Feature Engineering & Machine Learning

After performing an Exploratory Data Analysis to the DataFrame ‘df’ is time to enhance and go deeper with the analysis of the data through feature engineering. **Feature Engineering** is crucial in analytical projects because it transforms raw data into meaningful inputs that improve model performance. It helps highlight hidden patterns, reduce noise, and align data with model requirements.

It's recommended to perform feature engineering before machine learning because proper features can drastically increase model accuracy and reduce overfitting, making algorithms more effective with less complexity. Finally, based on the characteristics of the project, two machine learning models were selected for the credit card fraud detection task: **Logistic Regression** and **Random Forest**.

Logistic Regression is ideal for fraud detection due to its simplicity, interpretability, and effectiveness in binary classification. Random Forest adds power by capturing non-linear relationships, handling imbalanced data better, and being robust to outliers, making both models complementary for detecting fraudulent transactions.

Feature Engineering

Based on insights uncovered during the time-based analysis in the EDA, two new features will be engineered to enhance the fraud detection model. The first, **is_night**, captures whether a transaction occurs between 10:00 p.m. and 6:00 a.m. a time window where fraudulent activity was most prevalent. The second, **is_weekend**, identifies transactions made on weekends to evaluate the significance of this period in relation to fraudulent behavior.

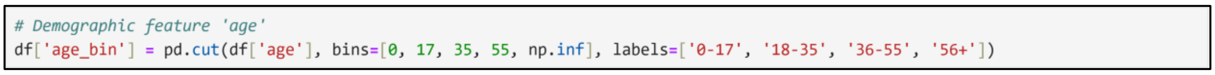
```
# Time-based new features
df['is_night'] = df['trans_hour'].apply(lambda x: 1 if x < 6 or x >= 22 else 0)
df['is_weekend'] = df['trans_day_week'].apply(lambda x: 1 if x >= 5 else 0)
```

During the exploratory data analysis (EDA), the '**amt**' variable, representing transaction amounts, was found to be highly right-skewed, indicating the presence of extreme values. To address this skewness and facilitate a more normalized distribution, a log transformation will be applied.

Additionally, to enhance pattern recognition and support interpretability in modeling, transaction amounts will be discretized into categorical bins. This binning strategy enables the model to capture nonlinear relationships and provides a clearer understanding of amount-based behaviors across defined ranges.

```
# Amount-based features
df['amount_log'] = np.log1p(df['amt'])
df['amount_bin'] = pd.cut(df['amt'], bins=[0, 100, 500, 1000, 5000, 10000, np.inf], labels=['0-100', '100-500', '500-1000',
                   '1000-5000','5000-10000', '10000+'],right=False)
```

Finally, we will apply the binning strategy to the demographic feature **'age'**. The goal of transforming the continuous age variable into discrete categories is to facilitate more effective analysis and modeling. By segmenting age into defined groups, it becomes easier to identify trends or behavioral patterns within specific demographics.



```python
# Demographic feature 'age'
df['age_bin'] = pd.cut(df['age'], bins=[0, 17, 35, 55, np.inf], labels=['0-17', '18-35', '36-55', '56+'])
```

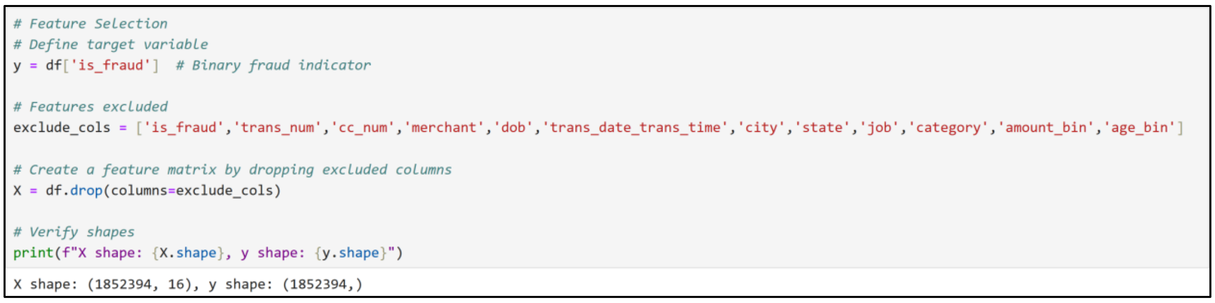
Feature Selection

It's time to proceed with the feature selection for our DataFrame 'df'. Feature selection is a critical step in feature engineering that identifies and **retains the most relevant variables** while eliminating noise, redundancy, and irrelevant data. By filtering out weak or correlated predictors, it enhances model performance, reduces overfitting, and improves generalization. Additionally, feature selection optimizes computational efficiency by lowering dimensionality, speeding up training, and making models more interpretable.

Let's begin by isolating the target variable **is_fraud** as **y**, and removing non-informative or potentially misleading columns from the feature matrix **X**.

Features excluded:

- is_fraud: Target variable.
- trans_num: Transaction ID (unique identifier)
- cc_num: Credit card number (identifier)
- merchant: Merchant name (likely too granular)
- dob: Date of birth (already age in the DataFrame)
- trans_date_trans_time: Datetime (already have derived features)
- city: City (already have location coordinates)
- state: State (could keep but have coordinates)
- job: Job title (likely too granular)
- category: Original category column (have dummies)
- amount_bin: Original amount bin column (have dummies)
- age_bin: Original age bin column (have dummies)



```python
# Feature Selection
# Define target variable
y = df['is_fraud']  # Binary fraud indicator

# Features excluded
exclude_cols = ['is_fraud','trans_num','cc_num','merchant','dob','trans_date_trans_time','city','state','job','category','amount_bin','age_bin']

# Create a feature matrix by dropping excluded columns
X = df.drop(columns=exclude_cols)

# Verify shapes
print(f"X shape: {X.shape}, y shape: {y.shape}")
```

```
X shape: (1852394, 16), y shape: (1852394,)
```

