When drop(), concat(), include axis=1

**TargetLeak**: timeline, dependency & circular, giveaway. select\_dtypes(include/exclude=['object', 'int64', 'float64']); ANOVA for features selection; chi for target leak;

**Pearsons**: df\_num = df.select\_dtypes(exclude='object'); corr\_mat=df\_num.corr(); corr\_mat.style.background\_gradient(cmap='Blues'); # high = sus have target leakage.

**ANOVA**: from scipy.stats import f\_oneway; categorical\_columns = pd.DataFrame(df['CPU3']); alpha = 0.05; for column in categorical\_columns: categories = df[column].unique(); p\_values = []

for category in categories: p\_value = f\_oneway(df['Price\_euros'][df[column] == category], df['Price\_euros'])[1]; p\_values.append(p\_value)

**chi**: df\_obj = df.select\_dtypes(include=['object']); new\_df = pd.concat([df\_obj, df[TARGET]], axis=1); from scipy.stats import chi2\_contingency; predictor\_columns = [col for col in new\_df.columns if col != 'Price\_euros']; for predictor in predictor\_columns: crosstab = pd.crosstab(new\_df[predictor], new\_df['Price\_euros']); chi2, p, \_, \_ = chi2\_contingency(crosstab) **label**: from sklearn.preprocessing import LabelEncoder; from collections import defaultdict; d = defaultdict(LabelEncoder); df\_obj = df\_obj.apply(lambda x: d[x.name].fit\_transform(x.astype(str))) **partition**: from sklearn.model\_selection import train\_test\_split; X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) **DTR**: (no need scale; default overfit; high Cardin Bad: more splits, false sense of importance) from sklearn.tree import DecisionTreeRegressor; my\_dtr = DecisionTreeRegressor(random\_state=1234).fit(X\_train , y\_train); y\_pred\_dtr = my\_dtr.predict(X\_test) **feat\_impt**: feats = {}; for feature, importance in zip(X.columns, my\_dtr.feature\_importances\_): feats[feature] = importance; <br> importances = pd.DataFrame.from\_dict(feats, orient='index').rename(columns={0: 'Importance'}); top15import = importances.sort\_values(by="Importance" ,ascending=False).head(15) **tree1ugly**: from sklearn.tree import export\_text; r = export\_text(my\_dtr, feature\_names=list(X.columns), max\_depth=5); **tree2**: fig = plt.figure(figsize=(25,20)); tree.plot\_tree(my\_dtr, feature\_names=list(X\_train.columns), max\_depth = 3, fontsize=12, filled=True); plt.show(); **measure**: mean\_squared\_error(y, y\_pred) **residual** = y\_test.**reset\_index(drop=True**) - y\_pred; **comparePlot**: fig, ax = plt.subplots(figsize=(18, 6)); ax.plot(residual\_lr, "red", label = 'LR'); ax.plot(residual\_dtr, "blue", label = 'DTR'); plt.title('Residual Plot of DTR Vs LR'); ax.legend(); plt.show() **multivarAnalysis**: sns.catplot(x="TypeName", y="Company", hue="Price\_euros", data=decoded\_X\_test, palette='bright'); plt.show(); **edit size of plot**: sns.set(rc={'figure.figsize':(12,36)}) **binary**: from sklearn.metrics import accuracy\_score, f1\_score, recall\_score; accuracy\_score(y\_test, y\_pred\_dtc) **matrix**: from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay;cm = confusion\_matrix(y\_test, y\_pred\_dtc, labels=my\_dtc.classes\_); disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=my\_dtc.classes\_); disp.plot(); plt.show() **DTC**: from sklearn.tree import DecisionTreeClassifier; my\_dtc = DecisionTreeClassifier(random\_state=1234).fit(X\_train , y\_train); **Machine learning** identifies patterns using statistical learning and computers by unearthing boundaries in data sets. You can use it to make predictions. **Overfitting** happens when some boundaries are based on on distinctions that don't make a difference. You can see if a model overfits by having test data flow through the model.

 