Non-homogeneous data: make it (more) Homogeneous

Normalization via z-score

Let's consider a simple dataset with examples that are drawn from two different groups

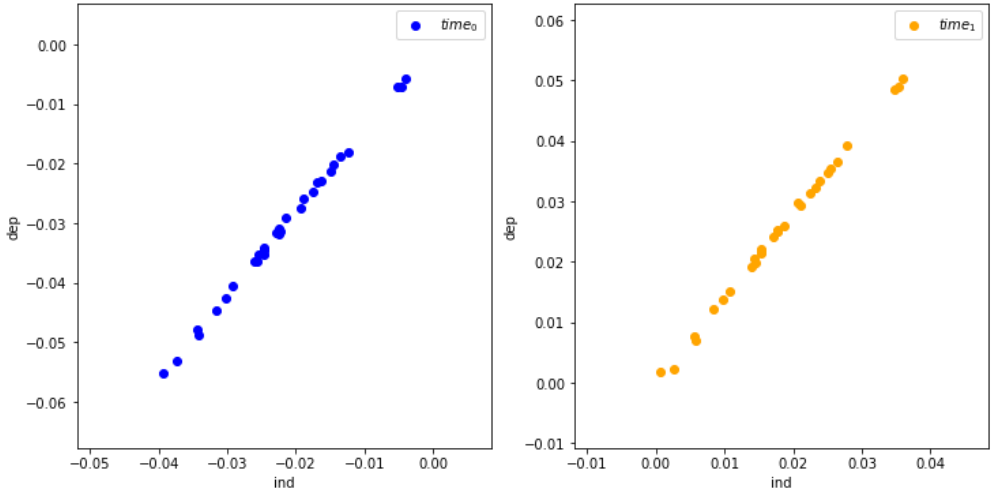
From the top graph: we can see that there is a constant linear relationship

- between target "dep" and feature "ind"
- both within groups and across groups

From the second and third rows, we see the distribution of features and targets

- has same shape between groups
- with different means

In [8]:
```
_= sph.plot_segments(df_2means)
```



Given the simple linear relationship intra-group

- No harm would come from pooling
- Even though the pooled data comes from distinct groups

However: it the intra-group relationship was more complex (e.g., a curve)

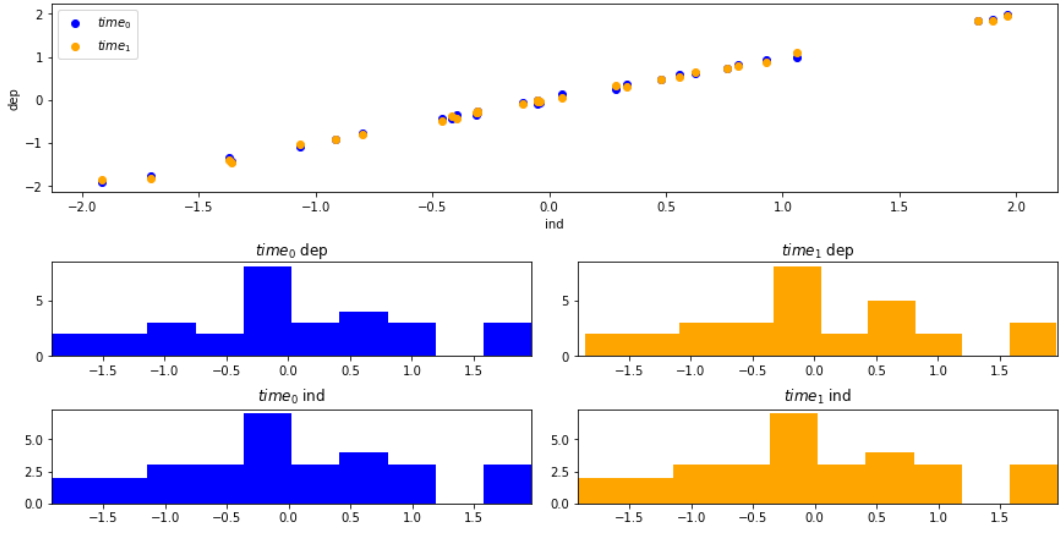
- pooling would be less successful

So although this example may be over-simplified, we still try to make the distinct groups look similar.

