



## Table of Contents

Abstract.....	3
Introduction .....	3
Project Overview.....	3
Program Objectives.....	3
Project Scope and Limitations.....	3
Project Application Areas.....	3
Methodology.....	4
Program Flowchart: .....	4
Program Description: .....	5
Dataset Used:.....	5
GUI: .....	5
Tools and Techniques: .....	5
Conclusion.....	6
Outputs .....	7
Code .....	10

## Abstract

A lot of research studies try to predict the income level of individuals with a specific set of abilities, skills, background, education, etc. Usually, most studies do this by just taking a sample from the entire population and then hoping that the trend noticed in the sample was good enough to apply on the entire population. There's no way to gauge how accurate this assumption is and could have far-reaching effects if some decisions are implemented on the entire population based on the trends noticed in the sample.

What's needed is a reliable and highly accurate statistical model that can predict the income level of people with different qualities. This way the room for error is greatly reduced and can help government institutions and NGO devise better policies.

## Introduction

### Project Overview

In this project, our team has built a statistical model using KNN to predict whether a person earns above \$50,000 or otherwise. The dataset that we used had nearly 50,000 entries and the final Mean Absolute Error of our model is 0.47.

The project has 4 main modules. The first module basically cleans, preprocesses and standardizes the dataset. The second module is for automatic feature selection, this module gives an importance score to each column and the median of total columns are chosen for model training based on the importance score. The third module is for training the model and then predicting output for any given input. Lastly, the fourth module is the GUI of the application.

### Program Objectives

The objectives of this project include:

- A model that can accurately predict whether a person is earning above the \$50,000 threshold or otherwise.
- The model should have a small error so that its results are dependable.
- A user-friendly and interactive GUI for the app so that any non-technical person can easily use the model.

### Project Scope and Limitations

The dataset used in this project is based on US nationals and majority of entries are of those people who are working in the US job market. So, this model can only predict the income levels of US nationals working in US. Moreover, the income classes are based on US Dollar only so, this model cannot cater to any other currencies.

Secondly, although TKinter is quite common in Python, it is quite limited in options. This is why the user interface of the app has been kept quite clean and simple. Otherwise, there were plans to add some advanced features to the app.

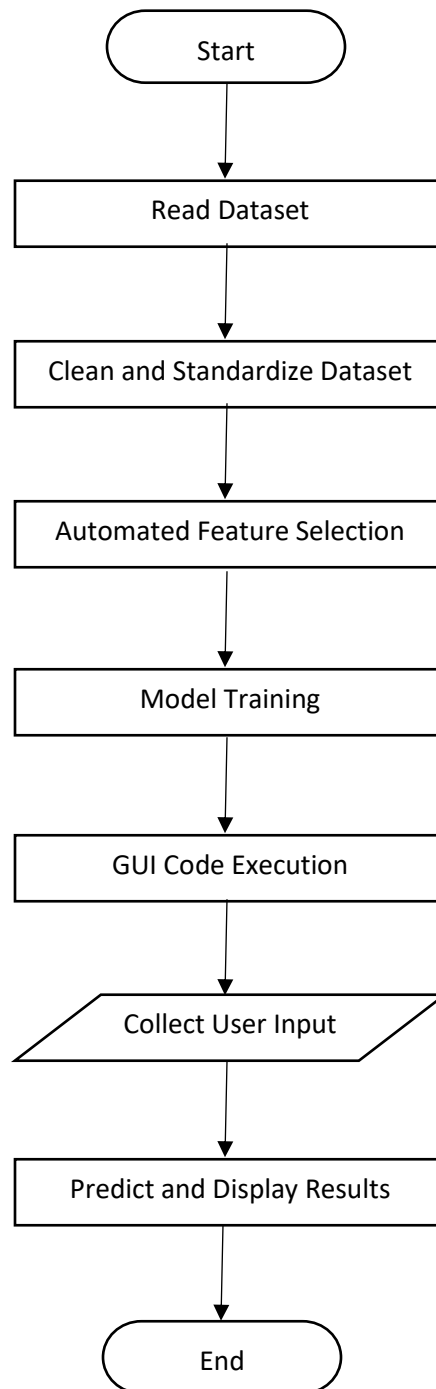
### Project Application Areas

This application can help anyone trying to gauge the income levels of different people. So, for example, anyone working in a governmental institution trying to gauge the economic standing of the people of an

area before devising a policy for them could use this application. This application could also be used by NGOs to predict which people need their help and attention the most.

