human-resources-analytics

November 20, 2024

Human Resources Analytics

Imported All the necessary libraries:

```
[ ]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import StandardScaler
     from sklearn.model_selection import train_test_split

     from sklearn.ensemble import RandomForestClassifier
     from sklearn.svm import SVC
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import accuracy_score, confusion_matrix,␣
      ↪classification_report
     from sklearn.model_selection import GridSearchCV
```

Read the Data set and store it in a pandas Data Frame:

```
[ ]: df = pd.read_csv("Human_Resuorces_Analytics.csv")
```

How many rows and columns in the data frame :

```
[ ]: df.shape
```

```
[ ]: (311, 36)
```

Explore the names of columns in the data frame:

```
[ ]: df.columns
```

```
[ ]: Index(['Employee_Name', 'EmpID', 'MarriedID', 'MaritalStatusID', 'GenderID',
            'EmpStatusID', 'DeptID', 'PerfScoreID', 'FromDiversityJobFairID',
            'Salary', 'Termd', 'PositionID', 'Position', 'State', 'Zip', 'DOB',
            'Sex', 'MaritalDesc', 'CitizenDesc', 'HispanicLatino', 'RaceDesc',
            'DateofHire', 'DateofTermination', 'TermReason', 'EmploymentStatus',
            'Department', 'ManagerName', 'ManagerID', 'RecruitmentSource',
            'PerformanceScore', 'EngagementSurvey', 'EmpSatisfaction',
```

```
'SpecialProjectsCount', 'LastPerformanceReview_Date', 'DaysLateLast30',
 'Absences'],
dtype='object')
```

General Information About the Columns of the Data Frame :

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 311 entries, 0 to 310
Data columns (total 36 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Employee_Name               311 non-null    object
 1   EmpID                       311 non-null    int64
 2   MarriedID                   311 non-null    int64
 3   MaritalStatusID             311 non-null    int64
 4   GenderID                    311 non-null    int64
 5   EmpStatusID                 311 non-null    int64
 6   DeptID                      311 non-null    int64
 7   PerfScoreID                 311 non-null    int64
 8   FromDiversityJobFairID      311 non-null    int64
 9   Salary                      311 non-null    int64
 10  Termd                       311 non-null    int64
 11  PositionID                  311 non-null    int64
 12  Position                    311 non-null    object
 13  State                       311 non-null    object
 14  Zip                         311 non-null    int64
 15  DOB                         311 non-null    object
 16  Sex                         311 non-null    object
 17  MaritalDesc                 311 non-null    object
 18  CitizenDesc                 311 non-null    object
 19  HispanicLatino              311 non-null    object
 20  RaceDesc                    311 non-null    object
 21  DateofHire                  311 non-null    object
 22  DateofTermination           104 non-null    object
 23  TermReason                  311 non-null    object
 24  EmploymentStatus            311 non-null    object
 25  Department                  311 non-null    object
 26  ManagerName                 311 non-null    object
 27  ManagerID                   303 non-null    float64
 28  RecruitmentSource           311 non-null    object
 29  PerformanceScore            311 non-null    object
 30  EngagementSurvey            311 non-null    float64
 31  EmpSatisfaction             311 non-null    int64
 32  SpecialProjectsCount        311 non-null    int64
 33  LastPerformanceReview_Date  311 non-null    object
 34  DaysLateLast30              311 non-null    int64
```

```
 35  Absences                    311 non-null    int64
dtypes: float64(2), int64(16), object(18)
memory usage: 87.6+ KB
```

Display the first 10 rows of data frame:

```
[ ]: df.head(10)
```

```
[ ]:              Employee_Name  EmpID  MarriedID  MaritalStatusID  GenderID  \
0      Adinolfi, Wilson  K  10026          0                0         1
1  Ait Sidi, Karthikeyan     10084          1                1         1
2         Akinkuolie, Sarah  10196          1                1         0
3              Alagbe,Trina  10088          1                1         0
4          Anderson, Carol   10069          0                2         0
5         Anderson, Linda    10002          0                0         0
6           Andreola, Colby  10194          0                0         0
7               Athwal, Sam  10062          0                4         1
8          Bachiochi, Linda  10114          0                0         0
9        Bacong, Alejandro   10250          0                2         1

   EmpStatusID  DeptID  PerfScoreID  FromDiversityJobFairID  Salary  …  \
0            1       5            4                       0   62506  …
1            5       3            3                       0  104437  …
2            5       5            3                       0   64955  …
3            1       5            3                       0   64991  …
4            5       5            3                       0   50825  …
5            1       5            4                       0   57568  …
6            1       4            3                       0   95660  …
7            1       5            3                       0   59365  …
8            3       5            3                       1   47837  …
9            1       3            3                       0   50178  …

       ManagerName  ManagerID   RecruitmentSource PerformanceScore  \
0   Michael Albert       22.0            LinkedIn          Exceeds
1       Simon Roup        4.0              Indeed      Fully Meets
2   Kissy Sullivan       20.0            LinkedIn      Fully Meets
3     Elijiah Gray       16.0              Indeed      Fully Meets
4   Webster Butler       39.0       Google Search      Fully Meets
5         Amy Dunn       11.0            LinkedIn          Exceeds
6  Alex Sweetwater       10.0            LinkedIn      Fully Meets
7    Ketsia Liebig       19.0   Employee Referral      Fully Meets
8   Brannon Miller       12.0  Diversity Job Fair      Fully Meets
9     Peter Monroe        7.0              Indeed      Fully Meets

   EngagementSurvey EmpSatisfaction SpecialProjectsCount  \
0              4.60               5                    0
1              4.96               3                    6
2              3.02               3                    0
```

```
3              4.84              5                    0
4              5.00              4                    0
5              5.00              5                    0
6              3.04              3                    4
7              5.00              4                    0
8              4.46              3                    0
9              5.00              5                    6

  LastPerformanceReview_Date DaysLateLast30 Absences
0                  1/17/2019              0        1
1                  2/24/2016              0       17
2                  5/15/2012              0        3
3                   1/3/2019              0       15
4                   2/1/2016              0        2
5                   1/7/2019              0       15
6                   1/2/2019              0       19
7                  2/25/2019              0       19
8                  1/25/2019              0        4
9                  2/18/2019              0       16

[10 rows x 36 columns]
```

What are the data types of the columns in the data frame:

[ ]:
```
df.dtypes
```

[ ]:
```
Employee_Name                 object
EmpID                          int64
MarriedID                      int64
MaritalStatusID                int64
GenderID                       int64
EmpStatusID                    int64
DeptID                         int64
PerfScoreID                    int64
FromDiversityJobFairID         int64
Salary                         int64
Termd                          int64
PositionID                     int64
Position                      object
State                         object
Zip                            int64
DOB                           object
Sex                           object
MaritalDesc                   object
CitizenDesc                   object
HispanicLatino                object
RaceDesc                      object
```

```
DateofHire                     object
DateofTermination              object
TermReason                     object
EmploymentStatus               object
Department                     object
ManagerName                    object
ManagerID                     float64
RecruitmentSource              object
PerformanceScore               object
EngagementSurvey              float64
EmpSatisfaction                 int64
SpecialProjectsCount            int64
LastPerformanceReview_Date     object
DaysLateLast30                  int64
Absences                        int64
dtype: object
```

Check the missing values in the columns of the data frame:

[ ]:
```
df.isnull().any()
```

[ ]:
```
Employee_Name                 False
EmpID                         False
MarriedID                     False
MaritalStatusID               False
GenderID                      False
EmpStatusID                   False
DeptID                        False
PerfScoreID                   False
FromDiversityJobFairID        False
Salary                        False
Termd                         False
PositionID                    False
Position                      False
State                         False
Zip                           False
DOB                           False
Sex                           False
MaritalDesc                   False
CitizenDesc                   False
HispanicLatino                False
RaceDesc                      False
DateofHire                    False
DateofTermination              True
TermReason                    False
EmploymentStatus              False
Department                    False
```

```
ManagerName                    False
ManagerID                       True
RecruitmentSource              False
PerformanceScore               False
EngagementSurvey               False
EmpSatisfaction                False
SpecialProjectsCount           False
LastPerformanceReview_Date     False
DaysLateLast30                 False
Absences                       False
dtype: bool
```

Count the missing values in the columns of the data frame:

[ ]:
```
df.isnull().sum()
```

[ ]:
```
Employee_Name                    0
EmpID                            0
MarriedID                        0
MaritalStatusID                  0
GenderID                         0
EmpStatusID                      0
DeptID                           0
PerfScoreID                      0
FromDiversityJobFairID           0
Salary                           0
Termd                            0
PositionID                       0
Position                         0
State                            0
Zip                              0
DOB                              0
Sex                              0
MaritalDesc                      0
CitizenDesc                      0
HispanicLatino                   0
RaceDesc                         0
DateofHire                       0
DateofTermination              207
TermReason                       0
EmploymentStatus                 0
Department                       0
ManagerName                      0
ManagerID                        8
RecruitmentSource                0
PerformanceScore                 0
EngagementSurvey                 0
```

```
EmpSatisfaction               0
SpecialProjectsCount          0
LastPerformanceReview_Date    0
DaysLateLast30                0
Absences                      0
dtype: int64
```

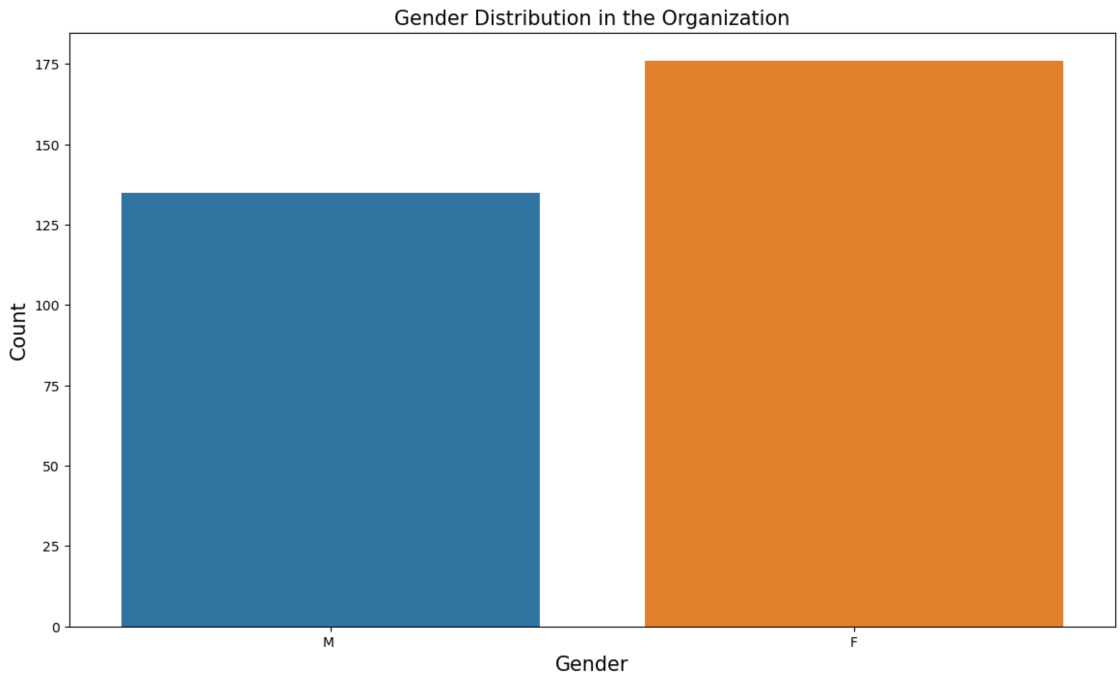
1 Section A (Data Exploration and Pre-processing):

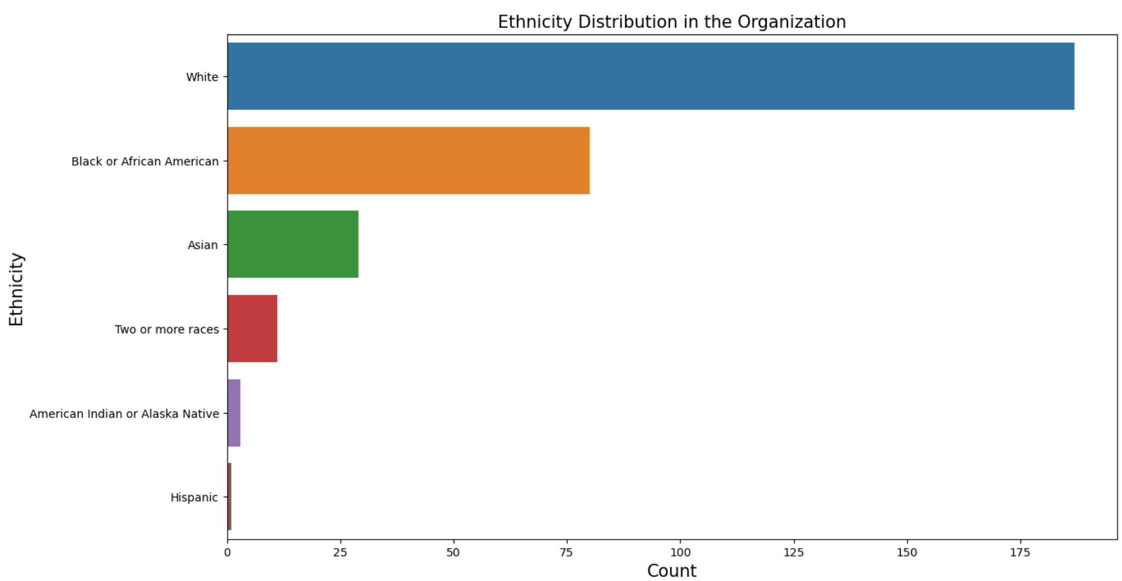
1.0.1 Explore the data using tables, visualizations, and other relevant methods

1. Overall Diversity Profile of the Organization:

To understand the diversity profile of the organization, we can analyze the distribution of employees based on their gender and ethnicity. We'll create a bar plot to visualize this

[ ]:
```
# Count the number of employees in each gender category
gender_counts = df['Sex'].value_counts()

