Northeastern University
College of Engineering

Statistical Learning for Engineers IE 7300
FALL 2023

ABSTRACT:

This project involves the exploration and analysis of the Census Income dataset, extracted from 1994 Census bureau database. The main objective is to build a predictive model which can determine whether an individual income is over $50,000 a year based on various demographic and socioeconomic features.

The dataset contains set of attributes such as age, education, occupation, work class, and more. In this project we focus on implementing machine learning techniques to extract valuable insights from the data and develop an accurate predictive model. The project methodology includes data preprocessing, exploratory data analysis, and feature engineering to enhance the quality of the input data for modeling.

We have implemented various machine learning algorithms to train and evaluate predictive models like Logistic Regression, K-Nearest Neighbor, Support Vector Machine and Gaussian Naive Bayes. Model performance will be assessed using metrics such as accuracy, precision, recall, support and F1 score. The findings and implications of this research can be valuable for policymakers, social scientists, and businesses seeking to understand the dynamics of income distribution in society. Moreover, the project contributes to the broader field of data science and machine learning by providing a practical application of predictive modeling on a real-world dataset.

During this project, we anticipate gaining a comprehensive understanding of the factors influencing income levels and developing an accurate predictive model.

INTRODUCTION:

In the era of data-driven insights, the Census Income dataset serves as a key resource for unraveling socio-economic complexities. Comprising demographic features such as age, education, and occupation, it forms the foundation for a classification problem for predicting income levels. Our aim here is to try and leverage the predictive potential of this information to discern patterns influencing an individual's income, specifically predicting whether it exceeds $50,000 annually.

Problem Statement:

A critical challenge in socio-economic research revolves around the absence of a robust predictive model to find the intricate factors shaping individual income levels. The inability to forecast accurately hampers targeted interventions and informed decision-making, limiting our capacity to address income disparities effectively.

By uncovering hidden patterns within demographic attributes, we hope to contribute to the enhancement of socio-economic decision-making, offering a nuanced approach to address the challenges associated with income disparities.

Data Source:

Link to the UCI repository: https://archive.ics.uci.edu/dataset/20/census+income

Data Description:

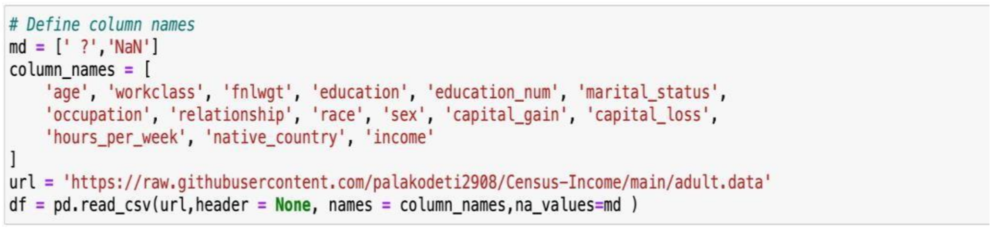
The dataset consists of 14 Features and 32561 Instances.

Variable Name	Role	Type	Missing Values
age	Feature	Integer	No
workclass	Feature	Categorical	Yes
fnlwgt	Feature	Integer	No
education	Feature	Categorical	No
education-num	Feature	Integer	No
marital-status	Feature	Categorical	No
occupation	Feature	Categorical	Yes
relationship	Feature	Categorical	No
race	Feature	Categorical	No
sex	Feature	Binary	No
capital-gain	Feature	Integer	No
capital-loss	Feature	Integer	No
hours-per-week	Feature	Integer	No
native-country	Feature	Categorical	Yes
income	Feature	Target	No

The dataset encompasses diverse census data elements related to individuals. With 15 columns, it includes both categorical and numerical features, culminating in a target variable predicting income levels. The features are as follows:

1. **Age**: Represents the age of the individual.
2. **Workclass**: Describes the type of employment.
3. **fnlwgt**: Stands for final weight, an estimate of the number of people the census believes the entry represents.
4. **Education**: Indicates the highest level of education attained
5. **Education-num:** Represents the number of years of education completed.
6. **Marital Status**: Describes the marital status of the individual.
7. **Occupation**: Specifies the type of occupation
8. **Relationship**: Describes the relationship status in the family
9. **Race**: Represents the individual's race.
10. **Sex**: Specifies the gender of the individual.
11. **Capital-Gain:** Represents the capital gains for the individual.
12. **Capital-Loss:** Represents the capital losses for the individual.
13. **Hours-per-Week**: Indicates the average number of hours worked per week.
14. **Native-Country**: Denotes the native country of the individual.
15. **Income**: Target variable indicating whether the individual's income exceeds $50,000 per year (">50K") or not ("<=50K").

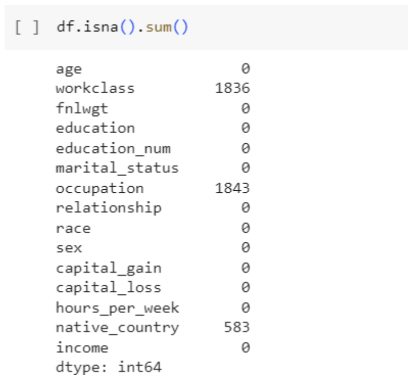
Firstly, we are defining the column names before reading the data as the dataset



```
# Define column names
md = [' ?','NaN']
column_names = [
    'age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status',
    'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss',
    'hours_per_week', 'native_country', 'income'
]
url = 'https://raw.githubusercontent.com/palakodeti2908/Census-Income/main/adult.data'
df = pd.read_csv(url,header = None, names = column_names,na_values=md )
```

Data Pre-processing: -

1. **Missing Values**

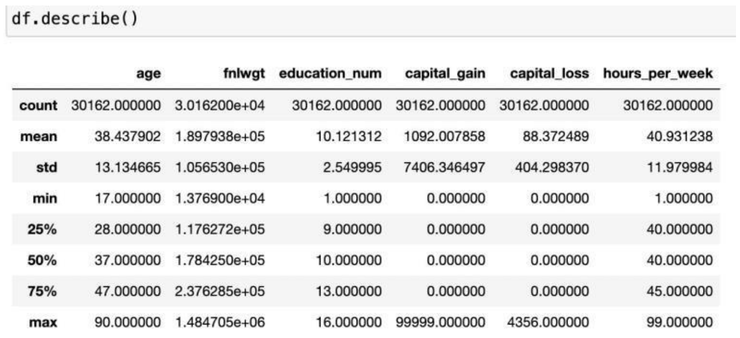


```
[ ]  df.isna().sum()

     age                  0
     workclass         1836
     fnlwgt               0
     education            0
     education_num        0
     marital_status       0
     occupation        1843
     relationship         0
     race                 0
     sex                  0
     capital_gain         0
     capital_loss         0
     hours_per_week       0
     native_country     583
     income               0
     dtype: int64
```

Occupation contains the largest number of null values in the dataset followed by workclass and native-country, and we have handled this missing data by dropping the columns containing the missing data and the result dataset has 30139 rows.

Overall description of the data:-

Data description provides various statistics summary of the numerical features in the dataset. The mean and standard deviation of 'hours-per-week' is 40.93 and 11.97 respectively, and min and max of 'age' is 17.00 and 90.00 .

