CS 677 Final Project

Prediction of Data Science Job Salaries

Olivia Lee (U76797374)
Boston University Computer Science Department
Fall 2022

LIBRARIES

In [42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import chi2
from matplotlib.pyplot import axhline
from sklearn.preprocessing import PolynomialFeatures
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
```

THE DATA SET

The data set I am using contains information on data scientists and similar professions, such as Machine Learning Engineer, Data Analyst, Data Engineer, etc.

Attributes:
work_year: The year the salary was paid.
experience_level: The experience level in the job during the year with the following possible values: EN Entry-level / Junior MI Mid-level / Intermediate SE Senior-level / Expert EX Executive-level / Director
employment_type: The type of employement for the role: PT Part-time FT Full-time CT Contract FL Freelance
job_title: The role worked in during the year.
salary: The total gross salary amount paid.
salary_currency: The currency of the salary paid as an ISO 4217 currency code.
salary_in_usd: The salary in USD (FX rate divided by avg. USD rate for the respective year via fxdata.foorilla.com).
employee_residence: Employee's primary country of residence in during the work year as an ISO 3166 country code.
remote_ratio: The overall amount of work done remotely, possible values are as follows: 0 No remote work (less than 20%) 50 Partially remote 100 Fully remote (more than 80%)
company_location: The country of the employer's main office or contracting branch as an ISO 3166 country code.
company_size: The average number of people that worked for the company during the year: S less than 50 employees (small) M 50 to 250 employees (medium) L more than 250 employees (large)

In [2]:
```
df = pd.read_csv("ds_salaries.csv")
df
```

Out[2]:

	Unnamed: 0	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	0	2020	MI	FT	Data Scientist	70000	EUR	79833	DE	0	DE	L
1	1	2020	SE	FT	Machine Learning Scientist	260000	USD	260000	JP	0	JP	S
2	2	2020	SE	FT	Big Data Engineer	85000	GBP	109024	GB	50	GB	M
3	3	2020	MI	FT	Product Data Analyst	20000	USD	20000	HN	0	HN	S
4	4	2020	SE	FT	Machine Learning Engineer	150000	USD	150000	US	50	US	L
...	...	...	...	...	...	...	...	...	...	...	...	...
602	602	2022	SE	FT	Data Engineer	154000	USD	154000	US	100	US	M
603	603	2022	SE	FT	Data Engineer	126000	USD	126000	US	100	US	M
604	604	2022	SE	FT	Data Analyst	129000	USD	129000	US	0	US	M
605	605	2022	SE	FT	Data Analyst	150000	USD	150000	US	100	US	M
606	606	2022	MI	FT	AI Scientist	200000	USD	200000	IN	100	US	L

607 rows × 12 columns

DATA CLEANING

1. Filtering and selecting columns

In [3]:
```
df = pd.read_csv("ds_salaries.csv")

#dropping index column that was included in csv file
df = df.drop(columns = [df.columns[0]])

