Predicting the Attrition Rate of the Company

IST 652 Scripting for Data Analysis
Prof. Ying Lin

- Project by Manasi Todankar

Attrition Rate of a Company

What is Attrition Rate?

Employee attrition refers to the loss of employees through a natural process, such as retirement, resignation, elimination of a position, personal health, or other similar reasons. With attrition, an employer will not fill the vacancy left by the former employee.

Why is it important?

As an employer, attrition is important to understand because it can decrease labor costs without incorporating staff departures. As employees retire, the company can perform what is called a hiring freeze. It means that when employees start to retire, the company doesn't replace them.

Objective

The objective of this project is to predict the attrition rate of each employee. To find out which employee id more likely to leave the company. It can help the company to find solutions to prevent attrition or plan in advance for hiring of a new candidate.

Process

1. Data Description
2. Data Exploration
3. Data Cleaning
4. Data Visualization
5. Data Modelling
6. Model Evaluation
7. Result Interpretation

Data

Dataset: EDA-Analyzing the Attrition Rate of a Company

Link: https://www.kaggle.com/code/muhammedsal98/eda-analyzing-the-attrition-rate-of-a-company/notebook

Libraries & Data Description

Libraries used:

- Numpy
- Pandas
- Seaborn
- Sklearn
- Graphviz
- Matplotlib

29 columns x 4410 rows

Target column: ‘Attrition’
Predictive column: Rest of the columns

Libraries & Data Description

EmployeeID	Identification number of the employee
Age	Age of the employee
Attrition	Whether the employee left in the previous year or not
BusinessTravel	Frequency of business travel in the last year
Department	Department in the company
DistanceFromHome	Distance from home in km
Education	Education Level: 1 'Below College', 2 'College', 3 'Bachelor', 4 'Master', 5 'Doctor'
Education Field	Field of education
EmployeeCount	Employee count
Gender	Gender of employee
JobLevel	Job level in the company on a scale of 1 to 5
JobRole	Name of job role in the company
MaritalStatus	Married, Single or other
MonthlyIncome	Monthly income in rupees per month
NumCompaniesWorked	Total number of companies the employee has worked for
Over18	Whether the employee is above 18 years of age or not
PercentSalaryHike	Percent salary hike for last year
StandardHours	Standard hours of work for the employee
StockOptionLevel	Stock option level of the employee
TotalWorkingHours	Total number of years the employee has worked so far
TrainingTimesLastYear	Number of times training was conducted for this employee last year
YearsAtCompany	Total number of years spent at the company by the employee
YearsSinceLastPromotion	Number of years since last promotion
YearsWithCurrManager	Number of years under current manager
EnvironmentSatisfaction	Work Environment Satisfaction Level 1 'Low' 2 'Medium' 3 'High' 4 'Very High'
JobSatisfaction	Job Satisfaction Level: 1 'Low', 2 'Medium', 3 'High', 4 'Very High'
WorkLifeBalance	Work life balance level 1 'Bad' 2 'Good' 3 'Better' 4 'Best'
JobInvolvement	Job Involvement Level 1 'Low' 2 'Medium' 3 'High' 4 'Very High'
PerformaceRating	Performance rating for last year 1 'Low' 2 'Good' 3 'Excellent' 4 'Outstanding'

Data Cleaning

Pre-processing:

- Making data ready

Steps:

- Taking care of missing data and dropping non-relevant features.
- Feature extraction
- Converting categorical gestures into numerical form
- Feature scaling
- Understanding correlations
- Splitting data into training and test data sets

Data Cleaning

```
df.isna().sum()
```

The dataset has 'na' values in 4 columns:

- NumCompaniesWorked
- EnvironmentSatisfaction
- WorklifeBalance
- JobSatisfaction

Imputing the values with mean

```
from sklearn.impute import SimpleImputer

imputeCol = SimpleImputer(strategy = 'mean')
imputeCol = imputeCol.fit(df[['NumCompaniesWorked', 'TotalWorkingYears', 'EnvironmentSatisfaction', 'JobSatisfaction', 'WorkLifeBalance']])
df[['NumCompaniesWorked', 'TotalWorkingYears', 'EnvironmentSatisfaction', 'JobSatisfaction', 'WorkLifeBalance']] = imputeCol.transform(df[['NumCompaniesWorked'
```

Data Exploration

	EmployeeID	Age	DistanceFromHome	Education	EmployeeCount	JobLevel	MonthlyIncome	NumCompaniesWorked
count	4410.000000	4410.000000	4410.000000	4410.000000	4410.0	4410.000000	4410.000000	4410.000000
mean	2205.500000	36.923810	9.192517	2.912925	1.0	2.063946	65029.312925	2.694830
std	1273.201673	9.133301	8.105026	1.023933	0.0	1.106689	47068.888559	2.493497
min	1.000000	18.000000	1.000000	1.000000	1.0	1.000000	10090.000000	0.000000
25%	1103.250000	30.000000	2.000000	2.000000	1.0	1.000000	29110.000000	1.000000
50%	2205.500000	36.000000	7.000000	3.000000	1.0	2.000000	49190.000000	2.000000
75%	3307.750000	43.000000	14.000000	4.000000	1.0	3.000000	83800.000000	4.000000
max	4410.000000	60.000000	29.000000	5.000000	1.0	5.000000	199990.000000	9.000000

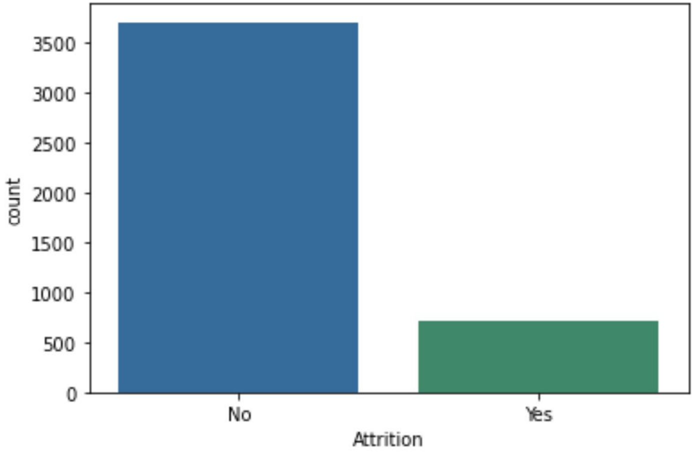
PercentSalaryHike	StandardHours	...	TotalWorkingYears	TrainingTimesLastYear	YearsAtCompany	YearsSinceLastPromotion	YearsWithCurrManager
4410.000000	4410.0	...	4410.000000	4410.000000	4410.000000	4410.000000	4410.000000
15.209524	8.0	...	11.279936	2.799320	7.008163	2.187755	4.123129
3.659108	0.0	...	7.774275	1.288978	6.125135	3.221699	3.567327
11.000000	8.0	...	0.000000	0.000000	0.000000	0.000000	0.000000
12.000000	8.0	...	6.000000	2.000000	3.000000	0.000000	2.000000
14.000000	8.0	...	10.000000	3.000000	5.000000	1.000000	3.000000
18.000000	8.0	...	15.000000	3.000000	9.000000	3.000000	7.000000
25.000000	8.0	...	40.000000	6.000000	40.000000	15.000000	17.000000

EnvironmentSatisfaction	JobSatisfaction	WorkLifeBalance	JobInvolvement	PerformanceRating
4410.000000	4410.000000	4410.000000	4410.000000	4410.000000
2.723603	2.728246	2.761436	2.729932	3.153741
1.089654	1.098753	0.703195	0.711400	0.360742
1.000000	1.000000	1.000000	1.000000	3.000000
2.000000	2.000000	2.000000	2.000000	3.000000
3.000000	3.000000	3.000000	3.000000	3.000000
4.000000	4.000000	3.000000	3.000000	3.000000
4.000000	4.000000	4.000000	4.000000	4.000000

```
df.describe()
```

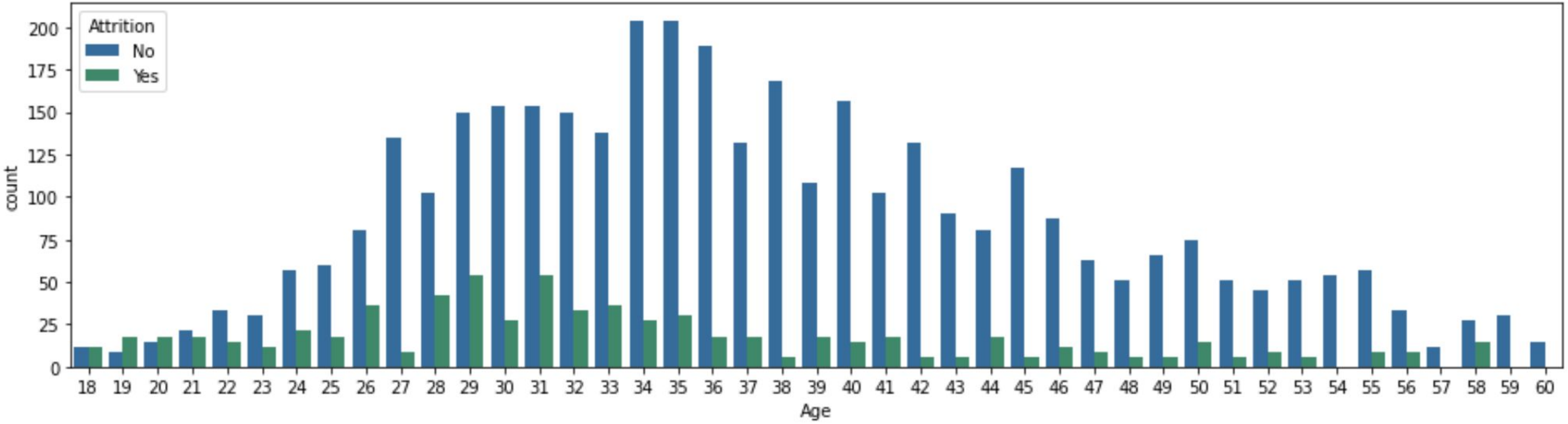
Data Visualizations

```
import matplotlib.pyplot as plt

sns.countplot(x = df['Attrition'])
plt.show()
```



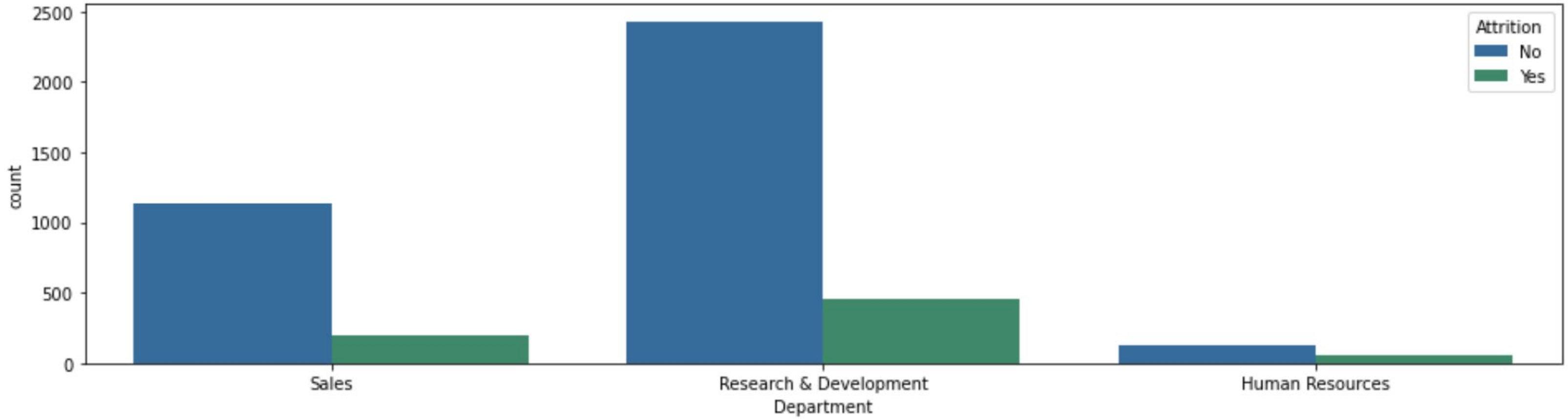
Data Visualizations

```
plt.subplots(figsize = (16,4))
sns.countplot(x = 'Age', hue = 'Attrition', data = df)
plt.show()
```



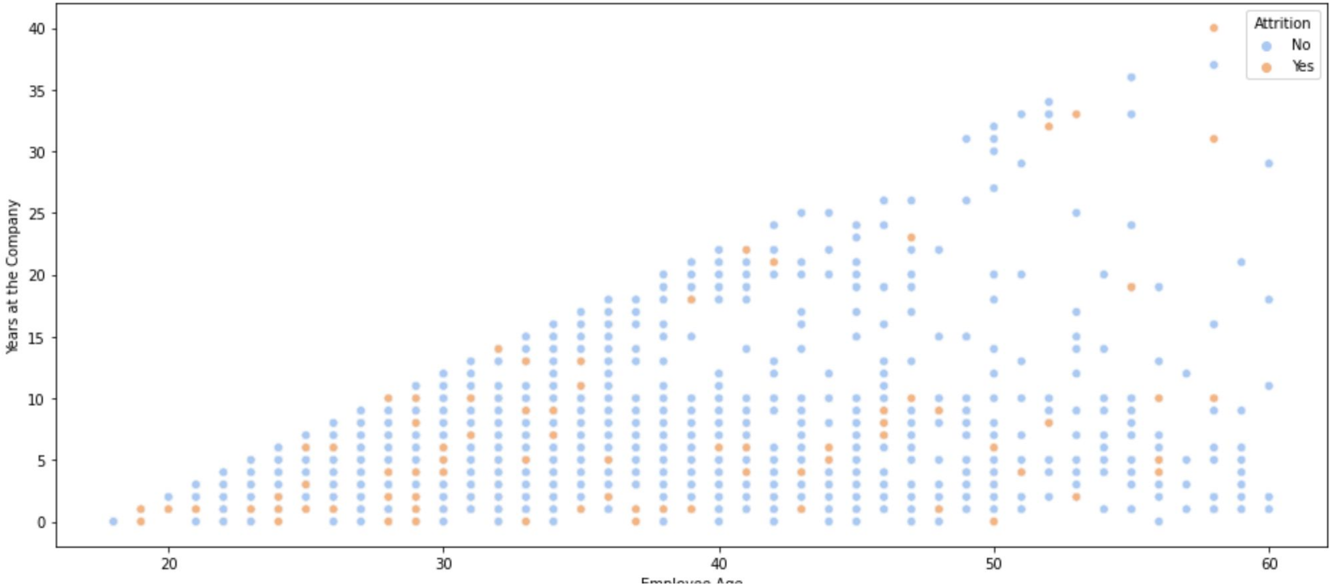
Data Visualization

```
plt.subplots(figsize = (16,4))
sns.countplot(x = 'Department', hue = 'Attrition', data = df)
plt.show()
```



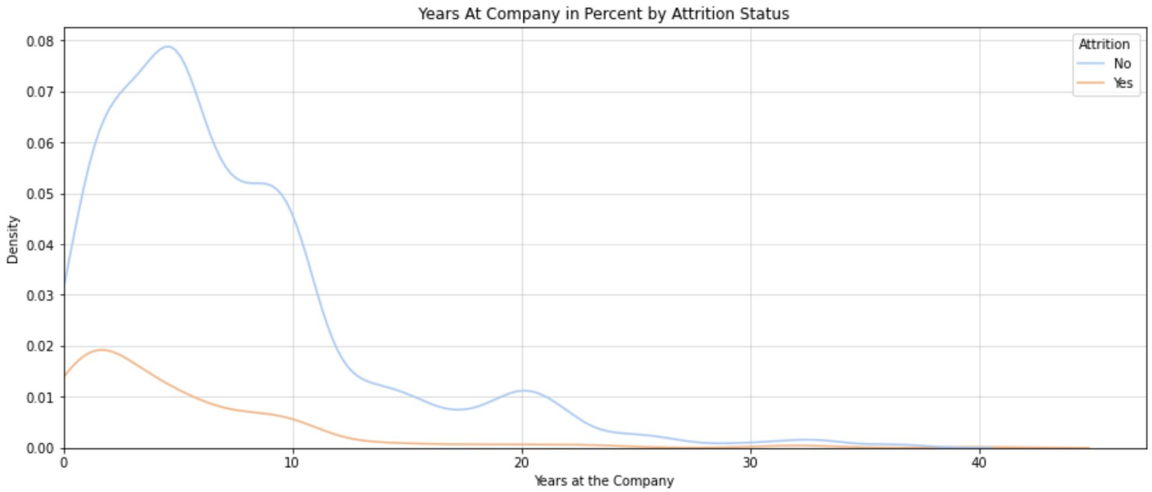
Data Visualizations

```python
import seaborn

plt.subplots(figsize = (16,7))
seaborn.scatterplot(x = 'Age', y = 'YearsAtCompany', hue = 'Attrition', data = df, palette = 'pastel')
plt.xlabel('Employee Age')
plt.ylabel('Years at the Company')
plt.legend(title = 'Attrition')
plt.show()
```



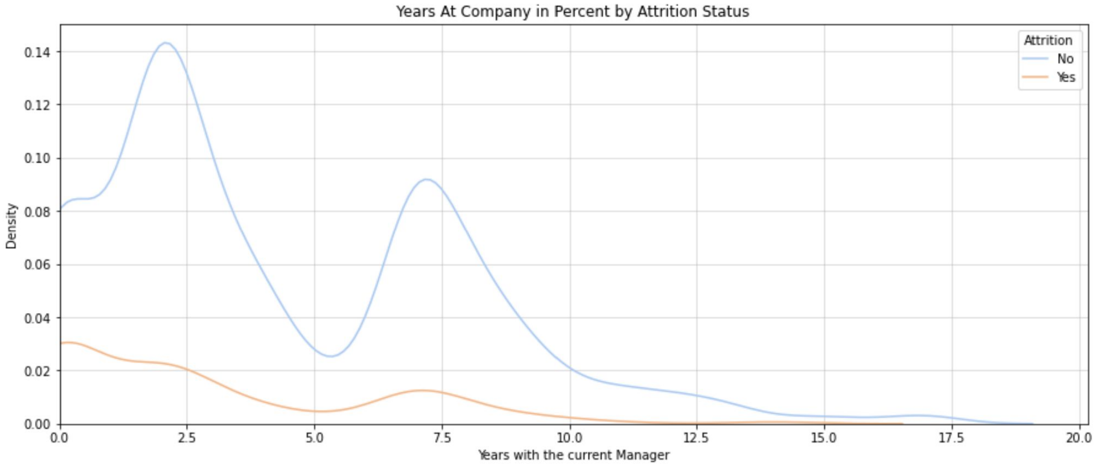
Data Visualizations

```
import seaborn as sns

plt.figure(figsize=(15,6))
plt.style.use('seaborn-colorblind')
plt.grid(True, alpha=0.5)
sns.kdeplot(data = df, x = df['YearsAtCompany'], hue = 'Attrition', palette = 'pastel')
plt.xlabel('YearsAtCompany')
plt.xlim(left=0)
plt.xlabel('Years at the Company')
plt.ylabel('Density')
plt.title('Years At Company in Percent by Attrition Status')
plt.show()
```



