# U.S. Census Income Prediction – Final Term Project

Jonathan Samuel

Department of Computer Science, Virginia Polytechnic Institute

CS5525: Data Analytics I

Dr. Reza Jafari

# Table of Contents

## Abstract

This project focuses on a US Census dataset to predict if an individual makes more or less than $50,000 annually. The dataset consists of roughly 50k entries containing 8 categorical and 5 numerical socioeconomic and population census features. The datasets were cleaned and transformed using a Power Transformer (to adjust skewness of two features), Normalization and a One-Hot encoder on the categorical features. After one-hot encoding, there were 87 total features (columns). The dataset was split prior to transformation in an 80/20 train/test fashion. A dimensionality reduction step using single variable decomposition and principal component analysis concluded with 18-20 components that explain 85% variance. A random forest classification analysis was also performed with a 0.0075 feature importance threshold reducing the feature space to 18 features with age, education and hours worked per week as the three most important features to predict income. This reduced feature space was assessed further using an f-test, t-test, collinearity assessment and backwards stepwise regression which ultimately led to using the same set of 18 features resulting with a logistic regression model have 0.837 accuracy. Four additional models were constructed, decision tree, KNN, support vector machine and Gaussian Naïve Bayes. The optimized logistic regression performed the best amongst the base models of the other four classifiers. However, after performing a hyper tuning adjustment with grid search – cross validation, the decision tree with the following parameters criterion: entropy, max_depth: 10, min_samples_leaf: 4, min_samples_split: 2, outperformed all models in terms of accuracy, precision, f-measure and specificity resulting with a finalized accuracy score of 0.851.

## Introduction

This project focuses on a US Census dataset to predict if an individual makes more or less than \$50,000 annually. In phase I, I focused on exploratory data analysis that included data cleaning, visual exploratory analysis, data transformations and dimensionality reduction. This section was to prepare the datasets for regression and classification analysis. Once the datasets were prepared and features were finalized, in phase II, I constructed a logistic regression model. In this section I further assessed relationship amongst the features by performing an f-test, t-test, collinearity, and backward stepwise regression. This resulted with a logistic regression model with an accuracy score of 0.837. In phase III, I constructed other classification models include decision tree, KNN, support vector machine and Gaussian Naïve Bayes. I evaluated default parameter performance and then performed hyper tuning by conducting a grid search with cross validation on various parameters for each model. The final section concludes with model preference, future work and any learnings that were found throughout the project.

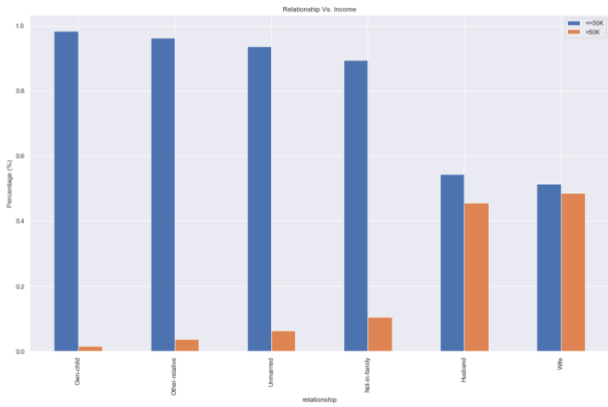
## Dataset

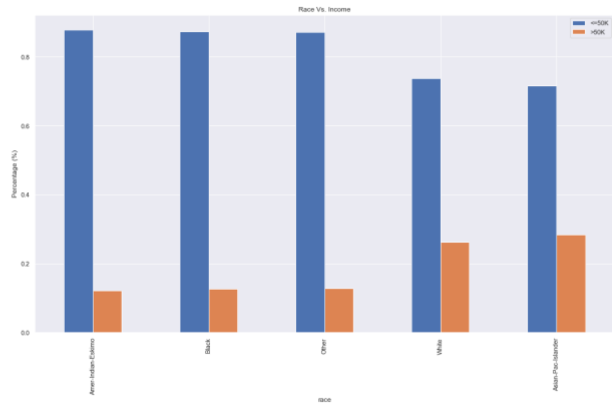
This project focuses on the Census Income Data Set provided publicly by the University of California Irvine. [1] This dataset contains approximately 49,000 entries between the train and test datasets provided. There are exactly eight categorical and five numerical features that act as independent variables and are composed of various socioeconomical and population characteristics such as work-class, education level, sex, and race. There is a single dependent variable labeled as income, that has a binary relationship stating whether an entry makes less than or equal to \$50,000 annually or more. The goal of this dataset is to use the provided features to construct a classification model that can predict if an individual will make less than or equal to \$50,000 annually. Because this dataset comes from the US census, it is important as it describes characteristics with respect to communities in the United States. This can showcase various features about our nation such as highlight areas that require focus and allow the government to distribute funds and assistance to communities appropriately.

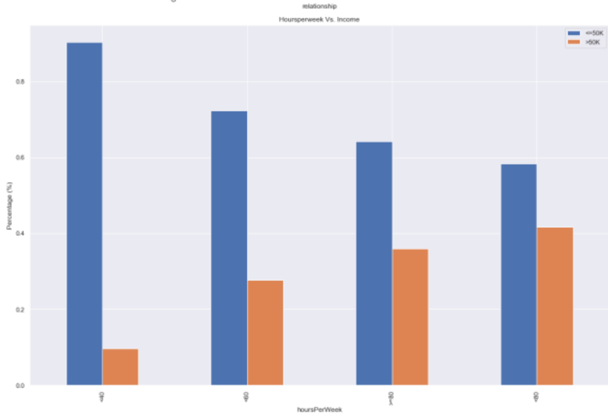
## Phase I – Exploratory Data Analysis

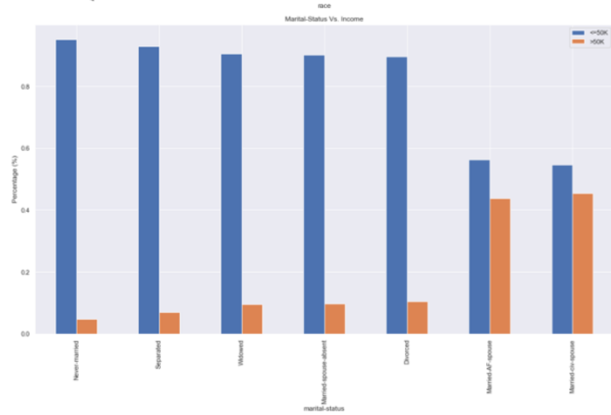
### Data Cleaning

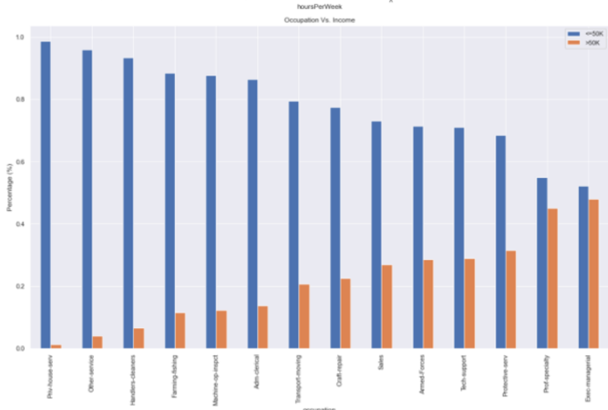
The first step to produce a statistical model that predicts whether an individual will make more or less than \$50,000 a year is to perform exploratory data analysis. In this step, I handled any missing data, outliers, or redundant entries. Specifically, I combined both train and test datasets so that I can split my data manually providing my own random seed for model evaluation downstream. I also mapped the target feature to a binary 0,1 relationship (<=\$50K:0, >\$50K:1). There were no missing data upon first look, however I noticed there were “?” placeholders within the work-class, occupation and native-country features that composed of roughly 7.4% of data. Because these were categorical features and did not make up a significant population of my dataset, I removed those entries containing “?”. I also removed the education feature since there was an education-num feature that essentially applied a label encoder onto the education feature making one redundant.

