Money in Education:

A Comprehensive Analysis of Illinois Teacher Compensation

Andrew Carr, Sara McCarthy, Nick Scope, Tzu-Hao Peng

DePaul University - College of Computing and Digital Media

Money in Education:

A Comprehensive Analysis of Illinois Teacher Compensation

Funding for public education has been a complex and frequently debated topic in the United States for decades. Mass teacher strikes, reduced education budgets, and a lack of qualified teachers can be seen throughout the country. Beyond federal mandates, each state has its own compensation system for teachers. Harnessing survey data gathered from Illinois educators across 10 years, this project aims to shed light on the key features that inform full-time educator salary decisions in Illinois through the use of linear regression, classification models, and feature specific visualizations.

I. Exploratory Analysis

The Data

The Raw Data:

The dataset for this project was obtained via voluntary survey responses from Illinois educators over a period of 10 years (2003-2012). Each instance (of which there were approximately 1.6 million) represents an educator in a particular year. There are 62 features, which contain a mix of numerical and categorical attributes, however several of the variables are descriptors of other variables (for example, feature 'race_ethnicity_desc' is the string descriptor of feature 'race_ethnicity_cd').

Preprocessing - Data Cleaning:

Data cleaning began by dealing with the null values. For feature 'gender', we filled in 6 nulls values by using judgement based on feature 'first_name'. We removed 54 instances without location based data and 42 instances without a school or district identified, as we wanted to ensure every instance had a location for future analysis.

Next some columns were dropped that did not have added value. We dropped columns 'tsr_status_cd' and 'tsr_status_desc' as they indicated that all instances were active. For the remaining columns where one column was a string description of another, we went through and dropped the code columns and kept the descriptor columns as categorical variables.

Next we renamed every column for ease of interpretability and incorporation into future visuals (for example, 'schl_st' was renamed to 'School_State').

Preprocessing - Data Filtering:

Data filtering included removing instances where it seemed as though there may have been a data entry error of the survey responses, where an instance was determined to be an outlier, or if an instance did not fit with where we chose to focus our analysis.

7,882 instances where 'Salary' was $0 were removed. These may have been volunteers or respondents that failed to complete this portion of the survey. As part of our objective was to determine which features influenced salary, having instances without a salary seemed counterproductive. 3,501 instances where 'Salary' was less than $20,000 were removed. Based on a comparison with otherwise similar instances, it seemed as though there may have been a data entry error. This was also the case for 5 high outlier salaries.

It was decided early on that the focus would be on Illinois full-time educators and those represented by unions in salary negotiations. 47,617 instances were removed where feature 'Employee_Type' was part-time. Feature 'Employee_Type' was subsequently removed as all remaining instances were full-time. 20,643 instances were removed where 'Months_Employed' was less than 9 months, where the remaining instances represented educators that worked a whole school year (9 months) up to a full calendar year (12 months). 100,147 instances were removed where 'Percent_Administration' was greater than 75%, as it was decided not to include those in an oversight/disciplinarian role, or would have influence themselves in determining educator salaries. Additionally there were 20 instances removed which were from educators located outside of the state of Illinois.

Feature Engineering / Mapping:

Several new features were created from existing features or added using external data. New feature 'Assignments' was added, which indicated how many non-blank assignment related features each instance had (an educator could have two assignments, such as guidance counselor and math teacher in features 'Primary_Assignment' and 'Secondary_Assignment', respectively). Additionally, each educator (which may appear multiple times due to responding to the survey in multiple years) was assigned a unique identifier and each school was assigned a unique identifier, to aid with futures tracking and analysis.

New feature 'Primary_Assignment_Group' was created by recasting over 200 job titles from feature 'Primary_Assignment' into 12 categories. General Education is the largest group.

Primary_Assignment_Group	Instances
Administration	13,008
Business/Trade	28,651
Elective	154,739
Facilities	5,528
Foreign Language	70,074
General Education	457,349
Language Arts	152,971
Librarian	19,702
Social Studies	60,699
Special Educaiton	186,645
STEM	189,824
Studen Support	105,557

Table 1.1: Primary Assignment Groups

Due to differences in the way feature 'Race' was recorded on the survey responses, all races were remapped appropriately (for example, 'multi-racial' and 'multiple' were mapped to 'multiple').

Race	Instances
White	1,228,420
Black	124,591
Hispanic	64,390
Asian or Pacific Islander	16,907
Native American	2,329
Multiple	2,182

Table 1.2: Race distribution

New features 'School_County' and 'District_County' were added to the data using external data and were based on school and district zip code, respectively. There are 102 different counties in Illinois. Additionally, whether or not a county was considered part of a rural or urban area was added to the data with new feature 'Urban_or_Rural'. This classification was taken from US Census data, where a rural county is defined as a county not part of a metropolitan statistical area (MSA) or a county that is part of a MSA but has a population less than 60,000. There are 19 urban counties and 83 rural counties in Illinois.

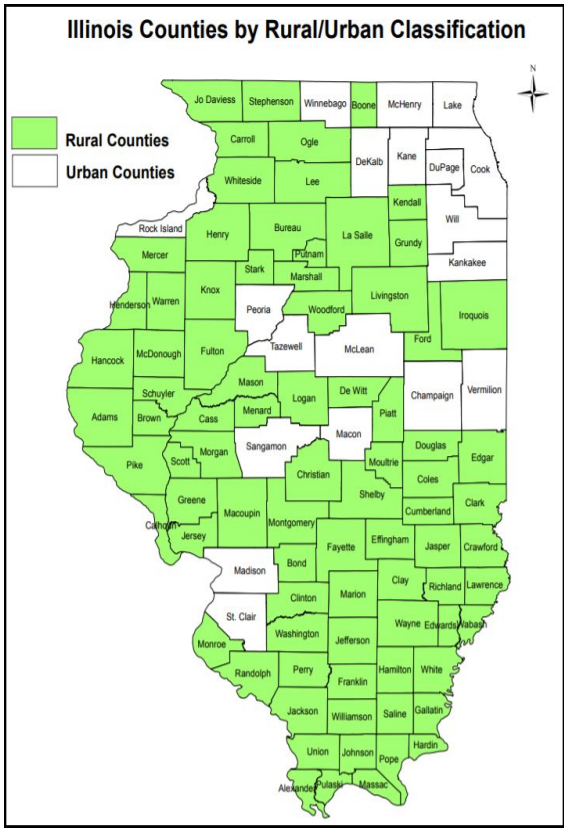


Figure 1.1: Illinois Counties, Urban v. Rural

Final Data:

Our final dataset included 41 features with 1,444,747 instances. Features include ‘Year’, 16 based on the location, and 24 based on the educator.

Year

District
District_Name
District_Address
District_City
District_State
District_Zip
District_County
School_ID
School_Name
School_Address
School_City
School_State
School_Zip
Location_Desc
School_County
Urban_or_Rural

Teacher_ID
Gender
Race
Months_Employed
Percent_Administration
Percent_Time_Employed
District_Experience
State_Experience
Out_of_State_Experience
Undergraduate_College
Highest_Degree_Achieved
Advance_College
Position
Lowest_Grade_Taught
Highest_Grade_Taught
Primary_Assignment
Secondary_Assignment
Third_Assignment
Fourth_Assignment
Fifth_Assignment
Sixth_Assignment
Seventh_Assignment
Primary_Assignment_Group
Assignments

Table 1.3: Cleaned data features

Exploration

Correlation Analysis:

An analysis of the correlation between the features was completed to identify some relationships to further explore. The features with the highest positive correlation to 'Salary' are 'State_Experience' (0.6) and 'District_Experience' (0.57). They are also highly correlated with each other (0.92), which makes sense given that a one year increase in 'District_Experience' leads to a one year increase in 'State_Experience'. Salary is also positively correlated with 'Months_Employed' (0.29) which ranged from 9-12 months and 'Year' (0.23), which ranged from 2003-2012. The features with the lowest negative correlation are 'District_Zip' and 'School_Zip', indicating a relationship with location.

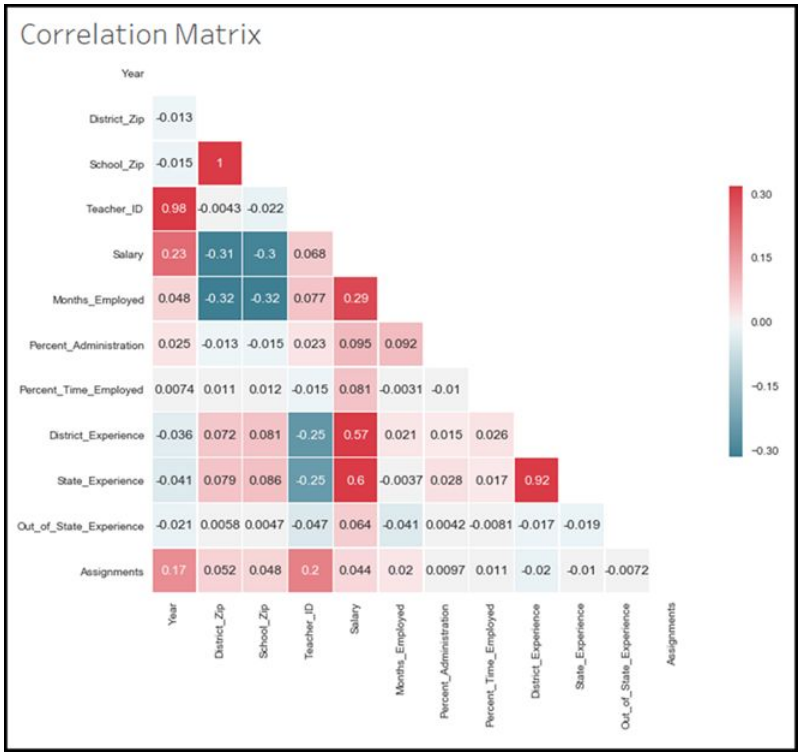


Figure 2.1: Correlations between features of interest and Salary

Salary:

When examining the median educator salary by year in Illinois, we can see wage growth overall between each year, with 2003-2004 and 2007-2008 showing the largest increase by percentage.

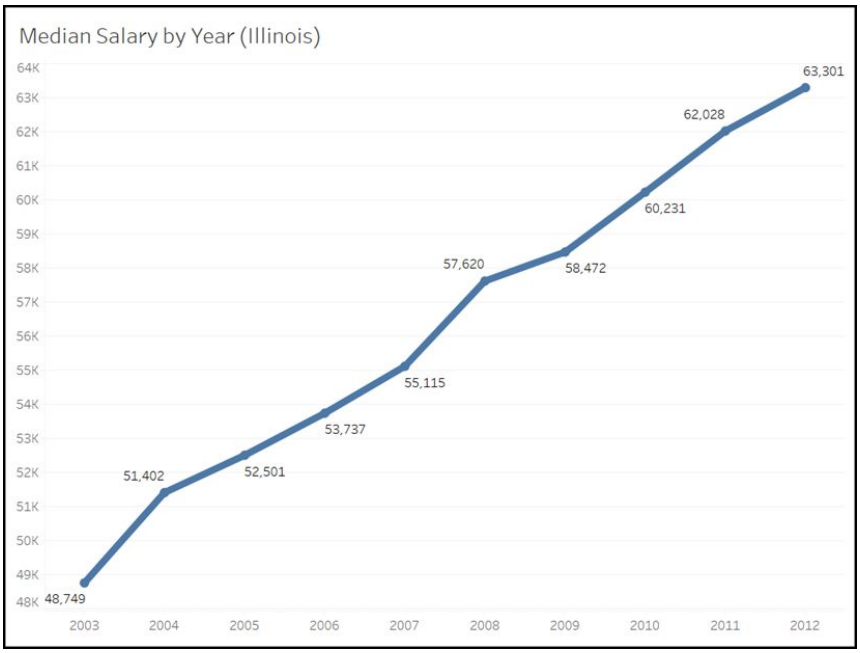


Figure 2.2: Median Illinois Educator Salary from 2003 - 2012

Gender

Looking an an overlaid histogram of feature 'Gender', we can see that females outnumber males by nearly four to one. Additionally, all of the high salaries identified as outliers are female. Salaries for both genders are slightly right skewed.

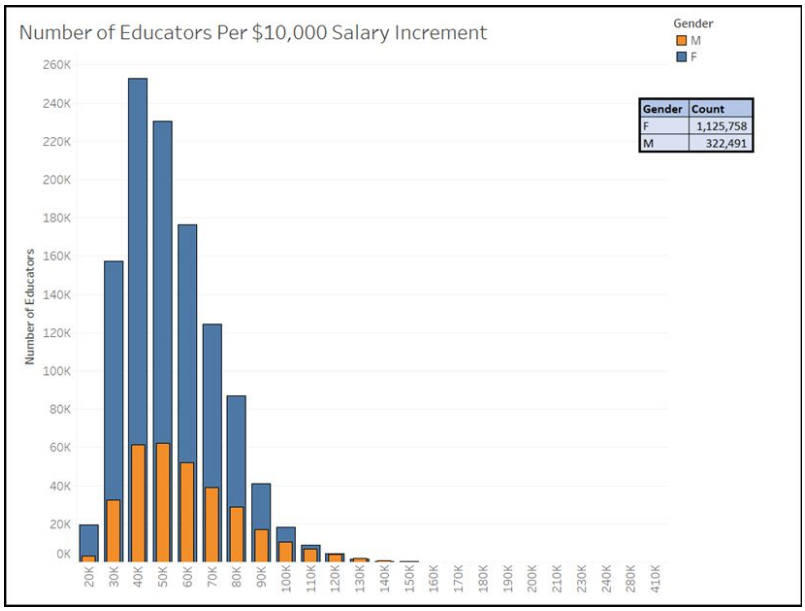


Figure 2.3: Salary breakdown by gender

Primary Assignment Group

The below multiple line graph shows mean salary by year for each of the twelve categories for feature 'Primary_Assignment_Group'. For every year but 2003, Administration had the largest mean salary. For every year, 'General Education', the group with the largest number of instances, had the lowest mean salary.

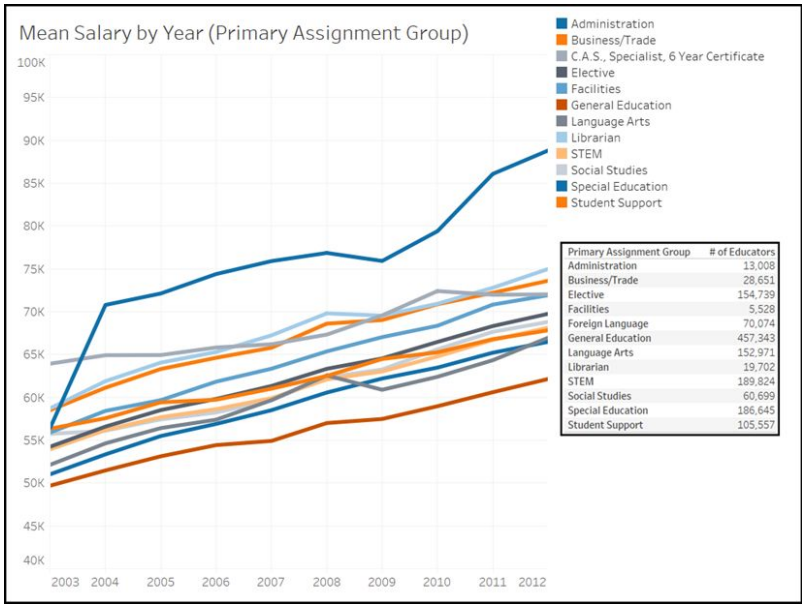


Figure 2.4: Mean salary across Primary Assignment Groups

Location

Taking a closer look at salaries in each county, it is clear that certain areas of the state have higher salaries. The heat map below shows the median salary by county and clearly shows the counties with or near major cities have the highest median salaries. These are also the counties which were identified as urban (in particular, counties in the Chicagoland area, Quad Cities region, and suburban St. Louis are of note). When examining the yearly percentage change of Illinois educator median salaries by county and comparing urban and rural areas, rural counties were twice as likely to see negative median wage growth year over year than urban counties.

Figure 2.5: Median Salary by Illinois County heat map

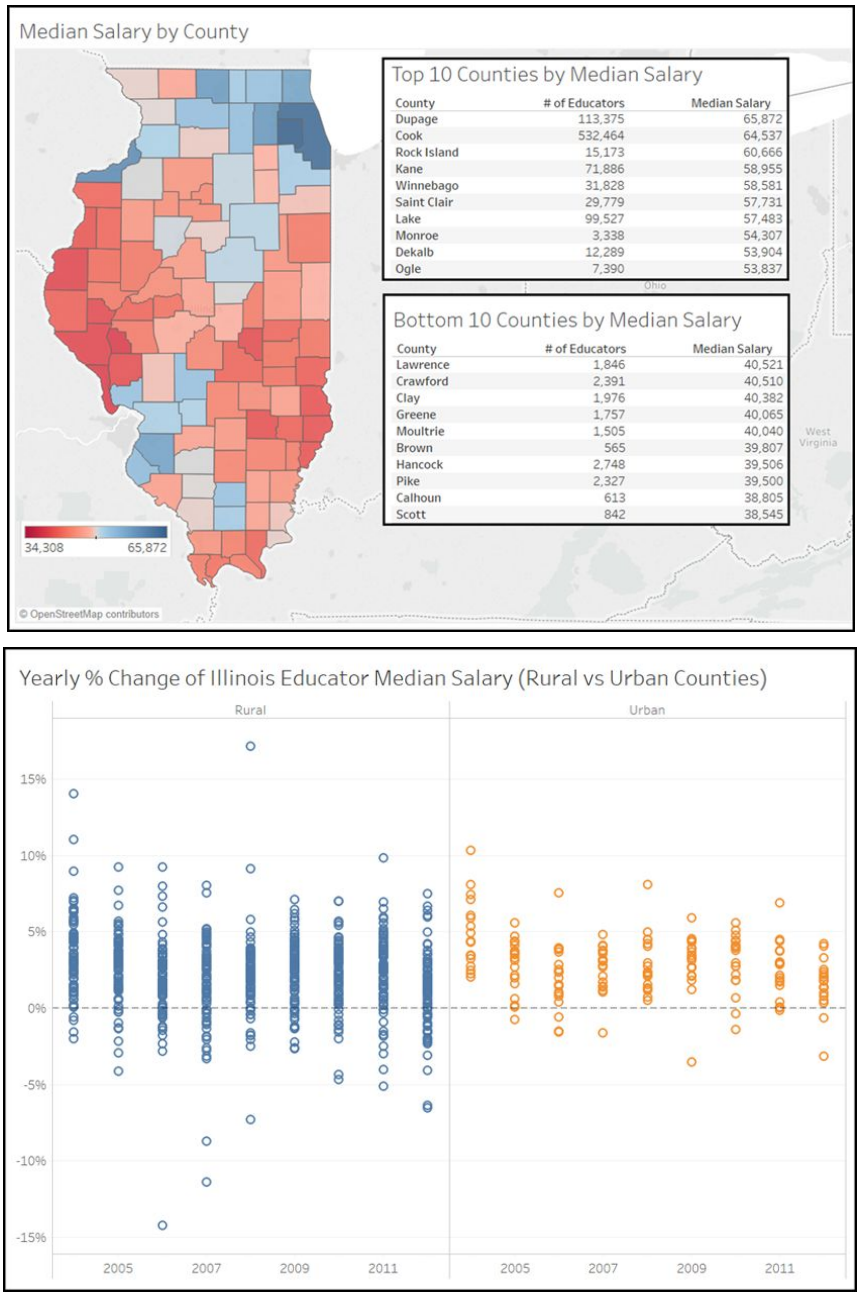


Figure 2.6: Yearly percent change of Illinois Educator Median Salary

II. Methodology & Final Models

Approach

Three main techniques were implemented in order to pinpoint, investigate, and validate the key features that determine an educator's salary in Illinois. Extensive linear modelling was used to pinpoint the key features. Bayesian nets and classification models were implemented to assess the predictive ability of the

key features as well as to validate the key features set. Each of the aforementioned model types required a unique process of tuning and optimization.

Linear Modeling:

The goal of developing a linear model to predict target feature Salary was to determine which features best helped explain the education compensation system in Illinois. The non-geographic features were examined first to test which combinations worked best:

- Year
- Gender
- Race
- Percent_Administration
- District_Experience
- State_Experience
- Out_of_State_Experience
- Highest_Degree_Achieved

The data was split into 70% training and 30% testing. Every combination of the non-geographic features was used and the top twenty combinations by lowest mean squared error on the testing set was identified.

R2	Test_MSE	Train_MSE	Variables
0.56037241	178365879.7	178092618.5	Year,Gender,Race,Percent_Administration,District_Experience,State_Experience, Out_of_State_Experience,Highest_Degree_Achieved_Primary_Assignment_Group
0.558582769	179094799.3	178817600.1	Year,Gender,Race,District_Experience,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved_Primary_Assignment_Group
0.556714734	179838112.1	179574338.9	Year,Gender,Race,Percent_Administration,District_Experience,State_Experience, Highest_Degree_Achieved_Primary_Assignment_Group
0.556464404	179860570.5	179675747.2	Year,Gender,Race,Percent_Administration,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved_Primary_Assignment_Group
0.554769212	180545276.9	180362467.7	Year,Gender,Race,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved_Primary_Assignment_Group

Table 3.1: Top 5 model combinations without geographic features

Next, each of the top twenty combinations was re-ran separately with each of the various geographic features to determine which were most significant:

- District_Zip
- District_County
- Urban_or_Rural
- District_County

District County was the best performing geographic feature.

