google-advance-analytics-salifort-analysis

January 13, 2024

0.1 # Google Advance Analytics Capstone

1 Executive Summary

In the realm of business, addressing challenges effectively can lead to significant benefits in terms of time, cost savings, and employee well-being. Salifort, a prominent manufacturer of alternative energy vehicles, maintains a global workforce exceeding 100,000 employees. However, the company has been grappling with a considerable employee turnover rate, accompanied by alarmingly low levels of employee satisfaction.

To address this critical issue, Salifort seeks to proactively predict and understand employee turnover patterns. This analysis delves into the realm of machine learning, employing both supervised methods such as tree-based, ensemble, and boosting models, as well as unsupervised techniques like KMeans and DBScan. Among these, the Random Forest Classifier (ensemble) model stands out, achieving an impressive accuracy rate of approximately 95%.

Our findings reveal that key factors influencing employee departures include levels of job satisfaction, the number of projects employees are assigned, average monthly work hours, and the results of the employees' last performance evaluation. By focusing on these core areas, Salifort can take strategic steps to enhance employee retention and foster a more stable and contented workforce.

1.1 About the company

Salifort Motors is a fictional French-based alternative energy vehicle manufacturer. Its global workforce of over 100,000 employees research, design, construct, validate, and distribute electric, solar, algae, and hydrogen-based vehicles. Salifort's end-to-end vertical integration model has made it a global leader at the intersection of alternative energy and automobiles.

1.2 Business case

As a data specialist working for Salifort Motors, you have received the results of a recent employee survey. The senior leadership team has tasked you with analyzing the data to come up with ideas for how to increase employee retention. To help with this, they would like you to design a model that predicts whether an employee will leave the company based on their department, number of projects, average monthly hours, and any other data points you deem helpful.

1.3 Scenario

Currently, there is a high rate of turnover among Salifort employees. (Note: In this context, turnover data includes both employees who choose to quit their job and employees who are let go). Salifort's

senior leadership team is concerned about how many employees are leaving the company. Salifort strives to create a corporate culture that supports employee success and professional development.

> Further,the high turnover rate is costly in the financial sense. Salifort makes a big investment in recruiting, training, and upskilling its employees.

If Salifort could predict whether an employee will leave the company, and discover the reasons behind their departure, they could better understand the problem and develop a solution.

As a first step, the leadership team asks Human Resources to survey a sample of employees to learn more about what might be driving turnover.

Next, the leadership team asks you to analyze the survey data and come up with ideas for how to increase employee retention. To help with this, they suggest you design a model that predicts whether an employee will leave the company based on their job title, department, number of projects, average monthly hours, and any other relevant data points.

> A good model will help the company increase retention and job satisfaction for current employees, and save money and time training new employees.

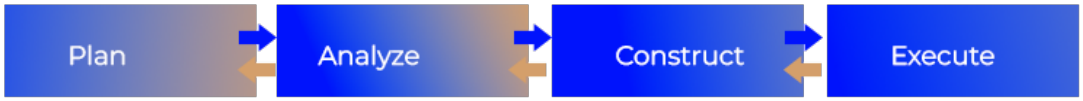
As a specialist in data analysis, the leadership team leaves it up to you to choose an approach for building the most effective model to predict employee departure.

For any approach, you'll need to analyze the key factors driving employee turnover, build an effective model, and share recommendations for next steps with the leadership team.

2 Implementing PACE Stage

[6]:
```
from IPython.display import Image
Image("assets/img/pace_framework.png")
```

[6]:



The project will implement the PACE framework for this analysis.

2.1 # Plan

This project aims to solve the business needs of Salifort by exploring the data and building a machine learning model that will be able to predict employees who are likely to exit. The increasing employee turnover marks as a problem as hiring and interviewing candidates can be expensive. Being in a such a competitive industry, identifying the factors leading to employee turnover will help the Salifort in the long run.

2.2 Feature Description

Column Name	Type	Description
satisfaction_level	int64	The employee's self-reported satisfaction level [0-1]
last_evaluation	int64	Score of employee's last performance review [0–1]
number_project	int64	Number of projects employee contributes to
average_monthly_hours	int64	Average number of hours employee worked per month
time_spend_company	int64	How long the employee has been with the company (years)
work_accident	int64	Whether or not the employee experienced an accident while at work
left	int64	Whether or not the employee left the company
promotion_last_5years	int64	Whether or not the employee was promoted in the last 5 years
department	str	The employee's department
salary	str	The employee's salary (low, medium, or high)

[1]:
```
!python --version
```

```
Python 3.11.5
```

Install requirements below to run notebook. Private Utility scripts are used for plotting .

```
!pip install plotnine pyjanitor==0.19 patchworklib --quiet
```

[2]:
```
# # requirements
# !pip install --upgrade plotnine patchworklib --quiet
# !pip install pyjanitor==0.19 --quiet
```

3 Analyze

3.1 Exploratory Data Analysis

The project will conduct an exploratory data analysis on our dataset. That includes cleaning, wrangling, and validation.

Initial discovery: - The data consist of 14,999 entries relating to the employee's background, performance, and overall satisfaction during their stay. - The entries in the dataset does not have any missing values. - That entries in the dataset does have duplicated features and will be removed from the analysis.

The project will use Python Programming Language version 3.11 for this analysis. And also the following modules: - for wrangling and cleaning libraries pandas, and pyjanitor. - for visualziation seaborn, plotnine, matplotlib

[3]:
```
from pathlib import Path
import pandas as pd
import numpy as np
import janitor
import janitor.ml
from plotnine import *
import seaborn as sns
import matplotlib.pyplot as plt

# utility script
from gp_utility.hr_script_plot import\
    plot_cat_features, plot_num_float_features,\
    plot_discrete_features, plot_binary_features_promotion,\
    plot_binary_features_accident

# module settings (pandas, seaborn, and matplotlib)
pd.options.display.max_columns = 100
sns.set(font_scale=0.8)
%matplotlib inline

# colar pallette for chart
color_pal = ['#5c86ff', '#db0049']


def read_hr_data() -> pd.DataFrame:
    """
    :description: Read dataframe and clean using pyjanitor
    :returns: Clean DataFrame of the Data
    """

    # filename
    file_name = 'salifort_hr_data.csv'
    data_path = Path('data') / file_name

    # load dataframe and clean columns
    df = pd.read_csv(data_path) \
        .clean_names() \
        .rename_columns({
        'left': 'exit',
        'average_montly_hours': 'average_monthly_hours'})

    return df
```

```
<Figure size 100x100 with 0 Axes>
```

[4]:
```
# use read hr function to load and clean data
df_original = read_hr_data()

# set a copy for wrangling purposes
df = df_original.copy()
```

[5]:
```
# check dataset information
df.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_monthly_hours  14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   work_accident          14999 non-null  int64
 6   exit                   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
```

[6]:
```
# check feature datatypes
df.dtypes
```

[6]:
```
satisfaction_level       float64
last_evaluation          float64
number_project             int64
average_monthly_hours      int64
time_spend_company         int64
work_accident              int64
exit                       int64
promotion_last_5years      int64
department                object
salary                    object
dtype: object
```

3.2 Missing

[7]:
```
# check for missing entries in all of our features
df.isna().sum()
```

[7]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_monthly_hours    0
time_spend_company       0
work_accident            0
exit                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

3.3 Duplicated

[8]:
```
# check for duplicate entries
duplicated_entries = df[df.duplicated()]
duplicated_entries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3008 entries, 396 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     3008 non-null   float64
 1   last_evaluation        3008 non-null   float64
 2   number_project         3008 non-null   int64
 3   average_monthly_hours  3008 non-null   int64
 4   time_spend_company     3008 non-null   int64
 5   work_accident          3008 non-null   int64
 6   exit                   3008 non-null   int64
 7   promotion_last_5years  3008 non-null   int64
 8   department             3008 non-null   object
 9   salary                 3008 non-null   object
dtypes: float64(2), int64(6), object(2)
memory usage: 258.5+ KB
```

[9]:
```
# original data department value counts
df.department.value_counts(normalize=True) * 100
```

[9]:
```
department
sales          27.601840
technical      18.134542
support        14.860991
IT              8.180545
product_mng     6.013734
marketing       5.720381
RandD           5.247016
accounting      5.113674
```

```
hr              4.926995
management      4.200280
Name: proportion, dtype: float64
```