#dropping all rows that contain NAs. In this dataset, there were none.
df = df.dropna()
df
```

Out[3]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2020	MI	FT	Data Scientist	70000	EUR	79833	DE	0	DE	L
1	2020	SE	FT	Machine Learning Scientist	260000	USD	260000	JP	0	JP	S
2	2020	SE	FT	Big Data Engineer	85000	GBP	109024	GB	50	GB	M
3	2020	MI	FT	Product Data Analyst	20000	USD	20000	HN	0	HN	S
4	2020	SE	FT	Machine Learning Engineer	150000	USD	150000	US	50	US	L
...	...	...	...	...	...	...	...	...	...	...	...
602	2022	SE	FT	Data Engineer	154000	USD	154000	US	100	US	M
603	2022	SE	FT	Data Engineer	126000	USD	126000	US	100	US	M
604	2022	SE	FT	Data Analyst	129000	USD	129000	US	0	US	M
605	2022	SE	FT	Data Analyst	150000	USD	150000	US	100	US	M
606	2022	MI	FT	AI Scientist	200000	USD	200000	IN	100	US	L

607 rows × 11 columns

2. Identifying and handling outliers

a. salary_in_usd

Although that there are many minimum outliers with salary prices that seem unrealistic in the US, I have decided to keep them in the dataset as they are from other countries that could have different wages.

In [4]:
```
#sorting salaries in ascending order to check for minimum and maximum outliers
df.sort_values(by = ['salary_in_usd'])
```

Out[4]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
176	2021	MI	FT	Data Scientist	58000	MXN	2859	MX	0	MX	S
185	2021	MI	FT	Data Engineer	4000	USD	4000	IR	100	IR	M
238	2021	EN	FT	Data Scientist	4000	USD	4000	VN	0	VN	M
77	2021	MI	PT	3D Computer Vision Researcher	400000	INR	5409	IN	50	IN	M
179	2021	MI	FT	Data Scientist	420000	INR	5679	IN	100	US	S
...	...	...	...	...	...	...	...	...	...	...	...
225	2021	EX	CT	Principal Data Scientist	416000	USD	416000	US	100	US	S
157	2021	MI	FT	Applied Machine Learning Scientist	423000	USD	423000	US	50	US	L
97	2021	MI	FT	Financial Data Analyst	450000	USD	450000	US	100	US	L
33	2020	MI	FT	Research Scientist	450000	USD	450000	US	0	US	M
252	2021	EX	FT	Principal Data Engineer	600000	USD	600000	US	100	US	L

607 rows × 11 columns

b. work_year

There are no outliers or incorrect entries.

In [5]:
```
df['work_year'].unique()
```

Out[5]:
```
array([2020, 2021, 2022])
```

c. experience_level

There are no outliers or incorrect entries.

In [6]:
```
df['experience_level'].unique()
```

Out[6]:
```
array(['MI', 'SE', 'EN', 'EX'], dtype=object)
```

d. employment_type

There are no outliers or incorrect entries.

In [7]:
```
df['employment_type'].unique()
```

Out[7]:
```
array(['FT', 'CT', 'PT', 'FL'], dtype=object)
```

e. salary_currency

There are no outliers of incorrect entries.

In [8]:
```
df['salary_currency'].unique()
```

Out[8]:
```
array(['EUR', 'USD', 'GBP', 'HUF', 'INR', 'JPY', 'CNY', 'MXN', 'CAD',
       'DKK', 'PLN', 'SGD', 'CLP', 'BRL', 'TRY', 'AUD', 'CHF'],
      dtype=object)
```

f. employee_residence

There are no outliers or incorrect entries.

In [9]:
```
df['employee_residence'].unique()
```

Out[9]:
```
array(['DE', 'JP', 'GB', 'HN', 'US', 'HU', 'NZ', 'FR', 'IN', 'PK', 'PL',
       'PT', 'CN', 'GR', 'AE', 'NL', 'MX', 'CA', 'AT', 'NG', 'PH', 'ES',
       'DK', 'RU', 'IT', 'HR', 'BG', 'SG', 'BR', 'IQ', 'VN', 'BE', 'UA',
       'MT', 'CL', 'RO', 'IR', 'CO', 'MD', 'KE', 'SI', 'HK', 'TR', 'RS',
       'PR', 'LU', 'JE', 'CZ', 'AR', 'DZ', 'TN', 'MY', 'EE', 'AU', 'BO',
       'IE', 'CH'], dtype=object)
```

g. remote_ratio

There are no outliers or inncorrect entries.

In [10]:
```
df['remote_ratio'].unique()
```

Out[10]:
```
array([  0,  50, 100])
```

h. company_size

There are no outliers or inncorrect entries.

In [11]:
```
df['company_size'].unique()
```

Out[11]:
```
array(['L', 'S', 'M'], dtype=object)
```

i. job_title

In [12]:
```
df['job_title'].unique()
```

Out[12]:
```
array(['Data Scientist', 'Machine Learning Scientist',
       'Big Data Engineer', 'Product Data Analyst',
       'Machine Learning Engineer', 'Data Analyst', 'Lead Data Scientist',
       'Business Data Analyst', 'Lead Data Engineer', 'Lead Data Analyst',
       'Data Engineer', 'Data Science Consultant', 'BI Data Analyst',
       'Director of Data Science', 'Research Scientist',