## Methodology

Program Flowchart:



### Program Description:

Now, let us go over each step in a little detail:

1. **Read Dataset:** In this step, the program reads the CSV file containing nearly 50,000 entries and stores it in a DataFrame object.
2. **Clean and Standardize Data:** The dataset that we used has '?' in place of blank/null values so that first step in this module is to replace all '?' with null. After this, we apply the label encoder on each column (while making sure the encoder skips the null values and numeric columns). Next, we apply interpolation on each column to fill all null values. After this, the data is standardized.
3. **Automated Feature Selection:** In this module, we use Lasso Regression with cross-validation to compute K-fold cross-validated mean squared error. An importance score is assigned to each column and we select N columns to keep based on this score. N is the median value of total columns. Remaining columns are dropped.
4. **Model Training:** Now, the dataset divided into training and testing sets and fed into a KNN Regression model for training and testing. This module also prints out the Mean Absolute Error of the model at this point. The MAE is around 0.47.
5. **GUI Code Execution:** Now the application moves towards the GUI code and executes all commands to build the UI and run the user application.
6. **Collect User Input:** At this point, the user gives input such as age, education, martial status, etc.
7. **Predict and Display Results:** The collected input goes through the same process of cleaning, encoding and standardization after which it is fed to the model for prediction. The result is displayed to the user in the form of class prediction (either  $\leq \$50,000$  or  $> \$50,000$ ).

### Dataset Used:

The [dataset](#) that we used can be found on Kaggle. It has a usability score of 5.88, 14 feature columns and one label/output column. The dataset had quite a few missing values which had to be interpolated. Moreover, there are a few columns which did not make any sense and hence, were removed.

### GUI:

Tkinter was used for the GUI as this is the only library we have been taught for building GUI in python. We wanted to try PyQt for building the GUI, this library has drag-and-drop applications for building the GUI but there was just not enough time to experiment. The GUI has been kept pretty clean and simple. Most input is numeric, some are in textual form and one is in the form of radio button. Once the user hits the predict button, the output is displayed in the textbox right below the Predict button.

### Tools and Techniques:

The program is purely a Python-based program and following are the modules used in the program:

1. math
2. pandas
3. numpy
4. tkinter
5. sklearn.preprocessing.LabelEncoder
6. sklearn.preprocessing.StandardScaler

7. `sklearn.model_selection.train_test_split`
8. `sklearn.neighbors.KNeighborsRegressor`
9. `sklearn.metrics.mean_absolute_error`
10. `sklearn.linear_model.LassoCV`

As mentioned, the dataset had '?' instead of blank/missing/null values and so we had to replace all of those values first with `np.nan`. Then, when we applied label encoding on the dataframe we had to ensure the null values weren't encoded because we were going to apply interpolation on them right after encoding. After interpolation, we applied standardization on the dataset to center the data.

A copy of the dataset is also kept after we replace '?' with `np.nan`. This copy is then used with user input. The user input is added as a new row to the data copy and then the copy goes through the encoding, interpolation and standardization process. This is necessary so that the user input is transformed into the form that is consistent with the dataset that was used to train the model. After the cleaning and preprocessing part is done on the data copy, the last row (which is the user input, now in transformed condition) is sent to the prediction function to generate the prediction.

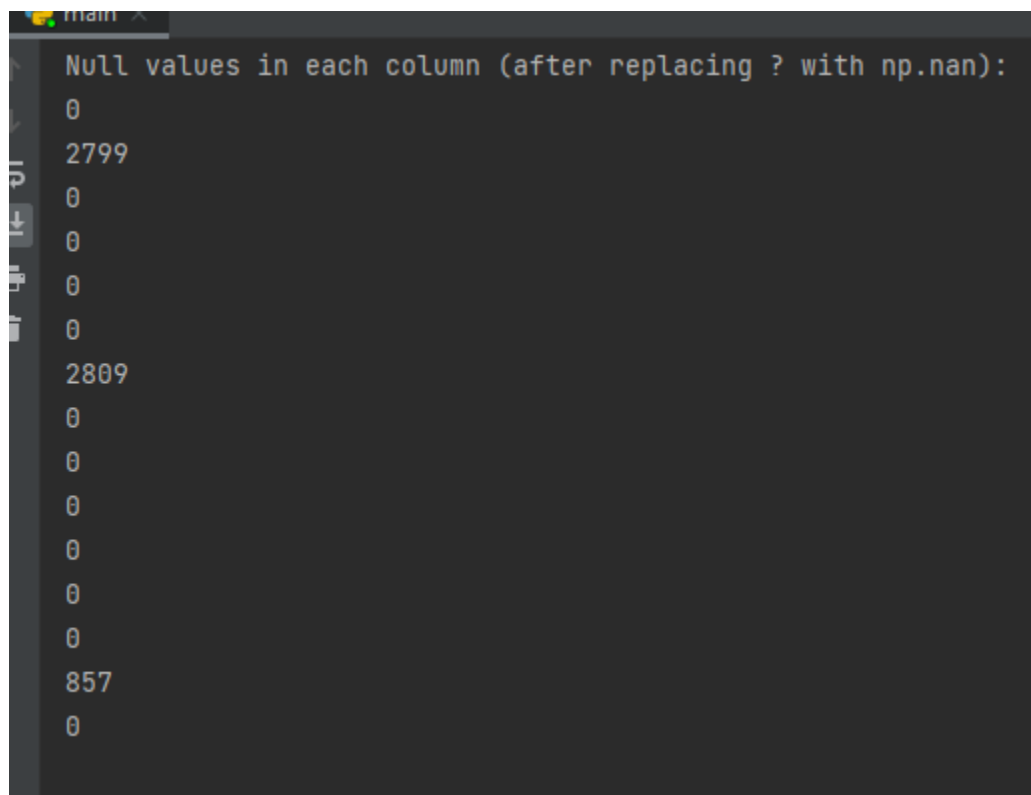
## Conclusion

The application proves that it is indeed possible to build a model that could predict income levels with great accuracy and hence provide great help in different research studies. The results of this application could be further improved by using a dataset that has entries of people from countries other than US as well. Also, as columns increase, a multi-layer perceptron model might prove to be more useful to discover all the underlying intricacies and relations of the data.

## Outputs

```
Mean Absolute Error: 0.47060680471220945
```

Error in the model



```
Null values in each column (after replacing ? with np.nan):  
0  
2799  
0  
0  
0  
0  
0  
2809  
0  
0  
0  
0  
0  
0  
0  
857  
0
```

Null values in the data after we replace all the '?' with np.nan

```
Null values in each column (after interpolation):
```

```
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0
```

Null values after applying interpolation on each column



tk

Enter Age:

Enter Total Years of Education:

Select Gender:  
☒ Male  
☐ Female

Enter Captial Gain:

Enter Captial Loss:

Enter Total Working Hours Per Week:

Class prediciton: <= \$50,000

Application UI and how it works

## Code

```
import math
import pandas as pd
import numpy as np
from tkinter import *
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LassoCV

# data cleaning and preprocessing
def clean_preprocess_data(data):
    # print('Null values in each column (after replacing ? with np.nan):')
    for i in data.columns:          #dataset has '?' instead of
null/blank spaces, so this loop replaces ? with np.nan
        data[i] = data[i].replace('?', np.nan)
        # print(data[i].isna().sum())

    for col_name in data.columns:  # Label Encoding while ignoring np.nan
values and numeric columns
        series = data[col_name]
        if (type(series[1]) != 'int' and type(series[1]) != 'float'):
            label_encoder = LabelEncoder()
            data[col_name] = pd.Series(
                label_encoder.fit_transform(series[series.notnull()]),
                index=series[series.notnull()].index
            )

    for i in data.columns:
        data[i] = data[i].interpolate(method='linear')  # interpolating
np.nan values

    # print('\n\nNull values in each column (after interpolation):')
    # for i in data.columns:          #proof that np.nan
values are not there anymore
    #     print(data[i].isna().sum())

    # standardizing data
    st = StandardScaler()
    data = st.fit_transform(data)
    data = pd.DataFrame(data)

    return data

# automated feature selection
def select_features(data, data_copy):

    # Separating input and output
    X = data.iloc[:, :-1]
```

```

y = data.iloc[:, -1]

# Fit the feature selection model
clf = LassoCV().fit(X, y)

# Selected features
cols = math.ceil((len(feature_names) - 1) / 2) # median value of
total columns selected, as recommended online
importance = np.abs(clf.coef_)
idx_third = importance.argsort()[-3]
threshold = importance[idx_third] + 0.01
idx_features = (-importance).argsort()[:cols]

to_keep = np.array(feature_names)[idx_features]
to_keep = list(to_keep)

check = 'income' in to_keep
if check == False:
    to_keep.append('income')

to_drop = [item for item in feature_names if item not in to_keep]

data = data.drop(to_drop, axis=1)
data_copy = data_copy.drop(to_drop, axis=1)

return data, data_copy

def click():

    global cols_to_keep

    a = int(age.get())
    b = int(YearOfEdu.get())
    c = mstatus.get()
    d = sex.get()
    e = float(CG.get())
    f = float(CL.get())
    g = int(hours.get())

    # generate class output for user input
    user_input = {'age': (a), 'education_num': (b), 'marital_status': (c),
                  'sex': (d), 'capital_gain': (e), 'capital_loss': (f),
                  'hours_per_week': (g)}

    user_input = pd.DataFrame([user_input])

    data = data_copy.copy()
    input = pd.concat([data, user_input], ignore_index=True, axis=0)
    input = clean_preprocess_data(input)

    input.columns = cols_to_keep
    user_input_row = input.iloc[-1, :]

    obj = pd.DataFrame([user_input_row])
    res = knn.predict(obj)
    output.delete('1.0', END)

```

```

    if res <= 0:
        output.insert(END, 'Class predicition: <= $50,000')
    else:
        output.insert(END, 'Class predicition: > $50,000')

#main
global data
global data_copy
global cols_to_keep
global knn

#Reading and cleaning the dataset
data = pd.read_csv(r'C:\Users\Naeem\Desktop\Jahanzeb\AI\project\adult.csv')
data = pd.DataFrame(data)

feature_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
                 'marital_status', 'occupation',
                 'relationship',
                 'race', 'sex', 'capital_gain', 'capital_loss',
                 'hours_per_week', 'native_country', 'income']

data.columns = feature_names
data_copy = data.copy() # saving a copy of data (to be used
                        # for encoding, standardizing user input)
data_copy.columns = feature_names

data = clean_preprocess_data(data)
data.columns = feature_names # standardization removed column
names but we need them for dropping unnecessary columns
data, data_copy = select_features(data, data_copy)

# Used later on for predicting output for user input
data_copy = data_copy.drop('income', axis=1)
cols_to_keep = list(data_copy.columns)

# Separating input and output
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# training and testing KNN model
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,
                                                    test_size=0.2)
knn = KNeighborsRegressor(n_neighbors=25)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print('Mean Absolute Error: ', mean_absolute_error(y_test, y_pred))

#GUI code

#main window

```

```

root = Tk()
root.geometry("800x600")
root.configure(background="white", border=4, borderwidth=4)

# user input age
l = Label(root, text="Enter Age:", fg='black', bg='white')
l.pack()
age = Entry(root, width=15, fg='black', bg='white')
age.pack()
space = Label(root, text="", fg='white', bg='white')
space.pack()

# user input Number of years of education
l1 = Label(root, text="Enter Total Years of Education:", fg='black',
bg='white')
l1.pack()
YearOfEdu = Entry(root, width=15, fg='black', bg='white')
YearOfEdu.pack()
space2 = Label(root, text="", bg='white')
space2.pack()

# user input marital_status
opt = data_copy['marital_status'].unique()
mstatus = StringVar()
mstatus.set("Set Marital Status: ")
marital_status = OptionMenu(root, mstatus, *opt)
marital_status.pack()
space3 = Label(root, text="", bg='white')
space3.pack()

# user input Gender
l2 = Label(root, text="Select Gender:", bg='white', fg='black')
l2.pack()
sex = StringVar()
gender = Radiobutton(root, text="Male", bg='white', value='Male',
variable=sex)
gender.pack()
gender = Radiobutton(root, text="Female", bg='white', value='Female',
variable=sex)
gender.pack()
space4 = Label(root, text="", bg='white')
space4.pack()

# user input capital gain
l3 = Label(root, text="Enter Captial Gain:", bg='white', fg='black')
l3.pack()
CG = Entry(root, width=15, fg='black', bg='white')
CG.pack()
space5 = Label(root, text="", bg='white')
space5.pack()

# user input capital loss
l5 = Label(root, text="Enter Captial Loss:", bg='white', fg='black')
l5.pack()
CL = Entry(root, width=15, fg='black', bg='white')
CL.pack()
space6 = Label(root, text="", bg='white')

```

```
space6.pack()

# user input Working hours per week
l6 = Label(root, text="Enter Total Working Hours Per Week:", bg='white',
fg='black')
l6.pack()
hours = Entry(root, width=15, fg='black', bg='white')
hours.pack()
space7 = Label(root, text="\n\n", bg='white')
space7.pack()

# Predict
button = Button(root, text='Predict', width=10, fg='black', bg='gray',
command=click)
button.pack()
space8 = Label(root, text="", bg='white')
space8.pack()
output = Text(root, width=35, height=1, background='white')
output.pack()

root.mainloop()
```