# Count the number of employees based on their ethnicity
ethnicity_counts = df['RaceDesc'].value_counts()

# Plot the gender distribution
plt.figure(figsize=(14, 8))
sns.countplot(data=df, x='Sex')
plt.title('Gender Distribution in the Organization', fontsize = 15)
plt.xlabel('Gender', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()

# Plot the ethnicity distribution
plt.figure(figsize=(14, 8))
sns.countplot(data=df, y='RaceDesc', order=ethnicity_counts.index)
plt.title('Ethnicity Distribution in the Organization', fontsize = 15)
plt.xlabel('Count', fontsize = 15)
plt.ylabel('Ethnicity', fontsize = 15)
plt.show()
```



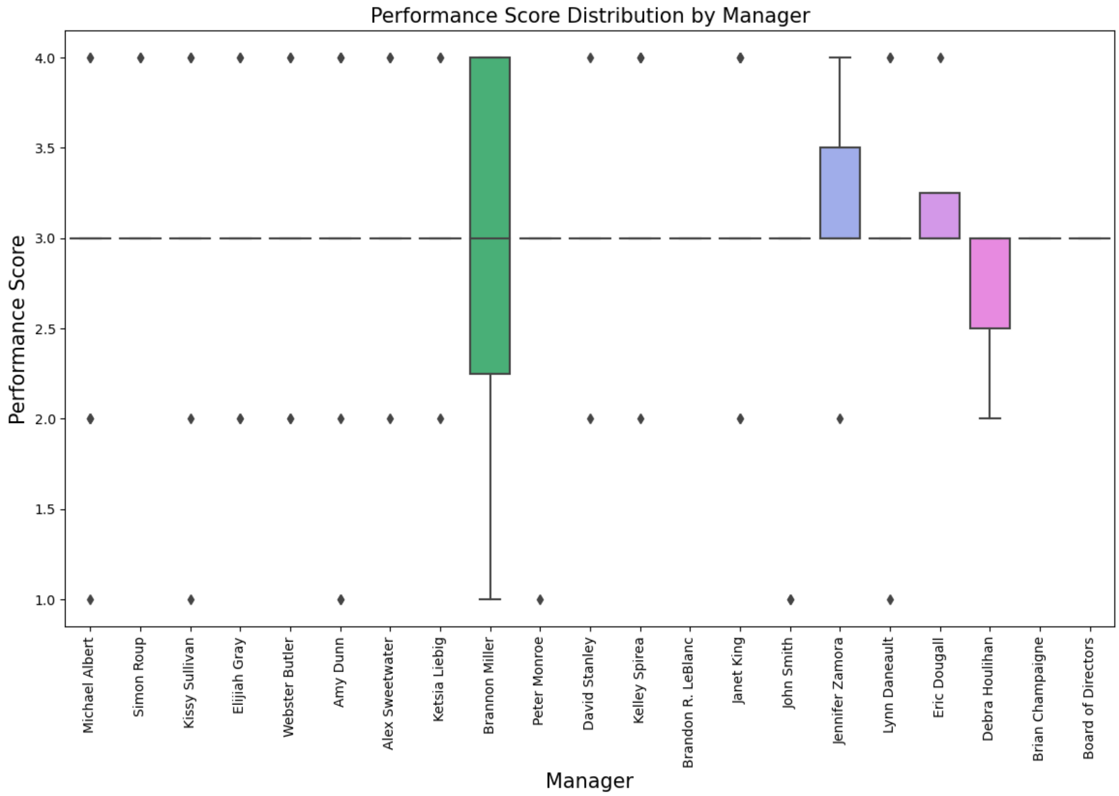


These plots will provide insights into the gender and ethnicity diversity within the organization.

2. Relationship between Manager and Performance Score:

We can investigate whether there is any relationship between the employee's manager and their performance score. To visualize this, we'll use a box plot to compare the performance scores across different managers.

```
[ ]: plt.figure(figsize=(14, 8))
sns.boxplot(data=df, x='ManagerName', y='PerfScoreID')
plt.xticks(rotation=90)
plt.title('Performance Score Distribution by Manager', fontsize = 15)
plt.xlabel('Manager', fontsize = 15)
plt.ylabel('Performance Score', fontsize = 15)
plt.show()
```



This box plot will help us identify any variations in performance scores based on the employee's manager.

3. Areas of Pay Inequity:

To identify areas of pay inequity within the company, we can compare the salaries across different departments. A violin plot can effectively showcase the distribution of salaries in each department.

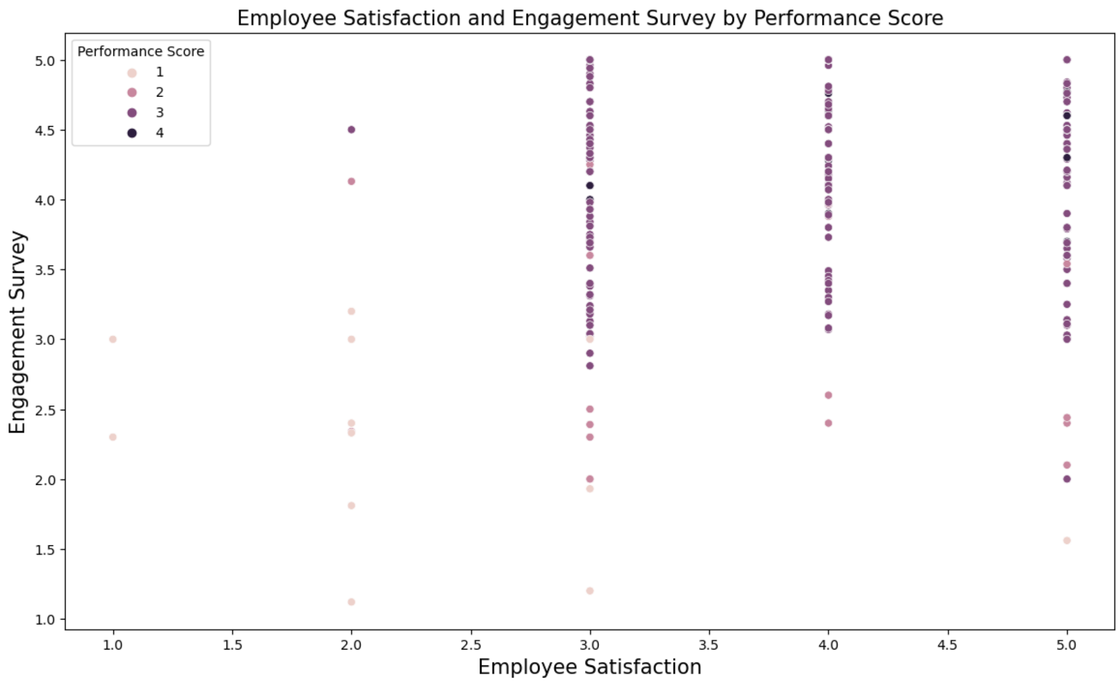
```
[ ]: plt.figure(figsize=(14, 8))
sns.violinplot(data=df, x='Department', y='Salary')
plt.xticks(rotation=90)
plt.title('Salary Distribution by Department', fontsize = 15)
plt.xlabel('Department', fontsize = 15)
plt.ylabel('Salary', fontsize = 15)
plt.show()
```

This visualization will highlight any variations in salary distribution across different departments, indicating potential areas of pay inequity.

4. Employee Satisfaction and Engagement Survey by Performance Score:

Let's create a scatter plot to visualize the relationship between employee satisfaction, engagement survey results, and performance scores. This will help us understand if there are any patterns or correlations among these variables.

[ ]:
```
plt.figure(figsize=(14, 8))
sns.scatterplot(data=df, x='EmpSatisfaction', y='EngagementSurvey',␣
 ↪hue='PerfScoreID')
plt.title('Employee Satisfaction and Engagement Survey by Performance Score',␣
 ↪fontsize = 15)
plt.xlabel('Employee Satisfaction', fontsize = 15)
plt.ylabel('Engagement Survey', fontsize = 15)
plt.legend(title='Performance Score')
plt.show()
```



This scatter plot will show how employee satisfaction and engagement survey results vary across different performance scores.

5. Salary Distribution by Department and Performance Score:

Let's visualize the distribution of salaries across different departments, considering the performance scores of employees as well. We'll use a box plot with color encoding to represent the performance scores.

[ ]:
```
plt.figure(figsize=(14, 8))
sns.boxplot(data=df, x='Department', y='Salary', hue='PerfScoreID')
plt.xticks(rotation=90)
plt.title('Salary Distribution by Department and Performance Score', fontsize =␣
    ↪15)
plt.xlabel('Department', fontsize = 15)
plt.ylabel('Salary', fontsize = 15)
plt.legend(title='Performance Score', fontsize = 15)
plt.show()
```

In this plot, each box represents the salary distribution for a specific department, and the color of the boxes represents different performance scores. This allows us to simultaneously explore the salary distributions and performance scores across departments.

1) Overall Diversity Profile of the Organization:

- The gender distribution graph provides insights into the gender diversity within the organization.
- The ethnicity distribution graph offers an understanding of the ethnic diversity within the organization.
- These graphs can help us assess the overall diversity profile of the organization.

2) Relationship between Who a Person Works for and Their Performance Score:

- The box plot graph comparing performance scores across different managers provides insights into the potential relationship between an employee's manager and their performance score.
- This graph allows us to explore whether there is any correlation between the employee's manager and their performance score.

3) Areas of the Company Where Pay Is Not Equitable:

- The violin plot graph displaying the salary distribution by department can help identify potential areas where pay may not be equitable. This graph enables us to visually assess variations in salary distribution across different departments, indicating areas where pay inequity might exist.

1.0.2 Apply different methods of pre-processing

Handling the missing values in ManagerID column:

```
[ ]: df['ManagerID'] = df['ManagerID'].fillna(df['ManagerID'].median())
```

```
[ ]: df['DateofTermination'] = df['DateofTermination'].
      ↪fillna(df['DateofTermination'].mode()[0])
```

Convert date columns to datetime data type:

```
[ ]: df['DOB'] = pd.to_datetime(df['DOB'])
     df['DateofHire'] = pd.to_datetime(df['DateofHire'])
     df['DateofTermination'] = pd.to_datetime(df['DateofTermination'])
     df['LastPerformanceReview_Date'] = pd.
      ↪to_datetime(df['LastPerformanceReview_Date'])
```

Replace 'no' and 'yes' with 'No' and 'Yes' in HispanicLatino column:

```
[ ]: df['HispanicLatino'] = df['HispanicLatino'].str.replace('no', 'No').
      ↪replace('yes','Yes')
```

Create some new columns: AgeAtHire, EmploymentDuration , TimeSinceLastReview and TimeUntilTermination from date columns in the data frame:

```
[ ]: # Calculate age at hire
     df['AgeAtHire'] = (df['DateofHire'] - df['DOB']).dt.days // 365

     # Calculate employment duration
     df['EmploymentDuration'] = (df['DateofTermination'].fillna(pd.Timestamp.now())␣
      ↪- df['DateofHire']).dt.days

     # Calculate time since last performance review
     df['TimeSinceLastReview'] = (pd.Timestamp.now() -␣
      ↪df['LastPerformanceReview_Date']).dt.days

     # Calculate the time difference in days
     df['TimeUntilTermination'] = abs((df['DateofTermination'].fillna(pd.Timestamp.
      ↪now()) - df['LastPerformanceReview_Date']).dt.days)

     last_performance_dates = df['LastPerformanceReview_Date']

     # Drop the original date columns
     df.drop(['DOB', 'DateofHire', 'DateofTermination',␣
      ↪'LastPerformanceReview_Date'], axis=1, inplace=True)
```

Drop some unnecessary columns from data frame:

```
[ ]: df.drop(['Employee_Name','Position','State','TermReason','EmpStatusID',␣
     ↪'Department','MaritalDesc','EmpID', 'Zip',  'ManagerName'], axis = 1,␣
     ↪inplace = True)
```

Convert the values of categorical columns to numeric columns:

```
[ ]: df['Sex'] = df['Sex'].astype('category').cat.codes
     df['CitizenDesc'] = df['CitizenDesc'].astype('category').cat.codes
     df['HispanicLatino'] = df['HispanicLatino'].astype('category').cat.codes
     df['RaceDesc'] = df['RaceDesc'].astype('category').cat.codes
     df['RecruitmentSource'] = df['RecruitmentSource'].astype('category').cat.codes
     df['PerformanceScore'] = df['PerformanceScore'].astype('category').cat.codes
     #df['EmploymentStatus'] = df['EmploymentStatus'].astype('category').cat.codes
```

Display the updated dataset with the new features

```
[ ]: df.head(10)
```

```
[ ]:    MarriedID  MaritalStatusID  GenderID  DeptID  PerfScoreID  \
     0          0                0         1       5            4
     1          1                1         1       3            3
     2          1                1         0       5            3
     3          1                1         0       5            3
     4          0                2         0       5            3
     5          0                0         0       5            4
     6          0                0         0       4            3
     7          0                4         1       5            3
     8          0                0         0       5            3
     9          0                2         1       3            3

        FromDiversityJobFairID  Salary  Termd  PositionID  Sex  …  \
     0                       0   62506      0          19    1  …
     1                       0  104437      1          27    1  …
     2                       0   64955      1          20    0  …
     3                       0   64991      0          19    0  …
     4                       0   50825      1          19    0  …
     5                       0   57568      0          19    0  …
     6                       0   95660      0          24    0  …
     7                       0   59365      0          19    1  …
     8                       1   47837      0          19    0  …
     9                       0   50178      0          14    1  …

        PerformanceScore  EngagementSurvey  EmpSatisfaction SpecialProjectsCount  \
     0                 0              4.60                5                    0
     1                 1              4.96                3                    6
     2                 1              3.02                3                    0
     3                 1              4.84                5                    0
     4                 1              5.00                4                    0
```

```
5                0        5.00                5                     0
6                1        3.04                3                     4
7                1        5.00                4                     0
8                1        4.46                3                     0
9                1        5.00                5                     6

   DaysLateLast30  Absences  AgeAtHire  EmploymentDuration  \
0               0         1         28                1583
1               0        17         39                 444
2               0         3         22                 447
3               0        15         19                2858
4               0         2         21                1884
5               0        15         34                1395
6               0        19         35                 359
7               0        19         30                 765
8               0         4        -61                2312
9               0        16         27                 303

   TimeSinceLastReview  TimeUntilTermination
0                 1620                  1170
1                 2678                   113
2                 4058                   132
3                 1634                  1156
4                 2701                   218
5                 1630                  1160
6                 1635                  1155
7                 1581                  1209
8                 1612                  1178
9                 1588                  1202

[10 rows x 26 columns]
```

2 Section B (Dimensionality Reduction):

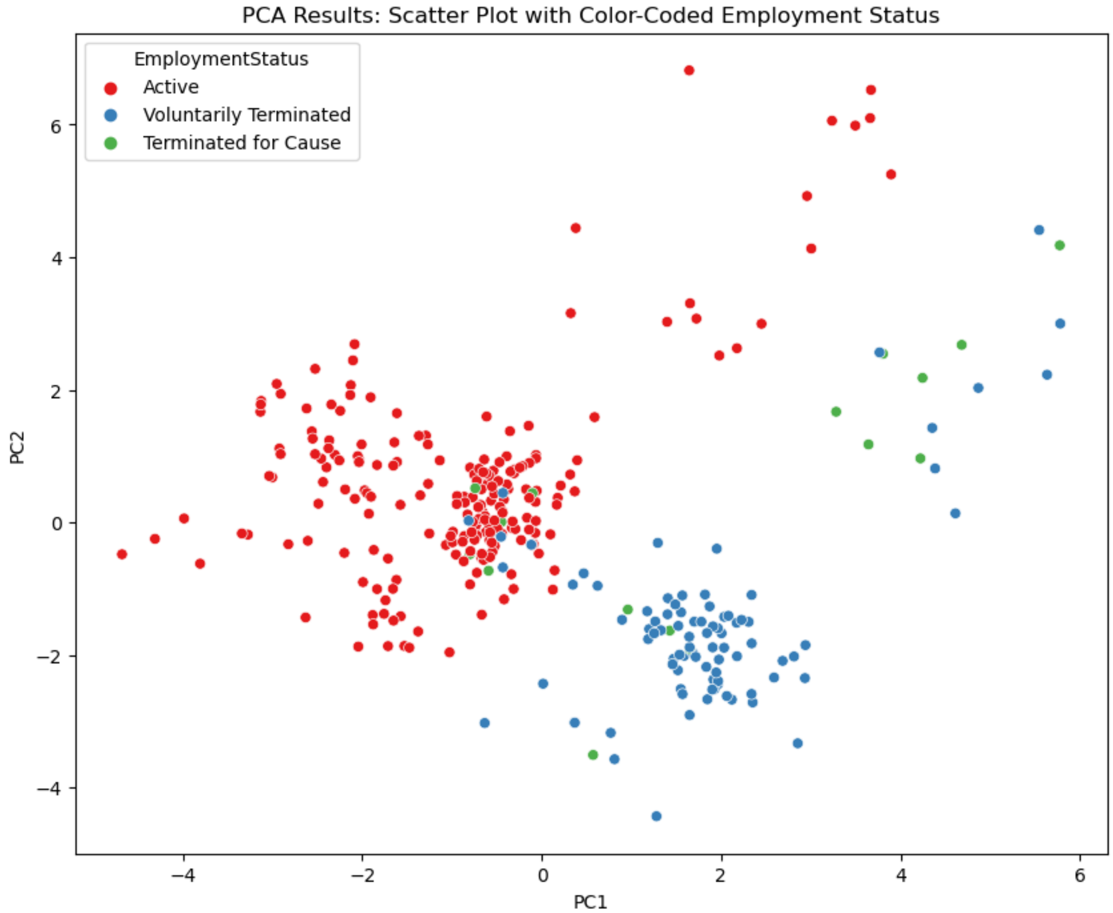
1. Apply PCA algorithm on the data.:

[ ]:
```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

pca = PCA()
pca_result = pca.fit_transform(StandardScaler().fit_transform(df.
 ↪drop('EmploymentStatus', axis=1)))
```

2. Create a scatter plot with color-coded observations based on employment status:

['Active', 'Voluntarily Terminated', 'Terminated for Cause'] = [0, 1, 2]

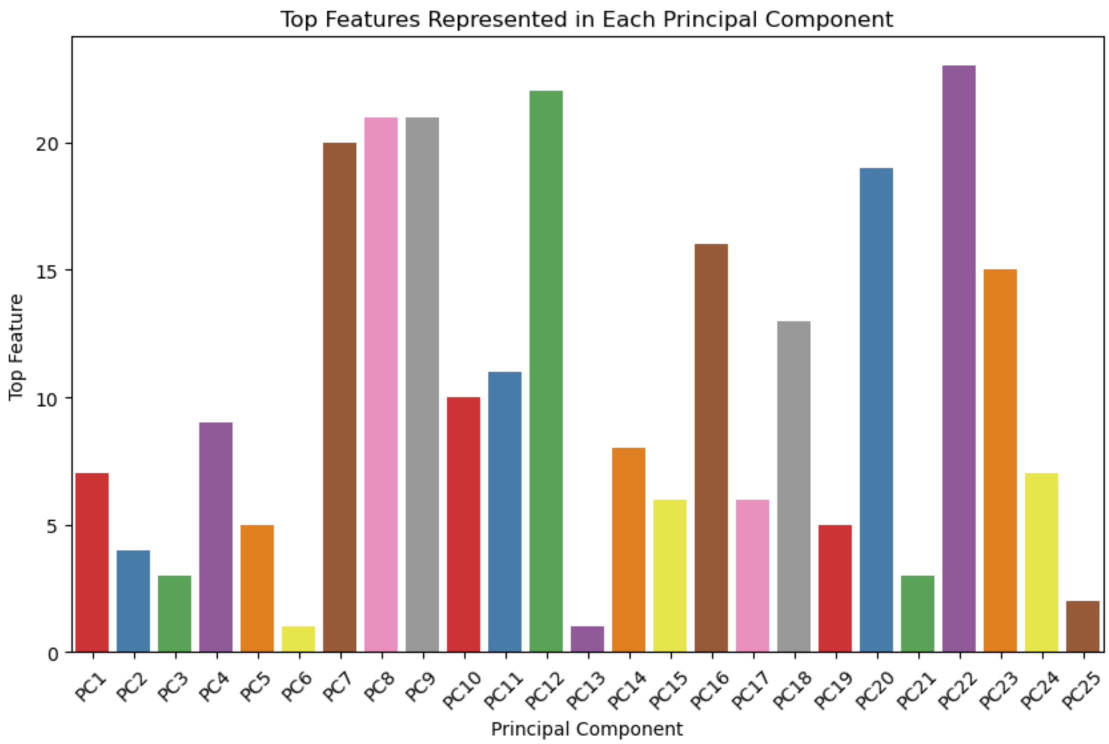
```
[ ]: pca_df = pd.DataFrame(data=pca_result, columns=[f'PC{i+1}' for i in␣
      ↪range(pca_result.shape[1])])
     pca_df['EmploymentStatus'] = df['EmploymentStatus']

     plt.figure(figsize=(10, 8))
     sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='EmploymentStatus',␣
      ↪palette='Set1')
     plt.title('PCA Results: Scatter Plot with Color-Coded Employment Status')
     plt.show()
```



3. Identify the features most strongly represented in each component:

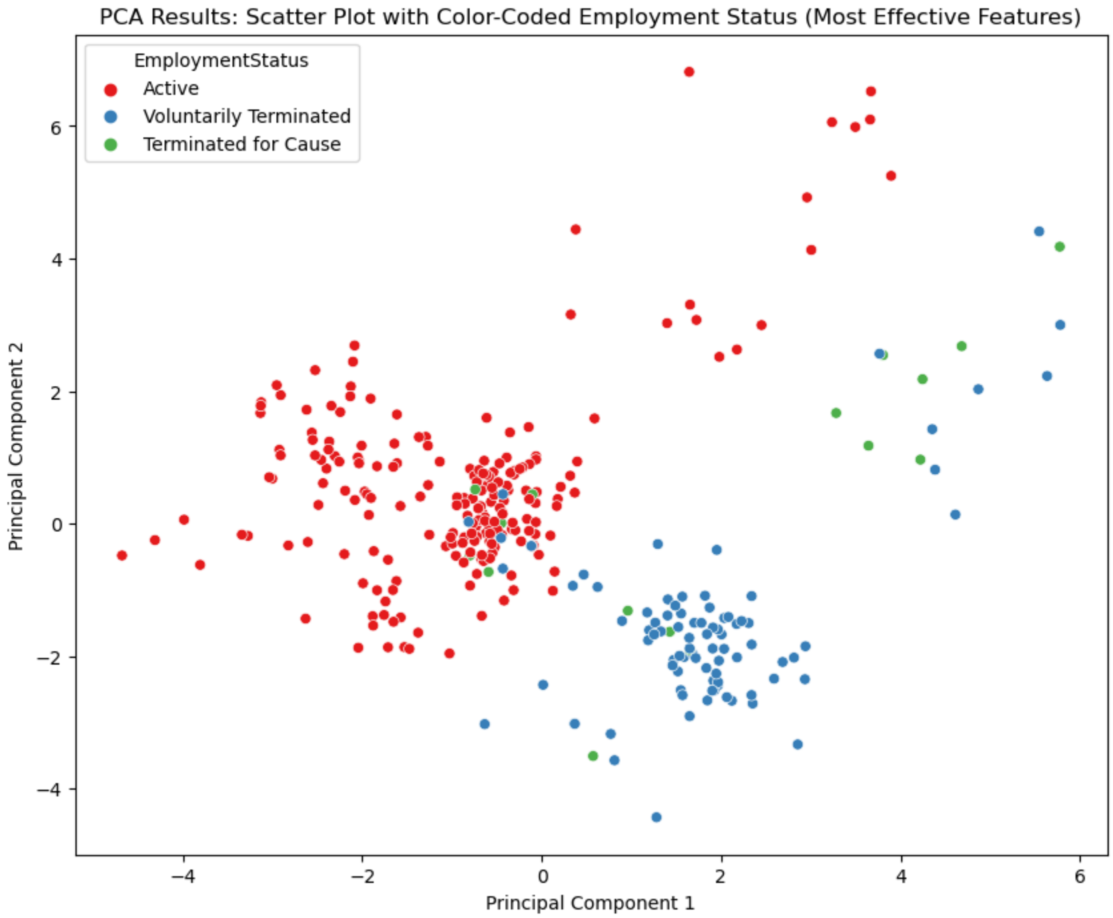
```
[ ]: component_features = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i␣
      ↪in range(pca_result.shape[1])])
     abs_component_features = component_features.abs()
     top_features = abs_component_features.idxmax(axis=0)
```

[ ]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=top_features.reset_index(), x='index', y=0, palette='Set1')
plt.title('Top Features Represented in Each Principal Component')
plt.xlabel('Principal Component')
plt.ylabel('Top Feature')
plt.xticks(rotation=45)
plt.show()
```



The most effective features for separating employees by their employment status:

[ ]:
```
# Visualize resulting clusters using top-ranked principal components
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue=df['EmploymentStatus'],␣
 ↪palette='Set1')
plt.title('PCA Results: Scatter Plot with Color-Coded Employment Status (Most␣
 ↪Effective Features)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



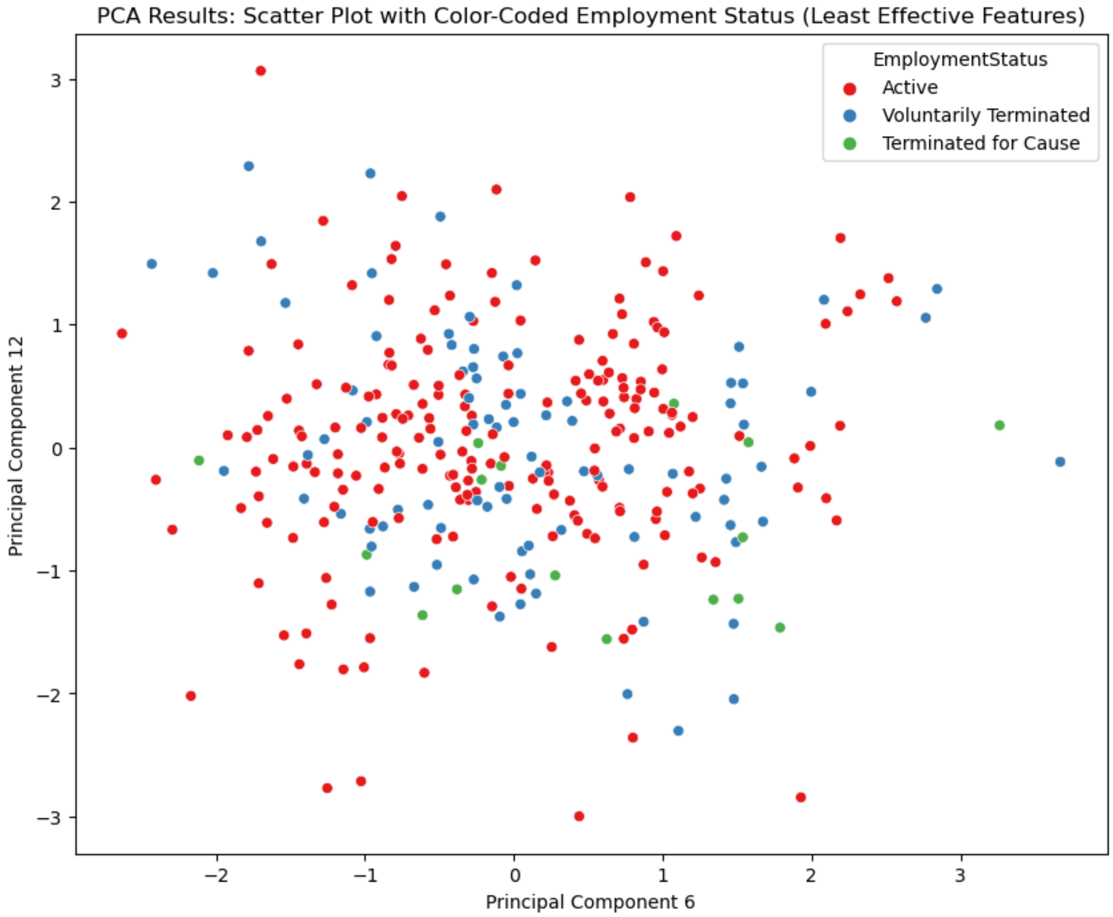
This code will plot a scatter plot where each point represents an employee, and the color of the point will indicate their employment status. The principal components PC1 and PC2, which are among the most effective features in separating the employees, will be used as the x and y axes, respectively.

By visualizing the resulting clusters, you can observe how the most effective features contribute to the differentiation of employees based on their employment status. These features have a strong influence on the separation, leading to more distinct clusters for different employment statuses.

The least effective features for separating employees by their employment status:

[ ]:
```
# Visualize resulting clusters using lower-ranked principal components
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pca_df, x='PC6', y='PC12', hue=df['EmploymentStatus'],␣
  ↪palette='Set1')
plt.title('PCA Results: Scatter Plot with Color-Coded Employment Status (Least␣
  ↪Effective Features)')
plt.xlabel('Principal Component 6')
plt.ylabel('Principal Component 12')
```

```
plt.show()
```



This code will plot a scatter plot where each point represents an employee, and the color of the point will indicate their employment status. The principal components PC6 and PC12, which are among the least effective features in separating the employees, will be used as the x and y axes, respectively.

By visualizing the resulting clusters, you can observe how the least effective features contribute to the differentiation of employees based on employment status. However, since these features have less impact on the separation, the clusters might overlap more, making it harder to distinguish between different employment statuses.

Assign numeric codes to EmploymentStatus variable:

[ ]:
```
df['EmploymentStatus'] = df['EmploymentStatus'].astype('category').cat.codes
```

Find the most effective feature:

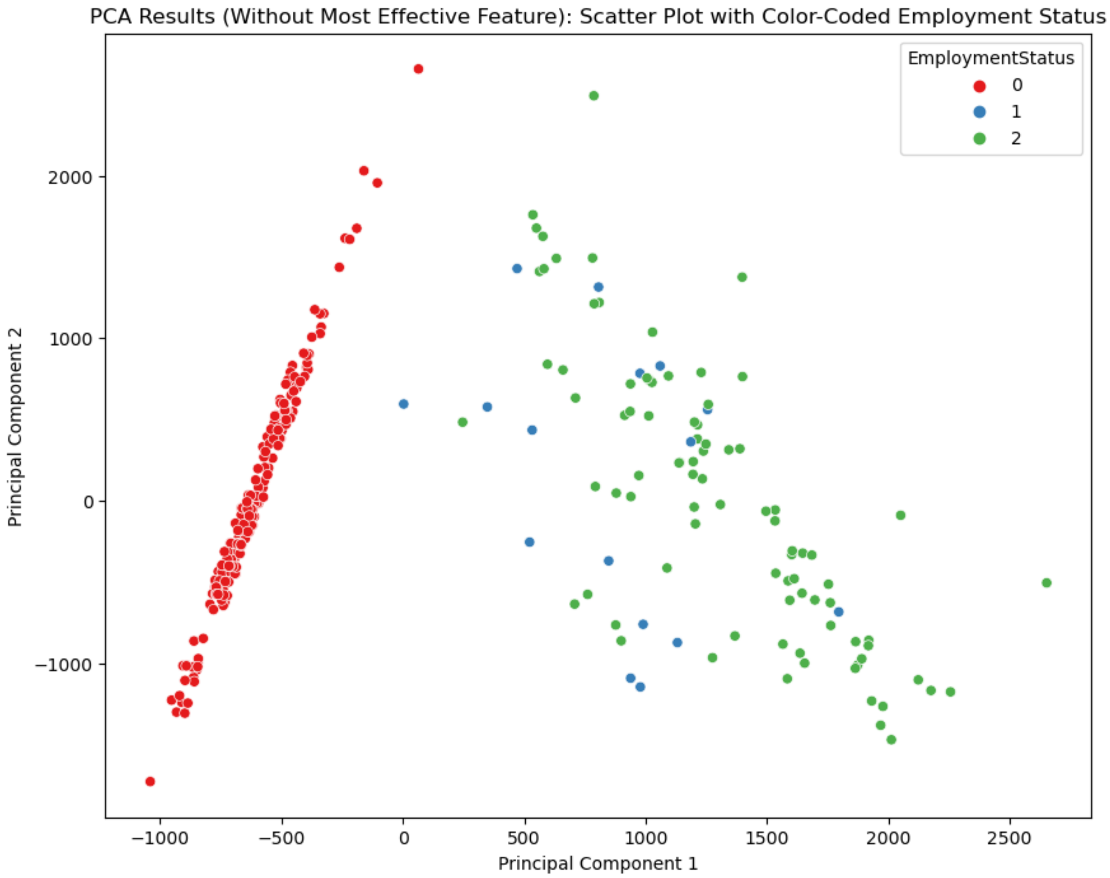
[ ]:
```
# Perform PCA on the dataset
pca = PCA(n_components=2)
```

```
pca_result = pca.fit_transform(df)

# Calculate feature importance in each principal component
component_features = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i␣
↪in range(pca_result.shape[1])])
feature_importance = component_features.abs().mean(axis=1)

# Identify the feature with the highest importance
most_effective_feature = feature_importance.idxmax()
most_effective_feature_ = df.columns[7]
print(most_effective_feature_)
```

```
Termd
```

PCA Results (Without Most Effective Feature): Scatter Plot with Color-Coded Employment Status:

[ ]:
```
# Exclude the most effective feature from the dataset
modified_df = df.drop(columns=df.columns[most_effective_feature])

# Apply PCA on the modified dataset
pca_modified = PCA(n_components=2)
pca_result_modified = pca_modified.fit_transform(modified_df)

# Visualize the resulting clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pd.DataFrame(pca_result_modified, columns=['PC1', 'PC2']),␣
↪x='PC1', y='PC2', hue=df['EmploymentStatus'], palette='Set1')
plt.title('PCA Results (Without Most Effective Feature): Scatter Plot with␣
↪Color-Coded Employment Status')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Explore the features most effective in separation (excluding the most␣
↪effective feature)
remaining_features = modified_df.columns
print("Remaining Features:", remaining_features)
```

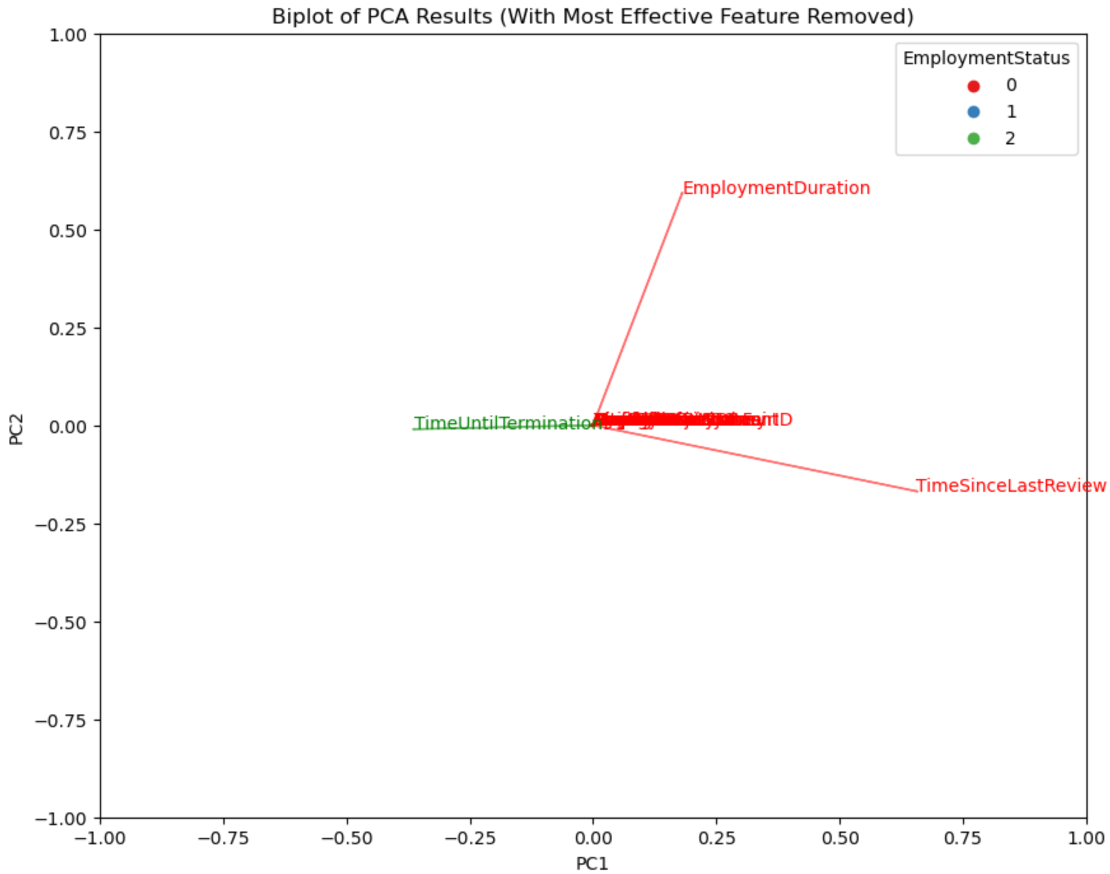



```
Remaining Features: Index(['MarriedID', 'MaritalStatusID', 'GenderID', 'DeptID',
'PerfScoreID',
       'FromDiversityJobFairID', 'Termd', 'PositionID', 'Sex', 'CitizenDesc',
       'HispanicLatino', 'RaceDesc', 'EmploymentStatus', 'ManagerID',
       'RecruitmentSource', 'PerformanceScore', 'EngagementSurvey',
       'EmpSatisfaction', 'SpecialProjectsCount', 'DaysLateLast30', 'Absences',
       'AgeAtHire', 'EmploymentDuration', 'TimeSinceLastReview',
       'TimeUntilTermination'],
      dtype='object')
```

4. Biplot of PCA Results (With Most Effective Feature Removed):

[ ]:
```
# Apply PCA on the modified dataset
pca_modified = PCA(n_components=2)
pca_result_modified = pca_modified.fit_transform(modified_df)

# Create a DataFrame for the modified PCA results
pca_df_modified = pd.DataFrame(data=pca_result_modified, columns=['PC1', 'PC2'])

# Get the explained variance ratio for PC1 and PC2
```

```
explained_variance_ratio_modified = pca_modified.explained_variance_ratio_

# Create a biplot (with the most effective feature)
plt.figure(figsize=(10, 8))

# Plot the data points
sns.scatterplot(data=pca_df_modified, x='PC1', y='PC2', hue=df['EmploymentStatus'], palette='Set1', alpha=0.6)

# Plot the variable vectors
for i, feature in enumerate(modified_df.columns[:-1]):
    arrow_length = np.sqrt(explained_variance_ratio_modified[0]) * pca_modified.components_[0, i]
    arrow_width = np.sqrt(explained_variance_ratio_modified[1]) * pca_modified.components_[1, i]
    plt.arrow(0, 0, arrow_length, arrow_width, color='r', alpha=0.5)
    plt.text(arrow_length, arrow_width, feature, color='r')

# Plot the most effective feature vector
arrow_length = np.sqrt(explained_variance_ratio_modified[0]) * pca_modified.components_[0, -1]
arrow_width = np.sqrt(explained_variance_ratio_modified[1]) * pca_modified.components_[1, -1]
plt.arrow(0, 0, arrow_length, arrow_width, color='g', alpha=0.5)
plt.text(arrow_length, arrow_width, modified_df.columns[-1], color='g')

# Set the limits of the plot
plt.xlim(-1, 1)
plt.ylim(-1, 1)

# Set labels and title
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Biplot of PCA Results (With Most Effective Feature Removed)')

# Show the plot
plt.show()
```



The biplot will provide insights into the relationships between the data points and variables. Here are some aspects to consider when interpreting the biplot:

Proximity of data points: Data points that are close to each other on the biplot are similar in terms of their feature values. Points that are far apart are dissimilar. You can observe the clusters of different employment status groups and identify patterns in their proximity.

Angle and direction of vectors: The vectors represent the variables/features in the biplot. The angle between two vectors indicates the correlation between the corresponding features. Vectors pointing in the same direction or opposite directions indicate positive or negative correlations, respectively.

Variables' contribution: The length of the vectors represents the contribution of the variables to the principal components. Longer vectors indicate higher importance in explaining the data's variance. By examining the lengths and orientations of the vectors, you can identify the variables that have the most significant influence on PC1 and PC2.

By considering these aspects and analyzing the biplot, you can gain insights into the relationships between data points and variables, identify patterns, and understand the variables' contributions to the separation of different employment status groups.

5. Bonus:

[ ]:
```
# Perform PCA on the dataset
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df)

# Calculate the coordinates of each data point in the PCA space
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])

# Calculate the Euclidean distance of each data point from the centroid in the␣
↪PCA space
centroid = pca_df.mean()
distances = np.linalg.norm(pca_df - centroid, axis=1)

# Define a threshold value to determine outliers
threshold = np.percentile(distances, 95)  # Adjust the percentile as needed

# Identify the outliers
outliers = df[distances > threshold]

# Print the list of outliers
print(outliers)

# Explain the findings
print("Number of outliers:", len(outliers))
print("Threshold:", threshold)
```

```
     MarriedID  MaritalStatusID  GenderID  DeptID  PerfScoreID  \
18           0                0         0       3            3
42           1                1         1       3            3
55           0                0         1       5            4
76           0                0         1       3            4
96           0                0         1       3            4
108          0                0         0       3            3
131          1                1         0       6            3
150          1                1         0       2            3
190          1                1         1       3            2
212          0                0         1       4            4
239          0                0         0       3            3
240          0                0         0       3            3
243          0                0         1       3            3
244          0                2         1       3            3
292          1                1         1       3            3
308          0                0         0       3            4

     FromDiversityJobFairID  Salary  Termd  PositionID  Sex  …  \
18                        0  110000      1           8    0  …
42                        0  110929      0           5    1  …
55                        0  170500      0          10    1  …
```

76	0	138888	0	13	1	…
96	0	178000	0	12	1	…
108	0	114800	1	8	0	…
131	0	180000	0	11	0	…
150	0	250000	0	16	0	…
190	1	157000	0	13	1	…
212	1	108987	1	24	1	…
239	0	120000	1	29	0	…
240	0	150290	0	7	0	…
243	0	140920	0	13	1	…
244	1	148999	1	13	1	…
292	0	113999	1	8	1	…
308	0	220450	0	6	0	…

	PerformanceScore	EngagementSurvey	EmpSatisfaction	\
18	1	4.50	4	
42	1	4.50	5	
55	0	3.70	5	
76	0	4.30	5	
96	0	5.00	5	
108	1	4.60	4	
131	1	4.50	4	
150	1	4.83	3	
190	2	2.39	3	
212	0	5.00	5	
239	1	3.88	3	
240	1	4.94	3	
243	1	3.60	5	
244	1	4.30	4	
292	1	4.33	3	
308	0	4.60	5	

	SpecialProjectsCount	DaysLateLast30	Absences	AgeAtHire	\
18	5	0	8	28	
42	7	0	8	-56	
55	0	0	15	25	
76	5	0	4	-57	
96	5	0	15	30	
108	4	0	10	-57	
131	0	0	19	-52	
150	0	0	10	-43	
190	6	4	13	25	
212	3	0	13	32	
239	7	0	12	41	
240	5	0	17	-56	
243	7	0	13	39	
244	6	0	8	-53	
292	7	0	9	28	

```
308                          6                0         16           30

     EmploymentDuration  TimeSinceLastReview  TimeUntilTermination
18                  432                 3083                   240
42                 -307                 1622                  1168
55                 2494                 1602                  1188
76                  668                 1633                  1157
96                 1664                 1630                  1160
108                  27                 3078                    54
131                 548                 1616                  1174
150                1220                 1620                  1170
190                1358                 1584                  1206
212                1400                 2870                    22
239                1405                 1958                   270
240                -430                 1600                  1190
243                1018                 1588                  1202
244                1395                 3094                   304
292                 737                 2321                     7
308                2034                 1585                  1205

[16 rows x 26 columns]
Number of outliers: 16
Threshold: 39097.90035366955
```

In this code, df represents your original dataset. The code calculates the Euclidean distances of each data point from the centroid in the PCA space. The threshold is defined as the 95th percentile of the distances, which can be adjusted as per your requirements. The code then identifies the data points that exceed the threshold as outliers and prints the list of outliers.

By analyzing the outliers, you can observe if they have any common characteristics or patterns that differentiate them from the rest of the data points. This analysis can provide insights into potential anomalies or peculiarities in your dataset.

3 Section C (Classification):

Split the dataset into features (X) and target variable (y):

[ ]:
```
X = df.drop('EmploymentStatus', axis=1)
y = df['EmploymentStatus']
```

Split the data into training and testing sets:

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

3.1 Random Forest Classifier:

Create a Random Forest classifier and perform parameter tuning using GridSearchCV:

[ ]:
```
# Create the Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Define the parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search to find the best parameters
grid_search = GridSearchCV(rf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameter values
best_params = grid_search.best_params_
best_params
```

[ ]:
```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 200}
```

Train the Random Forest classifier with the best parameters:

[ ]:
```
# Create a new Random Forest classifier with the best parameters
rf_best = RandomForestClassifier(random_state=42, **best_params)

# Fit the classifier to the training data
rf_best.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(min_samples_split=5, n_estimators=200, random_state=42)
```

Get feature importances using Random Forest and explain their significance::

[ ]:
```
# Get feature importances
importances = rf_best.feature_importances_

# Create a dataframe to store feature importances
feature_importances = pd.DataFrame({'Feature': X_train.columns, 'Importance':␣
↪importances})

# Sort the features by importance in descending order
feature_importances = feature_importances.sort_values(by='Importance',␣
↪ascending=False)

# Print the feature importances
```

```python
print(feature_importances)
```

```
                  Feature  Importance
24   TimeUntilTermination    0.282077
23     TimeSinceLastReview   0.278856
7                   Termd    0.251752
16        EngagementSurvey   0.024534
22      EmploymentDuration   0.022366
13               ManagerID   0.021174
6                   Salary   0.016125
19          DaysLateLast30   0.013537
14       RecruitmentSource   0.012429
8               PositionID   0.012173
20                Absences   0.010548
21               AgeAtHire   0.010124
18    SpecialProjectsCount   0.008907
15        PerformanceScore   0.008434
4              PerfScoreID   0.007148
3                   DeptID   0.004572
1          MaritalStatusID   0.003570
17         EmpSatisfaction   0.003263
0                MarriedID   0.001891
10             CitizenDesc   0.001606
5   FromDiversityJobFairID   0.001384
12                RaceDesc   0.001174
2                 GenderID   0.000991
9                      Sex   0.000987
11          HispanicLatino   0.000378
```

Compute accuracy for the model and provide sensitivity and specificity measurements for every class :

[ ]:
```python
# Predict the employment status using the trained random forest Classifier
y_pred = rf_best.predict(X_test)

# Compute accuracy
accuracy_rf = accuracy_score(y_test, y_pred)

# Compute sensitivity and specificity for each class
class_labels = rf_best.classes_
sensitivity = {}
specificity = {}

for label in class_labels:
    label_index = np.where(class_labels == label)[0][0]
    true_positive = np.sum((y_test == label) & (y_pred == label))
    false_negative = np.sum((y_test == label) & (y_pred != label))
    true_negative = np.sum((y_test != label) & (y_pred != label))
```

```python
    false_positive = np.sum((y_test != label) & (y_pred == label))
    sensitivity[label] = true_positive / (true_positive + false_negative)
    specificity[label] = true_negative / (true_negative + false_positive)

# Print accuracy, sensitivity, and specificity for each class
for label in class_labels:
    print(f"Class: {label}")
    print(f"Accuracy: {accuracy_rf:.4f}")
    print(f"Sensitivity: {sensitivity[label]:.4f}")
    print(f"Specificity: {specificity[label]:.4f}")
    print()
```

```
Class: 0
Accuracy: 0.9524
Sensitivity: 1.0000
Specificity: 1.0000

Class: 1
Accuracy: 0.9524
Sensitivity: 0.2500
Specificity: 1.0000

Class: 2
Accuracy: 0.9524
Sensitivity: 1.0000
Specificity: 0.9333
```

The above code performs parameter tuning using GridSearchCV to find the best combination of hyperparameters for the Random Forest classifier. Then, it trains the classifier using the best parameters and computes the feature importances.

The feature importances indicate the relative importance of each feature in predicting the target variable. Higher values indicate greater importance. By analyzing the feature importances, you can identify which features have a stronger influence on the model's predictions. It helps you understand which variables are more relevant in determining the employment status.

3.2 Support Vector Machine:

Create an SVM classifier and perform parameter tuning using GridSearchCV:

[ ]:
```python
# Create the SVM classifier
svm = SVC(random_state=42)

# Define the parameter grid for tuning
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
```

```
}

# Perform grid search to find the best parameters
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameter values
best_params = grid_search.best_params_
best_params
```

[ ]:
```
{'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
```

Train the SVM classifier with the best parameters:

[ ]:
```
# Create a new SVM classifier with the best parameters
svm_best = SVC(random_state=42, **best_params)

# Fit the classifier to the training data
svm_best.fit(X_train, y_train)
```

[ ]:
```
SVC(C=0.1, kernel='linear', random_state=42)
```

Compute accuracy for the model and provide sensitivity and specificity measurements for every class :

[ ]:
```
# Predict the employment status using the trained svm Classifier
y_pred = svm_best.predict(X_test)

# Compute accuracy
accuracy_svc = accuracy_score(y_test, y_pred)

# Compute sensitivity and specificity for each class
class_labels = rf_best.classes_
sensitivity = {}
specificity = {}

for label in class_labels:
    label_index = np.where(class_labels == label)[0][0]
    true_positive = np.sum((y_test == label) & (y_pred == label))
    false_negative = np.sum((y_test == label) & (y_pred != label))
    true_negative = np.sum((y_test != label) & (y_pred != label))
    false_positive = np.sum((y_test != label) & (y_pred == label))
    sensitivity[label] = true_positive / (true_positive + false_negative)
    specificity[label] = true_negative / (true_negative + false_positive)

# Print accuracy, sensitivity, and specificity for each class
for label in class_labels:
    print(f"Class: {label}")
```

```
    print(f"Accuracy: {accuracy_svc:.4f}")
    print(f"Sensitivity: {sensitivity[label]:.4f}")
    print(f"Specificity: {specificity[label]:.4f}")
    print()
```