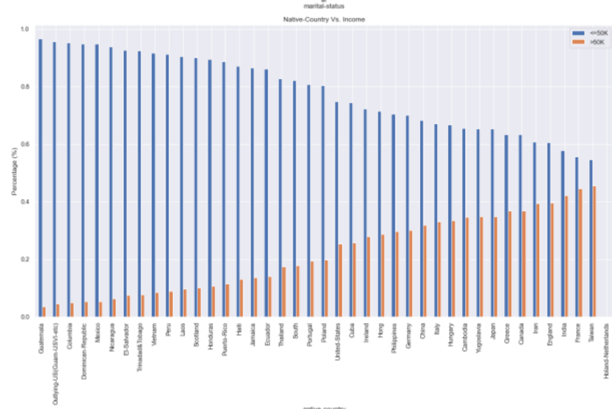


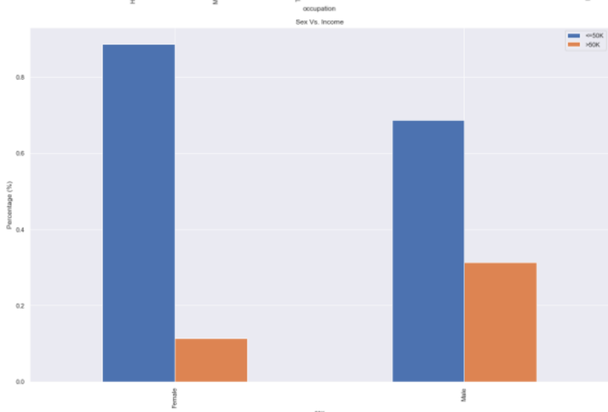



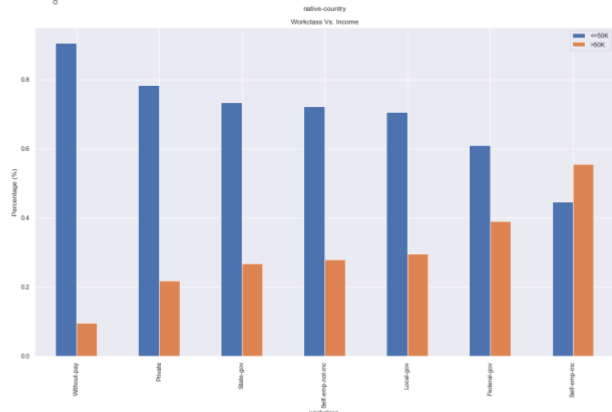



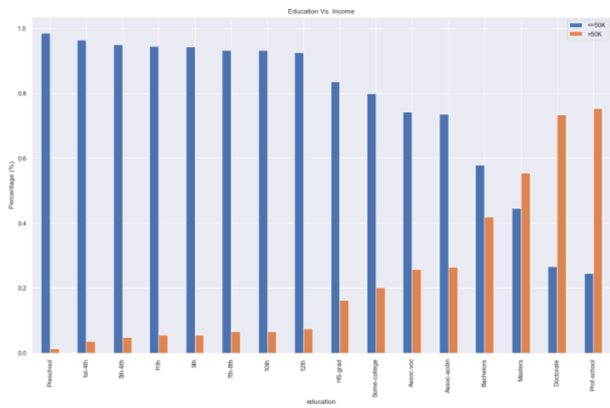



**Figure 1:** Shows the distribution of income amongst categorical feature's labels and ordered by distribution making over $50,000 in ascending order. From left to right, top to down: (a) Relationship status (b) Race (c) Hours worked per week (d) Martial Status (e) Occupation (f) Native country (g) Sex (h) Work class (i) Education

## Visual Exploratory Analysis

I next wanted to visually understand the dataset by plotting the percentage distribution of the target feature with respect to categorical features and their labels. Interestingly in figure 1c, those working between 60-80 hours has a higher distribution of individuals making more than $50k in comparison to those working over 80 hours. In figure 1g, there was a higher distribution of males making more than $50k annually than females. Most other categories followed stereotypical trends such as those with higher education generally had a higher distribution of individuals making more than $50k annually (figure 1i). Of course, this information can all be skewed if there were lacking entry points for each respective category and label.

## Data Transformations

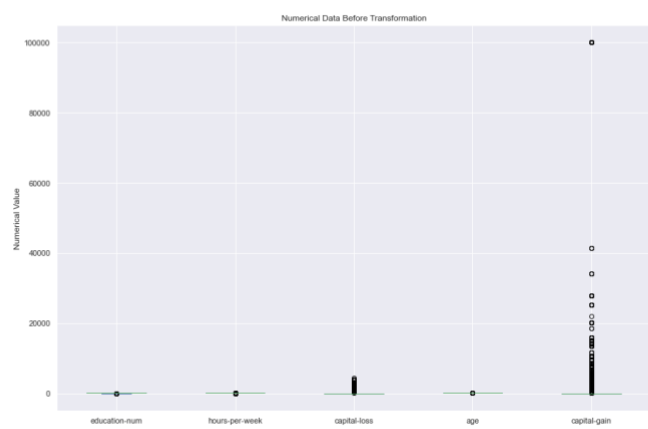



**Figure 2:** Box and whisker plot of numerical features before transformation

| **Numerical Feature** | **Skewness Score** | **Skewness Score after PowerTransformer** |
|---|---|---|
| Capital Gain | 11.95 | 4.26 |

| | | |
|---|---|---|
| Capital Loss | 4.52 | 3.02 |
| Age | 0.53 | 0.53 |
| Hours-per-week | 0.31 | 0.31 |
| Education-num | -0.30 | -0.30 |

**Table 1:** Skewness scores for numerical features

After understanding a bit more about the characteristics of the dataset, I prepared my data for data transformation by spitting an 80/20 (train/test) with a random seed of 17. Figure two shows a box and whisker plot of the five numerical features. There is a visible magnitude difference amongst capital-gain and loss in comparison to other features. Before handling this through normalization, I determined if there was any skewness within the distribution of these five numerical features. Table 1 shows capital gain and loss having a high skewness score and is recommended that the absolute value of skewness scores above 1 should be examined and treated. [2] I applied a Power Transformer to handle skewness for capital gain and loss which maps the distribution of the feature(s) to make them more *Gaussian-like.* [3] In table 1, you'll notice after applying the Power Transformer on capital gain and loss, there still appears to be high skewness but significantly reduced from ~12 to 4 and 4.5 to 3, respectfully.

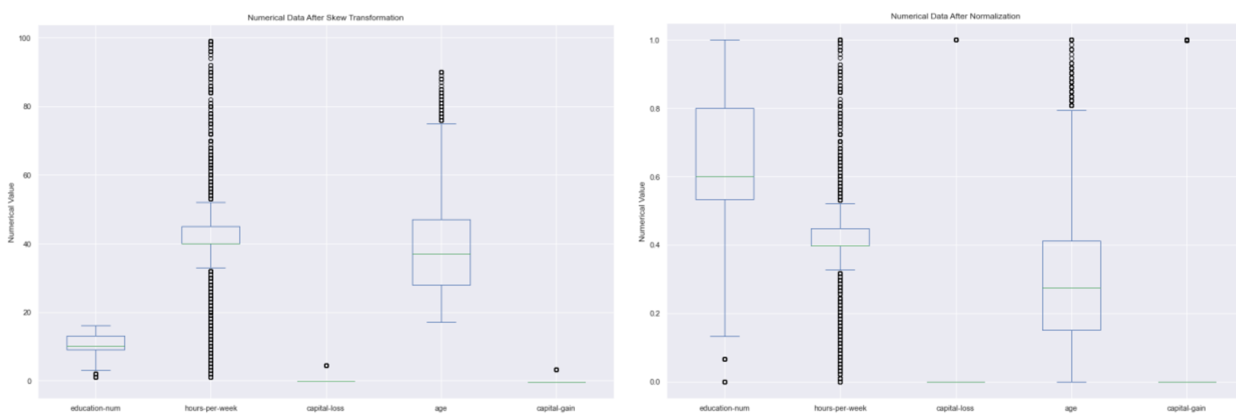



**Figure 3:** (left) Box and whisker plot of numerical features after applying Power Transformer on capital gain and loss (right) Box and whisker pot of numerical features after applying Normalization on all features.

| relationship_Husband | relationship_Not-in-family | ... | native-country_Yugoslavia | sex_Female |
|---|---|---|---|---|
| 0.0 | 0.0 | ... | 0.0 | 0.0 |
| 0.0 | 0.0 | ... | 0.0 | 1.0 |
| 0.0 | 1.0 | ... | 0.0 | 1.0 |

| relationship_Husband | relationship_Not-in-family | ... | native-country_Yugoslavia | sex_Female |
|---|---|---|---|---|
| ... | ... | ... | ... | ... |
| 1.0 | 0.0 | ... | 0.0 | 0.0 |
| 0.0 | 1.0 | ... | 0.0 | 0.0 |

**Table 3:** Segment of the training set after applying one-hot encoding on all categorical features.

Figure 3-left represents figure 2 after applying a Power Transformer. Because there is still a magnitude difference of ~2 amongst numerical features, I applied a normalization, and the results can be shown in figure 3-right. After handling the numerical features, I focused on the categorical features by applying a One-Hot encoder to all categories. Table 3 shows a segment of the categorical features after one-hot encoding within the training dataset. There are now 87 features (columns) after applying one hot encoding.

## Dimensionality Reduction

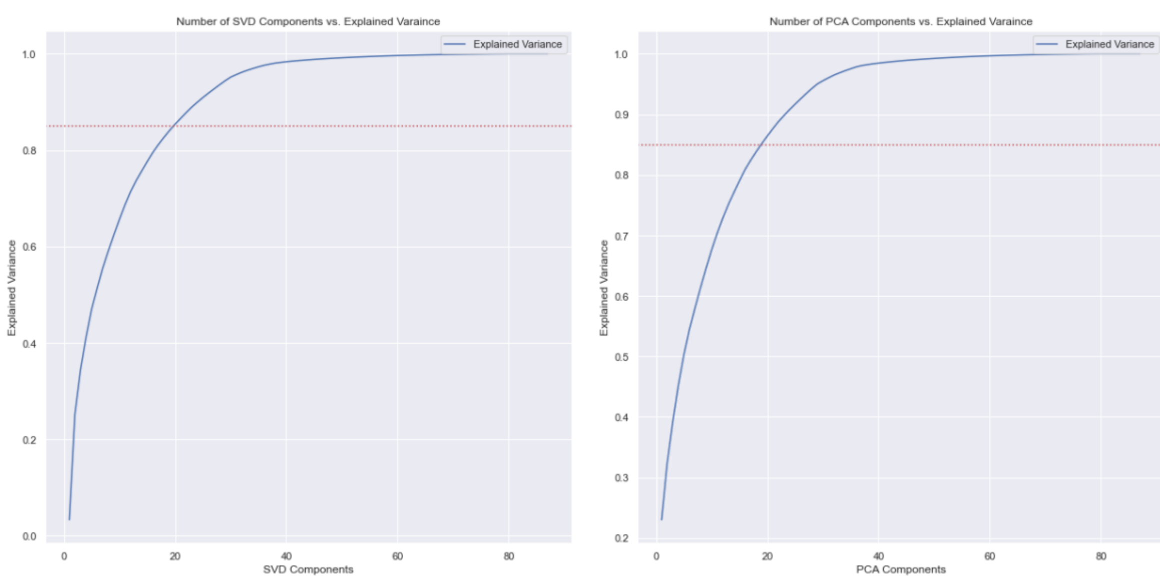



**Figure 4:** (left) Number of SVD components vs. explained variance (right) Number of PCA components vs. explained variance