Data Visualizations

```
plt.figure(figsize=(15,6))
plt.style.use('seaborn-colorblind')
plt.grid(True, alpha=0.5)
sns.kdeplot(data = df, x = df['YearsWithCurrManager'], hue = 'Attrition', palette = 'pastel')
plt.xlabel('YearsWithCurrManager')
plt.xlim(left=0)
plt.ylabel('Density')
plt.xlabel('Years with the current Manager')
plt.title('Years At Company in Percent by Attrition Status')
plt.show()
```



Data Cleaning

```
df['EmployeeCount'].unique()
```

```
array([1], dtype=int64)
```

```
df['Over18'].unique()
```

```
array(['Y'], dtype=object)
```

```
df['StandardHours'].unique()
```

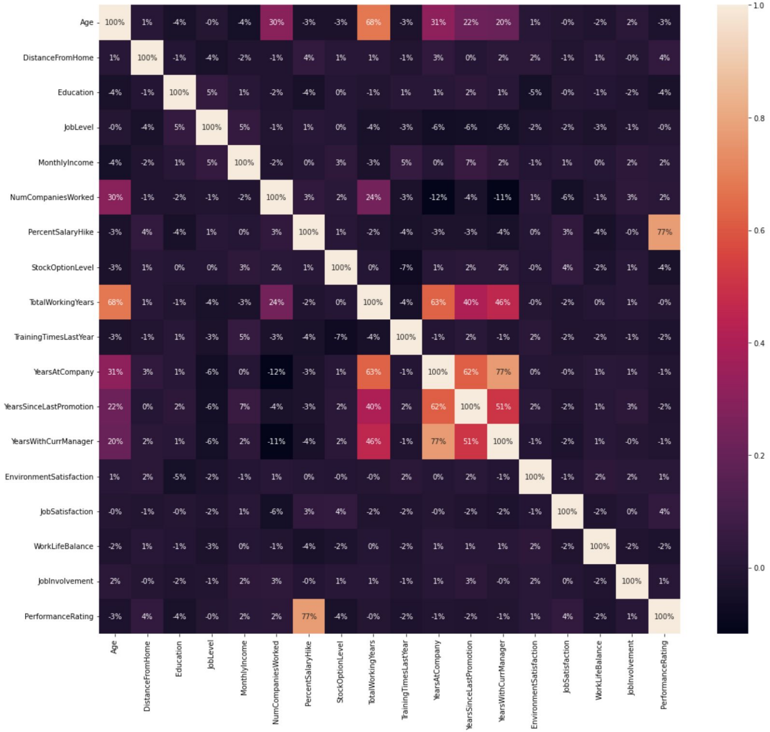
```
array([8], dtype=int64)
```

```
# Removing useless columns

# Over18: By default all the employees are above 18 years.
# EmployeeCount: Employee count does not contribute towards our study. As it does not tell naything about the data.
# Standard Hours are by default 8hr/day.
# Even the Employee ID is a redundant column.

df = df.drop('Over18', axis = 1)
df = df.drop('EmployeeCount', axis = 1)
df = df.drop('StandardHours', axis = 1)
df = df.drop('EmployeeID', axis = 1)
```

Data Cleaning

```
from sklearn.preprocessing import LabelEncoder

for column in df.columns:
  if df[column].dtype == np.number:
    continue
  df[column] = LabelEncoder().fit_transform(df[column])
```

Data Visualizations




```
plt.figure(figsize=(18,16))
sns.heatmap(df.corr(), annot = True, fmt = '.0%')
plt.show()
```

Data Modelling

```
# Splitting the dataset

X = df.iloc[:, 1: df.shape[1]].values
Y = df.iloc[:, 0].values
```

```
# Splitting the dataset for modelling
# 75% for training
# 25% for testing

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

Data Modelling

```python
# Importing all the packages we will need for implementing the Machine Learning models.

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
```

```python
# Defining the Machine learning model functions and giving parameters where necessary

# c: trust value. Keeping it low to give more weight to this complexity penalty.
logistic_reg = LogisticRegression(C = 0.1, random_state = 42, solver = 'liblinear')

decision_tree = DecisionTreeClassifier()
random_forest = RandomForestClassifier()
gaussian_nb = GaussianNB()
knn = KNeighborsClassifier(n_neighbors=3)
svm = svm.SVC(kernel='linear')
```

Data Modelling

```
# Importing sklearn.metrics package
from sklearn.metrics import accuracy_score

# Getting the accuracy for multiple models.
for a,b in zip([logistic_reg, decision_tree, random_forest, gaussian_nb, knn, svm],["Logistic Regression","Decision Tree",
    a.fit(X_train,Y_train)
    prediction = a.predict(X_train)
    Y_pred = a.predict(X_test)
    score_train = accuracy_score(Y_train,prediction)
    score_test = accuracy_score(Y_test,Y_pred)
    training_accu = "[%s] training data accuracy is : %f" % (b,score_train)
    test_accu = "[%s] test data accuracy is : %f" % (b,score_test)
    print(training_accu)
    print(test_accu)

# Reference: scikit-learn: machine learning in Python
```

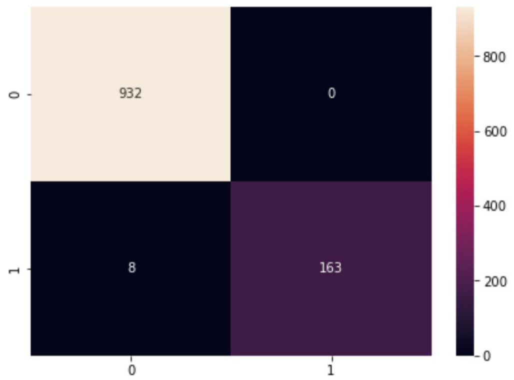
```
[Logistic Regression] training data accuracy is : 0.846386
[Logistic Regression] test data accuracy is : 0.853128
[Decision Tree] training data accuracy is : 1.000000
[Decision Tree] test data accuracy is : 0.984587
[Random Forest] training data accuracy is : 1.000000
[Random Forest] test data accuracy is : 0.993654
[Naive Bayes] training data accuracy is : 0.833989
[Naive Bayes] test data accuracy is : 0.835902
[KNN] training data accuracy is : 0.985788
[KNN] test data accuracy is : 0.906618
[SVM] training data accuracy is : 0.836710
[SVM] test data accuracy is : 0.844968
```

Model Evaluation of RF

```python
from sklearn.metrics import confusion_matrix

conf_matrix_RF = confusion_matrix(Y_test, forest.predict(X_test))

TN_RF = conf_matrix_RF[0][0]
TP_RF = conf_matrix_RF[1][1]
FN_RF = conf_matrix_RF[1][0]
FP_RF = conf_matrix_RF[0][1]

print(conf_matrix_RF)
print('Model Testing accuracy for Random Forest Algorithm = {}'.format((TP_RF + TN_RF) / (TP_RF + FP_RF + FN_RF + TN_RF)))
```

```
[[932   0]
 [  8 163]]
Model Testing accuracy for Random Forest Algorithm = 0.9927470534904805
```

```python
# Visulaizing the Confusion Matrix
plt.figure(figsize = (7,5))
sns.heatmap(conf_matrix_RF, annot=True, fmt='g')
plt.show()
```



Model Evaluation of RF

```
from sklearn.metrics import classification_report

random_forest_CR = random_forest.predict(X_test)
print(classification_report(Y_test, random_forest_CR))
```

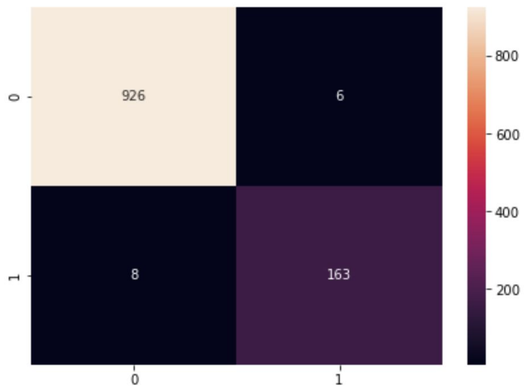
```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       932
           1       1.00      0.96      0.98       171

    accuracy                           0.99      1103
   macro avg       1.00      0.98      0.99      1103
weighted avg       0.99      0.99      0.99      1103
```

Model Evaluation of DT

