Certificate in Credit Risk Analytics in Python

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the foundational step in any credit risk analytics project. It involves summarizing the main characteristics of a dataset, often with visual methods, before any formal modeling begins. For a credit analyst using Python, EDA helps to uncover data quality issues, understand the distribution of key risk variables, and generate hypotheses about the drivers of default. The following glossary presents the most important terms and concepts that a practitioner must master to conduct effective EDA in the context of credit risk.

Population refers to the complete set of entities about which we want to draw conclusions. In credit risk, the population might be all loan applicants for a particular product line across a financial institution. Because obtaining data for every member of the population is rarely feasible, analysts work with a sample, a subset that is assumed to be representative.

Sample is a collection of observations drawn from the population. The quality of the sample determines the reliability of any insights derived from EDA. In Python, a sample is commonly created using the sample method of a pandas DataFrame, e.G., df.Sample(frac=0.1, Random_state=42), which extracts a 10% random subset while preserving reproducibility.

Variable is a generic term for any measured attribute of an observation. In credit risk, variables include applicant age, income, loan amount, credit score, and repayment status. A variable that the analyst wishes to predict, such as default (yes/no), is called the target.

Target (or response variable) is the outcome that a predictive model aims to estimate. In credit risk analytics the target is frequently a binary indicator of default, but it can also be a continuous loss severity measure. During EDA the target is examined both in isolation (e.G., Default rate) and in relation to other variables (e.G., Default vs. Credit score).

Feature denotes any explanatory variable used to predict the target. In the Python ecosystem, features are often stored as columns of a pandas DataFrame. The set of all features is sometimes called the feature space. Understanding each feature's distribution and relationship to the target is a core EDA activity.

Distribution describes how values of a variable are spread across possible outcomes. Common descriptive statistics for a distribution include the mean, median, mode, variance, and percentiles. Visual tools such as histograms, density plots, and boxplots are used to assess distribution shape. For example, a histogram of loan amounts can reveal whether the data are heavily skewed toward small loans or if there are a few very large exposures.

Skewness quantifies the asymmetry of a distribution. Positive skew (right-skew) indicates a long tail on the high-value side, which is common for variables like income or loan amount. Negative skew (left-skew) is less common but may appear in repayment-related metrics. In Python, df['income'].Skew() returns a numeric

measure; values far from zero suggest that a transformation (e.G., Log) may be needed before modeling.

Kurtosis measures the “tailedness” of a distribution. High kurtosis signals heavy tails or outliers, while low kurtosis indicates a flatter distribution. Analysts use kurtosis to diagnose the presence of extreme values that could unduly influence model coefficients. In pandas, df['loan_amount'].Kurt() provides a quick estimate.

Outlier is an observation that lies far outside the typical range of the data. Outliers can arise from data entry errors, fraud, or genuine extreme cases. Detecting outliers is crucial in credit risk because a single erroneous default flag could distort loss estimates. Common detection methods include the interquartile range (IQR) rule, Z-score thresholds, and visual inspection via boxplots. In Python, an IQR-based filter might be written as:

```
Q1 = df['credit_score'].Quantile(0.25)
Q3 = df['credit_score'].Quantile(0.75)
IQR = Q3 - Q1
Outliers = df[(df['credit_score']  Q3 + 1.5*IQR)]
```

Missing Data (or missing values) occur when a variable has no recorded value for some observations. In credit datasets, missingness can be systematic (e.G., Self-employed borrowers may not provide salary details) or random. Identifying patterns of missingness is part of EDA; the pandas function df.Isnull().Sum() quickly reveals column-wise counts. Ignoring missing data can bias risk estimates, so analysts must decide on an appropriate imputation strategy.

Imputation replaces missing values with plausible substitutes. Simple techniques include mean or median imputation for numeric variables and mode imputation for categorical variables. More sophisticated approaches involve predictive models such as k-nearest neighbors or multivariate imputation by chained equations (MICE). In Python, the SimpleImputer class from scikit-learn offers a convenient interface:

```
from sklearn.Impute import SimpleImputer
Imp = SimpleImputer(strategy='median')
Df['annual_income'] = imp.Fit_transform(df[['annual_income']])
```

Correlation measures the linear relationship between two numeric variables. The Pearson correlation coefficient ranges from –1 (perfect negative linear relationship) to +1 (perfect positive linear relationship). A correlation matrix visualized as a heatmap helps analysts spot highly correlated features, which may indicate redundancy. For credit risk, a high correlation between “debt-to-income ratio” and “loan-to-value ratio” could suggest multicollinearity concerns for later regression models.

Covariance is a related concept that quantifies the direction of the linear relationship without normalizing by the variables’ standard deviations. While covariance is less interpretable than correlation, it forms the basis of many dimensionality-reduction techniques such as principal component analysis (PCA).

Multicollinearity arises when two or more explanatory variables are strongly linearly related. In credit risk modeling, multicollinearity can inflate variance of coefficient estimates, making them unstable and difficult

to interpret. Detecting multicollinearity during EDA often involves checking variance inflation factors (VIF). In Python, VIF can be computed using statsmodels:

```
from statsmodels.Stats.Outliers_influence import variance_inflation_factor
X = df[['credit_score', 'debt_to_income', 'loan_amount']]
Vif = pd.DataFrame()
Vif['VIF'] = [variance_inflation_factor(X.Values, i) for i in range(X.Shape[1])]
Vif['feature'] = X.Columns
```

Dimensionality Reduction refers to techniques that reduce the number of variables while preserving essential information. In credit risk, dimensionality reduction can simplify model inputs and improve computational efficiency. The most common method is principal component analysis (PCA), which transforms correlated variables into a smaller set of uncorrelated components that capture the majority of variance. A typical Python workflow uses scikit-learn's PCA class:

```
from sklearn.Decomposition import PCA
Pca = PCA(n_components=0.90)  # retain 90% variance
Components = pca.Fit_transform(df[['credit_score','debt_to_income','loan_amount']])
```

Histogram is a bar-type plot that displays the frequency of observations falling within predefined bins. Histograms are the first visual tool used to assess the shape of a numeric variable's distribution. In credit risk, a histogram of "days past due" may reveal a concentration at zero (on-time payments) with a long tail of late payments, informing the choice of a transformation or binning strategy.

Boxplot (or box-and-whisker plot) summarizes the five-number summary: Minimum, first quartile, median, third quartile, and maximum, while also highlighting outliers. Boxplots are especially useful for comparing the distribution of a variable across categories, such as default rates by loan purpose. In seaborn, a boxplot can be generated with:

```
import seaborn as sns
Sns.Boxplot(x='loan_purpose', y='loan_amount', data=df)
```

Violin Plot combines a boxplot with a kernel density estimate, offering a richer view of the distribution shape. Violin plots are valuable when the analyst wants to see multimodality in a variable like "credit score" across different borrower segments.

Scatter Plot visualizes the relationship between two numeric variables by plotting each observation as a point. In credit risk, a scatter plot of "debt-to-income ratio" versus "annual income" can reveal clusters of high-risk borrowers. Adding a hue dimension (color) to represent default status creates an informative three-dimensional view.

Pair Plot (also known as a scatterplot matrix) displays scatter plots for every pair of variables in a dataset, along with histograms on the diagonal. This comprehensive view helps to spot patterns, correlations, or non-linear relationships among many features simultaneously. In seaborn, `sns.Pairplot(df[['credit_score','loan_amount','debt_to_income','default']])` generates such a matrix.

Heatmap is a two-dimensional visualization where cell colors encode numeric values, commonly used to display a correlation matrix. In credit risk, a heatmap of feature correlations quickly identifies groups of related variables that may need to be aggregated or removed. Seaborn's sns.Heatmap(corr, annot=True, cmap='coolwarm') is a typical implementation.

Time Series data consist of observations recorded at regular time intervals. Credit risk analysts often work with time-series data such as monthly default counts or quarterly loan volumes. EDA for time series includes plotting the series, checking for trends, seasonality, and volatility. The pandas Series object, indexed by dates, enables easy resampling and rolling calculations.

Rolling Statistics compute summary measures over a moving window of observations. Rolling averages smooth out short-term fluctuations, revealing underlying trends. For example, a 12-month rolling average of default rates can highlight periods of heightened risk. In pandas, the syntax is:

df['default_rate'].Rolling(window=12).Mean()

Lag refers to the value of a variable at a previous time step. Lagged variables are frequently introduced as features in credit risk models to capture temporal dependencies. A lag-1 default indicator might be created as:

df['default_lag1'] = df['default'].Shift(1)

Autocorrelation measures the correlation of a time series with its own lagged values. The autocorrelation function (ACF) helps diagnose whether past defaults influence current defaults. In Python, the statsmodels acf function computes these coefficients, and the plot_acf function visualizes them.

Stationarity is a property of a time series whose statistical characteristics (mean, variance) remain constant over time. Non-stationary series may require differencing or transformation before modeling. The Augmented Dickey-Fuller (ADF) test, accessed via statsmodels.Tsa.Stattools.Adfuller, provides a statistical check for stationarity.

Transformation changes the scale or distribution of a variable to meet modeling assumptions. Common transformations include logarithmic, square-root, and Box-Cox. For credit risk, the log transformation is often applied to skewed variables such as "loan amount" to reduce right-skewness. In pandas, a log transform is simply:

df['log_loan_amount'] = np.Log(df['loan_amount'] + 1)

Scaling adjusts the range of numeric variables, typically to improve algorithmic performance. Two primary scaling methods are normalization (rescaling to a 0-1 range) and standardization (centering to zero mean and unit variance). In scikit-learn, the StandardScaler and MinMaxScaler classes implement these techniques.

Normalization (or min-max scaling) maps a variable to the interval [0, 1] using the formula (x – min) / (max – min). This is useful when features have different units, such as "age" (years) versus "loan amount" (dollars). The MinMaxScaler in scikit-learn automates this process.

Standardization subtracts the mean and divides by the standard deviation, producing a Z-score. Standardized variables have a mean of zero and a standard deviation of one, which is advantageous for algorithms that assume Gaussian-distributed inputs (e.G., Logistic regression). The StandardScaler computes these statistics and applies them to the data.

Z-score is the standardized value of a datum, indicating how many standard deviations it lies from the mean. Outliers can be identified by Z-scores exceeding a threshold (commonly ±3). In pandas, (df['credit_score'] - df['credit_score'].Mean()) / df['credit_score'].Std() yields Z-scores.

Robust Scaling uses median and interquartile range instead of mean and standard deviation, making it less sensitive to outliers. The RobustScaler from scikit-learn implements this method and is recommended when the dataset contains extreme values that would otherwise dominate the scaling.

Data Cleaning encompasses all activities that improve data quality before analysis. It includes handling missing values, correcting typographical errors, standardizing categorical codes, and removing duplicate records. In credit risk, data cleaning is often the most time-consuming part of an EDA because loan datasets may contain inconsistent identifiers, outdated addresses, or mismatched dates.

Data Wrangling (or data munging) extends data cleaning by reshaping data structures, merging multiple sources, and creating derived variables. For example, a credit risk analyst may need to join a borrower-level dataset with a macro-economic table containing unemployment rates. In pandas, the merge function accomplishes this:

df_merged = pd.Merge(df_loans, df_macro, left_on='date', right_on='month')

Feature Engineering is the process of creating new variables that capture underlying patterns in the data. In credit risk, common engineered features include “age of credit history,” “average payment delay,” and “number of open credit lines.” Feature engineering is closely tied to domain knowledge; the analyst must understand how each derived variable relates to creditworthiness.

Encoding converts categorical variables into numeric form so that they can be used by machine-learning algorithms. Two principal techniques are one-hot encoding and label encoding. One-hot encoding creates a binary column for each category, while label encoding assigns an integer to each category. For ordinal variables (e.G., Credit rating), label encoding preserves the natural order.

One-Hot Encoding is implemented in pandas via pd.Get_dummies(df['loan_purpose'], prefix='purpose'). This produces a separate column for each loan purpose (e.G., “Purpose_home,” “purpose_auto”). The resulting matrix is sparse, which is acceptable for most modern algorithms but can increase memory usage for high-cardinality variables.

Label Encoding uses scikit-learn’s LabelEncoder:

```
from sklearn.Preprocessing import LabelEncoder
Le = LabelEncoder()
Df['rating_encoded'] = le.Fit_transform(df['credit_rating'])
```

Ordinal Encoding is a variant of label encoding that respects the order of categories (e.G., “Low,” “medium,” “high”). When using ordinal encoding, it is important to ensure that the numeric values reflect the true ranking, otherwise the model may infer incorrect distances between categories.

Binning (or discretization) groups continuous variables into intervals. This can simplify analysis and reveal non-linear relationships. For example, “annual income” might be binned into “low,” “medium,” and “high” tiers. In pandas, pd.Cut(df['annual_income'], bins=[0,30000,60000,1e6], labels=['Low','Medium','High']) accomplishes this.

Discretization is similar to binning but often uses algorithmic methods (e.G., Equal-frequency or entropy-based binning) to determine optimal cut points. The KBinsDiscretizer in scikit-learn offers several strategies:

from sklearn.Preprocessing import KBinsDiscretizer
Kbd = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
Df['income_bin'] = kbd.Fit_transform(df[['annual_income']])

Target Encoding replaces categorical levels with the mean of the target variable for that level. In credit risk, target encoding can be used for high-cardinality variables such as “employer ID,” where each employer’s average default rate is informative. Care must be taken to avoid leakage; cross-validation or smoothing techniques are employed to mitigate overfitting.

Cross-Validation is a resampling procedure used to evaluate model performance and to guard against overfitting. In EDA, cross-validation helps assess whether a derived feature (e.G., A binned variable) genuinely improves predictive power across different data splits. The KFold class in scikit-learn implements stratified or non-stratified splits.

Train-Test Split partitions the dataset into a training set for model fitting and a test set for final evaluation. For credit risk, a typical split might allocate 70 % of the data to training and 30 % to testing, ensuring that the default rate is preserved in both sets (stratified sampling). The pandas train_test_split function from scikit-learn makes this straightforward:

from sklearn.Model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[features], df['default'], test_size=0.3, Stratify=df['default'], random_state=42)

Stratified Sampling ensures that each class (e.G., Default vs. Non-default) appears in roughly the same proportion in both training and test sets. This is essential when the target is imbalanced, as is common in credit risk where defaults may constitute only a few percent of the observations.

Class Imbalance describes a situation where one class dominates the dataset. In credit risk, the default class is typically the minority. Imbalance can cause classifiers to be biased toward the majority class, leading to inflated accuracy but poor detection of defaults. EDA must quantify the extent of imbalance using the class distribution, and subsequent steps may involve resampling techniques.

Oversampling creates synthetic examples of the minority class to balance the dataset. The most popular method is SMOTE (Synthetic Minority Over-sampling Technique), which interpolates between existing minority examples. In Python, the imblearn library provides a SMOTE implementation:

```
from imblearn.Over_sampling import SMOTE
Sm = SMOTE(random_state=42)
X_res, y_res = sm.Fit_resample(X_train, y_train)
```

Under-Sampling reduces the majority class by randomly discarding observations, thereby balancing the class distribution. While simple, under-sampling can discard valuable information, especially when the original dataset is not large. A hybrid approach that combines mild under-sampling with SMOTE is often recommended for credit risk.

Model Evaluation follows EDA and model fitting, but many evaluation metrics are examined during EDA to gauge the predictive potential of variables. For binary classification in credit risk, key metrics include the ROC curve, AUC, confusion matrix, precision, recall, and F1-score. Understanding these metrics early helps the analyst choose appropriate thresholds and assess trade-offs between false positives (unnecessarily rejecting good borrowers) and false negatives (approving risky borrowers).

ROC Curve (Receiver Operating Characteristic) plots the true-positive rate against the false-positive rate at various threshold settings. The shape of the ROC curve provides a visual summary of a model's discriminative ability. In Python, the roc_curve function from scikit-learn produces the data needed for plotting:

```
from sklearn.Metrics import roc_curve
Fpr, tpr, thresholds = roc_curve(y_test, model.Predict_proba(X_test)[:,1])
```

AUC (Area Under the ROC Curve) condenses the ROC curve into a single scalar value ranging from 0.5 (No discrimination) to 1.0 (Perfect discrimination). An AUC above 0.75 Is generally considered acceptable for credit scoring models, though the precise threshold depends on business objectives.

Confusion Matrix tabulates true positives, false positives, true negatives, and false negatives. For credit risk, the matrix helps quantify how many defaults are correctly identified versus how many non-defaults are mistakenly flagged. The confusion_matrix function in scikit-learn returns a 2-by-2 array that can be visualized with a heatmap.

Precision (positive predictive value) measures the proportion of predicted defaults that are actual defaults. High precision indicates that when the model predicts a default, it is likely correct, which is valuable for regulatory reporting.

Recall (sensitivity) quantifies the proportion of actual defaults that the model correctly captures. High recall is crucial for risk mitigation because missed defaults can lead to unexpected losses.

F1-Score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. In credit risk, the F1-score is often used when the cost of false negatives (missed defaults) is

comparable to the cost of false positives (rejected good borrowers).

Precision-Recall Curve plots precision versus recall for different thresholds, offering a more informative view than the ROC curve when dealing with heavily imbalanced data. The area under the precision-recall curve (average precision) can be computed with average_precision_score.

Feature Importance ranks variables according to their contribution to the predictive model. While not strictly an EDA concept, examining feature importance early in the analysis can guide further exploration. Tree-based models such as Random Forests provide built-in importance scores that can be visualized as bar charts.

Partial Dependence Plot (PDP) shows the marginal effect of a single feature on the predicted outcome, holding all other features constant. PDPs help interpret complex models and validate whether the relationships discovered during EDA (e.G., A non-linear effect of debt-to-income ratio) are captured.

SHAP Values (SHapley Additive exPlanations) provide a unified measure of each feature's contribution to an individual prediction. SHAP visualizations can be overlaid on EDA plots to illustrate how a borrower's characteristics drive their predicted default probability.

Data Drift occurs when the statistical properties of input data change over time. In credit risk, data drift can arise from shifts in macro-economic conditions, regulatory changes, or evolving borrower behavior. Detecting drift during EDA involves comparing distributions across time windows (e.G., Pre- and post-crisis periods) using statistical tests such as the Kolmogorov-Smirnov test.

Kolmogorov-Smirnov Test assesses whether two samples come from the same distribution. In Python, scipy.Stats.Ks_2samp implements the test. Applying the KS test to the "credit score" distribution before and after a policy change can reveal whether the applicant pool has become riskier.

Chi-Square Test evaluates the independence of two categorical variables. For credit risk, a chi-square test can determine whether default status is independent of loan purpose. The pandas crosstab function combined with scipy.Stats.Chi2_contingency provides a quick implementation.

ANOVA (Analysis of Variance) compares the means of a numeric variable across multiple groups. For instance, ANOVA can test whether average loan amounts differ significantly across credit rating categories. In Python, statsmodels.Formula.Api.Ols followed by statsmodels.Stats.Anova.Anova_lm performs the analysis.

Quantile-Quantile Plot (Q-Q plot) compares the quantiles of a variable to the quantiles of a theoretical distribution (often normal). Deviations from the diagonal line indicate non-normality, guiding the analyst to apply appropriate transformations. The statsmodels.Graphics.Gofplots.Qqplot function creates Q-Q plots.

Kernel Density Estimate (KDE) provides a smooth estimate of a variable's probability density function. KDE plots complement histograms by revealing subtle features such as multimodality. In seaborn, sns.Kdeplot(df['credit_score']) produces a KDE curve.

Outlier Detection Methods extend beyond the simple IQR rule. Advanced techniques include Local Outlier

Factor (LOF), Isolation Forest, and One-Class SVM. These algorithms assign an outlier score to each observation, allowing analysts to flag unusual borrowers for further review. In scikit-learn, the IsolationForest class is used as follows:

```
from sklearn.Ensemble import IsolationForest
Iso = IsolationForest(contamination=0.01, Random_state=42)
Df['outlier_score'] = iso.Fit_predict(df[features])
```

Data Profiling automates the generation of a comprehensive report that includes statistics, histograms, correlations, and missing-value summaries for every column. The pandas-profiling package creates an HTML report with a single command, providing a rapid overview that can be shared with stakeholders.

Interactive Visualization tools such as Plotly and Bokeh enable analysts to explore data dynamically, zooming into regions of interest and filtering by categorical variables. For credit risk, an interactive scatter plot of “debt-to-income ratio” versus “credit score,” colored by default status, helps uncover hidden patterns that static plots might miss.

Pipeline refers to a sequence of data-processing steps that are executed in order, often culminating in model fitting. In EDA, a pipeline can encapsulate cleaning, imputation, scaling, and encoding, ensuring that the same transformations are applied consistently to training and test data. Scikit-learn’s Pipeline class makes this straightforward:

```
from sklearn.Pipeline import Pipeline
Pipeline = Pipeline([
   ('Imputer', SimpleImputer(strategy='median')),
   ('Scaler', StandardScaler()),
   ('Model', LogisticRegression())
])
```

Feature Selection reduces the number of input variables by retaining only those that contribute meaningfully to the predictive task. Techniques include filter methods (e.G., Selecting variables with high mutual information), wrapper methods (e.G., Recursive feature elimination), and embedded methods (e.G., L1-regularized models). Feature selection is often guided by insights discovered during EDA.

Mutual Information measures the dependence between two variables, capturing both linear and non-linear relationships. In credit risk, mutual information can reveal that “number of credit inquiries” provides information about default risk that is not captured by “credit score.” The mutual_info_classif function in scikit-learn computes this metric.

Regularization adds a penalty term to a model’s loss function to discourage overly complex solutions. L1 regularization (Lasso) can perform feature selection by shrinking some coefficients to zero, while L2 regularization (Ridge) reduces coefficient variance. Understanding regularization during EDA helps decide whether a simpler model might suffice.

Bootstrap resampling creates many repeated samples from the original dataset to estimate the variability of

a statistic. In credit risk, bootstrapping can be used to compute confidence intervals for the default rate or for model performance metrics. The resample function from scikit-learn's utils module implements bootstrap sampling.

Confidence Interval provides a range of plausible values for a population parameter based on a sample statistic. For example, a 95 % confidence interval for the mean loan amount informs the analyst about the precision of the estimate. In Python, the statsmodels.Stats.Weightstats.DescrStatsW class can compute confidence intervals for weighted data.

Weighting assigns different importance to observations, often to correct for sampling bias. In credit risk, weighting may be used when the sample over-represents certain borrower segments. The DescrStatsW class accepts a weight vector, allowing analysts to compute weighted means, variances, and confidence intervals.

Segmentation divides the borrower population into homogeneous groups based on shared characteristics. Segmentation is a key EDA activity because risk profiles can differ dramatically across segments (e.G., Small business loans vs. Personal loans). Techniques include k-means clustering, hierarchical clustering, and rule-based segmentation using business criteria.

K-Means Clustering partitions observations into k clusters by minimizing within-cluster variance. In credit risk, clustering can reveal natural borrower groups that share similar risk attributes. The KMeans class in scikit-learn requires the analyst to choose k, often guided by the elbow method or silhouette scores.

Silhouette Score measures how well an observation fits within its assigned cluster compared to other clusters. Higher silhouette scores indicate well-separated clusters. The silhouette_score function from scikit-learn provides a quantitative way to evaluate clustering results.

Hierarchical Clustering builds a dendrogram representing nested groupings of observations. This method is useful when the analyst wants to explore multiple levels of granularity before committing to a final segmentation. Scipy's linkage and dendrogram functions generate hierarchical cluster visualizations.

Principal Component Analysis (PCA) not only reduces dimensionality but also provides insight into the direction of greatest variance. The component loadings reveal which original variables contribute most to each principal component, helping the analyst interpret underlying risk factors.

Explained Variance Ratio indicates the proportion of total variance captured by each principal component. Summing the ratios of the first few components informs the analyst how many components are needed to retain a desired amount of information (e.G., 90 %). In Python, pca.Explained_variance_ratio_.Cumsum() yields the cumulative curve.

Eigenvalues and Eigenvectors are mathematical constructs underlying PCA. Eigenvalues correspond to the amount of variance explained by each principal component, while eigenvectors define the direction of the component in the original feature space. Understanding these concepts, even at a high level, helps the analyst interpret PCA results.

Linear Discriminant Analysis (LDA) is a dimensionality-reduction technique that simultaneously maximizes

class separability. In credit risk, LDA can be applied when the target is binary, producing a single discriminant axis that best separates defaulters from non-defaulters. The LinearDiscriminantAnalysis class in scikit-learn implements this method.

Time-Window Analysis evaluates data within specific temporal windows (e.G., Quarterly). This approach is useful for detecting seasonality or for comparing the behavior of borrowers before and after a major economic event. Analysts often create rolling windows and compute summary statistics within each window.

Lagged Features capture the historical value of a variable at previous time steps. In credit risk, lagged defaults, lagged macro-economic indicators, or lagged payment histories can improve model performance. Care must be taken to avoid data leakage: The lagged variable should not incorporate information from the future relative to the prediction point.

Rolling Correlation computes the correlation between two time-series variables over a moving window, revealing how their relationship evolves. For example, a rolling correlation between “unemployment rate” and “default rate” may increase during recessionary periods, indicating heightened sensitivity.

Seasonality Decomposition separates a time series into trend, seasonal, and residual components using methods such as STL (Seasonal-Trend decomposition using Loess). Decomposing default rates can highlight recurring patterns (e.G., Higher defaults in certain months) that may influence credit policy.

Statistical Tests for Normality include the Shapiro-Wilk test, Anderson-Darling test, and D’Agostino’s K2 test. These tests assess whether a variable follows a normal distribution, which is important for certain modeling assumptions (e.G., Logistic regression residuals). In Python, scipy.Stats.Shapiro and scipy.Stats.Anderson provide implementations.

Power Analysis estimates the sample size required to detect a specified effect size with a given statistical power. In credit risk, power analysis can guide the design of pilot studies or the collection of additional data to ensure that observed differences (e.G., Between segments) are not due to chance.

Data Dictionary is a documented list of all variables, their definitions, data types, permissible values, and source systems. While not a statistical concept, a well-maintained data dictionary is essential for accurate EDA, as it prevents misinterpretation of coded fields (e.G., “0 = No, 1 = Yes”).

Data Lineage tracks the origin and transformation history of each data element. Understanding lineage helps the analyst trace back any anomalies discovered during EDA to their source, facilitating corrective actions.

Business Rules are domain-specific constraints that must be respected during data cleaning. Examples include “loan amount cannot exceed 80% of collateral value” or “age must be between 18 and 75.” Incorporating business rules into EDA scripts reduces the risk of propagating invalid data into downstream models.

Regulatory Compliance imposes additional requirements on data handling, especially in credit risk where regulations such as Basel III, the Fair Credit Reporting Act (FCRA), and GDPR dictate data privacy, model

transparency, and reporting standards.