BANA 7038
February 28, 2019

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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.

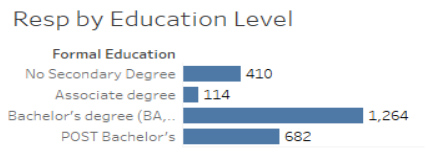
CHAPTER 1: Data exploration and data cleansing
In reviewing the data, we again limited the developers to only those in the Data Science/Data Analysis roles. This left us with about 2500 respondents. We also initially chose to break out multiselect fields (one field - many values) into separate fields for languages, frameworks, platforms and databases. Other items included in our data cleansing process:

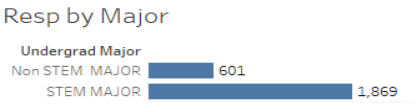
- Salaries were incorrectly entered, artificially inflating the values. (entering an annual figure but indicating pay is weekly). We reduced our sample to salaries between 25,000USD and 250,000 USD. We also eliminated individuals with the title of CEO/CIO/CFO.
- Gender was an issue. We consolidated what 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.

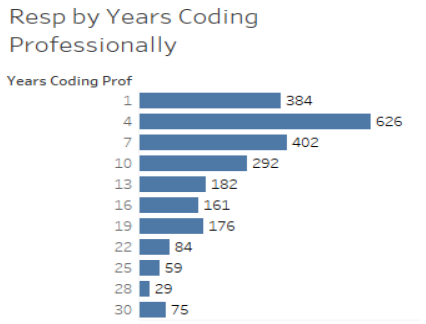
Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

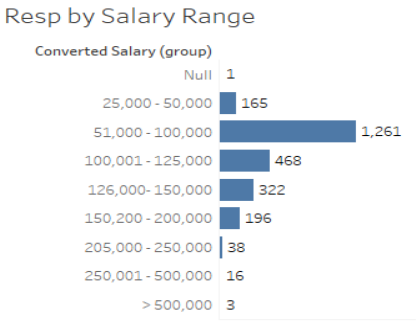
- The features we kept are FormalEducation, UndergradMajor, CompanySize, YearsCodingProf, ConvertedSalary, LanguageWorkedWith, DatabaseWorkedWith, PlatformWorkedWith, FrameworkWorkedWith, HoursComputer, Gender, RaceEthnicity and Age from the original database.

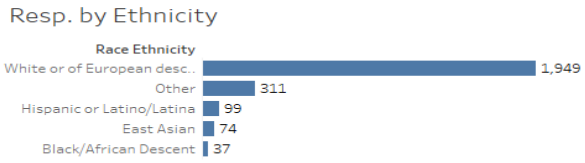
Summary of data file values

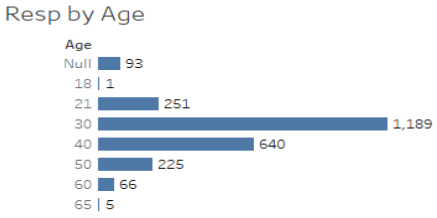


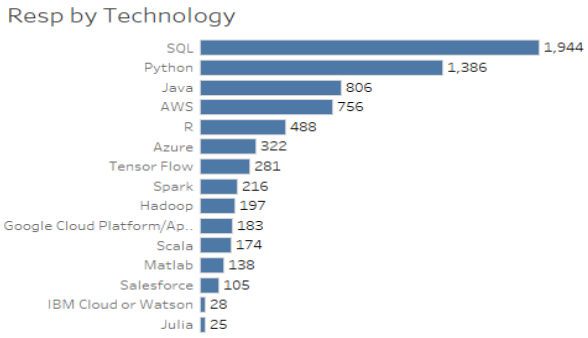












BANA 7038
February 28, 2019

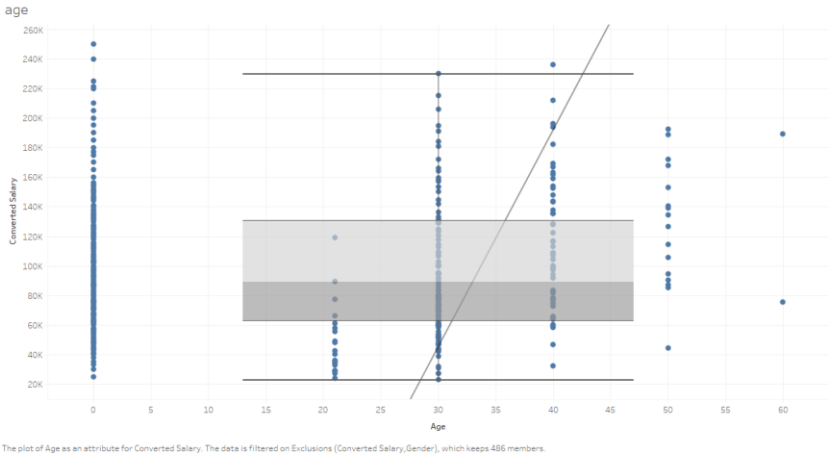
Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

Pairs from dataset

BANA 7038
February 28, 2019

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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)	**SSE (sum squared error):**	10466.6
Number of modeled observations:	204	**MSE (mean squared error):**	51.8147
Number of filtered observations:	145	**R-Squared:**	0.168804
Model degrees of freedom:	2	**Standard error:**	7.19824
Residual degrees of freedom (DF):	202	**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

