Problem in Context

The problem we are trying to solve is calculating the attrition rate of employees in an enterprise. The attrition rate is calculated on a plethora of parameters like 'Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome', 'Education', 'EducationField', 'EmployeeNumber', 'Gender', 'HourlyRate', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike', 'PerformanceRating', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager', etc. Hence carefuly analysing the data with help of EDA is paramount to develop a model for this dataset.

Link to dataset: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset

ML Metrics:

1. Confusion matrix
2. Train Accuracy
3. Test Accuracy

Software Metrics:

1. Number of lines of code
2. Less latency
3. Increased throughput

Business Metrics:

1. The attrition rate of employees in an organization
2. Management needs to understand reason behind attrition

Features in the dataset

Enlisted features: 'Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department','DistanceFromHome', 'Education', 'EducationField', 'EmployeeNumber','EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager'

```
Value to predict: Attrition
```

Protected Features

Following are some of the protected features in the dataset:

1. Gender
2. MaritalStatus
3. Age

Reason for Selecting the model

Logistic Regression to evaulate the datset and to serve as the baseline model from which an intuition can be gained.

Random Forest classifier to act on different samples and takes their majority vote for classification. Hence it serves a better model for our dataset.

Evaluate Model

The fairness of the model can be assessed by constructing the confusion matrix, with true positive, true negative, false positive and false negative. With these values we can calculate the precison and recall of the trained model. Also by predicting the train and test accuracy, we gauge the performance of our model.

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv("/content/WA_Fn-UseC_-HR-Employee-Attrition.csv")
# df.head()
df.tail()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educati
1465	36	No	Travel_Frequently	884	Research & Development	23	
1466	39	No	Travel_Rarely	613	Research & Development	6	
1467	27	No	Travel_Rarely	155	Research & Development	4	
1468	49	No	Travel_Frequently	1023	Sales	2	
1469	34	No	Travel_Rarely	628	Research & Development	8	

5 rows × 35 columns

Exploratory Data Analysis

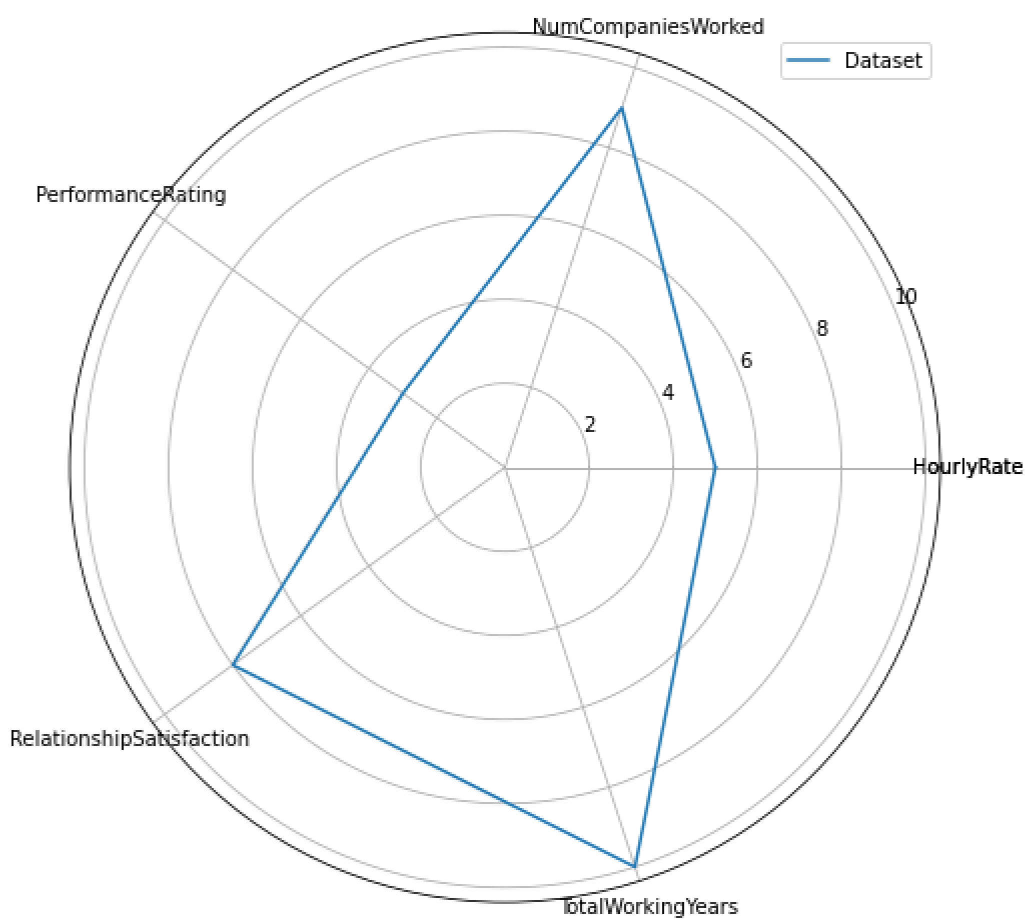
```
df.info()
# df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
```

```
 26  StandardHours            1470 non-null   int64
 27  StockOptionLevel         1470 non-null   int64
 28  TotalWorkingYears        1470 non-null   int64
 29  TrainingTimesLastYear    1470 non-null   int64
 30  WorkLifeBalance          1470 non-null   int64
 31  YearsAtCompany           1470 non-null   int64
 32  YearsInCurrentRole       1470 non-null   int64
 33  YearsSinceLastPromotion  1470 non-null   int64
 34  YearsWithCurrManager     1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

```
dimensions = ['HourlyRate', 'NumCompaniesWorked', 'PerformanceRating', 'RelationshipSatisfact
dimensions = [*dimensions, dimensions[0]]
values = [5, 9, 3, 8, 10]
values = [*values, values[0]]
label_loc = np.linspace(start=0, stop= 2*np.pi, num=len(values))

plt.figure(figsize=(8, 8))
plt.subplot(polar=True)
plt.plot(label_loc, values, label='Dataset')
lines, labels = plt.thetagrids(np.degrees(label_loc), labels=dimensions)
plt.legend()
plt.show()
```



```
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
# print(numeric_cols)
# print(categorical_cols)
```

```
missing_counts = df[numeric_cols].isna().sum().sort_values(ascending=False)
missing_counts[missing_counts > 0]
```

```
Series([], dtype: int64)
```

```
missing_counts = df[categorical_cols].isna().sum().sort_values(ascending=False)
missing_counts[missing_counts > 0]
```

```
Series([], dtype: int64)
```

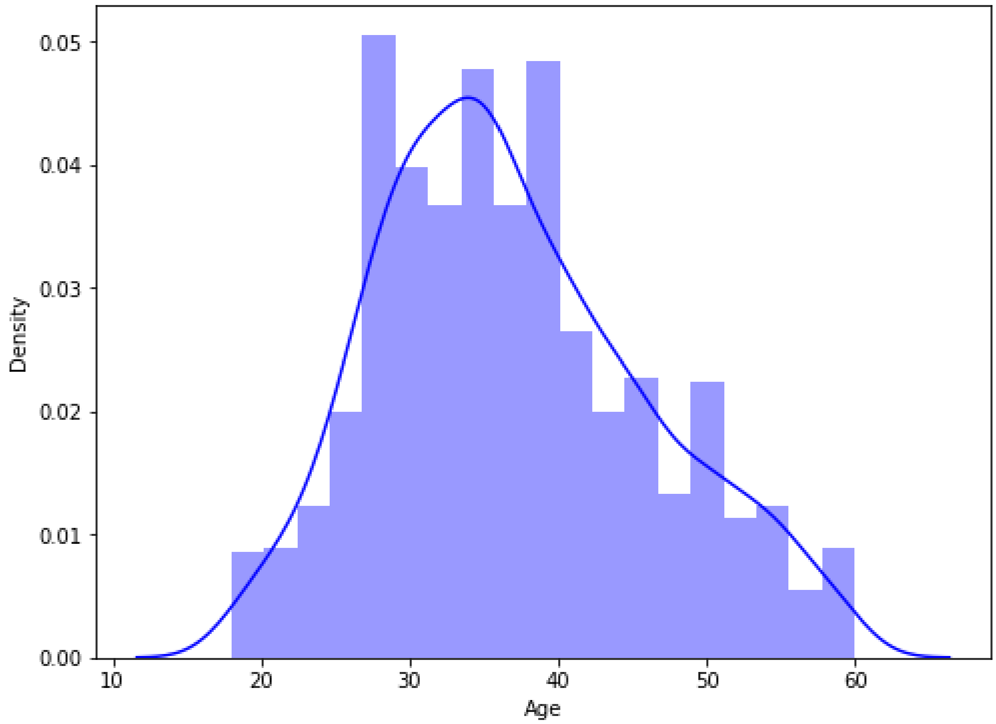
Discarding features that are least significant

