capstone-project

September 14, 2023

DATA LOADING AND INSPECTION

[37]:
```
import pandas as pd
import seaborn as sns
#reading the first dataset and finding out the shape i.e no of columns and rows
df1 = pd.read_csv("Credit_card.csv")
df1.shape
```

[37]:
```
(1548, 18)
```

[38]:
```
df1.columns
```

[38]:
```
Index(['Ind_ID', 'GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN',
       'Annual_income', 'Type_Income', 'EDUCATION', 'Marital_status',
       'Housing_type', 'Birthday_count', 'Employed_days', 'Mobile_phone',
       'Work_Phone', 'Phone', 'EMAIL_ID', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

[39]:
```
df2 = pd.read_csv("Credit_card_label.csv")
df2.shape
#reading the second dataset and finding out the shape i.e no of columns and rows
```

[39]:
```
(1548, 2)
```

[40]:
```
df2.columns
```

[40]:
```
Index(['Ind_ID', 'label'], dtype='object')
```

[41]:
```
#proves there is same number of rows
#finding out the common column in both datasets and proves 'Ind_ID' is the
 ↪common column

common_column = df1.columns.intersection(df2.columns)
common_column
```

[41]:
```
Index(['Ind_ID'], dtype='object')
```

[42]:
```python
#confirming both the 'Ind_id' data are equal
are_equal = df1['Ind_ID'].equals(df2['Ind_ID'])
are_equal
```

[42]:
```
True
```

[43]:
```python
#total data after merging is assigned to 'df'

df = pd.merge(df1, df2, on='Ind_ID')
```

DATA PREPROCESSING

[44]:
```python
df = df.rename(columns={'GENDER':'Gender'})
```

[45]:
```python
#need to calculate age from Birthday_count column
from datetime import datetime, timedelta
current_date = datetime(2023, 9, 7)

# Convert 'Birthday_count' to birth dates
df['Birthdate'] = current_date + pd.to_timedelta(df['Birthday_count'], unit='D')

# Calculate age based on the current date
df['Age'] = (current_date - df['Birthdate']).dt.days // 365
```

[46]:
```python
#calculated the employed years from the employed days data toprevent negative␣
↪data
df['Employed_years'] = df['Employed_days'] / 365

# You can round the result to 2 decimal places, for example
df['Employed_years'] = df['Employed_years'].round(2)
```

[47]:
```python
columns_to_drop =␣
↪['Ind_ID','EMAIL_ID','Birthdate','Birthday_count','Work_Phone','Phone','Mobile_phone','CHIL
df.drop(columns_to_drop,axis = 1,inplace=True)
```

[48]:
```python
df.isna().sum()/df.shape[0]*100
#all the missing values percentage is lessthan or close to 30% will be replaced␣
↪based on outliers
```

[48]:
```
Gender            0.452196
Car_Owner         0.000000
Propert_Owner     0.000000
Annual_income     1.485788
Type_Income       0.000000
EDUCATION         0.000000
Marital_status    0.000000
Housing_type      0.000000
```

```
Type_Occupation    31.524548
Family_Members      0.000000
label               0.000000
Age                 1.421189
Employed_years      0.000000
dtype: float64
```

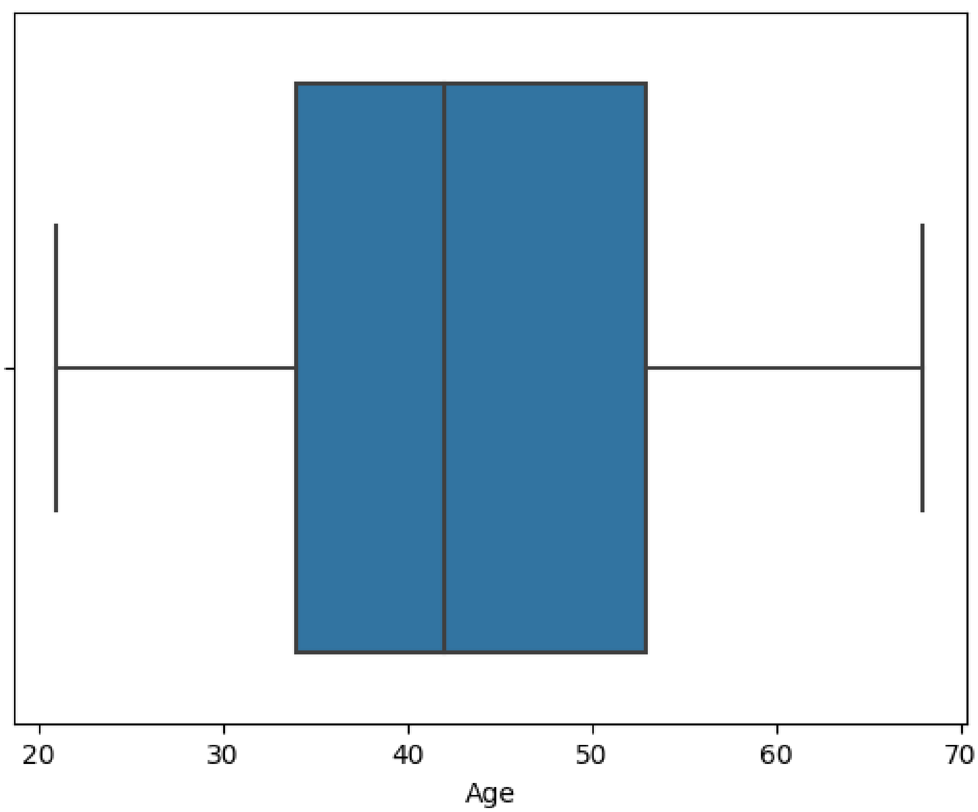
[49]:
```
#As "Type_occupation" column has more missing data and is categorical,replaced␣
↪missing values with other
import numpy as np
df.Type_Occupation.replace(np.nan, 'Other', inplace = True)
```

[50]:
```
#as "gender" is categorical, we replace it with mode values
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
```

EXPLORATORY DATA ANALYSIS(EDA):

[51]:
```
#as "Age" is continous and has missing data, we replace with mean function
sns.boxplot(x=df['Age'])
#As no outliers are available, we replace with mean
```

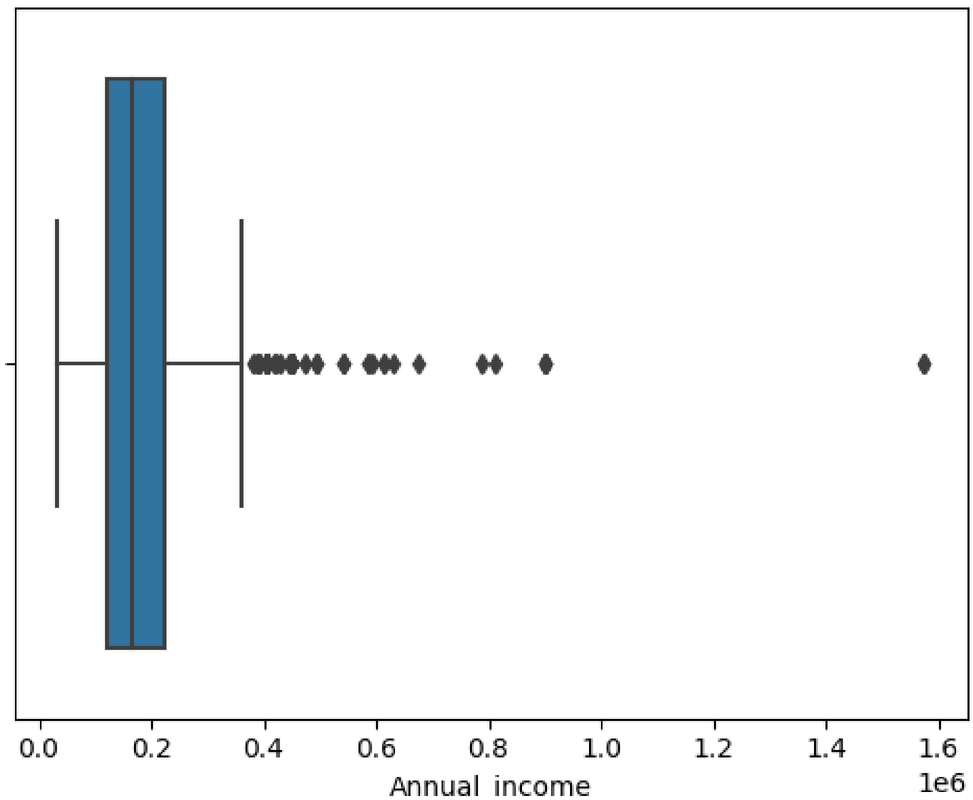
[51]: <Axes: xlabel='Age'>



[52]:
```
df['Age'].fillna(df['Age'].mean(),inplace=True)
```

[53]:
```
sns.boxplot(x=df['Annual_income']) #proves annual income has many outliers
```

[53]:
```
<Axes: xlabel='Annual_income'>
```



[54]:
```
df['Annual_income'].fillna(df['Annual_income'].median(),inplace=True)
```

[55]:
```
#confiming there are no check missing values are present
df.isnull().sum()
```

[55]:
```
Gender            0
Car_Owner         0
Propert_Owner     0
Annual_income     0
Type_Income       0
EDUCATION         0
Marital_status    0
Housing_type      0
Type_Occupation   0
```

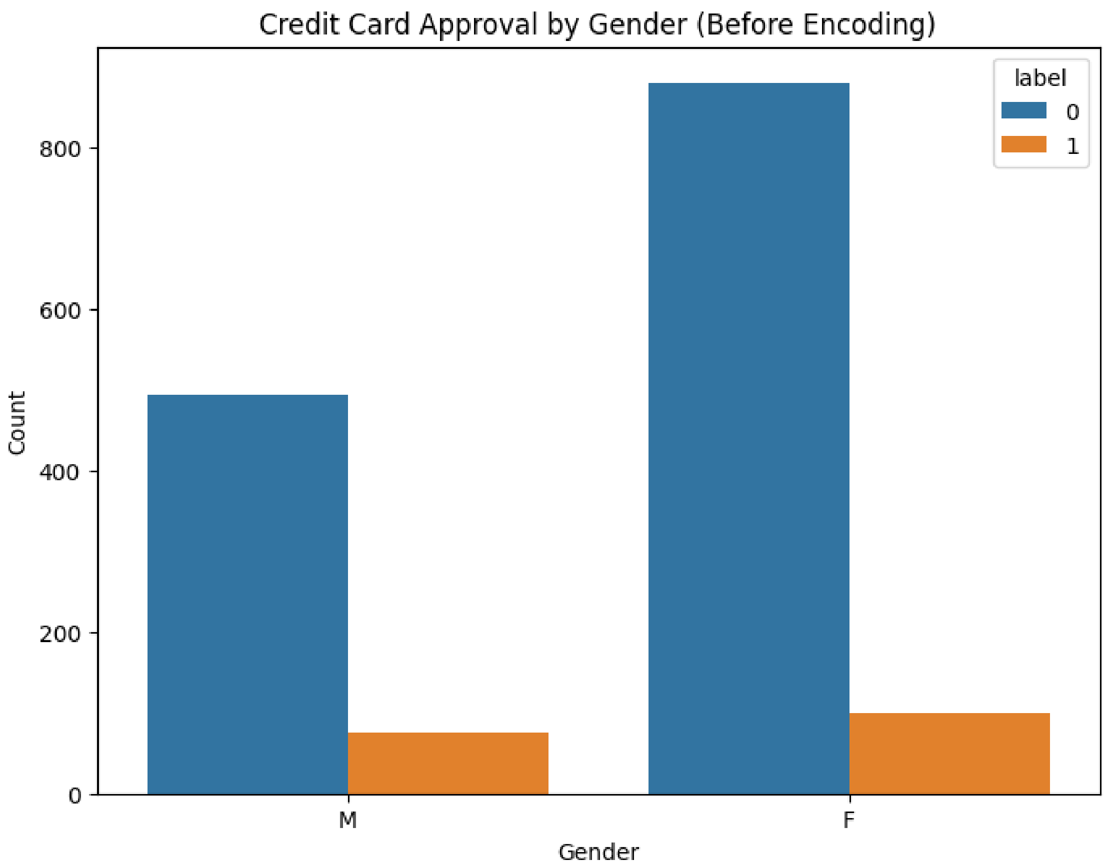
```
Family_Members     0
label              0
Age                0
Employed_years     0
dtype: int64
```

1 Treating Outliers

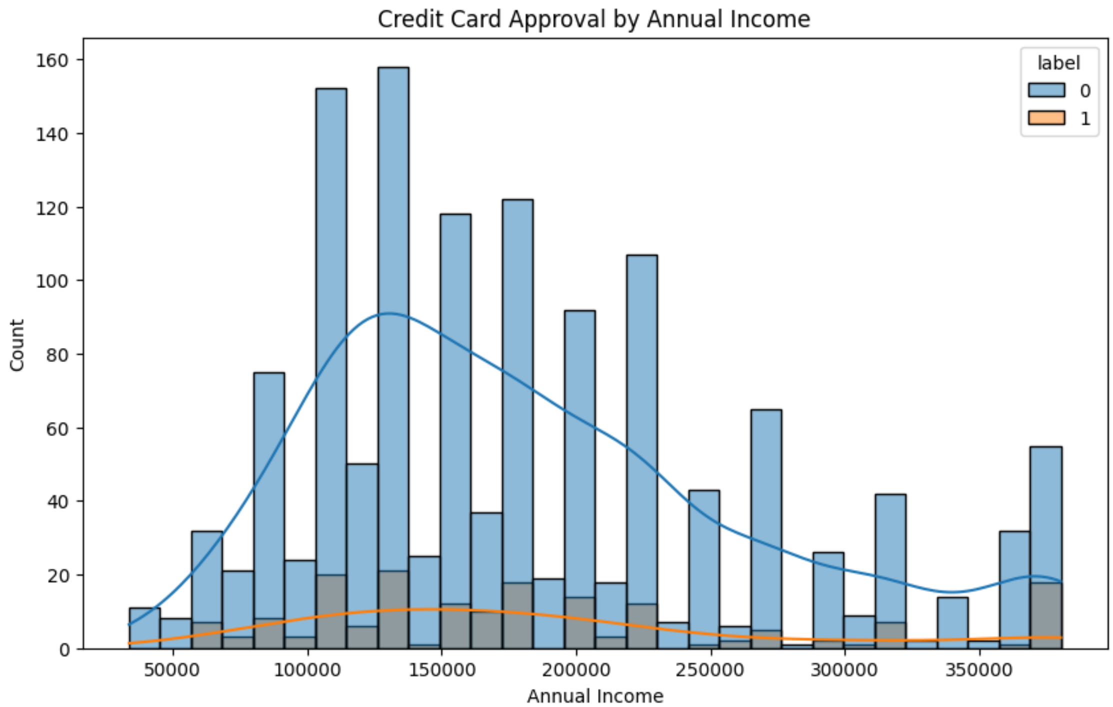
EXTRACTED OUTLIERS USING Z-SCORE

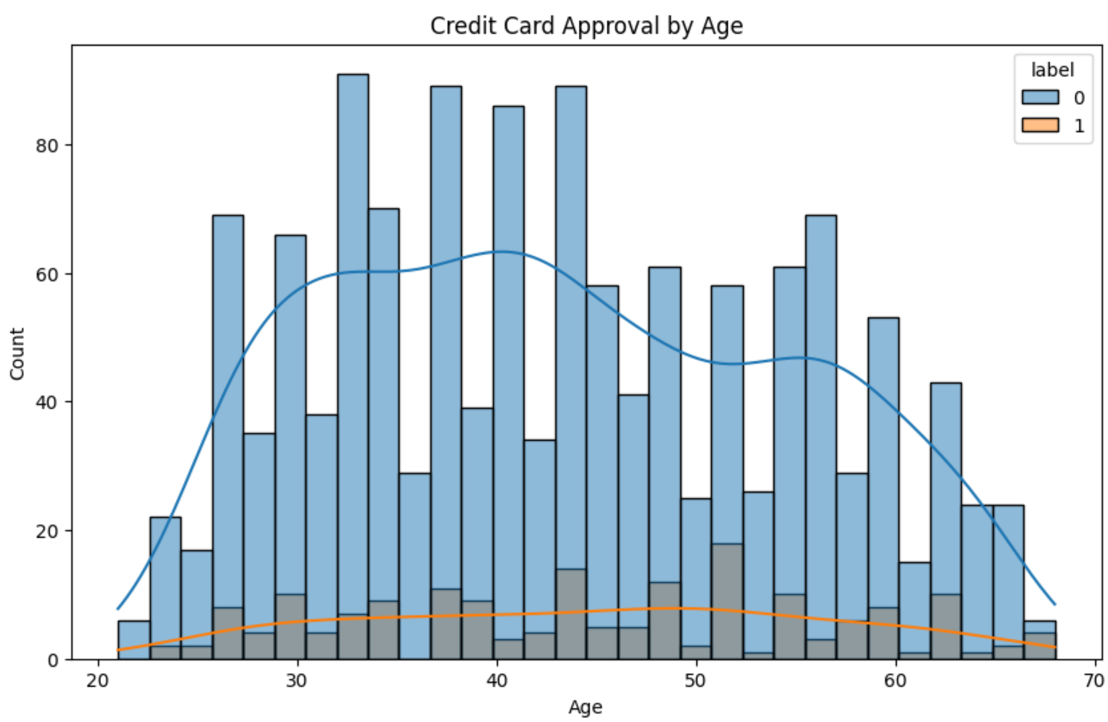
[56]:
```python
import pandas as pd
#IQR method to treat outliers

# Calculate the IQR for 'Annual_income' and 'Employed_years'
Q1 = df[['Annual_income', 'Employed_years']].quantile(0.25)
Q3 = df[['Annual_income', 'Employed_years']].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify and treat outliers in 'Annual_income'
df['Annual_income'] = df['Annual_income'].apply(lambda x:␣
↪upper_bound['Annual_income'] if x > upper_bound['Annual_income'] else␣
↪(lower_bound['Annual_income'] if x < lower_bound['Annual_income'] else x))

# Identify and treat outliers in 'Employed_years'
df['Employed_years'] = df['Employed_years'].apply(lambda x:␣
↪upper_bound['Employed_years'] if x > upper_bound['Employed_years'] else␣
↪(lower_bound['Employed_years'] if x < lower_bound['Employed_years'] else x))

# Now, the dataset 'df' has outliers in 'Annual_income' and 'Employed_years'␣
↪treated
```

[57]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Visualizing the relationship between 'Gender' and 'label' before encoding
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', hue='label', data=df)
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Credit Card Approval by Gender (Before Encoding)')
plt.show()
```



[58]:
```
# Visualizing the relationship between 'Annual_income' and 'label' using␣
↪histograms
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Annual_income', hue='label', bins=30, kde=True)
plt.xlabel('Annual Income')
plt.ylabel('Count')
plt.title('Credit Card Approval by Annual Income')
plt.show()

# Visualizing the relationship between 'Age' and 'label' using histograms
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', hue='label', bins=30, kde=True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Credit Card Approval by Age')
plt.show()
```

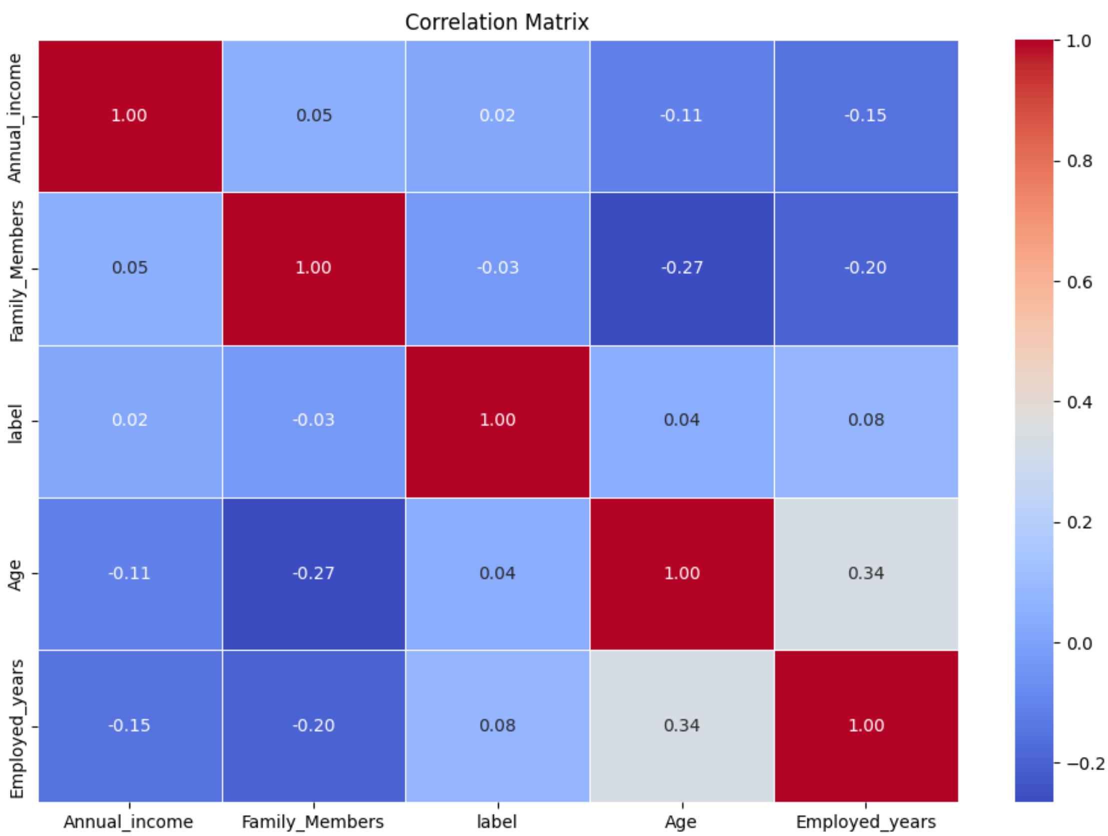




CORRELATION:-Correlation analysis helps you understand the relationships between numerical variables in your dataset

[59]:
```
correlation_matrix = df.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

```
<ipython-input-59-b5da2a171882>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  correlation_matrix = df.corr()
```



FEATURE ENGINEERING

[60]:
```
#checking the skewness of data
df.skew()
```

```
<ipython-input-60-f526aed6189a>:2: FutureWarning: The default value of
numeric_only in DataFrame.skew is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  df.skew()
```

[60]:
```
Annual_income     0.797122
Family_Members    2.232273
label             2.446379
Age               0.173564
Employed_years    0.175969
dtype: float64
```

**This proves the features 'Annual_income','Family_members are right skewed as skewness is >1 and Label is a target variable

[61]:
```python
#Converting the right skewed features to normal distribution
import numpy as np

# Apply a logarithmic transformation to 'Annual_income'
df['Annual_income'] = np.log(df['Annual_income'])

# Apply a square root transformation to 'Family_Members'
df['Family_Members'] = np.sqrt(df['Family_Members'])
```

[62]:
```python
df.skew()
```

```
<ipython-input-62-9e0b1e29546f>:1: FutureWarning: The default value of
numeric_only in DataFrame.skew is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  df.skew()
```

[62]:
```
Annual_income    -0.202358
Family_Members    0.552291
label             2.446379
Age               0.173564
Employed_years    0.175969
dtype: float64
```

**Skewness*

'Annual_income':-This had a positive skewness of approximately 0.797122 (right-skewed).After applying the natural logarithm (ln) transformation, the skewness is now approximately -0.202358. This indicates that the transformation has made the distribution closer to normal and has reduced the right-skewness. **'Family_Members':-**Thishad a positive skewness of approximately 2.232273 (right-skewed).After applying the square root transformation, the skewness is now approximately

0.552291. The transformation has significantly reduced the right-skewness, making the distribution closer to normal

ENCODING:-Encoding categorical variables is an essential step in preparing your dataset for machine learning models. We typically need to convert categorical variables into numerical format since many machine learning algorithms require numerical inputs. We can use techniques like one-hot encoding for this purpose.

[63]:
```
# List of categorical columns to be one-hot encoded
categorical_columns = ['Gender', 'Car_Owner', 'Propert_Owner', 'Type_Income',␣
↪'EDUCATION', 'Marital_status', 'Housing_type', 'Type_Occupation']

# Use pandas' get_dummies function for one-hot encoding
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# 'drop_first=True' is used to prevent multicollinearity by dropping one␣
↪category from each encoded feature set

# Now, 'df_encoded' contains your dataset with one-hot encoding applied to␣
↪categorical columns
```

MODEL BUILDING AND MODEL EVALUATION

TRAIN_TEST_SPLIT:-To split your dataset into training and testing sets, you can use the train_test_split function from the sklearn.model_selection module in Python. This function randomly splits your data into two subsets: one for training your model and the other for testing its performance.

[64]:
```
from sklearn.model_selection import train_test_split

# As the dataset is 'df_encoded' and the output variable as 'label'

# Split your dataset into features (X) and the target variable (y)
X = df_encoded.drop('label', axis=1)
y = df_encoded['label']

# Split the data into training (80%) and testing (20%) sets, and adjusting the␣
↪'test_size' parameter as needed
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# 'random_state' ensures reproducibility of the split, we can choose any␣
↪integer value

# Now, we have X_train, y_train for training the model and X_test, y_test for␣
↪testing its performance
```

[65]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings("ignore")

# Assuming you have already loaded and split your data into X_train, X_test,␣
 ↪y_train, and y_test

# Create instances of different models
logistic_regression = LogisticRegression(random_state=42)
random_forest = RandomForestClassifier(random_state=42)
xgboost = XGBClassifier(random_state=42)
decision_tree = DecisionTreeClassifier(random_state=42)

# Create a list of models to evaluate
models = [logistic_regression, random_forest, xgboost, decision_tree]

# Iterate through the models and evaluate them
for model in models:
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate and print accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model: {type(model).__name__}")
    print(f"Accuracy: {accuracy:.2f}")

    # Print classification report for more detailed evaluation
    print(classification_report(y_test, y_pred))

# Assuming you have already trained your models
y_pred_lr = logistic_regression.predict(X_test)
y_pred_rf = random_forest.predict(X_test)
y_pred_xgb = xgboost.predict(X_test)
y_pred_dt = decision_tree.predict(X_test)


# Calculate and print the accuracy of each model separately (if needed)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
```

```python
accuracy_dt = accuracy_score(y_test, y_pred_dt)

print("Logistic Regression Accuracy:", accuracy_lr)
print("Random Forest Accuracy:", accuracy_rf)
print("XGBoost Accuracy:", accuracy_xgb)
print("Decision Tree Accuracy:", accuracy_dt)
```

```
Model: LogisticRegression
Accuracy: 0.91
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       280
           1       1.00      0.03      0.06        30

    accuracy                           0.91       310
   macro avg       0.95      0.52      0.51       310
weighted avg       0.92      0.91      0.86       310

Model: RandomForestClassifier
Accuracy: 0.93
              precision    recall  f1-score   support

           0       0.93      1.00      0.96       280
           1       1.00      0.30      0.46        30

    accuracy                           0.93       310
   macro avg       0.97      0.65      0.71       310
weighted avg       0.94      0.93      0.92       310

Model: XGBClassifier
Accuracy: 0.93
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       280
           1       0.79      0.37      0.50        30

    accuracy                           0.93       310
   macro avg       0.86      0.68      0.73       310
weighted avg       0.92      0.93      0.92       310

Model: DecisionTreeClassifier
Accuracy: 0.88
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       280
           1       0.41      0.47      0.44        30
```

```
    accuracy                           0.88       310
   macro avg       0.68      0.70      0.69       310
weighted avg       0.89      0.88      0.89       310

Logistic Regression Accuracy: 0.9064516129032258
Random Forest Accuracy: 0.932258064516129
XGBoost Accuracy: 0.9290322580645162
Decision Tree Accuracy: 0.8838709677419355
```

OBSERVATION:- Random Forest and XGBoost achieved the highest accuracy among the models, both at 0.93. Logistic Regression had the lowest accuracy at 0.90. Random Forest and XGBoost also performed better in terms of classifying class 1 (credit risk) compared to the other models. Decision Tree had the lowest accuracy and performance for class 1. Based on these results, you might consider selecting either Random Forest or XGBoost as the best model for this specific problem

HYPERPARAMETER TUNING

[66]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

# Hyperparameter Tuning
# Define the hyperparameter grid to search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid,␣
 ↪cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and the best model
best_params = grid_search.best_params_
best_rf_model = grid_search.best_estimator_

# 2. Cross-Validation (Optional)
# If you want to perform cross-validation on the entire dataset, you can do it␣
 ↪with the best model.
# This step helps provide a more robust estimate of the model's performance.

from sklearn.model_selection import cross_val_score
```

```
cv_scores = cross_val_score(best_rf_model, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())

# 3. Final Model Training
# Train the best model on the entire training dataset
best_rf_model.fit(X_train, y_train)

# 4. Model Evaluation
# Evaluate the model on the test set
y_pred = best_rf_model.predict(X_test)
print("Model: Random Forest (Tuned)")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Cross-Validation Scores: [0.91935484 0.91935484 0.90322581 0.91093117
0.91497976]
Mean CV Accuracy: 0.9135692830090113
Model: Random Forest (Tuned)
Accuracy: 0.9258064516129032
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       280
           1       1.00      0.23      0.38        30

    accuracy                           0.93       310
   macro avg       0.96      0.62      0.67       310
weighted avg       0.93      0.93      0.90       310
```

[69]:
```
df.to_csv('data_to_load.csv', index=False)
```