Build a simple machine learning (Linear Regression) model to find out 'Census Growth'.
This may useful for Data Science and Machine Learning enthusiasts.. - Rajendra Tuturi

a. Dataset used: Census_Data1.csv
b. Output Predicted Result: Census_Data1-Out-Predicted-Results.xls
c. Code file: Cencus-Growth-Pred-01-py.txt
d. Info description doc: Cencus Growth - Linear Regression ML model- 01.doc

#Census Growth Predict Assignment – 01

Code: Cencus-Growth-Pred-01.jpynb

#Step-1: Data collection and Import libraries
 # Step-1A. Importing required libraries

```
#Step-1: Data collection and Import libraries
# Step-1A. Importing required libraries

import csv
import time
import numpy as np
import pandas as pd

#from sklearn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn import model_selection

#for ploting graphs
from matplotlib import pyplot as plt
from matplotlib import rcParams
```

 # Step-1B. Read the Census Datasets

```
census_data = pd.read_csv('Census_Data1.csv', header = 0,
    dtype={'Growth_1991_2001': float} )

# shape of the Census datasets
print('Shape of Census data :', census_data.shape)
```

Out 6:
```
Shape of Census data : (27, 7)
```

```
#census_data.head()
census_data.head()
```

#Step-2. Data preprocessing/Data Cleansing:
#Since, sklearn requires all inputs to be numeric, we should convert all our categorical variables
#into numeric by encoding the categories. Before that we fill all the missing values in the dataset.

 # Step-2A: First, Check missing values in the dataset:
This command should tell us the number of missing values in each column as isnull() returns 1,
#if the value is null.
I've already done some data preprocessing on census data, so may not have missing values in
#the dataseet

census_data.apply(lambda x: sum(x.isnull()),axis=0)

```
Out:
SNo                   0
State                 0
District              0
Persons               0
Males                 0
Females               0
Growth_1991_2001      0
dtype: int64
```

#Second, Fill the missing values in the dataset (census_data)
#example: Persons has no missing values, but if any, then fill the missing values as below:

census_data['Persons'].fillna(census_data['Persons'].mode()[0], inplace=True)

 # Step-2B. Eliminate unused columns and use only required

 # First, assign the data to dataframe(df) for data manupulations
df = pd.DataFrame(census_data)

print(df)

#Second, Eliminate the unused columns with alphanumaric data.
#We consider only required columns with numeric data

df = df.drop(columns=['SNo','State','District','Males', 'Females'],axis=0)
print(df)

#Step-2C, re-check whether missing values filled in the dataframe:
df.apply(lambda x: sum(x.isnull()),axis=0)

Step-3: Distribution Analysis:

 # a. Now, to analyze data, we can look at frequency distribution for non numeric values (from census_data) like 'State'

```
census_data['State'].value_counts()
```

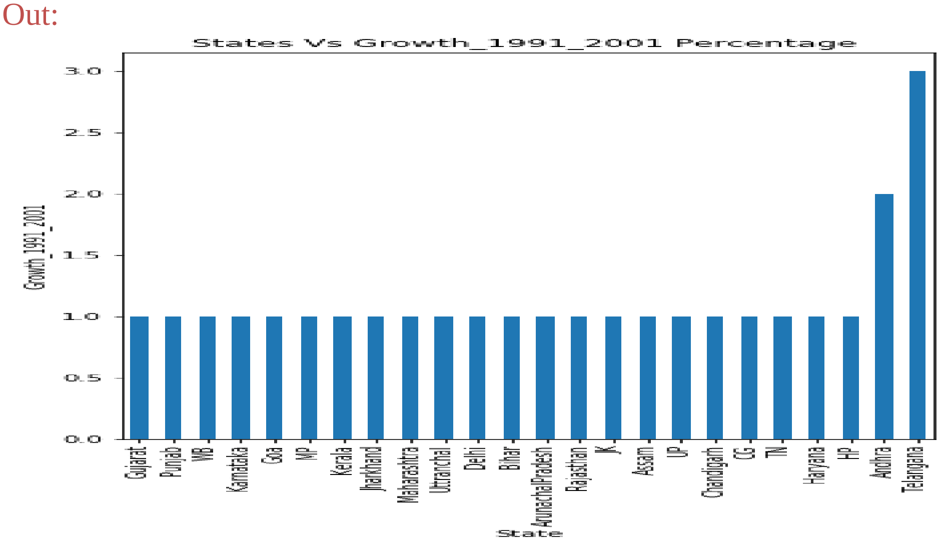
 # b. Growth_1991_2001 by State:

```
temp1 = census_data['Growth_1991_2001'].value_counts(ascending=True)
temp2 = census_data['State'].value_counts(ascending=True)

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(122)
ax1.set_xlabel('State')
ax1.set_ylabel('Growth_1991_2001')
temp1.plot(kind='bar')

ax2 = fig.add_subplot(122)
temp2.plot(kind = 'bar')
ax2.set_xlabel('State')
ax2.set_ylabel('Growth_1991_2001')
ax2.set_title("States Vs Growth_1991_2001 Percentage ")
```

Out:



'''

 # Step-4: Building a Predictive Model in Python   Linear Regression Model

seperate the indepndent and target variable on training data
Now, we need to predict the missing target variable in the test data

 # Step-4A: #Standard processing and Training/Test set Split
 # target variable - Growth_1991_2001
'''

x = df.loc[:, df.columns != "Growth_1991_2001"]
y = df.loc[:, "Growth_1991_2001"]
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=0.3, random_state=7)

'''
#Step-4B: Creat Model object

#Create the object of the Linear Regression
#You can also add other parameters and test your code here
#Some parameters are : fit_intercept and normalize
#Growth-1991-2001, Percentage_SC_to_total, Percentage_Non_Workers,
#Percentage_to_total_population_ST
'''
model = LinearRegression()

'''
#Step-4B: # fit the model with the data
'''

model.fit(x_train, y_train)

Out:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

'''
#Step-4C: coefficeint & intercept of model
'''
# coefficeints of the trained model
print('\nCoefficient of model :', model.coef_)

# intercept of the model
print('\nIntercept of model',model.intercept_)

```
Out:
Coefficient of model : [-1.73045757e-07]

Intercept of model 23.03774061888616
```

'''
#Step-4D: # predict the target 'Growth_1991_2001' on the training dataset
'''

```
predict_train = model.predict(x_train)
print('\Predict Growth_1991_2001 on training data',predict_train)
```
Out:
```
\Predict Growth_1991_2001 on training data [22.90647278 22.76377025
22.75928023 22.43349564 22.98668987 22.881002
 22.67365044 22.53645077 22.54267938 22.94092238 22.93627922 22.95798608
 22.51108174 22.81075494 22.2650664  22.72808856 22.40777135 22.85855797]
```

'''
 # Step-4E: Evaluate the Model on training data
 # Root Mean Squared Error on training dataset
'''

```
rmse_train = mean_squared_error(y_train,predict_train)**(0.5)
print('\nRMSE on train dataset : ', rmse_train)
```
Out:
```
RMSE on train dataset :  10.965318911412524
```

'''
 # Step-4F: # predict the target 'Growth_1991_2001' on the test dataset
'''
```
predict_test = model.predict(x_test)
print('\nPredict Growth_1991_2001 on test data',predict_test)
```
Out:
```
Predict Growth_1991_2001 on test data [22.4760334  22.88188955 22.37501811
23.03100499 22.30843841 22.92519287
 22.95668408 22.56104178 22.82938003]
```

'''
 # Step-4G: Evaluate the Model on test data
 # Root Mean Squared Error on test dataset
'''

```
rmse_test = mean_squared_error(y_test, predict_test)**(0.5)
print('\nRMSE on train dataset : ', rmse_test)
```

```
Out:
RMSE on train dataset :  13.203196822823575
```

'''

Step-5: Visualize: Plotting the best fit line - Linear Regression

#Linear Regression
#It is used to estimate the real values (Dependent Variable - Y = ‘Growth_1991_2001’)
#based on continuous variable(s) (Independent variable - X = ‘Persons’).
#We establish a relationship between independent and dependent variables by fitting a best line.
#This best fit line is known as regression line and represented by
#a linear equation Y= m *X + c. Here, m – Slope and c – Intercept

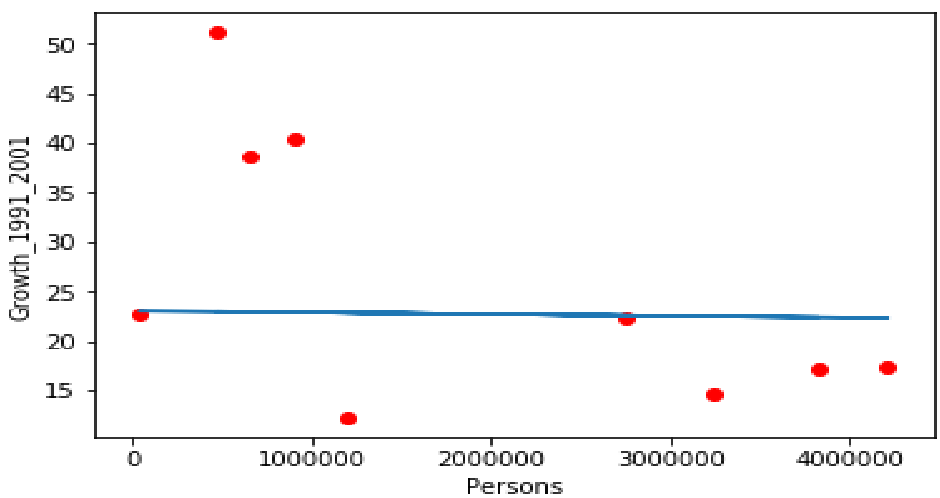
#To see the relationship between the training data values
#RMSE: See the scattered graph, red dots are actual values and blue line is the set of predicted
#values drawn by our model.
#Distance between the actual value and the predicted line, this line represents the error.
#Taking mean of all those distances and squaring them and finally taking the root will give us
#RMSE of our model.
'''

```
plt.scatter(x_train,y_train,c='red')
plt.show()
```

#to see the relationship between the predicted
#‘Growth_1991_2001’ values using scattered graph

```
plt.plot(x_test, predict_test)
plt.scatter(x_test,y_test,c='red')
plt.xlabel('Persons')
plt.ylabel('Growth_1991_2001')
```

Out:



#error in each value

```
for i in range(1, 3):
    print("Error in value number", i, (y_test[i]- predict_test[i]))
```

Out:
```
Error in value number 1 -5.701889553633933
Error in value number 2 -7.745018112318112
```

#combined rmse value

```
rss=((y_test-predict_test)**2).sum()
mse=np.mean((y_test-predict_test)**2)
print("Final rmse value is =",np.sqrt(np.mean((y_test-predict_test)**2)))
```

Out:
```
Final rmse value is = 13.203196822823573
```