```
df.Attrition.replace(to_replace = dict(Yes = 1, No = 0), inplace = True)
# Discard features that are least significant
df = df.drop(columns=['StandardHours',
                          'EmployeeCount',
                          'Over18',
                        ])
```

```
plt.figure(figsize=(8,6))
r = df.groupby('Attrition')['Attrition'].count()
plt.pie(r, explode=[0.05, 0.1], labels=['No', 'Yes'], radius=1.5, autopct='%1.1f%%',  shadow=
```

```
plt.figure(figsize=(8,6))
sns.distplot(df["Age"], color="b")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac0fe3b890>
```

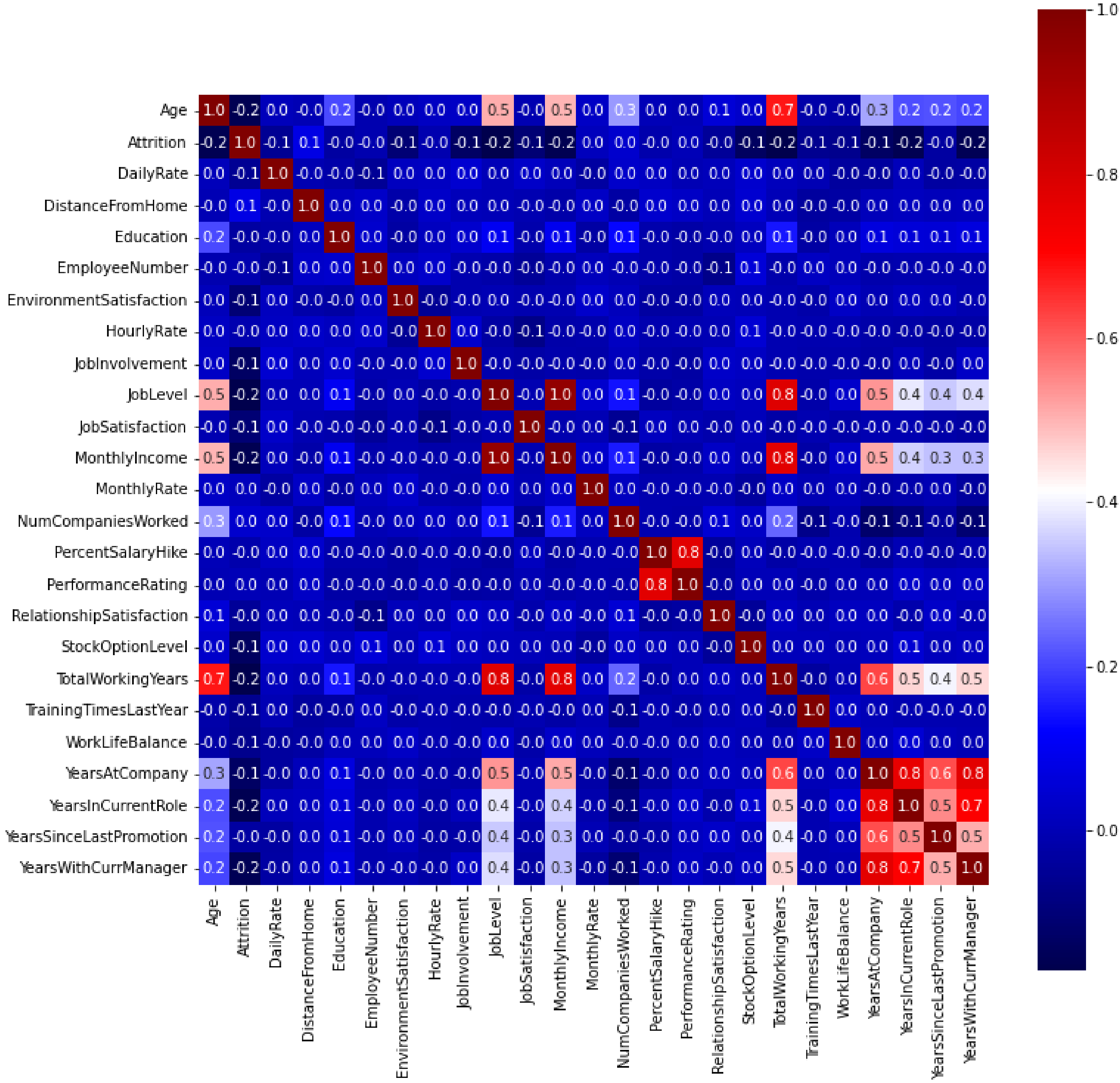



```
df[['Age']].value_counts().sort_values(ascending=False).head(10)
```

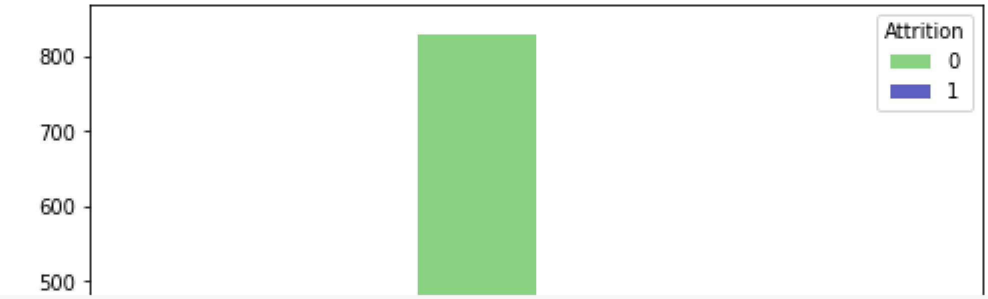
```
Age
35     78
34     77
36     69
31     69
29     68
32     61
30     60
38     58
33     58
40     57
dtype: int64
```

```
corr = df.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr,cbar=True,square=True,fmt='.1f',annot=True,cmap='seismic')
```

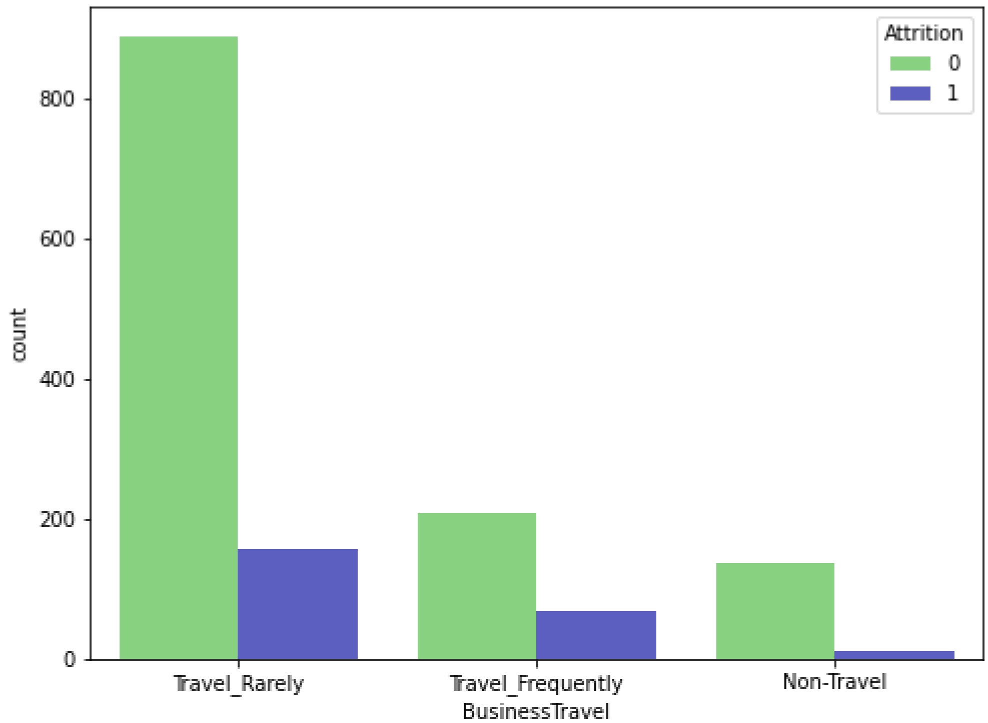
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac19dfbe50>
```