       'Machine Learning Manager', 'Data Engineering Manager',
       'Machine Learning Infrastructure Engineer', 'ML Engineer',
       'AI Scientist', 'Computer Vision Engineer',
       'Principal Data Scientist', 'Data Science Manager', 'Head of Data',
       '3D Computer Vision Researcher', 'Data Analytics Engineer',
       'Applied Data Scientist', 'Marketing Data Analyst',
       'Cloud Data Engineer', 'Financial Data Analyst',
       'Computer Vision Software Engineer',
       'Director of Data Engineering', 'Data Science Engineer',
       'Principal Data Engineer', 'Machine Learning Developer',
       'Applied Machine Learning Scientist', 'Data Analytics Manager',
       'Head of Data Science', 'Data Specialist', 'Data Architect',
       'Finance Data Analyst', 'Principal Data Analyst',
       'Big Data Architect', 'Staff Data Scientist', 'Analytics Engineer',
       'ETL Developer', 'Head of Machine Learning', 'NLP Engineer',
       'Lead Machine Learning Engineer', 'Data Analytics Lead'],
      dtype=object)
```

DATA PREPROCESSING

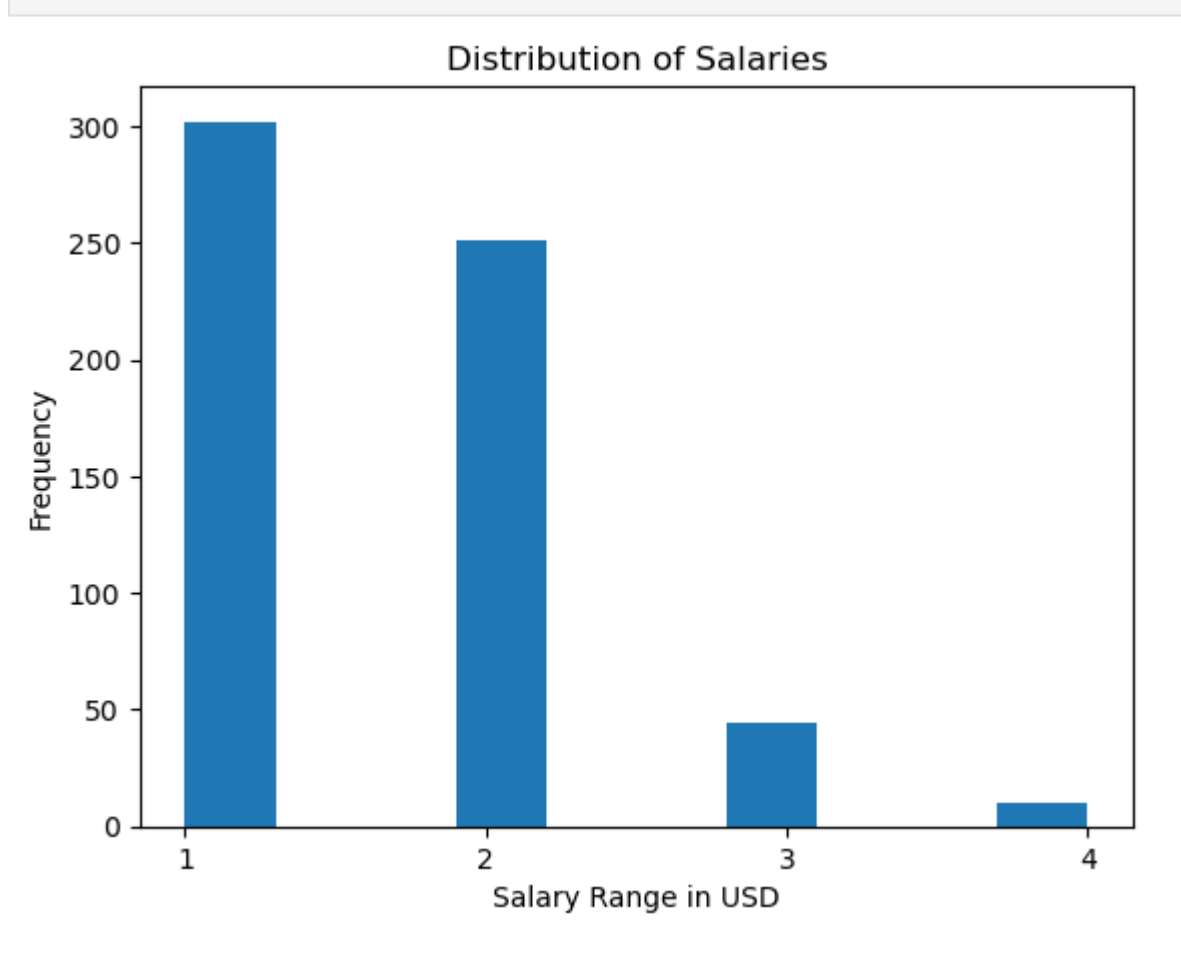
1. Binning

To predict data science salaries, we will change the quantitative variable "salary_in_usd" to categorical variables using binning.

There are 4 bins: Bin 1: (2859, 100000] Bin 2: (100000, 200000] Bin 3: (200000, 300000] Bin 4: (300000, 600000]

In [13]:
```
bins = [df.salary_in_usd.min(), 100000, 200000, 300000, df.salary_in_usd.max()]
labels = ['1', '2', '3', '4']
df['salary_bins'] = pd.cut(df['salary_in_usd'], bins = bins, labels = labels, include_lowest = True)
df = df.sort_values(by = ['salary_in_usd'])
```

In [43]:
```
plt.hist(df['salary_bins'])
plt.title(label = "Distribution of Salaries")
plt.xticks(labels)
plt.xlabel("Salary Range in USD")
plt.ylabel("Frequency")
plt.show()
```



2. Label Encoding

a. experience_level

Converting the categorical labels to ordinal, since the ranking matters.

In [15]:
```
df.loc[df['experience_level'] == 'EN', 'experience_level'] = 0
df.loc[df['experience_level'] == 'MI', 'experience_level'] = 0.25
df.loc[df['experience_level'] == 'SE', 'experience_level'] = 0.5
df.loc[df['experience_level'] == 'EX', 'experience_level'] = 1
```

b. company_size

Converting the categorical labels to ordinal, since the ranking matters.

In [16]:
```
df.loc[df['company_size'] == 'S', 'company_size'] = 0
df.loc[df['company_size'] == 'M', 'company_size'] = 0.5
df.loc[df['company_size'] == 'L', 'company_size'] = 1
```

3. One-hot Encoding

a. Creating dummy variables for the following categorical variables:

1. employment_type
2. salary_currency
3. employee_residence
4. company_location

In [18]:
```
df = pd.get_dummies(df, columns = ['employment_type', 'salary_currency', 'employee_residence', 'company_location'], drop_first = False)
```

b. Creating new dummy variable columns for job_title

In [19]:
```
job_ml = []
job_analyst = []
job_data_scientist = []
job_data_engineer = []
for i in range(0, len(df)):
    #if job_title contains "Machine Learning", "AI", "Computer Vision" or "NLP", we label it as a ML job
    if ("Machine Learning" in df['job_title'].iloc[i]) or ("AI" in df['job_title'].iloc[i]) or ("Computer Vision" in df['job_title'].iloc[i]) or ("NLP" in df['job_title
        job_ml.append(1)
    else:
        job_ml.append(0)
    #if job_title contains "Data Analyst" or "Analytics", we label it as a data analyst job
    if ("Data Analyst" in df['job_title'].iloc[i]) or ("Analytics" in df['job_title'].iloc[i]):
        job_analyst.append(1)
    else:
        job_analyst.append(0)
    #if job_title contains "Data Scientist" or "Data Science", we label it as a data scientist job
    if ("Data Scientist" in df['job_title'].iloc[i]) or ("Data Science" in df['job_title'].iloc[i]):
        job_data_scientist.append(1)
    else:
        job_data_scientist.append(0)
    #if job_title contains "Data Engineer" or "Architect", we label it as a data engineering job
    if ("Data Engineer" in df['job_title'].iloc[i]) or ("Architect" in df['job_title'].iloc[i]):
        job_data_engineer.append(1)
    else:
        job_data_engineer.append(0)

