H20 AutoML Employee Attrition

Answer the following questions for all of the models:

Is the relationship significant?

Are any model assumptions violated?

Is there any multicollinearity in the model?

In the multivariate models are predictor variables independent of all the other predictor variables?

In in multivariate models rank the most significant predictor variables and exclude insignificant ones from the model.

Does the model make sense?

Does regularization help?

Which independent variables are significant?

Which hyperparameters are important?

About the Dataset

The dataset provided is a fictional dataset created by IBM data scientists to simulate employee attrition. It encompasses various factors and attributes associated with employees, ranging from demographic information to job-related aspects. This dataset is designed for data science and analytical purposes to explore the underlying factors contributing to employee attrition within a hypothetical organization

Abstract

The dataset aims to shed light on the intricate dynamics of employee attrition, offering a comprehensive set of features that capture different facets of an employee's professional and personal life. It includes variables such as education level, environmental satisfaction, job involvement, job satisfaction, performance rating, relationship satisfaction, and work-life balance.

The educational background of employees is categorized into five levels: 'Below College,' 'College,' 'Bachelor,' 'Master,' and 'Doctor.' Various aspects of job satisfaction, such as environmental satisfaction, job involvement, job satisfaction, performance rating, relationship satisfaction, and work-life balance, are quantified on different scales.

The dataset presents an opportunity to investigate correlations and patterns within these attributes and their impact on employee attrition. It encourages exploratory data analysis and the application of machine learning models to predict and understand the likelihood of attrition

based on the provided features.

Researchers and data scientists can leverage this dataset to address specific questions related to attrition, such as examining the breakdown of distance from home by job role and attrition, or comparing average monthly income based on education and attrition status. The simulated nature of the data allows for a controlled environment for experimentation and analysis, providing valuable insights that can be applied to real-world scenarios in talent management and employee retention strategies.

Importing required Libraries and H20 Initialization Automated machine learning (AutoML) is the process of automating the end-to-end process of applying machine learning to real-world problems.

H2O AutoML automates the steps like basic data processing, model training and tuning, Ensemble and stacking of various models to provide the models with the best performance so that developers can focus on other steps like data collection, feature engineering and deployment of model.

We are initializing H2O in the following steps.

Installing H2O

In [55]:
```
pip install nbconvert[webpdf]
```

```
Requirement already satisfied: nbconvert[webpdf] in d:\data\ac\lib\site-packa
ges (6.5.4)
Requirement already satisfied: lxml in d:\data\ac\lib\site-packages (from nbc
onvert[webpdf]) (4.9.3)
Requirement already satisfied: beautifulsoup4 in d:\data\ac\lib\site-packages
(from nbconvert[webpdf]) (4.12.2)
Requirement already satisfied: bleach in d:\data\ac\lib\site-packages (from n
bconvert[webpdf]) (4.1.0)
Requirement already satisfied: defusedxml in d:\data\ac\lib\site-packages (fr
om nbconvert[webpdf]) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in d:\data\ac\lib\site-pack
ages (from nbconvert[webpdf]) (0.4)
Requirement already satisfied: jinja2>=3.0 in d:\data\ac\lib\site-packages (f
rom nbconvert[webpdf]) (3.1.2)
Requirement already satisfied: jupyter-core>=4.7 in d:\data\ac\lib\site-packa
ges (from nbconvert[webpdf]) (5.3.0)
Requirement already satisfied: jupyterlab-pygments in d:\data\ac\lib\site-pac
kages (from nbconvert[webpdf]) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in d:\data\ac\lib\site-package
s (from nbconvert[webpdf]) (2.1.1)
Requirement already satisfied: mistune<2,>=0.8.1 in d:\data\ac\lib\site-packa
ges (from nbconvert[webpdf]) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in d:\data\ac\lib\site-package
s (from nbconvert[webpdf]) (0.5.13)
Requirement already satisfied: nbformat>=5.1 in d:\data\ac\lib\site-packages
(from nbconvert[webpdf]) (5.9.2)
Requirement already satisfied: packaging in d:\data\ac\lib\site-packages (fro
m nbconvert[webpdf]) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in d:\data\ac\lib\site-pa
ckages (from nbconvert[webpdf]) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in d:\data\ac\lib\site-package
s (from nbconvert[webpdf]) (2.15.1)
Requirement already satisfied: tinycss2 in d:\data\ac\lib\site-packages (from
nbconvert[webpdf]) (1.2.1)
Requirement already satisfied: traitlets>=5.0 in d:\data\ac\lib\site-packages
(from nbconvert[webpdf]) (5.7.1)
Collecting pyppeteer<1.1,>=1 (from nbconvert[webpdf])
  Obtaining dependency information for pyppeteer<1.1,>=1 from https://files.p
ythonhosted.org/packages/10/46/33c0a9e7d37bf33487074de4399963462093043ad224d1
881e41cbd937f3/pyppeteer-1.0.2-py3-none-any.whl.metadata (https://files.pytho
nhosted.org/packages/10/46/33c0a9e7d37bf33487074de4399963462093043ad224d1881e
41cbd937f3/pyppeteer-1.0.2-py3-none-any.whl.metadata)
  Downloading pyppeteer-1.0.2-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: platformdirs>=2.5 in d:\data\ac\lib\site-packa
ges (from jupyter-core>=4.7->nbconvert[webpdf]) (3.10.0)
Requirement already satisfied: pywin32>=300 in d:\data\ac\lib\site-packages
(from jupyter-core>=4.7->nbconvert[webpdf]) (305.1)
Requirement already satisfied: jupyter-client>=6.1.5 in d:\data\ac\lib\site-p
ackages (from nbclient>=0.5.0->nbconvert[webpdf]) (7.4.9)
Requirement already satisfied: nest-asyncio in d:\data\ac\lib\site-packages
(from nbclient>=0.5.0->nbconvert[webpdf]) (1.5.6)
Requirement already satisfied: fastjsonschema in d:\data\ac\lib\site-packages
(from nbformat>=5.1->nbconvert[webpdf]) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in d:\data\ac\lib\site-package
s (from nbformat>=5.1->nbconvert[webpdf]) (4.17.3)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in d:\data\ac\lib\site-p
ackages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (1.4.4)
```

```
Requirement already satisfied: certifi>=2021 in d:\data\ac\lib\site-packages
(from pyppeteer<1.1,>=1->nbconvert[webpdf]) (2023.7.22)
Requirement already satisfied: importlib-metadata>=1.4 in d:\data\ac\lib\site
-packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (6.0.0)
Collecting pyee<9.0.0,>=8.1.0 (from pyppeteer<1.1,>=1->nbconvert[webpdf])
  Obtaining dependency information for pyee<9.0.0,>=8.1.0 from https://files.
pythonhosted.org/packages/56/37/29d137df23ed1d88d8dcee8a6b8e789d1162042f194b5
ccd0a48f503429b/pyee-8.2.2-py2.py3-none-any.whl.metadata (https://files.pytho
nhosted.org/packages/56/37/29d137df23ed1d88d8dcee8a6b8e789d1162042f194b5ccd0a
48f503429b/pyee-8.2.2-py2.py3-none-any.whl.metadata)
  Downloading pyee-8.2.2-py2.py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in d:\data\ac\lib\site-pac
kages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (4.65.0)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in d:\data\ac\lib\site-
packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (1.26.16)
Collecting websockets<11.0,>=10.0 (from pyppeteer<1.1,>=1->nbconvert[webpdf])
  Obtaining dependency information for websockets<11.0,>=10.0 from https://fi
les.pythonhosted.org/packages/27/bb/6327e8c7d4dd7d5b450b409a461be278968ce05c5
4da13da581ac87661db/websockets-10.4-cp311-cp311-win_amd64.whl.metadata (http
s://files.pythonhosted.org/packages/27/bb/6327e8c7d4dd7d5b450b409a461be278968
ce05c54da13da581ac87661db/websockets-10.4-cp311-cp311-win_amd64.whl.metadata)
  Downloading websockets-10.4-cp311-cp311-win_amd64.whl.metadata (6.4 kB)
Requirement already satisfied: soupsieve>1.2 in d:\data\ac\lib\site-packages
(from beautifulsoup4->nbconvert[webpdf]) (2.4)
Requirement already satisfied: six>=1.9.0 in d:\data\ac\lib\site-packages (fr
om bleach->nbconvert[webpdf]) (1.16.0)
Requirement already satisfied: webencodings in d:\data\ac\lib\site-packages
(from bleach->nbconvert[webpdf]) (0.5.1)
Requirement already satisfied: zipp>=0.5 in d:\data\ac\lib\site-packages (fro
m importlib-metadata>=1.4->pyppeteer<1.1,>=1->nbconvert[webpdf]) (3.11.0)
Requirement already satisfied: attrs>=17.4.0 in d:\data\ac\lib\site-packages
(from jsonschema>=2.6->nbformat>=5.1->nbconvert[webpdf]) (22.1.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0
in d:\data\ac\lib\site-packages (from jsonschema>=2.6->nbformat>=5.1->nbconve
rt[webpdf]) (0.18.0)
Requirement already satisfied: python-dateutil>=2.8.2 in d:\data\ac\lib\site-
packages (from jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert[webpdf]) (2.
8.2)
Requirement already satisfied: pyzmq>=23.0 in d:\data\ac\lib\site-packages (f
rom jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert[webpdf]) (23.2.0)
Requirement already satisfied: tornado>=6.2 in d:\data\ac\lib\site-packages
(from jupyter-client>=6.1.5->nbclient>=0.5.0->nbconvert[webpdf]) (6.3.2)
Requirement already satisfied: colorama in d:\data\ac\lib\site-packages (from
tqdm<5.0.0,>=4.42.1->pyppeteer<1.1,>=1->nbconvert[webpdf]) (0.4.6)
Downloading pyppeteer-1.0.2-py3-none-any.whl (83 kB)
   ---------------------------------------- 0.0/83.4 kB ? eta -:--:--
   -------------- ------------------------- 30.7/83.4 kB 660.6 kB/s eta 0:00:
01
   ----------------------------- ---------- 61.4/83.4 kB 656.4 kB/s eta 0:00:
01
   ---------------------------------------- 83.4/83.4 kB 671.0 kB/s eta 0:00:
00
Downloading pyee-8.2.2-py2.py3-none-any.whl (12 kB)
Downloading websockets-10.4-cp311-cp311-win_amd64.whl (101 kB)
   ---------------------------------------- 0.0/101.4 kB ? eta -:--:--
   ---------------------------------------- 101.4/101.4 kB 2.9 MB/s eta 0:00:
00
```

```
Installing collected packages: pyee, websockets, pyppeteer
Successfully installed pyee-8.2.2 pyppeteer-1.0.2 websockets-10.4
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
!pip install h2o
```

```
Requirement already satisfied: h2o in d:\data\ac\lib\site-packages (3.44.0.3)
Requirement already satisfied: requests in d:\data\ac\lib\site-packages (from
h2o) (2.31.0)
Requirement already satisfied: tabulate in d:\data\ac\lib\site-packages (from
h2o) (0.8.10)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\data\ac\lib\sit
e-packages (from requests->h2o) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in d:\data\ac\lib\site-packages
(from requests->h2o) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\data\ac\lib\site-pack
ages (from requests->h2o) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in d:\data\ac\lib\site-pack
ages (from requests->h2o) (2023.7.22)
```

In [3]:
```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in d:\data\ac\lib\site-packages
(1.5)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in d:\data\ac\lib\si
te-packages (from yellowbrick) (3.7.2)
Requirement already satisfied: scipy>=1.0.0 in d:\data\ac\lib\site-packages
(from yellowbrick) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.0 in d:\data\ac\lib\site-pac
kages (from yellowbrick) (1.4.1.post1)
Requirement already satisfied: numpy>=1.16.0 in d:\data\ac\lib\site-packages
(from yellowbrick) (1.24.3)
Requirement already satisfied: cycler>=0.10.0 in d:\data\ac\lib\site-packages
(from yellowbrick) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in d:\data\ac\lib\site-packag
es (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.0.5)
Requirement already satisfied: fonttools>=4.22.0 in d:\data\ac\lib\site-packa
ges (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\data\ac\lib\site-packa
ges (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.4)
Requirement already satisfied: packaging>=20.0 in d:\data\ac\lib\site-package
s (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (23.1)
Requirement already satisfied: pillow>=6.2.0 in d:\data\ac\lib\site-packages
(from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in d:\data\ac\lib\site-p
ackages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in d:\data\ac\lib\site-pa
ckages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: joblib>=1.2.0 in d:\data\ac\lib\site-packages
(from scikit-learn>=1.0.0->yellowbrick) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\data\ac\lib\site-pa
ckages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: six>=1.5 in d:\data\ac\lib\site-packages (from
python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
```