Let's normalize each group

- for each variable: turn values into z-scores
    - subtract variable mean, divide by variable standard deviation

In [9]:
```
df_2means_norm = sph.normalize_data(df_2means)
_= sph.plot_data(df_2means_norm)
```



You can now see that the two groups are

- congruent in the top joint plot
- have same distributions in the second and third rows

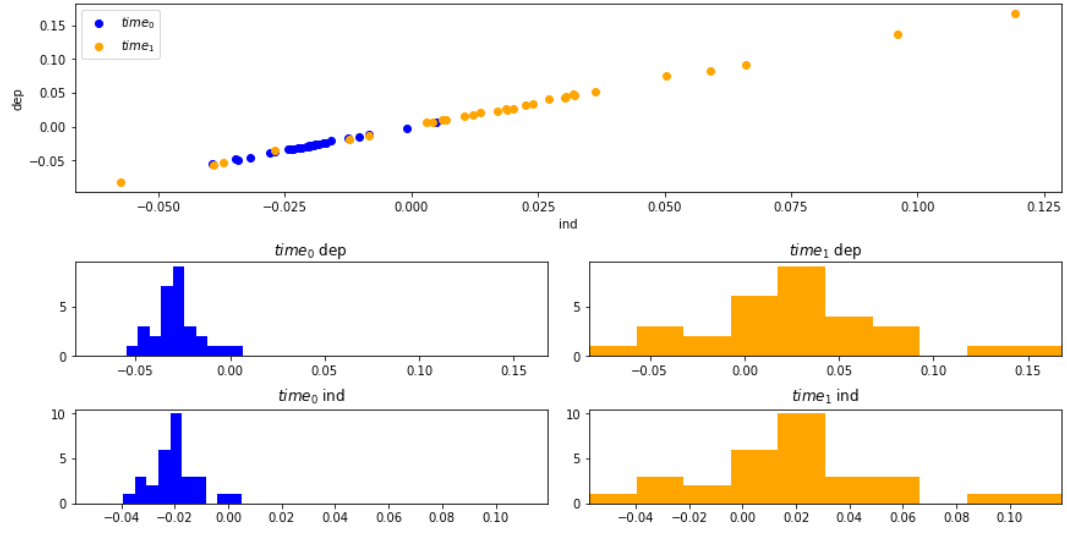
Non-homogeneous groups made homogeneous !

We can make the separation between groups less trivial by also having different standard deviations per group.

Here's what the data looks like

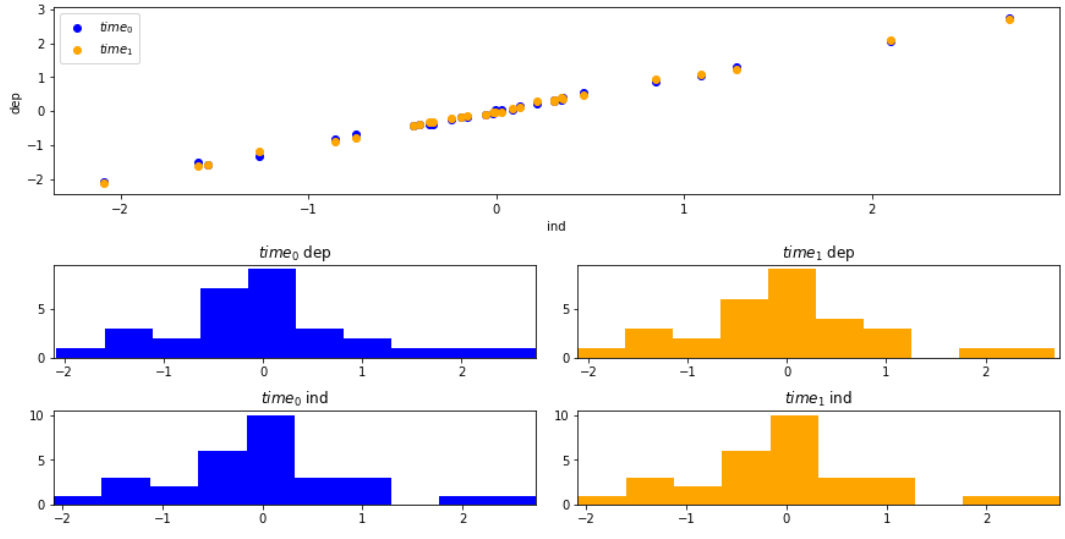
In [10]:
```
df_2means_2sdevs = sph.gen_returns(means, [s, 4*s])

_= sph.plot_data(df_2means_2sdevs)
```



Again: normalization does the trick

In [11]:
```
df_2means_2sdevs_norm = sph.normalize_data(df_2means_2sdevs)
_= sph.plot_data(df_2means_2sdevs_norm)
```

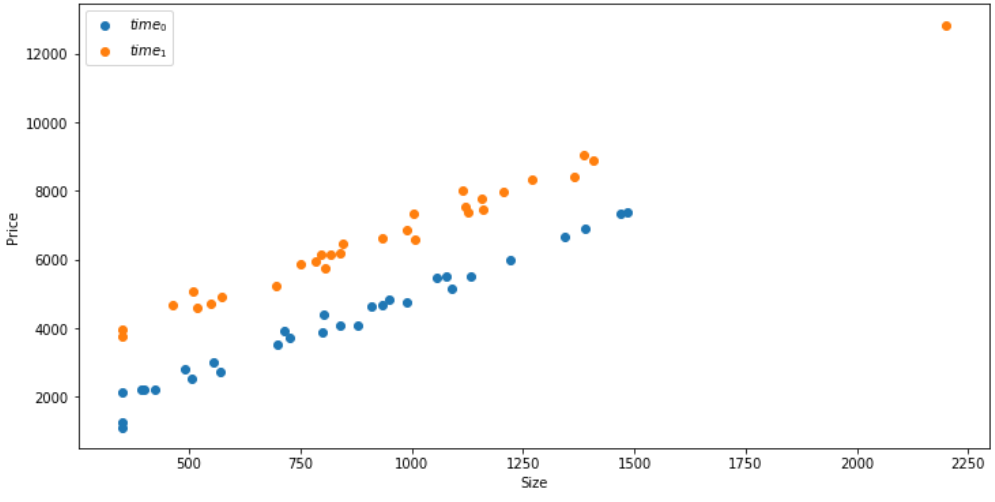


Pooled over time alternate method: normalization

Let's revisit our "pooled over time" dataset.

In [12]:
```
sph = transform_helper.ShiftedPrice_Helper()
series_over_time = sph.gen_data(m=30)

fig, ax = plt.subplots(1,1, figsize=(12,6) )
_= sph.plot_data(series_over_time, ax=ax)
```



We had previously addressed this by adding a missing feature

- distinct intercept per group
- by adding a "group indicator" feature

$$\mathbf{y} \quad = \quad \Theta_{(\text{time}_0)} * \text{Is}_0 + \Theta_{(\text{time}_1)} * \text{Is}_1 + \Theta_1 * \mathbf{x}$$

- after addressing the "dummy variable" trap, the equation is of the form

$$\mathbf{y} = \Theta_0 + \Theta_1 * \mathbf{x}$$

  We show an alternate solution using a (trivial) standardization

We will standardize both the target $\mathbf{y}$ and the single feature $\mathbf{x}$

- transform each to mean 0

$$\begin{aligned} \mathbf{y}' &= \mathbf{y} - \bar{\mathbf{y}} \\ \mathbf{x}' &= \mathbf{x} - \bar{\mathbf{x}} \end{aligned}$$

- no scaling by volatility

Model equation

$$\mathbf{y}' = \Theta'_0 + \Theta'_1 * \mathbf{x}'$$

Here is the math behind this "de-meaning" transformation.

If

$$\mathbf{y} = \Theta_0 + \Theta_1 * \mathbf{x}$$

then

$\mathbf{y^{(i)}}$	$=$	$\Theta_0 + \Theta_1 * \mathbf{x^{(i)}}$	hypothesize linear relationship
$\frac{1}{m}\sum_i \mathbf{y^{(i)}}$	$=$	$\frac{1}{m}\sum_i (\Theta_0 + \Theta_1 * \mathbf{x^{(i)}})$	sum over all examples, divide by no.
$\bar{\mathbf{y}}$	$=$	$\Theta_0 + \Theta_1 * \bar{\mathbf{x}}$	definition of average
Θ_0	$=$	$\bar{\mathbf{y}} - \Theta_1 * \bar{\mathbf{x}}$	re-arrange terms

When $\bar{\mathbf{x}} = 0$ (as is the case for $\bar{\mathbf{x}}'$)

- $\Theta_0 = \bar{\mathbf{y}}'$

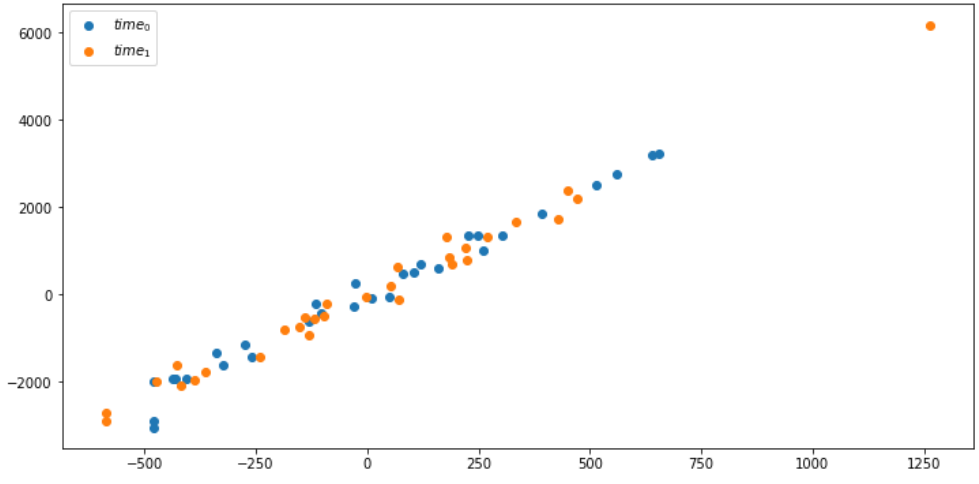
That is: we have transformed the two groups so as to have a common intercept.

Here is a little code to demonstrate this

In [13]:
```
fig, ax = plt.subplots(1,1, figsize=(12,6) )

demean_x0 = sph.x0 - sph.x0.mean()
demean_x1 = sph.x1 - sph.x1.mean()

_= ax.scatter(demean_x0, sph.y0 - sph.y0.mean(), label="$time_0$")
_= ax.scatter(demean_x1, sph.y1 - sph.y1.mean(), label="$time_1$")
_= ax.legend()
```



Now it looks like each group comes from the same distribution.

- We can pool the observations from the two groups

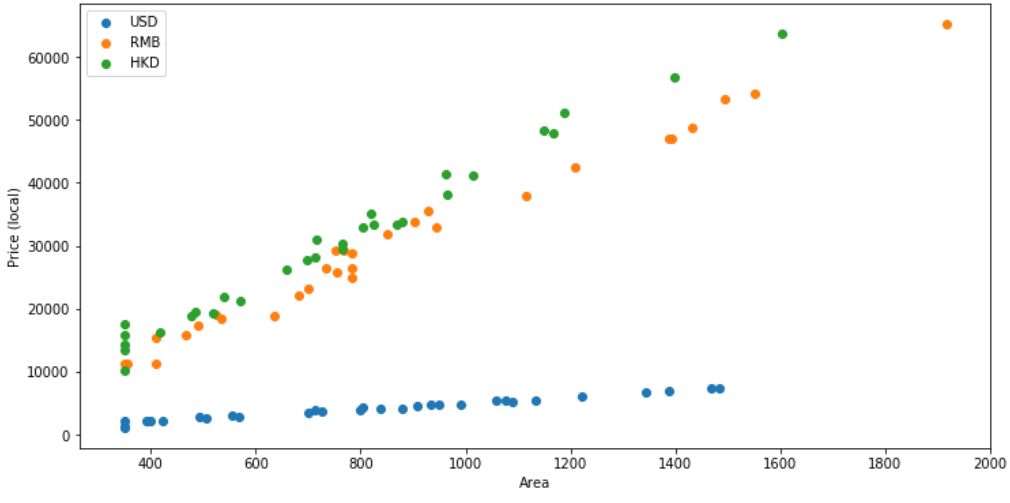
Normalization by uncovering the hidden relationship

Consider the following multi-group data (our multiple geography pooling of data)

- house price as a function of size
- in different geographies

In [14]:
```
fig_rp
```

Out[14]:



There is clearly a linear relationship intra-group, but the slope differs between groups (local currencies).

The apparent diversity in the target may obscure a simple relationship that is common to all groups

Let's re-denominate the target in a common unit.

- Let the target of example i in group g be
$$\mathbf{y}^{(\mathbf{i})}_{(\text{group}_g)}$$
- Change the units in which $\mathbf{y}^{(\mathbf{i})}_{(\text{group}_g)}$ is expressed
- Into a common unit
- Via an "exchange rate" equal to the slope of group g
$$\beta_{(\text{group}_g)}$$
- yielding
$$\tilde{\mathbf{y}}^{(\mathbf{i})}_{(\text{group}_g)} = \frac{\mathbf{y}^{(\mathbf{i})}_{(\text{group}_g)}}{\beta_{(\text{group}_1)}}$$

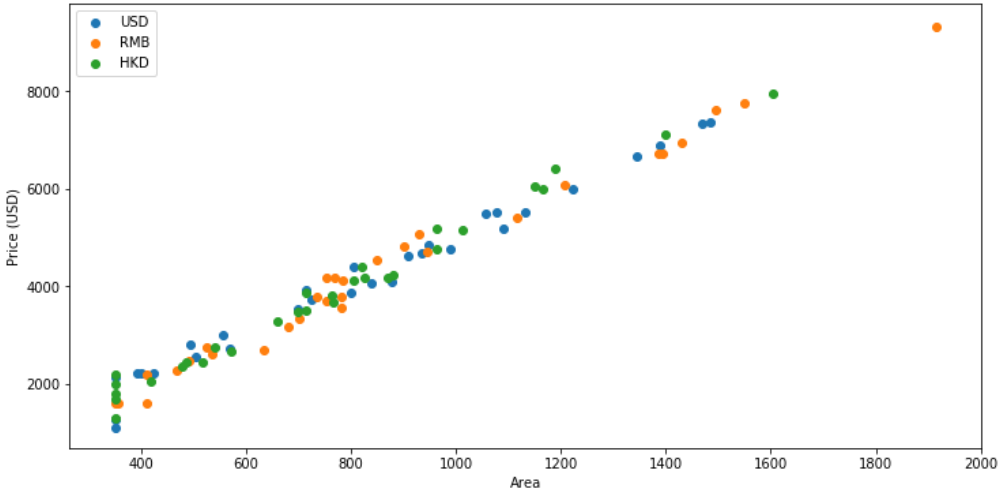
If we do this for each group (separately) the data becomes homogeneous !

In [15]:
```
# Relative price levels
rel_price = rph.relative_price()

# Normalize the price of each series by the relative price
series_normalized = [ series[i]/(1,rel_price[i]) for i in range(len(series))]

fig_rp_norm, ax_rp_norm = plt.subplots(1,1, figsize=(12,6))
_= rph.plot_data(series_normalized, ax=ax_rp_norm, labels=labels, xlabel="Area",
ylabel="Price (USD)")

# plt.close(fig_rp_norm)
```



We can see why this is true.

Here's an equation that describes the data for the first two sub-groups:

$$\begin{aligned}\mathbf{y}_{(\text{group}_0)} &= \beta_{(\text{group}_0)} * (\Theta_0 + \Theta_1 * \mathbf{x}) \\ \mathbf{y}_{(\text{group}_1)} &= \beta_{(\text{group}_1)} * (\Theta_0 + \Theta_1 * \mathbf{x})\end{aligned}$$

And the regression equation for the transformed data:

$$\begin{aligned}\frac{\mathbf{y}_{(\text{group}_0)}}{\beta_{(\text{group}_0)}} &= \Theta_0 + \Theta_1 * \mathbf{x} \\ \frac{\mathbf{y}_{(\text{group}_1)}}{\beta_{(\text{group}_1)}} &= \Theta_0 + \Theta_1 * \mathbf{x}\end{aligned}$$

The common relationship

$$\tilde{\mathbf{y}} = \Theta_0 + \Theta_1 * \mathbf{x}$$

is revealed.

The phenomenon of variables denominated in different units between sub-groups is not uncommon

- Examples observed in different countries, measured in local currency
- Examples observed at different times (e.g., adjust for time-varying general price level)

In our case, the only variable that needed to be normalized was the target.

You can imagine situations in which features (e.g., Area) need to be normalized

- Meters versus feet

We may need to be creative in conceptualizing the "exchange rate"

- Consider re-denominating so that "pricing power" is constant
    - Units of "number of McDonald's burgers" !
    - May work across time and currency !

As long as all commodity prices change the same, this should work.

This is a type of *scaling* transformation.

- the common relationship only becomes apparent when the target (or some features) are placed on a common scale
- often see this when target/features are scaled by their standard deviation
    - re-denominate in terms of *number of standard deviations*
    - e.g., returns of two equities are both normal but with different volatilities

Normalization: creating the correct units

There is a similar need for "re-denomination" that arises in a different context

- when the raw feature
- does not express the key semantics as well as a re-denominated feature

The Geron book has a more sophisticated example of predicting house Price from features (external/handson-ml2/02_end_to_end_machine_learning_project.ipynb#Experimenting-with-Attribute-Combinations)

- a lot more features

In [16]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In terms of predictive value, there are some features

- `total_rooms`, `total_bedrooms` that are not predictive because their units are not informative
- both features will have greater magnitude in a multi-family house than a single family house

A more meaningful feature can be synthesized by normalizing by the number of families

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
```

That is:

- the normalized variable has units "per household"
- that is more predictive of price than the raw feature

In [17]:
```
print("Done")
```

```
Done
```