Contents

- 1 Mortgage class
- 2 30 year fixed-rate mortgage
- 3 20 year fixed-rate mortgage
- 4 30 year ARM

Mortgage class

A python class "Mortgage", that can serve in creating the **amortization schedule** for both *fixed* and *adjusted rate mortgages* is created.

The class works by the following way:

- Creates a new instance of mortgage every time it is assigned to a variable.

The parameters that the Mortgage class takes are:

```
    * Principal amount
    * Interest rate
    * Period of mortgage in years

In case of ARM, three additional parameters should be passed:

    * Variable years which is the total number of years when the reset will occur.
    * Years for reset (1 if the rates are reset every year, 0.5 if the rates are reset
every six months)
    * Reset rates, which is a python list containing interest rates for the reset. The
number of elements in the list
      must be equal to the variable years.

The class initiates four python lists. All the values of the list are in dollars.

    * Payment
    * Principal
    * Interest
    * Balance
```

- The method MakePayment() computes the payment using the principal amount, rate and the period.

It does the following to the lists:

```
    * Appends a single payment to the payment list
    * Computes the interest amount paid and appends to interest list
    * Subtracts the interest from the payment and appends the value to principal list.
    * Subtracts the principal from previous balance and appends to the balance list.
```

- The method AmortizationSchedule() returns a Pandas DataFrame, containing five features.

```
* Period
* Payment
* Principal
* Interest
* Balance
```

A *for loop* with the range of the total number of payments of the mortgage can be used to amortize the schedule to zero.

```
    Example:

        mort_a = Mortgage(1000000, 0.04, 30)
        for i in range(30*12):
            mort_a.MakePayment()
        mort_a.AmortizationSchedule()
```

In [1]:
```
import pandas as pd
import altair as alt
pd.options.display.float_format = '{:.2f}'.format

class Mortgage(object):
    '''
    Creates an instance of class Mortgage.
    Multi-purpose - can be used to schedule Fixed rate and adjustable rate mortgages.

    Parameters:
        loan: Initial loan amount
        interest_rate: Annual interest rate.
                       If adjustable rate mortgage, pass the fixed term rate. Ex.,annual rate for 7 years for
        years: Period of mortgage in years

        Optional (To be used for Adjusted rate mortgage)
        variable_years: Total years when the rates are adjusted. Ex., 23 <30-7> for 7-1 ARM amortizing in 30 y
        reset_time: Reset time in years. Ex., 1 for 7-1 ARM
        reset_rates: Assumes a list of interest rates for the remaining years of an adjustable mortgage

    Methods:
        MakePayment(): Adds one payment to the Amortization schedule
        GetPayments(): Returns the completed payments
        GetPrincipal(): Returns the paid principal each month
        GetInterest(): Returns the paid interest each month
        GetTotalPayments(): Returns the sum of completed payments
        GetBalance(): Returns the outstanding balance
        AmortizationSchedule(): Returns a Pandas DataFrame
    '''
    def __init__(self, loan, interest_rate, years, variable_years=0, reset_time=0, reset_rates=[]):
        if variable_years != 0:
            try:
                assert len(reset_rates) == variable_years/reset_time
            except:
                raise ValueError("The number of reset rates does not match the variable years")

        self.loan = loan
        self.rate = interest_rate/12
        self.months = years*12
        self.variable_years = variable_years
        self.fixed_years = years - variable_years
        self.reset_time = reset_time
        self.reset_rates = reset_rates
        self.paid = [0]
        self.principal = [0]
        self.interest = [0]
        self.balance = [loan]

    def CalculatePayment(self, loan, monthly_rate, months):
        '''
        Assumes loan, monthly_rate and months to be int. Returns the monthly payment.
        '''
        num = loan*monthly_rate*(1+monthly_rate)**months
        den = (1+monthly_rate)**months - 1
        return num/den

    def MakePayment(self):
        '''
        Adds one payment to the mortgage schedule when invoked.
        '''
        if len(self.balance)-1 < self.fixed_years*12:
            Mortgage.interest_rate = self.rate
            Mortgage.pmt = Mortgage.CalculatePayment(self, self.loan, Mortgage.interest_rate, self.months)

        elif (len(self.balance)-1) % (12*self.reset_time) == 0:
            reset = int(((len(self.balance) - 1) - (self.fixed_years * 12)) / (12*self.reset_time))
            Mortgage.interest_rate = self.reset_rates[reset]/12
            bal = self.balance[-1]
            months = self.months - (12*self.fixed_years) - (12*self.reset_time*reset)
            Mortgage.pmt = Mortgage.CalculatePayment(self, bal, Mortgage.interest_rate, months)

        self.paid.append(Mortgage.pmt)
        interest = self.balance[-1] * Mortgage.interest_rate
        self.interest.append(interest)
        principal = Mortgage.pmt - interest
        self.principal.append(principal)
        reduction = self.balance[-1] - principal
        self.balance.append(reduction)

    def GetPayments(self):
        '''
        Returns a list of completed payments.
        '''
        return self.paid.copy()

    def GetPrinipal(self):
        '''
        Returns a list of Principal paid each month.
        '''
        return self.principal.copy()

    def GetInterest(self):
        '''
        Returns a list of interest paid each month.
        '''
        return self.interest.copy()

    def GetTotalPayments(self):
        '''
        Returns the sum of completed payments.
        '''
        return sum(self.paid)

    def GetUnpaid(self):
        '''
        Returns a list of unpaid balances on each month.
        '''
        return self.balance.copy()

    def AmortizationSchedule(self):
        '''
        Returns a Pandas DataFrame of the Amortization schedule
        containing features Period, Payments, Principal, Interest and Unpaid Balance.
        '''
        return pd.DataFrame(
            {
                "Period": range(len(self.balance)),
                "Payments": self.GetPayments(),
                "Principal": self.GetPrinipal(),
                "Interest": self.GetInterest(),
                "Unpaid Balance": self.GetUnpaid()
            }
        ).apply(lambda x: round(x,2))
```

30 year fixed-rate mortgage

In [2]:
```
p = 1_000_000
i = 0.04
n = 30

mortgage_a = Mortgage(p, i, n)

for i in range(n*12):
    mortgage_a.MakePayment()

schedule_a = mortgage_a.AmortizationSchedule()

schedule_a
```

Out[2]:

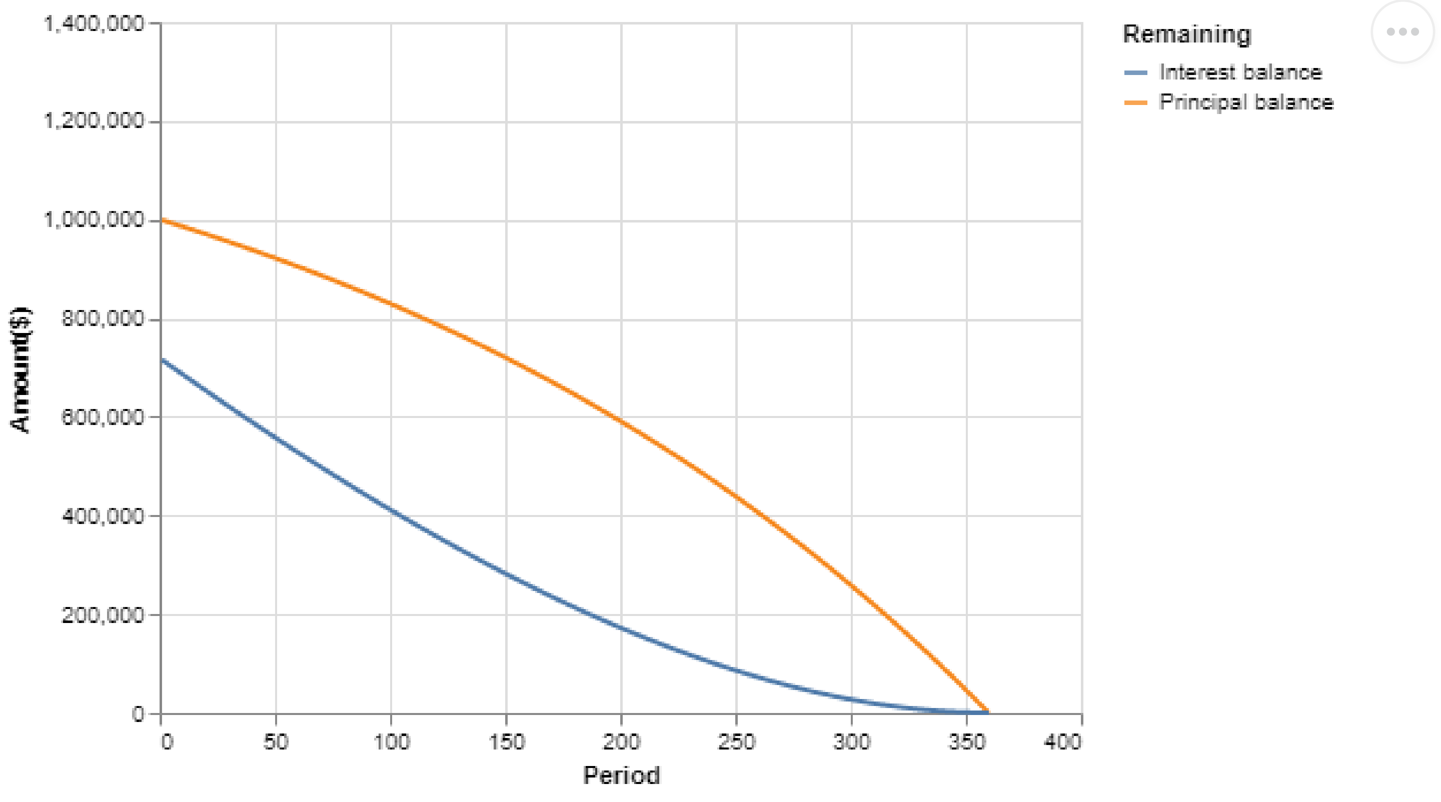
	Period	Payments	Principal	Interest	Unpaid Balance
0	0	0.00	0.00	0.00	1000000.00
1	1	4774.15	1440.82	3333.33	998559.18
2	2	4774.15	1445.62	3328.53	997113.56
3	3	4774.15	1450.44	3323.71	995663.12
4	4	4774.15	1455.28	3318.88	994207.84
...	...	...	...	...	...
356	356	4774.15	4695.37	78.78	18938.53
357	357	4774.15	4711.02	63.13	14227.50
358	358	4774.15	4726.73	47.43	9500.78
359	359	4774.15	4742.48	31.67	4758.29
360	360	4774.15	4758.29	15.86	0.00

361 rows × 5 columns

In [3]:
```
schedule_a_bal = schedule_a.copy()
schedule_a_bal["Interest balance"] = schedule_a_bal["Interest"].sum() - schedule_a_bal["Interest"].cumsum()
schedule_a_bal["Principal balance"] = schedule_a_bal["Principal"].sum() - schedule_a_bal["Principal"].cumsum(

schedule_a_melt = schedule_a_bal.drop(0).melt(id_vars="Period", value_vars=["Principal balance", "Interest ba
                                             value_name="Amount($)", var_name="Remaining")

a_plot = alt.Chart(schedule_a_melt).mark_line()\
            .encode(x="Period", y=alt.Y("Amount($)", scale=alt.Scale(domain=[0, 1_400_000])), color="Remainin

a_plot
```

```
C:\Users\Abishek\anaconda3\envs\zen\lib\site-packages\altair\utils\core.py:317: FutureWarning: iteritems is d
eprecated and will be removed in a future version. Use .items instead.
  for col_name, dtype in df.dtypes.iteritems():
```

Out[3]:



20 year fixed-rate mortgage

In [4]:
```
p = 1_000_000
i = 0.025
n = 20

mortgage_b = Mortgage(p, i, n)

for i in range(n*12):
    mortgage_b.MakePayment()

schedule_b = mortgage_b.AmortizationSchedule()

schedule_b
```

Out[4]:

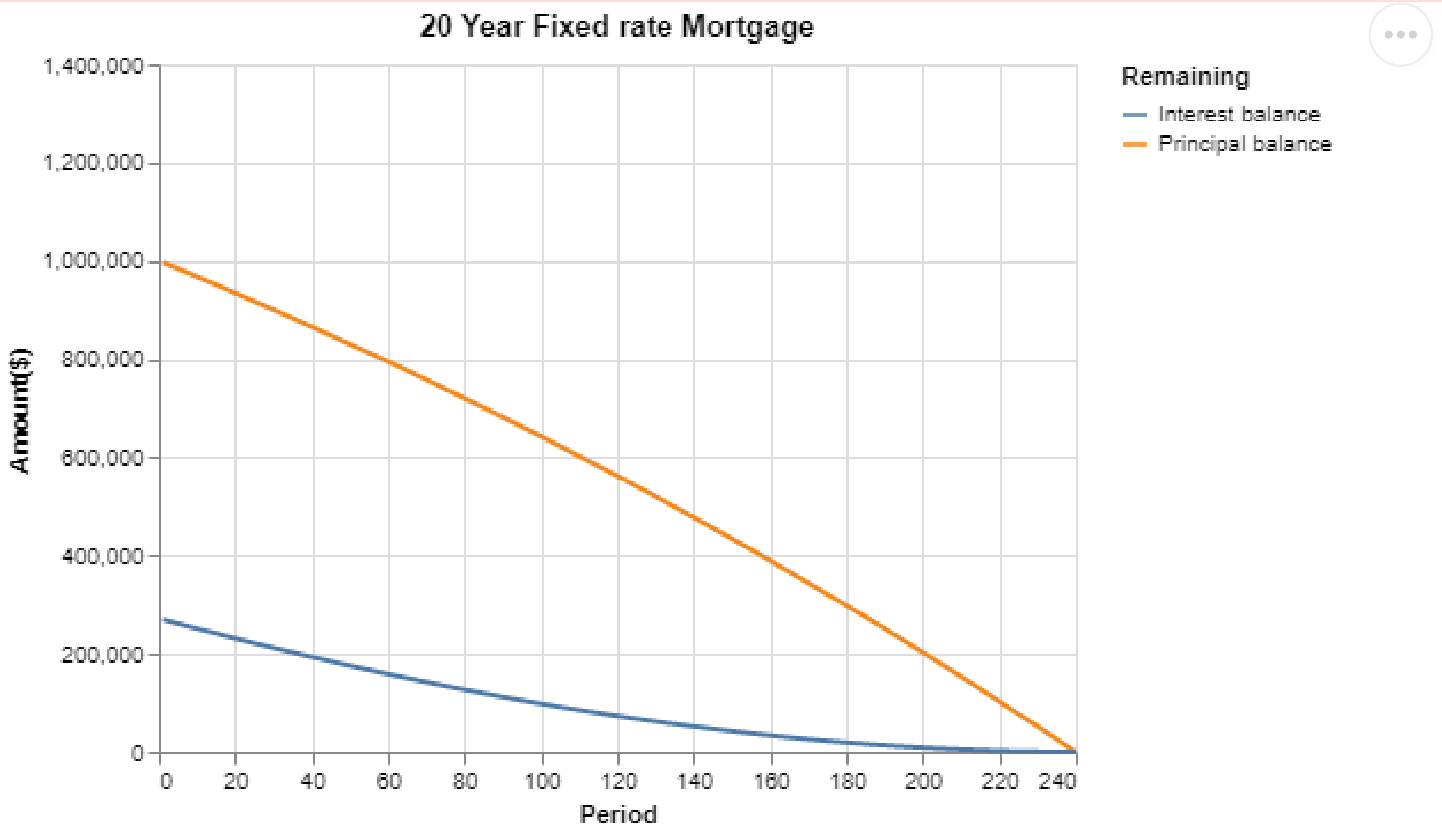
	Period	Payments	Principal	Interest	Unpaid Balance
0	0	0.00	0.00	0.00	1000000.00
1	1	5299.03	3215.70	2083.33	996784.30
2	2	5299.03	3222.39	2076.63	993561.91
3	3	5299.03	3229.11	2069.92	990332.80
4	4	5299.03	3235.84	2063.19	987096.97
...	...	...	...	...	...
236	236	5299.03	5244.17	54.85	21086.18
237	237	5299.03	5255.10	43.93	15831.08
238	238	5299.03	5266.05	32.98	10565.03
239	239	5299.03	5277.02	22.01	5288.01
240	240	5299.03	5288.01	11.02	0.00

