Employee Turnover Analysis with Python

Date: October 09, Thursday to October 29, Wednesday, 2025
Project Aim and Goals:

- This project aims to provide a data-driven framework for understanding, predicting, and mitigating employee turnover by systematically investigating the factors that influence turnover through *data visualization, exploratory data analysis (EDA), statistical analysis, feature engineering, plus predictive model development and validation*, based on this dataset.
- The ultimate goal is to generate actionable, evidence-based insights that can be directly applied to strategic Human Resources (HR) initiatives to enhance employee retention and organizational stability.

Self-directed professional: Qi Zhou
Email: qiqizhou1996@gmail.com
LinkedIn: linkedin.com/in/qi-zhou-1996to2096/
Github: github.com/QiZhou1996/
Kaggle: kaggle.com/writeups/cazzie1996/employee-turnover-analysis-with-python

Introduction:

Employee turnover is a significant challenge, imposing substantial costs and instability on organizations. Understanding and mitigating this turnover requires moving beyond anecdotal evidence to a **data-driven framework**.

This Employee Turnover Analysis project provides a comprehensive analysis of the factors influencing employee turnover. Leveraging Python to conduct **exploratory data analysis (EDA)**, **statistical analysis**, excellent **data visualization**, and advanced **predictive modelling**, the analysis systematically studies the complex dynamics of employee turnover.

The primary objective is to deliver **actionable, data-based insights** that can be directly integrated into strategic Human Resources (HR) initiatives. The ultimate goal is to enhance employee retention, improve organizational stability, and ensure the business maintains its critical talent pool. This report details the **key drivers of turnover** and validates a robust **random forest model** for accurately forecasting employee turnover risk.

Analysis:

Step 1: Import Packages and Load Data

In [1]:
```
# For data manipulation
import numpy as np
import pandas as pd

# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# For data modelling
from sklearn.ensemble import RandomForestClassifier

# For metrics and helpful functions
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import roc_auc_score
from scipy.stats import pearsonr

# For saving models
import pickle

# Load dataset into a dataframe
raw_data = pd.read_csv("RawData.csv")

# Display the first five rows of the raw data and check if the data is lo
raw_data.head()
```

Out[1]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

Step 2: Understand Data through Descriptive Statistics and Data Cleaning

In [2]:
```
# Gather the basic information about the raw data
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Department             14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

In [3]:
```
# Gather the descriptive statistics about the raw data
raw_data.describe(include='all')
```

Out[3]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours
count	14999.000000	14999.000000	14999.000000	14999.000000
unique	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN
mean	0.612834	0.716102	3.803054	201.050337
std	0.248631	0.171169	1.232592	49.943099
min	0.090000	0.360000	2.000000	96.000000
25%	0.440000	0.560000	3.000000	156.000000
50%	0.640000	0.720000	4.000000	200.000000
75%	0.820000	0.870000	5.000000	245.000000
max	1.000000	1.000000	7.000000	310.000000

In [4]:
```
# Rename columns to make column names more concise as needed
raw_data = raw_data.rename(columns={'Work_accident': 'work_accident',
                           'average_montly_hours': 'average_monthly_hours'
                           'time_spend_company': 'tenure',
                           'Department': 'department'})
raw_data.columns
```

Out[4]:
```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'tenure', 'work_accident', 'left',
       'promotion_last_5years', 'department', 'salary'],
      dtype='object')
```

In [5]:
```
# Check missing values
raw_data.isna().sum()
```

Out[5]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_monthly_hours    0
tenure                   0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

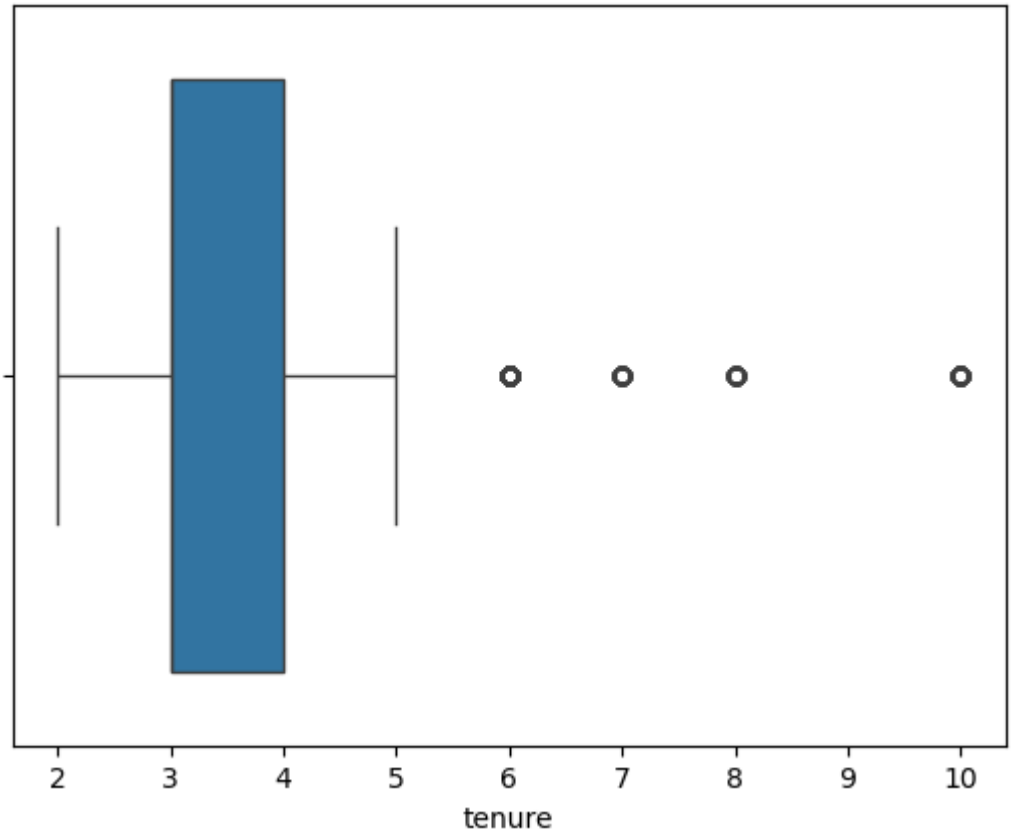
In [6]:
```
# Check duplicates
raw_data.duplicated().sum()
# Check some rows containing duplicates as needed
#raw_data[raw_data.duplicated()].head()
```

Out[6]:
```
np.int64(3008)
```

In [7]:
```
# Remove duplicates
data_deduplicated = raw_data.drop_duplicates(keep='first')
#data_deduplicated.info()
```

In [8]:
```
# Check outliers for the "tenure" variable
sns.boxplot(x=data_deduplicated['tenure'])
```

Out[8]:
```
<Axes: xlabel='tenure'>
```



In [9]:
```
# Determine the number of records containing outliers for the "tenure" va
# Compute the 25th percentile value, the 75th percentile value and the in
percentile25 = data_deduplicated['tenure'].quantile(0.25)
percentile75 = data_deduplicated['tenure'].quantile(0.75)
iqr = percentile75 - percentile25
```

```
# Define the upper limit and lower limit for non-outlier values for the "
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
print("Lower limit:", lower_limit)
print("Upper limit:", upper_limit)

# Identify subset of data containing outliers in `tenure`
outliers = data_deduplicated[(data_deduplicated['tenure'] > upper_limit)

# Count how many rows in the data contain outliers in `tenure`
print("Number of records in the data containing outliers for the 'tenure'
print("Proportion of identified outliers: {:.2f}%".format(len(outliers)/d
```