df["job_ml"] = job_ml
df["job_analyst"] = job_analyst
df["job_data_scientist"] = job_data_scientist
df["job_data_engineer"] = job_data_engineer
```

4. Final Dataset

In [20]:
```
df = df.drop(columns = ['salary', 'salary_in_usd', 'job_title']) #dropping these columns as we do not need them in prediction
df
```

Out[20]:

	work_year	experience_level	remote_ratio	company_size	salary_bins	job_ml	job_analyst	job_data_scientist	job_data_engineer	employment_type_CT	...	company_location_PL	company_location_PT	comp
176	2021	0.25	0	0	1	0	0	1	0	0	...	0	0	
185	2021	0.25	100	0.5	1	0	0	0	1	0	...	0	0	
238	2021	0	0	0.5	1	0	0	1	0	0	...	0	0	
77	2021	0.25	50	0.5	1	1	0	0	0	0	...	0	0	
179	2021	0.25	100	0	1	0	0	1	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
225	2021	1	100	0	4	0	0	1	0	1	...	0	0	
157	2021	0.25	50	1	4	1	0	0	0	0	...	0	0	
97	2021	0.25	100	1	4	0	1	0	0	0	...	0	0	
33	2020	0.25	0	0.5	4	0	0	0	0	0	...	0	0	
252	2021	1	100	1	4	0	0	0	1	0	...	0	0	

607 rows × 137 columns

5. Splitting Test and Training Sets

In [21]:
```
X = df.loc[:, df.columns != 'salary_bins'] #features
y = df['salary_bins'] #target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)
```

MACHINE LEARNING ALGORITHMS

1. Logistic Regression

In [22]:
```
def log_reg(X_train, y_train, X_test, y_test):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.fit_transform(X_test)
    classifier = LogisticRegression(multi_class = "ovr")
    classifier.fit(X_train_scaled, y_train)
    y_pred = classifier.predict(X_test_scaled)