R2	Test_MSE	Train_MSE	Variables
0.686486064	127694179.7	127004126	Year,Gender,Race,Percent_Administration,District_Experience,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved,Primary_Assignment_Group,District_County
0.686440228	127707242.8	127022694.3	Year,Gender,Percent_Administration,District_Experience,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved,Primary_Assignment_Group,District_County
0.684948388	128334595.3	127627036.5	Year,Gender,Race,District_Experience,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved,Primary_Assignment_Group,District_County
0.684904367	128347860.9	127644869.5	Year,Gender,District_Experience,State_Experience,Out_of_State_Experience,Highest_Degree_Achieved, Primary_Assignment_Group,District_County
0.684540329	128408183.8	127792340.9	Year,Gender,Race,Percent_Administration,State_Experience,Out_of_State_Experience, Highest_Degree_Achieved,Primary_Assignment_Group,District_County

Table 3.2: Top 5 model combinations with geographic features

The features from the best overall model and their coefficients were:

- Year
- Percent_Administration
- State_Experience
- Out_of_State_Experience
- Gender
- Race
- Highest_Degree_Achieved
- Primary_Assignment
- District_County

Relative to the complexity of the data, the R-squared of 0.685 was successful. Full details on the final model can be found in the Appendix.

OLS Regression Results	
Dependent Variable	Salary
Model	OLS
Method	Least Squares
Number of Observations	1011488
DF	130
Covariance Type	Nonrobust
R-squared	0.685
Adjusted R-squared	0.685
F-statistic	1.68E+04
Prob (F-statistic)	0
Log-Likelihood	-1.09E+07
AIC	2.18E+07
BIC	2.18E+07

County	
Variable	Coefficient
Dupage	21650.00
Cook	19720.00
Lake	16520.00
...	...
Pike	-6521.01
Hancock	-7374.58
Brown	-7378.35

Highest Degree Achieved	
Variable	Coefficient
Doctorate	9465.09
C.A.S., Sepecialist, 6 Year Certificate	6030.31
Masters	2671.55
Baccalaureate	-6253.14
None	-8683.33
Registered Nurse	-10160.00

General	
Variable	Coefficient
Year	1584.93
State Experience	1193.28
Out of State Experience	874.64
Percent Administration	235.27
const	-1987000.00

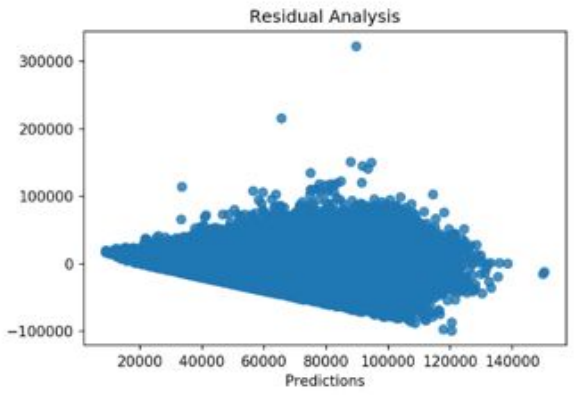
Gender	
Variable	Coefficient
Male	-991500.00
Female	-995700.00

Primary Assignment Group	
Variable	Coefficient
Administration	-161300.00
Facilities	-163200.00
Business/Trade	-164300.00
Student Support	-164900.00
STEM	-165300.00
Elective	-165700.00
Social Studies	-165800.00
Librarian	-166100.00
Language Arts	-166500.00
Foreign Language	-166700.00
Special Education	-168000.00
General Education	-169400.00

Race	
Variable	Coefficient
Native America	-235.30
Hispanic	-589.60
Asian of Pacific Islander	-760.32
White	-957.95
Black	-1101.10
Multiple	-1462.34

Table 3.3: Final model feature statistics

The residuals of the final model show that the higher salaries are, the less standardized the instance is according to our model and therefore harder to predict. The residuals do not pass a normality test, however the density curve appears normal visually. It is clear that a few outliers on the right of the density curve greatly skew the data causing the normality test failure. This does not appear to greatly impact the model, however, so it can be said with confidence that the residuals are normal. For the full model coefficient detail, see the Appendix - [16].



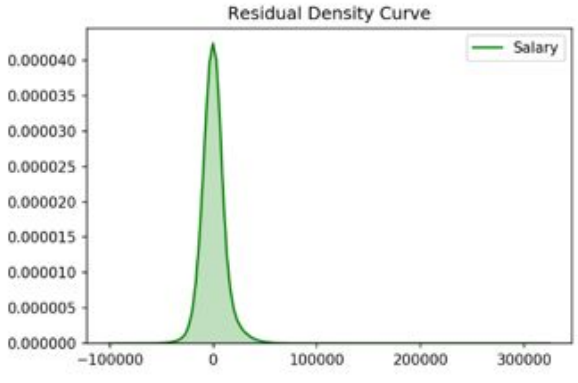


Figure 3.1: Final model residuals

- Positive yearly coefficient shows that year to year we can expect salaries to rise
- This matches the boxplot of the residuals above.
- Median is always roughly around 0

The positive coefficient for feature Year in the final model shows that year to year Salary is expected to rise. This matched the boxplot of the final model residuals. Additionally, the median residual for each year is approximately zero.

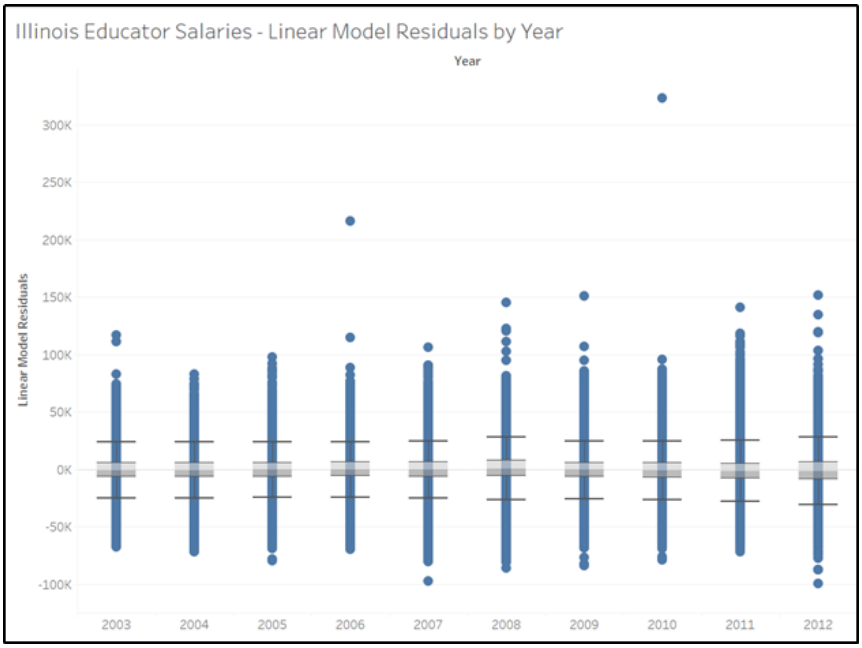


Figure 3.2: Final model residuals broken down by year

When examining the final model residuals by feature Highest Degree Achieved, registered nurses have the most volatile mean from year to year, which could be due to having a low population within other subgroups. Educators without a degree had residuals that were also very volatile.

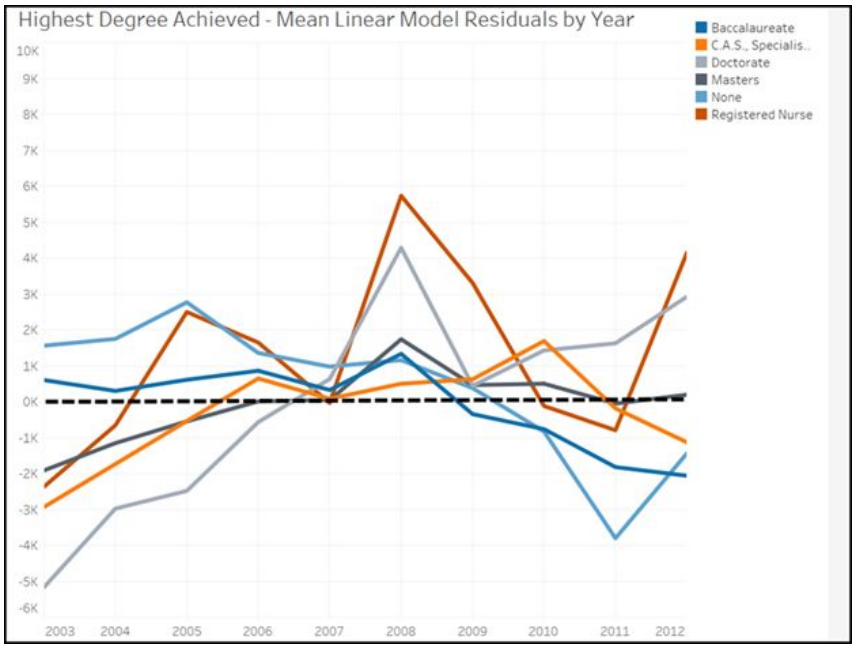


Figure 3.3: Final model residuals by Highest Degree Achieved

When examining the final model residuals by feature Primary Assignment Group, special education was the most volatile. All residuals show a spike in 2008, when our research indicated that the largest annual increases in salary occurred, however the model was able to correct for 2009, when our research showed flat wage growth for most of the state.

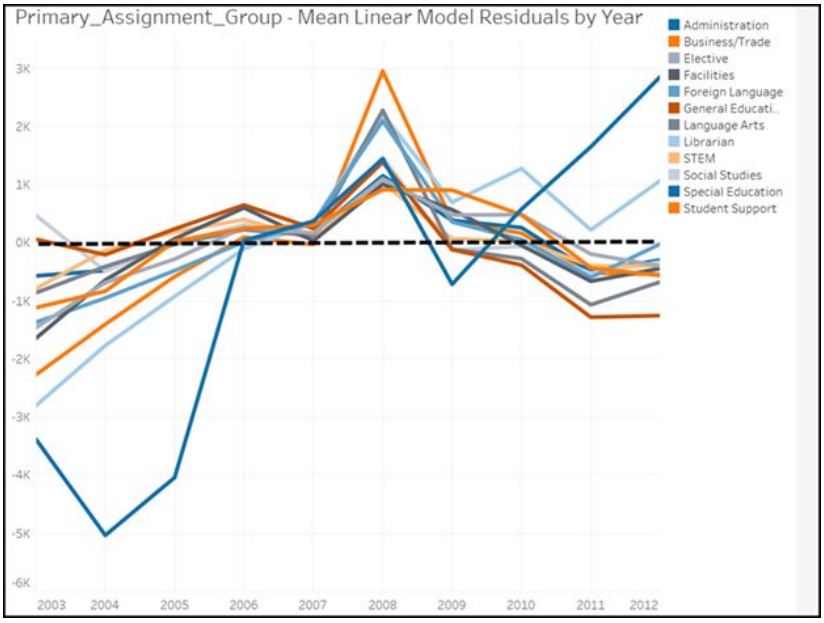


Figure 3.4: Final model residuals by Primary Assignment Group

When examining the final model residuals by feature Gender, the median is typically higher than the mean for both male and female. The median male residual was almost always negative. The model had the most trouble in 2008, which was a common trend see across various spectrums.

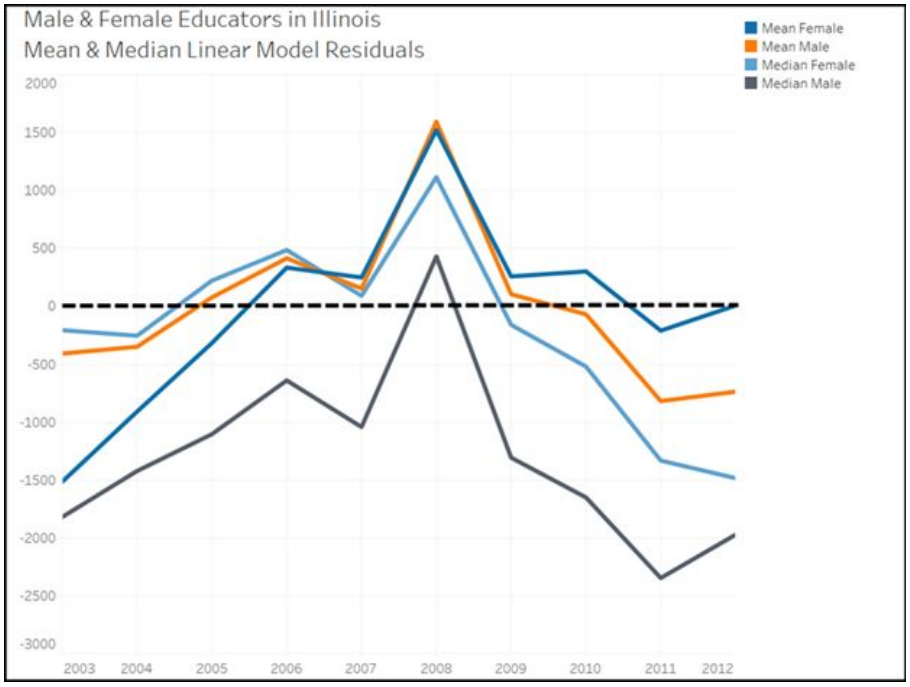


Figure 3.5: Final model residuals by Gender

Bayesian Nets:

The goal of Bayesian networks is to find the conditional dependency between each feature. Based on the Bayesian networks plot, we can see which features have the biggest impact on each other. The features selected were from the output of the linear modeling performed. Prior to the analysis, feature Salary was binned into three levels (low < $65,530, $65,530 < middle < $100,000, high > $100,000), feature Urban_or_Rural was binned into two levels (urban and rural), feature State_Experience (range 0 to 59) was binned into 5 levels, and feature Out_of_State_Experience (range 0 to 42) was binned into 3 levels.

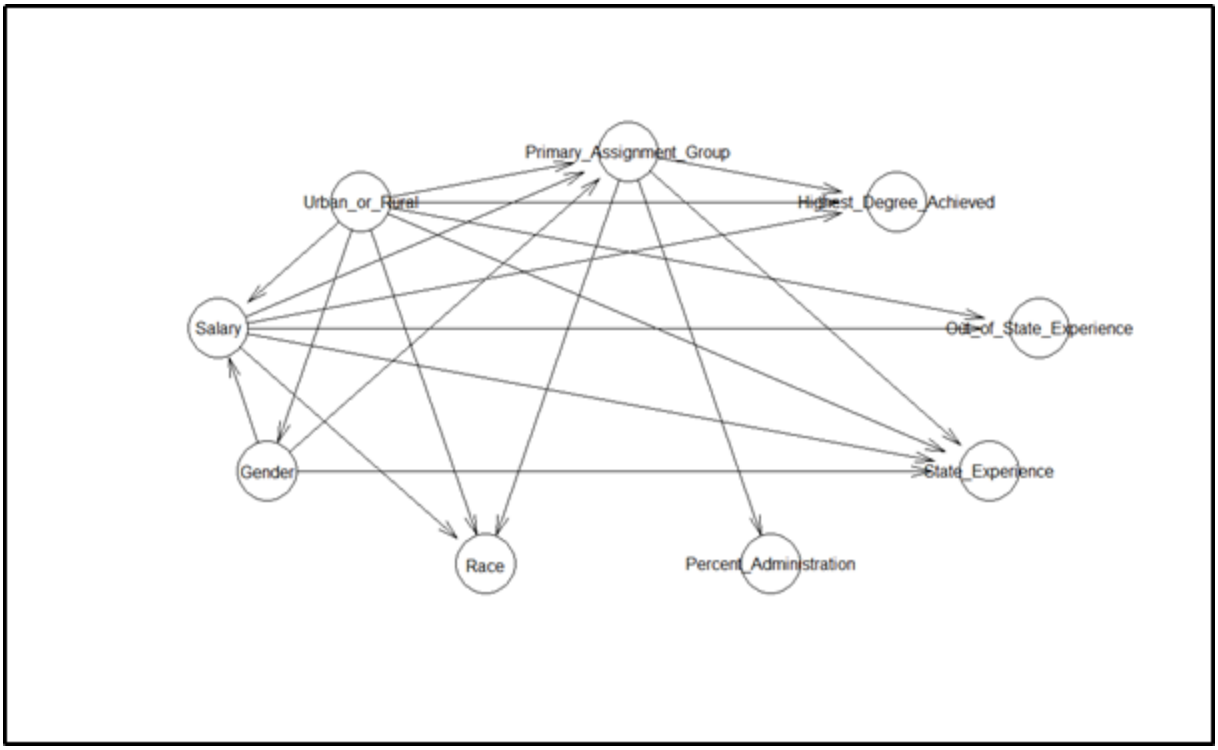


Figure 3.6: Bayesian Net plot

Based on the network plot, we can see that Urban_or_Rural and Gender have a significant effect on Salary. Although we know other variables also affect this, this demonstrates that geographic location and Gender have a significant impact.

T-tests were performed, and there is a statistically significant difference between the mean salary of each gender. Additionally, there is also a statistically significant difference between mean salary of urban and rural educators, and also between male and female educators in both urban and rural settings. This validates the findings of the linear model.

Gender Salary t-Test Results	
Female Mean Salary	58740.16162
Male Mean Salry	64463.31366
t-statistic	-130.5246133
p-value	0

Urban/Rural Salary t-Test Results	
Urban Mean Salary	61357.19119
Rural Mean Salry	48681.63721
t-statistic	338.7209309
p-value	0

Average Salary by Subgroup		
	Urban	Rural
Female	59391.39	47813.16
Male	67998.94	51408.69

Table 3.4: T- Test results and Salary Statistics from Bayesian Net

Further analysis was performed to examine the distribution of salary bins (low, medium, high) across region and gender. Virtually no high salaries were located in rural areas. High salaries were especially prominent in urban areas, and despite the fact that females outnumber males by nearly a four to one ratio, males have approximately the same amount of high salaries.

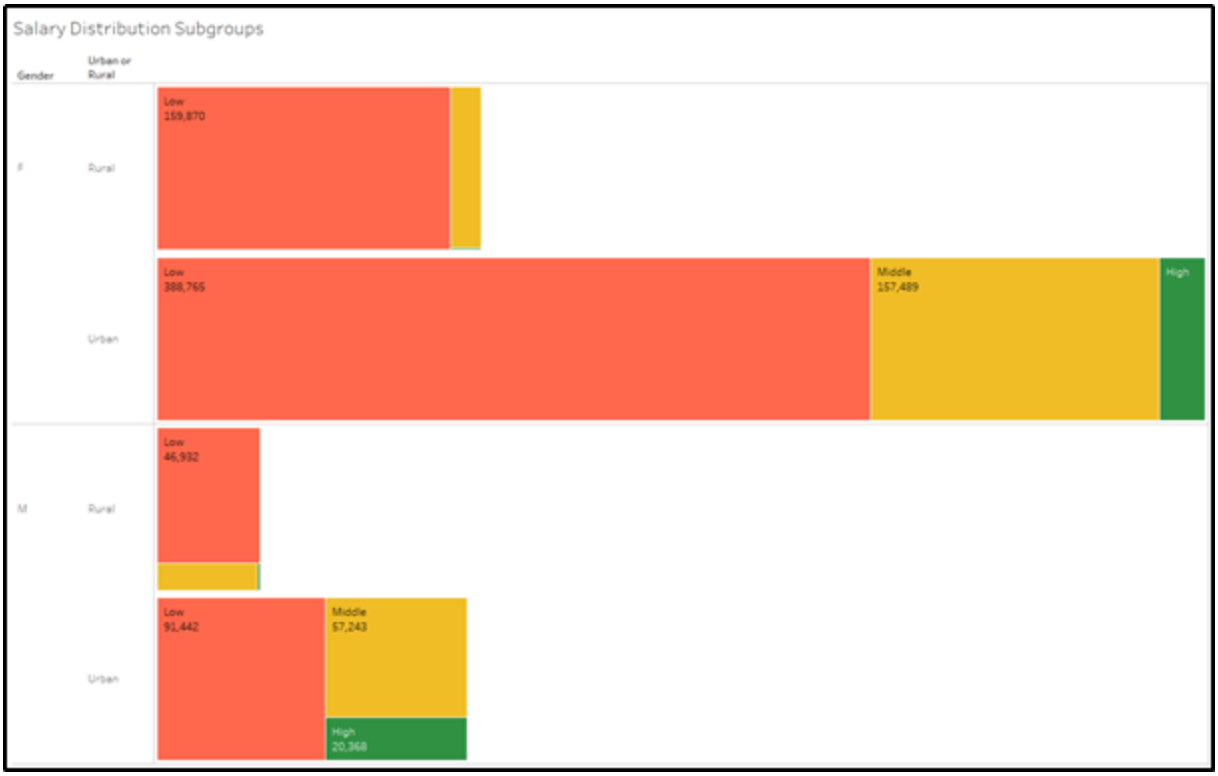


Figure 3.7: Examination of Salary bins across region and gender

Classification Models:

As the goal of the analysis of Illinois teacher compensation is primarily analytic as opposed to predictive, the classification models were largely used to validate the final set of influential features produced in the linear modelling. The process for producing the final classification models was heavily reliant upon the computational cost of the model, the ability to effectively and accurately process high dimensionality, and explainability. The original data contained only numeric salary designations. Thus, Salary classes were created using k-means clustering. Two sets of binnings were attempted: 2 class split and 3 class split. The specific salary cut-offs for each bin can be seen in Table 3.5.

	2 Classes	3 Classes
Class 0	**Less than or equal to** $65,530.92	**Less than or equal to** $65,530.92
Class 1	**Greater than** $65,530.92	**Greater than:** $65,530.92 **Less than or equal to:** $99,999.99
Class 2	n/a	**Greater than:** $100,000.00

Table 3.5: Binning/Class assignment values for target variable: Salary

The features '2Cluster' and '3Cluster' designate class assignment and were added to the data as target variables where applicable.

The final models were run on both non-feature selected and feature selected data across both the 2-Class and 3-Class problems in order to capture any change in performance. The non-feature selected data contained approximately 323 features while the feature selected data contained 134. The final models include: a decision tree classifier, Random forest classifier, and Support Vector Machine. All classification modelling was done using the SciKit Learn machine learning package in Python (Pedregosa et al, 2011).

Parameter Tuning & Optimization

Balancing: Due to the skewness of the salary distribution and the class imbalance present in the target salary bins, it was necessary to balance the data. A few techniques for balancing the data were researched and considered. For example, the innovative 'SMOTE' technique was considered (Bowyer, Chawla, Kegelmeyer, 2002). However, due to the large quantity of data present in our set, basic undersampling of the majority class(es) was implemented. All majority classes were balanced to equal the number of instances present in the minority class, leaving the 2 Class instance count at 491,129 each class and the 3 Class instance count at 59,835 each class. A split of 65% training and 35% testing was used for all of the following.

Decision Tree Classifier: The optimal parameters for the decision tree were gathered through a comprehensive grid search. For all models, the optimal splitting criterion is 'gini', number of maximum features is none, and the minimum samples upon which to split is 100.

Random Forest Classifier: The optimal parameters for the Random Forest Classifier were determined through a combination of a grid search and error plotting as Random Forest Classifiers are prone to overfitting and over-prediction of the majority class (Chen, Liaw, Breiman, 2004). The most difficult parameter to tune for the Random Forest Classifier is the number of estimators (or trees) used in prediction. Thus, models with the prefered parameters were run over 19 different numbers of estimators. The errors for each number of estimators was plotted to find the 'knee' (the bend before error levels off). An example of one of these plots can be seen in Figure 3.8.

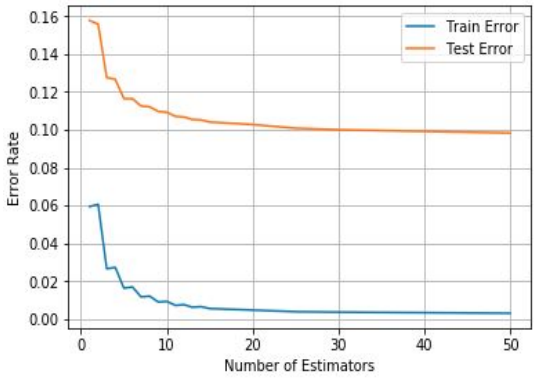


Figure 3.8: Finding the optimal estimators for non-feature selected 2-Class Random Forest.

The optimal number of estimators for the full-featured data (both 2-Class and 3-Class) was 10. The optimal number of estimators for the feature selected data 2-Class and 3-Class problems were 10, and 15 respectively.

Support Vector Machine: The SVM model was only run on the 3-Class problem due to computational cost. The main motivation for attempting the SVM model was to analyze the support vectors for patterns in misclassification. Parameter tuning was done through a grid search on kernel, gamma value, and penalty parameter. The final models used an 'rbf' kernel, gamma value of 1/number of features, and penalty parameter of 1.0.

Final Models

Overall, the results of the classification models on the full featured data versus the feature selected data showed minimal loss in accuracy, precision, and recall across all models. That being said, each model presented its own unique insight about the feature set. Unsurprisingly, results on the 3-Class problem were lower than that of the 2-Class for each model.

Decision Tree Classifier: Although the decision tree classifier is a fairly basic model, it yielded impressive accuracy, precision, and recall scores as well as highly explainable results. With all of features present, the decision tree model produced an overall accuracy on the 2-Class problem of 88%, with precision and recall scores for both classes averaging around 88% and 87% respectively. The results using the significantly reduced, feature selected data showed minimal decreases in accuracy (85%), precision (avg. 85%), and recall (avg. 85%). The full results breakdown for the final decision tree can be found in the Appendix - [15].

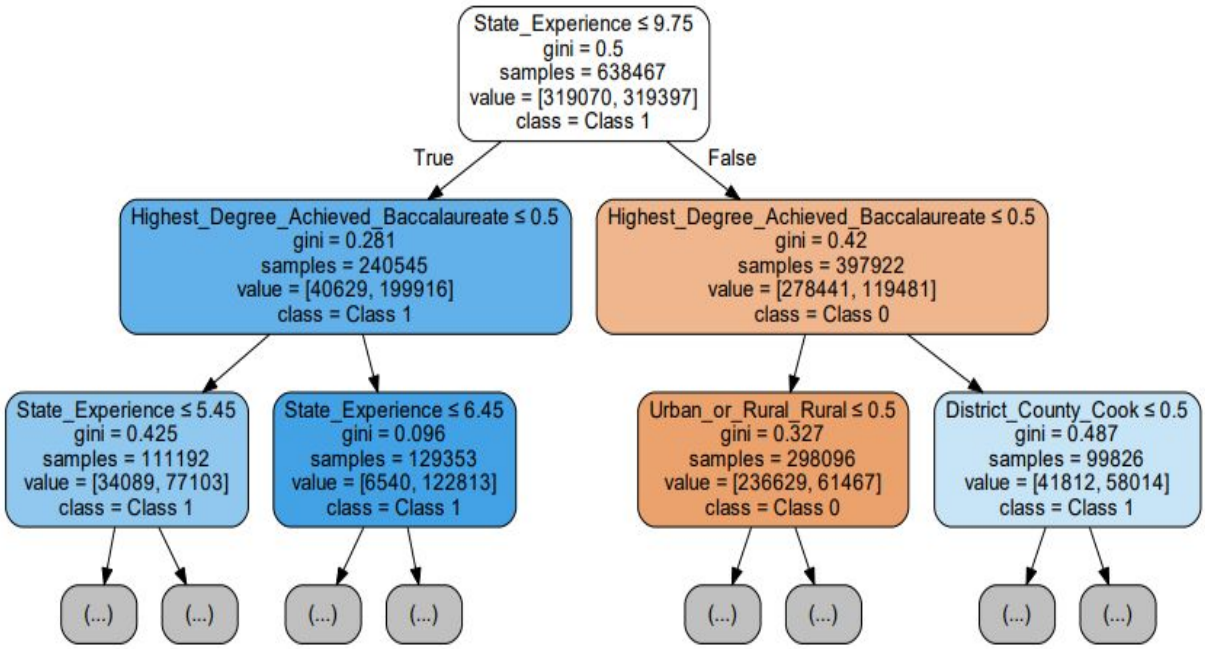


Figure 3.9: The first three levels of splits from the 2-Class decision tree classifier run on full featured data

The results on the feature selected data show a minimal reduction in accuracy, precision, and recall of between two to three percent. Validation that our feature selections are adequate to predict salary class is not the only thing learned from the decision tree model. Figure 3.9 shows that the first three splits of the 2-Class model run on full featured data used only features present in our final set of key features.

Random Forest Classifier: Results for the Random Forest classifier on both the two and three Class problems largely followed those of the base decision tree. There was only a slight reduction (approximately three percent) of accuracy, precision, and recall from full featured data model to the feature selected model. For the full comparison of results on the Random Forest Classifiers, see the Appendix - [15].

Support Vector Machine: As has been previously discussed, the SVM model was only run on the 3-Class problem due to computational cost. In terms of accuracy, precision, and recall on the three class problem, the support vector machine saw the best results. However, these results were within a few percentage points of both the decision tree and random forest models, making those few percentage points not worth the computational cost associated with running the model. That being said, the main goal of running SVM was to analyze the support vectors, or hard to classify instances. A thorough analysis showed a few key insights:

- Less than four percent of the support vectors included teachers with any administrative duties. We have come to find that administrators within the data set show much higher salaries than other

positions. Thus, it would make sense that the hard to classify instances rarely included anyone with administrative duties.

- As can be seen in Figure 3.10, the majority of the support vectors were educators with moderate to high levels of in state experience. Further analysis showed the majority of the support vectors were instances of female educators. The aforementioned trend leads one to believe that there may be insufficient experience based pay increases for women in certain districts.

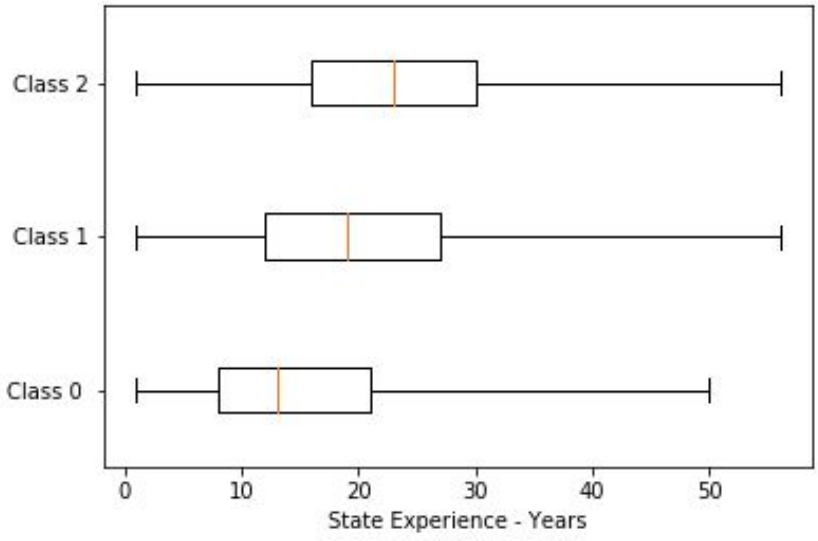


Figure 3.10: Support vector state experience distributions by class

III. Conclusions, Fairness Assessment, Recommendations, & Future Work

Influential Features

Through our exploratory analysis, modeling, and validation, the following features are the most significant in determining compensation:

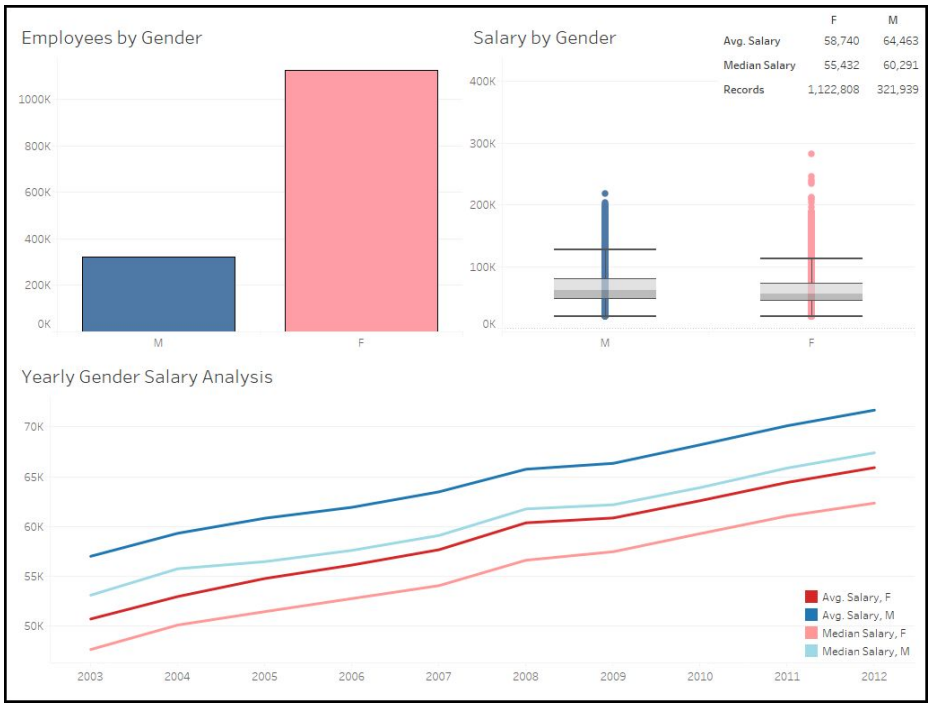
- Year
- Percent_Administration
- State_Experience
- Out_of_State_Experience
- Gender
- Race
- Highest_Degree_Acheived
- Primary_Assignment_Group
- County
- Urban_or_Rual

Fairness Assessment

An in depth examination of many of the influential features was performed to assess the fairness of the educator compensation system in the state of Illinois:

Gender:

An examination of Gender shows that females outnumber males at nearly a four to one ratio. When looking at the distribution of salaries, the top end of the range of female salaries is higher than male salaries, however the interquartile range of male salaries is larger. For each year in our data, males have a higher median and mean salary than females. The differences range from 11% to 7.2%.



Year	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Median Salary Disparity	10.3%	10.1%	8.9%	8.4%	8.5%	8.3%	7.6%	7.2%	7.3%	7.5%
Mean_Salary_Disparity	11.0%	10.7%	10.0%	9.4%	9.2%	8.2%	8.3%	8.2%	8.1%	8.1%

Figure 4.1: Gender statistics breakdown

Looking at the roles each Gender has in the education system, females are more likely to be in general education than any other primary assignment group. Males are more likely to in STEM and elective roles than any other primary assignment group.

When comparing the earnings between males and females in each primary assignment group, males have a higher median salary across every category, with the difference ranging from $1,500 (Social Studies) to $8,000 (Student Support).

Gender and Primary Assignment

	Admin	Business/ Trade	Elective	Facilities	Foreign Language	General Education	Language Arts	Librarian	Social Studies	Special Education	STEM	Student Support	Grand Total
F	0.79%	1.29%	7.50%	0.31%	5.20%	37.28%	11.20%	1.59%	2.22%	14.62%	10.19%	7.81%	100.00%
M	1.30%	4.40%	21.92%	0.65%	3.63%	12.03%	8.45%	0.57%	11.11%	6.98%	23.41%	5.55%	100.00%
Grand Total	0.90%	1.98%	10.71%	0.38%	4.85%	31.66%	10.59%	1.36%	4.20%	12.92%	13.14%	7.31%	100.00%

Median Salary

	Admin	Business/ Trade	Elective	Facilities	Foreign Language	General Education	Language Arts	Librarian	Social Studies	Special Education	STEM	Student Support	Grand Total
F	72,708	55,592	55,955	59,446	57,203	52,401	55,735	64,538	57,520	56,556	54,711	63,839	55,432
M	77,173	59,128	60,553	64,041	59,045	53,645	60,259	69,041	59,165	60,833	60,745	71,450	60,291
Grand Total	74,148	57,270	57,934	61,124	57,486	52,508	56,488	64,842	58,473	57,036	56,965	65,141	56,445

Table 4.1: Gender and Primary Assignment Groups v. Median Salary

Race:

An examination of Race shows that white educators make up the majority of all primary assignment groups, which makes sense as they make up 85% of all educators. White educators have the lowest median salary, and multiracial educators have the highest. When looking at primary assignment groups within each race, the most common for each race is general education, with the exception of hispanic educators, which are most likely in foreign language. Asian educators are most likely to be in STEM after general education.

Primary Assignment by Race

	Admin	Business/ Trade	Elective	Facilities	Foreign Language	General Education	Language Arts	Librarian	Social Studies	Special Education	STEM	Student Support	Grand Total
Multiple	0.32%	0.09%	0.12%	0.25%	0.40%	0.11%	0.13%	0.13%	0.20%	0.16%	0.13%	0.21%	0.15%
Black	15.11%	6.20%	6.09%	6.80%	1.96%	10.33%	8.69%	8.42%	7.84%	8.72%	5.92%	14.85%	8.66%
Native American	0.18%	0.16%	0.11%	0.05%	0.34%	0.16%	0.14%	0.25%	0.13%	0.11%	0.21%	0.17%	0.16%
Hispanic	6.50%	1.25%	1.97%	1.71%	34.30%	3.88%	2.44%	1.48%	2.82%	1.98%	1.95%	5.18%	4.48%
Asian or Pacific Islander	0.85%	0.50%	0.73%	0.53%	2.58%	1.02%	0.96%	0.81%	1.04%	0.81%	2.01%	1.40%	1.18%
White	77.04%	91.80%	90.98%	90.66%	60.42%	84.48%	87.63%	88.91%	87.96%	88.22%	89.77%	78.19%	85.38%
Grand Total	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Race by Primary Assignment

	Admin	Business/ Trade	Elective	Facilities	Foreign Language	General Education	Language Arts	Librarian	Social Studies	Special Education	STEM	Student Support	Grand Total
Multiple	1.88%	1.19%	8.20%	0.64%	12.74%	23.69%	9.30%	1.15%	5.45%	13.79%	11.69%	10.27%	100.00%
Black	1.57%	1.42%	7.54%	0.30%	1.09%	37.85%	10.62%	1.33%	3.80%	13.00%	8.98%	12.50%	100.00%
Native American	0.99%	1.98%	7.13%	0.13%	10.05%	31.39%	9.19%	2.10%	3.43%	9.10%	16.87%	7.64%	100.00%
Hispanic	1.30%	0.55%	4.71%	0.15%	37.01%	27.53%	5.77%	0.45%	2.65%	5.70%	5.73%	8.43%	100.00%
Asian or Pacific Islander	0.65%	0.84%	6.70%	0.17%	10.62%	27.66%	8.67%	0.94%	3.70%	8.89%	22.46%	8.69%	100.00%
White	0.81%	2.13%	11.42%	0.41%	3.42%	31.38%	10.86%	1.42%	4.32%	13.34%	13.81%	6.68%	100.00%
Grand Total	0.90%	1.98%	10.72%	0.38%	4.83%	31.72%	10.58%	1.36%	4.20%	12.91%	13.13%	7.29%	100.00%

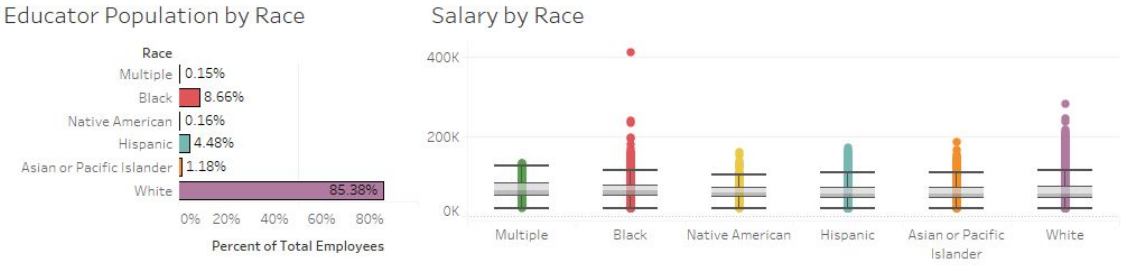


Figure 4.2: Primary Assignment and Race statistics

Primary Assignment Group:

An examination of Primary Assignment Group shows that educators in administration have the highest salaries, and educators in general education make the least. Typically (with the exception of facilities), the less educators in a particular assignment group, the higher the salary of that group.

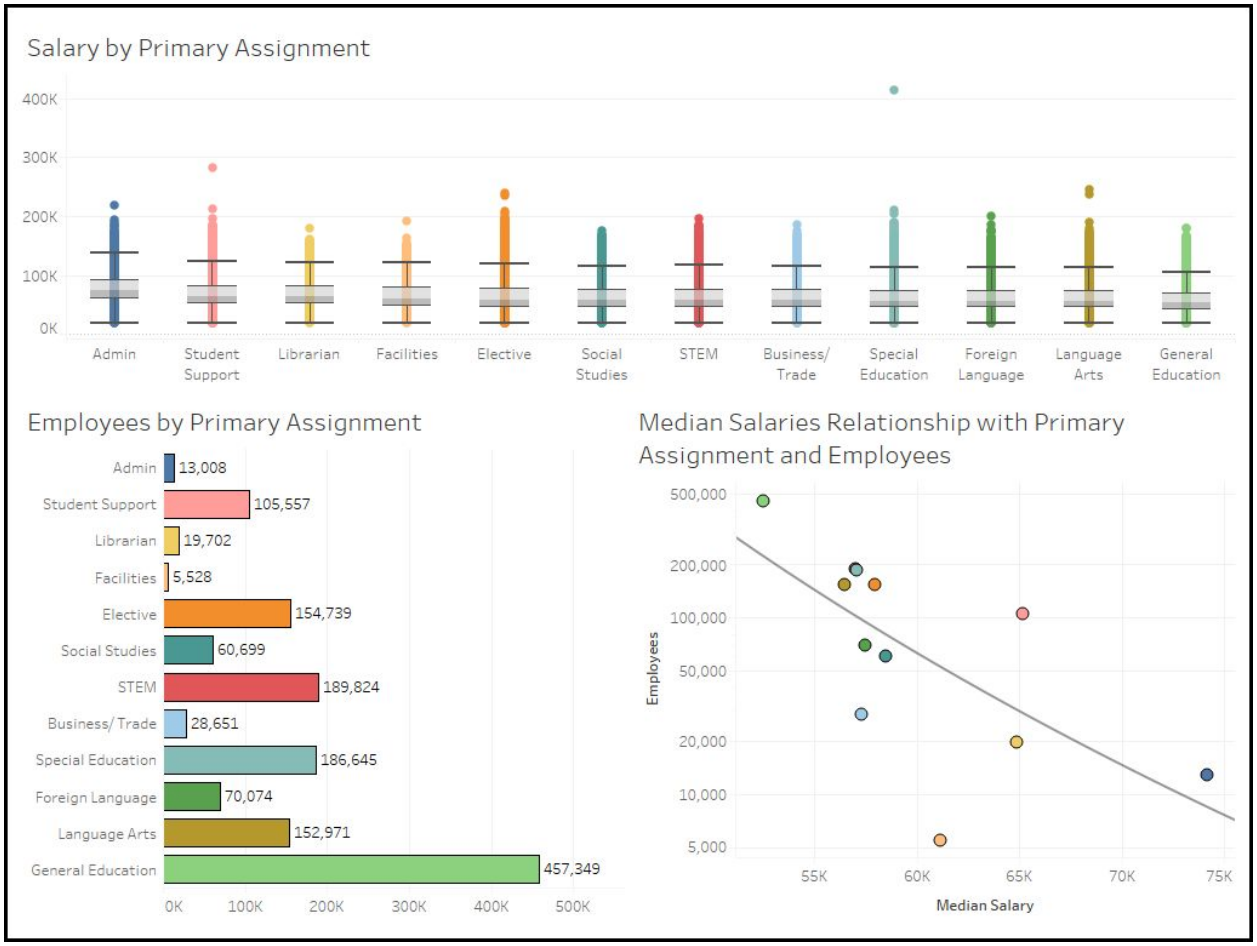


Figure 4.3: Primary Assignment Breakdown

Location:

An examination of location-based features shows that urban areas typically have a much higher median county salary than rural areas. When looking at the linear model residuals of each county, the distribution of salaries within each county varies greatly. In the Chicagoland area counties, where median salary is the highest, the median residuals were negative, indicating the model actually over predicted these salaries. These negative residuals have a right skew, showing that there are many higher level or outlier salaries in these counties, and that even within those counties, the salaries may not be fairly distributed.