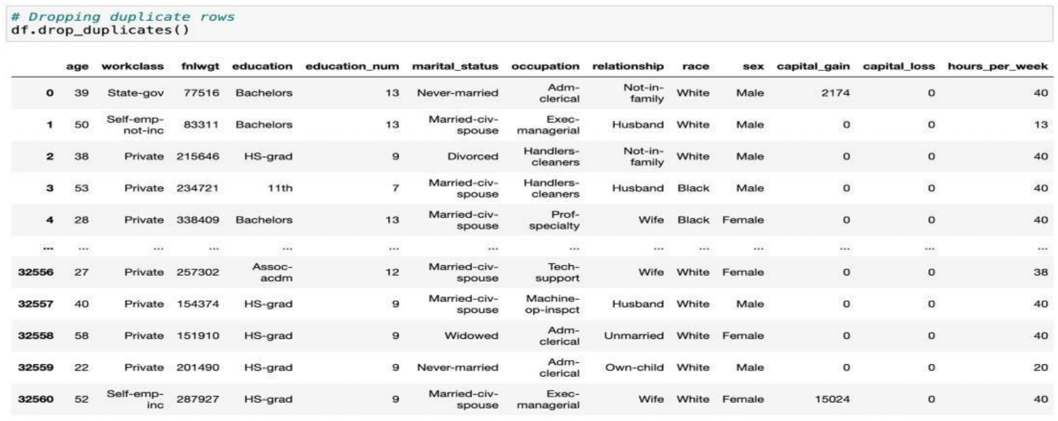


```
df.describe()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
count	30162.000000	3.016200e+04	30162.000000	30162.000000	30162.000000	30162.000000
mean	38.437902	1.897938e+05	10.121312	1092.007858	88.372489	40.931238
std	13.134665	1.056530e+05	2.549995	7406.346497	404.298370	11.979984
min	17.000000	1.376900e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.176272e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.784250e+05	10.000000	0.000000	0.000000	40.000000
75%	47.000000	2.376285e+05	13.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

2. **Duplicate data: -**

The dataset contains a few duplicate data and we have dropped them and the final dataset left with 30139 rows.



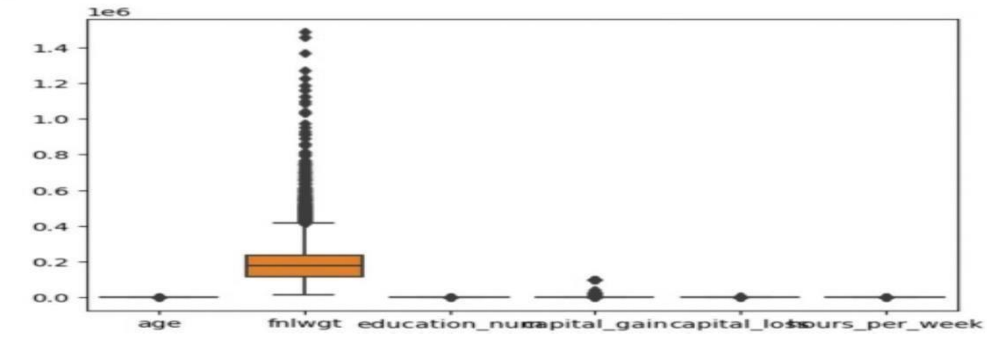
```
# Dropping duplicate rows
df.drop_duplicates()
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40

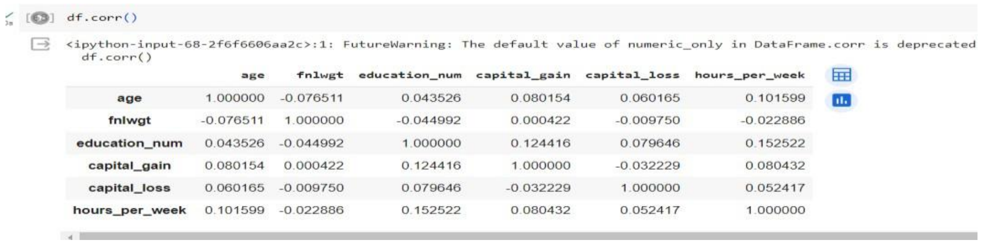
3. **Outliers: -**

Plotting box plot that describes the shape on how the data is distributed with the outliers in each column

```
sns.boxplot(data=df)
plt.show()
```



4. **Correlation between Numeric Columns:-**



```
df.corr()
```

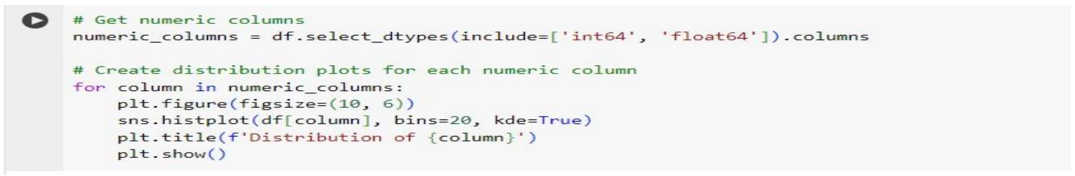
```
<ipython-input-68-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated
  df.corr()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
age	1.000000	-0.076511	0.043526	0.080154	0.060165	0.101599
fnlwgt	-0.076511	1.000000	-0.044992	0.000422	-0.009750	-0.022886
education_num	0.043526	-0.044992	1.000000	0.124416	0.079646	0.152522
capital_gain	0.080154	0.000422	0.124416	1.000000	-0.032229	0.080432
capital_loss	0.060165	-0.009750	0.079646	-0.032229	1.000000	0.052417
hours_per_week	0.101599	-0.022886	0.152522	0.080432	0.052417	1.000000

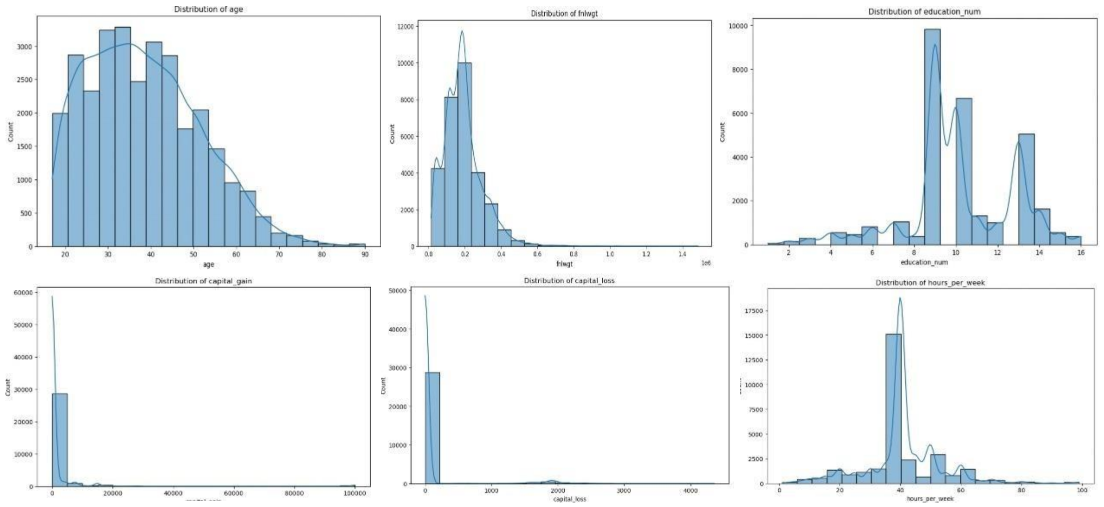
Correlation describes the relation between the numeric features in the dataset. The feature 'fnlwgt' has a negative correlation with other numerical features except 'capital gain' . Also, the maximum correlation among the features is between ‛hours-per-week' and ‛education-num'.

Data Analysis:-

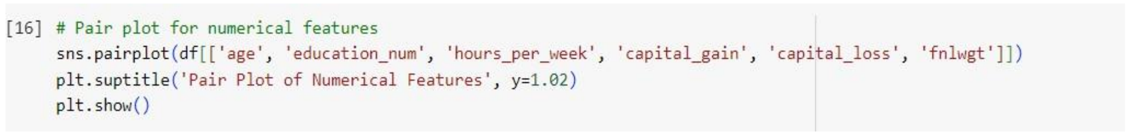
1. **Distribution hist plot: -**
   We have created separate histogram plots for each numeric feature providing the insights into the distribution of values for each variable.



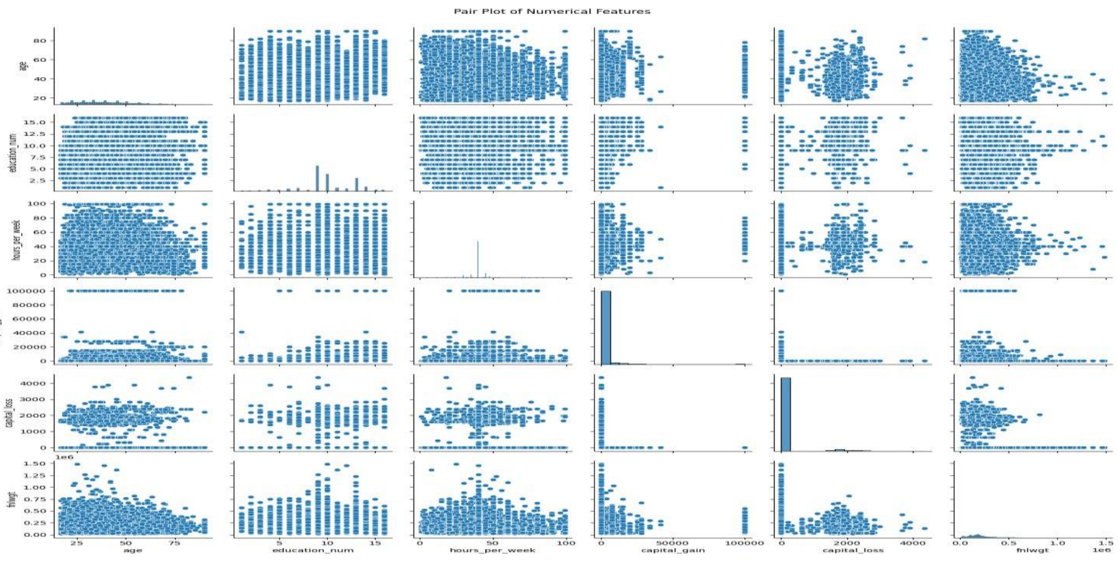
```
# Get numeric columns
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns

# Create distribution plots for each numeric column
for column in numeric_columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(df[column], bins=20, kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()
```



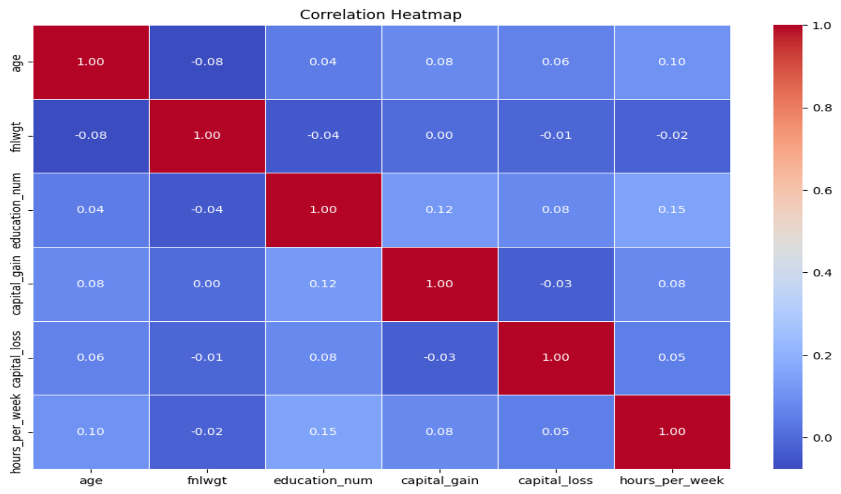
2. **Pair Plot: -**



```
[16] # Pair plot for numerical features
     sns.pairplot(df[['age', 'education_num', 'hours_per_week', 'capital_gain', 'capital_loss', 'fnlwgt']])
     plt.suptitle('Pair Plot of Numerical Features', y=1.02)
     plt.show()
```



3. **Correlation Matrix:-** A Correlation Matrix in the form of a Heatmap illustrating Correlations between features and labels, where all features are continuous variables.



1. At the time, only around one-third of the population was deemed high-income, while the other two-thirds earned less than $50,000 USD each year. The highest share of Asians were earning more over $50,000, with the white class following closely after.

2. Capital Gain was a solid measure of wealth, with a noticeable distinction between persons earning more than $50,000 and those with larger capital gains, indicating that the wealth divide in the United States is widening.

3. Capital loss was a mix of both high- and low-income persons and was not a clear measure of wealth.

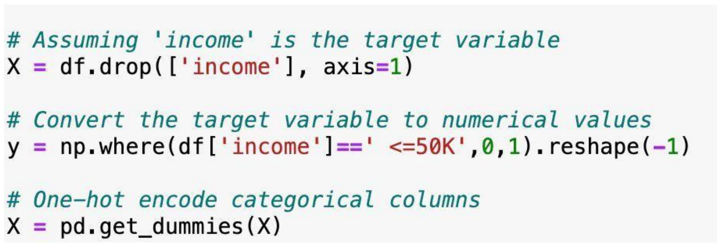
4. Education was a good predictor of income, with the majority of high-income persons completing a pHD, Masters, or Bachelor's degree. The vast majority of the population had completed either high school or some college.

5. Married persons had the largest percentage of high-income people, with husbands comprising the bulk of the workforce. In 1994, the male labor market more than doubled that of women.

6. The male dominant job was Craft-repair, while the female dominant position was Administrative-clerical.

Data Splitting: -

The code separates the dataset into features (X) and target variables (y). The target variable is 'income', and the features are produced by removing the column 'income' from the original Data frame. The target variable 'income' is converted to numerical numbers. If 'income' is '<=50K', it is encoded as 0, whereas '>50K' is encoded as 1. This binary encoding is commonly used in binary classification tasks. The categorical columns in the feature set (X) are encoded once using pd.get_dummies(). This is implemented to transform categorical data into a format that machine learning models can use with numerical input.



```
# Assuming 'income' is the target variable
X = df.drop(['income'], axis=1)

# Convert the target variable to numerical values
y = np.where(df['income']==' <=50K',0,1).reshape(-1)

# One-hot encode categorical columns
X = pd.get_dummies(X)
```

The code selects the top 90 columns using onehot encoding.This might be useful if you have an extensive amount of columns and need to work with a subset for modeling or analysis.

The prepared dataset (X) is now ready to train machine learning models, complete with numerical features and a binary-encoded target variable.

Onehot encoding guarantees that categorical variables are represented.
The code is frequently used as part of the preparation stage in a machine learning pipeline, wher e input is processed to be compatible with various algorithms.

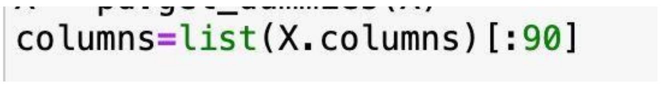
Feature Engineering:-

Dimensionality reduction technique is done by using **Principal Component Analysis(PCA).** Since there are 104 columns after splitting the data, now we need to reduce the number of features in training the dataset from prediction models implementation.

The 'class PCA' is defined to encapsulate the PCA functionality. It has methods for fitting the PCA model (fit) and transforming the data (transform). The _init_method initializes the PCA object with the number of principal components (n) that you want to retain. The fit method takes a dataset (X) and calculates the covariance matrix, eigenvalues, and eigenvectors.

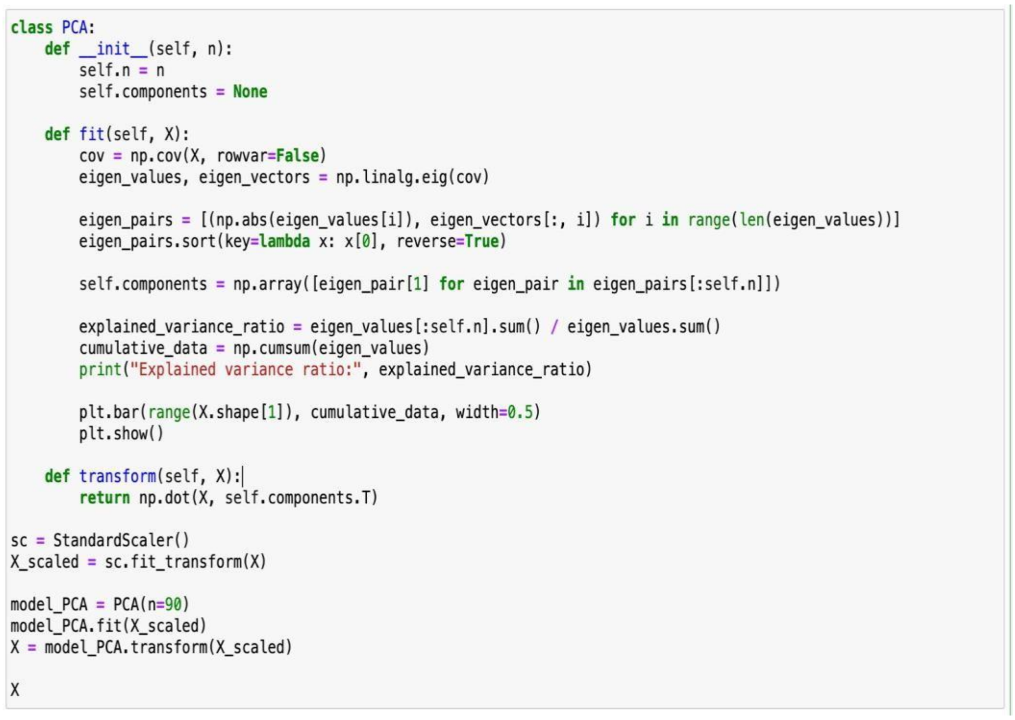
It sorts the eigenvalue-eigenvector pairs in descending order based on eigenvalues.

The transform method takes a dataset (X) and projects it onto the selected principal components. Before applying PCA, the feature matrix (X) is standardized using StandardScaler to have zero mean and unit variance. An instance of the PCA class is created with n=90, indicating that you want to retain 90 principal components. The fit method is called with the scaled data (X_scaled).



```
columns=list(X.columns)[:90]
```

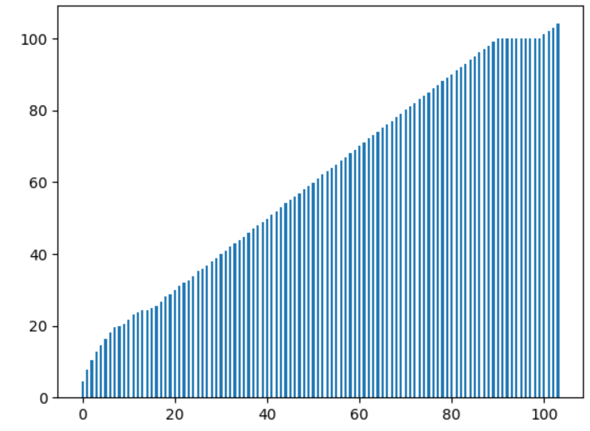
Based on the cumulative explained variance ratio graph below, we are getting 95.19% of variance ratio. Hence we are now continuing with 90 columns.



```python
class PCA:
    def __init__(self, n):
        self.n = n
        self.components = None

    def fit(self, X):
        cov = np.cov(X, rowvar=False)
        eigen_values, eigen_vectors = np.linalg.eig(cov)

        eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:, i]) for i in range(len(eigen_values))]
        eigen_pairs.sort(key=lambda x: x[0], reverse=True)

        self.components = np.array([eigen_pair[1] for eigen_pair in eigen_pairs[:self.n]])

        explained_variance_ratio = eigen_values[:self.n].sum() / eigen_values.sum()
        cumulative_data = np.cumsum(eigen_values)
        print("Explained variance ratio:", explained_variance_ratio)

        plt.bar(range(X.shape[1]), cumulative_data, width=0.5)
        plt.show()

    def transform(self, X):
        return np.dot(X, self.components.T)

sc = StandardScaler()
X_scaled = sc.fit_transform(X)

model_PCA = PCA(n=90)
model_PCA.fit(X_scaled)
X = model_PCA.transform(X_scaled)

X
```

Plotting the explained variance ratio gaph:

```
X
Explained variance ratio: 0.9519395674103263
```



```
array([[ 0.51821068, -2.44239648, -0.59109703, ...,  0.4011028 ,
         0.32538723,  0.4421365 ],
       [-2.84879264, -1.42727303, -0.0632561 , ..., -1.03830454,
         1.35312398,  0.12224899],
       [ 0.80097833,  0.87338866, -1.20573024, ..., -0.05964325,
        -0.28974481,  0.48282364],
       ...,
       [ 2.36158276, -0.85860917,  0.30270404, ..., -0.57082553,
        -0.31712288,  0.3332339 ],
       [ 1.9797012 ,  1.1805689 , -1.83205078, ..., -0.49360755,
        -0.04132791,  0.22015827],
       [-0.74518025, -1.717621  ,  0.56118951, ..., -0.56669104,
         0.51916899, -0.22159945]])
```

The explained variance ratio is the percentage of the dataset's total variance explained by every principal components. A high explained variance ratio for a primary component means that it retains an extensive amount of information from 'X' . The cumulative explained variance ratio measures how much variability in the data is captured as we add more main components.

Models that we have implemented for this project:

1. **Logistic Regression**
2. **Naïve Bayes**
3. **Support Vector Machine**
4. **K-Nearest Neighbor**

Approach: -

Since our data requirement is classification, we have tested the traditional models discussed that were discussed in class. For this purpose, **we used Naïve Bayes as our baseline model.**

Model description: -

1. **Naïve Bayes Model Description: -**

Naïve Bayes Algorithm uses simple Bayesian Formula to predict the probability of each class label. It is a type of generative classification. By using Bayesian Formula, we estimate the probability of the sample belonging to each class.

We start by assuming that each feature is independent of all other features in that specific class. A simple naïve match estimates the probability of each feature value.

If the probability of feature 1 is P(X1) then the likelihood according to the independence assumption of each sample is given by the product of the individual probabilities of each feature in that specific class.

Likelihood = $P(x1|y=k). P(x2|y=k). P(x3|y=k)\ldots\ldots$

According to the bayes theorem the conditional probability of x belonging to a specific class is given by.

$$P(y=k|x=x)= \text{likelihood}*\text{prior of the class}$$

Where prior is the probability of the class k = $P(y=k)$

$$P(y=k|x=x)= P(x1|y=k). P(x2|y=k). P(x3|y=k)\ldots\ldots* P(y=k)$$

This is repeated for each class to find the probability of X belonging to each class

The following logic calculates the likelihood

```
if class_value==0:
    if actual_value==0:
        self.likelyhood_list_class0_value0.append((len(self.X0_train[:index][self.X0_train[:index]==0])+
                                                   alpha)/(len(self.X0_train)+(m*alpha)))
    if actual_value==1:
        self.likelyhood_list_class0_value1.append((len(self.X0_train[:index][self.X0_train[:index]==1])+
                                                   alpha)/(len(self.X0_train)+(m*alpha)))
else :

    if actual_value==0:
        self.likelyhood_list_class1_value0.append((len(self.X1_train[:index][self.X1_train[:index]==0])+
                                                   alpha)/(len(self.X1_train)+(m*alpha)))
    if actual_value==1:
        self.likelyhood_list_class1_value1.append((len(self.X1_train[:index][self.X1_train[:index]==1])+
                                                   alpha)/(len(self.X1_train)+(m*alpha)))
```

Calculating Priors:-

```
self.prior_y0 = len(self.X0_train)/len(self.X_train)
self.prior_y1 = len(self.X1_train)/len(self.X_train)
```

While training the model, we keep track of all the priors and the likelihood of each training sample.

Now, we predict the class labels by using these tracked priors and the likelihoods.

```
for index in np.arange(0,len(self.X_train.columns)-6):
    # sample belongs to class 0
    if list(sample[1])[index] == 1:
        class_0_probability = class_0_probability*self.likelyhood_list_class0_value1[index]
    else:
        class_0_probability =class_0_probability*self.likelyhood_list_class0_value0[index]

# sample belongs to class 1

    if list(sample[1])[index] == 1:
        class_1_probability = class_1_probability*self.likelyhood_list_class1_value1[index]
    else:
        class_1_probability =class_1_probability* self.likelyhood_list_class1_value0[index]
```

The class label is then given by whichever probability is maximum.

```
py0 = class_0_probability*self.prior_y0
py1 = class_1_probability*self.prior_y1

if py0>py1:
    self.prediction_list.append(0)
else:
    self.prediction_list.append(1)
```

2. Logistic regression: -

This is a kind of Discriminative classification model where the probability given by the model is odds of how likely the sample belongs to the class.

In discriminative classification, the sum of probabilities of all the class labels is equal to 1.

∑P(y=k|X) = 1

We train the model using sigmoid transformation function. And the sigmoid is given by

1/(1+e$^{-X(theta)}$) = this sigmoid will give the probability value.

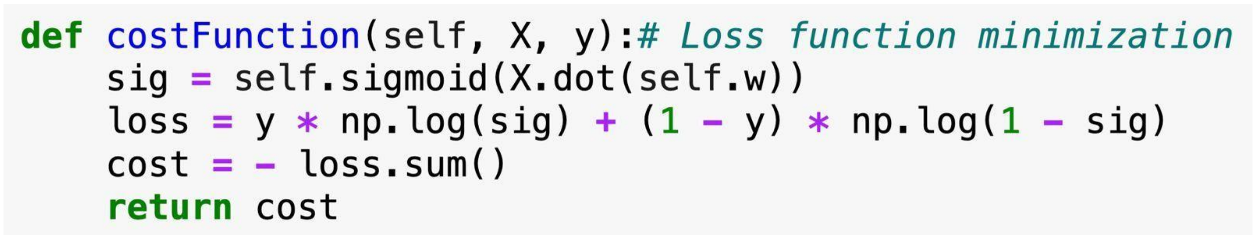
As our data is a binary classification, calculating the probability of one class label will automatically give the probability of the other class labels.

P(y=0) = 1- p(y=1)

For this algorithm, we train the model using by minimizing cost function which is log loss.

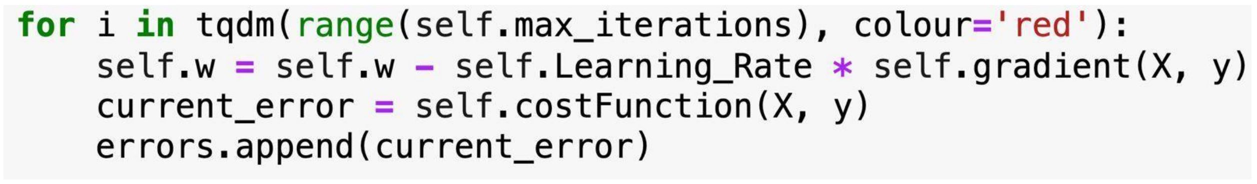
Log loss function is given by -**∑(ylog(p) + (1-y) log(1-p)).** This is achieved by using gradient decent.

We start by assuming theta's and calculating the probability by sigmoid then calculating the loss using log loss.



```
def costFunction(self, X, y):# Loss function minimization
    sig = self.sigmoid(X.dot(self.w))
    loss = y * np.log(sig) + (1 - y) * np.log(1 - sig)
    cost = - loss.sum()
    return cost
```

Based on the loss, we update the theta's using the provided learning rate.



```
for i in tqdm(range(self.max_iterations), colour='red'):
    self.w = self.w - self.Learning_Rate * self.gradient(X, y)
    current_error = self.costFunction(X, y)
    errors.append(current_error)
```

This is repeated for each iteration. Then the theta's for the log loss is chosen as the optimum theta. Now new predictions are performed using 1/(1+e$^{-X(theta)}$) with the optimum thetas and the test data X.

Because logistic regression is a distance-based algorithm, it is necessary to standardize the data before training it. We used StandardScalar which scales the data based on the following formula.

X_Scaled = X - µ/sigma (where µ is the mean of data and sigma is the standard deviation)

```python
def Normalize_data(self,X):
    mean=np.mean(X,axis=0)
    std=np.std(X,axis=0)
    X=(X-mean)/std
    X=self.addX0(X)
    return X,mean,std

def normalize_test(self,X,mean,std):
    X=(X-mean)/std
    X=self.addX0(X)
    return X
```

As we can see from the above code, that we have used the same mean and S.D to normalize the test data as well. This is done to prevent information leakage.

3. **Support Vector Machine (Hard Margin):-**

The objective of this model is to identify the hard separation layer between classes.

We start by assuming that the samples in a specific class are at least a scaled distance of 1 from the separation layer. The projection distance of sample X from the boundary line is given as **wx+b** where w and x are vectors. x is a sample vector. As the distance is positive and negative for each class, we take y(wx+b) as mod distance where y can be either -1 or 1. Y=1 is for the success class and -1 for the other.

The objective function which needs to be maximized is given by hinge loss.

Hinge loss for SVM = J(w) = lambda*||w||^2 + 1/n*sum_{i=1}^{n}max(0, 1-y_i(w.x_i + b))+

Lamda is regularization factor over w's.

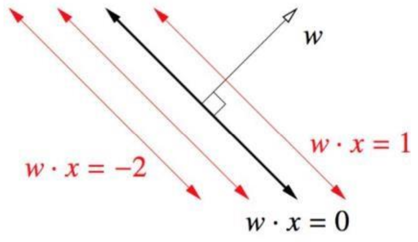
If we look at hinge loss, the values of the loss, if the distance y(wx+b) >1 is 0 and if y(wx+b)<1 it is y(wx+b).

We check for this condition at every step and modify the cost function accordingly.

```python
condition = (y_[idx] * (np.dot(x_i, self.w) + self.b)) >= 1
if condition:
    self.w -= 2 * self.lr * self.lambda_param * self.w
else:
    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
    self.b -= -y_[idx]
```

Now we follow the same steps that we have followed for gradient decent by assuming w's and b.

Graphical Behavior of SVM:-



The predictions are obtained by substituting the values of w's and b in wx+b and checking if wx+b is positive (+1 class) and negative (-1 class).

4. K Nearest Neighbor:-

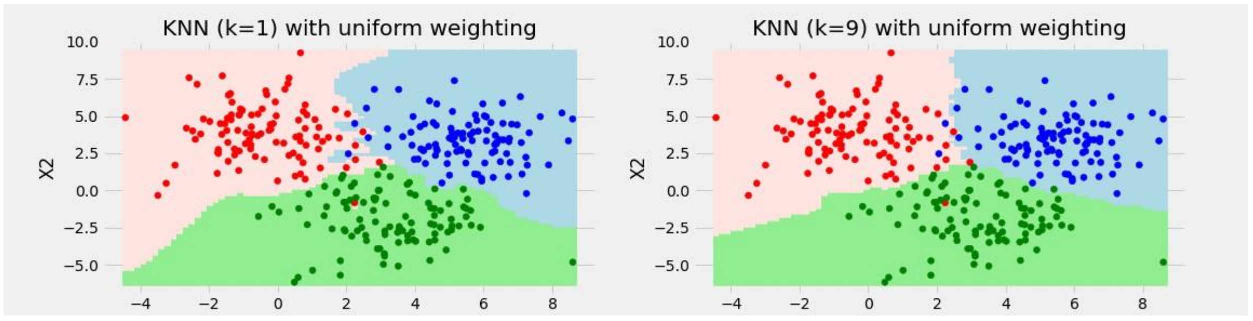
This is a clustering algorithm which used Euclidian distance between data points to assign the label which has the majority of k number of class labels which are closed to the datapoint under consideration. The Euclidian distance logic is given by

```
def euclidean_dist(self,pointA, pointB):
    distance = np.square(pointA - pointB) # (ai-bi)**2 for every point in the vectors
    distance = np.sum(distance) # adds all values
    distance = np.sqrt(distance)
    return distance
```

We calculate the distance using the above logic from the test data point to all the datapoints in the dataset. Now we isolate the top k points which are close to test data point. We observe the class of all the top k data points and assign the labels which is majority of all the datapoints.

```
neighbors_sorted = neighbors[neighbors[:, 0].argsort()]  # sorts training points on the basis of distance
k_neighbors = neighbors_sorted[:k] # selects k-nearest neighbors
frequency = np.unique(k_neighbors[:, 1], return_counts=True)
target_class = frequency[0][frequency[1].argmax()] # selects label with highest frequency
predictions = np.append(predictions, target_class)
```

KNN is one of the algorithms which requires no prior training for it's modelling



The above is the sample KNN behavior.

Results and observations: -

As mentioned above, we have considered **Naïve bayes** as the **baseline model**. Results are as follows

Naïve bayes: -

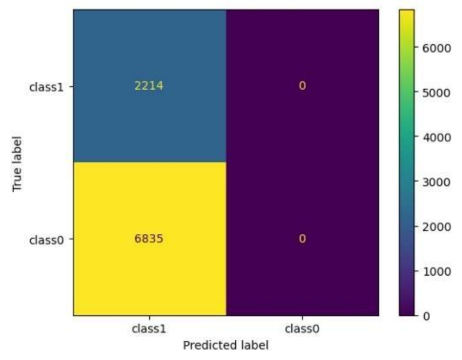
Based on evaluating the naïve Bayes algorithm over categorical features, the obtained evaluation metrics are tabulated below

```
fl = np.array(model.prediction_list)
accuracy_train(model.y_test,fl)
```

```
y_preds:  [1 1 1 ... 1 1 1]
y_preds shape:  (9049,)
misclassification rate 0.7553320808929164
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      6835
           1       0.24      1.00      0.39      2214

    accuracy                           0.24      9049
   macro avg       0.12      0.50      0.20      9049
weighted avg       0.06      0.24      0.10      9049

Train_Time:  0:00:18.077434
```

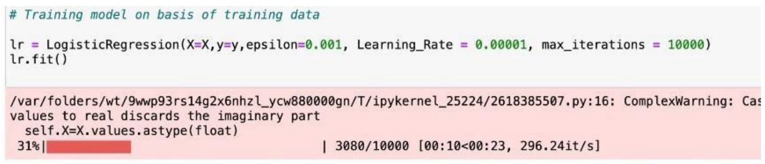
As we can see that accuracy is 24%, the model predicted each entry value to be 1. The precision for class label 0 is 0. This might be due to class imbalance issue in the training data and it could mean that the categorical columns that are used to train the data had little to no impact in model predictions.

```
cm = confusion_matrix(model.y_test,fl, labels=[1,0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["class1","class0"])
disp.plot()
plt.show()
```



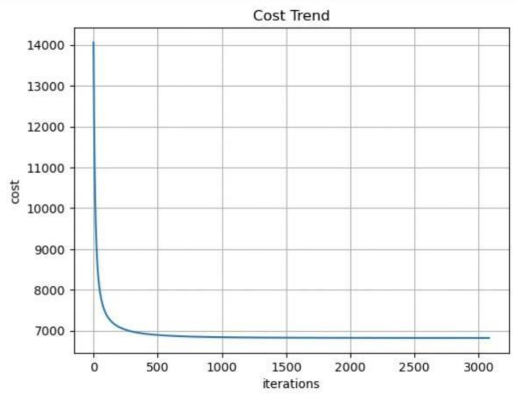
This model serves as an explanation that class imbalance is the major issue which makes this model unsuitable for modeling demographic data

Logistic Regression: -

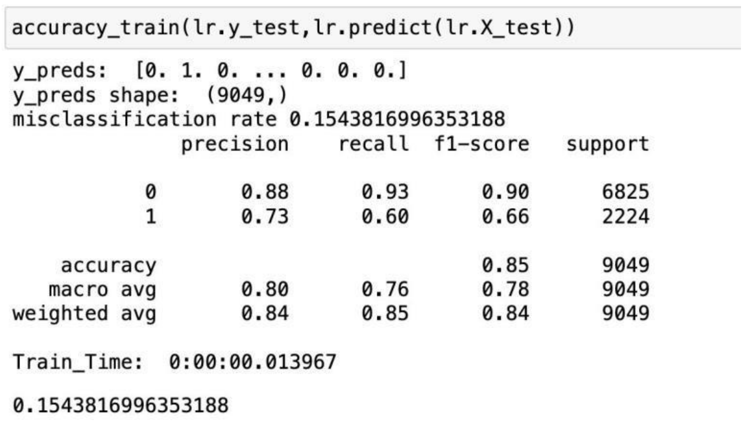


```
# Training model on basis of training data

lr = LogisticRegression(X=X,y=y,epsilon=0.001, Learning_Rate = 0.00001, max_iterations = 10000)
lr.fit()
```

```
/var/folders/wt/9wwp93rs14g2x6nhzl_ycw880000gn/T/ipykernel_25224/2618385507.py:16: ComplexWarning: Cas
values to real discards the imaginary part
  self.X=X.values.astype(float)
 31%|                                    | 3080/10000 [00:10<00:23, 296.24it/s]
```



Training the dataset with logistic regression gave the best results owing to the presence of non-categorical columns in the training sets. The above graph shows the cost to iteration trend while training the algorithm. As we can see in the graph the bias variance trade off quick at approx. 300 iterations post which the costs seem to stabilize. The convergence has been achieved faster with respect to the epsilon value of 0.001which is used to train the algorithm. This signifies the importance of odds of non-categorical columns which influence the model prediction.

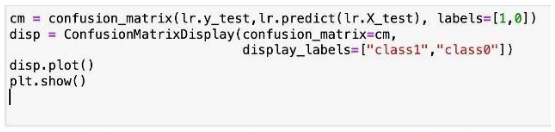


```
accuracy_train(lr.y_test,lr.predict(lr.X_test))
```

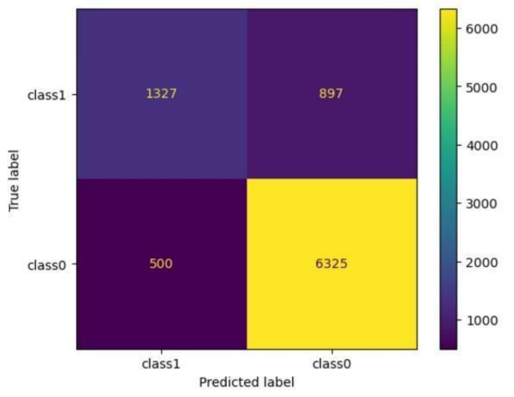
```
y_preds:  [0. 1. 0. ... 0. 0. 0.]
y_preds shape:  (9049,)
misclassification rate 0.1543816996353188
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      6825
           1       0.73      0.60      0.66      2224

    accuracy                           0.85      9049
   macro avg       0.80      0.76      0.78      9049
weighted avg       0.84      0.85      0.84      9049

Train_Time:  0:00:00.013967

0.1543816996353188
```

The tabulated metrics show the class imbalance issue prevalent in the training data does not hinder the accuracy of the prediction. As observed from the table, the model is more efficient in predicting the class label 0 compared to class 1.

In recent times, universities have conducted studies using logistic regression to model census data and understand the factors contributing to income inequality. Researchers may explore how demographic characteristics influence the probability of individuals belonging to different income groups.



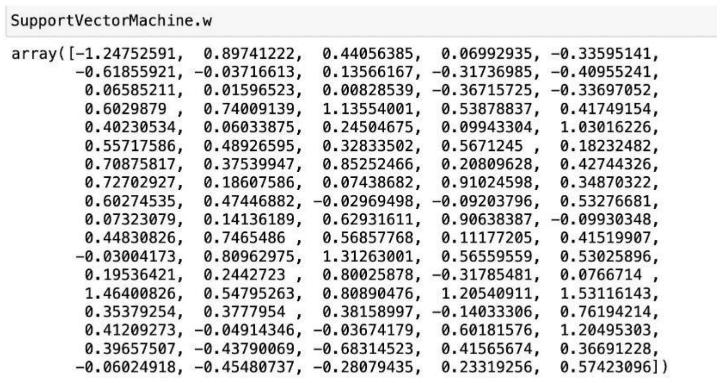
```
cm = confusion_matrix(lr.y_test,lr.predict(lr.X_test), labels=[1,0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["class1","class0"])
disp.plot()
plt.show()
```



Based on the confusion metrics, the instances of income which is less than $50,000 has been predicted up to a fair level of accuracy which helps us to determine the resource allocation to under privileged sector of population making this model efficient in predicting the practical scenarios.

Support Vector Machine (Hard Margin): -

We have applied this model assuming that there might be clear linear separation between the classes based on the given demographic features.



```
SupportVectorMachine.w

array([-1.24752591,  0.89741222,  0.44056385,  0.06992935, -0.33595141,
       -0.61855921, -0.03716613,  0.13566167, -0.31736985, -0.40955241,
        0.06585211,  0.01596523,  0.00828539, -0.36715725, -0.33697052,
        0.6029879 ,  0.74009139,  1.13554001,  0.53878837,  0.41749154,
        0.40230534,  0.06033875,  0.24504675,  0.09943304,  1.03016226,
        0.55717586,  0.48926595,  0.32833502,  0.5671245 ,  0.18232482,
        0.70875817,  0.37539947,  0.85252466,  0.20809628,  0.42744326,
        0.72702927,  0.18607586,  0.07438682,  0.91024598,  0.34870322,
        0.60274535,  0.47446882, -0.02969498, -0.09203796,  0.53276681,
        0.07323079,  0.14136189,  0.62931611,  0.90638387, -0.09930348,
        0.44830826,  0.7465486 ,  0.56857768,  0.11177205,  0.41519907,
       -0.03004173,  0.80962975,  1.31263001,  0.56559559,  0.53025896,
        0.19536421,  0.2442723 ,  0.80025878, -0.31785481,  0.0766714 ,
        1.46400826,  0.54795263,  0.80890476,  1.20540911,  1.53116143,
        0.35379254,  0.3777954 ,  0.38158997, -0.14033306,  0.76194214,
        0.41209273, -0.04914346, -0.03674179,  0.60181576,  1.20495303,
        0.39657507, -0.43790069, -0.68314523,  0.41565674,  0.36691228,
       -0.06024918, -0.45480737, -0.28079435,  0.23319256,  0.57423096])
```

These are the weights obtained for the features trained using SVM.

```
SupportVectorMachine.b
```

```
-4
```

The SVM results obtained are with regularization factor of lambda = 0.001 over norm 2 of weights. Proper care is taken to select the sample required to train ensuring the normal distribution which is prevalent in demographic data. So, the sample between 12,000 and 14,000 is chosen to train the data. As SVM is a non-parametric model, training SVM with large amount of data like census might increase the time complexity. Hence optimum ranges of samples are chosen while training the model. 500 iterations are chosen for this.

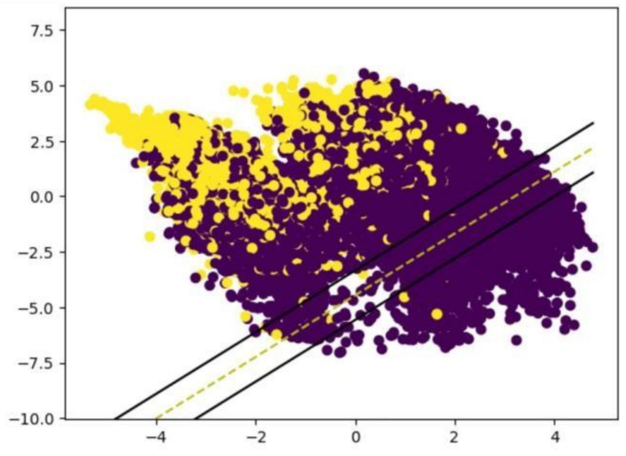
```
SupportVectorMachine.accuracy_test(SupportVectorMachine.predict)
```

```
misclassification rate 0.174
testing error 17.4
              precision    recall  f1-score   support

           0       0.82      0.99      0.90      1510
           1       0.92      0.32      0.47       490

    accuracy                           0.83      2000
   macro avg       0.87      0.65      0.68      2000
weighted avg       0.84      0.83      0.79      2000
```

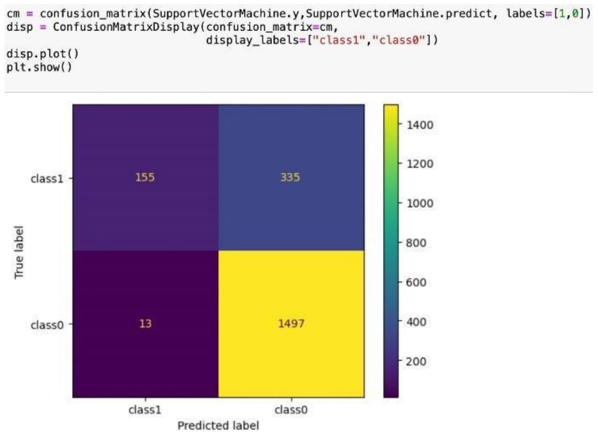
This model gave a high precision value for class label 1 compared to class label 0 making it efficient in detecting higher income ratio of the demography. This is the SVM view in 2D.

```
visualize_svm( X.iloc[:,:2],y,SupportVectorMachine)
```

```
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:196: ComplexWarning:
Casting complex values to real discards the imaginary part
  offsets = np.asanyarray(offsets, float)
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/cbook/__init__.py:1298: ComplexWarn
ing: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:2877: ComplexWarning:
Casting complex values to real discards the imaginary part
  vmin, vmax = map(float, [vmin, vmax])
```

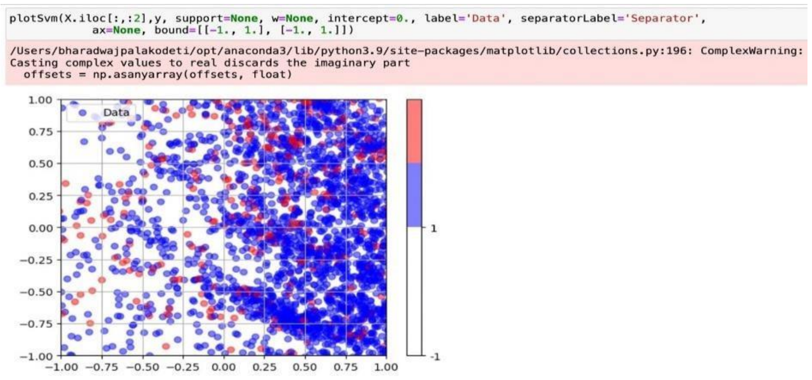


Below is the confusion matrix.



```
cm = confusion_matrix(SupportVectorMachine.y,SupportVectorMachine.predict, labels=[1,0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["class1","class0"])
disp.plot()
plt.show()
```

Below is the density of the sample chosen for SVM model.

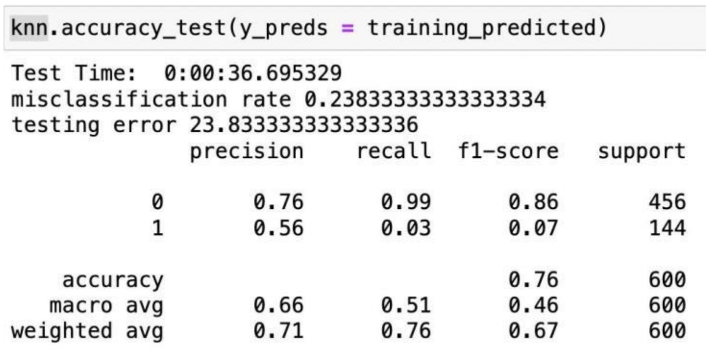


```
plotSvm(X.iloc[:,:2],y, support=None, w=None, intercept=0., label='Data', separatorLabel='Separator',
            ax=None, bound=[[-1., 1.], [-1., 1.]])
```

```
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:196: ComplexWarning:
Casting complex values to real discards the imaginary part
  offsets = np.asanyarray(offsets, float)
```

K-Nearest neighbor clustering: -

We have used an unsupervised clustering, KNN, which has previously proved efficient in clustering demographic data. This approach has been undertaken for performance comparison purposes.

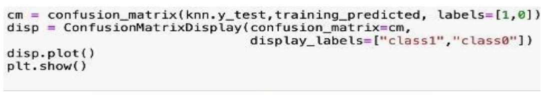


```
knn.accuracy_test(y_preds = training_predicted)
```

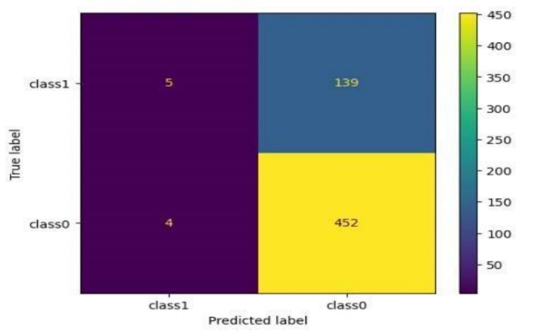
```
Test Time:  0:00:36.695329
misclassification rate 0.23833333333333334
testing error 23.833333333333336
              precision    recall  f1-score   support

           0       0.76      0.99      0.86       456
           1       0.56      0.03      0.07       144

    accuracy                           0.76       600
   macro avg       0.66      0.51      0.46       600
weighted avg       0.71      0.76      0.67       600
```

KNN is computationally intense algorithm whose complexity is proportional to the number of datapoints and dimensions in the dataset. This impacted our model in a huge degree as our training data has 90 features which contributed large amount of predicted time to cluster the datapoints

The KNN model in practical scenarios is used to understand the variance in the data instead oof relying on it for prediction purposes. So KNN is not a suitable fit for any data which has large number of dimensions and large class overlap. As we can understand, the income range can have overlapping classes in demographic features. As we can see from the above table KNN gave an accuracy of 76% and it has performed poorly in identifying the 2 class labels.

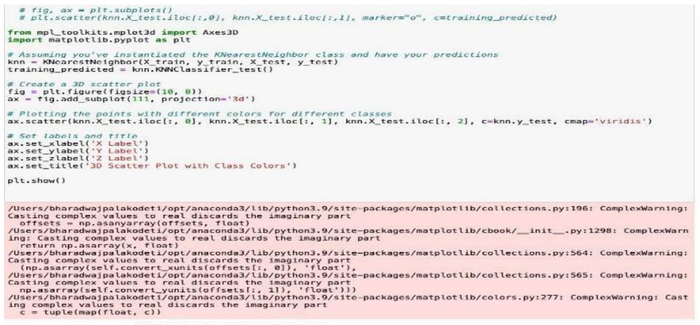


```
cm = confusion_matrix(knn.y_test,training_predicted, labels=[1,0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["class1","class0"])
disp.plot()
plt.show()
```



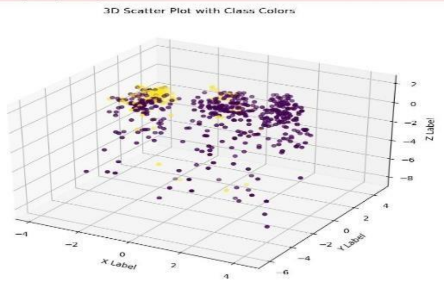
The confusion matrix above shows that both the income levels have poor predictions.

The below is the 3D of the class clusters of KNN that we have obtained.



```
  # fig, ax = plt.subplots()
  # plt.scatter(knn.X_test.iloc[:,0], knn.X_test.iloc[:,1], marker="o", c=training_predicted)

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# Assuming you've instantiated the KNearestNeighbor class and have your predictions
knn = KNearestNeighbor(X_train, y_train, X_test, y_test)
training_predicted = knn.KNNClassifier_test()

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plotting the points with different colors for different classes
ax.scatter(knn.X_test.iloc[:, 0], knn.X_test.iloc[:, 1], knn.X_test.iloc[:, 2], c=knn.y_test, cmap='viridis')

# Set labels and title
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
ax.set_title('3D Scatter Plot with Class Colors')

plt.show()
```

```
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:196: ComplexWarning:
Casting complex values to real discards the imaginary part
  offsets = np.asanyarray(offsets, float)
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/cbook/__init__.py:1298: ComplexWarn
ing: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:564: ComplexWarning:
Casting complex values to real discards the imaginary part
  (np.asarray(self.convert_xunits(offsets[:, 0]), 'float'),
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:565: ComplexWarning:
Casting complex values to real discards the imaginary part
  np.asarray(self.convert_yunits(offsets[:, 1]), 'float')))
/Users/bharadwajpalakodeti/opt/anaconda3/lib/python3.9/site-packages/matplotlib/colors.py:277: ComplexWarning: Cast
ing complex values to real discards the imaginary part
  c = tuple(map(float, c))
```



It is difficult to visualize KNN for large number of features as it is not known that which features might influence the distance scale to the maximum extent.

Overall Results: -

Model	**Accuracy**
Logistic regression	0.85
K-nearest Neighbor	0.76
Support Vector Machine	0.83
Naïve Bayes	0.24

Income greater than $50,000: -

Model	**Recall**	**Precision**	**F1 Score**
Logistic regression	0.93	0.88	0.90
K-nearest Neighbor	0.99	0.76	0.86
Support Vector Machine	0.99	0.82	0.90
Naïve Bayes	0.00	0.00	0.00

Income less than $50,000: -

Model	**Recall**	**Precision**	**F1 Score**
Logistic regression	0.60	0.73	0.66
K-nearest Neighbor	0.03	0.56	0.07
Support Vector Machine	0.32	0.92	0.47
Naïve Bayes	1.00	0.24	0.39

Conclusion: -

Using the Census Income information, we set out to create an excellent prediction model for identifying income levels. After experimenting with many machine learning methods, including K-Nearest Neighbors (KNN) clustering, Support Vector Machines (SVM), Naive Bayes, and Logistic Regression, the findings show that Logistic Regression is the best resilient and accurate approach for the task.

Key findings:

1. **Accuracy Comparison:**

   Logistic Regression outperformed KNN, SVM, and Naive Bayes in predicting income levels based on demographic variables, with an outstanding 85% accuracy.

2. **Interpretability and simplicity:**

   Logistic Regression is not only accurate, but also interpretable, which makes it easier to comprehend the link between input data and projected outcomes. This simplicity might be critical for stakeholders looking for insights into the issues driving income forecasts.

3. **Efficiency of Training and Inference:**

   Logistic regression often has faster training times than more sophisticated algorithms such as SVM. It demonstrates efficiency in both model training and prediction, making it suited for real-time applications or scenarios involving large data sets.