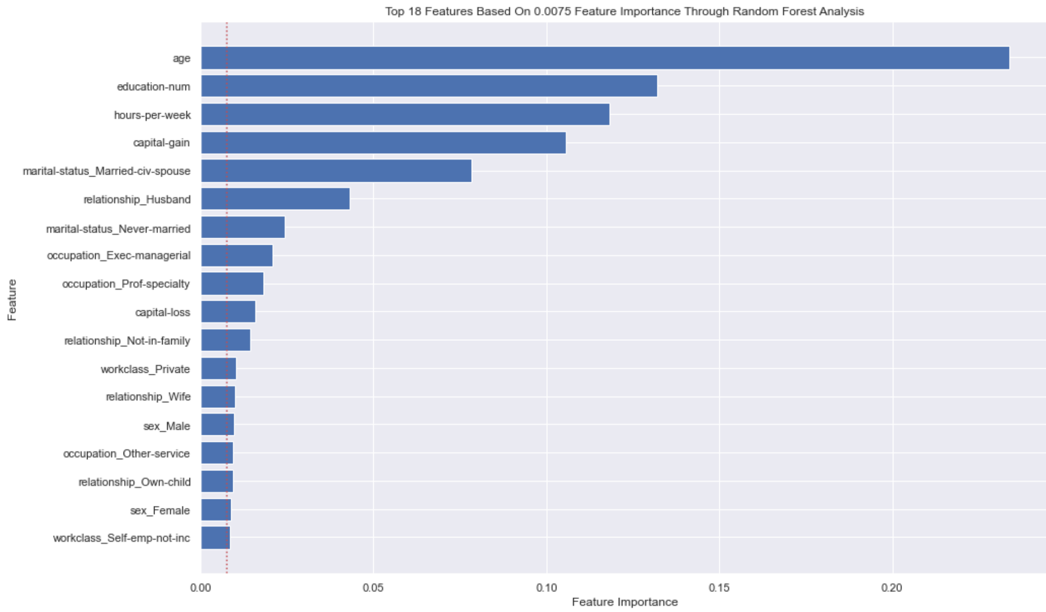



**Figure 5:** Random Forest Classification analysis of the top features that have a feature importance greater than 0.0075 threshold

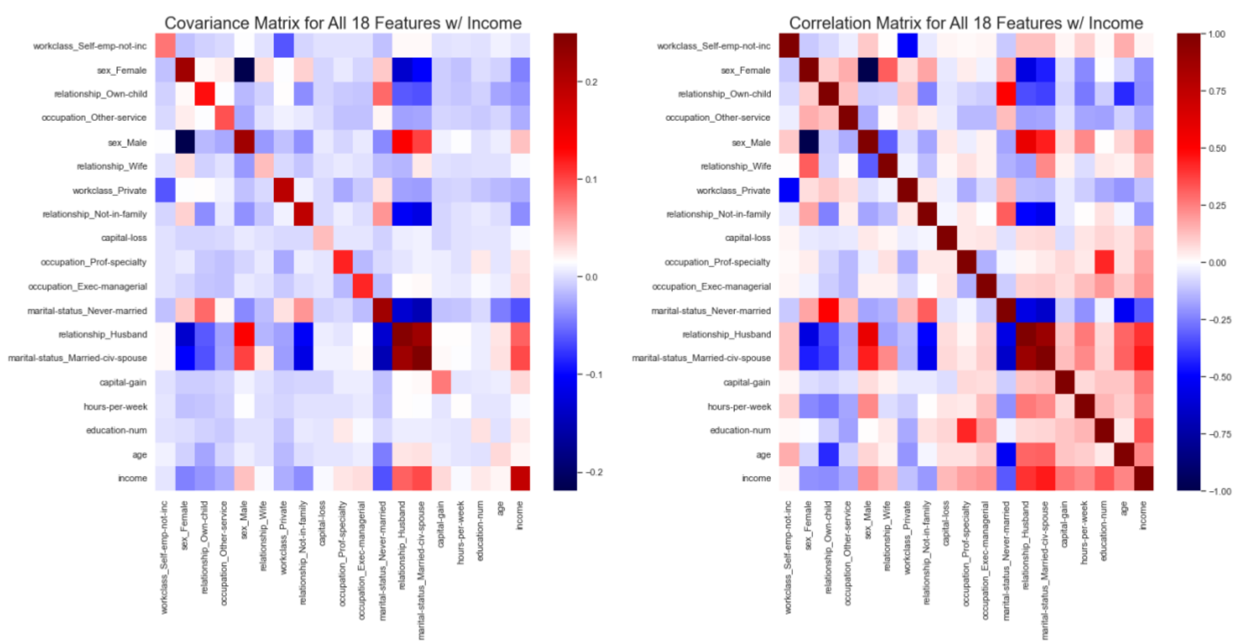



**Figure 6:** (left) Covariance matrix of 18 features and target features (right) Correlation matrix of 18 features and target features

Now that data is prepared and ready for modeling, I wanted to reduce the feature space by first determining the number of features required to represent 85% variance of information using principal component analysis (PCA) and single value decomposition (SVD). Figure 4-left, shows that that I require 20 SVD components to explain 85% variance. Similarly, figure 4-right explains that I required 19 PCA components to explain 85% variance. Although I am not actually using the transformed components for modeling, this analysis helps guide my decision with respect to dimensionality reduction. I next applied a random forest classification (RFA) on the training dataset to assess feature importance. I assessed various feature importance thresholds using the above threshold features within multiple logistic regression models. I assessed thresholds: 0.001, 0.005, 0.0075, 0.01, 0.05 ,0.1. At 0.001, the RFA resulted with 44 components above a 0.001 feature importance threshold and had a logistic regression accuracy score of 0.842. At a feature importance threshold of 0.0075 resulted with 18 components, shown in figure 5, and had an accuracy score of 0.838. Any threshold higher reduced the feature space between 4-12 and had a mean accuracy score of 0.82. Because I was interested in selecting approximately 18-20 features based on SVD and PCA analysis, and there was a significant decrease in accuracy (~0.2), I decided to use a 0.0075 feature importance threshold reducing the feature space to 18 total. Figure 5 shows that age, education number and hours worked per week were the most important features to predict income. Figure 6 shows the covariance and correlation matrix of the selected 18 features in addition to the target feature. There is nothing out of the ordinary within the two matrices. There is a perfect negative correlation between sex_male and sex_female which makes sense since these features were one-hot encoded.

## Phase II - Regression Analysis

### T-test & F-test

Now that the dataset is cleaned, transformed, and reduced, I can now focus on modeling. I focused on a logistic regression model within this phase and attempted to further reduce the feature space after examining f-test, t-test, variance inflation factor (collinearity), and backwards stepwise regression. In table 4 and 5 below, you can see the t-test and f-test scores and their p-value for the selected 18 features. Because these assessments are univariate and don't consider relationships amongst multiple variables, I decided to not use any of this information for my extended feature reduction procedure.

| feature1 | feature2 | t-test | p-value |
|---|---|---|---|
| workclass_Private | capital-loss | 268.048 | 0.0 |
| education-num | age | 239.226 | 0.0 |
| workclass_Private | capital-gain | 239.048 | 0.0 |
| sex_Male | relationship_Wife | 233.196 | 0.0 |
| sex_Male | capital-loss | 232.281 | 0.0 |
| ... | ... | ... | ... |
| relationship_Wife | hours-per-week | -283.039 | 0.0 |
| workclass_Self-emp-not-inc | education-num | -304.916 | 0.0 |

| | | | |
|---|---|---|---|
| capital-gain | education-num | -307.838 | 0.0 |
| capital-loss | education-num | -391.887 | 0.0 |
| relationship_Wife | education-num | -395.607 | 0.0 |

**Table 4:** T-test of 18 features (segment)

| feature1 | feature2 | f-test | p-value |
|---|---|---|---|
| marital-status_Married-civ-spouse | hours-per-week | 16.67 | 0.00 |
| relationship_Husband | hours-per-week | 16.24 | 0.00 |
| sex_Female | hours-per-week | 14.69 | 0.00 |
| sex_Male | hours-per-week | 14.69 | 0.00 |
| marital-status_Never-married | hours-per-week | 14.64 | 0.00 |
| ... | ... | ... | ... |
| sex_Male | workclass_Private | 1.13 | 0.00 |
| sex_Female | workclass_Private | 1.13 | 0.00 |
| relationship_Own-child | occupation_Exec-managerial | 1.09 | 0.00 |
| relationship_Own-child | occupation_Prof-specialty | 1.08 | 0.00 |
| workclass_Self-emp-not-inc | capital-gain | 1.02 | 0.05 |

**Table 5:** F-test of 18 features (segment)

## Further Dimensionality Reduction & Final Logistic Model

| Attribute | vifScores_original | vifScores_updated |
|---|---|---|
| sex_Male | 36.61 | 36.19 |
| marital-status_Married-civ-spouse | 31.94 | NaN |
| relationship_Husband | 31.48 | 3.73 |
| sex_Female | 16.20 | 16.07 |
| relationship_Wife | 6.44 | 1.37 |
| marital-status_Never-married | 2.44 | 2.38 |
| relationship_Not-in-family | 2.37 | 2.36 |
| relationship_Own-child | 2.24 | 2.24 |

| | | |
|---|---|---|
| age | 1.54 | 1.53 |
| workclass_Private | 1.45 | 1.45 |
| education-num | 1.41 | 1.41 |
| workclass_Self-emp-not-inc | 1.38 | 1.38 |
| occupation_Prof-specialty | 1.37 | 1.37 |
| hours-per-week | 1.18 | 1.18 |
| occupation_Exec-managerial | 1.17 | 1.17 |
| occupation_Other-service | 1.11 | 1.11 |
| capital-gain | 1.05 | 1.05 |
| capital-loss | 1.02 | 1.02 |

**Table 6:** Variance Inflation Score of features. The rightmost column shows the updated VIF score after removing married-civ-spouse feature.

