BANA7038 final exam evaluation

Frankenhoff, Brenda 50%

Dong, Juntao 50% (evaluator)

**Final Project Report:**

**StackOverflow 2018 Salary Survey Data**

**Summary:**

This project *was to* review data submitted by developers from all over the world in 2018 to the published salary survey ( <https://insights.stackoverflow.com/survey/2018/#overview>) on their experience and salary to see if we could determine a valid model to predict anticipated salary for an individual with a given number of years of experience and the specific skill set.

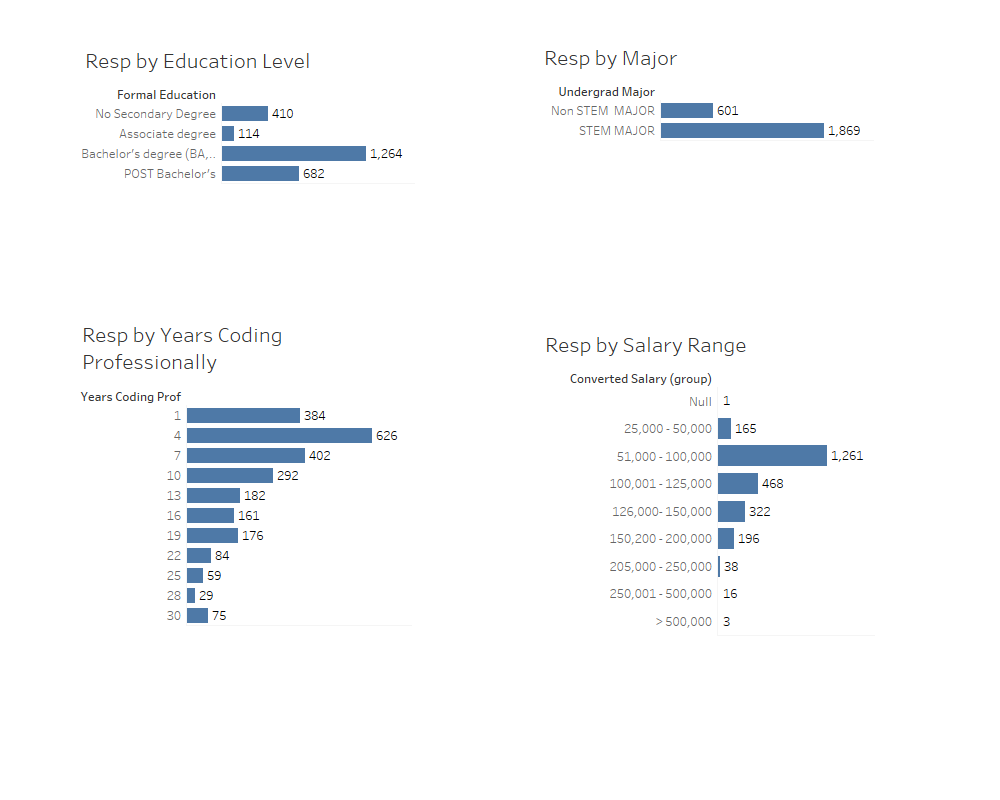
We found that the survey provided much data to choose from but finding the correlations and items that directly impacted salary were much more difficult than anticipated. The sample of data also provided challenges as much data required either exclusion or remapping. Over 100,000 individuals submitted responses, but we narrowed the data set down to only US developers. Our final conclusions were that we could not adequately predict salary for a given individual based upon their experience and skill set.

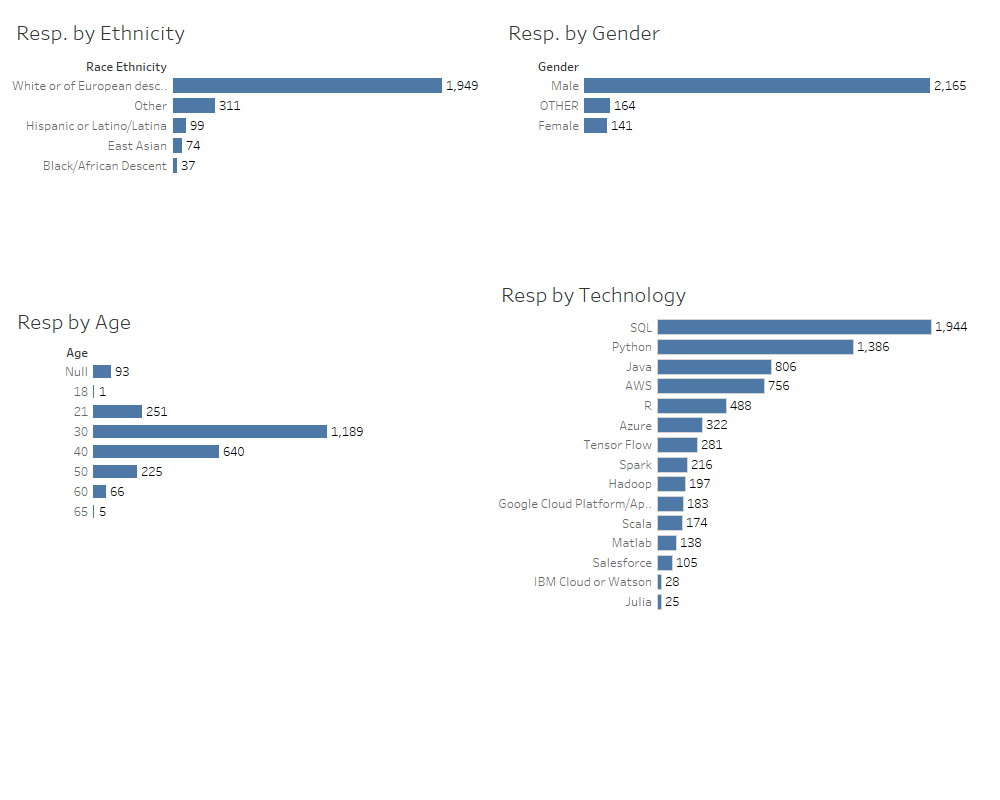
**CHAPTER 1: Data exploration and data cleansing**

In reviewing the data, we again limited the developers to only those in the Data Science/Data Analysis roles. This left us with about 2500 respondents. We also initially chose to break out multiselect fields (one field - many values) into separate fields for languages, frameworks, platforms and databases. Other items included in our data cleansing process:

* Salaries were incorrectly entered, artificially inflating the values. (entering an annual figure but indicating pay is weekly). We reduced our sample to salaries between 25,000USD and 250,000 USD. We also eliminated individuals with the title of CEO/CIO/CFO.
* Gender was an issue. We consolidated what was previously over 10 classifications for gender to Male, Female, and Other. (Male 88%, Female 7% Other 8% of data). We still were unable to see enough significance for the smaller categories and we ended up classifying the respondents as Male and Non-male.
* For Ethnicity, we consolidated multiple ethnicity into its parent category. Again, the percentage of data from non-White individuals was so small, we consolidated into White and Non-white.
* For ages, we selected the lowest value for each range and remapped from a text category to a numeric value.
* For Company size, we again remapped from a text field category range to a numeric value using the lowest number of the range.
* For undergradmajor, we mapped various degrees into two categories, STEM and NONSTEM.
* For Education level, we mapped/grouped into the classes identified below.
* Many of the submission records had missing/incomplete information, where individuals chose not to input salary, or other items requested. We kept these records and allowed the software to handle the null values.
* The features we kept are FormalEducation, UndergradMajor, CompanySize, YearsCodingProf, ConvertedSalary, LanguageWorkedWith, DatabaseWorkedWith, PlatformWorkedWith, FrameworkWorkedWith, HoursComputer, Gender, RaceEthnicity and Age from the original database.

**Summary of data file values**

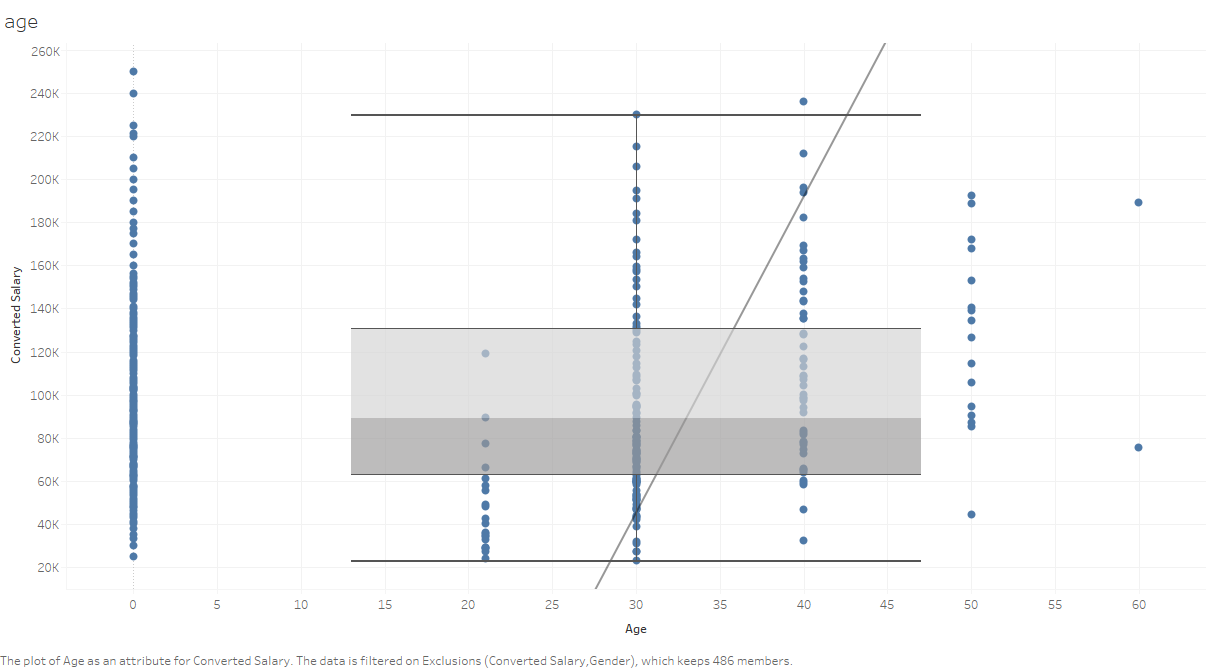




**Pairs from dataset**



Examination of single feature (Age), indicating a linear relationship between Age and Converted Salary.



**Trend Lines Model**

A linear trend model is computed for Converted Salary given Age as an attribute. The model may be significant at p <= 0.05.

**Model formula:**( Converted Salary + intercept )

**Number of modeled observations:** 204

**Number of filtered observations:** 145

**Model degrees of freedom:** 2

**Residual degrees of freedom (DF):** 202

**SSE (sum squared error):** 10466.6

**MSE (mean squared error):** 51.8147

**R-Squared:** 0.168804

**Standard error:** 7.19824

**p-value (significance):** < 0.0001

**Individual trend lines:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Panes** | | **Line** | | **Coefficients** | | | | |
| **Row** | **Column** | **p-value** | **DF** | **Term** | **Value** | **StdErr** | **t-value** | **p-value** |
| Converted Salary | Age | < 0.0001 | 202 | Converted Salary | 6.816e-05 | 1.064e-05 | 6.40494 | < 0.0001 |
|  | | | | intercept | 26.9235 | 1.13487 | 23.7239 | < 0.0001 |

**Chapter 2: Model development and supporting code**

After data cleaning and explorative data analysis, we had a general idea about the dataset. We found that there was no strong or clear relationship between the predictors and the response variable like in most data analysis learning cases. Therefore, the goal was to develop a model to perform the best prediction with the data that we could substantiate.

We started with a dataset that had 30 variables and around 2500 respondents. Initially, we were interested if salary was affected by the individual tools/technology used by the individual employees. We also expected the value of variables like CompanySize, YearsCodingProf, HoursComputer and Age to impact the salary value so we transferred them as numeric variables by using the lower value of each range.

All the numeric features were normalized before feature selection and fitting into models. The response variable, ConvertedSalary, was originally numeric. Later in the project, we also created a new categorical response variable called SalaryRange based on the original data since we couldn’t reach a satisfying result by regression to see if classification would yield a better response.

***Predictors of dataset1 (30 in total):***

* CompanySize
* YearsCodingProf
* HoursComputer
* Age
* Python
* Scala
* Matlab
* SQL
* Julia
* Java
* R
* Salesforce
* IBM Cloud or Watson
* AWS
* Azure
* Google Cloud Platform/App Engine
* TensorFlow
* Torch/PyTorch
* Spark
* Hadoop
* FormalEducation\_Bachelor’s degree (BA, BS, B.Eng., etc.)
* FormalEducation\_No Secondary Degree
* FormalEducation\_POST Bachelor's
* UndergradMajor\_STEM MAJOR
* Gender\_Male
* Gender\_OTHER
* RaceEthnicity\_East Asian
* RaceEthnicity\_Hispanic or Latino/Latina
* RaceEthnicity\_Other
* RaceEthnicity\_White or of European

***Response variable:*** ConvertedSalary / SalaryRange

**Heatmap of pairwise correlation of columns:**

Python code:

corr = df.corr()

fig2, ax = plt.subplots()

fig2.set\_size\_inches(14, 10)

sns.heatmap(abs(corr), xticklabels=corr.columns, yticklabels=corr.columns)

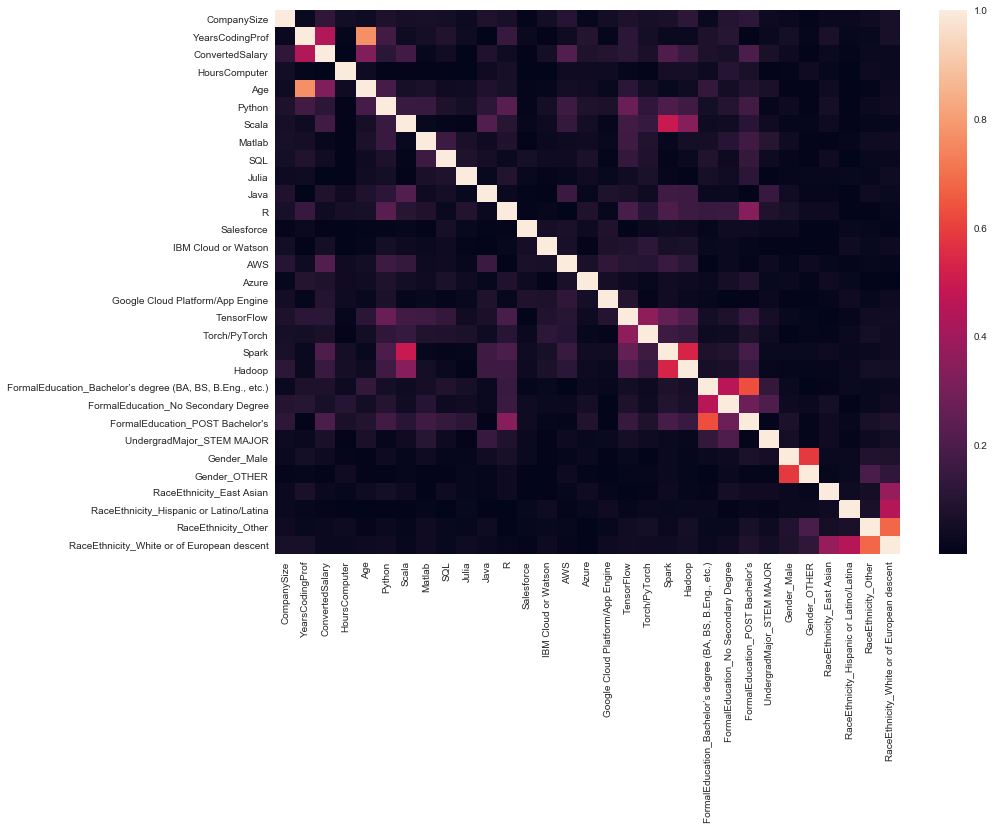
**

Figure 2.1 Heatmap of pairwise correlation of columns

1. **Feature selection:**

Since we had 30 variables at the beginning, too many for the model, and some of the variables could bring noise to the model, we needed to select the subset of variables that would provide better prediction performance. The method we used to do variable selection was **Forward stepwise selection.** Forward Stepwise begins with a model containing no predictors, and then adds predictors to the model, one at the time. At each step, the variable that gives the greatest additional improvement to the fit is added to the model. Here Mallow’s Cp, AIC, BIC and adjusted R2 are introduced to select the number of features to be used.

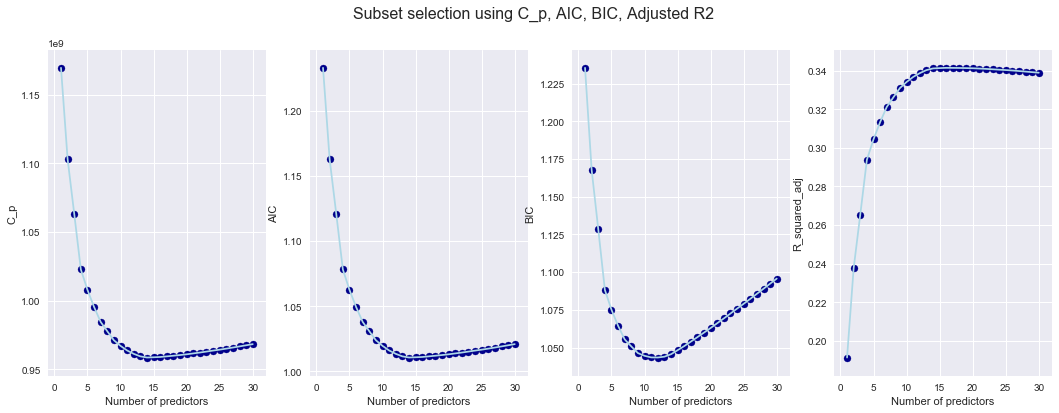


Figure 2.2 Feature selection using Cp, AIC, BIC and adjusted R2

Based on the values shown above, we selected the best 14 predictors for our model, they are:

* YearsCodingProf
* Spark
* AWS
* CompanySize
* Python
* Azure
* FormalEducationPOST\_Bachelor's
* FormalEducation\_Bachelor’s degree (BA, BS, B.Eng., etc.)
* Scala
* Google Cloud Platform/App Engine
* SQL
* FormalEducation\_No Secondary Degree
* TensorFlow
* RaceEthnicity\_East Asian
* UndergradMajor\_STEM MAJOR

1. **Data splitting**

We split data into a training set (80%) and a testing set (20%).

Python code:

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

1. **Regression modelling**
   * 1. **Linear regression**

We first trained linear regression and made predictions with the test data using the following code.

Python code:

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X\_train,y\_train)

y\_pred = lm.predict(X\_test)

We next calculated the MSE of the model prediction and plotted the figure of the fitted value versus the true value.

Python code:

print('MSE:', mean\_squared\_error(y\_test, y\_pred))

fig, ax = plt.subplots()

fig.set\_size\_inches(10, 10)

ax.set\_xlim(0, 250000)

ax.set\_ylim(0, 250000)

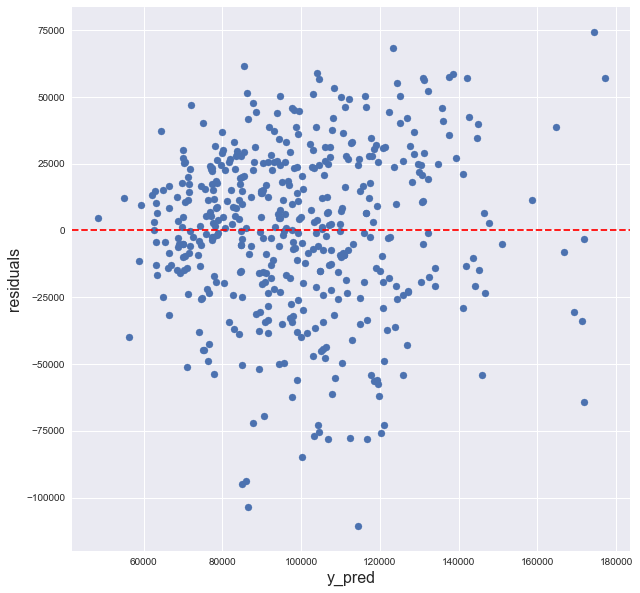
plt.scatter(y\_test, y\_pred)

plt.xlabel('y\_test')

plt.ylabel('y\_pred')

ab = np.linspace(0, 250000, 1000)

plt.plot(ab, ab, linestyle='dashed', color='red')

A close up of a map

Description automatically generated

1. (b)

Figure 2.3 (a) Scatter plot of residual against fitted values from linear regression (b) Scatter plot of linear fitted values against true values

The MSE of the model based on the test data is *955055604*. Figure 2.3(a) is the scatter plot of residuals against fitted values. The dots are evenly distributed around zero with constant variance across the x-axis. Therefore, the linearity and equal variance assumptions are satisfied. Figure 2.3(b) is the scatter plot of fitted values against true values. The angle of the red dashed line is 45 degrees, and the distance between the points and the dashed line represents the prediction accuracy. Smaller distance between the data points and line indicate better prediction and the points located exactly on the line indicate a perfect prediction. As we can see most of the fitted values are within the range between $50,000 and $150,000, whereas the true values fall between $25,000 and $250,000.

* 1. **Ridge regression**

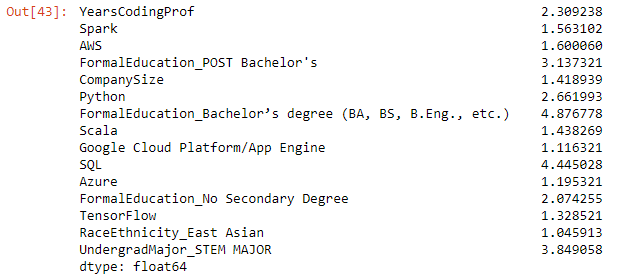
To further improve our linear model, we calculated the Variance Inflation Factors (VIF) for the covariates to uncover any multicollinearity issues.

Python code:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

pd.Series([variance\_inflation\_factor(X\_14.values, i) for i in range(X\_14.shape[1])], index=X\_14.columns)

The output:



General recommendation is that if VIF is greater than 5, then the explanatory variable is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors. From the result above we found that two covariates have VIF near 5 which may indicate a multicollinearity issue, so we decided to give ridge regression a try.

We first plotted the figure of λ against the coefficient estimates to help choose λ. Here we chose two values: λ = 10 and λ = 100.

A screenshot of a cell phone

Description automatically generated

Figure 2.4 Ridge coefficients as a function of the regularization

* **λ = 10 MSE = 975182423**

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

(a) (b)

Figure 2.5 (a) Scatter plot of residual against fitted values from ridge regression with λ =10 (b) Scatter plot of ridge fitted values against true values

* **λ = 100 MSE= 1026741817**

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

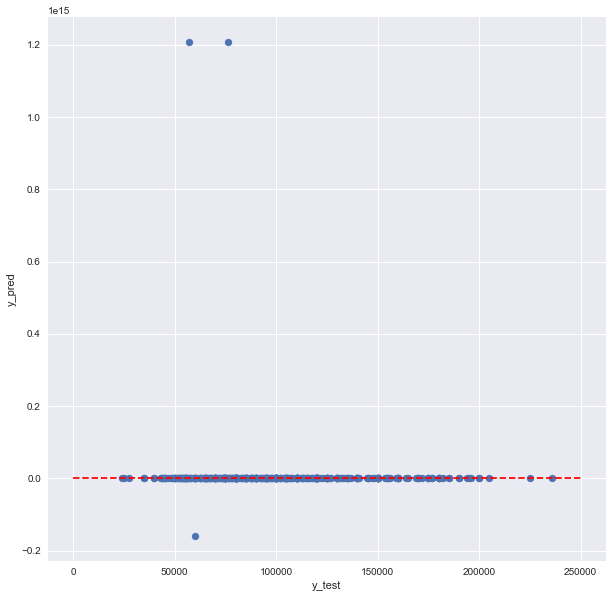
(a) (b)

Figure 2.6 (a) Scatter plot of residual against fitted values from ridge regression with λ =100 (b) Scatter plot of ridge fitted values against true values

We can see there is an obvious trend that the fitted values are focusing towards $100,000 as the λ increases. The MSE also increased, so ridge regression did not help with the improvement of the prediction.

* 1. **Polynomial regression**

Since linear regression and ridge regression did not provide the improvement we expected, we tried polynomial regression with a degree of 2 to see if it would better describe the relationship between covariates and response variable. Below are the two scatter plots as we have shown for each regression model. Obviously, polynomial regression doesn’t fit this dataset and generates an MSE of 7x1028 which is ridiculously large.

A close up of a device

Description automatically generated

(a) (b)

Figure 2.6 (a) Scatter plot of residual against fitted values from ridge regression with λ =100 (b) Scatter plot of ridge fitted values against true values

1. **Classification modelling**

After reviewing the above regression models, we found it difficult to predict a single value that matches well with the true value. Since all covariates are categorical, we thought it might work well if we transfer the response variable, ConvertedSalary, to different ranges, and predict which range the respondent should be in based on all the covariates reported. We used decision tree and k nearest neighbors in this section.

**Salary Ranges:**

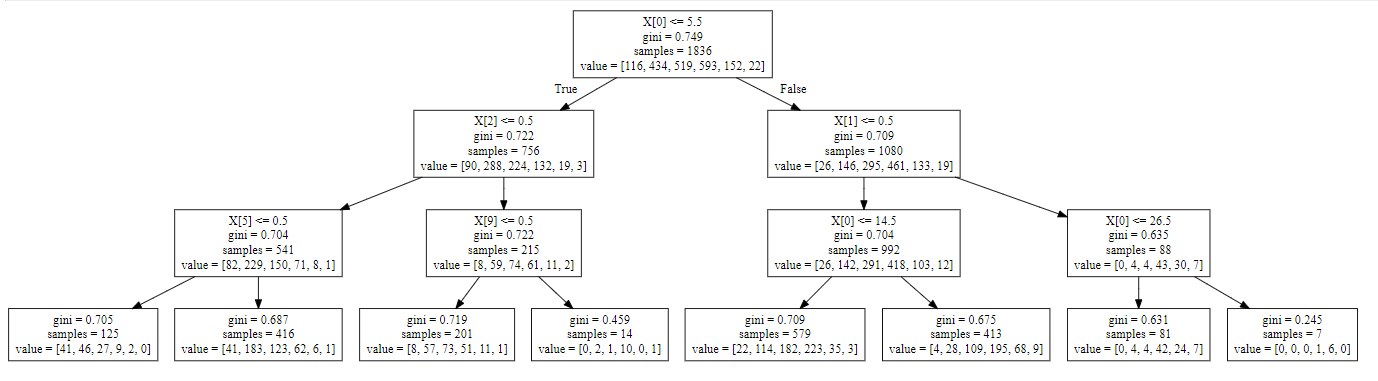
* $25,000 - $50,000
* $50,000 - $75,000
* $75,000 - $100,000
* $100,000 - $150,000
* $150,000 - $200,000
* $200,000 - $250,000
  1. **Decision Tree**

We first used decision tree classification. By tuning the max depth of the decision tree as figure 2.7 shows, we found max\_depth = 3 or 4 provide us the best prediction result. To make the model simpler and easier to explain, we chose max\_depth = 3.

A close up of a person

Description automatically generated

Figure 2.7 Line plot of accuracy of the decision tree classification against max depth of the decision tree



With the decision tree classification, the best accuracy we obtained was 44%. We did not feel it was good enough, but it seemed better than a random guess within the 6 available options (the accuracy is 1/6) without considering the distribution of the salary. Below shows the confusion matrix for the test data.

A screenshot of a cell phone

Description automatically generated

Figure 2.8 Confusion matrix of decision tree classification with max depth of 3

* 1. **K Nearest Neighbors**

Like the decision tree classification, we first defined the value of the k that would provide us the best prediction accuracy. From the figure below, when k =29, we will reach the best accuracy of 42%.

A close up of text on a white background

Description automatically generated

Figure 2.9 Accuracy of the KNN classification against the K value

Below shows the confusion matrix for the test data.

A screenshot of a cell phone

Description automatically generated

Figure 2.10 Confusion matrix of KNN classification with K of 29

**Chapter 3: Observations**

Best model selection:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model name | Simple linear regression | Ridge regression  λ = 10 | Ridge regression  λ = 100 | Polynomial regression |
| MSE | 9.55x108 | 9.75 x108 | 1.03 x109 | 7x1028 |

|  |  |  |
| --- | --- | --- |
| Model name | Decision tree  Max\_depth=3 | K nearest neighbors  K=29 |
| Accuracy | 44% | 42% |

We compared all the regression models based on the MSE and compared the classification models based on the accuracy. We found simple linear regression and decision tree with max depth of 3 provide the best prediction results.

Using the linear model from Chapter 2 section 3.1, we performed prediction on a small subset of the testing data to show the relationship of variables to salary. The coefficients of the linear regression we developed are shown below:

|  |  |
| --- | --- |
| Coefficients | Value |
|  | 52566.11 |
|  | 66776.97 |
|  | 12641.25 |
|  | 13342.42 |
|  | 26545.67 |
|  | 8161.53 |
|  | 5663.25 |
|  | 16471.66 |
|  | 12557.52 |
|  | 9866.98 |
|  | -5521.06 |
|  | 5982.20 |
|  | 9428.34 |
|  | 6109.17 |
|  | 8174.76 |
|  | 1342.67 |

**Data used for predictions:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | YearsCodingProf | Spark | AWS | FormalEducation\_POST Bachelor's | CompanySize | Python | FormalEducation\_Bachelor’s degree (BA, BS, B.Eng., etc.) | Scala | Google Cloud Platform/App Engine | SQL | Azure | FormalEducation\_No Secondary Degree | TensorFlow | RaceEthnicity\_East Asian | UndergradMajor\_STEM MAJOR |
| 0 | 0.620689655 | 0 | 0 | 0 | 0.000500125 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0.206896552 | 0 | 0 | 0 | 0.002750688 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0.206896552 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0.103448276 | 0 | 0 | 0 | 0.149787447 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0.374843711 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Comparison of predicted value against the original data record (test data file).**

|  |  |  |
| --- | --- | --- |
| Respondent Index | Fitted Values | True Values |
| 0 | $97,925.25 | $75,000.00 |
| 1 | $84,361.00 | $80,000.00 |
| 2 | $133,293.93 | $120,000.00 |
| 3 | $77,310.41 | $70,000.00 |
| 4 | $82,313.26 | $60,000.00 |

Since both regression and classification models did not provide us good results with the dataset1, we decided to see if the number of the tools that the developers used could more accurately predict the salary, rather than the individual tools. Using a single variable for the number of tools we didn’t find it provided any better prediction quality. We put the modeling and prediction results for this effort in appendix A.

**Chapter 4: Conclusion**

It is very key to understand the purpose and construction of a survey when using the data for analysis. While this seemed, on the surface, to be straight forward, we found that it was not easy to produce a predictive model to generate a salary estimate based upon a specific set of given variables with the level of accuracy that we would like to have seen. Looking at the StackOverflow report, they segmented the resulting data and did only observational reporting. This dataset provided us the opportunity to use many of the tools introduced in the class to see how they compared in actual analysis against the same dataset.

We did not see a direct correlation with experience, skills and technologies that could accurately predict salary. Because data was self-reported, we really cannot validate the salary accuracy from which to build our model. If data were from employer databases where the skills and salary values were accurately tracked, we might by company or industry, be able to predict salary. The survey results also did not indicate a PRIMARY language/technology, so we are not sure if the values reported actually had influence on the salary or were just part of the individual’s knowledge. We also know (outside of the data) that geography can affect salaries by metro-region. This information was not available in this dataset. At best, this data can be used to report salary for the StackOverflow user base but is not a good set of data for predicting general industry salaries for hopeful graduates.

## Appendix A - Results and visualization of dataset2

A close up of a logo

Description automatically generated

Figure A.1 Heatmap of pairwise correlation of columns

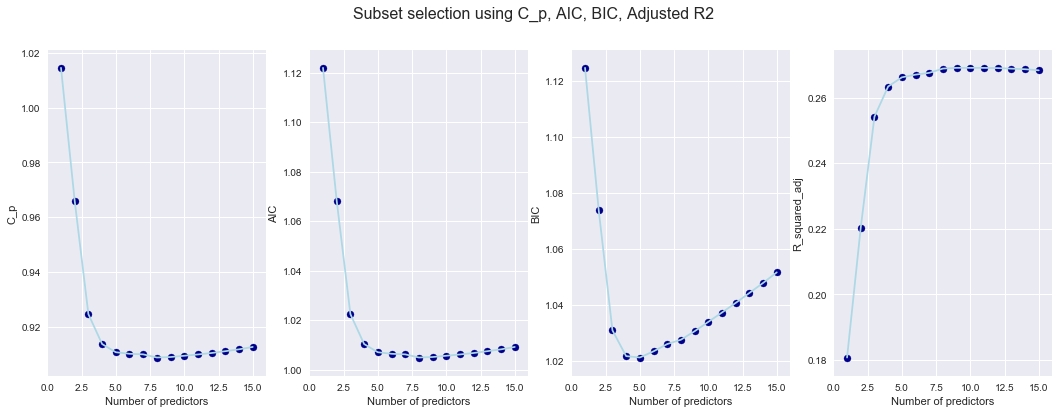


Figure B.2 Feature selection using Cp, AIC, BIC and adjusted R2

Based on the values shown above, the selected covariates are:

* YearsCodingProf
* FormalEducation
* NUMDATABASE
* CompanySize
* RaceEthnicity\_White or of European descent

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

Figure A.3 (a) Scatter plot of residual against fitted values from linear regression (b) Scatter plot of linear fitted values against true values

A close up of a mans face

Description automatically generatedA close up of a piece of paper

Description automatically generated

Figure A.4 Line plot of accuracy of the decision tree classification against max depth of the decision tree and confusion matrix of decision tree classification with max depth of 4

A screenshot of a cell phone

Description automatically generatedA close up of a piece of paper

Description automatically generated

Figure A.5 Accuracy of the KNN classification against the K value and Confusion matrix of KNN classification with K of 32

## Appendix B – Python code for data cleaning

## Import packages

get\_ipython().magic('matplotlib inline')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import matplotlib.ticker as ticker

pd.options.display.max\_rows = 1000

## Load data

df = pd.read\_csv(r'C:\Users\junta\Google Drive\BANA7038\BANA7038 Final Project\stack overflow\survey\_results\_public.csv')

features = ['FormalEducation', 'UndergradMajor', 'CompanySize', 'DevType', 'YearsCodingProf',

'JobSatisfaction', 'Salary', 'SalaryType', 'ConvertedSalary', 'LanguageWorkedWith',

'DatabaseWorkedWith', 'PlatformWorkedWith', 'FrameworkWorkedWith', 'HoursComputer',

'Gender', 'RaceEthnicity', 'Age']

df = df.loc[(df.Country=='United States')&(df.Student=='No')&(df.Employment=='Employed full-time'), features]

df = df.dropna(subset=['Salary', 'SalaryType', 'ConvertedSalary'], thresh=2)

# Filter rows with feature DevType containing 'Data' and not containing 'C-suite'

# Transfer the type of Salary to float (remove coma in the numbers first)

df\_ds = df[df.DevType.str.contains('Data', na=False)]

df\_ds = df\_ds[~df\_ds.DevType.str.contains('C-suite', na=False)].copy()

df\_ds['Salary'] = df\_ds['Salary'].str.replace(',', '').astype('float64')

####################

## Data Cleanning ##

####################

# Some people may input their annual salary but selet wrong salary type since many incomes seem not reliable.

# For SalaryType of Monthly, if salary > $30,000 we will treat it as annual salary and else we think of it as monthly income.

# For SalaryType of Weekly, if salary > $10,000 we treat it as annual salary and else we think of it as weekly income.

lang = ['Python', 'Scala', 'Matlab', 'SQL', 'Julia', 'Java', 'R'] # 'R'

pt = ['Salesforce', 'IBM Cloud or Watson', 'AWS', 'Azure', 'Google Cloud Platform/App Engine']

fw = ['TensorFlow', 'Torch/PyTorch', 'Spark', 'Hadoop']

for index, row in df\_ds.iterrows():

if row['SalaryType'] == 'Monthly' and row['Salary'] >= 30000:

df\_ds.at[index,'ConvertedSalary'] = row['Salary']

elif row['SalaryType'] == 'Weekly' and row['Salary'] >=10000:

df\_ds.at[index,'ConvertedSalary'] = row['Salary']

for item in lang:

if item in str(row['LanguageWorkedWith']).split(';'):

df\_ds.at[index, item] = 1

else:

df\_ds.at[index, item] = 0

for item in pt:

if item in str(row['PlatformWorkedWith']):

df\_ds.at[index, item] = 1

else:

df\_ds.at[index, item] = 0

for item in fw:

if item in str(row['FrameworkWorkedWith']):

df\_ds.at[index, item] = 1

else:

df\_ds.at[index, item] = 0

# if str(row['LanguageWorkedWith'])=='R' or str(row['LanguageWorkedWith']).startswith('R;') or str(row['LanguageWorkedWith']).endswith(';R') or str(row['LanguageWorkedWith']).find(';R;')!=0:

# df\_ds.at[index,'R'] = 1

#else:

# df\_ds.at[index,'R'] = 0

# Delete developers earning salary more than $2,000,000 and less than $20,000.

df\_ds = df\_ds[~((df\_ds.ConvertedSalary>=250000) | (df\_ds.ConvertedSalary<=20000))]

## 2470 rows

df\_ds

size = np.unique(df\_ds['CompanySize'].dropna().values)

size\_int = [3000, 15, 20000, 300, 60, 7500, 750, 5]

mapping1 = dict(zip(size, size\_int))

year = np.unique(df\_ds['YearsCodingProf'].dropna().values)

year\_int = [1, 13, 16, 19, 22, 25, 28, 4, 30, 7, 10]

mapping2 = dict(zip(year, year\_int))

hour = np.unique(df\_ds['HoursComputer'].dropna().values)

hour\_int = [4, 8, 12, 1, 14]

mapping3 = dict(zip(hour, hour\_int))

age = np.unique(df\_ds['Age'].dropna().values)

age\_int = [21, 30, 40, 50, 60, 65, 18]

mapping4 = dict(zip(age, age\_int))

df\_ds = df\_ds.replace({'CompanySize': mapping1, 'YearsCodingProf':mapping2, 'HoursComputer':mapping3, 'Age': mapping4})

# Drop features Salary and SalaryType

df\_ds = df\_ds.drop(['Salary', 'SalaryType', 'DevType', 'JobSatisfaction'], axis=1)

## Create bins for ConvertedSalary:

## bin1: 1 - $25,000 to $50,000

## bin2: 2 - $50,000 to $75,000

## bin3: 3 - $75,000 to $100,000

## bin4: 4 - $100,000 to $150,000

## bin5: 5 - $150,000 to $200,000

## bin6: 6 - $200,000 to $250,000

bins = [24999, 50000, 75000, 100000, 150000, 200000, 250000]

category = pd.cut(df\_ds.ConvertedSalary,bins)

df\_ds['SalaryRange'] = category.to\_frame()

df\_ds.to\_csv('intermediate.csv', index=False)

#sns.distplot(df\_ds['ConvertedSalary'])

sns.set\_style('ticks')

fig, ax = plt.subplots()

fig.set\_size\_inches(15, 7)

sns.distplot(df\_ds['ConvertedSalary'], bins=np.arange(0,1600000,10000), kde=False)

ax.set\_xlabel('Yearly Salary')

ax.set\_ylabel('Count')

ax.set\_title(r'Histogram of Salary of Developer in Data Science Area with Bin Width of $10,000', fontsize=18)

fmt = '${x:,.0f}'

formatter = ticker.StrMethodFormatter(fmt)

ax.xaxis.set\_major\_formatter(formatter)

fig.savefig('hist.png')

sns.set\_style('ticks')

fig, ax = plt.subplots()

fig.set\_size\_inches(15, 7)

sns.distplot(df\_ds['ConvertedSalary'], bins=np.arange(0,1600000,10000), kde=True)

ax.set\_xlabel('Yearly Salary')

ax.set\_ylabel('Density')

ax.set\_title(r'Distribution of Data Science Employees Salary', fontsize=18)

fmt = '${x:,.0f}'

formatter = ticker.StrMethodFormatter(fmt)

ax.xaxis.set\_major\_formatter(formatter)

fig.savefig('dist.png')

np.unique(df\_ds['FormalEducation'].dropna().values)

np.unique(df\_ds['UndergradMajor'].dropna().values)

np.unique(df\_ds['Gender'].dropna().values)

np.unique(df\_ds['RaceEthnicity'].dropna().values)

np.unique(df\_ds['YearsCodingProf'].dropna().values)

np.unique(df\_ds['HoursComputer'].dropna().values)

np.unique(df\_ds['Age'].dropna().values)

lang = df\_ds['LanguageWorkedWith'].dropna().values

lst = []

for line in lang:

langs = line.split(';')

#print(line)

lst.extend(langs)

list(set(lst))

pt = df\_ds['PlatformWorkedWith'].dropna().values

lst = []

for line in pt:

pts = line.split(';')

lst.extend(pts)

list(set(lst))

db = df\_ds['DatabaseWorkedWith'].dropna().values

lst = []

for line in db:

dbs = line.split(';')

lst.extend(dbs)

list(set(lst))

fw = df\_ds['FrameworkWorkedWith'].dropna().values

lst = []

for line in fw:

fws = line.split(';')

lst.extend(fws)

list(set(lst))

## Appendix C – Code of regression modelling for database1

## Import packages

%matplotlib inline

%xmode Verbose

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import matplotlib.ticker as ticker

import statsmodels.api as sm

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import normalize

pd.options.display.max\_rows = 500

df = pd.read\_csv('final cleaned data file 022019.csv')

df = df.dropna()

df = df[~(df.ConvertedSalary>=250000)]

fig = sns.pairplot(df, vars=['CompanySize', 'YearsCodingProf', 'ConvertedSalary', 'HoursComputer', 'Age'])

fig.savefig('pair.png')

df = pd.get\_dummies(df, columns=['FormalEducation', 'UndergradMajor', 'Gender', 'RaceEthnicity'], drop\_first=True)

df = df.drop(['JobSatisfaction'], axis=1)

df = df.astype('float64')

corr = df.corr()

fig2, ax = plt.subplots()

fig2.set\_size\_inches(14, 10)

sns.heatmap(abs(corr), xticklabels=corr.columns, yticklabels=corr.columns)

X1 = df.loc[:, ['CompanySize', 'YearsCodingProf', 'HoursComputer', 'Age']]

X2 = df.iloc[:, 5:]

y = df.loc[:, 'ConvertedSalary']

X1\_scaled = (X1-X1.min())/(X1.max()-X1.min())

X = pd.concat([X1\_scaled, X2], axis=1)

import itertools

def fit\_linear\_reg(X,Y):

#Fit linear regression model and return RSS and R squared values

model\_k = LinearRegression(fit\_intercept = True)

model\_k.fit(X,Y)

RSS = mean\_squared\_error(Y,model\_k.predict(X)) \* len(Y)

R\_squared = model\_k.score(X,Y)

return RSS, R\_squared

k = 30

remaining\_features = list(X.columns.values)

features = []

RSS\_list, R\_squared\_list = [np.inf], [np.inf] #Due to 1 indexing of the loop...

features\_list = dict()

for i in range(1,k+1):

best\_RSS = np.inf

for combo in itertools.combinations(remaining\_features,1):

RSS = fit\_linear\_reg(X[list(combo) + features],y) #Store temp result

if RSS[0] < best\_RSS:

best\_RSS = RSS[0]

best\_R\_squared = RSS[1]

best\_feature = combo[0]

#Updating variables for next loop

features.append(best\_feature)

remaining\_features.remove(best\_feature)

#Saving values for plotting

RSS\_list.append(best\_RSS)

R\_squared\_list.append(best\_R\_squared)

features\_list[i] = features.copy()

## Forward stepwise selection

## AIC, BIC, Mallows'CP

df1 = pd.concat([pd.DataFrame({'features':features\_list}),pd.DataFrame({'RSS':RSS\_list, 'R\_squared': R\_squared\_list})], axis=1, join='inner')

df1['numb\_features'] = df1.index

#Initializing useful variables

m = len(y)

p = 11

hat\_sigma\_squared = (1/(m - p -1)) \* min(df1['RSS'])

#Computing

df1['C\_p'] = (1/m) \* (df1['RSS'] + 2 \* df1['numb\_features'] \* hat\_sigma\_squared )

df1['AIC'] = (1/(m\*hat\_sigma\_squared)) \* (df1['RSS'] + 2 \* df1['numb\_features'] \* hat\_sigma\_squared )

df1['BIC'] = (1/(m\*hat\_sigma\_squared)) \* (df1['RSS'] + np.log(m) \* df1['numb\_features'] \* hat\_sigma\_squared )

df1['R\_squared\_adj'] = 1 - ( (1 - df1['R\_squared'])\*(m-1)/(m-df1['numb\_features'] -1))

## Plotting the computed values as a function of number of features

variables = ['C\_p', 'AIC','BIC','R\_squared\_adj']

fig = plt.figure(figsize = (18,6))

for i,v in enumerate(variables):

ax = fig.add\_subplot(1, 4, i+1)

ax.plot(df1['numb\_features'],df1[v], color = 'lightblue')

ax.scatter(df1['numb\_features'],df1[v], color = 'darkblue')

if v == 'R\_squared\_adj':

ax.plot(df1[v].idxmax(),df1[v].max(), marker = 'x', markersize = 20)

else:

ax.plot(df1[v].idxmin(),df1[v].min(), marker = 'x', markersize = 20)

ax.set\_xlabel('Number of predictors')

ax.set\_ylabel(v)

fig.suptitle('Subset selection using C\_p, AIC, BIC, Adjusted R2', fontsize = 16)

plt.show()

## Linear regression using the selected 14 features

v14 = df1.iloc[14,0]

X\_14 = X[v14]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_14, y, test\_size=0.2, random\_state=0)

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X\_train,y\_train)

y\_pred = lm.predict(X\_test)

fig3, ax = plt.subplots()

fig3.set\_size\_inches(10, 10)

ax.set\_xlim(0, 250000)

ax.set\_ylim(0, 250000)

plt.scatter(y\_test, y\_pred)

plt.xlabel('y\_test')

plt.ylabel('y\_pred')

ab = np.linspace(0, 250000, 1000)

plt.plot(ab, ab, linestyle='dashed', color='red')

print('Selected features: ', v14)

print('Training score: ', lm.score(X\_train,y\_train))

##print('Testing score: ', lm.score(X\_test,y\_test))

print('MSE:', mean\_squared\_error(y\_test, y\_pred))

## Rige Coefficients

#import glmnet as gln

from sklearn.preprocessing import scale

from sklearn import model\_selection

from sklearn.linear\_model import Ridge, RidgeCV

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.metrics import mean\_squared\_error

alphas = 10\*\*np.linspace(10,-2,100)\*0.5

ridge = Ridge()

coefs = []

for a in alphas:

ridge.set\_params(alpha=a)

ridge.fit(scale(X\_14), y)

coefs.append(ridge.coef\_)

ax = plt.gca()

ax.plot(alphas, coefs)

ax.set\_xscale('log')

ax.set\_xlim(ax.get\_xlim()[::-1]) # reverse axis

plt.axis('tight')

plt.xlabel('lambda', fontsize=13)

plt.ylabel('weights', fontsize=13)

plt.title('Ridge coefficients as a function of the regularization');

## Ridge regression with selected 14 variables

## Rigde alpha = 100

from sklearn.linear\_model import Ridge

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_14, y, test\_size=0.2, random\_state=0)

rr10 = Ridge(alpha=100)

rr10.fit(X\_train, y\_train)

y\_pred = rr10.predict(X\_test)

fig, ax = plt.subplots()

fig.set\_size\_inches(10, 10)

ax.set\_xlim(0, 250000)

ax.set\_ylim(0, 250000)

plt.scatter(y\_test, y\_pred)

plt.xlabel('y\_test', fontsize=16)

plt.ylabel('y\_pred', fontsize=16)

ab = np.linspace(0, 250000, 1000)

plt.plot(ab, ab, linestyle='dashed', color='red')

print('Training score: ', rr10.score(X\_train,y\_train))

print('Testing score: ', rr10.score(X\_test,y\_test))

print('MSE:', mean\_squared\_error(y\_test, y\_pred))

fig, ax = plt.subplots()

fig.set\_size\_inches(10, 10)

plt.scatter(y\_pred, (y\_pred-y\_test))

plt.axhline(y=0, color='r', linestyle='--')

fig.set\_size\_inches(10, 10)

plt.xlabel('y\_pred', fontsize = 16)

plt.ylabel('residuals', fontsize = 16)

## Polynomial regression with variables

from sklearn.preprocessing import PolynomialFeatures

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_14, y, test\_size=0.2, random\_state=0)

poly = PolynomialFeatures(degree = 2)

X\_train\_ = poly.fit\_transform(X\_train)

X\_test\_ = poly.fit\_transform(X\_test)

#poly.fit(X\_train\_, y)

# Instantiate

lg = LinearRegression()

# Fit

lg.fit(X\_train\_, y\_train)

# Obtain coefficients

#lg.coef\_

y\_pred = lg.predict(X\_test\_)

fig4, ax = plt.subplots()

fig4.set\_size\_inches(10, 10)

#ax.set\_xlim(0, 250000)

#ax.set\_ylim(0, 250000)

plt.scatter(y\_test, y\_pred)

plt.xlabel('y\_test')

plt.ylabel('y\_pred')

ab = np.linspace(0, 250000, 1000)

plt.plot(ab, ab, linestyle='dashed', color='red')

print('Training score: ', lg.score(X\_train\_,y\_train))

print('Testing score: ', lg.score(X\_test\_,y\_test))

print('MSE:', mean\_squared\_error(y\_test, y\_pred))

mse = sum(np.square(y\_pred-y\_test))/len(y\_pred)

np.square(y\_pred-y\_test)

fig, ax = plt.subplots()

fig.set\_size\_inches(10, 10)

plt.scatter(y\_pred, (y\_pred-y\_test))

plt.axhline(y=0, color='r', linestyle='--')

fig.set\_size\_inches(10, 10)

plt.xlabel('y\_pred', fontsize = 16)

plt.ylabel('residuals', fontsize = 16)

## Appendix D – Code of classification modelling for database1

bins = [24999, 50000, 75000, 100000, 150000, 200000, 250000]

labels = [1,2,3,4,5,6]#['$25,000-$50,000', '$50,000-$75,000', '$75,000-$100,000', '$100,000-$150,000', '$150,000-$200,000', '$200,000-$250,000']

ranges = pd.cut(df.ConvertedSalary, bins=bins, labels=labels)

df['SalaryRange'] = pd.DataFrame(ranges)

df = df.drop(['JobSatisfaction', 'ConvertedSalary'],axis=1)

df = df.dropna()

df = pd.get\_dummies(df, columns=['FormalEducation', 'UndergradMajor', 'Gender', 'RaceEthnicity'], drop\_first=True)

X = df.loc[:, df.columns != 'SalaryRange']

y = y = df.loc[:, 'SalaryRange']

X\_train\_n, X\_test\_n, y\_train\_n, y\_test\_n = train\_test\_split(X\_new, y, test\_size=0.2, random\_state = 0)

acc = []

depth = []

for i in range(1, 13):

dtree\_model = DecisionTreeClassifier(max\_depth = i).fit(X\_train\_n, y\_train\_n)

dtree\_predictions = dtree\_model.predict(X\_test\_n)

accuracy\_tree = accuracy\_score(y\_test\_n, dtree\_predictions)

print('The accuracy of the decision tree with max depth of %i is %f', i, accuracy\_tree)

depth.append(i)

acc.append(accuracy\_tree)

df\_depth = pd.DataFrame({'max\_depth': depth, 'accuracy': acc})

plt.plot(df\_depth.max\_depth, df\_depth.accuracy)

plt.xlabel('Max Depth of Decision Tree')

plt.ylabel('Accuracy')

plt.suptitle('Max depth vs accuracy', fontsize = 16)

dtree\_model = DecisionTreeClassifier(max\_depth = 10).fit(X\_train\_n, y\_train\_n)

dtree\_predictions = dtree\_model.predict(X\_test\_n)

labels1 = ['$25,000-$50,000', '$50,000-$75,000', '$75,000-$100,000', '$100,000-$150,000', '$150,000-$200,000', '$200,000-$250,000']

# creating a confusion matrix

cm\_tree = confusion\_matrix(y\_test\_n, dtree\_predictions)

accuracy\_tree = accuracy\_score(y\_test\_n, dtree\_predictions)

print('The accuracy of the model is ', accuracy\_tree)

# Plot non-normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_tree,classes=labels1,

title='Confusion matrix, without normalization')

# Plot normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_tree, classes=labels1, normalize=True,

title='Normalized confusion matrix')

plt.grid('off')

#plt.savefig('save\_file.png')

from sklearn.neighbors import KNeighborsClassifier

kk = []

acc = []

for k in range(1,50):

knn = KNeighborsClassifier(n\_neighbors = k).fit(X\_train\_n, y\_train\_n)

# accuracy on X\_test

accuracy = knn.score(X\_test\_n, y\_test\_n)

print(k, accuracy)

kk.append(k)

acc.append(accuracy)

df\_knn = pd.DataFrame({'K': kk, 'accuracy': acc})

plt.plot(df\_knn.K, df\_knn.accuracy)

plt.xlabel('K nearest neighbors')

plt.ylabel('Accuracy')

plt.suptitle('KNN vs accuracy', fontsize = 16)

knn = KNeighborsClassifier(n\_neighbors = 29).fit(X\_train\_n, y\_train\_n)

knn\_pred = knn.predict(X\_test\_n)

# creating a confusion matrix

cm\_knn = confusion\_matrix(y\_test\_n, knn\_pred)

accuracy\_knn = accuracy\_score(y\_test\_n, knn\_pred)

print('The accuracy of the model is ', accuracy\_knn)

# Plot non-normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_knn,classes=labels1,

title='Confusion matrix, without normalization')

plt.grid('off')

# Plot normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_knn, classes=labels1, normalize=True,

title='Normalized confusion matrix')

## Appendix E – Code of regression and classification modelling for database2

## Import packages

%matplotlib inline

%xmode Verbose

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import matplotlib.ticker as ticker

import statsmodels.api as sm

from sklearn.preprocessing import normalize

pd.options.display.max\_rows = 100

df = pd.read\_csv(r'C:\Users\junta\Google Drive\BANA7038\BANA7038 Final Project\reduced files 022519\file with only calculated fields.csv')

df[['NUMFRAMEWORKS', 'NUMPLATFORM', 'NUMDATABASE', 'numberlang']] = df[['NUMFRAMEWORKS', 'NUMPLATFORM', 'NUMDATABASE', 'numberlang']].fillna(value=0)

# dataframe without NAs

df1 = df.drop('ConvertedSalary', axis=1).dropna()

df = df.dropna()

y\_old = df['ConvertedSalary']

df1 = pd.get\_dummies(df1, columns=['UndergradMajor', 'Gender', 'RaceEthnicity'], drop\_first=True)

corr = df1.corr()

fig, ax = plt.subplots()

fig.set\_size\_inches(14, 10)

sns.heatmap(abs(corr),

xticklabels=corr.columns,

yticklabels=corr.columns,)

df1 = df1.dropna()

X = df1.loc[:, df1.columns != 'SalaryRange']

y = df1.loc[:, 'SalaryRange']

import itertools

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

def fit\_linear\_reg(X,Y):

#Fit linear regression model and return RSS and R squared values

model\_k = LinearRegression(fit\_intercept = True)

model\_k.fit(X,Y)

RSS = mean\_squared\_error(Y,model\_k.predict(X)) \* len(Y)

R\_squared = model\_k.score(X,Y)

return RSS, R\_squared

k = 15

remaining\_features = list(X.columns.values)

features = []

RSS\_list, R\_squared\_list = [np.inf], [np.inf] #Due to 1 indexing of the loop...

features\_list = dict()

for i in range(1,k+1):

best\_RSS = np.inf

for combo in itertools.combinations(remaining\_features,1):

RSS = fit\_linear\_reg(X[list(combo) + features],y) #Store temp result

if RSS[0] < best\_RSS:

best\_RSS = RSS[0]

best\_R\_squared = RSS[1]

best\_feature = combo[0]

#Updating variables for next loop

features.append(best\_feature)

remaining\_features.remove(best\_feature)

#Saving values for plotting

RSS\_list.append(best\_RSS)

R\_squared\_list.append(best\_R\_squared)

features\_list[i] = features.copy()

## Forward stepwise selection

## AIC, BIC, Mallows'CP

df\_sel = pd.concat([pd.DataFrame({'features':features\_list}),pd.DataFrame({'RSS':RSS\_list, 'R\_squared': R\_squared\_list})], axis=1, join='inner')

df\_sel['numb\_features'] = df\_sel.index

#Initializing useful variables

m = len(y)

p = 11

hat\_sigma\_squared = (1/(m - p -1)) \* min(df\_sel['RSS'])

#Computing

df\_sel['C\_p'] = (1/m) \* (df\_sel['RSS'] + 2 \* df\_sel['numb\_features'] \* hat\_sigma\_squared )

df\_sel['AIC'] = (1/(m\*hat\_sigma\_squared)) \* (df\_sel['RSS'] + 2 \* df\_sel['numb\_features'] \* hat\_sigma\_squared )

df\_sel['BIC'] = (1/(m\*hat\_sigma\_squared)) \* (df\_sel['RSS'] + np.log(m) \* df\_sel['numb\_features'] \* hat\_sigma\_squared )

df\_sel['R\_squared\_adj'] = 1 - ( (1 - df\_sel['R\_squared'])\*(m-1)/(m-df\_sel['numb\_features'] -1))

## Plotting the computed values as a function of number of features

variables = ['C\_p', 'AIC','BIC','R\_squared\_adj']

fig = plt.figure(figsize = (18,6))

for i,v in enumerate(variables):

ax = fig.add\_subplot(1, 4, i+1)

ax.plot(df\_sel['numb\_features'],df\_sel[v], color = 'lightblue')

ax.scatter(df\_sel['numb\_features'],df\_sel[v], color = 'darkblue')

if v == 'R\_squared\_adj':

ax.plot(df\_sel[v].idxmax(),df\_sel[v].max(), marker = 'x', markersize = 20)

else:

ax.plot(df\_sel[v].idxmin(),df\_sel[v].min(), marker = 'x', markersize = 20)

ax.set\_xlabel('Number of predictors')

ax.set\_ylabel(v)

fig.suptitle('Subset selection using C\_p, AIC, BIC, Adjusted R2', fontsize = 16)

plt.show()

selected\_fea = df\_sel.iloc[4,0]

X\_new = X[selected\_fea]

## Linear regression using the selected 5 features

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, y\_old, test\_size=0.2, random\_state=0)

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X\_train,y\_train)

y\_pred = lm.predict(X\_test)

fig3, ax = plt.subplots()

fig3.set\_size\_inches(10, 10)

ax.set\_xlim(0, 250000)

ax.set\_ylim(0, 250000)

plt.scatter(y\_test, y\_pred)

plt.xlabel('y\_test', fontsize=16)

plt.ylabel('y\_pred', fontsize=16)

ab = np.linspace(0, 250000, 1000)

plt.plot(ab, ab, linestyle='dashed', color='red')

#print('Selected features: ', v14)

print('Training score: ', lm.score(X\_train,y\_train))

##print('Testing score: ', lm.score(X\_test,y\_test))

print('MSE:', mean\_squared\_error(y\_test, y\_pred))

fig3, ax = plt.subplots()

fig3.set\_size\_inches(10, 10)

plt.scatter(y\_pred, (y\_pred-y\_test))

plt.axhline(y=0, color='r', linestyle='--')

fig.set\_size\_inches(10, 10)

plt.xlabel('y\_pred', fontsize = 16)

plt.ylabel('residuals', fontsize = 16)

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.tight\_layout()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, y, test\_size=0.2, random\_state = 0)

acc = []

depth = []

for i in range(1, 6):

dtree\_model = DecisionTreeClassifier(max\_depth = i).fit(X\_train, y\_train)

dtree\_predictions = dtree\_model.predict(X\_test)

accuracy\_tree = accuracy\_score(y\_test, dtree\_predictions)

print('The accuracy of the decision tree with max depth of %i is %f', i, accuracy\_tree)

depth.append(i)

acc.append(accuracy\_tree)

#pd.concat([pd.DataFrame({'index':index}),pd.DataFrame({'accuracy':accuracy\_tree})]

#print(df\_depth)

df\_depth = pd.DataFrame({'max\_depth': depth, 'accuracy': acc})

plt.plot(df\_depth.max\_depth, df\_depth.accuracy)

plt.xlabel('Max Depth of Decision Tree')

plt.ylabel('Accuracy')

plt.suptitle('Max depth vs accuracy', fontsize = 16)

labels1 = ['$25,000-$50,000', '$50,000-$75,000', '$75,000-$100,000', '$100,000-$150,000', '$150,000-$200,000', '$200,000-$250,000']

# creating a confusion matrix

cm\_tree = confusion\_matrix(y\_test, dtree\_predictions)

accuracy\_tree = accuracy\_score(y\_test, dtree\_predictions)

print('The accuracy of the model is ', accuracy\_tree)

# Plot non-normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_tree,classes=labels1,

title='Confusion matrix, without normalization')

plt.grid('off')

# Plot normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_tree, classes=labels1, normalize=True,

title='Normalized confusion matrix')

#plt.savefig('save\_file.png')

from sklearn.neighbors import KNeighborsClassifier

kk = []

acc = []

for k in range(1,50):

knn = KNeighborsClassifier(n\_neighbors = k).fit(X\_train, y\_train)

# accuracy on X\_test

accuracy = knn.score(X\_test, y\_test)

print(k, accuracy)

kk.append(k)

acc.append(accuracy)

#pd.concat([pd.DataFrame({'index':index}),pd.DataFrame({'accuracy':accuracy\_tree})]

#print(df\_depth)

df\_knn = pd.DataFrame({'K': kk, 'accuracy': acc})

plt.plot(df\_knn.K, df\_knn.accuracy)

plt.xlabel('K nearest neighbors')

plt.ylabel('Accuracy')

plt.suptitle('KNN vs accuracy', fontsize = 16)

knn = KNeighborsClassifier(n\_neighbors = 32).fit(X\_train, y\_train)

knn\_pred = knn.predict(X\_test)

# creating a confusion matrix

cm\_knn = confusion\_matrix(y\_test, knn\_pred)

accuracy\_knn = accuracy\_score(y\_test, knn\_pred)

print('The accuracy of the model is ', accuracy\_knn)

# Plot non-normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_knn,classes=labels1,

title='Confusion matrix, without normalization')

plt.grid('off')

# Plot normalized confusion matrix

plt.figure(figsize=(10,10))

plot\_confusion\_matrix(cm\_knn, classes=labels1, normalize=True,

title='Normalized confusion matrix')