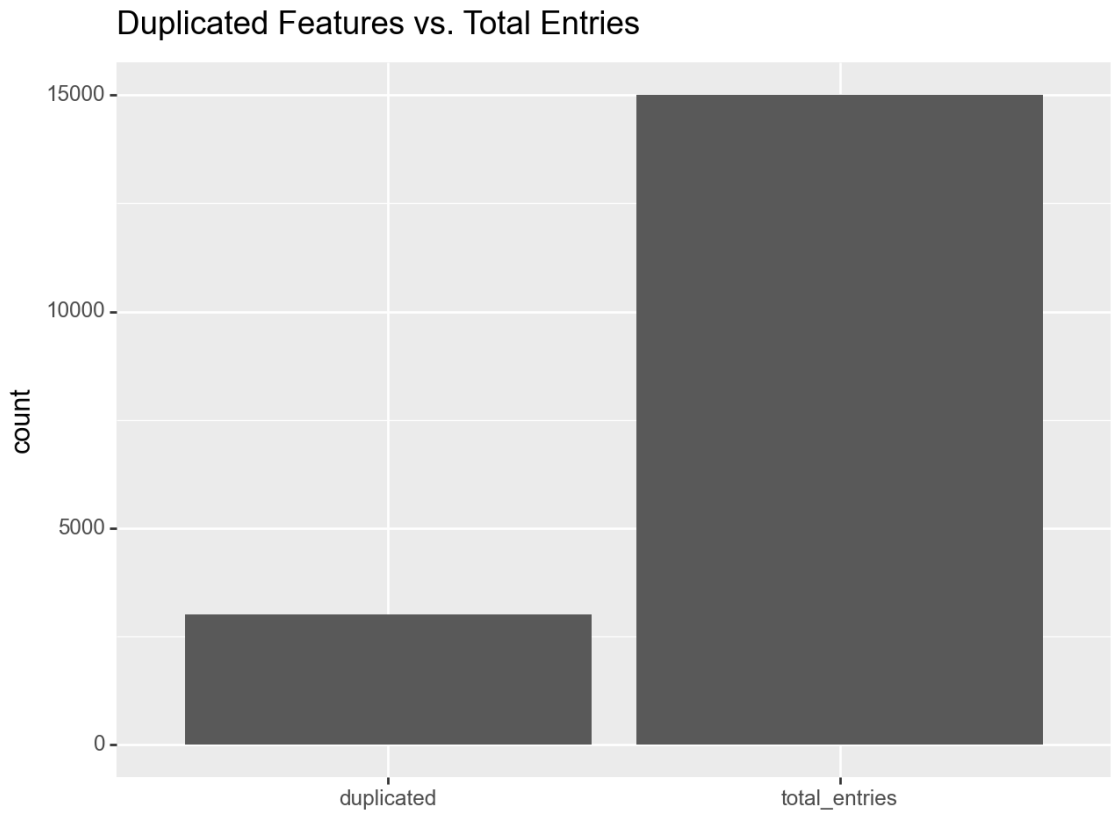
[10]:
```
# duplicated entries department value counts
duplicated_entries.department.value_counts(normalize=True) * 100
```

[10]:
```
department
sales          29.953457
technical      15.824468
support        13.563830
IT              8.344415
product_mng     7.180851
management      6.449468
marketing       6.150266
accounting      4.853723
hr              4.587766
RandD           3.091755
Name: proportion, dtype: float64
```

3.3.1 Investigate Duplicated Entries

[11]:
```
duplicated_data_summary = {'total_entries': [df.shape[0]], 'duplicated':␣
↪[duplicated_entries.shape[0]]}
duplicated_viz = pd.melt(pd.DataFrame.from_dict(duplicated_data_summary))

duplicated_plot = (ggplot(duplicated_viz)\
    + geom_col(aes(x='variable', y='value'), position='stack')\
    + labs(y='count', x='', title='Duplicated Features vs. Total Entries')).
↪draw();

duplicated_plot
```

[11]:



[12]:
```
duplicated_entries.exit.value_counts()
```

[12]:
```
exit
1    1580
0    1428
Name: count, dtype: int64
```

[13]:
```
np.round(duplicated_entries.department.value_counts(normalize=True) * 100, 1)
```

[13]:
```
department
sales          30.0
technical      15.8
support        13.6
IT              8.3
product_mng     7.2
management      6.4
marketing       6.2
accounting      4.9
hr              4.6
RandD           3.1
Name: proportion, dtype: float64
```

3.4 ### Drop Decision

Duplicated features may be a result of technical error during the collation and will be removed from this analysis. Further questions & inquiry regarding the duplicated entries can be forwarded to salifort data & technical team.

[14]:
```
# drop duplicated features
df = df.drop_duplicates(keep='first')
```

3.5 Object Features

[15]:
```
df.select_dtypes('object').describe()
```

[15]:

	department	salary
count	11991	11991
unique	10	3
top	sales	low
freq	3239	5740

[16]:
```
df['exit'].value_counts(normalize=True) * 100
```

[16]:
```
exit
0    83.39588
1    16.60412
Name: proportion, dtype: float64
```

[17]:
```
plot_salary, plot_department = plot_cat_features(df)
```

3.5.1 Exit based on Low, Medium, and High Salary

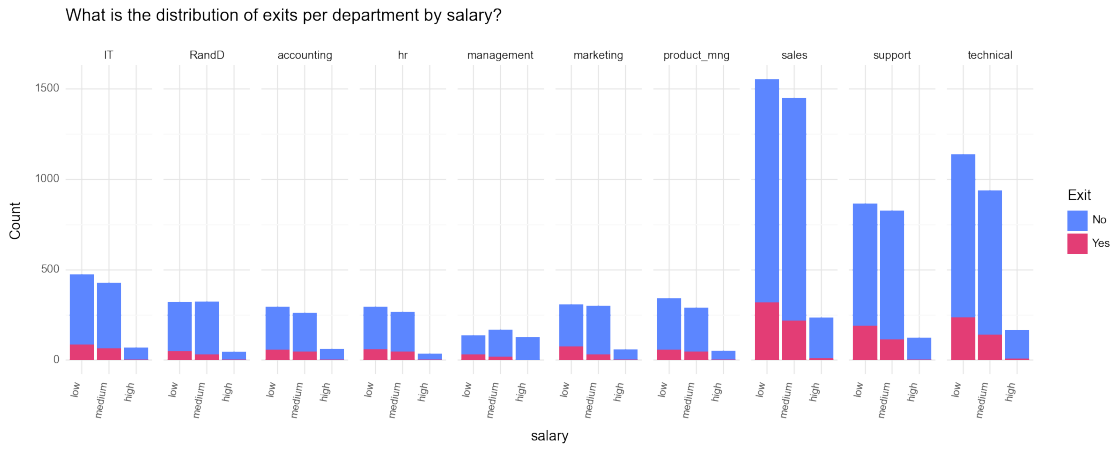
[18]:
```
plot_salary.draw()
```

[18]:

3.5.2 Count of employees leaving their work based on salary

[19]:
```
plot_department.draw()
```

[19]:



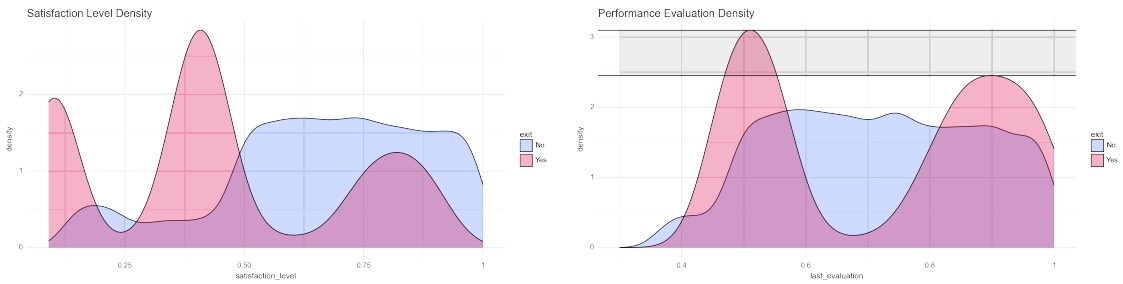
3.6 Numerical Features

3.6.1 Satisfaction Level & Performance Evaluation KDE

[20]:
```
float_features = df.select_dtypes('float').columns.to_list()
float_features.append('exit')
float_feat_df = df[float_features]
float_feat_df.exit = float_feat_df.exit.astype('str')
```