```
Class: 0
Accuracy: 0.9365
Sensitivity: 1.0000
Specificity: 1.0000

Class: 1
Accuracy: 0.9365
Sensitivity: 0.5000
Specificity: 0.9661

Class: 2
Accuracy: 0.9365
Sensitivity: 0.8889
Specificity: 0.9556
```

The above code performs parameter tuning using GridSearchCV to find the best combination of hyperparameters for the SVM classifier. Then, it trains the classifier using the best parameters and computes accuracy, confusion matrix, and classification report on the test data.

The accuracy score provides an overall measure of how well the model predicts the employment status. The confusion matrix gives a breakdown of the predicted and actual labels, showing true positives, true negatives, false positives, and false negatives. From the confusion matrix, you can calculate sensitivity (true positive rate) and specificity (true negative rate) for each class.

3.3 Gradient Boosting Classifier:

Create a Gradient Boosting Classifier and perform parameter tuning using GridSearchCV:

[ ]:
```
# Create the Gradient Boosting Classifier
gbc = GradientBoostingClassifier(random_state=42)

# Define the parameter grid for tuning
param_grid = {
    'learning_rate': [0.1, 0.5, 1],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7]
}

# Perform grid search to find the best parameters
grid_search = GridSearchCV(gbc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameter values
```

```python
best_params = grid_search.best_params_
best_params
```

[ ]: {'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 100}

Train the Gradient Boosting Classifier with the best parameters:

[ ]:
```python
# Create a new Gradient Boosting Classifier with the best parameters
gbc_best = GradientBoostingClassifier(random_state=42, **best_params)

# Fit the classifier to the training data
gbc_best.fit(X_train, y_train)
```

[ ]: GradientBoostingClassifier(learning_rate=0.5, random_state=42)

Compute accuracy for the model and provide sensitivity and specificity measurements for every class :

[ ]:
```python
# Predict the employment status using the trained Gradient Boosting Classifier
y_pred = gbc_best.predict(X_test)

# Compute accuracy
accuracy_gbc = accuracy_score(y_test, y_pred)

# Compute sensitivity and specificity for each class
class_labels = rf_best.classes_
sensitivity = {}
specificity = {}

for label in class_labels:
    label_index = np.where(class_labels == label)[0][0]
    true_positive = np.sum((y_test == label) & (y_pred == label))
    false_negative = np.sum((y_test == label) & (y_pred != label))
    true_negative = np.sum((y_test != label) & (y_pred != label))
    false_positive = np.sum((y_test != label) & (y_pred == label))
    sensitivity[label] = true_positive / (true_positive + false_negative)
    specificity[label] = true_negative / (true_negative + false_positive)

# Print accuracy, sensitivity, and specificity for each class
for label in class_labels:
    print(f"Class: {label}")
    print(f"Accuracy: {accuracy_gbc:.4f}")
    print(f"Sensitivity: {sensitivity[label]:.4f}")
    print(f"Specificity: {specificity[label]:.4f}")
    print()
```

```
Class: 0
Accuracy: 0.9206
```

```
Sensitivity: 1.0000
Specificity: 1.0000

Class: 1
Accuracy: 0.9206
Sensitivity: 0.5000
Specificity: 0.9492

Class: 2
Accuracy: 0.9206
Sensitivity: 0.8333
Specificity: 0.9556
```

Obtain the feature importance from a trained Gradient Boosting Classifier:

```
[ ]: # Train the Gradient Boosting Classifier with the best parameters (assuming
     ↪gbc_best is already trained)
     gbc_best.fit(X_train, y_train)

     # Get feature importance scores
     feature_importance = gbc_best.feature_importances_

     # Create a dictionary to store feature importance scores with corresponding
     ↪feature names
     importance_dict = {feature: importance for feature, importance in zip(X.
     ↪columns, feature_importance)}

     # Sort the feature importance scores in descending order
     sorted_importance = sorted(importance_dict.items(), key=lambda x: x[1],
     ↪reverse=True)

     # Display feature importance scores
     for feature, importance in sorted_importance:
         print(f"{feature}: \t{importance}")
```

```
TimeSinceLastReview:    0.4540968698912378
TimeUntilTermination:   0.4145000809896252
EngagementSurvey:       0.03190955006893112
EmploymentDuration:     0.026467782003560827
Termd:  0.02381732416831714
Salary:         0.01966346301148855
PositionID:     0.013886897577337064
AgeAtHire:      0.006653270279196991
RecruitmentSource:      0.004805240933436853
Absences:       0.0030264448578457333
ManagerID:      0.0009500512010532898
DeptID:         0.00019079427836132264
```

```
SpecialProjectsCount:   2.3332388883857895e-05
HispanicLatino:         6.43832575253447e-06
MaritalStatusID:        9.472581456192514e-07
EmpSatisfaction:        9.458217961975859e-07
DaysLateLast30:         5.542809330630451e-07
Sex:    1.259239151668275e-08
CitizenDesc:    3.4160603284144587e-11
MarriedID:      2.8205621772115578e-11
PerfScoreID:    9.338868158221302e-12
RaceDesc:       1.940450127019775e-17
GenderID:       0.0
FromDiversityJobFairID:         0.0
PerformanceScore:       0.0
```

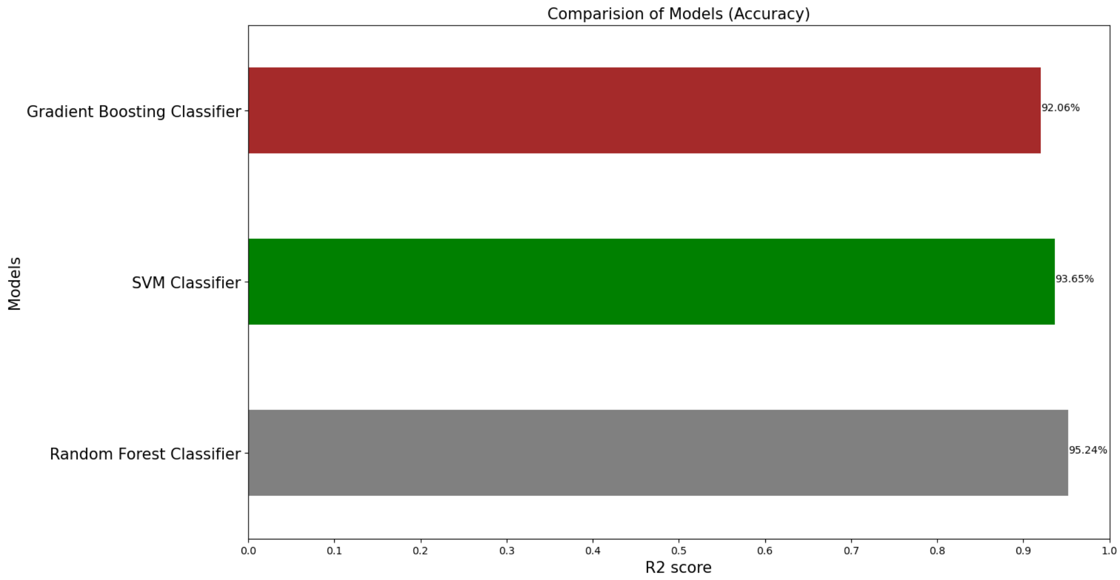
The above code performs parameter tuning using GridSearchCV to find the best combination of hyperparameters for the Random Forest classifier. Then, it trains the classifier using the best parameters and computes the feature importances.

The feature importances indicate the relative importance of each feature in predicting the target variable. Higher values indicate greater importance. By analyzing the feature importances, you can identify which features have a stronger influence on the model's predictions. It helps you understand which variables are more relevant in determining the employment status.

Comparison of the model in terms of accuracy:

[ ]:
```
scores = pd.Series([accuracy_rf, accuracy_svc, accuracy_gbc,  ])
Model_Names = ['Random Forest Classifier','SVM Classifier','Gradient Boosting␣
 ↪Classifier']
```

[ ]:
```
ax = scores.plot(kind =␣
 ↪'barh',figsize=(15,9),color=['gray','green','brown','pink','blue','red'])
ax.set_title('Comparision of Models (Accuracy)',fontsize=15)
ax.set_xticks([0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])
ax.set_yticklabels(Model_Names,fontsize=15,)
ax.set_ylabel("Models",fontsize=15)
ax.set_xlabel("R2 score",fontsize=15)
[ax.text(v, i, '{:.2f}%'.format(100*v)) for i, v in enumerate(scores)];
plt.show()
```



4 Section D (Regression):

Import the necessary Libraries for Regression:

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from datetime import timedelta
```

Split the dataset into features (X) and target variable (y):

The TimeUntilTermination is already calculate above from DateofTermination and LastPerformanceReview_Date.

[ ]:
```
X = df.drop('TimeUntilTermination', axis=1)
y = df['TimeUntilTermination']
```

Split the data into training and testing sets:

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

4.1 Random Forest Regressor:

Create a Random Forest Regressor and perform parameter tuning using GridSearchCV:

[ ]:
```
# Create the Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Define the parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search to find the best parameters
grid_search = GridSearchCV(rf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameter values
best_params = grid_search.best_params_
best_params
```

[ ]:
```
{'max_depth': None,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 300}
```

Train the Random Forest Regressor with the best parameters:

[ ]:
```
# Create a new Random Forest Regressor with the best parameters
rf_best = RandomForestRegressor(random_state=42, **best_params)

# Fit the Regressor to the training data
rf_best.fit(X_train, y_train)
```

[ ]:
```
RandomForestRegressor(min_samples_leaf=4, min_samples_split=10,
                      n_estimators=300, random_state=42)
```

Get feature importances using Random Forest and explain their significance::

[ ]:
```
# Get feature importances
importances = rf_best.feature_importances_

# Create a dataframe to store feature importances
feature_importances = pd.DataFrame({'Feature': X_train.columns, 'Importance':␣
↪importances})

# Sort the features by importance in descending order
feature_importances = feature_importances.sort_values(by='Importance',␣
↪ascending=False)
```

```python
# Print the feature importances
print(feature_importances)
```

```
                   Feature  Importance
13        EmploymentStatus    0.454860
7                    Termd    0.396037
24      TimeSinceLastReview   0.129885
23      EmploymentDuration    0.006168
14               ManagerID    0.002254
21                Absences    0.002114
6                   Salary    0.001865
22               AgeAtHire    0.001118
17        EngagementSurvey    0.001066
3                   DeptID    0.000683
10             CitizenDesc    0.000647
15       RecruitmentSource    0.000581
8               PositionID    0.000521
18         EmpSatisfaction    0.000506
1          MaritalStatusID    0.000350
12                RaceDesc    0.000310
5   FromDiversityJobFairID    0.000278
20          DaysLateLast30    0.000252
0                MarriedID    0.000223
2                 GenderID    0.000078
19     SpecialProjectsCount   0.000062
9                      Sex    0.000058
4              PerfScoreID    0.000054
11          HispanicLatino    0.000015
16        PerformanceScore    0.000014
```

Evaluate Random Forest Regressor using MAE, RMSE and R2 scores:

[ ]:
```python
y_pred = rf_best.predict(X_test)
mean_absolute_error(y_pred, y_test)
print('\nMean Absolute Error:', mean_absolute_error(y_pred, y_test))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_pred, y_test)))
print('R2 score :', r2_score(y_pred, y_test))
```

```
Mean Absolute Error: 43.38053964712802
Root Mean Squared Error: 91.50751070428193
R2 score : 0.9620994673655205
```

5 Linear Regression Model:

Create a Linear Regression Model and train it on training set:

[ ]:
```python
# Create the Linear Regression model
Lr = LinearRegression()
model_lr = Lr.fit(X_train, y_train)
y_pred = model_lr.predict(X_test)
```

Evaluate Linear Regression model using MAE, RMSE and R2 scores:

[ ]:
```python
y_pred = model_lr.predict(X_test)
mean_absolute_error(y_pred, y_test)
print('\nMean Absolute Error:', mean_absolute_error(y_pred, y_test))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_pred, y_test)))
print('R2 score :', r2_score(y_pred, y_test))
```

```
Mean Absolute Error: 66.80676107238526
Root Mean Squared Error: 99.47719653678247
R2 score : 0.9568176276262845
```

5.1 Gradient Boosting Regressor:

Create a Gradient Boosting regressor and perform parameter tuning using GridSearchCV:

[ ]:
```python
# Create the Gradient Boosting Regressor
gbr = GradientBoostingRegressor(random_state=42)

# Define the parameter grid for tuning
param_grid = {
    'learning_rate': [0.1, 0.5, 1],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7]
}

# Perform grid search to find the best parameters
grid_search = GridSearchCV(gbr, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameter values
best_params = grid_search.best_params_
best_params
```

[ ]:
```
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
```

Train the Gradient Boosting regressor with the best parameters:

[ ]:
```python
# Create a new Gradient Boosting Regressor with the best parameters
gbr_best = GradientBoostingRegressor(random_state=42, **best_params)

# Fit the classifier to the training data
```

```
gbr_best.fit(X_train, y_train)
```

[ ]: GradientBoostingRegressor(n_estimators=50, random_state=42)

Obtain the feature importance from a trained Gradient Boosting regressor:

[ ]:
```
# Train the Gradient Boosting Regressor with the best parameters (assuming
↪gbr_best is already trained)
gbr_best.fit(X_train, y_train)

# Get feature importance scores
feature_importance = gbr_best.feature_importances_

# Create a dictionary to store feature importance scores with corresponding
↪feature names
importance_dict = {feature: importance for feature, importance in zip(X.
↪columns, feature_importance)}

# Sort the feature importance scores in descending order
sorted_importance = sorted(importance_dict.items(), key=lambda x: x[1],
↪reverse=True)

# Display feature importance scores
for feature, importance in sorted_importance:
    print(f"{feature}: \t{importance}")
```

```
EmploymentStatus:       0.6707972305145762
Termd:  0.3002432521091373
TimeSinceLastReview:    0.007268161051847526
EmploymentDuration:     0.006547285960554348
DeptID:         0.003737772290699512
ManagerID:      0.0026318126412453106
DaysLateLast30:         0.0014975733931413603
Salary:         0.00147108909464373
Absences:       0.0011478487887006887
CitizenDesc:    0.0009712726619059215
RaceDesc:       0.0009557291830794313
RecruitmentSource:      0.0007111559693962408
MaritalStatusID:        0.0005448672412584865
AgeAtHire:      0.00044450208948630616
EngagementSurvey:       0.00034569345848166995
PositionID:     0.0003418473024510636
PerformanceScore:       0.00010742218492525613
FromDiversityJobFairID:         9.410581457913739e-05
PerfScoreID:    9.184394249165618e-05
SpecialProjectsCount:   4.9417050220901125e-05
MarriedID:      1.1725717787294e-07
```

```
GenderID:       0.0
Sex:    0.0
HispanicLatino:         0.0
EmpSatisfaction:        0.0
```

Evaluate Regression model using MAE, RMSE and R2 scores:

```
[ ]: y_pred = gbr_best.predict(X_test)
mean_absolute_error(y_pred, y_test)
print('\nMean Absolute Error:', mean_absolute_error(y_pred, y_test))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_pred, y_test)))
print('R2 score :', r2_score(y_pred, y_test))
```

```
Mean Absolute Error: 40.03239803442558
Root Mean Squared Error: 77.71043517573587
R2 score : 0.9726115670126408
```

Post-process the results and find the exact termination date for each employee

```
[ ]: predictions = gbr_best.predict(X)
# Calculate the exact termination date by adding the predicted time to the last␣
↪performance review date
termination_dates = last_performance_dates + pd.to_timedelta(predictions,␣
↪unit='D')

# Print the exact termination dates
for date in termination_dates:
    print(date)
```

```
2022-03-31 12:08:49.982956800
2016-07-01 15:18:04.971715200
2012-09-25 11:07:14.155046400
2022-02-26 21:39:15.687532800
2016-06-07 17:38:38.820710400
2022-03-09 10:50:54.523334400
2022-02-28 10:41:12.283670400
2022-06-14 11:52:15.164025600
2022-04-11 06:21:42.030374400
2022-05-31 18:31:47.443555200
2016-09-15 07:55:35.822294400
2016-10-28 05:33:39.993955200
2022-06-14 11:52:15.164025600
2022-05-23 21:38:18.785356800
2017-06-04 16:16:14.059372800
2017-09-19 22:44:34.952323200
2022-03-23 05:41:12.196953600
2022-03-25 16:13:44.009270400
2015-08-15 11:50:32.589427200
```

```
2022-03-17 08:09:55.691078400
2022-03-23 05:41:12.196953600
2022-03-15 08:07:07.982544
2022-06-18 04:29:27.597753600
2022-06-20 15:01:59.410070400
2014-10-26 06:26:47.803430400
2016-06-01 16:00:18.966038400
2022-05-31 18:31:47.443555200
2014-08-19 08:48:53.756035199
2013-09-14 05:40:34.377168
2022-03-22 08:18:29.194012800
2022-04-28 02:47:42.678988800
2022-04-06 02:08:35.124547200
2014-06-11 05:47:45.041078400
2022-06-06 17:44:19.186771200
2022-03-25 16:13:44.009270400
2022-04-08 12:41:06.936864
2022-06-08 16:30:08.474371200
2022-04-24 04:56:58.713849600
2022-04-28 02:47:42.678988800
2022-05-31 18:31:47.443555200
2022-04-06 02:08:35.124547200
2022-05-20 02:36:10.628640
2022-03-25 04:26:59.300016
2022-06-02 16:48:14.080521600
2022-03-23 05:41:12.196953600
2022-04-01 07:10:59.744121599
2016-01-07 18:43:05.388067200
2022-04-26 15:29:30.526166400
2022-06-18 04:29:27.597753600
2012-11-30 19:41:56.955926400
2011-10-13 05:22:58.689004800
2022-06-19 08:11:30.150643200
2022-03-23 05:41:12.196953600
2016-10-20 01:12:00.001296
2022-03-26 11:22:52.853980800
2022-05-01 02:51:04.921603200
2022-05-23 21:38:18.785356800
2022-04-08 02:09:56.110204800
2022-04-20 04:17:21.794726400
2022-02-28 10:41:12.283670400
2022-04-12 07:31:31.415606400
2022-05-20 02:36:10.628640
2022-06-14 11:52:15.164025600
2022-06-20 15:01:59.410070400
2022-04-25 23:32:25.265212800
2016-07-20 05:35:55.245206400
2022-04-14 02:30:12.136377600
```

```
2022-04-25 01:56:05.307590400
2015-11-16 07:19:10.200460800
2022-04-25 23:32:25.265212800
2022-04-09 23:50:59.483760
2022-06-02 16:48:14.080521600
2022-04-21 20:14:05.352864
2022-03-31 07:10:59.744121599
2022-06-02 16:48:14.080521600
2022-06-14 11:52:15.164025600
2022-03-03 12:33:31.471315200
2022-04-20 13:19:01.826745600
2022-03-25 04:26:59.300016
2022-04-09 07:50:15.781660800
2017-08-02 13:23:44.669673600
2022-05-22 02:59:08.469312
2022-03-17 08:09:55.691078400
2022-06-14 11:52:15.164025600
2018-08-01 07:48:17.977507200
2017-09-24 08:13:55.124832
2013-04-10 06:02:15.769392
2022-06-08 16:30:08.474371200
2022-03-09 10:50:54.523334400
2016-06-05 05:59:35.933452800
2022-03-02 14:46:52.387276800
2022-05-03 22:59:35.027606400
2022-04-21 20:14:05.352864
2013-10-06 16:03:54.181785599
2012-08-15 21:22:33.974659200
2019-07-10 12:30:47.223964800
2022-03-11 21:23:26.335737600
2022-05-09 03:22:34.709577600
2022-04-20 13:19:01.826745600
2022-04-24 16:03:44.274902400
2022-05-31 18:31:47.443555200
2022-06-18 18:04:59.561414400
2022-03-27 03:40:04.684108800
2016-11-05 14:13:32.572272
2014-12-03 20:55:30.092188800
2015-08-04 14:48:27.058608
2022-05-10 00:34:06.715603200
2022-04-14 02:30:12.136377600
2015-04-26 03:12:53.047324799
2022-03-15 08:07:07.982544
2022-05-31 18:31:47.443555200
2022-06-03 05:04:19.255958400
2011-07-24 12:59:34.904112
2022-04-20 04:17:21.794726400
2022-05-10 00:34:06.715603200
```

```
2022-03-23 05:41:12.196953600
2022-06-06 17:44:19.186771200
2022-05-25 19:36:46.693785600
2022-03-03 12:33:31.471315200
2014-01-23 02:30:18.077414400
2022-04-20 04:17:21.794726400
2014-07-05 03:24:48.233001600
2022-06-11 03:02:40.286688
2022-03-11 08:20:30.013180800
2015-10-29 22:58:53.182876800
2022-05-25 19:36:46.693785600
2022-05-17 16:43:36.661094400
2013-08-15 00:09:09.343987200
2022-04-24 04:56:58.713849600
2012-05-05 20:00:50.737334400
2022-03-01 17:47:45.793536
2022-04-06 02:08:35.124547200
2015-06-04 10:48:47.762179200
2022-03-09 10:50:54.523334400
2022-03-31 07:10:59.744121599
2022-05-23 21:38:18.785356800
2013-08-05 21:43:06.478953600
2012-11-19 04:26:47.316076800
2022-03-23 05:41:12.196953600
2022-03-29 01:36:18.170640
2022-06-08 18:05:58.861017600
2022-06-16 22:24:46.976342400
2022-04-06 02:08:35.124547200
2018-05-21 21:37:23.429913600
2022-05-17 16:43:36.661094400
2022-03-17 08:09:55.691078400
2016-06-26 01:52:25.688380800
2014-04-29 02:47:48.218524800
2022-04-09 23:50:59.483760
2022-04-08 02:09:56.110204800
2022-03-29 01:36:18.170640
2015-07-24 06:34:53.997312
2013-11-05 18:59:55.045247999
2022-05-08 01:38:58.986268800
2022-04-08 12:41:06.936864
2022-03-31 12:08:49.982956800
2022-03-09 10:50:54.523334400
2015-12-10 00:31:28.109136
2014-08-07 07:41:43.900108800
2022-05-03 22:59:35.027606400
2022-03-29 01:36:18.170640
2013-05-22 13:46:32.616336
2022-03-11 08:20:30.013180800
```

```
2017-12-17 15:28:21.991612800
2022-04-20 04:17:21.794726400
2022-06-08 16:30:08.474371200
2022-04-21 20:14:05.352864
2022-04-06 02:08:35.124547200
2018-06-17 15:05:37.824432
2022-05-31 18:31:47.443555200
2022-04-16 13:02:43.948694400
2016-08-18 02:23:57.046761600
2022-05-20 02:36:10.628640
2022-04-09 23:50:59.483760
2015-10-23 01:48:24.137740800
2017-09-20 13:57:28.973635200
2022-06-16 09:08:18.135744
2022-04-06 02:08:35.124547200
2012-11-30 18:30:20.293516800
2022-04-08 02:09:56.110204800
2022-03-23 05:41:12.196953600
2022-03-17 18:39:39.794860800
2022-05-10 00:34:06.715603200
2022-04-24 04:56:58.713849600
2022-04-21 20:14:05.352864
2016-08-25 17:38:09.714278400
2022-04-06 02:08:35.124547200
2022-03-23 05:41:12.196953600
2014-03-11 18:34:14.570659200
2022-03-13 18:53:01.825497600
2022-06-08 16:30:08.474371200
2022-03-23 05:41:12.196953600
2022-04-25 23:32:25.265212800
2022-05-17 16:43:36.661094400
2022-03-29 01:36:18.170640
2022-04-20 04:17:21.794726400
2022-04-08 02:09:56.110204800
2022-06-14 11:52:15.164025600
2013-10-08 11:23:26.874844800
2016-08-03 04:10:18.546441600
2022-05-31 18:31:47.443555200
2022-05-22 02:59:08.469312
2022-04-14 02:30:12.136377600
2022-02-28 10:41:12.283670400
2022-03-27 03:40:04.684108800
2018-07-08 15:25:27.612393600
2013-11-13 04:22:09.513984
2022-04-14 02:30:12.136377600
2022-05-10 00:34:06.715603200
2022-05-03 22:59:35.027606400
2022-03-23 05:41:12.196953600
```

```
2013-11-11 19:38:11.410022400
2015-12-06 01:29:53.742825600
2016-05-31 10:56:45.014956800
2017-10-06 18:24:49.630694400
2015-11-04 18:18:36.479894400
2015-07-25 20:08:27.285216
2013-06-26 15:46:09.349996800
2016-02-03 12:09:55.164153600
2022-03-23 05:41:12.196953600
2022-03-03 12:33:31.471315200
2012-09-14 09:50:07.334217600
2022-06-11 17:22:04.444406400
2022-04-28 02:47:42.678988800
2019-01-01 16:48:48.315571200
2022-03-09 10:50:54.523334400
2013-05-24 18:10:23.973110400
2015-05-07 02:56:52.020038400
2022-05-08 01:38:58.986268800
2018-05-10 19:17:24.899078400
2022-04-20 04:17:21.794726400
2016-01-28 14:21:59.382489600
2022-05-20 03:16:08.473411200
2011-12-09 14:28:21.077558400
2012-03-02 23:45:50.665276800
2015-11-28 21:17:43.846137600
2016-08-22 20:16:58.996214400
2022-05-17 16:43:36.661094400
2022-05-23 21:38:18.785356800
2018-07-05 01:20:02.462524800
2022-05-08 01:38:58.986268800
2022-04-25 23:32:25.265212800
2018-11-06 09:11:57.595027200
2022-05-31 18:31:47.443555200
2015-09-15 16:48:05.788108800
2022-05-22 02:59:08.469312
2022-06-01 15:30:54.037296
2018-07-04 04:34:04.295308799
2022-03-17 18:39:39.794860800
2015-10-18 05:00:14.660496
2022-05-31 18:31:47.443555200
2022-05-08 01:38:58.986268800
2022-03-09 10:50:54.523334400
2022-05-25 19:36:46.693785600
2022-03-23 05:41:12.196953600
2022-03-25 04:26:59.300016
2015-06-30 00:57:02.302272
2022-04-20 04:17:21.794726400
2022-03-27 03:40:04.684108800
```

```
2014-01-14 19:33:58.404470400
2022-04-12 07:31:31.415606400
2022-05-14 10:03:36.627984
2022-03-23 05:41:12.196953600
2022-06-15 08:51:21.757766399
2022-05-23 21:38:18.785356800
2015-08-03 07:49:31.395388800
2022-05-12 11:06:38.527920
2022-03-29 01:36:18.170640
2022-03-31 07:10:59.744121599
2022-03-17 08:09:55.691078400
2022-04-28 10:04:57.077529600
2022-04-09 11:23:01.309603200
2022-04-12 07:31:31.415606400
2022-04-25 23:32:25.265212800
2022-05-31 18:31:47.443555200
2022-02-28 10:41:12.283670400
2013-01-02 03:35:14.485603200
2017-09-29 20:52:49.846108800
2022-04-20 04:17:21.794726400
2016-09-17 11:31:23.434608
2010-10-28 20:28:35.283187200
2022-03-23 05:41:12.196953600
2022-02-28 16:25:01.365312
2022-05-17 16:43:36.661094400
2015-10-05 00:56:15.353116800
2018-01-18 03:26:57.881472
2014-10-10 13:23:05.604345600
2022-03-23 05:41:12.196953600
2022-03-31 12:08:49.982956800
2016-03-22 20:26:59.917891200
2022-06-20 15:01:59.410070400
2022-03-31 07:10:59.744121599
2017-06-10 13:29:44.992319999
2015-12-30 01:15:07.833340800
2022-04-21 20:14:05.352864
2022-05-20 02:36:10.628640
2012-03-18 13:49:36.690153600
2015-09-19 17:49:17.175878400
2022-03-03 12:33:31.471315200
2022-05-07 01:38:58.986268800
2014-10-04 08:08:21.973776
2015-11-12 05:59:53.525356800
2011-12-25 05:44:49.323955200
2015-01-29 17:51:49.734432
2016-04-17 06:15:29.461564799
2022-06-06 17:44:19.186771200
2022-06-19 08:11:30.150643200
```

```
2016-01-30 11:16:09.764025600
2022-06-03 07:00:20.782684800
2022-04-28 02:47:42.678988800
2022-04-24 06:32:49.100496
```