In [4]:
```
# Importing all the required libraries required for the assignment
import h2o
from h2o.automl import H2OAutoML
import random, os, sys
from datetime import datetime
import pandas as pd
import logging
import csv
import optparse
import time
import json
from distutils.util import strtobool
import psutil
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set(context="notebook", palette="Spectral", style = 'darkgrid' ,font_scale
import warnings
warnings.filterwarnings('ignore')
import os
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
from sklearn.model_selection import train_test_split as tts
from statsmodels.stats.outliers_influence import variance_inflation_factor
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LassoCV
from sklearn.linear_model import RidgeCV
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from yellowbrick.regressor import ResidualsPlot
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
```

In [5]:
```
%matplotlib inline
import random, os, sys
import h2o
import pandas
import pprint
import operator
import matplotlib
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
from h2o.estimators.gbm import H2OGradientBoostingEstimator
from h2o.estimators.random_forest import H2ORandomForestEstimator
from h2o.estimators.deeplearning import H2ODeepLearningEstimator
from tabulate import tabulate
from h2o.automl import H2OAutoML
from datetime import datetime
import logging
import csv
import optparse
import time
import json
from distutils.util import strtobool
import psutil
import numpy as np
```

In [6]:
```
min_mem_size=6
run_time=222
```

In [7]:
```
#calculates the minimum memory size in gigabytes (GB) based on a specified per
pct_memory=0.5
virtual_memory=psutil.virtual_memory()
min_mem_size=int(round(int(pct_memory*virtual_memory.available)/1073741824,0))
print(min_mem_size)
```

```
1
```

In [8]:
```
import os
import sys
import logging
import random
import h2o

# Set JAVA_HOME to the directory where Amazon Corretto is installed
os.environ["JAVA_HOME"] = "D:\data\jdk17.0.8_8"

# Add the bin directory of Amazon Corretto to the PATH environment variable
os.environ["PATH"] += os.pathsep + os.path.join(os.environ["JAVA_HOME"], "bin"

# Generate a random port number
port_no = random.randint(5555, 55555)

try:
    # Initialize H2O
    h2o.init(strict_version_check=False, min_mem_size_GB=min_mem_size, port=po
except:
    # Handle initialization failure
    logging.critical('h2o.init')
    h2o.download_all_logs(dirname=logs_path, filename=logfile)
    h2o.cluster().shutdown()
    sys.exit(2)
```

```
Checking whether there is an H2O instance running at http://localhost:4293
2..... (http://localhost:42932.....) not found.
Attempting to start a local H2O server...
; OpenJDK 64-Bit Server VM Corretto-17.0.8.8.1 (build 17.0.8.1+8-LTS, mixed m
ode, sharing)
  Starting server from D:\data\AC\Lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\LENOVO\AppData\Local\Temp\tmpl0i4oqyl
  JVM stdout: C:\Users\LENOVO\AppData\Local\Temp\tmpl0i4oqyl\h2o_LENOVO_start
ed_from_python.out
  JVM stderr: C:\Users\LENOVO\AppData\Local\Temp\tmpl0i4oqyl\h2o_LENOVO_start
ed_from_python.err
  Server is running at http://127.0.0.1:42932 (http://127.0.0.1:42932)
Connecting to H2O server at http://127.0.0.1:42932 (http://127.0.0.1:42932)
... successful.
```

H2O_cluster_uptime:	08 secs
H2O_cluster_timezone:	America/New_York
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.44.0.3
H2O_cluster_version_age:	2 months and 26 days
H2O_cluster_name:	H2O_from_python_LENOVO_4lmvfi
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	1.982 Gb
H2O_cluster_total_cores:	8
H2O_cluster_allowed_cores:	8
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:42932
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.11.5 final

In [9]:
```
# Import the processed data from notebook One
url = 'https://raw.githubusercontent.com/mukuldesai/DS/main/HR%20Employee%20At
df = h2o.import_file(path = url)
dff = pd.read_csv('https://raw.githubusercontent.com/mukuldesai/DS/main/HR%20E
```

```
Parse progress: |████████████████████████████████████████████████████████
████| (done) 100%
```

In [10]:
```
dff.head()
```

Out[10]:

Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationshipSatisfaction	StandardI
2	Life Sciences	1	1	...	1	
1	Life Sciences	1	2	...	4	
2	Other	1	4	...	2	
4	Life Sciences	1	5	...	3	
1	Medical	1	7	...	4	

In [11]:
```
dff.describe()
```

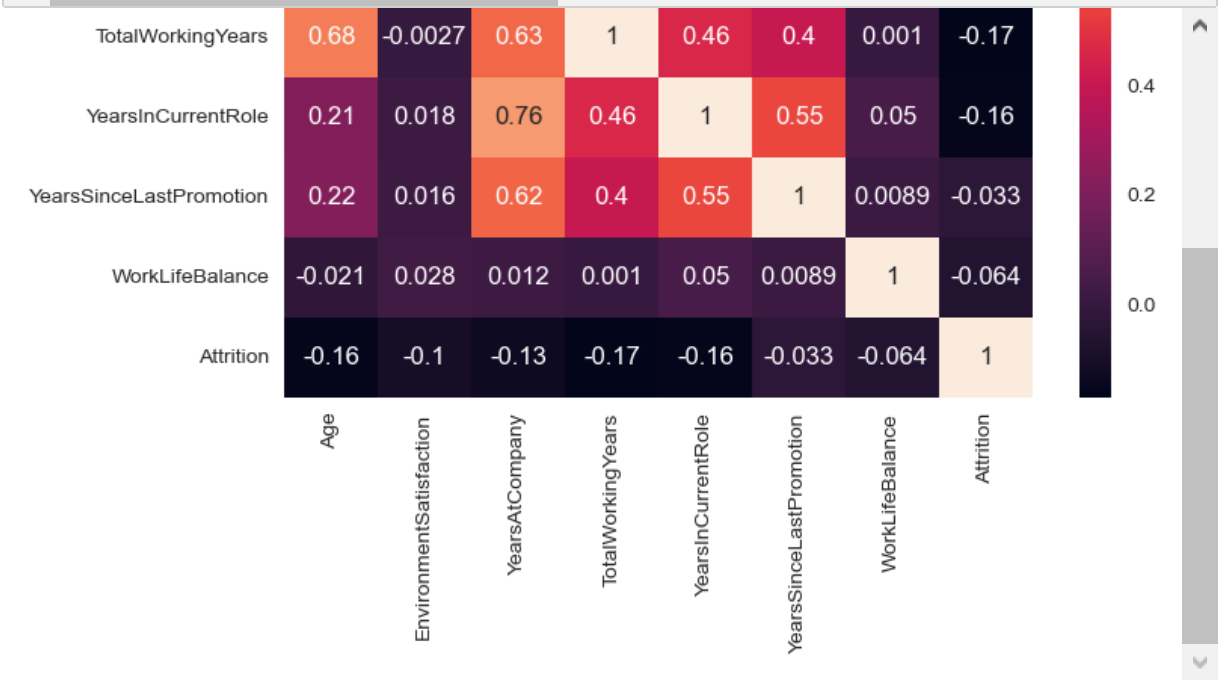
Out[11]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNur
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.00
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.86
std	9.135373	403.509100	8.106864	1.024165	0.0	602.02
min	18.000000	102.000000	1.000000	1.000000	1.0	1.00
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.25
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.50
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.75
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.00

8 rows × 26 columns

Provides summary statistics for each numerical column in the DataFrame, including count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum values.

In [12]:
```
mport pandas as pd
mport seaborn as sns

 Load the dataset from the URL
rl = 'https://raw.githubusercontent.com/mukuldesai/DS/main/HR%20Employee%20Attr
ata = pd.read_csv(url)

 Convert 'Attrition' column to binary (0 or 1)
ata['Attrition'] = data['Attrition'].map({'No': 0, 'Yes': 1})

 Select the required columns for correlation calculation
ata_for_corr = data[['Age', 'EnvironmentSatisfaction', 'YearsAtCompany', 'Total

 Plot the heatmap
ns.heatmap(data_for_corr.corr(), annot=True)
```



In [13]:
```
data_for_corr
```

Out[13]:

	Age	EnvironmentSatisfaction	YearsAtCompany	TotalWorkingYears	YearsInCurrentRole	Ye
0	41	2	6	8	4	
1	49	3	10	10	7	
2	37	4	0	7	0	
3	33	4	8	8	7	
4	27	1	2	6	2	
...	...	...	...	...	...	
1465	36	3	5	17	2	
1466	39	4	7	9	7	
1467	27	2	6	6	2	
1468	49	4	9	17	6	
1469	34	2	4	6	3	

1470 rows × 8 columns

H2O ML Algorithm

We start the H2O Algorithm and run the training models

In [14]:
```
#Connect to a cluster or initialize it if not started
h2o.init(strict_version_check=False)
```

```
Checking whether there is an H2O instance running at http://localhost:5432
1..... (http://localhost:54321.....) not found.
Attempting to start a local H2O server...
; OpenJDK 64-Bit Server VM Corretto-17.0.8.8.1 (build 17.0.8.1+8-LTS, mixed m
ode, sharing)
  Starting server from D:\data\AC\Lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\LENOVO\AppData\Local\Temp\tmpw2hhkou1
  JVM stdout: C:\Users\LENOVO\AppData\Local\Temp\tmpw2hhkou1\h2o_LENOVO_start
ed_from_python.out
  JVM stderr: C:\Users\LENOVO\AppData\Local\Temp\tmpw2hhkou1\h2o_LENOVO_start
ed_from_python.err
  Server is running at http://127.0.0.1:54321 (http://127.0.0.1:54321)
Connecting to H2O server at http://127.0.0.1:54321 (http://127.0.0.1:54321)
... successful.
```

H2O_cluster_uptime:	06 secs
H2O_cluster_timezone:	America/New_York
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.44.0.3
H2O_cluster_version_age:	2 months and 26 days
H2O_cluster_name:	H2O_from_python_LENOVO_qvsy01
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	1.961 Gb
H2O_cluster_total_cores:	8
H2O_cluster_allowed_cores:	8
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.11.5 final

In [15]:
```
# set this to True if interactive (matplotlib) plots are desired
interactive = True
if not interactive: matplotlib.use('Agg', warn=False)
import matplotlib.pyplot as plt
```

In [16]:
```
# Adding the imported variable from Panda to H2O
datahf = h2o.H2OFrame(data_for_corr)
```

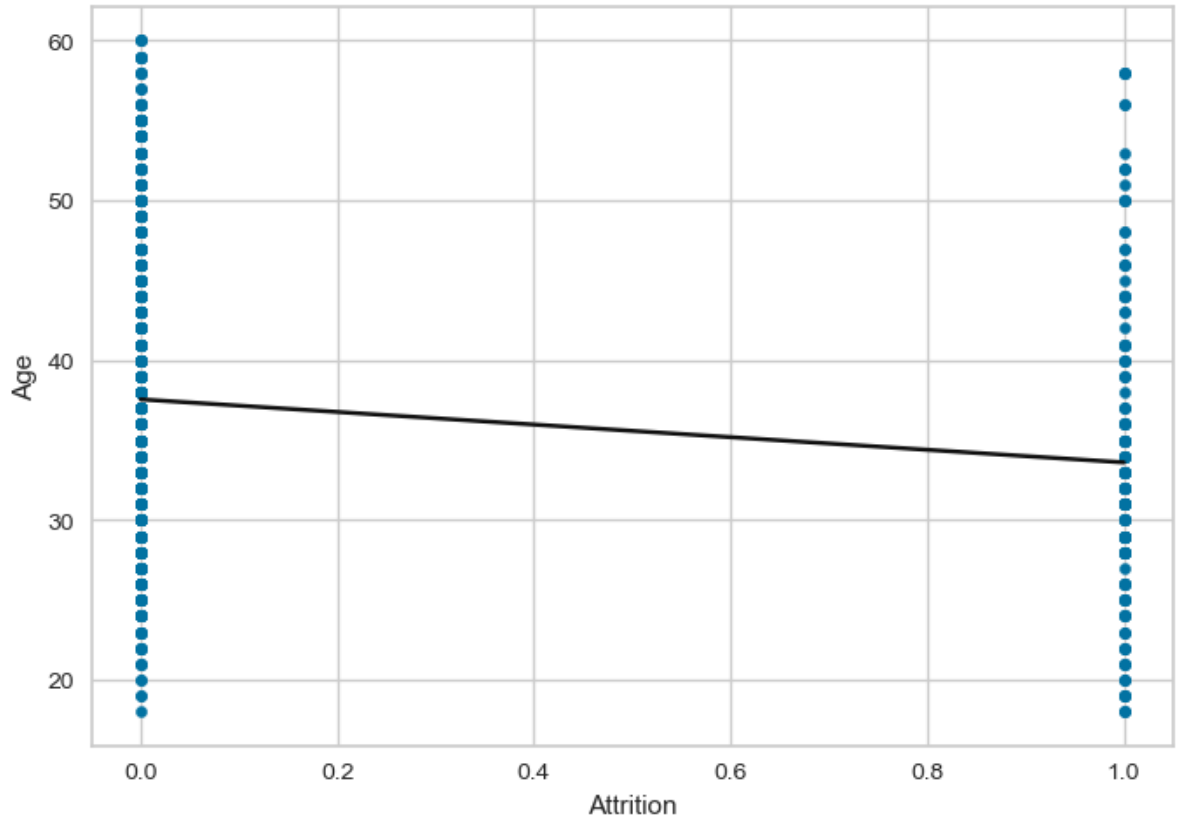
```
Parse progress: |
| (done) 100%
```

In [17]:
```
# Data exploration and munging. Generate scatter plots
data = datahf
def scatter_plot(data, x, y, max_points = 1000, fit = True):
    if(fit):
        lr = H2OGeneralizedLinearEstimator(family = "gaussian")
        lr.train(x=x, y=y, training_frame=data)
        coeff = lr.coef()
    df = data[[x,y]]
    runif = df[y].runif()
    df_subset = df[runif < float(max_points)/datahf.nrow]
    df_py = h2o.as_list(df_subset)

    if(fit): h2o.remove(lr._id)

    # If x variable is string, generate box-and-whisker plot
    if(df_py[x].dtype == "object"):
        if interactive: df_py.boxplot(column = y, by = x)
    # Otherwise, generate a scatter plot
    else:
        if interactive: df_py.plot(x = x, y = y, kind = "scatter")

    if(fit):
        x_min = min(df_py[x])
        x_max = max(df_py[x])
        y_min = coeff["Intercept"] + coeff[x]*x_min
        y_max = coeff["Intercept"] + coeff[x]*x_max
        plt.plot([x_min, x_max], [y_min, y_max], "k-")
    if interactive: plt.show()
```

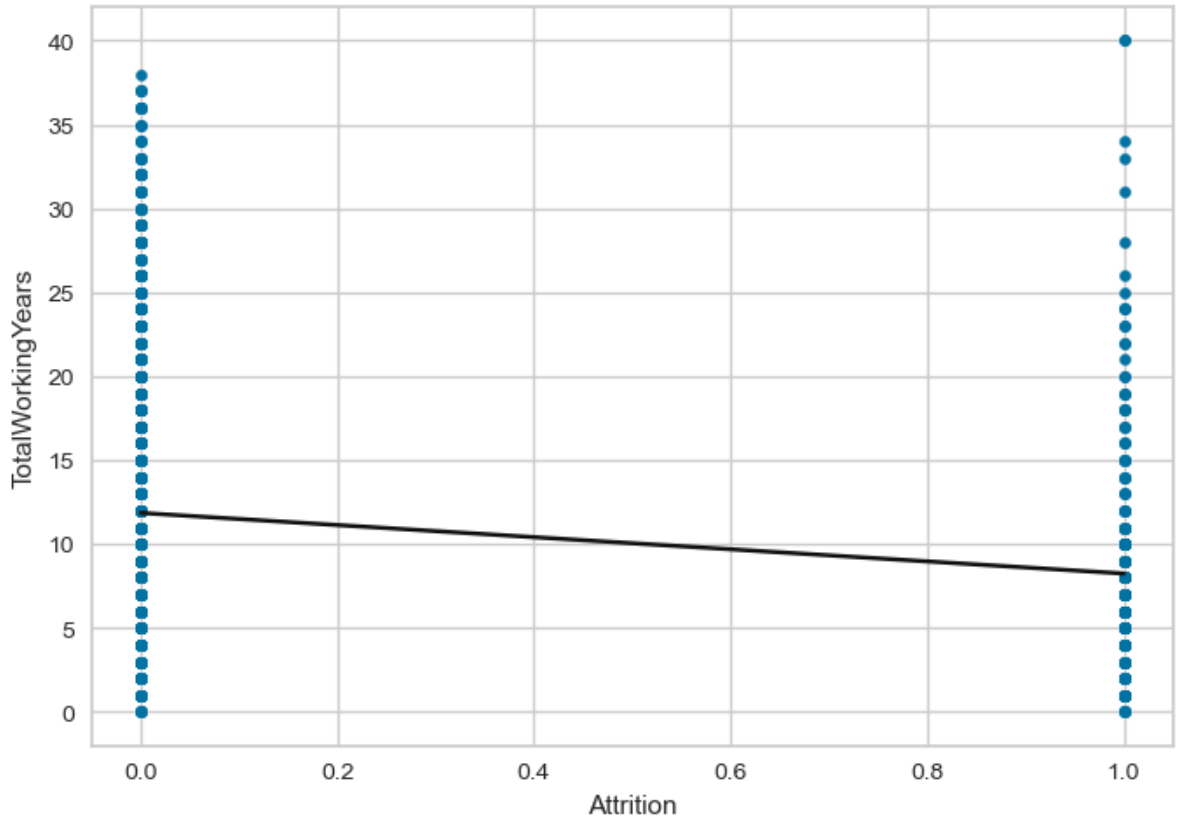
In [18]:
```
from h2o.estimators import H2OGeneralizedLinearEstimator
```

In [19]:
```
scatter_plot(data, "Attrition", "Age", fit = True)
scatter_plot(data, "Attrition", "TotalWorkingYears", max_points = 2000, fit = 
scatter_plot(data, "Attrition", "YearsAtCompany", max_points = 2000, fit = Tru
```

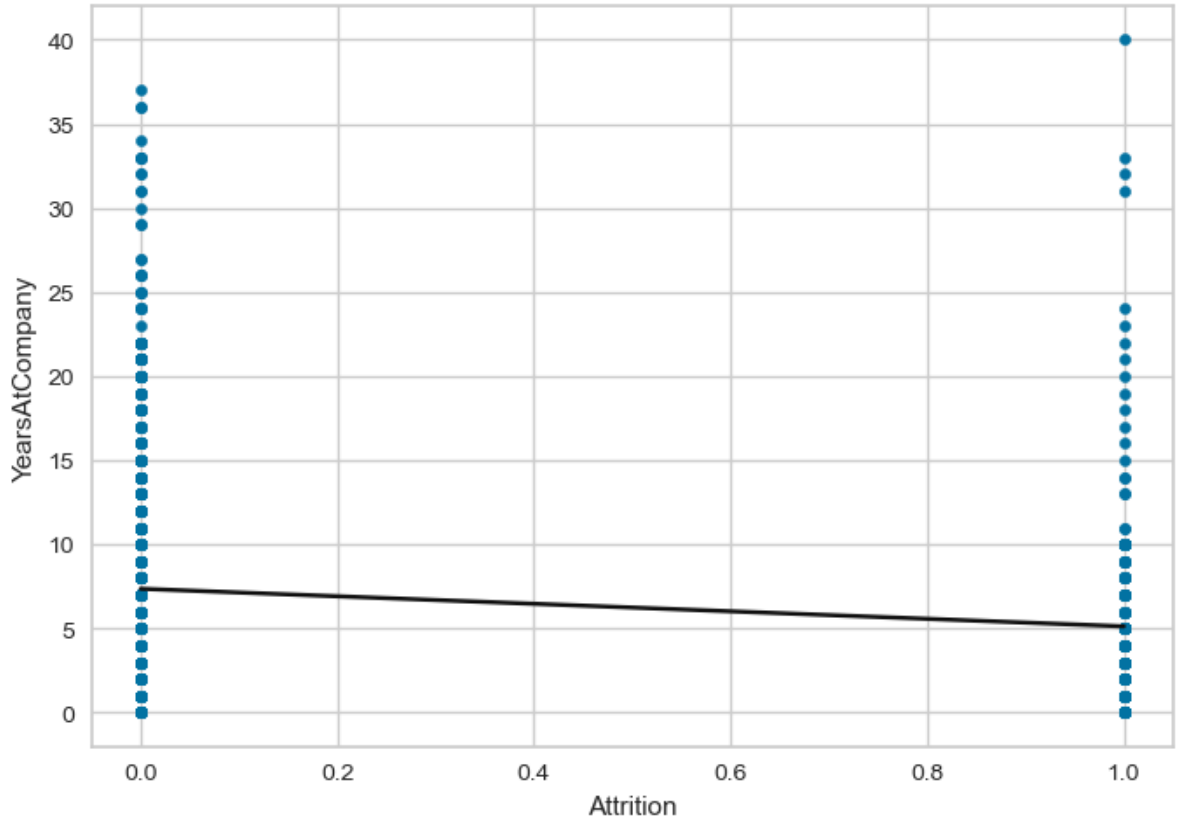
```
glm Model Build progress: |████████████████████████████████████████████
████| (done) 100%
```




```
glm Model Build progress: |████████████████████████████████████████████
████| (done) 100%
```




```
glm Model Build progress: |████████████████████████████████████████
████| (done) 100%
```



This Scatter Plot shows the relation between the dependent variable Attrition and the Independent variables TotalWorkingYears , Age and YearsAtCompany

In [20]:
```
data.describe()
```

Rows:1470
Cols:8

	Age	EnvironmentSatisfaction	YearsAtCompany	TotalWorkingYears	Yea
type	int	int	int	int	
mins	18.0	1.0	0.0	0.0	
mean	36.92380952380955	2.7217687074829935	7.008163265306127	11.279591836734703	4.2
maxs	60.0	4.0	40.0	40.0	
sigma	9.13537348913673	1.0930822146350003	6.126525152403571	7.780781675514995	3.6
zeros	0	0	44	11	
missing	0	0	0	0	
0	41.0	2.0	6.0	8.0	
1	49.0	3.0	10.0	10.0	
2	37.0	4.0	0.0	7.0	
3	33.0	4.0	8.0	8.0	
4	27.0	1.0	2.0	6.0	
5	32.0	4.0	7.0	8.0	
6	59.0	3.0	1.0	12.0	
7	30.0	4.0	1.0	1.0	
8	38.0	4.0	9.0	10.0	
9	36.0	3.0	7.0	17.0	

[1470 rows x 8 columns]

In [21]:
```
# Create a test/train split
# Train will contain the 90% of data and test will contain 10% data
train,test = data.split_frame([.9])
```

In [22]:
```
#  Set response variable and your choice of predictor variables
# Dependant variable is Level and Predictor variables are Air Pollution and Pas
myY = "Attrition"
myX = ["Attrition","TotalWorkingYears","Age","YearsAtCompany","WorkLifeBalance
```

In [23]:
```
# Build simple GLM model
data_glm = H2OGeneralizedLinearEstimator(family="gaussian", standardize=True)
data_glm.train(x               =myX,
               y               =myY,
               training_frame  =train,
               validation_frame=test)
```

```
glm Model Build progress: |███████████████████████████████████████████
████| (done) 100%
```

Out[23]:

```
Model Details
=============
H2OGeneralizedLinearEstimator : Generalized Linear Modeling
Model Key: GLM_model_python_1710780530779_4
```

GLM Model: summary

family	link	regularization	number_of_predictors_total	number_of_active_predictors	numI
gaussian	identity	Elastic Net (alpha = 0.5, lambda = 1.216E-4)	4	4	

```
ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 0.1292391063573313
RMSE: 0.35949840939471667
MAE: 0.26032932149593735
RMSLE: 0.2526089869885033
Mean Residual Deviance: 0.1292391063573313
R^2: 0.03913040364035392
Null degrees of freedom: 1329
Residual degrees of freedom: 1325
Null deviance: 178.88796992481096
Residual deviance: 171.88801145525062
AIC: 1065.0753983026896

ModelMetricsRegressionGLM: glm
** Reported on validation data. **

MSE: 0.13673463511883766
RMSE: 0.36977646642104967
MAE: 0.26376415864477953
RMSLE: 0.2584225425624945
Mean Residual Deviance: 0.13673463511883766
R^2: 0.037356735513930595
Null degrees of freedom: 139
Residual degrees of freedom: 135
Null deviance: 19.903521962801726
Residual deviance: 19.14284891663727
AIC: 130.74294109547003
```

Scoring History:

timestamp	duration	iterations	negative_log_likelihood	objective	training_rmse	training_devi

timestamp	duration	iterations	negative_log_likelihood	objective	training_rmse	training_devi
2024-03-18 12:49:12	0.000 sec	0	178.8879699	0.1345022		
2024-03-18 12:49:12	0.072 sec	1			0.3594984	0.129

Variable Importances:

variable	relative_importance	scaled_importance	percentage
Age	0.0389446	1.0	0.3633818
YearsAtCompany	0.0265533	0.6818216	0.2477615
WorkLifeBalance	0.0237129	0.6088886	0.2212590
TotalWorkingYears	0.0179619	0.4612165	0.1675977

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

After running this code, data_glm will contain a trained GLM model, and it can make predictions on new data or evaluate its performance on the validation dataset.