[21]:
```
plot_num_float_features(float_feat_df, color_pal)
```

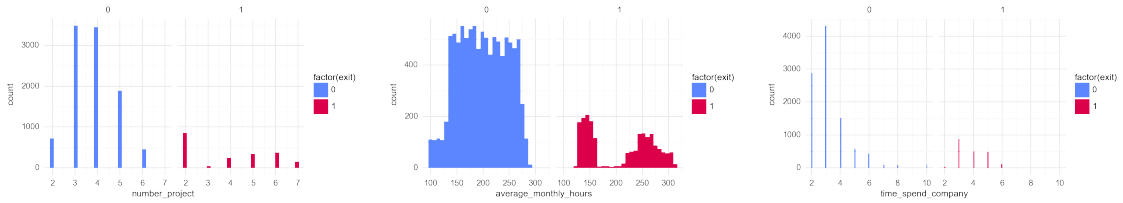
[21]:



3.6.2 Discrete Features Distribution by Factor of Exit

[22]:
```
# discrete features
int_features = df.select_dtypes('integer').reorder_columns(['exit'])
discrete_features = int_features[['number_project', 'average_monthly_hours',␣
↪'time_spend_company', 'exit']]
plot_discrete_features(discrete_features, color_pal)
```

[22]:



3.7 Binary Features

[23]:
```
int_features.work_accident.value_counts(normalize=True) * 100
```

[23]:
```
work_accident
0    84.571762
1    15.428238
Name: proportion, dtype: float64
```

[24]:
```
int_features.promotion_last_5years.value_counts(normalize=True) * 100
```

[24]:
```
promotion_last_5years
0    98.307064
1     1.692936
Name: proportion, dtype: float64
```

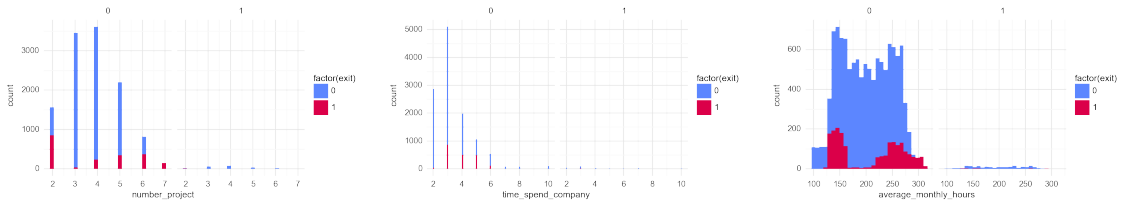
[25]:
```
int_features.exit.value_counts(normalize=True) * 100
```

[25]:
```
exit
0    83.39588
1    16.60412
Name: proportion, dtype: float64
```

3.7.1 Promotion

[26]:
```
plot_binary_features_promotion(int_features, color_pal)
```

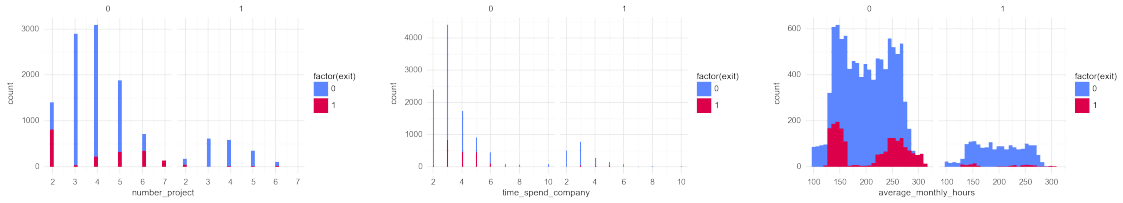
[26]:



3.7.2 Work Accident

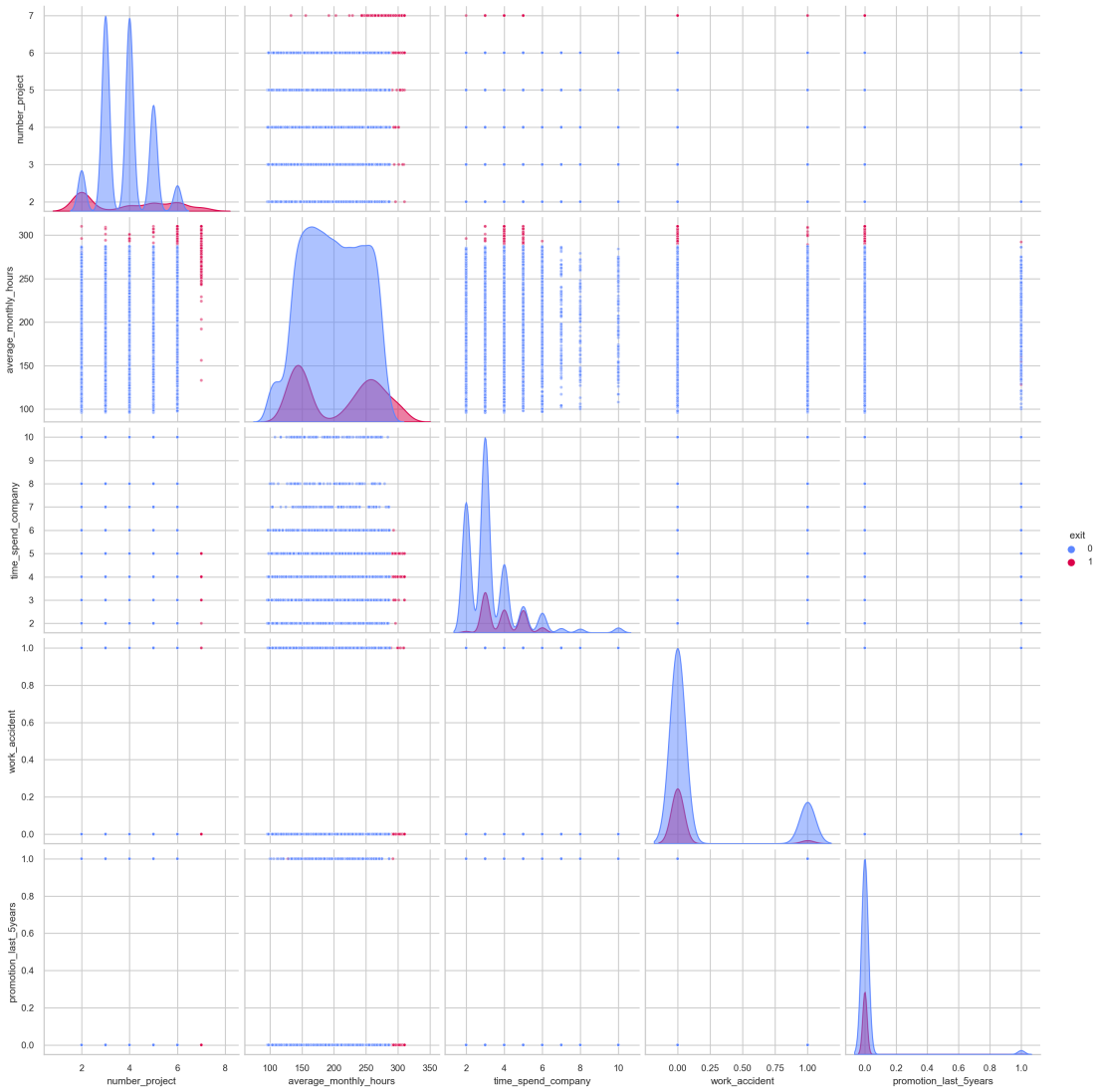
[27]:
```
plot_binary_features_accident(int_features, color_pal)
```

[27]:



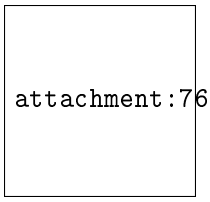
3.8 Scatter Matrix

[28]:
```
sns.set_style("whitegrid")
mat = sns.pairplot(int_features, hue='exit', palette=color_pal, markers='.',␣
 ↪diag_kws={'alpha': 0.5}, plot_kws={'alpha': 0.5})
mat.fig.set_size_inches(15, 15)
```