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 income   No. Observations:                36177
Model:                          Logit   Df Residuals:                    36159
Method:                           MLE   Df Model:                           17
Date:                Sat, 10 Dec 2022   Pseudo R-squ.:                  0.3854
Time:                        17:53:38   Log-Likelihood:                -12475.
converged:                       True   LL-Null:                       -20298.
Covariance Type:            nonrobust   LLR p-value:                     0.000
=====================================================================================================
                                        coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------
const                                -5.1425   6.31e+05  -8.15e-06      1.000   -1.24e+06    1.24e+06
workclass_Self-emp-not-inc           -0.6535      0.061    -10.778      0.000      -0.772      -0.535
sex_Female                           -2.9356   6.31e+05  -4.65e-06      1.000   -1.24e+06    1.24e+06
relationship_Own-child               -0.6097      0.142     -4.298      0.000      -0.888      -0.332
occupation_Other-service             -0.9778      0.094    -10.415      0.000      -1.162      -0.794
sex_Male                             -2.2069   6.31e+05   -3.5e-06      1.000   -1.24e+06    1.24e+06
relationship_Wife                     1.8693      0.206      9.055      0.000       1.465       2.274
workclass_Private                    -0.0421      0.040     -1.044      0.296      -0.121       0.037
relationship_Not-in-family            0.3952      0.084      4.699      0.000       0.230       0.560
capital-loss                          1.1469      0.065     17.554      0.000       1.019       1.275
occupation_Prof-specialty             0.4408      0.050      8.833      0.000       0.343       0.539
occupation_Exec-managerial            0.7551      0.044     17.141      0.000       0.669       0.841
marital-status_Never-married         -0.5021      0.071     -7.110      0.000      -0.641      -0.364
relationship_Husband                  0.6969      0.199      3.506      0.000       0.307       1.087
marital-status_Married-civ-spouse     1.3129      0.200      6.562      0.000       0.921       1.705
capital-gain                          1.7245      0.052     33.147      0.000       1.623       1.826
hours-per-week                        2.8814      0.142     20.256      0.000       2.603       3.160
education-num                         4.5947      0.121     37.842      0.000       4.357       4.833
age                                   1.9227      0.107     18.044      0.000       1.714       2.132
=====================================================================================================
Accuracy for StatsModel: 0.837
Accuracy for SciKit: 0.838
```

**Table 7:** Summary report of the logistic regression model using the 18 features suggested by the random forest analysis with a 0.0075 threshold. I ran both SciKit's LogisticRegression and StatsModel sm.Logit functions and compared their accuracy score both at 0.837-0.838

I next applied a collinearity assessment using the variance inflation factor (VIF). [4] According to a reference, if the absolute value of VIF is greater than four, these features should be inspected as there might be evidence of high collinearity amongst features. [4] Table 6 shows the VIF scores for the 18 features and identifies collinearity amongst sex_male, marital status married-civ-spouse, relationship-husband, sex_female, relationship_wife. Although it makes sense why these categorical features, sex and relationship (husband and wife), have a high collinearity score, I decided not to remove these features as they are features from one-hot-encoding and uncertain whether it is recommended to remove one-hot encoded features with redundant labels. For example, for a binary category such as sex, if we one-hot encode and examine an entry that has 0 for sex_male then it is implied that sex_female is 1. In this example, this would make one of the categories redundant as the collection of other categories explains the redundant label. Although this is true, I am uncertain whether it is recommended to remove redundant one-hot encoded labels as I would assume this transformation would already handle this scenario implying that there is some significance to keeping redundant labels. For the sake of argument, I decided not to remove any features from the collinearity assessment. However, in table 6, the rightmost column shows the VIF score if I were to remove marital status married-civ-spouse. I performed a backwards stepwise regression with a 0.05 threshold on p-value resulting with the removal of sex_male and workclass_private. Similar reasoning, I decided to not remove both features since they are one-hot encoded items and uncertain with handling. However, as an experiment I did assess a logistic regression with the current 18 features, 17 features (removing marital status married-civ-spous), and 16 features (removing sex_male and workclass_private) and in all scenarios the accuracy score did not change significantly. Since I did not remove any additional features in phase II, I performed a logistic regression analysis using the 18 features selected from phase I and produced an accuracy score of 0.838. Table7 shows the summary report from StatsModel.

## Phase III – Classification Analysis

### Base Model Evaluation

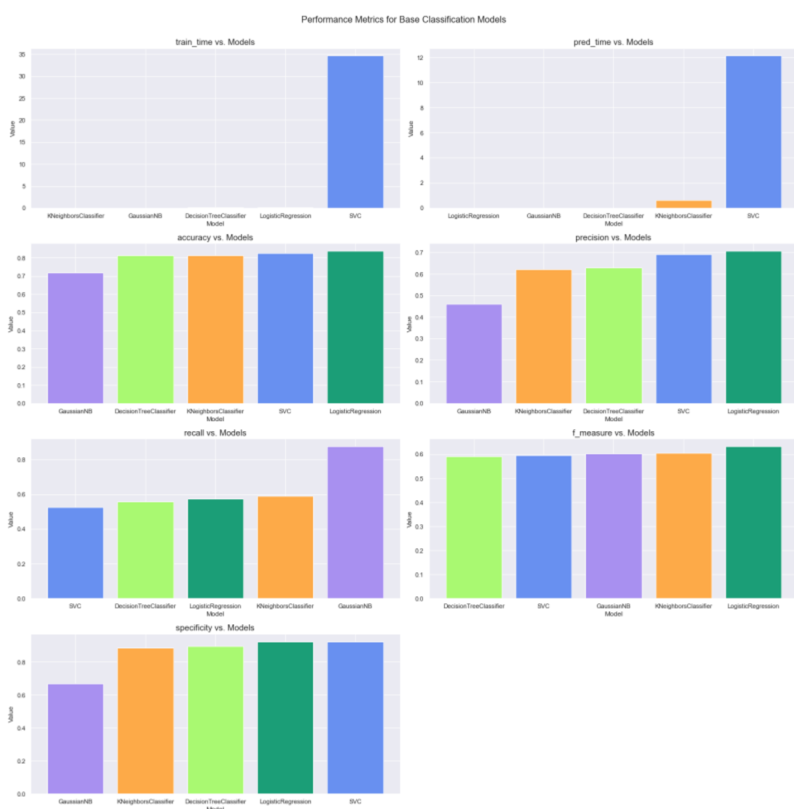


**Figure 7:** Visual representation of base model performance (logistic regression is optimized) for five classification models: Logistic Regression (dark green), Decision tree classifier (lime green), K-Nearest Neighbor Classifier (orange), Support Vector Machine (blue) and a Gaussian Naïve Bayes Model (purple). From left to right, top to down, **(a)** training time, **(b)** prediction time, **(c)** model accuracy, **(d)** model precision, **(e)** model recall, **(f)** model f-measure and **(g)** model specificity

| Model | precision | recall | pred_time | train_time | specificity | accuracy | f_measure |
|---|---|---|---|---|---|---|---|
| **LogisticRegression** | 0.706 | 0.573 | 0.001 | 0.155 | 0.923 | 0.838 | 0.633 |
| **DecisionTreeClassifier** | 0.630 | 0.558 | 0.004 | 0.078 | 0.894 | 0.812 | 0.592 |
| **KNeighborsClassifier** | 0.622 | 0.588 | 0.623 | 0.005 | 0.885 | 0.812 | 0.604 |
| **SVC** | 0.690 | 0.525 | 12.128 | 34.630 | 0.924 | 0.827 | 0.596 |
| **GaussianNB** | 0.460 | 0.875 | 0.003 | 0.011 | 0.669 | 0.719 | 0.603 |

**Table 8:** Tabular representation of figure 7 results.

Outside of the logistic regression from phase II, this section focuses on training and evaluating four additional models using various metrics. Specifically, training and evaluating base model performance for decision tree classifier (default parameters), support vector machine (default parameters), K-nearest neighbors (n_neighbor=3) and a Gaussian Naïve Bayes model (default parameters). Figure 7 and table 8 show various performance metrics for each model. The support vector machine had a magnitude higher training and prediction time. It appears that the logistic regression model performed the best in terms of accuracy, precision and f-measure and ~specificity (0.001 less than SVC). The recall was the highest for the Bayes model but examining figure 8 shows significantly higher false positive value relative to all four models. In figure 9, the logistic regression appears to have its graph closest towards the top left corner implying that it is the better performer amongst the other models. Similarly, it's AOC value is much higher in comparison to the other models at 0.89. As a note, the decision tree classifier and the KNN model might display inaccurate ROC curves in both figure 9 and figure 12 in the next section.

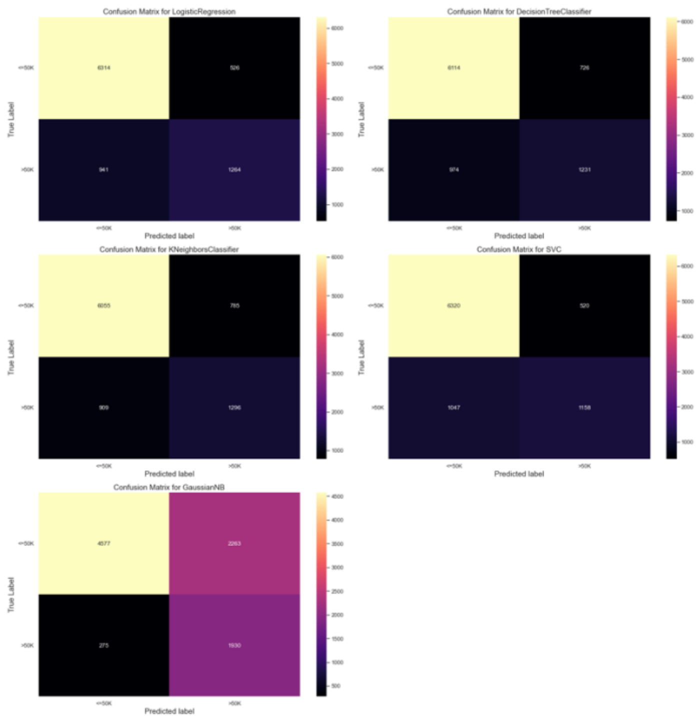



**Figure 8:** Confusion Matrix of the four base models and optimized logistic regression

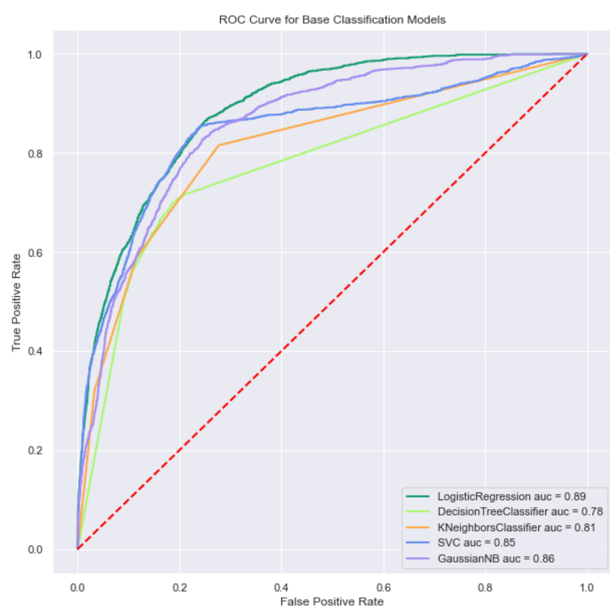



**Figure 9:** ROC curve and AOC value for four base models and optimized logistic regression

## Optimized Model Evaluation


**Figure 10:** Visual representation of optimized model performance for five classification models: Logistic Regression (dark green), Decision tree classifier (purple), K-Nearest Neighbor Classifier (red), Support Vector Machine (light blue) and a Gaussian Naïve Bayes Model (purple). From left to right, top to down, **(a)** training time, **(b)** prediction time, **(c)** model accuracy, **(d)** model precision, **(e)** model recall, **(f)** model f-measure and **(g)** model specificity

| Optimized Model | train_time | pred_time | accuracy | precision | recall | f_measure | specificity |
|---|---|---|---|---|---|---|---|
| **clf_DTree_Grid** | 158.751 | 0.004 | 0.851 | 0.760 | 0.570 | 0.651 | 0.942 |
| **clf_DTree_Base** | 0.101 | 0.003 | 0.812 | 0.629 | 0.558 | 0.592 | 0.894 |
| **clf_KNN_Grid** | 61.631 | 0.655 | 0.834 | 0.681 | 0.596 | 0.636 | 0.910 |
| **clf_KNN_Base** | 0.005 | 0.559 | 0.813 | 0.623 | 0.588 | 0.605 | 0.885 |
| **clf_SVMLinear_Grid** | 476.403 | 2.641 | 0.836 | 0.708 | 0.557 | 0.624 | 0.926 |

| Optimized Model | train_time | pred_time | accuracy | precision | recall | f_measure | specificity |
|---|---|---|---|---|---|---|---|
| **clf_SVM_Base** | 30.506 | 12.146 | 0.827 | 0.690 | 0.525 | 0.596 | 0.924 |
| **clf_GaussianNB_Base** | 0.014 | 0.003 | 0.719 | 0.460 | 0.875 | 0.603 | 0.669 |
| **clf_Log_Base** | 0.203 | 0.005 | 0.838 | 0.706 | 0.573 | 0.633 | 0.923 |

**Table 9:** Tabular representation of figure 10 results and the base model performance from the last section for reference

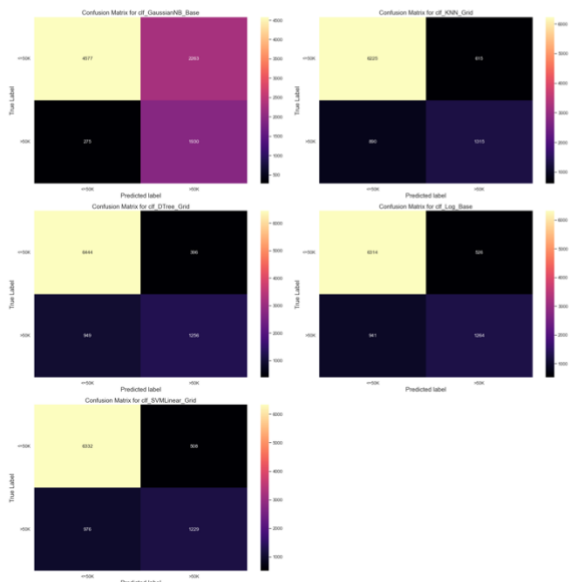


**Figure 11:** Confusion Matrix of the five optimized models

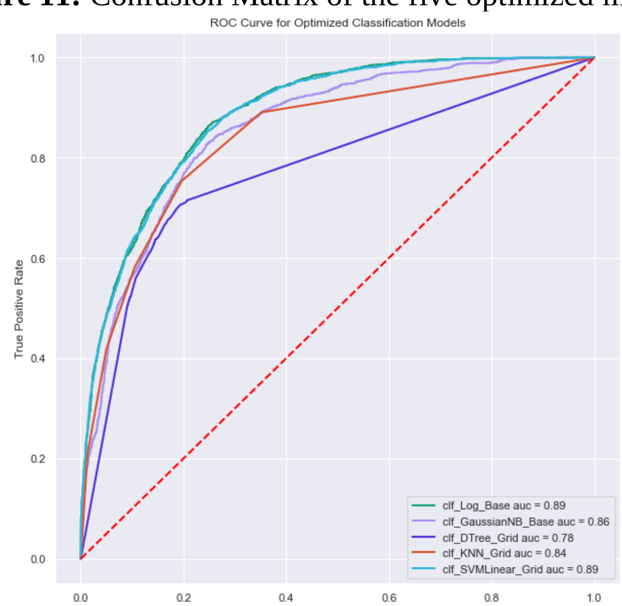



**Figure 12:** ROC curve and AOC value for five optimized models

After conducting an initial model evaluation for the four separate models (excluding logistic regression since it was optimized in phase II), I performed a hyper tuning using GridSearch with cross validation. The Bayes model did not have any additional parameters to tune and performed the worst in the base model evaluation so focused the hyper tuning assessment on the decision tree classifier (criterion: gini, entropy | max_depth: range(1,15) | min_samples_split: range(2,10) | min_samples_leaf: range(1,5)), KNN (n_neighbors: range(1,50,2)) and the support vector machine (kernel: poly, linear, rbf, sigmoid). Because the support vector machine required a significant amount of time to train in the base model evaluation, the parameter selection was only limited to the kernel initially. Although a greedy approach, once the optimal kernel was selected the c_parameter was assessed between 0.1, 1, 10, 100. After GridSearch the optimal parameters for each model were:

- DecisionTree : {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2}
- KNN : {'n_neighbors': 33}
- Support Vector Machine : {'kernel': 'linear’, 'C': 1}

Figure 10 and table 9 showcase the same metrics that the base models were evaluated on for the optimized classifiers, logistic regression and bayes model. The support vector machine required the most time for training and prediction by a significant magnitude. Decision Tree performed the best in terms of accuracy, precision, f-measure and specificity. Figure 12 shows the ROC for each model with the logistic regression showing as the best performer. However, as discussed in the prior section, I believe there was a slight error when constructing the ROC curve for the decision tree classifier which shows as the underperforming model based on figure 12.

## Recommendations

### Overall model selection and future work

Overall, after optimizing models, I would conclude that the best performing model is the decision tree classifier with the following parameters (criterion: entropy, max_depth: 10, min_samples_leaf: 4, min_samples_split: 2). I would conclude that if a random forest classifier was used within this assessment, it would most likely outperform the decision classifier since it is an ensemble model that uses a decision classifier. As mentioned in prior sections, I would aim to understand handling and assessing one-hot encoded features that have high variation inflation factors and fall above the significance value within a backwards stepwise regression.

### Contribution

I completed 100% of this project on my own without any outside help.

### Learnings

- Learned a lot within the data transformation phase such as you should always split your data before applying transformations, skewness assessment & PowerTransformers
- The difference between PCA, SVD for feature selection vs. using the output components for modeling

- Random Forest Analysis vs. Model
- Variance Inflation Factor
- Modeling KNN, SVM and Naïve Bayes
- Assessing model based on various metrics
- Visualization
- Syntax

## Appendix (softcopy of code)

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from time import time
sns.set(style="darkgrid")

## Transformations
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder

## Feature Reduction Analyis
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import TruncatedSVD

## Analaysis
from sklearn.metrics import accuracy_score
from scipy.stats import ttest_ind
import scipy.stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_score
from sklearn.metrics import auc

## Models
from sklearn.linear_model import LogisticRegression
from sklearn import svm
```

```python
import statsmodels.api as sm
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

## State for all randomness
randomState = 17

##############################################################
# Data Cleaning
##############################################################
## Smaller data set with ~10 features

colNames = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation',
        'relationship', 'race','sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income']

dfTrain = pd.read_csv(os.path.join(os.getcwd(),"subset","adult.data"), index_col=False, names = colNames)
dfTest = pd.read_csv(os.path.join(os.getcwd(),"subset","adult.test"), index_col=False, names = colNames)
df = pd.concat([dfTrain,dfTest.iloc[1::]])

## Don't require fnlwgt for analysis
df.drop(columns=["fnlwgt"], inplace = True)

# Read in age as int type
df['age'] = df.age.astype(int)

## Map target dataset to binary
df.income = df.income.map({' <=50K': 0, ' <=50K.': 0, ' >50K':1, ' >50K.':1})

# Identify null
print(f"Results of Na in the dataset:\n{df.isna().sum()}")

# Identify any filler entries that act as null
print("--------------------------------------")
print("Identify any NaN fillers, '?', and convert their entry to np.nan.\n\
Also, want to determine the percentage of data fillers make up to determine\n\
proceeding steps.\n")
```

```
for i in df.columns:
    res = len(df[df[i] == ' ?'])
    if res != 0:
        print(f"{i}: {res} records with '?' as entry")
        print(f"This makes up {100*res/len(df):0.2f}% of the dataset\n")
        df.loc[df[i]==' ?', i] = np.nan
missData = 100*len(df[df.workclass.isna() | df.occupation.isna() |  df["native-country"].isna()])/len(df)
print(f"If I were to handle missing data by removing those entries all together that would make up {missData:0.2f}% of the data removed")
print(f"Because I believe I can still obtain a reliable model while still removing {missData:0.2f}% of data,\n\
I will proceed with removing those entries.")
df.dropna(inplace = True)

# Strip data on left side since there seems to be an additional space between entry
catData = set(df.columns) - set(df.describe().columns)
numData= list(set(df.columns)-catData - {"income"})
for i in catData:
    df[i] = df[i].str.lstrip()


##########################################################################
# Visual Exploratory Analysis
##########################################################################

def getGroupPlot(df, columns=["workclass","income"], value = "income"):
    data = df.groupby(columns)[value].count().unstack()
    data = data.div(data.apply(sum,1),0)
    data.sort_values(by = 1, inplace = True)
    data.plot(kind="bar",
              figsize=(15, 10),
              title= (columns[0] + " vs. " + columns[1]).title(),
              ylabel = 'Percentage (%)')
    plt.legend(["<=50K", ">50K"])
    plt.tight_layout()
    plt.show()
    return data
```

```
# Bin hours per week into four categories for visualization purposes only
df.loc[df['hours-per-week'].between(0, 40, 'left'), 'hoursPerWeek'] = '<40'
df.loc[df['hours-per-week'].between(40, 60, 'left'), 'hoursPerWeek'] = '<60'
df.loc[df['hours-per-week'].between(60, 80, 'left'), 'hoursPerWeek'] = '<80'
df.loc[df['hours-per-week'].between(80, 100, 'both'), 'hoursPerWeek'] = '>=80'
catData = catData|{'hoursPerWeek'}

groupbyAll = {}
for i in catData:
    groupbyAll[i] = (getGroupPlot(df, columns=[i,"income"],value="income"))

# Drop hoursPerWeek & education for the remainder of the modeling since education-num gives the same
information
# as education and hoursPerWeek were only used for visualization
df.drop(columns=["hoursPerWeek", "education"], inplace = True)
catData = catData - {'hoursPerWeek', 'education'}

#########################################################
# Data Transformation
#########################################################

# Split data prior to transformations
X_train, X_test, y_train, y_test = train_test_split(df[list(set(df.columns)-{"income"})], df['income'], test_size=0.2,
random_state=randomState)

X_train[numData].plot(kind = 'box',
          figsize = (15,10),
          title="Numerical Data Before Transformation",
          xlabel = "Features",
          ylabel="Numerical Value")


# Skew value <-1 or >1 is highly skewed
X_train[numData].skew().sort_values(ascending=False)

## Visualize Skewness
fig = plt.figure(figsize = (15,20));
```

```
for i, feature in enumerate(X_train[numData]):
    ax = fig.add_subplot(3, 2, i+1)
    ax.hist(X_train[feature], bins = 50)
    ax.set_title(f"{feature.title()} Feature Distribution")
    ax.set_xlabel("Feature Value")
    ax.set_ylabel("Feature Count")
    ax.set_ylim((0, 3000))
    ax.set_yticks([0, 1000, 2000, 3000])
    ax.set_yticklabels([0, 1000, 2000, ">3000"])
fig.suptitle("Distributions of Numerical Features")
fig.tight_layout()
plt.show()

# Adjust for skewness on capital gain and capital loss
powerSkew = PowerTransformer()
fig = plt.figure(figsize=(15,10))

j = 1
for i in ["capital-gain","capital-loss"]:
    ax = fig.add_subplot(2, 2, j)
    ax.hist(X_train[i], bins = 50)
    ax.set_title(f"Orginal Distribution for {i}")
    ax.set_ylim((0, 3000))
    ax.set_yticks([0, 1000, 2000, 3000])
    ax.set_yticklabels([0, 1000, 2000, ">3000"])
    j+=2

powerSkew = PowerTransformer()
X_train[['capital-gain','capital-loss']]=powerSkew.fit_transform(X_train[['capital-gain','capital-loss']])
X_test[['capital-gain','capital-loss']] = powerSkew.transform(X_test[['capital-gain','capital-loss']])

j=2
for i in ["capital-gain","capital-loss"]:
    ax = fig.add_subplot(2, 2, j)
    ax.hist(X_train[i], bins = 50)
    ax.set_title(f"Power Transform for {i}")
    j += 2
```

```
plt.show()
X_train[numData].skew().sort_values(ascending=False)

X_train[numData].plot(kind = 'box',
          figsize = (15,10),
          title="Numerical Data After Skew Transformation",
          xlabel = "Features",
          ylabel="Numerical Value")

## Because I know the capital loss and capital-gain do not follw a normal distribution still, I am
## choosing to normalize all the numerical data rather than standardize
normalize = MinMaxScaler()
X_train[numData] = normalize.fit_transform(X_train[numData])
X_test[numData] = normalize.transform(X_test[numData])
X_train.head(5)

# Used to identify if normalization is required
X_train[numData].plot(kind = 'box',
            figsize = (15,10),
            title="Numerical Data After Normalization",
            xlabel = "Features",
            ylabel="Numerical Value")

encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
X_trainCat = pd.DataFrame(encoder.fit_transform(X_train[catData]))
X_testCat = pd.DataFrame(encoder.transform(X_test[catData]))

# Grab index location of categorical data to pass get_feature_names
catIdx = [X_train.columns.get_loc(col) for col in catData]

# Adding column names to the encoded data set
X_trainCat.columns = encoder.get_feature_names(X_train.columns.values[catIdx].tolist())
X_testCat.columns = encoder.get_feature_names(X_train.columns.values[catIdx].tolist())

# One-hot encoding removed index; put it back
X_trainCat.index = X_train.index
```

```
X_testCat.index = X_test.index

# Remove categorical columns (will replace with one-hot encoding)
X_trainNum = X_train.drop(catData, axis=1)
X_testNum = X_test.drop(catData, axis=1)

# Add one-hot encoded columns to numerical features
X_train2 = pd.concat([X_trainNum, X_trainCat], axis=1)
X_test2 = pd.concat([X_testNum, X_testCat], axis=1)

###########################################################
# Dimensionality Reduction
###########################################################

## SVD
svdThresh = 0.85
svdVar = []
for i in range(1,len(X_train2.columns)+1):
    svdVar.append({"n_components":i, "explainedVar":sum(TruncatedSVD(n_components =
i).fit(X_train2).explained_variance_ratio_)})
svdVar = pd.DataFrame(svdVar)

plt.figure(figsize=(10,10))
plt.plot(svdVar['n_components'], svdVar['explainedVar'], label = 'Explained Variance')
plt.axhline(y = svdThresh, color = 'r', linestyle = ':')
plt.legend()
plt.xlabel("SVD Components")
plt.ylabel("Explained Variance")
plt.title("Number of SVD Components vs. Explained Varaince")
plt.show()

print(f"SVD suggest that there should be at least
{svdVar[svdVar['explainedVar']>=svdThresh].iloc[0]['n_components']:0.0f} of {len(svdVar)} SVD components\n\
to provide {svdThresh*100}% variance")

## PCA
pcaThresh = 0.85
```

```
pcaVar = []
for i in range(1,len(X_train2.columns)+1):
    pcaVar.append({"n_components":i, "explainedVar":sum(PCA(n_components =
i).fit(X_train2).explained_variance_ratio_)})
pcaVar = pd.DataFrame(pcaVar)

plt.figure(figsize=(10,10))
plt.plot(pcaVar['n_components'], pcaVar['explainedVar'], label = 'Explained Variance')
plt.axhline(y = pcaThresh, color = 'r', linestyle = ':')
plt.legend()
plt.xlabel("PCA Components")
plt.ylabel("Explained Variance")
plt.title("Number of PCA Components vs. Explained Varaince")
plt.show()

print(f"PCA suggest that there should be at least
{pcaVar[pcaVar['explainedVar']>=pcaThresh].iloc[0]['n_components']:0.0f} of {len(pcaVar)} PCA components\n\
to provide {pcaThresh*100}% variance")

## Random Forest Analysis (RFA)
thresh = [0.001,0.005,0.0075, 0.01,0.05,0.1]
threshFeatures = {}
rfAnalysis = RandomForestClassifier(random_state=randomState).fit(X_train2, y_train)
X_train2_FeatureImportance = pd.DataFrame(rfAnalysis.feature_importances_,
X_train2.columns).round(4).sort_values(by=[0], ascending=False)
for i in thresh:
    print("==========================")
    threshCol = list(X_train2_FeatureImportance[X_train2_FeatureImportance[0]>i].index)
    threshFeatures[str(i)] = threshCol
    logThresh = LogisticRegression(random_state=randomState)
    logThresh = logThresh.fit(X_train2[threshCol], y_train)
    print(f"\nNumber of features selected: {len(threshCol)}")
    print(f"\nFeatures include {threshCol}")
    print(f"\nRandom Forest Top {i} Threshold, Accuracy for Logistic Regression: {accuracy_score(y_test,
logThresh.predict(X_test2[threshCol])):0.3f}")

print("Because PCA, SVD suggest 18-20 features of 87 to cover 85% variance and the Random Forest Analysis\n\
```

```
suggest 14-28 features with feature importance threshold of 0.005-0.01, I will select threshold 0.0075.\n\
At 0.0075, this meets the PCA/SVD variance coverage and minimizes the feature space significantly without\n\
diminishing the accuracy of a logistic regression. Although 5-14 features still produces a great accuracy\n\
according to PCA/SVD, reducing the feature space this much would reduce variance coverage.")

X_train2_FeatureImportance =
X_train2_FeatureImportance[X_train2_FeatureImportance[0]>0.0075].sort_values(by=[0], ascending=True)
plt.figure(figsize=(15,10))
plt.barh(range(len(X_train2_FeatureImportance)), X_train2_FeatureImportance[0])
plt.yticks(range(len(X_train2_FeatureImportance)), X_train2_FeatureImportance.index)
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.axvline(x = 0.0075, color = 'r', linestyle = ':')
plt.title(f"Top {len(X_train2_FeatureImportance)} Features Based On 0.0075 Feature Importance Through Random
Forest Analysis")
plt.show()

X_train3 = X_train2[list(X_train2_FeatureImportance.index)]
X_test3 = X_test2[list(X_train2_FeatureImportance.index)]

###########################################################
# Covariance and Correlation Matrix
###########################################################

## Cov
plt.figure(figsize=(10,10))
sns.heatmap(pd.concat([X_train3, y_train], axis=1).cov(), annot=False, cmap="seismic")
plt.title(f"Covariance Matrix for All {len(X_train3.columns)} Features w/ Income",  fontsize=20)
plt.show()

## Corr
plt.figure(figsize=(10,10))
sns.heatmap(pd.concat([X_train3, y_train], axis=1).corr(method='pearson'), annot=False, cmap="seismic")
plt.title(f"Correlation Matrix for All {len(X_train3.columns)} Features w/ Income",  fontsize=20)
plt.show()
```

```
############################################################
# T-test and F-test
############################################################
tTest = []
fTest = []
alpha = 0.05

for i in range(len(X_train3.columns)):
    for k in range(i+1, len(X_train3.columns)):
        iCol = X_train3.columns[i]
        kCol = X_train3.columns[k]

        ## f test
        f = np.var(X_train3[iCol], ddof=1)/np.var(X_train3[kCol], ddof=1)
        f_pvalue = 1-scipy.stats.f.cdf(f, X_train3[iCol].size-1, X_train3[kCol].size-1)
        if f_pvalue <= alpha:
            fTest.append({"feature1":iCol, "feature2":kCol, "f-test":f, "p-value":f_pvalue})

        ## t test
        tResults = ttest_ind(X_train3[iCol], X_train3[kCol])
        if tResults.pvalue <= alpha:
            tTest.append({"feature1":iCol, "feature2":kCol, "t-test":tResults.statistic, "p-value":tResults.pvalue})
tTest = pd.DataFrame(tTest).round(3)
fTest = pd.DataFrame(fTest).round(2)
tTest.sort_values(by=["t-test"],ascending=False)
fTest.sort_values(by=["f-test"],ascending=False)

############################################################
# Collinearity Assessment w/ Variance Inflation Factor
############################################################

vif_scores = pd.DataFrame()
vif_scores["Attribute"] = X_train3.columns
vif_scores["vifScores"] = [variance_inflation_factor(X_train3.values, i) for i in range(len(X_train3.columns))]

print("It is known that colinearity will exist with binary one hot encoded features such as male and female.\n\
However upon reseearch, it suggest this analysis can be ignored for said features and instead ensure\n\
```

```
when modeling a regression to exclude the intercept. Similar information can be made for husband/wife.\n\
It might make sense to remove Married Civilian Spouse since Married-spouse-absent and Married-AF-spouse\n\
have been removed earlier and can infer this info from relationship (husband/wife). However for the\n\
remainder of the modeling, will leave feature space as is.")

vif_scores2 = pd.DataFrame()
tempCol = set(X_train3.columns) - {"marital-status_Married-civ-spouse"}
vif_scores2["Attribute"] = list(tempCol)
vif_scores2["vifScores"] = [variance_inflation_factor(X_train3[list(tempCol)].values, i) for i in range(len(tempCol))]
vif_scores.merge(vif_scores2, on="Attribute", how="left",
suffixes=("_original","_updated")).round(2).sort_values(by=['vifScores_original'], ascending =False)

#############################################################
# Backward Stepwise Regression w/ 0.05 threshold
#############################################################

def backwardStepwise(X_train, y_train, X_test, y_test, thresh = 0.05):
    features=list(X_train.columns)
    while True:
        changed=False
        model = sm.Logit(y_train, sm.add_constant(X_train[features])).fit()
        pvalues = model.pvalues.iloc[1:]
        if pvalues.max() > thresh:
            changed=True
            features.remove(pvalues.idxmax())
            print(f'Feature "{pvalues.idxmax()}" dropped (p-value {pvalues.max():.2f} > thresh {thresh})')
        if not changed:
            break
    print(f"\nFeatures kept by backwards stepwise regression are: {', '.join(features)}")
    return features, accuracy_score(y_test, model.predict(sm.add_constant(X_test[features])).round())


features, accuracy = backwardStepwise(X_train3, y_train, X_test3, y_test, thresh=0.05)
print(f"\nAccuracy: {accuracy:0.3f}\n")
print("\nBecause there are no real accuraccy gains after stepwise regression, choosing to leave feature space as is.")

#############################################################
```

```
# Final Logistic Regression using RFA features
############################################################
log1Stats = sm.Logit(y_train, sm.add_constant(X_train3)).fit()
print(log1Stats.summary())

log1SciKit = LogisticRegression(random_state=randomState, fit_intercept=False)
log1SciKit = log1SciKit.fit(X_train3, y_train)

print(f"Accuracy for StatsModel: {accuracy_score(y_test, log1Stats.predict(sm.add_constant(X_test3)).round()):0.3f}")
print(f"Accuracy for SciKit: {accuracy_score(y_test, log1SciKit.predict(X_test3)):0.3f}")

############################################################
# Base Model Evaluation
############################################################

def classifierEval(clf, X_train, y_train, X_test, y_test):
    results = {}

    start = time()
    clf = clf.fit(X_train, y_train)
    results['train_time'] = time() - start

    start = time()
    pred = clf.predict(X_test)
    results['pred_time'] = time() - start

    confusion = confusion_matrix(y_test,pred).ravel()
    tn, fp, fn, tp = confusion

    results['confusion'] = confusion
    results['accuracy'] =  accuracy_score(y_test,pred)
    results['precision'] = precision_score(y_test, pred)
    results['recall'] = recall_score(y_test,pred)
    results['f_measure'] = f1_score(y_test,pred)
    results['specificity'] = tn/(tn+fp)

    return clf, results
```

```
clf_Log = log1SciKit
clf_DTree = tree.DecisionTreeClassifier(random_state = randomState)
clf_KNN = KNeighborsClassifier(n_neighbors=3)
clf_SVM = svm.SVC(random_state = randomState)
clf_Bayes = GaussianNB()

# Collect results on the learners
results = {}
for clf in [clf_Log, clf_DTree, clf_KNN, clf_SVM, clf_Bayes]:
    print(clf.__class__.__name__)
    clf, results[clf.__class__.__name__] = classifierEval(clf, X_train3, y_train, X_test3, y_test)

## Base Model Metrics
colors = ['#1b9e77', '#a9f971', '#fdaa48','#6890F0','#A890F0']
baseResults = pd.DataFrame(results)
baseResults = baseResults.append(pd.DataFrame([colors], index=["colors"], columns=baseResults.columns))

fig = plt.figure(figsize=(20,20))
for j, label in enumerate(["train_time","pred_time","accuracy","precision","recall","f_measure","specificity"]):
    ax = fig.add_subplot(4, 2, j+1)
    temp = baseResults.sort_values(by=[label], axis=1)
    ax.bar(temp.loc[label].index, temp.loc[label].values, color = list(temp.loc["colors"]))
    ax.set_title(f"{label} vs. Models", fontsize=15)
    ax.set_xlabel("Model")
    ax.set_ylabel("Value")
plt.suptitle("Performance Metrics for Base Classification Models", fontsize = 16, y = 1)
plt.tight_layout()
plt.show()

baseResults.loc[list(set(baseResults.index) - {"colors","confusion"})].T.round(2)

## ROC and AUC for Base Model
fig = plt.figure(figsize=(10,10))
for i, clf in enumerate([clf_Log, clf_DTree, clf_KNN, clf_SVM, clf_Bayes]):
        if clf.__class__.__name__ == "SVC":
            fpr, tpr, _ = roc_curve(y_test, clf.decision_function(X_test3))
```

```
        else:
            fpr, tpr, _ = roc_curve(y_test, clf.predict_proba(X_test3)[:,1])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, color=colors[i],lw=2, label=f'{clf.__class__.__name__} auc = {roc_auc:.2f}')

plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Base Classification Models')
plt.legend()
plt.show()

## Confusion Matrix
fig = plt.figure(figsize=(20,20))
j=0
for index, value in baseResults.loc['confusion'].iteritems():
    ax = fig.add_subplot(3, 2, j+1)
    sns.heatmap(value.reshape(2, -1), annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_xlabel("Predicted label", fontsize =15)
    ax.set_xticklabels(["<=50K",">50K"])
    ax.set_ylabel("True Label", fontsize=15)
    ax.set_yticklabels(["<=50K",">50K"], rotation = 0)
    ax.set_title(f"Confusion Matrix for {index}", fontsize=15)
    j+=1

plt.tight_layout()
plt.show()

##################################################################
# Optimized Model Performance with GridSearchCV
# GridSearchCV uses default scoring
##################################################################
def classifierGrid(clf, param_dict, X_train, y_train, X_test, y_test):
    gridCLF =  GridSearchCV(clf,
                    param_grid=param_dict,
                    cv=5)
    gridCLF, results= classifierEval(gridCLF, X_train, y_train, X_test, y_test)
```

```
    results["best_params"] = gridCLF.best_params_
    results["score_mean"] = gridCLF.cv_results_['mean_test_score'].mean()
    results["score_best"] = gridCLF.best_score_
    return gridCLF, results


clf_DTree_Grid = tree.DecisionTreeClassifier(random_state = randomState)
clf_KNN_Grid = KNeighborsClassifier()
clf_SVMKernel_Grid = svm.SVC(random_state = randomState)

param_clf_DTree_Grid = {"criterion":["gini", "entropy"],
             "max_depth":range(1,15),
             "min_samples_split":range(2,10),
             "min_samples_leaf":range(1,5)
            }
param_clf_KNN_Grid = {"n_neighbors":range(1,50,2)}

param_clf_SVMKernel = {"kernel":["poly","linear","rbf","sigmoid"]}

## Running Grid for SVM - Kernel only and will optimize specific kernel next
    ## Note, I understand that this is a greedy approach but saves time for training/testing since SVM takes a while
clfKey = ['clf_DTree_Grid','clf_KNN_Grid','clf_SVMKernel_Grid']
clfs = dict(zip(clfKey,[clf_DTree_Grid,clf_KNN_Grid,clf_SVMKernel_Grid]))
params = dict(zip(clfKey,[param_clf_DTree_Grid,param_clf_KNN_Grid,param_clf_SVMKernel]))

resultsOpt = {}
for key, clf in clfs.items():
    print("=================")
    print(clf.__class__.__name__)
    print(key)
    start = time()
    clf, resultsOpt[key] = classifierGrid(clf,params[key], X_train3, y_train, X_test3, y_test)
    print(time()-start)

######## Best kernel from prior
clf_SVMLinear_Grid = svm.SVC(kernel="linear", random_state = randomState)
param_clf_SVMLinear_Grid = {'C': [0.1, 1, 10, 100]}
```

```
start = time()
clf_SVMLinear_Grid, resultsOpt["clf_SVMLinear_Grid"] =
classifierGrid(clf_SVMLinear_Grid,param_clf_SVMLinear_Grid, X_train3, y_train, X_test3, y_test)
print(time()-start)

for key, value in resultsOpt.items():
    print(f"Best parameters for {key} are: {resultsOpt[key]['best_params']}")

resultsComb = pd.DataFrame(resultsOpt).join(pd.DataFrame(results)).rename(columns=
                {"LogisticRegression":"clf_Log_Base",
                  "DecisionTreeClassifier":"clf_DTree_Base",
                  "KNeighborsClassifier":"clf_KNN_Base",
                  "SVC":"clf_SVM_Base",
                  "GaussianNB":"clf_GaussianNB_Base"
                }).drop(columns=["clf_SVMKernel_Grid"])
resultsComb = resultsComb.append(pd.DataFrame([['#4f2bdb','#db4f2b','#2bb7db', '#1b9e77', '#a9f971',
'#fdaa48','#6890F0','#A890F0']], index=["colors"], columns=resultsComb.columns))

resultsComb.loc[["train_time","pred_time","accuracy","precision","recall","f_measure","specificity"]].T

## All Model (Base and Optimized Metrics)
fig = plt.figure(figsize=(20,20))
for j, label in enumerate(["train_time","pred_time","accuracy","precision","recall","f_measure","specificity"]):
    ax = fig.add_subplot(5, 2, j+1)
    temp = resultsComb.sort_values(by=[label], axis=1)
    ax.bar(temp.loc[label].index, temp.loc[label].values, color = list(temp.loc["colors"]))
    ax.set_title(f"{label} vs. Models", fontsize=15)
    ax.set_xlabel("Model")
    ax.set_ylabel("Value")
plt.suptitle("Performance Metrics for All Classification Models", fontsize = 16, y = 1)
plt.tight_layout()
plt.show()

## Only optimized model metrics
fig = plt.figure(figsize=(20,20))
cols = list(set(resultsComb.columns)-{"clf_DTree_Base","clf_KNN_Base","clf_SVM_Base"})
```

```
for j, label in enumerate(["train_time","pred_time","accuracy","precision","recall","f_measure","specificity"]):
    ax = fig.add_subplot(5, 2, j+1)
    temp = resultsComb[cols].sort_values(by=[label], axis=1)
    ax.bar(temp.loc[label].index, temp.loc[label].values, color = list(temp.loc["colors"]))
    ax.set_title(f"{label} vs. Models", fontsize=15)
    ax.set_xlabel("Model")
    ax.set_ylabel("Value")
plt.suptitle("Performance Metrics for Optimized Classification Models", fontsize = 16, y = 1)
plt.tight_layout()
plt.show()

## Only optimized model confusion matrix
fig = plt.figure(figsize=(20,20))
j=0
for index, value in resultsComb[cols].loc['confusion'].iteritems():
    ax = fig.add_subplot(3, 2, j+1)
    sns.heatmap(value.reshape(2, -1), annot=True, fmt="g", ax=ax, cmap="magma")
    ax.set_xlabel("Predicted label", fontsize =15)
    ax.set_xticklabels(["<=50K",">50K"])
    ax.set_ylabel("True Label", fontsize=15)
    ax.set_yticklabels(["<=50K",">50K"], rotation = 0)
    ax.set_title(f"Confusion Matrix for {index}", fontsize=15)
    j+=1
plt.tight_layout()
plt.show()

## Only optimized model ROC and AUC
fig = plt.figure(figsize=(10,10))
labels = ["clf_Log_Base", "clf_GaussianNB_Base", "clf_DTree_Grid", "clf_KNN_Grid", "clf_SVMLinear_Grid"]
for i, clf in enumerate([clf_Log, clf_Bayes, clf_DTree_Grid, clf_KNN_Grid, clf_SVMLinear_Grid]):
        if clf.__class__.__name__ == "SVC":
            fpr, tpr, _ = roc_curve(y_test, clf.decision_function(X_test3))
        else:
            fpr, tpr, _ = roc_curve(y_test, clf.predict_proba(X_test3)[:,1])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, color=resultsComb[labels].loc["colors"].values[i],lw=2, label=f'{labels[i]} auc = {roc_auc:.2f}')
```

```
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Optimized Classification Models')
plt.legend()
plt.show()
```

# References

[1] UCI Machine Learning Repository: Census Income Data Set. (n.d.). Retrieved December 11, 2022, from https://archive.ics.uci.edu/ml/datasets/census+income

[2] *Oracle® Crystal Ball Reference and Examples Guide*. Moved. (n.d.). Retrieved December 11, 2022, from https://docs.oracle.com/cd/E57185_01/CBREG/ch03s02s03s01.html#:~:text=A%20skewness%20value%20greater%20than,the%20distribution%20is%20fairly%20symmetrical

[3] *Sklearn.preprocessing.PowerTransformer*. scikit. (n.d.). Retrieved December 11, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html

[4] *Variance inflation factor (VIF)*. Corporate Finance Institute. (2022, December 5). Retrieved December 11, 2022, from https://corporatefinanceinstitute.com/resources/data-science/variance-inflation-factor-vif/