Glassdoor Reviews

Nicolas Alvarez, Zachary Brazelton, Daniel Gabriel, Pratheek Gandhodi

Part One: Data Introduction

What is Glassdoor?

- Companies post Jobs
- Previous Employees Review the Company
- Trustworthy

Dataset

- https://www.kaggle.com/datasets/davidgauthier/glassdoor-job-reviews
- Data pulled from industries in the United Kingdom
- Only from companies on Glassdoor
- Last Updated November 2022

Features

	overall_rating	work_life_balance	culture_values	career_opp	comp_benefits	senior_mgmt	recommend	ceo_approv	outlook
661354	2	2.0	2.0	3.0	3.0	1.0	x	x	x
512064	4	4.0	4.0	4.0	3.0	4.0	v	x	r
146071	1	3.0	1.0	2.0	2.0	1.0	x	x	r
153117	4	2.0	2.0	5.0	2.0	3.0	v	r	v
527213	5	5.0	5.0	3.0	5.0	5.0	v	v	v

(v - Positive, r - Mild, x - Negative, o - No opinion)

- Work Life Balance- (1-5)
- Culture and Values- (1-5)
- Career Opportunities- (1-5)
- Compensation and Benefits- (1-5)
- Senior Management- (1-5)
- Recommend?
- Approve of CEO?
- Company Outlook?

Part Two: Data Preparation

- Conversions
- Which Variables?
- Missing Values
- Sampling the Data
- Final Modifications

Conversions

- We converted some of our main variables to categorical.
  - Overall rating, recommend, ceo approval, outlook
- Converted text variables to string
  - However…

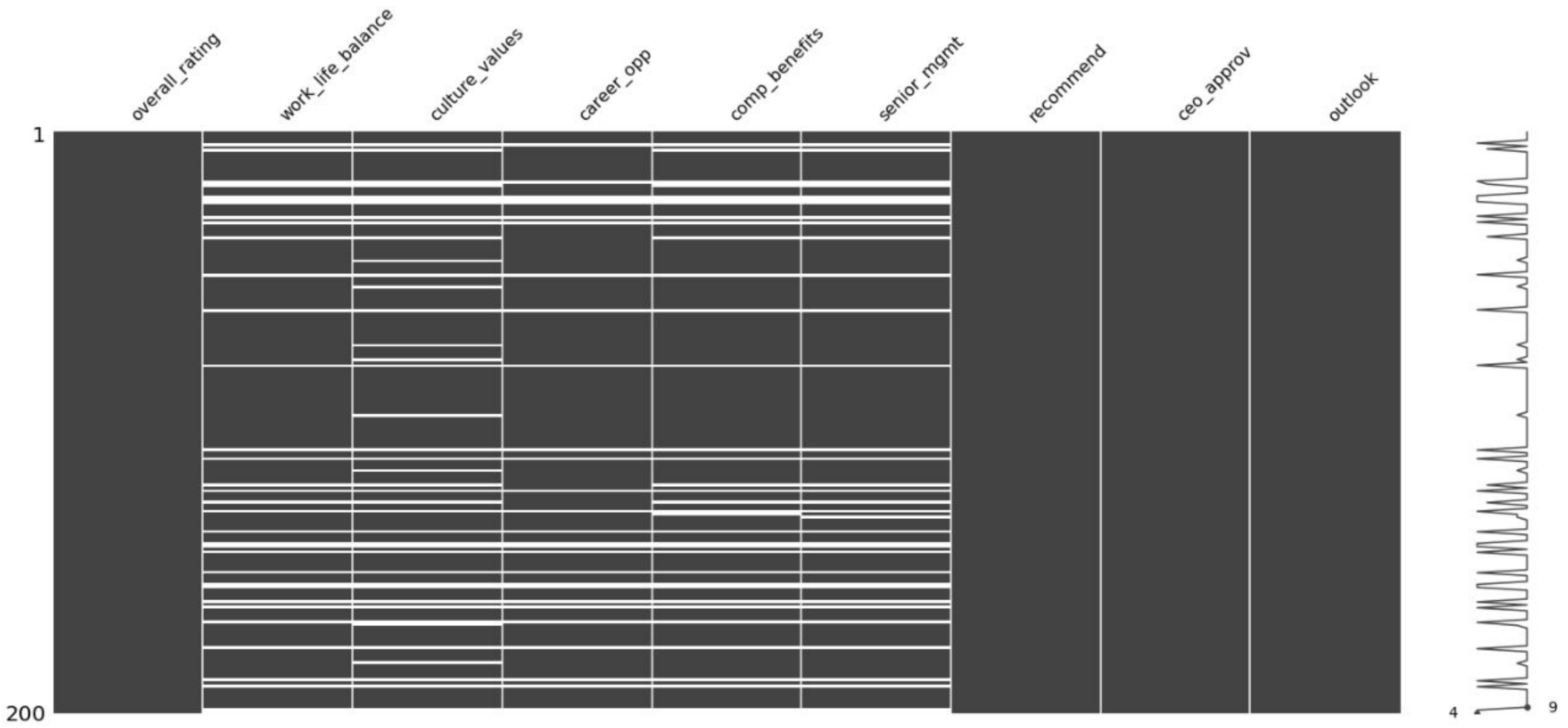
Which Variables?

