İTÜ

Socioeconomic Status Classification

A Data-driven Approach

mindset. institute

12.05.2024

Samir Guliyev

Project Introduction

- Welcome to the presentation on our project involving the Adult Census dataset.
- Our project focuses on utilizing data science techniques for predictive modeling and analysis.
- The Adult Census dataset is a widely used dataset in the field, offering valuable insights into socioeconomic factors affecting income levels.
- Our goal is to develop robust predictive models that can accurately classify individuals based on their income levels (>50K or <=50K).
- Through this presentation, we'll explore our approach, methodologies, and findings in addressing this classification task.

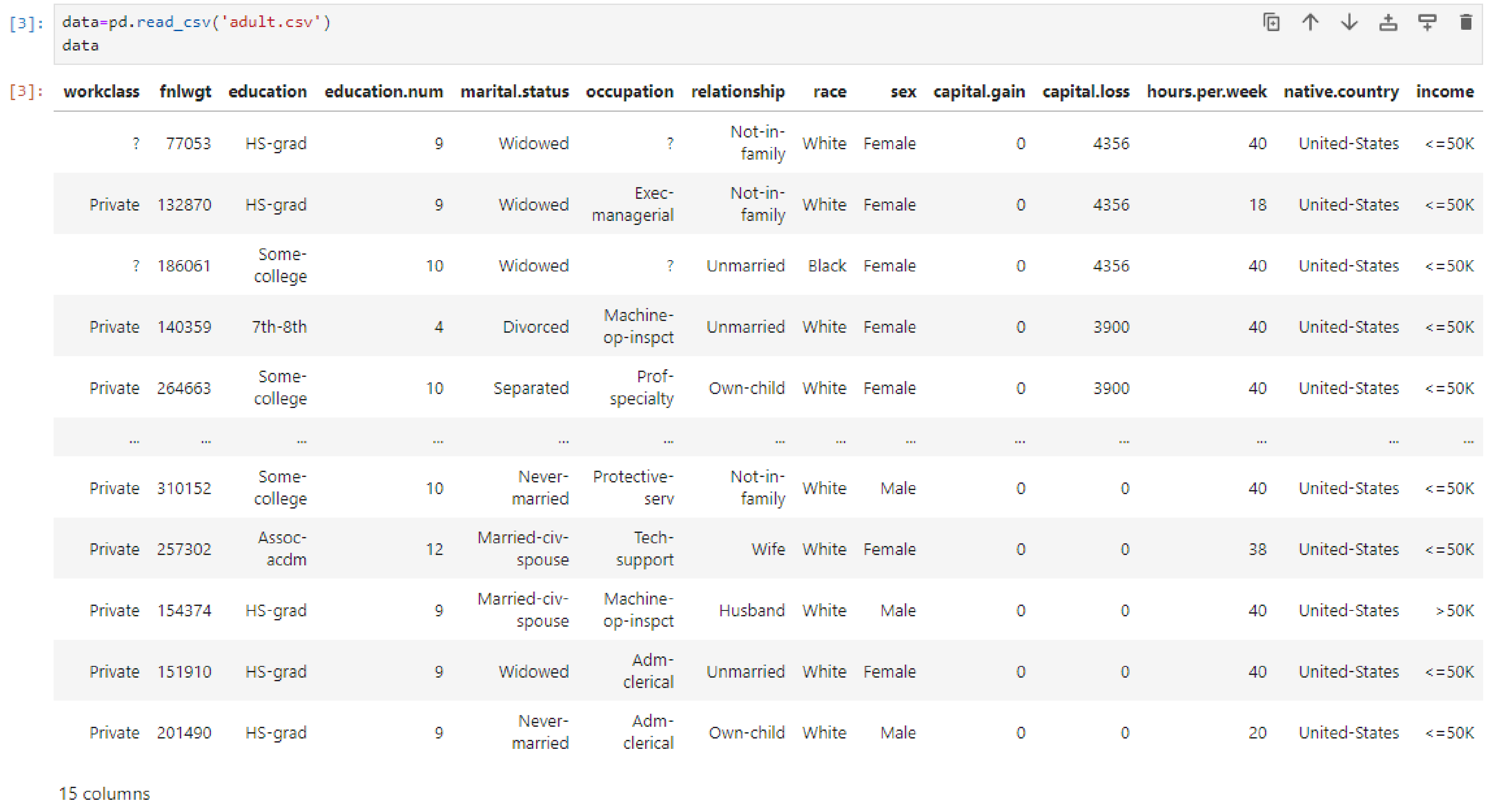
Problem Statement

- The problem at hand revolves around predicting income levels based on demographic and socioeconomic features.
- Specifically, we aim to classify individuals into two income categories: >50K or <=50K.
- This binary classification task is crucial for various real-world applications such as targeted marketing, credit risk assessment, and social policy planning.
- By leveraging the Adult Census dataset, our objective is to develop accurate predictive models that can effectively categorize individuals based on their income levels.

Dataset Overview

- The Adult Census dataset, also known as the “Adult” dataset, is a well-known resource in machine learning and data mining.
- Extracted from the 1994 Census database, it contains various demographic and socioeconomic features of individuals.
- This dataset is commonly used for predictive modeling tasks, particularly in forecasting income levels based on socioeconomic attributes.
- The Adult Census dataset comprises a total of 14 features, offering insights into various demographic and socioeconomic attributes. Features include both categorical and numerical variables, providing a comprehensive view of individuals' characteristics.

Dataset Overview



```
[3]: data=pd.read_csv('adult.csv')
     data
```

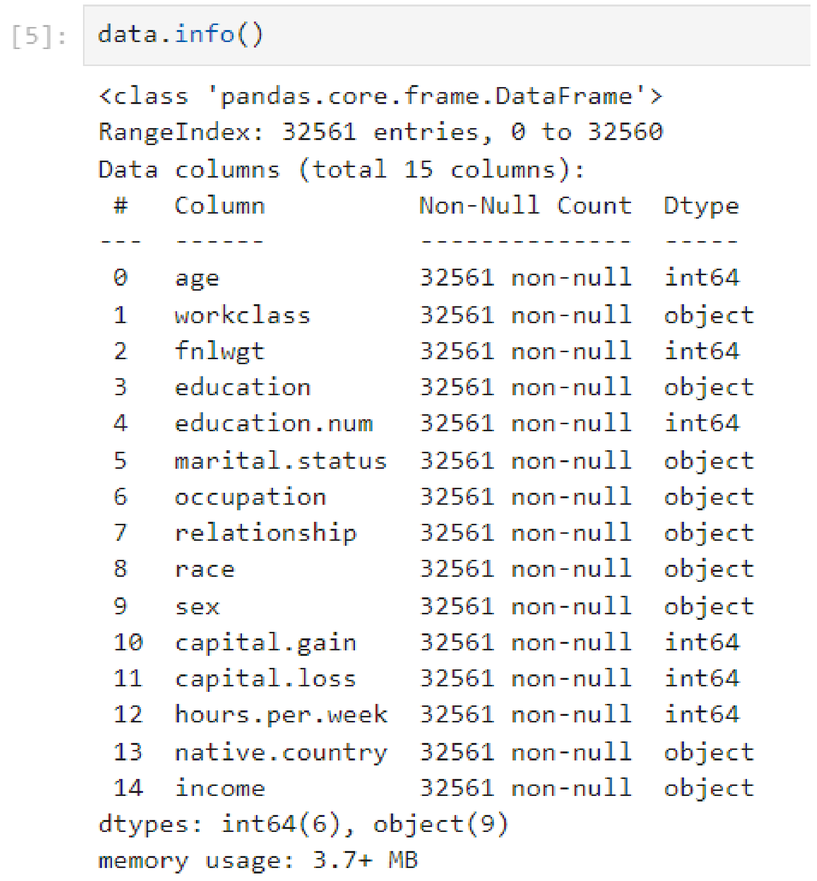
[3]:

workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40	United-States	<=50K
Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K

