Kyle Calabro

Dr. Tweneboah

CMPS 620

4 May 2021

Homework Six

---

1). **Data Preparation**

```
np.random.seed(42)
tf.random.set_random_seed(42)
```

```
hr = pd.read_csv("./HR_comma_sep.csv")
hr.rename(columns = {"sales" : "department"}, inplace = True)
hr.shape
```

```
hr.isnull().sum()

satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
sales                    0
salary                   0
dtype: int64
```

- From the above output we can see that the data set contains no missing data, therefore there is no need for any form of imputation
- Converting categorical data to ordinal values (performed after EDA was conducted):

```
scale_mapper = {"low" : 1, "medium" : 2, "high" : 3}
hr.salary = hr.salary.replace(scale_mapper)
```

```
scale_mapper = {"sales" : 1, "technical" : 2, "support" : 3, "IT" : 4, "product_mng" : 5,
                "marketing" : 6, "RandD" : 7, "accounting" : 8, "hr" : 9, "management" : 10}
hr.department = hr.department.replace(scale_mapper)
```

2). Exploratory Data Analysis (EDA)

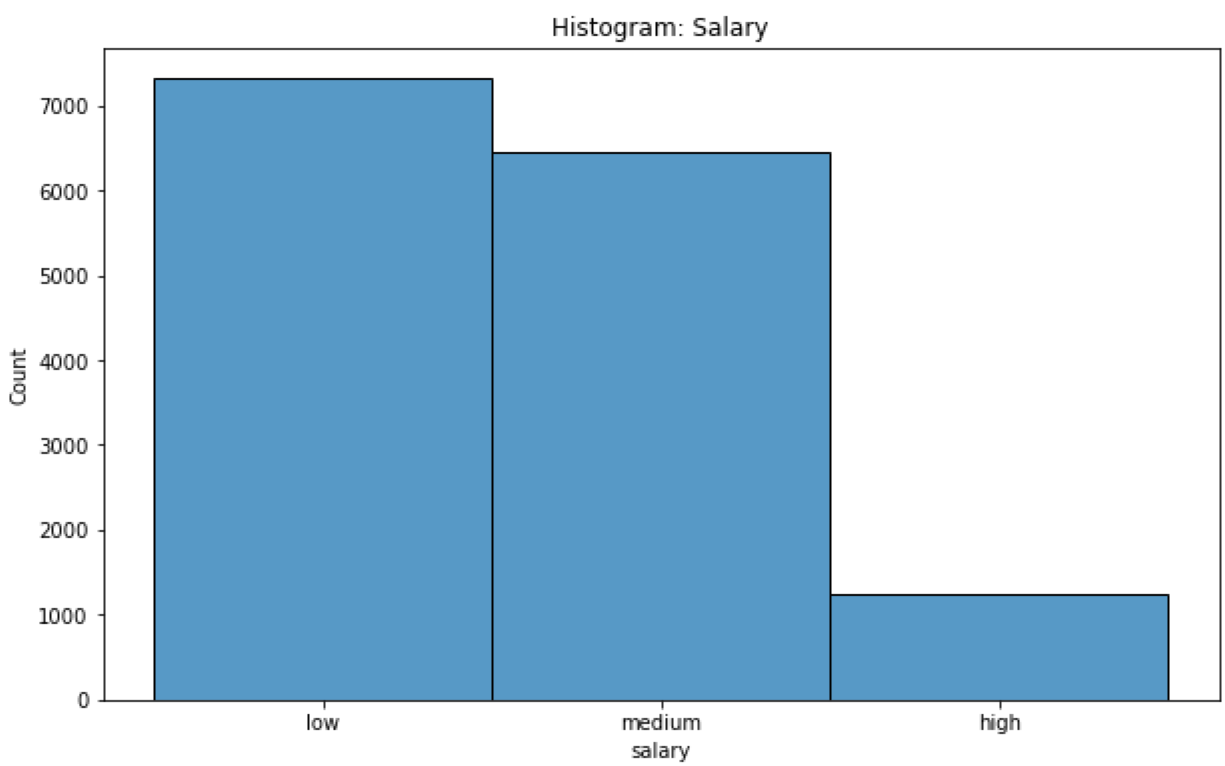
- EDA.1: Distribution of employees by department:

```
sales            4140
technical        2720
support          2229
IT               1227
product_mng       902
marketing         858
RandD             787
accounting        767
hr                739
management        630
Name: department, dtype: int64
```

- EDA.2: Distribution of salaries for employees:

```
low        7316
medium     6446
high       1237
Name: salary, dtype: int64
```

```
plt.figure(figsize = (10, 6))
sns.histplot(hr.salary).set_title("Histogram: Salary");
```



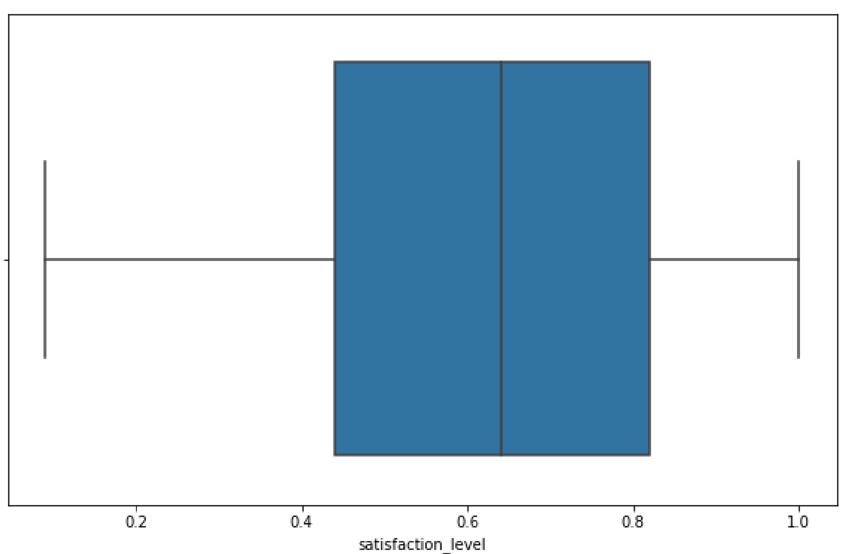
- EDA.3: Number of employees per salary range within each department:

```
pivot_table = hr.pivot_table(values = "satisfaction_level", index = "department", columns = "salary", aggfunc = np.count_nonzero)
pivot_table
```

salary department	high	low	medium
IT	83.0	609.0	535.0
RandD	51.0	364.0	372.0
accounting	74.0	358.0	335.0
hr	45.0	335.0	359.0
management	225.0	180.0	225.0
marketing	80.0	402.0	376.0
product_mng	68.0	451.0	383.0
sales	269.0	2099.0	1772.0
support	141.0	1146.0	942.0
technical	201.0	1372.0	1147.0

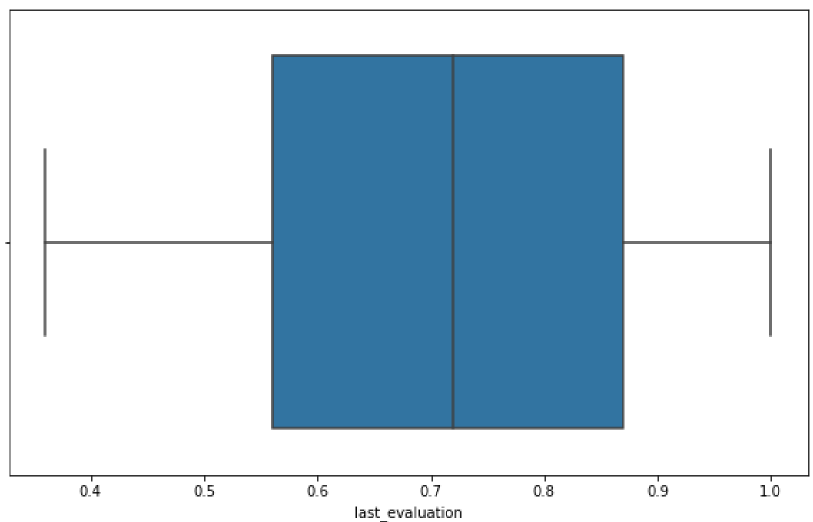
- Boxplots for features of interest:
    - EDA.4: Satisfaction Level:

```
plt.figure(figsize = (10, 6))
sns.boxplot(hr.satisfaction_level);
```



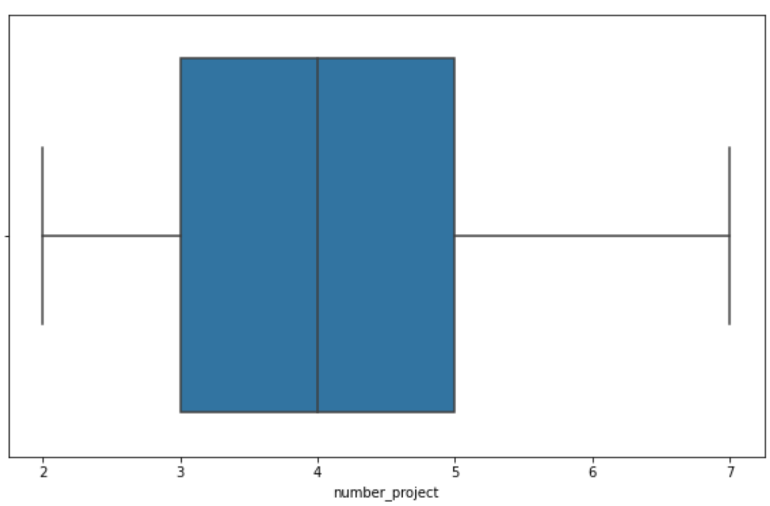
- EDA.5: Last Evaluation:

```
plt.figure(figsize = (10, 6))
sns.boxplot(hr.last_evaluation);
```



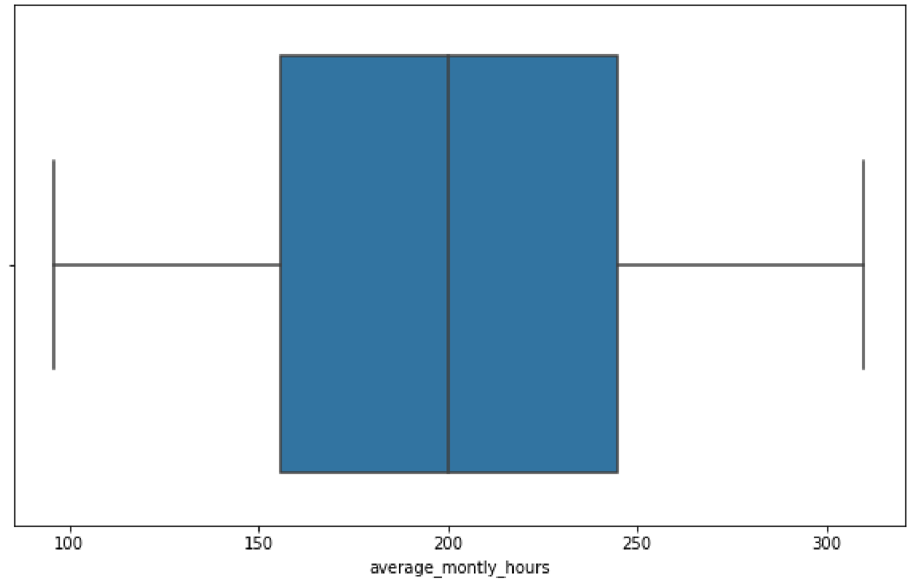
- EDA.6: Number of Projects:

```
plt.figure(figsize = (10, 6))
sns.boxplot(hr.number_project);
```



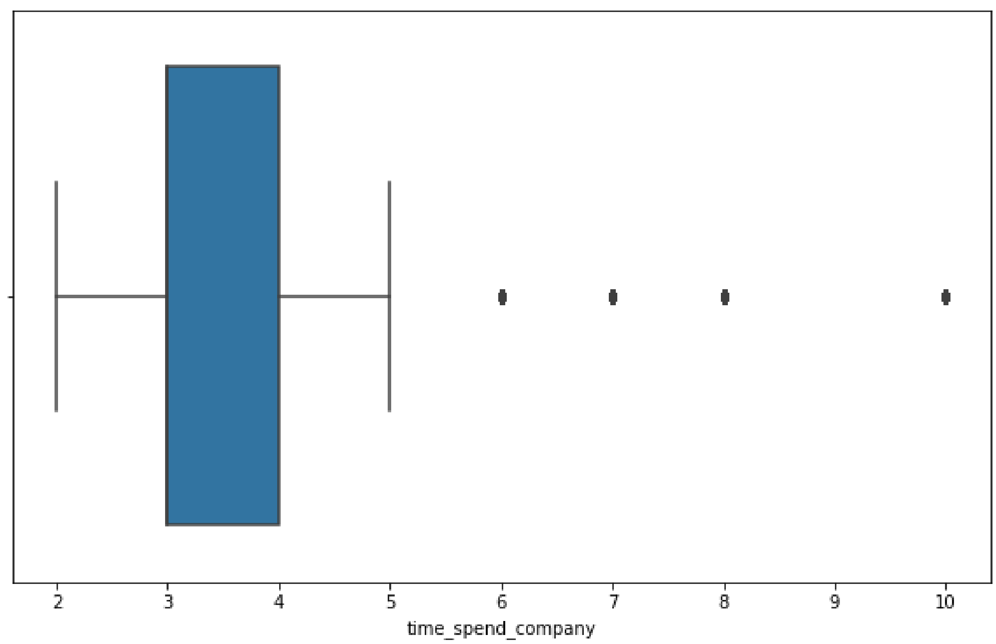
- EDA.7: Average Monthly Hours:

```
plt.figure(figsize = (10, 6))
sns.boxplot(hr.average_montly_hours);
```



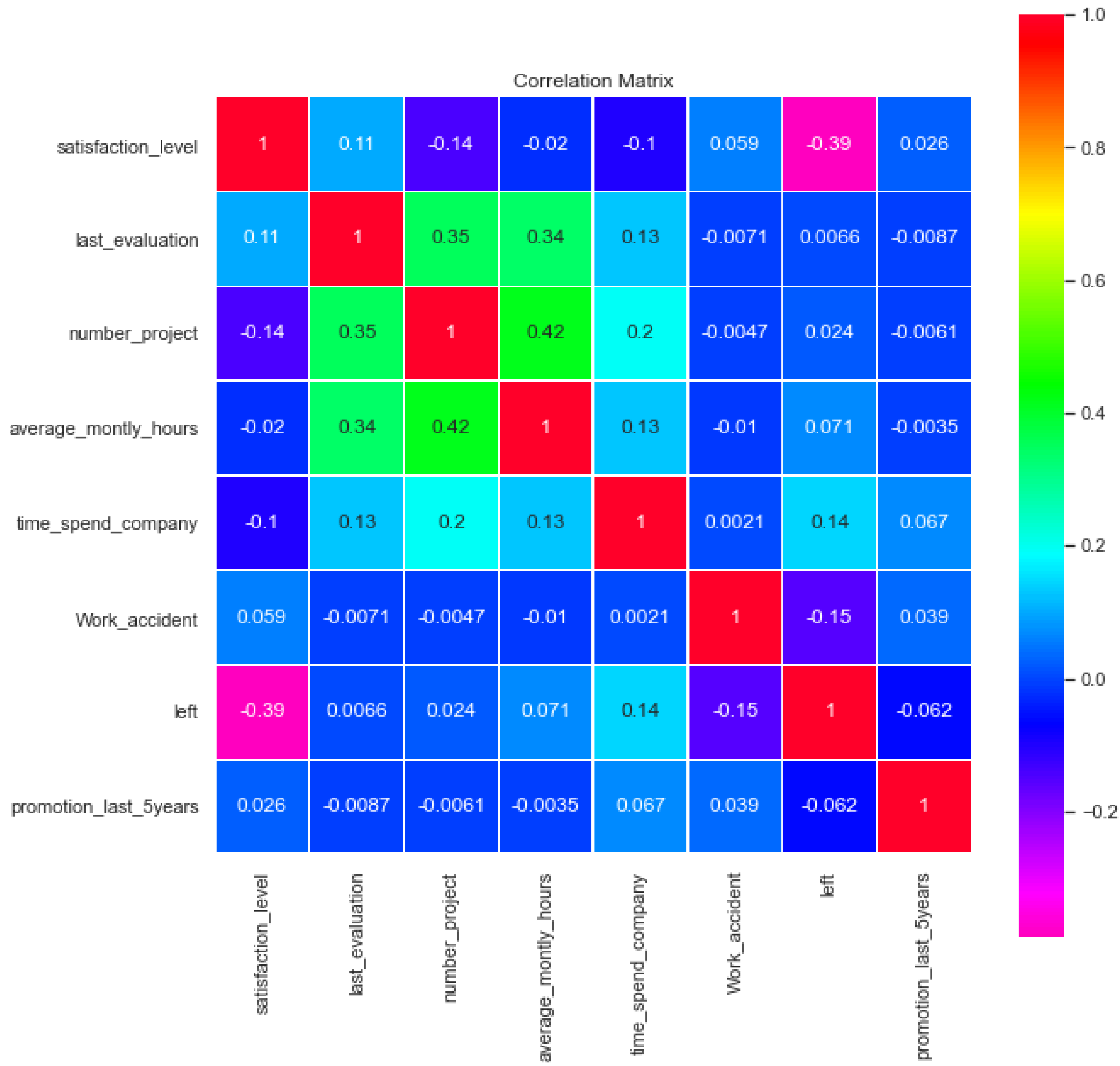
- EDA.8: Time Spent with Company:

```
plt.figure(figsize = (10, 6))
sns.boxplot(hr.time_spend_company);
```



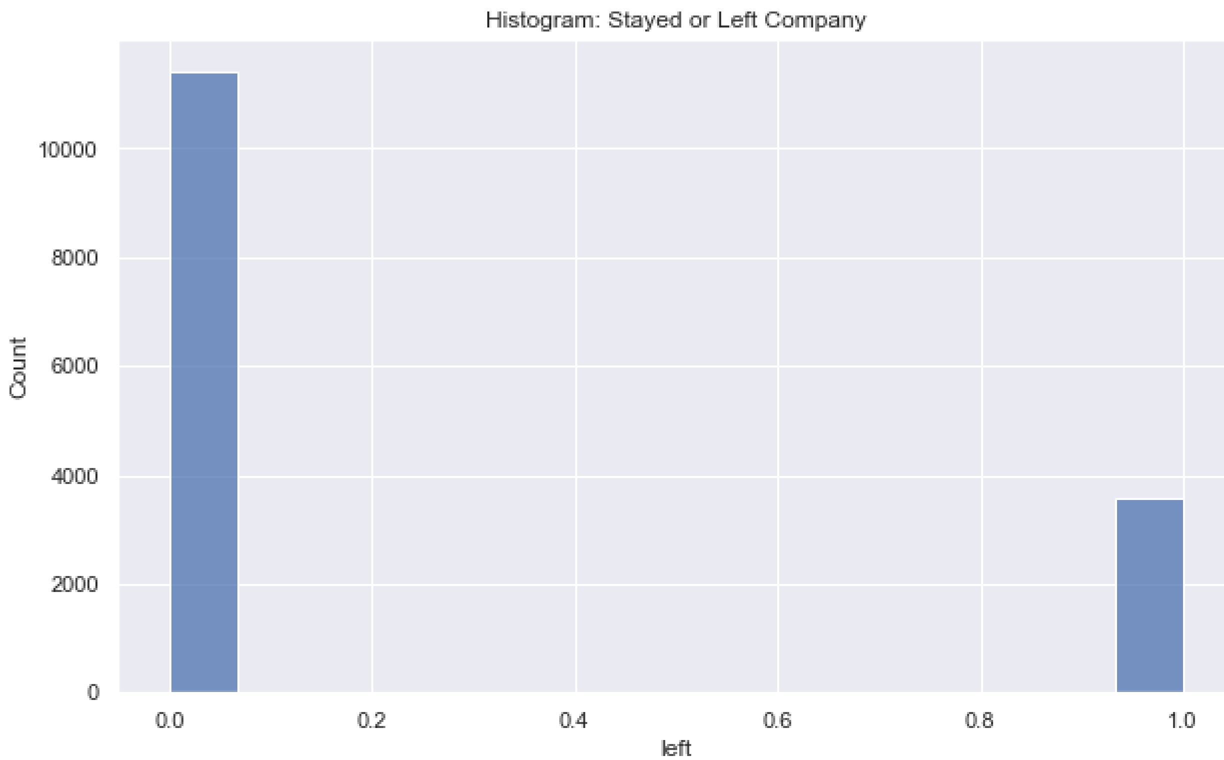
- EDA.9: Correlation Matrix:

```
plt.figure(figsize = (10, 10))
sns.set(font_scale = 1)
sns.heatmap(hr.corr(), cmap = "gist_rainbow_r", annot = True, square = True, linewidths = .5)
plt.title("Correlation Matrix");
```



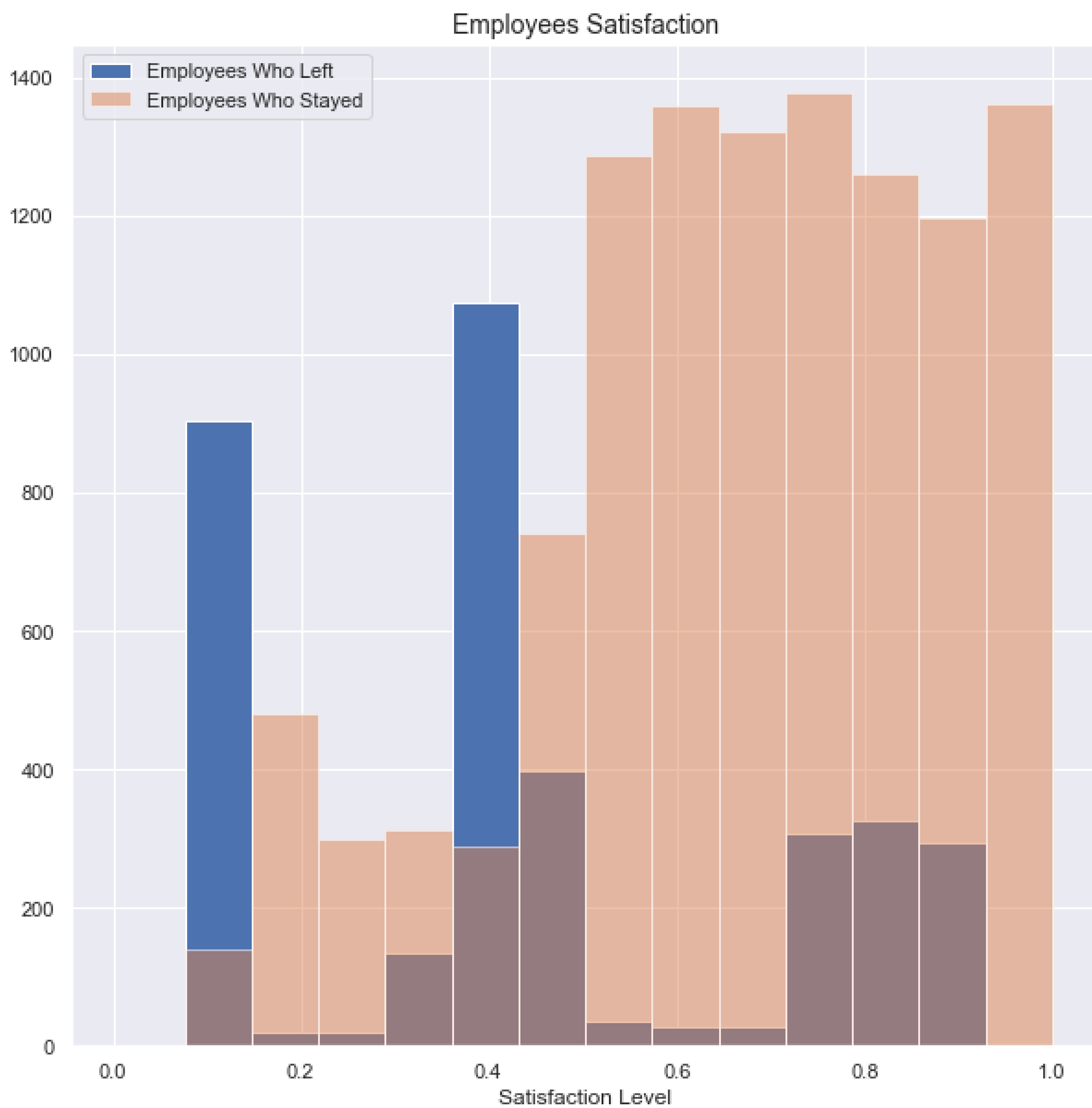
- EDA.10: How many people have left the company?

```
plt.figure(figsize = (10, 6))
sns.histplot(hr.left).set_title("Histogram: Stayed or Left Company");
```



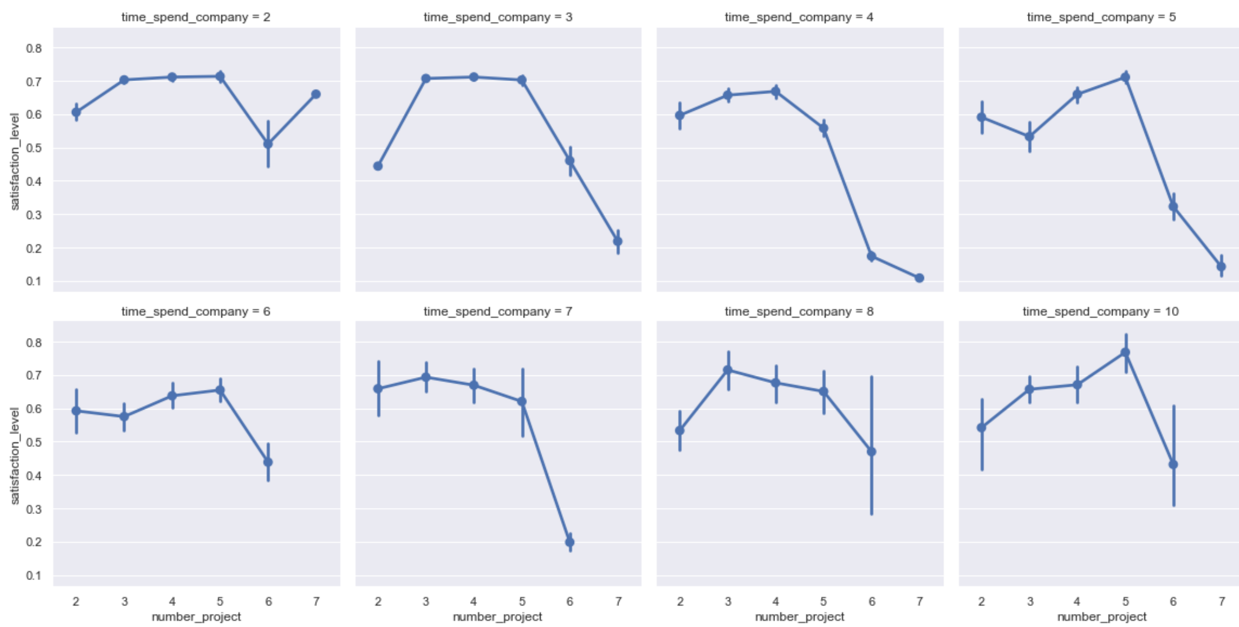
- EDA.11: What impact does satisfaction level have on employee retention?

```
plt.figure(figsize = (10, 10))

bins = np.linspace(0.006,1.000, 15)

plt.hist(hr[hr.left == 1]['satisfaction_level'], bins = bins, alpha = 1, label = 'Employees Who Left')
plt.hist(hr[hr.left == 0]['satisfaction_level'], bins = bins, alpha = 0.5, label = 'Employees Who Stayed')

plt.title('Employees Satisfaction', fontsize = 14)
plt.xlabel('Satisfaction Level')
plt.legend(loc = 'best');
```



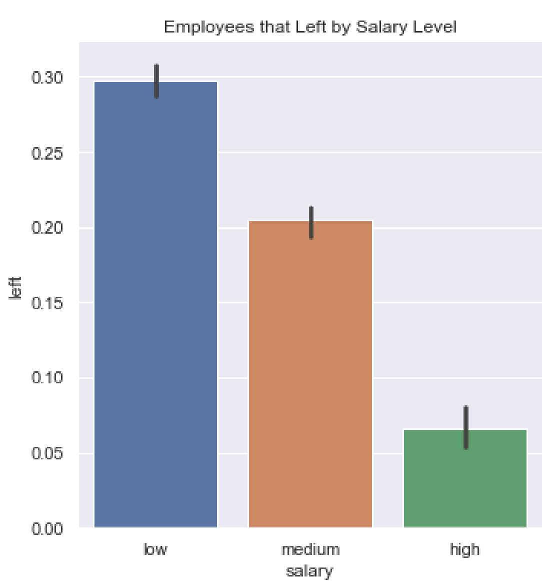
- EDA.12: How does the number of projects assigned to employees affect their satisfaction level(s)?

```
sns.factorplot(data = hr, x = "number_project", y = "satisfaction_level", col = "time_spend_company", col_wrap = 4, size = 4);
```



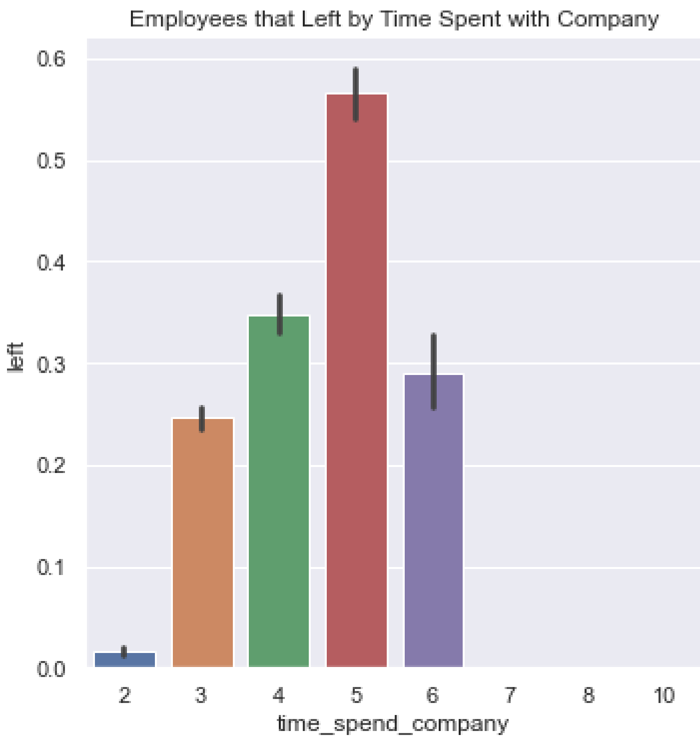
- EDA.13: How does salary affect those leaving the company?

```
f_plot = sns.factorplot(x = "salary", y = "left", kind = "bar", data = hr)
plt.title("Employees that Left by Salary Level");
```



- EDA.14: How does time spent with the company impact employee retention?

```
f_plot = sns.factorplot(x = "time_spend_company", y = "left", kind = "bar", data = hr)
plt.title("Employees that Left by Time Spent with Company");
```



- Conclusions From Exploratory Data Analysis:
    - From the correlation matrix (EDA.9) we can observe the following:
        - There exists a negative correlation between the satisfaction level of employees and those who have left the company
        - The highest positive correlation exists between the number of projects and average monthly hours for employees
    - From the overlaid histogram (EDA.11) we can observe that:
        - There exists a peak of employees who left that were incredibly disappointed with the company. There is also another peak around the satisfaction level of .4 that still left the company. There also exists another

group within the [.7 : .9] satisfaction level range that still decided to leave the company. This is indicative that retention is not solely dependent upon satisfaction level and that a combination of factors may be driving employees to leave

- From the factor plot (EDA.12) we can observe that:
  - There is a considerable drop in satisfaction level when employees are working on six or more projects regardless of how many years they have been with the company
  - Three or four projects seems to be the optimal number to preserve the satisfaction level of employees
- From the boxplot for time spent with company (EDA.8), we can observe that:
  - Most employees have been with the company for three or four years, there exist potential outliers who have spent six to ten years with the company however. This could be indicative of a relatively high turnover rate for employees or that it is a relatively young company
- From the boxplot for satisfaction level (EDA.6), we can observe that:
  - Most employees satisfaction level(s) lie within the [.4 : .8] range, this indicates that most employees are content with the company but perhaps not overly happy with it

3). Data Partitioning

- Splitting into testing and training sets:

```
X = hr.drop(columns = ["left"], axis = 1).values
y = hr.left.values
```

```
sc = StandardScaler()

train_ratio = 0.66

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1 - train_ratio, random_state = 42)

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
X_train: (9899, 9)
X_test: (5100, 9)
```

```
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)
y_train: (9899,)
y_test: (5100,)
```

4). Deep Neural Networks:

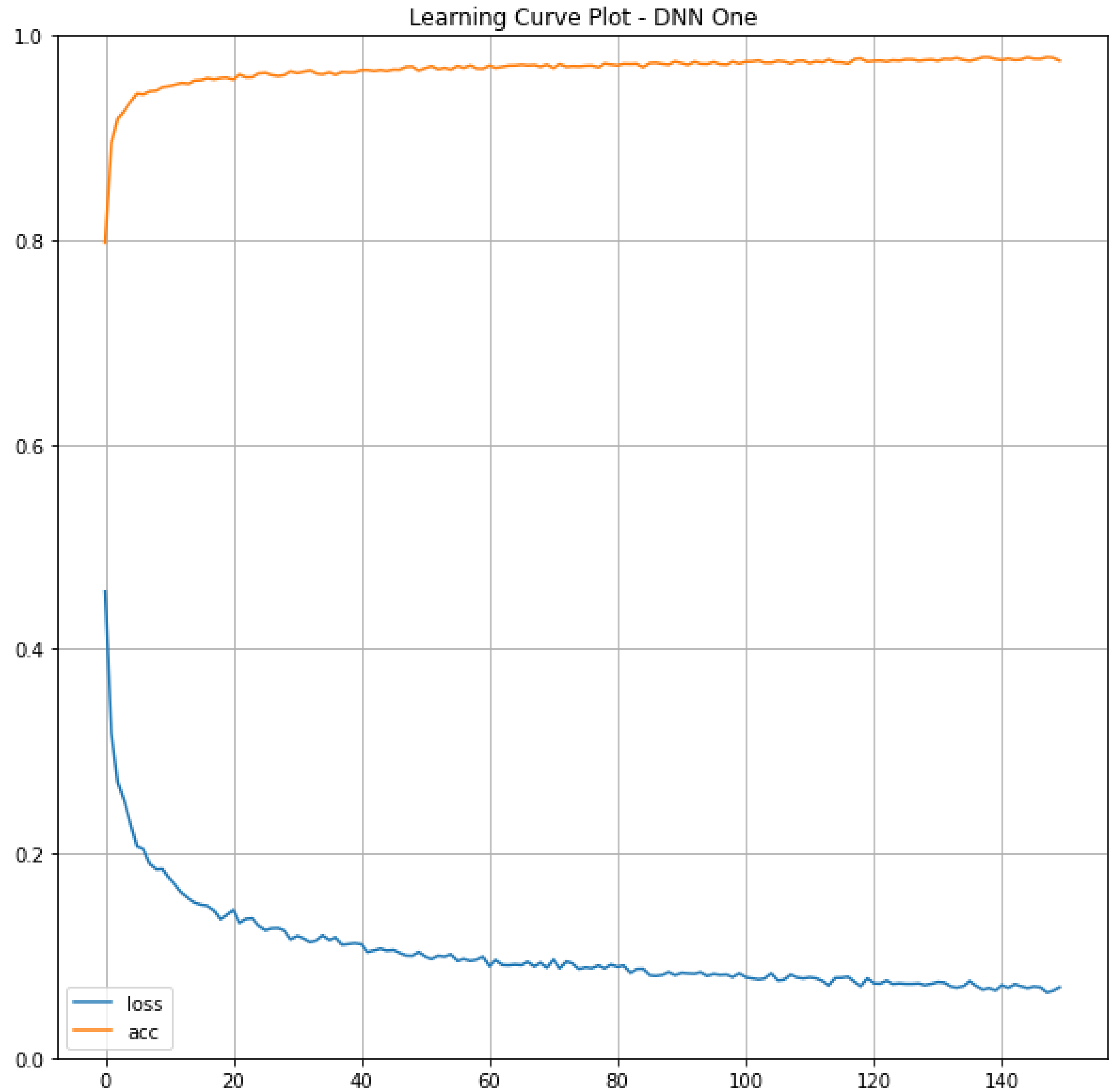
- Model One:

```
dnn_one = keras.models.Sequential([
    keras.layers.BatchNormalization(),
    keras.layers.Dense(600, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(450, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(150, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(1, activation = "sigmoid")
])
```

```
dnn_one.compile(loss = "binary_crossentropy",
                optimizer = keras.optimizers.SGD(lr = 1e-3),
                metrics = ["accuracy"])
```

```
start_time_dnn_one = time.time()

history_one = dnn_one.fit(X_train, y_train, epochs = 150)

end_time_dnn_one = time.time()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_5 (Batch multiple                  36
_________________________________________________________________
dense_12 (Dense)             multiple                  6000
_________________________________________________________________
batch_normalization_6 (Batch multiple                  2400
_________________________________________________________________
dense_13 (Dense)             multiple                  270450
_________________________________________________________________
batch_normalization_7 (Batch multiple                  1800
_________________________________________________________________
dense_14 (Dense)             multiple                  135300
_________________________________________________________________
batch_normalization_8 (Batch multiple                  1200
_________________________________________________________________
dense_15 (Dense)             multiple                  45150
_________________________________________________________________
batch_normalization_9 (Batch multiple                  600
_________________________________________________________________
dense_16 (Dense)             multiple                  151
=================================================================
Total params: 463,087
Trainable params: 460,069
Non-trainable params: 3,018
```




```
print("Elapsed Training Time: {} seconds".format(end_time_dnn_one - start_time_dnn_one), "\n")
```

- Runtime on local machine:

```
Elapsed Training Time: 168.24746108055115 seconds
```

- Runtime on AWS:

```
Elapsed Training Time: 377.35017919540405 seconds
```

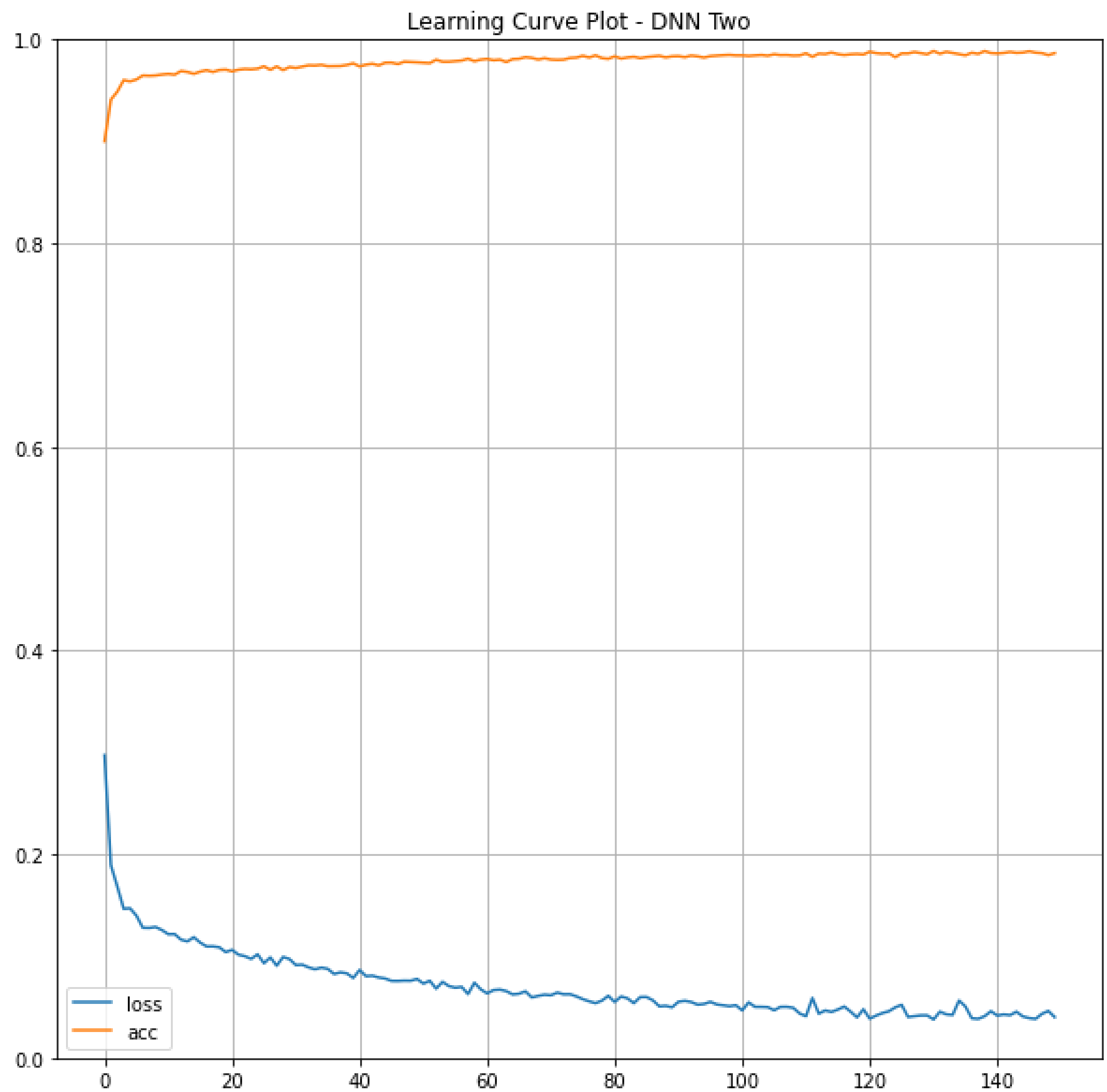
- Model Two:

```
dnn_two = keras.models.Sequential([
    keras.layers.Dense(600, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(450, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(300, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(150, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(100, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(50, activation = "elu", kernel_initializer = "he_normal"),
    keras.layers.Dropout(rate = 0.2),
    keras.layers.Dense(1, activation = "sigmoid")
])
```

```
dnn_two.compile(loss = "binary_crossentropy",
                optimizer = "nadam",
                metrics = ["accuracy"])
```

```
start_time_dnn_two = time.time()

history_two = dnn_two.fit(X_train, y_train, epochs = 150)

end_time_dnn_two = time.time()
```

```
dnn_two.summary()
_________________________________________________________________
dropout_6 (Dropout)          multiple                  0
_________________________________________________________________
dense_18 (Dense)             multiple                  270450
_________________________________________________________________
dropout_7 (Dropout)          multiple                  0
_________________________________________________________________
dense_19 (Dense)             multiple                  135300
_________________________________________________________________
dropout_8 (Dropout)          multiple                  0
_________________________________________________________________
dense_20 (Dense)             multiple                  45150
_________________________________________________________________
dropout_9 (Dropout)          multiple                  0
_________________________________________________________________
dense_21 (Dense)             multiple                  15100
_________________________________________________________________
dropout_10 (Dropout)         multiple                  0
_________________________________________________________________
dense_22 (Dense)             multiple                  5050
_________________________________________________________________
dropout_11 (Dropout)         multiple                  0
_________________________________________________________________
dense_23 (Dense)             multiple                  51
=================================================================
Total params: 477,101
Trainable params: 477,101
Non-trainable params: 0
_________________________________________________________________
```




```
print("Elapsed Training Time: {} seconds".format(end_time_dnn_two - start_time_dnn_two), "\n")
```

- Runtime on local machine:

```
Elapsed Training Time: 343.7559051513672 seconds
```

- Runtime on AWS:

```
Elapsed Training Time: 409.19395208358765 seconds
```

5). Random Forest:

- Random Forest One:

```
rf_clf = RandomForestClassifier(n_estimators = 500, max_leaf_nodes = 16, random_state = 42)

start_time_rf_one = time.time()

rf_clf.fit(X_train, y_train)

end_time_rf_one = time.time()

y_pred_rf = rf_clf.predict(X_test)
```

```
print("Elapsed Training Time: {} seconds".format(end_time_rf_one - start_time_rf_one), "\n")
```

  - Runtime on local machine:

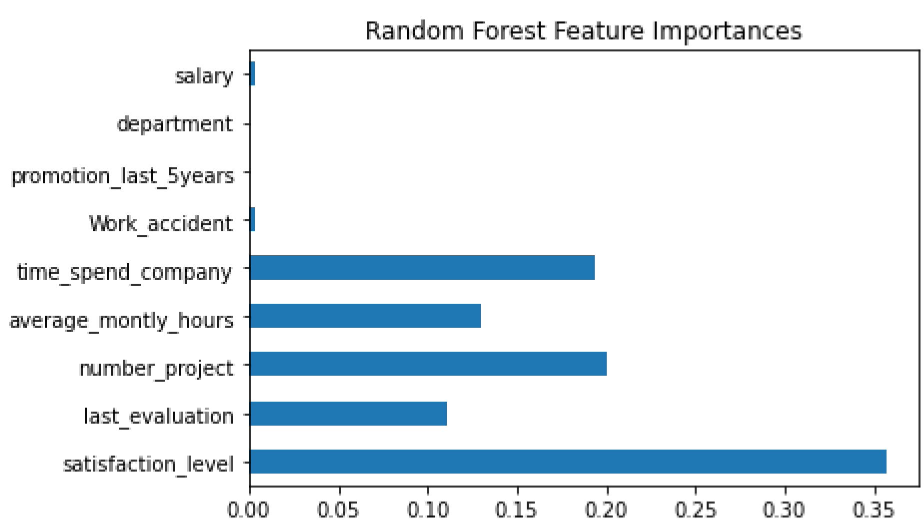
```
Elapsed Training Time: 2.005603075027466 seconds
```

  - Runtime on AWS:

```
Elapsed Training Time: 1.840357780456543 seconds
```

  - Random Forest One Feature Importances:

```
rf_importances = pd.Series(rf_clf.feature_importances_, index = hr.drop(columns = ["left"], axis = 1).columns)
rf_importances.plot(kind = "barh")
plt.title("Random Forest Feature Importances");
```



- Random Forest Two:
  - Removes the max_leaf_nodes hyperparameter entirely

```
rf_two_clf = RandomForestClassifier(n_estimators = 500, random_state = 42)

start_time_rf_two = time.time()

rf_two_clf.fit(X_train, y_train)

end_time_rf_two = time.time()

y_pred_rf_two = rf_two_clf.predict(X_test)
```

```
print("Elapsed Training Time: {} seconds".format(end_time_rf_two - start_time_rf_two), "\n")
```

  - Runtime on local machine:

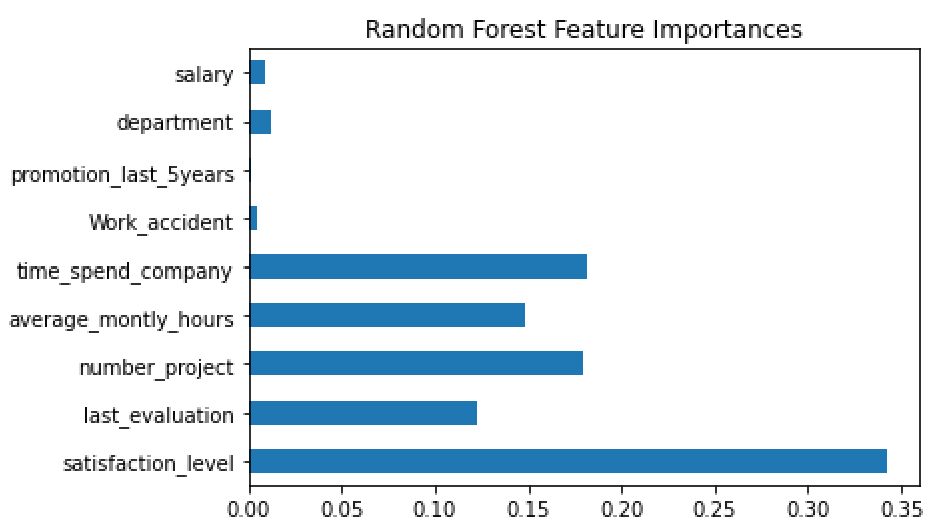
```
Elapsed Training Time: 3.035227060317993 seconds
```

  - Runtime on AWS:

```
Elapsed Training Time: 3.060112237930298 seconds
```

  - Random Forest Two Feature Importances:

```
rf_two_importances = pd.Series(rf_two_clf.feature_importances_, index = hr.drop(columns = ["left"], axis = 1).columns)
rf_two_importances.plot(kind = "barh")
plt.title("Random Forest Feature Importances");
```



6). Evaluating Models on Test Set:

- DNN One:

```
dnn_one.evaluate(X_test, y_test)
```

```
5100/5100 [==============================] - 0s 52us/sample - loss: 0.0994 - acc: 0.9727
[0.09935453007326407, 0.9727451]
```

- DNN Two:

```
dnn_two.evaluate(X_test, y_test)
```

```
5100/5100 [==============================] - 0s 60us/sample - loss: 0.1944 - acc: 0.9761
[0.1944141926448427, 0.97607845]
```

- Random Forest One:

```
print(sk.confusion_matrix(y_test, y_pred_rf), "\n")
print(sk.classification_report(y_test, y_pred_rf), "\n")
print("Accuracy:", sk.accuracy_score(y_test, y_pred_rf))
```

```
[[3875   13]
 [ 128 1084]]

              precision    recall  f1-score   support

           0       0.97      1.00      0.98      3888
           1       0.99      0.89      0.94      1212

    accuracy                           0.97      5100
   macro avg       0.98      0.95      0.96      5100
weighted avg       0.97      0.97      0.97      5100


Accuracy: 0.9723529411764706
```

- Random Forest Two:

```
print(sk.confusion_matrix(y_test, y_pred_rf_two), "\n")
print(sk.classification_report(y_test, y_pred_rf_two), "\n")
print("Accuracy:", sk.accuracy_score(y_test, y_pred_rf_two))
```

```
[[3880    8]
 [  56 1156]]

              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3888
           1       0.99      0.95      0.97      1212

    accuracy                           0.99      5100
   macro avg       0.99      0.98      0.98      5100
weighted avg       0.99      0.99      0.99      5100


Accuracy: 0.9874509803921568
```

Summary

The first deep neural network (DNN One) was implemented with batch normalization while the second deep neural network (DNN Two) was implemented with dropout to aid with regularization and in turn alleviate issues with overfitting should they arise. As this is a binary classification problem, both DNNs utilize sigmoid as the activation function in the final layer. DNN One utilizes SGD as the optimizer function whilst DNN Two utilizes nadam as its optimizer function. For the dense layers, DNN One utilizes the rectified linear unit (ReLU) as the activation function, while DNN Two utilizes the exponential linear unit (eLU) as the activation function. DNN Two utilizes dropout after every dense input layer with a rate of .2.

DNN One yielded an accuracy of .9727 when evaluated on the test set. On my personal machine this particular model took 168.24 seconds to train compared to 377.35 seconds on an AWS EC2 instance.

DNN Two yielded a slightly higher accuracy of .9761 when evaluated on the test set. On my personal machine this particular model took 343.75 seconds to train compared to 409.19 seconds on an AWS EC2 instance.

The first Random Forest model was implemented with a max number of trees as 500 and the maximum number of leaf nodes limited to sixteen. This model yielded an accuracy of .9723 when evaluated on the test set. On my personal machine this particular model took 2.00 seconds to train compared to 1.84 seconds on an AWS EC2 instance. The feature importance plot for this particular model indicates that “satisfcation_level”, “nnumber_project” and “time_spend_company” had the most impact on its predictions. The feature importance plot also indicates that “department” and “promotion_last_5years” had virtually no impact on predictions. This aligns with some of my findings from the exploratory data analysis as well.

The second Random Forest model was implemented with a max number of trees as 500 and no limitation set on the maximum number of leaf nodes. This model yielded a slightly higher accuracy of .99 when evaluated on the test set. On my personal machine this particular model took 3.03 seconds to train compared to 3.06 seconds on an AWS EC2 instance. The feature importance plot for this random forest model indicates that “satisfaction_level” had the most profound impact on its predictions, with “time_spend_company” and “number_project” also having a sizable impact. The feature importance plot also indicates that “department” and “promotion_last_5years” have virtually no impact, similar to the previous random forest model.

Within the context of employee retention, I believe that the deployment of a random forest model would be more advantageous. While the neural networks offered very high accuracies at .9727 and .9761, the random forests performed equally as well with accuracies of .9723 and .99. The random forests also exhibited precision and recall scores of .99 and .89 for the first model, respectively, as well as .99 and .95 for the second model, respectively.

The most valuable advantage for the random forest however lies in the ability to obtain the feature importances for these particular models. If the goal of a company is to maximize its employee retention rates, the feature importance plots, when combined with quality exploratory data analysis, can offer an incredible amount of insight as to what drives employees to stay with the company or leave.

The random forest models were also considerably faster to train than the neural networks. If these models were to be deployed at a fortune 500 company with tens of thousands employees, the speed advantage could be quite useful. For example, if employees are evaluated quarterly, or even more frequently, that would require the model(s) to be retrained just as frequently to satisfy that feature alone. In a very large company, retraining a neural network frequently in such a context is most likely not worth the tradeoff in speed given the nearly non-existent difference in performance and the other advantages of random forest.

Overall, I believe the largest advantage for the random forest models in this context lies within the feature importance plots that are available for such models. Given the similar performance of these models to that of the neural networks and the speed advantage, I do not see any particular reason that a neural network would be a better model to deploy in this particular context.