1. The raw data files and their format

While the rate of fatal road accidents has been decreasing steadily since the 80s, the past ten years have seen a stagnation in this reduction. Coupled with the increase in number of miles driven in the nation, the total number of traffic related-fatalities has now reached a ten year high and is rapidly increasing.

Per request of the US Department of Transportation, we are currently investigating how to derive a strategy to reduce the incidence of road accidents across the nation. By looking at the demographics of traffic accident victims for each US state, we find that there is a lot of variation between states. Now we want to understand if there are patterns in this variation in order to derive suggestions for a policy action plan. In particular, instead of implementing a costly nation-wide plan we want to focus on groups of states with similar profiles. How can we find such groups in a statistically sound way and communicate the result effectively?

To accomplish these tasks, we will make use of data wrangling, plotting, dimensionality reduction, and unsupervised clustering.

The data given to us was originally collected by the National Highway Traffic Safety Administration and the National Association of Insurance Commissioners. This particular dataset was compiled and released as a CSV-file (https://github.com/fivethirtyeight/data/tree/master/bad-drivers (https://github.com/fivethirtyeight/data/tree/master/bad-drivers)) by FiveThirtyEight under the CC-BY4.0 license (https://github.com/fivethirtyeight/data (https://github.com/fivethirtyeight/data)).

In [1]:
```
import os

current_dir = os.getcwd()
print("Present Working Directory:\n", current_dir, sep="")
```

```
Present Working Directory:
C:\myProject\WinPY\WPy64-3830\notebooks
```

In [2]:
```
# Change the location of current directory to the required locations where datasets resides
os.chdir(r"D:/NYIT (Fall2020-ACADEMICS)/INTRODUCTION TO DATA MINING (CSCI657 or CSCI415) [PROFF Helen Gu]/Final
Projects/PROJECT1 PYTHON/datasets")
current_dir = os.getcwd()
print("Changed Directory:\n", current_dir, sep="")

# List all files in the current directory
ls = os.listdir()
print("\nList Of Files:")
for element in range(len(ls)):
    print(ls[element], sep="")

# View the first 20 lines of datasets/miles-driven.csv
with open("miles-driven.csv") as myfirst_dataset:
    head = [next(myfirst_dataset) for x in range(20)]
print("\nFirst 20 Lines of Dataset miles-driven.csv:\n", str(head).strip("[]"), sep="")

# View the first 20 lines of datasets/road-accidents.csv
with open("road-accidents.csv") as mysecond_dataset:
    head = [next(mysecond_dataset) for x in range(20)]
print("\nFirst 20 Lines of Dataset road-accidents.csv:\n", str(head).strip("[]"), sep="")
```

```
Changed Directory:
D:\NYIT (Fall2020-ACADEMICS)\INTRODUCTION TO DATA MINING (CSCI657 or CSCI415) [PROFF Helen Gu]\Final Projects
\PROJECT1 PYTHON\datasets

List Of Files:
DataSet Link.txt
miles-driven.csv
road-accidents.csv

First 20 Lines of Dataset miles-driven.csv:
'state|million_miles_annually\n', 'Alabama|64914\n', 'Alaska|4593\n', 'Arizona|59575\n', 'Arkansas|32953\n',
'California|320784\n', 'Colorado|46606\n', 'Connecticut|31197\n', 'Delaware|9028\n', 'District of Columbia|356
8\n', 'Florida|191855\n', 'Georgia|108454\n', 'Hawaii|10066\n', 'Idaho|15937\n', 'Illinois|103234\n', 'Indiana
|76485\n', 'Iowa|31274\n', 'Kansas|30021\n', 'Kentucky|48061\n', 'Louisiana|46513\n'

First 20 Lines of Dataset road-accidents.csv:
'##### LICENSE #####\n', '# This data set is modified from the original at fivethirtyeight (https://github.co
m/fivethirtyeight/data/tree/master/bad-drivers)\n', '# and it is released under CC BY 4.0 (https://creativecom
mons.org/licenses/by/4.0/)\n', '##### COLUMN ABBREVIATIONS #####\n', '# drvr_fatl_col_bmiles = Number of drive
rs involved in fatal collisions per billion miles (2011)\n', '# perc_fatl_speed = Percentage Of Drivers Involv
ed In Fatal Collisions Who Were Speeding (2009)\n', '# perc_fatl_alcohol = Percentage Of Drivers Involved In F
atal Collisions Who Were Alcohol-Impaired (2011)\n', '# perc_fatl_1st_time = Percentage Of Drivers Involved In
Fatal Collisions Who Had Not Been Involved In Any Previous Accidents (2011)\n', '##### DATA BEGIN #####\n', 's
tate|drvr_fatl_col_bmiles|perc_fatl_speed|perc_fatl_alcohol|perc_fatl_1st_time\n', 'Alabama|18.8|39|30|80\n',
'Alaska|18.1|41|25|94\n', 'Arizona|18.6|35|28|96\n', 'Arkansas|22.4|18|26|95\n', 'California|12|35|28|89\n',
'Colorado|13.6|37|28|95\n', 'Connecticut|10.8|46|36|82\n', 'Delaware|16.2|38|30|99\n', 'District of Columbia|
5.9|34|27|100\n', 'Florida|17.9|21|29|94\n'
```

1. Read in and get an overview of the data

Next, we will orient ourselves to get to know the data with which we are dealing.

In [3]:
```
# Import the `pandas` module as "pd"
import pandas as pd

# Read in `road-accidents.csv`
car_acc = pd.read_csv("D:/NYIT (Fall2020-ACADEMICS)/INTRODUCTION TO DATA MINING (CSCI657 or CSCI415) [PROFF Hel
en Gu]/Final Projects/PROJECT1 PYTHON/datasets/road-accidents.csv", comment='#', sep='|')

# Save the number of rows columns as a tuple
rows_and_cols = car_acc.shape
print("Brief Description Of Dataset road-accidents.csv Stored In DataFrame Object car_acc:\n")
print('There are {} rows and {} columns.\n'.format(rows_and_cols[0], rows_and_cols[1]))

# Generate an overview of the DataFrame
car_acc_information = car_acc.info()
print(car_acc_information)

# Check whether there are Nan values or not
print("\nBrief Description Whether DataFrame Object car_acc Consists Any NaN Values:\n", car_acc.isna().sum(),
sep="")

# Display the first five rows of the DataFrame
print("\nFollowing Is The Generic Overview Of DataFrame Object car_acc:\n")
car_acc.head()
```

```
Brief Description Of Dataset road-accidents.csv Stored In DataFrame Object car_acc:

There are 51 rows and 5 columns.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   state                 51 non-null     object
 1   drvr_fatl_col_bmiles  51 non-null     float64
 2   perc_fatl_speed       51 non-null     int64
 3   perc_fatl_alcohol     51 non-null     int64
 4   perc_fatl_1st_time    51 non-null     int64
dtypes: float64(1), int64(3), object(1)
memory usage: 2.1+ KB
None

Brief Description Whether DataFrame Object car_acc Consists Any NaN Values:
state                   0
drvr_fatl_col_bmiles    0
perc_fatl_speed         0
perc_fatl_alcohol       0
perc_fatl_1st_time      0
dtype: int64

Following Is The Generic Overview Of DataFrame Object car_acc:
```

Out[3]:

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
0	Alabama	18.8	39	30	80
1	Alaska	18.1	41	25	94
2	Arizona	18.6	35	28	96
3	Arkansas	22.4	18	26	95
4	California	12.0	35	28	89

1. Create a textual and a graphical summary of the data

We now have an idea of what the dataset looks like. To further familiarize ourselves with this data, we will calculate summary statistics and produce a graphical overview of the data. The graphical overview is good to get a sense for the distribution of variables within the data and could consist of one histogram per column. It is often a good idea to also explore the pairwise relationship between all columns in the data set by using a using pairwise scatter plots (sometimes referred to as a "scatterplot matrix").

