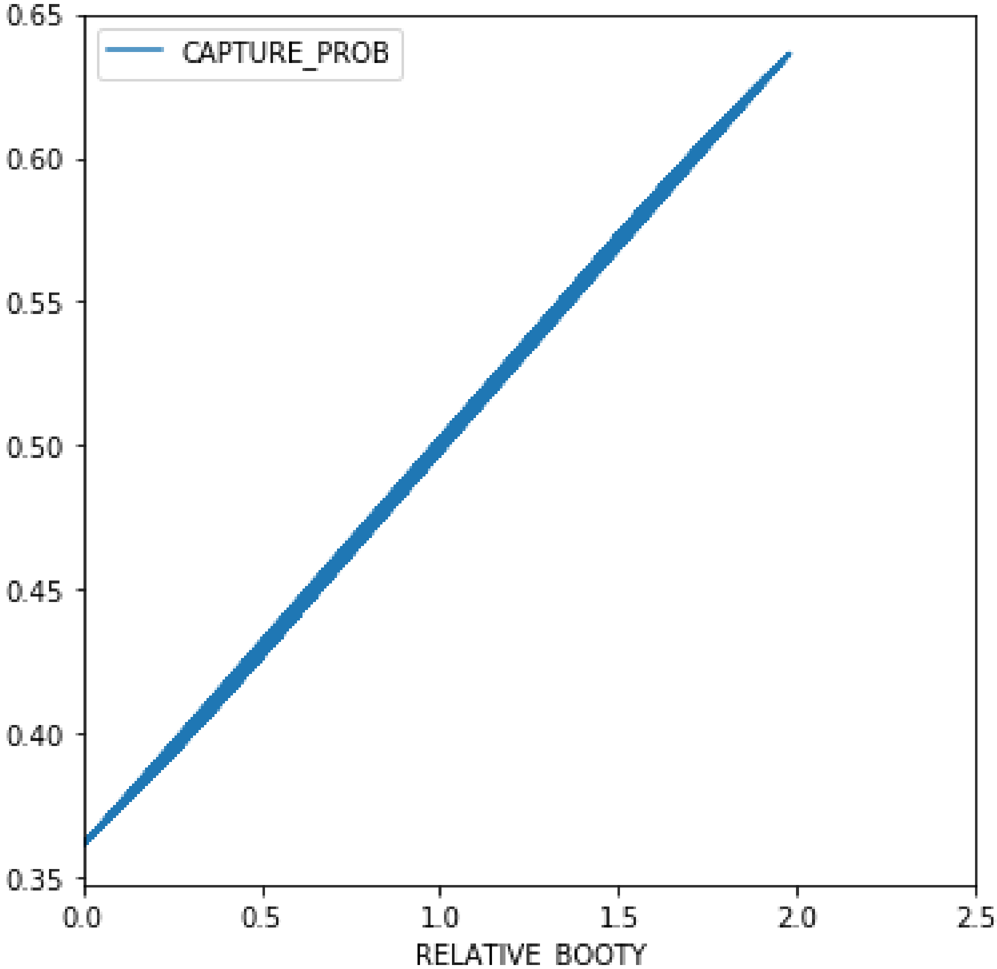
2.1.5 Relación entre el botín relativo (S/< S >) y la probabilidad de captura

A mayor botín, la probabilidad de captura se acerca a la asíntota P1

```
In [21]: df_simulated_month['RELATIVE_BOOTY'] = df_simulated_month.BOOTY / booty_avg
```

```
In [22]: df_simulated_month.plot(x='RELATIVE_BOOTY',y='CAPTURE_PROB',xlim=(df_simulated_month.
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0xcdc90f0>
```