Figure 4.4: Median Salary & Median Residual heat maps by county

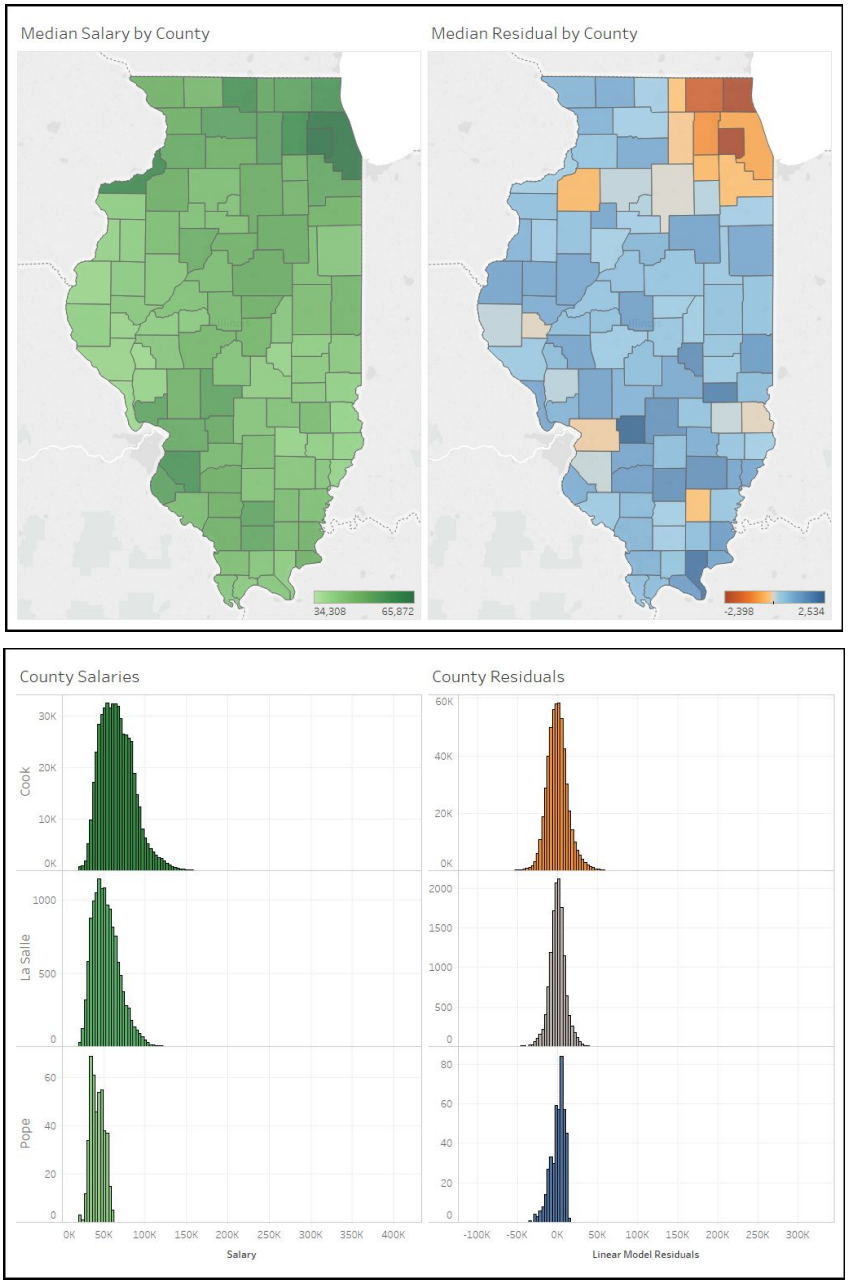
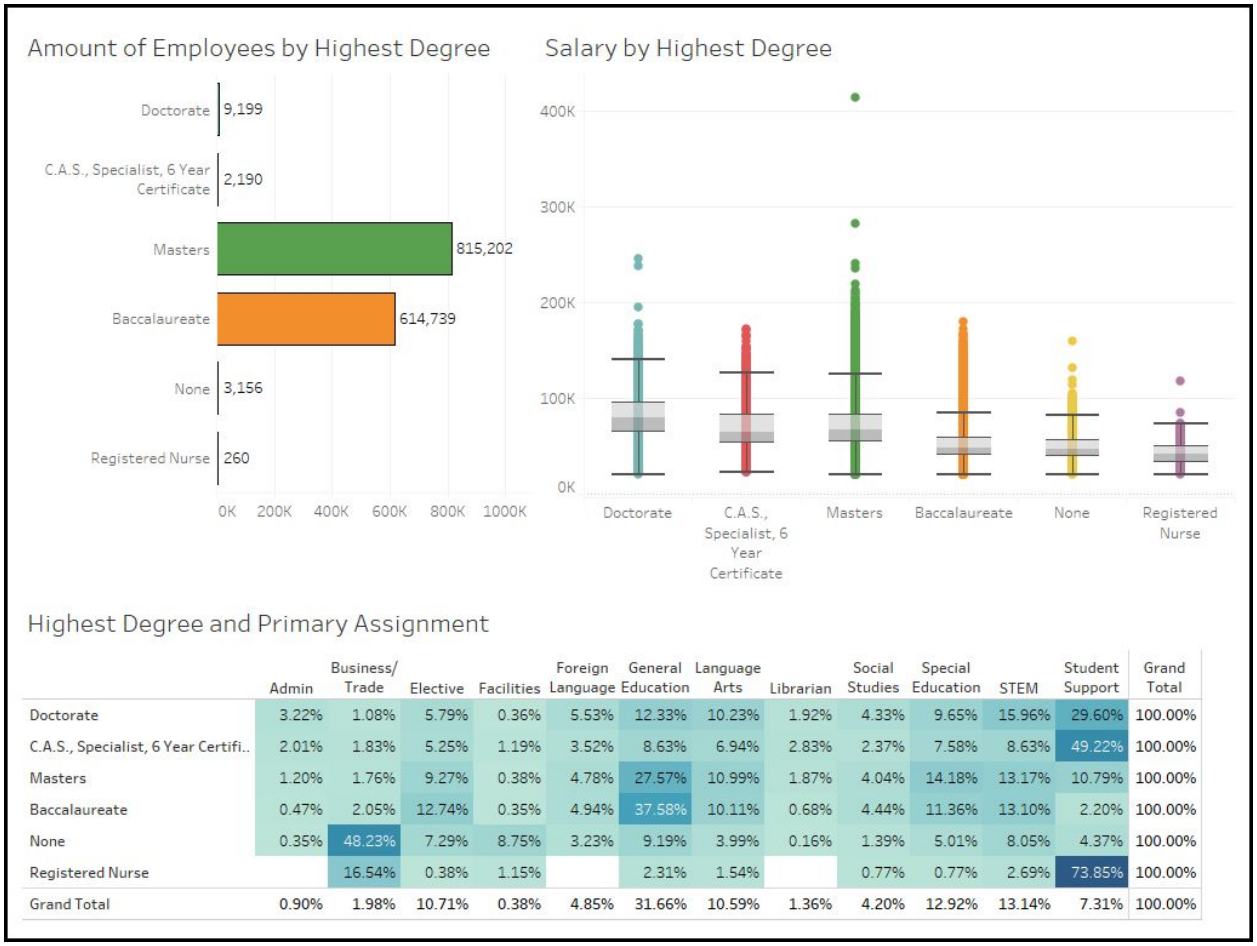


Figure 4.5: County Salary & County Residual distributions

Highest Degree Achieved:

An examination of Highest Degree Achieved shows that the more advanced a degree is, the higher it typically pays. Registered nurses are the exception, however this may be due to factors outside our dataset. Educators with a masters degree have the highest population, and the primary assignment group which typically has the highest degree is administration. General education is made up of mostly educators with masters and baccalaureate degrees, and business/trade is primarily made up of educators without a degree.



Highest Degree and Primary Assignment

	Admin	Business/ Trade	Elective	Facilities	Foreign Language	General Education	Language Arts	Librarian	Social Studies	Special Education	STEM	Student Support	Grand Total
Doctorate	3.22%	1.08%	5.79%	0.36%	5.53%	12.33%	10.23%	1.92%	4.33%	9.65%	15.96%	29.60%	100.00%
C.A.S., Specialist, 6 Year Certifi..	2.01%	1.83%	5.25%	1.19%	3.52%	8.63%	6.94%	2.83%	2.37%	7.58%	8.63%	49.22%	100.00%
Masters	1.20%	1.76%	9.27%	0.38%	4.78%	27.57%	10.99%	1.87%	4.04%	14.18%	13.17%	10.79%	100.00%
Baccalaureate	0.47%	2.05%	12.74%	0.35%	4.94%	37.58%	10.11%	0.68%	4.44%	11.36%	13.10%	2.20%	100.00%
None	0.35%	48.23%	7.29%	8.75%	3.23%	9.19%	3.99%	0.16%	1.39%	5.01%	8.05%	4.37%	100.00%
Registered Nurse		16.54%	0.38%	1.15%		2.31%	1.54%		0.77%	0.77%	2.69%	73.85%	100.00%
Grand Total	0.90%	1.98%	10.71%	0.38%	4.85%	31.66%	10.59%	1.36%	4.20%	12.92%	13.14%	7.31%	100.00%

Figure 4.6: Break down and statistics of Highest Degree Achieved

Experience:

An examination of the features related to experience shows that both in state and out of state experience impact salary, however in state experience improves salary at a much fast rate. Salary is highly impacted by in state experience.

Figure 4.7: Trends of State/Out-of-State Experience and Salary

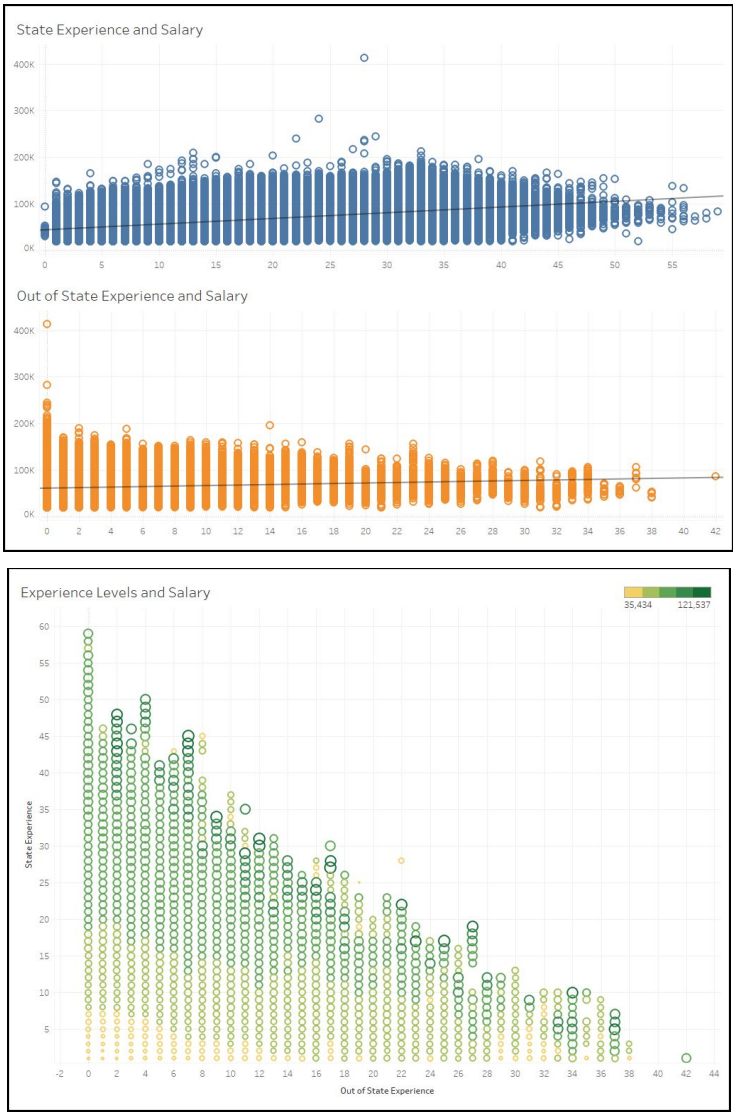
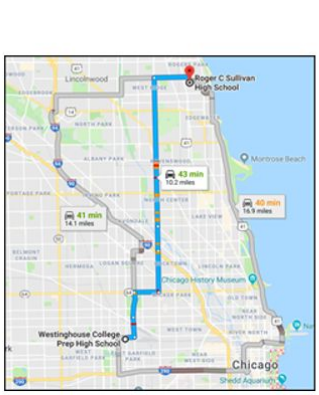
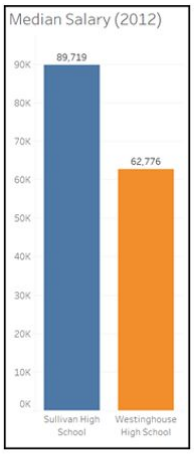


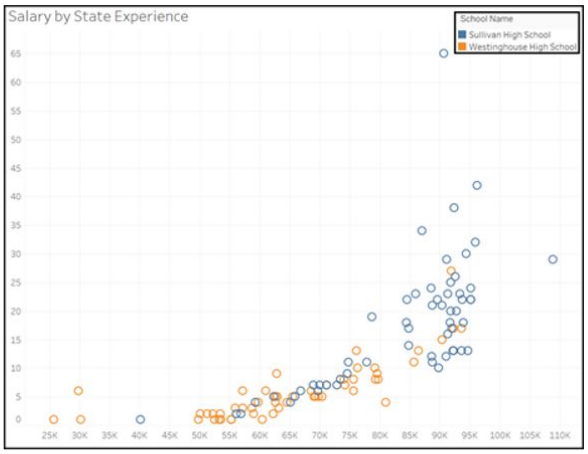
Figure 4.8: Experience levels and Salary

School Comparison:

Two Chicago inner city schools were compared to identify the factors impacting the differences in the compensation of the educators at each. Sullivan High School in Rogers Park and Westinghouse High School near the United Center (west of downtown) are ten miles apart. The median salary was $89,719 at Sullivan High School and $62,776 at Westinghouse High School in 2012, a very large difference. The only significant difference between the two schools was that the educators at Sullivan High School have much more years of in state experience. Additionally, performance data from Chicago Public Schools during 2012 indicated that Sullivan High School was on probation and had far lower teacher scores than Westinghouse High School. In state experience seems to have a much larger impact on salary than it should.







Name of School	Adequate Yearly Progress Made?	CPS Performance Policy Status	Family Involvement Icon	Family Involvement Score	Environment Score	Instruction Score	Leaders Score	Teachers Score	Parent Engagement Score	Parent Environment Score
George Westinghouse High School	N/A	Not on Probation	Strong	72	77	80	86	71	54	60
Roger C Sullivan High School	No	Probation	Average	44	34	34	42	33	44	45

Figure 4.9: Comparison of Sullivan High School and George Westinghouse High School by teacher statistics

Final Compensation Analysis

Several features have been identified which impact educator compensation. Some seem fair, such as experience, highest degree achieved, and location (could be related to cost of living). With others, fairness is uncertain, such as primary assignment group, where compensation based on role is more subjective (should a second grade teacher be paid more than a high school math teacher?). Unfairness is clear in the compensation discrepancy between the genders and both the compensation and representation amongst the races.

Recommendations

The most difficult instances to classify were those on the higher end of the salary spectrum. It is clear that the process for determining salary relies too heavily on in state experience, and doesn't take any performance based measures into account. A poor performing teacher with many years experience shouldn't make significantly more than a high performing teacher with less experience. The structure used to calculate pay in this public sector is vastly different than that of the private sector. Certain roles should have caps and pay increases beyond inflationary raises should be based on performance. Additionally, the education system should explore the circumstances surrounding the gender pay gap, and look for ways to ensure more fairness in compensation across primary assignments groups.

Future Work

For future work, we would like to perform a separate analysis on the administrators we removed from the dataset, perhaps separately and then incorporated back into our models. We would also like to look at where the most part time workers are located, to determine if their deployment is more prominent in

certain areas. Finally, we would like to conduct a study on what fairness means to people, perhaps via survey, as what seems fair to one person may not seem fair to another. More objective measures could be developed to gain consensus on many of the issues with the education compensation system in Illinois.

Appendix

[1] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[2] Bowyer, K., Chawla, N., Hall, L., Kegelmeyer, W. (2002, June). SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research, 16,* 321-357.

[3] Chen, C., Liaw, A., Breiman, L. (2004, July). *Using Random Forest to Learn Imbalanced Data.* Retrieved from: https://statistics.berkeley.edu/tech-reports/666

[4] Chicago Public Schools - Progress Report Cards (2011-2012). Retrieved from: https://data.cityofchicago.org/Education/Chicago-Public-Schools-Progress-Report-Cards-2011-/9xs2-f89t

[5] Wild, C., Pfannkuch, M. (1999, December). Statistical Thinking in Empirical Enquiry. *International Statistical Review, 67,* 223-248.

[6] Decision Tree Classifier Final Results Table

Metrics	No Feature Selection 2-Class	Feature Selection 2-Class	No Feature Selection 3-Class	Feature Selection 3-Class
Accuracy	0.88	0.85	0.79	0.75
Precision	**0:** 0.87 **1:** 0.88	**0:** 0.83 **1:** 0.87	**0:** 0.85 **1:** 0.69 **2:** 0.83	**0:** 0.84 **1:** 0.64 **2:** 0.76
Recall	**0:** 0.89 **1:** 0.86	**0:** 0.88 **1:** 0.83	**0:** 0.84 **1:** 0.70 **2:** 0.83	**0:** 0.81 **1:** 0.61 **2:** 0.81

[7] Random Forest Classifier Results Table

Metrics	No Feature Selection 2-Class	Feature Selection 2-Class	No Feature Selection 3-Class	Feature Selection 3-Class
Accuracy	0.89	0.85	0.81	0.74
Precision	**0:** 0.87 **1:** 0.91	**0:** 0.84 **1:** 0.86	**0:** 0.85 **1:** 0.72 **2:** 0.86	**0:** 0.83 **1:** 0.63 **2:** 0.75
Recall	**0:** 0.92 **1:** 0.87	**0:** 0.87 **1:** 0.83	**0:** 0.86 **1:** 0.71 **2:** 0.85	**0:** 0.81 **1:** 0.60 **2:** 0.78