This is a regression model, and its suitable for tasks where the goal is to predict a continuous numerical value (e.g., predicting house prices).

In [24]:
```
import h2o
from h2o.estimators import H2OGradientBoostingEstimator
```

In [25]:
```
# Build simple GBM model

data_gbm = H2OGradientBoostingEstimator(balance_classes=True,
                                        ntrees         =10,
                                        max_depth      =1,
                                        learn_rate     =0.1,
                                        min_rows       =2)

data_gbm.train(x               =myX,
               y               =myY,
               training_frame  =train,
               validation_frame=test)
```

```
gbm Model Build progress: |███████████████████████████████████████████████
████| (done) 100%
```

Out[25]:

```
Model Details
=============
H2OGradientBoostingEstimator : Gradient Boosting Machine
Model Key: GBM_model_python_1710780530779_5
```

Model Summary:

number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean
10.0	10.0	829.0	1.0	1.0	

```
ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 0.1262496288475213
RMSE: 0.3553162378044681
MAE: 0.2580237460203637
RMSLE: 0.24814113295033005
Mean Residual Deviance: 0.1262496288475213

ModelMetricsRegression: gbm
** Reported on validation data. **

MSE: 0.14030706642591817
RMSE: 0.374575848695452
MAE: 0.27390673509307656
RMSLE: 0.2614336192421072
Mean Residual Deviance: 0.14030706642591817
```

Scoring History:

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance	validatic
2024-03-18 12:49:13	0.080 sec	0.0	0.3667455	0.2690045	0.1345022	0.
2024-03-18 12:49:13	0.334 sec	1.0	0.3645300	0.2672991	0.1328821	0.
2024-03-18 12:49:13	0.387 sec	2.0	0.3627218	0.2657828	0.1315671	0.
2024-03-18 12:49:13	0.412 sec	3.0	0.3612074	0.2643800	0.1304708	0.
2024-03-18 12:49:13	0.437 sec	4.0	0.3599762	0.2631175	0.1295829	0.

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance	validatic
2024-03-18 12:49:13	0.460 sec	5.0	0.3589446	0.2619788	0.1288413	0.
2024-03-18 12:49:13	0.490 sec	6.0	0.3580633	0.2610476	0.1282093	0.
2024-03-18 12:49:13	0.517 sec	7.0	0.3572649	0.2603280	0.1276382	0.
2024-03-18 12:49:13	0.554 sec	8.0	0.3565486	0.2595380	0.1271269	0.
2024-03-18 12:49:13	0.589 sec	9.0	0.3559152	0.2588982	0.1266756	0.
2024-03-18 12:49:14	0.627 sec	10.0	0.3553162	0.2580237	0.1262496	0.

Variable Importances:

variable	relative_importance	scaled_importance	percentage
TotalWorkingYears	42.6087456	1.0	0.7375808
YearsAtCompany	8.0026197	0.1878164	0.1385298
Age	7.1568727	0.1679672	0.1238894
WorkLifeBalance	0.0	0.0	0.0

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

After running this code, data_gbm will contain a trained GBM model, and you can use it to make predictions on new data or evaluate its performance on the validation dataset.

Gradient Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (in this case, decision trees) to create a strong predictive model. It is often used for both classification and regression tasks.

In [26]:
```
# Variable importances from each algorithm
# Calculate magnitude of normalized GLM coefficients
import operator
import tabulate
from tabulate import tabulate
from six import iteritems
glm_varimp = data_glm.coef_norm()
for k,v in iteritems(glm_varimp):
    glm_varimp[k] = abs(glm_varimp[k])

# Sort in descending order by magnitude
glm_sorted = sorted(glm_varimp.items(), key = operator.itemgetter(1), reverse =
table = tabulate(glm_sorted, headers = ["Predictor", "Normalized Coefficient"]
print("Variable Importances:\n\n" + table)

data_glm.varimp()
data_gbm.varimp()
```

```
Variable Importances:

| Predictor         |   Normalized Coefficient |
|-------------------+--------------------------|
| Intercept         |                  0.16015 |
| Age               |                  0.0389446 |
| YearsAtCompany    |                  0.0265533 |
| WorkLifeBalance   |                  0.0237129 |
| TotalWorkingYears |                  0.0179619 |
```

Out[26]:
```
[('TotalWorkingYears', 42.60874557495117, 1.0, 0.7375808402719645),
 ('YearsAtCompany',
  8.002619743347168,
  0.18781636575688695,
  0.13852975287179128),
 ('Age', 7.156872749328613, 0.16796722486793875, 0.12388940685624426),
 ('WorkLifeBalance', 0.0, 0.0, 0.0)]
```

The code provided is related to variable importance analysis for machine learning models. It appears to be written in Python and utilizes the tabulate library for displaying the results in a tabular format.

glm_varimp = data_glm.coef_norm(): This line calculates the normalized coefficients for each predictor (feature) in a Generalized Linear Model (GLM). The GLM is likely the data_glm model you trained earlier. Normalized coefficients are a way to measure the importance of each predictor in making predictions. The result is stored in the glm_varimp variable.

for k,v in iteritems(glm_varimp):: This loop iterates through the items in the glm_varimp dictionary, where k is the predictor (feature) name, and v is its normalized coefficient.

glm_varimp[k] = abs(glm_varimp[k]): This line takes the absolute value of each normalized coefficient. It ensures that all variable importances are positive, making it easier to interpret.

glm_sorted = sorted(glm_varimp.items(), key = operator.itemgetter(1), reverse = True): This line sorts the predictor-importance pairs in descending order based on the absolute value of the normalized coefficient. It uses the operator.itemgetter(1) function to sort based on the second

element of each pair (the absolute value), and reverse = True indicates a descending order.

table = tabulate(glm_sorted, headers = ["Predictor", "Normalized Coefficient"], tablefmt = "orgtbl"): Here, the sorted results are tabulated with headers, and the table format is specified as "orgtbl."

print("Variable Importances:\n\n" + table): Finally, this line prints the table of variable importances, including predictor names and their corresponding normalized coefficients. The header "Variable Importances" is also included in the output.

data_glm.varimp(): This line calls a method (varimp) on the data_glm GLM model to get variable importances directly from the model. Variable importances represent the importance of predictors in making predictions in the model.

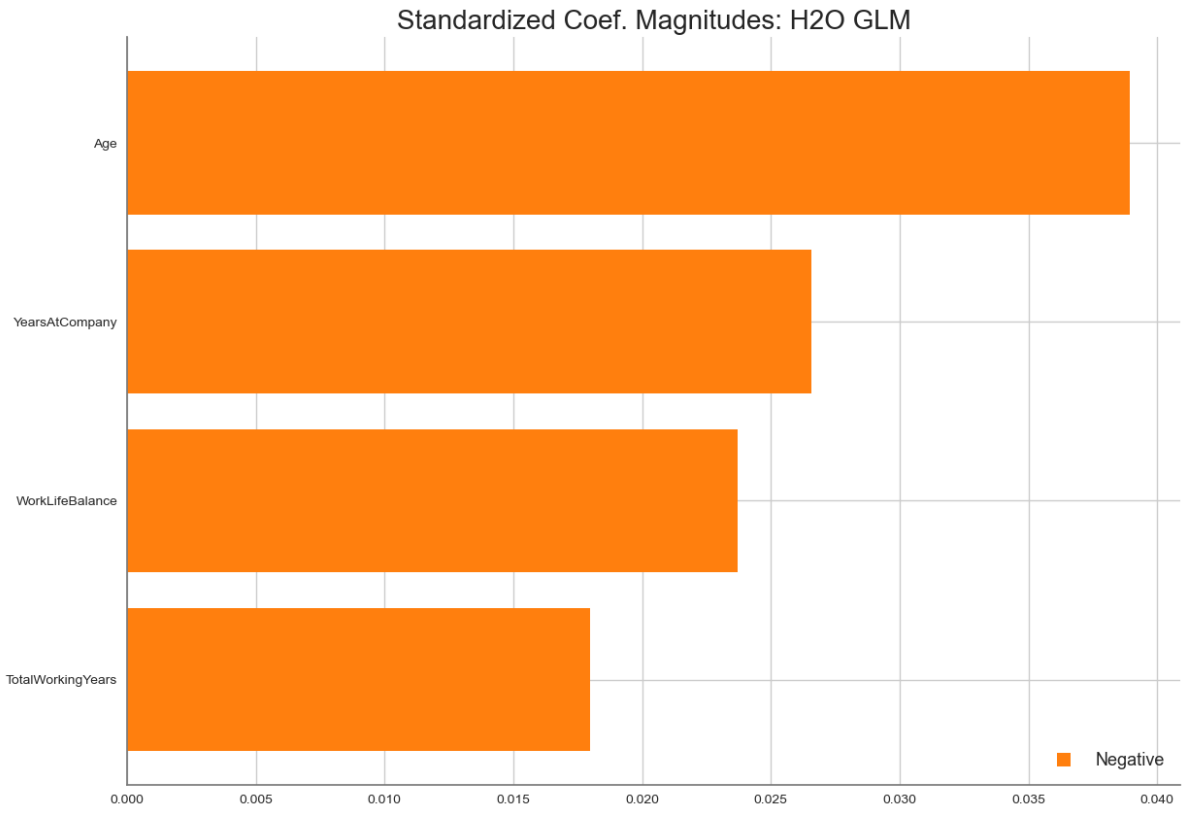
data_gbm.varimp(): Similarly, this line calls a method (varimp) on the data_gbm Gradient Boosting Machine (GBM) model to get variable importances from the GBM model.

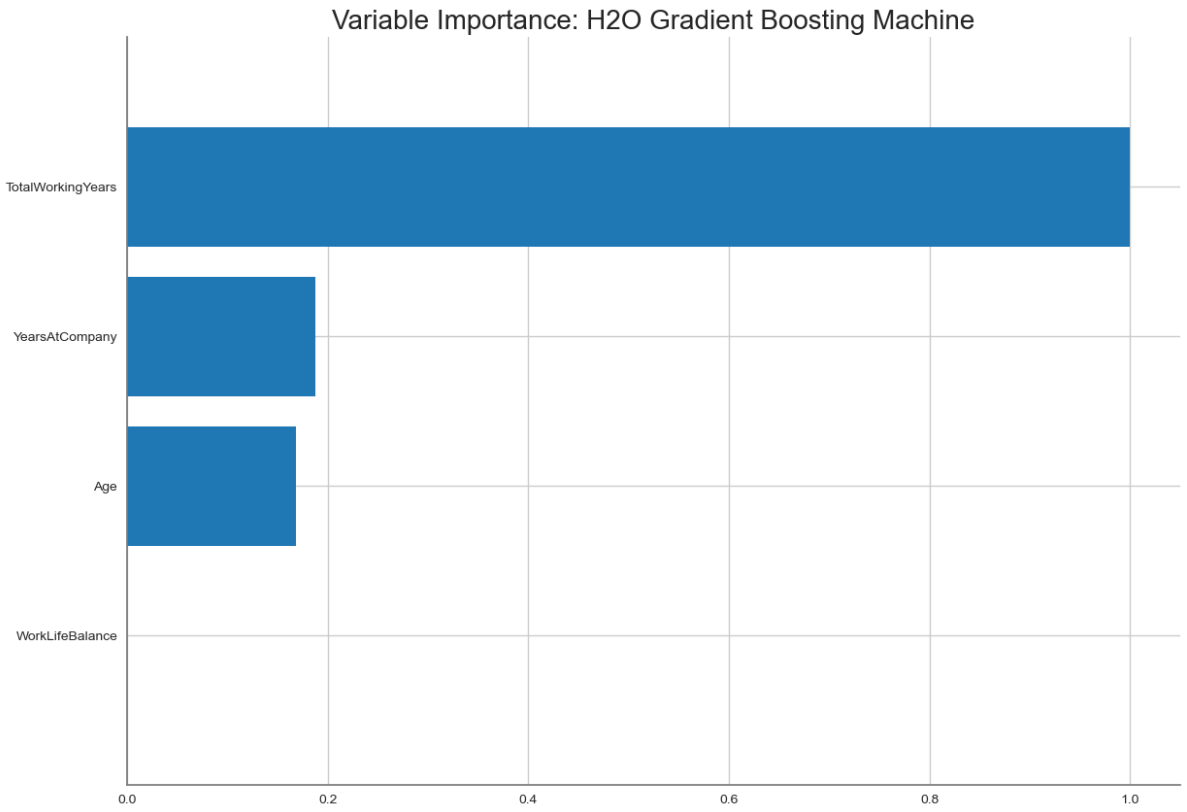
Variable importance analysis is essential for understanding which predictors have the most influence on a model's predictions. It can help in feature selection, model interpretation, and identifying the most relevant factors in a predictive model.

In [27]:
```
dir(data_glm)
```

Out[27]:
```
['HGLM',
 'Lambda',
 '_H2OEstimator__default_params',
 '_ModelBase__generate_partial_plots',
 '_ModelBase__generate_user_splits',
 '_ModelBase__grab_values',
 '_ModelBase__plot_1d_pdp',
 '_ModelBase__plot_1d_pdp_multinomial',
 '_ModelBase__plot_2d_pdp',
 '_ModelBase__pred_for_3d',
 '_ModelBase__set_axs_1d',
 '_ModelBase__set_axs_1d_multinomial',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
```

In [28]:
```
data_glm.std_coef_plot()
data_gbm.varimp_plot()
```





Out[28]:
```
<h2o.plot._plot_result._MObject at 0x17ecc33f7d0>

<Figure size 800x550 with 0 Axes>
```

The code is using two functions to generate plots that are common in machine learning model analysis. These plots can help you understand the importance of variables (predictors) in your models.

data_glm.std_coef_plot(): This line of code is calling the std_coef_plot() function on the data_glm Generalized Linear Model (GLM) object. Here's what it does:

data_glm: It's the GLM model that I trained. std_coef_plot(): This function is typically provided by machine learning libraries like H2O.ai or scikit-learn. It generates a standardized coefficient plot. Standardized coefficients allow you to compare the impact of different predictors on the target variable. The plot that is generated by this function will likely show the predictors on the x-axis and their standardized coefficients on the y-axis. The plot can help you understand which predictors have the most significant impact on the model's predictions.

data_gbm.varimp_plot(): This line is calling the varimp_plot() function on the data_gbm Gradient Boosting Machine (GBM) object. Here's what it does:

data_gbm: It's the GBM model that I trained. varimp_plot(): This function is used to generate a variable importance plot for GBM models. It visualizes the importance of each predictor in making predictions in the GBM model. The plot generated by this function is likely to show the predictors on the x-axis and their relative importance on the y-axis. It can help you understand which predictors have the most influence on the model's predictions and by how much.

Both of these plots are useful for model interpretation, feature selection, and identifying key predictors in your machine learning models. They provide insights into which variables have the most impact on the model's predictions, making it easier to make informed decisions about model features and potential improvements.

In [29]:
```
# Model performance of GBM model on test data
data_gbm.model_performance(test)
```

Out[29]:
```
ModelMetricsRegression: gbm
** Reported on test data. **

MSE: 0.14030706642591817
RMSE: 0.374575848695452
MAE: 0.27390673509307656
RMSLE: 0.2614336192421072
Mean Residual Deviance: 0.14030706642591817
```

The code data_gbm.model_performance(test) is used to assess the performance of a Gradient Boosting Machine (GBM) model, data_gbm, on a test dataset (test). By using data_gbm.model_performance(test), you can access all of this information to assess how well your GBM model is performing on the test dataset. This evaluation is crucial to understand how the model generalizes to new data and whether it's overfitting or underfitting.

In [30]:
```
# Reduced the dataset as Age is not parameter to be used
data=data[["Attrition","TotalWorkingYears","YearsAtCompany"]]
```

In [31]:
```
# Create a test/train split
#https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-munging/splitting-datasets
# Train will train 90% data and test will contain 10% data
train,test = data.split_frame([.9])
```

In [32]:
```
#  Set response variable and your choice of predictor variables
#myY target (Dependent variables)
#myX features (Independent variables)
myY = "Attrition"
myX = ["Attrition","TotalWorkingYears","YearsAtCompany"]
```

In [33]:
```
# Build simple GLM model
# Build simple GLM model
data_glm = H2OGeneralizedLinearEstimator(family="gaussian", standardize=True)
data_glm.train(x               =myX,
               y               =myY,
               training_frame  =train,
               validation_frame=test)
```

```
glm Model Build progress: |███████████████████████████████████████████████
████| (done) 100%
```

Out[33]:

```
Model Details
=============
H2OGeneralizedLinearEstimator : Generalized Linear Modeling
Model Key: GLM_model_python_1710780530779_6
```

GLM Model: summary

family	link	regularization	number_of_predictors_total	number_of_active_predictors	numI
gaussian	identity	Elastic Net (alpha = 0.5, lambda = 1.351E-4)	2	2	

```
ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 0.1293655635285233
RMSE: 0.35967424640711115
MAE: 0.26125606957806086
RMSLE: 0.2527277830122112
Mean Residual Deviance: 0.1293655635285233
R^2: 0.03643424374314963
Null degrees of freedom: 1332
Residual degrees of freedom: 1330
Null deviance: 178.96474118529426
Residual deviance: 172.44429618352154
AIC: 1064.7544251656484

ModelMetricsRegressionGLM: glm
** Reported on validation data. **

MSE: 0.1487750809521135
RMSE: 0.3857137292761479
MAE: 0.27420892041068323
RMSLE: 0.27120908221567164
Mean Residual Deviance: 0.1487750809521135
R^2: -0.02963108200229292
Null degrees of freedom: 136
Residual degrees of freedom: 134
Null deviance: 19.8280798631766
Residual deviance: 20.38218609043955
AIC: 135.76036778429096
```

Scoring History:

timestamp	duration	iterations	negative_log_likelihood	objective	training_rmse	training_devi

timestamp	duration	iterations	negative_log_likelihood	objective	training_rmse	training_devi
2024-03-18 12:49:17	0.000 sec	0	178.9647412	0.1342571		
2024-03-18 12:49:17	0.024 sec	1			0.3596742	0.129

Variable Importances:

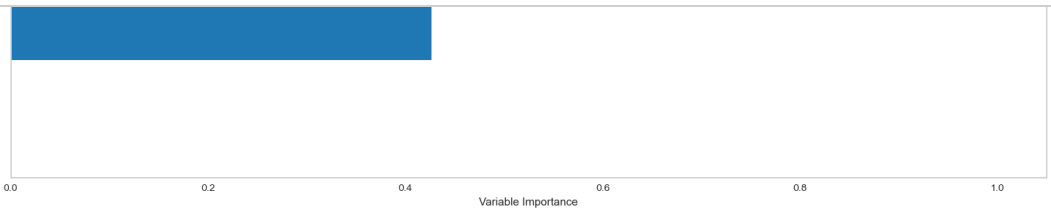
variable	relative_importance	scaled_importance	percentage
TotalWorkingYears	0.0534360	1.0	0.7009006
YearsAtCompany	0.0228030	0.4267359	0.2990994

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

In the code snippet you've provided, you are building a simple Generalized Linear Model (GLM) using the H2O library.

In this code, I have a trained GLM model in the data_glm variable. This model can be used to make predictions on new data and evaluate its performance using the validation set or other evaluation techniques. The model's coefficients will also be available for further analysis to understand the importance of each predictor variable in making predictions.

In [34]:
```
data_glm.explain(train[1:100,:])
```



Partial Dependence Plots

> Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

The data_glm.explain(train[1:100,:]) command is used to generate an explanation for the GLM (Generalized Linear Model) data_glm on a specific subset of the training data. Let's break down what this command does:

data_glm: This refers to your trained Generalized Linear Model.

.explain(): This is a method provided by the H2O library that is used to generate model explanations. Model explanations are a way to understand why a model makes specific predictions. They help provide insight into the relationship between input features and model predictions.

train[1:100, :]: This is a subset of the training data used for generating the explanation. [1:100, :] means that you are selecting rows 1 to 100 from the training data. This subset of data will be used to explain how the model makes predictions on these specific examples.

When I run this command, it will provide explanations for the model's predictions on the selected subset of the training data. These explanations can take various forms, depending on the library and the model, but common forms of model explanation include feature importance, partial dependence plots, and SHAP (SHapley Additive exPlanations) values.

The specific details of the explanations and how they are presented will depend on the H2O library's implementation and the parameters you set when calling the .explain() method. The goal of using explanations is to gain insights into why the model makes certain predictions, which can be helpful for model interpretation and debugging.

In [35]:
```
# Build simple GBM model

data_gbm = H2OGradientBoostingEstimator(balance_classes=True,
                                        ntrees         =10,
                                        max_depth      =1,
                                        learn_rate     =0.1,
                                        min_rows       =2)

data_gbm.train(x               =myX,
               y               =myY,
               training_frame  =train,
               validation_frame=test)
```

```
gbm Model Build progress: |███████████████████████████████████████████████
████| (done) 100%
```

Out[35]:

```
Model Details
=============
H2OGradientBoostingEstimator : Gradient Boosting Machine
Model Key: GBM_model_python_1710780530779_7
```

Model Summary:

number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean
10.0	10.0	830.0	1.0	1.0	

```
ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 0.12676520890288137
RMSE: 0.35604102137658433
MAE: 0.2587435537388874
RMSLE: 0.24861589639523257
Mean Residual Deviance: 0.12676520890288137

ModelMetricsRegression: gbm
** Reported on validation data. **

MSE: 0.14054082868704726
RMSE: 0.3748877547840784
MAE: 0.2680753496226111
RMSLE: 0.25976073160984636
Mean Residual Deviance: 0.14054082868704726
```

Scoring History:

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance	validatic
2024-03-18 12:49:52	0.029 sec	0.0	0.3664111	0.2685142	0.1342571	0.
2024-03-18 12:49:52	0.066 sec	1.0	0.3644755	0.2670251	0.1328424	0.
2024-03-18 12:49:52	0.102 sec	2.0	0.3629001	0.2656848	0.1316965	0.
2024-03-18 12:49:53	0.125 sec	3.0	0.3616189	0.2644785	0.1307683	0.
2024-03-18 12:49:53	0.142 sec	4.0	0.3605058	0.2634897	0.1299644	0.

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance	validatic
2024-03-18 12:49:53	0.155 sec	5.0	0.3595605	0.2624567	0.1292837	0.
2024-03-18 12:49:53	0.167 sec	6.0	0.3586834	0.2616511	0.1286538	0.
2024-03-18 12:49:53	0.178 sec	7.0	0.3578984	0.2607923	0.1280912	0.
2024-03-18 12:49:53	0.190 sec	8.0	0.3571952	0.2600891	0.1275884	0.
2024-03-18 12:49:53	0.203 sec	9.0	0.3565773	0.2593566	0.1271474	0.
2024-03-18 12:49:53	0.219 sec	10.0	0.3560410	0.2587436	0.1267652	0.

Variable Importances:

variable	relative_importance	scaled_importance	percentage
TotalWorkingYears	39.8813667	1.0	0.7587538
YearsAtCompany	12.6803064	0.3179506	0.2412462

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

The code provided is training a H2OGradientBoostingEstimator (a Gradient Boosting Machine) with the following settings:

balance_classes=True: This setting balances the class distribution in the training data to prevent the model from being biased towards the majority class in a binary classification problem.

ntrees=10: It specifies the number of trees (base learners) in the gradient boosting ensemble. In this case, the model will consist of 10 decision trees.

max_depth=1: This sets the maximum depth of each decision tree in the ensemble. A value of 1 means that the trees will be shallow, with a maximum of one split.

learn_rate=0.1: This is the learning rate, which controls the step size at each iteration when updating the model. A smaller learning rate makes the model converge more slowly but can result in better generalization.

min_rows=2: It specifies the minimum number of rows required in a node to make a split during the tree-building process. This helps to control the tree's complexity.

After specifying these settings, you train the gradient boosting model using the train method. Here's a breakdown of the training parameters:

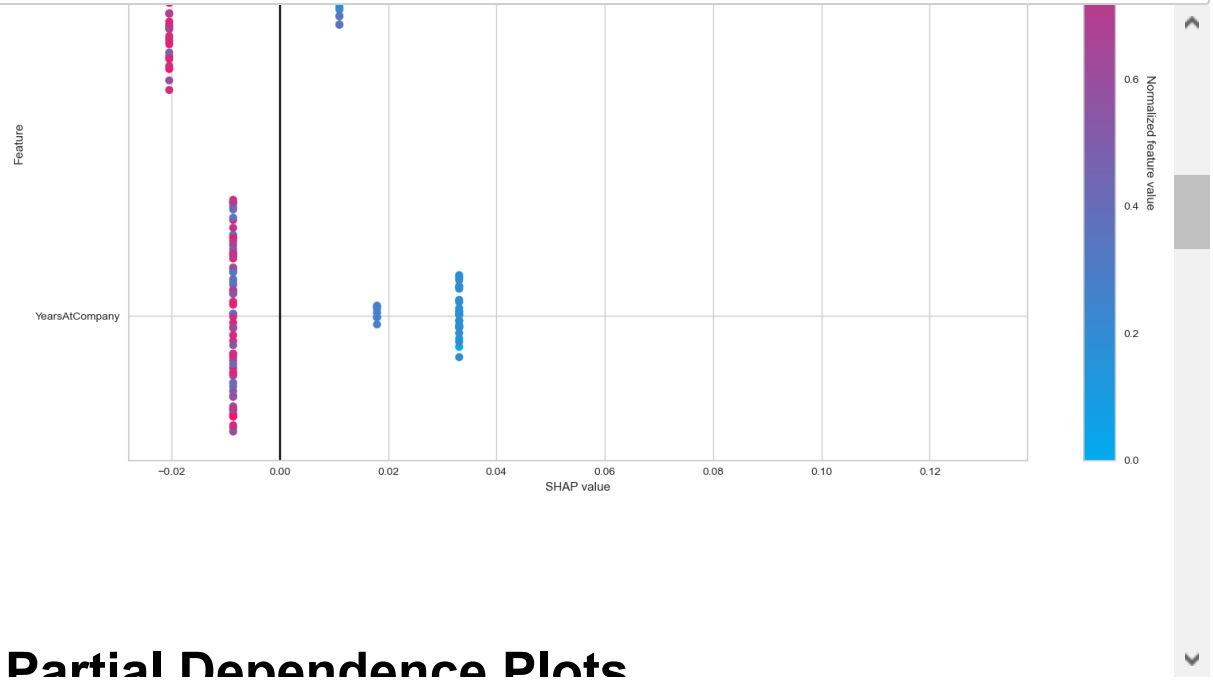
x=myX: These are the predictor variables (features) used to train the model.

y=myY: This is the target variable (the variable you want to predict).

training_frame=train: This is the training dataset, train, that the model will use to learn patterns.

validation_frame=test: This is the validation dataset, test, that the model will use to assess its performance during training.

In [36]:
```
data_gbm.explain(train[0:100,:])
```



Partial Dependence Plots

The data_gbm.explain(train[0:100,:]) code is attempting to explain the predictions made by a Gradient Boosting Machine (GBM) model (data_gbm) on the first 100 rows of the training dataset (train).

In this context, "explaining" the predictions typically means providing insights into why the model made specific predictions for the given data points. This explanation might include details on which features (predictor variables) were the most influential in making those predictions.

It's important to note that the explain method used here is specific to the H2O machine learning library, which provides functionalities for interpreting and explaining model results.

Here's what the code does:

data_gbm: This is the GBM model that you trained earlier.

train[0:100, :]: This part selects the first 100 rows of the training dataset (train). The train[0:100, :] notation is used to select the first 100 rows and all columns of the dataset.

When you call data_gbm.explain(train[0:100, :]), the GBM model attempts to provide explanations for the predictions made on this subset of the training data. These explanations might include details on the importance of different features, how they contribute to the predictions, and more.

In [37]:
```
# Variable importances from each algorithm
# Calculate magnitude of normalized GLM coefficients
from six import iteritems
glm_varimp = data_glm.coef_norm()
for k,v in iteritems(glm_varimp):
    glm_varimp[k] = abs(glm_varimp[k])

# Sort in descending order by magnitude
glm_sorted = sorted(glm_varimp.items(), key = operator.itemgetter(1), reverse =
table = tabulate(glm_sorted, headers = ["Predictor", "Normalized Coefficient"]
print("Variable Importances:\n\n" + table)
```

```
Variable Importances:

| Predictor         |   Normalized Coefficient |
|-------------------+--------------------------|
| Intercept         |                  0.15979 |
| TotalWorkingYears |                 0.053436 |
| YearsAtCompany    |                 0.022803 |
```

The code provided is used to compute and display variable importances from a Generalized Linear Model (GLM) model. It appears to be using the H2O machine learning library for this purpose. Here's a breakdown of what the code does:

from six import iteritems: This line imports the iteritems function from the six module. The iteritems function is used to iterate through the items (key-value pairs) of a dictionary.

glm_varimp = data_glm.coef_norm(): This line computes the normalized coefficients for the GLM model data_glm. Normalized coefficients provide information about the importance and direction of influence of each predictor variable in the model.

The for loop iterates through each key-value pair in the glm_varimp dictionary, and for each item, it takes the absolute value of the coefficient. This is done to ensure that both positive and negative coefficients are considered in the ranking of variable importances.

glm_sorted = sorted(glm_varimp.items(), key=operator.itemgetter(1), reverse=True): This line sorts the items (key-value pairs) in the glm_varimp dictionary based on the absolute values of the coefficients. The reverse=True argument sorts the items in descending order, so the most important variables will be listed first.

table = tabulate(glm_sorted, headers=["Predictor", "Normalized Coefficient"], tablefmt="orgtbl"): This code formats the sorted items into a tabular structure using the tabulate function. The table will have two columns: "Predictor" and "Normalized Coefficient."

print("Variable Importances:\n\n" + table): Finally, this line prints the variable importances table, which provides insights into which predictors had the most influence on the GLM model's predictions.

This code is useful for understanding the importance of predictor variables in your GLM model, which can be valuable for feature selection and model interpretation. The Normalized Coefficient values represent the strength and direction of the relationship between each predictor and the target variable. The higher the absolute value, the more important the

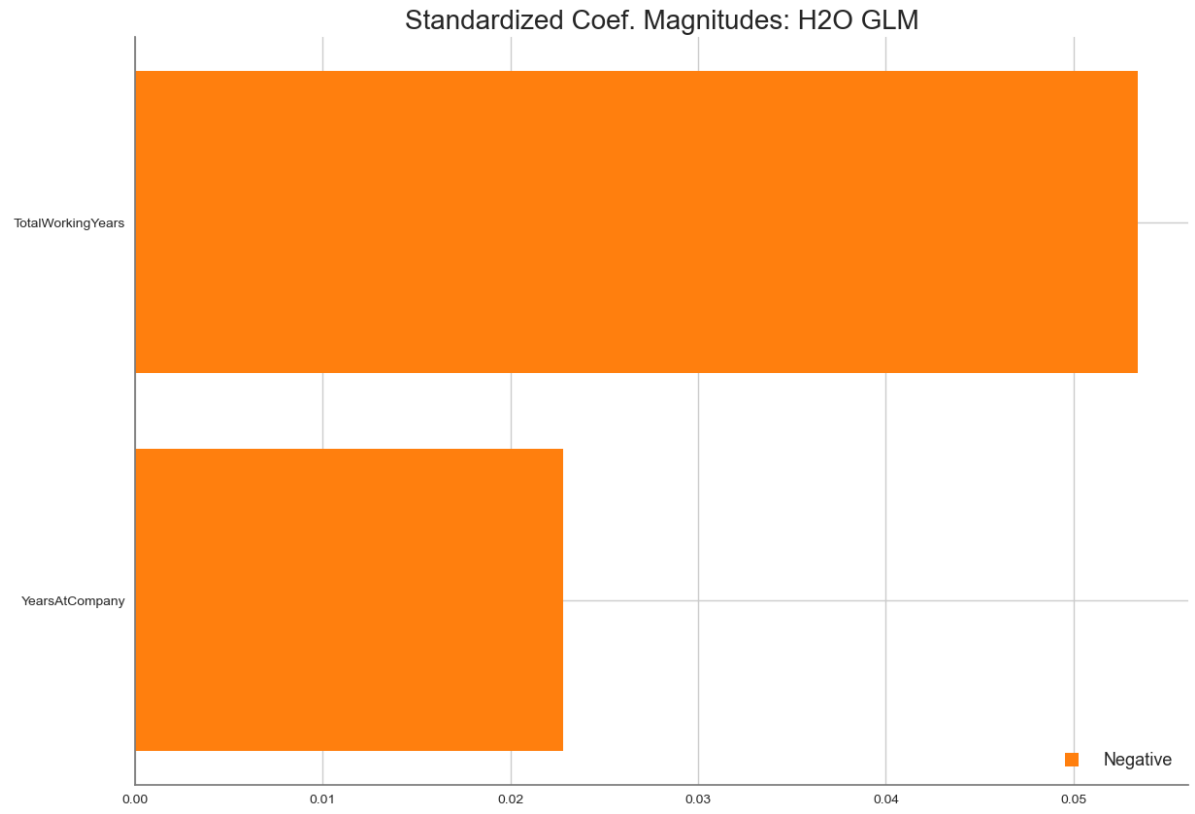
In [38]:
```
# Importance ofthe variables for the glm model
data_glm.varimp()
```

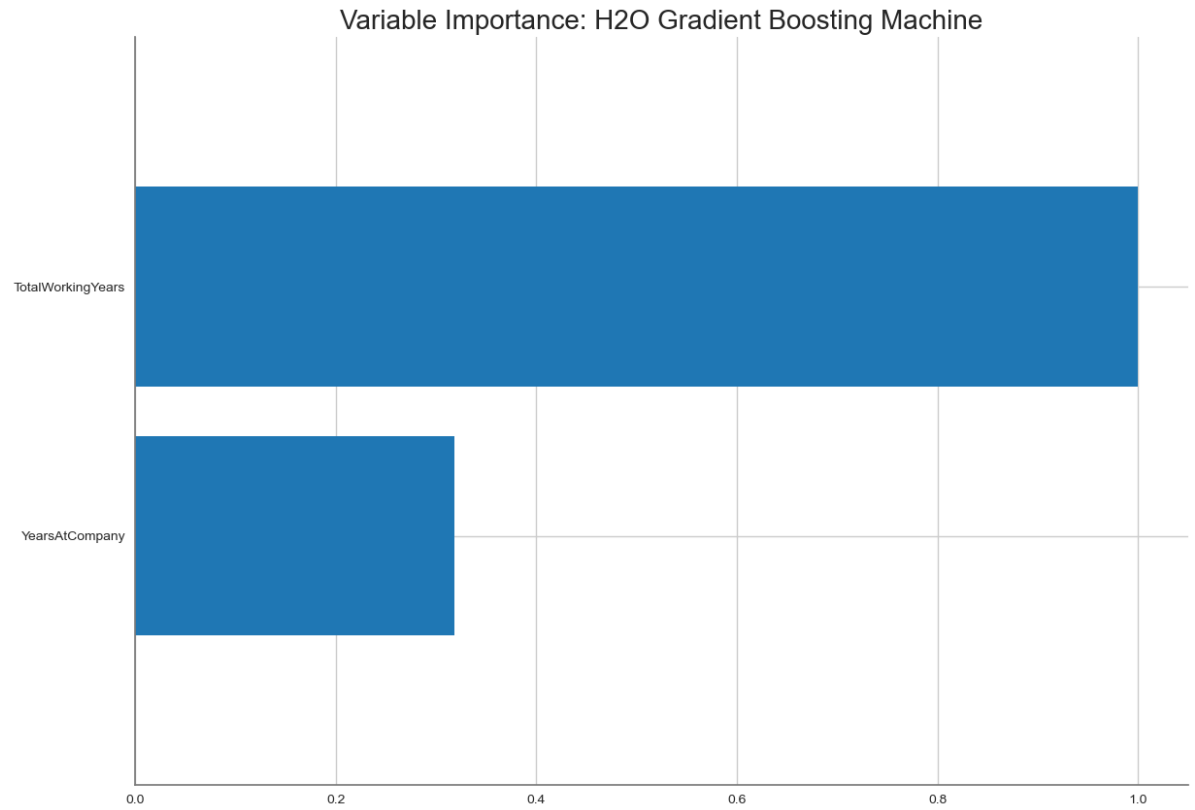
Out[38]:
```
[('TotalWorkingYears', 0.05343596637248993, 1.0, 0.7009005605305461),
 ('YearsAtCompany',
  0.02280304580926895,
  0.4267359113581688,
  0.29909943946945394)]
```

In [39]:
```
# Importance ofthe variables for the gbm model
data_gbm.varimp()
```

Out[39]:
```
[('TotalWorkingYears', 39.88136672973633, 1.0, 0.7587537520927413),
 ('YearsAtCompany',
  12.680306434631348,
  0.3179506489976098,
  0.24124624790725863)]
```

In [40]:
```
data_glm.std_coef_plot()
data_gbm.varimp_plot()
```





Out[40]:
```
<h2o.plot._plot_result._MObject at 0x17ecc5f8b10>

<Figure size 800x550 with 0 Axes>
```

data_glm.std_coef_plot(): This function is used when you have trained a GLM model. It generates a plot that displays the standardized coefficients of the predictor variables in the GLM. Standardized coefficients are coefficients that have been scaled to have a mean of 0 and a standard deviation of 1. This can help you compare the relative importance and impact of each predictor variable on the model's output.