241 rows × 5 columns

In [5]:
```
schedule_b_bal = schedule_b.copy()
schedule_b_bal["Interest balance"] = schedule_b_bal["Interest"].sum() - schedule_b_bal["Interest"].cumsum()
schedule_b_bal["Principal balance"] = schedule_b_bal["Principal"].sum() - schedule_b_bal["Principal"].cumsum(

schedule_b_melt = schedule_b_bal.drop(0).melt(id_vars="Period", value_vars=["Principal balance", "Interest ba
                                              value_name="Amount($)", var_name="Remaining")

b_plot = alt.Chart(schedule_b_melt).mark_line()\
            .encode(x="Period", y=alt.Y("Amount($)", scale=alt.Scale(domain=[0, 1_400_000])), color="Remainin
            .properties(title = "20 Year Fixed rate Mortgage")

b_plot
```

```
C:\Users\Abishek\anaconda3\envs\zen\lib\site-packages\altair\utils\core.py:317: FutureWarning: iteritems is d
eprecated and will be removed in a future version. Use .items instead.
  for col_name, dtype in df.dtypes.iteritems():
```

Out[5]:



30 year ARM

For the ARM, the reset rate of an year is computed as the average of the predicted rates for that year.

In [6]:
```
df = pd.read_csv("MORTGAGE30US.csv", parse_dates=["DATE"])[["DATE", "MORTGAGE30US"]]
sim_rates = df[df["DATE"] > "1991-01-01"]
sim_rates = sim_rates.groupby(sim_rates["DATE"].map(lambda x: x.year)).mean()
sim_rates.index.name = "YEAR"
sim_rates[-24:-1]
```

```
C:\Users\Abishek\AppData\Local\Temp\ipykernel_8172\169174269.py:3: FutureWarning: The default value of numeri
c_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Eithe
r specify numeric_only or select only columns which should be valid for the function.
  sim_rates = sim_rates.groupby(sim_rates["DATE"].map(lambda x: x.year)).mean()
```

Out[6]:

	MORTGAGE30US
YEAR	
1999	7.44
2000	8.05
2001	6.97
2002	6.54
2003	5.83
2004	5.84
2005	5.87
2006	6.41
2007	6.34
2008	6.03
2009	5.04
2010	4.69
2011	4.45
2012	3.66
2013	3.98
2014	4.17
2015	3.85
2016	3.65
2017	3.99
2018	4.54
2019	3.94
2020	3.11
2021	2.96

In [7]:

```
p = 1_000_000
i = sim_rates["MORTGAGE30US"][1992] / 100
n = 30
var = 23
reset = 1
reset_rates = (sim_rates["MORTGAGE30US"].iloc[-24:-1] / 100).to_list()
reset_rates

mortgage_c = Mortgage(p, i, n, var, reset, reset_rates)

for i in range(n*12):
    mortgage_c.MakePayment()

schedule_c = mortgage_c.AmortizationSchedule()

schedule_c
```

Out[7]:

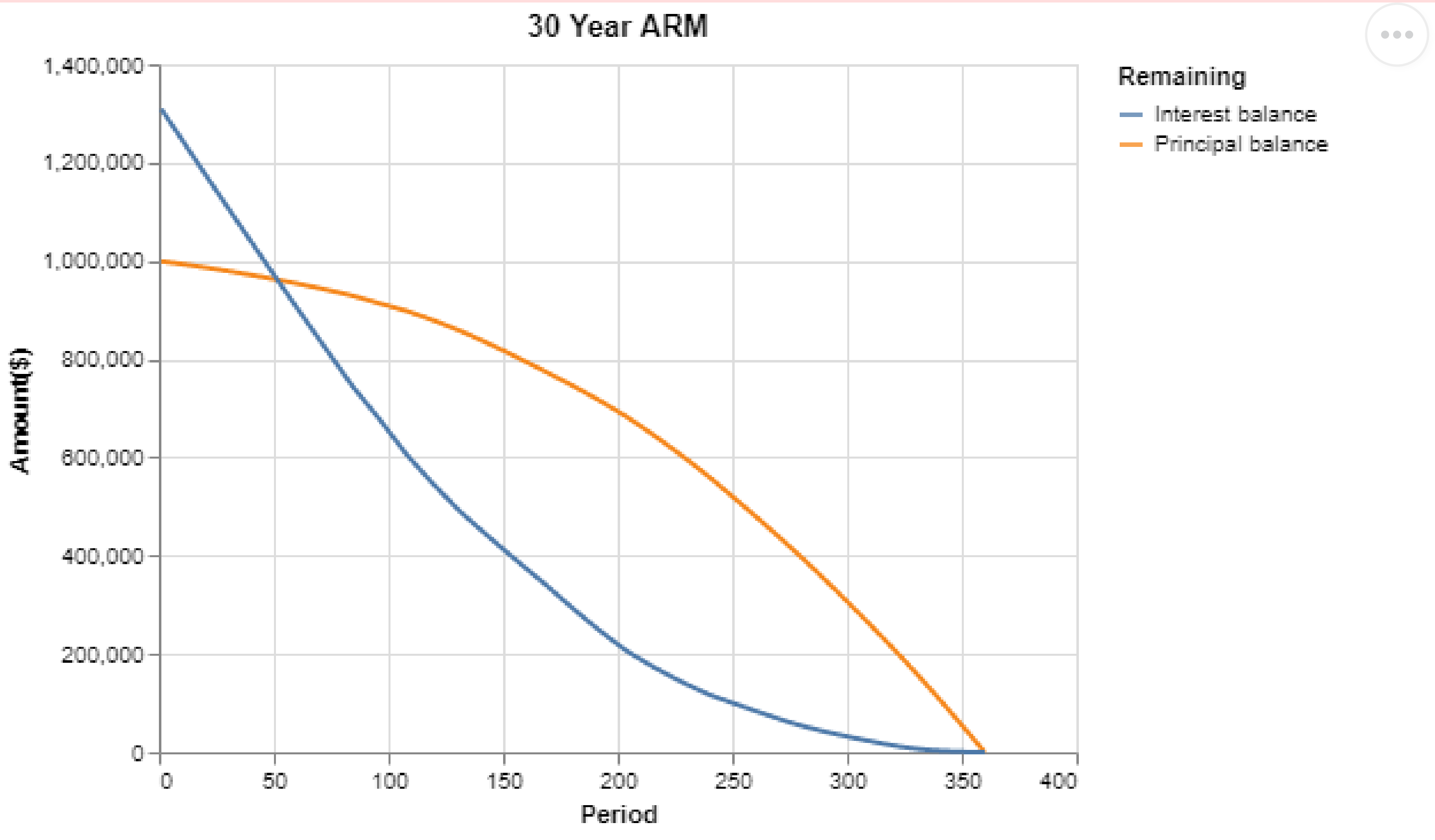
	Period	Payments	Principal	Interest	Unpaid Balance
0	0	0.00	0.00	0.00	1000000.00
1	1	7611.45	619.62	6991.82	999380.38
2	2	7611.45	623.95	6987.49	998756.42
3	3	7611.45	628.32	6983.13	998128.11
4	4	7611.45	632.71	6978.74	997495.39
...	...	...	...	...	...
356	356	5606.14	5537.56	68.58	22287.06
357	357	5606.14	5551.21	54.93	16735.85
358	358	5606.14	5564.89	41.25	11170.96
359	359	5606.14	5578.61	27.53	5592.35
360	360	5606.14	5592.35	13.78	-0.00

361 rows × 5 columns

In [8]:
```
schedule_c_cal = schedule_c.copy()
schedule_c_cal["Interest balance"] = schedule_c_cal["Interest"].sum() - schedule_c_cal["Interest"].cumsum()
schedule_c_cal["Principal balance"] = schedule_c_cal["Principal"].sum() - schedule_c_cal["Principal"].cumsum(

schedule_c_melt = schedule_c_cal.drop(0).melt(id_vars="Period", value_vars=["Principal balance", "Interest ba
                                              value_name="Amount($)", var_name="Remaining")

c_plot = alt.Chart(schedule_c_melt).mark_line()\
            .encode(x="Period", y=alt.Y("Amount($)", scale=alt.Scale(domain=[0, 1_400_000])), color="Remainin
            .properties(title = "30 Year ARM")

c_plot
```

```
C:\Users\Abishek\anaconda3\envs\zen\lib\site-packages\altair\utils\core.py:317: FutureWarning: iteritems is d
eprecated and will be removed in a future version. Use .items instead.
  for col_name, dtype in df.dtypes.iteritems():
```

Out[8]:


30 Year ARM
Remaining
Interest balance
Principal balance
Amount($)
Period