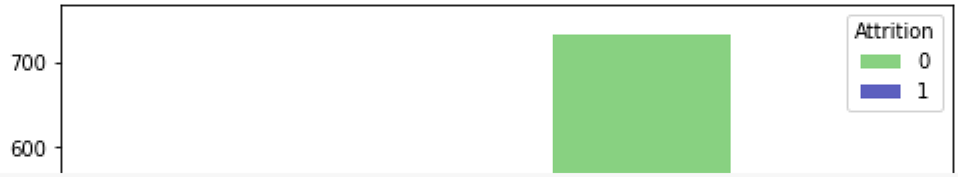
```
plt.figure(figsize=(8,6))
sns.countplot(x='Department', hue='Attrition', palette=['#7DDF73',"#4C50CF"], data=df);
```




```
plt.figure(figsize=(8,6))
sns.countplot(x='BusinessTravel', hue='Attrition', palette=['#7DDF73',"#4C50CF"], data=df);
```

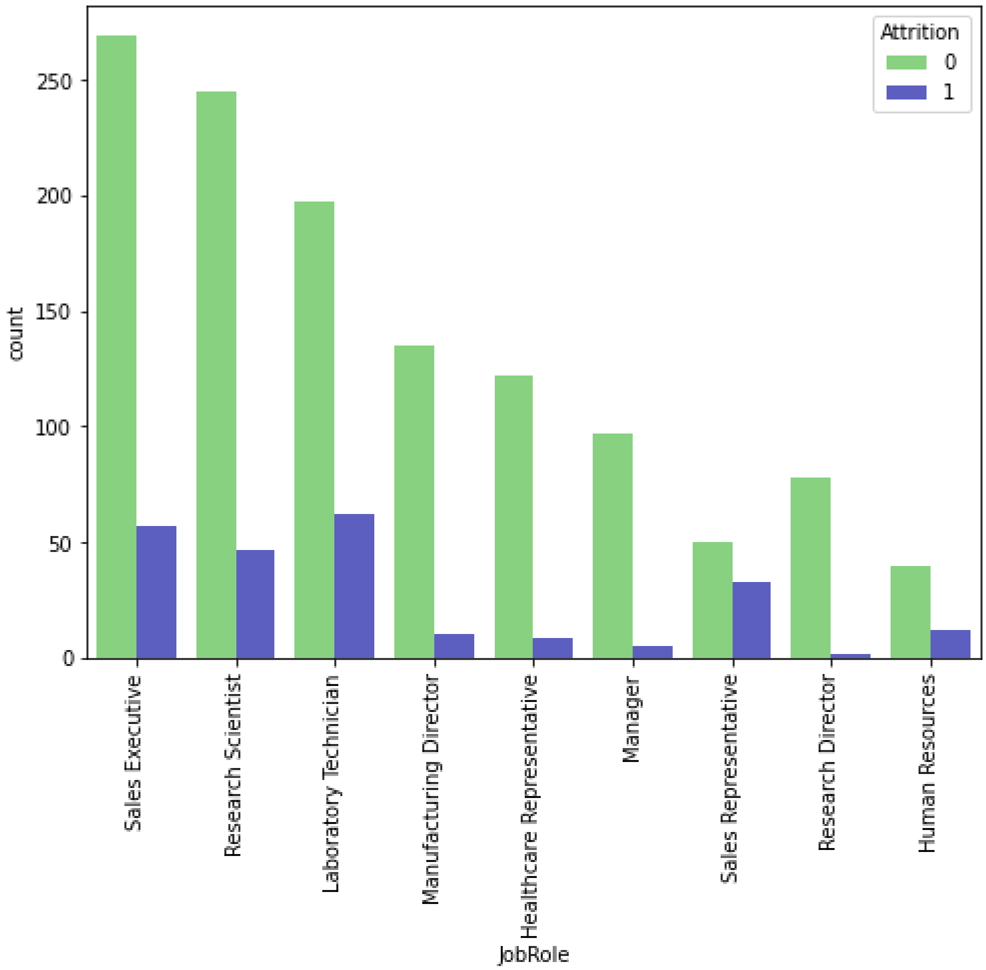



```
plt.figure(figsize=(8,6))
sns.countplot(x='Gender', hue='Attrition', palette=['#7DDF73',"#4C50CF"], data=df);
```



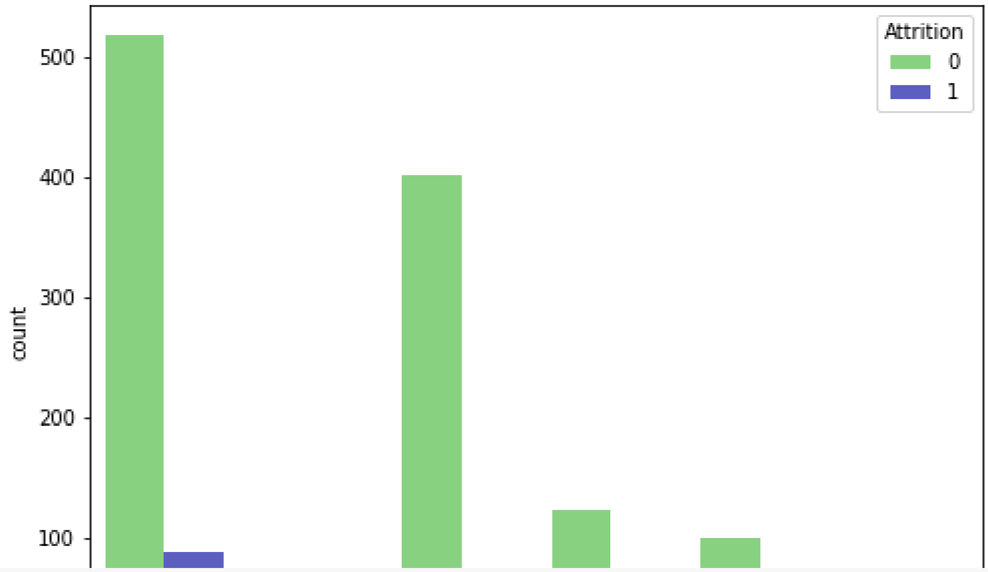

```
plt.figure(figsize=(8,6))
sns.countplot(x='JobRole', hue='Attrition', palette=['#7DDF73',"#4C50CF"], data=df);
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 <a list of 9 Text major ticklabel objects>)
```




```
plt.figure(figsize=(8,6))
sns.countplot(x='EducationField', hue='Attrition', palette=['#7DDF73',"#4C50CF"], data=df);
plt.xticks(rotation=45)
```

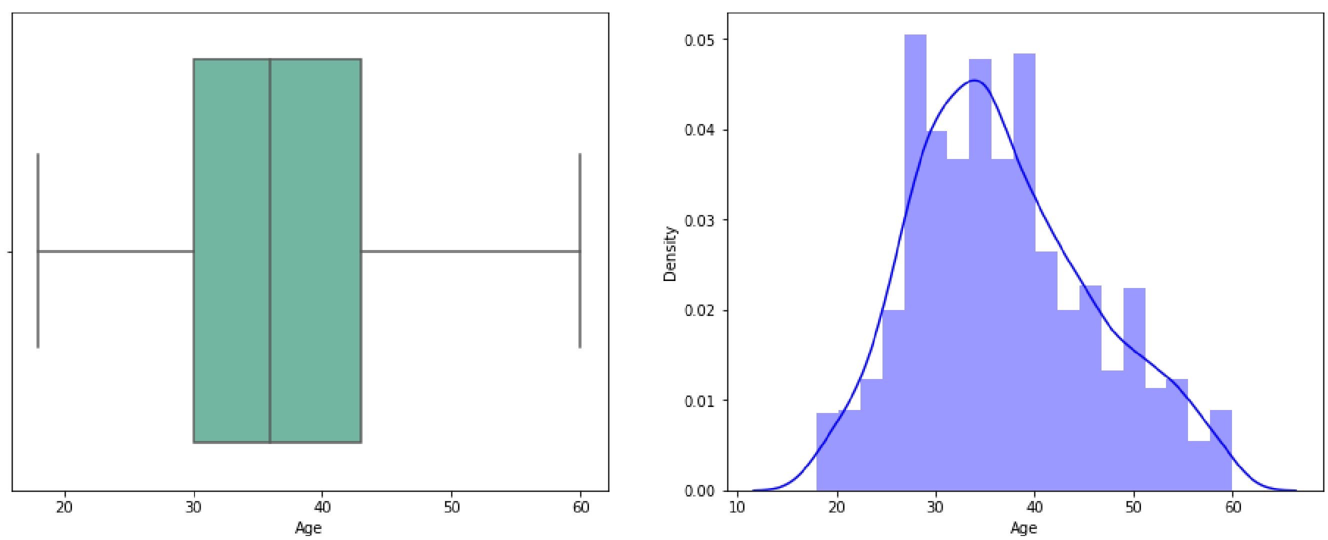
```
(array([0, 1, 2, 3, 4, 5]), <a list of 6 Text major ticklabel objects>)
```




```
#plt.figure(figsize=(12,12))
#sns.heatmap(df.corr(), annot=True)
```

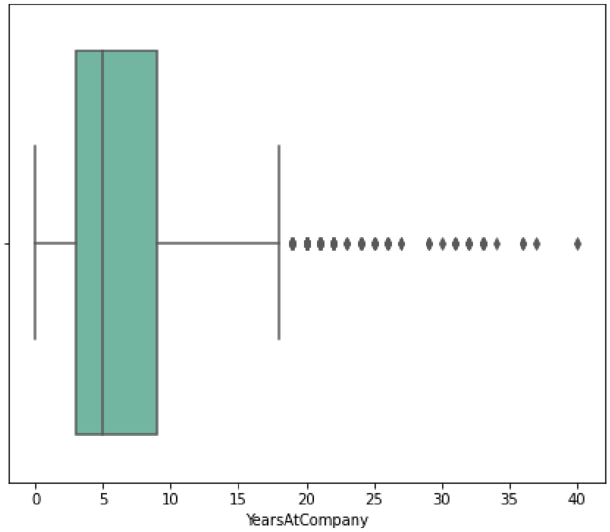
```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
sns.boxplot(ax = axes[0],palette = "Set2", x = df['Age'])
sns.distplot(ax = axes[1],color = "Blue",a=df["Age"])
```

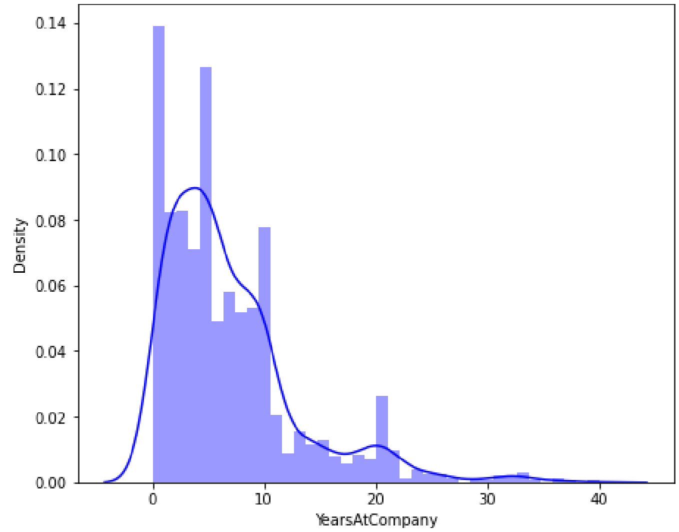
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac19af3b10>
```




```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
sns.boxplot(ax = axes[0],palette = "Set2",x = df['YearsAtCompany'])
sns.distplot(ax = axes[1],color = "Blue" ,a=df["YearsAtCompany"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac19be3710>
```






```
df["Attrition"] = LabelEncoder().fit_transform(df['Attrition'])
df["BusinessTravel"] = LabelEncoder().fit_transform(df['BusinessTravel'])
df["Department"] = LabelEncoder().fit_transform(df['Department'])
df["EducationField"] = LabelEncoder().fit_transform(df['EducationField'])
df["Gender"] = LabelEncoder().fit_transform(df['Gender'])
df["JobRole"] = LabelEncoder().fit_transform(df['JobRole'])
df["MaritalStatus"] = LabelEncoder().fit_transform(df['MaritalStatus'])
df["OverTime"] = LabelEncoder().fit_transform(df['OverTime'])
```

```
numeric_cols.remove('StandardHours')
numeric_cols.remove('EmployeeCount')
cols = list(df.columns)
cols.remove("Attrition")

df[numeric_cols] = MinMaxScaler().fit_transform(df[numeric_cols])
sampled,target = SMOTE().fit_resample(df[cols],df["Attrition"])
```

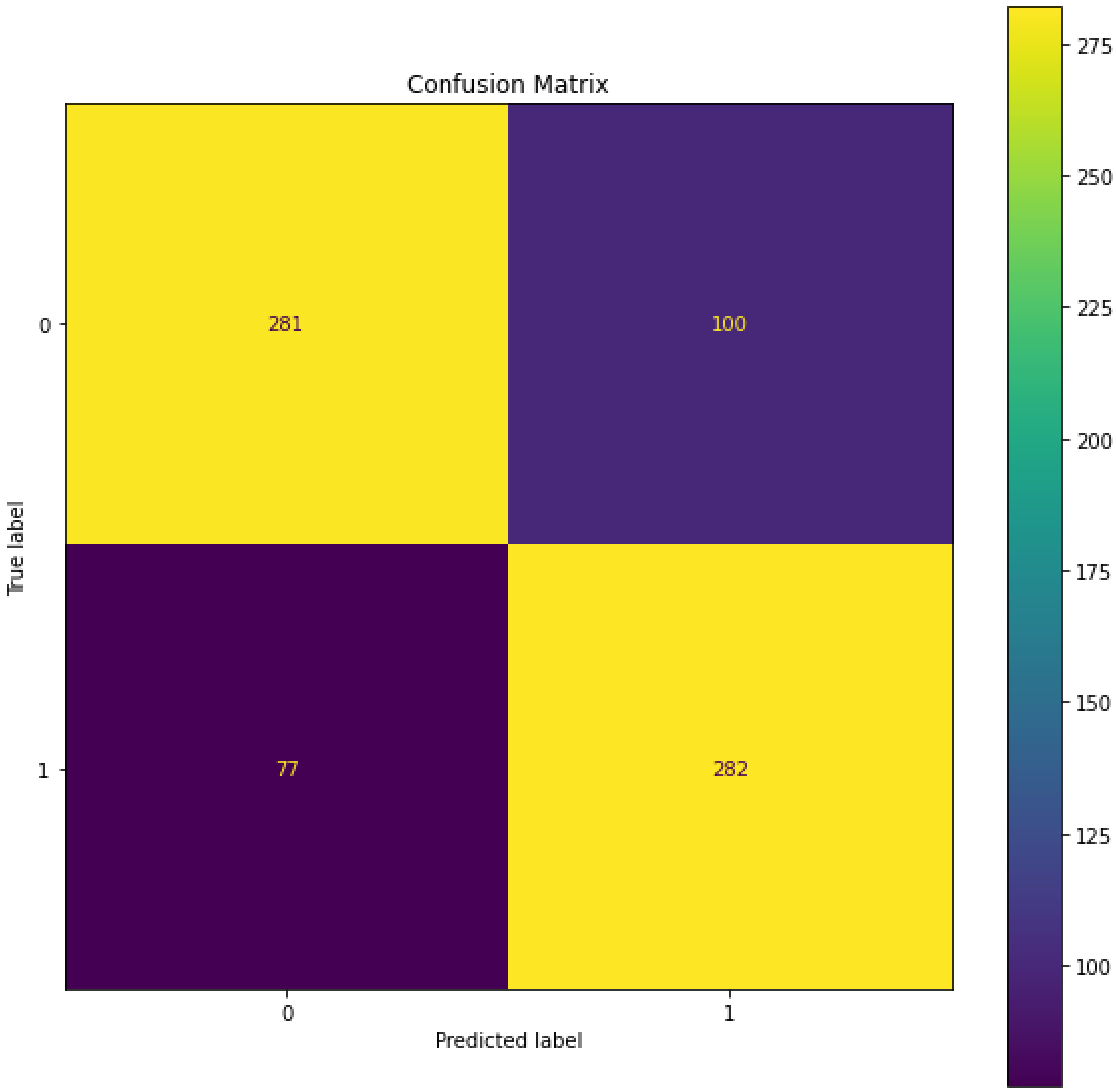
```
X_train,X_test,Y_train,Y_test = train_test_split(sampled[cols],
                                                 target,
                                                 test_size = 0.3,
                                                 shuffle=True)
```

```
logistic_model = LogisticRegression(solver='liblinear',random_state=0).fit(X_train,Y_train)
print("Train Accuracy : {:.2f} %".format(accuracy_score(logistic_model.predict(X_train),Y_tra
print("Test Accuracy : {:.2f} %".format(accuracy_score(logistic_model.predict(X_test),Y_test)

cm = confusion_matrix(Y_test,logistic_model.predict(X_test))
classes = ["0","1"]
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=classes)
fig, ax = plt.subplots(figsize=(10,10))
plt.title("Confusion Matrix")
disp = disp.plot(ax=ax)
plt.show()
```

```
Train Accuracy : 0.76 %
Test Accuracy : 0.76 %
```

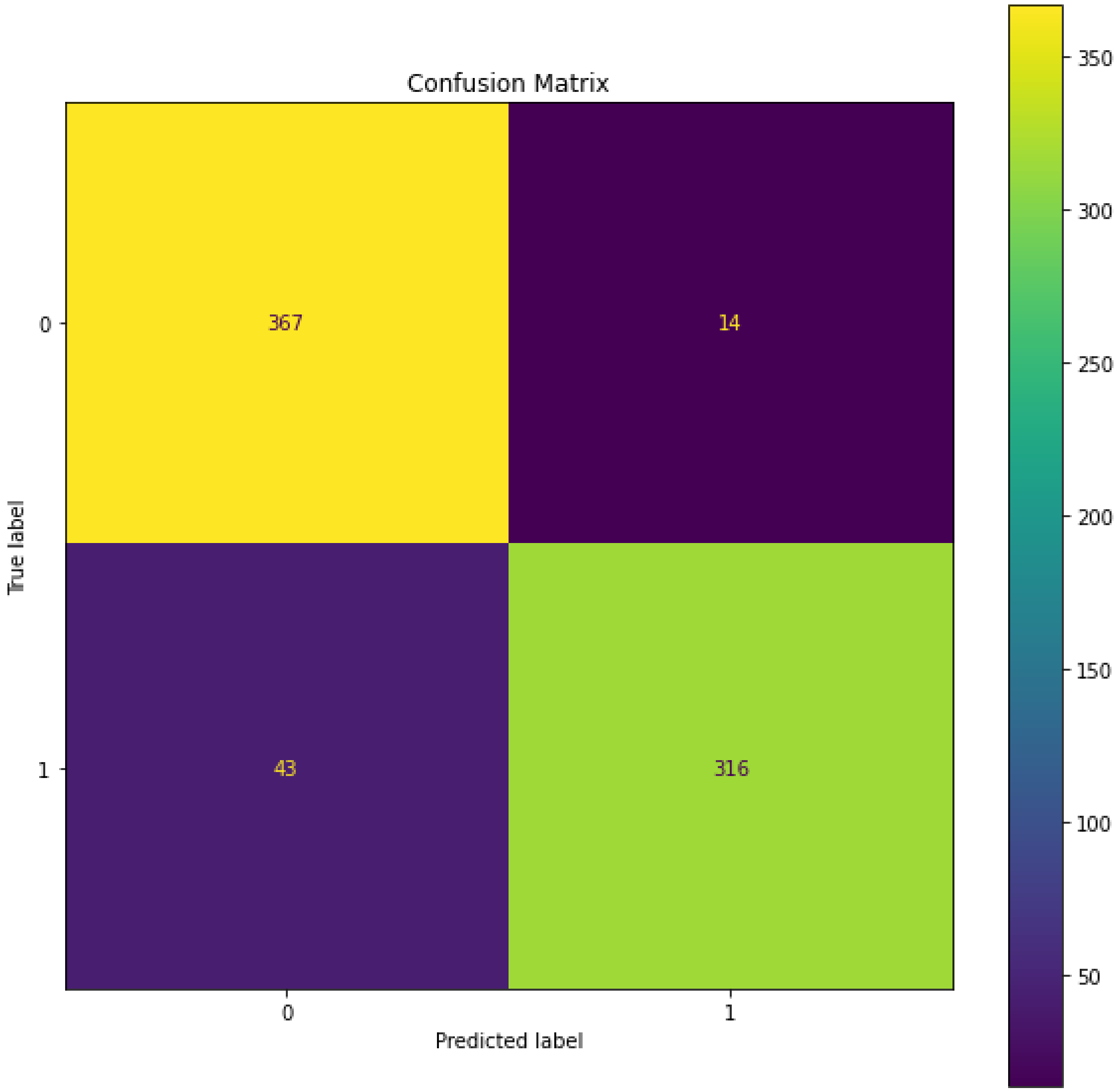



```
random_forest = RandomForestClassifier(n_estimators=590,
                                       random_state=0).fit(X_train,Y_train)
print("Train Accuracy : {:.2f} %".format(accuracy_score(random_forest.predict(X_train),Y_trai
print("Test Accuracy : {:.2f} %".format(accuracy_score(random_forest.predict(X_test),Y_test)

cm = confusion_matrix(Y_test,random_forest.predict(X_test))
classes = ["0","1"]
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=classes)
fig, ax = plt.subplots(figsize=(10,10))
plt.title("Confusion Matrix")
```

```
disp = disp.plot(ax=ax)
plt.show()
```

```
Train Accuracy : 1.00 %
Test Accuracy : 0.92 %
```

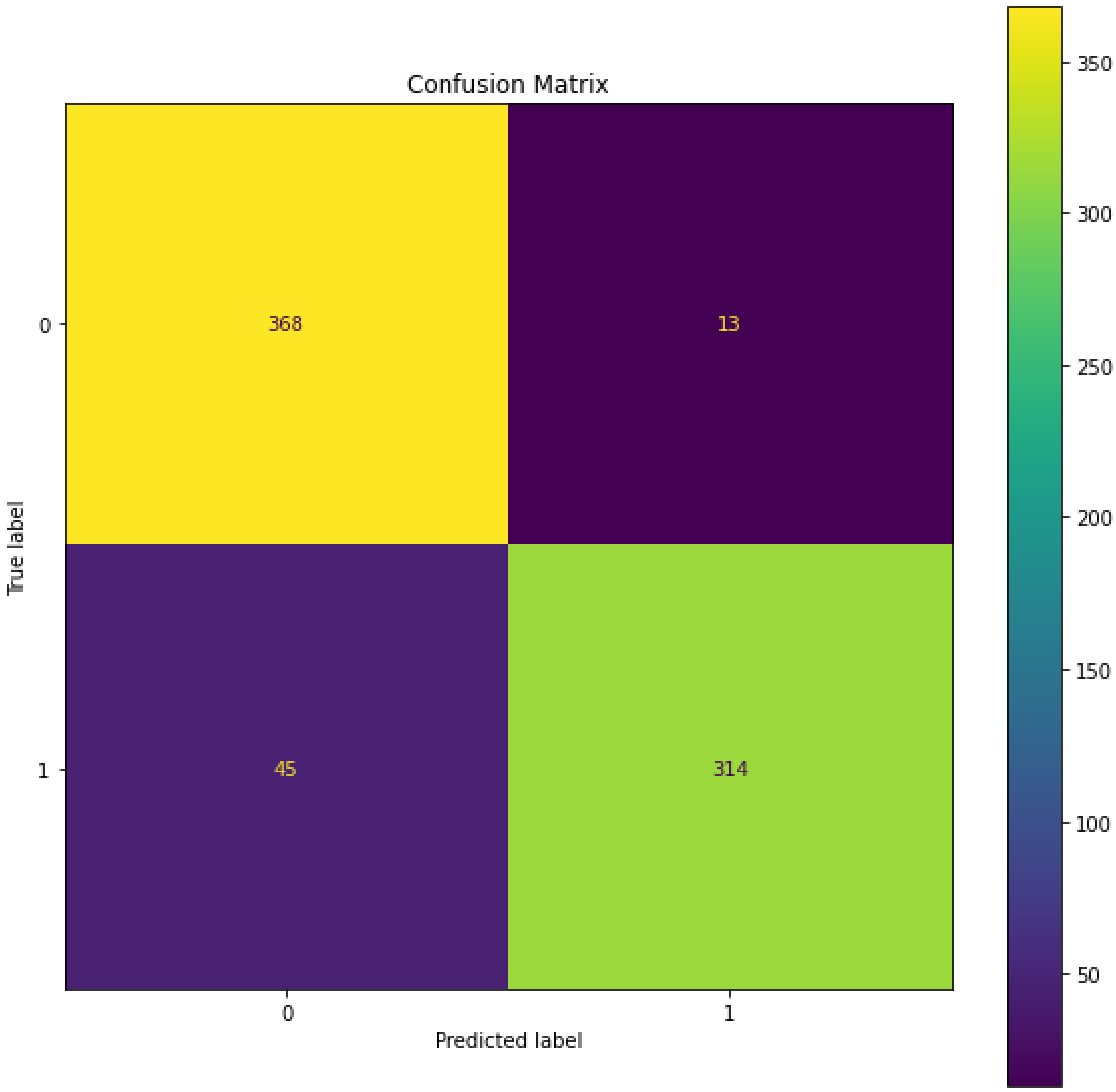



```
from xgboost import XGBClassifier
model = XGBClassifier(learning_rate=0.01,n_estimators=2000,use_label_encoder=False,random_sta

print("Train Accuracy : {:.2f} %".format(accuracy_score(model.predict(X_train),Y_train)))
print("Test Accuracy : {:.2f} %".format(accuracy_score(model.predict(X_test),Y_test)))

cm = confusion_matrix(Y_test,model.predict(X_test))
classes = ["0","1"]
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=classes)
fig, ax = plt.subplots(figsize=(10,10))
plt.title("Confusion Matrix")
disp = disp.plot(ax=ax)
plt.show()
```

```
Train Accuracy : 0.98 %
Test Accuracy : 0.92 %
```




```
#df.info()
```

```
#!pip install tfx --user
```

```
#!pip install tfx --user

import os
from absl import logging
import urllib.request
import tempfile
import pandas as pd
logging.set_verbosity(logging.INFO)  # Set default logging level.

import tensorflow as tf
print('TensorFlow version: {}'.format(tf.__version__))
```

```
from tfx import v1 as tfx
print('TFX version: {}'.format(tfx.__version__))
```

```
TensorFlow version: 2.9.1
TFX version: 1.9.1
```

```
# PIPELINE_NAME is the name of the pipeline.
PIPELINE_NAME = "ibm-hr-analysis-attrition-dataset"

# PIPELINE_ROOT for output directory to store artifacts generated from the pipeline.
PIPELINE_ROOT = os.path.join('pipeline', PIPELINE_NAME)

# METADATA_PATH for storing meta data.
METADATA_PATH = os.path.join('metadata', PIPELINE_NAME, 'metadata.db')

# SERVING_MODEL_DIR to deploy model.
SERVING_MODEL_DIR = os.path.join('serving_model', PIPELINE_NAME)
```

```
# Creating a temporary directory.
DATA_ROOT = tempfile.mkdtemp(prefix='tfx-data')

df.to_csv(DATA_ROOT + "/WA_Fn-UseC_-HR-Employee-Attrition.csv")
_data_filepath =  DATA_ROOT + "/WA_Fn-UseC_-HR-Employee-Attrition.csv"
```

```
!head {_data_filepath}
```

```
,Age,Attrition,BusinessTravel,DailyRate,Department,DistanceFromHome,Education,Education
0,0.5476190476190477,1,2,0.7158196134574086,2,0.0,0.25,1,0.0,0.3333333333333333,0,0.914
1,0.7380952380952379,0,1,0.12670007158196134,1,0.25,0.0,1,0.0004837929366231253,0.66666
2,0.4523809523809524,1,2,0.9098067287043664,1,0.03571428571428571,0.25,4,0.001451378809
3,0.35714285714285715,0,1,0.9234073013600572,1,0.07142857142857142,0.75,1,0.00193517174
4,0.21428571428571425,0,2,0.35003579098067283,1,0.03571428571428571,0.0,3,0.00290275761
5,0.3333333333333333,0,1,0.64638511095204,1,0.03571428571428571,0.25,1,0.00338655055636
6,0.976190476190476,0,2,0.8747315676449534,1,0.07142857142857142,0.5,3,0.00435413642960
7,0.28571428571428564,0,2,0.8990694345025053,1,0.8214285714285714,0.0,1,0.0048379293662
8,0.4761904761904761,0,1,0.0816034359341446,1,0.7857142857142857,0.5,1,0.00532172230285
```

```
_trainer_module_file = 'attrition_train.py'
```

```
#df.info()
```

```
_temp_dir = os.path.join(tempfile.gettempdir(), 'ibm-hr-analysis/')
os.mkdir(_temp_dir)

_data_dir = os.path.join(_temp_dir, 'data/')
os.mkdir(_data_dir)
```

```
_serving_model_dir = os.path.join(_temp_dir, 'serving_model/')
os.mkdir(_serving_model_dir)
```

```
df.to_csv(_data_dir + "WA_Fn-UseC_-HR-Employee-Attrition.csv")
```

```
df.columns
```

```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeNumber',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')
```

```
%%writefile {_trainer_module_file}

from typing import List
from absl import logging
import tensorflow as tf
from tensorflow import keras
from tensorflow_transform.tf_metadata import schema_utils

from tfx import v1 as tfx
from tfx_bsl.public import tfxio
from tensorflow_metadata.proto.v0 import schema_pb2

# define the list of features in _FEATURE_KEYS variable
# 8 => CODE HERE #
_FEATURE_KEYS = ['BusinessTravel', 'Department', 'EducationField','Gender','JobRole','Marital

_FLOAT_KEYS = [ 'Age', 'DailyRate',
       'DistanceFromHome', 'Education',  'EmployeeNumber',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager']

# define your target variable _LABEL_KEY
# 9 => CODE HERE #
_LABEL_KEY = 'Attrition'

_TRAIN_BATCH_SIZE = 20
_EVAL_BATCH_SIZE = 10
```

```
FEATURE_MAP = dict()

for feature in _FEATURE_KEYS:
    FEATURE_MAP[feature] =  tf.io.FixedLenFeature(shape=[1], dtype=tf.int64)

for feature in _FLOAT_KEYS:
    FEATURE_MAP[feature] =  tf.io.FixedLenFeature(shape=[1], dtype=tf.float32)

FEATURE_MAP[_LABEL_KEY] = tf.io.FixedLenFeature(shape=[1], dtype=tf.int64)
# Since we're not generating or creating a schema, we will instead create
# a feature spec.  Since there are a fairly small number of features this is
# manageable for this dataset.
_FEATURE_SPEC = {
    **{
        feature: FEATURE_MAP
       },

    _LABEL_KEY: tf.io.FixedLenFeature(shape=[1], dtype=tf.int64)
}


def _input_fn(file_pattern: List[str],
              data_accessor: tfx.components.DataAccessor,
              schema: schema_pb2.Schema,
              batch_size: int = 200) -> tf.data.Dataset:
  """Generates features and label for training.

