HR dashboard

April 21, 2025

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

[3]:
```
# Set visual styles
plt.style.use('ggplot')
sns.set_palette("Set2")
```

[4]:
```
# Function to calculate age from birthdate
def calculate_age(birthdate, reference_date=None):
    if reference_date is None:
        reference_date = datetime.now()

    if pd.isnull(birthdate):
        return np.nan

    # Convert to datetime if it's a string
    if isinstance(birthdate, str):
        try:
            birth_date = pd.to_datetime(birthdate)
        except:
            return np.nan
    else:
        birth_date = birthdate

    # Calculate age
    age = reference_date.year - birth_date.year - ((reference_date.month,
 ↪reference_date.day) < (birth_date.month, birth_date.day))
    return age
```

[5]:
```
# Function to calculate tenure
def calculate_tenure(hiredate, termdate=None, reference_date=None):
    if reference_date is None:
```

```
        reference_date = datetime.now()

    if pd.isnull(hiredate):
        return np.nan

    # Convert to datetime if it's a string
    if isinstance(hiredate, str):
        try:
            hire_date = pd.to_datetime(hiredate)
        except:
            return np.nan
    else:
        hire_date = hiredate

    # Determine end date (termination date or current date)
    if termdate is not None and not pd.isnull(termdate) and termdate != "":
        if isinstance(termdate, str):
            try:
                end_date = pd.to_datetime(termdate)
            except:
                end_date = reference_date
        else:
            end_date = termdate
    else:
        end_date = reference_date

    # Calculate tenure in years
    tenure_years = end_date.year - hire_date.year - ((end_date.month, end_date.
 ↪day) < (hire_date.month, hire_date.day))
    return tenure_years
```

[6]:
```
# Load the data
def load_hr_data(file_path):
    # Read CSV with semicolon delimiter
    df = pd.read_csv(file_path, sep=';')

    # Check for and handle any data cleaning needs

    # Convert date columns to datetime - FIX: specify dayfirst=True to handle␣
 ↪DD/MM/YYYY format
    date_columns = ['Birthdate', 'Hiredate', 'Termdate']
    for col in date_columns:
        if col in df.columns:
            # Explicitly set dayfirst=True to handle DD/MM/YYYY format
            df[col] = pd.to_datetime(df[col], dayfirst=True, errors='coerce')

    # Add calculated columns
```

```
    reference_date = pd.to_datetime('2025-04-21')  # Today's date

    # Calculate age
    df['Age'] = df['Birthdate'].apply(lambda x: calculate_age(x,␣
↪reference_date))

    # Calculate tenure
    df['Tenure'] = df.apply(lambda row: calculate_tenure(row['Hiredate'],␣
↪row['Termdate'], reference_date), axis=1)

    # Flag active vs terminated employees
    df['Status'] = df['Termdate'].apply(lambda x: 'Terminated' if pd.notna(x)␣
↪else 'Active')

    # Print data info for verification
    print(f"Loaded {len(df)} employee records")
    print(f"Date range: {df['Hiredate'].min()} to {df['Hiredate'].max()}")
    print(f"Age range: {df['Age'].min()} to {df['Age'].max()}")
    print(f"Active employees: {len(df[df['Status'] == 'Active'])}")
    print(f"Terminated employees: {len(df[df['Status'] == 'Terminated'])}")

    # Check for any missing values in key columns
    missing_values = df.isnull().sum()
    if missing_values.sum() > 0:
        print("\nMissing values in columns:")
        print(missing_values[missing_values > 0])

    return df
```

[7]:
```
# Load data
df = load_hr_data('HumanResources.csv')

# Basic dataset overview
print("\nDataset Overview:")
print(f"Total Employees: {len(df)}")
print(f"Active Employees: {len(df[df['Status'] == 'Active'])}")
print(f"Terminated Employees: {len(df[df['Status'] == 'Terminated'])}")
print(f"Turnover Rate: {len(df[df['Status'] == 'Terminated']) / len(df) * 100:.
↪2f}%")
```

```
Loaded 8950 employee records
Date range: 2015-01-01 00:00:00 to 2024-12-29 00:00:00
Age range: 20 to 65
Active employees: 7984
Terminated employees: 966

Missing values in columns:
```

```
Termdate    7984
dtype: int64

Dataset Overview:
Total Employees: 8950
Active Employees: 7984
Terminated Employees: 966
Turnover Rate: 10.79%
```

[8]:
```python
# Quick sample to verify data is loaded correctly
print("\nSample data (first 5 rows):")
print(df.head())
```

```
Sample data (first 5 rows):
   Employee_ID First Name Last Name  Gender           State           City  \
0  00-95822412   Danielle   Johnson  Female        New York  New York City
1  00-42868828       John    Taylor    Male  North Carolina      Charlotte
2  00-83197857      Erica   Mcclain    Male        New York  New York City
3  00-13999315   Brittany   Johnson    Male        New York  New York City
4  00-90801586    Jeffery    Wagner  Female        New York  New York City

  Education Level  Birthdate   Hiredate   Termdate        Department  \
0     High School 1980-02-13 2016-04-16 2021-07-05  Customer Service
1        Bachelor 1987-09-22 2017-02-09 2019-06-14                IT
2        Bachelor 1994-05-19 2016-02-03 2021-03-06        Operations
3        Bachelor 1980-04-18 2016-02-06 2018-11-06        Operations
4        Bachelor 1985-04-07 2015-01-11        NaT        Operations

               Job Title  Salary Performance Rating  Age  Tenure      Status
0   Help Desk Technician   81552  Needs Improvement   45       5  Terminated
1   System Administrator  107520               Good   37       2  Terminated
2  Logistics Coordinator   61104               Good   30       5  Terminated
3    Inventory Specialist   73770               Good   45       2  Terminated
4      Operations Analyst   55581       Satisfactory   40      10      Active
```

1 1. Salary Structure and Budget Control Analysis

[10]:
```python
print("\n--- Salary Structure Analysis ---")
# Education level vs. salary
edu_salary = df.groupby('Education Level')['Salary'].agg(['mean', 'median',␣
↪'std', 'count']).reset_index()
edu_salary.columns = ['Education Level', 'Mean Salary', 'Median Salary', 'Std␣
↪Deviation', 'Count']
print(edu_salary)

# Education level vs. performance rating
```

```python
edu_performance = pd.crosstab(df['Education Level'], df['Performance Rating'],
    normalize='index')
print("\nPerformance Rating Distribution by Education Level (%):")
print(edu_performance * 100)
```

```
--- Salary Structure Analysis ---
  Education Level    Mean Salary  Median Salary  Std Deviation  Count
0        Bachelor  69921.570532        66534.0   12293.098034   5416
1     High School  62144.286971        60968.0    6564.515033   1819
2          Master  82675.957154        82026.0   14172.368094   1237
3             PhD  86033.196653        84159.0   16164.175611    478

Performance Rating Distribution by Education Level (%):
Performance Rating  Excellent       Good  Needs Improvement  Satisfactory
Education Level
Bachelor            12.296898  49.963072           7.847120     29.892910
High School         12.864211  21.385377          33.974711     31.775701
Master              35.408246  40.743735           4.607922     19.240097
PhD                 47.698745  34.309623           4.811715     13.179916
```

[11]:

```python
# Calculate ROI for education levels (using performance as proxy)
# Assuming 'Excellent' performance = 4, 'Good' = 3, 'Average' = 2, 'Poor' = 1
performance_map = {'Excellent': 4, 'Good': 3, 'Average': 2, 'Poor': 1}
if 'Performance Rating' in df.columns:
    df['Performance Score'] = df['Performance Rating'].map(performance_map)

    edu_roi = df.groupby('Education Level').agg({
        'Salary': 'mean',
        'Performance Score': 'mean'
    }).reset_index()

    # Calculate simple ROI metric (Performance Score / Salary in 10K units)
    edu_roi['ROI_Metric'] = edu_roi['Performance Score'] / (edu_roi['Salary'] /
    10000)
    print("\nEducation ROI Analysis:")
    print(edu_roi)
```

```
Education ROI Analysis:
  Education Level        Salary  Performance Score  ROI_Metric
0        Bachelor  69921.570532           3.197509    0.457299
1     High School  62144.286971           3.375602    0.543188
2          Master  82675.957154           3.464968    0.419102
3             PhD  86033.196653           3.581633    0.416308
```

2 2. Department Manpower Cost Analysis

[13]:
```
print("\n--- Department Analysis ---")
dept_analysis = df.groupby('Department').agg({
    'Employee_ID': 'count',
    'Salary': ['sum', 'mean', 'median']
}).reset_index()
dept_analysis.columns = ['Department', 'Headcount', 'Total Salary', 'Avg␣
↪Salary', 'Median Salary']
dept_analysis['% of Workforce'] = dept_analysis['Headcount'] / len(df) * 100
dept_analysis['% of Salary Budget'] = dept_analysis['Total Salary'] /␣
↪df['Salary'].sum() * 100
print(dept_analysis.sort_values('Headcount', ascending=False))
```

```
--- Department Analysis ---
         Department  Headcount  Total Salary    Avg Salary  Median Salary  \
5        Operations       2718     177757880  65400.250184        63375.0
6             Sales       1835     139836079  76204.947684        74520.0
0  Customer Service       1673     110146520  65837.728631        63314.0
3                IT       1382     113221190  81925.607815        83865.0
4         Marketing        718      48579179  67659.023677        64897.0
1           Finance        452      34555917  76451.143805        72963.5
2                HR        172      11032946  64145.034884        64034.5

   % of Workforce  % of Salary Budget
5       30.368715           27.987650
6       20.502793           22.016932
0       18.692737           17.342366
3       15.441341           17.826467
4        8.022346            7.648702
1        5.050279            5.440765
2        1.921788            1.737117
```

3 3. Termination and Recruitment Efficiency Analysis

[15]:
```
print("\n--- Termination Analysis ---")
# Termination by department
term_by_dept = df[df['Status'] == 'Terminated'].
↪groupby('Department')['Employee_ID'].count().reset_index()
term_by_dept.columns = ['Department', 'Termination Count']

# Get total employees by department
total_by_dept = df.groupby('Department')['Employee_ID'].count().reset_index()
total_by_dept.columns = ['Department', 'Total Count']

# Calculate termination rate by department
```

```python
term_rate = pd.merge(term_by_dept, total_by_dept, on='Department')
term_rate['Termination Rate'] = term_rate['Termination Count'] /
    term_rate['Total Count'] * 100
print(term_rate.sort_values('Termination Rate', ascending=False))

# Estimate replacement costs
avg_salary_by_dept = df.groupby('Department')['Salary'].mean().reset_index()
term_cost = pd.merge(term_by_dept, avg_salary_by_dept, on='Department')
term_cost['Est. Replacement Cost (Low)'] = term_cost['Termination Count'] *
    term_cost['Salary']
term_cost['Est. Replacement Cost (High)'] = term_cost['Termination Count'] *
    term_cost['Salary'] * 1.5
print("\nEstimated Replacement Costs by Department:")
print(term_cost[['Department', 'Termination Count', 'Est. Replacement Cost
    (Low)', 'Est. Replacement Cost (High)']])
```

```
--- Termination Analysis ---
          Department  Termination Count  Total Count  Termination Rate
1            Finance                 63          452         13.938053
2                 HR                 20          172         11.627907
0   Customer Service                184         1673         10.998207
6              Sales                201         1835         10.953678
5         Operations                289         2718         10.632818
3                 IT                139         1382         10.057887
4          Marketing                 70          718          9.749304

Estimated Replacement Costs by Department:
          Department  Termination Count  Est. Replacement Cost (Low)  \
0   Customer Service                184                 1.211414e+07
1            Finance                 63                 4.816422e+06
2                 HR                 20                 1.282901e+06
3                 IT                139                 1.138766e+07
4          Marketing                 70                 4.736132e+06
5         Operations                289                 1.890067e+07
6              Sales                201                 1.531719e+07

   Est. Replacement Cost (High)
0                  1.817121e+07
1                  7.224633e+06
2                  1.924351e+06
3                  1.708149e+07
4                  7.104197e+06
5                  2.835101e+07
6                  2.297579e+07
```

4 4. Education Investment ROI

[17]:
```
print("\n--- Education Investment ROI Analysis ---")
# Compare performance ratings across education levels
if 'Performance Rating' in df.columns and 'Performance Score' in df.columns:
    edu_perf = df.groupby('Education Level').agg({
        'Performance Score': 'mean',
        'Salary': 'mean'
    }).reset_index()

    # Calculate performance per salary dollar (higher is better)
    edu_perf['Performance per 10K Salary'] = edu_perf['Performance Score'] /␣
 ↪(edu_perf['Salary'] / 10000)
    print(edu_perf)
```

```
--- Education Investment ROI Analysis ---
  Education Level  Performance Score        Salary  Performance per 10K Salary
0        Bachelor           3.197509  69921.570532                    0.457299
1     High School           3.375602  62144.286971                    0.543188
2          Master           3.464968  82675.957154                    0.419102
3             PhD           3.581633  86033.196653                    0.416308
```

5 5. Age vs Salary Optimization

[19]:
```
print("\n--- Age and Salary Analysis ---")
# Age groups
df['Age Group'] = pd.cut(df['Age'], bins=[20, 30, 40, 50, 60, 100],␣
 ↪labels=['20-29', '30-39', '40-49', '50-59', '60+'])
```

```
--- Age and Salary Analysis ---
```

[20]:
```
age_salary = df.groupby(['Age Group', 'Job Title'], observed=True)['Salary'].
 ↪agg(['mean', 'std', 'count']).reset_index()
age_salary = age_salary[age_salary['count'] >= 5]  # Only include groups with␣
 ↪enough data
print("Salary by Age Group and Job Title (where count >= 5):")
print(age_salary.sort_values(['Job Title', 'Age Group']))
```

```
Salary by Age Group and Job Title (where count >= 5):
    Age Group            Job Title          mean           std  count
0       20-29           Accountant  70768.405405   9906.953923     37
26      30-39           Accountant  70938.549020  10054.383771     51
54      40-49           Accountant  73728.897959  11299.287550     49
81      50-59           Accountant  77256.772727  10682.356758     22
108       60+           Accountant  82818.071429  12168.637472     14
..        …                    …             …             …     …
```

```
25       20-29  System Administrator  84063.820896  10002.239401     67
53       30-39  System Administrator  83613.329268  11894.191408     82
80       40-49  System Administrator  86280.974026  10661.676009     77
107      50-59  System Administrator  89329.090909  10832.153066     44
133        60+  System Administrator  89585.111111   9477.059254      9

[123 rows x 5 columns]
```

[21]:
```python
top_jobs = df['Job Title'].value_counts().head(5).index.tolist()
filtered_df = df[df['Job Title'].isin(top_jobs)]

plt.figure(figsize=(12, 7))
sns.boxplot(x='Age Group', y='Salary', hue='Job Title', data=filtered_df)
plt.title('Salary Distribution by Age Group and Top 5 Job Titles')
plt.xlabel('Age Group')
plt.ylabel('Salary ($)')
plt.legend(title='Job Title', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig('age_salary_analysis.png')
plt.close()
```

[22]:
```python
# Calculate the average salary and standard deviation for each age group
age_general = df.groupby('Age Group', observed=True)['Salary'].agg(['mean',␣
↪'std', 'count']).reset_index()
print("\naverage salary and standard deviation for each age group:")
print(age_general)
```

```
average salary and standard deviation for each age group:
  Age Group          mean           std  count
0     20-29  69045.781344  13132.937764   1994
1     30-39  69976.680400  13585.358103   2500
2     40-49  71380.414963  13375.519695   2446
3     50-59  72992.205962  14068.689435   1476
4       60+  75549.930636  15953.661586    519
```

[23]:
```python
# Calculate the salary coefficient of variation (variation coefficient)
age_general['variation coefficient'] = age_general['std'] / age_general['mean']
print("\nSalary coefficient of variation by age group:")
print(age_general[['Age Group', 'variation coefficient']])
```

```
Salary coefficient of variation by age group:
  Age Group  variation coefficient
0     20-29               0.190206
1     30-39               0.194141
2     40-49               0.187384
3     50-59               0.192742
4       60+               0.211167
```

[ ]:

[24]:
```
# Calculate salary dispersion (coefficient of variation) to identify roles with␣
↪high variation
salary_dispersion = df.groupby('Job Title').agg({
    'Salary': ['mean', 'std', 'count']
}).reset_index()
salary_dispersion.columns = ['Job Title', 'Mean Salary', 'Std Deviation',␣
↪'Count']
salary_dispersion = salary_dispersion[salary_dispersion['Count'] >= 10]  # Only␣
↪include roles with enough data
salary_dispersion['Coefficient of Variation'] = salary_dispersion['Std␣
↪Deviation'] / salary_dispersion['Mean Salary']
print("\nJobs with Highest Salary Dispersion (potential optimization targets):")
print(salary_dispersion.sort_values('Coefficient of Variation',␣
↪ascending=False).head(10))
```

```
Jobs with Highest Salary Dispersion (potential optimization targets):
                 Job Title    Mean Salary  Std Deviation  Count  \
0               Accountant   73457.312139   11095.923627    173
20          SEO Specialist   73271.651429   11060.478745    175
6        Financial Analyst   86409.354037   13014.114169    161
10    Help Desk Technician   72426.079918   10748.543878    488
22           Sales Manager  103795.500000   15283.656187     52
24        Sales Specialist   75045.182301   10945.570051    565
17      Operations Analyst   73674.849192   10507.502463    557
21        Sales Consultant   86075.500000   12005.832721    478
27    System Administrator   85466.807143   11087.945213    280
11              IT Manager  113906.821429   14056.571375     28

    Coefficient of Variation
0                   0.151053
20                  0.150952
6                   0.150610
10                  0.148407
22                  0.147248
24                  0.145853
17                  0.142620
21                  0.139480
27                  0.129734
11                  0.123404
```

6 6. HQ vs Branch Cost Comparison

[ ]:

[26]:
```
print("\n--- HQ vs Branch Analysis ---")
# Define a function to identify HQ vs Branch from location
# Assuming New York City is the HQ (based on sample data showing many employees␣
↪there)
def determine_location_type(city):
    if pd.isna(city):
        return 'Unknown'
    elif city.lower() in ['headquarters', 'hq', 'main office', 'corporate']:
        return 'HQ'
    # Add New York City as HQ - this is the key addition
    elif city == 'New York City':
        return 'HQ'
    else:
        return 'Branch'
```

```
--- HQ vs Branch Analysis ---
```

[ ]:

[ ]:

[63]:
```
# Apply location type determination
if 'City' in df.columns:
    df['Location Type'] = df['City'].apply(determine_location_type)

    # Now perform the analysis
    location_analysis = df.groupby('Location Type').agg({
        'Employee_ID': 'count',
        'Salary': ['sum', 'mean']
    }).reset_index()

    location_analysis.columns = ['Location Type', 'Headcount', 'Total Salary',␣
↪'Avg Salary']
    location_analysis['% of Workforce'] = location_analysis['Headcount'] /␣
↪len(df) * 100
    location_analysis['% of Salary Budget'] = location_analysis['Total Salary']␣
↪/ df['Salary'].sum() * 100
    location_analysis['Cost per Employee'] = location_analysis['Total Salary'] /
↪ location_analysis['Headcount']

    print(location_analysis)
```

```
  Location Type  Headcount  Total Salary   Avg Salary  % of Workforce  \
```

```
0        Branch       5991     425087808  70954.399599       66.938547
1            HQ       2959     210041903  70984.083474       33.061453

   % of Salary Budget  Cost per Employee
0            66.92929       70954.399599
1            33.07071       70984.083474
```

[ ]:

```python
# Calculate the cost difference between HQ and branches
if 'HQ' in location_analysis['Location Type'].values and 'Branch' in location_analysis['Location Type'].values:
    hq_cost = location_analysis[location_analysis['Location Type'] == 'HQ']['Cost per Employee'].values[0]
    branch_cost = location_analysis[location_analysis['Location Type'] == 'Branch']['Cost per Employee'].values[0]
    cost_diff = hq_cost - branch_cost
    cost_diff_pct = (cost_diff / branch_cost) * 100

    print(f"\nHQ vs Branch Cost Comparison:")
    print(f"HQ cost per employee: ${hq_cost:,.2f}")
    print(f"Branch cost per employee: ${branch_cost:,.2f}")
    print(f"HQ premium: ${cost_diff:,.2f} ({cost_diff_pct:.1f}%)")
```

```
HQ vs Branch Cost Comparison:
HQ cost per employee: $70,984.08
Branch cost per employee: $70,954.40
HQ premium: $29.68 (0.0%)
```

[28]:

7 7. Performance vs. Salary Correlation

```python
print("\n--- Performance vs. Salary Analysis ---")
if 'Performance Score' in df.columns:
    performance_salary_corr = df.groupby('Performance Rating').agg({
        'Salary': ['mean', 'median', 'count'],
        'Employee_ID': 'count'
    }).reset_index()
    performance_salary_corr.columns = ['Performance Rating', 'Mean Salary', 'Median Salary', 'Salary Count', 'Employee Count']
    performance_salary_corr['% of Workforce'] = performance_salary_corr['Employee Count'] / len(df) * 100
    print(performance_salary_corr)
```

```
--- Performance vs. Salary Analysis ---
```

```
  Performance Rating   Mean Salary  Median Salary  Salary Count  \
0          Excellent  74403.783525        70204.0          1566
1               Good  71484.260165        67391.0          3763
2  Needs Improvement  67031.341941        63290.0          1123
3       Satisfactory  69792.601281        65631.0          2498

   Employee Count  % of Workforce
0            1566       17.497207
1            3763       42.044693
2            1123       12.547486
3            2498       27.910615
```

[ ]: