Shrey Patel

Project Report:

Predictive Credit Card Approval System

Introduction:

This project aims to create a machine learning-based application for automatic credit card approval. Using hyperparameter optimization and Logistic Regression, the model predicts whether a loan will be fully paid back. This report explains the methodology, analysis, findings, and implementation in a structured manner.

How It Helps in Predicting Credit Approval: -

1. **Data-Driven Decision-Making:**
    - The insights from the dataset (like patterns between fico and loan repayment) enable the system to predict the likelihood of a loan default based on an applicant's credit profile.
2. **Accurate Predictions:**
    - The machine learning models, especially the fine-tuned XGBoost model, predict whether a credit card application should be approved or rejected based on the likelihood of repayment.
3. **Reduced Risks:**
    - By focusing on predicting defaults (not_fully_paid), the system helps reduce financial risks for credit issuers.
4. **Explainability:**
    - Feature importance and correlation analyses make it easier to explain predictions to stakeholders or regulators, ensuring transparency.

Data Exploration and Preprocessing: -

Data Overview:

- **Dataset Source and Size:** The dataset was sourced from [specify source, e.g., UCI Machine Learning Repository]. It contains [number of rows] rows and [number of columns] columns, providing diverse features related to credit risk.
- **Goal:** To predict `not_fully_paid` (1 if the loan is not fully paid, 0 otherwise).
- **Key Features:**
  1. `credit_policy`: Criteria met by the customer.
  2. `purpose`: Loan purpose.
  3. `int_rate`, `fico`, `dti`: Risk-related metrics.
  4. `days_with_cr_line`, `revol_bal`: Credit usage history.
  5. `pub_rec`, `delinq_2yrs`: Historical payment behavior.

Steps Taken:

1. **Data Loading:** The dataset was loaded, and its structure was inspected.
2. **Null Value Check:** Missing values were identified for handling.
3. **Column Name Cleanup:** Replaced "." with "_" in column names for consistency.

This helps in Understanding the dataset's structure and ensures proper feature alignment with the target variable, facilitating meaningful preprocessing and analysis.

Exploratory Data Analysis (EDA):

Correlation Matrix:

- A correlation matrix was generated to understand relationships between numeric features.

Key Insights:

- Positive Correlation: Higher `int_rate` correlated with `not_fully_paid`.
- Negative Correlation: Higher `fico` scores correlated with loans fully paid.

How It Helps:

- Identifying correlated features informs feature engineering and model interpretation. For example, a strong negative correlation was observed between `fico` scores and

`not_fully_paid`, suggesting that higher creditworthiness reduces loan default risk. Visualizing these correlations through scatterplots and heatmaps further clarified how numeric features like `int_rate` and `dti` relate to the target variable, providing actionable insights for feature selection.

Visualizations:

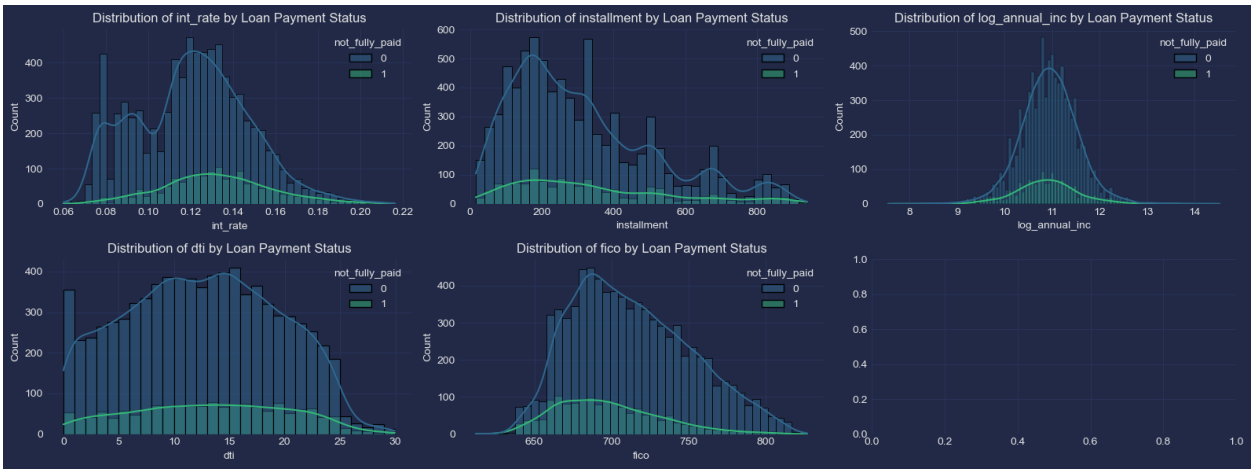
- Histograms: Segmented numeric variables (`int_rate`, `fico`, etc.) by `not_fully_paid`.

Insights:

- Riskier loans (`int_rate` high) were more likely unpaid.

How It Helps:

EDA visualizations highlight patterns that guide feature selection and preprocessing.



Shrey Patel

Data Preprocessing: -

Categorical Variable Encoding:

- One-hot encoding was applied to `purpose` instead of label encoding to handle categorical data. One-hot encoding was chosen because it prevents ordinal relationships from being falsely inferred between categories, which could happen with label encoding. Since the `purpose` feature represents non-ordinal categories (e.g., 'debt_consolidation', 'small_business'), one-hot encoding ensures that each category is treated independently, maintaining the integrity of the data and improving model performance in scenarios where categorical relationships do not follow a natural order.
- One-hot encoding was applied to `purpose` to handle categorical data.
- Ensures compatibility with machine learning models.

```
# Encode Categorical varaibles

df = pd.get_dummies(df,['purpose'], drop_first=True)
```

Python

```
df.head()
```

Python

	credit_policy	int_rate	installment	log_annual_inc	dti	fico	days_with_cr_line	revol_bal	revol_util	inq_last_6mths	delinq_2yrs	pub_rec	not_fully_paid	purpose_credit_card	purpose_debt_consolidation	purpose_educational	purpose_home_improv
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0	False	True	False	
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0	True	False	False	
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0	False	True	False	
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0	False	True	False	
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0	True	False	False	

Feature Scaling:

```
# Feature Scaling

numeric_features = ['int_rate', 'installment', 'log_annual_inc', 'dti','fico', 'days_with_cr_line', 'revol_bal', 'revol_util',
                    'inq_last_6mths', 'delinq_2yrs', 'pub_rec']
```

```
scaler  = StandardScaler()

df[numeric_features] = scaler.fit_transform(df[numeric_features])
```

+ Code + Markdown

```
df[numeric_features].round(2)
```

	int_rate	installment	log_annual_inc	dti	fico	days_with_cr_line	revol_bal	revol_util	inq_last_6mths	delinq_2yrs	pub_rec
0	-0.14	2.46	0.68	1.00	0.69	0.43	0.35	0.18	-0.72	-0.30	-0.24
1	-0.58	-0.44	0.24	0.24	-0.10	-0.72	0.50	1.03	-0.72	-0.30	-0.24
2	0.49	0.23	-0.91	-0.14	-0.76	0.06	-0.40	-0.73	-0.26	-0.30	-0.24
3	-0.81	-0.76	0.68	-0.65	0.03	-0.75	0.50	0.91	-0.26	-0.30	-0.24
4	0.74	-1.04	0.60	0.34	-1.15	-0.20	-0.36	-0.25	-0.72	1.53	-0.24
...	...	...	...	...	...	...	...	...	...	...	...
9573	0.87	0.12	2.03	-0.32	-1.02	2.37	5.88	1.22	0.19	-0.30	-0.24
9574	0.10	-0.30	0.34	-1.80	0.29	-0.07	-0.50	-1.58	1.56	-0.30	-0.24
9575	-0.58	-1.07	-0.55	0.07	-0.63	-0.44	-0.20	1.24	2.92	-0.30	-0.24

- StandardScaler normalized numeric variables to improve model performance.
- Variables like `int_rate` and `fico` were scaled

How It Helps:

- Scaling ensures features contribute equally to model training and prevents dominance by high-value columns.

Train-Test Split:

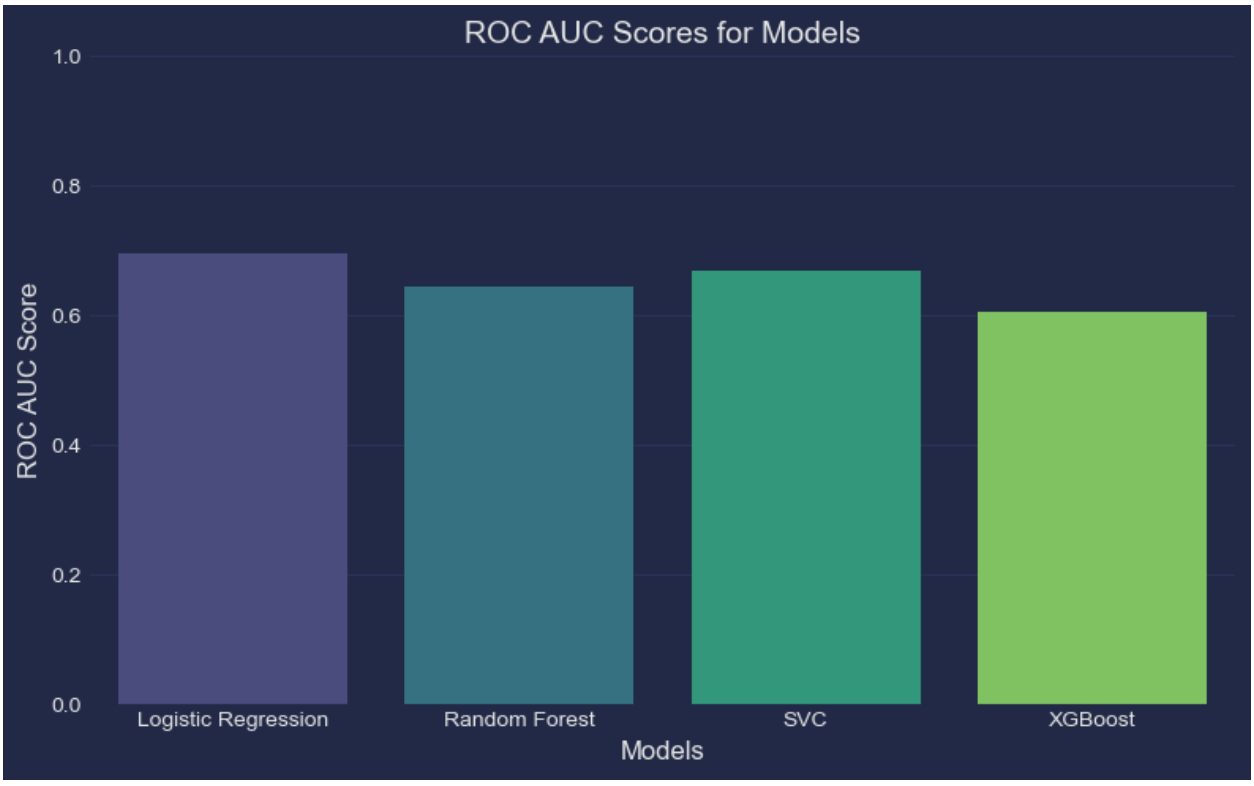
- The dataset was split (70% training, 30% testing).
- Ensures fair evaluation of the model.

Model Building and Evaluation: -

Initial Models:

Four models were trained:

1. Logistic Regression.
2. Random Forest.
3. SVC (Support Vector Classifier).
4. XGBoost.



Class Imbalance Handling:

- Applied `class_weight='balanced'` for Logistic Regression, Random Forest, and SVC.
- Used `scale_pos_weight` in XGBoost to prioritize predicting `not_fully_paid` loans.

Evaluation Metrics:

- **Classification Report:** Precision, recall, and F1-scores.
- **Confusion Matrix:** Visualized true positives/negatives and false positives/negatives.
- **ROC-AUC Score:** Assessed model performance on separating classes.

Findings:

- Random Forest outperformed others with the highest ROC-AUC score.
- Random Forest performed well in feature importance analysis.

How It Helps:

- Comparing models provides insights into their strengths and weaknesses, ensuring the selection of the best-fit model for the task.

Hyperparameter Tuning: -

Grid Search Implementation:

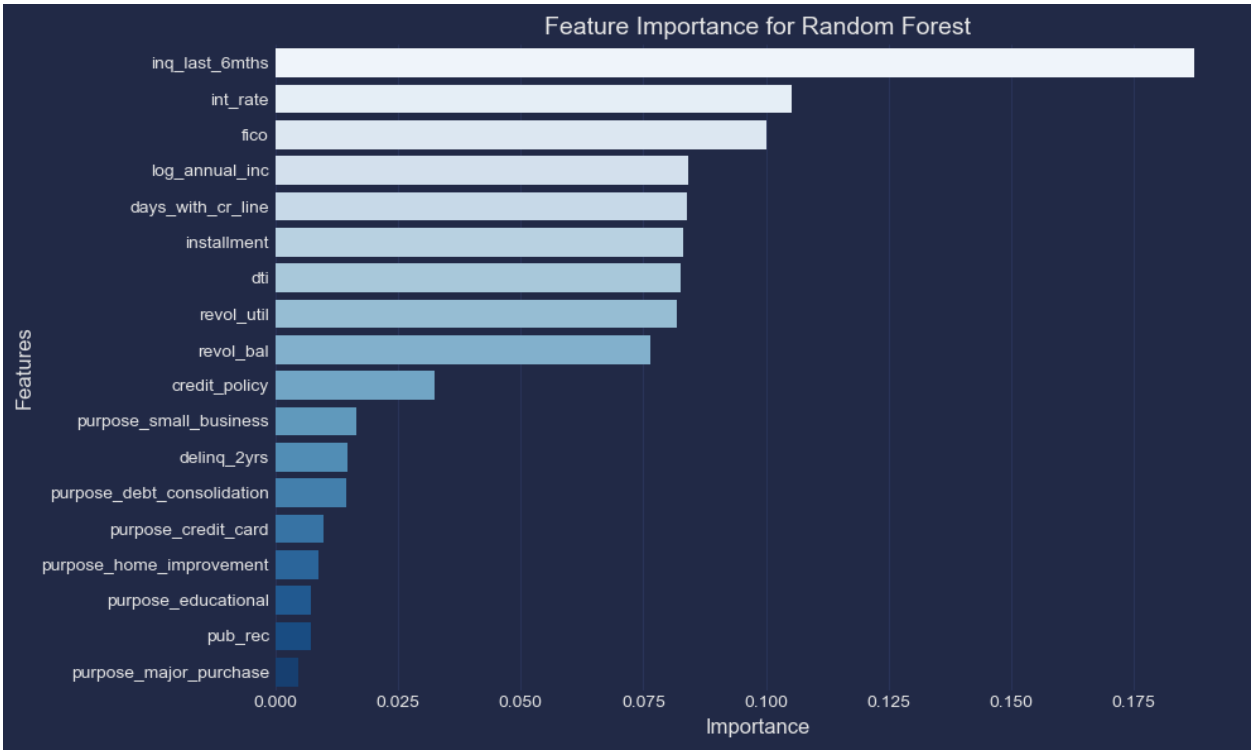
- Performed for all models to optimize parameters.

Parameters Tuned:

- Logistic Regression: `C`, `solver`.
- Random Forest: `n_estimators`, `max_depth`, etc.
- SVC: `C`, `kernel`, `gamma`.
- XGBoost: `learning_rate`, `n_estimators`, `max_depth`, etc.

Best Model:

- Random Forest with optimized parameters achieved the best ROC-AUC score.



How It Helps:

- Tuning improves model performance by finding the optimal configuration and enhancing prediction accuracy.

Final Evaluation and Feature Importance: -

Confusion Matrix:

Provided insights into the model's ability to minimize false negatives (critical for this problem).

Feature Importance:

Identified top predictors of `not_fully_paid`, e.g., `fico`, `int_rate`, `dti`.

How It Helps:

Understanding the importance of features aids in model interpretability and stakeholder communication.

Addressing Class Imbalance with SMOTE: -

Used Synthetic Minority Oversampling Technique (SMOTE) to balance classes in training data.

How It Helps:

Balancing ensures the model learns to identify minority class patterns, improving predictions for `not_fully_paid`.

Deployment: -

Model Saving:

- Exported the best model (Random Forest) using `joblib` for deployment.
- This enables reusability of the trained model in real-world applications, integrating it into automated systems.

Conclusion: -

- This project demonstrates the application of machine learning techniques to predict credit card approvals effectively. The Random Forest model was the best performer after hyperparameter tuning, addressing class imbalance, and prioritizing predictions for `not_fully_paid`. The project highlights skills in EDA, preprocessing, model building, evaluation, and deployment, making it a strong portfolio addition.

Future Enhancements: -

- Incorporate additional features (e.g., employment history).
- Experiment with advanced techniques like stacking or deep learning for further improvements of the model.