  Args:
    file_pattern: List of paths or patterns of input tfrecord files.
    data_accessor: DataAccessor for converting input to RecordBatch.
    schema: schema of the input data.
    batch_size: representing the number of consecutive elements of returned
      dataset to combine in a single batch

  Returns:
    A dataset that contains (features, indices) tuple where features is a
      dictionary of Tensors, and indices is a single Tensor of label indices.
  """
  return data_accessor.tf_dataset_factory(
      file_pattern,
      tfxio.TensorFlowDatasetOptions(
          batch_size=batch_size, label_key=_LABEL_KEY),
      schema=schema).repeat()


def _build_keras_model() -> tf.keras.Model:
  """Creates a DNN Keras model for classifying penguin data.

  Returns:
    A Keras Model.
```

```
  """
  # The model below is built with Functional API, please refer to
  # https://www.tensorflow.org/guide/keras/overview for all API options.
  inputs = [keras.layers.Input(shape=(1,), name=f) for f in _FEATURE_KEYS]
  d = keras.layers.concatenate(inputs)
  # compelete your model architecture here
  # 10 => CODE HERE #
  for _ in range(2):
    d = keras.layers.Dense(8, activation='relu')(d)
  outputs = keras.layers.Dense(3)(d)

  model = keras.Model(inputs=inputs, outputs=outputs)
  model.compile(
      optimizer=keras.optimizers.Adam(1e-2),
      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics=[keras.metrics.SparseCategoricalAccuracy()])

  model.summary(print_fn=logging.info)
  return model


# TFX Trainer will call this function.
def run_fn(fn_args: tfx.components.FnArgs):
  """Train the model based on given args.

  Args:
    fn_args: Holds args used to train the model as name/value pairs.
  """

  # This schema is usually either an output of SchemaGen or a manually-curated
  # version provided by pipeline author. A schema can also derived from TFT
  # graph if a Transform component is used. In the case when either is missing,
  # `schema_from_feature_spec` could be used to generate schema from very simple
  # feature_spec, but the schema returned would be very primitive.
  schema = schema_utils.schema_from_feature_spec(FEATURE_MAP)

  train_dataset = _input_fn(
      fn_args.train_files,
      fn_args.data_accessor,
      schema,
      batch_size=_TRAIN_BATCH_SIZE)
  eval_dataset = _input_fn(
      fn_args.eval_files,
      fn_args.data_accessor,
      schema,
      batch_size=_EVAL_BATCH_SIZE)

  model = _build_keras_model()
  model.fit(
      train_dataset,
      steps_per_epoch=fn_args.train_steps,
```

```
      validation_data=eval_dataset,
      validation_steps=fn_args.eval_steps)

  # The result of the training should be saved in `fn_args.serving_model_dir`
  # directory.
  model.save(fn_args.serving_model_dir, save_format='tf')
```

```
Overwriting attrition_train.py
```

```
import tensorflow_model_analysis as tfma

def _create_pipeline(pipeline_name: str, pipeline_root: str, data_root: str,
                     module_file: str, serving_model_dir: str,
                     metadata_path: str) -> tfx.dsl.Pipeline:
  """Creates a three component census pipeline with TFX."""
  # Brings data into the pipeline.
  example_gen = tfx.components.CsvExampleGen(input_base=data_root)

  # Uses user-provided Python function that trains a model.
  trainer = tfx.components.Trainer(
      module_file=module_file,
      examples=example_gen.outputs['examples'],
      train_args=tfx.proto.TrainArgs(num_steps=1000),
      eval_args=tfx.proto.EvalArgs(num_steps=5))

  model_resolver = tfx.dsl.Resolver(
      strategy_class=tfx.dsl.experimental.LatestBlessedModelStrategy,
      model=tfx.dsl.Channel(type=tfx.types.standard_artifacts.Model),
      model_blessing=tfx.dsl.Channel(
          type=tfx.types.standard_artifacts.ModelBlessing)).with_id(
              'latest_blessed_model_resolver')
  eval_config = tfma.EvalConfig(
      model_specs=[tfma.ModelSpec(label_key='Attrition')],
      slicing_specs=[
          # An empty slice spec means the overall slice, i.e. the whole dataset.
          tfma.SlicingSpec(),
          # Calculate metrics for each penguin species.
          tfma.SlicingSpec(feature_keys=[ 'BusinessTravel', 'Department', 'EducationField','G
          ],
      metrics_specs=[
          tfma.MetricsSpec(per_slice_thresholds={
              'sparse_categorical_accuracy':
                  tfma.PerSliceMetricThresholds(thresholds=[
                      tfma.PerSliceMetricThreshold(
                          slicing_specs=[tfma.SlicingSpec()],
                          threshold=tfma.MetricThreshold(
                              value_threshold=tfma.GenericValueThreshold(
                                   lower_bound={'value': 0.6}),
                              # Change threshold will be ignored if there is no
                              # baseline model resolved from MLMD (first run).
```

```
                              change_threshold=tfma.GenericChangeThreshold(
                                  direction=tfma.MetricDirection.HIGHER_IS_BETTER,
                                  absolute={'value': -1e-10}))
                       )]),
            })],
        )
  evaluator = tfx.components.Evaluator(
      examples=example_gen.outputs['examples'],
      model=trainer.outputs['model'],
      baseline_model=model_resolver.outputs['model'],
      eval_config=eval_config)

  # Pushes the model to a filesystem destination.
  pusher = tfx.components.Pusher(
      model=trainer.outputs['model'],
      push_destination=tfx.proto.PushDestination(
          filesystem=tfx.proto.PushDestination.Filesystem(
              base_directory=serving_model_dir)))

  # Following three components will be included in the pipeline.
  components = [
      example_gen,
      trainer,
      model_resolver,
      evaluator,
      pusher,
  ]

  return tfx.dsl.Pipeline(
      pipeline_name=pipeline_name,
      pipeline_root=pipeline_root,
      metadata_connection_config=tfx.orchestration.metadata
      .sqlite_metadata_connection_config(metadata_path),
      components=components)
```

```
tfx.orchestration.LocalDagRunner().run(
  _create_pipeline(
      pipeline_name=PIPELINE_NAME,
      pipeline_root=PIPELINE_ROOT,
      data_root=DATA_ROOT,
      module_file=_trainer_module_file,
      serving_model_dir=SERVING_MODEL_DIR,
      metadata_path=METADATA_PATH))
```

```
INFO:absl:Generating ephemeral wheel package for '/content/attrition_train.py' (incl
INFO:absl:User module package has hash fingerprint version 86f3b89d7ffd403bbcb9c29c6
INFO:absl:Executing: ['/usr/bin/python3', '/tmp/tmp3tvmve54/_tfx_generated_setup.py'
INFO:absl:Successfully built user code wheel distribution at 'pipeline/ibm-hr-analys
INFO:absl:Full user module path is 'attrition_train@pipeline/ibm-hr-analysis-attriti
INFO:absl:Using deployment config:
 executor_specs {
  key: "CsvExampleGen"
```

```
  value {
    beam_executable_spec {
      python_executor_spec {
        class_path: "tfx.components.example_gen.csv_example_gen.executor.Executor"
      }
    }
  }
}
executor_specs {
  key: "Evaluator"
  value {
    beam_executable_spec {
      python_executor_spec {
        class_path: "tfx.components.evaluator.executor.Executor"
      }
    }
  }
}
executor_specs {
  key: "Pusher"
  value {
    python_class_executable_spec {
      class_path: "tfx.components.pusher.executor.Executor"
    }
  }
}
executor_specs {
  key: "Trainer"
  value {
    python_class_executable_spec {
      class_path: "tfx.components.trainer.executor.GenericExecutor"
    }
  }
}
custom_driver_specs {
  key: "CsvExampleGen"
  value {
    python_class_executable_spec {
      class_path: "tfx.components.example_gen.driver.FileBasedDriver"
    }
  }
}
metadata_connection_config {
  database_connection_config {
    sqlite {
      filename_uri: "metadata/ibm-hr-analysis-attrition-dataset/metadata.db"
      connection_mode: READWRITE_OPENCREATE
    }
  }
```