15 columns

Features Overview

- Categorical variables encompass attributes such as education level, occupation, marital status, race, and gender, offering insights into social and demographic factors.
- Numerical variables encompass attributes such as age, education years, capital gains, capital losses, and hours per week worked, providing quantitative information about individuals' socioeconomic status and activities.



```
[5]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

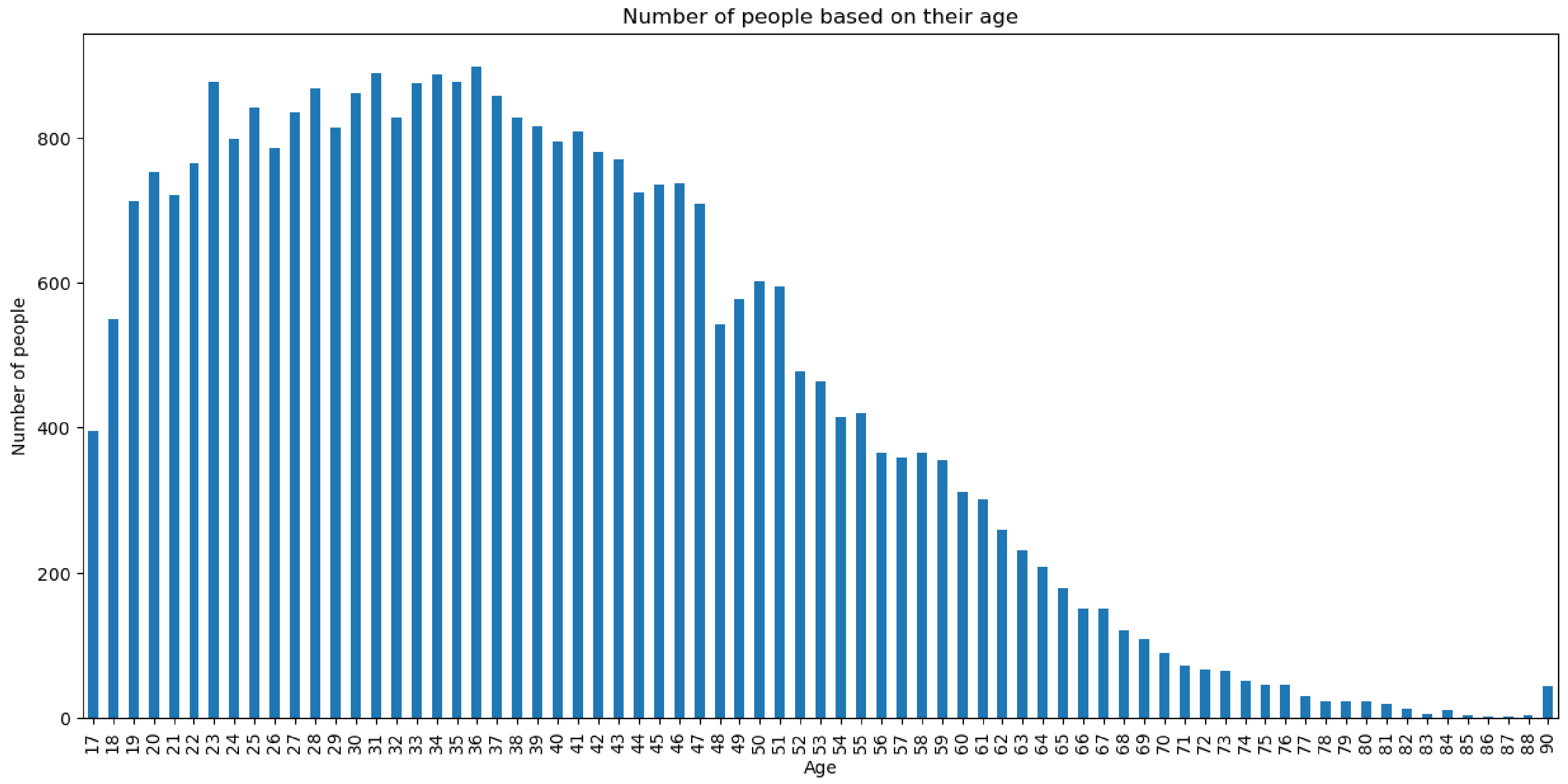
EDA

Exploratory Data Analysis

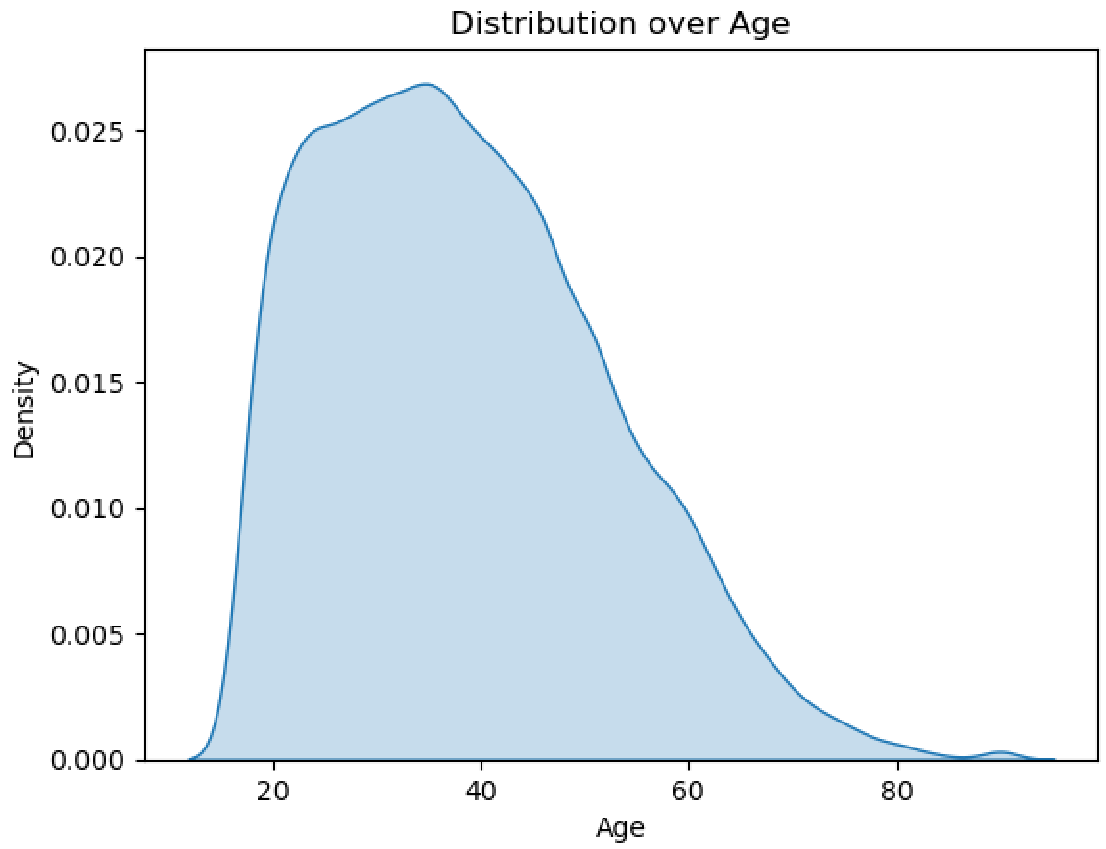
Introduction to EDA

- Exploratory Data Analysis (EDA) is a crucial step in the data analysis process, aimed at gaining insights and understanding the underlying patterns and characteristics of the dataset.
- EDA involves examining various features of the dataset, visualizing distributions, assessing data balance, exploring correlations, and identifying outliers.
- Through EDA, we aim to understand the structure of the data, identify potential preprocessing needs, and inform modeling decisions based on a comprehensive understanding of the dataset.

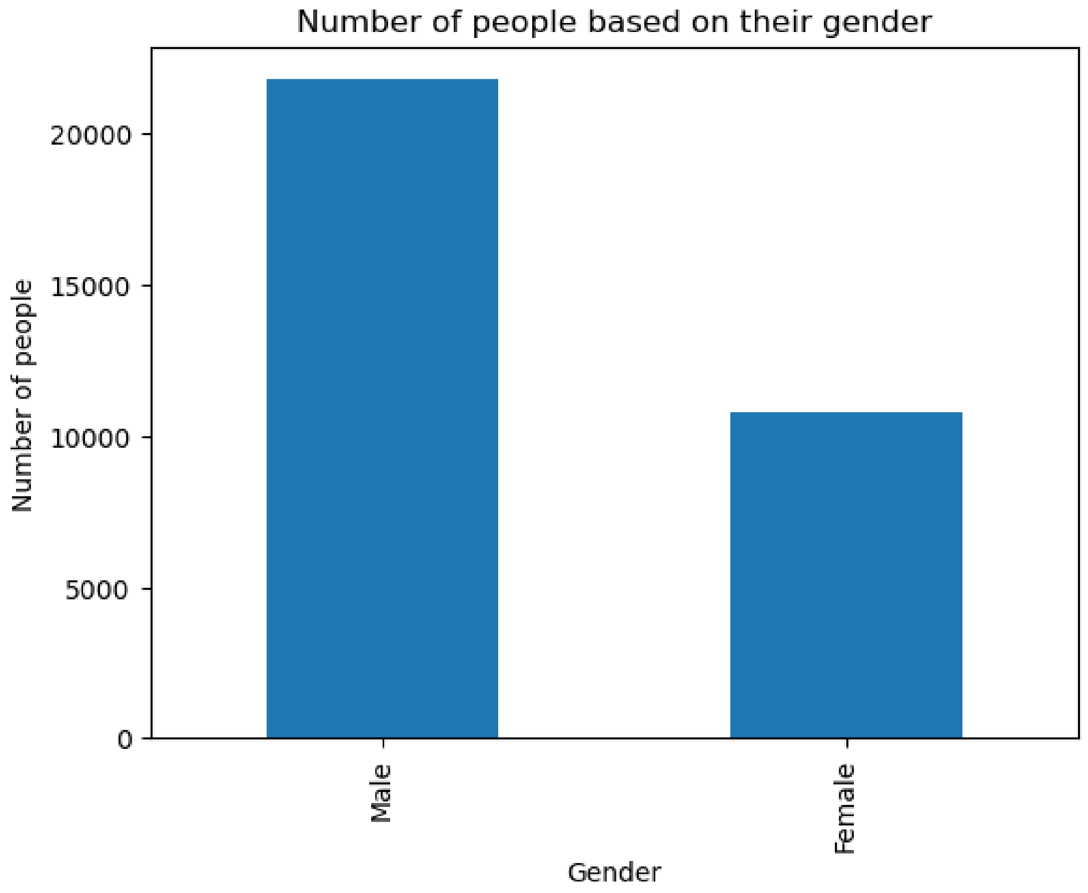
Age Distribution



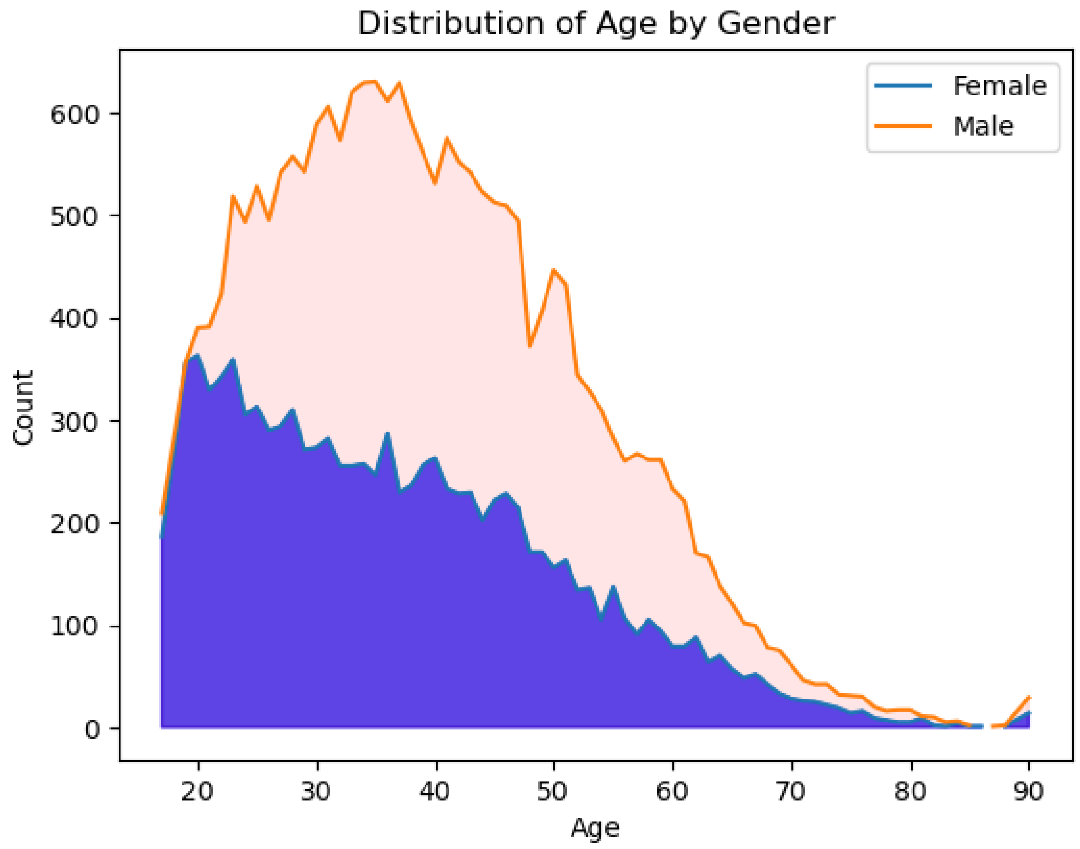
Age Distribution (KDE)



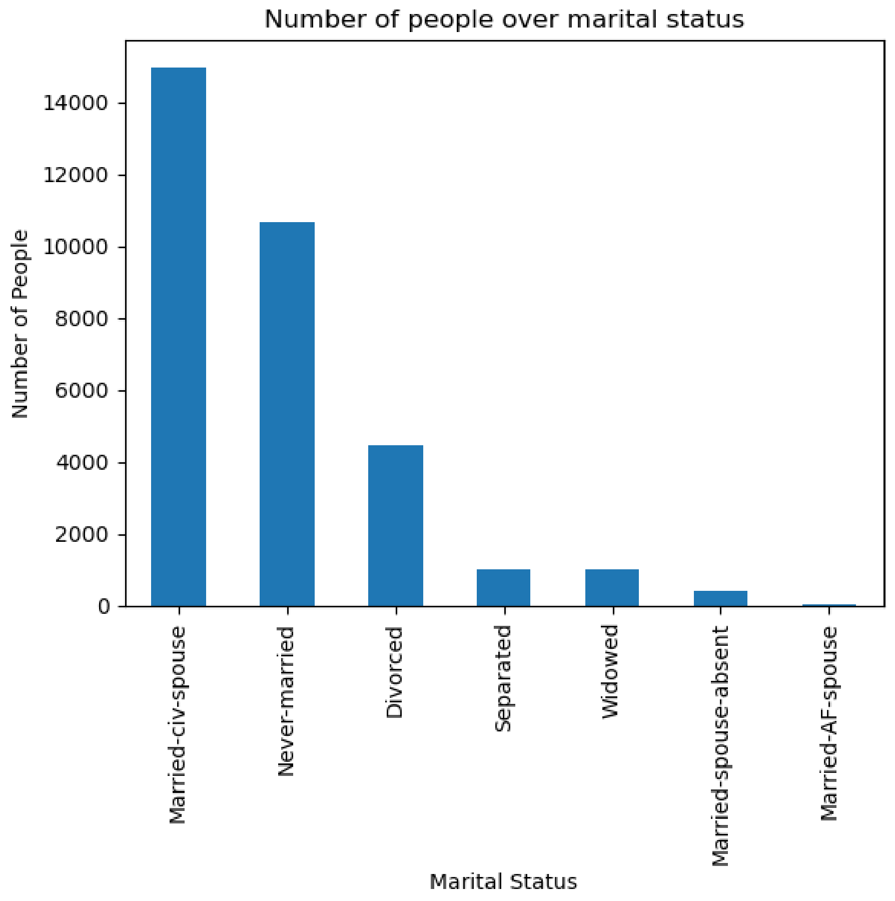
Gender Distribution



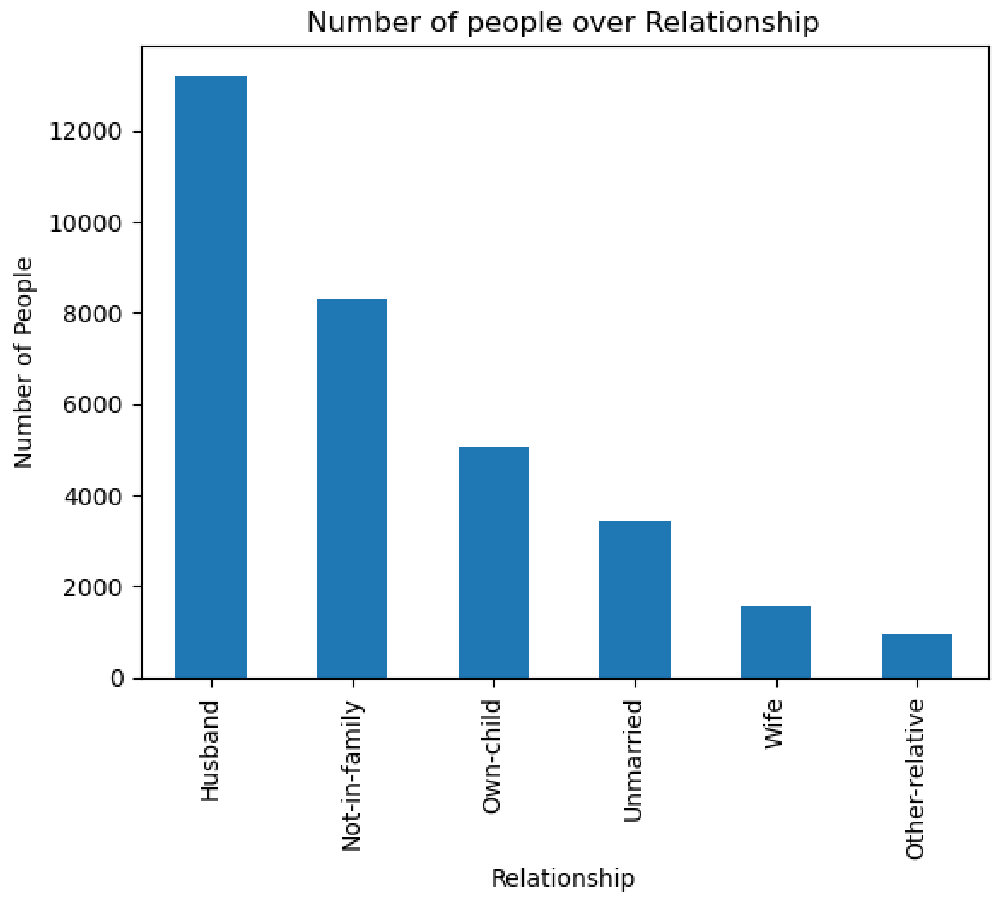