Splitting Data

Now, the DataFrame can be split into training and test sets. This allows the evaluation of the model's performance on unseen data, simulating how it would perform in real-world scenarios. Splitting Data is a key step during the feature engineering because it **prevents data leakage**, ensuring that transformations and model training are done only on the training data, leading to more reliable and unbiased evaluation results.

```
# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Splitting the data before applying transformations like scaling or encoding is essential to maintain class balance across train and test sets, which is especially important in imbalanced problems like fraud detection.

One-Hot encoding

After generating new binned features like **amount_bin** and **age_bin**, and splitting the data into training and test sets, the next step is to apply One-Hot Encoding to the DataFrame. The **category** variable will be included in this process due to its low cardinality, which makes it well-suited for this type of encoding. One-Hot Encoding is important in feature engineering because it transforms categorical variables into a numeric format that most machine learning models can interpret without assuming any ordinal relationship.

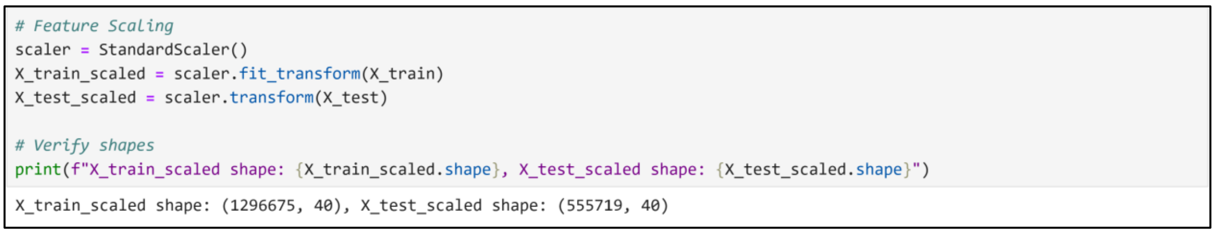
It creates binary columns for each category, improving model accuracy, interpretability, and handling of non-linear relationships. While it helps avoid misleading assumptions from label encoding, it can increase dimensionality, so caution is needed with high-cardinality features.

```
# One-Hot Encoding (on train and test separately)
cols_to_dummy = ['category', 'amount_bin', 'age_bin']

# Recover those columns for dummying from original df
X_train[cols_to_dummy] = df.loc[X_train.index, cols_to_dummy]
X_test[cols_to_dummy] = df.loc[X_test.index, cols_to_dummy]

# Get dummies
X_train = pd.get_dummies(X_train, columns=cols_to_dummy, prefix=cols_to_dummy)
X_test = pd.get_dummies(X_test, columns=cols_to_dummy, prefix=cols_to_dummy)

# Align test set to train set in case some dummies are missing
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)
```

Feature Scaling

A feature scaling will be executed to bring all numerical features to a **common scale**, usually to improve model performance and stability. Its execution is important because many algorithms like logistic regression and PCA are sensitive to the magnitude of feature values.

Without scaling, features with larger ranges can dominate the learning process, leading to biased or inaccurate results. By scaling (e.g., standardizing to mean 0 and variance 1), you ensure that all features contribute equally, allowing the model to learn patterns more effectively and converge faster during training.



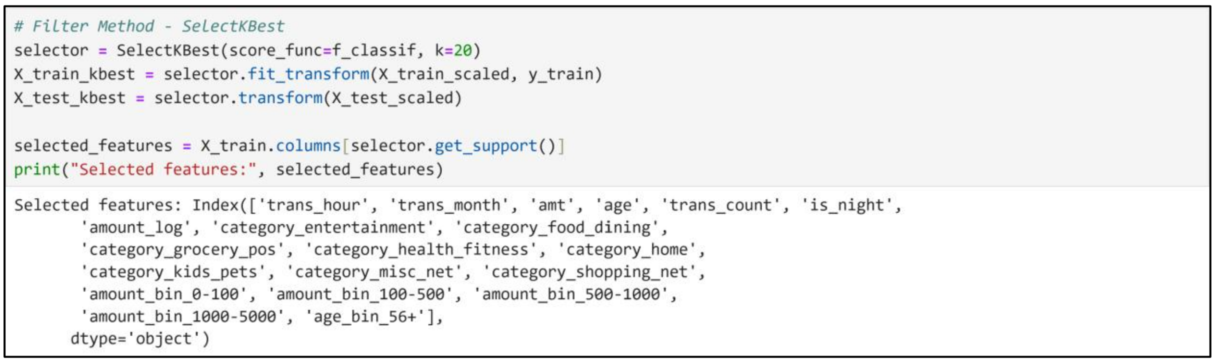
```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Verify shapes
print(f"X_train_scaled shape: {X_train_scaled.shape}, X_test_scaled shape: {X_test_scaled.shape}")
```

```
X_train_scaled shape: (1296675, 40), X_test_scaled shape: (555719, 40)
```

Filter Method – SelectKBest

Next step is to apply a feature selection method like SelectKBest from scikit-learn that **selects the top K features** based on their scores from a specified statistical test. By reducing dimensionality and retaining only the most relevant features, it helps minimize noise and improve model performance. This is particularly beneficial in imbalanced classification problems like fraud detection, where focusing on the most predictive signals can significantly enhance precision and recall.

For the interest of this project, the **20 features** with the highest correlation to the target variable will be selected.



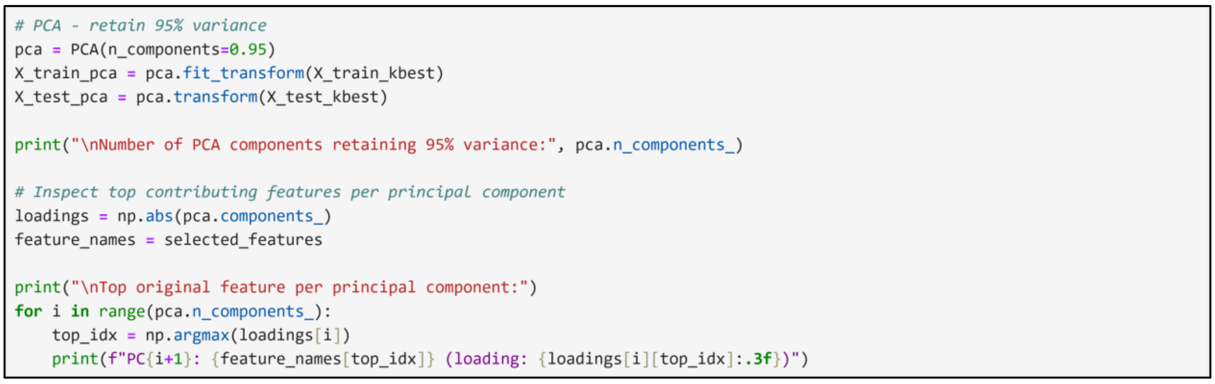
```
# Filter Method - SelectKBest
selector = SelectKBest(score_func=f_classif, k=20)
X_train_kbest = selector.fit_transform(X_train_scaled, y_train)
X_test_kbest = selector.transform(X_test_scaled)

selected_features = X_train.columns[selector.get_support()]
print("Selected features:", selected_features)
```

```
Selected features: Index(['trans_hour', 'trans_month', 'amt', 'age', 'trans_count', 'is_night',
       'amount_log', 'category_entertainment', 'category_food_dining',
       'category_grocery_pos', 'category_health_fitness', 'category_home',
       'category_kids_pets', 'category_misc_net', 'category_shopping_net',
       'amount_bin_0-100', 'amount_bin_100-500', 'amount_bin_500-1000',
       'amount_bin_1000-5000', 'age_bin_56+'],
      dtype='object')
```

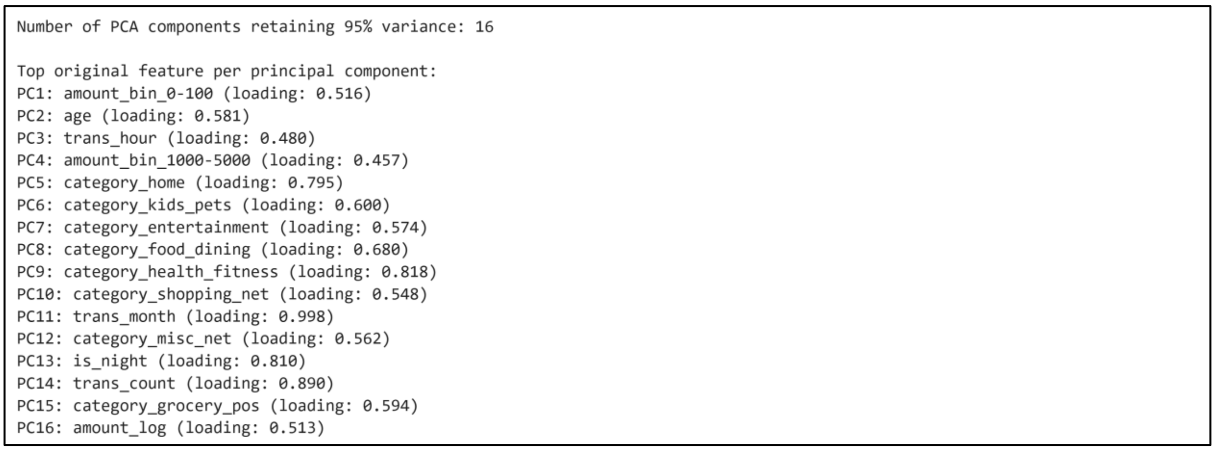
Dimensionality Reduction – PCA

As part of the ongoing data refinement, we now perform a PCA (Principal Component Analysis), a **dimensionality reduction technique** that transforms correlated features into a smaller set of uncorrelated components while retaining most of the variance in the data. It's recommended during feature engineering to reduce complexity, improve model efficiency, and mitigate multicollinearity. PCA complements SelectKBest by capturing hidden structure and variance that univariate methods might miss.

PCA is effective for handling high-dimensional or noisy data, as in our credit card fraud detection project. It enhances anomaly detection and improves the performance of algorithms sensitive to feature correlation, especially when used alongside other feature selection methods.



```python
# PCA - retain 95% variance
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train_kbest)
X_test_pca = pca.transform(X_test_kbest)

print("\nNumber of PCA components retaining 95% variance:", pca.n_components_)

# Inspect top contributing features per principal component
loadings = np.abs(pca.components_)
feature_names = selected_features

print("\nTop original feature per principal component:")
for i in range(pca.n_components_):
    top_idx = np.argmax(loadings[i])
    print(f"PC{i+1}: {feature_names[top_idx]} (loading: {loadings[i][top_idx]:.3f})")
```

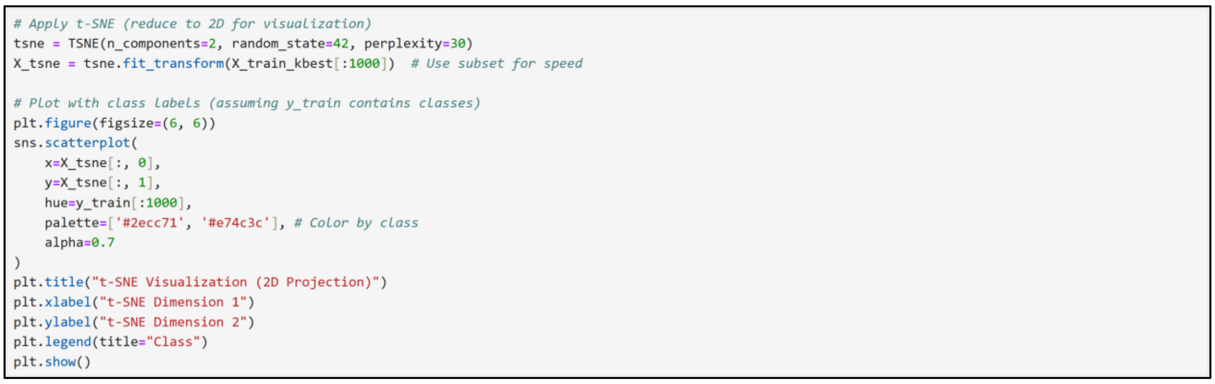


```
Number of PCA components retaining 95% variance: 16

Top original feature per principal component:
PC1: amount_bin_0-100 (loading: 0.516)
PC2: age (loading: 0.581)
PC3: trans_hour (loading: 0.480)
PC4: amount_bin_1000-5000 (loading: 0.457)
PC5: category_home (loading: 0.795)
PC6: category_kids_pets (loading: 0.600)
PC7: category_entertainment (loading: 0.574)
PC8: category_food_dining (loading: 0.680)
PC9: category_health_fitness (loading: 0.818)
PC10: category_shopping_net (loading: 0.548)
PC11: trans_month (loading: 0.998)
PC12: category_misc_net (loading: 0.562)
PC13: is_night (loading: 0.810)
PC14: trans_count (loading: 0.890)
PC15: category_grocery_pos (loading: 0.594)
PC16: amount_log (loading: 0.513)
```

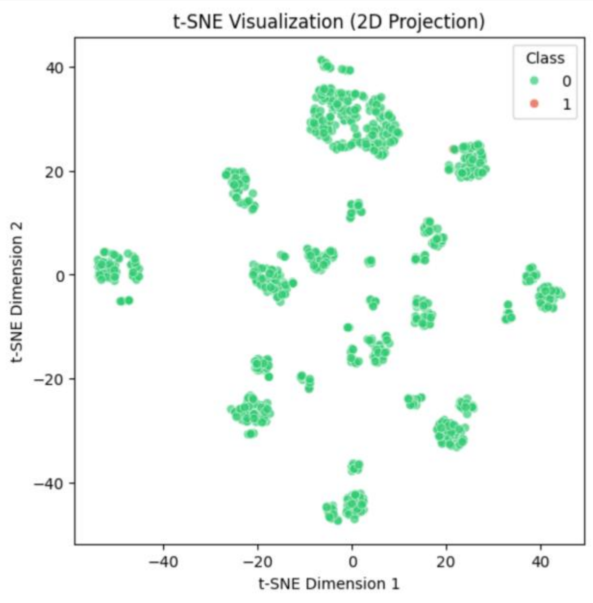
t-SNE for Visualization

Will finish the feature engineering for this project with a t-SNE. T-distributed Stochastic Neighbor Embedding is a nonlinear **dimensionality reduction** technique used primarily for **visualizing high-dimensional data in 2D or 3D**. Its key strength lies in preserving local structure, meaning it keeps similar data points close together in the lower-dimensional space.

When applied after methods like SelectKBest and PCA, where SelectKBest filters out irrelevant features and PCA reduces dimensionality while preserving global variance, t-SNE provides a meaningful 2D visualization of the refined data. This is especially useful in feature engineering to **visually assess patterns**, detect clusters or outliers, and evaluate how well different classes (such as fraudulent vs. non-fraudulent transactions) are separated, ultimately guiding more informed modeling decisions.



```python
# Apply t-SNE (reduce to 2D for visualization)
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
X_tsne = tsne.fit_transform(X_train_kbest[:1000])  # Use subset for speed

# Plot with class labels (assuming y_train contains classes)
plt.figure(figsize=(6, 6))
sns.scatterplot(
    x=X_tsne[:, 0],
    y=X_tsne[:, 1],
    hue=y_train[:1000],
    palette=['#2ecc71', '#e74c3c'], # Color by class
    alpha=0.7
)
plt.title("t-SNE Visualization (2D Projection)")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.legend(title="Class")
plt.show()
```



The t-SNE visualization confirms that the current features capture meaningful structure in the data, as seen in the distinct clusters of legitimate transactions. However, the lack of visible separation for fraudulent cases and the overwhelming presence of legitimate ones highlights a **severe class imbalance** and poor feature differentiation for fraud detection.

This suggests that further feature engineering and class balancing techniques, such as SMOTE or anomaly detection, are necessary to improve model performance. The plot serves as a critical diagnostic tool, revealing that while the preprocessing steps are on the right track, additional work is needed to make fraud cases more distinguishable in the feature space.

Predictive Modeling

This is the final step of the credit card fraud detection project. Now it's time to perform a predictive modeling the process of using statistical and machine learning algorithms to forecast future outcomes based on historical data; a key step in an analytical project because it transforms raw data into actionable insights, enabling data-driven decision-making.

Predictive modeling is particularly crucial in a credit card fraud detection project because it allows the system to automatically **identify potentially fraudulent transactions** in real time. Its impact includes reducing financial losses, improving security, and enhancing customer trust by accurately flagging suspicious activity while minimizing false positives.

SMOTE

During the t-SNE visualization conducted in the feature engineering stage, a pronounced class imbalance between fraudulent and legitimate transactions was observed. This insight highlighted the need for a data balancing technique to enhance model performance. As a result, SMOTE (Synthetic Minority Over-sampling Technique) was selected to address this issue effectively.

SMOTE (Synthetic Minority Over-sampling Technique) is a technique used to **address class imbalance** by generating synthetic examples of the minority class through interpolation. This helps ensure that predictive models like Logistic Regression and Random Forest do not become biased toward the majority class.

Applying SMOTE before model training improves the model's ability to learn from both classes, leading to **better recall and F1-score** for the minority class. It is crucial to apply SMOTE only on the training set, after scaling and feature selection, to avoid data leakage and ensure reliable model evaluation.

```
# Apply SMOTE to Balance the Classes
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_kbest, y_train)
```

Logistic Regression Model

Logistic Regression is a supervised machine learning algorithm used for **binary classification problems**, where the outcome is either 0 or 1, such as “fraud” or “not fraud”. It models the **probability** that a given input belongs to a particular class using the **logistic (sigmoid) function**, making it ideal for scenarios where we want to estimate the likelihood of an event happening.

Credit card fraud detection is a classic **binary classification problem** where transactions are labeled as either **legitimate (0)** or **fraudulent (1)**. Logistic Regression is particularly valuable in this domain because:

- It is **interpretable**: we can understand how each feature (e.g., transaction amount, hour, category) contributes to the fraud risk through model coefficients.
- It is **efficient** on large, high-dimensional datasets, making it suitable for real-time transaction analysis.
- It outputs **probabilities**, which allows businesses to set flexible thresholds based on risk tolerance (e.g., flag only the top 1% riskiest transactions).

```
# Train the Logistic Regression model on balanced data
log_model = LogisticRegression(max_iter=1000, random_state=42)
log_model.fit(X_train_bal, y_train_bal)

# Make predictions on test set
y_pred_log = log_model.predict(X_test_kbest)
y_proba_log = log_model.predict_proba(X_test_kbest)[:, 1]

# Model Evaluation
print("🔍 Logistic Regression Performance:")
print(classification_report(y_test, y_pred_log))
print(f"ROC AUC: {roc_auc_score(y_test, y_proba_log):.4f}")
```

```
🔍 Logistic Regression Performance:
              precision    recall  f1-score   support

           0       1.00      0.90      0.94    552824
           1       0.04      0.88      0.08      2895

    accuracy                           0.90    555719
   macro avg       0.52      0.89      0.51    555719
weighted avg       0.99      0.90      0.94    555719

ROC AUC: 0.9574
```

The Logistic Regression model for credit card fraud detection demonstrates strong discriminatory power, with a high ROC AUC of **0.9574**, indicating it effectively ranks fraudulent and legitimate transactions. The model achieves an excellent **recall of 0.88** for fraud cases, successfully identifying most of them — a crucial aspect in fraud detection tasks.

However, the model suffers from **very low precision (0.04)** for fraud predictions, meaning that only 4% of transactions flagged as fraud are truly fraudulent. This results in a high number of false positives, which could disrupt customer experience and trigger unnecessary interventions. The **F1-score for the fraud class is also low (0.08)**, reflecting an imbalance between precision and recall.

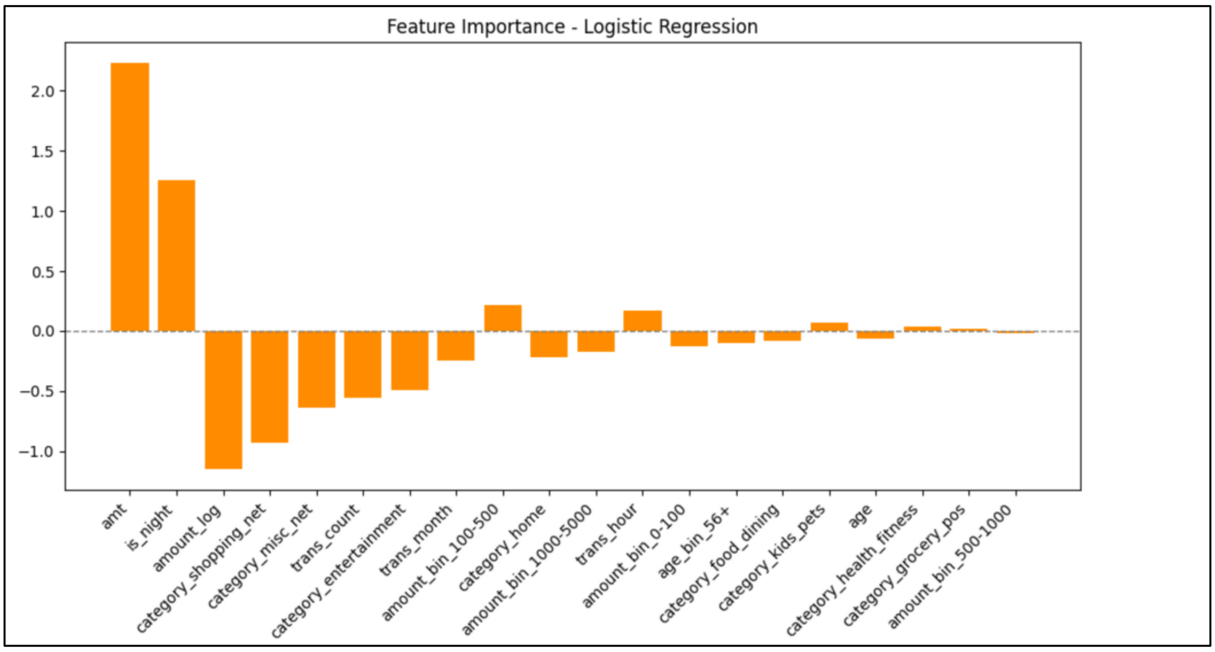
Despite being trained on balanced data, the evaluation on a highly imbalanced test set (~99.5% legitimate) reveals that class imbalance still impacts model performance. To address this, adjusting the classification threshold, calibrating probabilities, or exploring alternative models like **Random Forest** or XGBoost is recommended to improve precision without significantly sacrificing recall.

Feature Importance – Logistic Regression

```
# Feature Importance
importance_log = log_model.coef_[0]
for i, v in enumerate(importance_log):
    print(f'Feature: {selected_features[i]:<25} Importance: {v:.3f}')
```

```
Feature: trans_hour                Importance: 0.171
Feature: trans_month               Importance: -0.248
Feature: amt                       Importance: 2.232
Feature: age                       Importance: -0.068
Feature: trans_count               Importance: -0.553
Feature: is_night                  Importance: 1.255
Feature: amount_log                Importance: -1.152
Feature: category_entertainment    Importance: -0.489
Feature: category_food_dining      Importance: -0.081
Feature: category_grocery_pos      Importance: 0.020
Feature: category_health_fitness   Importance: 0.041
Feature: category_home             Importance: -0.223
Feature: category_kids_pets        Importance: 0.068
Feature: category_misc_net         Importance: -0.634
Feature: category_shopping_net     Importance: -0.931
Feature: amount_bin_0-100          Importance: -0.132
Feature: amount_bin_100-500        Importance: 0.223
Feature: amount_bin_500-1000       Importance: -0.019
Feature: amount_bin_1000-5000      Importance: -0.177
Feature: age_bin_56+               Importance: -0.103
```

```
# Get selected feature names
feature_names = X_train.columns[selector.get_support()]

# Sort coefficients by absolute value
coef = log_model.coef_[0]
idx = np.argsort(np.abs(coef))[::-1]

# Plot
plt.figure(figsize=(10, 6))
plt.bar(np.array(feature_names)[idx], coef[idx], color='darkorange')
plt.xticks(rotation=45, ha='right')
plt.title("Feature Importance - Logistic Regression")
plt.axhline(0, color='gray', lw=1, ls='--')
plt.tight_layout()
plt.show()
```

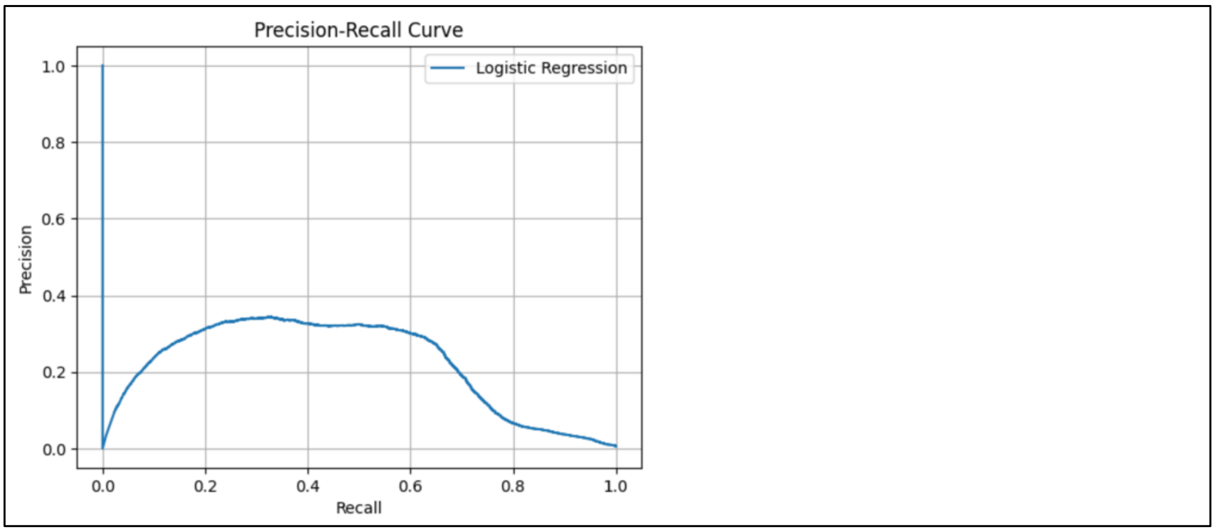


The feature importance analysis from the Logistic Regression model highlights key drivers of fraud detection. The most influential positive predictors are **amt** (transaction amount) and **is_night**, indicating that high-value transactions and those occurring at night significantly increase the likelihood of fraud—aligning with common fraud patterns. Other positively weighted features like **trans_hour** and **amount_bin_100-500** also suggest certain transaction times and amounts are associated with risk.

Conversely, features such as **category_shopping_net**, and **trans_count** have strong negative weights, indicating that frequent transactions and spending in online shopping categories reduce fraud likelihood—likely reflecting typical user behavior. Categories like **entertainment**, **misc_net**, and **trans_month** also contribute negatively, suggesting legitimate usage patterns.

Precision-Recall Curve - Logistic Regression

```
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_proba_log)
plt.plot(recall, precision, label='Logistic Regression')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```



The Precision-Recall Curve shows that the Logistic Regression model struggles to balance precision and recall in detecting credit card fraud. Precision is high only when recall is near zero, indicating the model is confident in very few fraud predictions. However, as recall increases, precision drops sharply, meaning the model captures more fraud cases but at the cost of many false positives.

This confirms earlier findings of high recall (0.88) but very low precision (0.04). Overall, while the model detects most frauds, its reliability in correctly identifying them is limited. Adjusting the decision threshold or exploring alternative models may improve this trade-off.

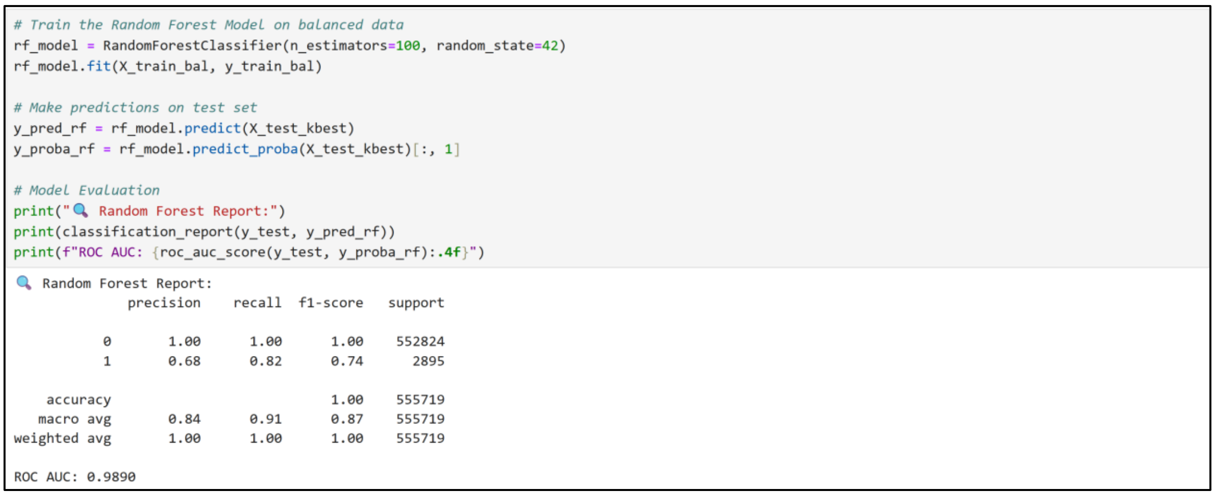
Random Forest Model

A **Random Forest** is an ensemble learning model that constructs multiple decision trees using random subsets of data and features, then aggregates their outputs (majority voting for classification) to make a final prediction. This approach improves accuracy and reduces overfitting compared to individual decision trees.

It is recommended to apply Random Forest when dealing with complex, non-linear relationships, high-dimensional data, or class imbalance—common traits in credit card fraud detection problems. Its robustness to noise, missing values, and outliers makes it suitable for real-world datasets.

In credit card fraud detection, Random Forest is particularly useful because it can capture subtle, complex patterns in transactional behavior that might indicate fraud. Random Forest is particularly valuable in this domain because:

- It handles **imbalanced data** well (common in fraud cases).
- Captures **complex patterns** and interactions that simpler models might miss.
- Offers **high accuracy** and **low variance**, reducing false positives/negatives.
- Provides **feature importance**, helping identify key fraud indicators.



```
# Train the Random Forest Model on balanced data
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_bal, y_train_bal)

# Make predictions on test set
y_pred_rf = rf_model.predict(X_test_kbest)
y_proba_rf = rf_model.predict_proba(X_test_kbest)[:, 1]

# Model Evaluation
print("🔍 Random Forest Report:")
print(classification_report(y_test, y_pred_rf))
print(f"ROC AUC: {roc_auc_score(y_test, y_proba_rf):.4f}")
```

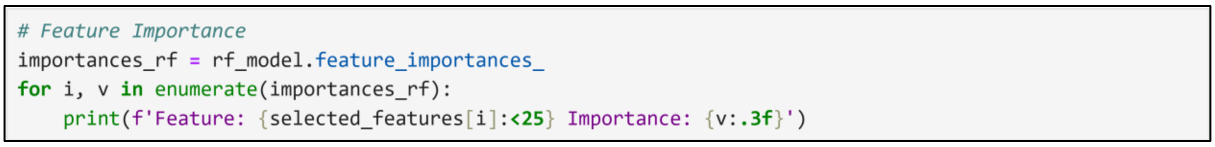
```
🔍 Random Forest Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    552824
           1       0.68      0.82      0.74      2895

    accuracy                           1.00    555719
   macro avg       0.84      0.91      0.87    555719
weighted avg       1.00      1.00      1.00    555719

ROC AUC: 0.9890
```

The evaluation results of the Random Forest model indicate strong performance, especially in handling **class imbalance**. The model perfectly identifies legitimate transactions (class 0) with a **precision**, **recall**, **and F1-score of 1.00**, which is expected given the large volume of non-fraud cases.

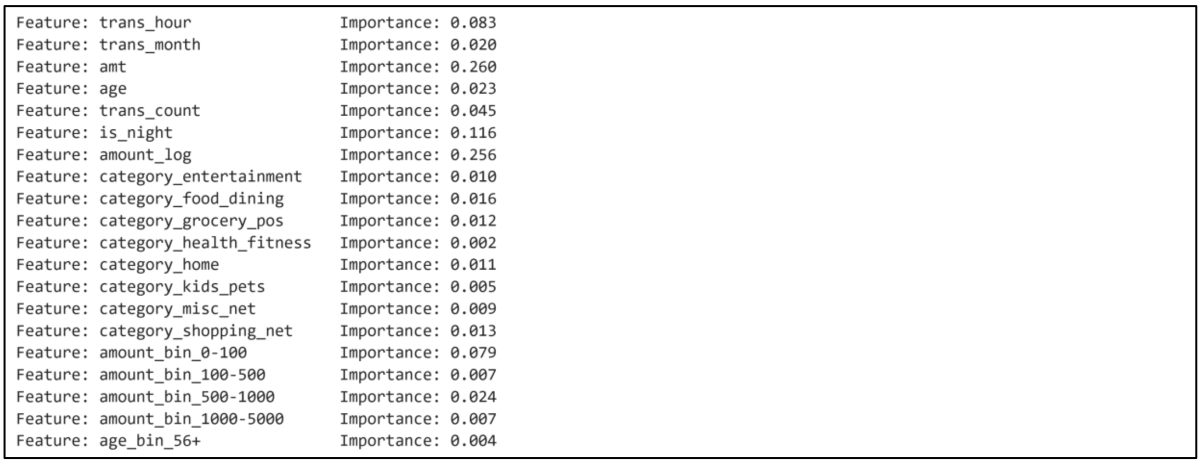
For fraudulent transactions (class 1), the model achieves a recall of 0.82, meaning it successfully detects **82% of actual fraud cases**—an essential metric in this domain. The **precision is 0.68**, suggesting that about **32% of transactions flagged as fraud are false positives**. The **F1-score of 0.74** reflects a good balance between precision and recall.

Overall accuracy is 1.00, but due to class imbalance, this metric can be misleading. A more meaningful indicator is the **ROC AUC score of 0.9890**, which shows excellent model capability to distinguish between fraud and non-fraud. In summary, the model is highly effective at detecting fraud, with a strong recall and AUC, making it a solid choice for this type of problem.

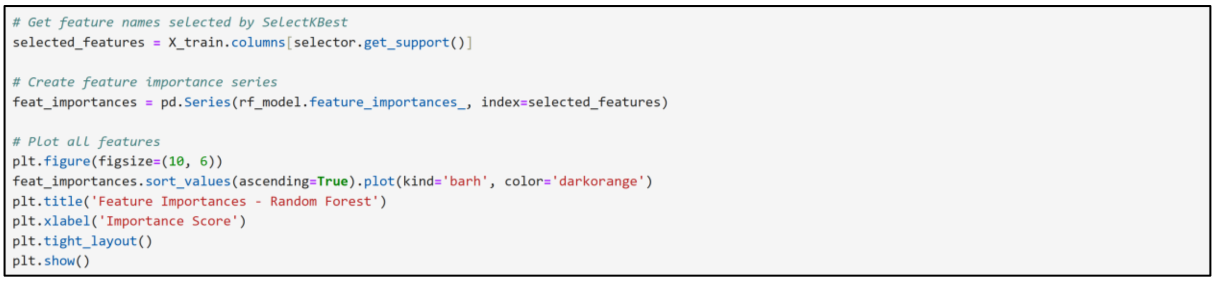
Feature Importance – Random Forest



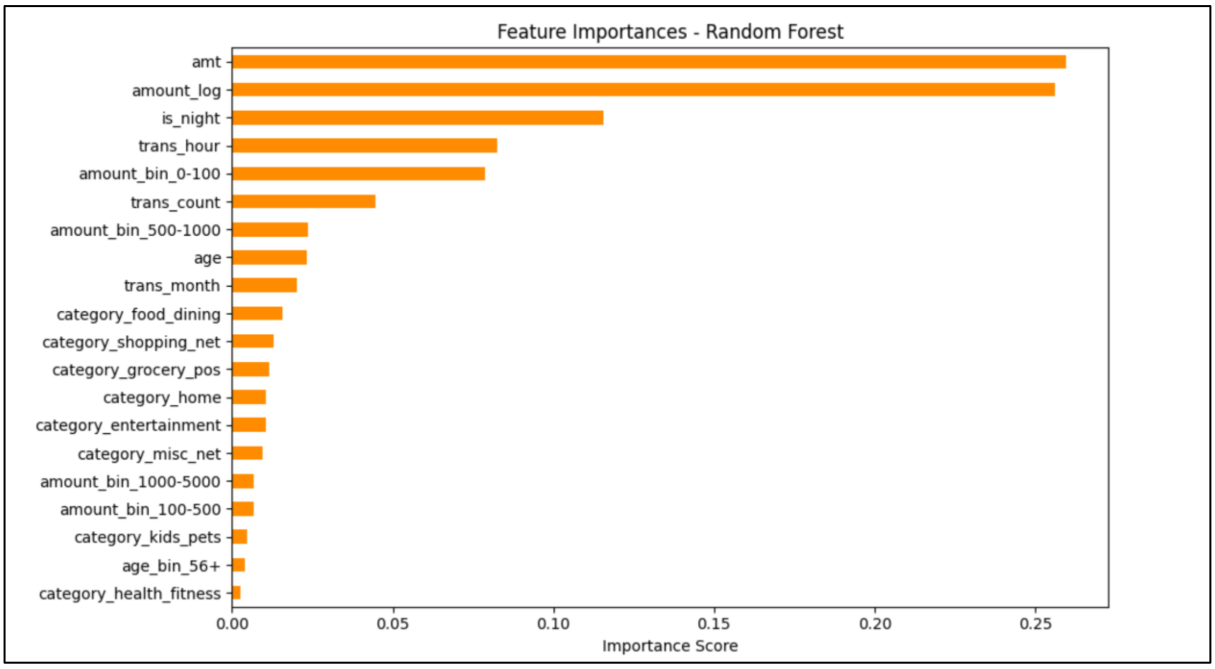
```
# Feature Importance
importances_rf = rf_model.feature_importances_
for i, v in enumerate(importances_rf):
    print(f'Feature: {selected_features[i]:<25} Importance: {v:.3f}')
```



```
Feature: trans_hour                Importance: 0.083
Feature: trans_month               Importance: 0.020
Feature: amt                       Importance: 0.260
Feature: age                       Importance: 0.023
Feature: trans_count               Importance: 0.045
Feature: is_night                  Importance: 0.116
Feature: amount_log                Importance: 0.256
Feature: category_entertainment    Importance: 0.010
Feature: category_food_dining      Importance: 0.016
Feature: category_grocery_pos      Importance: 0.012
Feature: category_health_fitness   Importance: 0.002
Feature: category_home             Importance: 0.011
Feature: category_kids_pets        Importance: 0.005
Feature: category_misc_net         Importance: 0.009
Feature: category_shopping_net     Importance: 0.013
Feature: amount_bin_0-100          Importance: 0.079
Feature: amount_bin_100-500        Importance: 0.007
Feature: amount_bin_500-1000       Importance: 0.024
Feature: amount_bin_1000-5000      Importance: 0.007
Feature: age_bin_56+               Importance: 0.004
```



```python
# Get feature names selected by SelectKBest
selected_features = X_train.columns[selector.get_support()]

# Create feature importance series
feat_importances = pd.Series(rf_model.feature_importances_, index=selected_features)

# Plot all features
plt.figure(figsize=(10, 6))
feat_importances.sort_values(ascending=True).plot(kind='barh', color='darkorange')
plt.title('Feature Importances - Random Forest')
plt.xlabel('Importance Score')
plt.tight_layout()
plt.show()
```



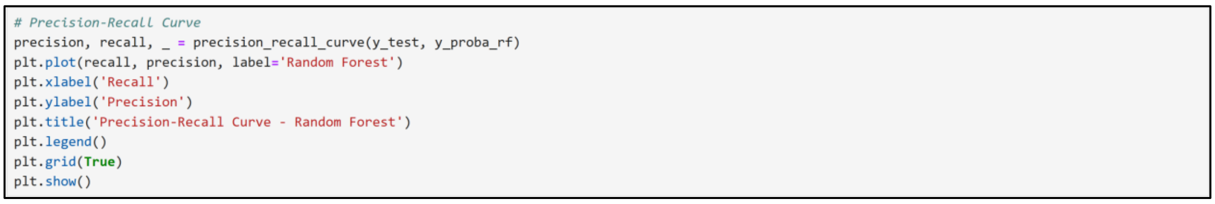
The feature importance results from the Random Forest model show that **transaction amount (amt)** is the most influential features, with importance scores of **0.260**, suggesting that the **transaction amount** (both raw and log-transformed) plays a crucial role in identifying fraud. Fraudulent transactions often exhibit abnormal values, making amount-based features highly discriminative.

The feature **is_night (0.116)** is also highly important, indicating that transactions occurring at night are more likely to be fraudulent, a common behavioral insight. **trans_hour (0.083)** and **amount_bin_0-100 (0.079)** also contribute significantly, reinforcing the relevance of **time-of-day and binned amount distributions** in fraud detection.

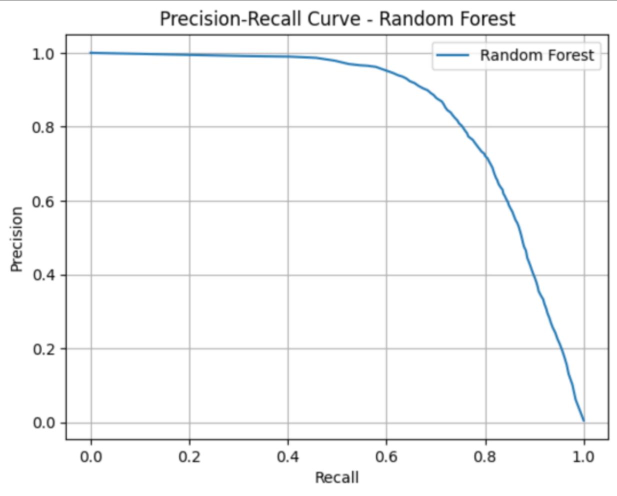
Other features like **trans_count (0.045)** and **trans_month (0.020)** add moderate value, likely capturing behavioral patterns over time. Meanwhile, **category-based features** (like category_entertainment, category_food_dining, etc.) and some **age-related bins** have relatively **low importance (< 0.02)**, indicating that merchant category and user age may offer limited discriminative power in this model.

In summary, **monetary values** and **temporal patterns** (amount, time of transaction) are the most influential factors in predicting fraud, while **demographic** and **categorical** variables have lower relevance in this specific dataset and model.

Precision-Recall Curve – Random Forest



```
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_proba_rf)
plt.plot(recall, precision, label='Random Forest')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - Random Forest')
plt.legend()
plt.grid(True)
plt.show()
```



The Precision-Recall curve of the Random Forest model shows **strong performance** for credit card fraud detection, where class imbalance is a major challenge. At low recall levels, **precision remains close to 1.0**, meaning the model is highly accurate when predicting only the most confident fraud cases. As recall increases, precision gradually declines, especially beyond 0.7, indicating more false positives as the model becomes more inclusive.

The overall shape of the curve is **smooth and high**, suggesting a **favorable balance between precision and recall** across various thresholds. This indicates that the model is effective at detecting a large portion of fraudulent transactions while maintaining an acceptable false positive rate. In short, the curve confirms the Random Forest model's suitability for real-world fraud detection.

Conclusions

Random Forest is the clear choise for the credit card fraud detection project. While Logistic Regression achieves a high recall of **0.88** for fraud cases, its **precision is extremely low (0.04)**, meaning that **most fraud predictions are false positives**. In contrast, the Random Forest model achieves a **much better balance**, with a **precision of 0.68** and **recall of 0.82** for frauds, along with a **significantly higher F1-score (0.74 vs.**

0.08). Additionally, its **ROC AUC score of 0.9890** outperforms Logistic Regression's 0.9574, indicating superior discrimination between fraud and non-fraud.

Therefore, based on the results from the Random Forest model, the **most influential factors for fraud detection** are **transaction amount (amt)**, **transaction time (trans_hour, is_night)**, and **amount binning (amount_bin_0-100)**. These findings suggest that **fraudulent activity tends to occur at unusual hours and often involves specific transaction amounts**, either very low (to test stolen cards) or very high (to maximize unauthorized gains).

Recommendations to a Bank:

1. **Real-time monitoring of transaction amounts**: Flag transactions that fall outside typical ranges, especially those with unusually high amounts or small micro-transactions, as they carry higher fraud risk.
2. **Implement time-based fraud rules**: Transactions during night hours (is_night) should be evaluated with stricter thresholds or require multi-factor authentication, given their higher risk score.
3. **Use amount and time features in fraud scoring engines**: Integrate features like amount_log, trans_hour, and amount_bin_0-100 directly into production fraud scoring systems to improve early detection.
4. **Less emphasis on categorical data**: Since merchant categories and customer age bins have low importance, current efforts focused on these may yield limited returns and can be deprioritized.

Data-Driven Strategy for Fraud Prevention:

- **Dynamic Risk Scoring System**: Build a real-time scoring model that assigns risk based on the most important features. This system should adapt thresholds depending on the time of day and transaction amount.
- **Behavioral Profiling**: Establish individual baselines for users' transaction behavior (amount, frequency, time), and flag deviations. For instance, a large transaction at night from a user who typically shops in the afternoon is a red flag.
- **Threshold-Based Alerts + ML Triggers**: Combine rule-based systems (e.g., flagging transactions over $1,000 at night) with machine learning models to reduce false positives while still catching sophisticated fraud attempts.
- **Customer Notifications & Feedback Loops**: Promptly notify users of suspicious activity and use their responses to retrain fraud detection models, improving future accuracy.

In summary, a financial institution should focus its fraud detection strategy on **transaction amount and time patterns**, deploy **adaptive models** that prioritize these features, and continuously **refine them with real-time behavioral data**.