    return y_pred
```

2. Gaussian Naive Bayes

In [23]:
```
def naive_bayes(X_train, y_train, X_test, y_test):
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    return y_pred
```

3. Random Forest

In [24]:
```
def rand_forest(X_train, y_train, X_test, y_test):
    classifier = RandomForestClassifier()
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    return y_pred
```

4. K-Nearest Neighbours

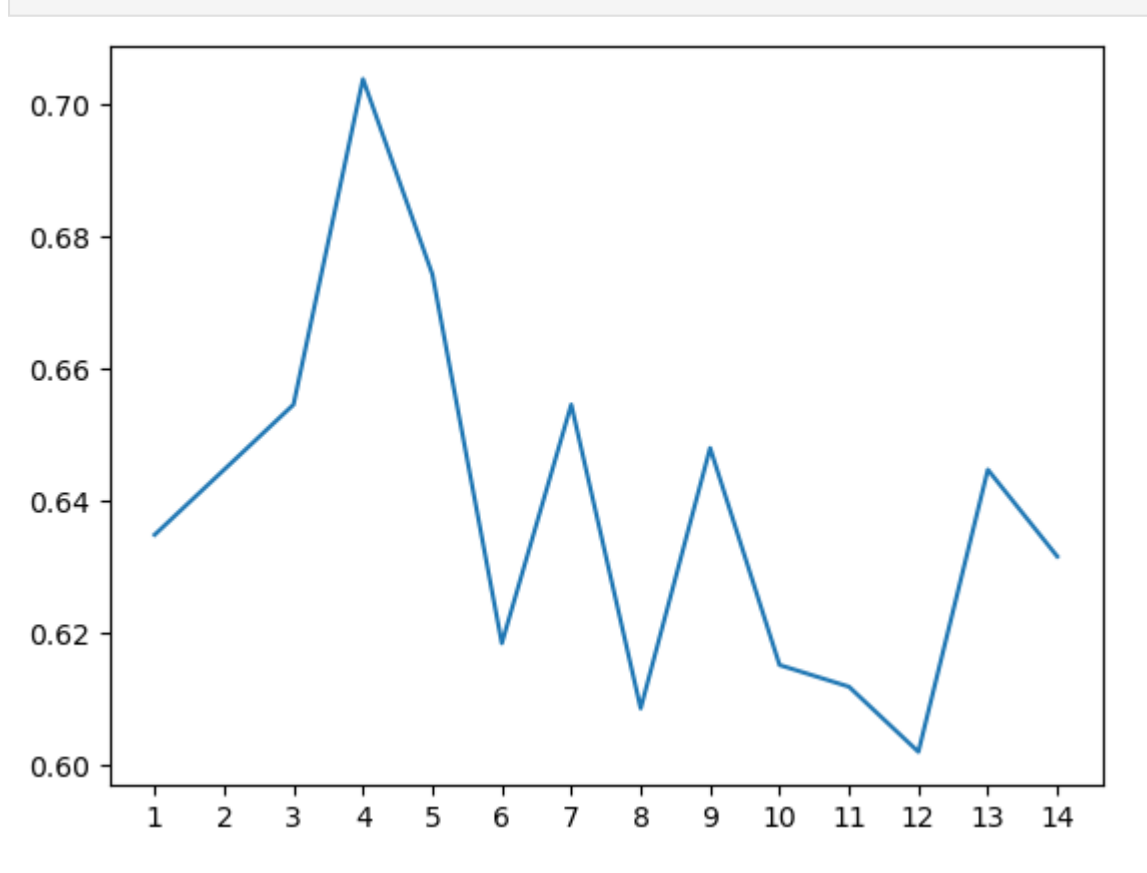
To find the optimal clusters to use in this algorithm, I plotted the accuracies as a function of k and chose the largest value of k with the highest accuracy. As seen in the plot below, I determined that k = 4 will be the optimal number of clusters.

In [25]:
```
def kNN(X_train, y_train, X_test, y_test, i):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.fit_transform(X_test)
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train_scaled, y_train)
    y_pred = classifier.predict(X_test_scaled)
    return y_pred
```

In [26]:
```
#using number of clusters, k from k = 1 to k = 15
k = np.arange(1, 15, 1)