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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

BANA 7038
February 28, 2019

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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)
```

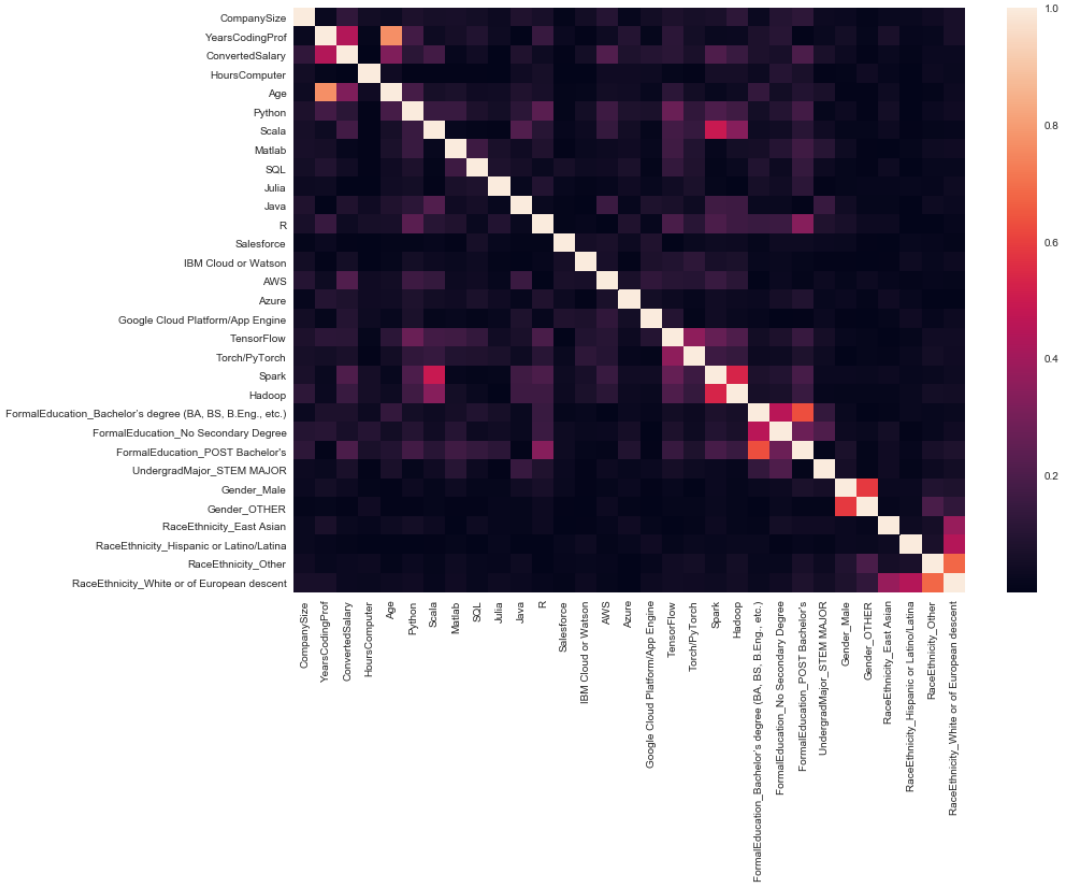


Figure 2.1 Heatmap of pairwise correlation of columns

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 C_p, AIC, BIC and adjusted R^2 are introduced to select the number of features to be used.

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

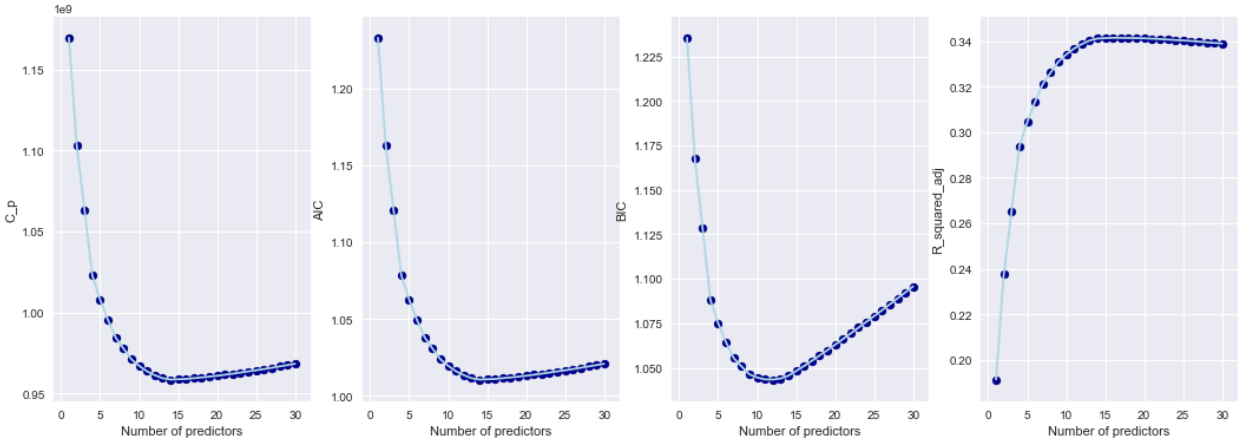


Figure 2.2 Feature selection using C_p, AIC, BIC and adjusted R^2

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

2. 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)
```

3. Regression modelling

3.1.1 Linear regression

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

Python code:

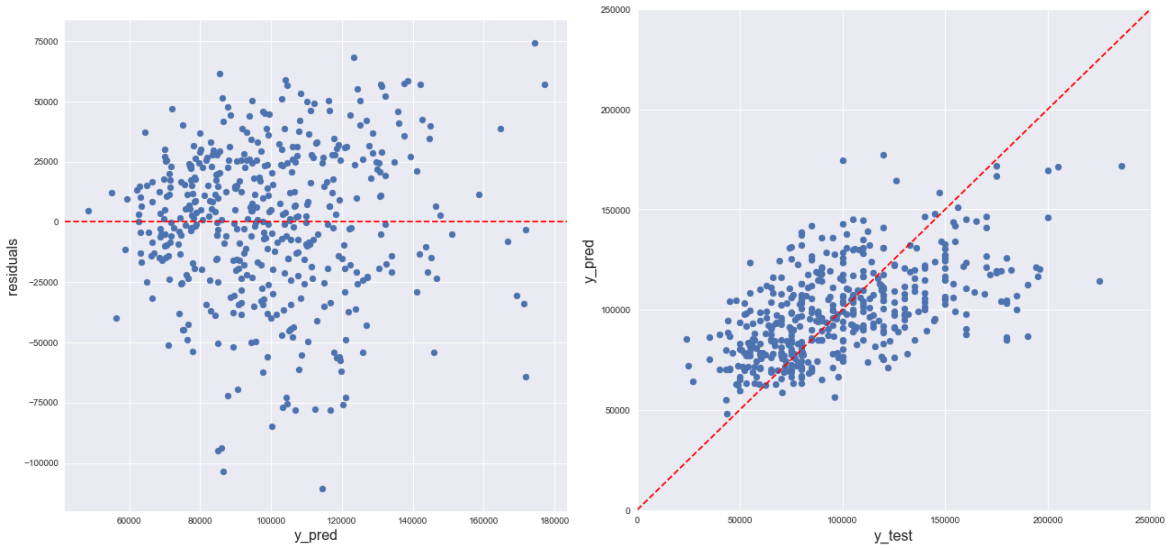
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
y_pred = lm.predict(X_test)
```

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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) (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.

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

3.2 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:

```
Out[43]: YearsCodingProf                                          2.309238
         Spark                                                    1.563102
         AWS                                                      1.600060
         FormalEducation_POST Bachelor's                          3.137321
         CompanySize                                              1.418939
         Python                                                   2.661993
         FormalEducation_Bachelor's degree (BA, BS, B.Eng., etc.) 4.876778
         Scala                                                    1.438269
         Google Cloud Platform/App Engine                         1.116321
         SQL                                                      4.445028
         Azure                                                    1.195321
         FormalEducation_No Secondary Degree                      2.074255
         TensorFlow                                               1.328521
         RaceEthnicity_East Asian                                 1.045913
         UndergradMajor_STEM MAJOR                                3.849058
         dtype: float64
```

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.

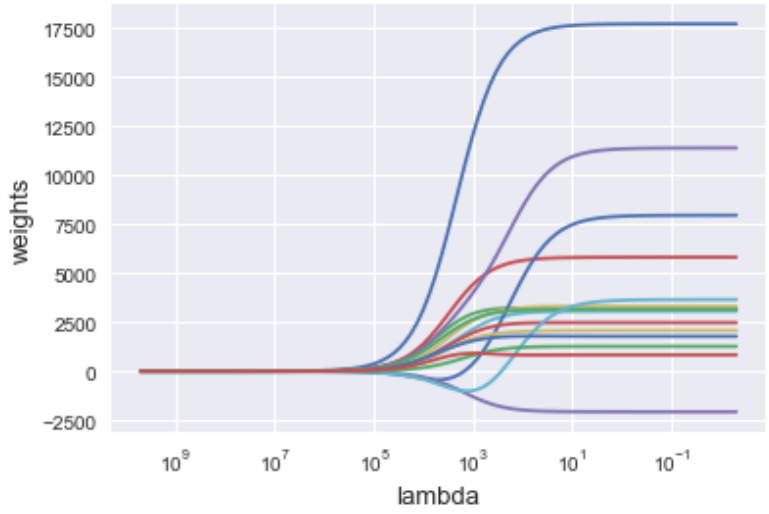
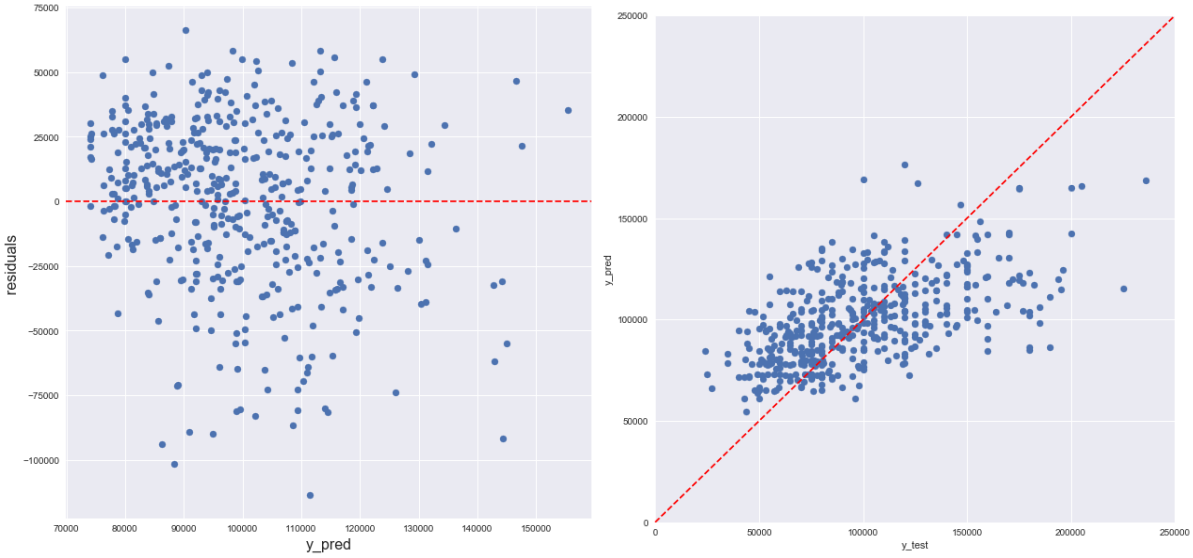


Figure 2.4 Ridge coefficients as a function of the regularization

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

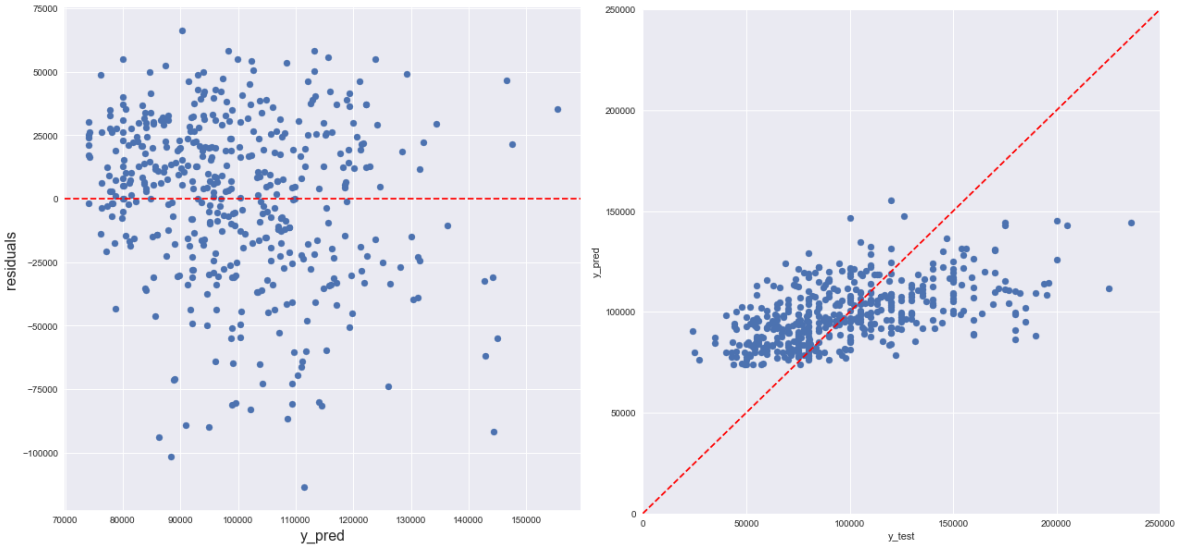
- **λ = 10 MSE = 975182423**



(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) (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

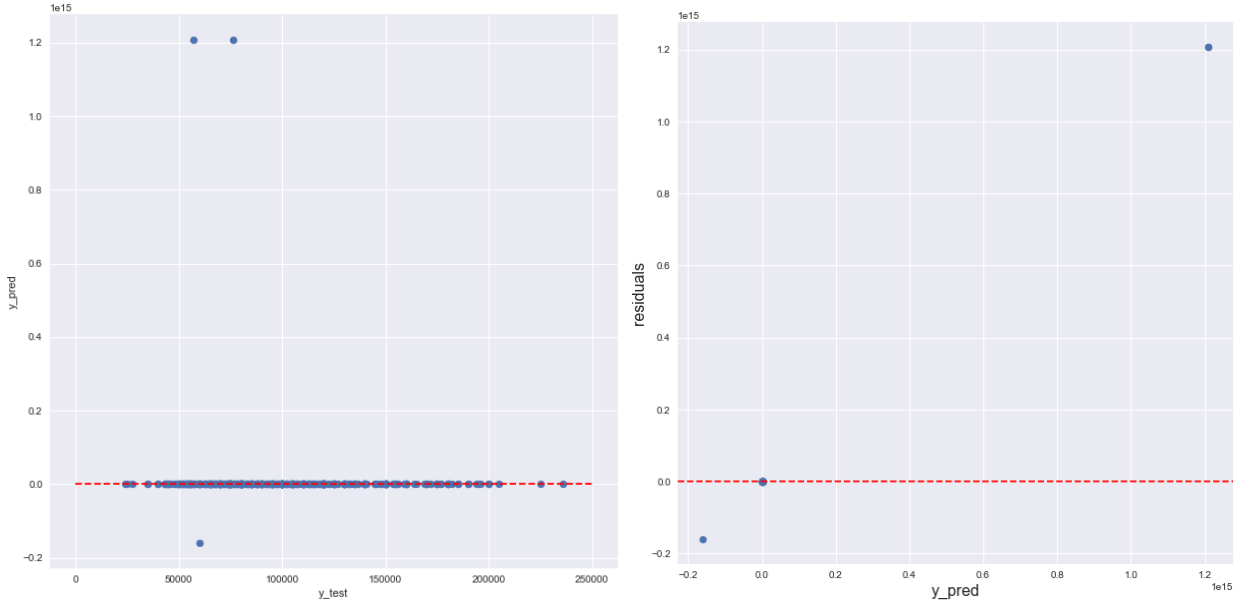
We can see there is an obvious trend that the fitted values are focusing towards $100,000 as the λ increases. The MSE also increased, so ridge regression did not help with the improvement of the prediction.

3.3 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

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

regression model. Obviously, polynomial regression doesn't fit this dataset and generates an MSE of $7x10^{28}$ which is ridiculously large.



(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

4 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:

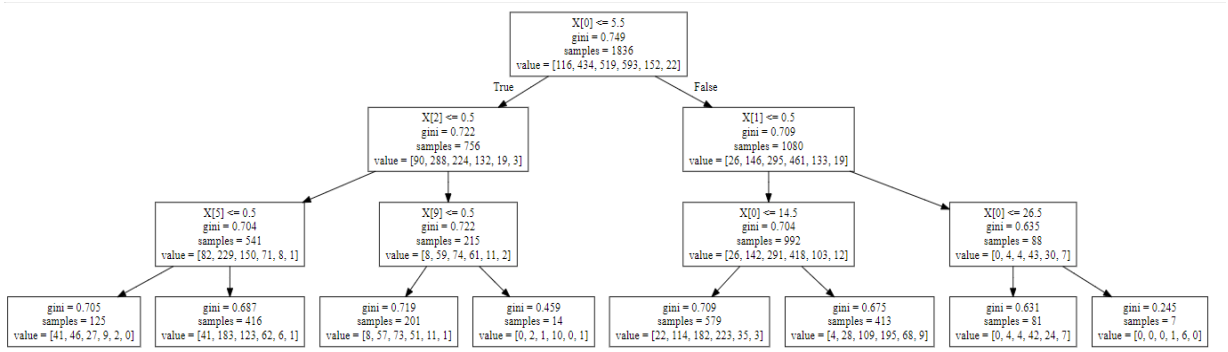
- $25,000 - $50,000
- $50,000 - $75,000
- $75,000 - $100,000