Distribution of Age by Gender



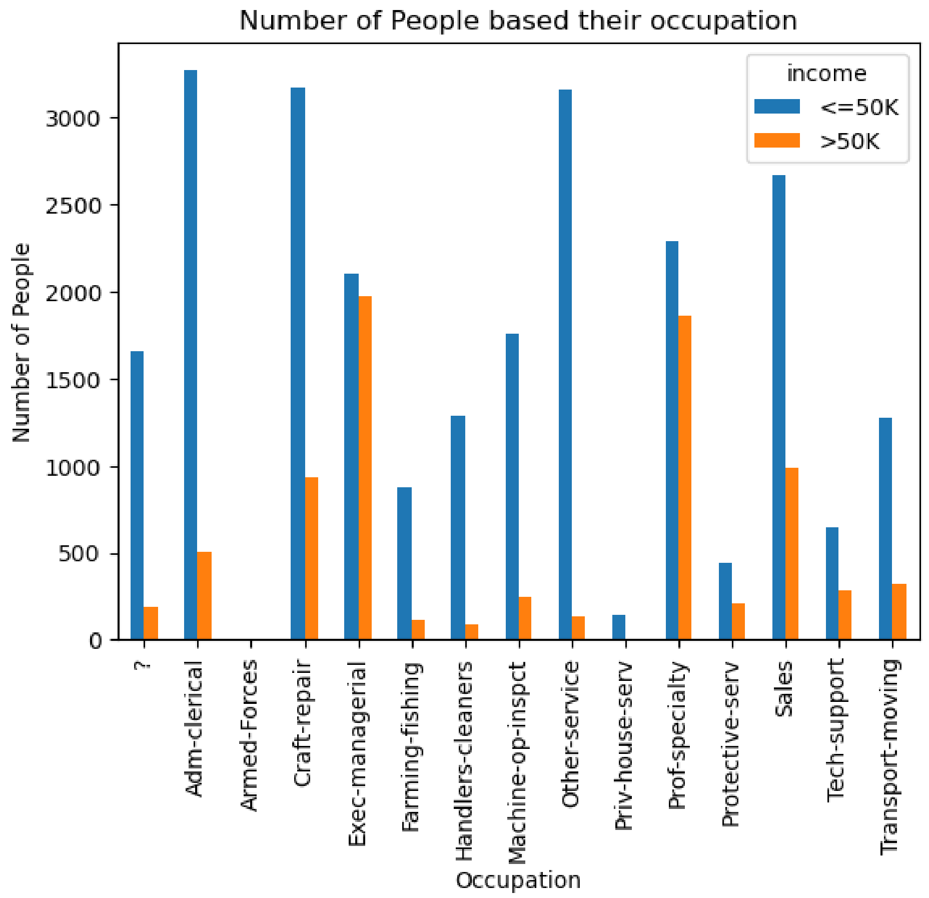
Marital Status Distribution



Relationship Distribution



Occupation Distribution



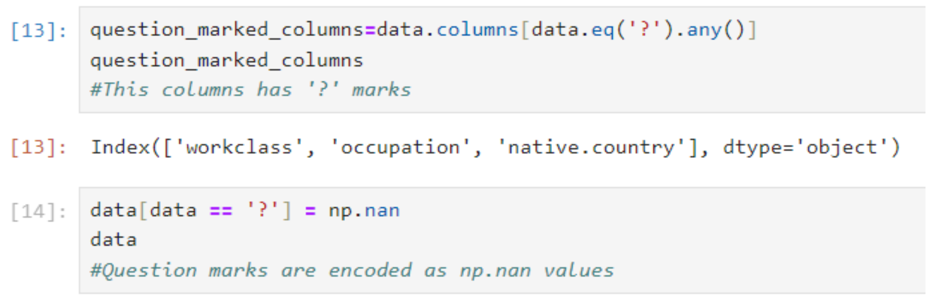
Data Cleaning and Preparation

Data Cleaning and Preparation

- Data cleaning and preparation are essential steps to ensure the quality and reliability of the dataset for analysis and modeling.
- This phase involves identifying and handling missing values, encoding categorical variables, scaling numerical features, and splitting the dataset into training and testing sets.
- By cleaning and preparing the data effectively, we can ensure that our models are trained on high-quality data, leading to more accurate and reliable predictions.

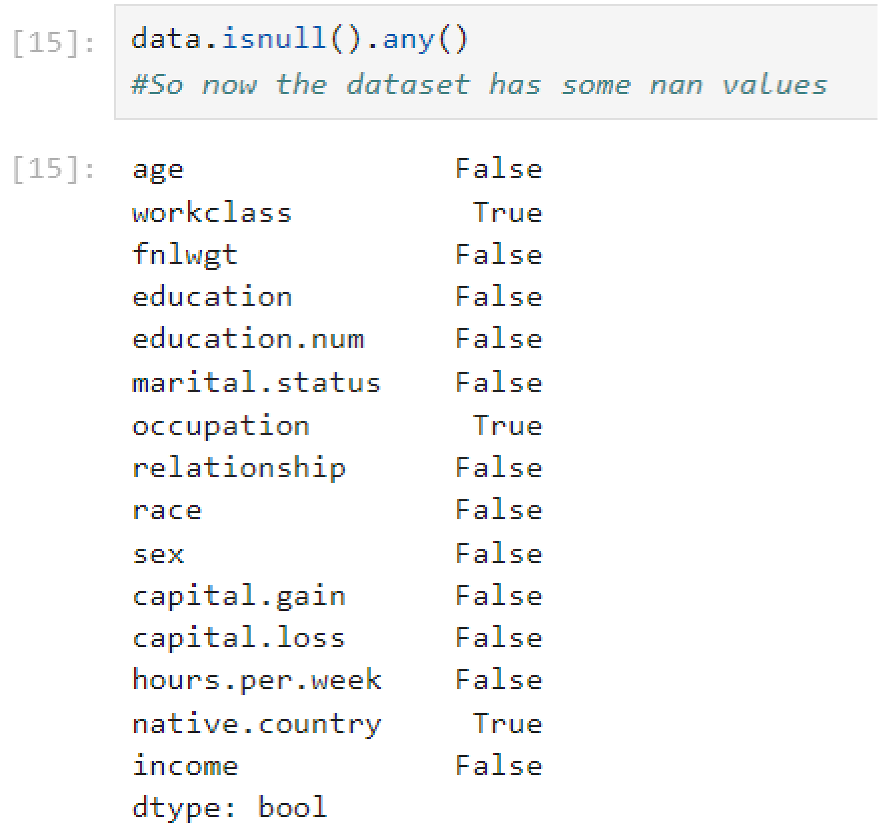
Identify Missing Values

- In the Adult Census dataset, missing values are represented as “?” marks, which deviate from standard NaN values.
- This non-standard representation of missing values requires special attention during data cleaning to ensure accurate analysis and modeling.
- Strategies for handling missing values may include imputation, deletion, or encoding them as NaN values for consistency in analysis and modeling.



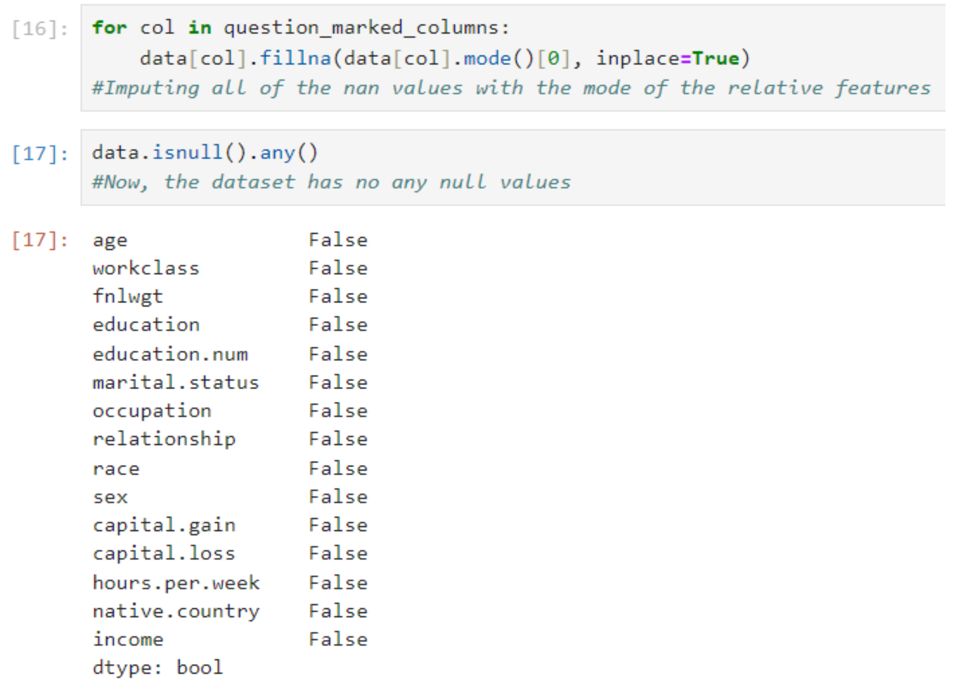
```
[13]: question_marked_columns=data.columns[data.eq('?').any()]
      question_marked_columns
      #This columns has '?' marks