#getting accuracies for each k number of clusters
accuracies = []
for i in k:
    y_pred_kNN = kNN(X_train, y_train, X_test, y_test, i)
    accuracies.append(accuracy_score(y_pred_kNN, y_test))

plt.plot(k, accuracies)
plt.xticks(k)
plt.show()
```



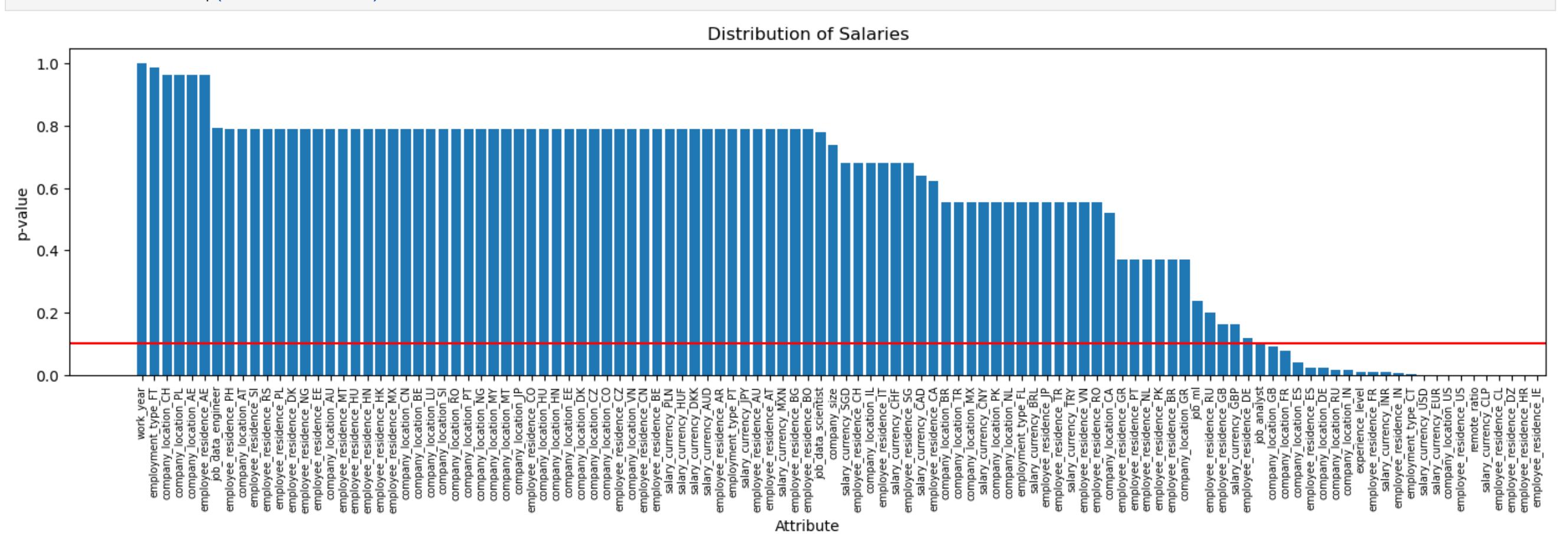
DIMENSIONALITY REDUCTION

1. Chi-square Test

We will use the Chi-square test for dimensionality reduction. We will remove any attribute that has a p-value of less than 0.1.

In [27]:
```
#Chi-Square Test
chi_scores = chi2(X_train, y_train)
results = pd.DataFrame({'Attribute' : X.columns, 'p-value' : chi_scores[1]})
results = results.sort_values(by = ['p-value'], ascending = False)
f = plt.figure()
f.set_figwidth(18)
f.set_figheight(4)
plt.bar(results['Attribute'], results['p-value'])
plt.xticks(rotation = 90, fontsize = 7)
plt.title(label = "Distribution of Salaries")
plt.xlabel("Attribute")
plt.ylabel("p-value")
axhline(0.1, color = "red")

#Reduced Dataset
remove = list(results[results['p-value'] > 0.1]['Attribute'])
X_train_1 = X_train.drop(columns = remove)
X_test_1 = X_test.drop(columns = remove)
```

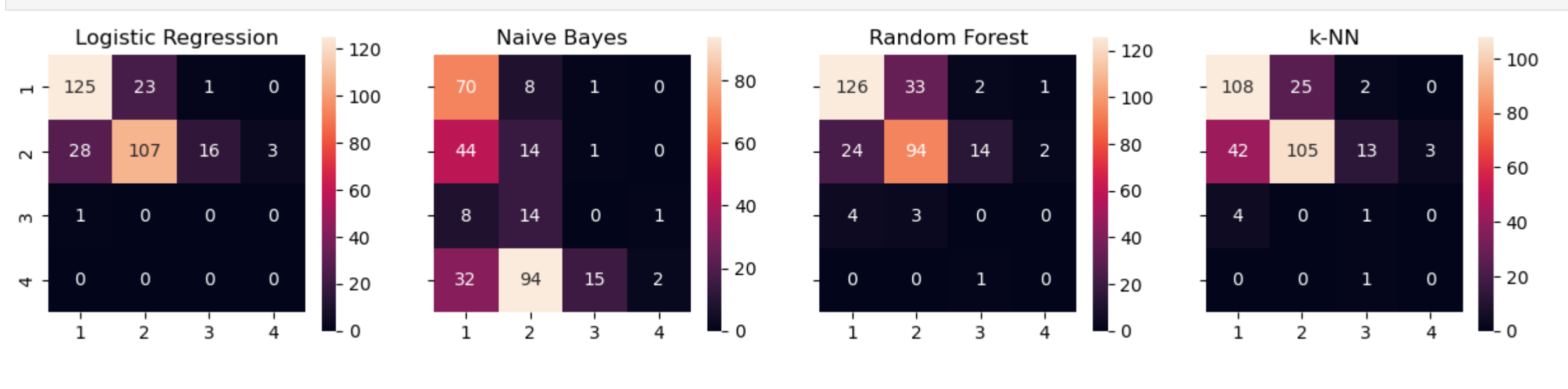


MODEL EVALUATION

1. Without dimensionality reduction

In [35]:
```
y_pred_lr = log_reg(X_train, y_train, X_test, y_test)
y_pred_nb = naive_bayes(X_train, y_train, X_test, y_test)
y_pred_rf = rand_forest(X_train, y_train, X_test, y_test)
y_pred_kNN = kNN(X_train, y_train, X_test, y_test, 4)
y_preds = [y_pred_lr, y_pred_nb, y_pred_rf, y_pred_kNN]
```

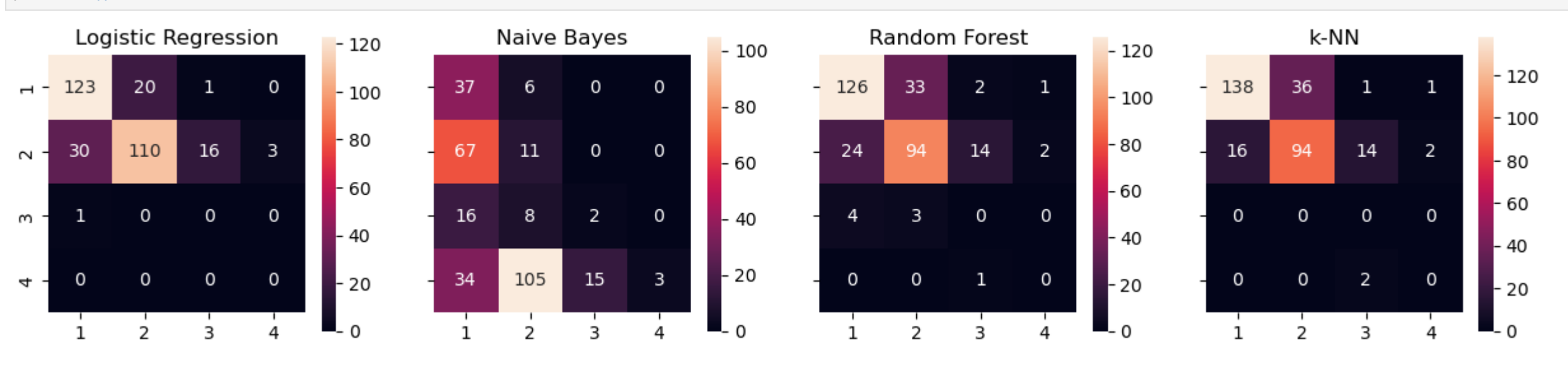
In [36]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, sharey=True, figsize=(15, 3))
mat_lr = confusion_matrix(y_test, y_pred_lr)
mat_nb = confusion_matrix(y_test, y_pred_nb)
mat_rf = confusion_matrix(y_test, y_pred_rf)
mat_kNN = confusion_matrix(y_test, y_pred_kNN)
sns.heatmap(mat_lr.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax1).set(title = "Logistic Regression")
sns.heatmap(mat_nb.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax2).set(title = "Naive Bayes")
sns.heatmap(mat_rf.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax3).set(title = "Random Forest")
sns.heatmap(mat_kNN.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax4).set(title = "k-NN")
plt.show()
```



2. With dimensionality reduction

In [37]:
```
y_pred_lr_1 = log_reg(X_train_1, y_train, X_test_1, y_test)
y_pred_nb_1 = naive_bayes(X_train_1, y_train, X_test_1, y_test)
y_pred_rf_1 = rand_forest(X_train_1, y_train, X_test_1, y_test)
y_pred_kNN_1 = kNN(X_train_1, y_train, X_test_1, y_test, 4)
y_preds_1 = [y_pred_lr_1, y_pred_nb_1, y_pred_rf_1, y_pred_kNN_1]
```

In [38]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, sharey=True, figsize=(15, 3))
mat_lr_1 = confusion_matrix(y_test, y_pred_lr_1)
mat_nb_1 = confusion_matrix(y_test, y_pred_nb_1)
mat_rf_1 = confusion_matrix(y_test, y_pred_rf)
mat_kNN_1 = confusion_matrix(y_test, y_pred_kNN_1)
sns.heatmap(mat_lr_1.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax1).set(title = "Logistic Regression")
sns.heatmap(mat_nb_1.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax2).set(title = "Naive Bayes")
sns.heatmap(mat_rf_1.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax3).set(title = "Random Forest")
sns.heatmap(mat_kNN_1.T,square=True, annot=True, fmt = 'd', cbar=True, xticklabels=[1,2,3,4], yticklabels=[1,2,3,4], ax = ax4).set(title = "k-NN")
plt.show()
```



3. Performance

In [39]:
```
accuracies = []
for i in y_preds:
    accuracies.append(accuracy_score(i, y_test))

accuracies_1 = []
for i in y_preds_1:
    accuracies_1.append(accuracy_score(i, y_test))
```

In [40]:
```
models = ["Logistic Regression", "Gaussian Naive Bayes", "Random Forest", "K-Nearest Neighbours"]
table = pd.DataFrame({"Model": models, "Accuracy without Dimensionality Reduction": accuracies, "Accuracy with Dimensionality Reduction": accuracies_1})
```

In [41]:
```
table
```

Out[41]:

	Model	Accuracy without Dimensionality Reduction	Accuracy with Dimensionality Reduction
0	Logistic Regression	0.763158	0.766447
1	Gaussian Naive Bayes	0.282895	0.174342
2	Random Forest	0.723684	0.756579
3	K-Nearest Neighbours	0.703947	0.763158

Conclusion

The best model for this dataset is Logistic Regression with Chi-square test dimensionality reduction method with an accuracy of 0.766447. The Chi-square test dimensionality reduction method improved accuracy but not by a significant amount.

The worst performing model is Gaussian Naive Bayes overall, with an accuracy of 0.282895 without dimensionality reduction and 0.174342 with dimensionality reduction.