MCIS6273 Data Mining (Prof. Maull) / Fall 2022

HW2

Name: Manoj Korepu

Student ID: 999901236

(30%) Learn more about the analysis of the FARS database.

§ Task: Read the paper and write a 4-7 sentence (about a paragraph) summary. State in your own words what you learned, what expanding your knowledge of the topic and what you found interesting about the information you received. Please include the major points of the paper, and any weaknesses the authors point out with their research.

Summary of the paper.

The author seeks to compare the fatalities of pedestrian crashes between Kansas and the USA. On page 383, the author expresses that he had no knowledge of any study that had been one to compare the factors associated with fatal crashes among pedestrians between Kansas and the USA, this appears to be what may have motivated the author to conduct the study. Among the objectives, "To compare the trend of pedestrian fatalities in Kansas to that of the USA" got widely covered by the analysis. The findings of the paper hold that: male fatalities are 65%+; most vulnerable age groups are 45-54 and 65+ years; Friday and Saturday are associated with the most fatal pedestrian crashes; most crashes occur between 6 pm to 6 am; fatal crashes are common in October, November, and December; other findings are clear days, dark light conditions, and non-intersection locations all correlated with most fatal pedestrian crashes. What was particularly interesting was the fact that clear-day atmospheric conditions are associated with fatal pedestrian crashes, but the author justified the fact that people tend to be more outgoing on a clear day. Statistically, Chi-square was used to establish dependency between different variables and one of the observations was that there existed a strong relationship between age groups and gender among other observations. The weaknesses the author pointed out are; Kansas was not considered for a statistical test because of a low number of observations; a lower number of fatal pedestrian crashes might have contributed decrease in exposure level of walking over the years.

§ Task: Answer the following questions:

(a) What time of day is most common for pedestrian fatalities in Kansas (over all years)?
Answer:

Midnight to 5:59 am

(b) How does this compare with the most common time of day for the US overall?
Answer:

The most common time of day for the US overall is 6 pm to 8:59 pm. The most common time of day for Kansas has a lower percentage of pedestrian fatalities compared to that of the US overall as shown in Figure 9.

(c) Looking at figure 11, would you say poor atmospheric conditions have a significant impact on pedestrian fatalities?

Answer:

No. It is clear from figure 11 that clear conditions are associated with the most fatal accidental crashes because people tend to be outgoing during such conditions compared to poor atmospheric conditions.

(d) On page 392, the author *states "For Kansas, speed limits between 30 mph and 40 mph account for 52% of total crashes (26% crashes for 30 mph and 26% for 35 mph or 40mph), …".* Why is this statement as written incorrect?

Answer:

It is incorrect because the speeds have been grouped. 30mph falls in the group of 30 mph or less. He states that 30 mph itself accounts for 26% which is meant for the group 30 mph or less. Therefore, speed limits between 30 and 40 mph would constitute a different percentage altogether not totaling the percentages as done in the statement.

(e) The authors go on to explain the abnormally high number of fatalities at higher speeds with *"…Kansas has a lot of rural roads, where the speed limit is high and in rural roads, laws are not strictly enforced, all of which might lead to a larger number of fatal pedestrian crashes.".* Which of the suggested countermeasures would you think might successfully address this issue? If you do not find anything sufficient, what might you recommend instead?

Answer:

The suggested countermeasures that I find sufficient are; providing paved shoulders, installing raised medians at unsignalized intersections, converting parallel lanes into high visibility crosswalks, and finally Adding intersection lighting.

(20%) Recreate data analysis in a published paper.