```
import os
import json
import absl
import shutil
```

```
import pprint
import urllib
import tempfile
import requests
import tensorflow as tf
import tensorflow_model_analysis as tfma

import tfx
from tfx.components import CsvExampleGen, Evaluator, ExampleValidator, Pusher, SchemaGen, Sta
from tfx.components.base import executor_spec
from tfx.components.trainer.executor import GenericExecutor
from tfx.types import Channel
from tfx.types.standard_artifacts import Model, ModelBlessing
from tfx.proto import example_gen_pb2, pusher_pb2, trainer_pb2
from tfx.dsl.components.common.resolver import Resolver
from tfx.dsl.experimental import latest_blessed_model_resolver
from tfx.orchestration.experimental.interactive.interactive_context import InteractiveContext
```

```
#! find {'serving_model'}
```

```
tf.get_logger().propagate = False
pp = pprint.PrettyPrinter()
%load_ext tfx.orchestration.experimental.interactive.notebook_extensions.skip
```

```
The tfx.orchestration.experimental.interactive.notebook_extensions.skip extension is al
  %reload_ext tfx.orchestration.experimental.interactive.notebook_extensions.skip
```

```
context = InteractiveContext()
```

```
WARNING:absl:InteractiveContext pipeline_root argument not provided: using temporary di
WARNING:absl:InteractiveContext metadata_connection_config not provided: using SQLite M
```

```
output = example_gen_pb2.Output(
             split_config=example_gen_pb2.SplitConfig(splits=[
                 example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
                 example_gen_pb2.SplitConfig.Split(name='eval', hash_buckets=1)
             ]))

example_gen = CsvExampleGen(input_base=_data_dir, output_config=output)
context.run(example_gen)
```

```
INFO:absl:Running driver for CsvExampleGen
INFO:absl:MetadataStore with DB connection initialized
INFO:absl:select span and version = (0, None)
INFO:absl:latest span and version = (0, None)
INFO:absl:Running executor for CsvExampleGen
INFO:absl:Generating examples.
INFO:absl:Processing input csv data /tmp/ibm-hr-analysis/data/* to TFExample.
INFO:absl:Examples generated.
INFO:absl:Running publisher for CsvExampleGen
INFO:absl:MetadataStore with DB connection initialized
```

▼**ExecutionResult** at 0x7fac0e113790

```
# Get the URI of the output artifact representing the training examples, which is a directory
train_uri = os.path.join(example_gen.outputs['examples'].get()[0].uri, 'Split-train')

# Get the list of files in this directory (all compressed TFRecord files)
tfrecord_filenames = [os.path.join(train_uri, name)
                      for name in os.listdir(train_uri)]

# Create a `TFRecordDataset` to read these files
dataset = tf.data.TFRecordDataset(tfrecord_filenames, compression_type='GZIP')

# Iterate over the first record and decode it.
for tfrecord in dataset.take(1):
  serialized_example = tfrecord.numpy()
  example = tf.train.Example()
  example.ParseFromString(serialized_example)
  pp.pprint(example)
```

```
features {
  feature {
    key: ""
    value {
      int64_list {
        value: 0
      }
    }
  }
  feature {
    key: "Age"
    value {
      float_list {
        value: 0.5476190447807312
      }
    }
  }
  feature {
    key: "Attrition"
    value {
      int64_list {
        value: 1
      }
    }
```

```
        }
        feature {
          key: "BusinessTravel"
          value {
            int64_list {
              value: 2
            }
          }
        }
        feature {
          key: "DailyRate"
          value {
            float_list {
              value: 0.7158195972442627
            }
          }
        }
        feature {
          key: "Department"
          value {
            int64_list {
              value: 2
            }
          }
        }
        feature {
          key: "DistanceFromHome"
          value {
            float_list {
              value: 0.0
            }
          }
        }
```


```
statistics_gen = StatisticsGen(examples=example_gen.outputs['examples'])
context.run(statistics_gen)
```

```
INFO:absl:Excluding no splits because exclude_splits is not set.
INFO:absl:Running driver for StatisticsGen
```

```
context.show(statistics_gen.outputs['statistics'])
```

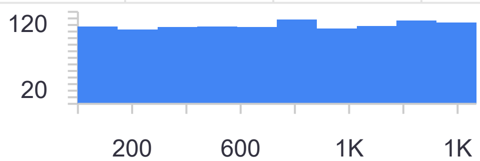
Artifact at /tmp/tfx-interactive-2022-08-13T03_13_46.404200-t1ko_o29/StatisticsGen/statistics/2

'train' split:

Sort by
Feature order ☐ Reverse order Feature search (regex enabled)

Features: ☐ int(8) ☐ float(24) ☐ unknown(1)

Numeric Features (33)



	count	missing	mean	std dev	zeros	min	median	max
Age	1,193	0%	0.45	0.22	0.59%	0	0.43	1
Attrition	1,193	0%	0.16	0.37	84.16%	0	0	1
BusinessTravel	1,193	0%	1.62	0.66	10.14%	0	2	2

'eval' split:

```
schema_gen = SchemaGen(statistics=statistics_gen.outputs['statistics'],
                       infer_feature_shape=True)

context.run(schema_gen)
```

```
INFO:absl:Excluding no splits because exclude_splits is not set.
INFO:absl:Running driver for SchemaGen
INFO:absl:MetadataStore with DB connection initialized
INFO:absl:Running executor for SchemaGen
INFO:absl:Processing schema from statistics for split train.
INFO:absl:Processing schema from statistics for split eval.
INFO:absl:Schema written to /tmp/tfx-interactive-2022-08-13T03_13_46.404200-t1ko_o29/Sc
INFO:absl:Running publisher for SchemaGen
INFO:absl:MetadataStore with DB connection initialized
```

▼**ExecutionResult** at 0x7fac19b36dd0

.execution_id	3
.component	▶**SchemaGen** at 0x7fac0c8cae50
.component.inputs	**['statistics']** ▶**Channel** of type **'ExampleStatistics'** (1 artifact) at 0x7fac0c8ff450
.component.outputs	**['schema']** ▶**Channel** of type **'Schema'** (1 artifact) at 0x7fac19b36a90

```
context.show(schema_gen.outputs['schema'])
```

Artifact at /tmp/tfx-interactive-2022-08-13T03_13_46.404200-t1ko_o29/SchemaGen/schema/3

Feature name	Type	Presence	Valency	Domain
''	INT	required		-
'Age'	FLOAT	required		-
'Attrition'	INT	required		-
'BusinessTravel'	INT	required		-
'DailyRate'	FLOAT	required		-
'Department'	INT	required		-
'DistanceFromHome'	FLOAT	required		-
'Education'	FLOAT	required		-
'EducationField'	INT	required		-
'EmployeeNumber'	FLOAT	required		-
'EnvironmentSatisfaction'	FLOAT	required		-
'Gender'	INT	required		-
'HourlyRate'	FLOAT	required		-
'JobInvolvement'	FLOAT	required		-
'JobLevel'	FLOAT	required		-
'JobRole'	INT	required		-
'JobSatisfaction'	FLOAT	required		-
'MaritalStatus'	INT	required		-
'MonthlyIncome'	FLOAT	required		-
'MonthlyRate'	FLOAT	required		-
'NumCompaniesWorked'	FLOAT	required		-
'OverTime'	INT	required		-

```
trainer = Trainer(
    module_file=_trainer_module_file,
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    train_args=trainer_pb2.TrainArgs(num_steps=1000),
    eval_args=trainer_pb2.EvalArgs(num_steps=5)
)

context.run(trainer)
```

```
INFO:absl:Feature PerformanceRating has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature RelationshipSatisfaction has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature StockOptionLevel has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature TotalWorkingYears has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature TrainingTimesLastYear has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature WorkLifeBalance has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature YearsAtCompany has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature YearsInCurrentRole has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature YearsSinceLastPromotion has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature YearsWithCurrManager has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature Age has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature Attrition has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature BusinessTravel has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature DailyRate has a shape dim {
  size: 1
}
. Setting to DenseTensor.
INFO:absl:Feature Department has a shape dim {
```