In [1]:

```
import pandas as pd
```

In [3]:

```
road
```

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
5	Colorado	13.6	37	28	95
6	Connecticut	10.8	46	36	82
7	Delaware	16.2	38	30	99
8	District of Columbia	5.9	34	27	100
9	Florida	17.9	21	29	94
10	Georgia	15.6	19	25	93
11	Hawaii	17.5	54	41	87
12	Idaho	15.3	36	29	98
13	Illinois	12.8	36	34	96
14	Indiana	14.5	25	29	95
15	Iowa	15.7	17	25	87
16	Kansas	17.8	27	24	85
17	Kentucky	21.4	19	23	76
18	Louisiana	20.5	35	33	98
19	Maine	15.1	38	30	84
20	Maryland	12.5	34	32	99
21	Massachusetts	8.2	23	35	80
22	Michigan	14.1	24	28	77
23	Minnesota	9.6	23	29	88
24	Mississippi	17.6	15	31	100
25	Missouri	16.1	43	34	84
26	Montana	21.4	39	44	85
27	Nebraska	14.9	13	35	90
28	Nevada	14.7	37	32	99
29	New Hampshire	11.6	35	30	83
30	New Jersey	11.2	16	28	78
31	New Mexico	18.4	19	27	98
32	New York	12.3	32	29	80
33	North Carolina	16.8	39	31	81
34	North Dakota	23.9	23	42	86
35	Ohio	14.1	28	34	82
36	Oklahoma	19.9	32	29	94
37	Oregon	12.8	33	26	90
38	Pennsylvania	18.2	50	31	88
39	Rhode Island	11.1	34	38	79
40	South Carolina	23.9	38	41	81
41	South Dakota	19.4	31	33	86
42	Tennessee	19.5	21	29	81
43	Texas	19.4	40	38	87
44	Utah	11.3	43	16	96
45	Vermont	13.6	30	30	95
46	Virginia	12.7	19	27	88
47	Washington	10.6	42	33	86
48	West Virginia	23.8	34	28	87
49	Wisconsin	13.8	36	33	84
50	Wyoming	17.4	42	32	90

In [6]:

```
# Save the number of rows columns as a tuple
rows_and_cols = road.shape
print('There are {} rows and {} columns.\n'.format(
    rows_and_cols[0], rows_and_cols[1]))
# Generate an overview of the DataFrame
road_information = road.info()
print(road_information)
```

```
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
```

In [2]:

```
road = pd.read_csv("road-accidentss.csv")
```

In [8]:

```
# Display the last five rows of the DataFrame
road.tail()
```

Out[8]:

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
46	Virginia	12.7	19	27	88
47	Washington	10.6	42	33	86
48	West Virginia	23.8	34	28	87
49	Wisconsin	13.8	36	33	84
50	Wyoming	17.4	42	32	90

3.create a textual and graphical summary of the data

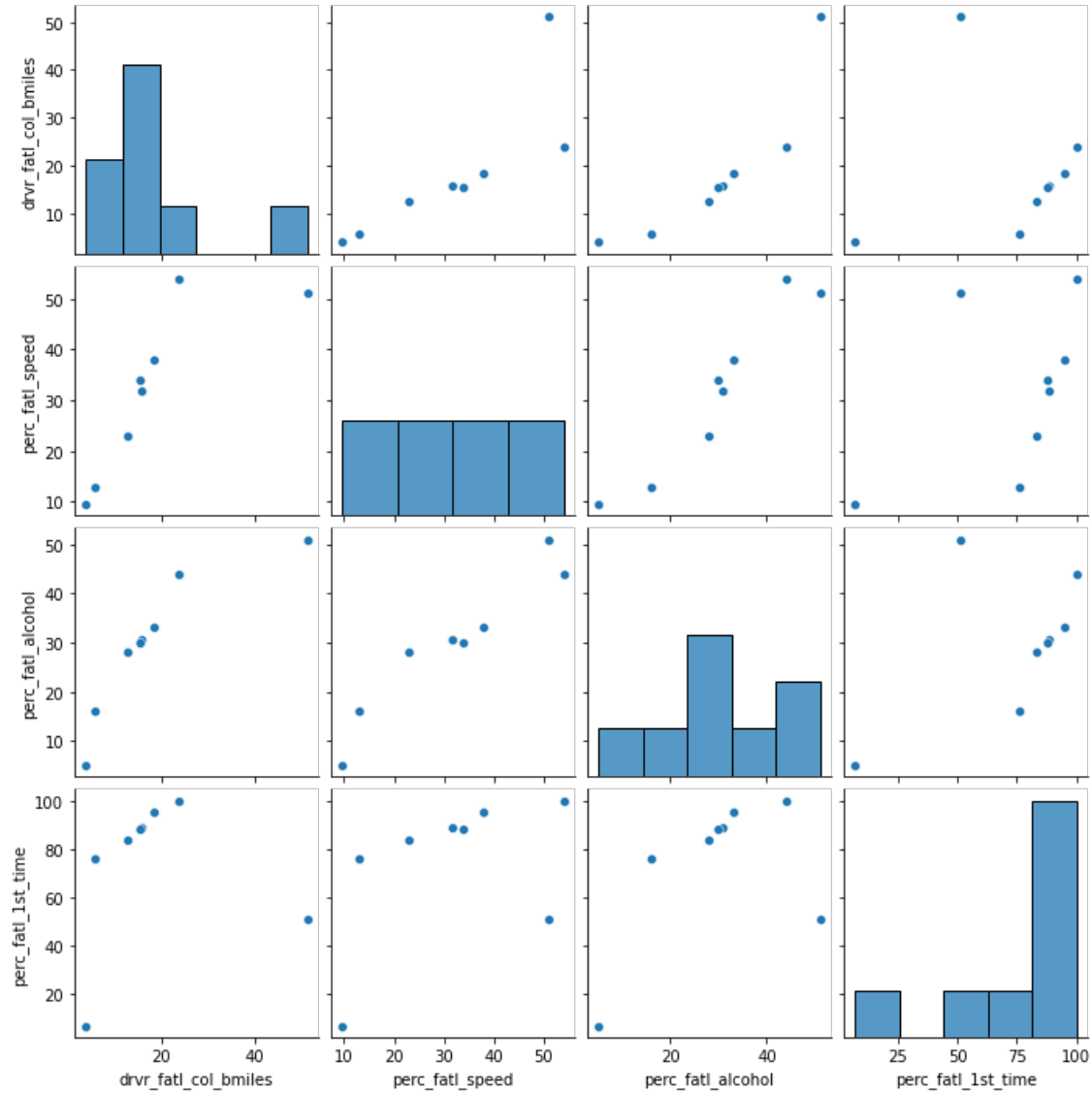
In [10]:

```
# import seaborn and make plots appear inline
import seaborn as sns
%matplotlib inline
# Compute the summary statistics of all columns in the `road` DataFrame
sum_stat_car = road.describe()
print(sum_stat_car)
# Create a pairwise scatter plot to explore the data
sns.pairplot(sum_stat_car)
```

```
       drvr_fatl_col_bmiles  perc_fatl_speed  perc_fatl_alcohol  \
count             51.000000        51.000000          51.000000
mean              15.790196        31.725490          30.686275
std                4.122002         9.633438           5.132213
min                5.900000        13.000000          16.000000
25%               12.750000        23.000000          28.000000
50%               15.600000        34.000000          30.000000
75%               18.500000        38.000000          33.000000
max               23.900000        54.000000          44.000000

       perc_fatl_1st_time
count            51.00000
mean             88.72549
std               6.96011
min              76.00000
25%              83.50000
50%              88.00000
75%              95.00000
max             100.00000
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x2f183a06f40>
```



1. Quantify the association of features and accidents

In [11]:

```
# Compute the correlation coefficent for all column pairs
corr_columns = road.corr()
corr_columns
```

Out[11]:

	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

5. Fit a multivariate linear regression

In [17]:

```
# Import the linear model function from sklearn
from sklearn import linear_model
# Create the features and target DataFrames
features = road[['perc_fatl_speed', 'perc_fatl_alcohol', 'perc_fatl_1st_time']]
target = road['drvr_fatl_col_bmiles']
# Create a linear regression object
reg = linear_model.LinearRegression()
# Fit a multivariate linear regression model
reg.fit(features, target)
# Retrieve the regression coefficients
fit_coef = reg.coef_
fit_coef
```

Out[17]:

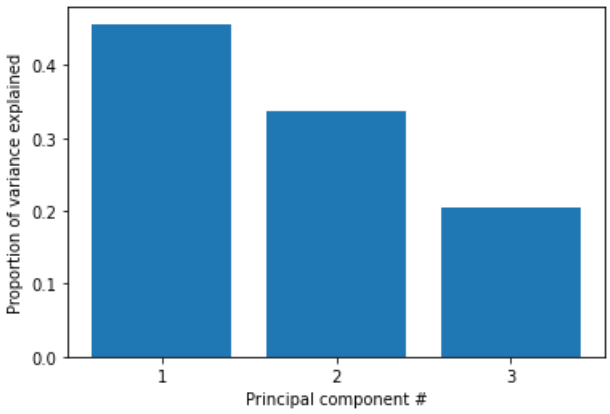
```
array([-0.04180041,  0.19086404,  0.02473301])
```

6. Perform PCA on standardized data

In [19]:

```
import numpy as np

# Standardize and center the feature columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Import the PCA class function 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_[0].cumsum()[0] + pca.explained_variance_ratio_[1].cumsum()[0]
print("The cumulative variance of the first two principal components is {}".format(
    round(two_first_comp_var_exp, 5)))
```

```
The cumulative variance of the first two principal components is 0.7947
```



7. Visualize the first two principal components

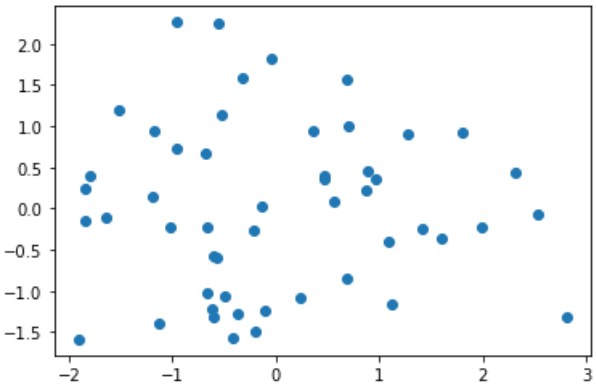
In [20]:

```
# 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
plt.scatter(p_comp1, p_comp2)
```

Out[20]:

```
<matplotlib.collections.PathCollection at 0x2f187023e50>
```



8. Find clusters of similar states in the data

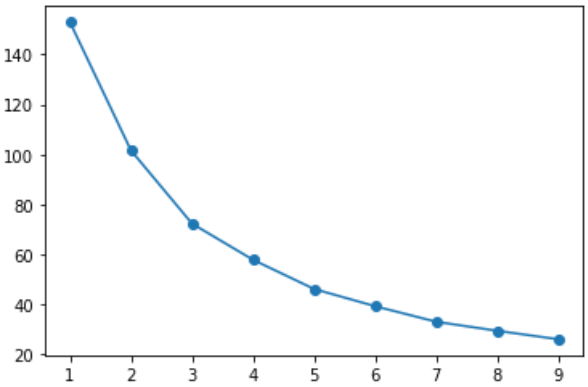
In [21]:

```
# Import KMeans from sklearn
from sklearn.cluster import KMeans

# A loop will be used to plot the explanatory 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 clusters (k)
    km = KMeans(n_clusters=k, random_state=8)
    # Fit the 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
plt.plot(ks, inertias, marker='o')
```

```
D:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x2f1872d4e20>]
```



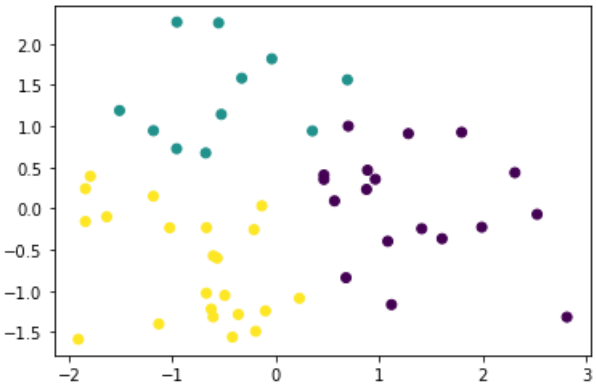
9. KMeans to visualize clusters in the PCA scatter plot

In [22]:

```
# Create a KMeans object with 3 clusters, use random_state=8
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
plt.scatter(p_comps[:, 0], p_comps[:, 1], c = km.labels_)
```

Out[22]:

<matplotlib.collections.PathCollection at 0x2f187766880>



10. Visualize the feature differences between the clusters

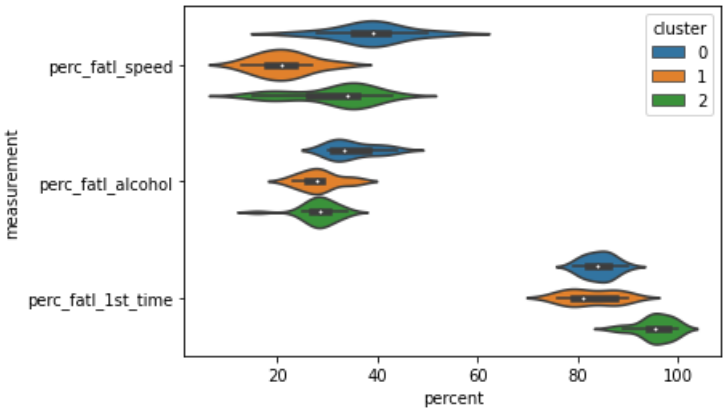
In [25]:

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

# Reshape the DataFrame to the long format
melt_car = pd.melt(road, 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(melt_car['percent'], melt_car['measurement'], hue = melt_car['cluster'])
```

```
D:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[25]:

<AxesSubplot:xlabel='percent', ylabel='measurement'>



11. Compute the number of accidents within each cluster

In [38]:

```
# Read in the new dataset
miles_drivens = pd.read_csv('miles-drivens.csv', sep=',')

display(miles_drivens.head())
```

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

In [40]:

```
# Merge the `road` DataFrame with the `miles_drivens` DataFrame
road_miles = road.merge(miles_drivens, on='state')
```

In [41]:

```
# Create a new column for the number of drivers involved in fatal accidents
```

```
road_miles['num_drvr_fatl_col'] = (road_miles['drvr_fatl_col_bmiles'] * road_miles['million_miles_annually']) / 1000

display(road_miles.head())
```

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time	cluster	million_miles_annually	num_drvr_fatl_col
0	Alabama	18.8	39	30	80	0	64914	1220.3832
1	Alaska	18.1	41	25	94	2	4593	83.1333
2	Arizona	18.6	35	28	96	2	59575	1108.0950
3	Arkansas	22.4	18	26	95	2	32953	738.1472
4	California	12.0	35	28	89	2	320784	3849.4080

In [42]:

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

Out[42]:

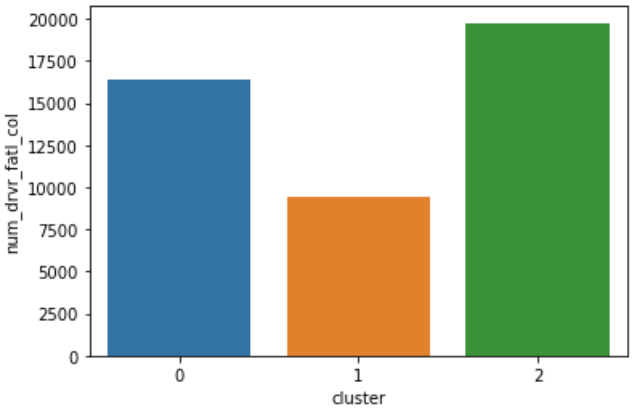
	count	mean	sum
cluster			
0	18	911.406439	16405.3159
1	11	860.505945	9465.5654
2	22	898.378595	19764.3291

In [43]:

```
# Create a barplot of the total number of accidents per cluster
sns.barplot(x='cluster', y='num_drvr_fatl_col', data=road_miles, estimator=sum, ci=None)
```

Out[43]:

```
<AxesSubplot:xlabel='cluster', ylabel='num_drvr_fatl_col'>
```



In [ ]:

12. Make a decision when there **is** no clear right choice
cluster_sum**=.....**