3.9 Unsupervised Machine Learning

[29]:
```
from typing import Iterator

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.metrics import silhouette_score
import patchworklib as pw
```



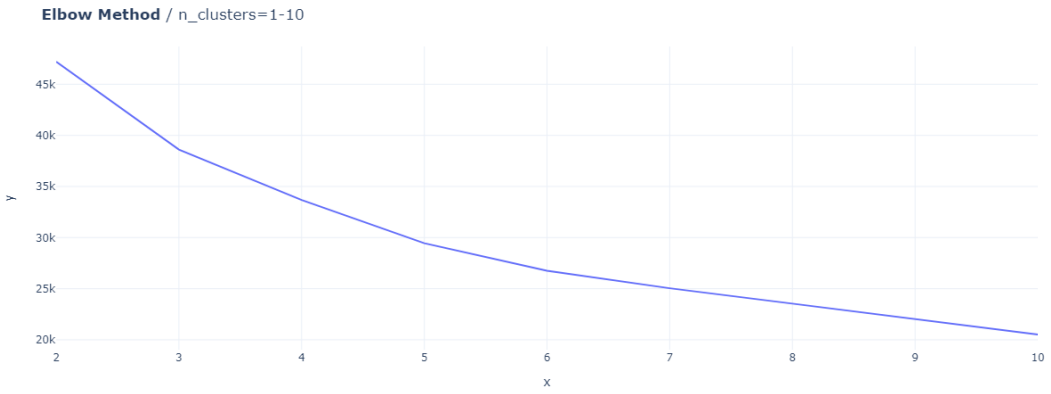
3.9.1 KMeans

[31]:
```
def kmeans_inertia(num_clusters, x_vals) -> list:

    inertia = []
    for num in num_clusters:
        kms = KMeans(n_clusters=num, random_state=42, n_init='auto')
        kms.fit(x_vals)
        inertia.append(kms.inertia_)

    return inertia

def kmeans_sil(num_clusters, x_vals) -> list:

    sil_score = []
    for num in num_clusters:
        kms = KMeans(n_clusters=num, random_state=42)
        kms.fit(x_vals)
        sil_score.append(silhouette_score(x_vals, kms.labels_))

    return sil_score

def plot_kmean_feat(x_feat) -> ggplot:

    values = ['#5c86ff', '#db0049', "#EBD323"]
    g_plot = (ggplot(scaled_important_features)\
        + geom_point(aes(x=x_feat, y='satisfaction_level',␣
↪fill='factor(clusters)'), alpha=0.3)\
        + scale_fill_manual(values=values, labels=[0, 1, 2])\
        + theme_minimal())
    return g_plot

def save_kmean_3_cluster_plots() -> Iterator[ggplot]:

    x_features = ['last_evaluation', 'average_monthly_hours', 'number_project',␣
↪'time_spend_company']
    values = ['#FFA500', '#9467bd', '#008080']

    for feat in x_features:
        g = ggplot(scaled_important_features)\
```

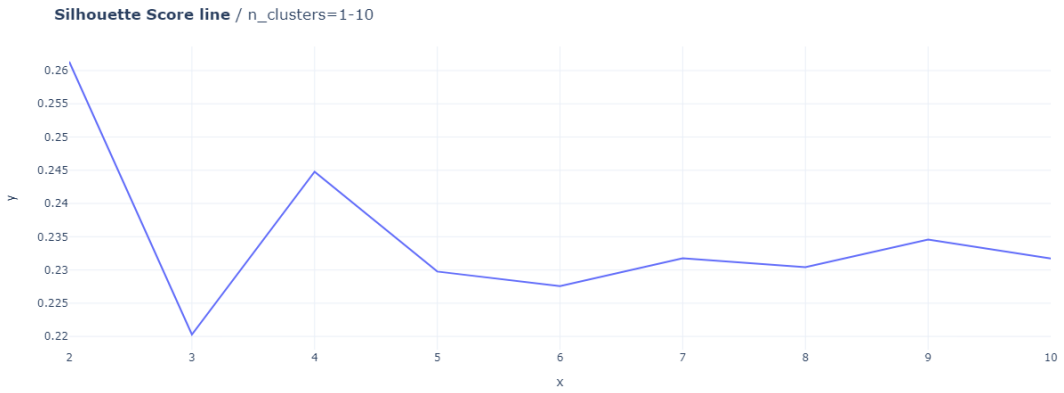
```
            + geom_point(aes(x=feat, y='satisfaction_level',␣
↪fill='factor(clusters)'), alpha=0.3)\
            + scale_fill_manual(values=values, labels=[0, 1, 2])\
            + theme_minimal()

        yield g
```

[32]:
```
# select important features from ensemble models
important_features = df.select_dtypes('number').iloc[:, :-3]
scaled_features = StandardScaler().fit_transform(important_features)
scaled_important_features = pd.DataFrame(scaled_features,␣
↪columns=important_features.columns)

# use elbow method to decide for number of clusters
num_clusters = [i for i in range(2, 11)]
inertia = kmeans_inertia(num_clusters, scaled_important_features)
```

[33]:
```
# create an inertia line plot using plotly express
inertia_plot = px.line(x=num_clusters, y=inertia)\
    .update_layout(template='plotly_white', width=800, height=480,
                   title='<b>Elbow Method</b> / n_clusters=1-10')\
    .update_traces(marker={'color': 'black'})

inertia_plot
```



[34]:
```
# create a silhouette score line plot using plotly express
sil_score = kmeans_sil(num_clusters, scaled_important_features)
sil_plot = px.line(x=num_clusters, y=sil_score)\
    .update_layout(template='plotly_white', width=800, height=480,
                   title='<b>Silhouette Score line</b> / n_clusters=1-10')\

sil_plot
```

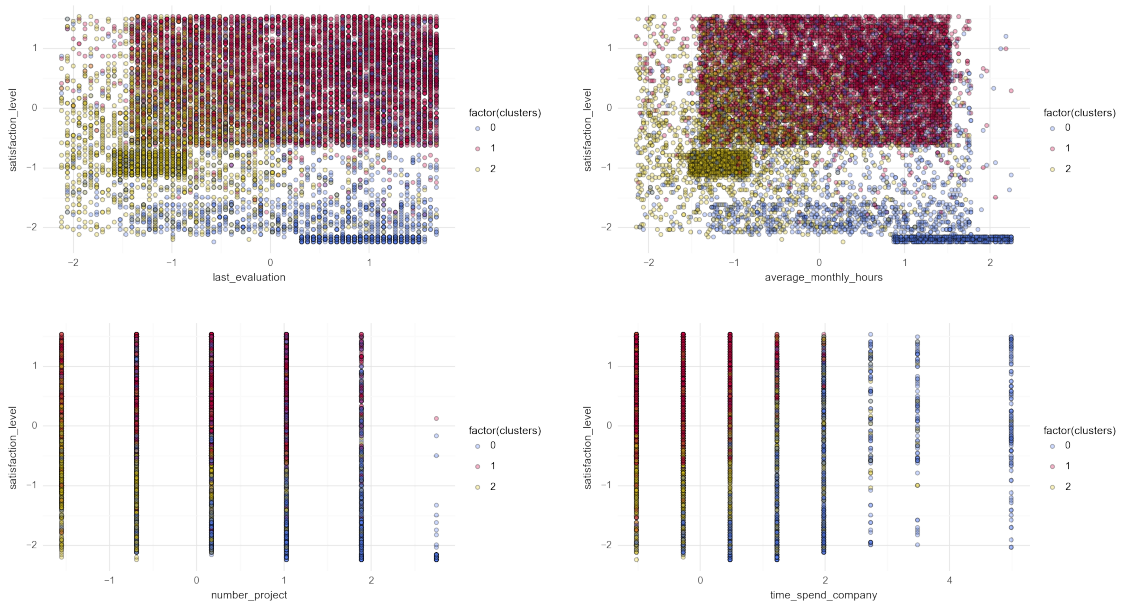


[35]:
```
# KMeans clusters of 2 (0, 1, 2)
km3 = KMeans(n_clusters=3, random_state=4).fit(scaled_important_features)
scaled_important_features['clusters'] = km3.labels_

# patch plots from kmeans output
feat_to_plot = ["last_evaluation", "average_monthly_hours", "number_project",␣
↪"time_spend_company"]
plot_collection = [plot_kmean_feat(i) for i in feat_to_plot]
plot_g = [pw.load_ggplot(plot, figsize=(5,3)) for plot in plot_collection]
plot_mat_g = ((plot_g[0]|plot_g[1])/(plot_g[2]|plot_g[3]))
```

