In [93]:

```
#Hlanhla Hlungwane
#Python project on Jupyter notebook
#Employee Turnover Analytics
#09 July 2025
```

In [94]:

```
import numpy as np   #Allows computations in Linear Algebra
import pandas as pd  #Allows for data processing
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [95]:

```
df = pd.read_csv('HR_comma_sep.csv')
```

In [96]:

```
df.head()
```

Out[96]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_cc
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	

In [97]:

```
df.describe()
```

Out[97]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spen
count	14999.000000	14999.000000	14999.000000	14999.000000	14
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 [98]:

```
#Train and Test Data
from sklearn.model_selection import train_test_split
X = df.drop(columns = ['left'])
y = df.left
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3)
```

In [99]:

```
print('Ratio of workers who left in train:', y_train.sum()/y_train.count())
```

```
Ratio of workers who left in train: 0.23522352235223523
```

In [100…

```
print('Ratio of workers who left in test:', y_test.sum()/y_test.count())
```

```
Ratio of workers who left in test: 0.2438
```

In [101…

```
print('Accuracy:', 1-y_test.sum()/y_test.size)
```

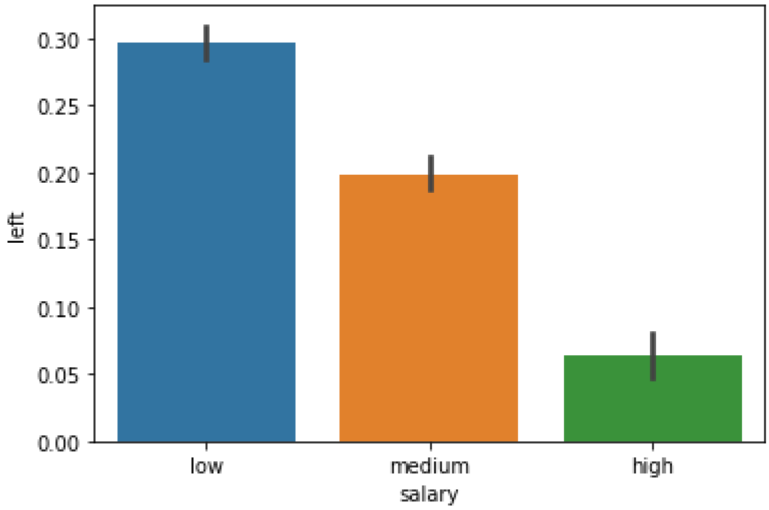
```
Accuracy: 0.7562
```

In [102…

```
#Data Cleaning
sns.barplot(x=X_train.salary, y=y_train, order = ['low', 'medium', 'high'])
```

Out[102]:

```
<AxesSubplot: xlabel='salary', ylabel='left'>
```



In [103…

```
#Employees with low salaries left the company the most
#The employees earning higher are less likely leave the company
```

In [104…

```
#The label encoder will be labelled with low, medium and high salaries
#These will be converted to 0, 1, 2
```

In [105…

```
#Encoding is when categorical data is converted to numerical data
```

In [106…

```
salary_encoder = {'low': 0, 'medium': 1, 'higher':2}
X_train['salary'] = X_train.salary.map(salary_encoder)
X_train['salary'] = X_test.salary.map(salary_encoder)
```

In [107…

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

dept_encoder = OneHotEncoder(handle_unknown='ignore')
dept_encoder.fit_transform(X_train[['sales']])
dept_encoder.transform(X_test[['sales']])


transformer = make_column_transformer(
    (OneHotEncoder(), ['sales']),
    remainder='passthrough')

transformed = transformer.fit_transform(X_train)
X_train = pd.DataFrame(
    transformed,
    columns=transformer.get_feature_names_out()
)

transformed = transformer.transform(X_test)
X_test = pd.DataFrame(
    transformed,
    columns=transformer.get_feature_names_out()
)
```

In [108…

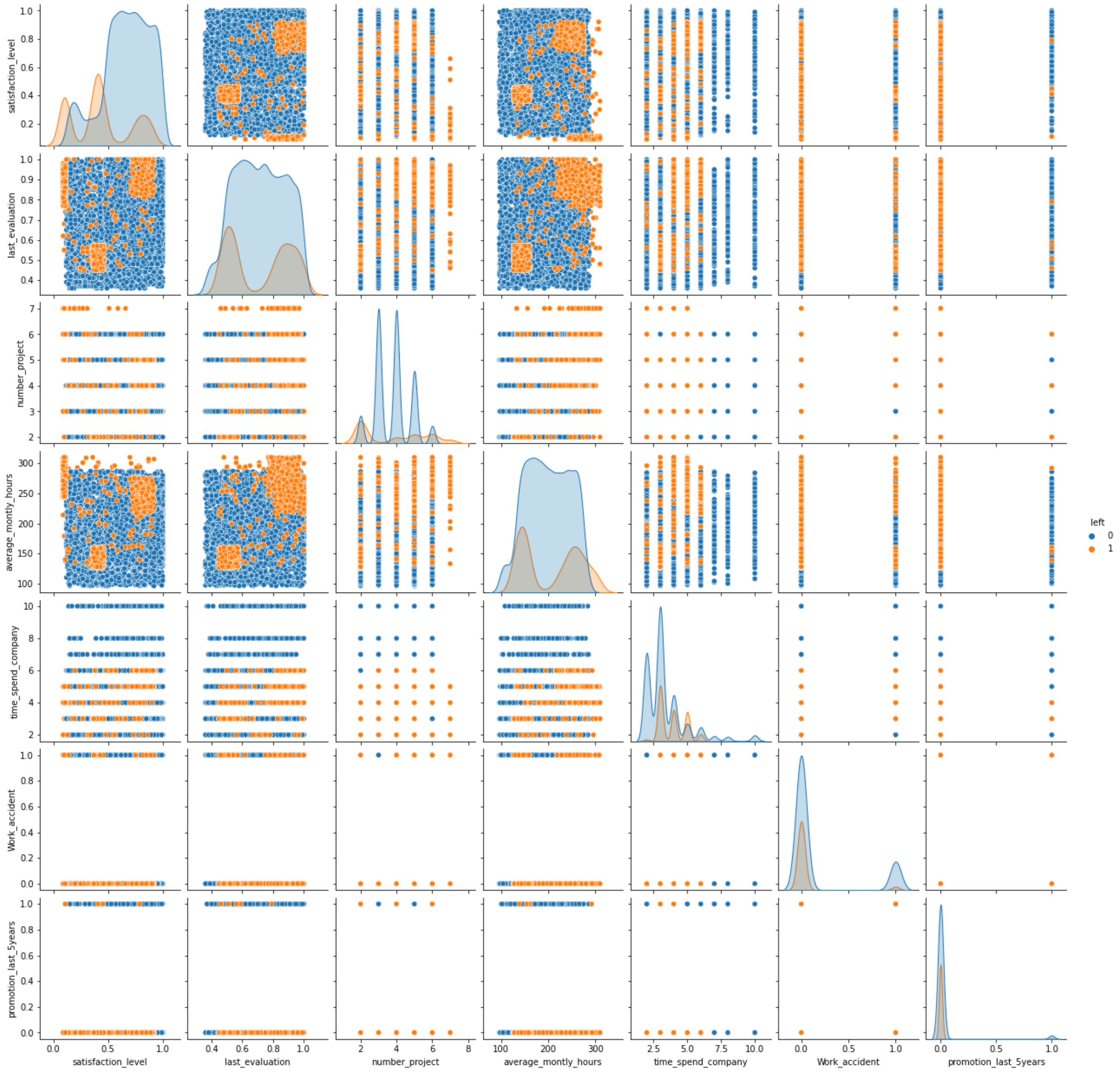
```
X = X_train.copy()
X['left'] = y_train
```

In [109…

```
#Visualizations

sns.pairplot(data=df, hue = 'left')
```

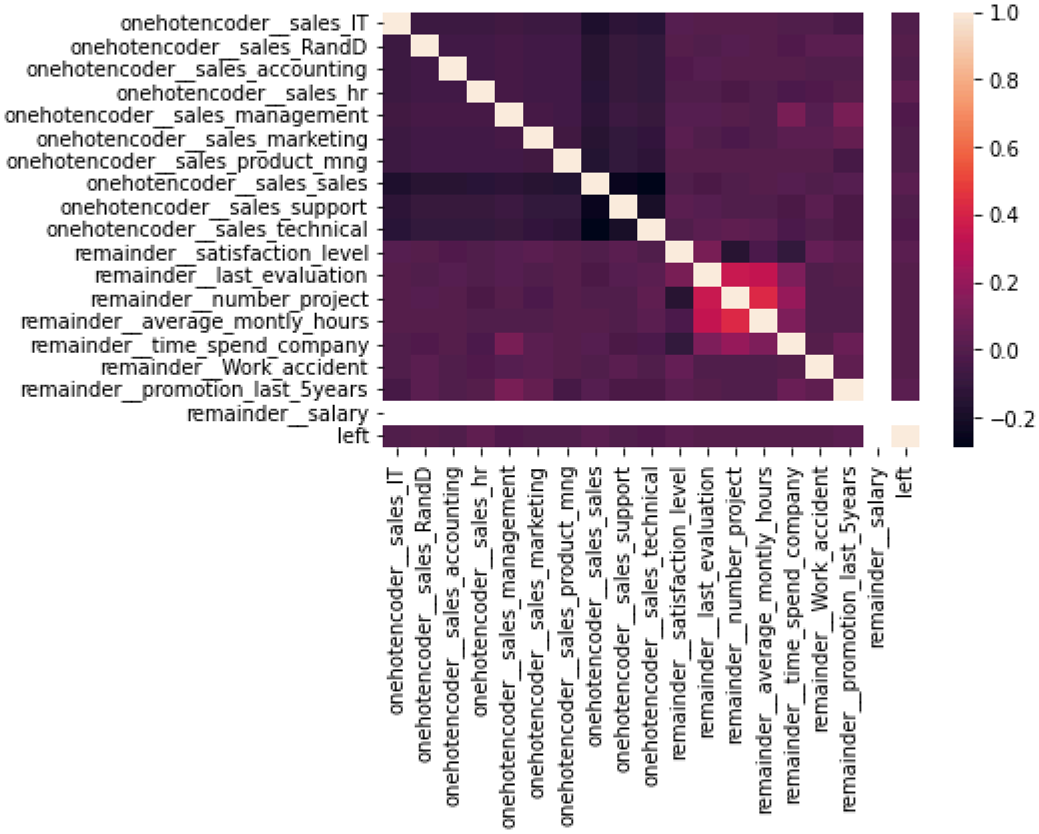
Out[109]:

```
<seaborn.axisgrid.PairGrid at 0x7ffa4f69de40>
```



```
In [110…  #Heat map plot

          sns.heatmap(X.corr(),
                      xticklabels = X.columns,
                      yticklabels = X.columns)
```

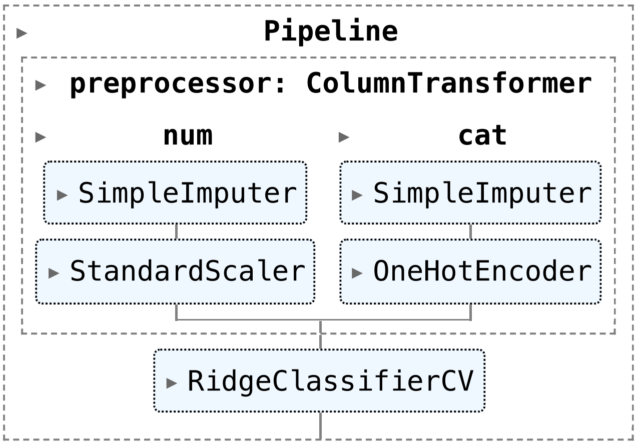
```
Out[110]: <AxesSubplot: >
```



In [111…

```
from sklearn.linear_model import RidgeClassifierCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer  # Import for handling missing valu
import random

# Define transformers
numeric_transformer = Pipeline(
  steps=[
      ('imputer', SimpleImputer(strategy='mean')),  # Handle missing values
      ('scaler', StandardScaler())                 # Scale numeric features
  ]
)

categorical_transformer = Pipeline(
  steps=[
      ('imputer', SimpleImputer(strategy='most_frequent')),  # Handle missin
      ('onehot', OneHotEncoder(handle_unknown='ignore'))     # Encode catego
  ]
)

# Combine transformers into a preprocessor
preprocessor = ColumnTransformer(
  transformers=[
      ('num', numeric_transformer, X_train.select_dtypes(include=['float64',
      ('cat', categorical_transformer, X_train.select_dtypes(include=['objec
  ]
)

# Define the model pipeline
model = Pipeline(
  steps=[
      ('preprocessor', preprocessor),
      ('model', RidgeClassifierCV(alphas=[random.uniform(0.1, 1) * 10**x for
    ]
)

# Fit the model
model.fit(X_train, y_train)
```

Out[111]:



Pipeline

preprocessor: ColumnTransformer

num

cat

SimpleImputer

SimpleImputer

StandardScaler

OneHotEncoder

RidgeClassifierCV

In [112…

```
model.fit(X_train,y=y_train)
model['model'].best_score_
```

Out[112]: 0.7771770885442721

In [113…

```
#A stronger model should be built
df.head(2)
```

Out[113]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_c
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	

In [114…

```
print(X_train.columns)
```

```
Index(['onehotencoder__sales_IT', 'onehotencoder__sales_RandD',
       'onehotencoder__sales_accounting', 'onehotencoder__sales_hr',
       'onehotencoder__sales_management', 'onehotencoder__sales_marketing',
       'onehotencoder__sales_product_mng', 'onehotencoder__sales_sales',
       'onehotencoder__sales_support', 'onehotencoder__sales_technical',
       'remainder__satisfaction_level', 'remainder__last_evaluation',
       'remainder__number_project', 'remainder__average_montly_hours',
       'remainder__time_spend_company', 'remainder__Work_accident',
       'remainder__promotion_last_5years', 'remainder__salary'],
      dtype='object')
```

In [125…

```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import RidgeClassifierCV
import random

# Define the preprocessor
preprocessor = ColumnTransformer(
 transformers=[
     # Use the prefixed column names for PolynomialFeatures
     ("poly", PolynomialFeatures(8), [
         'remainder__satisfaction_level',
         'remainder__last_evaluation',
         'remainder__average_montly_hours'
```

```
        ]),
        ("poly2", PolynomialFeatures(2), [
            'remainder__number_project',
            'remainder__time_spend_company'
        ]),
        # Apply StandardScaler to all columns in X_train
        ('scaler', StandardScaler(), X_train.columns)
    ]
)

# Define the pipeline
model = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("model", RidgeClassifierCV(alphas=[random.uniform(0.1, 1) * 10**x for
    ]
)


# Fit the model
model.fit(X_train, y_train)

# Access the best score
print(model['model'].best_score_)
```

```
0.9458947973986993
```

In [126…

```
print('Accuracy on test data:', model.score(X_test,y_test))
```

```
Accuracy on test data: 0.9422
```

In [127…

```
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV




#The model consists of the scaler and XGBClassifier
clf = Pipeline(
    steps=[('scaler', StandardScaler()),
            ("xgb", XGBClassifier(cv=5))
           ]
)

param_grid = {
    "xgb__n_estimators": [300],
    "xgb__learning_rate": [0.3],
    "xgb__eta":[0.1],
    "xgb__max_depth": [1],
    "xgb__objective":['reg:squarederror'],
    'xgb__min_child_weight': [4],
    'xgb__subsample': [0.7],
    'xgb__reg_lambda':[0.55]
}

search = GridSearchCV(clf, param_grid=param_grid, cv = 5, scoring='accuracy'

search.fit(X_train,y_train)
print(search.best_params_)
```

```
[08:48:34] WARNING: ../src/learner.cc:627:
Parameters: { "cv" } might not be used.

  This could be a false alarm, with some parameters getting used by languag
e bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actu
ally being used
  but getting flagged wrongly here. Please open an issue if you find any su
ch cases.


{'xgb__eta': 0.1, 'xgb__learning_rate': 0.3, 'xgb__max_depth': 1, 'xgb__min
_child_weight': 4, 'xgb__n_estimators': 300, 'xgb__objective': 'reg:squared
error', 'xgb__reg_lambda': 0.55, 'xgb__subsample': 0.7}
```

In [128…

```
search.best_score_
```

Out[128]:

```
0.9491942471235617
```

In [ ]:

```
#The model has been strengthened
```