Salary earning prediction depending on different variables - Classification dataset

**Student ID: 9753941**

**Dataset link**: < <https://www.kaggle.com/datasets/ayessa/salary-prediction-classification> >

**Coventry GitHub Repository URL**: < <https://github.coventry.ac.uk/singhh48/9753941-HS-s1> >

Table of Contents

[1 Introduction 3](#_Toc120751327)

[1.1 Problem statement 3](#_Toc120751328)

[1.2 Existing Approaches 3](#_Toc120751329)

[2 Analysis 3](#_Toc120751330)

[2.1 Data visualization 4](#_Toc120751331)

[3 Pre-Processing 6](#_Toc120751332)

[4 Applying Machine Learning algorithms 7](#_Toc120751333)

[5 Model Tuning 7](#_Toc120751334)

[6 Evaluating the models 7](#_Toc120751335)

[7 Conclusions 7](#_Toc120751336)

[7.1 Comparing the approaches and results of other existing pieces of work 7](#_Toc120751337)

[8 Bibliography 8](#_Toc120751338)

[9 Appendix A 9](#_Toc120751339)

Table of Figures

[Figure 1 Salary vs. diff sex counts 5](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796202)

[Figure 2 Count of salary values with percentage 5](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796203)

[Figure 3 Marital-status counts 6](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796204)

[Figure 4 Education level counts 6](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796205)

[Figure 5 Work class counts 6](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796206)

[Figure 6 correlation plot of the selected features 7](#_Toc120796207)

[Figure 7 ROC and AUC curve 9](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796208)

[Figure 8 LR with parameters score 9](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796209)

[Figure 9 LR confusion matrix 9](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796210)

[Figure 10 MLP confusion matrix 10](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796211)

[Figure 11 MLP with parameters score 10](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796212)

[Figure 12 KNN confusion matrix 10](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796213)

[Figure 13 KNN with parameters score 10](file:///C:\Users\harma\Documents\6006cem\Coursework\6006CEM_9753941-HS-s1.docx#_Toc120796214)

List of Appendixes

[Appendix A A Names of Attributes 11](#_Toc120792343)

[Appendix A B Functions to compare the different models with default parameters 11](#_Toc120792344)

[Appendix A C Hyperparameters for Random and Grid Search 12](#_Toc120792345)

[Appendix A D Random and Grid Search Functions 12](#_Toc120792346)

[Appendix A E View all unique values in each feature 13](#_Toc120792347)

[Appendix A F Replacing all NaN values 15](#_Toc120792348)

[Appendix A G dataset after feature dropped and encoded 15](#_Toc120792349)

# Introduction

Predicting the salary of an individual can be a challenging task, however, with the help of Machine Learning and computing power, it’s possible to train the machine in such a manner that when we pass it different categorical and numerical values, we can predict if an individual will earn less of more than 50.000 dollars per year, also with the use of Machine learning techniques it’s possible to identify if some immutable characteristics that differentiate an individual from another can also be impactful in their respective ability to earn below or above the 50.000 dollar threshold. In this report, the dataset will meticulously be analysed and prepared to be processed through different Machine Learning algorithms to achieve the highest prediction accuracy possible.

## Problem statement

The dataset selected and used throughout this report has been sourced from kaggle.com/datasets/ayessa/salary-prediction-classification and contains the 1994 US income census data of working Adults. There are 32.6K instances in total, all containing different variables and values, which can be a factor in determining the annual income of specific population demographic. Predicting the salary of an induvial does not necessarily mean that is what they will earn during their career; however, it can be used as a method for choosing a career path that can lead to being part of the desired income bracket.

## Existing Approaches

During the research and literature review for this dataset, I have discovered that the most common approach used by other users on Kaggle.com has been Logistic Regression, followed by K-Nearest Neighbours and Random Forest Classifier, in average the accuracy score for Logistic Regression on multiple notebooks has been reported ranging from 80% to 85%, I will be using this figure numbers as a reference to what the accuracy score should be most close to, and to try and use other algorithms such as Neural Networks to get higher score where possible.

# Analysis

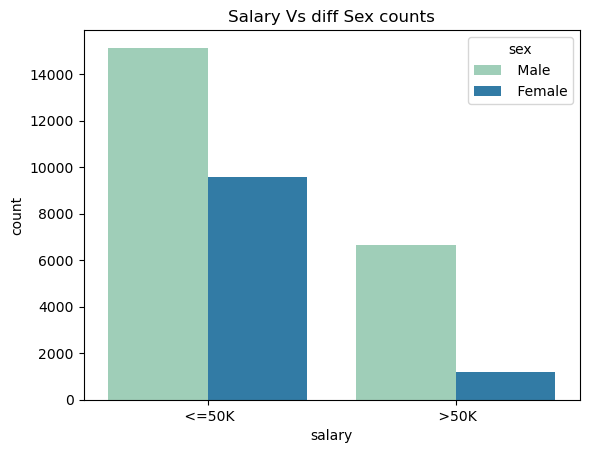
The dataset selected, has fifteen columns and 32561 instances of data, however, I may have to reduce the number of columns to choose only the data that is more relevant to the kind of prediction I am looking to achieve. The following [table](#Table) contains all the names of the columns and what they represent.

The first fourteen columns are all attributes that may correlate with how much an individual can earn each year. However, not all data present is composed of numerical values, some of which are categorical, meaning that they will need to be correctly processed and encoded in numbers before being passed to the machine learning algorithm.

During the analysis phase of the dataset, I found out that, although there appear to be no Null or NaN values present, three columns out of 15 have “?” values ([Appendix A E](#unique_values)), indicating that the data might be missing or not complete, possibly affecting the accuracy of the machine learning algorithm if used in this state with no changes, to find the missing values I used functions to print out each unique value contained in each column, identifying where the missing values are, and finally from the said column I took the total count of the missing values before replacing them for NaN, which makes them easier to handle with the pre-existing sklearn functions.

## Data visualization

The figures below (*Figure 1)* are the total count of salary attribute values, shown in a bar chart with their respective percentage values on top. Finally, (*Figure 2)* shows how much the two different sexes make each year below or above the $50.000 threshold.

Chart, bar chart

Description automatically generated

Figure Salary vs. diff sex counts

Figure 2 Count of salary values with percentage

Showing the total number of values count is helpful to understand if the data is more skewed to one side than the other and if there are a small number of instances that require to be oversampled or under-sampled, in the case of the dataset used, there is no need for oversampling of the >50K data as the total ratio is 24:100 (Ching, 2021) and oversampling is usually recommended on dataset sample with ratios of 1:100 for the minority of the values, I have not performed any oversampling on the dataset not to increase the chances of overfitting the data, as oversampling would create more of the instances containing >50K, by duplicating them randomly.

Chart, bar chart

Description automatically generatedDuring data visualization, I have found more discrepancies in the data that I will have to fix during pre-processing, such as:

The ‘marital-status’ attribute has three values that mean ‘married’; however, they have been divided into three categories, married civ spouse and AF spouse, respectively representing civilian and Armed forces, and finally, Married-spouse-absent, which can be categorized simply as married to reduce the number of categorical values that have small counts.

Figure Marital-status counts

Chart

Description automatically generated(*Figure 3*)

In the ‘Education’ attribute, I have observed that from preschool to grade 12th, the categorical data can be summarized as ‘school’ instead of having eight different values. (*Figure 4*)

Figure Education level counts

Chart

Description automatically generated

Finally, in the ‘workclass’ attribute, I have found out that there are Without-pay and Never-worked values that can be considered an outlier in a dataset that is used to predict an individual’s yearly salary, during pre-processing. I will eliminate all possible outliers that will influence the algorithm’s accuracy. (*Figure 5*)

Figure Work class counts

# Pre-Processing

In this phase, I need to prepare the data and ensure all re-adjustments are made from the data analysis and visualization findings. Therefore, I have replaced all missing data with NaN and removed them ([Appendix A F](#replacing_nan)), reducing the today instances from 32561 down to 30139; the value ‘Never-worked’ of the work class attribute was part of the instances removed, however ‘Without-pay’ had to be removed separately, I have combined the small value in marital-status and education attribute to a straightforward category, married and school respectively, and finally I have entirely removed the attribute that would not be fit for purpose or was not of any meaningful importance such as ‘fnlwgt’ (Chet Lemon, 2018), ‘capital-gains/loss’, and ‘education-num’ as ‘education’ represents the same values but that is not yet been encoded in numerical digits.

To prepare the data to be passed through different machine learning algorithms, I used the train\_test\_split method. I divided the code into scaled and unscaled versions while encoding all the features in numerical values with LabelEncoder ([Appendix A G](#drop_encode)). Finally, since I had removed the outliers in the data, the preferred method for scaling would be the StandardScaler (Joneliunas, 2018), which assumed all data is normalized, with distribution around 0 and standard deviant of 1, where the Min-Max scaler would have shrunk the range of the values between 0 and 1, therefore eliminating any issues that could be caused by any outliers presents.

Graphical user interface, chart, treemap chart

Description automatically generated

Figure correlation plot of the selected features

# Applying Machine Learning algorithms

In [Appendix A B](#Function_models), I have included the code I developed to compare and run different Algorithms. To make the code easier to read and better usability, I have divided the functions for scaled and unscaled data separately, where the functions have predefined split datasets of X\_test and Y\_test, and each prints out the Accuracy and Training score along with the time taken to run the algorithm, which is very important to assess with algorithm performs the fastest and how accurate it can be. Finally, to each of the functions I pass in the algorithm that I want to run and view their relative score, the best performing Algorithm with default parameters is MLP with 0.819 accuracy, followed by KneighborsClassifier and Linear Regression, having 0.803 and 0.771 each, for the scaled data. Below is a table that shows how accuracy changed for each algorithm while using scaled and unscaled data, however, the change is way too small to be considered significant, or could possibly be a small margin of error, there was noted no change in accuracy for Logistic Regression using both variants of data, however, from this point forward I will only be using the scaled data, to maintain consistency throughout the work, and because even if the difference of score was not huge it is still a slight improvement nonetheless.

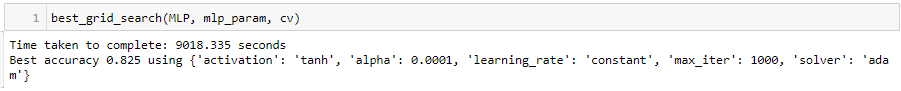
|  |  |  |  |
| --- | --- | --- | --- |
|  | Logistic Regression | MLP | KneighnorbsClassifier |
| Accuracy Scaled data | 0.771 | 0.819 | 0.803 |
| Accuracy Unscaled data | 0.771 | 0.788 | 0.790 |

# Model Tuning

To tune the hyperparameters ([Appendix A C](#hyperparameters)) of the models, I used Random and GridSearchCV [(Appendix A D)](#Search_Functions), which, again, have been produced in functions for repeatability of the results, both of the functions take as parameters, the algorithm model to be tuned, their relative hyperparameters to be searched for, and finally the cross-validation parameters, which splits the training dataset into ten folds to be validated against each other and given the best accuracy mean score, Random search appears to be the best-performing methods based on the time taken, however it does not always offer the best solution, where Grid Search, even though being heavily taxing on the system and using all of the cores, we are assured that all of the combinations of hyperparameters are tested for the best results, we can see from the table below that Logistic regression did not in fact change, but with Grid search, MLP and KNN showed some improvement although not as noticeable as I would have expected.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Logistic Regression | MLP | KneighnorbsClassifier |
| Random Search | 0.780 | 0.814 | 0.805 |
| Grid Search | 0.780 | 0.825 | 0.808 |

Despite being the most accurate, MLP, took the longest to run, 9018 seconds which amounts to 2.5 hours.



# Evaluating the models

Diagram

Description automatically generatedROC and AUC can be used to summarize the results, the graph on the side, show the True positive rate and False positive rate, we can see that even though MLP has the best accuracy of the three ML algorithm used, KNN, however, has the best ROC and AUC curve, meaning that KNN has a better threshold for accept the True positive, also the AUC of KNN model is greater than the rest of the models, making it the best model for this dataset.

Below is the confusion matric for LR, MLP and KNN, and when comparing the F1-score and Recall along with the precision, tells us that accuracy alone is not a perfect metric for evaluating the ML model, and in this case we can see that KNN has highest Precision, Recall and F1-score of the compared model, which clearing makes the best model to be used of this dataset.

Figure ROC and AUC curve

Chart, treemap chart

Description automatically generated

Table

Description automatically generated

Figure LR with parameters score

Figure LR confusion matrix

Chart, treemap chart

Description automatically generatedTable

Description automatically generated with medium confidence

Figure MLP with parameters score

Figure MLP confusion matrix

Chart, treemap chart

Description automatically generatedTable

Description automatically generated with low confidence

Figure 12 KNN with parameters score

Figure KNN confusion matrix

# Conclusions

In my work I have found that Logistic Regression, while being the fastest the algorithm to process the data results on being the least accurate, by not being able to linearly separate the Y vector data, MLP and KNN while being more accurate are also exponentially slower than Logistic Regression, I believe there are limitation to how I have pre-processed the data, which could be the reason why they might be affecting my scores, there is an issues with skewed data, shown in the figure below, there are lot more younger people in the census that earn less than 50K, which I had not accounted for in the pre-processing, I believe my results could be improved upon and finally, I found out, while testing with different features that only, age, race, sex, hours-per-week, occupation, really affect the prediction outcome.

Chart, histogram

Description automatically generated

Figure Skewed data graph

## Comparing the approaches and results of other existing pieces of work

Below is the table that show two other people on Kaggle who worked on the same dataset but achieved different accuracy results. The same Kaggle user, did not use the same machine learning algorithm that I have applied when working on this dataset, however overall, my score is slight lower than there, for the heavy pre-processing and cleaning that I had previously done.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Logistic Regression | MLP | KNN |
| My work | 0.780 | 0.825 | 0.808 |
| (TOLBA, 2022) | 0.851 | N/A | N/A |
| (MOHAMMADI, 2022) | 0.824 | N/A | 0.838 |

# Bibliography

Brownlee, J. (2018, August 31). *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

Chet Lemon, C. Z. (2018). *They are predicting if income exceeds $50,000 per year based on 1994 US Census Data with.* San Diego: https://cseweb.ucsd.edu/.

Ching. (2021, February 06). *Introduction to Synthetic Minority Over-sampling Technique and its Implementation from Scratch*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/introduction-to-synthetic-minority-over-sampling-technique-and-its-implementation-from-scratch-77593647c10d

GARG, R. (2018, January 18). *7 Types of Classification Algorithms*. Retrieved from analyticsindiamag.com: https://analyticsindiamag.com/7-types-classification-algorithms/

HAMDY, A. (2022, May). *Salary Predection 86.40% Accuracy*. Retrieved from kaggle.com: https://www.kaggle.com/code/abdo977/salary-predection-86-40-accuracy

JAIN, T. R. (2022, August). *Feature Selection Methods*. Retrieved from kaggle.com: https://www.kaggle.com/code/tarundalal/feature-selection-methods/notebook

Joneliunas, S. (2018, July 9). *Difference between Standard scaler and MinMaxScaler*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/51237635/difference-between-standard-scaler-and-minmaxscaler

Kumar, V. (2021, August 19). *KNN Classifier in Sklearn using GridSearchCV with Example*. Retrieved from machinelearningknowledge.ai: https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/

MOHAMMADI, H. (2022, October). *salary-classification-3-models-comparison*. Retrieved from kaggle.com: https://www.kaggle.com/code/hikmatullahmohammadi/salary-classification-3-models-coparison

OMAR, A. A. (2022, October). *Salary Prediction Classification | 88% Accuracy*. Retrieved from kaggle.com: https://www.kaggle.com/code/ahmedaliomar/salary-prediction-classification-88-accuracy

TOLBA, M. (2022, June). *Salary Prediction Classification*. Retrieved from kaggle.com: https://www.kaggle.com/code/mahmoudftolba/salary-prediction-classification/notebook

# Appendix A

Appendix A Names of Attributes

|  |  |
| --- | --- |
| Attributes | |
| Attribute name: | Description: |
| Age | Int value > 0: individual’s age |
| Work class | Categorical: Employment status of the individual |
| Fnlwgt | Final weight number int value > 0: count of individuals the census instance represents |
| Education | Categorical: Highest education of the individual |
| Education-num | Int value > 0: Education level enumerated |
| Marital-status | Categorical: Specifies the marital status of the individual |
| Occupation | Categorical: Specific occupation of the individual |
| Relationship | Categorical: Relationship relative to others |
| Race | Categorical: Individual’s race |
| Sex | Categorical: Individual’s biological sex |
| Capital-gain | Int value > 0: Income made from investing |
| Capital-loss | Int value > 0: Income lost from investing |
| Hours-per-week | Int value > 0: Individual’s reported weekly hours |
| Native-country | Categorical: individual’s country of origin |
| Salary | Prediction value greater, less, or equal to 50.000 dollars |

Appendix A Functions to compare the different model with default parameters

|  |
| --- |
| **def** **fit\_and\_test**(model, show\_score):  start = time.time()    model.fit(X\_train,y\_train)  predict = model.predict(X\_test)    end = time.time()  time\_taken = end-start  **if** show\_score:  **print**("Using: ", str(model))  **print**("Training score: {:.3f}".format(model.score(X\_train, y\_train)))  **print**("Accuracy score: {:.3f}".format(metrics.accuracy\_score(y\_test, predict)))  **print**(classification\_report(y\_test, predict, target\_names=['<=50k','>50k']))  **print**("Time taken to complete: {:.3f} seconds".format(time\_taken))  **def** **scaled\_and\_test**(model, show\_score):  start = time.time()    model.fit(X\_train\_scaled,y\_train\_scaled)  predict = model.predict(X\_test\_scaled)    end = time.time()  time\_taken = end-start  **if** show\_score:  **print**("Using: ", str(model))  **print**("Training score for scaled data: {:.3f}".format(model.score(X\_train\_scaled, y\_train\_scaled)))  **print**("Accuracy score for scaled data: {:.3f}".format(metrics.accuracy\_score(y\_test\_scaled, predict)))  **print**(classification\_report(y\_test\_scaled, predict, target\_names=['<=50k','>50k']))  **print**("Time taken to complete: {:.3f} seconds".format(time\_taken)) |

Appendix A Hyperparameters for Random and Grid Search

|  |
| --- |
| # defining Hyperparameters for Random and Grid search  **from** **sklearn.model\_selection** **import** RepeatedStratifiedKFold  #### reference: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/  lr\_param = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],  'penalty': ['l1', 'l2', 'elasticnet', 'none'],  'C': [**100**, **10**, **1.0**, **0.1**, **0.01**],  'max\_iter': [**200**, **1000**, **5000**, **10000**]}  #### reference: https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html  mlp\_param = {'activation': ['identity', 'logistic','tanh', 'relu'],  'solver': ['sgd', 'adam'], 'alpha': [**0.0001**, **0.05**],  'learning\_rate': ['constant', 'invscaling', 'adaptive'],  'max\_iter': [**200**, **1000**, **5000**, **10000**]}  ### reference: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html  KNN\_param = {  'algorithm': ('auto', 'ball\_tree', 'kd\_tree', 'brute'),  'n\_neighbors': range(**1**,**30**),  'leaf\_size': range(**1**,**30**),  'p': (**1**,**2**,**3**,**4**),  'weights': ('uniform', 'distance'),  'metric': ('minkowski', 'manhattan')}  # Cross validation parameters  cv = RepeatedStratifiedKFold(n\_splits=**10**, n\_repeats=**3**, random\_state=**1**) |

Appendix A Random and Grid Search Functions

|  |
| --- |
| **from** **sklearn.model\_selection** **import** RandomizedSearchCV  #### reference: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/  **def** **best\_random\_search**(model, param, CV):  start = time.time()  rand = RandomizedSearchCV(model, param, n\_jobs=-**1**, cv=CV, scoring='accuracy', error\_score=**0**)  rand\_search = rand.fit(X\_train\_scaled, y\_train\_scaled)  end = time.time()  **print**("Time taken to complete: {:.3f} seconds".format(end - start))  **print**("Best accuracy {:.3f} using {}".format(rand\_search.best\_score\_, rand\_search.best\_params\_))  **from** **sklearn.model\_selection** **import** GridSearchCV  **def** **best\_grid\_search**(model, param, CV):  start = time.time()  grid = GridSearchCV(estimator=model, param\_grid=param, n\_jobs=-**1**, cv=CV, scoring='accuracy',error\_score=**0**)  grid\_search = grid.fit(X\_train\_scaled, y\_train\_scaled)  end = time.time()  **print**("Time taken to complete: {:.3f} seconds".format(end - start))  **print**("Best accuracy {:.3f} using {}".format(grid\_search.best\_score\_, grid\_search.best\_params\_)) |

Appendix A View all unique values in each feature

|  |
| --- |
| **def** **print\_uniques**(df):  list\_col = df.columns.values.tolist() # print all unique values in each of the columns  **for** index, value **in** enumerate(list\_col): # and their counts  **print**(value, df[value].unique(), "Number of unique values: ",  len(df[value].unique()),  end="**\n**------------------------------------------------------------------------------**\n**" )  print\_uniques(data)  Output: |

Appendix A Replacing all NaN values

|  |
| --- |
| data.replace(' ?', np.nan, inplace=True)  **print**("Checking for any NaN values after replacing all ? in the dataset :",'**\n**',data.isna().sum())  Output: |

Appendix A dataset after feature dropped and encoded

|  |
| --- |
| drop\_columns=['capital-gain','capital-loss','fnlwgt', 'education-num',  'workclass', 'relationship', 'native-country']  data.drop(drop\_columns, axis=**1**, inplace=True)    #encoding the unscaled data  data\_unscaled = data  data\_unscaled = data\_unscaled.apply(le.fit\_transform)  data\_unscaled  output: |

Appendix A Complete code in salaryPrediction

|  |
| --- |
| # Data Analysis  **import** **pandas** **as** **pd**  **import** **numpy** **as** **np**  **import** **warnings**  warnings.filterwarnings('ignore')  data = pd.read\_csv('salary.csv') # importing dataset to be processed  **print**("Dataset shape:", data.shape, '**\n**') # row and columns total count  **print**("Dataset info: ", data.info(), '**\n**')  data.head(**15**)  **print**("Checking for any null values:",'**\n**',data.isnull().sum())  **print**("Checking for unique values in each column: ", '**\n**', data.duplicated().sum())  **print**("Checking for unique values in each column: ", '**\n**', data.nunique())  **def** **print\_uniques**(df):  list\_col = df.columns.values.tolist() # print all unique values in each of the columns  **for** index, value **in** enumerate(list\_col): # and their counts  **print**(value, df[value].unique(), "Number of unique values: ",  len(df[value].unique()),  end="**\n**------------------------------------------------------------------------------**\n**" )  print\_uniques(data)  **def** **print\_counts**(df, bol, \*args):  **for** i **in** args:  **if** i **in** df:  **print**('**\n**', i,' **\n**',df[i].value\_counts(), '**\n**',  end='------------------------------------------------------------------------------',)  **else**:  **continue**  print\_counts(data, True, 'workclass', 'occupation', 'native-country', 'education', 'marital-status')  af\_work\_marital = data[(data['marital-status'] == ' Married-AF-spouse')  | (data['occupation'] == ' Armed-Forces')]  af\_work\_marital.shape  ## Data Visualization  **import** **matplotlib.pyplot** **as** **plt**  **import** **seaborn** **as** **sns**  %matplotlib inline  #### reference: https://stackoverflow.com/questions/33179122/seaborn-countplot-with-frequencies/33259038  #### reference: https://www.kaggle.com/code/abdo977/salary-predection-86-40-accuracy  plt.title('Count of Salary less and more than 50k')  salary\_percen = sns.countplot(data.salary, data=data, palette='YlGnBu')  total = len(data.salary)  **for** p **in** salary\_percen.patches:  x=p.get\_bbox().get\_points()[:,**0**]  y=p.get\_bbox().get\_points()[**1**,**1**]  salary\_percen.annotate('{:.1f}%'.format(**100.**\*y/total), (x.mean(), y),  ha='center', va='bottom')  plt.show()  plt.title('Salary Vs diff Sex counts')  sns.countplot(data.salary, hue=data.sex, palette='YlGnBu')  #### reference : https://www.kaggle.com/code/ahmedaliomar/salary-prediction-classification-88-accuracy#Data-Preprocessing  plt.title('Salary distribuited relative to Age')  sns.kdeplot(data=data, x=data['age'], hue=data['salary'], fill=True, palette='Set2' )  plt.axvline(data.age.median(), color = 'red')  plt.title('Salary relative to Hours worked per week')  sns.kdeplot(data=data, x=data['hours-per-week'], hue=data['salary'], fill=True, palette='Set2')  plt.axvline(data['hours-per-week'].median(), color = 'red')  ##### Reference: https://www.youtube.com/watch?v=4DnWYK88-E4&ab\_channel=SessionWithSumit  ycols = ['education', 'occupation', 'workclass', 'relationship','race', 'marital-status']  plt.figure(figsize=(**7**,**28**))  **for** i **in** enumerate(ycols):  plt.subplot(**6**,**1**,i[**0**]+**1**)  plt.title('{} variable count separated by their Salary <=50 or >50'.format(i[**1**].capitalize()))  sns.countplot(data=data, y=data[i[**1**]], hue=data.salary, orient='v', palette='YlGnBu')  plt.legend(title='Salary',loc='lower right')  # Data Pre-Processing  ## Cleaning the Data  Deleting unecessary columns, combininig some of the values to simplify them  data.replace(' ?', np.nan, inplace=True)  **print**("Checking for any NaN values after replacing all ? in the dataset :",'**\n**',data.isna().sum())  data.drop\_duplicates(inplace=True)  data.dropna(inplace=True) #dropping all the NaN values  **print**("Dataset shape after deleting all NaN values: ", data.shape, '**\n**')  # dropping instances of rows where col workclass contains without-pay,  # small instances of data that could affect the final results  data.drop(data.loc[data['workclass']==' Without-pay'].index, inplace=True)  # combining Married-AF-spouse and Married-civ-spouse because these are small instances  # that means the same  data['marital-status'] = data['marital-status'].replace([' Married-civ-spouse',  ' Married-AF-spouse',  ' Married-spouse-absent'],  'Married')  data['marital-status'].value\_counts()  data['education'] = data['education'].replace([ ' 12th', ' 11th',' 9th', ' 7th-8th',  ' 5th-6th', ' 10th', ' 1st-4th', ' Preschool'],  ' School')  data['education'].value\_counts()  # dropping columns for capital-gain/loss  # maybe drop native-country completly?  drop\_columns=['capital-gain','capital-loss','fnlwgt', 'education-num',  'workclass', 'relationship', 'native-country']  data.drop(drop\_columns, axis=**1**, inplace=True)  ## Splitting the Data  **from** **sklearn.model\_selection** **import** train\_test\_split  **from** **sklearn.preprocessing** **import** LabelEncoder  **from** **sklearn.preprocessing** **import** StandardScaler  scale=StandardScaler()  le=LabelEncoder()  #### encoding the unscaled data  #encoding the unscaled data  data\_unscaled = data  data\_unscaled = data\_unscaled.apply(le.fit\_transform)  data\_unscaled  data\_unscaled.corr()  # after data has been preprocessed and encoded  correlation\_matrix = data\_unscaled.corr().round(**3**)  plt.figure(figsize=(**15**,**10**))  sns.heatmap(data=correlation\_matrix, annot=True)  X = data\_unscaled.iloc[**1**:,:-**1**]  y = data\_unscaled.iloc[**1**:,-**1**]  **print**(X)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=**0.25**, random\_state=**0**)  #### scaling the data  data\_scaled = data\_unscaled  X\_scaled = data\_scaled.iloc[**1**:,:-**1**]  y\_scaled = data\_scaled.iloc[**1**:,-**1**]  **print**(X\_scaled)  **print**(X\_scaled.shape)  # scaling the data  **def** **scaled\_data**(data\_x):  scale.fit(data\_x)  X\_scaled = scale.transform(data\_x)  **return** X\_scaled  X\_scaled = scaled\_data(X\_scaled)  **print**(X\_scaled)  **print**(X\_scaled.shape)  X\_train\_scaled, X\_test\_scaled, y\_train\_scaled, y\_test\_scaled = train\_test\_split(X\_scaled,y\_scaled, test\_size=**0.25**, random\_state=**0**)  #print\_uniques(data\_scaled)  #print\_counts(data\_scaled, 'workclass', 'occupation', 'native-country', 'education', 'education-num', 'marital-status')  # Applied machine learning algorithms  **from** **sklearn** **import** metrics  **from** **sklearn.metrics** **import** classification\_report  **import** **time**  **def** **fit\_and\_test**(model, show\_score):  start = time.time()    model.fit(X\_train,y\_train)  predict = model.predict(X\_test)    end = time.time()  time\_taken = end-start  **if** show\_score:  **print**("Using: ", str(model))  **print**("Training score: {:.3f}".format(model.score(X\_train, y\_train)))  **print**("Accuracy score: {:.3f}".format(metrics.accuracy\_score(y\_test, predict)))  **print**(classification\_report(y\_test, predict, target\_names=['<=50k','>50k']))  **print**("Time taken to complete: {:.3f} seconds".format(time\_taken))  **def** **scaled\_and\_test**(model, show\_score):  start = time.time()    model.fit(X\_train\_scaled,y\_train\_scaled)  predict = model.predict(X\_test\_scaled)    end = time.time()  time\_taken = end-start  **if** show\_score:  **print**("Using: ", str(model))  **print**("Training score for scaled data: {:.3f}".format(model.score(X\_train\_scaled, y\_train\_scaled)))  **print**("Accuracy score for scaled data: {:.3f}".format(metrics.accuracy\_score(y\_test\_scaled, predict)))  **print**(classification\_report(y\_test\_scaled, predict, target\_names=['<=50k','>50k']))  **print**("Time taken to complete: {:.3f} seconds".format(time\_taken))  ## Logistic Regression  **from** **sklearn.linear\_model** **import** LogisticRegression  LR=LogisticRegression()  fit\_and\_test(LR, show\_score=True)  **print**('**\n**')  scaled\_and\_test(LR, show\_score=True)  ## MLPClassifier  **from** **sklearn.neural\_network** **import** MLPClassifier  MLP = MLPClassifier()  fit\_and\_test(MLP, show\_score=True)  **print**('**\n**')  scaled\_and\_test(MLP, show\_score=True)  ## KNN  **from** **sklearn.neighbors** **import** KNeighborsClassifier  KNN = KNeighborsClassifier()  fit\_and\_test(KNN, show\_score=True)  **print**('**\n**')  scaled\_and\_test(KNN, show\_score=True)  # Model tuning  # defining Hyperparameters for Random and Grid search  **from** **sklearn.model\_selection** **import** RepeatedStratifiedKFold  #### reference: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/  lr\_param = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],  'penalty': ['l1', 'l2', 'elasticnet', 'none'],  'C': [**100**, **10**, **1.0**, **0.1**, **0.01**],  'max\_iter': [**200**, **1000**, **5000**, **10000**]}  #### reference: https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html  mlp\_param = {'activation': ['identity', 'logistic','tanh', 'relu'],  'solver': ['sgd', 'adam'], 'alpha': [**0.0001**, **0.05**],  'learning\_rate': ['constant', 'invscaling', 'adaptive'],  'max\_iter': [**200**, **1000**, **5000**, **10000**]}  ### reference: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html  KNN\_param = {  'algorithm': ('auto', 'ball\_tree', 'kd\_tree', 'brute'),  'n\_neighbors': range(**1**,**30**,**20**),  'leaf\_size': range(**1**,**30**,**2**),  'weights': ('uniform', 'distance'),  'metric': ('minkowski', 'manhattan')}  # Cross validation parameters  cv = RepeatedStratifiedKFold(n\_splits=**10**, n\_repeats=**3**, random\_state=**1**)  ## Randomsearch  **from** **sklearn.model\_selection** **import** RandomizedSearchCV  #### reference: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/  **def** **best\_random\_search**(model, param, CV):  start = time.time()  rand = RandomizedSearchCV(model, param, n\_jobs=-**1**, cv=CV, scoring='accuracy', error\_score=**0**)  rand\_search = rand.fit(X\_train\_scaled, y\_train\_scaled)  end = time.time()  **print**("Time taken to complete: {:.3f} seconds".format(end - start))  **print**("Best accuracy {:.3f} using {}".format(rand\_search.best\_score\_, rand\_search.best\_params\_))  ### Random search with Logistic Regression  best\_random\_search(LR, lr\_param, cv)  ### Random search with MLP  best\_random\_search(MLP, mlp\_param, cv)  ### Random search with KNN  best\_random\_search(KNN, KNN\_param, cv)  ## Gridsearch  **from** **sklearn.model\_selection** **import** GridSearchCV  **def** **best\_grid\_search**(model, param, CV):  start = time.time()  grid = GridSearchCV(estimator=model, param\_grid=param, n\_jobs=-**1**, cv=CV, scoring='accuracy',error\_score=**0**)  grid\_search = grid.fit(X\_train\_scaled, y\_train\_scaled)  end = time.time()  **print**("Time taken to complete: {:.3f} seconds".format(end - start))  **print**("Best accuracy {:.3f} using {}".format(grid\_search.best\_score\_, grid\_search.best\_params\_))  ### Grid search with Logistic Regression  best\_grid\_search(LR, lr\_param, cv)  ### Grid search with MLP  best\_grid\_search(MLP, mlp\_param, cv)  ### Grid search with KNN  best\_grid\_search(KNN, KNN\_param, cv)  # Evaluation  #### reference: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/  #### reference: https://www.youtube.com/watch?v=TEkvKx2tQHU&ab\_channel=NormalizedNerd  **from** **sklearn.datasets** **import** make\_classification  **from** **sklearn.metrics** **import** precision\_recall\_curve  **from** **sklearn.metrics** **import** f1\_score  **from** **sklearn.metrics** **import** auc  ### Logistic Regression  LR = LogisticRegression(solver= 'liblinear', penalty= 'l1', max\_iter= **200**, C= **1.0**)  LR.fit(X\_test\_scaled, y\_test\_scaled)  LR\_predict = LR.predict(X\_test\_scaled)  **print**(classification\_report(y\_test\_scaled, LR\_predict, target\_names=['<=50k','>50k']))  ### MLP  MLP = MLPClassifier(activation = 'tanh', alpha=**0.0001**, learning\_rate= 'constant', max\_iter= **1000**, solver= 'adam')  MLP.fit(X\_test\_scaled, y\_test\_scaled)  MLP\_predict = MLP.predict(X\_test\_scaled)  **print**(classification\_report(y\_test\_scaled, MLP\_predict, target\_names=['<=50k','>50k']))  ### KNN  KNN = KNeighborsClassifier(algorithm= 'ball\_tree', leaf\_size= **3**, metric= 'manhattan', n\_neighbors= **21**, weights= 'uniform')  KNN.fit(X\_test\_scaled, y\_test\_scaled)  KNN\_predict = KNN.predict(X\_test\_scaled)  **print**(classification\_report(y\_test\_scaled, KNN\_predict, target\_names=['<=50k','>50k']))  ## ROC and AUC  #### reference: https://scikit-learn.org/stable/auto\_examples/release\_highlights/plot\_release\_highlights\_0\_22\_0.html#sphx-glr-auto-examples-release-highlights-plot-release-highlights-0-22-0-py  **from** **sklearn.metrics** **import** plot\_roc\_curve  curve\_plot = plot\_roc\_curve(LR, X\_test\_scaled, y\_test\_scaled)  plot\_roc\_curve(MLP, X\_test\_scaled, y\_test\_scaled, ax=curve\_plot.ax\_)  plot\_roc\_curve(KNN, X\_test\_scaled, y\_test\_scaled, ax=curve\_plot.ax\_)  curve\_plot.figure\_.suptitle("ROC curve comparison")  ## Confusion matrix  **from** **sklearn.metrics** **import** plot\_confusion\_matrix  plot\_confusion\_matrix(LR, X\_test\_scaled, y\_test\_scaled, display\_labels = ['<=50K','>50K'])  plt.title("Logistic Regression Confusion Metrix")  plot\_confusion\_matrix(MLP, X\_test\_scaled, y\_test\_scaled, display\_labels = ['<=50K','>50K'])  plt.title("MLP Confusion Metrix")  plot\_confusion\_matrix(KNN, X\_test\_scaled, y\_test\_scaled, display\_labels = ['<=50K','>50K'])  plt.title("KNN Confusion Metrix") |