```
Lower limit: 1.5
Upper limit: 5.5
Number of records in the data containing outliers for the 'tenure' variabl
e: 824
Proportion of identified outliers: 6.87%
```

While the **boxplot analysis** may indicate the presence of **outliers**, their removal during the data cleaning phase is not automatically warranted. The decision regarding the **treatment of outliers** must be driven by the **specific use case** and an assessment of the downstream **modelling requirements**, particularly the model's **sensitivity** to these extreme values. Given the **practical significance** in this particular context, the current determination is to **retain the outliers** for now.

In [10]:
```
# After data cleaning, a new dataframe named "data_cleaned" is created fo
data_cleaned = data_deduplicated.reset_index(drop=True)
data_cleaned.info()
```

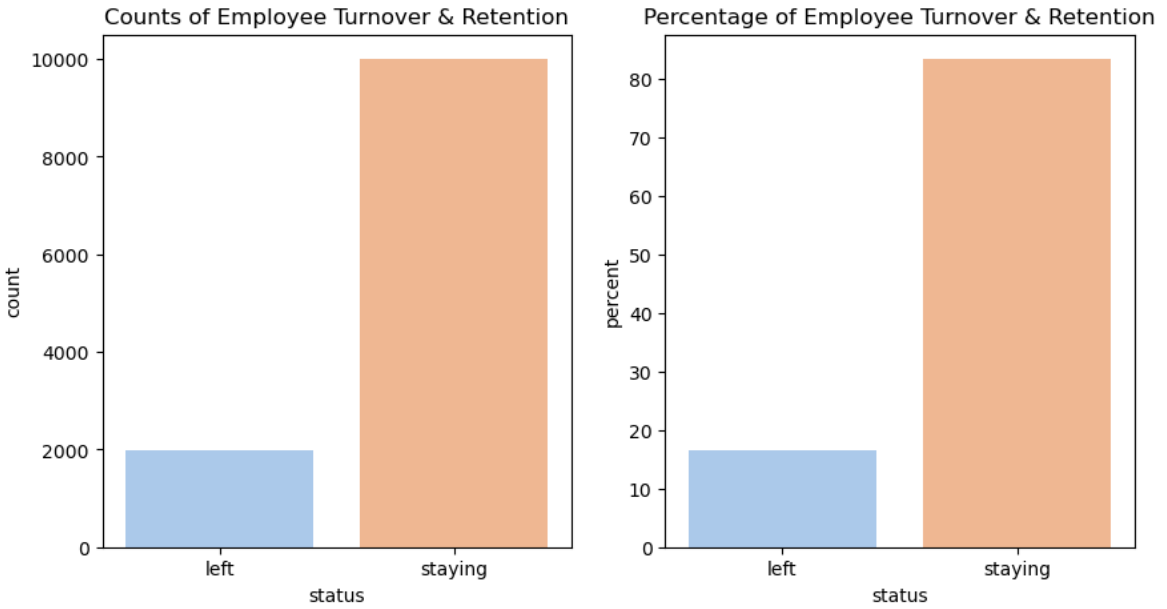
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11991 entries, 0 to 11990
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     11991 non-null  float64
 1   last_evaluation        11991 non-null  float64
 2   number_project         11991 non-null  int64
 3   average_monthly_hours  11991 non-null  int64
 4   tenure                 11991 non-null  int64
 5   work_accident          11991 non-null  int64
 6   left                   11991 non-null  int64
 7   promotion_last_5years  11991 non-null  int64
 8   department             11991 non-null  object
 9   salary                 11991 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 936.9+ KB
```

Step 3: Perform Exploratory Data Analysis (EDA), Statistical Analysis and Data Visualizations

Issue 1: Summary of the current employee turnover situation.

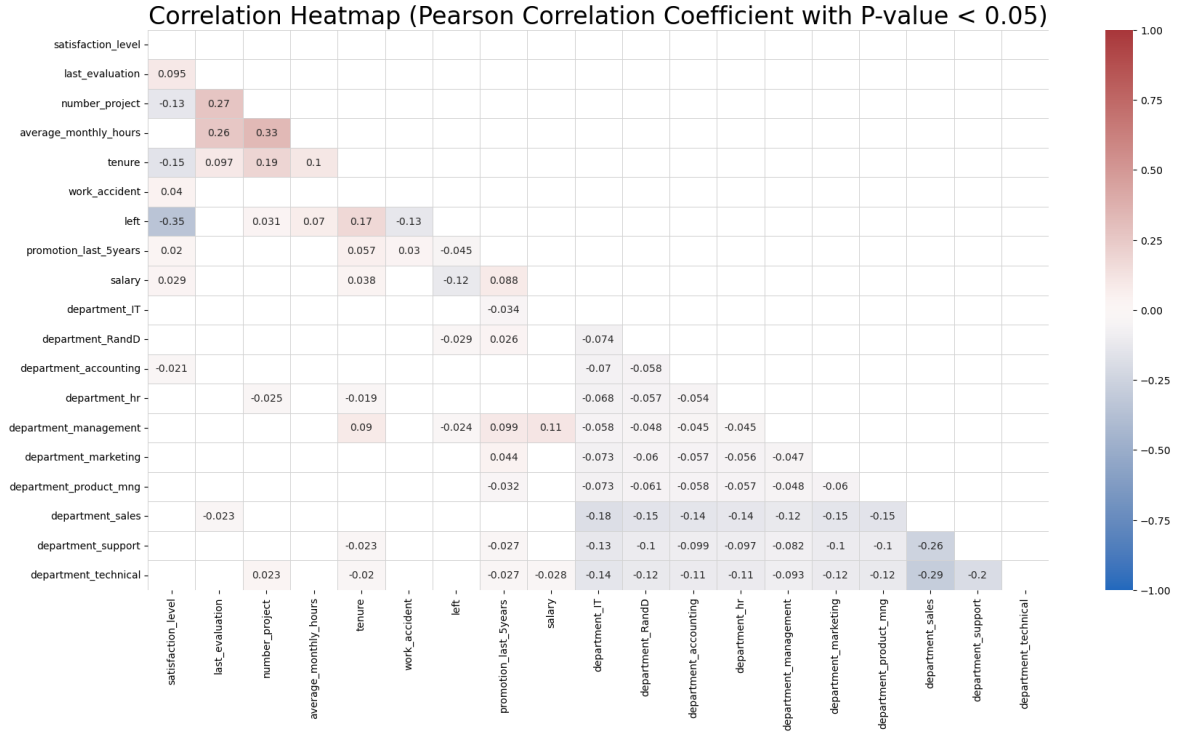
In [11]:
```
# Add a variable named "status" to replace the original "left" variable f
data_cleaned['status'] = 'staying'
data_cleaned.loc[(data_cleaned['left'] == 1), 'status'] = 'left'
```

```
#data_cleaned.info()
fig, ax = plt.subplots(1, 2, figsize = (10,5))
sns.countplot(data=data_cleaned, x='status', stat='count', hue='status',
ax[0].set_title("Counts of Employee Turnover & Retention", fontsize='12')
sns.countplot(data=data_cleaned, x='status', stat='percent', hue='status'
ax[1].set_title("Percentage of Employee Turnover & Retention", fontsize='
plt.show()
```

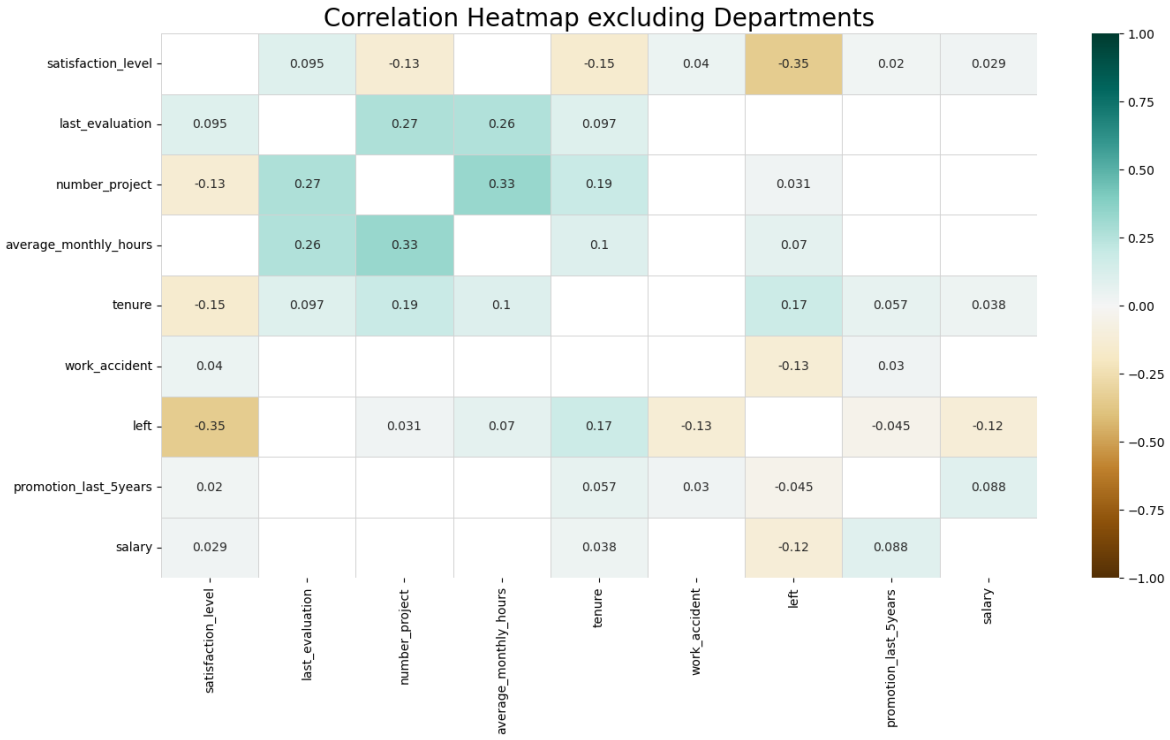


In [12]:

```
# Copy the cleaned dataframe for encoding the non-numeric variables to ch
data_enc = data_cleaned.drop('status', axis=1).copy()

# Encode the "salary" variable as an ordinal numeric category
data_enc['salary'] = (data_enc['salary'].astype('category').cat.set_categ

# Dummy encode the "department" variable
data_enc = pd.get_dummies(data_enc, drop_first=False)

# Compute correlations and p-values
def pearsonr_pval(x, y):
    return pearsonr(x, y)[1]
corr_matrix = data_enc.corr()
p_values_matrix = data_enc.corr(method=pearsonr_pval)

# Plot a correlation heatmap
annot_matrix = corr_matrix.copy()
annot_matrix[p_values_matrix >= 0.05] = np.nan # Significance level = 5%
#annot_matrix[np.abs(annot_matrix) < 0.2] = np.nan
annot_matrix[np.triu(np.ones_like(corr_matrix, dtype=bool))] = np.nan
plt.figure(figsize=(20, 10))
sns.heatmap(data=corr_matrix, vmin=-1, vmax=1, annot=annot_matrix, mask=a
plt.title("Correlation Heatmap (Pearson Correlation Coefficient with P-va
plt.show()
```



In [13]:
```
# Copy the cleaned dataframe for encoding the non-numeric variables to ch
df_enc = data_cleaned.drop(['status', 'department'], axis=1).copy()

# Encode the "salary" variable as an ordinal numeric category
df_enc['salary'] = (df_enc['salary'].astype('category').cat.set_categorie

# Compute correlations and p-values
def pearsonr_pval(x, y):
    return pearsonr(x, y)[1]
corr_matrix = df_enc.corr()
p_values_matrix = df_enc.corr(method=pearsonr_pval)

# Plot a correlation heatmap
annot_matrix = corr_matrix.copy()
annot_matrix[p_values_matrix >= 0.05] = np.nan # Significance level = 5%
#annot_matrix[np.abs(annot_matrix) < 0.1] = np.nan
#annot_matrix[np.triu(np.ones_like(corr_matrix, dtype=bool))] = np.nan
plt.figure(figsize=(16, 8))
sns.heatmap(data=corr_matrix, vmin=-1, vmax=1, annot=annot_matrix, mask=a
plt.title("Correlation Heatmap excluding Departments", fontsize=20)
plt.show()
```



Analysis Finding 1:

The **employee turnover rate** currently stands above **15%**, according to the countplot visualization above.

Correlational analysis utilizing the **Pearson Correlation Coefficient** (at a significance level of p-value < 0.05) reveals several significant, yet complex, relationships with employee turnover:

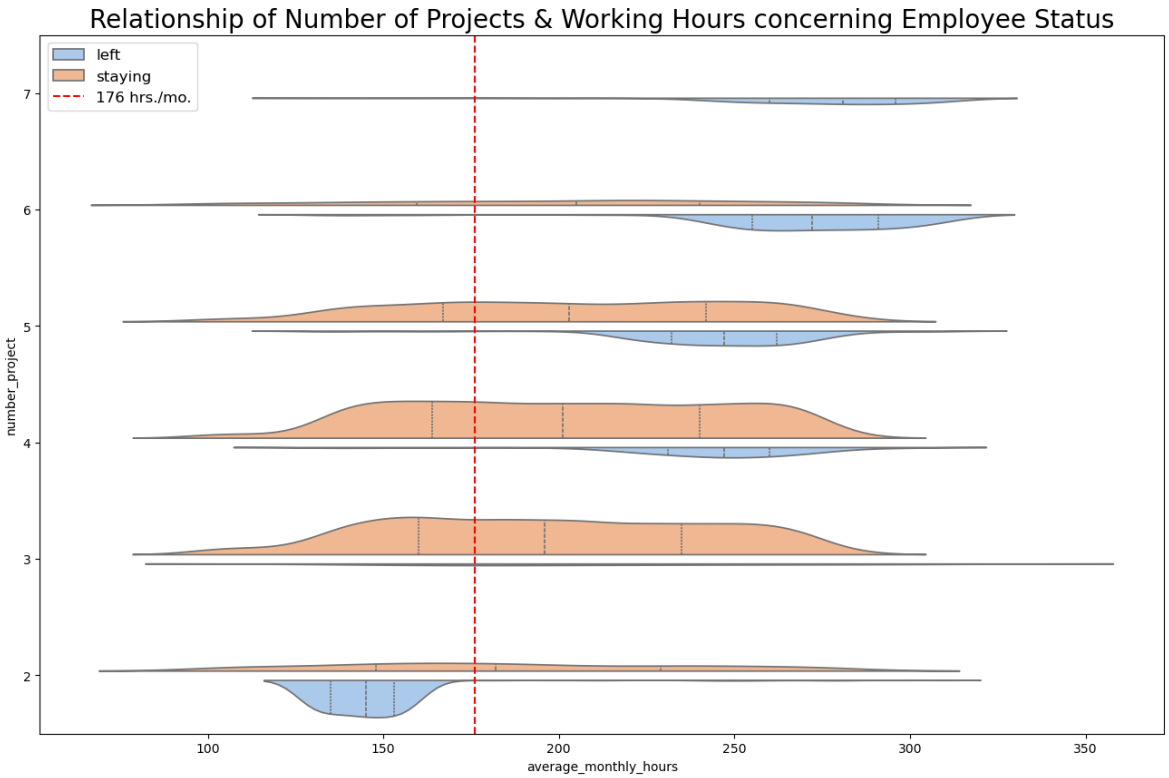
- **Weak but Significant Positive Correlations:** Three key variables—**number of projects** contributed to, average monthly **working hours**, and **tenure** (years)—exhibit a **weak but statistically significant positive linear correlation** with employee turnover. This suggests that as these factors increase, there is a slight, non-random tendency for turnover to rise as well.
- **Negative Correlations:** The following four variables display a **negative correlation** with employee turnover: employee-reported **job satisfaction level**, experience of an **accident while at work**, being **promoted in the last 5 years**, and **salary**. An increase in any of these factors is associated with a decrease in the probability of employee departure.

Additionally, a notable relationship is identified between two independent variables:

- **Number of Projects** and **Working Hours:** The **number of projects an employee contributes to** is **positively correlated** with **average monthly working hours**. This relationship has the **highest positive correlation coefficient** observed in the dataset (r ~= 0.33), indicating a moderate tendency for project load to extend employee working hours directly.

Issue 2: In-depth analysis of the relationship between the number of projects and working hours concerning employee status.

In [14]:
```
fig, ax = plt.subplots(figsize = (16, 10))
sns.violinplot(data = data_cleaned, x='average_monthly_hours', y='number_
ax.axvline(x=176, color='Red', label='176 hrs./mo.', ls='--')
ax.legend(fontsize=12)
plt.title("Relationship of Number of Projects & Working Hours concerning
plt.show()
```



Analysis Finding 2:

The established **baseline for average monthly working hours is 176 hours**, derived from a standard 40-hour, five-day workweek (assuming an average of 22 working days per month multiplied by 8 hours per day).

The **violin plot** visualization strongly suggests that a **significant proportion of the employee population logs hours above this 176-hour baseline**, indicating **widespread overwork**.

A notable correlation emerges when analyzing high-project workloads:

- For employees managing **more than three projects**, the mean working hours of those who **left** are substantially higher than those of employees who **remain**. This discrepancy suggests that **chronic overwork** for high-volume project employees can be a significant contributing factor to **employee turnover** in this segment.

Conversely, an unexpected pattern appears in the two-project cohort:

- Among employees assigned **two projects**, the largest single group of employees who **left** have working hours logged **below the 176-hour average**. This specific finding triggers further investigation to identify underlying factors beyond overwork.

Issue 3: In-depth analysis of the relationship between the number of projects and tenure concerning employee status.

In [15]:
```
plt.figure(figsize=(16, 8))
sns.stripplot(data = data_cleaned, x='tenure', y='number_project', hue='s
plt.title("Relationship of Number of Projects & Tenure concerning Employe
plt.legend(fontsize=12)
plt.show()
```

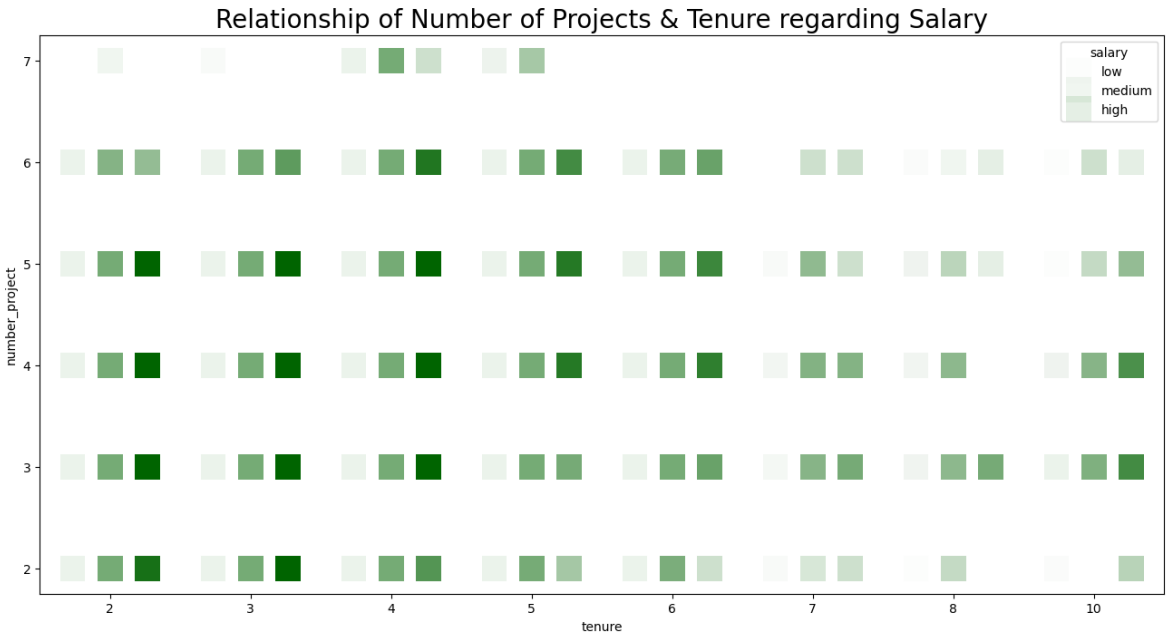
Analysis Finding 3:

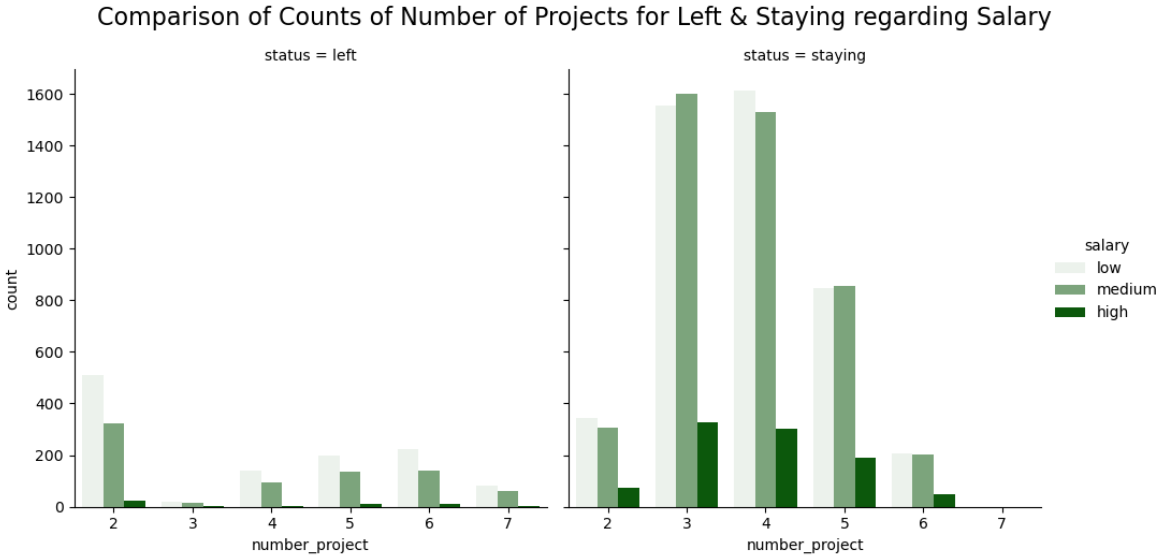
The stripplot visualization reveals several critical patterns regarding the number of projects contributed to, tenure, and turnover.

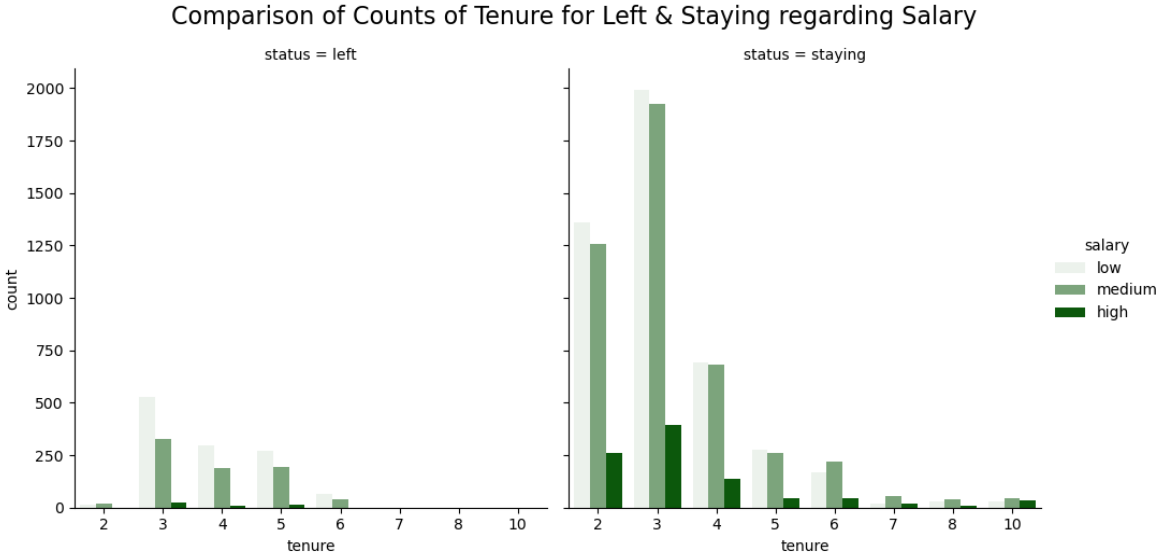
- **High-Impact Departures:** A significant finding is that **all employees who were the top dedicators** across the seven tracked projects have **left**. This suggests a critical loss of subject matter expertise and project momentum.
- **Tenure-Specific Turnover:** When comparing two specific projects, the data indicates a strong correlation between **three years of tenure and employee turnover**. Conversely, the majority of employees with **two years of tenure** are retained.
- **Long-Term Retention:** The data shows complete retention of long-tenured employees, as all employees with more than **six years of tenure** remain, despite a small proportion.

Issue 4: In-depth analysis of the relationship between the number of projects and tenure, regarding salary.

In [16]:
```
plt.figure(figsize=(16, 8))
sns.stripplot(data = data_cleaned, x='tenure', y='number_project', hue='s
plt.title("Relationship of Number of Projects & Tenure regarding Salary",
g1 = sns.catplot(data=data_cleaned, x="number_project", hue='salary', pal
g2 = sns.catplot(data=data_cleaned, x="tenure", hue='salary', palette='li
g1.fig.suptitle("Comparison of Counts of Number of Projects for Left & St
g2.fig.suptitle("Comparison of Counts of Tenure for Left & Staying regard
plt.show()
```







Analysis Finding 4:

The **stripplot** visualization illustrates a clear trend in salary distribution based on employee **tenure** and the **number of projects contributed** to: salaries are heavily **concentrated** towards the lower end of the spectrum.

When considering **employee status** (left or staying), two key observations regarding **turnover** emerge:

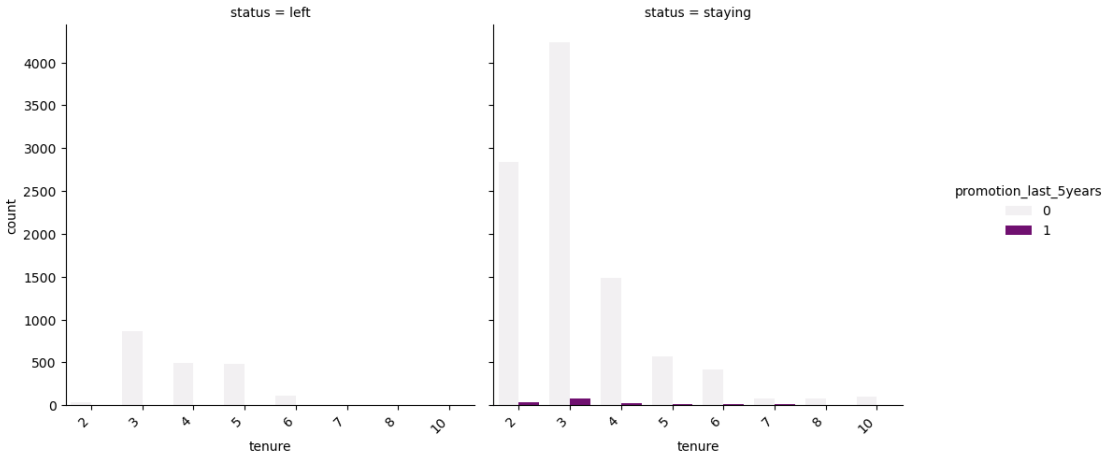
1. **Project Load and Turnover:** There is a **proportional increase** in the rate of employee **departure** as the **number of projects** an employee contributes to rises.
2. **Tenure and Turnover:** A distinct **surge** in the volume of employee **turnover** begins following the **third year of tenure**.

Issue 5: In-depth analysis of promotions in the last 5 years by tenure & department, given employee status.

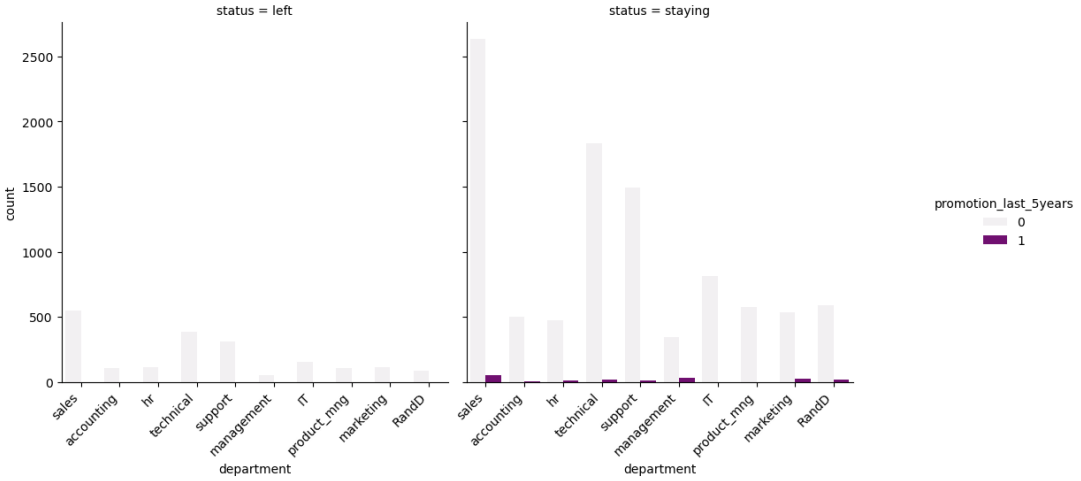
In [17]:
```
g1 = sns.catplot(data=data_cleaned, x="tenure", hue='promotion_last_5year
g1.set_xticklabels(rotation=45, ha="right")
g1.fig.suptitle("Comparison of Counts of Promotions in the last 5 Years f
```

```
g2 = sns.catplot(data=data_cleaned, x="department", hue='promotion_last_5
g2.set_xticklabels(rotation=45, ha="right")
g2.fig.suptitle("Comparison of Counts of Promotions in the last 5 Years f
plt.show()
```

Comparison of Counts of Promotions in the last 5 Years for Left & Staying by Tenure regarding Employee Status



Comparison of Counts of Promotions in the last 5 Years for Left & Staying by Departments regarding Employee Status

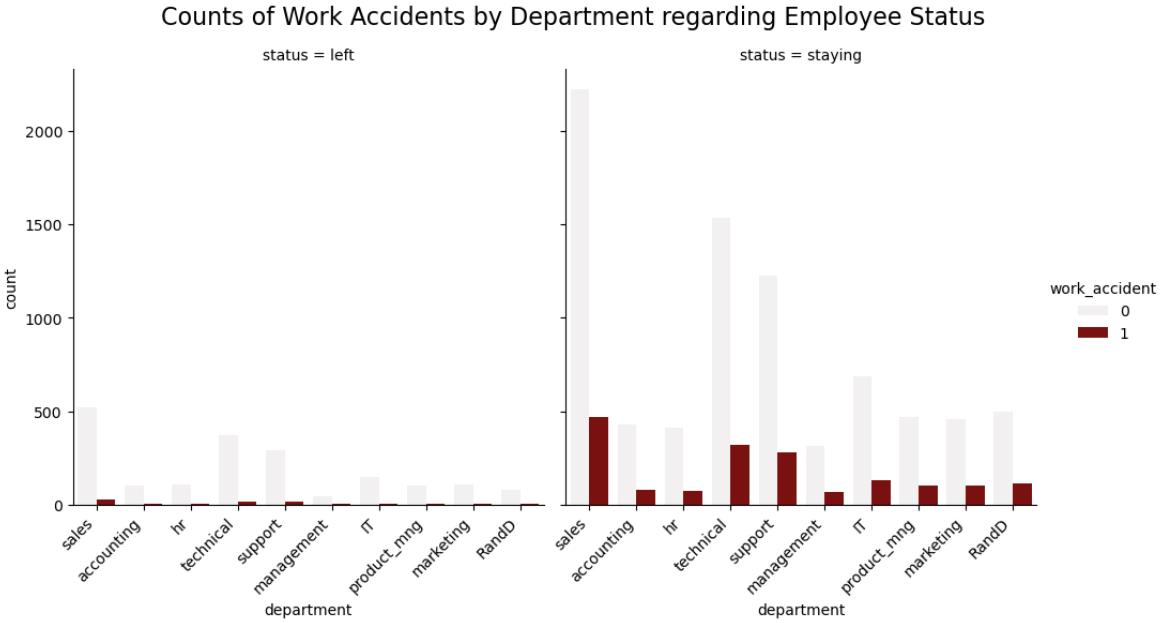


Analysis Finding 5:

As evidenced by the companion catplot displays, the frequency of **promotions** has been **notably low** throughout the last five-year period, with no significant variation observed relative to **employee tenure** or **department**.

Issue 6: In-depth analysis of work accidents by department, given employee status.

In [18]:
```
g = sns.catplot(data=data_cleaned, x="department", hue='work_accident', p
g.set_xticklabels(rotation=45, ha="right")
g.fig.suptitle("Counts of Work Accidents by Department regarding Employee
plt.show()
```



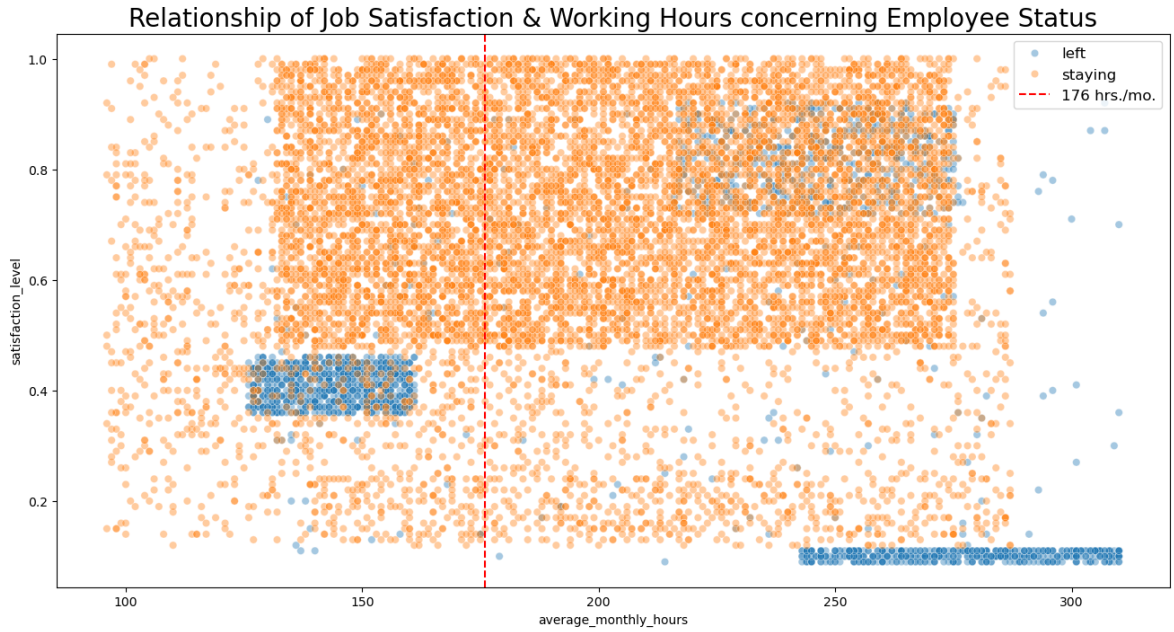
Analysis Finding 6:

Based on the **catplot** above, there appears to be **no significant correlation** between the occurrence of **work accidents** and **employee turnover**.

On the other hand, the **consistently high frequency of work accidents** is notably observed across **all departments**. Although this does not seem to directly drive turnover, the widespread nature of these incidents **warrants further, targeted investigation**.

Issue 7: In-depth analysis of the relationship between job satisfaction and working hours concerning employee status.

In [19]:
```
fig, ax = plt.subplots(figsize = (16, 8))
sns.scatterplot(data=data_cleaned, x='average_monthly_hours', y='satisfac
ax.axvline(x=176, color='Red', label='176 hrs./mo.', ls='--')
ax.legend(fontsize=12)
plt.title("Relationship of Job Satisfaction & Working Hours concerning Em
plt.show()
```



Analysis Finding 7:

This **scatterplot** visualization exhibits **anomalous distribution shapes**, warranting further investigation. The pattern is possibly indicative of **data manipulation** or the **use of synthetic data**.

Analysis Outcomes

The analysis proves a significant **employee turnover rate of over 15%**. Statistical analysis reveals that **seven factors** have weak but significant relationships with this turnover.

Positive Correlations (Higher value = Higher Turnover):

1. **Number of Projects**: more projects an employee contributes to.
2. **Average Working Hours**: longer monthly working hours.
3. **Tenure (Years)**: longer employment.

Negative Correlations (Higher value = Lower Turnover):

1. **Job Satisfaction**: higher reported satisfaction.
2. **Promotion**: being promoted in the last five years.
3. **Salary**: higher pay.
4. **Work Accident**: experiencing an accident while at work.

Analysis Details on the seven factors

- *Overwork*: **Number of Projects** and **Working Hours**, whose relationship has the **highest positive correlation coefficient (r ~= 0.33)** and is statistically significant.
    - The average working hours baseline of **176 hours** per month is frequently surpassed by a significant portion of the workforce, indicative of

widespread overwork.
    - The data clearly establishes a relationship between **high workload and increased turnover**, particularly for **high-performing talent**, as observed from the departure of all top dedicators across the seven tracked projects.
    - The **two-project cohort**, with working hours below the 176-hour baseline, occupies the **largest proportion** of the turnover.
- *Tenure*
    - Turnover **surges starting from the third year of tenure**, especially among the **two-project cohort**.
    - All employees with **more than six years of tenure** remain, but their **proportion is tiny**.
- *Salary* and *Promotion*
    - **Salaries** are generally **skewed towards the lower end** relative to the number of projects and tenure. Insufficient salary, especially when compounded by high workloads, is likely to result in turnover.
    - There have been **notably few promotions in the last five years**, regardless of employee tenure or department. This lack of upward mobility may invoke the negative correlation between promotion and turnover.
- *Work Accidents*:
    - **Work accidents tend not to lead to turnover**.
    - However, the **consistently high frequency across all departments** warrants a separate investigation.
- *Job Satisfaction*
    - Statistically, the employee-reported **job satisfaction level** has the **highest negative correlation (r ~= -0.35)** with turnover.
    - Nevertheless, the strange shapes of its data distribution suggest possible ***data manipulation*** or ***the use of synthetic data***, which should be considered carefully.

Modelling:

Step 1: Apply Feature Engineering

Feature Engineering, through dropping the "satisfaction_level" (due to data leakage) and "last_evaluation" (due to no statistical significance) variables and extracting the "average_monthly_hours" variable to create a new "overworked" variable, improves model performance, increases model interpretability, reduces computational needs, and avoids overfitting.

In [20]:
```
# Feature Selection
#data_enc.info()
data_modelling = data_enc.drop(['satisfaction_level', 'last_evaluation'],

# Feature Extraction
```

```
data_modelling['overworked'] = (data_modelling['average_monthly_hours'] >
data_modelling = data_modelling.drop('average_monthly_hours', axis=1)
data_modelling.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11991 entries, 0 to 11990
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   number_project          11991 non-null  int64
 1   tenure                  11991 non-null  int64
 2   work_accident           11991 non-null  int64
 3   left                    11991 non-null  int64
 4   promotion_last_5years   11991 non-null  int64
 5   salary                  11991 non-null  int8
 6   department_IT           11991 non-null  bool
 7   department_RandD        11991 non-null  bool
 8   department_accounting   11991 non-null  bool
 9   department_hr           11991 non-null  bool
 10  department_management   11991 non-null  bool
 11  department_marketing    11991 non-null  bool
 12  department_product_mng  11991 non-null  bool
 13  department_sales        11991 non-null  bool
 14  department_support      11991 non-null  bool
 15  department_technical    11991 non-null  bool
 16  overworked              11991 non-null  int64
dtypes: bool(10), int64(6), int8(1)
memory usage: 691.0 KB
```

Step 2: Split Train-test Dataset and Train the Model

In [21]:

```
# Define functions
# For getting cross-validation (CV) scores from the grid search
def get_cv_scores(model_name:str, model_object, metric:str):
    '''
    Arguments:
    model_name (string): what you want the model to be called in the outp
    model_object: a fit GridSearchCV object
    metric (string): precision, recall, f1, accuracy, or AUC

    Returns a pandas df with the F1, recall, precision, accuracy, and AUC
    for the model with the best mean 'metric' score across all validation
    '''
    # Create a dictionary that maps the input metric to the actual metric
    metric_dict = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'
                   }
    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)
    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metri
    # Extract Accuracy, precision, recall, and f1 score from that row
    auc = best_estimator_results.mean_test_roc_auc
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
```

```
    accuracy = best_estimator_results.mean_test_accuracy
    # Create table of results
    table = pd.DataFrame()
    table = pd.DataFrame({'Model': [model_name],
                          'Precision': [precision],
                          'Recall': [recall],
                          'F1': [f1],
                          'Accuracy': [accuracy],
                          'AUC': [auc]
                         })
    return table

# For getting test-set scores
def get_test_set_scores(model_name:str, model, X_test_data, y_test_data):
    '''
    Generate a table of test scores.
    In:
        model_name (string): How you want your model to be named in the o
        model:                A fit GridSearchCV object
        X_test_data:          numpy array of X_test data
        y_test_data:          numpy array of y_test data
    Out: pandas df of precision, recall, f1, accuracy, and AUC scores for
    '''
    preds = model.best_estimator_.predict(X_test_data)
    auc = roc_auc_score(y_test_data, preds)
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)
    table = pd.DataFrame({'Model': [model_name],
                          'Precision': [precision],
                          'Recall': [recall],
                          'F1': [f1],
                          'Accuracy': [accuracy],
                          'AUC': [auc]
                         })
    return table

# For saving and reading the model
path = "RandomForestModel.pkl"
def write_pickle(path, model_object):
    '''
    In:
        path: path of the folder where you want to save the pickle
        model_object: a model you want to pickle
        save_as: filename for how you want to save the model
    Out: A call to pickle the model in the folder indicated
    '''
    with open(path, 'wb') as to_write:
        pickle.dump(model_object, to_write)

def read_pickle(path):
    '''
    In:
        path: path to the folder where you want to read from
        saved_model_name: filename of pickled model you want to read in
    Out:
        model: the pickled model
    '''
    with open(path, 'rb') as to_read:
```

```
        model = pickle.load(to_read)
    return model
```

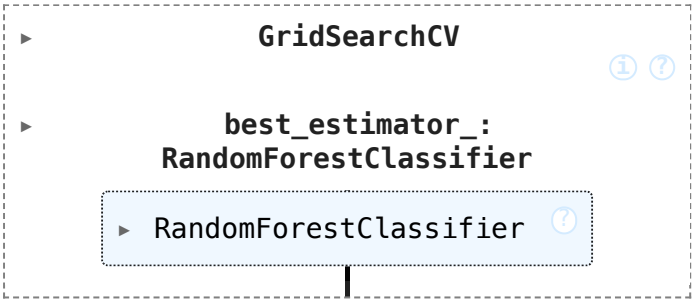
In [22]:
```
# Isolate the Features and Target Variable
y = data_modelling['left']
X = data_modelling.drop('left', axis=1)

# Split Train-test Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
```

In [23]:
```
# Instantiate the random forest model
model = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

# Assign a dictionary of scoring metrics to capture
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

# Instantiate GridSearch
rf = GridSearchCV(estimator=model, param_grid=cv_params, scoring=scoring,
```

In [24]:
```
%%time
rf.fit(X_train, y_train)
```

```
CPU times: user 11min 46s, sys: 9.55 s, total: 11min 55s
Wall time: 12min 4s
```

Out[24]:



In [25]:
```
# Save the model by writing pickle
write_pickle(path, rf)
```

Step 3: Evaluate and Interpret the Model

In [26]:
```
# Read the model in pickle
rf = read_pickle(path)
```

In [27]:
```
# Check best AUC score on CV
rf.best_score_
```

Out[27]:
```
np.float64(0.9548051583489843)
```

In [28]:
```
# Check best params
```

```
rf.best_params_
```

Out[28]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 3,
 'min_samples_split': 2,
 'n_estimators': 500}
```

In [29]:
```
# Check Cross-Validation Scores
rf_cv_results = get_cv_scores('Random Forest CV Scores', rf, 'auc')
rf_cv_results
```

Out[29]:

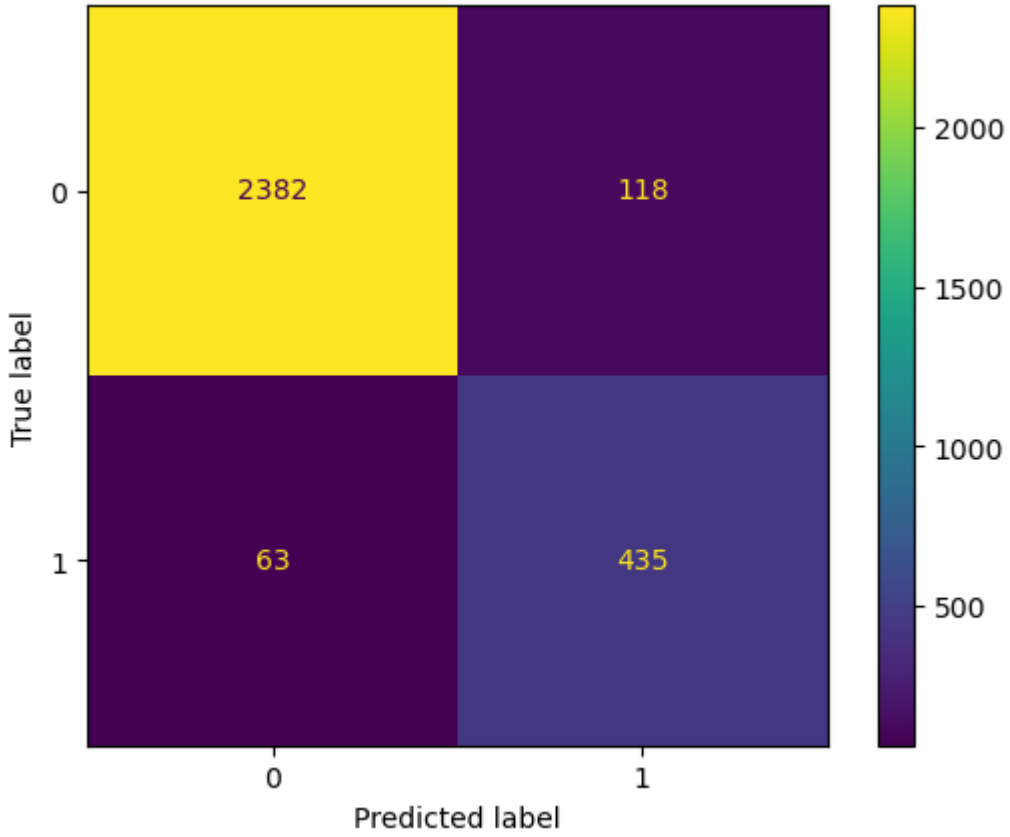
	Model	Precision	Recall	F1	Accuracy	AUC
0	Random Forest CV Scores	0.801514	0.837237	0.818916	0.938508	0.954805

In [30]:
```
# Check predictions on test data
rf_test_scores = get_test_set_scores('Random Forest Test-set Scores', rf,
rf_test_scores
```

Out[30]:

	Model	Precision	Recall	F1	Accuracy	AUC
0	Random Forest Test-set Scores	0.786618	0.873494	0.827783	0.939626	0.913147

In [31]:
```
# Plot a confusion matrix to visualize how well the model predicts on the
# Generate an array of values for the confusion matrix
preds = rf.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, preds, labels=rf.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf.clas
disp.plot(values_format='')
plt.show()
```



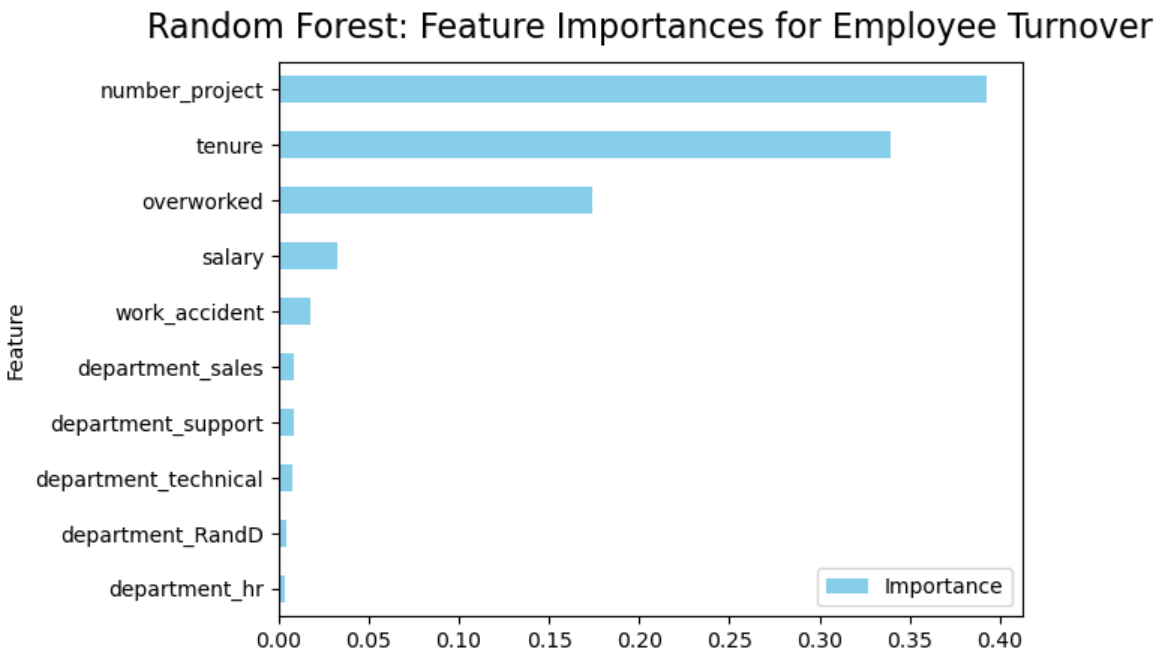
Modelling Finding 1:

Considering the inherent limitations of conventional models—specifically, the **sensitivity of Logistic Regression to data outliers** and the potential for **Decision Trees to overfit** the training data—the **Random Forest (Classification)** algorithm represents the most robust and appropriate model choice for this analysis. Its ensemble nature mitigates overfitting and provides a more generalized, stable, and accurate predictive performance.

The random forest model achieves an **AUC of 91.3%**, a precision of 78.7%, a recall of 87.3%, an F1-score of 82.8%, and an accuracy of 94.0% on the test set.

In [32]:
```
# Get feature importances
feat_impt = rf.best_estimator_.feature_importances_

# Get indices of top 10 features
ind = np.argpartition(rf.best_estimator_.feature_importances_, -10)[-10:]

# Get column labels of top 10 features
feat = X.columns[ind]

# Filter 'feat_impt' to consist of the top 10 feature importances
feat_impt = feat_impt[ind]

# Create a dataframe of the feature importances for plotting
y_df = pd.DataFrame({"Feature":feat,"Importance":feat_impt}).sort_values(
fig = plt.figure()
ax = fig.add_subplot(111)
y_df.plot(kind='barh', ax=ax, x="Feature", y="Importance", color='skyblue
ax.set_title("Random Forest: Feature Importances for Employee Turnover",
```

```
ax.legend(loc='lower right')
plt.show()
```



Modelling Finding 2:

Based on the **Random Forest Feature Importances** above, the following variables demonstrate the strongest predictive power concerning turnover:

1. **Number of Projects:** This factor exhibits the **highest predictive influence** on an employee's decision to leave.
2. **Tenure (Years):** The duration of employment is the second most impactful predictor of employee departure.
3. **Working Hours (Over 176 hrs./mo.):** The indicator for high monthly workload (overwork) is the third most significant variable in foreseeing turnover.

These three features are the most instrumental components of the model for **discriminating between employees who will remain and those who will churn**.

Modelling Outcomes

The rigorous evaluation of multiple classification algorithms has identified the **Random Forest model** as the optimal and most reliable tool for employee churn prediction. This model, relying on its **ensemble architecture**, effectively overcomes the limitations of simpler models—namely, the outlier sensitivity of Logistic Regression and the overfitting likelihood associated with single Decision Trees.

Model Interpretation

The validated Random Forest classifier demonstrates **superior stability and predictive power** on the test dataset, confirming its fitness for deployment in an operational HR environment. Key performance metrics are as follows:

Metric	Value	Interpretation
Accuracy	94.0%	The overall proportion of correct predictions (both turnover and retention) is exceptionally high.
Area Under Curve (AUC)	91.3%	Indicates excellent model discrimination, signifying a 91.3% chance that the model will rank a randomly chosen departing employee higher than a randomly chosen retained employee.
F1-Score	82.8%	Represents a strong balance between Precision and Recall, providing a reliable measure of the model's performance on the less frequent turnover class.
Recall (Sensitivity)	87.3%	The model is highly effective at identifying the employees who *will* leave, minimizing the possibility of false negatives (failing to flag an at-risk employee).
Precision	78.7%	When the model predicts an employee will leave, it is correct nearly 79% of the time, limiting organizational alert fatigue from false positives.

The **Accuracy score of 94.0%** combined with an **AUC score of 91.3%** strongly validates the model's capacity to deliver reliable turnover forewarning, allowing for timely, targeted intervention strategies.

Analysis of Random Forest Feature Importances

The **Feature Importances** derived from the Random Forest model pinpoints the variables that exert the most significant statistical influence on an employee's decision to leave:

1. **Number of Projects:** This factor emerges as the **most critical predictive determinant** of turnover. The volume of concurrent project assignments likely reflects an employee's overall workload pressure and resource allocation, making it the primary variable to monitor.
2. **Tenure (Years):** The duration of employment is the second most impactful predictor. This suggests that specific employment thresholds (e.g., the 3-5 year mark in an employee's career) probably represent elevated flight risks.
3. **Working Hours (Over 176 hrs./mo.):** The indicator for **chronic overwork** is the third most influential feature. This quantitatively confirms that unsustainable workload practices are a significant catalyst for employee departure.

These three factors are the **instrumental components** driving the model's decision-making process, highlighting that **workload management** (*Projects* and *Hours*) and **career stage** (*Tenure*) are the dominant contributors.

Conclusions:

This ***Employee Turnover Analysis with Python*** project successfully establishes a data-driven framework for understanding and mitigating employee turnover. The analysis confirms a critical talent retention crisis with an **employee turnover rate exceeding 15%** and statistically identifies seven factors that significantly, albeit weakly, relate to this turnover. Crucially, the **Random Forest classification model** provides a robust predictive tool of high reliability, with an **Accuracy score of 94.0%** and an **AUC score of 91.3%**, for forecasting employee turnover risk.

Key Data-Driven Insights

The analysis isolates **three dominant drivers of churn**, which together explain the vast majority of the turnover:

- **Excessive Workload and Resource Strain:** The analysis unequivocally confirms that **chronic overwork** is the leading predictor of employee departure. The **Number of Projects** and **Average Working Hours** are the **first and third most critical features** in the predictive model, with their combined positive correlation (r ~= 0.33) being the strongest relationship. This pattern is particularly detrimental to **high-performing talent**, as evidenced by the departure of all top dedicators across tracked projects.

- **Career Stagnation and Uncompetitive Salary: Tenure (Years)** is the **second most impactful predictor**, with a **surge in turnover starting around the third year**. This suggests that the 3-5 year tenure mark represents a critical period of elevated flight risk. This is likely compounded by organizational issues in **compensation and upward mobility**:

    - **Low Salary:** Salaries are skewed towards the lower end, and insufficient pay, especially when combined with a high workload, acts as a significant detractor.
    - **Lack of Promotion:** The **notably low frequency of promotions** across all departments indicates a critical failure in internal career development pathways, reinforcing the negative relationship between promotion and turnover. The absence of upward movement, especially around the high-risk tenure mark, transforms career stagnation into a significant driver of churn.
- **Model Validation and Proactive Intervention:** The **Random Forest model** is validated as an exceptionally stable and predictive tool for operational deployment. With an **Accuracy of 94.0%** and an **AUC of 91.3%**, the model exhibits superior overall correctness and discrimination. The **Recall of 87.3%** is particularly critical, as it assures that the model is **highly effective at identifying employees who *will* leave**, minimizing the costly failure of not flagging an at-risk employee (false negatives).

Strategic Business Recommendations

1. **Implement Mandatory Workload Audits:** A **zero-tolerance policy** for chronic overwork must be adopted through **eliminating excessive working hours** and **balancing project resource allocation** to protect high-performing talent.
2. **Establish a Proactive Mid-Tenure Career Review Program:** The **critical three-year flight risk period** can be changed into a **retention milestone** through involving **accelerated promotion reviews** and **market-competitive salary adjustments** to mitigate the compounded risk from career stagnation and insufficient compensation.
3. **Deploy the Random Forest Model:** HR strategy should shift from reactive replacement to **proactive, targeted retention intervention** through integrating the validated Random Forest model into the HR data infrastructure and automating the generation of **bi-weekly or monthly flight risk scores** for all employees.
4. **Ensure Job Satisfaction Data Integrity:** The suspicious distribution of the Job Satisfaction data necessitates an immediate **data governance review** to determine if data manipulation or synthetic generation occurred. Future data collection must ensure the validity and reliability of this important metric.
5. **Initiate Work Accident Investigation:** The consistently high frequency of work accidents across all departments, while not a direct turnover driver, is a **serious concern** to be addressed.