Cheat Sheet: Python para Data Science - Análisis Completo

Este documento es una guía completa y analítica basada en los ejercicios de las carpetas `clase_jf/` y `clase_io/` , así como en los exámenes de 2022, 2023 y 2024.

📚 Tabla de Contenidos

1. Python Básico
2. NumPy: Operaciones Numéricas
3. Pandas: Manipulación de Datos
4. Limpieza y Preparación de Datos
5. Visualización de Datos
6. Estadística
7. Machine Learning con Scikit-learn
8. Patrones Comunes en Exámenes
9. Resumen General

1. Python Básico

1.1 Variables y Tipos de Datos

Por qué es importante: Python es dinámicamente tipado, lo que significa que no necesitas declarar explícitamente el tipo de variable. Sin embargo, entender los tipos es crucial para evitar errores.

```
# Tipos básicos
entero = 42
flotante = 3.14
texto = "Data Science"
booleano = True
lista = [1, 2, 3, 4, 5]
tupla = (1, 2, 3)
diccionario = {'nombre': 'Juan', 'edad': 25}
```

Análisis: Las listas son mutables (se pueden modificar), las tuplas son inmutables. Los diccionarios son fundamentales para estructurar datos clave-valor.

1.2 Estructuras de Control

Bucles - For y While:

```
# For loop - itera sobre una secuencia
for i in range(5):
    print(f"Iteración {i}")

# List comprehension - forma pythónica de crear listas
cuadrados = [x**2 for x in range(10)]

# Comprehension con condición
pares = [x for x in range(20) if x % 2 == 0]
```

Por qué usar list comprehensions: Son más rápidas y legibles que los bucles tradicionales para operaciones simples.

Condicionales:

```
# If-elif-else
valor = 75
if valor >= 90:
    categoria = 'Excelente'
elif valor >= 70:
    categoria = 'Bueno'
else:
    categoria = 'Necesita mejorar'
```

1.3 Funciones

Definición y uso:

```
def calcular_estadisticas(datos):
    """
    Calcula media, mediana y desviación estándar.

    Args:
        datos (list): Lista de números

    Returns:
        dict: Diccionario con las estadísticas
    """
    import numpy as np
    return {
        'media': np.mean(datos),
        'mediana': np.median(datos),
        'std': np.std(datos)
    }

# Uso
resultado = calcular_estadisticas([1, 2, 3, 4, 5])
```

Análisis: Las funciones deben tener docstrings que expliquen qué hacen, sus parámetros y valores de retorno. Esto es especialmente importante en proyectos de data science.

1.4 Gestión del Entorno

Patrón común en notebooks:

```
from IPython import get_ipython
get_ipython().run_line_magic('reset', '-f')
get_ipython().run_line_magic('clear', '-f')

import os
# Verificar directorio de trabajo
print(os.getcwd())

# Cambiar directorio si es necesario
# os.chdir('/ruta/al/directorio')
```

Por qué: Limpiar el entorno asegura que no haya variables residuales de ejecuciones anteriores que puedan causar errores.

2. NumPy: Operaciones Numéricas

2.1 Creación de Arrays

Conceptos básicos:

```
import numpy as np

# Crear arrays de diferentes formas
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.zeros((3, 4))        # Array de ceros
arr3 = np.ones((2, 3))         # Array de unos
arr4 = np.arange(0, 10, 2)     # Array con secuencia
arr5 = np.linspace(0, 1, 5)    # 5 números equiespaciados entre 0 y 1
arr6 = np.random.rand(3, 3)    # Array aleatorio
```

Por qué NumPy: Los arrays de NumPy son mucho más eficientes que las listas de Python para operaciones numéricas, gracias a la implementación en C.

2.2 Indexación y Slicing

```
arr = np.array([10, 20, 30, 40, 50])

# Indexación básica
primer_elemento = arr[0]      # 10
ultimo_elemento = arr[-1]     # 50

# Slicing
primeros_tres = arr[:3]       # [10, 20, 30]
desde_segundo = arr[1:]       # [20, 30, 40, 50]
cada_dos = arr[::2]           # [10, 30, 50]

# Indexación booleana (muy común en análisis)
arr_mayor_25 = arr[arr > 25]  # [30, 40, 50]
```

Análisis: La indexación booleana es fundamental para filtrar datos basándose en condiciones.