```
conf_matrix_DT = confusion_matrix(Y_test, classifier.predict(X_test))

TN_DT = conf_matrix_DT[0][0]
TP_DT = conf_matrix_DT[1][1]
FN_DT = conf_matrix_DT[1][0]
FP_DT = conf_matrix_DT[0][1]

print(conf_matrix_DT)
print('Model Testing accuracy for Decision Tree Algorithm = {}'.format((TP_DT + TN_DT) / (TP_DT + FP_DT + FN_DT + TN_DT)))
```

```
[[926   6]
 [  8 163]]
Model Testing accuracy for Decision Tree Algorithm = 0.9873073436083409
```

```
# Visualizing the Confusion Matrix
plt.figure(figsize = (7,5))
sns.heatmap(conf_matrix_DT, annot=True, fmt='g')
plt.show()
```



Data Modelling of DT

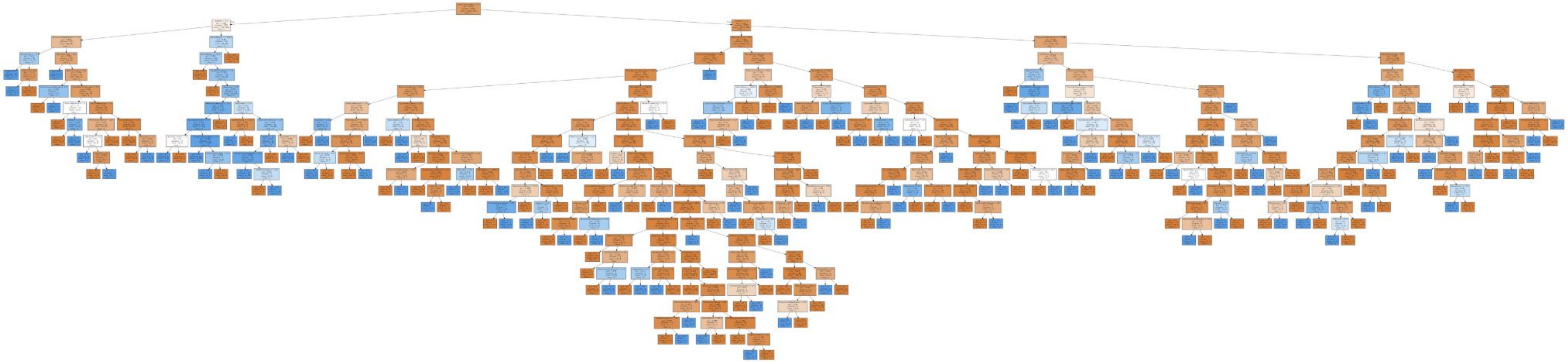
```
from sklearn.metrics import classification_report

decision_tree_CR = decision_tree.predict(X_test)
print(classification_report(Y_test, decision_tree_CR))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       932
           1       0.95      0.95      0.95       171

    accuracy                           0.98      1103
   macro avg       0.97      0.97      0.97      1103
weighted avg       0.98      0.98      0.98      1103
```

Decision Tree

```
import graphviz
from sklearn import tree

feature_names = df.columns[:24]
target_names = df['Attrition'].unique().tolist()


viz = tree.export_graphviz(classifier, out_file=None,
                                feature_names = feature_names,
                                class_names = str(target_names),
                                filled=True)

# Draw graph
graph = graphviz.Source(viz, format="png")
graph
```

Decision Tree



Findings

- The Random Forest algorithm performs better than the Decision Tree.
- Salary Hike and Performance shows a positive correlation thus it can be used as a remedial measure to reward good employees and make them stay.
- Environment satisfaction is negatively correlated to the level of education.

Thank you!