In [1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab5-pca.ipynb")
```

In [2]:
```
import numpy as np
import pandas as pd
import altair as alt
from scipy import linalg
from statsmodels.multivariate.pca import PCA
# disable row limit for plotting
alt.data_transformers.disable_max_rows()
# uncomment to ensure graphics display with pdf export
# alt.renderers.enable('mimetype')
```

Out[2]:
```
DataTransformerRegistry.enable('default')
```

Lab 5: Principal components

Principal components analysis (PCA) is a widely-used multivariate analysis technique. Depending on the application, PCA is variously described as:

- a dimension reduction method
- a an approximation method
- a latent factor model
- a filtering or compression method

The core technique of PCA is *finding linear data transformations that preserve variance*.

What does it mean to say that 'principal components are linear data transformations'? Suppose you have a dataset with observations and variables. We can represent the values as a data matrix with rows and columns:

To say that the principal components are linear data transformations means that each principal component is of the form:

for some vector . In PCA, the following terminology is used:

- linear combination coefficients are known as *loadings*
- values of the linear combinations are known as *scores*

- the vector of loadings is known as a *principal axis*

As discussed in lecture, the values of the loadings are found by decomposing the correlation structure.

Objectives

In this lab, you'll focus on computing and interpreting principal components:

- finding the loadings (linear combination coefficients) for each PC;
- quantifying the variation captured by each PC;
- visualization-based techniques for selecting a number of PC's to A(nalyze);
- plotting and interpreting loadings.

You'll work with a selection of county summaries from the 2010 U.S. census. The first few rows of the dataset are shown below:

In [3]:
```
# import tidy county-level 2010 census data
census = pd.read_csv('data/census2010.csv', encoding = 'latin1')
census.head()
```

Out[3]:

	State	County	Women	White	Citizen	IncomePerCap	Poverty	ChildPov
0	Alabama	Autauga	51.567339	75.788227	73.749117	24974.49970	12.912305	18.707
1	Alabama	Baldwin	51.151337	83.102616	75.694057	27316.83516	13.424230	19.484
2	Alabama	Barbour	46.171840	46.231594	76.912223	16824.21643	26.505629	43.55
3	Alabama	Bibb	46.589099	74.499889	77.397806	18430.99031	16.603747	27.197
4	Alabama	Blount	50.594351	87.853854	73.375498	20532.27467	16.721518	26.857

5 rows × 24 columns

The observational units are U.S. counties, and each row is an observation on one county. The values are, for the most part, percentages of the county population. You can find variable descriptions in the metadata file `census2010metadata.csv` in the data directory.

Correlations

PCA identifies variable combinations that capture covariation by decomposing the correlation matrix. So, to start with, let's examine the correlation matrix for the 2010 county-level census data to get a sense of which variables tend to vary together.

The correlation matrix is a matrix of all pairwise correlations between variables. If denotes the value for the th observation of variable , then the entry at row and column of the correlation matrix is:

In the census data, the `State` and `County` columns indicate the geographic region for each observation; essentially, they are a row index. So we'll drop them before computing the matrix :

In [4]:
```python
# store quantitative variables separately
x_mx = census.drop(columns = ['State', 'County'])
```

From here, the matrix is simple to compute in pandas using `.corr()`:

In [5]:
```python
# correlation matrix
corr_mx = x_mx.corr()
```

The matrix can be inspected directly to determine which variables vary together. For example, we could look at the correlations between employment rate and every other variable in the dataset by extracting the `Employed` column from the correlation matrix and sorting the correlations:

In [6]:
```python
# correlation between employment rate and other variables
corr_mx.loc[:, 'Employed'].sort_values()
```

Out[6]:
```
ChildPoverty   -0.744510
Poverty        -0.735569
Unemployment   -0.697985
Minority       -0.439053
Service        -0.403261
MeanCommute    -0.252111
Drive          -0.215038
Carpool        -0.144336
Production     -0.136277
Citizen        -0.087343
Office         -0.014838
OtherTransp    -0.010041
FamilyWork      0.055654
Women           0.131181
Transit         0.151700
SelfEmployed    0.154107
PrivateWork     0.264826
WorkAtHome      0.303839
White           0.432856
Professional    0.473413
IncomePerCap    0.767001
Employed        1.000000
Name: Employed, dtype: float64
```

Recall that correlation is a number in the interval [-1, 1] whose magnitude indicates the strength of the linear relationship between variables:

- correlations near -1 are *strongly negative*, and mean that the variables *tend to vary in opposition*
- correlations near 1 are *strongly positive*, and mean that the variables *tend to vary together*

From examining the output above, it can be seen that the percentage of the county population that is employed is:

- strongly *negatively* correlated with child poverty, poverty, and unemployment, meaning it *tends to vary in opposition* with these variables
- strongly *positively* correlated with income per capita, meaning it *tends to vary together* with this variable

If instead we wanted to look up the correlation between just two variables, we could retrieve the relevant entry directly using `.loc[...]` with the variable names:

In [7]:
```
# correlation between employment and income per capita
corr_mx.loc['Employed', 'IncomePerCap']
```

Out[7]: 0.7670009685702536

So across U.S. counties employment is, perhaps unsurprisingly, strongly and positively correlated with income per capita, meaning that higher employment rates tend to coincide with higher incomes per capita.

Question 1

Find the correlation between the poverty rate and demographic minority rate and store the value as `pov_dem_rate`. Interpret the value in context.

Type your answer here, replacing this text.

In [8]:
```
# correlation between poverty and percent minority
pov_dem_rate = corr_mx.loc['Poverty', 'Minority']

# print
pov_dem_rate
```

Out[8]: 0.6231625196890354

In [9]:
```
grader.check("q1")
```

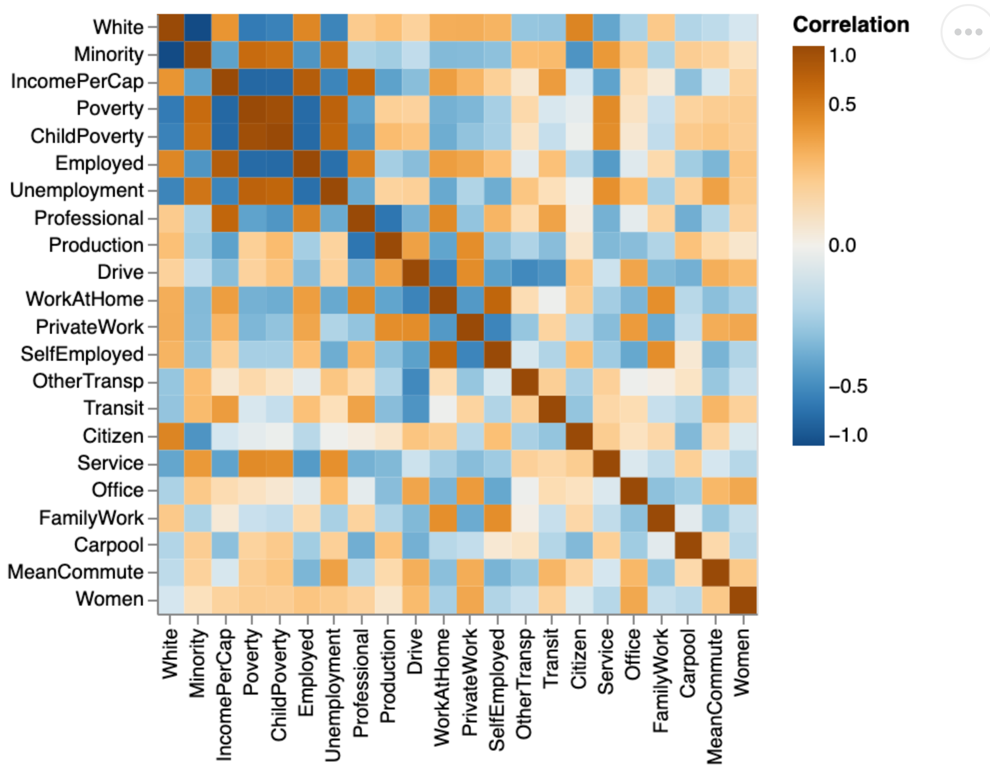
Out[9]:

q1 passed! 🚀

While direct inspection is useful, it can be cumbersome to check correlations for a large number of variables this way. A heatmap -- a colored image of the matrix -- provides a (sometimes) convenient way to see what's going on without having to examine the numerical values directly. The cell below shows one way of constructing this plot. Notice the diverging color scale; this should always be used.

In [10]:
```
# melt corr_mx
corr_mx_long = corr_mx.reset_index().rename(
    columns = {'index': 'row'}
).melt(
    id_vars = 'row',
    var_name = 'col',
    value_name = 'Correlation'
)

# construct plot
alt.Chart(corr_mx_long).mark_rect().encode(
    x = alt.X('col', title = '', sort = {'field': 'Correlation', 'order': 'a
    y = alt.Y('row', title = '', sort = {'field': 'Correlation', 'order': 'a
    color = alt.Color('Correlation',
                      scale = alt.Scale(scheme = 'blueorange', # diverging g
                                        domain = (-1, 1), # ensure white = 0
                                        type = 'sqrt'), # adjust gradient sc
                     legend = alt.Legend(tickCount = 5)) # add ticks to colo
).properties(width = 300, height = 300)
```

Out[10]:



Question 2

Which variable is self employment rate most *positively* correlated with? Refer to the heatmap.

Based on the heatmap, work at home seems to have the most positive correlation

Computing principal components

Each principal component is of the form:

The loading for each component indicate which variables are most influential (heavily weighted) on that principal axis, and thus offer an indirect picture of which variables are driving variation and covariation in the original data.

Loadings and scores

In `statsmodels`, the module `multivariate.pca` contains an easy-to-use implementation.

In [11]:
```
# compute principal components
pca = PCA(data = x_mx, standardize = True)
```

Most quantities you might want to use in PCA can be retrieved as attributes of `pca`. In particular:

- `.loadings` contains the loadings
- `.scores` contains the scores
- `.eigenvals` contains the variances along each principal axis (see lecture notes)

Examine the loadings below. Each column gives the loadings for one principal component; components are ordered from largest to smallest variance.

In [12]:
```
# inspect loadings
pca.loadings
```

Out[12]:

	comp_00	comp_01	comp_02	comp_03	comp_04	comp_05	comp_0
Women	-0.020055	0.139958	0.187600	-0.176614	-0.310716	-0.274826	-0.5375
White	0.289614	0.196549	-0.288902	-0.078059	0.242441	-0.061416	-0.1171
Citizen	0.050698	0.064994	-0.281904	-0.467986	0.404244	-0.025597	-0.1559
IncomePerCap	0.334863	0.020432	0.284074	-0.022197	0.040680	-0.030926	0.0684
Poverty	-0.365212	-0.120172	-0.040170	-0.128231	-0.076355	-0.073253	-0.1369
ChildPoverty	-0.364836	-0.081086	-0.077433	-0.098585	-0.074420	-0.101475	-0.1282
Professional	0.240139	-0.175611	0.287636	-0.258789	-0.094541	-0.004750	0.0826
Service	-0.203254	-0.139714	0.005957	-0.122145	0.377802	0.257543	0.1577
Office	-0.052168	0.189803	0.281398	-0.267195	-0.006059	0.155655	-0.1507
Production	-0.094307	0.282329	-0.285500	0.355106	-0.051805	-0.168182	-0.2313
Drive	-0.102197	0.406130	-0.099229	-0.261077	-0.288984	0.168809	0.2169
Carpool	-0.079129	-0.063744	-0.095696	0.457962	0.110105	-0.235710	0.1828
Transit	0.030233	-0.101142	0.390869	0.052245	0.281641	-0.377725	-0.0194
OtherTransp	-0.021871	-0.209403	0.139315	0.221098	0.318697	0.279204	-0.5497
WorkAtHome	0.218353	-0.331636	-0.116068	-0.113166	-0.067242	-0.168720	-0.0316
MeanCommute	-0.097003	0.176739	0.135322	-0.144408	0.232196	-0.590953	0.2499
Employed	0.345588	0.054653	0.157726	0.128709	-0.115729	0.060930	-0.1021
PrivateWork	0.035539	0.441922	0.158709	0.146725	0.033461	-0.060590	-0.1273
SelfEmployed	0.155300	-0.316174	-0.266798	-0.104453	-0.200415	-0.181008	-0.0262
FamilyWork	0.085077	-0.221137	-0.203301	-0.064817	-0.213611	-0.203811	-0.1496
Unemployment	-0.333420	-0.043047	0.069938	-0.125235	0.125138	-0.132276	-0.1231
Minority	-0.292461	-0.191628	0.282231	0.074901	-0.250838	0.054750	0.1169

22 rows × 22 columns

Similarly, inspect the scores below and check your understanding; each row is an observation and the columns give the scores on each principal axis.

In [13]:

```
# inspect scores
pca.scores
```

Out[13]:

	comp_00	comp_01	comp_02	comp_03	comp_04	comp_05	comp_06	comp
0	0.000504	0.015907	0.008343	-0.006795	-0.007242	0.005351	0.003190	0.001
1	0.005153	0.013795	0.011124	-0.015817	-0.000635	0.004615	-0.001805	0.01
2	-0.029425	0.000688	-0.007837	0.002652	0.002070	-0.000715	0.007475	-0.01
3	-0.011412	0.010430	-0.021061	0.020958	0.015194	-0.009843	0.017917	-0.014
4	-0.004669	0.023879	-0.002247	0.001292	-0.001606	-0.026357	0.006381	0.006
...	...	...	...	...	...	...	...	
3213	0.008225	0.010000	0.007505	0.035786	0.003588	0.002305	0.019690	0.006
3214	0.031668	-0.017726	0.037763	0.014502	0.037688	0.031695	-0.006695	-0.025
3215	0.007884	0.003308	0.003249	0.024815	0.004831	0.005043	0.010563	-0.004
3216	0.008614	-0.009843	-0.003693	0.016797	-0.003189	0.019038	-0.011354	0.006
3217	0.016117	-0.013507	-0.003956	0.015611	0.018331	-0.055831	0.001351	-0.043

3218 rows × 22 columns

Importantly, `statsmodels` rescales the scores so that they have unit inner product; in other words, so that the variances are all ——.

In [14]:

```python
# variance of scores
pca.scores.var()
```

Out[14]:

```
comp_00    0.000311
comp_01    0.000311
comp_02    0.000311
comp_03    0.000311
comp_04    0.000311
comp_05    0.000311
comp_06    0.000311
comp_07    0.000311
comp_08    0.000311
comp_09    0.000311
comp_10    0.000311
comp_11    0.000311
comp_12    0.000311
comp_13    0.000311
comp_14    0.000311
comp_15    0.000311
comp_16    0.000311
comp_17    0.000311
comp_18    0.000311
comp_19    0.000311
comp_20    0.000311
comp_21    0.000311
dtype: float64
```

In [15]:

```python
# for comparison
1/(x_mx.shape[0] - 1)
```

Out[15]: 0.00031084861672365556

To change this behavior, set `normalize = False` when computing the principal components.

Question 3

Check your understanding. Which variable contributes most to the sixth principal component? Store the variable name exactly as it appears among the original column names as `pc6_most_influential_variable`, and store the corresponding loading as `pc6_most_influential_variable_loading`. Print the variable name.

In [16]:
```
# find most influential variable
pc6_most_influential_variable = pca.loadings.iloc[:, 5].abs().idxmax()

# find loading
pc6_most_influential_variable_loading = pca.loadings.loc[pc6_most_influentia

# print
print(pc6_most_influential_variable)
```

```
MeanCommute
```

In [17]:
```
grader.check("q3")
```

Out[17]:

q3 passed! 🙌

Variance ratios

The *variance ratios* indicate the proportions of total variance in the data captured by each principal axis. You may recall from lecture that the variance ratios are computed from the eigenvalues of the correlation (or covariance, if data are not standardized) matrix.

When using `statsmodels`, these need to be computed manually.

In [18]:
```
# compute variance ratios
var_ratios = pca.eigenvals/pca.eigenvals.sum()

# print
var_ratios
```

Out[18]:
```
0     0.262856
1     0.151574
2     0.114128
3     0.076665
4     0.054345
5     0.051541
6     0.047318
7     0.040208
8     0.036687
9     0.033641
10    0.026326
11    0.022018
12    0.017596
13    0.016841
14    0.014282
15    0.010239
16    0.007834
17    0.006307
18    0.004719
19    0.002662
20    0.002101
21    0.000112
Name: eigenvals, dtype: float64
```

Note again that the principal components have been computed in order of *decreasing* variance.

Question 4

Check your understanding. What proportion of variance is captured *jointly* by the first three components taken together? Provide a calculation to justify your answer.

In [19]:
```
print('the proportion is', sum(var_ratios[0:3])*100, "%")
```

```
the proportion is 52.855915886257755 %
```

Selecting a subset of PCs

PCA generally consists of choosing a small subset of components. The basic strategy for selecting this subset is to determine how many are needed to capture some analyst-chosen minimum portion of total variance in the original data.

Most often this assessment is made graphically by inspecting the variance ratios and their cumulative sum, *i.e.*, the amount of total variation captured jointly by subsets of successive components. We'll store these quantities in a data frame.

In [20]:
```
# store proportion of variance explained as a dataframe
pca_var_explained = pd.DataFrame({
    'Component': np.arange(1, 23),
    'Proportion of variance explained': var_ratios})
```

```
# add cumulative sum
pca_var_explained['Cumulative variance explained'] = var_ratios.cumsum()

# print
pca_var_explained.head()
```

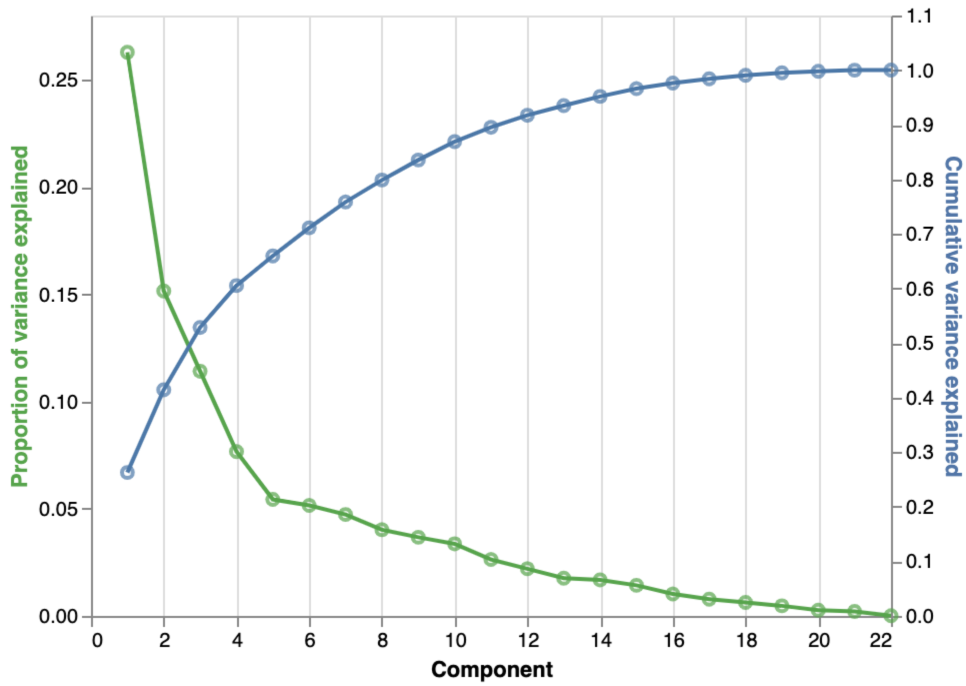
Out[20]:

	Component	Proportion of variance explained	Cumulative variance explained
0	1	0.262856	0.262856
1	2	0.151574	0.414431
2	3	0.114128	0.528559
3	4	0.076665	0.605224
4	5	0.054345	0.659569

Now we'll make a dual-axis plot showing, on one side, the proportion of variance explained (y) as a function of component (x), and on the other side, the cumulative variance explained (y) also as a function of component (x). Make sure that you've completed Q1(a) before running the next cell.

In [21]:
```
# encode component axis only as base layer
base = alt.Chart(pca_var_explained).encode(
    x = 'Component')

# make a base layer for the proportion of variance explained
prop_var_base = base.encode(
    y = alt.Y('Proportion of variance explained',
              axis = alt.Axis(titleColor = '#57A44C'))
)

# make a base layer for the cumulative variance explained
cum_var_base = base.encode(
    y = alt.Y('Cumulative variance explained', axis = alt.Axis(titleColor =
)

# add points and lines to each base layer
prop_var = prop_var_base.mark_line(stroke = '#57A44C') + prop_var_base.mark_
cum_var = cum_var_base.mark_line() + cum_var_base.mark_point()

# layer the layers
var_explained_plot = alt.layer(prop_var, cum_var).resolve_scale(y = 'indeper

# display
var_explained_plot
```

Out[21]:



The purpose of making this plot is to quickly determine the fewest number of principal components that capture a considerable portion of variation and covariation. 'Considerable' here is a bit subjective.

Question 5

How many principal components explain more than 6% of total variation individually? Store this number as `num_pc`, and store the proportion of variation that they capture jointly as `var_explained`.

In [22]:

```
# number of selected components
num_pc = 4

# variance explained
var_explained =  sum(var_ratios[0:num_pc])

#print
print('number selected: ', num_pc)
print('proportion of variance captured: ', var_explained)
```

```
number selected:  4
proportion of variance captured:  0.6052243442775457
```

In [23]:

```
grader.check("q5")
```

Out[23]:

q5 passed! ✨

Interpreting loadings

Now that you've chosen the number of components to work with, the next step is to examine loadings to understand just *which* variables the components combine with significant weight.

We'll store the scores for the components you selected as a dataframe.

In [24]:
```
# subset loadings
loading_df = pca.loadings.iloc[:, 0:num_pc]

# rename columns
loading_df = loading_df.rename(columns = dict(zip(loading_df.columns, ['PC'

# print
loading_df.head()
```

Out[24]:

	PC1	PC2	PC3	PC4
Women	-0.020055	0.139958	0.187600	-0.176614
White	0.289614	0.196549	-0.288902	-0.078059
Citizen	0.050698	0.064994	-0.281904	-0.467986
IncomePerCap	0.334863	0.020432	0.284074	-0.022197
Poverty	-0.365212	-0.120172	-0.040170	-0.128231

Again, the loadings are the *weights* with which the variables are combined to form the principal components. For example, the `PC1` column tells us that this component is equal to:

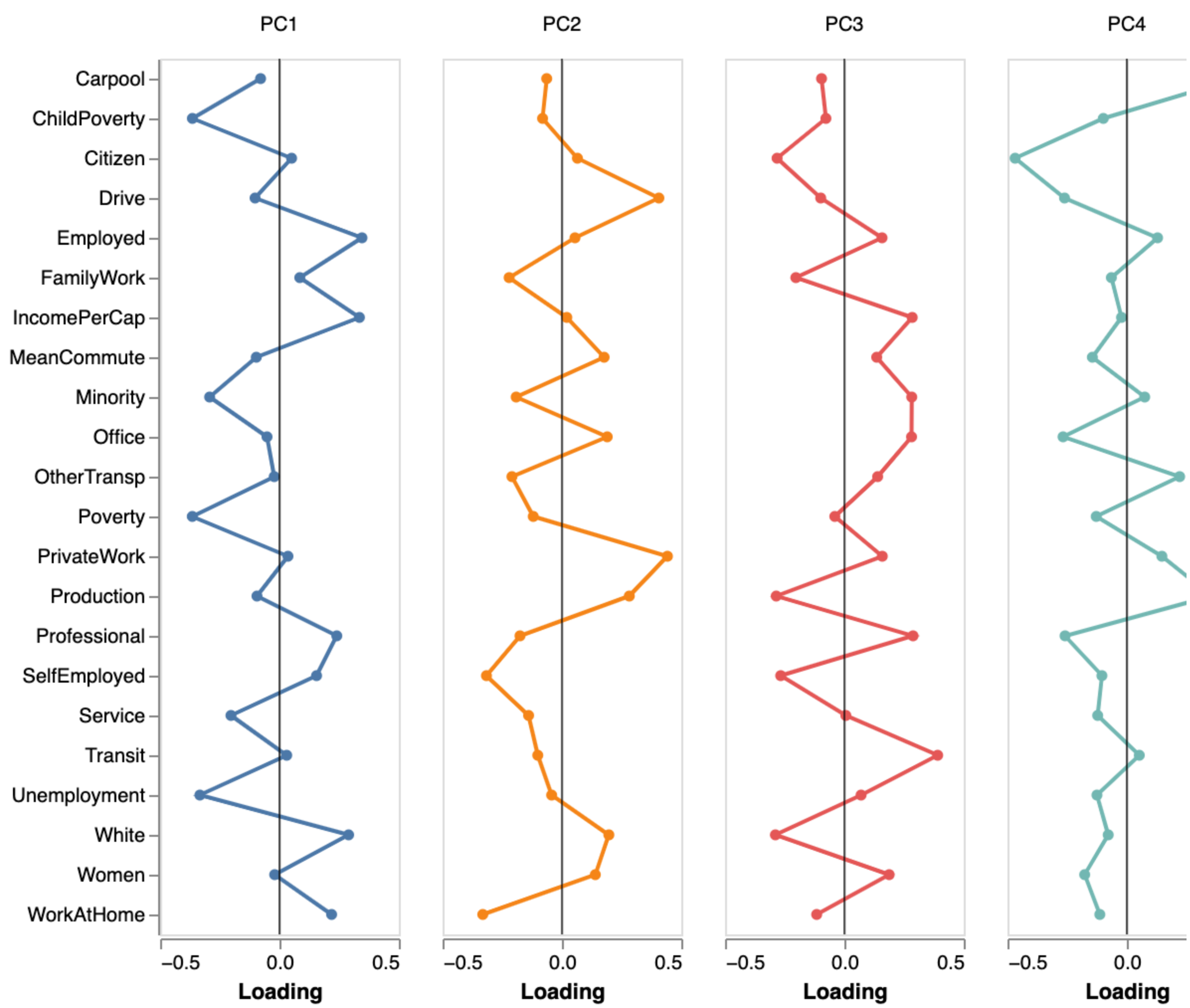
Since the components together capture over half the total variation, the heavily weighted variables in the selected components are the ones that drive variation in the original data.

By visualizing the loadings, we can see which variables are most influential for each component, and thereby also which variables seem to drive total variation in the data.

In [25]:
```
# melt from wide to long
loading_plot_df = loading_df.reset_index().melt(
    id_vars = 'index',
    var_name = 'Principal Component',
    value_name = 'Loading'
).rename(columns = {'index': 'Variable'})

# add a column of zeros to encode for x = 0 line to plot
loading_plot_df['zero'] = np.repeat(0, len(loading_plot_df))

# create base layer
base = alt.Chart(loading_plot_df)
```

```
# create lines + points for loadings
loadings = base.mark_line(point = True).encode(
    y = alt.X('Variable', title = ''),
    x = 'Loading',
    color = 'Principal Component'
)

# create line at zero
rule = base.mark_rule().encode(x = alt.X('zero', title = 'Loading'), size =

# layer
loading_plot = (loadings + rule).properties(width = 120)

# show
loading_plot.facet(column = alt.Column('Principal Component', title = ''))
```

Out[25]:



Look first at PC1: the variables with the largest loadings (points farthest in either direction from the zero line) are Child Poverty (positive), Employed (negative), Income per capita (negative), Poverty (positive), and Unemployment (positive). We know from exploring the correlation matrix that employment rate, unemployment rate, and income per capita are all related, and similarly child poverty rate and poverty rate are related. Therefore, the positively-loaded variables are all measuring more or less the same thing, and likewise for the negatively-loaded variables.

Essentially, then, PC1 is predominantly (but not entirely) a representation of income and poverty. In particular, counties have a higher value for PC1 if they have lower-than-average income per capita and higher-than-average poverty rates, and a smaller value for PC1 if they have higher-than-average income per capita and lower-than-average poverty rates.

A system for loading interpretation

Often interpreting principal components can be difficult, and sometimes there's no clear interpretation available! That said, it helps to have a system instead of staring at the plot and scratching our heads. Here is a semi-systematic approach to interpreting loadings:

1. Divert your attention away from the zero line.
2. Find the largest positive loading, and list all variables with similar loadings.
3. Find the largest negative loading, and list all variables with similar loadings.
4. The principal component represents the difference between the average of the first set and the average of the second set.
5. Try to come up with a description of less than 4 words.

This system is based on the following ideas:

- a high loading value (negative or positive) indicates that a variable strongly influences the principal component;
- a negative loading value indicates that
  - increases in the value of a variable *decrease* the value of the principal component
  - and decreases in the value of a variable *increase* the value of the principal component;
- a positive loading value indicates that
  - increases in the value of a variable *increase* the value of the principal component
  - and decreases in the value of a variable *decrease* the value of the principal component;
- similar loadings between two or more variables indicate that the principal component reflects their *average*;
- divergent loadings between two sets of variables indicates that the principal component reflects their *difference*.

Question 6

Work with your neighbor to interpret PC2. Come up with a name for the component and explain which variables are most influential.

PC2 measures the highest emplyment type

Standardization

Data are typically standardized because otherwise the variables on the largest scales tend to dominate the principal components, and most of the time PC1 will capture the majority of the variation. However, that is artificial. In the census data, income per capita has the largest magnitudes, and thus, the highest variance.

In [26]:
```
# three largest variances
x_mx.var().sort_values(ascending = False).head(3)
```

Out[26]:
```
IncomePerCap    3.804072e+07
Minority        5.265263e+02
White           5.264985e+02
dtype: float64
```

When PCs are computed without normalization, the total variation is mostly just the variance of income per capita because it is orders of magnitude larger than the variance of any other variable. But that's just because of the *scale* of the variable -- incomes per capita are large numbers -- not a reflection that it varies more or less than the other variables.

Run the cell below to see what happens to the variance ratios if the data are not normalized.

In [27]:
```
# recompute pcs without normalization
pca_unscaled = PCA(data = x_mx, standardize = False)

# show variance ratios for first three pcs
pca_unscaled.eigenvals[0:3]/pca_unscaled.eigenvals.sum()
```

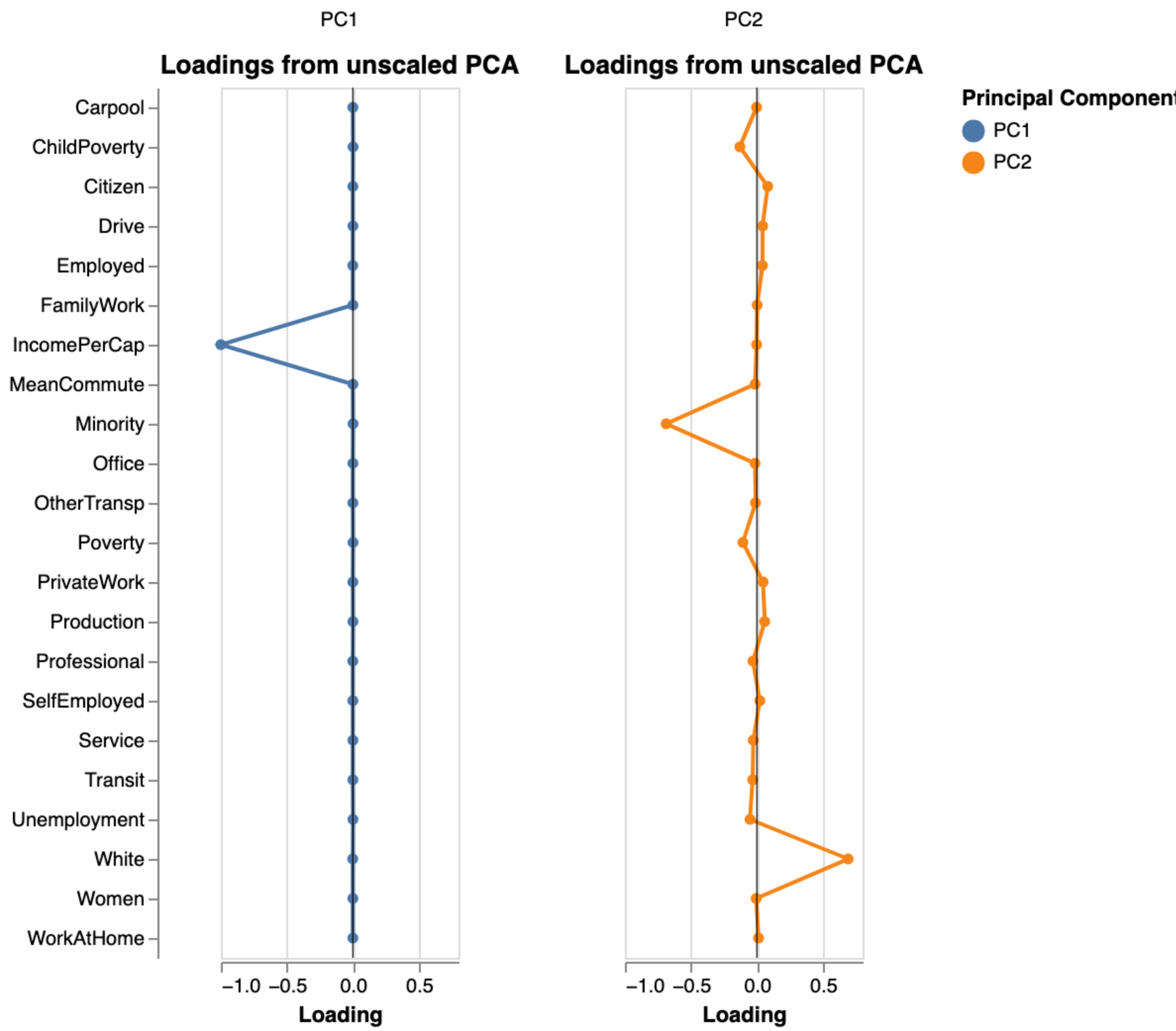
Out[27]:
```
0    0.999965
1    0.000025
2    0.000003
Name: eigenvals, dtype: float64
```

Further, let's look at the loadings when data are not standardized:

In [28]:
```
# subset loadings
unscaled_loading_df = pca_unscaled.loadings.iloc[:, 0:2]

# rename columns
unscaled_loading_df = unscaled_loading_df.rename(
    columns = dict(zip(unscaled_loading_df.columns, ['PC' + str(i) for i in
)

# melt from wide to long
unscaled_loading_plot_df = unscaled_loading_df.reset_index().melt(
    id_vars = 'index',
```

```
    var_name = 'Principal Component',
    value_name = 'Loading'
).rename(
    columns = {'index': 'Variable'}
)

# add a column of zeros to encode for x = 0 line to plot
unscaled_loading_plot_df['zero'] = np.repeat(0, len(unscaled_loading_plot_df

# create base layer
base = alt.Chart(unscaled_loading_plot_df)

# create lines + points for loadings
loadings = base.mark_line(point = True).encode(
    y = alt.X('Variable', title = ''),
    x = 'Loading',
    color = 'Principal Component'
)

# create line at zero
rule = base.mark_rule().encode(x = alt.X('zero', title = 'Loading'), size = 

# layer
loading_plot = (loadings + rule).properties(width = 120, title = 'Loadings f

# show
loading_plot.facet(column = alt.Column('Principal Component', title = ''))
```

Out[28]:



Notice that the variables with nonzero loadings in unscaled PCA are simply the three variables with the largest variances.

In [29]:

```
# three largest variances
x_mx.var().sort_values(ascending = False).head(3)
```

Out[29]:

```
IncomePerCap    3.804072e+07
Minority        5.265263e+02
White           5.264985e+02
dtype: float64
```

Exploratory analysis based on PCA

Now that we have the principal components, we can use them for exploratory data visualizations. To this end, let's retrieve the scores from the components you selected:

In [30]:

```
# subset scores
score_df = pca.scores.iloc[:, 0:num_pc]

# rename columns
score_df = score_df.rename(
```

```
    columns = dict(zip(score_df.columns, ['PC' + str(i) for i in range(1, nu
)

# add state and county
score_df[['State', 'County']] = census[['State', 'County']]

# print
score_df.head()
```

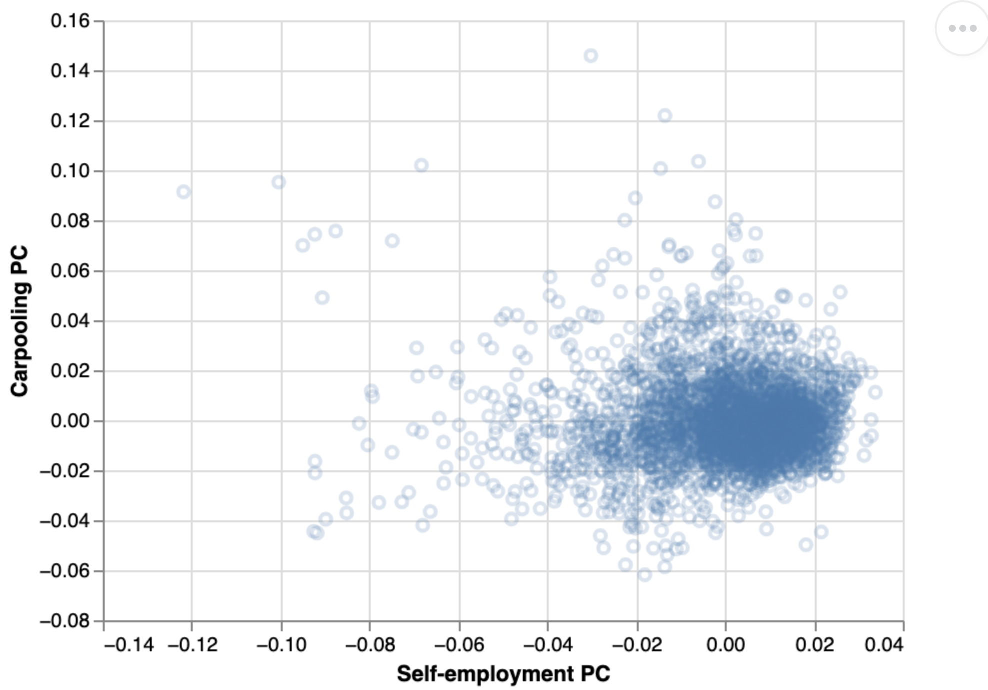
Out[30]:

	PC1	PC2	PC3	PC4	State	County
0	0.000504	0.015907	0.008343	-0.006795	Alabama	Autauga
1	0.005153	0.013795	0.011124	-0.015817	Alabama	Baldwin
2	-0.029425	0.000688	-0.007837	0.002652	Alabama	Barbour
3	-0.011412	0.010430	-0.021061	0.020958	Alabama	Bibb
4	-0.004669	0.023879	-0.002247	0.001292	Alabama	Blount

The PC's can be used to construct scatterplots of the data and search for patterns. We'll illustrate that by identifying some outliers. The cell below plots PC2 (employment type) against PC4 (carpooling?):

In [31]:
```
# base chart
base = alt.Chart(score_df)

# data scatter
scatter = base.mark_point(opacity = 0.2).encode(
    x = alt.X('PC2:Q', title = 'Self-employment PC'),
    y = alt.Y('PC4:Q', title = 'Carpooling PC')
)

# show
scatter
```

Out[31]:



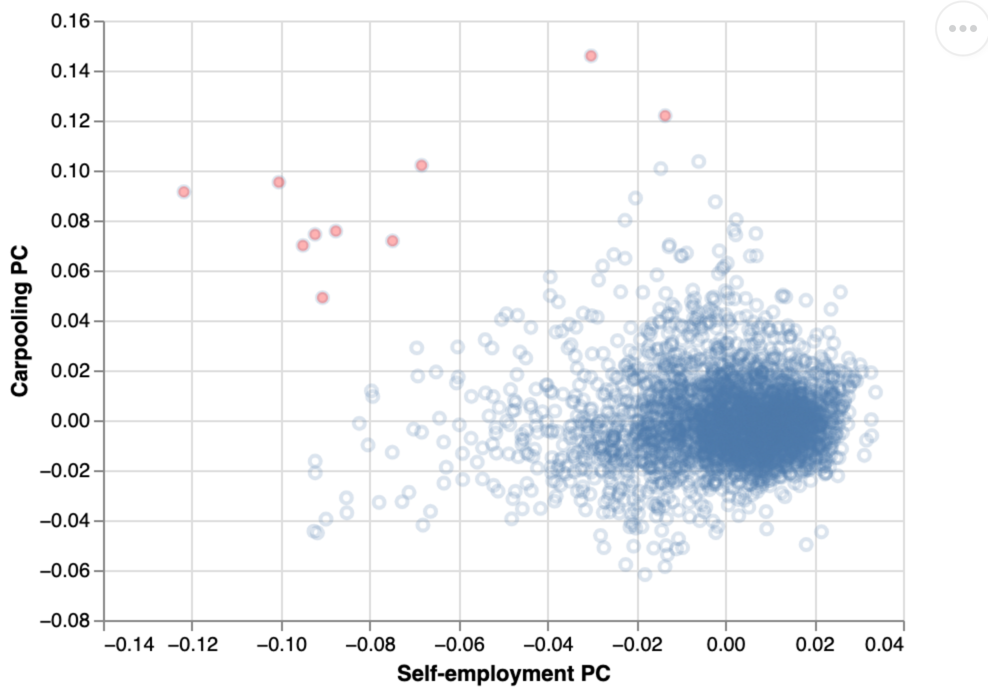
Notice that there are a handful of outlying points in the upper right region away from the dense scatter. What are those?

In order to inspect the outlying counties, we first need to figure out how to identify them. The outlying values have a large *sum* of PC2 and PC4. We can distinguish them by finding a cutoff value for the sum; a simple quantile will do.

In [32]:

```
# find cutoff value
pc2_pc4_sum = (score_df.PC2 + score_df.PC4)
cutoff = pc2_pc4_sum.quantile(0.99999)

# store outlying rows using cutoff
outliers = score_df[(-score_df.PC2 + score_df.PC4) > cutoff]

# plot outliers in red
pts = alt.Chart(outliers).mark_circle(
    color = 'red',
    opacity = 0.3
).encode(
    x = 'PC2',
    y = 'PC4'
)

# layer
scatter + pts
```

Out[32]:



Notice that almost all the outlying counties are remote regions of Alaska:

In [33]:

```
outliers
```

Out[33]:

	PC1	PC2	PC3	PC4	State	County
67	0.009880	-0.030176	-0.013891	0.145835	Alaska	Aleutians East Borough
70	-0.024979	-0.087602	0.048901	0.075695	Alaska	Bethel Census Area
73	-0.011094	-0.074851	0.041229	0.071774	Alaska	Dillingham Census Area
81	-0.042726	-0.121797	0.060025	0.091386	Alaska	Kusilvak Census Area
82	-0.006917	-0.100417	0.042412	0.095258	Alaska	Lake and Peninsula Borough
84	-0.021137	-0.095011	0.041992	0.069952	Alaska	Nome Census Area
85	-0.017581	-0.068302	0.033698	0.102002	Alaska	North Slope Borough
86	-0.022937	-0.092293	0.046340	0.074361	Alaska	Northwest Arctic Borough
95	-0.016652	-0.090613	0.027860	0.049054	Alaska	Yukon-Koyukuk Census Area
739	0.002107	-0.013507	-0.023003	0.121893	Indiana	LaGrange

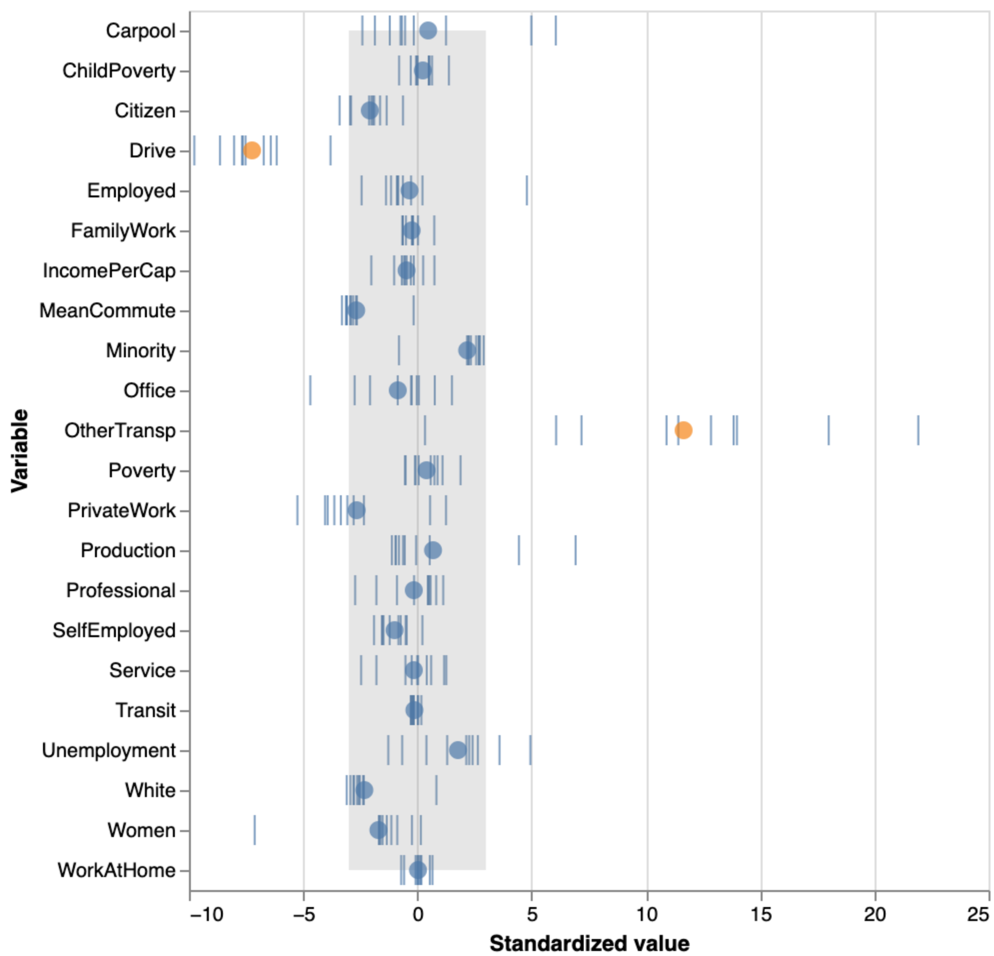
What sets them apart? The cell below retrieves the normalized data and county name for the outlying rows, and then plots the Standardized values of each variable for all 9 counties as vertical ticks, along with a point indicating the mean for the outlying counties. This plot can be used to determine which variables are over- or under-average for the outlying counties relative to the nation by simply locating means that are far from zero in either direction.

In [34]:

```
x_ctr = (x_mx - x_mx.mean())/x_mx.std()
```

```
# retrieve normalized data for outlying rows
outlier_data = x_ctr.loc[outliers.index.values].join(
    census.loc[outliers.index.values, ['County']]
)

# melt to long format for plotting
outlier_plot_df = outlier_data.melt(
    id_vars = 'County',
    var_name = 'Variable',
    value_name = 'Standardized value'
)

# plot ticks for values (x) for each variable (y)
ticks = alt.Chart(outlier_plot_df).mark_tick().encode(
    x = 'Standardized value',
    y = 'Variable'
)

# shade out region within 3SD of mean
grey = alt.Chart(
    pd.DataFrame(
        {'Variable': x_ctr.columns,
         'upr': np.repeat(3, 22),
         'lwr': np.repeat(-3, 22)}
    )
).mark_area(opacity = 0.2, color = 'gray').encode(
    y = 'Variable',
    x = alt.X('upr', title = 'Standardized value'),
    x2 = 'lwr'
)

# compute means of each variable across counties
means = alt.Chart(outlier_plot_df).transform_aggregate(
    group_mean = 'mean(Standardized value)',
    groupby = ['Variable']
).transform_calculate(
    large = 'abs(datum.group_mean) > 3'
).mark_circle(size = 80).encode(
    x = 'group_mean:Q',
    y = 'Variable',
    color = alt.Color('large:N', legend = None)
)

# layer
ticks + grey + means
```

Out[34]:



Question 7

The two variables that clearly set the outlying counties apart from the nation are the percentage of the population using alternative transportation (extremely above average) and the percentage that drive to work (extremely below average). What about those counties explains this?

(*Hint:* take a peek at the Wikipedia page on transportation in Alaska.)

Other transportation and drive are the otwo that are outliers. The reasion this is so off is probably due to Alaska's weather. The colder climate means it is harder to drive and more people have to use other transportations.

Submission

1. Save the notebook.
2. Restart the kernel and run all cells. (**CAUTION**: if your notebook is not saved, you will lose your work.)
3. Carefully look through your notebook and verify that all computations execute correctly and all graphics are displayed clearly. You should see **no errors**; if there

are any errors, make sure to correct them before you submit the notebook.

4. Download the notebook as an `.ipynb` file. This is your backup copy.
5. Export the notebook as PDF and upload to Gradescope.

To double-check your work, the cell below will rerun all of the autograder tests.

In [35]:
```
grader.check_all()
```

Out[35]:
```
q1 results: All test cases passed!

q3 results: All test cases passed!

q5 results: All test cases passed!
```