Concise Report

Name: Darsh Kachroo

Mailing Address: kachroo.darsh@gmail.com

Problem Statement: Predicting Employee Attrition

Dataset: IBM HR Analytics Employee Attrition & Performance

Source: Kaggle

Summary of Analysis

In the visualization and analysis of the dataset it was found that:

- Age positively impacts the variance in stock options. New hires tend to be at similar levels, but promotion creates more variation as employees age.
- In Human Resources, overtime work becomes less important for receiving stock options as employees gain experience. This suggests the company incentivizes new hires to stay.
- Research and Development offers higher stock options to younger employees regardless of overtime to retain them, while experienced workers receive more for less work, prioritizing quality over quantity.
- Job satisfaction data allows for identifying outliers who may need intervention.
- Average job level helps determine where to focus hiring efforts.
- Women receive the highest starting hourly rates but experience a decline relative to men as training time increases. Satisfaction also appears to decrease for women with more training.
- Employees with moderate experience show the highest job satisfaction with minimal pay difference between genders.

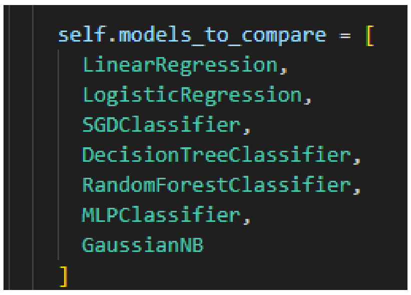
In the analysis of the dataset, the following steps were taken:

- Feature extraction methods
    - Chi^2 test
    - Correlation based
- Data processing methods
    - unscaled data
    - scaled data
    - unscaled pca
    - scaled pca
    - unscaled t-sne
    - scaled t-sne
- Predicting models
    - Linear Regression
    - Logistic Regression
    - SGD Classifier
    - Decision Tree
    - Random Forest
    - MLP
    - Gaussian Naive Bayes
- Hyperparameter tuning
- Ensemble Models

Challenges Encountered

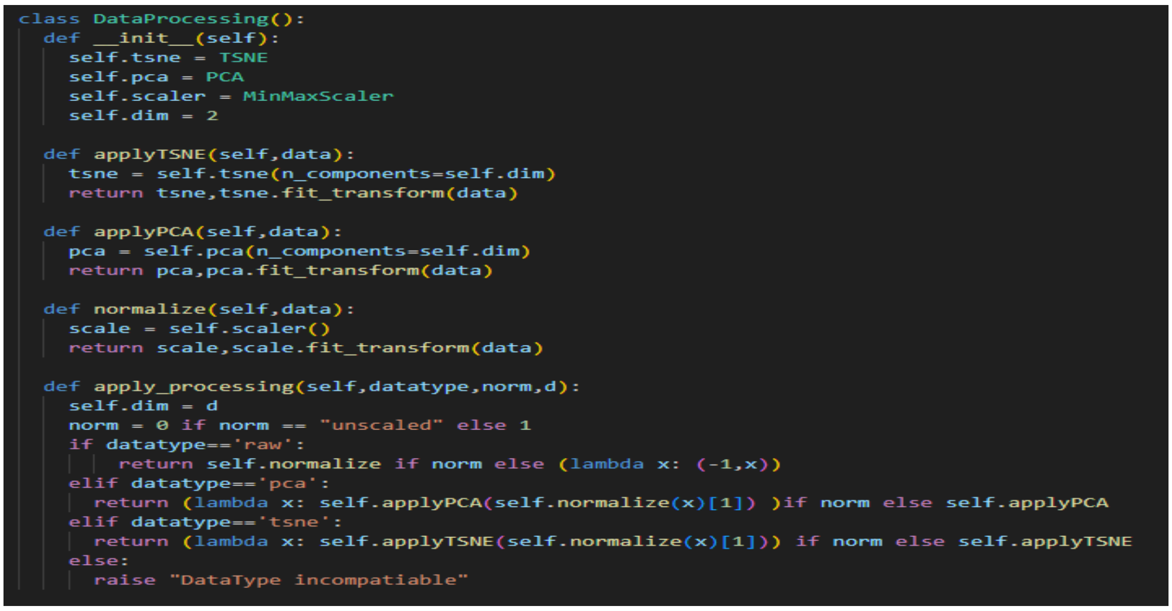
One of the major challenges was making sure the data was flowing smoothly to and from each part in analysis for prediction of attrition. Since speed of the inferences is also to be taken into account for a good workflow.

An example which made big changes in the code was working on the model training class. Since the models had to be new versions of themselves to train again and again, initially I had decided to make instances of the models inside the for loop but then I realised while making the hyperparameter tuning function that I can have an array of the class and not the objects. This way I can easily iterate through the classes and still make new objects from them. This gave a massive speed boost as well as prevented overfitting



```python
self.models_to_compare = [
  LinearRegression,
  LogisticRegression,
  SGDClassifier,
  DecisionTreeClassifier,
  RandomForestClassifier,
  MLPClassifier,
  GaussianNB
]
```

A similar challenge was faced when making the ensemble class because the preprocessing function for each model was different, but the scaler function used to process the data should be stored. This can only be done once the data had originally been given. However, I wanted to make a general model class like ones provided by libraries which work directly with the input data. This gave me the idea to modify the data processing class to return the scalars along with the fitted values. This can now be integrated with the ensemble code whenever the model is used. And any use will trigger the setup, overwriting the processing functions making them into scalars functions as intended.



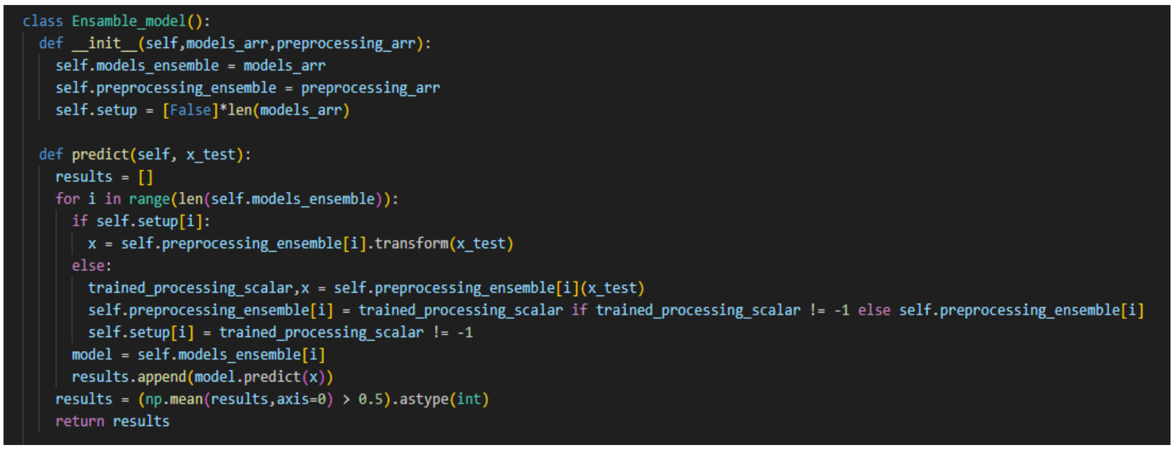
```python
class DataProcessing():
  def __init__(self):
    self.tsne = TSNE
    self.pca = PCA
    self.scaler = MinMaxScaler
    self.dim = 2

  def applyTSNE(self,data):
    tsne = self.tsne(n_components=self.dim)
    return tsne,tsne.fit_transform(data)

  def applyPCA(self,data):
    pca = self.pca(n_components=self.dim)
    return pca,pca.fit_transform(data)

  def normalize(self,data):
    scale = self.scaler()
    return scale,scale.fit_transform(data)

  def apply_processing(self,datatype,norm,d):
    self.dim = d
    norm = 0 if norm == "unscaled" else 1
    if datatype=='raw':
        return self.normalize if norm else (lambda x: (-1,x))
    elif datatype=='pca':
      return (lambda x: self.applyPCA(self.normalize(x)[1]) )if norm else self.applyPCA
    elif datatype=='tsne':
      return (lambda x: self.applyTSNE(self.normalize(x)[1])) if norm else self.applyTSNE
    else:
      raise "DataType incompatiable"
```



```python
class Ensamble_model():
  def __init__(self,models_arr,preprocessing_arr):
    self.models_ensemble = models_arr
    self.preprocessing_ensemble = preprocessing_arr
    self.setup = [False]*len(models_arr)

  def predict(self, x_test):
    results = []
    for i in range(len(self.models_ensemble)):
      if self.setup[i]:
        x = self.preprocessing_ensemble[i].transform(x_test)
      else:
        trained_processing_scalar,x = self.preprocessing_ensemble[i](x_test)
        self.preprocessing_ensemble[i] = trained_processing_scalar if trained_processing_scalar != -1 else self.preprocessing_ensemble[i]
        self.setup[i] = trained_processing_scalar != -1
      model = self.models_ensemble[i]
      results.append(model.predict(x))
    results = (np.mean(results,axis=0) > 0.5).astype(int)
    return results
```

Recommendations to decrease attrition

According to the Chi^2 test, the attrition is most dependant on Daily Rate, Monthly Income Monthly Rate, Total Working Years, and Years at Company. Hence increasing these factors will decrease the employee attrition.

According to the correlation matrix test, the attrition is most dependent on Age, Job Level, Marital Status, Monthly Income, Over Time, Total Working Years, Years in Current Role, and Years with Curr Manager. Hence increasing these factors will decrease the employee attrition.

According to the model results with chi^2 and correlation tests, the correlation tests showed more consistent results and is why is recommended to follow