Absolutely—there are several **powerful, model-driven EDA techniques** that, like decision trees, help uncover **human-interpretable insights** rather than just modeling predictions. These are sometimes called **"model-assisted EDA"** or **"explanatory modeling techniques"**.

Below is a breakdown of **9 EDA techniques**—including decision trees—that can supercharge your insight discovery.

---

1. Decision Trees (CART) – for Rule Discovery

What it does: Automatically segments data based on feature values to predict a target.
Why it's great: Provides **if-then rules** and thresholds you can directly interpret.

> *Insight:* "Customers under age 25 with fewer than 3 support interactions are 75% more likely to churn."

---

2. Clustering (e.g., k-means, DBSCAN) – for Pattern Discovery

What it does: Groups rows based on similarity in features.
Why it's great: Helps discover **natural customer segments**, behaviors, or operational patterns.

> *Insight:* "There are 3 customer types: bulk buyers, discount seekers, and one-time visitors."

---

3. Dimensionality Reduction (e.g., PCA, t-SNE, UMAP) – for Structure Detection

What it does: Projects high-dimensional data into 2D or 3D space.
Why it's great: Reveals **clusters, trends, or outliers** in complex datasets.

> *Insight:* "A clear separation exists between transactions before and after policy change X."

---

4. Association Rule Mining (e.g., Apriori, FP-Growth) – for Market Basket Insights

What it does: Finds co-occurring items or behaviors in categorical data.
Why it's great: Outputs interpretable rules like "If A and B, then C."

> *Insight:* "If a customer buys Product X and Y, there's a 78% chance they'll buy Z."

---

5. SHAP Values – for Feature Influence Quantification

What it does: Quantifies each feature's contribution to a prediction (local or global).
Why it's great: Visualizes **feature importance and directionality** in a way humans can understand.

> *Insight:* "Higher `last_login_gap` increases churn risk significantly—especially over 14 days."

---

6. Survival Analysis (e.g., Kaplan-Meier) – for Time-to-Event Understanding

What it does: Models the probability of an event over time (e.g., churn, failure).
Why it's great: Reveals **when** events happen and **which features** delay or accelerate them.

> *Insight:* "95% of churn occurs within the first 30 days post-signup."

---

7. Partial Dependence Plots (PDPs) – for Sensitivity Analysis

What it does: Shows how changing a single variable affects model predictions, holding others constant.
Why it's great: Identifies **nonlinearities, plateaus, and interaction regions**.

> *Insight:* "Purchase likelihood increases with discount up to 30%, but plateaus after that."

---

8. ICE Plots (Individual Conditional Expectation) – for Customer-Specific Behavior

What it does: Plots the effect of a feature on prediction **for each row/customer**.
Why it's great: Reveals **heterogeneity** in behavior across the population.

> *Insight:* "Some users are highly sensitive to price increases; others are unaffected."

---

9. Feature Correlation + Network Graphs – for Structural Understanding

What it does: Maps relationships between features using correlation or mutual information.
Why it's great: Identifies **redundant, interacting, or proxy variables**.

> *Insight:* "`total_spent` and `number_of_items` are highly correlated; may be capturing the same signal."

Choosing the Right Tool

Goal	Recommended Techniques
Find rules or explainable segments	Decision Trees, Association Rules
Understand structure or behavior groups	Clustering, PCA, t-SNE
Quantify feature impact	SHAP, PDP, ICE
Analyze behavior over time	Survival Analysis, Time Series Aggregates
Detect feature redundancy or proxies	Correlation matrix, Mutual Info, Feature Networks

---

Bonus: When to Use These in the EDA Process

Phase	Techniques
Phase 1 (Triage)	Correlation matrix, Feature types, Missingness heatmaps
Phase 2 (Prioritize)	Decision trees, Clustering, SHAP (global)
Phase 3 (Synthesize)	Association rules, PDP, ICE, Survival curves, SHAP (local)

Yes, **using decision trees for discovering data insights** is not only possible—it's a fantastic, underused strategy in exploratory data analysis (EDA), especially for:

- Understanding what drives your target variable
- Identifying **interaction effects**
- Discovering **thresholds** or **rules** hidden in your data

This technique is sometimes called **"white-box EDA"**, because decision trees are **interpretable models** that reveal logic patterns in data.

Why Decision Trees Are Insightful for EDA

Feature	Why It Helps
Splits data using clear rules	Exposes thresholds like "sales > 500" that segment behavior
Works on mixed types	Can split on numeric, categorical, and datetime
Captures interactions	Sees how two variables combine to predict the outcome
Feature hierarchy	Shows which features matter most and in what order

---

Example Use Cases for Data Insight Discovery

---

1. Find Drivers of a Binary Outcome (e.g., Churn, Purchase)

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X, y)  # where X = features, y = binary target

plt.figure(figsize=(12, 6))
plot_tree(clf, feature_names=X.columns, class_names=["No", "Yes"], filled=True)
plt.show()
```

You Learn:

- Which **features split first** → strongest predictor
- What thresholds define the groups
- How **subgroups behave differently**

Example Insight: > "Customers with usage < 300 mins and tenure < 6 months are 80% likely to churn. Loyalty improves after 6 months."

---

2. Quantify Rules in Numeric Target Prediction (e.g., Revenue, Spend)

Use `DecisionTreeRegressor`:

```
from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor(max_depth=3)
reg.fit(X, y_continuous)
plot_tree(reg, feature_names=X.columns, filled=True)
```

Example Insight: > "Orders with quantity > 12 and from Region C have 3x higher average revenue."

---

3. Compare Segments Across Categorical Target

Use trees to **explain imbalance or unexpected trends** in target distributions:

- Why is Region A returning more products?
- What characterizes users who upgrade plans?
- What distinguishes long-stay vs. short-stay patients?

Train a classifier and analyze the first few splits.

---

4. Find Feature Interactions

Suppose neither `age` nor `product_type` alone explains churn well—but their combo does.

Trees can reveal: > "Young customers on the Premium plan churn 4x more than older customers on Basic."

---

5. Spot Outliers or Anomalies

By examining **leaves with low sample counts** or high errors: - Who doesn't fit the rule? - Where is prediction difficult? - Could this subgroup be an anomaly or edge case?

Bonus: Tree-Based EDA + SHAP

Combine trees with SHAP (SHapley Additive exPlanations) to get:

- **Feature importance**
- **Local explanations**
- **Visual interactions**

Trees build the interpretable structure; SHAP quantifies contributions.

Tips for Using Trees for EDA

Tip	Why
Use **shallow trees** (max_depth 3–4)	Focus on interpretability, not accuracy
Use **min_samples_leaf** to avoid overfitting	Keeps splits meaningful
Limit features to a **subset of interest**	Keeps insight focused
Export trees as **rules** (if-then format)	Great for explaining decisions or policies

---

Summary

Using decision trees in EDA helps you: - Surface **explainable rules** - Discover **segment-based insights** - Prioritize **important features** - Uncover **interactions and thresholds**

Absolutely! Let's dive deep into:

---

2. Clustering (e.g., k-means, DBSCAN) – for Pattern Discovery

(from model-driven EDA techniques)

What Is Clustering?

Clustering is an **unsupervised learning** technique that **groups similar data points** together based on their feature values—without using a target variable. It's one of the most effective tools for **pattern discovery** in EDA.

> It answers:
> - "What natural segments exist in my data?"
> - "Are there different types of users, products, or behaviors?"
> - "Who looks unusual or doesn't belong in any group?"

Why Use Clustering for EDA?

Benefit	Why It's Insightful
Discover hidden patterns	See groups that weren't obvious
Segment users or items	Helps with personalization, targeting
Detect outliers	Points that don't belong in any cluster
Compare cluster behavior	Profile each group by summary stats
Feature interaction insight	Sometimes clusters reflect interactions of multiple variables

---

Common Clustering Methods

Algorithm	Best For	Notes
k-means	Dense, round-ish clusters	Fast & popular, but needs k
DBSCAN	Irregular shapes + outliers	Great for noise detection
Agglomerative	Hierarchical data	Produces a tree of clusters
HDBSCAN	Smart density + noise	Robust, no need to specify k

Example Walkthrough: k-means Clustering in Python

Let's use a synthetic customer behavior dataset:

Step 1: Simulate or Load Your Data

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.DataFrame({
    'annual_income': [15, 16, 17, 30, 31, 35, 80, 85, 88],
    'spending_score': [39, 81, 6, 77, 40, 45, 20, 90, 15]
})
```

Step 2: Preprocess (Scaling is Important!)

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)
```

Step 3: Apply k-means

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)
```

Step 4: Visualize the Clusters

```
plt.figure(figsize=(8, 6))
for c in df['cluster'].unique():
    subset = df[df['cluster'] == c]
    plt.scatter(subset['annual_income'], subset['spending_score'], label=f'Cluster {c}')
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score")
plt.title("Customer Segments (k-means)")
plt.legend()
plt.grid(True)
plt.show()
```

Now Ask: What Do the Clusters Mean?

Cluster ID	Summary	Business Interpretation
0	Low income, low spend	Price-sensitive, low LTV
1	Mid-income, high spend	Core buyers, target for upselling
2	High income, low spend	Under-engaged premium segment

EDA Insight Statements from Clustering

> "We identified 3 distinct customer segments. One group consists of high-income but low-spending users—this could represent an opportunity for engagement campaigns. Another group has high spend but mid-income, suggesting brand loyalty or emotional connection."

> "Customers in Cluster 2 have a 3x higher average return rate. They also appear more frequently in Region C, suggesting geographic behavior differences."

Advanced EDA: Profile Clusters

```
df.groupby('cluster').agg(['mean', 'count'])
```

- Compare **behavioral or demographic fields** across clusters
- Add **target rates** if you're modeling (e.g., churn rate per cluster)

Alternative: DBSCAN for Anomaly Detection

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=3)
df['dbscan_cluster'] = dbscan.fit_predict(X_scaled)
```

Cluster `-1` = **outliers**. Use this to flag suspicious entries, edge behavior, or operational noise.

When to Use Clustering for EDA

Use Case	What You Get
Customer behavior patterns	Segment strategies or personalization
Outlier detection	DBSCAN/HDBSCAN flags rare cases
Understand diversity in the dataset	Visual and statistical group differences
Preprocessing step before modeling	Create cluster membership as a feature

---

Caveats

- Requires **scaling** (especially k-means)
- k-means assumes **spherical clusters**—not ideal for all data
- Choosing `k` may require **elbow method** or **silhouette score**

- Clusters can be **sensitive to noise or outliers**

Summary

Tool	Purpose
`KMeans`	Quick segmentation
`DBSCAN`	Outlier-aware clustering
`Agglomerative`	Hierarchies and dendrograms
`groupby('cluster')`	Profiles for business impact

Absolutely! Let's break down and **deep-dive into**:

---

3. Dimensionality Reduction (e.g., PCA, t-SNE, UMAP) – for Structure Detection

(from model-driven EDA techniques)

What Is Dimensionality Reduction?

Dimensionality reduction techniques **transform high-dimensional data into lower dimensions (2D or 3D)**—while trying to **preserve its structure, patterns, and relationships**.

> Think of it like compressing a complex space so that you can **visualize it**, **detect clusters**, or **identify hidden structure** in a way your eyes and mind can grasp.

These methods are particularly useful when you: - Have many features (10+) - Want to **see data structure visually** - Need to **understand relationships** between observations - Want to find **groupings, gradients, or anomalies**

Three Main Tools (Each with Unique Strengths)

Method	Best For	Description
PCA (Principal Component Analysis)	Global patterns, feature variance	Linear projection; captures directions of max variance
t-SNE (t-distributed Stochastic Neighbor Embedding)	Visualizing clusters	Nonlinear; emphasizes local structure
UMAP (Uniform Manifold Approximation and Projection)	Visual + real structure	Nonlinear; preserves global and local relationships

---

Why Use These for EDA?

EDA Goal	How It Helps
See patterns	Reduce 10+ dimensions to 2D for visualization
Spot clusters	See whether the data naturally groups
Detect outliers	Outliers become visible as points far from any cluster
Preprocessing for clustering or modeling	Use top principal components instead of original noisy features
Interpret "hidden" relationships	Sometimes important axes aren't obvious in raw features

---

Example: Visualizing High-Dimensional Customer Data with PCA + t-SNE + UMAP

Let's walk through an example using simulated customer data:

Step 1: Generate Sample Data

```
from sklearn.datasets import make_blobs
import pandas as pd
import numpy as np

# Simulate customer features
X, y = make_blobs(n_samples=300, centers=4, n_features=6, random_state=42)
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])])
```

Step 2: Standardize the Data

```
from sklearn.preprocessing import StandardScaler
X_scaled = StandardScaler().fit_transform(df)
```

3A. PCA – Principal Component Analysis

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='Set1')
plt.title("PCA Projection (2D)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.show()
```

What You Learn:

- Which directions carry the most variance
- Whether clusters exist globally
- Whether a few components can summarize many features

3B. t-SNE – Nonlinear Embedding for Cluster Visualization

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='Set1')
plt.title("t-SNE Projection")
plt.grid(True)
plt.show()
```

What You Learn:

- Highly effective for **discovering clusters**
- t-SNE **pulls similar points together** and pushes dissimilar ones apart
- Excellent for revealing **latent segments** even when features are noisy

---

3C. UMAP – The Balance Between Global and Local

```
import umap.umap_ as umap

umap_model = umap.UMAP(n_neighbors=15, min_dist=0.1, random_state=42)
X_umap = umap_model.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap='Set1')
plt.title("UMAP Projection")
plt.grid(True)
plt.show()
```

What You Learn:

- UMAP preserves both local neighborhood structure **and** global layout
- UMAP is faster than t-SNE and more stable
- Works well with high-dimensional categorical embeddings

When Should You Use Each?

Goal	Use
You want **variance explanation**	PCA
You want **cluster visualization**	t-SNE or UMAP
You want to **reduce dimensions before modeling**	PCA (or UMAP)

Goal	Use
You have **nonlinear relationships**	t-SNE or UMAP
You want **interpretable axes**	PCA only

Practical EDA Use Cases

Use Case	Insight
Understand segments in user behavior	See groups that emerge from activity features
Spot anomalies	Points far from clusters = potential outliers
Identify hidden drivers	Loadings from PCA show which features explain variability
Preprocessing before clustering	UMAP before DBSCAN = powerful combination
Creating visualizations for stakeholders	Give interpretable snapshots of complex behavior

---

Tips & Caveats

Consideration	Why It Matters
Always **standardize** data first	PCA/t-SNE/UMAP are sensitive to scale
t-SNE is **non-deterministic**	Use `random_state` for reproducibility
UMAP and t-SNE **distort distances**	Don't interpret distances literally
PCA components are **linear combinations**	You can use `.components_` to interpret them
Use 3D UMAP/t-SNE for visualizations	Ideal for dashboards or deep dives

---

Summary

Technique	What It Gives You
PCA	Linear structure, interpretable directions
t-SNE	Nonlinear cluster visualization
UMAP	Combined global + local structure, fast & interpretable

Absolutely! Let's unpack one of the most **classic yet underrated model-driven EDA techniques**:

---

4. Association Rule Mining (e.g., Apriori, FP-Growth) – for Market Basket Insights

(from model-driven EDA techniques)

What Is Association Rule Mining?

Association Rule Mining is an unsupervised learning method that **discovers co-occurrence patterns**—like which items are often purchased together or which behaviors happen in sequence.

> It answers questions like:
> "If a user buys Product A and B, what's the likelihood they'll also buy Product C?"
> "Which user behaviors tend to co-occur?"
> "What combinations of features or actions are meaningful?"

This is the logic behind: - Amazon's "Frequently Bought Together" - Retail basket analysis - Fraud detection sequences - Trigger chains in user behavior

---

Why Use Association Rules in EDA?

Use Case	Value
Market basket analysis	See what products co-occur in transactions
Behavior analysis	Discover user action patterns (e.g., open → click → buy)
Feature interaction discovery	Understand feature combinations that imply outcomes
Anomaly detection	Spot unexpected or rare combinations

Key Metrics for Association Rules

Metric	Meaning
Support	How often items A and B appear together
Confidence	Likelihood of B given A (conditional probability)
Lift	How much more likely A and B co-occur than by chance (lift > 1 = interesting)

Rule Format:

> **If [antecedent], then [consequent]**

Example: Market Basket Analysis with `mlxtend`

We'll use a dataset of transactions from a fictional grocery store.

Step 1: Create a Basket Format

```python
import pandas as pd

# Example transaction data
transactions = [
    ['milk', 'bread', 'butter'],
    ['bread', 'butter'],
```

```python
    ['milk'],
    ['milk', 'bread'],
    ['butter', 'bread'],
    ['milk', 'butter'],
    ['milk', 'bread', 'butter'],
]

# Convert to basket-style format (one-hot encoded)
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

Step 2: Generate Frequent Itemsets

```python
from mlxtend.frequent_patterns import apriori

frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
frequent_itemsets
```

Step 3: Generate Association Rules

```python
from mlxtend.frequent_patterns import association_rules

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
```

Example Output:

Antecedents	Consequents	Support	Confidence	Lift
{bread}	{butter}	0.57	0.80	1.14
{butter}	{milk}	0.43	0.67	1.12
{milk, bread}	{butter}	0.43	0.86	1.23

How to Interpret

> Rule: "If customer buys milk and bread, then they also buy butter (confidence = 86%, lift = 1.23)"

EDA Insight: > Suggests bundling milk + bread + butter in promotions. This combo drives higher co-occurrence than chance alone.

Real-World Applications

Domain	Insight Example
Retail	Customers who buy diapers and beer often buy chips
E-commerce	Visitors who view Product A often also add Product B
Healthcare	Patients with condition A and B often also take Drug C
Banking	Customers who defaulted often had high credit utilization + recent late payment
Web analytics	Users who start trial and click email 2 often convert to paid

---

Advanced Tips

- Use `min_support` and `min_lift` to tune rule quality
- Use `frozenset` strings or convert to readable text for dashboards
- Visualize rules as a **network graph** or **Sankey diagram**
- Combine with **clustering** to segment behavior then run rules within clusters

---

Cautions

Issue	Fix
Too many rules?	Increase support/lift/confidence
Meaningless rules?	Filter by domain logic
Rare combos?	Use with **confidence AND lift** to find impactful rules
One-hot encoding limits?	Use `mlxtend.preprocessing.TransactionEncoder` on raw data lists

Summary

Step	Purpose
One-hot encode transactions	Input for Apriori
Use `apriori()`	Find frequent itemsets
Use `association_rules()`	Extract meaningful rule combinations
Interpret via support, confidence, lift	Discover which combinations matter
Actionable insight	Bundle, recommend, intervene, or investigate

---

Absolutely! Let's go **deep into SHAP (SHapley Additive exPlanations)**—one of the most powerful tools for **model-driven EDA** and one of the best ways to understand **feature influence on predictions**.

---

5. SHAP Values – for Feature Influence Quantification

(from model-driven EDA techniques)

What Are SHAP Values?

SHAP values are a game-theoretic approach to explain **how much each feature contributed** to a specific model prediction.

> Think of SHAP as: - "How did each feature **push the model's prediction** away from the baseline?" - For every prediction, you get a **+/- contribution** from each feature

---

Why SHAP Is Unique (and Powerful)

Advantage	Description
Local + global	You can explain **individual predictions** and **overall model behavior**
Model-agnostic	Works with **any model** (tree, linear, NN, etc.)
Additive	SHAP values **sum to the prediction** (interpretable math)
Fair attribution	Based on Shapley values from game theory (fair split of credit)
Visuals are excellent	Force plots, waterfall plots, summary plots, interaction plots

---

SHAP in EDA: What You Can Learn

Use Case	Insight
Feature importance	Which features impact predictions most?
Direction of impact	Does high age increase or decrease churn risk?
Thresholds	At what point does income start influencing risk?
Interactions	Are there combinations that flip effects?
Local outliers	Why did this customer behave differently?

Example: Using SHAP for EDA on Classification

We'll use the **Titanic survival dataset** to understand what influences survival.

Step 1: Load and Prepare the Data

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

df = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df = df[['Survived', 'Pclass', 'Sex', 'Age', 'Fare', 'SibSp', 'Parch']]
df = df.dropna()
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
```

```python
X = df.drop('Survived', axis=1)
y = df['Survived']
```

Step 2: Train a Model

```python
model = XGBClassifier(n_estimators=100, max_depth=3, random_state=42)
model.fit(X, y)
```

Step 3: Apply SHAP

```python
import shap

explainer = shap.Explainer(model)
shap_values = explainer(X)
```

Step 4: Global Feature Impact (SHAP Summary Plot)

```python
shap.plots.beeswarm(shap_values)
```

This shows **which features matter most**, and how they influence predictions.

Step 5: Local Explanation (Single Row)

```python
shap.plots.waterfall(shap_values[0])
```

Explains exactly **why this passenger** was predicted to survive or not.

Step 6: Dependence Plot (Feature Thresholds)

```python
shap.plots.scatter(shap_values[:, "Age"], color=shap_values)
```

Visualizes how **Age affects survival**, and whether it interacts with other features like `Sex`.

How to Interpret SHAP Values

Value	Meaning
Positive SHAP	Feature **increased the prediction** (pushed toward "1" for classification)
Negative SHAP	Feature **decreased the prediction**
Magnitude	Strength of the effect
Zero	Feature had no meaningful influence for this row

Common SHAP Visuals

Plot	Use
Beeswarm	Global overview of feature effects across all rows
Bar plot	Feature importance (average absolute SHAP value)
Waterfall	Breakdown of individual prediction
Force plot	Push-pull visualization of prediction
Dependence plot	Continuous variable vs SHAP value
Interaction plot	Pairs of variables and how they affect prediction together

---

Summary: Why SHAP Is a Game-Changer for EDA

Benefit	Why It Matters
Bridges modeling and interpretation	Works **before modeling** (as insight discovery) or **after modeling** (for audit/exploration)
Highly visual	Communicates well with stakeholders
Spot trends and interactions	Helps define rules and thresholds
Find "why this happened"	Best tool for **individual row interpretation**

Considerations

Caveat	Recommendation
Can be slow	Use `TreeExplainer` for tree-based models like XGBoost, LightGBM
Local explanations vary	Always check global patterns too
Correlated features	SHAP tries to compensate, but correlation can skew attribution

---

Absolutely! Let's dive into one of the most **powerful time-based EDA techniques** for understanding not just *what* happens, but *when* it happens:

---

6. Survival Analysis (e.g., Kaplan-Meier) – for Time-to-Event Understanding

(from model-driven EDA techniques)

What Is Survival Analysis?

Survival Analysis is a family of statistical methods for modeling **time until an event occurs**—such as churn, failure, death, upgrade, conversion, etc.

It answers:
- "How long does it take for users to churn?"
- "What % of machines survive past 1,000 hours?"
- "When are customers most likely to convert?"
- "Which user types drop off sooner than others?"

Key Concepts

Term	Definition
Event	The outcome we're waiting for (e.g., churn, failure, death)
Duration	Time from start to the event or censoring
Censoring	Cases where we don't observe the event by the end of the study (e.g., still active)
Survival function (S(t))	Probability of **surviving past time t**
Hazard function	Instantaneous **risk of the event** at time t, given survival until then

---

Why Use Survival Analysis in EDA?

Goal	What It Reveals
Understand **when** outcomes occur	Go beyond binary classification (e.g., churn vs. no churn)
Detect **time-based risk patterns**	Are customers more likely to churn early or late?
Compare **groups over time**	Do Plan A users stay longer than Plan B?
Support **retention, maintenance, warranty** analysis	Predict when failure or drop-off is likely
Build **segment-level timelines**	Visualize time-to-event curves by user type, region, etc.

Example: Kaplan-Meier Survival Curves in Python

Let's look at an example of user churn over time (synthetic data).

Step 1: Simulate Time-to-Event Data

```
import pandas as pd
import numpy as np

np.random.seed(42)

# Simulated user data
df = pd.DataFrame({
```

```python
    'user_id': range(1, 101),
    'tenure_days': np.random.exponential(scale=365, size=100).astype(int),
    'churned': np.random.binomial(1, 0.7, size=100)  # 70% observed churn
})
```

- `tenure_days`: how long they stayed active
- `churned`: 1 = churned, 0 = still active (right-censored)

Step 2: Kaplan-Meier Estimator

```python
from lifelines import KaplanMeierFitter
import matplotlib.pyplot as plt

kmf = KaplanMeierFitter()
kmf.fit(durations=df['tenure_days'], event_observed=df['churned'])

kmf.plot_survival_function()
plt.title("Survival Curve: User Churn Over Time")
plt.xlabel("Days Since Signup")
plt.ylabel("Probability of Retention")
plt.grid(True)
plt.show()
```

Interpretation:

- The curve shows the **probability of users *not* churning over time**

- A steep drop early on suggests **high early churn**
- A plateau implies **stabilized long-term users**
- You can compare groups (e.g., subscription plans) using multiple curves

---

Step 3: Compare Groups (e.g., Plan A vs Plan B)

```python
df['plan'] = np.random.choice(['A', 'B'], size=100)

# Plot grouped survival curves
for plan in df['plan'].unique():
    kmf.fit(durations=df[df['plan'] == plan]['tenure_days'],
            event_observed=df[df['plan'] == plan]['churned'],
            label=f'Plan {plan}')
    kmf.plot_survival_function()

plt.title("Survival by Plan Type")
plt.xlabel("Days")
plt.ylabel("Retention Probability")
plt.grid(True)
plt.legend()
plt.show()
```

Insight: If Plan A has a steeper drop than Plan B, it may be riskier for early churn.

More Advanced Survival Tools

Tool	Purpose
Cox Proportional Hazards Model	Model hazard rate using covariates (age, plan, etc.)
Nelson-Aalen Estimator	Estimate cumulative hazard
Log-rank test	Test if two survival curves are significantly different

---

Real-World EDA Examples

Domain	Example Use
SaaS	Understand when users churn or upgrade
Healthcare	Time until readmission or complication
Manufacturing	Time-to-failure for machines or parts
Education	When students drop a course or stop attending
Finance	Duration until default or early repayment

Summary

Element	Insight
Kaplan-Meier	Retention probability over time
Survival function	Who is more or less likely to "survive"?
Censoring	Models incomplete observations correctly
Group comparison	Retention by plan, cohort, region, etc.

Absolutely! Let's explore **Partial Dependence Plots (PDPs)**—a vital part of model-driven EDA that helps you understand **how changes in a feature affect your model's predictions**.

---

7. Partial Dependence Plots (PDPs) – for Sensitivity Analysis

(from model-driven EDA techniques)

What Is a Partial Dependence Plot?

A **Partial Dependence Plot (PDP)** shows the **marginal effect** of one or two features on the predicted outcome of a machine learning model, **averaging out all other features**.

> Think of it like this:
> "If we vary `feature X` while keeping everything else constant, how does the model's prediction change?"

Why Use PDPs in EDA?

Goal	What PDP Shows
Understand model behavior	Shows relationship between a feature and predictions
Perform sensitivity analysis	Are predictions stable across feature ranges?
Detect nonlinear effects	U-shapes, plateaus, thresholds, etc.
Spot regions of high/low risk	PDP shows where risk increases or levels off
Communicate findings	PDPs are intuitive, stakeholder-friendly plots

---

Ideal Use Cases for PDP

- Churn risk increases when tenure < 3 months
- High claim amounts sharply increase insurance fraud probability
- Revenue impact plateaus once ad spend > $100k
- Age or credit score drives risk in a nonlinear way

Step-by-Step Example: PDP with a Classification Model

We'll use a simple dataset and scikit-learn's PDP tools.

Step 1: Load Sample Data

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
target = data.target
```

Step 2: Train a Model

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df, target, random_state=42)
model = GradientBoostingRegressor().fit(X_train, y_train)
```

Step 3: Plot a PDP

```
from sklearn.inspection import plot_partial_dependence
import matplotlib.pyplot as plt
```

```
features = ['AveRooms', 'HouseAge']
plot_partial_dependence(model, X_train, features, kind='average', grid_resolution=50)
plt.suptitle("Partial Dependence Plots")
plt.tight_layout()
plt.show()
```

What You Might See

- As `HouseAge` increases, predicted housing prices rise sharply up to ~25 years, then level off
- `AveRooms` shows a nonlinear relationship with price—higher rooms always higher prices

---

Interpreting PDPs

Shape	Interpretation
Upward slope	Feature increases outcome
Downward slope	Feature decreases outcome
U-shape or inverted U	Nonlinear or optimal ranges
Plateau	Diminishing returns
Steep drop	Sensitivity region or model instability

Advanced: 2D PDP (Feature Interactions)

```
plot_partial_dependence(model, X_train, [('AveRooms', 'HouseAge')], grid_resolution=30)
```

Shows a 3D surface plot of how **two features interact** in influencing predictions.

Key Insights You Can Derive

Insight	Example
Thresholds	Income below $30k sharply increases churn
Diminishing returns	Marketing spend beyond $50k yields flat gains
Feature effects	HouseAge has stronger effect on price than MedInc
Nonlinear behavior	Customer engagement peaks at moderate usage

---

Tips for Using PDPs Well

Tip	Why It Helps
Use with **tree-based models**	PDPs are most stable with decision trees, random forests, gradient boosting
Avoid high correlation	If features are highly correlated, PDP assumptions may break
Use `kind='both'`	Combines averaged and individual lines for richer visual

Tip	Why It Helps
Use `ice_lines=True` (or SHAP instead)	For instance-level behavior (see ICE/SHAP)
Pair with SHAP or feature importance	Validate what you see with other model explainability tools

---

PDP vs. SHAP vs. ICE

Tool	Focus	Best For
PDP	Average global effect	General model insight
ICE	Individual curves per sample	Heterogeneity analysis
SHAP	Local + global explanation	Feature importance + interactions

---

Summary

Tool	What It Does
PDP	Shows how changing a feature impacts prediction
Good for	Visualizing thresholds, plateaus, nonlinearities
Not good for	Highly correlated features, individual behavior
Output	Easy-to-read, stakeholder-friendly graphs

Absolutely! Let's explore **ICE plots** (Individual Conditional Expectation)—a powerful tool for understanding **how a feature affects predictions for individual rows (e.g., customers)**, rather than just on average.

---

8. ICE Plots (Individual Conditional Expectation) – for Customer-Specific Behavior

(from model-driven EDA techniques)

What Is an ICE Plot?

ICE plots show how a model's prediction changes as a **single feature is varied**, for **one data point at a time**, while keeping all other features constant.

> Think of it like: "If this customer's income changed, how would their predicted risk change—*for them specifically*?"
>
> It's like a personalized **sensitivity test** for each row.

ICE plots give you **many individual lines**, one per observation, showing: - Personal response curves to a given feature - Variation across users - Hidden subgroups with different reactions

---

How ICE Plots Differ from PDPs

PDP	ICE
Shows **average effect**	Shows **individual effects**
Smooth, single line	Many lines (one per sample)
Good for global trends	Good for **heterogeneity**
Can miss subgroups	Reveals subgroups and exceptions
Easy to summarize	Rich, nuanced, more complex

Why Use ICE Plots in EDA?

Goal	Why It Matters
Understand individual behavior	Does *this* customer respond like the average one?
Reveal subgroups	Some customers increase risk with age, others decrease
Detect heterogeneity	Not all data points are equal
Spot unfair or biased model effects	See if certain groups are treated differently
Create user personas or segments	Based on how they respond to features

---

Example: ICE Plot in Python (scikit-learn + PDPBox)

Let's walk through how to use ICE plots on a housing price prediction model.

Step 1: Load and Prepare Data

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

Step 2: Train a Model

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df, y, random_state=42)
model = GradientBoostingRegressor().fit(X_train, y_train)
```

Step 3: ICE Plot with PDPBox

```python
from pdpbox import pdp

# ICE plot for 'AveRooms'
ice_plot = pdp.pdp_isolate(
    model=model,
    dataset=X_test,
    model_features=X_test.columns.tolist(),
    feature='AveRooms',
    num_grid_points=30,
    grid_type='percentile'
)

pdp.pdp_plot(ice_plot, 'AveRooms', plot_lines=True, frac_to_plot=0.1)
```

What You Might See:

- Most lines go up: More rooms = higher predicted house value
- Some lines go flat: Room count doesn't affect prediction for certain homes
- A few drop: Maybe in low-income regions, more rooms don't increase value

---

How to Read ICE Plots

Feature	ICE Line Behavior	Insight
Flat line	Feature has no effect on that sample	
Steep slope	Feature highly influences prediction for that sample	
Crossing lines	Feature affects samples differently → **heterogeneity**	
Outlier lines	Sample behaves differently from the group → investigate it!	

ICE + PDP Hybrid: Centered ICE (c-ICE)

To better compare shapes (ignoring vertical shifts), you can center each line at the feature's baseline value. Most libraries like SHAP or PDPBox support this.

ICE Plot Use Cases

Domain	Use
Customer churn	See how each customer's churn risk responds to changing contract length
Loan approval	Understand how income affects approval probability by applicant
Healthcare	See how a patient's predicted risk responds to age or dosage

Domain	Use
Pricing	Understand how ad spend or discount % impacts predicted revenue by customer type

---

Summary

Element	Description
ICE plot	One line per observation, showing model response as one feature varies
Good for	Detecting individual-level behavior and segment diversity
Best used with	Tree-based models (Random Forest, XGBoost, GBM) or any scikit-learn model
Pairs well with	PDP (global) and SHAP (local + global)
Output	Lines that expose model structure, bias, and subgroup behaviors

---

Quick Comparison Table

Tool	Focus	Good For
PDP	Average trend	Summary insight
ICE	Individual behavior	Segment and exception discovery
SHAP	Local + global effects	Feature attribution, fairness, interpretability

---

Absolutely! Let's do a **deep dive** into a powerful yet often underutilized model-driven EDA technique:

---

9. Feature Correlation + Network Graphs – for Structural Understanding

(from model-driven EDA techniques)

What Is It?

This technique combines:

1. **Feature Correlation**: Measures **linear** or **nonlinear** associations between features

2. **Network Graphs**: Visual structures that **map relationships as nodes and edges**

   Think of it as creating a **map of how features are connected**—a "social network" of your data variables.

What It Tells You

Goal	What You Learn
Detect **redundancy**	Are multiple variables capturing the same signal?
Find **latent structures**	Are there feature "families" or tightly-knit subgroups?
Improve **feature selection**	Drop or combine highly correlated variables
Uncover **unexpected links**	Reveal proxy relationships or engineered variable overlap
Support **explainable models**	Helps diagnose multicollinearity or bias in ML models

---

Tools You Can Use

Tool	Use
`pandas.corr()`	Pearson correlation (linear)
`mutual_info_regression/classif`	Nonlinear correlation (entropy-based)
`networkx`	Create and visualize networks in Python
`matplotlib`, `plotly`, or `pyvis`	Visualize graphs

Example: Create a Feature Correlation Network Graph (Pearson)

Step 1: Create Synthetic Dataset

```
import pandas as pd
import numpy as np

np.random.seed(42)
df = pd.DataFrame({
    'age': np.random.randint(20, 70, 100),
    'income': np.random.normal(50000, 10000, 100),
    'expenses': np.random.normal(30000, 5000, 100),
    'savings': np.random.normal(10000, 3000, 100)
})

# Inject correlation
df['total_spent'] = df['income'] - df['savings']
df['debt_ratio'] = df['expenses'] / df['income']
```

Step 2: Calculate Correlation Matrix

```
corr_matrix = df.corr().abs()  # absolute correlation
```

Step 3: Filter for Strong Correlations

```
corr_pairs = corr_matrix.stack().reset_index()
corr_pairs.columns = ['var1', 'var2', 'correlation']
```

```
corr_pairs = corr_pairs[corr_pairs['var1'] != corr_pairs['var2']]
corr_pairs = corr_pairs[corr_pairs['correlation'] > 0.5]
```

Step 4: Create Network Graph

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
for _, row in corr_pairs.iterrows():
    G.add_edge(row['var1'], row['var2'], weight=row['correlation'])

pos = nx.spring_layout(G, seed=42)

plt.figure(figsize=(8, 6))
nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray', node_size=2000, font_size=
edge_labels = {(row['var1'], row['var2']): f"{row['correlation']:.2f}" for _, row in corr_pairs.iterrows
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Feature Correlation Network Graph")
plt.show()
```

What You Might See

Relationship	Meaning
`income total_spent`	Highly correlated → redundant
`income debt_ratio`	Inverse relationship
`savings total_spent`	Negative correlation: more savings = less spent

Optional: Use Mutual Information Instead

```
from sklearn.feature_selection import mutual_info_regression

def compute_mi_matrix(df):
    columns = df.columns
    mi_matrix = pd.DataFrame(index=columns, columns=columns)

    for col1 in columns:
        for col2 in columns:
            if col1 != col2:
                mi = mutual_info_regression(df[[col1]], df[col2])[0]
                mi_matrix.loc[col1, col2] = mi
    return mi_matrix.astype(float)
```

Use this when relationships are **nonlinear** or involve **categorical + numeric** interactions.

What You Can Learn from the Network Graph

Insight Type	Example
Redundant features	`income`, `total_spent` — drop one or combine
Feature clusters	`savings`, `expenses`, `debt_ratio` may belong to a financial health cluster
Proxy variables	If `age` is strongly linked to `income`, be careful about fairness/bias
Unexpected links	See if two engineered features overlap more than expected
Target proxy	If the target appears strongly connected to a feature, check for leakage

---

Real-World Use Cases

Domain	Use
Finance	Spot correlated credit features before building risk model
Healthcare	Identify diagnostic features that track the same conditions
Manufacturing	See how sensor variables interact or duplicate each other
Marketing	Combine user behavior metrics into more compact feature sets
EDA prep	Identify what to drop, keep, or combine before modeling

---

Summary

Tool	Purpose
Pearson or MI matrix	Quantify pairwise similarity
Network graph	Visually explore structure of feature relationships
Use case	Reduce redundancy, discover structure, improve interpretability
Key benefit	Helps with **feature selection**, **multicollinearity**, and **data storytelling**