[36]:
```
# save kmeans plot to pdf
save_as_pdf_pages(save_kmean_3_cluster_plots(), path='viz_output',␣
↪filename='kmeans_cluster.pdf')
```

[37]:
```
plot_mat_g
```

[37]:

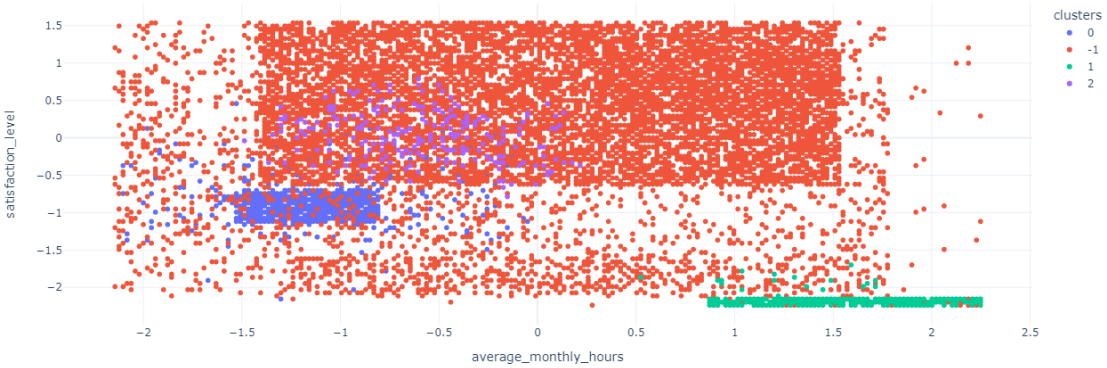


3.9.2 DBScan

[38]:
```
from sklearn.cluster import DBSCAN
```

[39]:
```
dbs = DBSCAN(eps=.90, min_samples=290).fit(scaled_important_features)
scaled_important_features['clusters'] = dbs.labels_
scaled_important_features['clusters'] = scaled_important_features['clusters'].
 ↪astype('str')
```

[40]:
```
px.scatter(scaled_important_features, x='average_monthly_hours',␣
 ↪y='satisfaction_level', color='clusters')\
    .update_layout(height=500, width=800, template='plotly_white')\
    .update_traces(marker=dict(opacity=1))
```



4 Construct

[41]:
```
import pickle
import os

from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import plot_tree, DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import precision_score, accuracy_score, recall_score,\
     f1_score, confusion_matrix, classification_report, silhouette_score

from xgboost import XGBClassifier
from xgboost import plot_importance

import pandas_flavor as pf

from gp_utility.cf_matrix import make_confusion_matrix
import gp_utility.project_ml as pml


M_RN_STATE = 5
```

[42]:
```
hr = read_hr_data()
# report = sv.analyze(hr)
# report.show_html()

# set copy of original data
original_df = hr.copy(deep=True)
df = original_df.drop_duplicates()
```

[43]:
```
df = df\
    .clean_names()\
    .encode_categorical(column_names=['department', 'salary'])
```

4.1 Model Preparation

[44]:
```
# selecting features DV, IV
X, y = df.get_features_targets(target_column_names='exit')

# store different types in variables
numerical_features = X.select_dtypes('number')
categorical_features = X.select_dtypes('category')

# store column names
num_feat_names = numerical_features.columns.to_list()
cat_feat_names = categorical_features.columns.to_list()

# make column transformer
transformer = make_column_transformer(
    (StandardScaler(), num_feat_names),
    (OneHotEncoder(sparse=False), cat_feat_names))

# fit features into transformer
fit_ct = transformer.fit(X)
X_transformed = fit_ct.transform(X)

# get ohe featurenames
ohe_col_names = fit_ct.named_transformers_['onehotencoder']\
    .get_feature_names_out()

# add ohe feature names to num feature names
all_feature_names = num_feat_names
all_feature_names.extend(list(ohe_col_names))

# create finished dataframe
X_CT = pd.DataFrame(X_transformed, columns=all_feature_names)

# # Fixed variables for our models
TRAIN_SPLIT = 0.75
M_RN_STATE = 2

# train test split
X_train, X_test, y_train, y_test = train_test_split(
    X_CT, y,
    train_size=TRAIN_SPLIT,
    random_state=M_RN_STATE)
```

4.2 Feature Selection

[45]:
```
# creeate a classification dict
classification_model = {
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=M_RN_STATE),
    'RandomForestClassifier' : RandomForestClassifier(random_state=M_RN_STATE),
    'SupportVectorClassifier': SVC(random_state=M_RN_STATE),
    'LogisticRegression': LogisticRegression(random_state=M_RN_STATE)}
```

[46]:
```
# create an empty feature importance dict
m_feature_importance = dict()
pml.train_models(
    classification_model,
    [X_train, X_test, y_train, y_test],
    m_feature_importance,
    all_feature_names)
```

```
        Model: DecisionTreeClassifier --------------------
        f1-score: 89.91
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2493
           1       0.89      0.91      0.90       505

    accuracy                           0.97      2998
   macro avg       0.94      0.94      0.94      2998
weighted avg       0.97      0.97      0.97      2998


        Model: RandomForestClassifier --------------------
        f1-score: 94.85
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      2493
           1       0.99      0.91      0.95       505

    accuracy                           0.98      2998
   macro avg       0.99      0.95      0.97      2998
weighted avg       0.98      0.98      0.98      2998


        Model: SupportVectorClassifier --------------------
        f1-score: 90.13
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2493
           1       0.91      0.90      0.90       505
```

```
    accuracy                           0.97      2998
   macro avg       0.94      0.94      0.94      2998
weighted avg       0.97      0.97      0.97      2998

coef_ is only available when using a linear kernel

        Model: LogisticRegression --------------------
        f1-score: 29.65
              precision    recall  f1-score   support

           0       0.86      0.96      0.90      2493
           1       0.50      0.21      0.30       505

    accuracy                           0.83      2998
   macro avg       0.68      0.58      0.60      2998
weighted avg       0.80      0.83      0.80      2998
```

[47]:
```
# transform feature importance data into dataframe
df_importance = [pd.DataFrame(data)
    for data in pml.get_table_feature_importance(m_feature_importance)]

# create a table of feature importance with classification anmes
table_feat_importance = [frame.set_index('feature')\
        .rename_columns({'importance': f'{name}'})\
        for frame, name in zip(df_importance, classification_model.keys())]

# style the table values
table_feat_importance = pd.concat(
    table_feat_importance, axis=1)\
    .style.background_gradient(cmap='viridis')

# export data as html
table_feat_importance.to_html('viz_output/feat_importance.html')
```

Synthesis The Logistic and SVM models chose similar features in importance. Both assigned a low salary as the main predictor for an exit. In comparison to the tree and ensemble model, both assigned satisfaction level as the likely predictors of exit.

Decision Tree and RandomForest gave little importance to wether which department an employee belongs or the level of their salary. The list of features that are of high importance of both models are of the following:

What features are important for both Tree & Ensemble Models? * Satisfaction level * Last performance evaluation * Number of projects * Time spent in the company * Average monthly hours

4.3 Boosting Models

[48]:
```
@pf.register_dataframe_method
def get_base_metrics(df, y_test, y_pred) -> pd.DataFrame:
    """
    :description:
        ...
    :y_test: Test data from training split
    :y_test type: array
    :y_pred: Predict data from model
    :y_pred type: array
    :returns: Dataframe with model metrics score.
    """
    return df.assign(base_f1_score=[f1_score(y_test, y_pred)])\
        .assign(base_recall_score=[recall_score(y_test, y_pred)])\
        .assign(base_precision_score=[precision_score(y_test, y_pred)])\
        .assign(base_accuracy_score=[accuracy_score(y_test, y_pred)])
```

4.3.1 XGBoost

[49]:
```
xgb = XGBClassifier(objective='binary:logistic', random_state=M_RN_STATE)
xgb.fit(X_train, y_train)
```

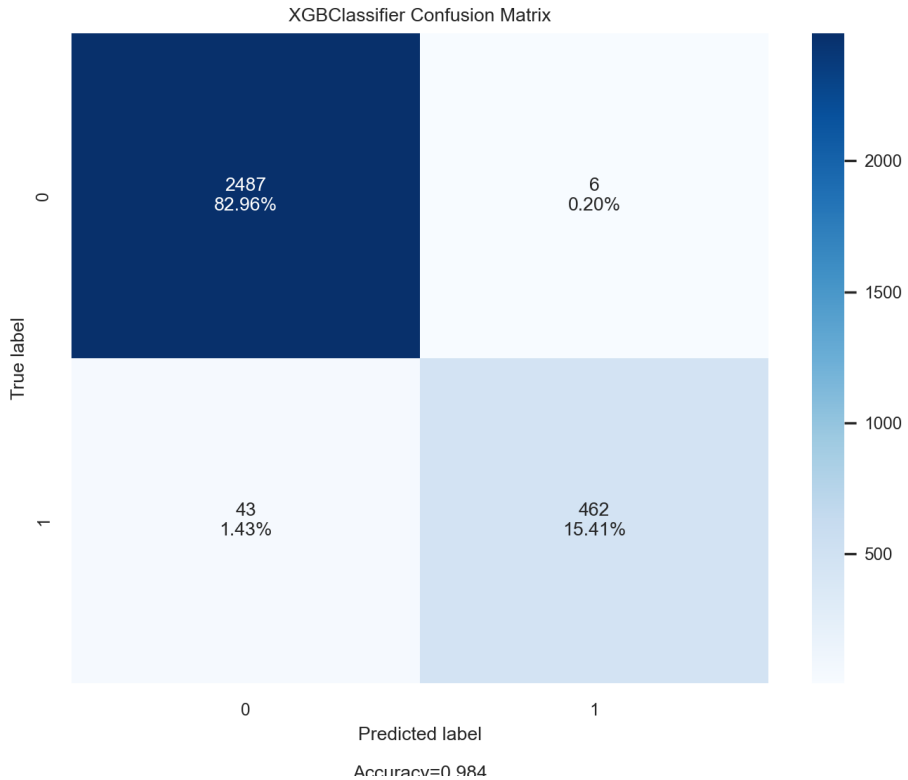
[49]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=2, ...)
```

[50]:
```
# predict test data
y_pred = xgb.predict(X_test)

# store score in dataframe
base_model_score = pd.DataFrame()
base_model_score.get_base_metrics(y_test, y_pred)
```

[50]:

	base_f1_score	base_recall_score	base_precision_score	base_accuracy_score
0	0.94964	0.914851	0.987179	0.983656

[51]:
```
cm = confusion_matrix(y_test, y_pred, labels=xgb.classes_)
make_confusion_matrix(cm, figsize=(8,6),
    title='XGBClassifier Confusion Matrix')
```



4.3.2 AdaBoost

[52]:
```
ad = AdaBoostClassifier(random_state=M_RN_STATE)
ad.fit(X_train, y_train)
```

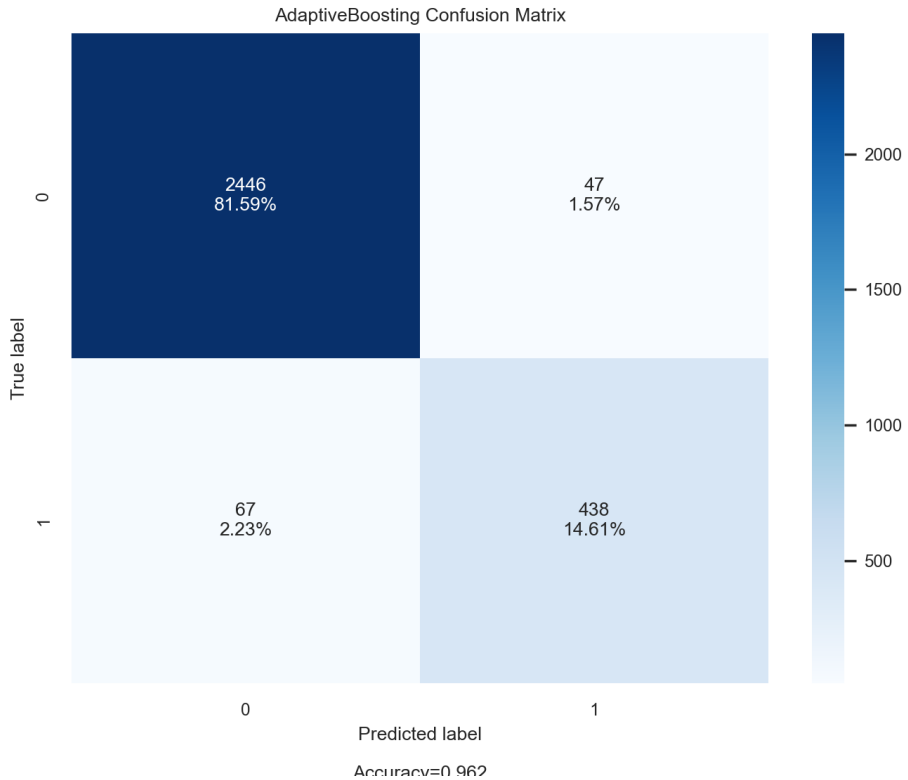
[52]:
```
AdaBoostClassifier(random_state=2)
```

[53]:
```
ada_boost_metrics = pd.DataFrame()
ada_y_pred = ad.predict(X_test)
ada_boost_metrics.get_base_metrics(y_test, ada_y_pred)
```

[53]:

	base_f1_score	base_recall_score	base_precision_score	base_accuracy_score
0	0.884848	0.867327	0.903093	0.961975

[54]:
```
cm = confusion_matrix(y_test, ada_y_pred, labels=ad.classes_)
make_confusion_matrix(cm, figsize=(8,6),
    title='AdaptiveBoosting Confusion Matrix')
```



4.3.3 ROC - AUC

[55]:
```
transformed_list = [X_train, X_test, y_train, y_test]

# create a classification dict
classification_model = {
    'DecisionTreeClassifier' : DecisionTreeClassifier(
        random_state=M_RN_STATE),
    'RandomForestClassifier' : RandomForestClassifier(
        random_state=M_RN_STATE),
    'AdaptiveBoost': AdaBoostClassifier(
        random_state=M_RN_STATE),
    'XGBoostClassifier': XGBClassifier(
        random_state=M_RN_STATE)}
```

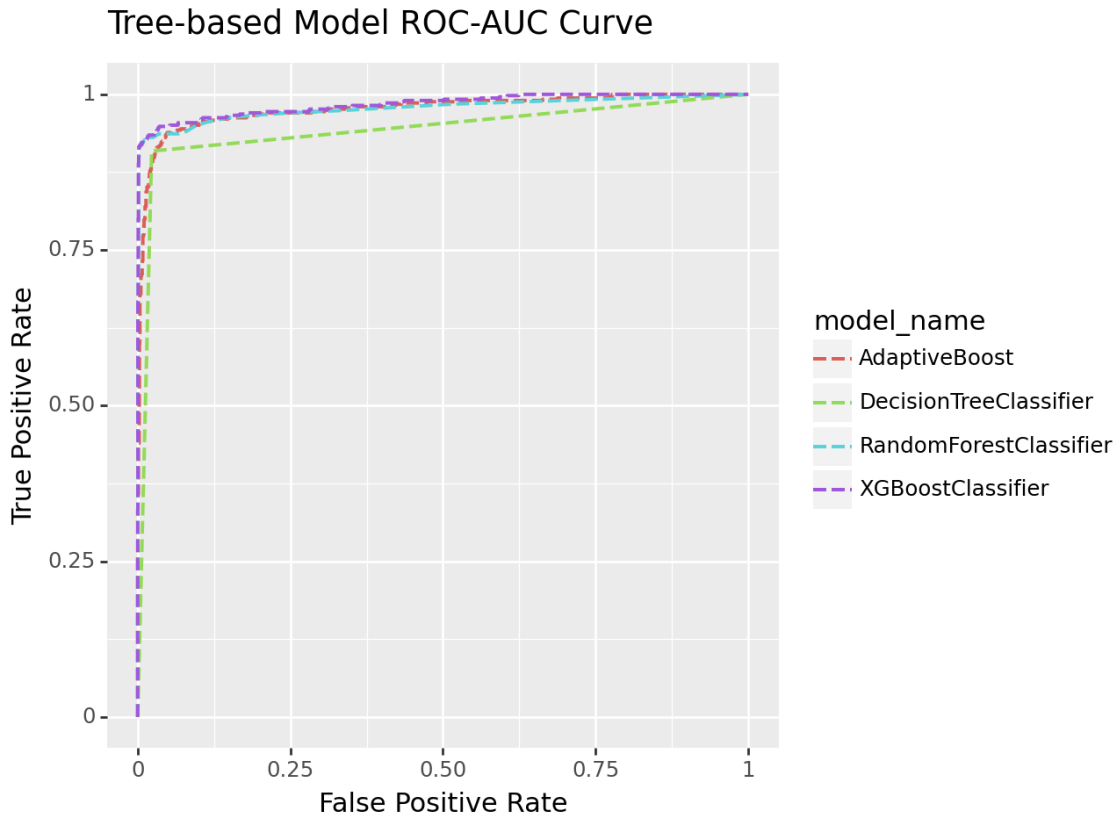
[56]:
```
clm_auc_roc_score = {k: '' for k in classification_model.keys()}
```

[57]:
```
# prepare model dataframe probability for roc auc

roc_auc_dt = pml.get_roc_auc(y_test, 'DecisionTreeClassifier',
    transformed_list, clm_auc_roc_score, classification_model)
```

```
roc_auc_rfc = pml.get_roc_auc(y_test, 'RandomForestClassifier',
    transformed_list, clm_auc_roc_score, classification_model)

roc_auc_ada = pml.get_roc_auc(y_test, 'AdaptiveBoost',
    transformed_list, clm_auc_roc_score, classification_model)

roc_auc_xgb = pml.get_roc_auc(y_test, 'XGBoostClassifier',
    transformed_list, clm_auc_roc_score, classification_model)
```

[58]:
```
# concat dataframes for plotting
roc_auc_all_models = [roc_auc_dt, roc_auc_rfc, roc_auc_ada, roc_auc_xgb]
roc_auc_all_models_df = pd.concat(roc_auc_all_models, axis=0)
```

[59]:
```
ggplot() + geom_line(roc_auc_all_models_df,
    aes(x='fpr', y='tpr', color='model_name'),size=0.8, linetype='dashed')\
    + ggtitle('Tree-based Model ROC-AUC Curve')\
    + labs(x='False Positive Rate', y='True Positive Rate')
```



[59]: <Figure Size: (640 x 480)>

The tree based models are performing well in our ROC-AUC curve. On the other hand, the decision-tree classifier performing less than the other models and will not proceed to tuning.

The models that we'll be using for tuning hyperparameters

- Random Forest Classifier
- XGBoost
- Adaptive Boost

And compare their performance in unknown data (f1-score).

4.4 GridSearch CV

4.4.1 XGBoost

[60]:
```
# xgboost parameters
params = {
    'n_estimators': [200, 350, 500, 800],
    'learning_rate': [0.01, 0.15, 0.3],
    'max_depth': [3, 5],
    'min_child_weight': [3, 5, 10],
    'subsample': [0.3, 0.7],
    'alpha': [1, 2]
}

# scoring
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# xgboost model
xgbc = XGBClassifier(random_state=M_RN_STATE)
```

[61]:
```
# gridsearch for xgbc
gcv_xgboost = GridSearchCV(
    xgbc,
    params,
    scoring=scoring,
    refit='f1')
```

[62]:
```
%%time
xgb_file_path = 'model/xgbcv.pickle'
if not os.path.exists(xgb_file_path):

    gcv_xgboost.fit(X_train, y_train)

    with open('model/xgbcv.pickle', 'wb') as f:
        pickle.dump(gcv_xgboost, f)

else:
    with open('model/xgbcv.pickle', 'rb') as f:
```

```
        gs_xgboost = pickle.load(f)
```

```
CPU times: total: 93.8 ms
Wall time: 42 ms
```

[63]:
```
pd.DataFrame().from_dict([gs_xgboost.best_params_])
```

[63]:

	alpha	learning_rate	max_depth	min_child_weight	n_estimators	subsample
0	2	0.01	5	3	800	0.7

[64]:
```
gs_xgboost.best_score_
```

[64]:
```
0.9414281714476669
```

4.4.2 Random Forest

[65]:
```
params = {
    'max_depth': [5, 6, 7],
    'max_features': [1.0],
    'max_samples': [0.7],
    'min_samples_leaf': [1,2],
    'min_samples_split': [2,3],
    'n_estimators': [50, 100],
    }

# scoring
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

# xgboost model
rfc = RandomForestClassifier(random_state=M_RN_STATE)
```

[66]:
```
# gridsearch for xgbc
gcv_rfc = GridSearchCV(
    rfc,
    params,
    scoring=scoring,
    refit='f1')
```

[67]:
```
%%time

rfc_file_path = 'model/rfc.pickle'
if not os.path.exists(rfc_file_path):

    gcv_rfc.fit(X_train, y_train)

    with open(rfc_file_path, 'wb') as f:
        pickle.dump(gcv_rfc, f)
else:
```

```
    with open(rfc_file_path, 'rb') as f:
        gs_random_forest = pickle.load(f)
```

```
CPU times: total: 109 ms
Wall time: 18 ms
```

[68]:
```
pd.DataFrame.from_dict([gs_random_forest.best_params_])
```

[68]:

	max_depth	max_features	max_samples	min_samples_leaf	min_samples_split	n_estimators
0	7	1.0	0.7	1	2	100

[69]:
```
gs_random_forest.best_score_
```

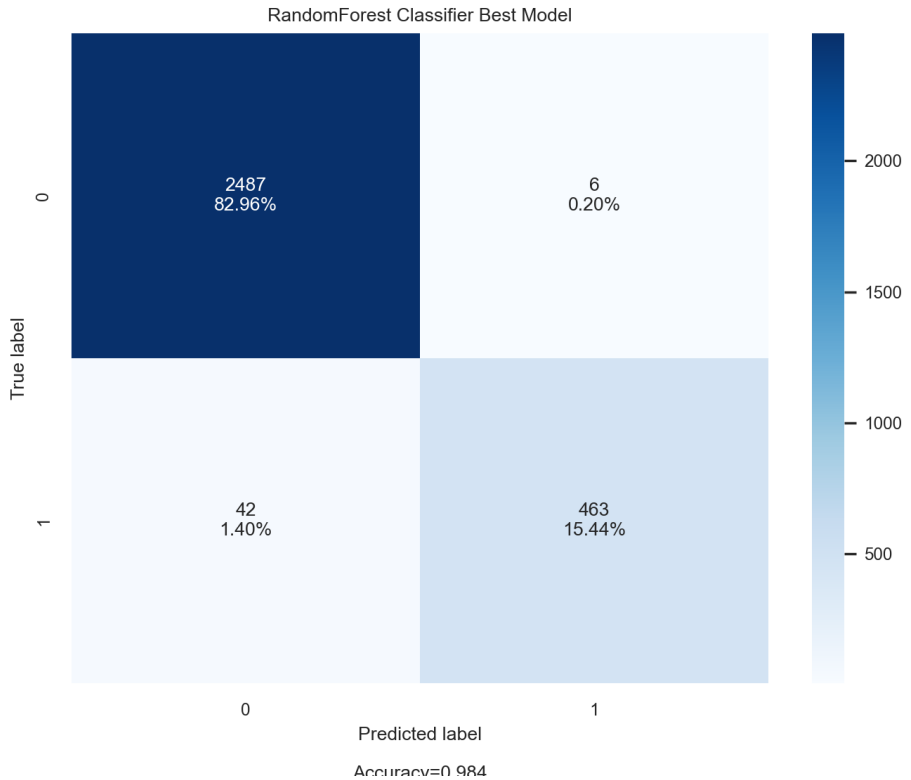
[69]:
```
0.9502917250538034
```

4.5 Best Model

[70]:
```
best_params = gs_random_forest.best_params_
model = RandomForestClassifier(**best_params, random_state=M_RN_STATE)
rf = model.fit(X_train, y_train)
```

[71]:
```
pred = rf.predict(X_test)
```

[72]:
```
cm = confusion_matrix(y_test, pred)
make_confusion_matrix(cm, figsize=(8,6), title='RandomForest Classifier Best␣
↪Model')
```



5 Execution

5.1 SHAP model explainer

[89]:
```
import shap
import pingouin as pg
```

[74]:
```
shap_values = shap.TreeExplainer(rf).shap_values(X_train)
shap.summary_plot(shap_values, X_train, plot_type='bar')
```

5.2 Hypothesis Test

The data violated normality and homoscedasticity and as a result we'll be conducting a non-parametric hypothesis test which are Kruskal-wallis and Games-howell post-hoc test.

- What are the important predictors of employee exit?

  The most important predictors are satisfaction level, number of project, time spent in

the company, last evaluation, and average monthly hours. Though, it's clear that low satisfaction levels will likely trigger an exit in any domain.

- Is there a difference in satisfaction between tenurity groups?

  The tenurity groups `New Employee <= 2 years` , `Middle Level Employee 3-4 years` and `senior or tenured employee > 4 years` have significant difference in means.

- Is there a difference in satisfaction levels between groups of average working hours per month.

  `below avg. (<=160 hrs. per month)`, `average (161-179 hrs. per month)`, `above average (> 180 hrs. per month)`

In our Kruskal-Wallis test, we observe a difference between groups. However, during our post-hoc test, we find no difference in means for the above-average group (180+ hours) and the average group ($\pm$10 of 170), as if the working hours intersect for both of these groups. This situation contrasts with the average and below-average groups, where we can see a more pronounced difference in means.

- Is there a difference in satisfaction levels between groups that of handling numerous projects

  `Light`, 1-2 projects `Moderate`, 3-4 projects `Heavy`, 5+ projects

Yes, the diference in the means of the groups are stastically significant.

[76]:
```
def map_tenure(year) -> str:

    if year <= 2:
        return 'New Employee'
    if (year >= 3) & (year <=4):
        return 'Middle Level Employee'
    else:
        return 'Senior or Tenured Employee'

def map_working_hours(hours) -> str:

    if hours < 160:
        return 'Below Average'
    if hours > 180:
        return 'Above Average'
    else:
        return 'Average'

def map_num_project(num) -> str:

    if num <= 2:
        return "Light"
    if num >= 5:
        return "Heavy"
    else:
        return "Moderate"
```

[77]:
```
# create a list of the important features based on RFC
important_features = ['satisfaction_level', 'number_project', 'last_evaluation',␣
↪'time_spend_company', 'average_monthly_hours']

# select featrues from dataframe
ht = df[important_features]
```

[78]:
```
ht['avg_monthly_hours_cat'] = ht.average_monthly_hours.map(map_working_hours)
ht['tenure_cat'] = ht.time_spend_company.map(map_tenure)
ht['number_projects_cat'] = ht.number_project.map(map_num_project)
```

[79]:
```
pd.pivot_table(ht, values=['satisfaction_level', 'time_spend_company',␣
↪'last_evaluation'], columns='number_project')
```

[79]:

number_project	2	3	4	5	6	7
last_evaluation	0.584640	0.717415	0.735682	0.750072	0.765654	0.863517
satisfaction_level	0.499071	0.689418	0.694280	0.671155	0.314673	0.116690
time_spend_company	3.186473	3.144602	3.300407	3.604120	4.154964	4.110345

5.2.1 Assumptions of Normality and Homosecadistcity

[80]:
```
pg.normality(ht)
```

[80]:

	W	pval	normal
satisfaction_level	0.952687	0.000000e+00	False
number_project	0.920626	0.000000e+00	False
last_evaluation	0.958242	0.000000e+00	False
time_spend_company	0.798271	0.000000e+00	False
average_monthly_hours	0.968443	5.605194e-45	False

[81]:
```
pg.homoscedasticity(ht)
```

[81]:

	W	pval	equal_var
levene	35562.196512	0.0	False

5.2.2 Kruskal-wallis test

H_0 : No statistical difference between groups of tenurity in explaining satisfaction level H_1 : There's a statistical difference between groups of tenurity in explaining satisfaction level

[82]:
```
pg.kruskal(ht, dv='satisfaction_level', between='tenure_cat')
```

[82]:

	Source	ddof1	H	p-unc
Kruskal	tenure_cat	2	239.997356	7.677791e-53

[83]:
```
post_hoc_1 = pg.pairwise_gameshowell(ht, dv='satisfaction_level',␣
↪between='tenure_cat')
post_hoc_1
```

[83]:
```
                         A                              B   mean(A)   mean(B)  \
0  Middle Level Employee                  New Employee  0.611954  0.699165
1  Middle Level Employee  Senior or Tenured Employee  0.611954  0.589952
2           New Employee  Senior or Tenured Employee  0.699165  0.589952

       diff        se          T           df          pval    hedges
0 -0.087211  0.004473 -19.495649  7110.820487  0.000000e+00 -0.380079
1  0.022002  0.007025   3.132079  2704.333094  4.989409e-03  0.087119
2  0.109213  0.007254  15.056402  2953.591841  1.424416e-13  0.483817
```

H_0 : No statistical difference between groups of average monthly worked hours in explaining satisfaction level
H_1 : There's a statistical difference between groups of average monthly worked hours in explaining satisfaction lev

[84]:
```
pg.kruskal(ht, dv='satisfaction_level', between='avg_monthly_hours_cat')
```

[84]:
```
                        Source  ddof1           H         p-unc
Kruskal  avg_monthly_hours_cat      2  265.602845  2.113854e-58
```

[85]:
```
post_hoc_2 = pg.pairwise_gameshowell(ht, dv='satisfaction_level',␣
↪between='avg_monthly_hours_cat')
post_hoc_2
```

[85]:
```
               A              B   mean(A)   mean(B)      diff        se  \
0  Above Average        Average  0.643258  0.657316 -0.014058  0.006331
1  Above Average  Below Average  0.643258  0.585330  0.057928  0.004844
2        Average  Below Average  0.657316  0.585330  0.071986  0.006778

           T           df          pval    hedges
0  -2.220492  2453.288839  6.795424e-02 -0.056792
1  11.958999  7041.543520  1.227796e-12  0.238831
2  10.619999  2965.866232  2.113310e-12  0.332110
```

[86]:
```
post_hoc_2.iloc[0]
```

[86]:
```
A          Above Average
B                Average
mean(A)           0.643258
mean(B)           0.657316
diff             -0.014058
se                0.006331
T                -2.220492
df             2453.288839
pval              0.067954
hedges           -0.056792
Name: 0, dtype: object
```

In our post-hoc test we're seeing that above average and average is not statistically significant as to the p-value of kruskal-wallis. Where above average versus average working hour groups' p-vaue is $0.067 > 0.05$.

H_0 : No statistical difference between groups of average monthly worked hours in explaining satisfaction level
H_1 : There's a statistical difference between groups of average monthly worked hours in explaining satisfaction lev

[87]:
```
pg.kruskal(ht, dv='satisfaction_level', between='number_projects_cat')
```

[87]:

	Source	ddof1	H	p-unc
Kruskal	number_projects_cat	2	1229.243966	1.183217e-267

[88]:
```
post_hoc_3 = pg.pairwise_gameshowell(ht, dv='satisfaction_level',␣
↪between='number_projects_cat')
post_hoc_3
```

[88]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	hedges
0	Heavy	Light	0.554160	0.499071	0.055090	0.006917	7.964125	4661.486533	5.226819e-12	0.206616
1	Heavy	Moderate	0.554160	0.691904	-0.137744	0.005818	-23.674678	4481.106426	0.000000e+00	-0.585981
2	Light	Moderate	0.499071	0.691904	-0.192833	0.004993	-38.623747	2549.437593	1.673217e-12	-0.991368

5.3 Conclusion & Recommendation

The analysis created the best model that can accurately predict about 95% of employee turn-over. Moreover, higlights the key features to look out for such as the number of projects, working hours, and tenurity.

- Satisfaction levels are low across the company mainly in the sales & technical department. And while higher salary can be a motivator for any employee this is not the prevailing case.
- The average working hour is higher than what the typical average working hour (165-170 per month) and this culture has drastrically affected the satisfaction levels of the employee. While it is essential to address over work and burn out, this only part of the truth.
  - The new employees who have light loads (1-2 projects) are equally dissatisffied with their standing. And this can be address by amending onboarding, creating continuous learning and mentorship programs. This is evident in our Unsupervised clustering, Bimodal distribution of our histogram, and Hypothesis test.
-