In [4]:
```
# Compute the summary statistics of all columns in the `car_acc` DataFrame
sum_stat_car = car_acc.describe()
print("Summary Statistics Of DataFrame Object car_acc:")
sum_stat_car
```

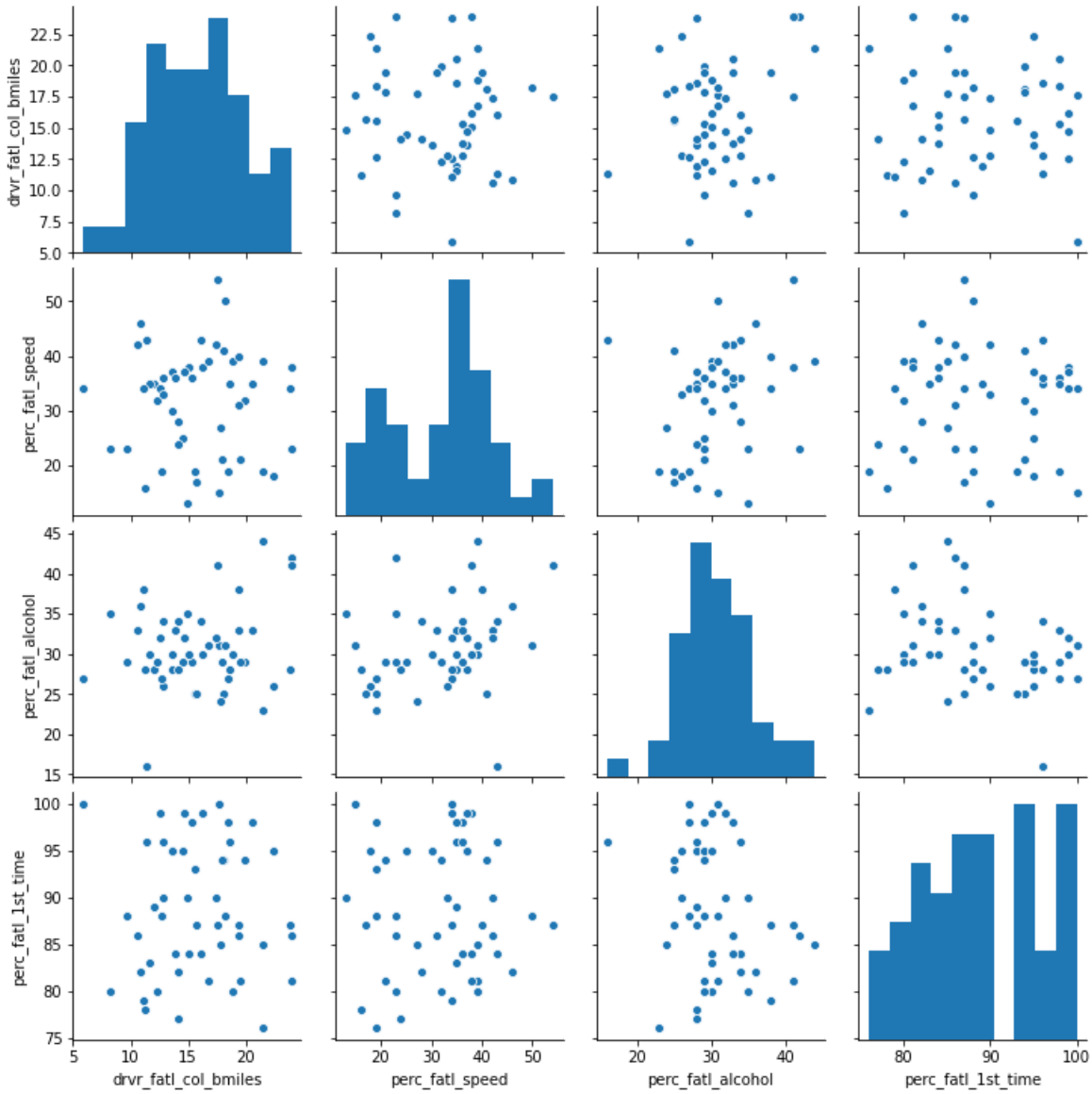
```
Summary Statistics Of DataFrame Object car_acc:
```

Out[4]:

	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
count	51.000000	51.000000	51.000000	51.00000
mean	15.790196	31.725490	30.686275	88.72549
std	4.122002	9.633438	5.132213	6.96011
min	5.900000	13.000000	16.000000	76.00000
25%	12.750000	23.000000	28.000000	83.50000
50%	15.600000	34.000000	30.000000	88.00000
75%	18.500000	38.000000	33.000000	95.00000
max	23.900000	54.000000	44.000000	100.00000

In [5]:
```
# import seaborn and make plots appear inline
import seaborn as sns
%matplotlib inline

# Create a pairwise scatter plot to explore the data
print("Data Visualization And Exploration:\n\n", sns.pairplot(car_acc), sep="")
```

```
Data Visualization And Exploration:

<seaborn.axisgrid.PairGrid object at 0x000002F78960CFD0>
```



1. Quantify the association of features and accidents

We can already see some potentially interesting relationships between the target variable (the number of fatal accidents) and the feature variables (the remaining three columns).

To quantify the pairwise relationships that we observed in the scatter plots, we can compute the Pearson correlation coefficient matrix. The Pearson correlation coefficient is one of the most common methods to quantify correlation between variables, and by convention, the following thresholds are usually used:

0.2 = weak

0.5 = medium

0.8 = strong

0.9 = very strong

In [6]:
```
# Compute the correlation coefficent for all column pairs
print("Pearson Correlation Coefficient Matrix:\n")
corr_columns = car_acc.corr()
corr_columns
```

```
Pearson Correlation Coefficient Matrix:
```

Out[6]:

	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
drvr_fatl_col_bmiles	1.000000	-0.029080	0.199426	-0.017942
perc_fatl_speed	-0.029080	1.000000	0.286244	0.014066
perc_fatl_alcohol	0.199426	0.286244	1.000000	-0.245455
perc_fatl_1st_time	-0.017942	0.014066	-0.245455	1.000000

1. Fit a multivariate linear regression

From the correlation table, we see that the amount of fatal accidents is most strongly correlated with alcohol consumption (first row). But in addition, we also see that some of the features are correlated with each other, for instance, speeding and alcohol consumption are positively correlated. We, therefore, want to compute the association of the target with each feature while adjusting for the effect of the remaining features. This can be done using multivariate linear regression.

Both the multivariate regression and the correlation measure how strongly the features are associated with the outcome (fatal accidents). When comparing the regression coefficients with the correlation coefficients, we will see that they are slightly different. The reason for this is that the multiple regression computes the association of a feature with an outcome, given the association with all other features, which is not accounted for when calculating the correlation coefficients.

A particularly interesting case is when the correlation coefficient and the regression coefficient of the same feature have opposite signs. How can this be? For example, when a feature A is positively correlated with the outcome Y but also positively correlated with a different feature B that has a negative effect on Y, then the indirect correlation (A->B->Y) can overwhelm the direct correlation (A->Y). In such a case, the regression coefficient of feature A could be positive, while the correlation coefficient is negative. This is sometimes called a masking relationship. Let's see if the multivariate regression can reveal such a phenomenon.

In [7]:
```
# Import the linear model function from sklearn
import numpy as np
from sklearn import linear_model

# Create the features and target DataFrames
features = car_acc[['perc_fatl_speed', 'perc_fatl_alcohol', 'perc_fatl_1st_time']]
target = car_acc['drvr_fatl_col_bmiles']

# Create a linear regression object
reg = linear_model.LinearRegression()
print(type(reg))

# Fit a multivariate linear regression model
reg.fit(features, target)

# Retrieve the regression coefficients
fit_coef = reg.coef_
print(type(fit_coef))

fit_coef
```

```
<class 'sklearn.linear_model._base.LinearRegression'>
<class 'numpy.ndarray'>
```

Out[7]:
```
array([-0.04180041,  0.19086404,  0.02473301])
```

1. Perform PCA on standardized data

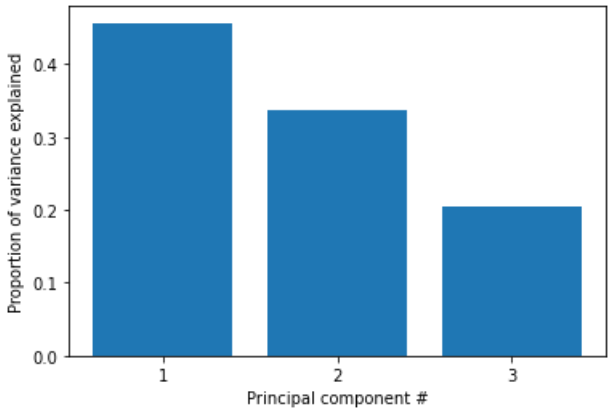
We have learned that alcohol consumption is weakly associated with the number of fatal accidents across states. This could lead us to conclude that alcohol consumption should be a focus for further investigations and maybe strategies should divide states into high versus low alcohol consumption in accidents. But there are also associations between alcohol consumptions and the other two features, so it might be worth trying to split the states in a way that accounts for all three features.

One way of clustering the data is to use PCA to visualize data in reduced dimensional space where we can try to pick up patterns by eye. PCA uses the absolute variance to calculate the overall variance explained for each principal component, so it is important that the features are on a similar scale (unless we would have a particular reason that one feature should be weighted more).

We'll use the appropriate scaling function to standardize the features to be centered with mean 0 and scaled with standard deviation 1.

In [8]:
```
# Standardize and center the feature columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Import the PCA class from sklearn
from sklearn.decomposition import PCA
pca = PCA()

# Fit the standardized data to the pca
pca.fit(features_scaled)

# Plot the proportion of variance explained on the y-axis of the bar plot
import matplotlib.pyplot as plt
plt.bar(range(1, pca.n_components_ + 1),  pca.explained_variance_ratio_)
plt.xlabel('Principal component #')
plt.ylabel('Proportion of variance explained')
plt.xticks([1, 2, 3])

# Compute the cumulative proportion of variance explained by the first two principal components
two_first_comp_var_exp = pca.explained_variance_ratio_.cumsum()[1]
print("The Cumulative Variance Of The First Two Principal Componenets Is {}".format(round(two_first_comp_var_ex
p, 5)))
```

```
The Cumulative Variance Of The First Two Principal Componenets Is 0.7947
```



1. Visualize the first two principal components

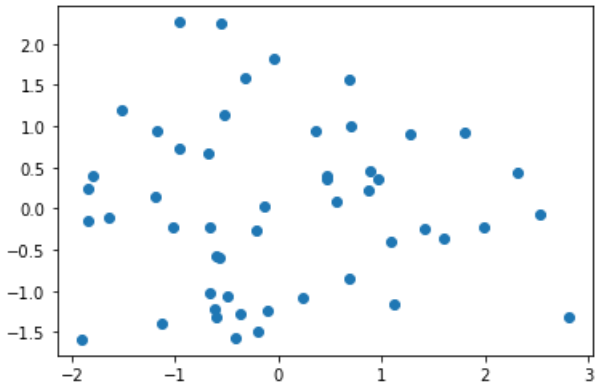
The first two principal components enable visualization of the data in two dimensions while capturing a high proportion of the variation (79%) from all three features: speeding, alcohol influence, and first-time accidents. This enables us to use our eyes to try to discern patterns in the data with the goal to find groups of similar states. Although clustering algorithms are becoming increasingly efficient, human pattern recognition is an easily accessible and very efficient method of assessing patterns in data.

We will create a scatter plot of the first principle components and explore how the states cluster together in this visualization.

In [9]:
```
# Transform the scaled features using two principal components
pca = PCA(n_components=2)
p_comps = pca.fit_transform(features_scaled)

# Extract the first and second component to use for the scatter plot
p_comp1 = p_comps[:, 0]
p_comp2 = p_comps[:, 1]

# Plot the first two principal components in a scatter plot
print("Scatter Plot Of First Two Principal Component:\n")
plt.scatter(p_comp1, p_comp2)
```

```
Scatter Plot Of First Two Principal Component:
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x2f78e815850>
```



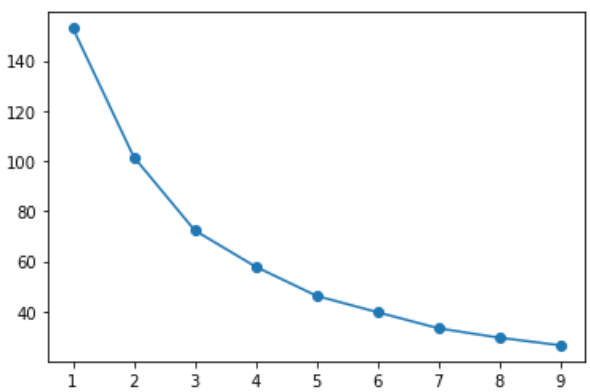
1. Find clusters of similar states in the data

It was not entirely clear from the PCA scatter plot how many groups in which the states cluster. To assist with identifying a reasonable number of clusters, we can use KMeans clustering by creating a scree plot and finding the "elbow", which is an indication of when the addition of more clusters does not add much explanatory power.

In [10]:
```
# Import KMeans from sklearn
from sklearn.cluster import KMeans

# A loop will be used to plot the explantory power for up to 10 KMeans clusters
ks = range(1, 10)
inertias = []
for k in ks:
    # Initialize the KMeans object using the current number of clusers (k)
    km = KMeans(n_clusters=k, random_state=8)
    # Fit the scaled scaled features to the KMeans object
    km.fit(features_scaled)
    # Append the inertia for `km` to the list of inertias
    inertias.append(km.inertia_)

# Plot the results in a line plot
print("Scree Plot Or Line Plot Using KMeans Along With Principal Components:\n")
plt.plot(ks, inertias, marker='o')
```

```
Scree Plot Or Line Plot Using KMeans Along With Principal Components:
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x2f78eef9e80>]
```



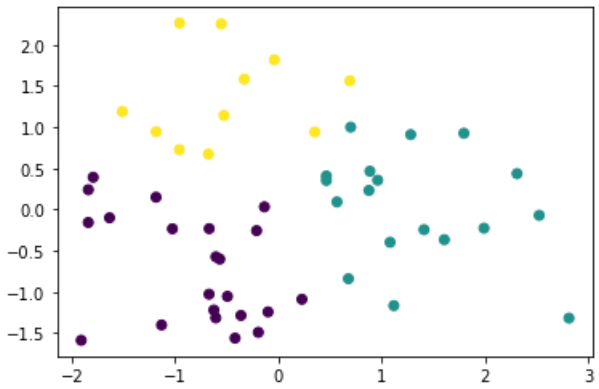
1. KMeans to visualize clusters in the PCA scatter plot

Since there wasn't a clear elbow in the scree plot, assigning the states to either two or three clusters is a reasonable choice, and we will resume our analysis using three clusters. Let's see how the PCA scatter plot looks if we color the states according to the cluster to which they are assigned.

In [11]:
```
# Create a KMeans object with 3 clusters
km = KMeans(n_clusters=3, random_state=8)
# Fit the data to the `km` object
km.fit(features_scaled)
# Create a scatter plot of the first two principal components
# and color it according to the KMeans cluster assignment
print("Visualize Clusters Using Scatter Plot, KMeans And PCA:\n")
plt.scatter(p_comps[:, 0], p_comps[:, 1], c=km.labels_)
```

```
Visualize Clusters Using Scatter Plot, KMeans And PCA:
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x2f78e0dd280>
```



1. Visualize the feature differences between the clusters

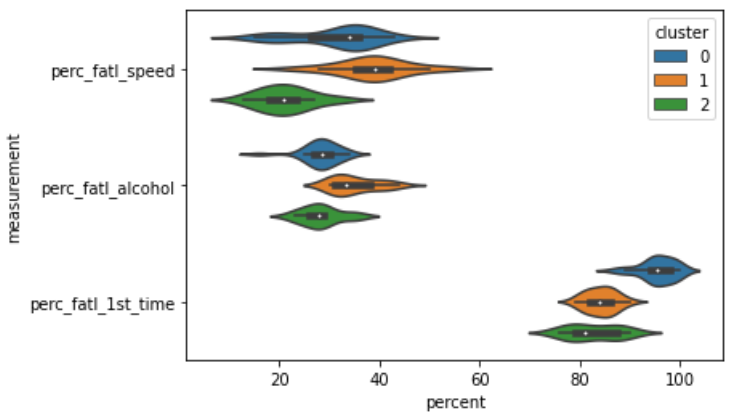
Thus far, we have used both our visual interpretation of the data and the KMeans clustering algorithm to reveal patterns in the data, but what do these patterns mean?

Remember that the information we have used to cluster the states into three distinct groups are the percentage of drivers speeding, under alcohol influence and that has not previously been involved in an accident. We used these clusters to visualize how the states group together when considering the first two principal components. This is good for us to understand structure in the data, but not always easy to understand, especially not if the findings are to be communicated to a non-specialist audience.

A reasonable next step in our analysis is to explore how the three clusters are different in terms of the three features that we used for clustering. Instead of using the scaled features, we return to using the unscaled features to help us interpret the differences.

In [12]:
```
# Create a new column with the labels from the KMeans clustering
car_acc['cluster'] = km.labels_

# Reshape the DataFrame to the long format
melt_car = pd.melt(car_acc, id_vars='cluster', var_name='measurement', value_name='percent',
                   value_vars=['perc_fatl_speed', 'perc_fatl_alcohol', 'perc_fatl_1st_time'])

# Create a violin plot splitting and coloring the results according to the km-clusters
sns.violinplot(y='measurement', x='percent', data=melt_car, hue='cluster')
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2f78df3c400>
```



In [13]:
```
# Read in the `miles-drives.csv`
miles_driven = pd.read_csv("D:/NYIT (Fall2020-ACADEMICS)/INTRODUCTION TO DATA MINING (CSCI657 or CSCI415) [PROF
F Helen Gu]/Final Projects/PROJECT1 PYTHON/datasets/miles-driven.csv", sep='|')

# Save the number of rows columns as a tuple
rows_and_cols = miles_driven.shape
print("Brief Description Of Dataset miles-driven.csv Stored In DataFrame Object miles_driven:\n")
print('There are {} rows and {} columns.\n'.format(rows_and_cols[0], rows_and_cols[1]))

# Generate an overview of the DataFrame
miles_driven_information = miles_driven.info()
print(miles_driven_information)

# Check whether there are Nan values or not
print("\nBrief Description Whether DataFrame Object miles_driven Consists Any NaN Values:\n", miles_driven.isna
().sum(), sep="")

# Display the first five rows of the DataFrame
print("\nFollowing Is The Generic Overview Of DataFrame Object miles_driven:\n")
miles_driven.head()
```

```
Brief Description Of Dataset miles-driven.csv Stored In DataFrame Object miles_driven:

There are 51 rows and 2 columns.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 2 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   51 non-null     object
 1   million_miles_annually  51 non-null     int64
dtypes: int64(1), object(1)
memory usage: 944.0+ bytes
None

Brief Description Whether DataFrame Object miles_driven Consists Any NaN Values:
state                   0
million_miles_annually    0
dtype: int64

Following Is The Generic Overview Of DataFrame Object miles_driven:
```

Out[13]:

	state	million_miles_annually
0	Alabama	64914
1	Alaska	4593
2	Arizona	59575
3	Arkansas	32953
4	California	320784

In [14]:
```
# Update the DataFrame `miles_driven` by converting `million_miles_annually` to `billion_miles_annually`
miles_driven['million_miles_annually'] = (miles_driven['million_miles_annually'] / 1000)
miles_driven.rename(columns={'million_miles_annually':'billion_miles_annually'}, inplace=True)
miles_driven.head()
```

Out[14]:

	state	billion_miles_annually
0	Alabama	64.914
1	Alaska	4.593
2	Arizona	59.575
3	Arkansas	32.953
4	California	320.784

In [15]:
```
# Merge the `car_acc` DataFrame with the `miles_driven` DataFrame
car_acc_miles = car_acc.merge(miles_driven, on='state')
print("After Merging We Get DataFrame Object car_acc_miles As:\n")
car_acc_miles.head()
```

```
After Merging We Get DataFrame Object car_acc_miles As:
```

Out[15]:

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time	cluster	billion_miles_annually
0	Alabama	18.8	39	30	80	1	64.914
1	Alaska	18.1	41	25	94	0	4.593
2	Arizona	18.6	35	28	96	0	59.575
3	Arkansas	22.4	18	26	95	0	32.953
4	California	12.0	35	28	89	0	320.784

In [16]:
```
# Create a new column for the number of drivers involved in fatal accidents
car_acc_miles['num_drvr_fatl_col'] = car_acc_miles['drvr_fatl_col_bmiles'] * car_acc_miles['billion_miles_annua
lly']
print("\nFollowing Is The Generic Overview Of DataFrame Object car_acc_miles After Adding New Column:\n")
car_acc_miles.head()
```

```
Following Is The Generic Overview Of DataFrame Object car_acc_miles After Adding New Column:
```

Out[16]:

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time	cluster	billion_miles_annually	num_drvr_fatl_col
0	Alabama	18.8	39	30	80	1	64.914	1220.3832
1	Alaska	18.1	41	25	94	0	4.593	83.1333
2	Arizona	18.6	35	28	96	0	59.575	1108.0950
3	Arkansas	22.4	18	26	95	0	32.953	738.1472
4	California	12.0	35	28	89	0	320.784	3849.4080

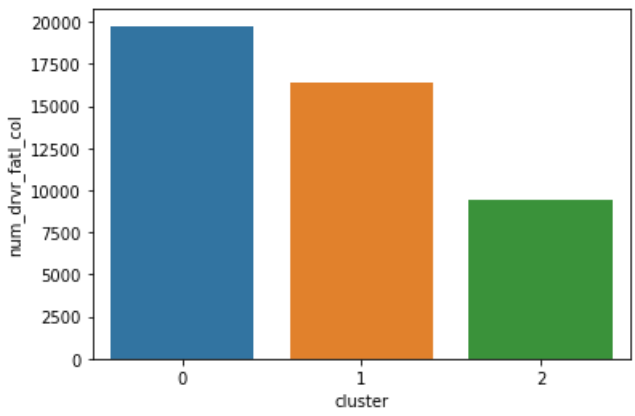
1. Compute the number of accidents within each cluster

Now it is clear that different groups of states may require different interventions. Since resources and time are limited, it is useful to start off with an intervention in one of the three groups first. Which group would this be? To determine this, we will include data on how many miles are driven in each state, because this will help us to compute the total number of fatal accidents in each state. Data on miles driven is available in another tab-delimited text file. We will assign this new information to a column in the DataFrame and create a violin plot for how many total fatal traffic accidents there are within each state cluster.

In [17]:
```
# Create a barplot of the total number of accidents per cluster
sns.barplot(x='cluster', y='num_drvr_fatl_col', data=car_acc_miles, estimator=sum, ci=None)

# Calculate the number of states in each cluster and their 'num_drvr_fatl_col' mean and sum.
count_mean_sum = car_acc_miles.groupby('cluster')['num_drvr_fatl_col'].agg(['count', 'mean', 'sum'])
count_mean_sum
```

Out[17]:

	count	mean	sum
cluster			
0	22	898.378595	19764.3291
1	18	911.406439	16405.3159
2	11	860.505945	9465.5654



In [18]:
```
# Directing corresponding states to cluster 0
print("Following Are The Respective States Corresponding To Cluster 0:\n")
for i in range(len(car_acc_miles)):
    if car_acc_miles.loc[i].cluster == 0:
        print(car_acc_miles.loc[i].state)
```

```
Following Are The Respective States Corresponding To Cluster 0:

Alaska
Arizona
Arkansas
California
Colorado
Delaware
District of Columbia
Florida
Georgia
Idaho
Illinois
Indiana
Louisiana
Maryland
Mississippi
Nevada
New Mexico
Oklahoma
Oregon
Utah
Vermont
West Virginia
```