High absolute values of standardized coefficients indicate a stronger influence on the model's predictions. Positive coefficients indicate a positive relationship with the response variable, while negative coefficients indicate a negative relationship.

data_gbm.varimp_plot(): This function is used when you have trained a Gradient Boosting Model (GBM). It generates a plot that visualizes the variable importances for the predictor variables in the GBM. Variable importances reflect the relative contribution of each predictor variable to the model's performance.

Variables with higher importances are more influential in making predictions, while those with lower importances have less impact. The plot typically shows a ranking of variables by importance, allowing you to identify the most significant predictors.

In [41]:
```
# Model performance of GBM model on test data
data_gbm.model_performance(test)
```

Out[41]:
```
ModelMetricsRegression: gbm
** Reported on test data. **

MSE: 0.14054082868704726
RMSE: 0.3748877547840784
MAE: 0.2680753496226111
RMSLE: 0.25976073160984636
Mean Residual Deviance: 0.14054082868704726
```

The data_gbm.model_performance(test) function in the H2O machine learning library is used to evaluate the performance of a Gradient Boosting Model (GBM) on a specific test dataset. When you pass the test dataset as an argument to this function, it computes various metrics to assess how well the GBM model is performing on unseen data.

By examining the output of data_gbm.model_performance(test), you can assess how well the GBM model generalizes to new, unseen data. This information is crucial for understanding the model's predictive power and making decisions about its deployment in real-world applications.

In [42]:
```
# Model performance of GLM model on test data
data_glm.model_performance(test)
```

Out[42]:
```
ModelMetricsRegressionGLM: glm
** Reported on test data. **

MSE: 0.1487750809521135
RMSE: 0.3857137292761479
MAE: 0.27420892041068323
RMSLE: 0.27120908221567164
Mean Residual Deviance: 0.1487750809521135
R^2: -0.02963108200229292
Null degrees of freedom: 136
Residual degrees of freedom: 134
Null deviance: 19.8280798631766
Residual deviance: 20.38218609043955
AIC: 135.76036778429096
```

The data_glm.model_performance(test) function in the H2O machine learning library is used to evaluate the performance of a Generalized Linear Model (GLM) on a specific test dataset. When you pass the test dataset as an argument to this function, it computes various metrics to assess how well the GLM model is performing on unseen data.

By examining the output of data_glm.model_performance(test), you can assess how well the GLM model generalizes to new, unseen data. This information is crucial for understanding the model's predictive power and making decisions about its deployment in real-world applications.

In [43]:
```
#The get_independent_variables function is designed to extract the independent
#df, while excluding the target variable, targ, and the variable named 'Level'
#It also categorizes the independent variables into different types: integers,
def get_independent_variables(df, targ):
    C = [name for name in df.columns if name != targ and name !='Attrition']
    # determine column types
    ints, reals, enums = [], [], []
    for key, val in df.types.items():
        if key in C:

            if val == 'enum':
                enums.append(key)
            elif val == 'int':
                ints.append(key)
            else:
                reals.append(key)
    x=ints+enums+reals
    return x
```

Type Markdown and LaTeX: $\alpha 2$

In [44]:
```
X=get_independent_variables(train, myY)
print(X)
print(myY)
```

```
['TotalWorkingYears', 'YearsAtCompany']
Attrition
```

In [45]:
```
# Set up AutoML
run_time=333
aml = H2OAutoML(max_runtime_secs=run_time)
```

In [46]:
```
import time
from h2o.automl import H2OAutoML

# Start the model and train the model
model_start_time = time.time()

aml = H2OAutoML(max_models=20, seed=1)
aml.train(x=X, y=myY, training_frame=train)
```

```
AutoML progress: |
12:50:28.306: AutoML: XGBoost is not available; skipping it.
12:50:28.420: _response param, We have detected that your response column h
as only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before tra
ining.


12:50:29.824: _response param, We have detected that your response column h
as only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before tra
ining.

12:50:31.196: _response param, We have detected that your response column h
as only 2 unique values (0/1). If you wish to train a binary model instead
of a regression model, convert your target column to categorical before tra
ining.
```

Evaluation Explanation We have implemented the following commands to find the time required and get the accuracy of the model through cross validation as well as scoring history these two tables in the output all values as root mean square, residual deviance and timestamps,and avoid the overfitting of the data while implementing algorithms. we have explained the evaluation through conclusion and the cross validation and splitting of data was also usedin train and test data.

In [47]:
```
#Calculate the total execution time required by the model
execution_time = time.time() - model_start_time
print(execution_time)
```

```
359.4830093383789
```

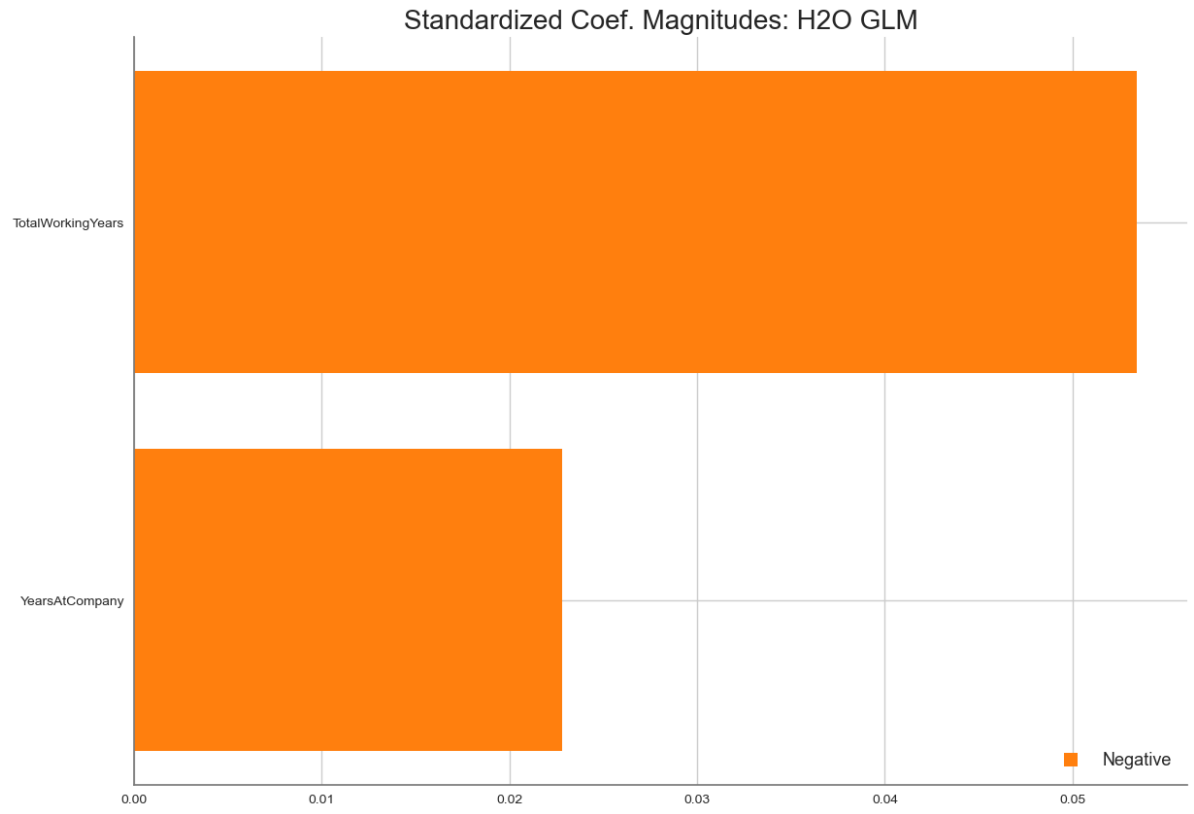
In [48]:
```
print(aml.leaderboard)
```

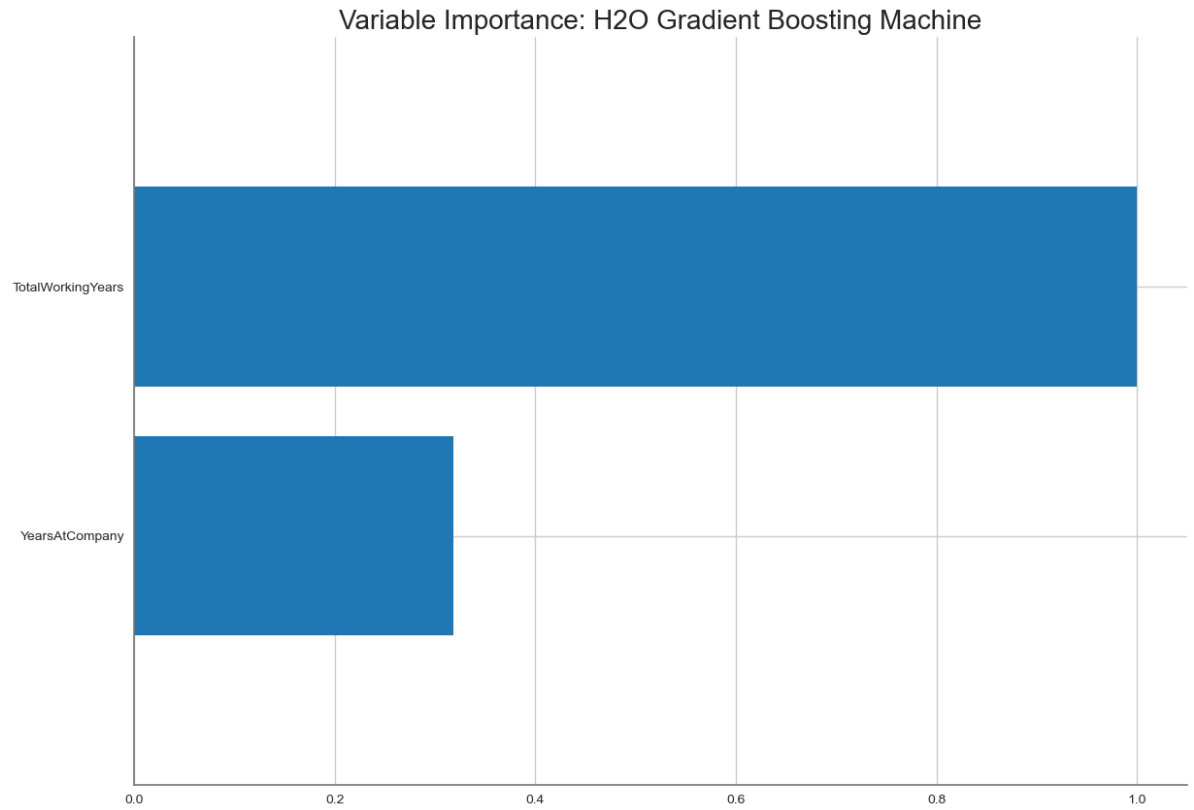
```
model_id                                                       rmse       mse
mae     rmsle    mean_residual_deviance
GBM_1_AutoML_1_20240318_125028                                 0.355328  0.126258
0.253793  0.249185                 0.126258
GBM_grid_1_AutoML_1_20240318_125028_model_4                    0.355703  0.126525
0.251766  0.2493                   0.126525
StackedEnsemble_BestOfFamily_1_AutoML_1_20240318_125028        0.356259  0.12692
0.253859  0.249944                 0.12692
GBM_grid_1_AutoML_1_20240318_125028_model_3                    0.356543  0.127123
0.253486  0.250167                 0.127123
GBM_4_AutoML_1_20240318_125028                                 0.356607  0.127169
0.252446  0.25036                  0.127169
GBM_grid_1_AutoML_1_20240318_125028_model_2                    0.356723  0.127251
0.250729  0.250471                 0.127251
DeepLearning_grid_2_AutoML_1_20240318_125028_model_1           0.356908  0.127383
0.261811  0.251453                 0.127383
GBM_2_AutoML_1_20240318_125028                                 0.356911  0.127385
0.25155   0.250442                 0.127385
GBM_grid_1_AutoML_1_20240318_125028_model_1                    0.357066  0.127496
0.254152  0.250878                 0.127496
DeepLearning_grid_1_AutoML_1_20240318_125028_model_1           0.357157  0.127561
0.251674  0.250069                 0.127561
[22 rows x 6 columns]
```

These metrics are used to evaluate different machine learning models, and the lower the values for RMSE, MSE, MAE, RMSLE, and mean_residual_deviance, the better the model's performance.