2.2 Períodos

A continuación se realizará la simulación sobre un período de 240 meses. Según la carga inicial previa del dataset df_agents, se comenzará a simular año por año para analizar los resultados obtenidos y así poder incorporar las modificaciones deseadas sobre el modelo.

```
In [23]: initialize_model(df_agents, df_simulated_month)
         df_hist = pd.DataFrame(columns=['FREE_AGENTS','CRIMES','CAPTURED','BOOTY_AVG','WAGE_A

         for month in range(0,sim_months):
             ### Booty simulation
             wage_avg = df_agents.WAGE.mean()
             df_simulated_month['BOOTY'] = simulate_booty(wage_avg, sim_agents)
             booty_avg = df_simulated_month.BOOTY.mean()

             ### Variable calculation for the current month
```

```
    # Prision Time calculation
    df_simulated_month['PRISION_TIME'] = get_prision_time(df_simulated_month.BOOTY, wa
    # Fine Calculation
    df_simulated_month['FINE'] = get_fine(df_simulated_month.BOOTY, df_agents.WEALTH)
    # Caputure Probability Calculation
    df_simulated_month['CAPTURE_PROB'] = df_simulated_month.BOOTY.apply(get_capture_pr
    # Expected Utility Calculation
    df_simulated_month['EXPECTED_UTILITY'] = get_utilities(df_simulated_month.CAPTURE_

    ### Processing calculated month
    # Agents not in prision
    df_free_agents = df_agents.query('PRISION_TIME_LEFT == 0')
    # Agents that commited crime
    df_commited_crime = df_free_agents.join(df_simulated_month)[df_simulated_month.EXP
    # Goverment Punishing simulation
    df_commited_crime['CAPTURED'] = get_captured_sim(df_commited_crime.CAPTURE_PROB)
    # Honesty value modification
    crimes_commited = df_commited_crime.shape[0]
    agents_captured = df_commited_crime[ df_commited_crime.CAPTURED ].shape[0]
    df_agents['HONESTY'] = get_adjusted_honesty(df_agents.HONESTY, crimes_commited, ag
    # Wealth modification
    df_free_agents['ADJ_WEALTH'] = df_free_agents.WEALTH + df_free_agents.WAGE
    df_commited_crime['ADJ_WEALTH'] = df_commited_crime.WEALTH + df_commited_crime.WAG
    df_commited_crime['ADJ_WEALTH'].loc[ df_commited_crime.CAPTURED ] = df_commited_cr
    df_commited_crime['ADJ_WEALTH'].loc[ df_commited_crime.ADJ_WEALTH < 0 ] = 0
    # Prision time modification
    df_agents['PRISION_TIME_LEFT'] = df_agents.PRISION_TIME_LEFT - 1
    df_agents['PRISION_TIME_LEFT'].loc[ df_agents.PRISION_TIME_LEFT < 0 ] = 0
    df_commited_crime['PRISION_TIME_LEFT'].loc[ df_commited_crime.CAPTURED ] = df_comm

    ### Merging results in master dataframe df_agents
    df_agents['WEALTH'].loc[ df_free_agents.index ] = df_free_agents.ADJ_WEALTH
    df_agents['WEALTH'].loc[ df_commited_crime.index ] = df_commited_crime.ADJ_WEALTH
    df_agents['PRISION_TIME_LEFT'].loc[ df_commited_crime.index ] = df_commited_crime
    df_agents['CRIMES'].loc[ df_commited_crime.index ] = df_agents.CRIMES + 1
    df_agents['CAPTURED_TIMES'].loc[ df_commited_crime.loc[df_commited_crime.CAPTURED]

    ### Historical data
    df_hist.loc[month] = [ df_free_agents.shape[0], crimes_commited, agents_captured,
```

```
C:\Users\mapre\Anaconda2\lib\site-packages\ipykernel_launcher.py:24: UserWarning: Boolean Serie
```

2.3 Análisis de simulación

2.3.1 Creación de variables de apoyo

```
In [24]: # Variables para analisis
```

```
# Se discretiza el salario por niveles
niveles_salario = range(int(df_agents.WAGE.min())-1,int(df_agents.WAGE.max())+1,int((d
df_agents['WAGE_LEVEL'] = pd.cut(df_agents.WAGE, niveles_salario, labels = False, incl
```

2.3.2 Dataset con ultimos datos de los agentes

In [25]:
```
df_agents
```

Out[25]:
```
          WAGE        WEALTH     HONESTY  PRISION_TIME_LEFT  CRIMES  \
0     5.171306   1303.169151  100.000000                0.0       0
1    28.282309   7127.141770  100.000000                0.0       0
2    16.422237   4138.403728   94.653280                0.0       0
3    65.659918  16546.299449  100.000000                0.0       0
4    16.283634   4103.475739   85.947953                0.0       0
5    76.093411  19175.539468  100.000000                0.0       0
6    33.304026   8392.614598  100.000000                0.0       0
7    18.579720   4682.089382  100.000000                0.0       0
8    57.251371  14427.345483  100.000000                0.0       0
9    45.541035  11476.340767  100.000000                0.0       0
10   24.212444   6101.535996  100.000000                0.0       0
11   39.749597  10016.898541  100.000000                0.0       0
12   53.415715  13460.760128   88.390262                0.0       0
13   61.857084  15587.985111  100.000000                0.0       0
14   57.581384  14510.508740   74.389485                0.0       0
15    9.593814   2417.641145  100.000000                0.0       0
16   58.988368  14865.068624   85.107101                0.0       0
17    3.945970    994.384552  100.000000                0.0       0
18   11.461155   2888.210952  100.000000                0.0       0
19   39.207770   9880.358064  100.000000                0.0       0
20    6.850129   1727.341186   95.495312                0.0       5
21   37.649457   9487.663147   96.840273                0.0       0
22   17.643286   4446.108197  100.000000                0.0       0
23   25.221674   6355.861916  100.000000                0.0       0
24   18.071982   4554.139493  100.000000                0.0       0
25    4.536767   1143.265275  100.000000                0.0       0
26   30.049564   7572.490189   97.112965                0.0       0
27   81.111860  20440.188634  100.000000                0.0       0
28   38.316036   9655.641093  100.000000                0.0       0
29   17.172543   4327.480846  100.000000                0.0       0
..         ...           ...         ...                ...     ...
970  75.609900  19053.694735  100.000000                0.0       0
971  32.846950   8277.431398  100.000000                0.0       0
972  10.074169   2538.690566  100.000000                0.0       0
973  47.674266  12013.915131  100.000000                0.0       0
974   6.265620   2019.216786   86.148927                0.0      20
975  52.691339  13278.217407   68.515210                0.0       0
976  27.067775   6821.079337   97.766956                0.0       0
977   7.800324   1965.681758  100.000000                0.0       0
```

978	2.859408	692.039403	61.191236	0.0	24
979	91.943805	23169.838767	83.610860	0.0	0
980	15.667821	3948.290931	100.000000	0.0	0
981	78.366432	19748.340918	100.000000	0.0	0
982	30.125097	7591.524454	100.000000	0.0	0
983	30.207836	7612.374552	100.000000	0.0	0
984	5.615543	1415.116880	100.000000	0.0	0
985	78.019730	19660.971958	100.000000	0.0	0
986	48.834703	12306.345120	100.000000	0.0	0
987	31.319956	7892.628886	100.000000	0.0	0
988	7.143337	1800.121037	100.000000	0.0	0
989	10.115664	2549.147210	100.000000	0.0	0
990	25.398826	6254.534648	61.191236	0.0	1
991	20.939888	5026.222376	61.191236	0.0	8
992	34.192026	8616.390643	61.191236	0.0	0
993	4.854045	2850.746207	61.191236	0.0	32
994	3.017689	1496.623394	61.191236	0.0	30
995	30.778048	7756.068080	61.191236	0.0	0
996	2.421017	1357.886288	61.191236	0.0	28
997	3.993066	2230.992377	61.191236	0.0	35
998	21.817666	6015.202058	61.191236	0.0	8
999	4.106813	1665.532845	61.191236	0.0	30

	CAPTURED_TIMES	WAGE_LEVEL
0	0	0.0
1	0	3.0
2	0	1.0
3	0	7.0
4	0	1.0
5	0	8.0
6	0	3.0
7	0	2.0
8	0	6.0
9	0	5.0
10	0	2.0
11	0	4.0
12	0	5.0
13	0	6.0
14	0	6.0
15	0	1.0
16	0	6.0
17	0	0.0
18	0	1.0
19	0	4.0
20	3	0.0
21	0	4.0
22	0	1.0
23	0	2.0

```
24             0        2.0
25             0        0.0
26             0        3.0
27             0        9.0
28             0        4.0
29             0        1.0
..           ...        ...
970            0        8.0
971            0        3.0
972            0        1.0
973            0        5.0
974           11        0.0
975            0        5.0
976            0        3.0
977            0        0.0
978           17        0.0
979            0       10.0
980            0        1.0
981            0        8.0
982            0        3.0
983            0        3.0
984            0        0.0
985            0        8.0
986            0        5.0
987            0        3.0
988            0        0.0
989            0        1.0
990            1        2.0
991            4        2.0
992            0        3.0
993           15        0.0
994           19        0.0
995            0        3.0
996           17        0.0
997           15        0.0
998            1        2.0
999           18        0.0

[1000 rows x 7 columns]
```

2.3.3 Dataset con datos promedios de los meses del período

In [26]:
```
df_hist
```

Out[26]:
```
     FREE_AGENTS  CRIMES  CAPTURED   BOOTY_AVG   WAGE_AVG   WEALTH_AVG  \
0         1000.0    30.0      15.0  161.261935  32.756854  427.874383
1          985.0    16.0       9.0  161.487130  32.756854  461.362204
2          976.0    12.0       8.0  161.727385  32.756854  494.330109
```

3	968.0	4.0	2.0	162.695341	32.756854	527.181366
4	968.0	5.0	2.0	165.442615	32.756854	560.318107
5	970.0	7.0	3.0	158.167977	32.756854	593.672076
6	969.0	10.0	6.0	164.825989	32.756854	626.461533
7	967.0	7.0	3.0	161.333431	32.756854	659.605103
8	965.0	5.0	2.0	161.664365	32.756854	692.698800
9	968.0	8.0	2.0	156.449021	32.756854	726.456837
10	969.0	14.0	7.0	160.688891	32.756854	760.075483
11	969.0	14.0	5.0	162.690456	32.756854	794.117503
12	968.0	19.0	10.0	163.958002	32.756854	827.960987
13	965.0	25.0	11.0	165.999059	32.756854	862.340887
14	959.0	19.0	13.0	166.552915	32.756854	895.267572
15	948.0	12.0	4.0	169.958996	32.756854	929.180256
16	950.0	16.0	9.0	164.061298	32.756854	962.439499
17	947.0	10.0	7.0	161.584431	32.756854	995.151470
18	944.0	6.0	1.0	158.201406	32.756854	1028.217080
19	950.0	20.0	11.0	165.254815	32.756854	1061.680037
20	946.0	12.0	6.0	164.162416	32.756854	1094.688417
21	950.0	17.0	10.0	164.761426	32.756854	1127.450744
22	950.0	10.0	6.0	159.594116	32.756854	1160.609470
23	952.0	16.0	12.0	164.420325	32.756854	1192.797363
24	946.0	4.0	2.0	168.310344	32.756854	1225.549933
25	952.0	4.0	2.0	160.362566	32.756854	1258.254127
26	952.0	7.0	6.0	162.229986	32.756854	1290.648273
27	952.0	6.0	2.0	165.781992	32.756854	1323.843774
28	955.0	6.0	4.0	162.451787	32.756854	1356.481198
29	960.0	7.0	6.0	165.629568	32.756854	1389.167982
..	...	...	...	...	...	...
210	1000.0	0.0	0.0	158.638227	32.756854	7345.378113
211	1000.0	0.0	0.0	164.121450	32.756854	7378.134967
212	1000.0	0.0	0.0	163.072435	32.756854	7410.891822
213	1000.0	0.0	0.0	163.303170	32.756854	7443.648676
214	1000.0	0.0	0.0	162.394055	32.756854	7476.405531
215	1000.0	0.0	0.0	163.804381	32.756854	7509.162385
216	1000.0	0.0	0.0	164.617538	32.756854	7541.919239
217	1000.0	0.0	0.0	166.931502	32.756854	7574.676094
218	1000.0	0.0	0.0	164.798984	32.756854	7607.432948
219	1000.0	0.0	0.0	164.734612	32.756854	7640.189802
220	1000.0	0.0	0.0	158.051008	32.756854	7672.946657
221	1000.0	0.0	0.0	165.868532	32.756854	7705.703511
222	1000.0	0.0	0.0	161.458088	32.756854	7738.460365
223	1000.0	0.0	0.0	161.232640	32.756854	7771.217220
224	1000.0	0.0	0.0	163.824906	32.756854	7803.974074
225	1000.0	0.0	0.0	161.794128	32.756854	7836.730928
226	1000.0	0.0	0.0	163.148654	32.756854	7869.487783
227	1000.0	0.0	0.0	163.320250	32.756854	7902.244637
228	1000.0	0.0	0.0	161.862385	32.756854	7935.001491
229	1000.0	0.0	0.0	165.885493	32.756854	7967.758346