- Before further data prep, we kept 11 variables, a mix of categorical and string types.
- We removed some string variables due to lack of relevance, or due to effort needed
- We wanted to keep useful numeric and categorical variables for our analysis

Missing Values

- We examined what percentage of data values were missing.
- We removed all with more that 30% missing
    - Diversity/inclusion and location

```
diversity_inclusion    83.773967
location               35.457913
culture_values         22.821459
senior_mgmt            18.588400
comp_benefits          17.897458
work_life_balance      17.875039
career_opp             17.589671
overall_rating          0.000000
recommend               0.000000
ceo_approv              0.000000
outlook                 0.000000
dtype: float64
```



Examining the remaining missing values, we noticed that most occurred in multiple variables. As a result, we decided to simply remove each row that contained any amount of missing values

Sampling the Data

- With around 700,000 rows of data, we decided to sample the full amount to save our computers
- We decided on 200,000 as our new amount, sampled randomly from the full dataset

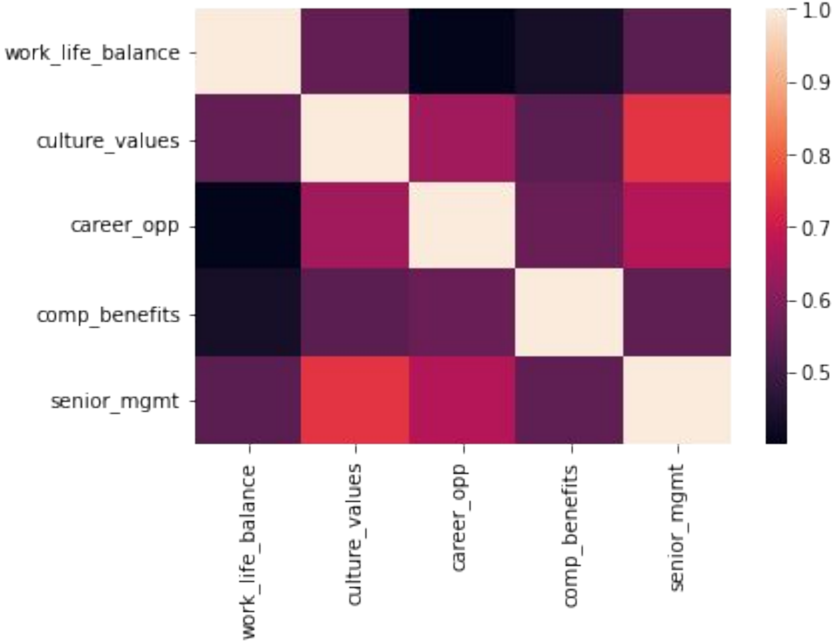
```
[ ] Glassdoor.head()
```

	overall_rating	work_life_balance	culture_values	career_opp	comp_benefits	senior_mgmt	recommend	ceo_approv	outlook
661354	2	2.0	2.0	3.0	3.0	1.0	x	x	x
512064	4	4.0	4.0	4.0	3.0	4.0	v	x	r
146071	1	3.0	1.0	2.0	2.0	1.0	x	x	r
153117	4	2.0	2.0	5.0	2.0	3.0	v	r	v
527213	5	5.0	5.0	3.0	5.0	5.0	v	v	v

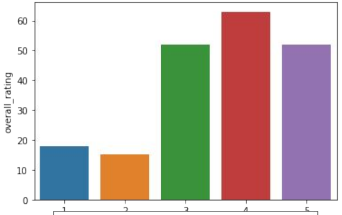
Final Modifications

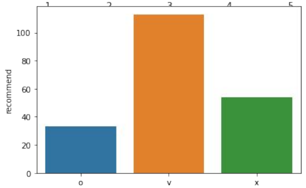
- We used an 80/20 split of training and testing data in our analysis
- We scaled our data using sklearn preprocessing
  - MinMaxScaler
- Now we're ready to examine our data!

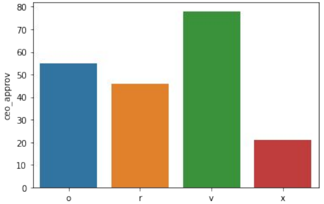
Part Three: Exploratory Analysis

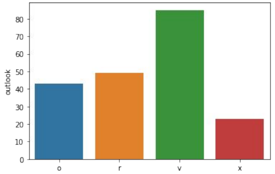


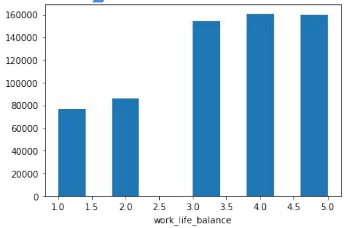
Part Three: Exploratory Analysis

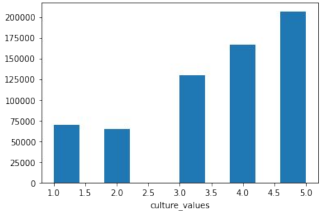


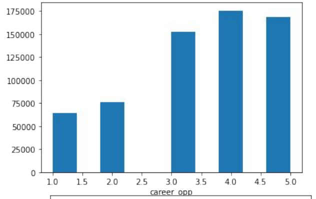


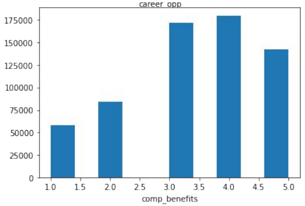












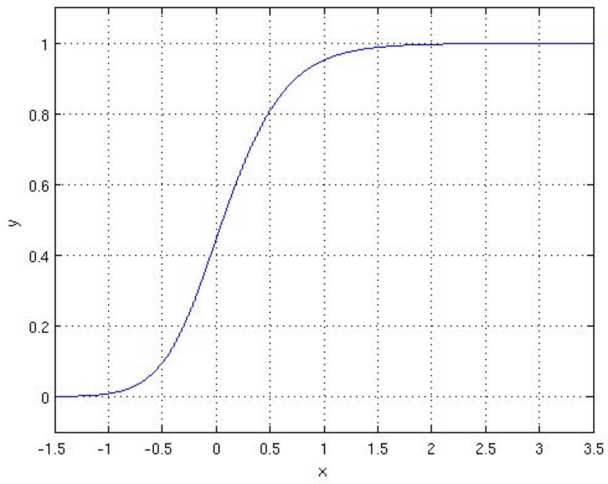
Part Four: Models

- Linear/Logistic
- Naive Bayes
- K-Nearest Neighbors
- Decision Tree
- Random Forest
- Why not XGBoost or SVM?
- Conclusions

Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression

    # instantiate the model (using the default parameters)
    logistic = LogisticRegression(max_iter=150)

    # fit the model with data
    logistic.fit(Glassdoor_X_train, Glassdoor_y_train)

    LogisticRegression(max_iter=150)
```

```
[ ] print(logistic.coef_)
    print(logistic.intercept_)