[13]: Index(['workclass', 'occupation', 'native.country'], dtype='object')

[14]: data[data == '?'] = np.nan
      data
      #Question marks are encoded as np.nan values
```

Handling Missing Values



```
[15]: data.isnull().any()
      #So now the dataset has some nan values
```

```
[15]: age                False
      workclass           True
      fnlwgt             False
      education          False
      education.num      False
      marital.status     False
      occupation          True
      relationship       False
      race               False
      sex                False
      capital.gain       False
      capital.loss       False
      hours.per.week     False
      native.country      True
      income             False
      dtype: bool
```



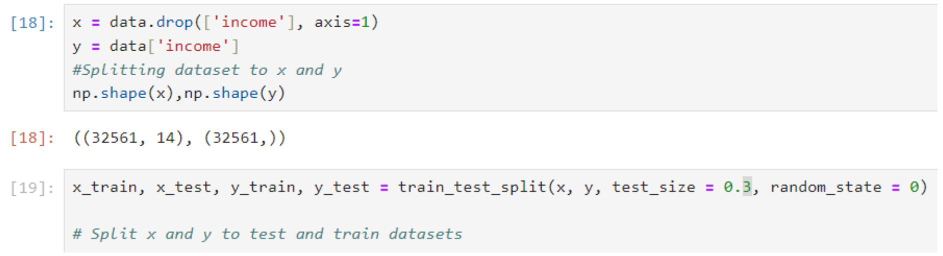
```
[16]: for col in question_marked_columns:
          data[col].fillna(data[col].mode()[0], inplace=True)
      #Imputing all of the nan values with the mode of the relative features
```

```
[17]: data.isnull().any()
      #Now, the dataset has no any null values
```

```
[17]: age               False
      workclass         False
      fnlwgt            False
      education         False
      education.num     False
      marital.status    False
      occupation        False
      relationship      False
      race              False
      sex               False
      capital.gain      False
      capital.loss      False
      hours.per.week    False
      native.country    False
      income            False
      dtype: bool
```

Dataset Splitting

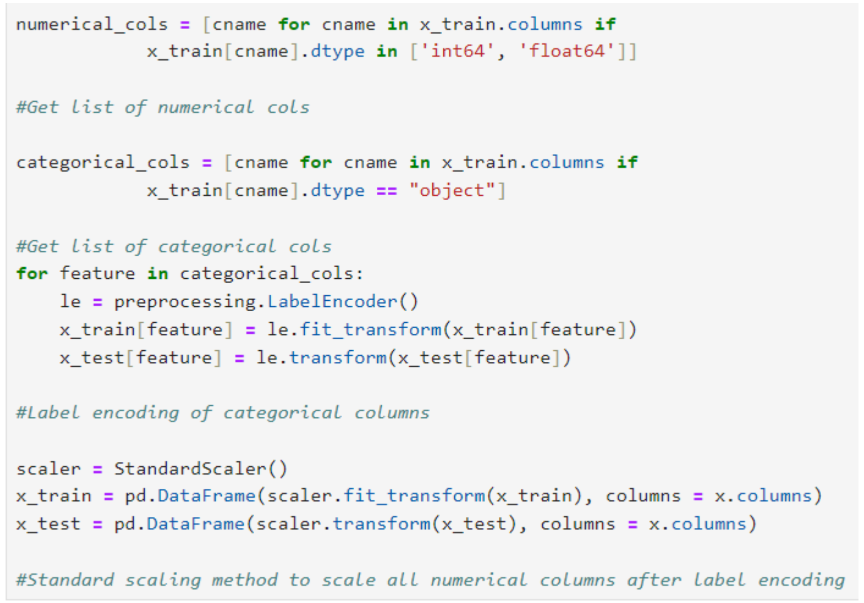
- The dataset is split into input features (x) and target variable (y).
- The shape of the input features (x) is (32561, 14), while the shape of the target variable (y) is (32561,).
- The dataset is further split into training and testing sets using a 70/30 ratio.
- This ensures that a portion of the data is reserved for model evaluation.



```
[18]: x = data.drop(['income'], axis=1)
      y = data['income']
      #Splitting dataset to x and y
      np.shape(x),np.shape(y)

[18]: ((32561, 14), (32561,))

[19]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)

      # Split x and y to test and train datasets
```

Dataset Preprocessing

- Numerical and categorical columns are identified in the training set.
- Categorical columns are label encoded to transform categorical values into numerical representations.
- Numerical columns are standardized using StandardScaler() to ensure consistent feature scales for model training.



```python
numerical_cols = [cname for cname in x_train.columns if
            x_train[cname].dtype in ['int64', 'float64']]

#Get list of numerical cols

categorical_cols = [cname for cname in x_train.columns if
            x_train[cname].dtype == "object"]

#Get list of categorical cols
for feature in categorical_cols:
    le = preprocessing.LabelEncoder()
    x_train[feature] = le.fit_transform(x_train[feature])
    x_test[feature] = le.transform(x_test[feature])

#Label encoding of categorical columns

scaler = StandardScaler()
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)

#Standard scaling method to scale all numerical columns after label encoding
```

Modeling

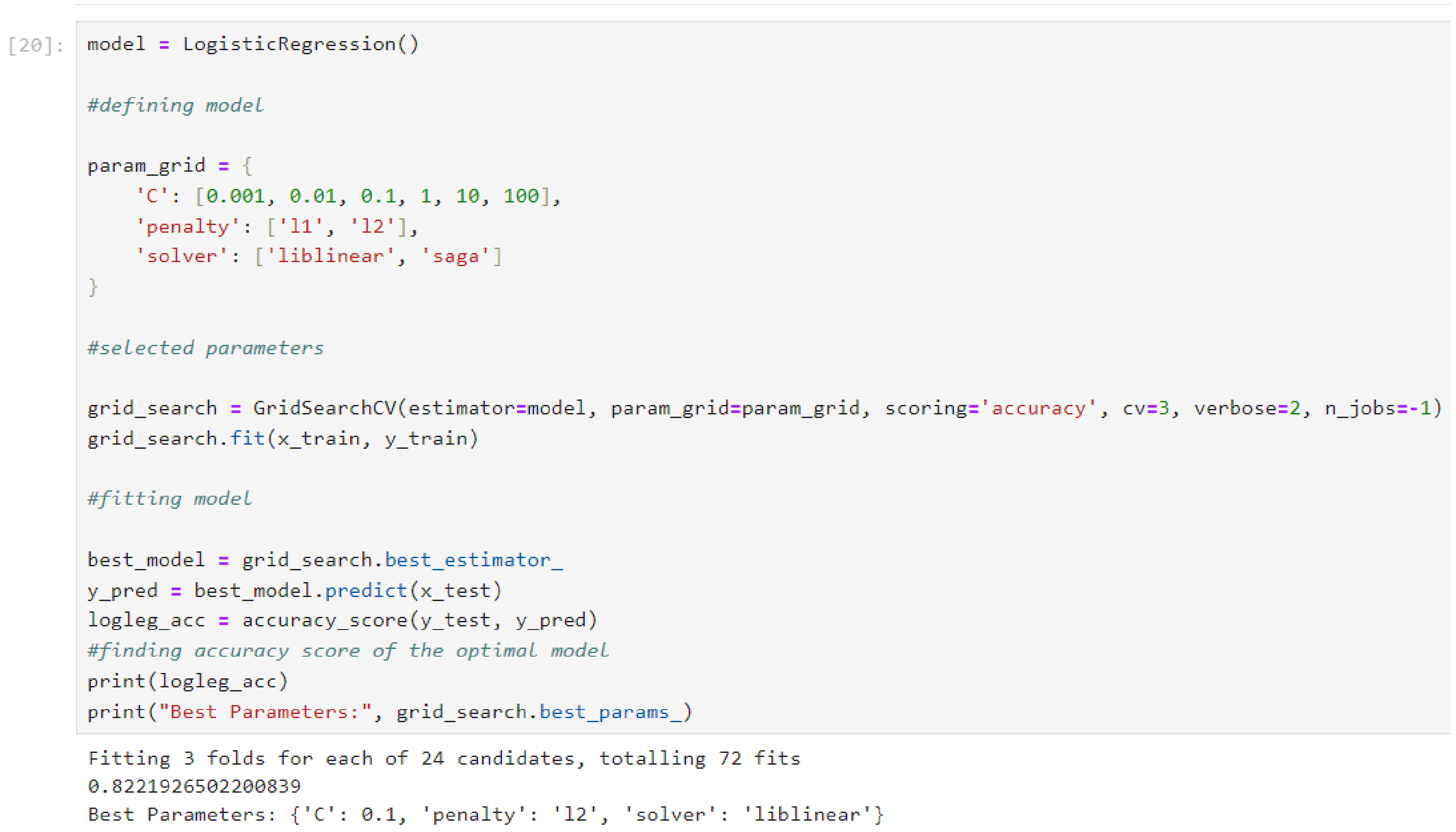
Modeling Approach

- Our modeling approach involves employing several machine learning algorithms to predict income levels based on demographic and socioeconomic features.
- We'll explore the following steps in our modeling process:
  - **Data preprocessing:** Handling missing values, encoding categorical variables, and scaling numerical features.
  - **Model selection:** Experimenting with various algorithms such as Logistic Regression, XGBoost, and Decision Trees to identify the most suitable model for our task.
  - **Hyperparameter tuning:** Optimizing model parameters using techniques like Grid Search and Randomized Search to improve performance.
  - **Evaluation:** Assessing model performance using accuracy score.

Logistic Regression

- Logistic Regression Experiments involve assessing the performance of the Logistic Regression algorithm in predicting income levels based on demographic and socioeconomic features.
- The objective is to optimize Logistic Regression hyperparameters and evaluate its accuracy under various settings to determine its effectiveness in our predictive modeling task.
- Through Logistic Regression experiments, we aim to identify the optimal configuration for maximizing predictive accuracy while ensuring model interpretability and computational efficiency.

Logistic Regression in Our Approach



```
[20]: model = LogisticRegression()

      #defining model

      param_grid = {
          'C': [0.001, 0.01, 0.1, 1, 10, 100],
          'penalty': ['l1', 'l2'],
          'solver': ['liblinear', 'saga']
      }

      #selected parameters

      grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3, verbose=2, n_jobs=-1)
      grid_search.fit(x_train, y_train)

      #fitting model

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test)
      logleg_acc = accuracy_score(y_test, y_pred)
      #finding accuracy score of the optimal model
      print(logleg_acc)
      print("Best Parameters:", grid_search.best_params_)
```

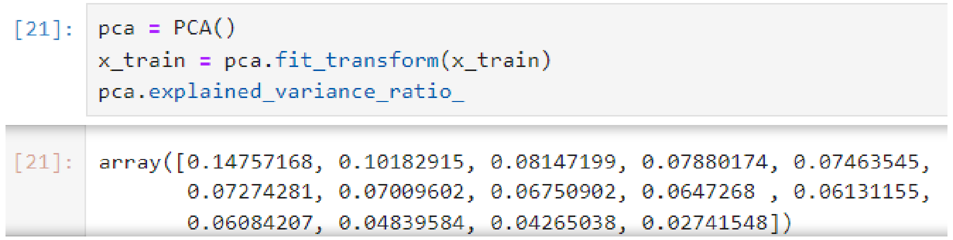
```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
0.8221926502200839
Best Parameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

Introduction to PCA

- Principal Component Analysis (PCA) is a dimensionality reduction technique used to simplify complex datasets while retaining important information.
- In our experiments, we explore the impact of PCA on model performance by systematically removing features and evaluating the effects on accuracy.
- The goal is to determine the optimal number of principal components that maximize predictive power while minimizing computational complexity.
- Through PCA experiments, we aim to gain insights into the relationship between feature dimensionality and model performance.

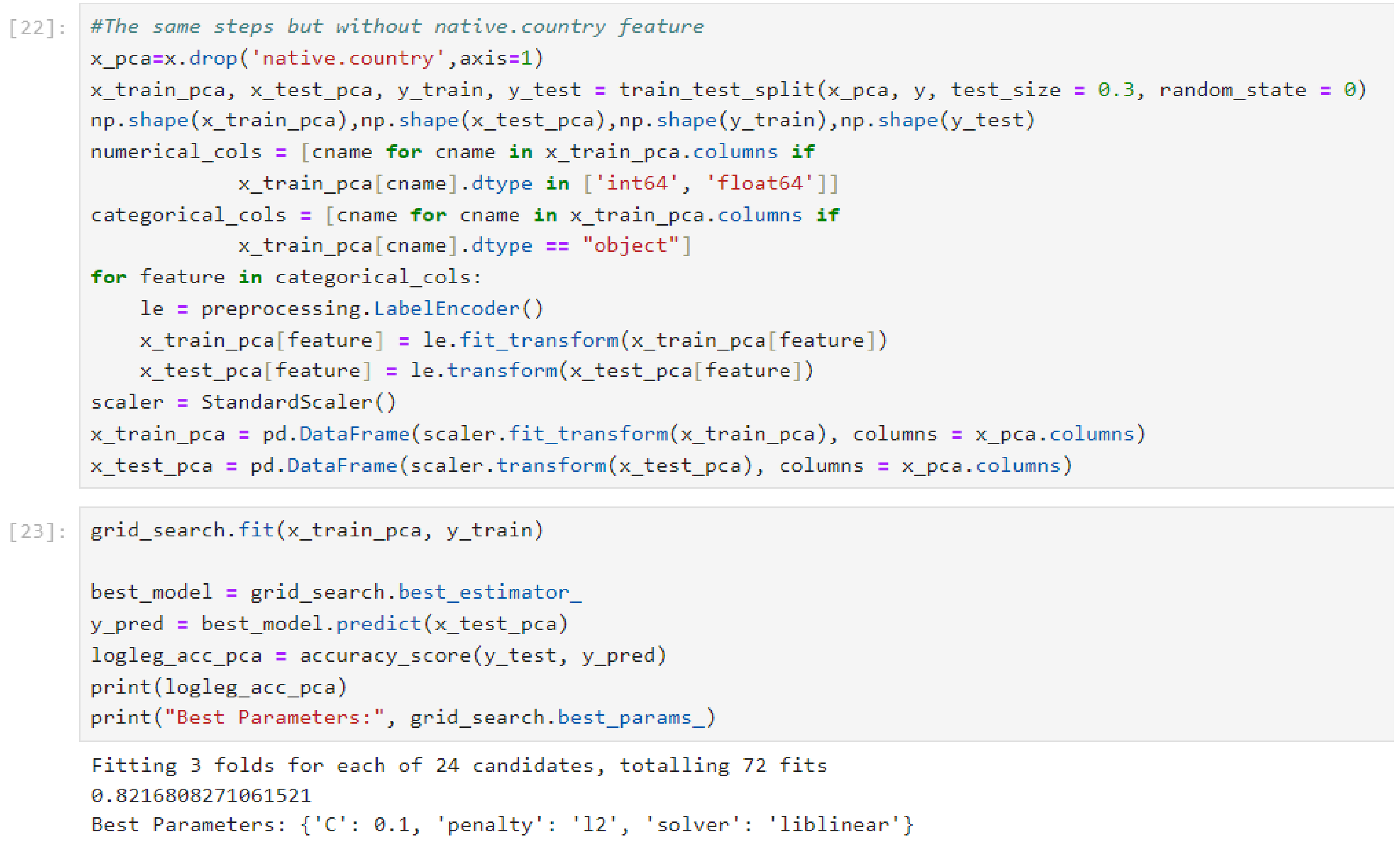
PCA Results

- **Experiment 1:** Removing the last feature, which contributes only 2.74% to the variance, leaving 97.5% of the variance explained by the remaining features.
- **Experiment 2:** Removing the last two features, which collectively contribute to only 7% of the variance.
- **Experiment 3:** Removing the last three features, which contribute to only 11% of the variance.



```
[21]: pca = PCA()
      x_train = pca.fit_transform(x_train)
      pca.explained_variance_ratio_

[21]: array([0.14757168, 0.10182915, 0.08147199, 0.07880174, 0.07463545,
             0.07274281, 0.07009602, 0.06750902, 0.0647268 , 0.06131155,
             0.06084207, 0.04839584, 0.04265038, 0.02741548])
```

1st Experiment with PCA

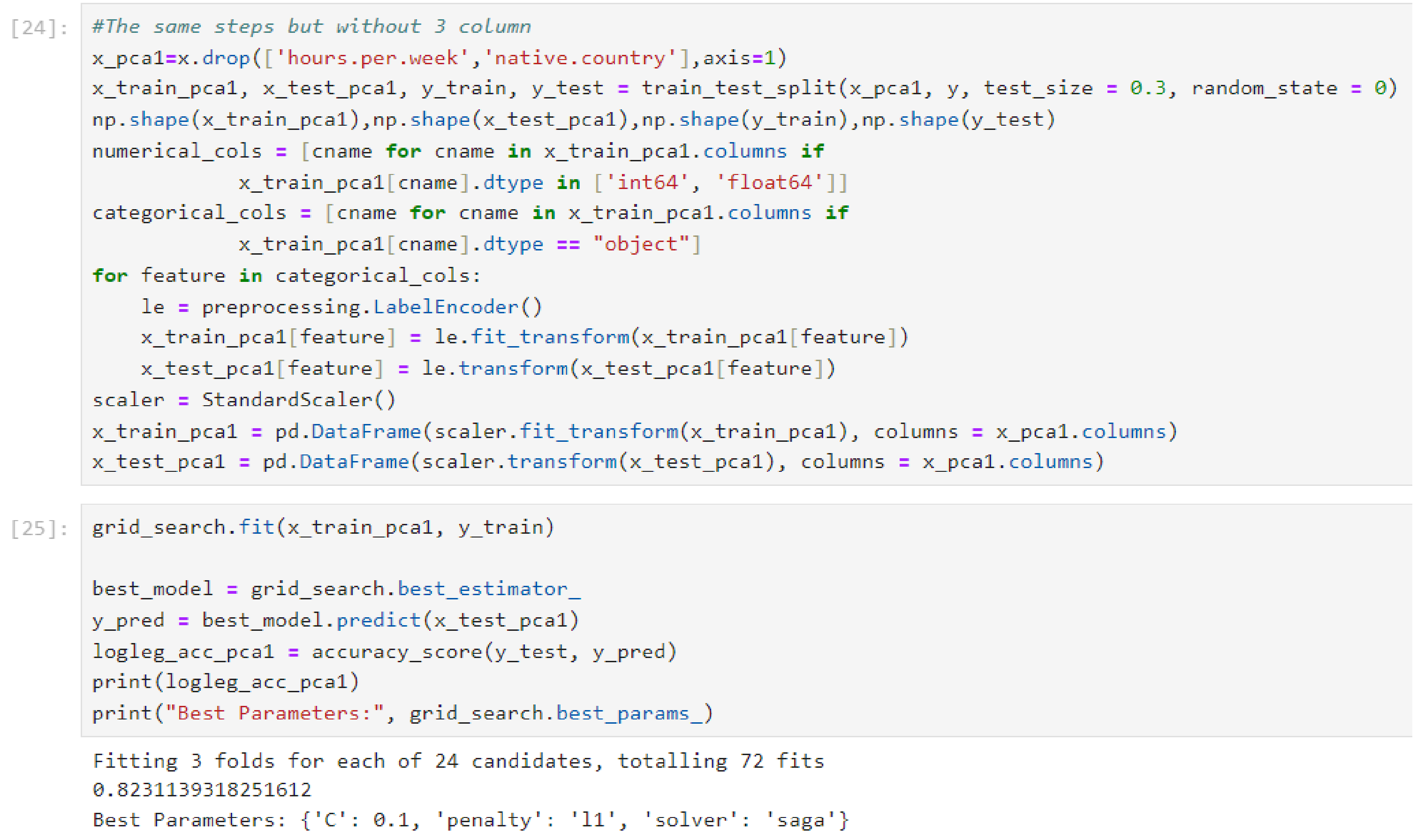


```
[22]: #The same steps but without native.country feature
      x_pca=x.drop('native.country',axis=1)
      x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size = 0.3, random_state = 0)
      np.shape(x_train_pca),np.shape(x_test_pca),np.shape(y_train),np.shape(y_test)
      numerical_cols = [cname for cname in x_train_pca.columns if
                  x_train_pca[cname].dtype in ['int64', 'float64']]
      categorical_cols = [cname for cname in x_train_pca.columns if
                  x_train_pca[cname].dtype == "object"]
      for feature in categorical_cols:
          le = preprocessing.LabelEncoder()
          x_train_pca[feature] = le.fit_transform(x_train_pca[feature])
          x_test_pca[feature] = le.transform(x_test_pca[feature])
      scaler = StandardScaler()
      x_train_pca = pd.DataFrame(scaler.fit_transform(x_train_pca), columns = x_pca.columns)
      x_test_pca = pd.DataFrame(scaler.transform(x_test_pca), columns = x_pca.columns)
```

```
[23]: grid_search.fit(x_train_pca, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca)
      logleg_acc_pca = accuracy_score(y_test, y_pred)
      print(logleg_acc_pca)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
0.8216808271061521
Best Parameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

2nd Experiment with PCA

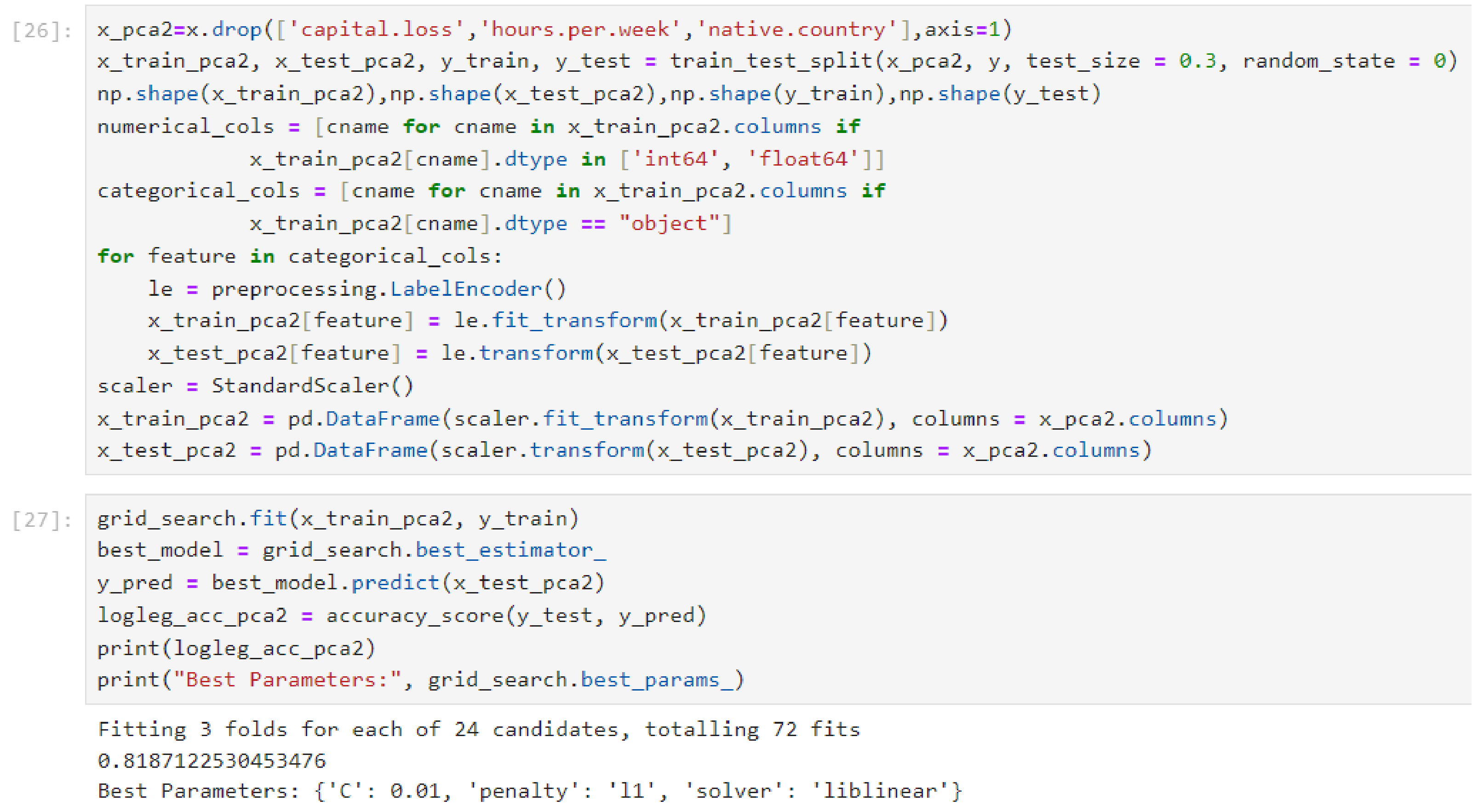


```
[24]: #The same steps but without 3 column
      x_pca1=x.drop(['hours.per.week','native.country'],axis=1)
      x_train_pca1, x_test_pca1, y_train, y_test = train_test_split(x_pca1, y, test_size = 0.3, random_state = 0)
      np.shape(x_train_pca1),np.shape(x_test_pca1),np.shape(y_train),np.shape(y_test)
      numerical_cols = [cname for cname in x_train_pca1.columns if
                  x_train_pca1[cname].dtype in ['int64', 'float64']]
      categorical_cols = [cname for cname in x_train_pca1.columns if
                  x_train_pca1[cname].dtype == "object"]
      for feature in categorical_cols:
          le = preprocessing.LabelEncoder()
          x_train_pca1[feature] = le.fit_transform(x_train_pca1[feature])
          x_test_pca1[feature] = le.transform(x_test_pca1[feature])
      scaler = StandardScaler()
      x_train_pca1 = pd.DataFrame(scaler.fit_transform(x_train_pca1), columns = x_pca1.columns)
      x_test_pca1 = pd.DataFrame(scaler.transform(x_test_pca1), columns = x_pca1.columns)
```

```
[25]: grid_search.fit(x_train_pca1, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca1)
      logleg_acc_pca1 = accuracy_score(y_test, y_pred)
      print(logleg_acc_pca1)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
0.8231139318251612
Best Parameters: {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
```

3rd Experiment with PCA



```
[26]: x_pca2=x.drop(['capital.loss','hours.per.week','native.country'],axis=1)
      x_train_pca2, x_test_pca2, y_train, y_test = train_test_split(x_pca2, y, test_size = 0.3, random_state = 0)
      np.shape(x_train_pca2),np.shape(x_test_pca2),np.shape(y_train),np.shape(y_test)
      numerical_cols = [cname for cname in x_train_pca2.columns if
                  x_train_pca2[cname].dtype in ['int64', 'float64']]
      categorical_cols = [cname for cname in x_train_pca2.columns if
                  x_train_pca2[cname].dtype == "object"]
      for feature in categorical_cols:
          le = preprocessing.LabelEncoder()
          x_train_pca2[feature] = le.fit_transform(x_train_pca2[feature])
          x_test_pca2[feature] = le.transform(x_test_pca2[feature])
      scaler = StandardScaler()
      x_train_pca2 = pd.DataFrame(scaler.fit_transform(x_train_pca2), columns = x_pca2.columns)
      x_test_pca2 = pd.DataFrame(scaler.transform(x_test_pca2), columns = x_pca2.columns)
```

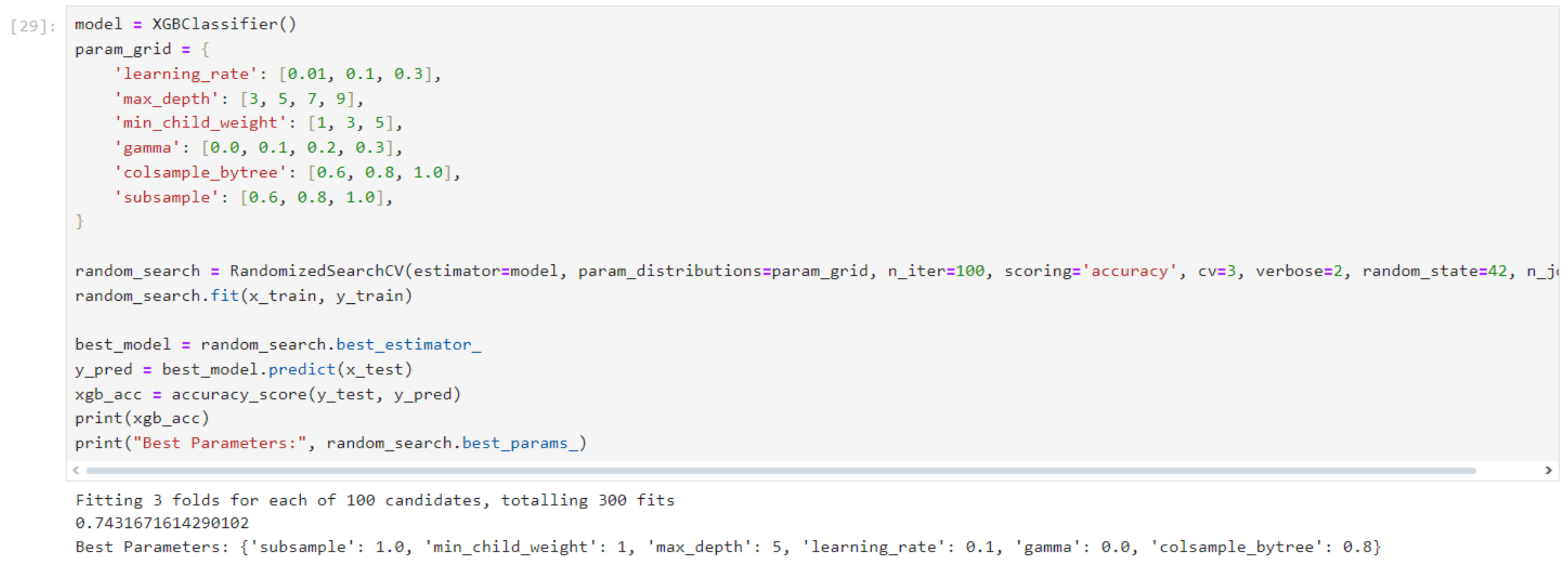
```
[27]: grid_search.fit(x_train_pca2, y_train)
      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca2)
      logleg_acc_pca2 = accuracy_score(y_test, y_pred)
      print(logleg_acc_pca2)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
0.8187122530453476
Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
```

Introduction to XGBoost

- XGBoost is a powerful gradient boosting algorithm known for its speed and performance in structured/tabular data.
- In our experiments, we explore the performance of XGBoost in predicting income levels based on demographic and socioeconomic features.
- We aim to optimize XGBoost hyperparameters and evaluate its accuracy in different scenarios to understand its effectiveness in our predictive modeling task.
- Through XGBoost experiments, we seek to identify the best configuration for maximizing predictive accuracy while minimizing computational resources.

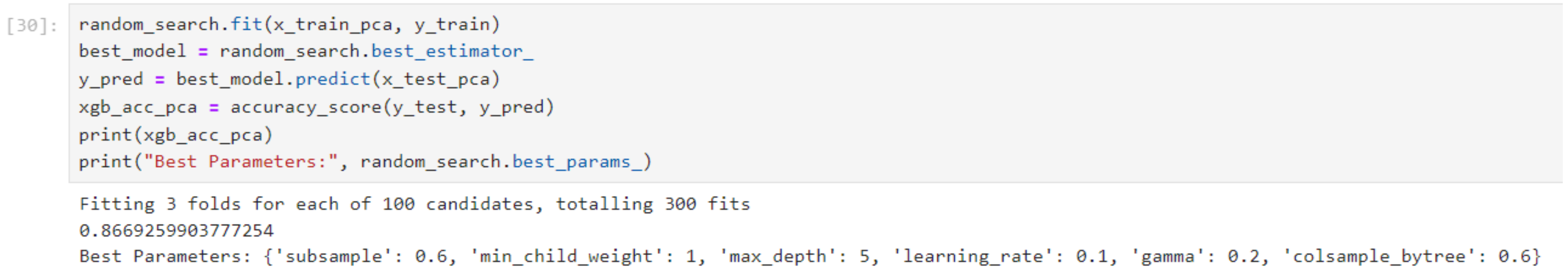
XGBoost in Our Approach



```
[29]: model = XGBClassifier()
      param_grid = {
          'learning_rate': [0.01, 0.1, 0.3],
          'max_depth': [3, 5, 7, 9],
          'min_child_weight': [1, 3, 5],
          'gamma': [0.0, 0.1, 0.2, 0.3],
          'colsample_bytree': [0.6, 0.8, 1.0],
          'subsample': [0.6, 0.8, 1.0],
      }

      random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100, scoring='accuracy', cv=3, verbose=2, random_state=42, n_j
      random_search.fit(x_train, y_train)

      best_model = random_search.best_estimator_
      y_pred = best_model.predict(x_test)
      xgb_acc = accuracy_score(y_test, y_pred)
      print(xgb_acc)
      print("Best Parameters:", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
0.7431671614290102
Best Parameters: {'subsample': 1.0, 'min_child_weight': 1, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.0, 'colsample_bytree': 0.8}
```

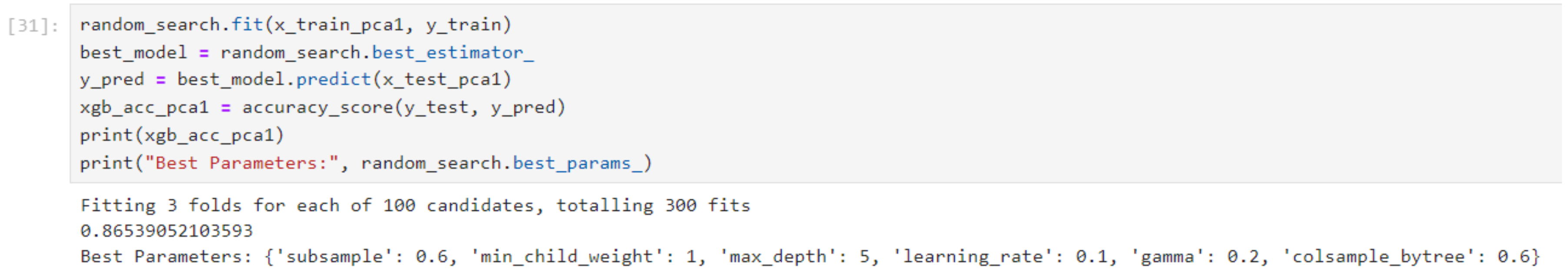
1st Experiment with XGBoost and PCA



```
[30]: random_search.fit(x_train_pca, y_train)
      best_model = random_search.best_estimator_
      y_pred = best_model.predict(x_test_pca)
      xgb_acc_pca = accuracy_score(y_test, y_pred)
      print(xgb_acc_pca)
      print("Best Parameters:", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
0.8669259903777254
Best Parameters: {'subsample': 0.6, 'min_child_weight': 1, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.2, 'colsample_bytree': 0.6}
```

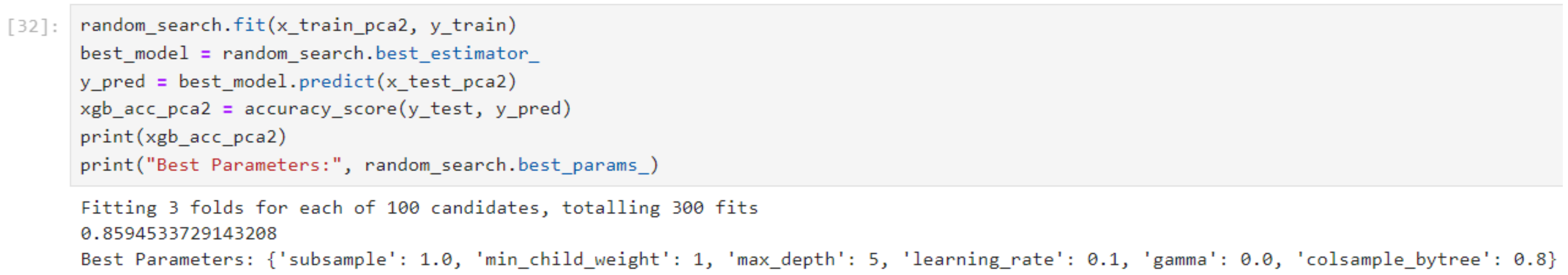
2nd Experiment with XGBoost and PCA



```
[31]: random_search.fit(x_train_pca1, y_train)
      best_model = random_search.best_estimator_
      y_pred = best_model.predict(x_test_pca1)
      xgb_acc_pca1 = accuracy_score(y_test, y_pred)
      print(xgb_acc_pca1)
      print("Best Parameters:", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
0.86539052103593
Best Parameters: {'subsample': 0.6, 'min_child_weight': 1, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.2, 'colsample_bytree': 0.6}
```

3rd Experiment with XGBoost and PCA



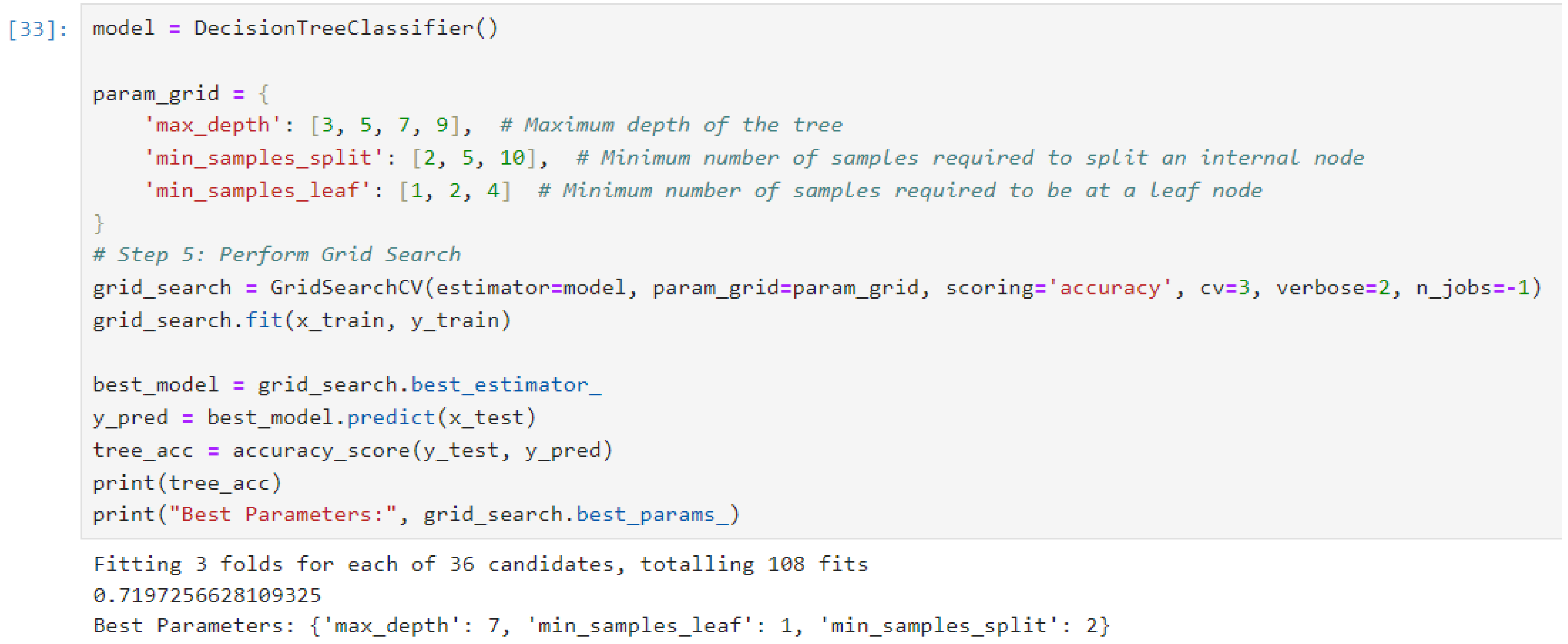
```
[32]: random_search.fit(x_train_pca2, y_train)
      best_model = random_search.best_estimator_
      y_pred = best_model.predict(x_test_pca2)
      xgb_acc_pca2 = accuracy_score(y_test, y_pred)
      print(xgb_acc_pca2)
      print("Best Parameters:", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
0.8594533729143208
Best Parameters: {'subsample': 1.0, 'min_child_weight': 1, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.0, 'colsample_bytree': 0.8}
```

Introduction to Decision Tree

- Decision Tree Experiments involve exploring the performance of the Decision Tree algorithm in predicting income levels based on demographic and socioeconomic features.
- We aim to optimize Decision Tree hyperparameters and evaluate its accuracy under different configurations to understand its effectiveness in our predictive modeling task.
- Through Decision Tree experiments, we seek to identify the best parameter settings for maximizing predictive accuracy while maintaining model interpretability.

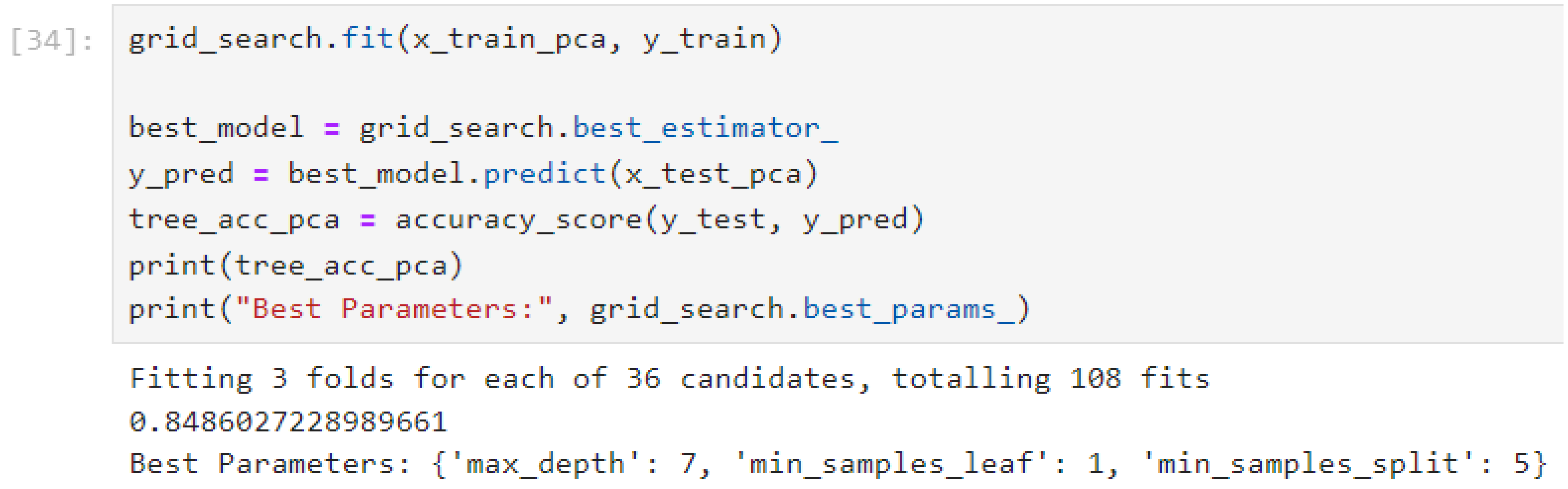
Decision Tree in Our Aproach



```
[33]: model = DecisionTreeClassifier()

      param_grid = {
          'max_depth': [3, 5, 7, 9],  # Maximum depth of the tree
          'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
          'min_samples_leaf': [1, 2, 4]  # Minimum number of samples required to be at a leaf node
      }
      # Step 5: Perform Grid Search
      grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3, verbose=2, n_jobs=-1)
      grid_search.fit(x_train, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test)
      tree_acc = accuracy_score(y_test, y_pred)
      print(tree_acc)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
0.7197256628109325
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

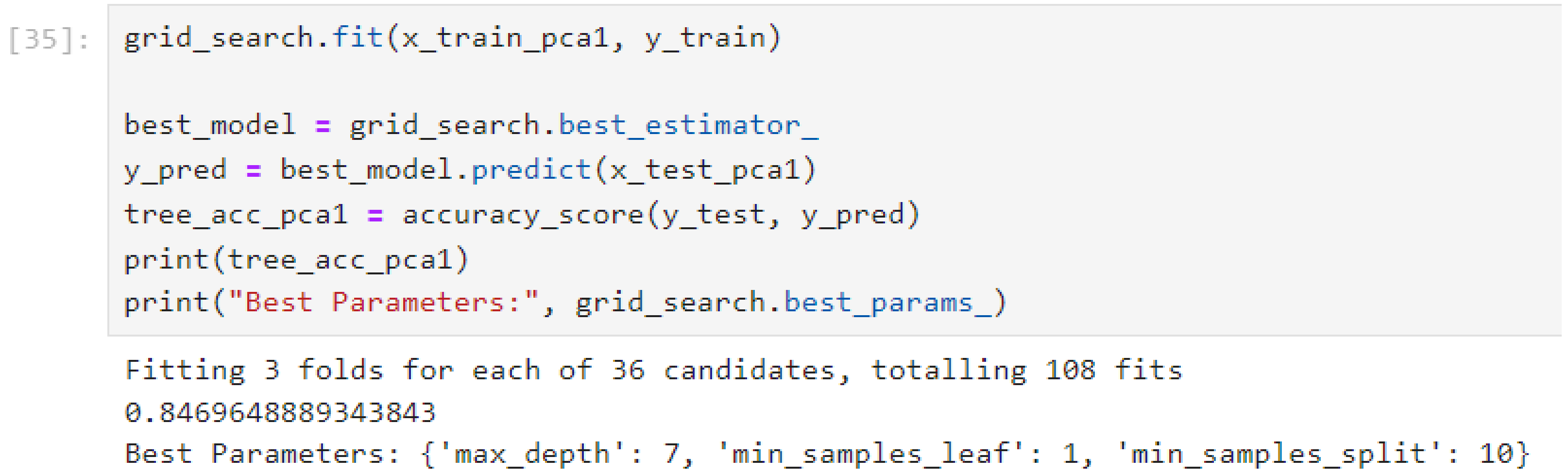
1st Experiment with Decision Tree and PCA



```
[34]: grid_search.fit(x_train_pca, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca)
      tree_acc_pca = accuracy_score(y_test, y_pred)
      print(tree_acc_pca)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
0.8486027228989661
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5}
```

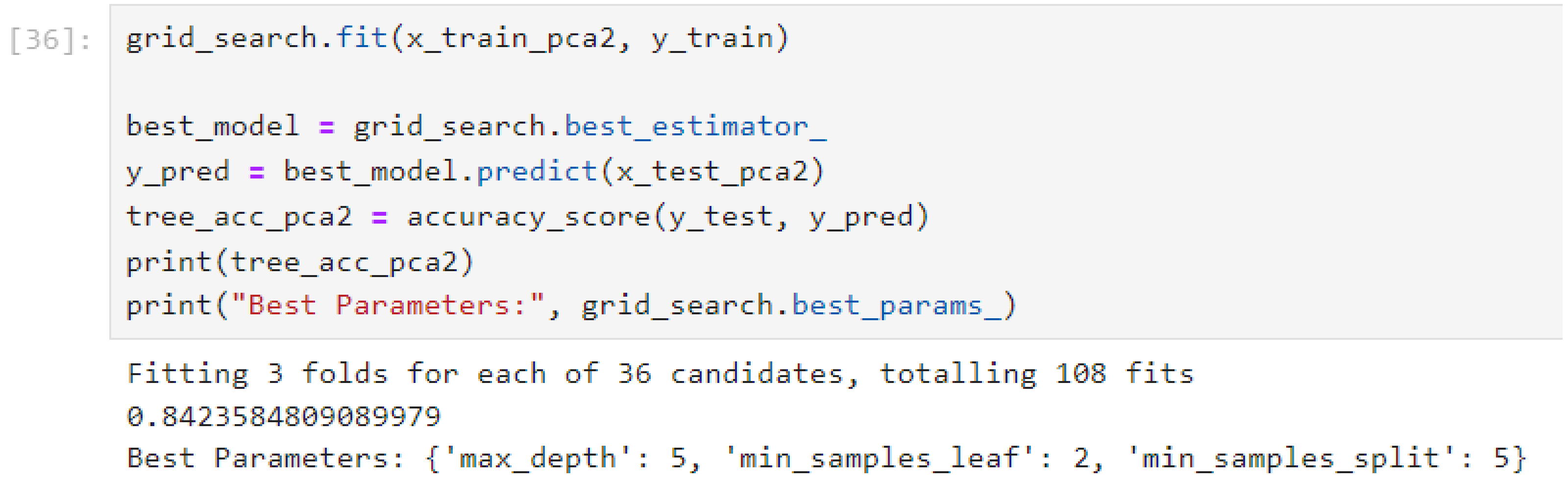
2nd Experiment with Decision Tree and PCA



```
[35]: grid_search.fit(x_train_pca1, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca1)
      tree_acc_pca1 = accuracy_score(y_test, y_pred)
      print(tree_acc_pca1)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
0.8469648889343843
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 10}
```

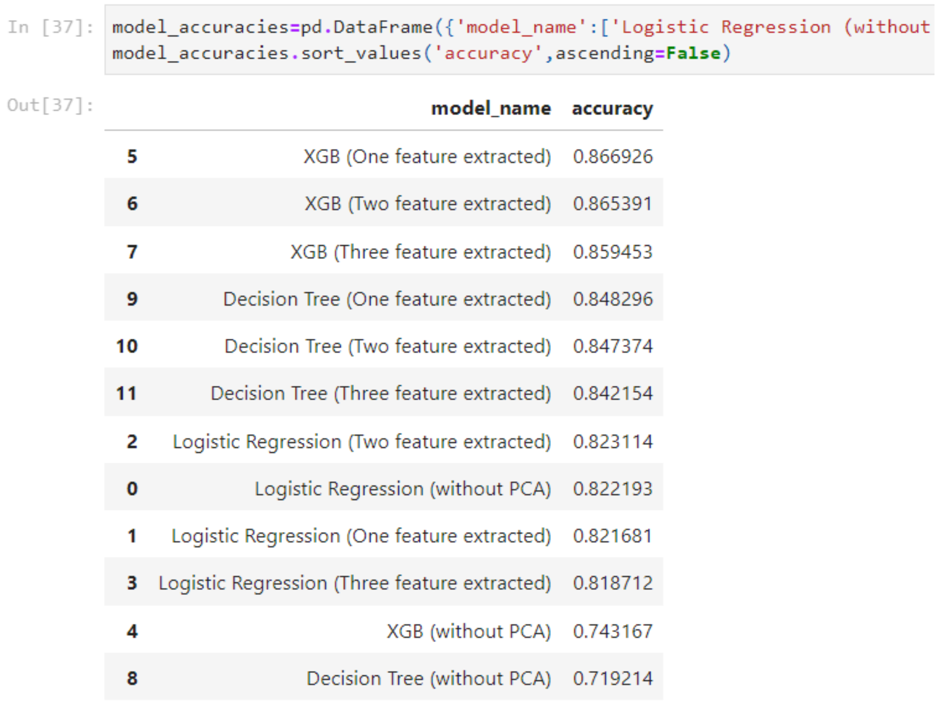
3rd Experiment with Decision Tree and PCA



```
[36]: grid_search.fit(x_train_pca2, y_train)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(x_test_pca2)
      tree_acc_pca2 = accuracy_score(y_test, y_pred)
      print(tree_acc_pca2)
      print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
0.8423584809089979
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 5}
```

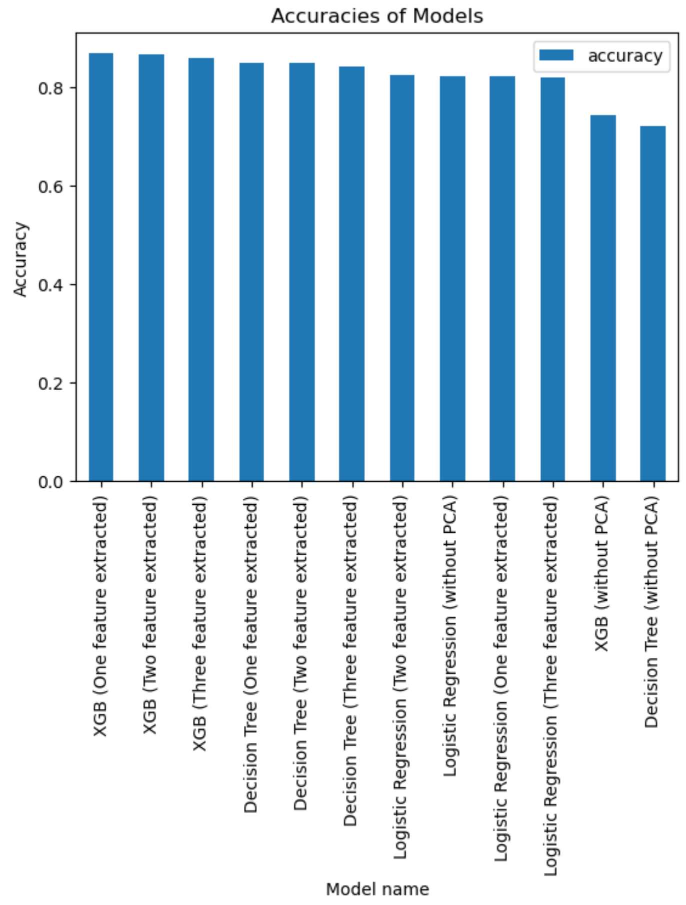
Model Accuracies



```
In [37]: model_accuracies=pd.DataFrame({'model_name':['Logistic Regression (without
         model_accuracies.sort_values('accuracy',ascending=False)
```

Out[37]:

	model_name	accuracy
5	XGB (One feature extracted)	0.866926
6	XGB (Two feature extracted)	0.865391
7	XGB (Three feature extracted)	0.859453
9	Decision Tree (One feature extracted)	0.848296
10	Decision Tree (Two feature extracted)	0.847374
11	Decision Tree (Three feature extracted)	0.842154
2	Logistic Regression (Two feature extracted)	0.823114
0	Logistic Regression (without PCA)	0.822193
1	Logistic Regression (One feature extracted)	0.821681
3	Logistic Regression (Three feature extracted)	0.818712
4	XGB (without PCA)	0.743167
8	Decision Tree (without PCA)	0.719214



Conclusion

- Based on our model accuracies, XGBoost with one feature extracted emerges as the top-performing model.
- XGBoost demonstrates superior predictive power in classifying income levels based on demographic and socioeconomic features.
- The success of XGBoost underscores the importance of feature selection and algorithm optimization in enhancing model performance.

Thanks for attention

For further information or improvement suggestions, please feel free to contact me at samir.guliyev@hotmail.com.