SUMMARY REPORT

The Data Cleaning Process

Overview

The dataset consists of demographic attributes of individuals and their corresponding income levels (whether above or below USD 50K per year). The data cleaning involved handling missing values, encoding categorical variables, and standardizing features to prepare them for model training.

Steps Involved: -

1. **Loading and Inspecting the Data**

The dataset was loaded using pandas. read_csv (). This method reads the CSV file into a data frame, making it easy to perform various data operations.

The dataset was initially inspected using methods like data. head (), data.info (), and data. describe () to understand its structure, data types, and summary statistics.

2. **Handling Missing Values**

Missing values can significantly impact model performance. The presence of missing values was checked using data. isnull().sum().

3. **Encoding Categorical Variables**

Machine learning models require numerical input, so categorical variables were encoded using LabelEncoder from sklearn. preprocessing.

4. **Standardizing Numerical Features**

Standardization ensures that numerical features have a mean of 0 and a standard deviation of 1. This step is crucial for algorithms like k-NN.
StandardScaler from sklearn. preprocessing was used to transform the numerical features.

5. **Splitting Data**

The dataset was split into training and testing sets using train_test_split from sklearn.model_selection, with an 80-20 ratio. This ensures that the models can be evaluated on unseen data.

Model Training and Evaluation

1. Naive Bayes Classifier:

Best Parameters: Naive Bayes doesn't require tuning of hyperparameters.

Training and Evaluation:

```
# Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
accuracy_nb = accuracy_score(y_test, y_pred_nb)
f1_nb = f1_score(y_test, y_pred_nb)
print(f'Naive Bayes - Accuracy: {accuracy_nb}, F1 Score: {f1_nb}')
```

```
Naive Bayes - Accuracy: 0.7990173499155535, F1 Score: 0.4345572354211663
```

2. k-NN Classifier:

- **Best Parameters**: {'n_neighbors': 5}
- **Training and Evaluation**

```
# k-NN Classifier
param_grid = {'n_neighbors': [3, 5, 7]}
knn = KNeighborsClassifier()
grid_search_knn = GridSearchCV(knn, param_grid, cv=5)
grid_search_knn.fit(X_train, y_train)
best_knn = grid_search_knn.best_estimator_
y_pred_knn = best_knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
print(f'k-NN - Accuracy: {accuracy_knn}, F1 Score: {f1_knn}')
```

```
k-NN - Accuracy: 0.7875019192384461, F1 Score: 0.3977371627502176
```

3. Decision Tree Classifier:

- **Best Parameters**: {'max_depth': 10}
- **Training and Evaluation**:

```
# Decision Tree Classifier
param_grid = {'max_depth': [None, 10, 20]}
dt = DecisionTreeClassifier()
grid_search_dt = GridSearchCV(dt, param_grid, cv=5)
grid_search_dt.fit(X_train, y_train)
best_dt = grid_search_dt.best_estimator_
y_pred_dt = best_dt.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)
print(f'Decision Tree - Accuracy: {accuracy_dt}, F1 Score: {f1_dt}')
```

```
Decision Tree - Accuracy: 0.8593582066635959, F1 Score: 0.6721546170365069
```

4. Random Forest Classifier:

- **Best Parameters**: {'n_estimators': 200, 'max_depth': None}
- **Training and Evaluation**.

```
# Random Forest Classifier
param_grid = {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]}
rf = RandomForestClassifier()
grid_search_rf = GridSearchCV(rf, param_grid, cv=5)
grid_search_rf.fit(X_train, y_train)
best_rf = grid_search_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
print(f'Random Forest - Accuracy: {accuracy_rf}, F1 Score: {f1_rf}')
```

```
Random Forest - Accuracy: 0.8668816213726394, F1 Score: 0.6875675675675677
```

3. Discussion and Conclusion of Results

i) **Model Performance Comparison**:

- **Random Forest**: Achieved the highest accuracy (0.866) and F1 score (0.687), indicating its robustness in handling complex data and reducing overfitting through ensemble learning.
- **Decision Tree**: Performed well with an accuracy of (0.859) and F1 score of (0.672). It's simpler than Random Forest but can overfit on training data.

- **k-NN**: Showed good performance with an accuracy of (0.787) and F1 score of (0.397), heavily influenced by the choice of n_neighbors.
- **Naive Bayes**: Performed well with an accuracy of (0.799) and F1 score of (0.434).

ii) **Insights**

Ensemble methods like Random Forest generally perform better due to their ability to capture more complex patterns and reduce overfitting. Simpler models like Naive Bayes can be useful for quick assessments and when computational resources are limited, but they may not always capture the intricacies of the data. Decision Trees are interpretable but can benefit from pruning and ensemble methods to improve their generalizability. k-NN requires careful selection of the k parameter and benefits from standardized features.

4. Challenges and Recommendations

i) **Challenges**

- Properly addressing missing values is crucial to avoid data leakage and biased model performance.
- Ensuring that categorical data is appropriately encoded to retain its informative value while being compatible with machine learning algorithms.
- Hyperparameter tuning, especially for complex models like Random Forest, is computationally intensive and requires significant resources.
- Dealing with imbalanced classes to ensure that the model does not become biased towards the majority class.

ii) **Recommendations**:

- Further exploration and creation of new features could enhance model performance. Techniques such as polynomial features, interaction terms, and domain-specific features should be considered.