# EXPLORING SOCIOECONOMIC AND HOUSING DYNAMICS WITH MACHINE LEARNING TO PREDICT HOUSEHOLD INCOME

## AGENDA

## RESEARCH QUESTION

**Main Question:**

- Can we predict a household's income level based on demographic, housing, and employment characteristics?

**Objective:**

- To explore and analyze the influence of various socioeconomic and housing factors on the income levels of households using machine learning techniques.

**Importance:**

- Understanding the relationship between these factors and household income can help in policy formulation, targeted interventions, and improving economic research.

## DATA SET OVERVIEW

**Source:**

- Data derived from the **American Community Survey (ACS) 2022**.
- Specifically, the Public Use Microdata Sample (PUMS) files.

**Key Features:**

•**Volume and Scope:**

- The dataset includes **819,228 records and 241 features**.
- For the scope of this study, data for **26 states** was considered, focusing on the most significant variables.

**Selected Features:**

•**Demographics:** Age, Sex, Race, Marital Status.

•**Economic:** Household Income, Personal Income, Employment Status.

•**Housing:** Property Value, Monthly Rent, Number of Bedrooms.

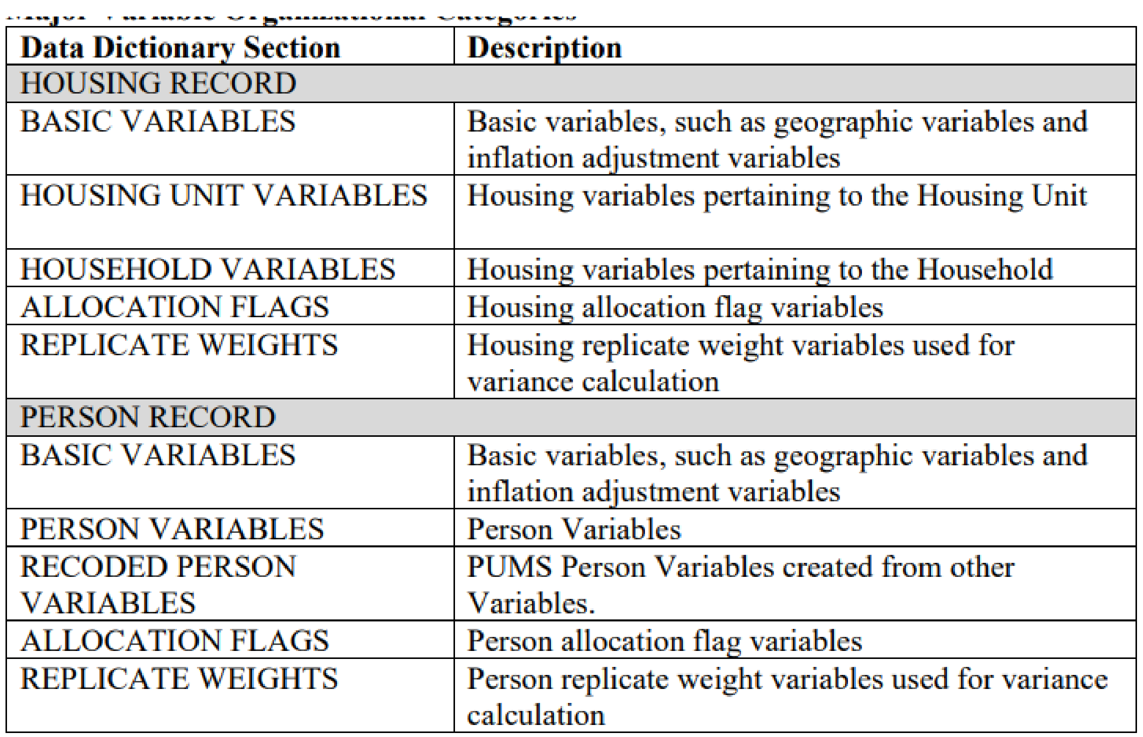
## DATA HANDLING:

**•STORAGE AND ACCESS:**

- Data is stored and managed using GitHub LFS, ensuring efficient handling of large-scale data.

**•PREPROCESSING CONSIDERATIONS:**

- Initial focus on filtering states with more than 50,000 records to ensure robust sample sizes for analysis.



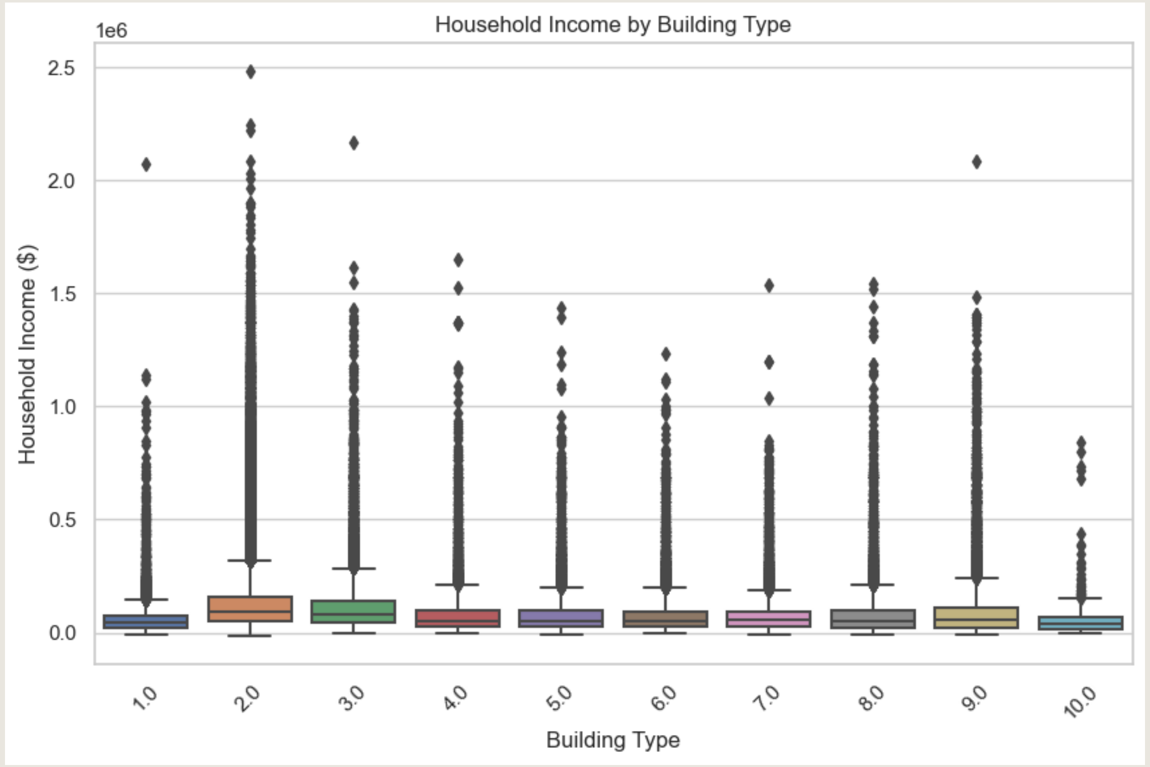
| Data Dictionary Section | Description |
|---|---|
| HOUSING RECORD | |
| BASIC VARIABLES | Basic variables, such as geographic variables and inflation adjustment variables |
| HOUSING UNIT VARIABLES | Housing variables pertaining to the Housing Unit |
| HOUSEHOLD VARIABLES | Housing variables pertaining to the Household |
| ALLOCATION FLAGS | Housing allocation flag variables |
| REPLICATE WEIGHTS | Housing replicate weight variables used for variance calculation |
| PERSON RECORD | |
| BASIC VARIABLES | Basic variables, such as geographic variables and inflation adjustment variables |
| PERSON VARIABLES | Person Variables |
| RECODED PERSON VARIABLES | PUMS Person Variables created from other Variables. |
| ALLOCATION FLAGS | Person allocation flag variables |
| REPLICATE WEIGHTS | Person replicate weight variables used for variance calculation |

## EDA

**Key Insights from Data Visualizations:**

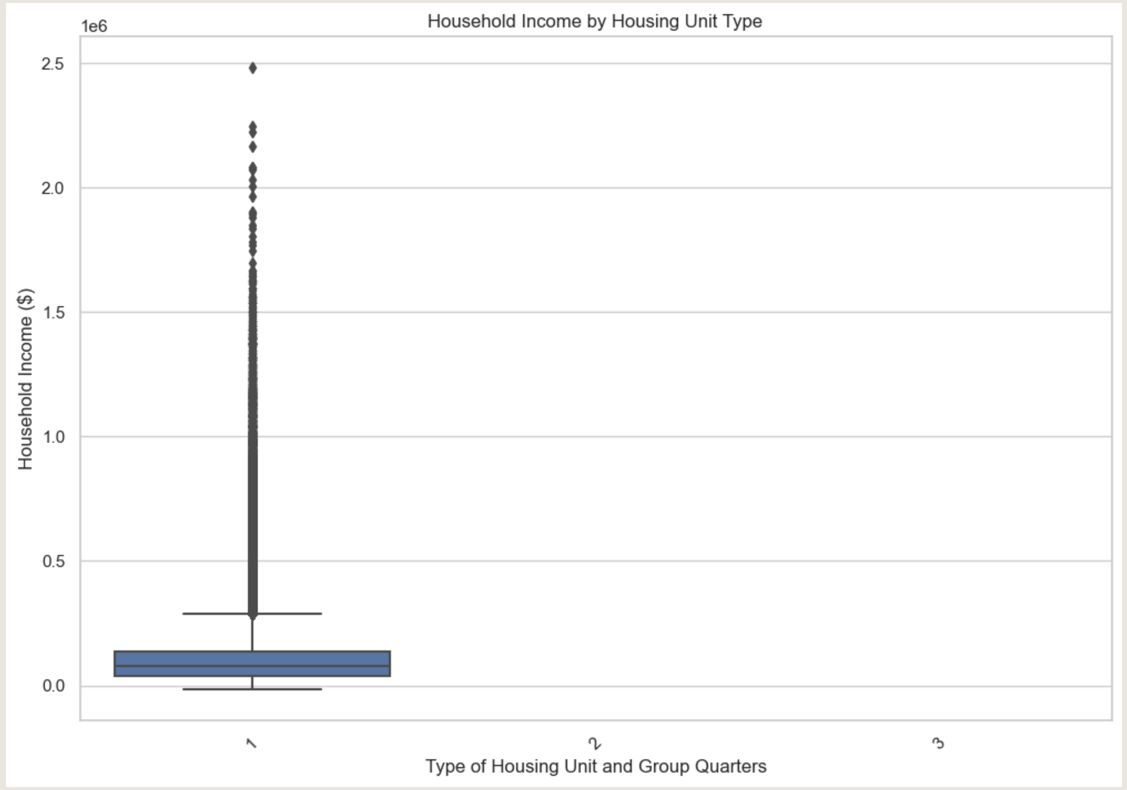
**Household Income by Building Type**:

1. The boxplot indicates a wide variation in income across different building types, with some building types showing more outliers, suggesting higher income variability.
2. Most building types have a similar median income, but the range differs significantly, pointing to a disparity in income distribution within types.



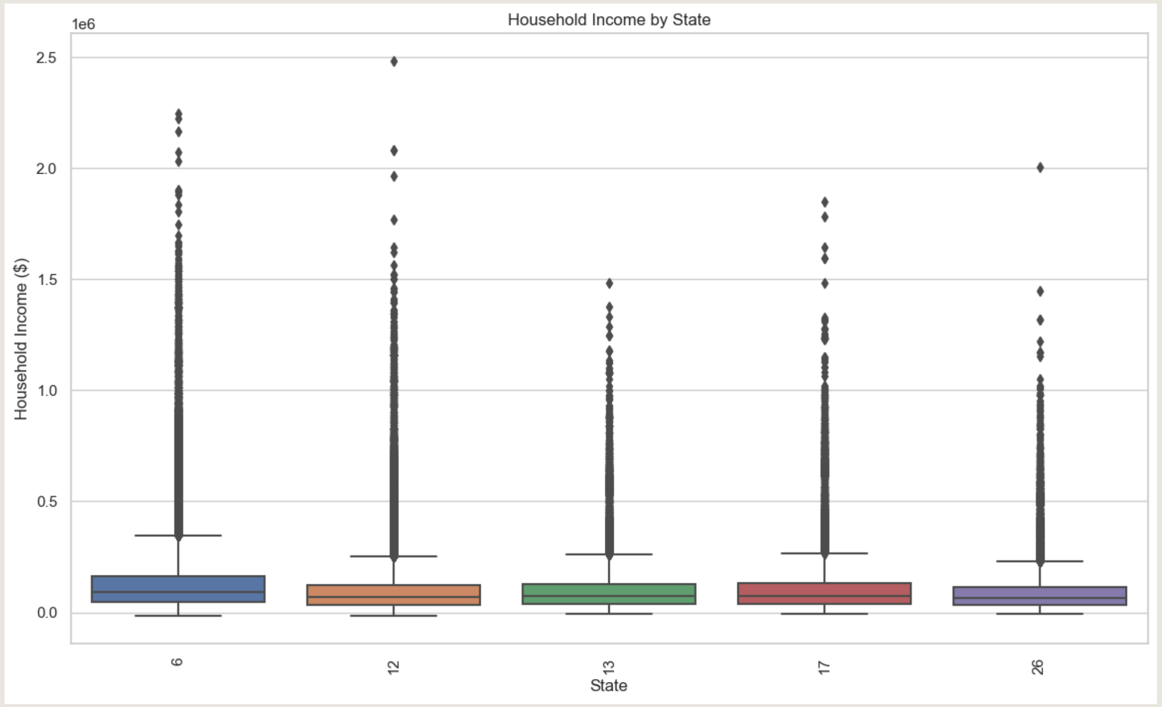
**Household Income by Housing Unit Type**:

1. The boxplot shows a concentrated distribution for most housing unit types, with a small number of high-income outliers.
2. This suggests a general homogeneity in income for the majority of housing types with few exceptions.



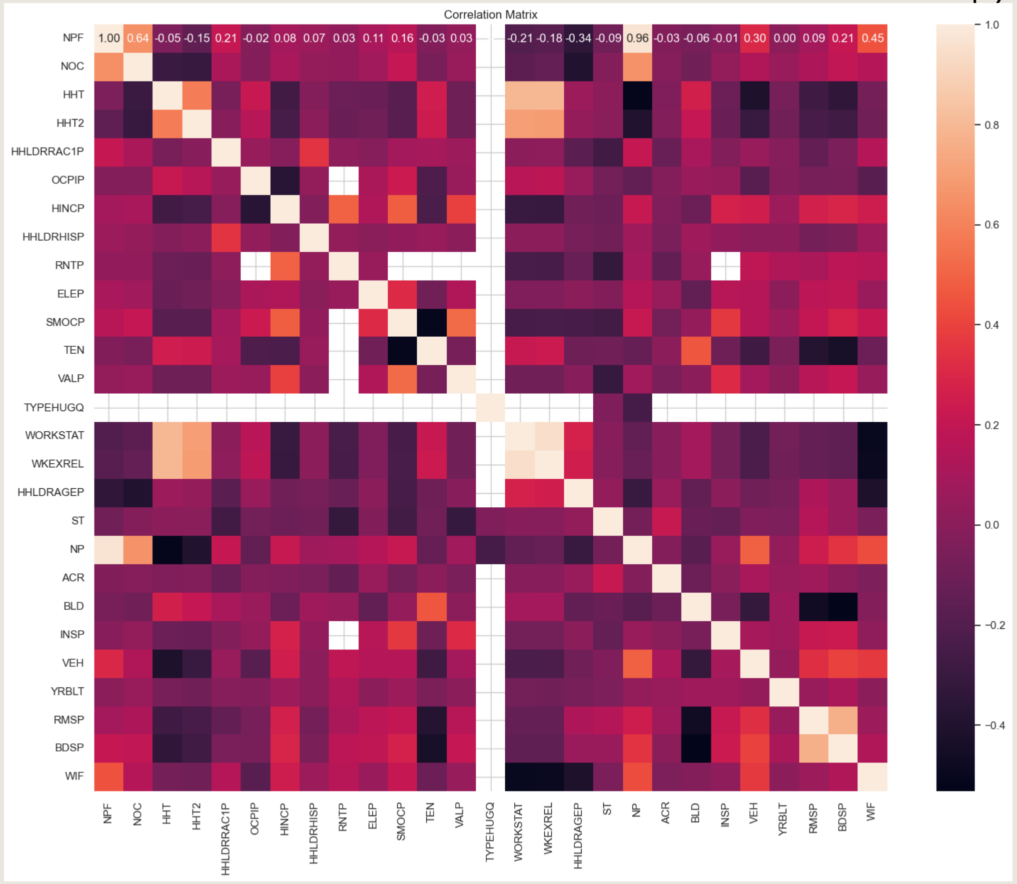
**Income Distribution Across States**:

1. The boxplot across different states reveals that some states have a higher median income and more upper-range outliers.
2. This geographic variation can indicate differing economic conditions or living costs across states.

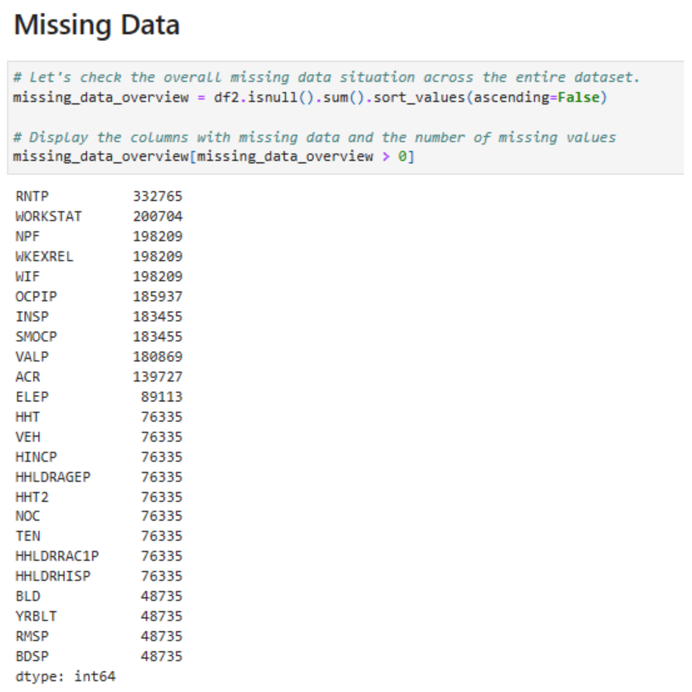


**Correlation Analysis**:

1. The heatmap of the correlation matrix highlights significant relationships between household income and variables like property value, rent price, and type of housing unit.
2. Strong positive correlations between income and property values suggest that as household incomes increase, so do property values.



# DATA PREPARATION AND FEATURE SELECTION



Missing Data

```
# Let's check the overall missing data situation across the entire dataset.
missing_data_overview = df2.isnull().sum().sort_values(ascending=False)

# Display the columns with missing data and the number of missing values
missing_data_overview[missing_data_overview > 0]
```

```
RNTP          332765
WORKSTAT      200704
NPF           198209
WKEXREL       198209
WIF           198209
OCPIP         185937
INSP          183455
SMOCP         183455
VALP          180869
ACR           139727
ELEP           89113
HHT            76335
VEH            76335
HINCP          76335
HHLDRAGEP      76335
HHT2           76335
NOC            76335
TEN            76335
HHLDRRAC1P     76335
HHLDRHISP      76335
BLD            48735
YRBLT          48735
RMSP           48735
BDSP           48735
dtype: int64
```

## Objective:

- To clean and prepare the dataset for modeling by handling missing data, imputing values, removing outliers, and encoding categorical variables.

## Missing Data Handling:

## Overview of Missing Data:

- Significant missing values in columns like **RNTP**, **WORKSTAT**, and **NPF**.
- Missing data proportions range from minor (e.g., **ELEP** 20.2%) to major (e.g., **RNTP** 75.7%).

## Actions Taken:

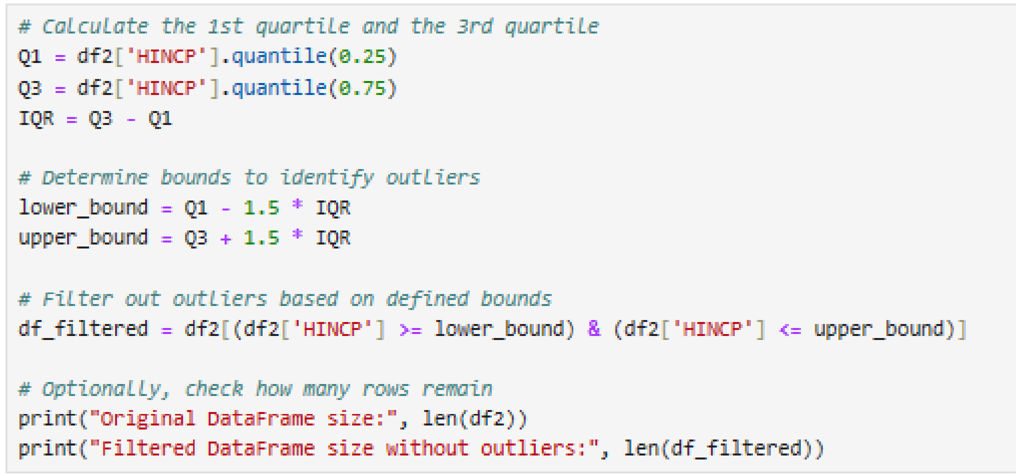
- Columns with very high missing rates like **RNTP** have been handled specifically.
- **HINCP** (Household Income) column missing values were dropped to ensure data integrity for income predictions.

## Imputation:

- **Method:**
  - Used the median for imputing missing numerical values in columns such as **NPF**, **OCPIP**, etc.
  - This method helps retain the central tendency without being affected by outliers.

## Outlier Removal:

- **Process:**
  - Calculated the interquartile range (IQR) and removed outliers beyond 1.5 times the IQR for **HINCP** and other variables.
  - This step helps in normalizing the data distribution and improving model accuracy.

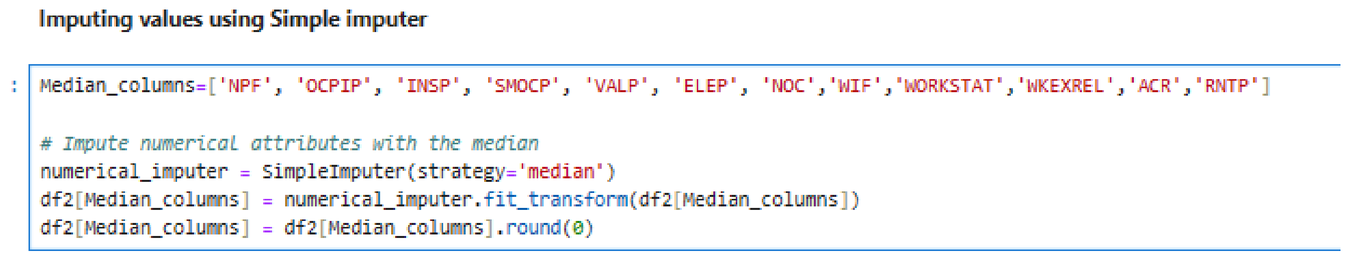


```
# Calculate the 1st quartile and the 3rd quartile
Q1 = df2['HINCP'].quantile(0.25)
Q3 = df2['HINCP'].quantile(0.75)
IQR = Q3 - Q1

# Determine bounds to identify outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers based on defined bounds
df_filtered = df2[(df2['HINCP'] >= lower_bound) & (df2['HINCP'] <= upper_bound)]

# Optionally, check how many rows remain
print("Original DataFrame size:", len(df2))
print("Filtered DataFrame size without outliers:", len(df_filtered))
```



**Imputing values using Simple imputer**

```
Median_columns=['NPF', 'OCPIP', 'INSP', 'SMOCP', 'VALP', 'ELEP', 'NOC','WIF','WORKSTAT','WKEXREL','ACR','RNTP']

# Impute numerical attributes with the median
numerical_imputer = SimpleImputer(strategy='median')
df2[Median_columns] = numerical_imputer.fit_transform(df2[Median_columns])
df2[Median_columns] = df2[Median_columns].round(0)
```

**Feature Engineering:**

•**Income Category:**

•Transformed **HINCP** into categorical bins (**Very Low**, **Low**, **Medium**, **High**, **Very High**) based on quantiles to simplify income analysis.

•**Household Type and Tenure:**

•Created new categorical features like **Family_Type** based on the number of persons and **TEN** based on the type of tenure.

•**Standardization and Encoding:**

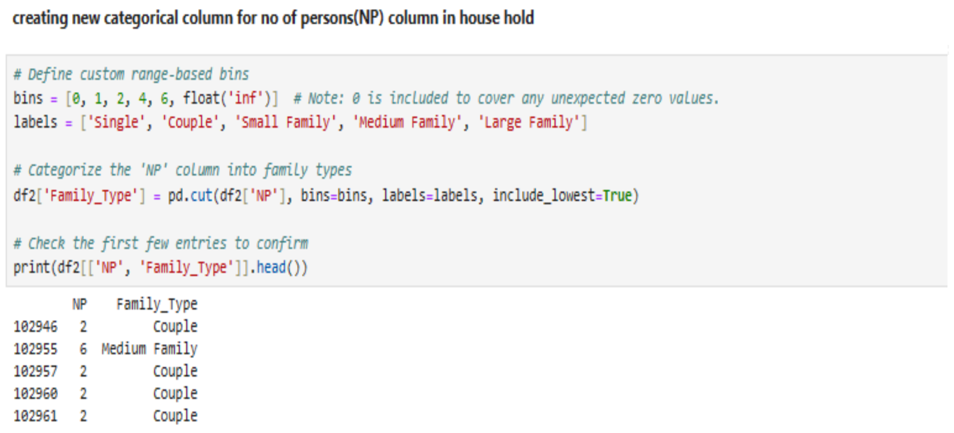
•Standardized numerical features to bring them to a common scale.

•Applied encoding to transform categorical variables for model compatibility.

**Visualization:**

•**Income Distribution:**

•Visualized the log-transformed household income to analyze the distribution after outlier removal and log transformation



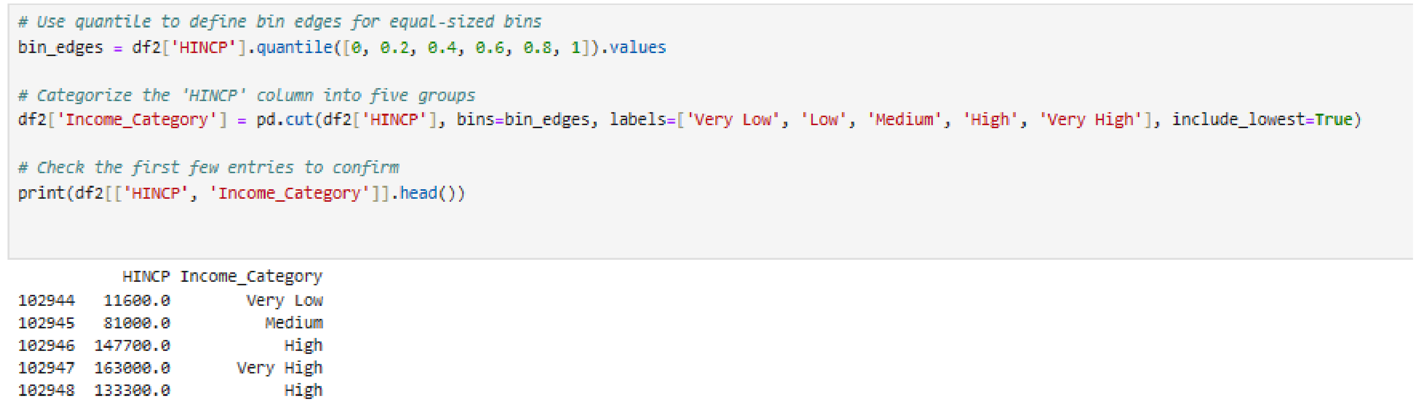
creating new categorical column for no of persons(NP) column in house hold

```
# Define custom range-based bins
bins = [0, 1, 2, 4, 6, float('inf')]  # Note: 0 is included to cover any unexpected zero values.
labels = ['Single', 'Couple', 'Small Family', 'Medium Family', 'Large Family']

# Categorize the 'NP' column into family types
df2['Family_Type'] = pd.cut(df2['NP'], bins=bins, labels=labels, include_lowest=True)

# Check the first few entries to confirm
print(df2[['NP', 'Family_Type']].head())
```

```
        NP    Family_Type
102946   2         Couple
102955   6  Medium Family
102957   2         Couple
102960   2         Couple
102961   2         Couple
```



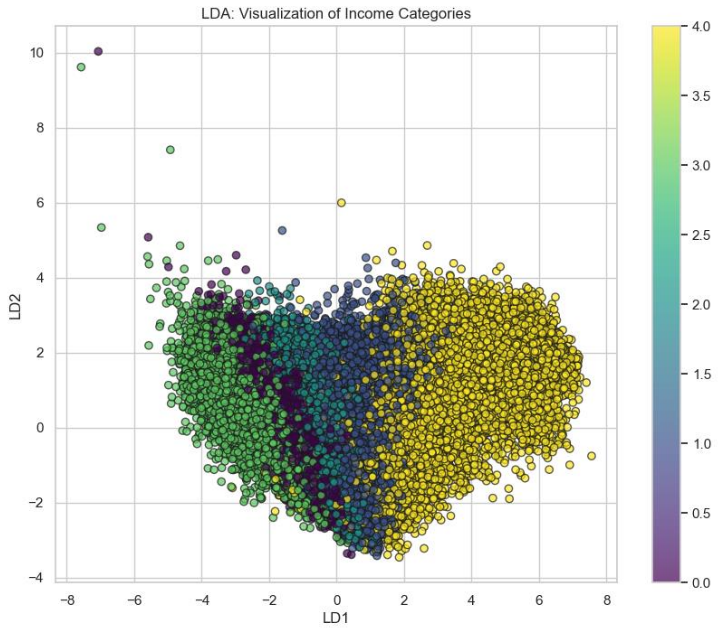
```
# Use quantile to define bin edges for equal-sized bins
bin_edges = df2['HINCP'].quantile([0, 0.2, 0.4, 0.6, 0.8, 1]).values

# Categorize the 'HINCP' column into five groups
df2['Income_Category'] = pd.cut(df2['HINCP'], bins=bin_edges, labels=['Very Low', 'Low', 'Medium', 'High', 'Very High'], include_lowest=True)

# Check the first few entries to confirm
print(df2[['HINCP', 'Income_Category']].head())
```

```
           HINCP Income_Category
102944   11600.0        Very Low
102945   81000.0          Medium
102946  147700.0            High
102947  163000.0       Very High
102948  133300.0            High
```

## FEATURE SELECTION



Feature selection is a crucial step to identify the most relevant predictors for our model. In this section, we explore two methods: Linear Discriminant Analysis (LDA) and Random Forest Feature Selection.

Linear Discriminant Analysis (LDA):

•LDA is a dimensionality reduction technique used for feature extraction.

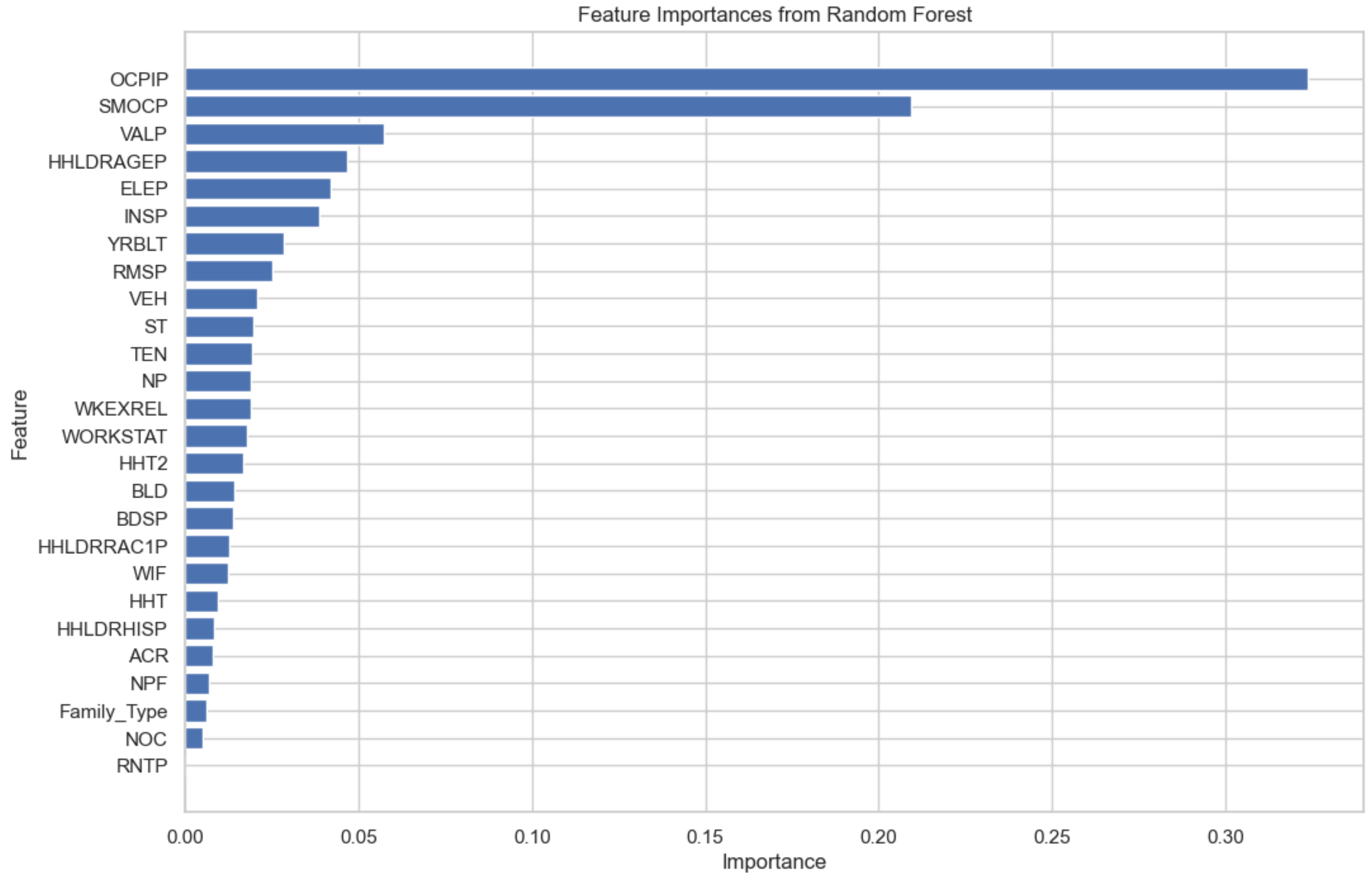
•We applied LDA to the dataset with the goal of visualizing the separation of income categories.

•The number of samples in our dataset is 200,428, with 26 features considered.

## RandomForest feature Selection

1. Random Forest Feature Selection:

- Random Forest is a popular ensemble learning method that provides a feature importance ranking.
- We trained a Random Forest classifier to determine the importance of each feature in predicting income categories.
- The top features identified by Random Forest include OCPIP, SMOCP, VALP, and others, indicating their significance in our model.



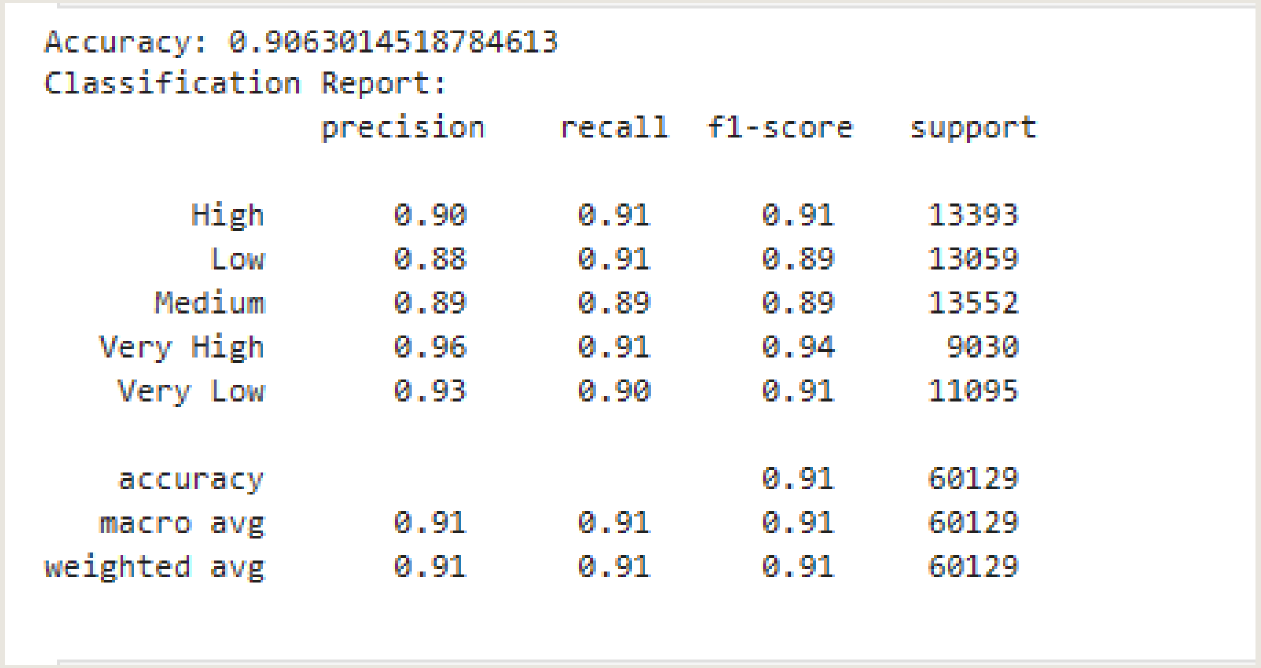
## MODELS

we explore various machine learning models for predicting income categories. We assess the performance of each model and provide insights into their strengths and weaknesses.

We have used:

- RandomForest Model
- XGBoost Model
- Neural Network Model

## RANDOM FOREST MODEL



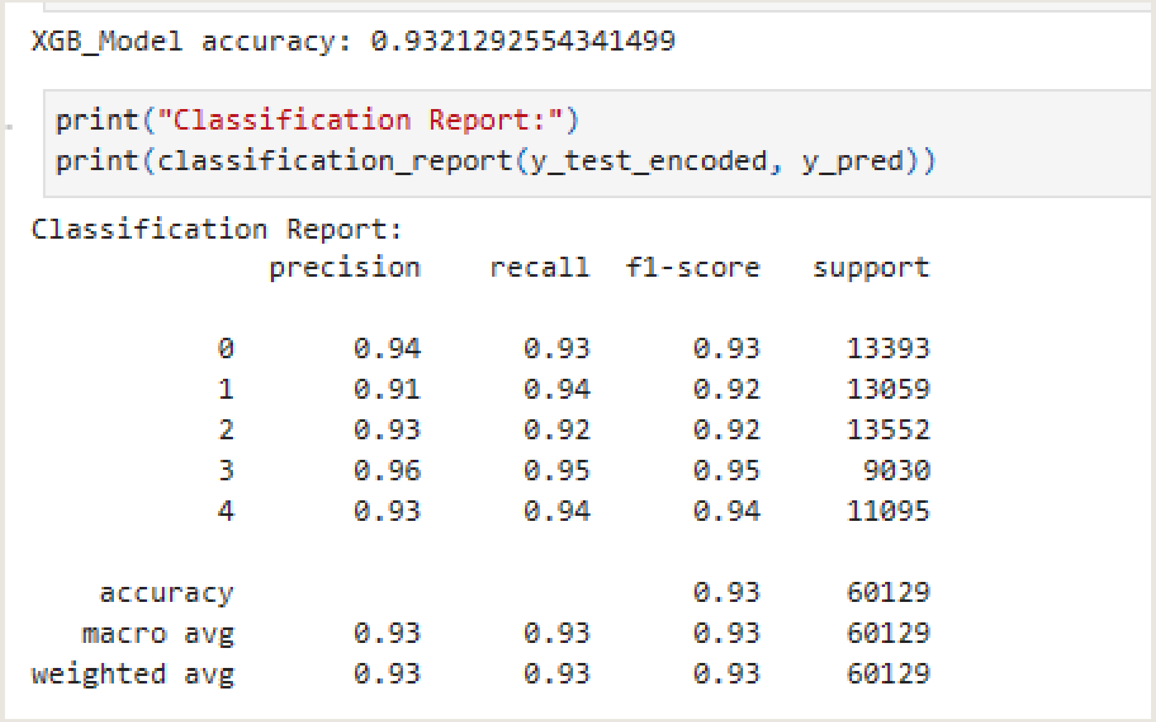
```
Accuracy: 0.9063014518784613
Classification Report:
               precision    recall  f1-score   support

        High       0.90      0.91      0.91     13393
         Low       0.88      0.91      0.89     13059
      Medium       0.89      0.89      0.89     13552
   Very High       0.96      0.91      0.94      9030
    Very Low       0.93      0.90      0.91     11095

    accuracy                           0.91     60129
   macro avg       0.91      0.91      0.91     60129
weighted avg       0.91      0.91      0.91     60129
```

- We utilized a RandomForestClassifier with 100 estimators to predict income categories.
- The model achieved an accuracy of approximately 91% on the test set.
- Precision, recall, and F1-score metrics were computed for each income category, showing consistent performance across classes.
- The confusion matrix visually represents the model's performance in classifying true and predicted labels.

## XG BOOST MODEL



```
XGB_Model accuracy: 0.9321292554341499
```

```
print("Classification Report:")
print(classification_report(y_test_encoded, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.93     13393
           1       0.91      0.94      0.92     13059
           2       0.93      0.92      0.92     13552
           3       0.96      0.95      0.95      9030
           4       0.93      0.94      0.94     11095

    accuracy                           0.93     60129
   macro avg       0.93      0.93      0.93     60129
weighted avg       0.93      0.93      0.93     60129
```

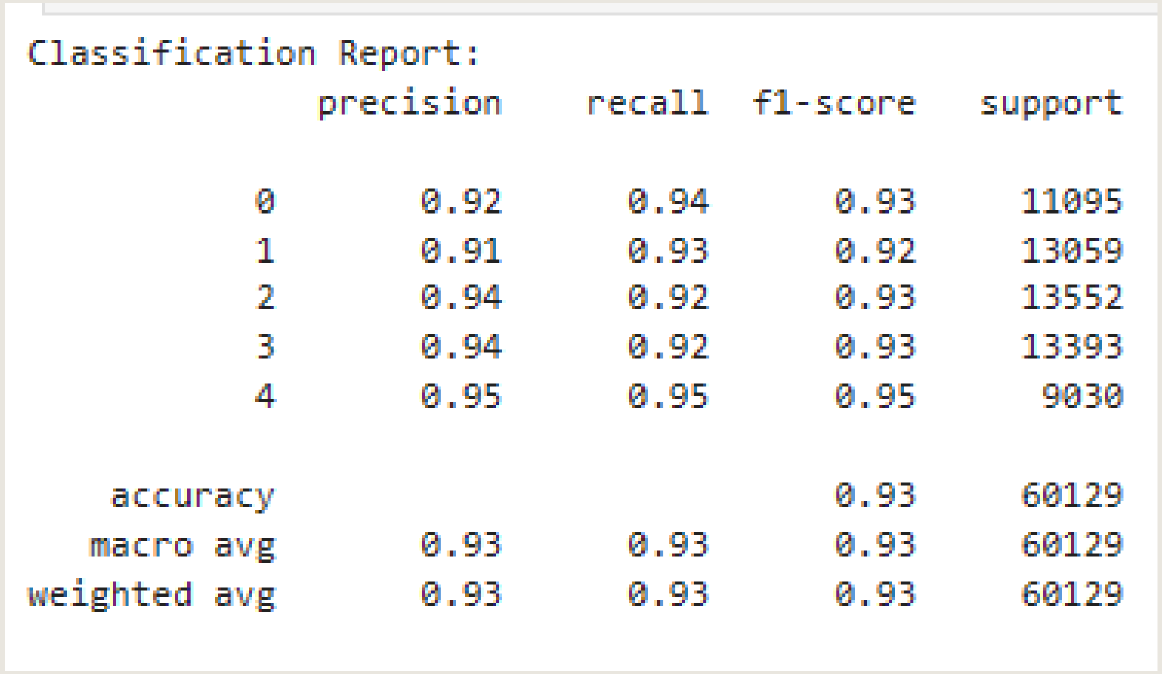
•An XG Boost classifier was trained using encoded target variables.

•The XG Boost model demonstrated higher accuracy compared to RandomForest, reaching approximately 93% on the test set.

•Like the RandomForest model, precision, recall, and F1-score metrics were computed for each income category.

•The confusion matrix provides a detailed overview of the model's classification performance.

# NEURAL NETWORK MODEL



```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     11095
           1       0.91      0.93      0.92     13059
           2       0.94      0.92      0.93     13552
           3       0.94      0.92      0.93     13393
           4       0.95      0.95      0.95      9030

    accuracy                           0.93     60129
   macro avg       0.93      0.93      0.93     60129
weighted avg       0.93      0.93      0.93     60129
```

•A neural network model was constructed using TensorFlow and Keras.

•The model consists of multiple layers with ReLU activation functions and softmax activation in the output layer.

•Training history indicates the model's progression in accuracy and loss over epochs.

•The model achieved an accuracy of around 93% on the test set, demonstrating competitive performance.

## PREFERRED MODEL

1. Accuracy Comparison:
    1. Random Forest: Achieved an accuracy of approximately 91%.
    2. XG Boost: Outperformed Random Forest with an accuracy of around 93%.
    3. Neural Network: Demonstrated competitive performance with an accuracy of about 93%.
2. Precision, Recall, and F1-score:
    1. All models exhibited consistent precision, recall, and F1-score across different income categories.
    2. Precision measures the model's ability to avoid false positives, recall measures its ability to find all relevant cases, and F1-score provides a balance between precision and recall.
3. Computational Complexity:
    1. Random Forest and XG Boost are ensemble methods and generally faster to train compared to neural networks.
    2. Neural networks, especially deep architectures, may require more computational resources and longer training times.

5. Interpretability:

    1. Random Forest and XG Boost offer feature importances, which provide insights into the importance of each feature for prediction.
    2. Neural networks, particularly deep architectures, are often considered black-box models, making interpretation challenging.

6. Deployment and Scalability:

    1. Random Forest and XG Boost models are easier to deploy and scale due to their simplicity and efficiency.
    2. Neural networks may require specialized hardware for deployment and might not scale well with large datasets.

Based on these considerations, the **XG Boost** model emerges as the preferred choice due to its superior accuracy, efficient training, and interpretability. However, the final decision should also consider factors such as deployment environment, computational resources, and the need for model interpretability.

## ENSEMBLE MODEL

The implementation of an ensemble model combining Random Forest, XG Boost, and Neural Network classifiers, along with the evaluation of its performance.

1.Ensemble Model Construction:

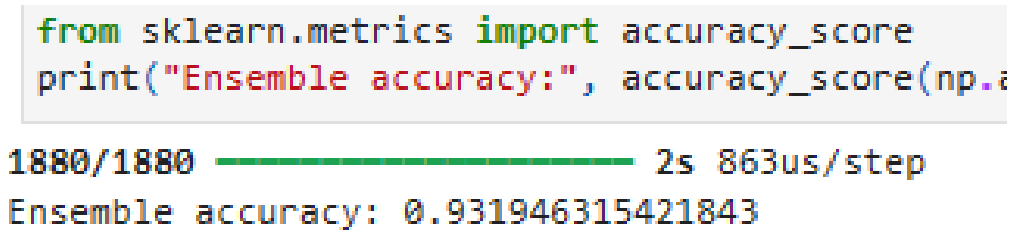
1. We combine predictions from Random Forest, XG Boost, and Neural Network classifiers to create an ensemble model.
2. Each model provides probability estimates for each class, which are averaged to generate ensemble predictions.

2.Model Configuration and Training:

1. Random Forest and XG Boost models are trained using their respective classifiers with suitable hyperparameters.
2. The Neural Network architecture consists of multiple dense layers with ReLU activation, trained using the Adam optimizer.

3. Performance Evaluation:

1. Ensemble accuracy: 93.19%
2. We achieve a high accuracy by combining the predictions of individual models.
3. Ensemble methods often lead to improved generalization and robustness compared to individual models.



```
from sklearn.metrics import accuracy_score
print("Ensemble accuracy:", accuracy_score(np.a
```

```
1880/1880 ━━━━━━━━━━━━━━━━━━━━ 2s 863us/step
Ensemble accuracy: 0.931946315421843
```

## CONCLUSION

Our analysis included the implementation of various algorithms such as XGBoost, Random Forest, and Neural Networks. Each model underwent preprocessing steps, including one-hot encoding for categorical variables and scaling for numerical features, ensuring consistent data handling across the pipeline.

Upon evaluation, our models demonstrated promising performance, with XGBoost achieving an accuracy of approximately 93.21%, Random Forest achieving 90.63%, and the Neural Network reaching 93.17%. Additionally, our ensemble model, combining XGBoost and Random Forest, achieved an accuracy of 93.19%. These results indicate the effectiveness of our approach in accurately predicting the target variable.

THANK YOU