- $100,000 - $150,000
- $150,000 - $200,000
- $200,000 - $250,000

4.1 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.

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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.

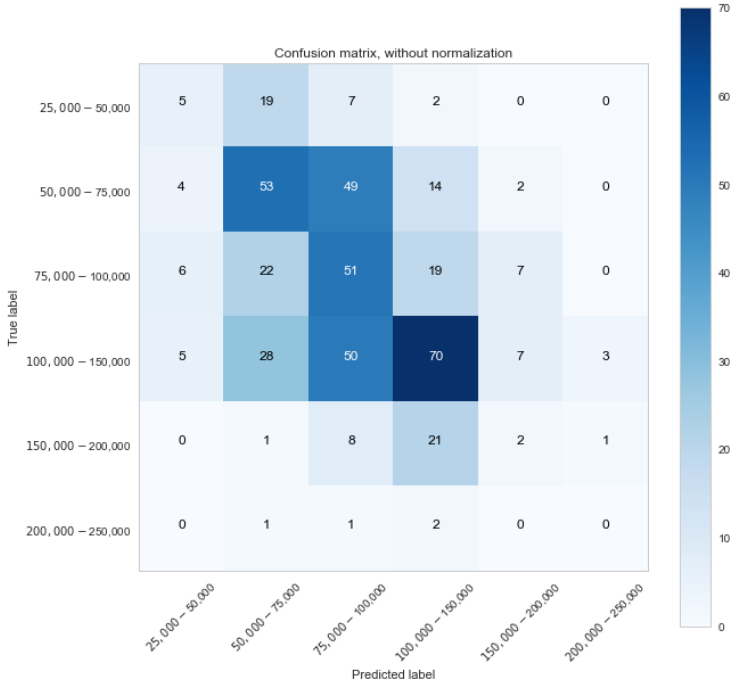


Figure 2.8 Confusion matrix of decision tree classification with max depth of 3

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

4.2 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%.

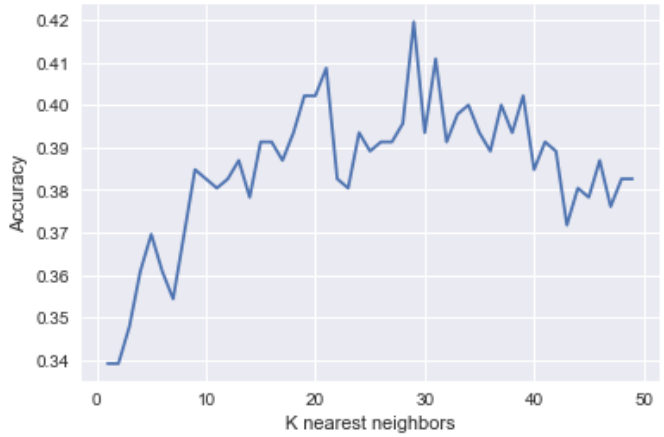


Figure 2.9 Accuracy of the KNN classification against the K value

Below shows the confusion matrix for the test data.

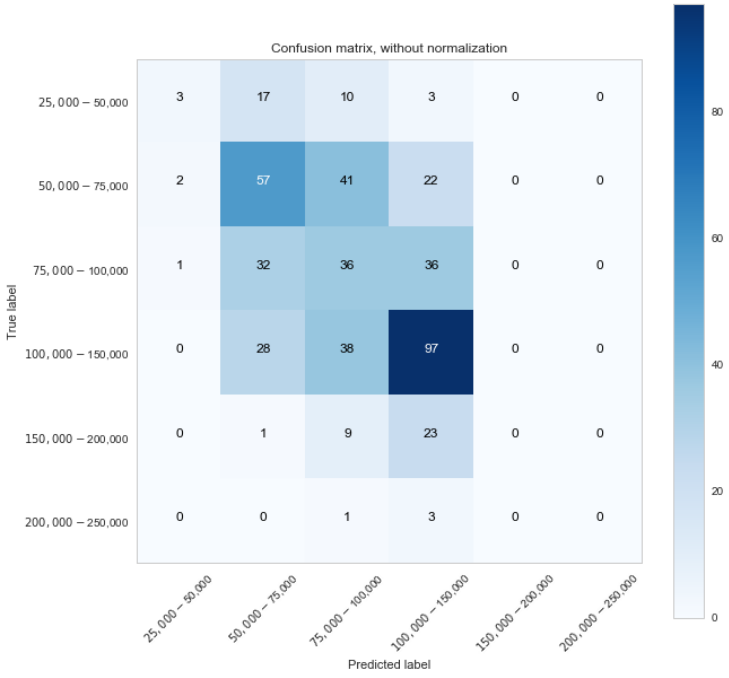


Figure 2.10 Confusion matrix of KNN classification with K of 29

Chapter 3: Observations

Best model selection:

Model name	Simple linear regression	Ridge regression λ = 10	Ridge regression λ = 100	Polynomial regression
MSE	9.55×10^8	9.75×10^8	1.03×10^9	7×10^{28}

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
$\widehat{\beta_0}$	52566.11
$\widehat{\beta_1}$	66776.97
$\widehat{\beta_2}$	12641.25
$\widehat{\beta_3}$	13342.42
$\widehat{\beta_4}$	26545.67
$\widehat{\beta_5}$	8161.53
$\widehat{\beta_6}$	5663.25
$\widehat{\beta_7}$	16471.66
$\widehat{\beta_8}$	12557.52
$\widehat{\beta_9}$	9866.98
$\widehat{\beta_{10}}$	-5521.06
$\widehat{\beta_{11}}$	5982.20
$\widehat{\beta_{12}}$	9428.34
$\widehat{\beta_{13}}$	6109.17
$\widehat{\beta_{14}}$	8174.76
$\widehat{\beta_{15}}$	1342.67

BANA 7038
February 28, 2019

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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	Formal Education_No Secondary Degree	TensorFlow	RaceEthnicity_East Asian	UndergradMajor_STEM MAJOR
0	0.62068965	0	0	0	0.00050012 5	0	0	0	0	1	0	1	0	0	0
1	0.20689655 2	0	0	0	0.00275068 8	1	1	0	0	1	0	0	0	0	1
2	0.20689655 2	1	0	1	1	1	0	1	0	0	0	0	0	0	1
3	0.10344827 6	0	0	0	0.14978744 7	1	1	0	0	1	0	0	0	0	0
4	0	0	0	1	0.37484371 1	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.

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

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.

BANA 7038
February 28, 2019
Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

Appendix A - Results and visualization of dataset2

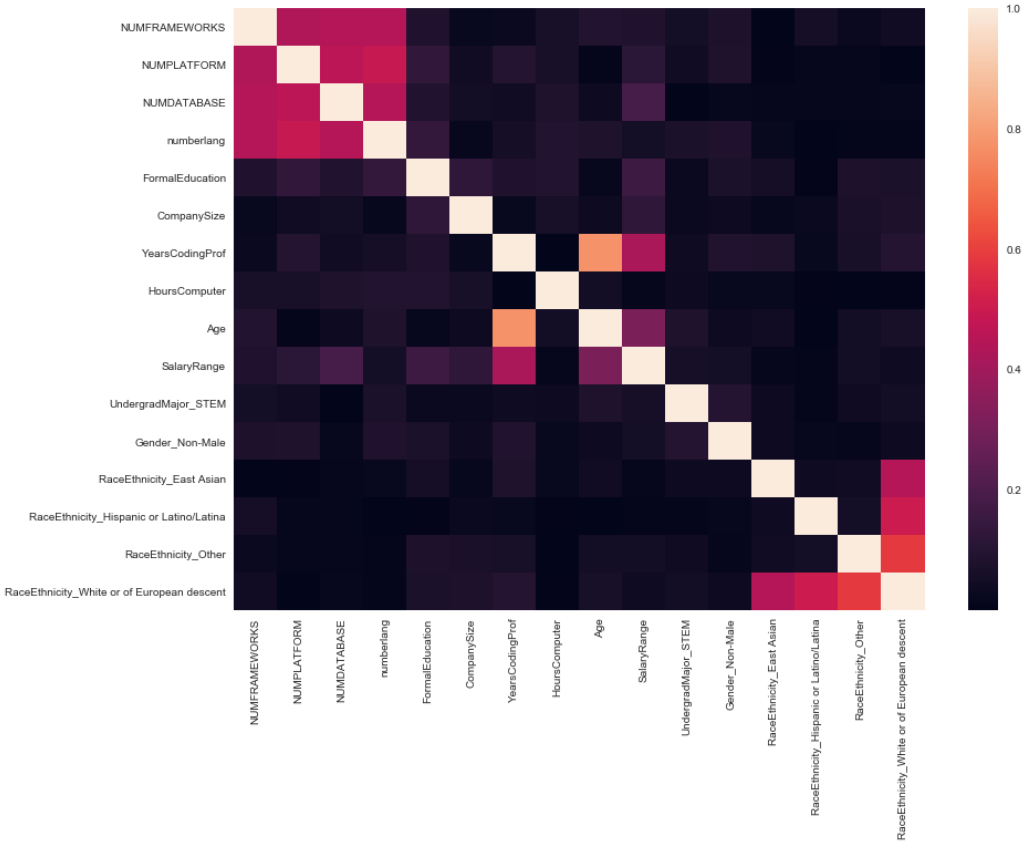


Figure A.1 Heatmap of pairwise correlation of columns

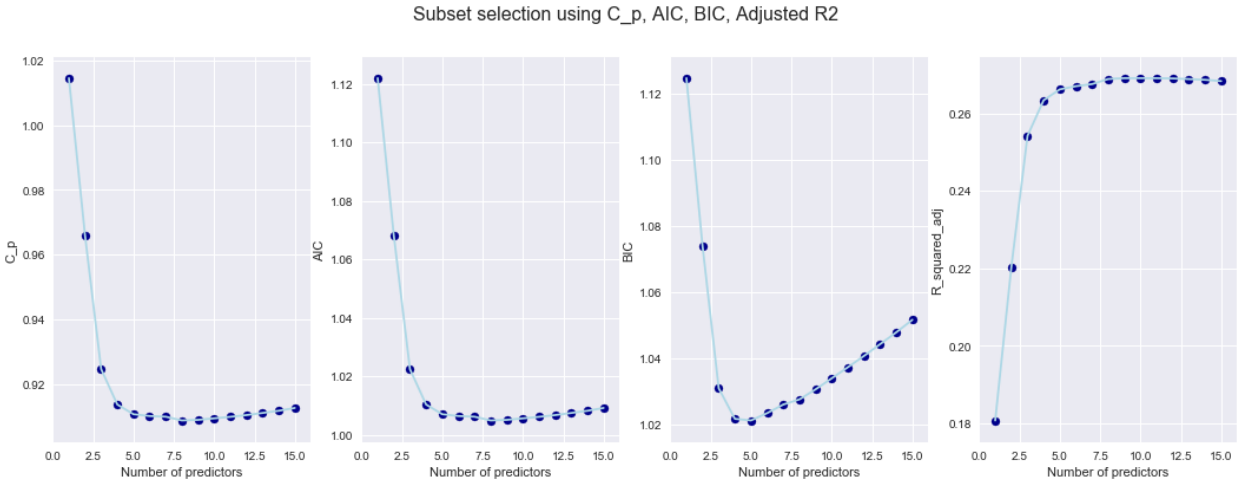


Figure B.2 Feature selection using C_p, AIC, BIC and adjusted R^2

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

- YearsCodingProf
- FormalEducation
- NUMDATABASE
- CompanySize
- RaceEthnicity_White or of European descent

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

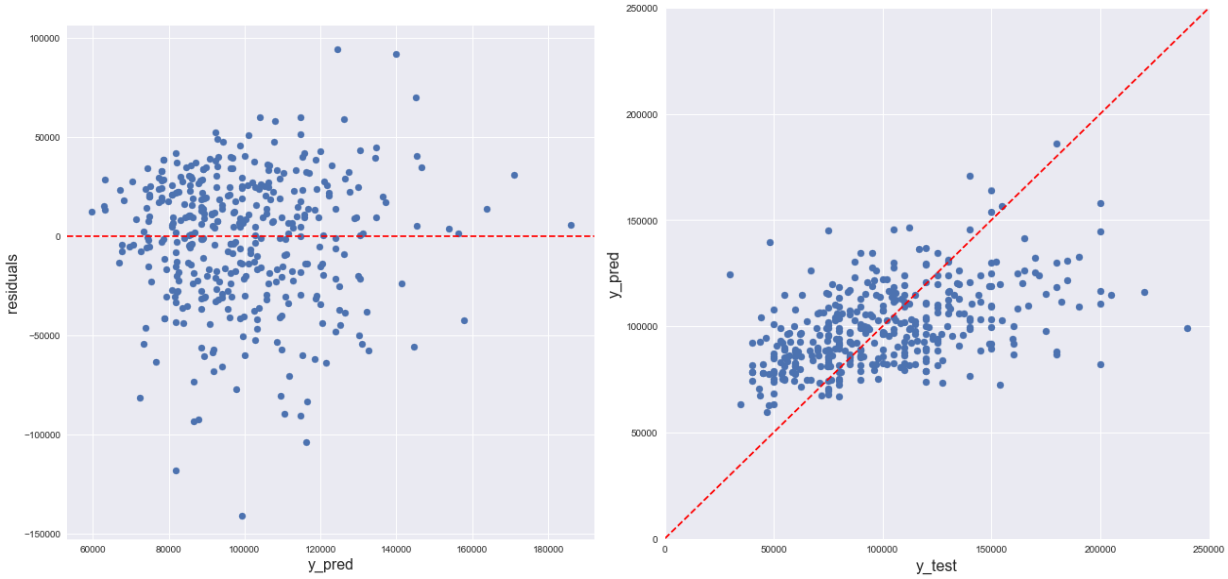


Figure A.3 (a) Scatter plot of residual against fitted values from linear regression (b) Scatter plot of linear fitted values against true values

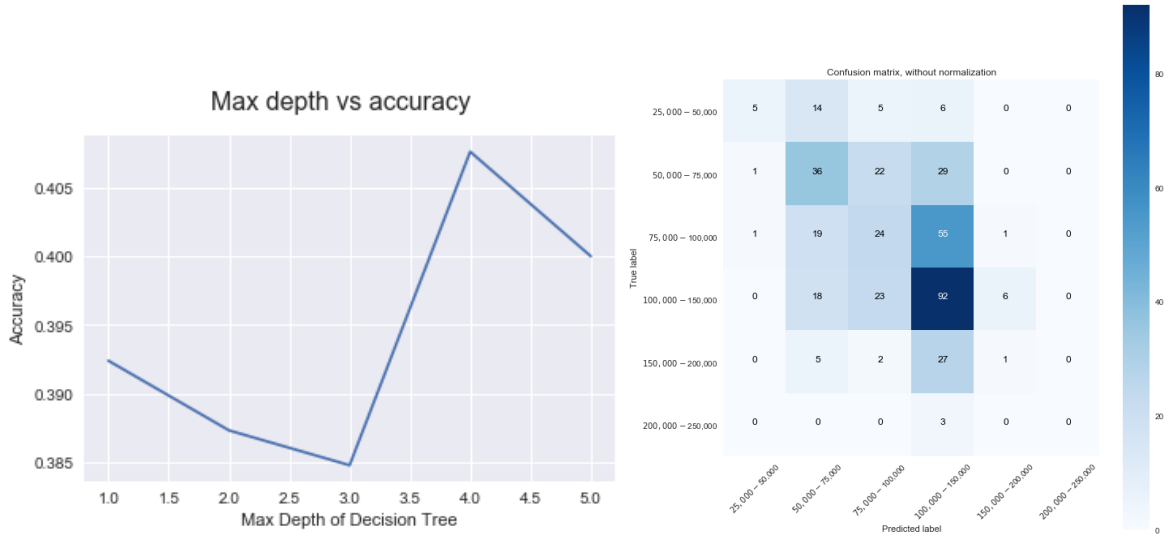


Figure A.4 Line plot of accuracy of the decision tree classification against max depth of the decision tree and confusion matrix of decision tree classification with max depth of 4

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

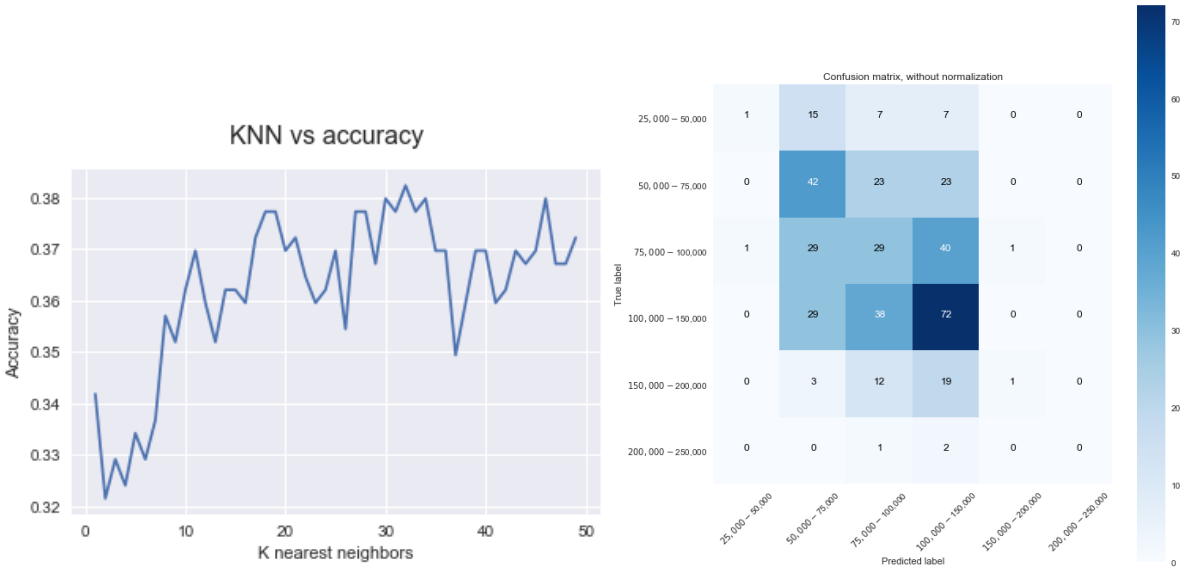


Figure A.5 Accuracy of the KNN classification against the K value and Confusion matrix of KNN classification with K of 32

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)
```

Dong, Juntao (M10453134)
Frankenhoff, Brenda (M01775924)

```
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')
```