Figure 15

Figure 4

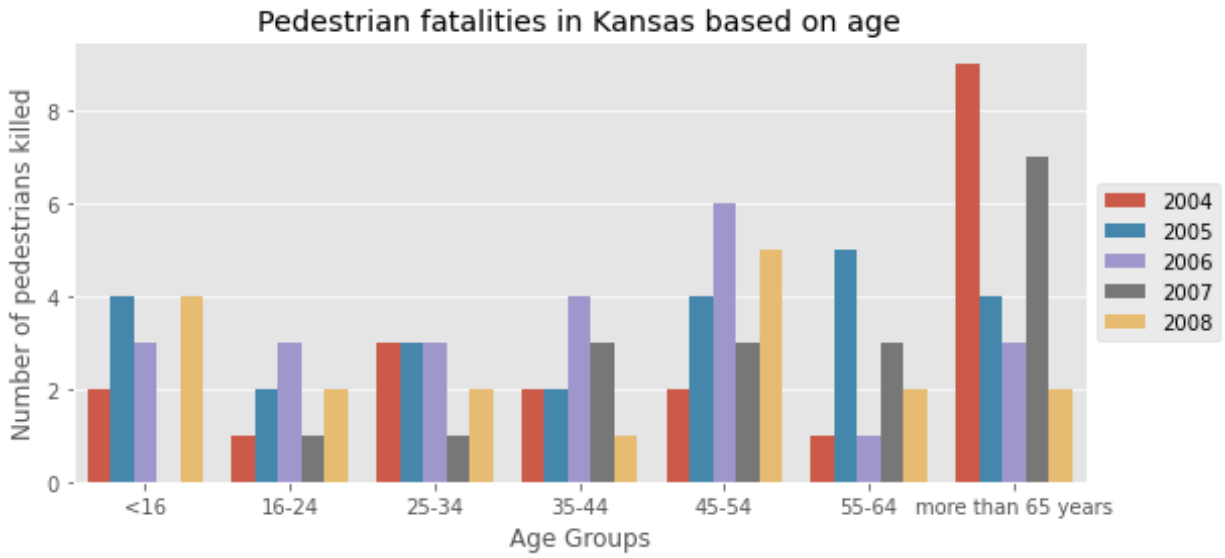


Figure 9

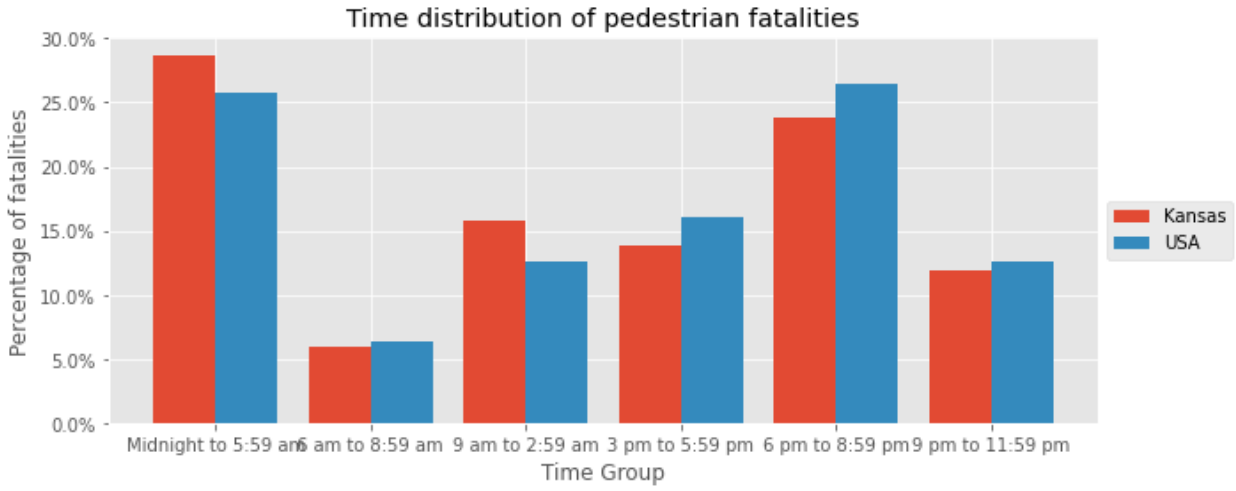
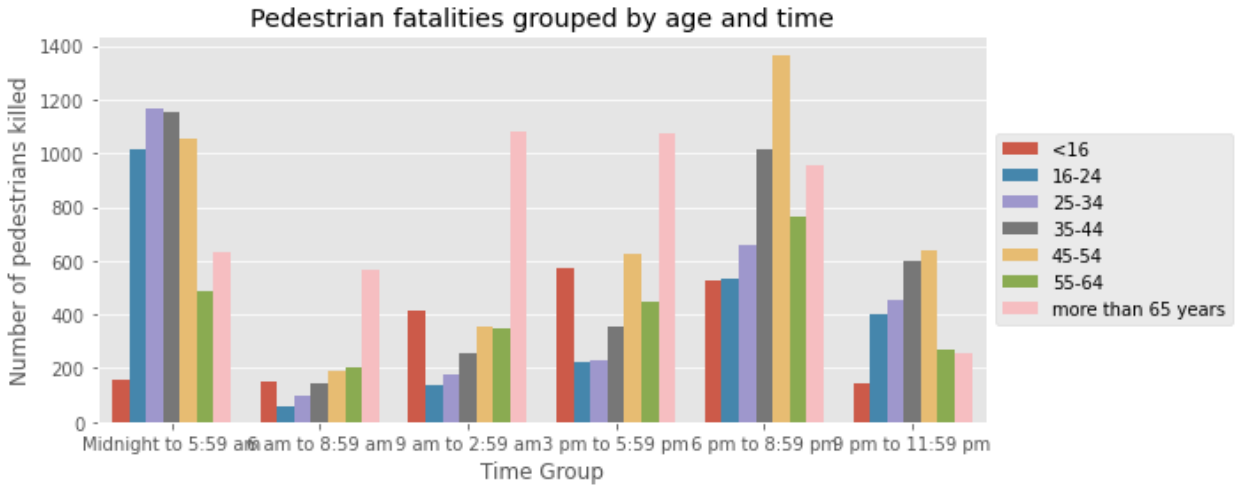


Figure of data from figure 4 and figure 9 combined.



(50%) Perform cluster analysis on NHTSA FARS data.

§ Task: (Perform ad hoc K-Means clustering). Centroids of each Cluster

```
-------------------------
Cluster 5; Centroids
-------------------------
[[-8.30798631e-02  3.12034430e-02  3.08106690e-03  3.82908581e-02
   2.78724454e-01 -2.06108912e-02  9.15724352e-01  8.80701941e-02
  -7.06098745e-02 -2.88476397e-02 -8.93942233e-01  4.29200400e-01
   1.54228758e+00  1.53408212e+00 -5.92381668e-01  2.96674666e+00
   3.01142789e+00  1.95539286e-01 -8.33542475e-01 -4.14432173e-01
  -6.03855099e-01 -4.84930434e-01 -5.95864494e-01 -2.44834510e-01
  -1.54259610e-01 -1.38138557e-01 -1.07273083e+00 -4.86579814e-01]
 [ 6.17523357e-02 -5.21502754e-03  2.21212246e-02  7.74248027e-03
  -2.00482803e-01 -2.06108912e-02 -1.02586554e+00 -2.91271140e-01
   1.49377551e-01 -5.62072628e-02  9.85755538e-01 -1.90034719e-01
  -4.92651113e-01 -4.80390439e-01  6.60064869e-01 -3.11341586e-01
  -3.05768001e-01 -6.60533423e-02  1.06412838e+00  1.08214989e+00
   1.11680601e+00  3.08009324e+00  2.71491035e+00  2.70619511e-02
  -1.03904221e-01 -1.35610834e-01  1.51527792e-01  3.07921844e+00]
 [ 4.76514878e-03  5.52075750e-03 -2.22268388e-02 -6.98305935e-04
  -1.47404845e-01 -4.49044989e-03 -1.02708887e+00 -1.91956097e-01
   1.19840705e-01 -2.14083655e-02  9.99530299e-01 -1.81948855e-01
  -4.92389360e-01 -4.79539549e-01  6.36939334e-01 -2.77029007e-01
  -2.77866604e-01 -3.02888015e-01  1.00307573e+00  3.36539207e-01
   5.95040898e-01 -3.31209888e-02  2.05946768e-01  3.16062461e-03
  -1.13021625e-01 -1.34560409e-01 -5.89744250e-03 -3.06166223e-02]
 [-9.35374841e-02  3.17763018e-02  9.49082840e-02  1.28105144e-01
```

```
  -8.43966298e-01  1.39582669e-01 -7.39946136e-01  4.10516535e-02
   9.52557933e-02 -6.03754409e-02  8.25121855e-01 -1.23177287e-01
  -4.82196542e-01 -4.71934893e-01  1.57281595e+00 -2.50271118e-01
  -2.63327626e-01  1.23955196e-01  7.77934208e-01  3.65860706e-01
   5.69930488e-01  5.37227881e-01  6.49623186e-01  6.94176018e+00
   7.18679122e+00  7.29465949e+00 -2.02896525e-01  5.38943703e-01]
 [ 5.43445690e-03 -1.14374258e-02  9.59446556e-03 -1.42754006e-02
   1.38068638e-01  6.17644256e-03  8.90342578e-01  1.95449926e-01
  -1.17553770e-01  3.71707086e-02 -8.67836878e-01  1.03151129e-01
   2.01560053e-01  1.89809602e-01 -5.98180443e-01 -3.19930223e-01
  -3.29107560e-01  2.18295545e-01 -8.96538206e-01 -4.16652453e-01
  -6.07014636e-01 -4.86752357e-01 -5.96541327e-01 -2.46826156e-01
  -1.54259610e-01 -1.38138557e-01  2.08570527e-01 -4.88391362e-01]]
--------------------------------------

-------------------------
Cluster 10; Centroids
-------------------------
[[ 1.82650223e-02  3.10608245e-03  3.99928811e-03 -7.79599481e-03
  -1.44803561e-01 -2.06108912e-02 -1.02779205e+00 -1.57695829e-01
   7.60232792e-03 -1.99405862e-03  9.99147453e-01 -1.90024235e-01
  -4.93236212e-01 -4.79817406e-01  6.43109962e-01 -2.77460801e-01
  -2.78105465e-01  4.11643695e-01  9.83257502e-01  3.17439146e-01
   5.56496448e-01 -4.07778675e-02  1.74950140e-01 -1.49788447e-03
  -1.15068994e-01 -1.34679325e-01  1.30202483e-01 -3.85286049e-02]
 [ 2.02545755e-02 -3.67828439e-02 -1.19777370e-02 -2.82602091e-02
   1.38962655e-01 -2.06108912e-02  8.88615105e-01  1.82394367e-01
  -1.60128529e-01 -1.05007599e-01 -8.72122903e-01  1.28054825e-02
  -4.53027601e-01 -4.80390439e-01 -5.97763311e-01 -3.12467035e-01
  -3.27872244e-01  1.96842736e-01 -9.03291527e-01 -4.17157302e-01
  -6.07646798e-01 -4.87120872e-01 -5.97020090e-01 -2.46810370e-01
  -1.54259610e-01 -1.38138557e-01  4.24972140e-01 -4.88761406e-01]
 [ 6.17523357e-02 -5.21502754e-03  2.21212246e-02  7.74248027e-03
  -2.00482803e-01 -2.06108912e-02 -1.02586554e+00 -2.91271140e-01
   1.49377551e-01 -5.62072628e-02  9.85755538e-01 -1.90034719e-01
  -4.92651113e-01 -4.80390439e-01  6.60064869e-01 -3.11341586e-01
  -3.05768001e-01 -6.60533423e-02  1.06412838e+00  1.08214989e+00
   1.11680601e+00  3.08009324e+00  2.71491035e+00  2.70619511e-02
  -1.03904221e-01 -1.35610834e-01  1.51527792e-01  3.07921844e+00]
 [-8.46243138e-02  2.73947186e-02  5.37428980e-03  5.09654116e-02
   2.78687710e-01 -2.06108912e-02  9.14099129e-01  3.86291903e-02
  -8.25096289e-02 -9.03058100e-02 -9.27410247e-01 -9.08427434e-03
