ChatGPT

Data Science Cheatsheet (General + Clustering + Modeling Strategy)

General Modeling Strategy

Start Simple. Build Up Gradually.

Start with a baseline model using only the most clearly relevant features. Once it works, slowly add new features and tune your model.

Example: Starting Small

```
# Minimal features known to correlate
X = train_set[["RoomService", "Spa", "VRDeck", "CryoSleep"]].copy()
X["CryoSleep"] = X["CryoSleep"].astype(bool).astype(int)
X = X.fillna(0)

y = train_set["Transported"].astype(bool).astype(int)

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y,
test_size=0.2, random_state=42)

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

print("Train Acc:", clf.score(X_train, y_train))
print("Val Acc:", clf.score(X_val, y_val))
```

📊 Data Visualization

Correlation Heatmap

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df.corr(numeric_only=True), cmap="YlGnBu")
plt.show()
```

📈 Scatter & Pair Plots

```
sns.scatterplot(x='feature_x', y='feature_y', hue='category_feature',
data=df)
plt.title('Feature X vs. Feature Y by Category')
plt.show()

sns.pairplot(df, vars=['f1','f2','f3','f4','f5'], hue='target')
plt.show()
```

📉 Aggregation Visualization

```
ax = df.query('group_column != "Other"') \
       .groupby('group_column')['numeric_feature'] \
       .agg(['mean', 'count']) \
       .query('count >= 10') \
       .sort_values('mean')['mean'] \
       .plot(kind='barh', figsize=(12, 5), title='Average Value by Group')
ax.set_xlabel('Average Value')
plt.show()
```

📋 Data Inspection & Cleaning

```
df.dtypes
df['date_column'] = pd.to_datetime(df['date_column'])
df.isna().sum()
df.duplicated()
df.query('col == "value"')
df["category_col"].value_counts()
df["num_col"].hist(bins=20)
```

Train-Test Split

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in split.split(df, df["target"]):
    train_set = df.loc[train_idx]
    test_set = df.loc[test_idx]
```

Pipeline + Transformers

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

class ColumnImputer(BaseEstimator, TransformerMixin):
    def __init__(self): self.imputers = {}
    def fit(self, X, y=None):
        for col in ["RoomService", "Spa", "VRDeck"]:
            self.imputers[col] = SimpleImputer(strategy="mean").fit(X[[col]])
        return self
    def transform(self, X):
        for col, imp in self.imputers.items():
            X[col] = imp.transform(X[[col]])
        return X

class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self): self.encoders = {}
    def fit(self, X, y=None):
        for col in ["HomePlanet", "Destination"]:
            enc = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
            enc.fit(X[[col]])
            self.encoders[col] = enc
        return self
    def transform(self, X):
        for col, enc in self.encoders.items():
            df_encoded = pd.DataFrame(enc.transform(X[[col]]),
columns=enc.get_feature_names_out([col]), index=X.index)
            X = pd.concat([X.drop(columns=[col]), df_encoded], axis=1)
        return X

class FeatureDropper(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None): return self
    def transform(self, X): return X.drop(["irrelevant1", "irrelevant2"],
axis=1, errors="ignore")
```

Full Pipeline

```python
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("imputer", ColumnImputer()),
    ("encoder", FeatureEncoder()),
    ("dropper", FeatureDropper())
])
```

```python
train_set = pipeline.fit_transform(train_set)
test_set = pipeline.transform(test_set)
```

⚙ Preprocessing

```python
from sklearn.preprocessing import StandardScaler

X = train_set.drop(["Transported"], axis=1)
y = train_set["Transported"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y_array = y.to_numpy()
```

Model Training & Tuning

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10],
    "min_samples_split": [2, 5]
}

clf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=3, scoring="accuracy",
return_train_score=True)
grid_search.fit(X_scaled, y_array)
best_model = grid_search.best_estimator_
```

📊 Evaluation

```python
X_test = test_set.drop(["Transported"], axis=1)
y_test = test_set["Transported"].to_numpy()
X_test_scaled = scaler.transform(X_test)

accuracy = best_model.score(X_test_scaled, y_test)
print("Accuracy:", accuracy)
```

Clustering Concepts (from provided notebook)

🧼 Clean Product Codes

```
df = df[~df['StockCode'].isin(['D', 'M', 'C2', 'BANK CHARGES'])]
```

Aggregate Per Customer

```
df_grouped = df.groupby("CustomerID").agg({
    "InvoiceNo": "nunique",
    "Quantity": "sum",
    "UnitPrice": "mean",
    "InvoiceDate": "min"
})
```

📊 Normalize & Cluster

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

scaler = StandardScaler()
data_scaled = scaler.fit_transform(df_grouped)

kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(data_scaled)

print("Silhouette Score:", silhouette_score(data_scaled, clusters))
```

Final Tips

- Start with numeric features first.
- Add one feature at a time, evaluate impact.
- Drop original categorical column after one-hot encoding.
- Use `.groupby("user")["score"].mean()` to compute per-group metrics.
- Use `.agg({"col": "mean"})` for custom aggregations.
- Check performance with cross-validation or a stratified train/val split. ```