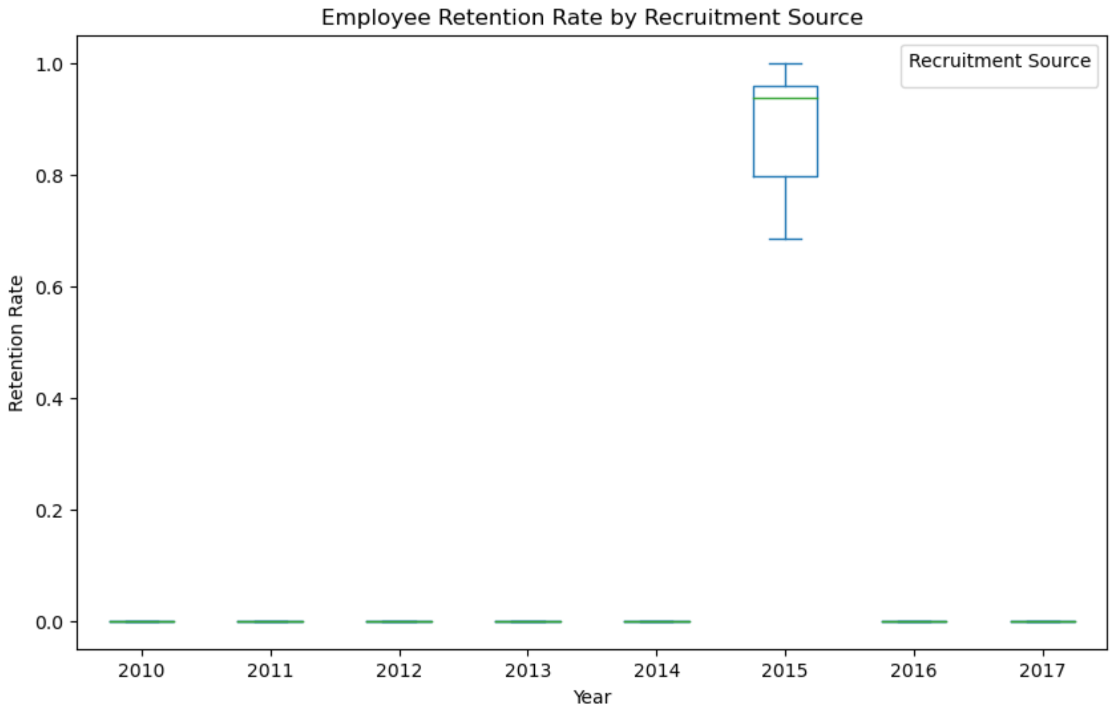
6 Section E :

Calculate the Employee Retention Rate for each year from 2008 until 2017 for every recruitment source and display it on a suitable graph:

[ ]:
```
df1 = pd.read_csv("Human_Resuorces_Analytics.csv")
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have already loaded the data into a DataFrame called 'df'
df1['DateofTermination'] = df1['DateofTermination'].
  ↪fillna(df1['DateofTermination'].mode()[0])
# Filter the data for the years 2008 to 2017
df1['Year'] = pd.to_datetime(df1['DateofTermination']).dt.year
filtered_df = df1[df1['Year'].between(2008, 2017)]

# Calculate the retention rate for each recruitment source and year
retention_rates = filtered_df.groupby(['RecruitmentSource', 'Year']).
  ↪apply(lambda x: 1 - x['Termd'].mean())

# Plot the retention rates on a graph
retention_rates.unstack().plot(kind='box', figsize=(10, 6))
plt.xlabel('Year')
plt.ylabel('Retention Rate')
plt.title('Employee Retention Rate by Recruitment Source')
plt.legend(title='Recruitment Source')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



Calculate the diversity index for each department based on race, gender, and age and display the department in descending order of their diversity index:

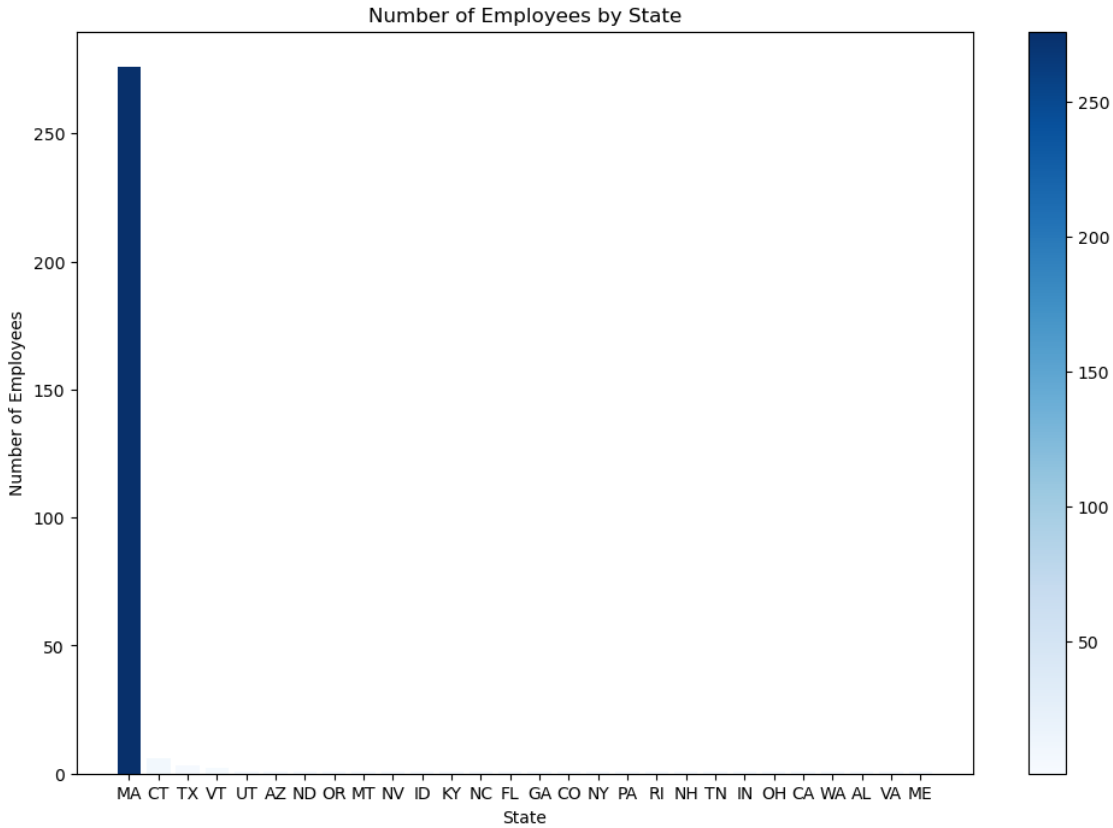
[ ]:
```
import pandas as pd

# Assuming you have already loaded the data into a DataFrame called 'df'
# Calculate age at hire
df1['DOB'] = pd.to_datetime(df1['DOB'])
df1['DateofHire'] = pd.to_datetime(df1['DateofHire'])
df1['AgeAtHire'] = (df1['DateofHire'] - df1['DOB']).dt.days // 365
# Calculate the diversity index
diversity_df = df1.groupby('Department').agg({'RaceDesc': lambda x: len(x.
↪unique()), 'GenderID': lambda x: len(x.unique()), 'AgeAtHire': lambda x:␣
↪len(x.unique())})
diversity_df['DiversityIndex'] = diversity_df['RaceDesc'] +␣
↪diversity_df['GenderID'] + diversity_df['AgeAtHire']

# Sort departments in descending order of diversity index
diversity_df = diversity_df.sort_values('DiversityIndex', ascending=False)

# Display the departments and their diversity index
print(diversity_df[['DiversityIndex']])
```

```
                      DiversityIndex
```

```
Department
Production                51
Sales                     28
IT/IS                     27
Software Engineering      12
Admin Offices             11
Executive Office           3
```

Create a map visualization that shows the number of employees currently working in the company from each state:

[ ]:
```python
import matplotlib.pyplot as plt

# Count the number of employees from each state
state_counts = df1['State'].value_counts()

# Read the state codes and corresponding counts into lists
state_codes = state_counts.index.tolist()
employee_counts = state_counts.values.tolist()

# Create a colormap for coloring the states
colormap = plt.cm.get_cmap('Blues')

# Plot the number of employees by state
fig, ax = plt.subplots(figsize=(12, 8))
bars = ax.bar(state_codes, employee_counts, color=colormap(employee_counts))

# Add labels and title
ax.set_xlabel('State')
ax.set_ylabel('Number of Employees')
ax.set_title('Number of Employees by State')

# Create a colorbar legend
sm = plt.cm.ScalarMappable(cmap=colormap, norm=plt.
 ↪Normalize(vmin=min(employee_counts), vmax=max(employee_counts)))
sm.set_array([])
cbar = plt.colorbar(sm)

# Show the plot
plt.show()
```



7 Section F:

Classification Task:

[ ]:
```
X = df.drop('EmploymentStatus', axis=1)
y = df['EmploymentStatus']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create a new Random Forest classifier with the best parameters
rf_best = RandomForestClassifier(random_state=42, )

# Fit the classifier to the training data
rf_best.fit(X_train, y_train)

# Predict the employment status using the trained random forest Classifier
y_pred = rf_best.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score

# Assuming you have your true labels and predicted labels stored in y_true and␣
↪y_pred, respectively

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted')

# Calculate recall
recall = recall_score(y_test, y_pred, average='weighted')

# Calculate F1 score
f1 = f1_score(y_test, y_pred, average='weighted')

y_pred_prob = rf_best.predict_proba(X_test)
# Calculate AUC-ROC
auc_roc = roc_auc_score(y_test, y_pred_prob,  multi_class='ovr')



print('Accuracy: ', accuracy)
print('Precision: ', precision)
print('Recall: ', recall)
print('F1: ', recall)
print('Auc_Roc: ', auc_roc)
```

```
Accuracy:  0.9523809523809523
Precision:  0.9487607908660539
Recall:  0.9523809523809523
F1:  0.9523809523809523
Auc_Roc:  0.9704052451698403
```

Regression Task:

[ ]:
```
X = df.drop('TimeUntilTermination', axis=1)
y = df['TimeUntilTermination']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create a new Gradient Boosting Regressor with the best parameters
gbr_best = GradientBoostingRegressor(random_state=42, **best_params)

# Fit the classifier to the training data
```

```
gbr_best.fit(X_train, y_train)

# Predict the employment status using the trained model
y_pred = gbr_best.predict(X_test)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assuming you have your true values and predicted values stored in y_true and␣
↪y_pred, respectively

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)

# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)

# Calculate root mean squared error
rmse = np.sqrt(mse)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)


print('MAE: ', mae)
print('MSE: ', mse)
print('RMSE: ', rmse)
print('R2: ', r2)
```

```
MAE:  40.03239803442558
MSE:  6038.911735202247
RMSE:  77.71043517573587
R2:  0.9734915883138399
```

[ ]: