Electronic Assignment Cover sheet

Course Title: MSc in Data Analytics

Lecturer Name: Courtney Ford

Module/Subject Title: Machine Learning and Pattern Recognition

Assignment Title: CA_ONE

No of Words: 2643

Income Classification Using Logistic Regression

History:

For this assessment, we explored several problem statements available across various platforms including government databases and data science communities like Kaggle. After researching a lot using Google, we sifted through various problem statements like predicting customer conversion, accommodation prices, credit card fraud detection and so on but we found them unsuitable for use as they had huge class imbalances and so we rejected those. We finally came across this Census Income dataset that sparked our interest and we selected it as a problem to be solved in this assessment using Machine Learning.

Dataset:

Link to the dataset: Census Income · master · Data Science Dojo / datasets · Code

The Census Income dataset was extracted from the census bureau database. This data set has been sourced from the Machine Learning Repository of University of California, Irvine. We came across this dataset on the Data Science dojo portal through Google and decided to use it. The dataset has 15 variables with a mix of both continuous and discrete features and it gives a demographic insight of a person's income sources. For every person, we have a view of their age, occupation, number of years of education along with their capital gain and loss for our analysis. Apart from this, the dataset has the variable 'Income' which is a categorical variable having two distinct classes, one being "greater than 50K" and the other being "less than or equal to 50K". As a group, we concluded that this variable is fit to be our target variable from our understanding of the domain. Having selected Income as our target feature, we decided to use Machine Learning to classify a person's income based on their demographic information.

Problem Statement:

To classify a given record into the correct classes based on the person's demographics as well as employment details. As our target feature is a categorical variable, we decided to solve this classification problem using the Logistic regression algorithm of Machine learning.

Dataset definition:

Age: Age of a person in years
Workclass: Includes 8 different work categories: (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)
Fnlwgt: The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population.
Education: Education: (Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool)
Education-num: Years of education
Marital-status: Marital Status: (Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse)
Occupation: Occupation: (Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces)
Relationship: Relationship:(Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried)
Race: Race: (White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black)
Sex: Sex: (Male, Female)
Capital-gain: Amount of capital gained
Capital-loss: Amount of capital lost
Hours-per-week: Number of hours worked per week
Native-country: Native country: (United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica,

Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands)
Income: Either the income is greater than $50,000 or lesser than and equal to $50,000: (>50K, <=50K)

Data Preparation (EDA and pre-processing)

Once we finalized our problem statement, we moved on to take a deeper look into the dataset and understand columns and their representation. We began with importing essential Python libraries like pandas, numpy and matplotlib to begin with. The dataset was available to us in two separate parts (adult.data and adult.test) and so we loaded these separately. As the columns were not labelled, we included column headers for each variable and concatenated both files into a single dataset for pre-processing.

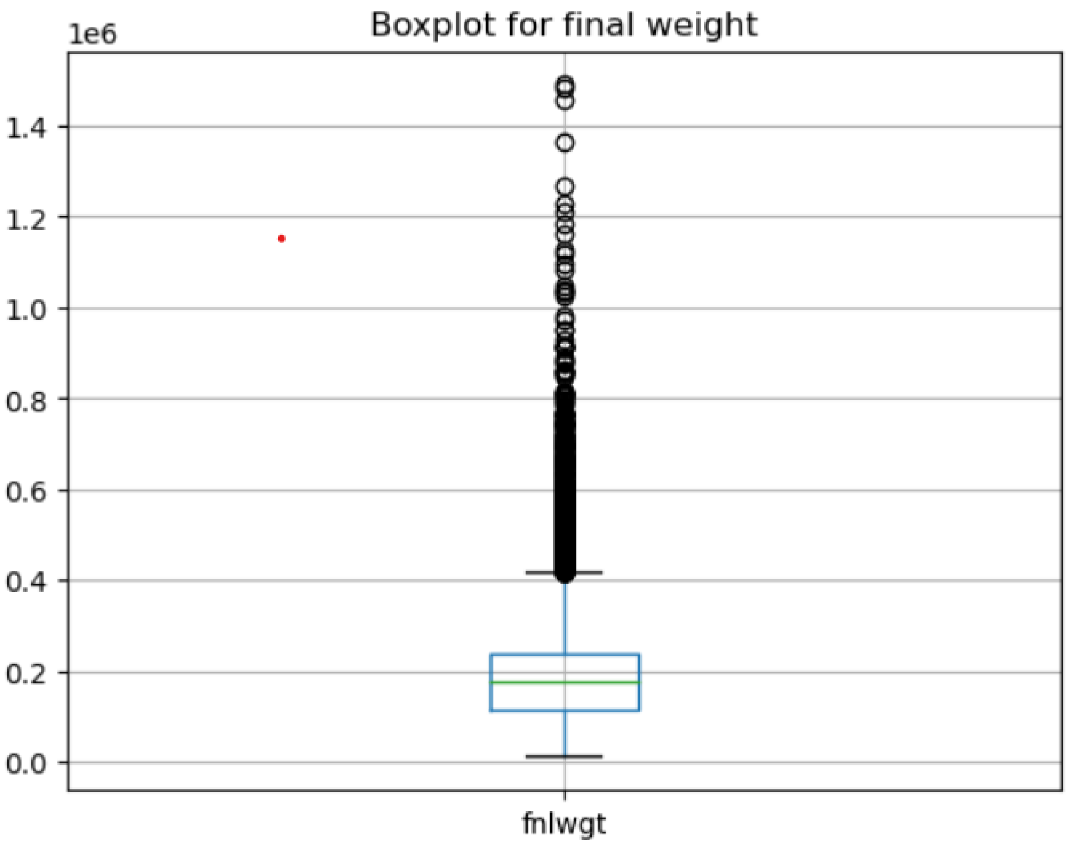
Pre-processing:

A quick check for the dimensionality of the dataset using data.shape() showed that we have 48840 row and 15 columns. We then looked at the value in each column for the first five rows using data.head(). Using data.info(), we see the non-null value counts for each feature along with the datatypes for each index. We also look at the description of numerical data in the dataframe using data.describe() which gives us the statistical summary for those values like mean, minimum and maximum of the values as well as the standard deviation and variance of our features. We found 52 duplicate values and hence dropped them.

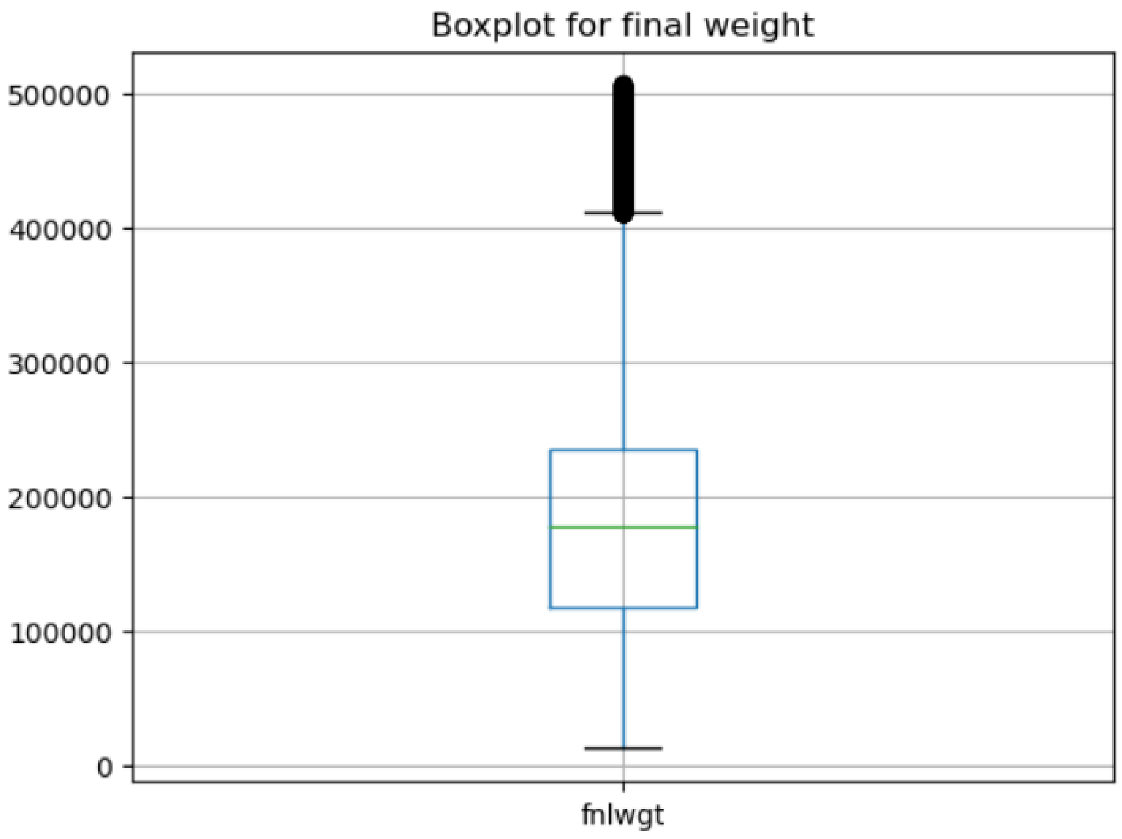
Outlier detection:

We did outlier detection on the variable final weight as it is the pre-computed value for the entire demographic data for a person. We plotted a boxplot to see the outliers before data processing as this shows the skewness of the data for final weight and saw that there were many outliers beyond the maximum value for final weight. Further, we chose Z-Score to remove outliers as it uses standard deviation with a range < 3. We re-plotted the box plot and can now see outliers that had a z-score greater than 3 were eliminated.

```
Before removing Outliers
```



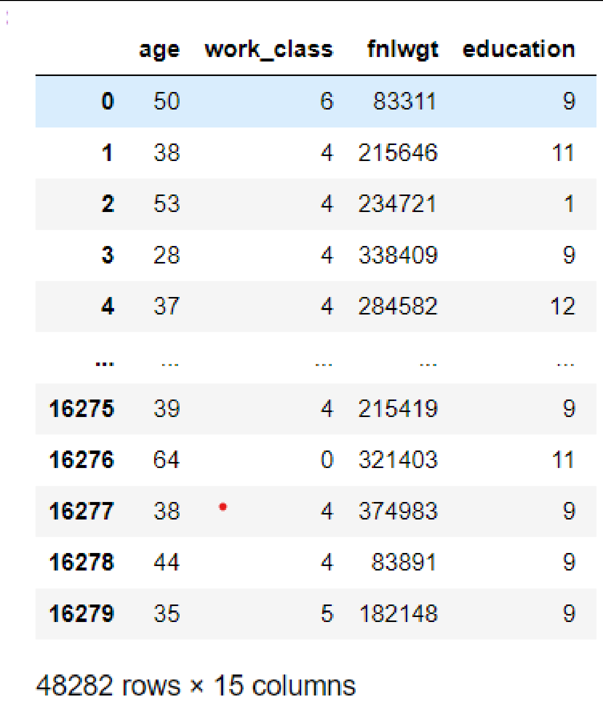
```
After removing Outliers
```



Binning:

We have used Binning on the feature ‘age’ to gather a better insight on the general income trend. As incomes can be similar for certain age groups, we have converted this value into age buckets of 20 years starting from teens who could be up to 20 years to senior citizens aged 100 years.

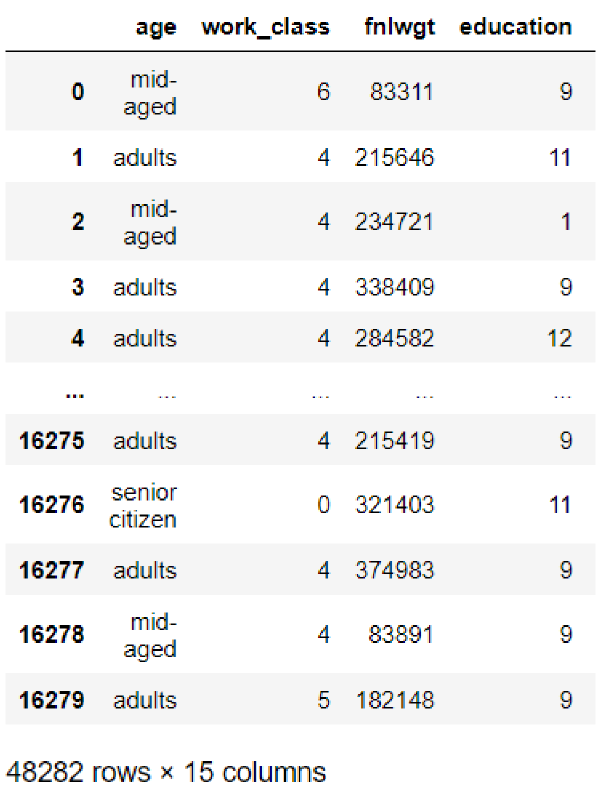
Before Binning:



	age	work_class	fnlwgt	education
0	50	6	83311	9
1	38	4	215646	11
2	53	4	234721	1
3	28	4	338409	9
4	37	4	284582	12
...	...	...	...	...
16275	39	4	215419	9
16276	64	0	321403	11
16277	38	4	374983	9
16278	44	4	83891	9
16279	35	5	182148	9

48282 rows × 15 columns

After Binning:



	age	work_class	fnlwgt	education
0	mid-aged	6	83311	9
1	adults	4	215646	11
2	mid-aged	4	234721	1
3	adults	4	338409	9
4	adults	4	284582	12
...	...	...	...	...
16275	adults	4	215419	9
16276	senior citizen	0	321403	11
16277	adults	4	374983	9
16278	mid-aged	4	83891	9
16279	adults	5	182148	9

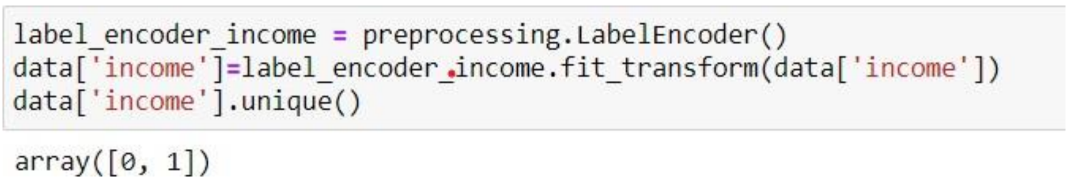
48282 rows × 15 columns

Standardization:

We standardized the data using the MinMaxScaler mechanism so that variables contribute equally to the model fitting and all features are transformed into the range 0 and 1. Variables, such as age and final weight, might have values at different scales and doing this standardizes the data.

Label encoding:

As models work best with numerical values, we encoded the non-numerical features into numerical categories instead of text using label encoding to get the best results. Even our target feature ‘Income’ has two categories ‘>50K’ and ‘<=50K’ which are then encoded as 0 and 1.



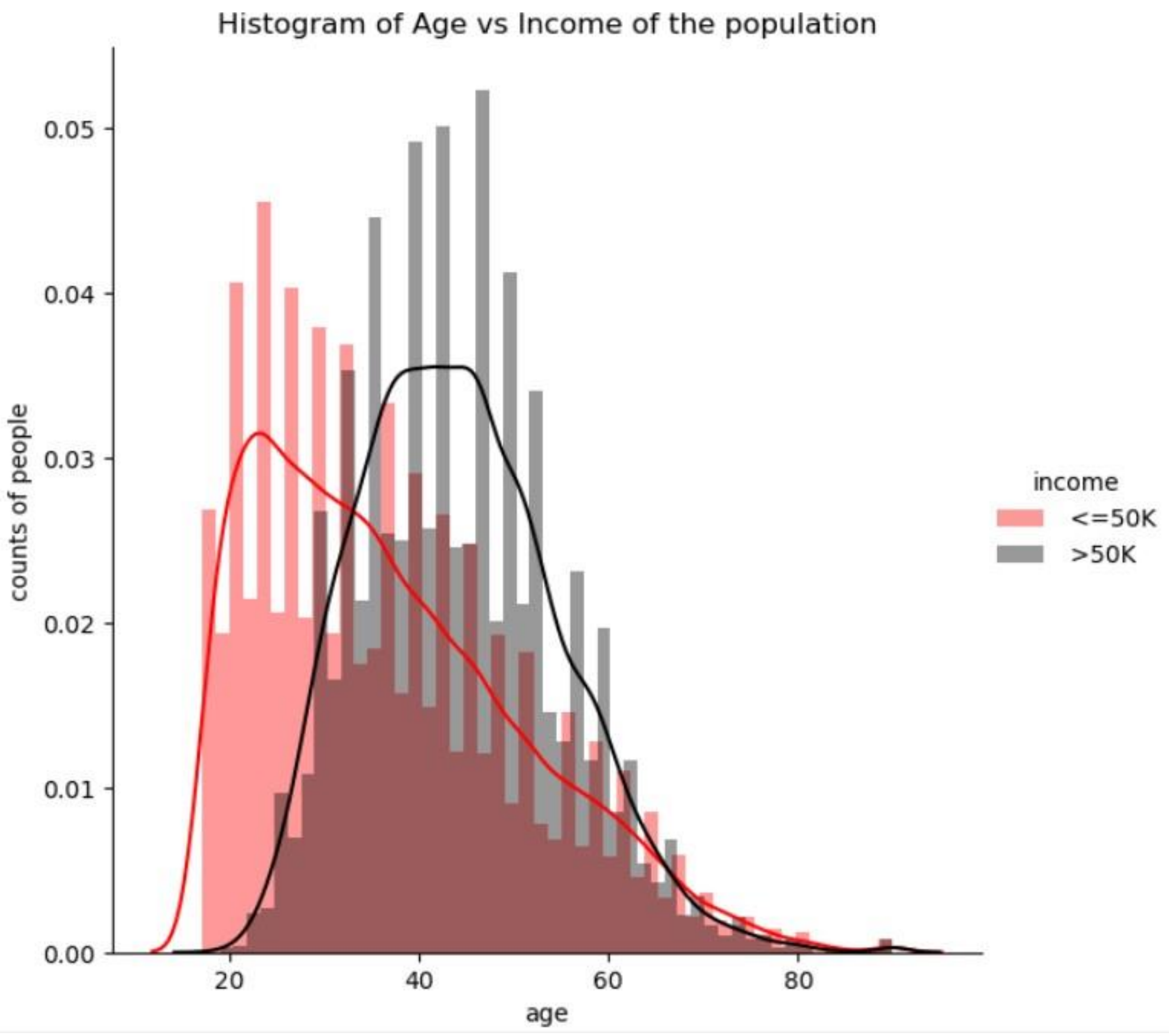
```
label_encoder_income = preprocessing.LabelEncoder()
data['income']=label_encoder_income.fit_transform(data['income'])
data['income'].unique()
```

```
array([0, 1])
```

Data Visualization:

In this step we agree that it essential to generate more graphical views of the data in order to see more clearly the relevance of the features, as well as to analyze the correlation and distribution between them. For this, we have the support of the Seaborn library, which was a fundamental part of the base and construction of our codes. Among the representations that can be seen in our CA-one we have:

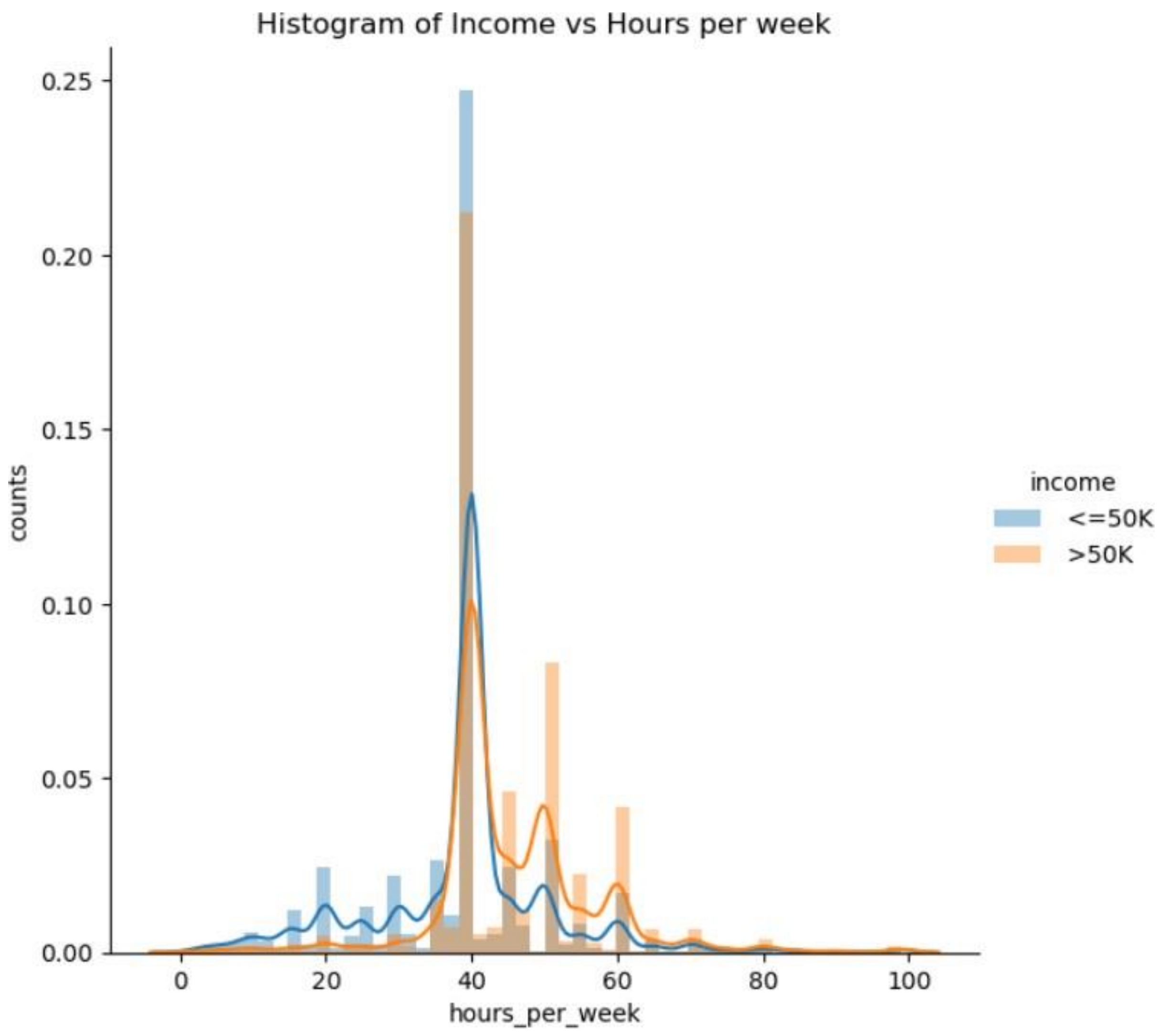
Histogram comparing Age vs Income:



Observation:

Here, it was possible to observe the concentration of higher income between the ages of 25 and 60 for those who earn more than 50k, while for those who earn less than 50k, but still make a considerable income, ages vary between 17 and 60.

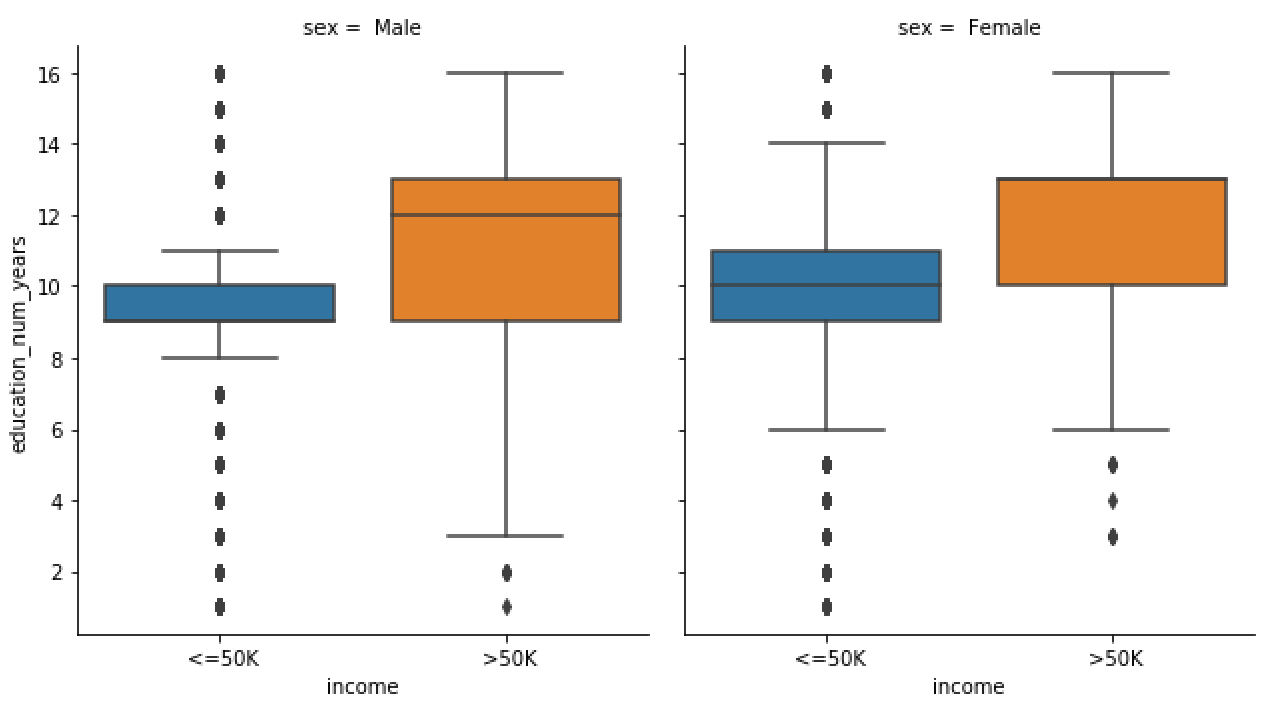
Histogram comparing Income vs Hours per week:



Observation:

The conclusion we had here was that many people working less than 40 hours seem to have an income less than 50K. In comparison, majority of people that work for more than 40 hours have an income greater than 50K.

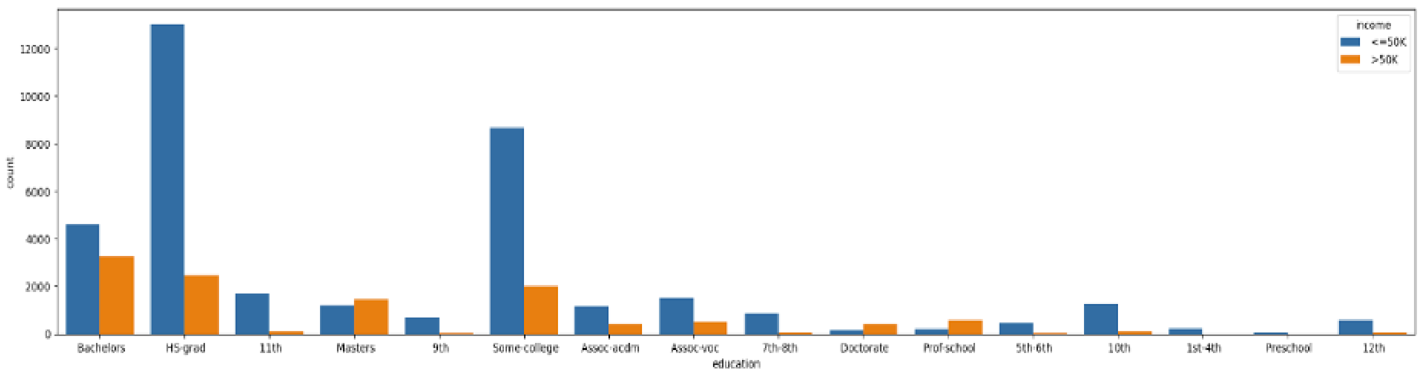
Category plot for education_num_years vs Income for both sexes :



Observation:

We used Catplot to see the differences between bivariate feature plot of income and education_num_years for both the sexes and found that there is no visible difference in the income patterns for both male and female. In general, as education_num_years increases, the range of income increases as well.

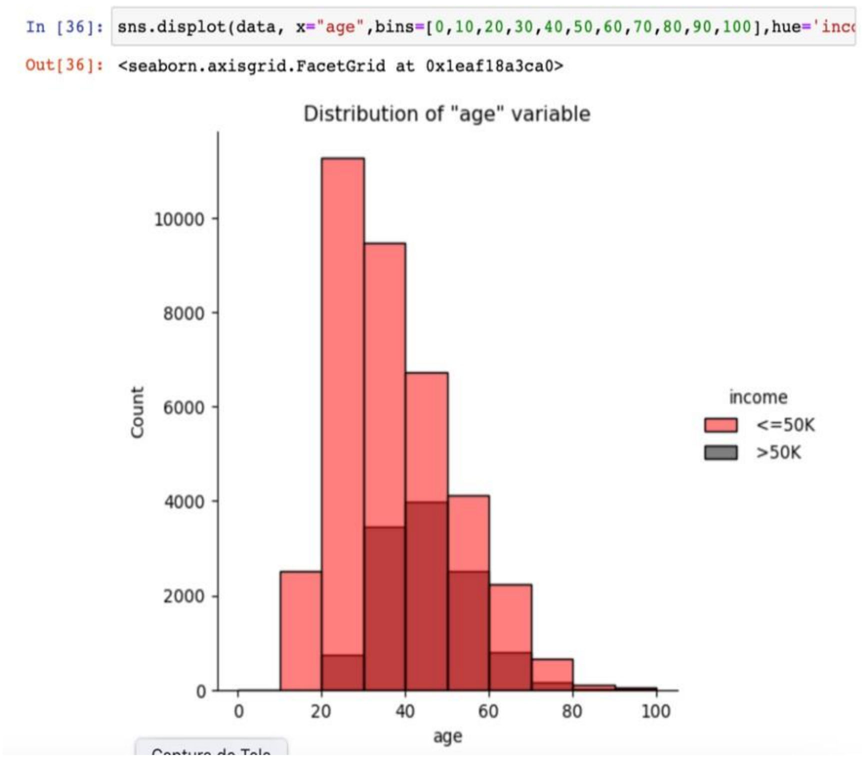
Bar Stack:



Observation:

From this bar stack, we concluded that more people who have a Master’s in education have an income greater than 50K as well as majority of the people who graduated High-school have an income less than 50K. Also, none of the people with only preschool education have an income greater than 50K.

Histogram of Distribution of Income with Age:

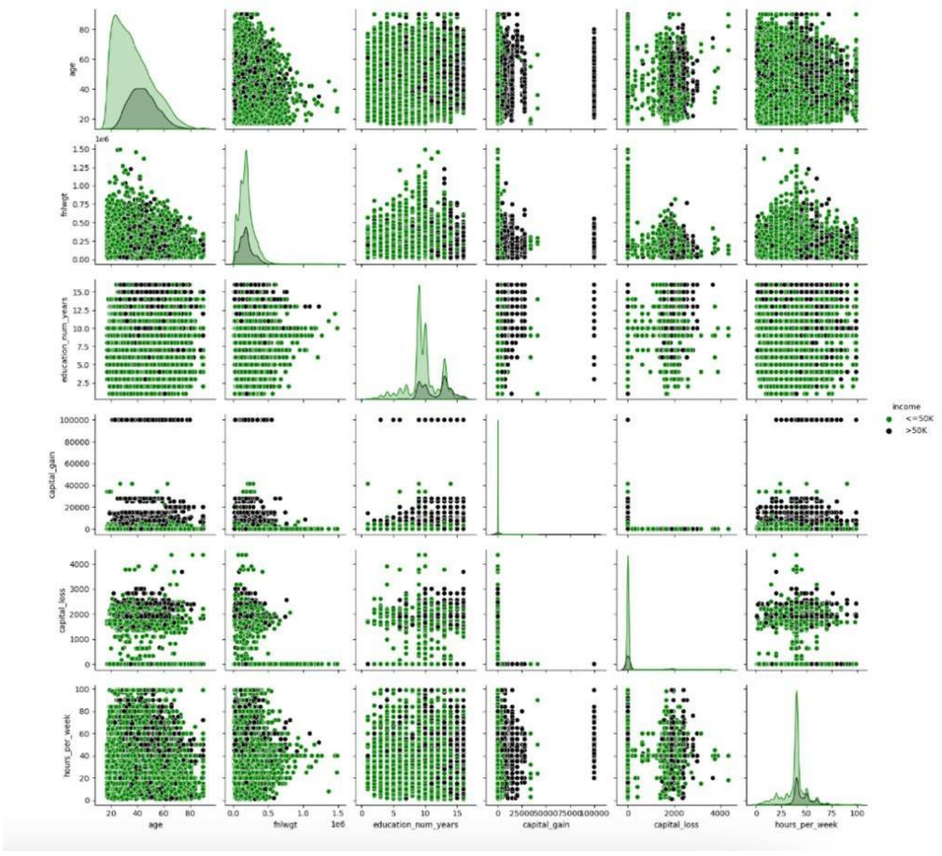


```
In [36]: sns.displot(data, x="age",bins=[0,10,20,30,40,50,60,70,80,90,100],hue='inc
Out[36]: <seaborn.axisgrid.FacetGrid at 0x1eaf18a3ca0>
```

Observation:

From this plot, we found that there is no clear disparity in income ranges across age.

Pair plot:



Observation:

We used this pair plot as part of our multivariate analysis and observed that features fnlwgt and age are correlated as well as capital_gain and fnlwgt.

All these graphs generated insight for the next step of the project leading us to the start of feature selection.

Feature selection

As we have multiple features, we have used feature selection models to identify the best features relevant to our target to quantify our models. We have picked Correlation matrix, Random Forest Classification as well PCA for this purpose.

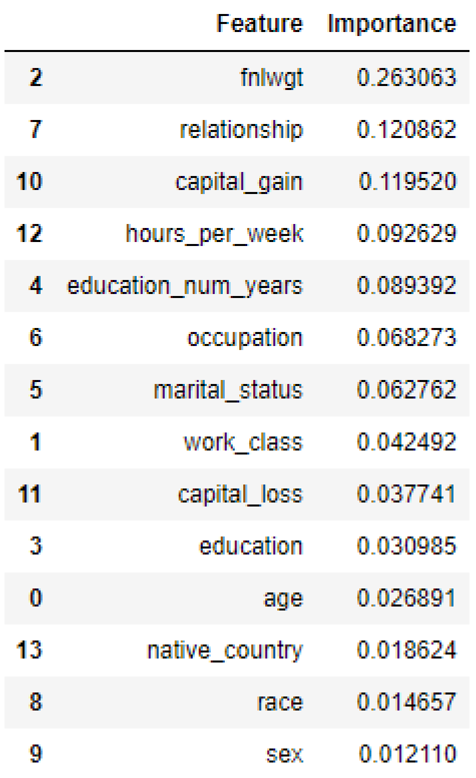
We use the corr() function to calculate the correlation of the features in our dataframe which we later plotted into a heatmap. Using the correlation heatmap, we observed that education_num_years is the most important feature to our target feature Income. Also, hours_per_week and capital_gain are highly correlated to the target feature.

Features like marital_status, relationship as well as fnlwgt were found to be the least correlated to our target feature.

Feature Importance:

We also used the Random Forest Classification algorithm to check for the most relevant features to our target variable.

Feature Importance using the Random Forest Classifier is as displayed below:



	Feature	Importance
2	fnlwgt	0.263063
7	relationship	0.120862
10	capital_gain	0.119520
12	hours_per_week	0.092629
4	education_num_years	0.089392
6	occupation	0.068273
5	marital_status	0.062762
1	work_class	0.042492
11	capital_loss	0.037741
3	education	0.030985
0	age	0.026891
13	native_country	0.018624
8	race	0.014657
9	sex	0.012110

From this method, we found that featuresfnlwgt, capital_gain, relationship and education_num_years are the most important to our target feature.

Features like native_country, sex and race were found to be least important to the target feature.

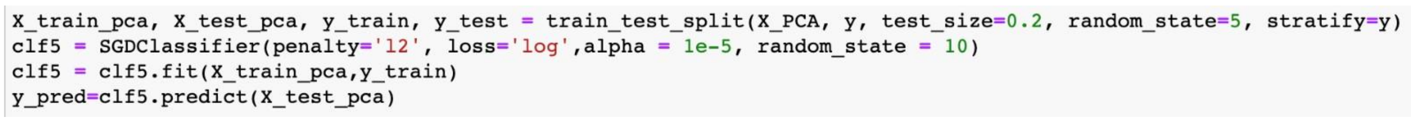
PCA:

We also tried Principal Component Analysis as a Feature Selection method for our models as it is a good tool for multivariate data analysis by summarizing the dataset into a small set of indices.

Cumulative Variances (Percentage):
[21.95762826 38.84223526 55.49999425 68.04039853 78.59576033 85.34754624
 90.82048929]

In this we stated the n_components to be 7 so the algorithm has combined 14 features weight (which excludes the target) into 7 pca components. Later we have used this newly generated components to train the model.

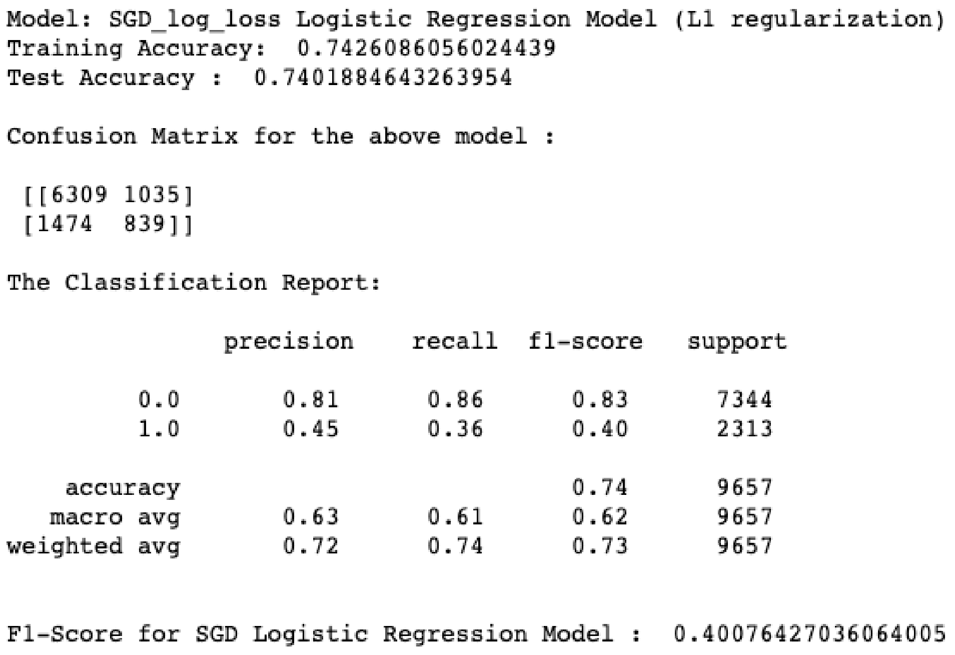
Code:



```
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_PCA, y, test_size=0.2, random_state=5, stratify=y)
clf5 = SGDClassifier(penalty='l2', loss='log',alpha = 1e-5, random_state = 10)
clf5 = clf5.fit(X_train_pca,y_train)
y_pred=clf5.predict(X_test_pca)
```

After the training the SGD model using pca vectors, no significant change in the model's performance was observed.

Results:



```
Model: SGD_log_loss Logistic Regression Model (L1 regularization) :
Training Accuracy:  0.7426086056024439
Test Accuracy :  0.7401884643263954

Confusion Matrix for the above model :

 [[6309 1035]
 [1474  839]]

The Classification Report:

              precision    recall  f1-score   support

         0.0       0.81      0.86      0.83      7344
         1.0       0.45      0.36      0.40      2313

    accuracy                           0.74      9657
   macro avg       0.63      0.61      0.62      9657
weighted avg       0.72      0.74      0.73      9657


F1-Score for SGD Logistic Regression Model :  0.40076427036064005
```

Model Development and Evaluation

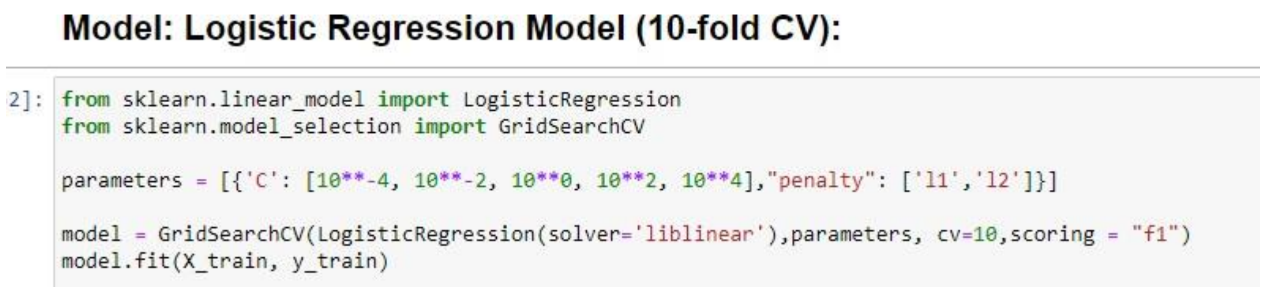
Post successfully identifying the problem as a classification problem, we started researching various techniques that can be used to perform classification which included both vanilla logistic regression as well as advanced algorithms with optimization.

Simple Logistic Regression Mode with Cross Validation:

Having split the data into training and test set with the 80:20 split method, we proceeded with the model development process and selected vanilla logistic regression as our first model to be trained on our training data. Moreover, to improvise and generalize our linear model we trained the model using K-fold cross validation with 10 folds (K=10).

K-fold cross validation: K-fold cross validation means randomly dividing the training data points into “K” folds or parts which are roughly equal in size. The first fold is treated as a validation set, and the model is fitted on the other k − 1 folds.

To perform the cross validation as well as tuning for the parameters of the models to find the best fit for our dataset and the model, we used the Grid Search method of cross validation with 10-folds.



Model: Logistic Regression Model (10-fold CV):

```
2]: from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import GridSearchCV

    parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4],"penalty": ['l1','l2']}]

    model = GridSearchCV(LogisticRegression(solver='liblinear'),parameters, cv=10,scoring = "f1")
    model.fit(X_train, y_train)
```

Grid Search CV trained the model on the parameter grid including the regularization strength hyperparameter as well as the type of penalty and found the best estimator parameters as follows:

LogisticRegression(C=1, penalty='l1', solver='liblinear')

Results:

Confusion Matrix for the above model :

```
 [[6976  368]
 [1343  970]]
```

Accuracy for the logistic regression model for our dataset with the above best parameters was found to be 82 percent.

F-1 Score for the above model was found to be 0.53 which was good but not the best as it indicates misclassification of the points from the minor class.

Optimization Algorithms:

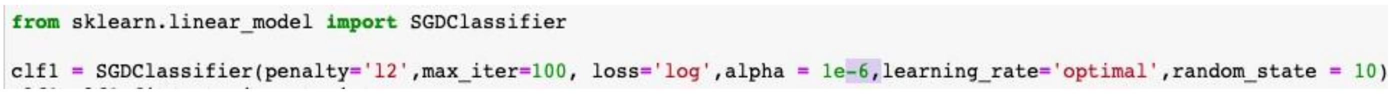
Batch GD with Log Loss 100 epochs:

Gradient Descent: The gradient descent algorithm is a technique in which we train the model parameters on each of the data points and continue training in the direction opposite to the slope increase. This slope is nothing but the derivative of the cost function at that point.

Batch Gradient Descent:

In Batch Gradient descent the model is trained over the entire batch and then the average of the gradients at each point in the batch is calculated to update the parameters after batch training. This entire training of the batch is considered as one epoch.

Code:



```
from sklearn.linear_model import SGDClassifier

clf1 = SGDClassifier(penalty='l2',max_iter=100, loss='log',alpha = 1e-6,learning_rate='optimal',random_state = 10)
```

We trained the model on different parameter values grid including the regularization strength hyperparameter as well as the type of penalty and found the best estimator parameters as follows:

Cost Function = Logistic loss

Penalty="L2", max_iter = 100

alpha(regularization constant) = 1e-6

Results after 100 epochs:

Training Accuracy: 0.81
Test Accuracy: 0.80
Confusion Matrix for the above model :

```
[[6928  416]
 [1367  946]]
```

The confusion matrix shows that the minor class is getting extremely misclassified as the count of TN is very less

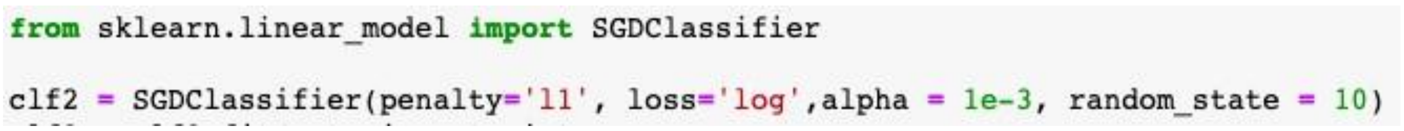
Accuracy for the logistic regression model with Batch Gradient Descent for our dataset was found to be 81 percent on the test data.

F-1 Score for the above model was found to be 0.38 which was good but not the best as it indicates misclassification of the points from the minor class.

Stochastic Gradient Descent:

In Stochastic Gradient descent the model is trained on random points in the dataset and the gradient at that random point in the batch is calculated to update the parameters. This training time of SGD is very less compared to the other gradient descents but there is a chance of missing the global minima as the point is selected randomly.

Code:



```
from sklearn.linear_model import SGDClassifier

clf2 = SGDClassifier(penalty='l1', loss='log',alpha = 1e-3, random_state = 10)
```

We trained the model on different parameter values grid including the regularization strength hyperparameter as well as the type of penalty and found the best estimator parameters as follows:

Cost Function = Logistic loss

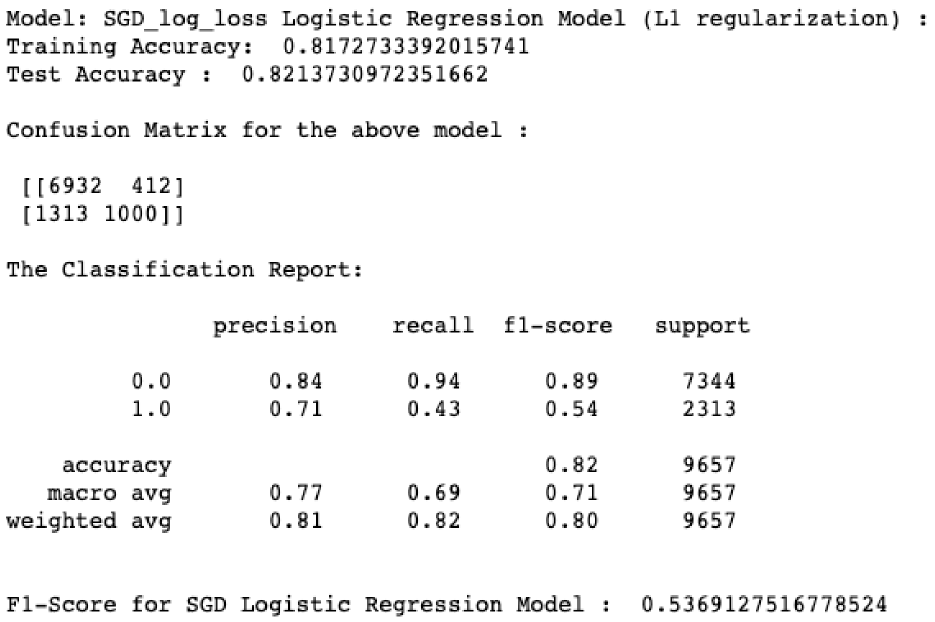
Penalty= “L1”, max_iter = 1000

alpha = 1e-3 (high regularization)

Accuracy for the logistic regression model with Stochastic Gradient Descent for our dataset was found to be 81 percent on the test data.

The F-1 Score for the above model was found to be 0.38 which was good but not the best as it indicates misclassification of the points from the minor class.

Results:



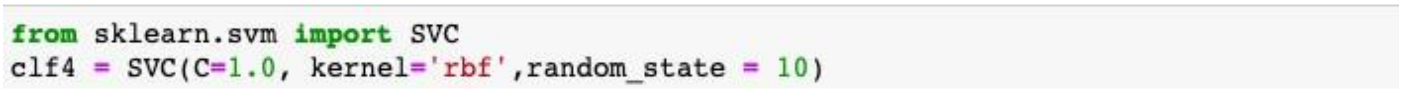
```
Model: SGD_log_loss Logistic Regression Model (L1 regularization) :
Training Accuracy:  0.8172733392015741
Test Accuracy :  0.8213730972351662

Confusion Matrix for the above model :

 [[6932  412]
 [1313 1000]]

The Classification Report:

              precision    recall  f1-score   support

         0.0       0.84      0.94      0.89      7344
         1.0       0.71      0.43      0.54      2313

    accuracy                           0.82      9657
   macro avg       0.77      0.69      0.71      9657
weighted avg       0.81      0.82      0.80      9657


F1-Score for SGD Logistic Regression Model :  0.5369127516778524
```

Support Vector Classifier with RBF Kernel:

Support vector machines generally work on the concept of classifying the points using multi-dimensional hyperplanes.

After observing overfitting in the performance of the linear models on the dataset, decided to use the Support Vector Classifier with RBF kernel which is a non-linear model. The model worked well with the data without overfitting.

Code:

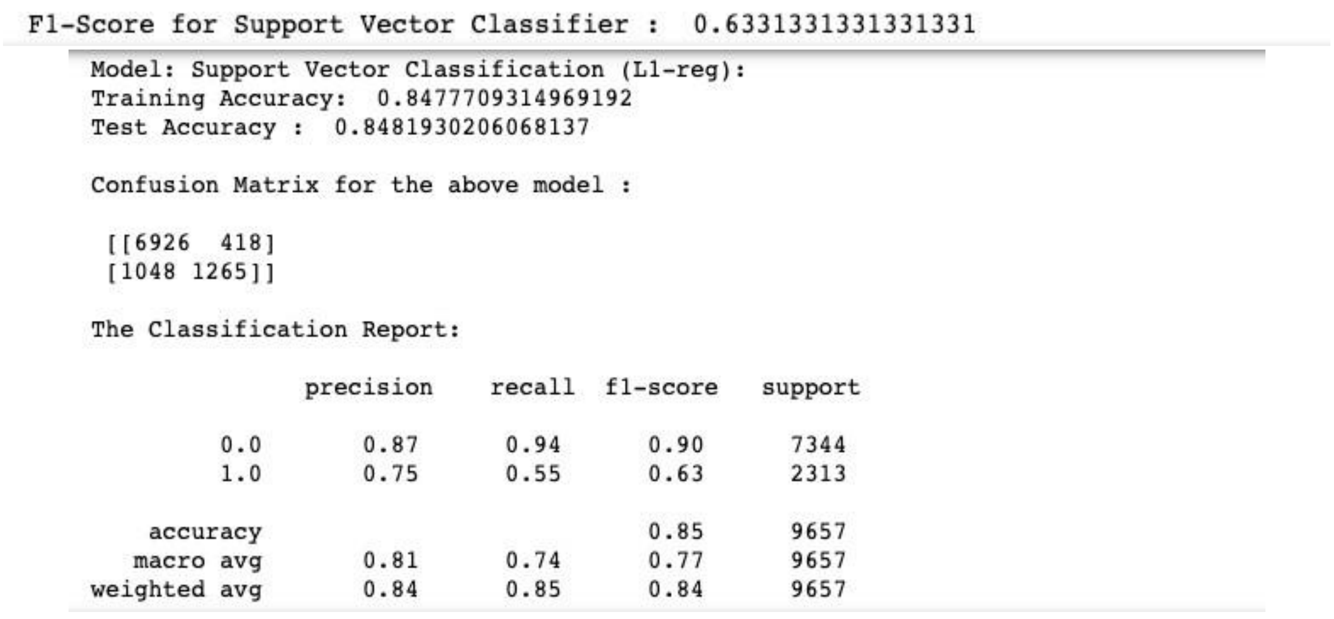


```
from sklearn.svm import SVC
clf4 = SVC(C=1.0, kernel='rbf',random_state = 10)
```

Accuracy for the Support Vector Classifier on our dataset was found to be 84 percent on the test data as well as the training data as we used the non linear RBF kernel.

The F-1 Score for the above model was found to be 0.63 which was the best of all the models with minimal misclassification.

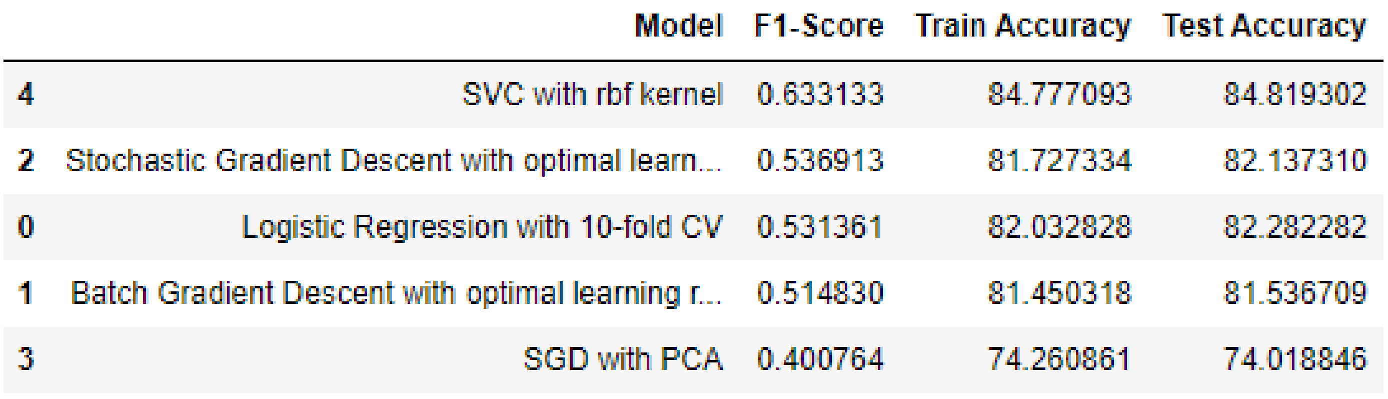
Results:



```
F1-Score for Support Vector Classifier :  0.6331331331331331

Model: Support Vector Classification (L1-reg):
Training Accuracy:  0.8477709314969192
Test Accuracy :  0.8481930206068137

Confusion Matrix for the above model :

 [[6926  418]
 [1048 1265]]

The Classification Report:

              precision    recall  f1-score   support

         0.0       0.87      0.94      0.90      7344
         1.0       0.75      0.55      0.63      2313

    accuracy                           0.85      9657
   macro avg       0.81      0.74      0.77      9657
weighted avg       0.84      0.85      0.84      9657
```

Model Comparison:

Results:



	Model	F1-Score	Train Accuracy	Test Accuracy
4	SVC with rbf kernel	0.633133	84.777093	84.819302
2	Stochastic Gradient Descent with optimal learn...	0.536913	81.727334	82.137310
0	Logistic Regression with 10-fold CV	0.531361	82.032828	82.282282
1	Batch Gradient Descent with optimal learning r...	0.514830	81.450318	81.536709
3	SGD with PCA	0.400764	74.260861	74.018846

Conclusion:

- There is a huge overlap in the datapoints between the two classes and hence it can't be classified precisely with the linear models.
- Training the SGD based models was very fast but the SVC classification model takes comparatively more time to train.
- Accuracy of the SGD and the Batch GD is observed to be for same most of the time even with hyper parameter tuning.
- As per our analysis, we conclude that Support Vector Classifier with RBF kernel has worked best for our problem as it has consistently given a high F1-score and accuracy on our dataset.
- After performing the Logistic Regression and applying optimization methods such as Batch Gradient Descent, Stochastic Gradient Descent and SVC with RBF Kernel, it is observed in the results in the table that the SVC method stands out bringing the best results in terms of F1-score, Train Accuracy and Test Accuracy. In this way, we believe that this is the best optimization that we can apply in Logistic Regression.

References:

1. Census Income · master · Data Science Dojo / datasets · Code
2. Principal Component Analysis (PCA) Explained | Built In
3. analyticsvidhya.com/blog/2022/02/exploratory-data-analysis-in-python/