    [[-1.95242692 -3.75711558 -3.78972863 -2.08439561 -3.68671128 -0.69177887
       0.94095359  0.08656158  0.18106987  0.27764356 -0.04512147 -0.06787479
       0.60246681]
     [-0.77054765 -1.82157755 -1.62318222 -1.27364003 -1.62045231 -0.64152053
       0.78862091  0.10434279 -0.01164666  0.03271673  0.09374033 -0.11158389
       0.26577922]
     [-0.05053877 -0.18576228 -0.15014621 -0.42097036  0.06425468  0.03055761
      -0.10167433  0.09946712 -0.04957546 -0.0637693   0.23291445 -0.05444095
       0.04590992]
     [ 0.80715948  1.69383382  1.58135961  0.97273984  1.66169261  0.64844206
      -0.86237828 -0.04914565 -0.10440137 -0.22525117 -0.0585166   0.00994424
      -0.47234849]
     [ 1.96635386  4.07062159  3.98169744  2.80626616  3.58121631  0.65429973
      -0.76552188 -0.24122584 -0.01544638 -0.02133982 -0.22301671  0.22395538
      -0.44180746]]
    [ 5.37604375  4.22413716  2.29862832 -2.10959584 -9.7892134 ]
```



*Not Actual Representation of Data

Naive Bayes

```
[ ]  from sklearn.naive_bayes import GaussianNB

     cnb = GaussianNB()

     naive_bayes = cnb.fit(Glassdoor_X_train, Glassdoor_y_train)
```

K-Nearest Neighbors

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
# Setup the parameters and distributions to sample from: param_dist
param_dist = {"n_neighbors": range(1, 75)}

knn_class = KNeighborsClassifier(weights = "distance")

# Instantiate the GridSearchCV object: knn_cv
knn_cv = RandomizedSearchCV(knn_class, param_dist, cv=5)

# Fit it to the data
knn_cv.fit(Glassdoor_X_train, Glassdoor_y_train)

# Print the tuned parameters and score
print("Tuned K Nearest Neighbors Parameters: {}".format(knn_cv.best_params_))
print("Best score is {}".format(knn_cv.best_score_))
```

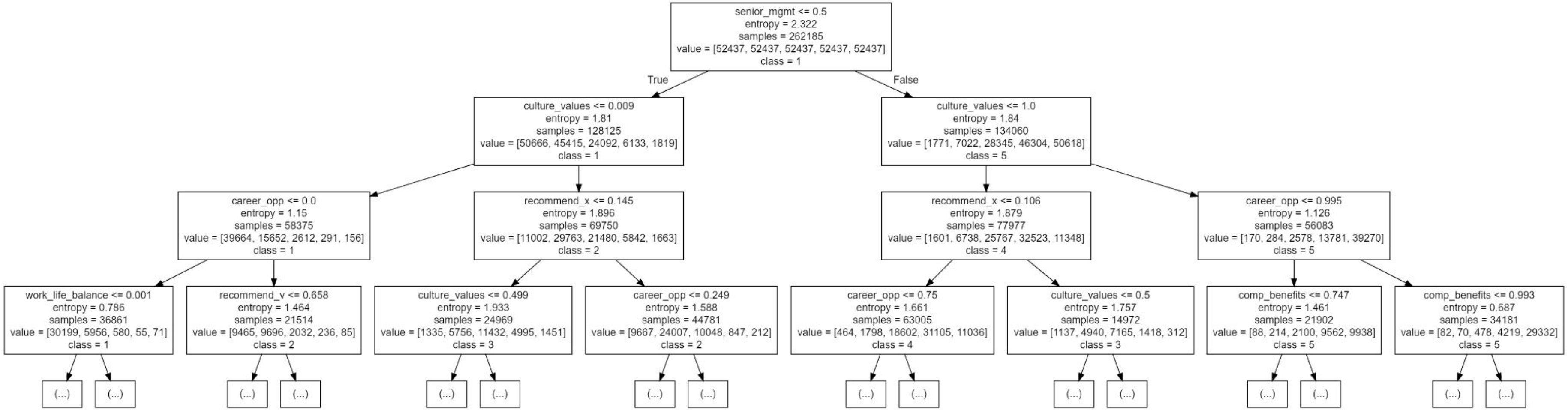
```
Tuned K Nearest Neighbors Parameters: {'n_neighbors': 65}
Best score is 0.6802952368204677
```

Decision Tree

```
[ ]  from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import GridSearchCV

     # Setup the parameters and distributions to sample from: param_dist
     param_dist = {"max_depth": [29, 31, 37, 41, 43, 47],
                   "min_samples_leaf": [1, 2, 3]}

     # Instantiate a Gradient Boosted Random Forest classifier: tree
     GDTree = DecisionTreeClassifier(criterion="entropy")

     # Instantiate the GridSearchCV object: tree_cv
     GDTree_cv = GridSearchCV(GDTree, param_dist, cv=5)

     # Fit it to the data
     GlassdoorTree = GDTree_cv.fit(Glassdoor_X_train, Glassdoor_y_train)

     # Print the tuned parameters and score
     print("Tuned Tree Parameters: {}".format(GDTree_cv.best_params_))
     print("Best score is {}".format(GDTree_cv.best_score_))
```

```
Tuned Tree Parameters: {'max_depth': 43, 'min_samples_leaf': 2}
Best score is 0.682815457865652
```

Decision Tree



Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
# Setup the parameters and distributions to sample from: param_dist
param_dist = {"max_depth": [13, 17, 19, 23, 29],
              "min_samples_leaf": list(range(1,3)),
              "n_estimators": list(range(1,100))}

# Instantiate a Gradient Boosted Random Forest classifier: forest
forest = RandomForestClassifier()

# Instantiate the RandomizedSearchCV object: forest_cv
# I will use randomized search because grid search is taking too long
forest_cv = RandomizedSearchCV(forest, param_dist, cv=5, n_iter=15)

# Fit it to the data
forest_cv.fit(Glassdoor_X_train, Glassdoor_y_train)

# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(forest_cv.best_params_))
print("Best score is {}".format(forest_cv.best_score_))
```

```
Tuned Random Forest Parameters: {'n_estimators': 71, 'min_samples_leaf': 2, 'max_depth': 23}
Best score is 0.7070233474195462
```

Why not XGB or SVM?

- Performance
  - XGBoost took over 2 hours to complete, while SVM never completed
- Accuracy
  - XGBoost had no improvement over other models

Results

```
log_pred = logistic.predict(Glassdoor_X_test)
print("Logistic Regression: " + str(accuracy_score(Glassdoor_y_test, log_pred)))

naive_pred = naive_bayes.predict(Glassdoor_X_test)
print("Naive Bayes: " + str(accuracy_score(Glassdoor_y_test, naive_pred)))

knn_pred = knn_cv.predict(Glassdoor_X_test)
print("K Nearest Neighbors: " + str(accuracy_score(Glassdoor_y_test, knn_pred)))

tree_pred = GDTree_cv.predict(Glassdoor_X_test)
print("Decision Tree: " + str(accuracy_score(Glassdoor_y_test, tree_pred)))

forest_pred = forest_cv.predict(Glassdoor_X_test)
print("Random Forest: " + str(accuracy_score(Glassdoor_y_test, forest_pred)))
```

```
Logistic Regression: 0.636725
Naive Bayes: 0.564525
K Nearest Neighbors: 0.628225
Decision Tree: 0.6227
Random Forest: 0.64315
```

Classification Report for Logistic

```
from sklearn.metrics import classification_report
print(classification_report(Glassdoor_y_test, log_pred))
```

```
              precision    recall  f1-score   support

           1       0.64      0.76      0.70      3087
           2       0.41      0.53      0.46      3635
           3       0.59      0.52      0.55      9112
           4       0.65      0.61      0.63     13074
           5       0.75      0.77      0.76     11092

    accuracy                           0.64     40000
   macro avg       0.61      0.64      0.62     40000
weighted avg       0.64      0.64      0.64     40000
```

Classification Report for Random Forest

```
[ ]  print(classification_report(Glassdoor_y_test, forest_pred))
```

	precision	recall	f1-score	support
1	0.67	0.73	0.70	3087
2	0.44	0.48	0.46	3635
3	0.57	0.59	0.58	9112
4	0.65	0.62	0.63	13074
5	0.77	0.74	0.75	11092
accuracy			0.64	40000
macro avg	0.62	0.63	0.63	40000
weighted avg	0.65	0.64	0.64	40000

Comparison

```
from sklearn.metrics import classification_report
print(classification_report(Glassdoor_y_test, log_pred))
```

	precision	recall	f1-score	support
1	0.64	0.76	0.70	3087
2	0.41	0.53	0.46	3635
3	0.59	0.52	0.55	9112
4	0.65	0.61	0.63	13074
5	0.75	0.77	0.76	11092
accuracy			0.64	40000
macro avg	0.61	0.64	0.62	40000
weighted avg	0.64	0.64	0.64	40000

```
[ ] print(classification_report(Glassdoor_y_test, forest_pred))
```

	precision	recall	f1-score	support
1	0.67	0.73	0.70	3087
2	0.44	0.48	0.46	3635
3	0.57	0.59	0.58	9112
4	0.65	0.62	0.63	13074
5	0.77	0.74	0.75	11092
accuracy			0.64	40000
macro avg	0.62	0.63	0.63	40000
weighted avg	0.65	0.64	0.64	40000

Confusion Matrices

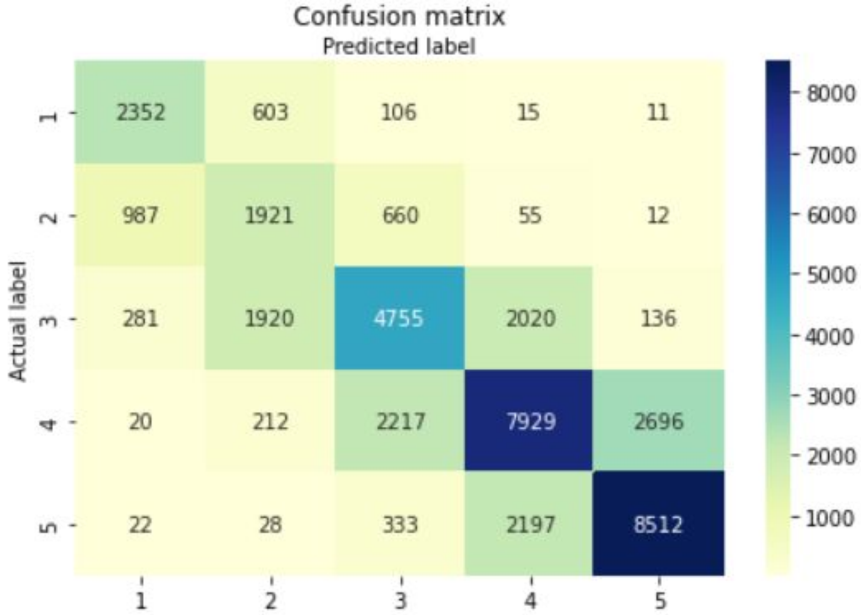
```
[ ] from sklearn.metrics import confusion_matrix
    log_cnf_matrix = confusion_matrix(Glassdoor_y_test,log_pred)
```

```
[ ] forest_cnf_matrix = confusion_matrix(Glassdoor_y_test,forest_pred)
```

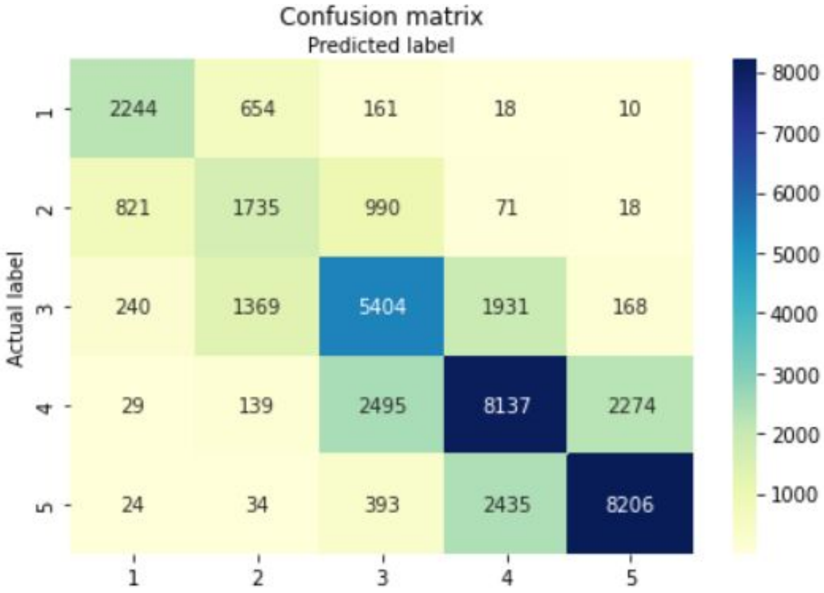
```
[ ] class_names=[1,2,3,4,5]
    fig, ax = plt.subplots()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names)
    plt.yticks(tick_marks, class_names)

    sns.heatmap(pd.DataFrame(log_cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g',xticklabels=class_names, yticklabels=class_names)
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

Comparison

Logistic Regression



Random Forest



Conclusions