[8] *data_cleaning.py*

```
1.  import pandas as pd
2.  import numpy as np
3.
4.  directory = r'C:/Users/nscop/Documents/Graduate School/Capstone/Data/'
5.  file = 'original_full_data.txt'
6.  assignment_groupings_file = 'primary_assignment_groupings.xlsx'
7.  county_file = 'County.csv'
8.  urban_rural_file = 'county_urban_rural.csv'
9.
10. original_df = pd.read_csv(directory + file, sep = '|')
11. assignment_groupings_df = pd.read_excel(directory + assignment_groupings_file)
12. county_df = pd.read_csv(directory + county_file, sep = ',')
13. urban_rural_df = pd.read_csv(directory + urban_rural_file, sep = ',')
```

```
14.
15.     original_df.columns
16.
17.     # Filling in blank genders
18.     original_df[original_df['gender'] == ' '][['last_name','first_name']]
19.     original_df['gender'].unique()
20.     original_df.set_value(35757,'gender','M')
21.     original_df.set_value(194845,'gender','M')
22.     original_df.set_value(353548,'gender','M')
23.     original_df.set_value(461225,'gender','F')
24.     original_df.set_value(968010,'gender','F')
25.     original_df.set_value(1485473,'gender','F')
26.     original_df[original_df['gender'] == ' '][['last_name','first_name']]
27.
28.     # Remapping race
29.     original_df['race_ethnicity_desc'].unique()
30.     race_mapping = {'White, Non-Hispanic':'White',
31.             'Black, Non-Hispanic':'Black',
32.             'Asian or Pacific Islander':'Asian or Pacific Islander',
33.             'American Indian or Alaskan Native':'Native American',
34.             'Hispanic':'Hispanic',
35.             'White':'White',
36.             'Two or More Races':'Multiple',
37.             'Unknown':np.nan,
38.             'Black or African American':'Black',
39.             'Hispanic or Latino':'Hispanic',
40.             'Asian':'Asian or Pacific Islander',
41.             'Native Hawaiian or Other Pacific Islander':'Asian or Pacific Islander',
42.             'American Indian or Alaska Native':'Native American'}
43.     original_df['race_ethnicity_desc'] = original_df['race_ethnicity_desc'].replace(race_mapping)
44.     original_df['race_ethnicity_desc'].unique()
45.     original_df[original_df['race_ethnicity_desc'].isnull()]
46.
47.     # Removing with blank locations
48.     original_df['location_cd'].unique()
49.     len(original_df[original_df['location_cd'].isnull()])
50.     len(original_df[original_df['location_cd'] == ' '])
51.     original_df = original_df[original_df['location_cd'].notnull()]
52.     original_df = original_df[original_df['location_cd'] != ' ']
53.
54.     # Removing where blank type
55.     original_df['emply_type'].unique()
56.     len(original_df[original_df['emply_type'].isnull()])
```

```
len(original_df[original_df['emply_type'] == ' '])
original_df = original_df[original_df['emply_type'].notnull()]
original_df = original_df[original_df['emply_type'] != ' ']

# How many not nulls in assignments
original_df['assign1_desc'].unique()
len(original_df[original_df['assign1_desc'].isnull()])
len(original_df[original_df['assign1_desc'] == ' '])
original_df = original_df[original_df['assign1_desc'].notnull()]
original_df = original_df[original_df['assign1_desc'] != ' ']

# Renaming columns
original_df = original_df.rename(columns = {'fy':'Year',
                                    'rcdt':'District',
                                    'dst_name':'District_Name',
                                    'dst_addr':'District_Address',
                                    'dst_city':'District_City',
                                    'dst_st':'District_State',
                                    'dst_zip':'District_Zip',
                                    'sch_name':'School_Name',
                                    'sch_addr':'School_Address',
                                    'sch_city':'School_City',
                                    'sch_st':'School_State',
                                    'sch_zip':'School_Zip',
                                    'gender':'Gender',
                                    'race_ethnicity_desc':'Race',
                                    'tsr_status_desc':'Status',
                                    'location_desc':'Location_Desc',
                                    'emply_desc':'Employee_Type',
                                    'salary':'Salary',
                                    'months_employed':'Months_Employed',
                                    'pct_admin':'Percent_Administration',
                                    'pct_emp':'Percent_Time_Employed',
                                    'dist_exp':'District_Experience',
                                    'state_exp':'State_Experience',
                                    'out_of_state_exp':'Out_of_State_Experience',
                                    'bacc_coll_desc':'Undergraduate_College',
                                    'high_degre_desc':'Highest_Degree_Achieved',
                                    'adv_coll_desc':'Advance_College',
                                    'pos_desc':'Position',
                                    'low_grd_desc':'Lowest_Grade_Taught',
                                    'high_grd_desc':'Highest_Grade_Taught',
                                    'assign1_desc':'Primary_Assignment',
```

```
                                'assign2_desc':'Secondary_Assignment',
                                'assign3_desc':'Third_Assignment',
                                'assign4_desc':'Fourth_Assignment',
                                'assign5_desc':'Fifth_Assignment',
                                'assign6_desc':'Sixth_Assignment',
                                'assign7_desc':'Seventh_Assignment'})

    original_df['School_ID'] = original_df['District'].astype(str) + ':' + original_df['sch_num'].astype(str)

    teachers_df = original_df[['last_name','first_name','School_ID']].drop_duplicates(keep = False).reset_index(drop = True)
    teachers_df['Teacher_ID'] = teachers_df.index
    original_df = original_df.merge(teachers_df, how = 'left')

    # Getting the number of assignments an employee has
    original_df['Assignments'] = original_df[['Primary_Assignment','Secondary_Assignment','Third_Assignment',
'Fourth_Assignment','Fifth_Assignment','Sixth_Assignment','Seventh_Assignment']].notnull().sum(axis = 1)

    # Getting a new dataframe that is a subset of the columns
    original_df.columns
    new_df = original_df[['Year','District','District_Name','District_Address','District_City','District_State','District_Zip',
'School_ID','School_Name','School_Address','School_City','School_State','School_Zip',
'Teacher_ID','Salary','Gender','Race','Location_Desc','Employee_Type','Months_Employed','Percent_Administration',
'Percent_Time_Employed','District_Experience','State_Experience','Out_of_State_Experience',
'Undergraduate_College','Highest_Degree_Achieved','Advance_College','Position','Lowest_Grade_Taught',
'Highest_Grade_Taught','Primary_Assignment','Secondary_Assignment','Third_Assignment','Fourth_Assignment',
                    'Fifth_Assignment','Sixth_Assignment','Seventh_Assignment','Assignments']]

    # Adding a column for bins
```

```
128.    new_df['Salary_Greater_65000'] = 0
129.    new_df['Salary_Greater_65000'] = np.where(new_df['Salary'] > 65500,1,0)
130.    new_df['Salary_Greater_65000'].value_counts()
131.
132.    # Adding the county information and the assignment groupings
133.    assignment_groupings_df = assignment_groupings_df[['Assignment','New Group']]
134.    assignment_groupings_df.columns = ['Primary_Assignment','Primary_Assignment_Group']
135.    new_df = new_df.merge(assignment_groupings_df,how = 'left')
136.
137.    county_df = county_df[['zip','county']]
138.
139.    district_county_df = county_df.copy()
140.    district_county_df.columns = ['District_Zip','District_County']
141.
142.    school_county_df = county_df.copy()
143.    school_county_df.columns = ['School_Zip','School_County']
144.
145.    urban_rural_df = urban_rural_df[['County','Urban or Rural']]
146.    urban_rural_df.columns = ['School_County','Urban_or_Rural']
147.    urban_rural_df = urban_rural_df.drop_duplicates().reset_index(drop = True)
148.
149.    new_df = new_df.merge(district_county_df,how = 'left')
150.    new_df = new_df.merge(school_county_df,how = 'left')
151.    new_df = new_df.merge(urban_rural_df,how = 'left')
152.
153.    new_df =
new_df[['Year','District','District_Name','District_Address','District_City','District_State','District_Zip','District_County',
154.
'School_ID','School_Name','School_Address','School_City','School_State','School_Zip','School_County','Urban_or_Rural',
155.
'Teacher_ID','Salary','Gender','Race','Location_Desc','Employee_Type','Months_Employed','Percent_Administration',
156.
'Percent_Time_Employed','District_Experience','State_Experience','Out_of_State_Experience',
157.
'Undergraduate_College','Highest_Degree_Achieved','Advance_College','Position','Lowest_Grade_Taught',
158.
'Highest_Grade_Taught','Primary_Assignment','Secondary_Assignment','Third_Assignment','Fourth_Assignment',
159.
```

```
'Fifth_Assignment','Sixth_Assignment','Seventh_Assignment','Primary_Assignment_Group','Assignments
']]
160.
161.    # Exporting
162.    new_df.to_csv(directory + 'cleaned_full_data.txt',index = False, sep = '|')
```

[9] *data_filtering.py*

```
1.  import pandas as pd
2.  import os
3.  import numpy as np
4.  from scipy import stats, integrate
5.  import matplotlib.pyplot as plt
6.  import seaborn as sns
7.
8.  directory = r'C:/Users/nscop/Documents/Graduate School/Capstone/Data/'
9.  file = 'cleaned_full_data.txt'
10. cleaned_df = pd.read_csv(directory + file, sep = '|', converters = {'District_Zip':str,'School_Zip':str})
11. len(cleaned_df)
12. cleaned_df.head()
13. cleaned_df.columns
14. cleaned_df.dtypes
15.
16. # Removing all lines without district or school number
17. len(cleaned_df[(cleaned_df['District_Name'].isnull())|(cleaned_df['District_Name'] == ' ')|(cleaned_df['School_Name'].isnull())|(cleaned_df['School_Name'] == ' ')])
18. cleaned_df = cleaned_df[(cleaned_df['District_Name'].notnull()) & (cleaned_df['District_Name'] != ' ') & (cleaned_df['School_Name'].notnull()) & (cleaned_df['School_Name'] != ' ')]
19.
20. # Removing all rows without zip code for school and district
21. len(cleaned_df[(cleaned_df['District_Zip'].isnull())|(cleaned_df['District_Zip'] == ' ')|(cleaned_df['School_Zip'].isnull())|(cleaned_df['School_Zip'] == ' ')])
22. cleaned_df = cleaned_df[(cleaned_df['District_Zip'].notnull()) & (cleaned_df['District_Zip'] != ' ') & (cleaned_df['School_Zip'].notnull()) & (cleaned_df['School_Zip'] != ' ')]
23.
24. # Removing 0 salary
25. len(cleaned_df[(cleaned_df['Salary'] <= 0)|(cleaned_df['Salary'].isnull())])
26. cleaned_df = cleaned_df[(cleaned_df['Salary'] > 0) & (cleaned_df['Salary'].notnull())]
27.
28. # Full time only
29. cleaned_df['Employee_Type'].unique()
30. len(cleaned_df[cleaned_df['Employee_Type'] != 'Full-Time'])
31. cleaned_df = cleaned_df[cleaned_df['Employee_Type'] == 'Full-Time']
32. del cleaned_df['Employee_Type']
33.
34. # Removing employees who have not been employed less than a certain number of months within the year or above the amount possible
35. len(cleaned_df[(cleaned_df['Months_Employed'] < 9) | (cleaned_df['Months_Employed'] > 12)])
```

```
cleaned_df = cleaned_df[(cleaned_df['Months_Employed'] >= 9) & (cleaned_df['Months_Employed'] <= 12)]

# Removing people who have too much administrative work
sns.kdeplot(cleaned_df['Percent_Administration'], shade=True);
len(cleaned_df[cleaned_df['Percent_Administration'] > 75])
cleaned_df[cleaned_df['Percent_Administration'] > 75]['Primary_Assignment'].unique()
cleaned_df = cleaned_df[cleaned_df['Percent_Administration'] <= 75]

# Removing michigan and oregon
cleaned_df['District_State'].unique()
cleaned_df['District_State'].value_counts()
len(cleaned_df[cleaned_df['District_State'] != 'IL '])
cleaned_df = cleaned_df[cleaned_df['District_State'] == 'IL ']

# Removing salary below a threshold and non missing salary
len(cleaned_df[cleaned_df['Salary'] <= 20000])
cleaned_df = cleaned_df[(cleaned_df['Salary'] >= 20000) & (cleaned_df['Salary'].notnull())]

# Looking at cases where the salary is overly high
high_salary_df = cleaned_df[(cleaned_df['Salary'] >= 200000) & (cleaned_df['Salary'].notnull())]
high_salary_df.to_excel(directory + 'questionable_salary.xlsx',index = False)

# Removing the row with suspicious salaries
cleaned_df = cleaned_df[cleaned_df['Salary'] != 441612]
cleaned_df = cleaned_df[cleaned_df['Salary'] != 439803]
cleaned_df = cleaned_df[cleaned_df['Salary'] != 368000]
cleaned_df = cleaned_df[cleaned_df['Salary'] != 356258.15]
cleaned_df = cleaned_df[cleaned_df['Salary'] != 329404]

# Exporting
cleaned_filtered_df = cleaned_df.copy()
cleaned_filtered_df.to_csv(directory + 'cleaned_filtered_data.txt',index = False, sep = '|')

# Quick summary statistics
summary_df = cleaned_df[['Salary','Months_Employed','Percent_Administration','Percent_Time_Employed','District_Experience','State_Experience',
        'Out_of_State_Experience','Assignments']].describe()
summary_df.to_csv(directory + 'continuous_summary.txt',index = False, sep = '|')

# Discrete summary stats
gender = cleaned_df['Gender'].value_counts()
```

```
76.     race = cleaned_df['Race'].value_counts()
77.     highest_degree_achived = cleaned_df['Highest_Degree_Achieved'].value_counts()
78.     unique_teachers = cleaned_df['Teacher_ID'].nunique()
79.     unique_districts = cleaned_df['District'].nunique()
80.     unique_schools = cleaned_df['School_ID'].nunique()
```

[10] *initial_analysis.py*

```
1.      import pandas as pd
2.      import os
3.      import numpy as np
4.
5.      directory = r'C:\Users\nscop\Documents\Graduate School\Capstone\Data\Raw'
6.      data_dictionary = {}
7.
8.      # Seeing which headers are only in certain files
9.      headers_df = pd.DataFrame()
10.
11.     for file in os.listdir(directory):
12.         data_df = pd.read_csv(directory + '\\' + file)
13.         data_dictionary[file] = data_df
14.         headers = data_df.columns
15.
16.         headers = headers.insert(0,'File')
17.         temp_df = pd.DataFrame(columns = headers)
18.
19.         ones = list()
20.         for header in headers:
21.             ones.append(1)
22.         ones[0] = file
23.
24.         temp_2_df = pd.DataFrame(ones).transpose()
25.         temp_2_df.columns = temp_df.columns
26.
27.         headers_df = pd.concat([headers_df,temp_2_df])
28.
29.     headers_df.to_csv(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\headers_analysis.txt',index = False,sep = '|')
30.
31.     # Getting a master file
32.     files = list()
33.     instances = list()
34.     incomplete_instances = list()
```

```
35.     missing_data_df = pd.DataFrame()
36.     full_data_df = pd.DataFrame()
37.
38.     for file in data_dictionary:
39.         files.append(file)
40.         current_df = data_dictionary[file]
41.         instances.append(len(current_df))
42.         # Does not include adv_coll_desc or anything past assignment 1
43.         missing_df = current_df[current_df.iloc[:,pd.np.r_[0:40,41:48]].isnull().any(axis = 1)]
44.         incomplete_instances.append(len(missing_data_df))
45.         missing_df.loc[:,'File'] = file
46.         current_df.loc[:,'File'] = file
47.         missing_data_df = pd.concat([missing_data_df,missing_df])
48.         full_data_df = pd.concat([full_data_df,current_df])
49.
50.     missing_data_df.to_csv(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\missing_data_rows.txt',index = False,sep = '|')
51.     full_data_df.to_csv(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\full_data.txt',index = False,sep = '|')
```

[11] *initial_analysis_2.py*

```
1.  #Setting Directory
2.  import os
3.  os.getcwd()
4.  os.chdir('G:\Regulatory\Data Investigations\Projects\ScopeCarrPAProject')
5.
6.  from string import ascii_letters
7.  import numpy as np
8.  import pandas as pd
9.  import seaborn as sns
10. import matplotlib.pyplot as plt
11. import scipy as sp
12.
13. #Importing data
14. data_df = pd.read_csv('G:\Regulatory\Data
Investigations\Projects\ScopeCarrPAProject\\cleaned_filtered_data.txt',sep = '|')
15. list(data_df.columns.values)
16.
17. # Compute the correlation matrix
18. corr = data_df.corr()
19.
20. #Diagonal Correlation Matrix Visual
21. sns.set(style="white")
22. #Generate a mask for the upper triangle
```

```
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
#Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))
#Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
#Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, annot =True, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})


#test of independent variables
a = data_df['District_Experience']
b = data_df['State_Experience']
sp.stats.ttest_ind(a, b, axis=0, equal_var=True)


#ANOVA
#Performs a 1-way ANOVA.
#The one-way ANOVA tests the null hypothesis that two or more groups have the same population mean.
#The test is applied to samples from two or more groups, possibly with differing sizes.
#Assumptions
#The samples are independent.
#Each sample is from a normally distributed population.
#The population standard deviations of the groups are all equal. This property is known as homoscedasticity.

%matplotlib inline

import pandas as pd
data_df = pd.read_csv('C:\\Users\\carin\\Documents\\CSC672\\cleaned_filtered_data.txt',sep = '|')
data_df.head()

data_df.boxplot('Salary', by='Gender')
grps = pd.unique(data_df.Gender.values)
d_data = {grp:data_df['Salary'][data_df.Gender == grp] for grp in grps}
k = len(pd.unique(data_df.Gender))
N = len(data_df.values)
n = data_df.groupby('Gender').size()[0]
from scipy import stats
F, p = stats.f_oneway(d_data['F'], d_data['M'])


data_df.boxplot('Salary', by='Highest_Degree_Achieved')
grps = pd.unique(data_df.Highest_Degree_Achieved.values)
```

```
69. d_data = {grp:data_df['Salary'][data_df.Highest_Degree_Achieved == grp] for grp in grps}
70. k = len(pd.unique(data_df.Highest_Degree_Achieved))
71. N = len(data_df.values)
72. n = data_df.groupby('Highest_Degree_Achieved').size()[0]
73. from scipy import stats
74. F, p = stats.f_oneway(d_data['Masters'], d_data['Baccalaureate'],d_data['Doctorate'], d_data['None'])
75.
76.
77. data_df.boxplot('Salary', by='Race')
78. grps = pd.unique(data_df.Race.values)
79. d_data = {grp:data_df['Salary'][data_df.Race == grp] for grp in grps}
80. k = len(pd.unique(data_df.Race))
81. N = len(data_df.values)
82. n = data_df.groupby('Race').size()[0]
83. from scipy import stats
84. F, p = stats.f_oneway(d_data['White'], d_data['Black'],d_data['Hispanic'])
```

[12] *linear_model.py*

```
1.  import pandas as pd
2.  import itertools
3.  from sklearn.linear_model import LinearRegression
4.  from sklearn.linear_model import Lasso
5.  from sklearn.metrics import mean_squared_error
6.  import numpy as np
7.  import seaborn as sns
8.  import matplotlib as plt
9.  import statsmodels.api as sm
10. from scipy.stats import boxcox
11. from scipy import stats
12. import copy
13.
14. np.random.seed(2016)
15.
16. data_df = pd.read_csv(r'C:/Users/nscop/Documents/Graduate
School/Capstone/Data/cleaned_filtered_data.txt',sep = '|',
17.                converters = {'District_Zip':str})
18. columns = ['Year','Gender','Race','Percent_Administration','District_Experience',
19.         'State_Experience','Out_of_State_Experience','Highest_Degree_Achieved',
20.         'Primary_Assignment_Group']
21.
22. lm = LinearRegression()
23.
24. data_df[columns].dtypes
```

```
25.
26.     random_df = pd.DataFrame(np.random.randn(len(data_df), 2))
27.     msk = np.random.rand(len(random_df)) < 0.7
28.
29.     # Setting up the lists
30.     variables = list()
31.     dummy_variables = list()
32.     R2s = list()
33.     train_mses = list()
34.     test_mses = list()
35.
36.     # Creating a linear model from each combination of variables
37.     for Xs in range(1, len(columns) + 1):
38.         for subset in itertools.combinations(columns, Xs):
39.             temp_df = data_df[['Salary'] + list(subset)]
40.             temp_df = pd.get_dummies(temp_df)
41.
42.             train_df = temp_df[msk]
43.             test_df = temp_df[~msk]
44.
45.             X_train = train_df.copy()
46.             X_test = test_df.copy()
47.
48.             Y_train = X_train['Salary']
49.             Y_test = X_test['Salary']
50.
51.             del X_train['Salary']
52.             del X_test['Salary']
53.
54.             lm.fit(X_train,train_df['Salary'])
55.             R2 = lm.score(X_train,train_df['Salary'])
56.
57.             y_predict = lm.predict(X_train)
58.             train_mse = mean_squared_error(y_predict, Y_train)
59.
60.             y_predict = lm.predict(X_test)
61.             test_mse = mean_squared_error(y_predict, Y_test)
62.
63.             variables.append(str(subset))
64.             dummy_variables.append(str(X_train.columns))
65.             R2s.append(R2)
66.             train_mses.append(train_mse)
67.             test_mses.append(test_mse)
```

```
68.
69.             print(str(subset))
70.
71.         output_df = pd.DataFrame({'Variables': variables,
72.                                   'Dummy_Variables': dummy_variables,
73.                                   'R2': R2s,
74.                                   'Train_MSE': train_mses,
75.                                   'Test_MSE': test_mses})
76.
77.         # Looking at the results from the model after saving the output combinations
78.         output_df.to_csv(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\first_models.txt',sep = '|', index = False)
79.         output_df.to_excel(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\first_models.xlsx', index = False)
80.
81.         sp = sns.regplot(x = output_df['Train_MSE'], y = output_df['Test_MSE'], fit_reg = False)
82.         sp.set(xlabel = 'Training MSE',ylabel = 'Testing MSE', title = 'Relationship Between Training and
Testing MSE')
83.         sp.ticklabel_format(style = 'plain')
84.
85.         print(output_df['Train_MSE'].corr(output_df['Test_MSE']))
86.
87.         output_df = output_df.sort_values(by = ['Test_MSE'], ascending = True).reset_index(drop =
True)
88.
89.         # Subsetting the previous output to run in combination with zip code
90.         output_2_df = output_df.iloc[:20,:]
91.         subsets = output_2_df['Variables'].values.tolist()
92.         iterations = list()
93.         for i in subsets:
94.             i = i.replace('(','')
95.             i = i.replace(')','')
96.             i = i.replace("'","")
97.             i = i.replace(" ","")
98.             iterations.append(i.split(','))
99.
100.        iterations_2 = copy.deepcopy(iterations)
101.        iterations_3 = copy.deepcopy(iterations)
102.
103.        for i in iterations:
104.            i.append('District_Zip')
105.
106.        for i in iterations_2:
```

```
    i.append('District_County')

for i in iterations_3:
    i.append('Urban_or_Rural')

iterations = iterations + iterations_2 + iterations_3

# Rerunning with the combinations that were in the top 20 with and without district_zip
variables = list()
dummy_variables = list()
R2s = list()
train_mses = list()
test_mses = list()

# Creating a linear model from each combination of variables but also with the district_zip
for subset in iterations:
    temp_df = data_df[['Salary'] + subset]
    temp_df = pd.get_dummies(temp_df)

    train_df = temp_df[msk]
    test_df = temp_df[~msk]

    X_train = train_df.copy()
    X_test = test_df.copy()

    Y_train = X_train['Salary']
    Y_test = X_test['Salary']

    del X_train['Salary']
    del X_test['Salary']

    lm.fit(X_train,train_df['Salary'])
    R2 = lm.score(X_train,train_df['Salary'])

    y_predict = lm.predict(X_train)
    train_mse = mean_squared_error(y_predict, Y_train)

    y_predict = lm.predict(X_test)
    test_mse = mean_squared_error(y_predict, Y_test)

    variables.append(str(subset))
    dummy_variables.append(str(X_train.columns))
    R2s.append(R2)
```

```
150.        train_mses.append(train_mse)
151.        test_mses.append(test_mse)
152.
153.        print(str(subset))
154.
155.    output_3_df = pd.DataFrame({'Variables': variables,
156.                            'Dummy_Variables': dummy_variables,
157.                            'R2': R2s,
158.                            'Train_MSE': train_mses,
159.                            'Test_MSE': test_mses})
160.
161.    output_2_df = pd.concat([output_2_df, output_3_df])
162.    output_2_df.to_csv(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\second_models.txt',sep = '|', index = False)
163.    output_2_df.to_excel(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\second_models.xlsx', index = False)
164.
165.    # Test_MSE is always better with district zip
166.    output_2_df = output_2_df.sort_values(by = ['Test_MSE'], ascending = True).reset_index(drop =
True)
167.
168.    # Subsetting the previous output to run in combination with zip code
169.    best_linear_models_df = output_2_df.iloc[:5,:]
170.
171.    # Preparing to run lasso regressions
172.    subsets = best_linear_models_df['Variables'].values.tolist()
173.    iterations = list()
174.    for i in subsets:
175.        i = i.replace('[',"")
176.        i = i.replace(']',"")
177.        i = i.replace("'","")
178.        i = i.replace(" ","")
179.        iterations.append(i.split(','))
180.
181.    alphas = [1e-15, 1e-10, 1e-8, 1e-5,1e-4, 1e-3,1e-2, 1, 5, 10]
182.
183.    variables = list()
184.    dummy_variables = list()
185.    R2s = list()
186.    train_mses = list()
187.    test_mses = list()
188.    alpha = list()
189.
```

```
# Running the iterations using lasso alphas
for subset in iterations:
    temp_df = data_df[['Salary'] + subset]
    temp_df = pd.get_dummies(temp_df)

    train_df = temp_df[msk]
    test_df = temp_df[~msk]

    X_train = train_df.copy()
    X_test = test_df.copy()

    Y_train = X_train['Salary']
    Y_test = X_test['Salary']

    del X_train['Salary']
    del X_test['Salary']

    for current_alpha in alphas:

        print(current_alpha)
        # Due to a low number of max iterations will need to rerun once a best model is selected
        lassoreg = Lasso(alpha = current_alpha, normalize = True, max_iter = 100)

        lassoreg.fit(X_train,train_df['Salary'])
        R2 = lassoreg.score(X_train,train_df['Salary'])

        y_predict = lassoreg.predict(X_train)
        train_mse = mean_squared_error(y_predict, Y_train)

        y_predict = lassoreg.predict(X_test)
        test_mse = mean_squared_error(y_predict, Y_test)

        variables.append(str(subset))
        dummy_variables.append(str(X_train.columns))
        R2s.append(R2)
        train_mses.append(train_mse)
        test_mses.append(test_mse)
        alpha.append(current_alpha)

    print(str(subset))

output_lasso_df = pd.DataFrame({'Variables': variables,
                    'Dummy_Variables': dummy_variables,
```

```
                    'R2': R2s,
                    'Train_MSE': train_mses,
                    'Test_MSE': test_mses,
                    'Alpha': alpha})
    output_lasso_df.to_csv(r'C:\Users\nscop\Documents\Graduate School\Capstone\Analysis\lasso_models.txt',sep = '|', index = False)
    output_lasso_df.to_excel(r'C:\Users\nscop\Documents\Graduate School\Capstone\Analysis\lasso_models.xlsx', index = False)

    sp = sns.boxplot(x = output_lasso_df['Alpha'], y = output_lasso_df['Test_MSE'])
    sp.set(title = 'Effects of Lasso Regression on Model Accuracy', ylabel = 'Test MSE')
    sp.figure.savefig(r'C:\Users\nscop\Documents\Graduate School\Capstone\Analysis\alpha_model_analysis.png', dpi = 150)

    # Narrowing down to the best model
    output_lasso_df[output_lasso_df['Test_MSE'] == output_lasso_df['Test_MSE'].min()]
    best_lasso_mse = output_lasso_df[output_lasso_df['Test_MSE'] == output_lasso_df['Test_MSE'].min()]['Test_MSE']
    best_linear_mse = output_2_df[output_2_df['Test_MSE'] == output_2_df['Test_MSE'].min()]['Test_MSE']
    best_lasso_mse.values.tolist()[0] <= best_linear_mse.values.tolist()[0]

    # Recreating the best model so we can analyze it
    best_variables = output_2_df[output_2_df['Test_MSE'] == output_2_df['Test_MSE'].min()]['Variables'].values.tolist()
    best_variables = best_variables[0]
    best_variables = best_variables.replace('[','')
    best_variables = best_variables.replace(']','')
    best_variables = best_variables.replace("'","")
    best_variables = best_variables.replace(" ","")
    best_variables = best_variables.split(',')
    subet = best_variables

    """
    ['Year',
     'Gender',
     'Race',
     'Percent_Administration',
     'District_Experience',
     'State_Experience',
     'Out_of_State_Experience',
     'Highest_Degree_Achieved',
     'Primary_Assignment_Group',
```

```
 'District_County']
'''

# Rebuilding the model
temp_df = data_df[['Salary'] + subset]
temp_df = pd.get_dummies(temp_df)

train_df = temp_df[msk]
test_df = temp_df[~msk]

X_train = train_df.copy()
X_test = train_df.copy()

Y_train = X_train['Salary']
Y_test = X_test['Salary']

del X_train['Salary']
del X_test['Salary']

lm.fit(X_train,train_df['Salary'])

R2 = lm.score(X_train,train_df['Salary'])

y_predict = lm.predict(X_train)
train_mse = mean_squared_error(y_predict, Y_train)

y_predict = lm.predict(X_test)
test_mse = mean_squared_error(y_predict, Y_test)

X_train_2 = sm.add_constant(X_train)
est = sm.OLS(train_df['Salary'],X_train_2)
model = est.fit()
print(model.summary())

# Checking the model
validation_df = temp_df.copy()
validation_salary = validation_df['Salary']
del validation_df['Salary']

predictions = lm.predict(validation_df)
residuals = validation_salary - predictions

sp = sns.regplot(x = predictions, y = residuals, fit_reg = False)
```

```
313.    sp.set(xlabel = 'Predictions',ylabel = 'Residuals', title = 'Residual Analysis')
314.
315.    sns.kdeplot(residuals, shade = True, color = "g")
316.    stats.normaltest(residuals)
317.
318.    # Seeing if creating a log model gets rid of the fan shape
319.    log_y = np.log(train_df['Salary'])
320.    log_lm = LinearRegression()
321.    log_lm.fit(X_train,log_y)
322.
323.    log_validation_salary = np.log(validation_salary)
324.    log_predictions = log_lm.predict(validation_df)
325.    log_residuals = log_validation_salary - log_predictions
326.
327.    sp = sns.regplot(x = log_predictions, y = log_residuals, fit_reg = False)
328.    sp.set(xlabel = 'Predictions',ylabel = 'Residuals', title = 'Log Residual Analysis')
329.
330.    # Seeing if creating a square root model gets rid of the fan shape
331.    sqrt_y = np.sqrt(train_df['Salary'])
332.    sqrt_lm = LinearRegression()
333.    sqrt_lm.fit(X_train,sqrt_y)
334.
335.    sqrt_validation_salary = np.sqrt(validation_salary)
336.    sqrt_predictions = sqrt_lm.predict(validation_df)
337.    sqrt_residuals = sqrt_validation_salary - sqrt_predictions
338.
339.    sp = sns.regplot(x = sqrt_predictions, y = sqrt_residuals, fit_reg = False)
340.    sp.set(xlabel = 'Predictions',ylabel = 'Residuals', title = 'Sqrt Residual Analysis')
341.
342.    # Extracting the residuals from the original linear model and drawing the necessary conclusions
343.    validation_df = temp_df.copy()
344.    validation_salary = validation_df['Salary']
345.    del validation_df['Salary']
346.
347.    predictions = lm.predict(validation_df)
348.    residuals = validation_salary - predictions
349.    data_df['Linear_Model_Residuals'] = residuals
350.    data_df.to_csv(r'C:/Users/nscop/Documents/Graduate
School/Capstone/Data/cleaned_filtered_and_residuals_data.txt',index = False, sep = '|')
351.
352.    sp = sns.regplot(x = predictions, y = residuals, fit_reg = False)
353.    sp.set(xlabel = 'Predictions',ylabel = 'Residuals', title = 'Residual Analysis')
354.    sp.figure.savefig(r'C:\Users\nscop\Documents\Graduate
```

```
School\Capstone\Analysis\residual_plot.png', dpi = 150)
355.
356.    sp = sns.kdeplot(residuals, shade = True, color = "g")
357.    sp.set(title = 'Residual Density Curve')
358.    sp.figure.savefig(r'C:\Users\nscop\Documents\Graduate
School\Capstone\Analysis\residual_density_curve.png', dpi = 150)
359.
360.    est = sm.OLS(train_df['Salary'],X_train_2)
361.    model = est.fit()
362.    print(model.summary())
363.
364.    stats.normaltest(residuals)
```

[13] bayesian_nets.R

```
1.  library(bnlearn)
2.  library(kernlab)
3.  library(ggplot2)
4.  library(caret)
5.  library(e1071)
6.  library(rpart.plot)
7.  library(rpart)
8.  library(rattle)
9.  library(mlbench)
10. library(randomForest)
11. library(pROC)
12. library(psych)
13. library(gplots)
14. library(sm)
15.
16. cf_data = read.csv("D:/depaul/csc672/cleaned_filtered_and_residuals_data.csv", sep = "|")
17. salary = cf_data[, c(1, 7, 8, 16, 18, 19, 20, 22, 23, 25, 26, 27, 29, 31, 41)]
18. salary$Salary = as.numeric(cut_number(salary$Salary, 3))
19. #salary$District_Experience = as.numeric(cut_number(salary$District_Experience, 5))
20. salary$State_Experience = as.numeric(cut_number(salary$State_Experience, 5))
21. salary_level = salary[, c(-1, -2, -3, -8, -10, -14)]
22.
23. convert = c(1:9)
24. salary_level[, convert] = lapply(salary_level[, convert], factor)
25. str(salary_level)
26.
27. #bayesian networks
28.
```

```
29. bn_df <- data.frame(salary_level)
30. res <- hc(bn_df)
31. plot(res)
```

[14] *conditional_analaysis.py*

```
1. import pandas as pd
2. from scipy import stats
3.
4.  data_df = pd.read_csv(r'C:\Users\nscop\Grad School\Capstone\Data\cleaned_filtered_data.txt',sep = '|')
5. data_df.head()
6. data_df.columns
7.
8. # t-Test for gender and salary
9. print data_df[data_df['Gender'] == 'F']['Salary'].mean()
10. print data_df[data_df['Gender'] == 'M']['Salary'].mean()
11. t_stat, p_val = stats.ttest_ind(data_df[data_df['Gender'] == 'F']['Salary'], data_df[data_df['Gender'] == 'M']['Salary'], equal_var = False)
12. print t_stat
13. print p_val
14.
15. # t-Test for urban/rural and salary
16. print data_df[data_df['Urban_or_Rural'] == 'Urban']['Salary'].mean()
17. print data_df[data_df['Urban_or_Rural'] == 'Rural']['Salary'].mean()
18. t_stat, p_val = stats.ttest_ind(data_df[data_df['Urban_or_Rural'] == 'Urban']['Salary'], data_df[data_df['Urban_or_Rural'] == 'Rural']['Salary'], equal_var = False)
19. print t_stat
20. print p_val
21.
22. # Finding the mean for the combinations of the two
23. data_df.groupby(['Urban_or_Rural','Gender'])['Salary'].mean().reset_index()
24.
25. # Grouping the salary into bins
26. data_df['Salary_Bin'] = 'Low'
27. data_df.loc[data_df['Salary'] > 65530.92,'Salary_Bin'] = 'Middle'
28. data_df.loc[data_df['Salary'] > 99999.99,'Salary_Bin'] = 'High'
29. data_df['Salary_Bin'].value_counts()
30.
31. # Finding the distribution within those bins
32. data_df[data_df['Gender'] == 'F'].groupby(['Urban_or_Rural','Salary_Bin']).size().reset_index()
33. data_df[data_df['Gender'] == 'M'].groupby(['Urban_or_Rural','Salary_Bin']).size().reset_index()
34.
```

data_df.groupby(['Gender','Urban_or_Rural','Salary_Bin']).size().reset_index()

[15] *classification_models.py*

```
FEATURE SELECTED DATA
#Importing, Loading, Subsetting-
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.utils import resample
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from sklearn import svm
import graphviz

data = pd.read_csv('cleaned_filtered_data.txt', sep='|')
sub = data[['Salary','Gender', 'Race', 'Percent_Administration', 'State_Experience', 'Highest_Degree_Achieved', 'Primary_Assignment_Group', 'District_County', 'Urban_or_Rural', 'Out_of_State_Experience']]
sub = pd.get_dummies(sub)

#Clustering-
kmeans = KMeans(n_clusters=2, random_state=222).fit(sub)
centers = kmeans.cluster_centers_
centers[0][1]
labels = kmeans.labels_
sub['Cluster2'] = labels
zero = []
one = []
for i in range(len(sub)):
    sal = sub['Salary'][i]
    cent = sub['Cluster2'][i]
    if cent ==0:
        zero.append(sal)
    else:
        one.append(sal)
print('Min Cluster 0: ', min(zero))
print('Max Cluster 0: ', max(zero))
print('Mean Cluster 0: ', (sum(zero)/len(zero)))
```

```
38.     print('-----------------------------------------------')
39.     print('Min Cluster 1: ', min(one))
40.     print('Max Cluster 1: ', max(one))
41.     print('Mean Cluster 1: ', (sum(one)/len(one)))
42.     print('Number of Instances in Cluster Zero: ', len(zero))
43.     print('-----------------------------------------------')
44.     print('Number of Instances in Cluster One: ', len(one))
45.
46.     #creating my own partitions
47.     three_clust = []
48.     for value in sub['Salary']:
49.         if value <= 65530.92:
50.             three_clust.append(0)
51.         elif value > 65530.92 and value < 100000.00:
52.             three_clust.append(1)
53.         else:
54.             three_clust.append(2)
55.     len(three_clust)
56.     less_65532 = []
57.     less_100k = []
58.     more_100k = []
59.     for value in three_clust:
60.         if value ==0:
61.             less_65532.append(1)
62.         elif value ==1:
63.             less_100k.append(1)
64.         else:
65.             more_100k.append(1)
66.     print('Number of Instances in Cluster Zero: ', len(less_65532))
67.     print('-----------------------------------------------')
68.     print('Number of Instances in Cluster One: ', len(less_100k))
69.     print('-----------------------------------------------')
70.     print('Number of Instances in Cluster Two: ', len(more_100k))
71.     sub['Cluster3'] = three_clust
72.     zero = []
73.     one = []
74.     two = []
75.     for i in range(len(sub)):
76.         sal = sub['Salary'][i]
77.         cent = sub['Cluster3'][i]
78.         if cent ==0:
79.             zero.append(sal)
80.         elif cent ==1:
```

```
            one.append(sal)
        else:
            two.append(sal)
    print('Min Cluster 0: ', min(zero))
    print('Max Cluster 0: ', max(zero))
    print('Mean Cluster 0: ', (sum(zero)/len(zero)))
    print('--------------------------------------------')
    print('Min Cluster 1: ', min(one))
    print('Max Cluster 1: ', max(one))
    print('Mean Cluster 1: ', (sum(one)/len(one)))
    print('--------------------------------------------')
    print('Min Cluster 2: ', min(two))
    print('Max Cluster 2: ', max(two))
    print('Mean Cluster 2: ', (sum(two)/len(two)))

    #Train/test split, Balancing-
    #balancing 2-class feature selected
    #downsampling the majority class
    from sklearn.utils import resample
    majority2 = sub[sub['Cluster2']==1]
    minority2 = sub[sub['Cluster2']==0]
    majorityDown2 = resample(majority2, replace=False, n_samples=491129, random_state=234)
    downTrain2 = pd.concat([majorityDown2,minority2])
    countGreat = 0
    countLess = 0
    for value in downTrain2['Cluster2']:
        if value == 1:
            countGreat +=1
        else:
            countLess +=1
    print(countGreat, ' ',countLess)
    down2Class = downTrain2['Cluster2']
    downTrain2 = downTrain2.drop('Cluster3', 1)
    downTrain2 = downTrain2.drop('Cluster2', 1)
    downTrain2 = downTrain2.drop('Salary', 1)
    downTrain2.head()

    #balancing 3 class
    majority3 = sub[sub['Cluster3']==0]
    minority3_1 = sub[sub['Cluster3']==1]
    minority3_2 = sub[sub['Cluster3']==2]
    majorityDown3 = resample(majority3, replace=False, n_samples=59835, random_state=234)
    majorityDown3_1 = resample(minority3_1, replace=False, n_samples=59835,
```

```
random_state=234)
    downTrain3 = pd.concat([majorityDown3, majorityDown3_1, minority3_2])
    countGreat = 0
    countLess = 0
    countLess2 =0
    for value in downTrain3['Cluster3']:
        if value == 0:
            countGreat +=1
        elif value ==1:
            countLess +=1
        else:
            countLess2 += 1
    print(countGreat, ' ',countLess, ' ', countLess2)
    down3Class = downTrain3['Cluster3']
    downTrain3 = downTrain3.drop('Cluster3', 1)
    downTrain3 = downTrain3.drop('Cluster2', 1)
    downTrain3 = downTrain3.drop('Salary', 1)

    #train test splits
    train2, test2, trainclass2, testclass2 = train_test_split(downTrain2, down2Class, test_size=0.35,
random_state=262)
    train3, test3, trainclass3, testclass3 = train_test_split(downTrain3, down3Class, test_size=0.35,
random_state=262)

    #Decision Tree Models-

    # decision tree grid search for three class feature selection
    from sklearn import tree
    from sklearn.model_selection import GridSearchCV
    params = {'criterion': ['gini', 'entropy'],
            'min_samples_split': (2,3,4,5,10,15,20,30,40,50,100),
            'max_features': [None, 'sqrt', 'log2']}
    tree = tree.DecisionTreeClassifier()
    groot = GridSearchCV(tree, params)
    groot.fit(train3, trainclass3)
    print(groot.best_params_)
    #best decision tree with feature selection 3-class
    from sklearn import tree
    tree = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
    tree = tree.fit(train3, trainclass3)
    trainP = tree.predict(train3)
    testP = tree.predict(test3)
    print('Number of Min Splits: ', 100)
```

```
print(' ')
print('Train Accuracy: ', tree.score(train3, trainclass3))
print(classification_report(trainclass3, trainP))
print(' ')
print('Test Accuracy: ', tree.score(test3, testclass3))
print(classification_report(testclass3, testP))
print('-----------------------')
# decision tree grid search for two class feature selection
from sklearn import tree
from sklearn.model_selection import GridSearchCV
params = {'criterion': ['gini', 'entropy'],
        'min_samples_split': (2,3,4,5,10,15,20,30,40,50,100),
        'max_features': [None, 'sqrt', 'log2']}
tree = tree.DecisionTreeClassifier()
groot = GridSearchCV(tree, params)
groot.fit(train2, trainclass2)
print(groot.best_params_)
#best decision tree with feature selection 2-class
from sklearn import tree
tree = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
tree = tree.fit(train2, trainclass2)
trainP = tree.predict(train2)
testP = tree.predict(test2)
print('Number of Min Splits: ', 100)
print(' ')
print('Train Accuracy: ', tree.score(train2, trainclass2))
print(classification_report(trainclass2, trainP))
print(' ')
print('Test Accuracy: ', tree.score(test2, testclass2))
print(classification_report(testclass2, testP))
print('-----------------------')


#Random Forest Models-

#random forest with 2 class with feature selection
from sklearn.ensemble import RandomForestClassifier
n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
testError2 = []
trainError2 =[]
numIters = []
for i in n_est:
    numIters.append(i)
```

```
    clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
    clf.fit(train2, trainclass2)
    accTrain = clf.score(train2, trainclass2)
    errTrain = 1 - accTrain
    trainError2.append(errTrain)
    accTest = clf.score(test2, testclass2)
    errTest = 1 - accTest
    testError2.append(errTest)
plt.plot(numIters, trainError2, label='Training Error')
plt.plot(numIters, testError2, label = 'Testing Error')
plt.xlabel('Number of Estimators')
plt.ylabel('Error Rate')
plt.legend(('Train Error', 'Test Error'))
plt.grid(True)
plt.show()
#best with fs 2-class
clf = RandomForestClassifier(min_samples_split=2, n_estimators=10,random_state=111)
clf.fit(train2, trainclass2)
testp = clf.predict(test2)
trainp = clf.predict(train2)
print('Train Accuracy: ', clf.score(train2, trainclass2))
print(classification_report(trainclass2, trainp))
print(' ')
print('Test Accuracy: ', clf.score(test2, testclass2))
print(classification_report(testclass2, testp))
print('-----------------------')

#random forest with 3 class with feature selection
n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
testError3f = []
trainError3f =[]
numIters = []
for i in n_est:
    numIters.append(i)
    clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
    clf.fit(train3, trainclass3)
    accTrain = clf.score(train3, trainclass3)
    errTrain = 1 - accTrain
    trainError3f.append(errTrain)
    accTest = clf.score(test3, testclass3)
    errTest = 1 - accTest
    testError3f.append(errTest)
plt.plot(numIters, trainError3f, label='Training Error')
```

```
plt.plot(numIters, testError3f, label = 'Testing Error')
plt.xlabel('Number of Estimators')
plt.ylabel('Error Rate')
plt.legend(('Train Error', 'Test Error'))
plt.grid(True)
plt.show()
#best with fs 3-class
clf = RandomForestClassifier(min_samples_split=2, n_estimators=15,random_state=111)
clf.fit(train3, trainclass3)
testp = clf.predict(test3)
trainp = clf.predict(train3)
print('Train Accuracy: ', clf.score(train3, trainclass3))
print(classification_report(trainclass3, trainp))
print(' ')
print('Test Accuracy: ', clf.score(test3, testclass3))
print(classification_report(testclass3, testp))
print('-----------------------')


#SVM-

#svm on the three class with feature selected
#attempting 3 class svm - nick feature selection
from sklearn import svm
sv3 = svm.SVC(kernel = 'rbf')
sv3.fit(train3, trainclass3)
accTrain = sv3.score(train3,trainclass3)
accTest = sv3.score(test3, testclass3)
trainP = sv3.predict(train3)
testP = sv3.predict(test3)
print('Train Accuracy: ', accTrain)
print(classification_report(trainclass3, trainP))
print('---------------------------')
print('Test Accuracy: ', accTest)
print(classification_report(testclass3, testP))

#analyzing support vectors
index_support = sv3.support_
trainarray = np.array(train3)
trainclassarray = np.array(trainclass3)
c_1 = []
c_0 = []
c_2 = []
```

```
    for i in index_support:
        if trainclassarray[i] == 0:
            c_0.append(trainarray[i])

        elif trainclassarray[i] == 1:
            c_1.append(trainarray[i])
        else:
            c_2.append(trainarray[i])
    print(len(c_0), ' ', len(c_1), ' ', len(c_2))
    female = 0
    male = 0
    white = 0
    not_white = 0
    percent_admin = []
    state_exp = []
    assmnt = []
    for row in c_0:
        percent_admin.append(row[2])
        state_exp.append(row[5])
        assmnt.append(row[7])
        if int(row[112]) == 1:
            female +=1
        else:
            male += 1
        if int(row[120]) ==1:
            white +=1
        else:
            not_white += 1
    print(len(assmnt))
    print('Class 0: ')
    print('-----------------------------')
    print('Number of Females: ', female)
    print('-----------------------------')
    print('Number of Males: ', male)
    print('-----------------------------')
    print('Number of White: ', white)
    print('-----------------------------')
    print('Number of Not-White: ', not_white)
    print('-----------------------------')
    print('Mean State Experience: ', sum(state_exp)/len(state_exp))
    print('Min State Exp.: ', min(state_exp), ' Max State Exp.: ', max(state_exp))
    print('-----------------------------')
    print('Mean Percent_Admin: ', sum(percent_admin)/len(percent_admin))
```

```
336.    print('Min % Admin: ', min(percent_admin), ' Max % Admin: ', max(percent_admin))
337.    print('-----------------------------')
338.    print('Mean Number of Assignments: ', sum(assmnt)/len(assmnt))
339.    print('Min Num. Assign.: ', min(assmnt), ' Max Num. Assign.: ', max(assmnt))
340.
341.    female1 = 0
342.    male1 = 0
343.    white1 = 0
344.    not_white1 = 0
345.    percent_admin1 = []
346.    state_exp1 = []
347.    assmnt1 = []
348.    for row in c_1:
349.        percent_admin1.append(row[2])
350.        state_exp1.append(row[5])
351.        assmnt1.append(row[7])
352.        if int(row[112]) == 1:
353.            female1 +=1
354.        else:
355.            male1 += 1
356.        if int(row[120]) ==1:
357.            white1 +=1
358.        else:
359.            not_white1 += 1
360.    print(len(assmnt1))
361.    print('Class 1: ')
362.    print('-----------------------------')
363.    print('Number of Females: ', female1)
364.    print('-----------------------------')
365.    print('Number of Males: ', male1)
366.    print('-----------------------------')
367.    print('Number of White: ', white1)
368.    print('-----------------------------')
369.    print('Number of Not-White: ', not_white1)
370.    print('-----------------------------')
371.    print('Mean State Experience: ', sum(state_exp1)/len(state_exp1))
372.    print('Min State Exp.: ', min(state_exp1), ' Max State Exp.: ', max(state_exp1))
373.    print('-----------------------------')
374.    print('Mean Percent_Admin: ', sum(percent_admin1)/len(percent_admin1))
375.    print('Min % Admin: ', min(percent_admin1), ' Max % Admin: ', max(percent_admin1))
376.    print('-----------------------------')
377.    print('Mean Number of Assignments: ', sum(assmnt1)/len(assmnt1))
378.    print('Min Num. Assign.: ', min(assmnt1), ' Max Num. Assign.: ', max(assmnt1))
```

```
    female2 = 0
    male2 = 0
    white2 = 0
    not_white2 = 0
    percent_admin2 = []
    state_exp2 = []
    assmnt2 = []
    for row in c_2:
        percent_admin2.append(row[2])
        state_exp2.append(row[5])
        assmnt2.append(row[7])
        if int(row[112]) == 1:
            female2 +=1
        else:
            male2 += 1
        if int(row[120]) ==1:
            white2 +=1
        else:
            not_white2 += 1
    print(len(assmnt2))
    print('Class 2: ')
    print('-----------------------------')
    print('Number of Females: ', female2)
    print('-----------------------------')
    print('Number of Males: ', male2)
    print('-----------------------------')
    print('Number of White: ', white2)
    print('-----------------------------')
    print('Number of Not-White: ', not_white2)
    print('-----------------------------')
    print('Mean State Experience: ', sum(state_exp2)/len(state_exp2))
    print('Min State Exp.: ', min(state_exp2), ' Max State Exp.: ', max(state_exp2))
    print('-----------------------------')
    print('Mean Percent_Admin: ', sum(percent_admin2)/len(percent_admin2))
    print('Min % Admin: ', min(percent_admin2), ' Max % Admin: ', max(percent_admin2))
    print('-----------------------------')
    print('Mean Number of Assignments: ', sum(assmnt2)/len(assmnt2))
    print('Min Num. Assign.: ', min(assmnt2), ' Max Num. Assign.: ', max(assmnt2))

    plt.boxplot([state_exp, state_exp1, state_exp2], labels = ['Class 0 ', 'Class 1', 'Class 2'],
vert=False, whis = 30)
    plt.xlabel('State Experience - Years')
    plt.show()
```

```
421.  plt.hist(state_exp2)
422.  plt.xlabel('State Experience - Years')
423.  plt.ylabel('Counts')
424.  plt.grid(True)
425.  plt.show()
426.
427.  females = [female, female1, female2]
428.  males = [male, male1, male2]
429.  whites = [white, white1, white2]
430.  not_whites = [not_white, not_white1, not_white2]
431.  classes = ('Class 0', 'Class 1', 'Class 2')
432.
433.  x = np.arange(3)
434.  plt.bar(x, females)
435.  plt.xticks(x, classes)
436.  plt.bar(x, males)
437.  plt.ylabel('Number of Females & Males')
438.  plt.legend(["Female", "Male"])
439.  plt.show()
440.
441.  x = np.arange(3)
442.  plt.bar(x, whites)
443.  plt.xticks(x, classes)
444.  plt.bar(x, not_whites)
445.  plt.ylabel('Number of Whites & Minorities')
446.  plt.legend(["White", "Minority"])
447.  plt.show()
448.
449.  print('Class 0')
450.  print('Percent Female ', female/(female+male))
451.  print('Percent Minority ', not_white/(white+not_white))
452.
453.  print('Class 1')
454.  print('Percent Female ', female1/(female1+male1))
455.  print('Percent Minority ', not_white1/(white1+not_white1))
456.
457.  print('Class 2')
458.  print('Percent Female ', female2/(female2+male2))
459.  print('Percent Minority ', not_white2/(white2+not_white2))
460.
461.  plt.hist(state_exp2)
462.  plt.xlabel('State Experience - Years')
463.  plt.ylabel('Counts')
```

```
464.    plt.grid(True)
465.    plt.show()
466.
467.    plt.hist(state_exp1)
468.    plt.xlabel('State Experience - Years')
469.    plt.ylabel('Counts')
470.    plt.grid(True)
471.    plt.show()
472.
473.    plt.hist(state_exp)
474.    plt.xlabel('State Experience - Years')
475.    plt.ylabel('Counts')
476.    plt.grid(True)
477.    plt.show()
478.
479.    pac0 = 0
480.    for value in percent_admin:
481.        if int(value) !=0:
482.            pac0 += 1
483.    pac0
484.
485.    pac1 = 0
486.    for value in percent_admin1:
487.        if int(value) !=0:
488.            pac1 += 1
489.    pac1
490.
491.    pac2 = 0
492.    for value in percent_admin2:
493.        if int(value) !=0:
494.            pac2 += 1
495.    pac2
496.
497.    (pac0/13854) + (pac1/29780) + (pac2/17429)
498.
499.    #NON FEATURE SELECTED DATA
500.    #Loading
501.
502.    noFS = data[['Year', 'District_County','Urban_or_Rural', 'Salary', 'Gender', 'Race',
'Months_Employed', 'Percent_Administration', 'Percent_Time_Employed', 'District_Experience',
'State_Experience', 'Out_of_State_Experience', 'Undergraduate_College', 'Highest_Degree_Achieved',
'Position', 'Highest_Grade_Taught', 'Primary_Assignment_Group', 'Assignments']]
503.
```

```
noFS = pd.get_dummies(noFS)

#Cluster groupings, balancing, train/test split-

twoClass = []
for value in noFS['Salary']:
    if value >65530.92:
        twoClass.append(0)
    else:
        twoClass.append(1)
twoClass[0:15]
threeClass = []
for value in noFS['Salary']:
    if value <= 65530.92:
        threeClass.append(0)
    elif value > 65530.92 and value < 100000.00:
        threeClass.append(1)
    else:
        threeClass.append(2)
len(threeClass)
less_65532 = []
less_100k = []
more_100k = []
for value in threeClass:
    if value ==0:
        less_65532.append(1)
    elif value ==1:
        less_100k.append(1)
    else:
        more_100k.append(1)
print('Number of Instances in Cluster Zero: ', len(less_65532))
print('-------------------------------------------------')
print('Number of Instances in Cluster One: ', len(less_100k))
print('-------------------------------------------------')
print('Number of Instances in Cluster Two: ', len(more_100k))
noFS['2Class'] = twoClass
noFS['3Class'] = threeClass

#balancing 2-class
#downsampling the majority class
from sklearn.utils import resample
majority2 = noFS[noFS['2Class']==1]
minority2 = noFS[noFS['2Class']==0]
```

```
    majorityDown2 = resample(majority2, replace=False, n_samples=491129, random_state=234)
    downTrain2 = pd.concat([majorityDown2,minority2])
    countGreat = 0
    countLess = 0
    for value in downTrain2['2Class']:
        if value == 1:
            countGreat +=1
        else:
            countLess +=1
    print(countGreat, ' ',countLess)

    down2Class = downTrain2['2Class']
    downTrain2 = downTrain2.drop('2Class', 1)
    downTrain2 = downTrain2.drop('3Class', 1)
    downTrain2 = downTrain2.drop('Salary', 1)

    #balancing 3 class
    majority3 = noFS[noFS['3Class']==0]
    minority3_1 = noFS[noFS['3Class']==1]
    minority3_2 = noFS[noFS['3Class']==2]
    majorityDown3 = resample(majority3, replace=False, n_samples=59835, random_state=234)
    majorityDown3_1 = resample(minority3_1, replace=False, n_samples=59835, random_state=234)
    downTrain3 = pd.concat([majorityDown3, majorityDown3_1, minority3_2])
    countGreat = 0
    countLess = 0
    countLess2 =0
    for value in downTrain3['3Class']:
        if value == 0:
            countGreat +=1
        elif value ==1:
            countLess +=1
        else:
            countLess2 += 1
    print(countGreat, ' ',countLess, ' ', countLess2)
    down3Class = downTrain3['3Class']
    downTrain3 = downTrain3.drop('3Class', 1)
    downTrain3 = downTrain3.drop('2Class', 1)
    downTrain3 = downTrain3.drop('Salary', 1)

    #train test split for two class
    train2, test2, trainclass2, testclass2 = train_test_split(downTrain2, down2Class, test_size=0.35, random_state=262)
```

```
588. train3, test3, trainclass3, testclass3 = train_test_split(downTrain3, down3Class, test_size=0.35, random_state=262)
589. #Decision Trees-
590.
591. #random forest with 2 class w/0 feature selection
592. from sklearn.ensemble import RandomForestClassifier
593. n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
594. testError2 = []
595. trainError2 =[]
596. numIters = []
597. for i in n_est:
598.     numIters.append(i)
599.     clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
600.     clf.fit(train2, trainclass2)
601.     accTrain = clf.score(train2, trainclass2)
602.     errTrain = 1 - accTrain
603.     trainError2.append(errTrain)
604.     accTest = clf.score(test2, testclass2)
605.     errTest = 1 - accTest
606.     testError2.append(errTest)
607. plt.plot(numIters, trainError2, label='Training Error')
608. plt.plot(numIters, testError2, label = 'Testing Error')
609. plt.xlabel('Number of Estimators')
610. plt.ylabel('Error Rate')
611. plt.legend(('Train Error', 'Test Error'))
612. plt.grid(True)
613. plt.show()
614. #best with fs 2-class
615. clf = RandomForestClassifier(min_samples_split=2, n_estimators=10 ,random_state=111)
616. clf.fit(train2, trainclass2)
617. testp = clf.predict(test2)
618. trainp = clf.predict(train2)
619. print('Train Accuracy: ', clf.score(train2, trainclass2))
620. print(classification_report(trainclass2, trainp))
621. print(' ')
622. print('Test Accuracy: ', clf.score(test2, testclass2))
623. print(classification_report(testclass2, testp))
624. print('-----------------------')
625.
626. #random forest with 3 class w/o feature selection
627. n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
628. testError3f = []
629. trainError3f =[]
```

```
    numIters = []
    for i in n_est:
        numIters.append(i)
        clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
        clf.fit(train3, trainclass3)
        accTrain = clf.score(train3, trainclass3)
        errTrain = 1 - accTrain
        trainError3f.append(errTrain)
        accTest = clf.score(test3, testclass3)
        errTest = 1 - accTest
        testError3f.append(errTest)
    plt.plot(numIters, trainError3f, label='Training Error')
    plt.plot(numIters, testError3f, label = 'Testing Error')
    plt.xlabel('Number of Estimators')
    plt.ylabel('Error Rate')
    plt.legend(('Train Error', 'Test Error'))
    plt.grid(True)
    plt.show()

    #best with fs 3-class
    clf = RandomForestClassifier(min_samples_split=2, n_estimators=10,random_state=111)
    clf.fit(train3, trainclass3)
    testp = clf.predict(test3)
    trainp = clf.predict(train3)
    print('Train Accuracy: ', clf.score(train3, trainclass3))
    print(classification_report(trainclass3, trainp))
    print(' ')
    print('Test Accuracy: ', clf.score(test3, testclass3))
    print(classification_report(testclass3, testp))
    print('-----------------------')

    #best decision tree w/o feature selection 3-class
    from sklearn import tree
    tree = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
    tree = tree.fit(train3, trainclass3)
    trainP = tree.predict(train3)
    testP = tree.predict(test3)
    print('Number of Min Splits: ', 100)
    print(' ')
    print('Train Accuracy: ', tree.score(train3, trainclass3))
    print(classification_report(trainclass3, trainP))
    print(' ')
    print('Test Accuracy: ', tree.score(test3, testclass3))
```

```
673.  print(classification_report(testclass3, testP))
674.  print('-----------------------')
675.  #best decision tree w/o feature selection 2-class
676.  from sklearn import tree
677.  tree = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
678.  tree = tree.fit(train2, trainclass2)
679.  trainP = tree.predict(train2)
680.  testP = tree.predict(test2)
681.  print('Number of Min Splits: ', 100)
682.  print(' ')
683.  print('Train Accuracy: ', tree.score(train2, trainclass2))
684.  print(classification_report(trainclass2, trainP))
685.  print(' ')
686.  print('Test Accuracy: ', tree.score(test2, testclass2))
687.  print(classification_report(testclass2, testP))
688.  print('-----------------------')
689.
690.  #tree visual
691.  clf = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
692.  clf = clf.fit(train2, trainclass2)
693.  tree.export_graphviz(clf, out_file='tree.dot')
694.  dot_data = tree.export_graphviz(clf, max_depth=3, out_file=None,
695.                                  feature_names=list(downTrain2.columns.values),
696.                                  class_names=['Class 0', 'Class 1'],
697.                                  filled=True, rounded=True,
698.                                  special_characters=True)
699.  graph = graphviz.Source(dot_data)
700.  graph.render('Compensation')
701.  graph
702.
703.  dot_data = tree.export_graphviz(clf, max_depth=2, out_file=None,
704.                                  feature_names=list(downTrain2.columns.values),
705.                                  class_names=['Class 0', 'Class 1'],
706.                                  filled=True, rounded=True,
707.                                  special_characters=True)
708.  graph = graphviz.Source(dot_data)
709.  graph.render('Compensation2')
710.  graph
711.
712.  clf = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
713.  clf = clf.fit(train3, trainclass3)
714.  tree.export_graphviz(clf, out_file='tree.dot')
715.  dot_data = tree.export_graphviz(clf, max_depth=2, out_file=None,
```

```
                    feature_names=list(downTrain2.columns.values),
                    class_names=['Class 0', 'Class 1', 'Class 2'],
                    filled=True, rounded=True,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph.render('Compensation3_2')
graph

clf = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_split = 100)
clf = clf.fit(train3, trainclass3)
tree.export_graphviz(clf, out_file='tree.dot')
dot_data = tree.export_graphviz(clf, max_depth=3, out_file=None,
                    feature_names=list(downTrain2.columns.values),
                    class_names=['Class 0', 'Class 1', 'Class 2'],
                    filled=True, rounded=True,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph.render('Compensation3_3')
graph


#Random Forest-

#random forest with 2 class with feature selection
from sklearn.ensemble import RandomForestClassifier
n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
testError2 = []
trainError2 =[]
numIters = []
for i in n_est:
    numIters.append(i)
    clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
    clf.fit(train2, trainclass2)
    accTrain = clf.score(train2, trainclass2)
    errTrain = 1 - accTrain
    trainError2.append(errTrain)
    accTest = clf.score(test2, testclass2)
    errTest = 1 - accTest
    testError2.append(errTest)
plt.plot(numIters, trainError2, label='Training Error')
plt.plot(numIters, testError2, label = 'Testing Error')
plt.xlabel('Number of Estimators')
plt.ylabel('Error Rate')
```

```
759.    plt.legend(('Train Error', 'Test Error'))
760.    plt.grid(True)
761.    plt.show()
762.    #best with fs 2-class
763.    clf = RandomForestClassifier(min_samples_split=2, n_estimators=10,random_state=111)
764.    clf.fit(train2, trainclass2)
765.    testp = clf.predict(test2)
766.    trainp = clf.predict(train2)
767.    print('Train Accuracy: ', clf.score(train2, trainclass2))
768.    print(classification_report(trainclass2, trainp))
769.    print(' ')
770.    print('Test Accuracy: ', clf.score(test2, testclass2))
771.    print(classification_report(testclass2, testp))
772.    print('-----------------------')
773.
774.    #random forest with 3 class with feature selection
775.    n_est = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,25,30,50]
776.    testError3f = []
777.    trainError3f =[]
778.    numIters = []
779.    for i in n_est:
780.        numIters.append(i)
781.        clf = RandomForestClassifier(min_samples_split=2, n_estimators=i,random_state=111)
782.        clf.fit(train3, trainclass3)
783.        accTrain = clf.score(train3, trainclass3)
784.        errTrain = 1 - accTrain
785.        trainError3f.append(errTrain)
786.        accTest = clf.score(test3, testclass3)
787.        errTest = 1 - accTest
788.        testError3f.append(errTest)
789.    plt.plot(numIters, trainError3f, label='Training Error')
790.    plt.plot(numIters, testError3f, label = 'Testing Error')
791.    plt.xlabel('Number of Estimators')
792.    plt.ylabel('Error Rate')
793.    plt.legend(('Train Error', 'Test Error'))
794.    plt.grid(True)
795.    plt.show()
796.    #best with fs 3-class
797.    clf = RandomForestClassifier(min_samples_split=2, n_estimators=15,random_state=111)
798.    clf.fit(train3, trainclass3)
799.    testp = clf.predict(test3)
800.    trainp = clf.predict(train3)
801.    print('Train Accuracy: ', clf.score(train3, trainclass3))
```

```
802.    print(classification_report(trainclass3, trainp))
803.    print(' ')
804.    print('Test Accuracy: ', clf.score(test3, testclass3))
805.    print(classification_report(testclass3, testp))
806.    print('-----------------------')
807.
808.
809.    #SVM-
810.
811.    #svm on the three class w/o feature selected
812.    from sklearn import svm
813.    sv3 = svm.SVC(kernel = 'rbf')
814.    sv3.fit(train3, trainclass3)
815.    accTrain = sv3.score(train3,trainclass3)
816.    accTest = sv3.score(test3, testclass3)
817.    trainP = sv3.predict(train3)
818.    testP = sv3.predict(test3)
819.    print('Train Accuracy: ', accTrain)
820.    print(classification_report(trainclass3, trainP))
821.    print('--------------------------')
822.    print('Test Accuracy: ', accTest)
823.    print(classification_report(testclass3, testP))
824.
825.    #analyzing support vectors
826.    index_support = sv3.support_
827.    trainarray = np.array(train3)
828.    trainclassarray = np.array(trainclass3)
829.    c_1 = []
830.    c_0 = []
831.    c_2 = []
832.    for i in index_support:
833.        if trainclassarray[i] == 0:
834.            c_0.append(trainarray[i])
835.
836.        elif trainclassarray[i] == 1:
837.            c_1.append(trainarray[i])
838.        else:
839.            c_2.append(trainarray[i])
840.    print(len(c_0), ' ', len(c_1), ' ', len(c_2))
841.    female = 0
842.    male = 0
843.    white = 0
844.    not_white = 0
```

```
    percent_admin = []
    state_exp = []
    assmnt = []
    for row in c_0:
        percent_admin.append(row[2])
        state_exp.append(row[5])
        assmnt.append(row[7])
        if int(row[112]) == 1:
            female +=1
        else:
            male += 1
        if int(row[120]) ==1:
            white +=1
        else:
            not_white += 1
    print(len(assmnt))
    print('Class 0: ')
    print('------------------------------')
    print('Number of Females: ', female)
    print('------------------------------')
    print('Number of Males: ', male)
    print('------------------------------')
    print('Number of White: ', white)
    print('------------------------------')
    print('Number of Not-White: ', not_white)
    print('------------------------------')
    print('Mean State Experience: ', sum(state_exp)/len(state_exp))
    print('Min State Exp.: ', min(state_exp), ' Max State Exp.: ', max(state_exp))
    print('------------------------------')
    print('Mean Percent_Admin: ', sum(percent_admin)/len(percent_admin))
    print('Min % Admin: ', min(percent_admin), ' Max % Admin: ', max(percent_admin))
    print('------------------------------')
    print('Mean Number of Assignments: ', sum(assmnt)/len(assmnt))
    print('Min Num. Assign.: ', min(assmnt), ' Max Num. Assign.: ', max(assmnt))

    female1 = 0
    male1 = 0
    white1 = 0
    not_white1 = 0
    percent_admin1 = []
    state_exp1 = []
    assmnt1 = []
    for row in c_1:
```

```
        percent_admin1.append(row[2])
        state_exp1.append(row[5])
        assmnt1.append(row[7])
        if int(row[112]) == 1:
            female1 +=1
        else:
            male1 += 1
        if int(row[120]) ==1:
            white1 +=1
        else:
            not_white1 += 1
    print(len(assmnt1))
    print('Class 1: ')
    print('------------------------------')
    print('Number of Females: ', female1)
    print('------------------------------')
    print('Number of Males: ', male1)
    print('------------------------------')
    print('Number of White: ', white1)
    print('------------------------------')
    print('Number of Not-White: ', not_white1)
    print('------------------------------')
    print('Mean State Experience: ', sum(state_exp1)/len(state_exp1))
    print('Min State Exp.: ', min(state_exp1), ' Max State Exp.: ', max(state_exp1))
    print('------------------------------')
    print('Mean Percent_Admin: ', sum(percent_admin1)/len(percent_admin1))
    print('Min % Admin: ', min(percent_admin1), ' Max % Admin: ', max(percent_admin1))
    print('------------------------------')
    print('Mean Number of Assignments: ', sum(assmnt1)/len(assmnt1))
    print('Min Num. Assign.: ', min(assmnt1), ' Max Num. Assign.: ', max(assmnt1))
    female2 = 0
    male2 = 0
    white2 = 0
    not_white2 = 0
    percent_admin2 = []
    state_exp2 = []
    assmnt2 = []
    for row in c_2:
        percent_admin2.append(row[2])
        state_exp2.append(row[5])
        assmnt2.append(row[7])
        if int(row[112]) == 1:
            female2 +=1
```

```
931.        else:
932.            male2 += 1
933.        if int(row[120]) ==1:
934.            white2 +=1
935.        else:
936.            not_white2 += 1
937.    print(len(assmnt2))
938.    print('Class 2: ')
939.    print('-----------------------------')
940.    print('Number of Females: ', female2)
941.    print('-----------------------------')
942.    print('Number of Males: ', male2)
943.    print('-----------------------------')
944.    print('Number of White: ', white2)
945.    print('-----------------------------')
946.    print('Number of Not-White: ', not_white2)
947.    print('-----------------------------')
948.    print('Mean State Experience: ', sum(state_exp2)/len(state_exp2))
949.    print('Min State Exp.: ', min(state_exp2), ' Max State Exp.: ', max(state_exp2))
950.    print('-----------------------------')
951.    print('Mean Percent_Admin: ', sum(percent_admin2)/len(percent_admin2))
952.    print('Min % Admin: ', min(percent_admin2), ' Max % Admin: ', max(percent_admin2))
953.    print('-----------------------------')
954.    print('Mean Number of Assignments: ', sum(assmnt2)/len(assmnt2))
955.    print('Min Num. Assign.: ', min(assmnt2), ' Max Num. Assign.: ', max(assmnt2))
956.
957.    plt.boxplot([state_exp, state_exp1, state_exp2], labels = ['Class 0 ', 'Class 1', 'Class 2'],
vert=False, whis = 30)
958.    plt.xlabel('State Experience - Years')
959.    plt.show()
960.    plt.hist(state_exp2)
961.    plt.xlabel('State Experience - Years')
962.    plt.ylabel('Counts')
963.    plt.grid(True)
964.    plt.show()
965.
966.    females = [female, female1, female2]
967.    males = [male, male1, male2]
968.    whites = [white, white1, white2]
969.    not_whites = [not_white, not_white1, not_white2]
970.    classes = ('Class 0', 'Class 1', 'Class 2')
971.
972.    x = np.arange(3)
```

```
973.    plt.bar(x, females)
974.    plt.xticks(x, classes)
975.    plt.bar(x, males)
976.    plt.ylabel('Number of Females & Males')
977.    plt.legend(["Female", "Male"])
978.    plt.show()
979.
980.    x = np.arange(3)
981.    plt.bar(x, whites)
982.    plt.xticks(x, classes)
983.    plt.bar(x, not_whites)
984.    plt.ylabel('Number of Whites & Minorities')
985.    plt.legend(["White", "Minority"])
986.    plt.show()
987.
988.    print('Class 0')
989.    print('Percent Female ', female/(female+male))
990.    print('Percent Minority ', not_white/(white+not_white))
991.
992.    print('Class 1')
993.    print('Percent Female ', female1/(female1+male1))
994.    print('Percent Minority ', not_white1/(white1+not_white1))
995.
996.    print('Class 2')
997.    print('Percent Female ', female2/(female2+male2))
998.    print('Percent Minority ', not_white2/(white2+not_white2))
999.
1000.   plt.hist(state_exp2)
1001.   plt.xlabel('State Experience - Years')
1002.   plt.ylabel('Counts')
1003.   plt.grid(True)
1004.   plt.show()
1005.
1006.   plt.hist(state_exp1)
1007.   plt.xlabel('State Experience - Years')
1008.   plt.ylabel('Counts')
1009.   plt.grid(True)
1010.   plt.show()
1011.
1012.   plt.hist(state_exp)
1013.   plt.xlabel('State Experience - Years')
1014.   plt.ylabel('Counts')
1015.   plt.grid(True)
```

```
plt.show()

pac0 = 0
for value in percent_admin:
    if int(value) !=0:
        pac0 += 1
pac0

pac1 = 0
for value in percent_admin1:
    if int(value) !=0:
        pac1 += 1
pac1

pac2 = 0
for value in percent_admin2:
    if int(value) !=0:
        pac2 += 1
pac2

(pac0/13854) + (pac1/29780) + (pac2/17429)
```

[16] full model coefficients

Variable	Coefficient	Std Err	t	P>|t|	[0.025	0.975]
const	-1987000.00	9471.48	-209.81	0.00	-2.01E+06	-1.97E+06
Year	1584.93	4.01	395.50	0.00	1.58E+03	1.59E+03
Percent_Administration	235.27	3.43	68.70	0.00	2.29E+02	2.42E+02
State_Experience	1193.28	1.24	960.12	0.00	1.19E+03	1.20E+03
Out_of_State_Experience	874.64	6.34	137.96	0.00	8.62E+02	8.87E+02
Gender_F	-995700.00	4735.99	-210.24	0.00	-1.00E+06	-9.86E+05
Gender_M	-991500.00	4735.53	-209.37	0.00	-1.00E+06	-9.82E+05
Race_Asian or Pacific Islander	-760.32	204.27	-3.72	0.00	-1.16E+03	-3.60E+02
Race_Black	-1101.10	180.22	-6.11	0.00	-1.45E+03	-7.48E+02
Race_Hispanic	-589.60	184.39	-3.20	0.00	-9.51E+02	-2.28E+02
Race_Multiple	-1462.34	339.72	-4.30	0.00	-2.13E+03	-7.96E+02
Race_Native American	-235.30	330.78	-0.71	0.48	-8.84E+02	4.13E+02
Race_White	-957.95	176.61	-5.42	0.00	-1.30E+03	-6.12E+02
Highest_Degree_Achieved_Baccalaureate	-6253.14	11300.00	-0.55	0.58	-2.84E+04	1.59E+04
Highest_Degree_Achieved_C.A.S., Sepecialist, 6 Year Certificate	6030.31	11300.00	0.53	0.59	-1.61E+04	2.82E+04
Highest_Degree_Achieved_Doctorate	9465.09	11300.00	0.84	0.40	-1.27E+04	3.16E+04
Highest_Degree_Achieved_Masters	2671.55	11300.00	0.24	0.81	-1.95E+04	2.48E+04

Highest_Degree_Achieved_None	-8683.33	11300.00	-0.77	0.44	-3.08E+04	1.35E+04
Highest_Degree_Achieved_Registered Nurse	-10160.00	11300.00	-0.90	0.37	-3.24E+04	1.21E+04
Primary_Assignment_Group_Administration	-161300.00	799.16	-201.90	0.00	-1.63E+05	-1.60E+05
Primary_Assignment_Group_Business/Trade	-164300.00	792.58	-207.32	0.00	-1.66E+05	-1.63E+05
Primary_Assignment_Group_Elective	-165700.00	790.28	-209.62	0.00	-1.67E+05	-1.64E+05
Primary_Assignment_Group_Facilities	-163200.00	805.19	-202.67	0.00	-1.65E+05	-1.62E+05
Primary_Assignment_Group_Foreign Language	-166700.00	791.54	-210.60	0.00	-1.68E+05	-1.65E+05
Primary_Assignment_Group_General Education	-169400.00	788.86	-214.68	0.00	-1.71E+05	-1.68E+05
Primary_Assignment_Group_Language Arts	-166500.00	790.36	-210.61	0.00	-1.68E+05	-1.65E+05
Primary_Assignment_Group_Librarian	-166100.00	794.09	-209.19	0.00	-1.68E+05	-1.65E+05
Primary_Assignment_Group_STEM	-165300.00	790.37	-209.15	0.00	-1.67E+05	-1.64E+05
Primary_Assignment_Group_Social Studies	-165800.00	792.27	-209.24	0.00	-1.67E+05	-1.64E+05
Primary_Assignment_Group_Special Education	-168000.00	790.97	-212.43	0.00	-1.70E+05	-1.66E+05
Primary_Assignment_Group_Student Support	-164900.00	791.02	-208.51	0.00	-1.66E+05	-1.63E+05
District_County_Adams	-2879.56	5654.99	-0.51	0.61	-1.40E+04	8.20E+03
District_County_Alexander	-2318.61	5667.08	-0.41	0.68	-1.34E+04	8.79E+03
District_County_Bond	2775.20	5663.21	0.49	0.62	-8.32E+03	1.39E+04
District_County_Boone	9873.17	5655.15	1.75	0.08	-1.21E+03	2.10E+04
District_County_Brown	-7378.35	5681.53	-1.30	0.19	-1.85E+04	3.76E+03
District_County_Bureau	-198.58	5656.19	-0.04	0.97	-1.13E+04	1.09E+04
District_County_Calhoun	-3150.70	5679.74	-0.56	0.58	-1.43E+04	7.98E+03
District_County_Carroll	2082.04	5660.17	0.37	0.71	-9.01E+03	1.32E+04
District_County_Cass	-3631.98	5661.74	-0.64	0.52	-1.47E+04	7.46E+03
District_County_Champaign	3184.01	5653.54	0.56	0.57	-7.90E+03	1.43E+04
District_County_Christian	429.14	5655.73	0.08	0.94	-1.07E+04	1.15E+04
District_County_Clark	-4824.49	5660.41	-0.85	0.39	-1.59E+04	6.27E+03
District_County_Clay	-3962.34	5660.93	-0.70	0.48	-1.51E+04	7.13E+03
District_County_Clinton	3532.75	5657.15	0.62	0.53	-7.56E+03	1.46E+04
District_County_Coles	-1179.38	5655.35	-0.21	0.84	-1.23E+04	9.90E+03
District_County_Cook	19720.00	5652.73	3.49	0.00	8.64E+03	3.08E+04
District_County_Crawford	-5869.95	5659.43	-1.04	0.30	-1.70E+04	5.22E+03
District_County_Cumberland	-4391.80	5664.61	-0.78	0.44	-1.55E+04	6.71E+03
District_County_Dekalb	10670.00	5653.99	1.89	0.06	-4.08E+02	2.18E+04
District_County_Dewitt	971.32	5659.89	0.17	0.86	-1.01E+04	1.21E+04
District_County_Douglas	-2827.86	5658.52	-0.50	0.62	-1.39E+04	8.26E+03
District_County_Dupage	21650.00	5652.83	3.83	0.00	1.06E+04	3.27E+04
District_County_Edgar	-3945.24	5658.93	-0.70	0.49	-1.50E+04	7.15E+03
District_County_Edwards	-1889.97	5674.41	-0.33	0.74	-1.30E+04	9.23E+03
District_County_Effingham	-1477.94	5656.59	-0.26	0.79	-1.26E+04	9.61E+03
District_County_Fayette	505.73	5659.60	0.09	0.93	-1.06E+04	1.16E+04
District_County_Ford	527.49	5659.33	0.09	0.93	-1.06E+04	1.16E+04

District_County_Franklin	7785.09	5655.94	1.38	0.17	-3.30E+03	1.89E+04
District_County_Fulton	-973.26	5655.75	-0.17	0.86	-1.21E+04	1.01E+04
District_County_Gallatin	3141.63	5677.29	0.55	0.58	-7.99E+03	1.43E+04
District_County_Greene	-4056.34	5662.01	-0.72	0.47	-1.52E+04	7.04E+03
District_County_Grundy	7842.09	5654.75	1.39	0.17	-3.24E+03	1.89E+04
District_County_Hamilton	-98.87	5671.11	-0.02	0.99	-1.12E+04	1.10E+04
District_County_Hancock	-7374.58	5658.45	-1.30	0.19	-1.85E+04	3.72E+03
District_County_Hardin	-720.79	5688.39	-0.13	0.90	-1.19E+04	1.04E+04
District_County_Henderson	-1804.84	5671.62	-0.32	0.75	-1.29E+04	9.31E+03
District_County_Henry	3031.59	5654.60	0.54	0.59	-8.05E+03	1.41E+04
District_County_Iroquois	-1258.00	5656.58	-0.22	0.82	-1.23E+04	9.83E+03
District_County_Jackson	5235.14	5655.36	0.93	0.36	-5.85E+03	1.63E+04
District_County_Jasper	-1146.47	5662.76	-0.20	0.84	-1.22E+04	9.95E+03
District_County_Jefferson	583.98	5656.20	0.10	0.92	-1.05E+04	1.17E+04
District_County_Jersey	3801.20	5661.09	0.67	0.50	-7.29E+03	1.49E+04
District_County_Jo Daviess	1723.43	5658.05	0.31	0.76	-9.37E+03	1.28E+04
District_County_Johnson	1771.68	5664.53	0.31	0.75	-9.33E+03	1.29E+04
District_County_Kane	15630.00	5652.91	2.77	0.01	4.55E+03	2.67E+04
District_County_Kankakee	3663.91	5653.92	0.65	0.52	-7.42E+03	1.47E+04
District_County_Kendall	9542.17	5653.89	1.69	0.09	-1.54E+03	2.06E+04
District_County_Knox	651.27	5655.26	0.12	0.91	-1.04E+04	1.17E+04
District_County_La Salle	5219.78	5653.89	0.92	0.36	-5.86E+03	1.63E+04
District_County_Lake	16520.00	5652.85	2.92	0.00	5.44E+03	2.76E+04
District_County_Lawrence	-6364.50	5661.41	-1.12	0.26	-1.75E+04	4.73E+03
District_County_Lee	2755.80	5657.05	0.49	0.63	-8.33E+03	1.38E+04
District_County_Livingston	3574.63	5655.93	0.63	0.53	-7.51E+03	1.47E+04
District_County_Logan	2373.55	5657.98	0.42	0.68	-8.72E+03	1.35E+04
District_County_Macon	2534.67	5654.07	0.45	0.65	-8.55E+03	1.36E+04
District_County_Macoupin	3260.82	5655.32	0.58	0.56	-7.82E+03	1.43E+04
District_County_Madison	7936.20	5653.25	1.40	0.16	-3.14E+03	1.90E+04
District_County_Marion	842.11	5655.38	0.15	0.88	-1.02E+04	1.19E+04
District_County_Marshall	-2182.46	5662.64	-0.39	0.70	-1.33E+04	8.92E+03
District_County_Mason	-888.53	5659.21	-0.16	0.88	-1.20E+04	1.02E+04
District_County_Massac	1346.48	5662.02	0.24	0.81	-9.75E+03	1.24E+04
District_County_Mcdonough	-2002.50	5657.27	-0.35	0.72	-1.31E+04	9.09E+03
District_County_Mchenry	12500.00	5653.14	2.21	0.03	1.42E+03	2.36E+04
District_County_Mclean	6454.30	5653.57	1.14	0.25	-4.63E+03	1.75E+04
District_County_Menard	-1857.57	5662.05	-0.33	0.74	-1.30E+04	9.24E+03
District_County_Mercer	-3390.19	5659.06	-0.60	0.55	-1.45E+04	7.70E+03
District_County_Monroe	8946.85	5657.54	1.58	0.11	-2.14E+03	2.00E+04
District_County_Montgomery	4920.61	5658.18	0.87	0.38	-6.17E+03	1.60E+04

District_County_Morgan	-2325.39	5656.31	-0.41	0.68	-1.34E+04	8.76E+03
District_County_Moultrie	-4007.63	5663.42	-0.71	0.48	-1.51E+04	7.09E+03
District_County_Ogle	7270.61	5654.84	1.29	0.20	-3.81E+03	1.84E+04
District_County_Peoria	6316.73	5653.50	1.12	0.26	-4.76E+03	1.74E+04
District_County_Perry	3566.16	5660.49	0.63	0.53	-7.53E+03	1.47E+04
District_County_Piatt	-2392.48	5660.41	-0.42	0.67	-1.35E+04	8.70E+03
District_County_Pike	-6521.01	5659.71	-1.15	0.25	-1.76E+04	4.57E+03
District_County_Pope	-2479.98	5687.07	-0.44	0.66	-1.36E+04	8.67E+03
District_County_Pulaski	-3808.89	5665.07	-0.67	0.50	-1.49E+04	7.29E+03
District_County_Putnam	-2553.55	5674.41	-0.45	0.65	-1.37E+04	8.57E+03
District_County_Randolph	1630.11	5657.11	0.29	0.77	-9.46E+03	1.27E+04
District_County_Richland	-2713.91	5660.66	-0.48	0.63	-1.38E+04	8.38E+03
District_County_Rock Island	12370.00	5653.76	2.19	0.03	1.29E+03	2.35E+04
District_County_Saint Clair	12480.00	5653.24	2.21	0.03	1.40E+03	2.36E+04
District_County_Saline	3902.17	5658.10	0.69	0.49	-7.19E+03	1.50E+04
District_County_Sangamon	4984.18	5653.38	0.88	0.38	-6.10E+03	1.61E+04
District_County_Schuyler	-5452.65	5670.49	-0.96	0.34	-1.66E+04	5.66E+03
District_County_Scott	-4986.95	5671.63	-0.88	0.38	-1.61E+04	6.13E+03
District_County_Shelby	-2038.08	5660.29	-0.36	0.72	-1.31E+04	9.06E+03
District_County_Stark	-2954.31	5670.18	-0.52	0.60	-1.41E+04	8.16E+03
District_County_Stephenson	1700.78	5655.53	0.30	0.76	-9.38E+03	1.28E+04
District_County_Tazewell	2226.84	5653.84	0.39	0.69	-8.85E+03	1.33E+04
District_County_Union	468.05	5659.86	0.08	0.93	-1.06E+04	1.16E+04
District_County_Vermilion	1861.41	5654.28	0.33	0.74	-9.22E+03	1.29E+04
District_County_Wabash	-4055.29	5664.59	-0.72	0.47	-1.52E+04	7.05E+03
District_County_Warren	-4819.97	5660.73	-0.85	0.40	-1.59E+04	6.27E+03
District_County_Washington	3655.87	5663.50	0.65	0.52	-7.44E+03	1.48E+04
District_County_Wayne	-4066.34	5660.46	-0.72	0.47	-1.52E+04	7.03E+03
District_County_White	649.91	5659.04	0.12	0.91	-1.04E+04	1.17E+04
District_County_Whiteside	5102.21	5654.92	0.90	0.37	-5.98E+03	1.62E+04
District_County_Will	12090.00	5652.91	2.14	0.03	1.01E+03	2.32E+04
District_County_Williamson	7975.65	5655.13	1.41	0.16	-3.11E+03	1.91E+04
District_County_Winnebago	10040.00	5653.20	1.78	0.08	-1.04E+03	2.11E+04
District_County_Woodford	2228.20	5655.71	0.39	0.69	-8.86E+03	1.33E+04