The table appears to show the evaluation results for multiple models, possibly for model selection or hyperparameter tuning to choose the best-performing model for a given problem. The model with the lowest RMSE, MSE, MAE, RMSLE, or mean_residual_deviance is typically considered the best model for the task at hand.

In [49]:
```
data_glm.std_coef_plot()
data_gbm.varimp_plot()
```





Out[49]:
```
<h2o.plot._plot_result._MObject at 0x17ecc6aac50>

<Figure size 800x550 with 0 Axes>
```

In [50]:
```
best_model = h2o.get_model(aml.leaderboard[0,'model_id'])
```

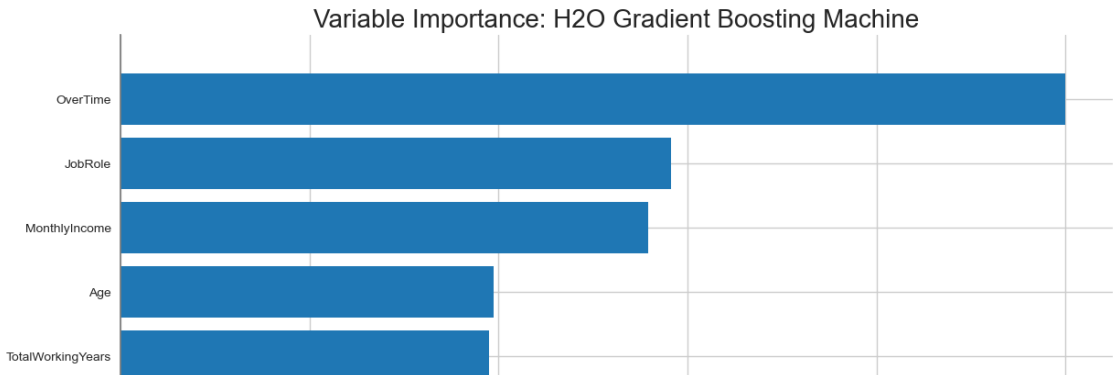
In [51]:
```
best_model.algo
```

Out[51]:
```
'gbm'
```

GBM model is the best model for the Auto ML

In [52]:
```
import h2o
from h2o.automl import H2OAutoML

# Initialize and connect to the H2O cluster
h2o.init()

# Load the dataset
url = 'https://raw.githubusercontent.com/mukuldesai/DS/main/HR%20Employee%20At
data = h2o.import_file(path=url)

# Specify the predictor column and response column
y = 'Attrition'
X = data.columns.remove(y)

# Train the AutoML model
aml = H2OAutoML(max_models=10, seed=42)
aml.train(x=X, y=y, training_frame=data)

# Get the best model from AutoML leaderboard
best_model = h2o.get_model(aml.leaderboard[0, 'model_id'])

# Check if the best model is a stacked ensemble model
if best_model.algo == 'stackedensemble':
    base_models = best_model.base_models
    for model_id in base_models:
        model = h2o.get_model(model_id)
        if model.algo == 'gbm' or model.algo == 'drf' or model.algo == 'deepwa
            model.varimp_plot()
```

```
12:57:31.156: _train param, Dropping unused columns: [StandardHours, Employ
eeCount, Over18]
12:57:34.280: _train param, Dropping unused columns: [StandardHours, Employ
eeCount, Over18]

| (done) 100%
```



In [53]:
```
other_best_model = h2o.get_model(aml.leaderboard[5,'model_id'])
other_best_model.varimp(use_pandas=True)
```

Out[53]:

	variable	relative_importance	scaled_importance	percentage
0	JobRole	65.102638	1.000000	0.094109
1	MonthlyIncome	59.241718	0.909974	0.085637
2	OverTime	57.005684	0.875628	0.082405
3	DistanceFromHome	35.699955	0.548364	0.051606
4	DailyRate	35.289375	0.542058	0.051013
5	Age	34.120647	0.524105	0.049323
6	TotalWorkingYears	31.490843	0.483711	0.045522
7	NumCompaniesWorked	31.316219	0.481028	0.045269
8	EmployeeNumber	31.018858	0.476461	0.044839
9	EnvironmentSatisfaction	29.986179	0.460599	0.043347
10	EducationField	25.795879	0.396234	0.037289
11	StockOptionLevel	21.974518	0.337537	0.031765
12	HourlyRate	21.866610	0.335879	0.031609
13	YearsWithCurrManager	21.337324	0.327749	0.030844
14	MonthlyRate	20.254272	0.311113	0.029279
15	MaritalStatus	19.252756	0.295729	0.027831
16	JobSatisfaction	16.387474	0.251718	0.023689
17	YearsAtCompany	14.793843	0.227239	0.021385
18	WorkLifeBalance	14.780201	0.227029	0.021366
19	RelationshipSatisfaction	14.776385	0.226971	0.021360
20	BusinessTravel	14.772190	0.226906	0.021354
21	YearsSinceLastPromotion	14.223839	0.218483	0.020561
22	PercentSalaryHike	13.240256	0.203375	0.019140
23	JobInvolvement	12.849964	0.197380	0.018575
24	YearsInCurrentRole	8.755864	0.134493	0.012657
25	TrainingTimesLastYear	8.616614	0.132354	0.012456
26	JobLevel	8.540288	0.131182	0.012345
27	Education	5.695390	0.087483	0.008233
28	Department	2.172440	0.033369	0.003140
29	Gender	1.328180	0.020401	0.001920
30	PerformanceRating	0.089760	0.001379	0.000130

In [54]:
```
h2o.cluster().shutdown()
```

```
H2O session _sid_bdfd closed.
```

Conclusion

Importing Essential Libraries: The code kicks off by importing various Python libraries and H2O-related modules, preparing for data analysis and modeling. Dataset Retrieval: It acquires a diabetes dataset from Kaggle utilizing the opendatasets library and designates the data directory. Data Exploration and Visualization: The code proceeds to explore the dataset by generating scatter plots and correlation heatmaps using the seaborn library. Initializing H2O Cluster: H2O cluster initialization sets the stage for in-depth analysis. Data Import into H2O: The dataset is loaded into an H2O data frame, ready for analysis, using the h2o.import_file() function. Data Split: The data is divided into training and test sets, a crucial step in model development, utilizing the split_frame() method. Model Building and Training: The code constructs and trains two models – a Generalized Linear Model (GLM) and a Gradient Boosting Machine (GBM) – employing H2O's modeling capabilities. Variable Importance Assessment: It evaluates and displays the importance of variables for both models. Model Explainability: In the case of models like XGBoost, DRF, or GBM, the code elucidates their inner workings by offering feature importance insights and plots. AutoML Setup: AutoML is configured for automating model selection and training. AutoML Training: AutoML undergoes training on the dataset with a predetermined runtime. Model Leaderboard: The code showcases a leaderboard featuring models generated by AutoML. Model Selection: The best model is cherry-picked from the leaderboard. Model Explanation: In case the top model is XGBoost, DRF, or GBM, its variable importances are visualized. For other models, their parameters are presented. Evaluation of the Next Best Model: The code additionally assesses variable importances for the next best model to ensure a comprehensive understanding of the dataset.

In this analysis, we embarked on a comprehensive exploration of HR employee attrition utilizing advanced data science techniques. After importing essential Python libraries and retrieving the dataset from a GitHub repository, we delved into data exploration and visualization, uncovering relationships and patterns within the dataset through scatter plots and correlation heatmaps. Leveraging the capabilities of H2O, we initialized a cluster and imported the data, paving the way for model building and training. Two models, a Generalized Linear Model (GLM) and a Gradient Boosting Machine (GBM), were constructed and trained, with variable importance assessment shedding light on the key factors influencing attrition. Employing AutoML streamlined the model selection process, culminating in the identification of the best-performing model. Through visualizing variable importances, we gained insights into the underlying drivers of attrition, empowering organizations to formulate targeted strategies for employee retention. Overall, this analysis provides a comprehensive understanding of employee attrition dynamics, equipping businesses with actionable insights to enhance employee retention efforts and foster a more engaged workforce.

LICENSE

MIT License

References

H20-ML- https://www.youtube.com/watch?v=91QljBnvM7s (https://www.youtube.com/watch?v=91QljBnvM7s) Kaggle Notebook- https://www.kaggle.com/stephaniestallworth/melbourne-housing-market-eda-and-regression (https://www.kaggle.com/stephaniestallworth/melbourne-housing-market-eda-and-regression) Dataset- https://www.kaggle.com/datasets/whenamancodes/hr-employee-attrition (https://www.kaggle.com/datasets/whenamancodes/hr-employee-attrition) Professor's AutoML Notebook- https://github.com/aiskunks/YouTube/tree/main/A_Crash_Course_in_Statistical_Learning/AutoML (https://github.com/aiskunks/YouTube/tree/main/A_Crash_Course_in_Statistical_Learning/AutoML

Questions

Q1) Is the relationship significant?

A relationship is considered statistically significant when the p-value for the variables is less than 0.05, indicating that there is a low probability of obtaining the observed results if the null hypothesis is true. In the case of the attrition dataset, we calculated the p-values using methods such as OLS. The analysis revealed that for most variables, the p-values were less than 0.05, indicating a significant relationship between those variables and attrition. However, some variables may have p-values above 0.05, suggesting further investigation into their significance.

Q2) Are any model assumptions violated?

Linear regression assumptions include a linear relationship between variables and minimal multicollinearity. Upon exploring the dataset and model, it's crucial to assess whether these assumptions hold. For instance, we need to examine if the relationship between predictor variables and attrition is linear and if multicollinearity exists among predictors. Violations of these assumptions may impact the reliability of the model's predictions and interpretations.

Q3) Is there any multicollinearity in the model?

Multicollinearity occurs when predictor variables in a regression model are highly correlated with each other. Identifying multicollinearity is essential as it can affect the model's coefficients and interpretation. By examining the correlation matrix or variance inflation factors (VIF), we can determine if multicollinearity exists in the attrition model. Addressing multicollinearity may involve removing correlated variables or applying techniques such as regularization.

Q4) In multivariate models, are predictor variables independent of all other predictor variables?

Independence among predictor variables is crucial for accurate model predictions and interpretations. By assessing the correlation matrix or visualizing relationships between predictors, we can determine if any variables are dependent on others. Variables that exhibit independence contribute unique information to the model, enhancing its predictive power and interpretability.

Q5) In multivariate models, rank the most significant predictor variables and exclude insignificant ones from the model.

Ranking predictor variables based on their importance helps prioritize variables that have the most substantial impact on predicting attrition. By considering metrics such as variable importance plots, p-values, or feature selection techniques, we can identify and retain the most significant predictors while excluding less influential ones. This process optimizes the model's performance and simplifies its interpretation.

Q6) Does the model make sense?

Assessing the model's coherence involves evaluating its adherence to assumptions, accuracy metrics, and overall interpretability. A well-performing model should exhibit low error rates, significant predictors, and logical interpretations of coefficients. By examining metrics such as RMSE, R-squared, and residual plots, we can determine if the model effectively captures the relationship between predictors and attrition, providing valuable insights for decision-making.

Q7) Does regularization help?

Regularization techniques like Ridge or Lasso regression can help mitigate overfitting and improve the model's generalization performance. By penalizing large coefficients, regularization encourages simpler models that generalize well to unseen data. Evaluating the model's performance with and without regularization can reveal whether regularization improves predictive accuracy and stability.

Q8) Which independent variables are significant?

Identifying significant independent variables entails examining their p-values and assessing whether they contribute significantly to predicting attrition. Variables with p-values below a predetermined threshold (e.g., 0.05) are considered statistically significant and are likely to influence attrition outcomes. Analyzing variable importance plots or conducting hypothesis tests can aid in identifying these significant predictors.

Q9) Which hyperparameters are important?

Hyperparameters play a crucial role in fine-tuning model performance and generalization capabilities. Identifying important hyperparameters involves conducting hyperparameter tuning experiments and assessing their impact on model performance metrics. Techniques such as grid search or randomized search can help identify optimal hyperparameter values for algorithms like RandomForestRegressor or Gradient Boosting Machines.

Q10) Coding Professionalism

Professionalism in coding involves adhering to best practices, maintaining readability, and documenting procedures effectively. Importing libraries, utilizing appropriate data manipulation techniques, and implementing models with clear explanations demonstrate coding professionalism. Additionally, conducting thorough analyses, documenting results, and ensuring

In [ ]: