Tyler Hayes

NJIT-CS634

**Final Project**

*Last Modified: July 8th, 2024*

***Contents***

[**Overview** 2](#_Toc171364280)

[**Setup** 2](#_Toc171364281)

[**Exploratory Data Analysis** 4](#_Toc171364282)

[Hypothesis 4](#_Toc171364283)

[Numerical Features 5](#_Toc171364284)

[Categorical Features 5](#_Toc171364285)

[**Pre-Processing** 7](#_Toc171364286)

[Target Label Encoding 7](#_Toc171364287)

[Categorical & Numeric Encoding 7](#_Toc171364288)

[**Model Architecture** 8](#_Toc171364289)

[LSTM 8](#_Toc171364290)

[Random Forest 9](#_Toc171364291)

[K-Nearest Neighbors 9](#_Toc171364292)

[**Model Training** 10](#_Toc171364293)

[Preparation 10](#_Toc171364294)

[Training 11](#_Toc171364295)

[**Model Evaluation** 12](#_Toc171364296)

[Fold Evaluation 12](#_Toc171364297)

[Aggregate Evaluation 12](#_Toc171364298)

[**Findings** 12](#_Toc171364299)

[**Appendix** 13](#_Toc171364300)

[**Screenshots** 13](#_Toc171364301)

# **Overview**

* + **Algorithms**
    - **Random Forest** (sklearn.ensemble - RandomForestClassifier)
    - **LSTM** (tensorflow.keras.layers - LSTM)
    - **K-Nearest Neighbors** (sklearn.neighbors - KNeighborsClassifier)
  + **Dataset**
    - Census Income - <https://archive.ics.uci.edu/dataset/20/census+income>
  + **Task**
    - This project will attempt to utilize the above three algorithms to correctly classify whether a given individual has an income greater than $50,000. Because of this, the task is fundamentally a binary classification problem. The individual being predicted either does (affirmative) or does not (negative) make greater than $50,000 a year.

# **Setup**

This project is contained entirely within the notebook (CS634 Final Proj.ipynb). Any dependencies which require a package to be installed (tensorflow, sklearn, tqdm) are checked prior to the main body of the code running. If it detects that any of these packages are not available, it will attempt to install them:

As a result, installation should be as simple as clicking “Run All” in your respective notebook environment. Any dependencies which are needed should be installed by the above snippet.

Figure : The notebook successfully checks whether dependencies are installed.

Data will be grabbed from the “/data/” folder relative to the notebook’s working directory. Because of this, if you clone the repository and run the notebook you should be able to successfully run all cells without any modifications. The data should be in the same relative path, and ingestion of the data into this notebook should be seamless.



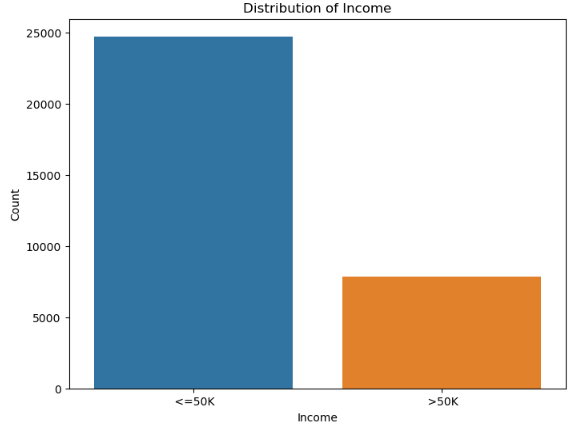
Figure : The notebook successfully loads in the selected dataset into a Pandas dataframe.

This should account for all necessary setup of this project, as the data will be automatically detected from the local file system upon execution. All other aspects of the notebook’s code are self-contained and will not require any intervention for running.

# **Exploratory Data Analysis**

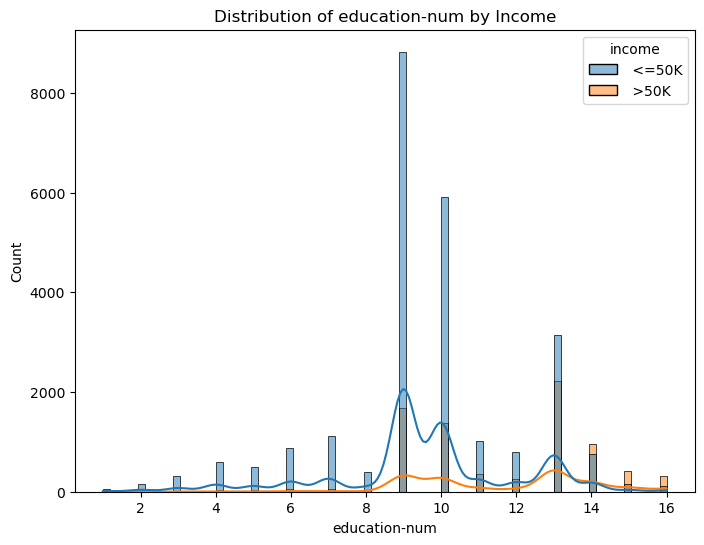
## Hypothesis

Upon initial examination, my hypothesis was that the features which we would find to be most important were the socioeconomic factors that typically correlated with higher incomes: education, sex, marital status, and race. I began looking at the data by checking the target class imbalance which was roughly 1:4 in favor of the negative class (“<=50K”); 24.08% of records were in the affirmative class:

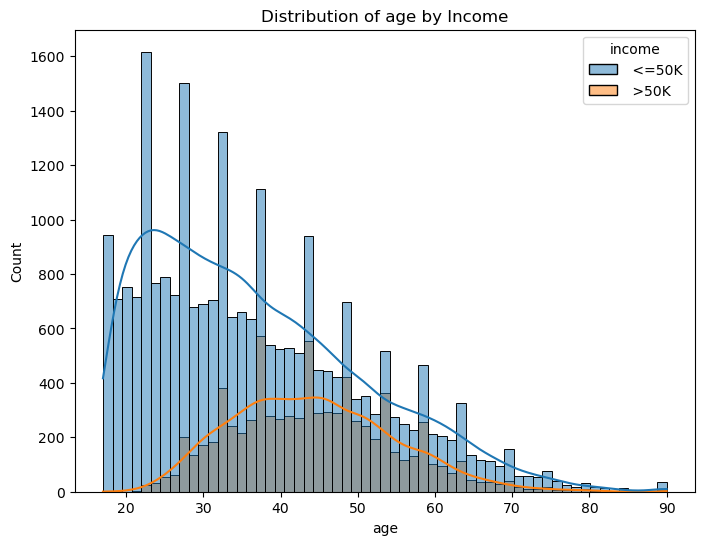


## Numerical Features

For education, it became apparent from looking at the distribution that education would have some relevance, since the number of educational years did seem to trend with individuals making over $50,000.

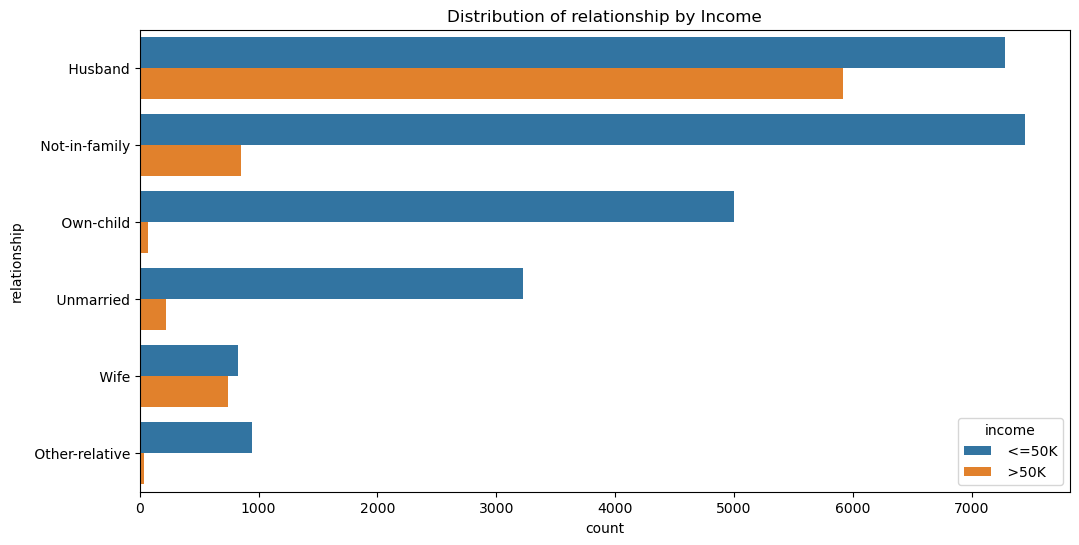


I could also see, from looking at the distribution by age, that the proportion of the class label was not evenly distributed across the various ages. Older individuals were far more likely to be in the affirmative class, which makes sense given their higher potential for experience.

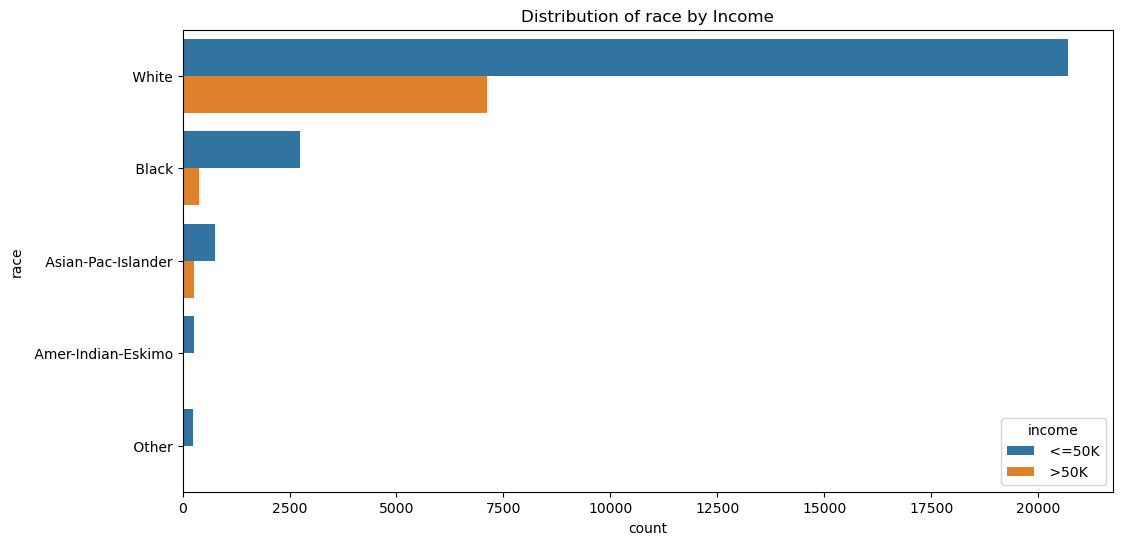


## Categorical Features

Marital status also had a notably disproportionate relationship with the class label. Individuals who were married (“Husband”/”Wife”) were far more likely to be in the affirmative class when compared to other marital statuses.



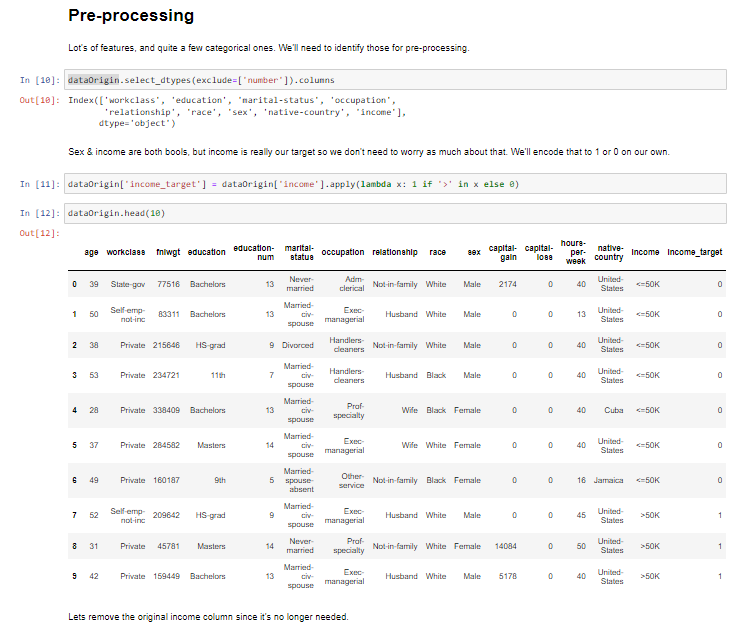
Finally, race also appeared to have a significant difference in the distribution relative to the class label. White individuals were proportionally far more likely to be in the affirmative class when compared to other racial groups.



# **Pre-Processing**

## Target Label Encoding

A simple 1 & 0 were used to encode the class label to give it a true affirmative & negative class. To do this, a basic search function to see if “>” appears in the income field was used. If the character was found the income\_target field was set to 1 (affirmative) and if it was not the field was set to 0 (negative). This keeps the same consistent descriptive terms (affirmative/negative) that I have used earlier on.



## Categorical & Numeric Encoding

All features were placed into a pipeline using the scikit-learn library. A list of numeric features and a separate list of categorical features was created so that they could be passed into the pipelines separately. All numeric features were encoded with a standard scaler & median imputation. All categorical features were encoded with one hot encoding & imputation using the most frequent label.



This enabled me to have a singular place to point each model to that had identical transformations applied. As we proceed, I will utilize this preprocessor specifically on each subsequent model, without making any changes to the object itself.

# **Model Architecture**

## LSTM

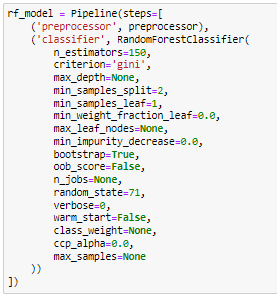
﻿ The LSTM begins at the input layer. The input layer is defined to accept sequences of data based on the input data. The first LSTM layer has 50 units and uses the ReLU activation function. A dropout layer with a 0.2 dropout rate follows the first LSTM layer. This helps prevent overfitting by randomly setting 20% of the input units to 0 each update during training. Another LSTM layer with 50 units and ReLU activation is used. Another dropout layer with a dropout rate of 0.2, like the first dropout layer.

A dense layer with a single unit and a sigmoid activation function is then used to coerce an output. The layer provides a probability value indicating the likelihood of the input sequence belonging to class 1 (binary classification).

Because I had some issues with the Keras standard wrapper, I created a simple wrapper for Keras so that the model can interact (fit/predict) using scikit-learn’s API. After that, the pipeline is used with the ReshapeTransformer class to build the LSTM model before training.

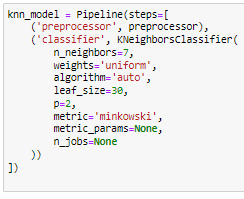
## Random Forest

The random forest model was configured to use 150 estimators and the Gini Index. I decided against setting a maximum depth since I was curious about how performance would be without one. Similarly, I left the max leaf nodes at none so that it wouldn’t terminate from hitting the maximum. I did leave the default minimum samples for the split/leaf. I felt that there could still be issues from sparsity created if I did that, so I left those at their default values.



## K-Nearest Neighbors

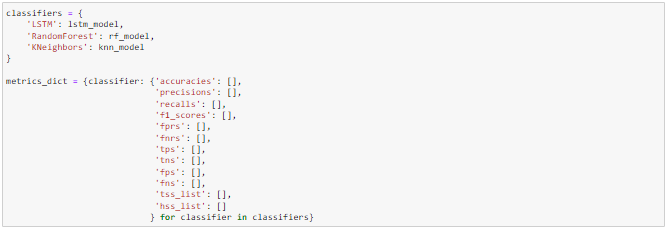
The K-Nearest Neighbor model was configured using many of the defaults. I changed the **n\_neighbors** parameter at 7, though I did experiment with a few other options to see if changes would benefit the performance. I kept **p** = 2 to retain Euclidean distance as the distance metric used, but also experimented with the Manhattan distance to no benefit.



# **Model Training**

## Preparation

Each of the models I want to test will have a dictionary entry created in a new dictionary called **classifiers**. I’ll iterate over that dictionary to create a new one with a set of blank lists, having one list for each metric I will eventually calculate. This will create a foundation that I can insert data into, which will eventually be shifted into tabular data.



## Training

The notebook will iterate over each model that is in the classifier dictionary. It will initialize a 10-fold cross-validation. As it gets to each classifier it will train it on the pipeline’s data by fitting the model and predicting for that given fold of the cross-validation process. As soon as the predictions are made each of the evaluation metrics is calculated and appended to the lists within our metric dictionary:

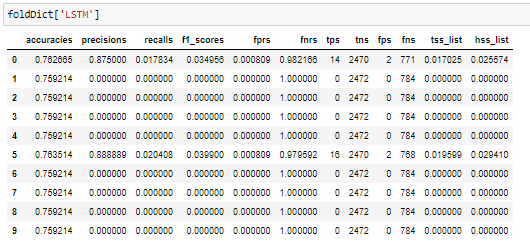
REPLACE THIS WITH LATEST CODE CELL

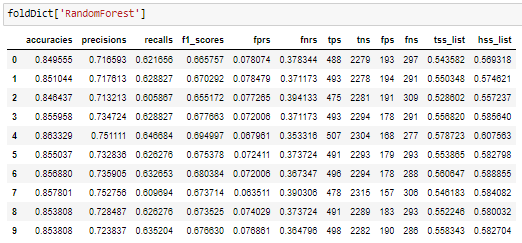
Because this cell is doing a great deal in one execution, I took the liberty of implementing [tqdm](https://github.com/tqdm/tqdm) so that a progress bar will be displayed as each fold is processed. This will give you some insight into expected duration as well as some feedback that the notebook is still running. I also report out the duration of each fold as they complete for the sake of transparency.

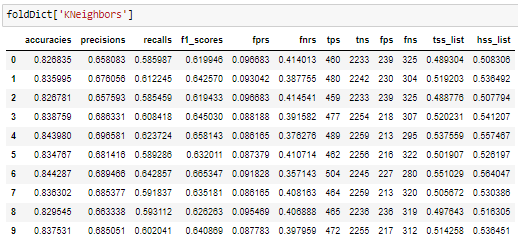
# **Model Evaluation**

## Fold-level Evaluation

Each model has ten folds that can be seen and evaluated within the notebook. The metrics are displayed separately via distinct code cells which return the dataframe of all evaluation metrics related to the model.

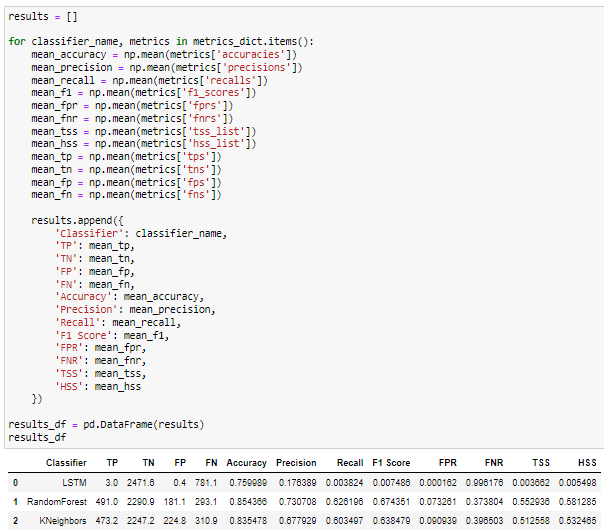






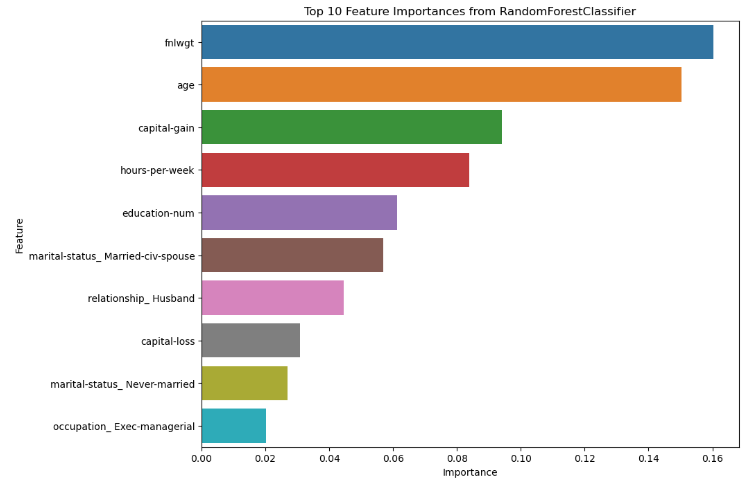
## Aggregate Evaluation

After the results have been stored in fold-level metrics, we can easily grab them and aggregate them using an average of our choosing. I’ll take the mean of each and display them as the final metrics for the project.



## Feature Importance

### Random Forest



### K-Nearest Neighbors

ADD IN THE FEATURE IMPORTANCES HERE FOR KNN

# **Findings**

* Random Forest proved to routinely be the most effective model for this task. Regardless of whether I was picking based on accuracy or F1 the Random Forest, it outperformed the other models and was a clear winner through every iteration of testing.
* LSTMs are a bit harder to work with relative to the other models. Both the initialization and interoperability of the LSTM model were more complex than the Random Forest & K-Nearest Neighbor models. Random Forest & K-Nearest Neighbors both work “out of the box” so to speak, with little customization being necessary.
* Despite multiple iterations during development, LSTM models routinely proved to be “conservative” predictions. They would typically either guess 100% of their predictions as the majority class label or very close to 100%. It took implementing a very complex architecture to get the model to predict anything other than the negative class label.
* The ‘**fnlwgt**’ attribute proved to have high feature importance in the Random Forest & K-Nearest Neighbors model. Oddly, the data dictionary for this dataset does not provide any description of what this feature is. Future tests with this project might investigate dropping it unless it can be shown to have some meaning.
* Many of the attributes I hypothesized being important (age, education, and marital status) did have a high feature importance on the Random Forest & K-Nearest Neighbors models. This makes sense given how frequently these play a role in how much a person makes.
  + Two important features which I did not predict to be important were ‘capital-gain’ and ‘hours-per-week’.
    - In retrospect, the capital gain should be obviously correlated with high income. Individuals who have the luxury of earning capital gains are likely to be those who make enough to have a disposable income which allows them to invest money.
    - Hours per week is a less intuitive one since individuals working 40 hours a week at a tech job can easily clear $50,000 anywhere in the U.S. while a minimum wage worker doing over-time (~50 hours) may not break $50,000 reliably. In retrospect though, this does make sense that on average it has predictive power when looking at incomes above/below $50,000.
* ADD NOTE ABOUT RUN TIME REQUIRED TO TRAIN LSTM

# **Appendix**

ADD GITHUB LINK

## **Screenshots**