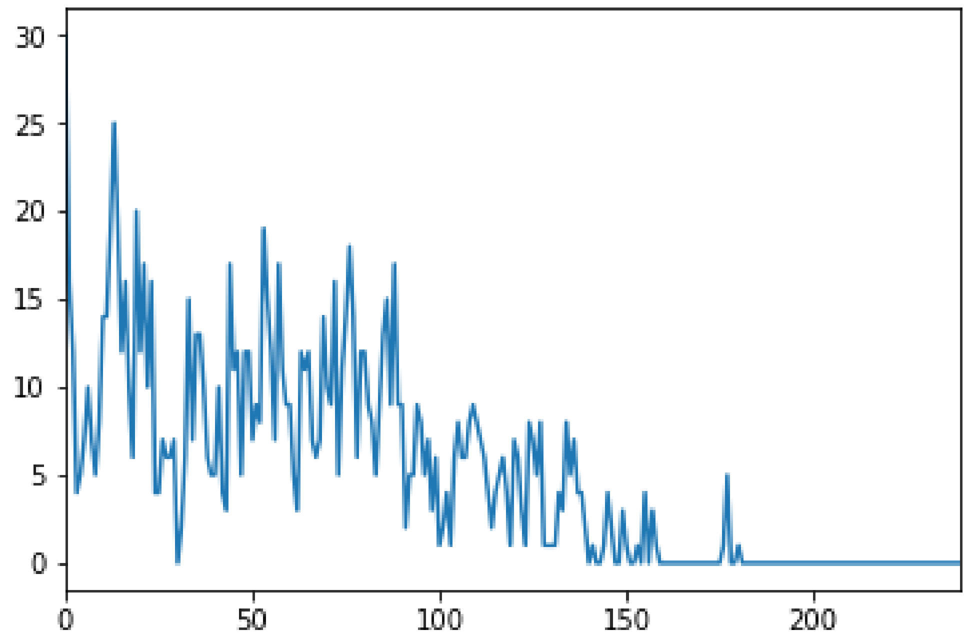
230	1000.0	0.0	0.0	166.721419	32.756854	8000.515200
231	1000.0	0.0	0.0	164.061566	32.756854	8033.272054
232	1000.0	0.0	0.0	164.539736	32.756854	8066.028909
233	1000.0	0.0	0.0	163.331902	32.756854	8098.785763
234	1000.0	0.0	0.0	163.618360	32.756854	8131.542617
235	1000.0	0.0	0.0	164.281991	32.756854	8164.299472
236	1000.0	0.0	0.0	160.527345	32.756854	8197.056326
237	1000.0	0.0	0.0	165.006211	32.756854	8229.813181
238	1000.0	0.0	0.0	164.609844	32.756854	8262.570035
239	1000.0	0.0	0.0	159.845200	32.756854	8295.326889

	HONESTY_AVG
0	65.270642
1	66.512027
2	69.707578
3	69.707578
4	67.707578
5	66.279006
6	68.279006
7	66.850435
8	64.853173
9	59.904727
10	59.904727
11	57.087454
12	57.613769
13	56.424588
14	60.108798
15	56.775465
16	58.025465
17	62.025465
18	55.377752
19	56.377752
20	56.377752
21	58.142458
22	60.142458
23	65.142458
24	65.142458
25	65.142458
26	72.058306
27	68.724973
28	72.058306
29	77.933638
..	...
210	95.787241
211	95.787241
212	95.787241
213	95.787241
214	95.787241

```
215    95.787241
216    95.787241
217    95.787241
218    95.787241
219    95.787241
220    95.787241
221    95.787241
222    95.787241
223    95.787241
224    95.787241
225    95.787241
226    95.787241
227    95.787241
228    95.787241
229    95.787241
230    95.787241
231    95.787241
232    95.787241
233    95.787241
234    95.787241
235    95.787241
236    95.787241
237    95.787241
238    95.787241
239    95.787241

[240 rows x 7 columns]
```

2.3.4 Progresión de los crímenes cometidos por mes

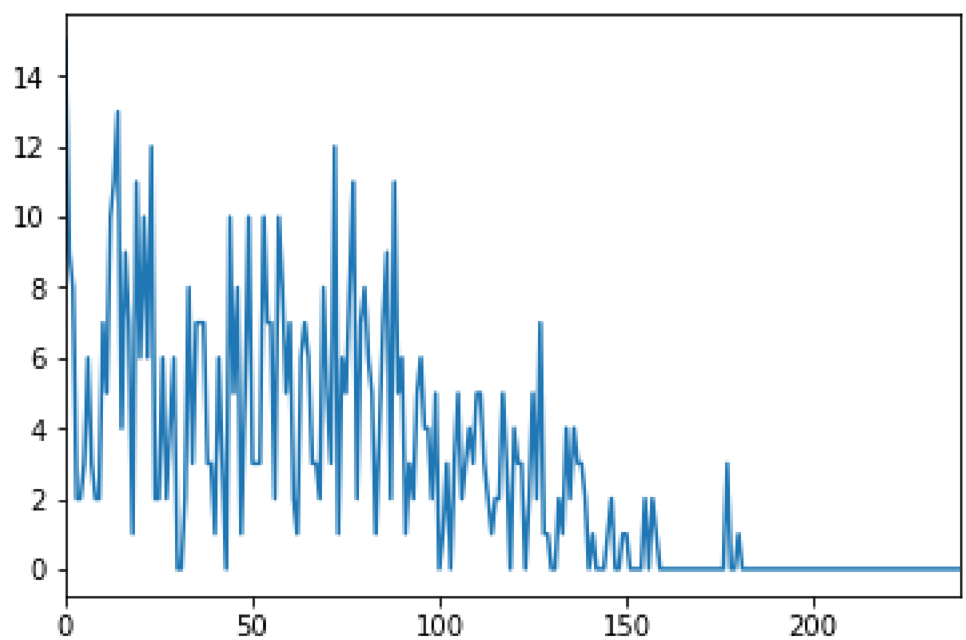
```
In [27]: df_hist.CRIMES.plot()

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xdf04f28>
```



2.3.5 Progresión de los crímenes capturados por mes

```
In [28]: df_hist.CAPTURED.plot()
```

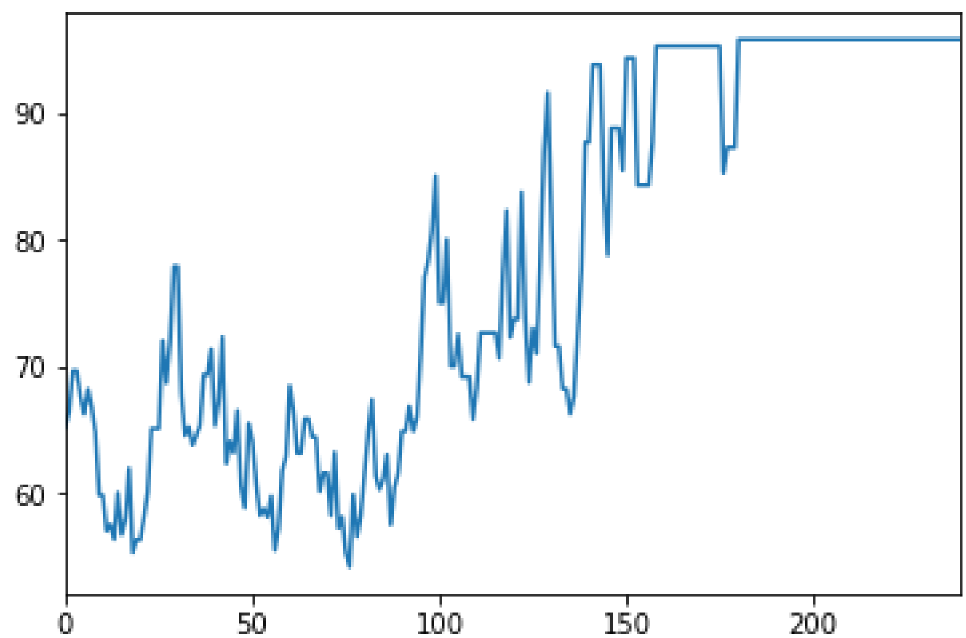
```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xc1e16a0>
```



2.3.6 Progresión de la honestidad promedio de los agentes

In [29]:
```
df_hist.HONESTY_AVG.plot()
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xd529438>
```



2.3.7 Análisis de linealidad entre variables de cada agente

In [30]:
```
df_agents.corr()
```

Out[30]:
```
                         WAGE    WEALTH   HONESTY  PRISION_TIME_LEFT    CRIMES  \
WAGE                 1.000000  0.999160  0.005654                NaN -0.286789
WEALTH               0.999160  1.000000 -0.004314                NaN -0.258148
HONESTY              0.005654 -0.004314  1.000000                NaN -0.453039
PRISION_TIME_LEFT         NaN       NaN       NaN                NaN       NaN
CRIMES              -0.286789 -0.258148 -0.453039                NaN  1.000000
CAPTURED_TIMES      -0.286284 -0.262384 -0.444050                NaN  0.980128
WAGE_LEVEL           0.993523  0.992882  0.009684                NaN -0.284576

                   CAPTURED_TIMES  WAGE_LEVEL
WAGE                    -0.286284    0.993523
WEALTH                  -0.262384    0.992882
HONESTY                 -0.444050    0.009684
```

```
PRISION_TIME_LEFT             NaN         NaN
CRIMES                   0.980128   -0.284576
CAPTURED_TIMES           1.000000   -0.283942
WAGE_LEVEL              -0.283942    1.000000
```

3 Cambios propuestos

- La riqueza (wealth) puede generar ingresos mensuales (wage) a través de inversiones
  - Se nota que una persona de bajos ingresos (random de inicializacion) siempre es propenso a delinquir, por lo cual si consideramos la riqueza acumulada para generar ingresos el modelo sería más apropiado.
- Modificador de honestidad: agregar dependencia sobre la multa y el tiempo de castigo