Context

Run a linear regression with Total Income as the dependent variable and your choice of four other variables as the independent variables (NOTE: Don't limit your analysis to the variables presented; feel free to try aggregations or categorical variables from the original variables). Explain why you chose those independent variables and give a general analysis of the results and what they mean.

Basic Exploration

Let's have a look at what is avaiable in the data frame first.

In [ ]:
```
import pandas as pd
import numpy as np
from datetime import date, datetime,  timedelta
import seaborn as sns
```

In [ ]:
```
data = pd.read_csv('data/aes-2015-csv.csv')
data.head()
```

Out[ ]:

	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Ur
0	2015	Level 1	99999	All industries	Doll (millio
1	2015	Level 1	99999	All industries	Doll (millio
2	2015	Level 1	99999	All industries	Doll (millio
3	2015	Level 1	99999	All industries	Doll (millio
4	2015	Level 1	99999	All industries	Doll (millio

Clearly, this is a table with many variables embeded as rows, some reshaping/pivoting will be required. \ But first let's see how many unique variables are there

In [ ]:
```
unique_variables = data['Variable_name'].unique()
unique_variables.shape
```

Out[ ]: (41,)

It seems like the basic structure of the tbale is, for every industry, across every year (between 2013-15), across every industry level (1, 3, 4), there are 41 variables (including Total Income) that are measured (in the Value column)

So to make things easier, we can change this long table to a wide table, by first controlling for year and industry level, then stack together the pivotted tables

In [ ]:
```
l = list(data.groupby(['Year', 'Industry_aggregation_NZSIOC']))
tables = []
for i in l:
    out = i[1].pivot_table(index= ['Industry_code_NZSIOC'], columns='Variable
    tables.append(out)

wide_df = pd.concat(tables).reset_index('Industry_code_NZSIOC')
wide_df.head()
```

Out[ ]:

Variable_name	Industry_code_NZSIOC	Additions to fixed assets	Closing stocks	Current assets	Current liabilities	Current ratio	Depr
0	99999	NaN	50442	488032	631136	77	
1	AA	NaN	13158	24740	22334	111	
2	BB	1279	337	20433	20361	100	
3	CC	3924	12367	32192	25996	124	
4	DD	2993	369	6080	8731	70	

5 rows × 42 columns

In [ ]:
```
#clean up the data a little bit.
for col in wide_df.columns[1:]:
    wide_df[col] =pd.to_numeric(wide_df[col], errors ='coerce').fillna(0).as
```

In [ ]:
```
# Remove unwanted columns
unwanted_columns = ['Industry_code_NZSIOC', 'Additions to fixed assets', 'Cl
        'Current ratio',
       'Depreciation', 'Disposals of fixed assets',
       'Liabilities structure', 'Margin on sales of goods for resale',
        'Opening stocks',
       'Purchases of goods bought for resale', 'Quick ratio',
         'Return on equity',
       'Return on total assets',
       'Shareholders funds or owners equity', 'Surplus before income tax',
       'Surplus per employee count', 'Surplus per employee count(3)',
        'Total income per employee count(3)']
wide_df.drop(unwanted_columns, axis=1, inplace=True)
```

It's always a good idea to look to have a closer look at our data — especially for outliers. Let's start by printing out some summary statistics about the data set.

In [ ]:
```
wide_df.describe()
```

Out[ ]:

Variable_name	Current assets	Current liabilities	Fixed tangible assets	Government funding, grants and subsidies	Indirect taxes
count	417.000000	417.000000	417.000000	417.000000	417.000000
mean	13384.165468	18013.364508	9889.047962	13.956835	169.959233
std	54946.648482	81619.358185	38453.560124	50.350245	635.830840
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	718.000000	588.000000	429.000000	0.000000	7.000000
50%	1848.000000	1636.000000	1096.000000	0.000000	19.000000
75%	5228.000000	3933.000000	3848.000000	2.000000	59.000000
max	524779.000000	700256.000000	410187.000000	492.000000	6438.000000

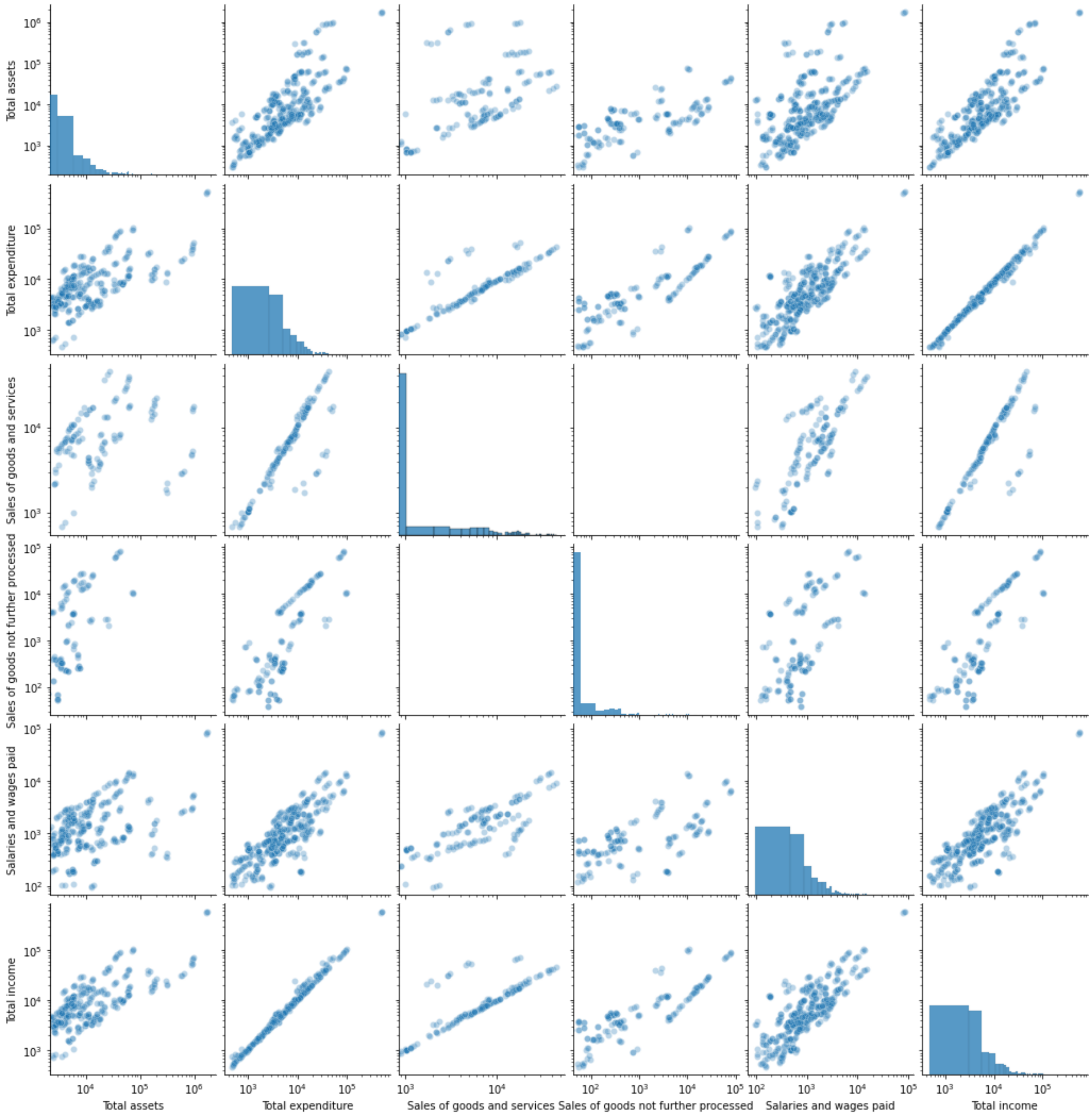
8 rows × 24 columns

In [ ]:

```
# Measurements scatterplot
# Without knowing a lot about business mechanism, I am taking a guess at pos
measurements = ['Total assets',  'Total expenditure', 'Sales of goods and se
selected_columns = measurements + ['Total income']
g = sns.pairplot(wide_df[selected_columns], plot_kws={'alpha': 0.3})
g.set(xscale="log")
g.set(yscale="log")
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x10887d730>
```



We can that Total Income highly correlates with Total Expenditure, Sales of goods and services. Loosely with Total assests and Salaries and wages paid. These will be a good factors for running regression

Let's start doing regression

Preparing the data set

In [ ]:

```
# Set a random seed number to reproduce our results
seed = 123

# 1. Load the dataset
df = wide_df

# 2. Select the columns of interest for modelling
features = ['Total assets',  'Total expenditure', 'Sales of goods and servic

# 3. Create the features matrix as X
X = df[features]

# 4. Create the labels vector as y
y = df['Total income']
```

Split the dataset

Create a training and test dataset using the train_test_split function

In [ ]:
```
# Import the function
from sklearn.model_selection import train_test_split

# Split the dataset into X_train, X_test, y_train, y_test
# Use a training dataset size of 80%
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, ra
```

In [ ]:
```
# Step 1: Import the classes
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Step 2: Instantiate the estimators
lr = LinearRegression()

# Step 3: Fit the estimators on data (i.e. train the models)
lr.fit(X_train, y_train)

# Step 4: Generate predictions
y_pred = lr.predict(X_test)

# Calculate the Root Mean Squared Error (RMSE)
# np.sqrt(mean_squared_error(...))
score = np.sqrt(mean_squared_error(y_test, y_pred))

# Display the model scores
print('Linear regression: %.3f' % (score))
```

```
Linear regression: 922.774
```

In [ ]:
```
# calculate the correlation between the labels and predictions
corr = np.corrcoef(y_test, y_pred)[0][1]

print('Linear regression: %.3f' % (corr))

print(lr.intercept_)
print(lr.coef_)
```

```
Linear regression: 0.999
-4.16392091815942
[ 0.01713442  1.05001024  0.04869003 -0.03276933  0.06065477]
```

In [ ]:
```
# pair coefficients with feature names
coeff = list(zip(features, lr.coef_))
coeff
```

Out[ ]:
```
[('Total assets', 0.01713441866806446),
 ('Total expenditure', 1.0500102383852488),
 ('Sales of goods and services', 0.048690030507279464),
 ('Sales of goods not further processed', -0.0327693317747218),
 ('Salaries and wages paid', 0.060654767030396045)]
```

What this is suggesting is that Total Income can be estimated in the following way:

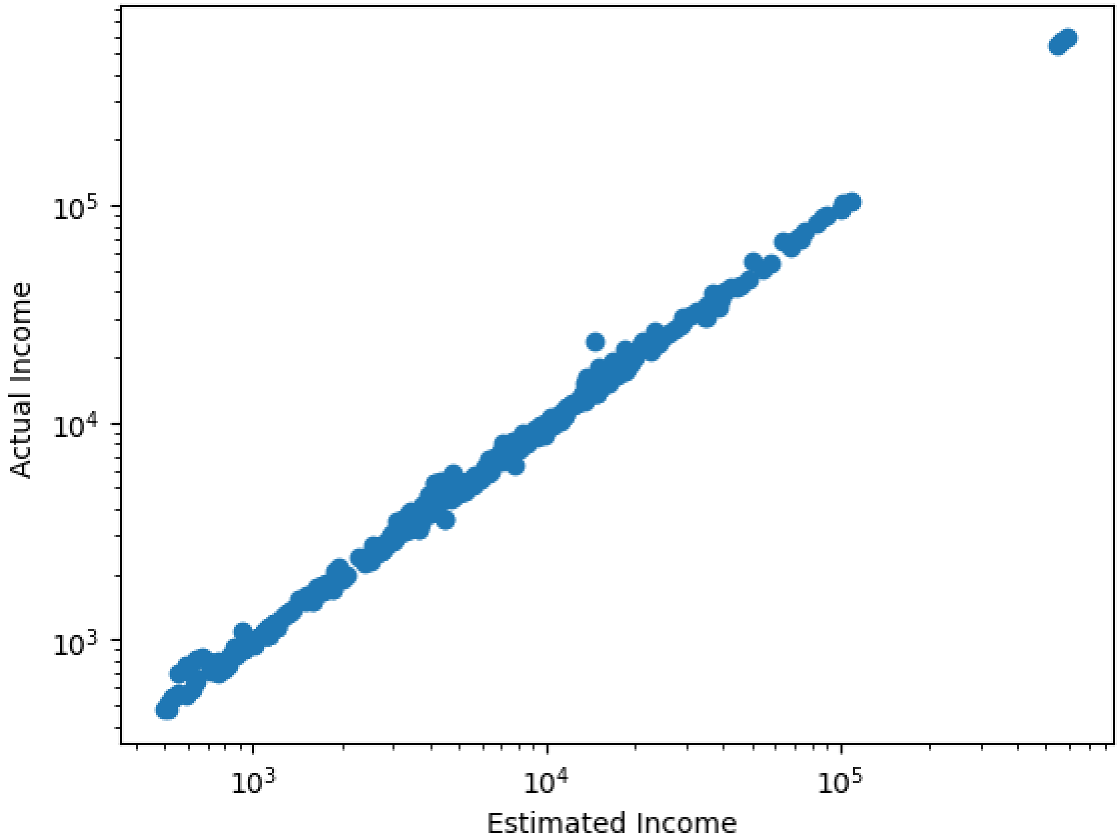
$$\begin{gathered}Total\ Income = 0.0171 * Total\ assets + 1.0500 * Total\ expenditure \\ + 0.0487 * Sales\ of\ goods\ and\ services - 0.0328 * Sales\ of\ goods\ not\ further\ proces \\ + 0.0607 * Salaries\ and\ wages\ paid\end{gathered}$$

In [ ]:
```
#Visualise and test
from matplotlib import pyplot as plt

y_est = X[coeff[0][0]]*coeff[0][1]+X[coeff[1][0]]*coeff[1][1]+X[coeff[2][0]]

plt.figure()

plt.scatter(y_est, y)
plt.xlabel('Estimated Income ')
plt.ylabel('Actual Income ')
plt.yscale('log')
plt.xscale('log')
plt.show()
```



The plot shows a really good estimation of the total income, meaning the above equation is pretty accurate in terms of estimating the number. \ However, one must always remember, correlation is NOT causation. e.g. It doesn't mean increasing the total expenditure will guarantee higher total income. \ I picked those variable based on purely their surface meaning, and my best guess on how they play in the business. \ To improve on this regression, one must have a deeper look at the company finance, and get the underlying mechanism of this micro-economy.