   1.52686829e+00  1.53174949e+00 -5.92394636e-01  2.96998719e+00
   3.01920200e+00  1.83270936e-01 -8.46393493e-01 -4.14632010e-01
  -6.04599665e-01 -4.85227711e-01 -5.95811061e-01 -2.44992052e-01
  -1.54259610e-01 -1.38138557e-01 -1.08173859e+00 -4.86873659e-01]
 [ 2.42567563e-02  3.01235159e-02  8.65741468e-02  5.03413910e-02
   1.05187809e-01 -2.06108912e-02  8.80613847e-01  1.07338456e-01
  -1.38868588e-01 -8.30418119e-02 -9.16528990e-01  2.12186353e-03
   2.06176314e+00 -4.80390439e-01 -5.98709890e-01 -3.21726437e-01
  -3.28530071e-01  2.04117143e-01 -8.92483768e-01 -4.15223853e-01
  -6.05190558e-01 -4.86204465e-01 -5.95806034e-01 -2.46846193e-01
  -1.54259610e-01 -1.38138557e-01  1.11124160e-01 -4.87840948e-01]
 [-9.02086756e-02  3.05906949e-02  9.55129976e-02  1.33644510e-01
```

```
  -8.36333078e-01 -2.06108912e-02 -7.38986083e-01  4.23524237e-02
   9.37069723e-02 -5.90587572e-02  8.24574797e-01 -1.22955053e-01
  -4.82158482e-01 -4.71906894e-01  1.53162310e+00 -2.60945995e-01
  -2.74203741e-01  1.22767879e-01  7.76986546e-01  3.67280927e-01
   5.68588551e-01  5.38826985e-01  6.48173320e-01  6.94176018e+00
   7.18679122e+00  7.29465949e+00 -2.01983101e-01  5.40512013e-01]
 [-3.73608541e-02  1.03955699e-02 -1.00214146e-01  2.01775224e-02
  -1.57267490e-01 -2.06108912e-02 -1.02625527e+00 -2.95486355e-01
   4.41920669e-01 -8.48013155e-02  9.87577552e-01 -1.89363286e-01
  -4.92376516e-01 -4.80390439e-01  6.19367029e-01 -2.71062463e-01
  -2.74590000e-01 -2.37348698e+00  1.06183225e+00  3.92992157e-01
   7.10023123e-01 -9.43910773e-03  2.98250665e-01  1.71127222e-02
  -1.07111198e-01 -1.36050338e-01 -4.03011678e-01 -6.18356226e-03]
 [ 2.48253796e-01  3.42043595e-01  1.81161960e-01 -8.12113826e-01
  -4.88301872e-01  4.85180379e+01  3.82109829e-02  1.38973269e-02
  -1.26740777e-01 -7.55235739e-02  7.74020917e-02 -5.35812440e-02
  -3.27847742e-01 -4.80390439e-01  1.78387273e+00  1.35337514e-01
   1.46872416e-01 -4.02017098e-01 -9.49938842e-02 -7.99086078e-02
   5.88130058e-02 -2.39614879e-01 -1.44610761e-01  8.63082929e-01
   9.26241378e-01  1.33126019e+00 -8.26902962e-02 -2.38840220e-01]
 [-3.64701281e-02  1.59744214e-02 -8.84849992e-03 -1.66680865e-02
   1.44554851e-01 -2.06108912e-02  8.94256389e-01  1.65466586e-01
  -1.32319394e-01 -7.22944886e-02 -8.77048563e-01  1.18793522e-02
   3.87259261e-01  2.08164010e+00 -5.98709890e-01 -3.31484477e-01
  -3.28133158e-01  2.65354083e-01 -9.07989362e-01 -4.17028722e-01
  -6.07465301e-01 -4.86806322e-01 -5.97020090e-01 -2.46846193e-01
  -1.54259610e-01 -1.38138557e-01 -1.69187130e-01 -4.88449887e-01]
 [-3.32700181e-02  4.43423103e-02  1.34803822e-01 -1.57483398e-01
   3.22201182e-01 -2.06108912e-02  9.49773411e-01  1.60270634e+00
   1.12350601e+00  5.06171762e+00 -5.37661815e-02  6.80787368e+00
   9.74590576e-01  9.71006170e-01 -5.78352150e-01  6.73617662e-01
   6.62646796e-01  4.28194272e-01 -5.52147709e-01 -4.10179743e-01
  -6.07236417e-01 -4.85876230e-01 -5.95028559e-01 -2.45415411e-01
  -1.54259610e-01 -1.38138557e-01 -4.40823548e-01 -4.87507627e-01]]
--------------------------------------

-------------------------
Cluster 12; Centroids
-------------------------
[[ 5.39647561e-02  4.00324532e-03  1.48685955e-02 -1.04403391e-02
  -1.94113661e-01 -2.06108912e-02 -1.02560184e+00 -2.95114765e-01
   1.66516597e-01 -7.17278681e-02  9.85455015e-01 -1.90017842e-01
  -4.92582872e-01 -4.80390439e-01  6.65074510e-01 -3.09611412e-01
  -3.04033293e-01 -7.99849803e-02  1.06412838e+00  4.13427020e-01
   6.88579365e-01  3.08624691e+00  2.71828720e+00  3.41241648e-02
  -1.01504121e-01 -1.35674140e-01  1.69533961e-01  3.08534864e+00]
 [-2.40416776e-02  3.49024309e-02  2.54152027e-02  1.60255939e-02
   1.24005869e-01 -2.06108912e-02  8.90106189e-01 -1.32897311e-01
  -1.00595563e-01 -1.05014333e-01 -9.62642644e-01 -2.95407320e-02
   1.07985849e+00  1.06507771e+00 -5.98709890e-01 -3.30456719e-01
  -3.31792109e-01  2.23229739e-01 -9.07364413e-01 -4.16228209e-01
  -6.06454563e-01 -4.86513915e-01 -5.96489728e-01 -2.46846193e-01
  -1.54259610e-01 -1.38138557e-01 -5.42649122e-02 -4.88154670e-01]
 [ 9.89541018e-02 -9.32301019e-03  5.70968928e-02 -3.19218258e-02
```

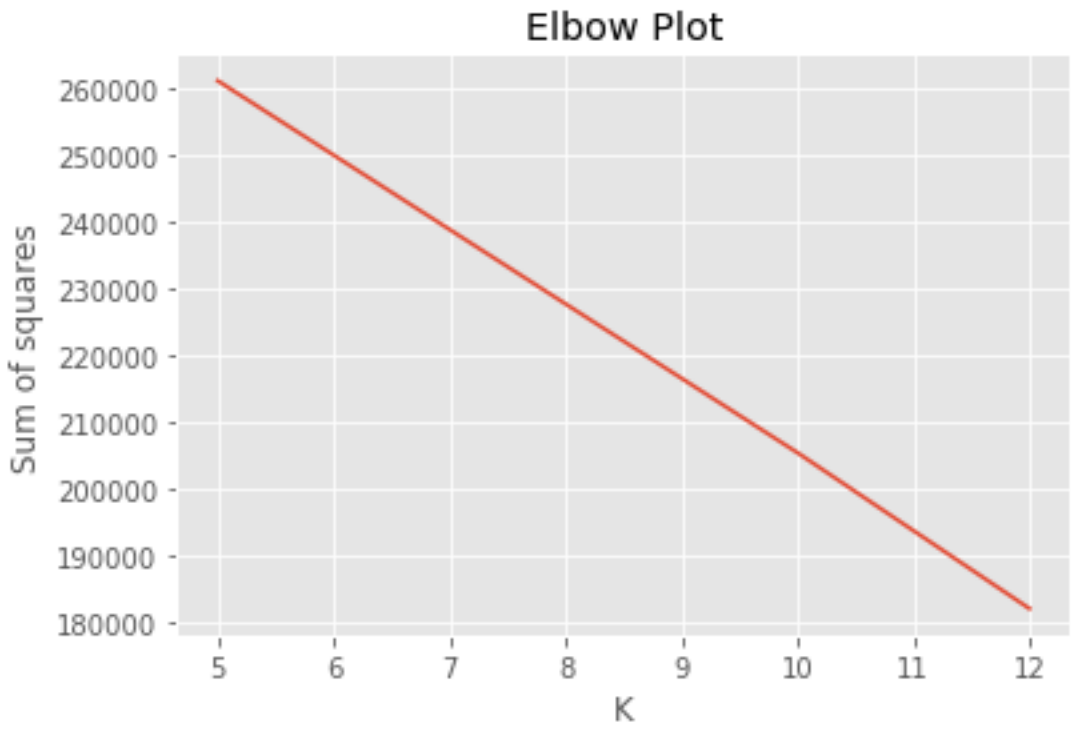
```
 -1.41185620e-01 -2.06108912e-02 -1.02781012e+00 -2.66645201e-01
  1.32505658e+00  1.69186634e-02  9.95646897e-01 -1.89960240e-01
 -4.92886208e-01 -4.80390439e-01  5.07014993e-01 -2.80184284e-01
 -2.82827986e-01  4.14737874e-01  1.02364906e+00  3.73115096e-01
  6.05576038e-01 -1.28087006e-02  2.10252676e-01  1.66346090e-02
 -1.17389117e-01 -1.36224132e-01  1.51123806e-01 -1.04543858e-02]
[-7.37933872e-02  2.42720430e-02  9.24986507e-02  1.40110742e-01
 -8.46957427e-01 -2.06108912e-02 -7.35081434e-01  4.76428044e-02
  7.79446495e-02 -5.37036541e-02  8.22349851e-01 -1.22051205e-01
 -4.82003688e-01 -4.71793021e-01  1.54734088e+00 -2.59915786e-01
 -2.73425374e-01  1.18804667e-01  7.73132293e-01  2.20955706e-01
  4.73673052e-01  5.04633429e-01  6.20117989e-01  6.94176018e+00
  7.18679122e+00  7.29465949e+00 -1.95610588e-01  5.06350283e-01]
[-3.68558774e-02  1.31342377e-02 -9.93001527e-02  1.83879238e-02
 -1.53815492e-01 -2.06108912e-02 -1.02623641e+00 -3.06011715e-01
  4.40985277e-01 -8.54713075e-02  9.87563218e-01 -1.89358456e-01
 -4.92369681e-01 -4.80390439e-01  6.20616804e-01 -2.72868812e-01
 -2.76470777e-01 -2.37471344e+00  1.06280948e+00  3.71726160e-01
  6.99814739e-01 -8.97271211e-03  2.97913106e-01  1.80135996e-02
 -1.06865953e-01 -1.36039476e-01 -4.04322841e-01 -5.72524275e-03]
[-7.84839157e-02  3.27951714e-02  1.28398536e-02  4.77609654e-02
  2.82865447e-01 -2.06108912e-02  9.21056893e-01 -3.44788604e-02
 -6.77618583e-02 -6.56956967e-02 -9.42354718e-01 -1.80863456e-02
  1.54816323e+00  1.54076698e+00 -5.93585112e-01  2.96638802e+00
  3.01504908e+00  1.81870895e-01 -8.44247700e-01 -4.15271988e-01
 -6.05969078e-01 -4.85792390e-01 -5.95920484e-01 -2.45182164e-01
 -1.54259610e-01 -1.38138557e-01 -1.08397635e+00 -4.87434058e-01]
[-4.26837072e-02  6.88070822e-03 -3.44493893e-02  1.05045521e-02
 -1.47384051e-01 -2.06108912e-02 -1.02772813e+00 -1.74280175e-01
 -8.93404959e-01 -2.03468050e-02  9.99335156e-01 -1.90062009e-01
 -4.93467608e-01 -4.79406178e-01  7.40916202e-01 -2.72820197e-01
 -2.72214409e-01  4.08091032e-01  9.73342346e-01  2.69706736e-01
  5.31038054e-01 -4.97699532e-02  1.67600526e-01 -1.00127487e-02
 -1.12497461e-01 -1.33523650e-01  1.19801705e-01 -4.75120213e-02]
[ 3.35227294e-02 -3.78620197e-02 -1.17279937e-02 -2.14755089e-02
  1.31298016e-01 -2.06108912e-02  8.90525703e-01 -1.34299647e-01
 -1.19291502e-01 -1.36780173e-01 -9.57699305e-01 -2.78737990e-02
 -4.45556107e-01 -4.80390439e-01 -5.98709890e-01 -3.11231416e-01
 -3.26730397e-01  1.76449022e-01 -9.00601951e-01 -4.17157302e-01
 -6.07646798e-01 -4.87120872e-01 -5.97020090e-01 -2.46846193e-01
 -1.54259610e-01 -1.38138557e-01  4.37023123e-01 -4.88761406e-01]
[ 9.89893009e-02 -7.75721859e-02  1.21411471e-01  2.71266811e-01
 -2.95678028e-01 -2.06108912e-02 -1.02988237e+00 -2.48356783e-01
 -3.86350106e-02  1.64015842e-01  9.90333273e-01 -1.90291785e-01
 -4.93690601e-01 -4.80390439e-01  5.88002398e-01 -3.04249703e-01
 -2.98318799e-01  9.71665557e-02  1.06412838e+00  1.12685100e+01
  7.63979328e+00  2.70561269e+00  2.43610590e+00  2.06429874e-01
  1.54332671e-01  1.65210001e-01 -1.07921306e-01  2.70538832e+00]
[-1.04683710e-02 -6.56294538e-02  4.69729507e-02 -5.91232466e-02
  1.75238221e-01 -2.06108912e-02  7.18613488e-01  3.18164316e+00
 -2.15069596e-01  1.25859311e+00  1.69252159e-02  3.55879806e-01
  4.10038659e-03  1.81197193e-01 -5.05856693e-01 -2.57127744e-01
 -2.59085864e-01  4.02188407e-01 -7.44799294e-01 -4.08043049e-01
 -5.99632408e-01 -4.83785546e-01 -5.88704190e-01 -2.40779858e-01
```

```
 -1.54259610e-01 -1.38138557e-01  5.25626524e-02 -4.85369985e-01]
[-5.10998597e-02  1.54439301e-01 -3.73103404e-02 -2.99384291e-01
  3.26645159e-01 -2.06108912e-02  9.35135527e-01  1.74852500e+00
  7.14533270e-02  1.24684694e+00  9.82096400e-02  1.16522589e+01
  1.38160942e+00  8.99164465e-01 -5.76003180e-01  1.27547091e+00
  1.25058598e+00  4.34565035e-01 -7.34616374e-01 -4.17157302e-01
 -6.07646798e-01 -4.87120872e-01 -5.97020090e-01 -2.46846193e-01
 -1.54259610e-01 -1.38138557e-01 -4.62208969e-01 -4.88761406e-01]
[ 2.48253796e-01  3.42043595e-01  1.81161960e-01 -8.12113826e-01
 -4.88301872e-01  4.85180379e+01  3.82109829e-02  1.38973269e-02
 -1.26740777e-01 -7.55235739e-02  7.74020917e-02 -5.35812440e-02
 -3.27847742e-01 -4.80390439e-01  1.78387273e+00  1.35337514e-01
  1.46872416e-01 -4.02017098e-01 -9.49938842e-02 -7.99086078e-02
  5.88130058e-02 -2.39614879e-01 -1.44610761e-01  8.63082929e-01
  9.26241378e-01  1.33126019e+00 -8.26902962e-02 -2.38840220e-01]]
```

Each value k has k arrays where each array has cluster centers of each group. Now for instance, when k = 12, there are 12 clusters (arrays). Each cluster has centroids for every feature.

§ Task: (Perform elbow analysis to find optimal cluster size)

- **The elbow graphs**



- **The optimal K**

The optimal K is 10. The elbow plot does not display much of the information needed to deduce the optimal k but using the Sum of Squares; [261009.01584825863, 205469.6311434515, 182197.55794660226] for 5, 10, and 12 respectively. Looking at the values, it is notable that the difference in the sum of squares of k = 10 and k = 12, is minimal compared to that of k = 5 and k = 10. Therefore, 10 is the optimal value

- **The reanalysis of the previous answer based on the optimal *K* (re-run your clusters and report their centroid characteristics)**

The cluster centers, more so the centroids comprise of 10 arrays in a 2-D array where each array represents the centroids of each label, that is label 0 through 9. These cluster centers represent where each data point will be associated to while predicting during clustering.

§ Task: (Find out where Kansas fits in)

Kansas fits in 7; the seventh cluster/label**.**

By extension, which states are most like Kansas (belong to the same cluster) regarding pedestrian fatalities?

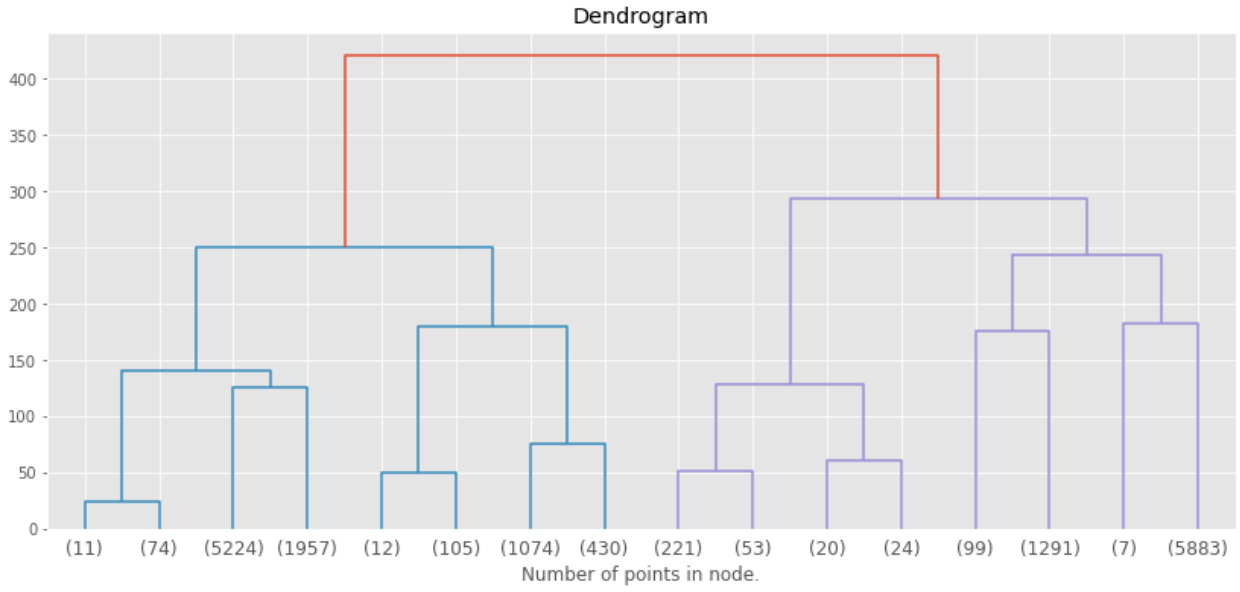
Python Output below; Kansas fits in 7, therefore the states below also fall in label 7.

```
----------------------------------------
The states which are most like Kansas are:
----------------------------------------
        South Dakota
             Wyoming
            Delaware
             Vermont
               Idaho
       New Hampshire
              Nevada
               Maine
                Iowa
              Hawaii
District of Columbia
            Colorado
            Missouri
          Washington
              Oregon
          New Mexico
            Arkansas
           Wisconsin
            Oklahoma
             Arizona
```

```
          Virginia
     Massachusetts
          Kentucky
          Maryland
         Louisiana
        New Jersey
              Ohio
    North Carolina
      Pennsylvania
             Texas
-----------------------------------------
```

§ Task: (Perform Agglomorative Clustering)

- **The plot of the Dendrogram**



Appendix (Code)

- **Please note that the data has been provided in the zipped file together with the IPYNB file. The data was obtained from the NTHSA FARS website, Kansas dataset are a result of filtering the FARS dataset where the State is KANSAS.**

```
# The necessary modules
import os
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
plt.style.use('ggplot')
import numpy as np
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
import scipy.stats as st
from scipy.cluster.hierarchy import dendrogram

################################
#### READING IN THE DATA ######
################################
csvs = os.listdir('temp_csv')

df = pd.read_csv('temp_csv/temp_csv0/accident.csv')
for folder in csvs[1:]:
    try:
        df1 = pd.read_csv(f'temp_csv/{folder}/accident.csv')
    except:
        df1 = pd.read_csv(f'temp_csv/{folder}/accident.csv',
encoding='ISO8859-1')
    df = pd.concat([df, df1], axis=0)

df = df.reset_index(drop = True)

# Get the pedestrians
dfp = df[df['HARM_EV'] == 8]

state = dfp[['STATE','STATENAME']].dropna()
state_dict = dict(zip(state['STATE'], state['STATENAME']))
```

```
# Map the states to their statenames
dfp['STATENAME'] = dfp['STATE'].map(state_dict)

# Get the Kansas DataFrame
kansas = dfp[dfp['STATENAME'] == 'Kansas']

####################################
##### Plotting figure 15 ###########
####################################

####################
### USA ############
####################
speed_limit_us = dfp[['SP_LIMIT', 'FATALS']]
# Sum the fatalities by the speed limit
sum_speed = speed_limit_us.groupby('SP_LIMIT')['FATALS'].sum()

speed_dict = {'30 mph or less': 0,
              '30 or 40 mph' : 0,
              '45 or 50 mph' : 0,
              '55 mph' : 0,
              '60 mph or higher': 0}
total_fatals = 0
for index in sum_speed.index:
    if int(index) in range(0, 31):
        speed_dict['30 mph or less'] += sum_speed[index]
    elif int(index) in range(31, 41):
        speed_dict['30 or 40 mph'] += sum_speed[index]
    elif int(index) in range(41, 51):
        speed_dict['45 or 50 mph'] += sum_speed[index]
    elif int(index) in range(51, 60):
        speed_dict['55 mph'] += sum_speed[index]
    else:
        speed_dict['60 mph or higher'] += sum_speed[index]
    total_fatals += sum_speed[index] # Find the total fatalities

speed_percent = {k:(speed_dict[k]/total_fatals)*100 for k in speed_dict}

####################
### KANSAS ########
####################
speed_limit_ka = kansas[['SP_LIMIT', 'FATALS']]
# Sum the fatalities by the speed limit
sum_speed_ka = speed_limit_ka.groupby('SP_LIMIT')['FATALS'].sum()
```

```
speed_dict_ka = {'30 mph or less': 0,
             '30 or 40 mph' : 0,
             '45 or 50 mph' : 0,
             '55 mph' : 0,
             '60 mph or higher': 0}
total_fatals_ka = 0
for index in sum_speed_ka.index:
    if int(index) in range(0, 31):
        speed_dict_ka['30 mph or less'] += sum_speed_ka[index]
    elif int(index) in range(31, 41):
        speed_dict_ka['30 or 40 mph'] += sum_speed_ka[index]
    elif int(index) in range(41, 51):
        speed_dict_ka['45 or 50 mph'] += sum_speed_ka[index]
    elif int(index) in range(51, 60):
        speed_dict_ka['55 mph'] += sum_speed_ka[index]
    else:
        speed_dict_ka['60 mph or higher'] += sum_speed_ka[index]
    total_fatals_ka += sum_speed_ka[index] # Find the total fatalities

speed_percent_ka = {k:(speed_dict_ka[k]/total_fatals_ka)*100 for k in
speed_dict_ka}

########################################
######## CODE PLOT FIGURE 15 ##########
########################################
fig, ax = plt.subplots(figsize = (8, 3))

x_axis = list(speed_dict.keys())  # X axis labels
x = np.arange(len(x_axis))
y1, y2 = list(speed_percent.values()), list(speed_percent_ka.values())

ax.bar(x-0.2, y2, 0.4, label = 'Kansas')
ax.bar(x+0.2, y1, 0.4, label = 'USA')
ax.yaxis.set_major_formatter(ticker.PercentFormatter())
ax.legend(loc='center left', bbox_to_anchor=(1,0.5))
plt.xlabel("Speed"), plt.ylabel("Percentage of fatal crashes")
plt.title("Fatal pedestrian crashes based on speed limit")
plt.xticks(x, x_axis)
plt.show()

####################################
##### Figure 4 ####################
####################################
# Reading in the datasets
```

```
csvs_ = os.listdir('2004-2008')
df_person = pd.read_csv('2004-2008/person.csv')
for file,i in zip(csvs_[1:],range(1,5)):
    try:
        df1 = pd.read_csv(f'2004-2008/person{i}.csv')
    except:
        df1 = pd.read_csv(f'2004-2008/person{i}.csv', encoding='ISO8859-1')
    df_person = pd.concat([df_person, df1], axis=0)

# Get the pedestrians only
dfp_person = df_person[df_person['HARM_EV'] == 8].reset_index(drop = True)

# Select the year between 2004 - 2009
dfp_person = dfp_person[dfp_person['DEATH_YR'].isin(range(2004, 2009))]

# Select the state of KANSAS
dfp_person_kansas = dfp_person[dfp_person['STATE'] == 20]

# group the data by age
dfp_person_kansas['CATEGORY'] = pd.cut(dfp_person_kansas['AGE'], bins=[0, 16,
24, 34, 44, 54, 64, 99],
       include_lowest=True, labels=['<16', '16-24', '25-34', '35-44', '45-
54', '55-64', 'more than 65 years'])

group_age_year = dfp_person_kansas.groupby(["DEATH_YR",
"CATEGORY"])["CATEGORY"].count()

# Get the counts of every age into a dataframe
age_cat_count = group_age_year.to_frame().rename(columns =
{'CATEGORY':'CAT_COUNT'}).reset_index()

##############################
##### CODE TO PLOT ##########
##############################
plt.figure(figsize = (9, 4))
sns.barplot(data = age_cat_count, x = "CATEGORY", y = "CAT_COUNT", hue =
"DEATH_YR")
plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
plt.xlabel("Age Groups")
plt.ylabel("Number of pedestrians killed")
plt.title("Pedestrian fatalities in Kansas based on age")
plt.show()

####################################
##### Figure 9 ####################
```

```
#####################################
#######
# USA #
#######
df_time = dfp_person[['HOUR']]
df_time["HOUR_CATEGORY"] = pd.cut(df_time['HOUR'], bins=[0, 6, 9, 15, 18, 21,
24],
       include_lowest=True, labels=['Midnight to 5:59 am',
                                    '6 am to 8:59 am',
                                    '9 am to 2:59 am',
                                    '3 pm to 5:59 pm',
                                    '6 pm to 8:59 pm',
                                    '9 pm to 11:59 pm'])
group_time = df_time['HOUR_CATEGORY'].value_counts(sort = False).to_frame()

##########
# KANSAS #
##########
df_time_ka = dfp_person[['HOUR']][dfp_person['STATE'] == 20]
df_time_ka["HOUR_CATEGORY"] = pd.cut(df_time_ka['HOUR'], bins=[0, 6, 9, 15,
18, 21, 24],
       include_lowest=True, labels=['Midnight to 5:59 am',
                                    '6 am to 8:59 am',
                                    '9 am to 2:59 am',
                                    '3 pm to 5:59 pm',
                                    '6 pm to 8:59 pm',
                                    '9 pm to 11:59 pm'])
group_time_ka = df_time_ka['HOUR_CATEGORY'].value_counts(sort =
False).to_frame()


##########
## PLOT ##
##########
fig, ax = plt.subplots(figsize = (10, 4))

x_labels = group_time.index.tolist()  # X axis labels
x = np.arange(len(x_labels))
y1 =
(group_time_ka['HOUR_CATEGORY']/group_time_ka['HOUR_CATEGORY'].sum())*100
y2 = (group_time['HOUR_CATEGORY']/group_time['HOUR_CATEGORY'].sum())*100

ax.bar(x-0.2, y1, 0.4, label = 'Kansas')
ax.bar(x+0.2, y2, 0.4, label = 'USA')
ax.yaxis.set_major_formatter(ticker.PercentFormatter())
```

```
ax.legend(loc='center left', bbox_to_anchor=(1,0.5))
plt.xlabel("Time Group"), plt.ylabel("Percentage of fatalities")
plt.title("Time distribution of pedestrian fatalities")
plt.xticks(x, x_labels)
plt.show()

################################################
### Combining data from  figure 4 and figure 9 ##
################################################
age_time = dfp_person[['AGE', 'HOUR']]

# Grouping Hour into categories
age_time["HOUR_C"] = pd.cut(age_time['HOUR'], bins=[0, 6, 9, 15, 18, 21, 24],
       include_lowest=True, labels=['Midnight to 5:59 am',
                                    '6 am to 8:59 am',
                                    '9 am to 2:59 am',
                                    '3 pm to 5:59 pm',
                                    '6 pm to 8:59 pm',
                                    '9 pm to 11:59 pm'])
# Grouping Age into categories
age_time['AGE_C'] = pd.cut(age_time['AGE'], bins=[0, 16, 24, 34, 44, 54, 64,
99],
       include_lowest=True, labels=['<16', '16-24', '25-34', '35-44', '45-
54', '55-64', 'more than 65 years'])

# Group by Age categories and Hour categories
counts_agetime = age_time.groupby(['AGE_C', 'HOUR_C']).count().reset_index()

##############################
##### CODE TO PLOT ##########
##############################
plt.figure(figsize = (9, 4))
sns.barplot(data = counts_agetime, x = "HOUR_C", y = "AGE", hue = "AGE_C")
plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
plt.xlabel("Time Group")
plt.ylabel("Number of pedestrians killed")
plt.title("Pedestrian fatalities grouped by age and time")
plt.show()

#############################################################################
############ (50%) Perform cluster analysis on NHTSA FARSdata. #############
#############################################################################
person = pd.read_csv('PERSON.CSV')

# Select rows where harmed person is pedestrian and Eliminate KANSAS state
```

```
dfp = person[(person['HARM_EV'] == 8)]

# Restrict to numeric features
dfp = dfp._get_numeric_data()

# Drop columns with NAN values
dfp = dfp.dropna(axis = 1)

# Select Kansas and Other states differently
kansas_df = dfp[(dfp['STATE'] == 20)]
US_df = dfp[(dfp['STATE'] != 20)]
state = US_df[['STATE']]

# Eliminate year as a feature and other columns that won't be 'numerically
feasible'
US_df = US_df.drop(columns = ['DEATH_YR', 'HARM_EV', 'STATE', 'COUNTY'])
kansas_df = kansas_df.drop(columns = ['DEATH_YR', 'HARM_EV', 'STATE',
'COUNTY'])

# Scaling the data using Standard Scaler
scaler = StandardScaler()
scaler.fit(US_df) # Fit the data
US_scaled = scaler.transform(US_df)

# CLusters
K = [5, 10, 12]
sum_squares = []
for k in K:
    kmeans = KMeans(n_clusters = k, init='k-means++', max_iter = 200, n_init
= 10, random_state = 0)
    kmeans.fit(US_scaled)
    sum_squares.append(kmeans.inertia_)
    print('-------------------------')
    print(f"Cluster {k}; Centroids")
    print('-------------------------')
    print(kmeans.cluster_centers_)
    print("-------------------------------------\n")

# Plot the sum of squares
plt.plot(K, sum_squares)
plt.xlabel("K"), plt.ylabel("Sum of squares")
plt.title("Elbow Plot")
plt.show()

# Finding out where Kansas fits in
```

```
kansas_labels = kmeans.predict(scaler.transform(kansas_df))

# Find the cluster that is most appearing
mode_label = st.mode(kansas_labels)[0][0]
print(f"Kansas would belong to the cluster: {mode_label}")

# Finding states that are like Kansas
state['Labels'] = kmeans.predict(US_scaled)

state_count = state.groupby(['STATE',
'Labels'])['Labels'].count().sort_values().groupby(level=0).tail(1).to_frame(
)
state_count = state_count.rename(columns =
{"Labels":"Labels_Count"}).reset_index()

# Map the states to their statenames
state_count['State_Name'] = state_count['STATE'].map(state_dict)

# Getting the states that are the same as Kansas
kansas_like = state_count['State_Name'][state_count['Labels'] == mode_label]
print('\n-----------------------------------------')
print("The states which are most like Kansas are:")
print('-----------------------------------------')
print(kansas_like.to_string(index = False))
print('-----------------------------------------')

###################################
# Sklearn Aglomerative Clustering #
###################################
def plot_dendrogram(model, **kwargs):
    cts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_) # Get the length of the labels
    for i, merge in enumerate(model.children_):
        c_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                c_count += 1  # leaf node
            else:
                c_count += cts[child_idx - n_samples]
        cts[i] = c_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, cts]
    ).astype(float)
```

```
    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)

# setting compute_distances = True will add the attribute distances_ in the
model.
aggmodel = AgglomerativeClustering(distance_threshold=None, n_clusters=10,
compute_distances = True)

aggmodel = aggmodel.fit(US_scaled)
plt.figure(figsize = (14, 6))
plt.title("Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(aggmodel, truncate_mode="level", p=3)
plt.xlabel("Number of points in node.")
plt.show()
```