2.3 Operaciones Vectorizadas

```
# Operaciones elemento a elemento (mucho más rápidas que loops)
arr = np.array([1, 2, 3, 4, 5])

# Operaciones aritméticas
arr_multiplicado = arr * 2     # [2, 4, 6, 8, 10]
arr_cuadrado = arr ** 2        # [1, 4, 9, 16, 25]

# Funciones matemáticas
arr_raiz = np.sqrt(arr)
arr_exp = np.exp(arr)
arr_log = np.log(arr)
```

Por qué vectorización: Evitar bucles explícitos mejora drásticamente el rendimiento.

2.4 Estadísticas con NumPy

```
datos = np.array([15, 23, 18, 30, 25, 19, 22])

# Estadísticas descriptivas
media = np.mean(datos)
mediana = np.median(datos)
std = np.std(datos)
varianza = np.var(datos)
minimo = np.min(datos)
maximo = np.max(datos)
suma = np.sum(datos)
```

3. Pandas: Manipulación de Datos

3.1 Lectura de Datos

Patrón más común en exámenes:

```python
import pandas as pd

# Leer CSV
df = pd.read_csv('archivo.csv')

# Opciones útiles
df = pd.read_csv('archivo.csv',
                 sep=',',           # Delimitador
                 encoding='utf-8',  # Codificación
                 na_values=['NA', 'N/A', ''])  # Valores nulos
```

Exploración inicial:

```python
# Primeras y últimas filas
df.head()      # Primeras 5 filas
df.tail(10)    # Últimas 10 filas

# Información del DataFrame
df.info()      # Tipos de datos y valores nulos
df.describe()  # Estadísticas descriptivas
df.shape       # (filas, columnas)
df.columns     # Nombres de columnas
df.dtypes      # Tipos de datos de cada columna
```

Análisis: Siempre comienza con exploración. `info()` te dice si hay valores nulos y el tipo de cada columna.

3.2 Selección y Filtrado

```
# Seleccionar columnas
df['columna']                    # Una columna (Serie)
df[['col1', 'col2']]            # Múltiples columnas (DataFrame)

# Seleccionar filas y columnas
df.loc[0:5, 'columna']          # Por etiquetas
df.iloc[0:5, 0:3]               # Por índices numéricos

# Filtrado con condiciones (MUY COMÚN EN EXÁMENES)
df_filtrado = df[df['edad'] > 25]
df_filtrado = df[(df['edad'] > 25) & (df['ciudad'] == 'Madrid')]
df_filtrado = df[df['nombre'].isin(['Juan', 'María', 'Pedro'])]
```

Por qué importante: El filtrado condicional es la base del análisis de datos. Nota el uso de `&` (and) y `|` (or) en lugar de `and` / `or` .

3.3 Operaciones de Agregación

groupby - Patrón fundamental:

```
# Agrupar y agregar
resultado = df.groupby('categoria')['ventas'].mean()

# Múltiples agregaciones
resultado = df.groupby('categoria').agg({
    'ventas': ['sum', 'mean', 'count'],
    'precio': ['min', 'max']
})

# Agrupar por múltiples columnas
resultado = df.groupby(['categoria', 'año'])['ventas'].sum()
```

Análisis: `groupby` es como el "GROUP BY" de SQL. Es esencial para análisis agregados (ej: "ventas por categoría y año").

Ejemplo práctico del Examen 2023:

```python
# Contar quejas por compañía
top_companies = df.groupby('company').size().sort_values(ascending=False).head(5)
```

3.4 Transformación de Datos

```python
# Aplicar funciones
df['columna_nueva'] = df['columna_vieja'].apply(lambda x: x * 2)

# Map - reemplazar valores
df['categoria_num'] = df['categoria'].map({'A': 1, 'B': 2, 'C': 3})

# Replace - reemplazar valores específicos
df['columna'] = df['columna'].replace('valor_viejo', 'valor_nuevo')

# Cambiar tipos de datos (MUY COMÚN)
df['columna'] = df['columna'].astype(float)
df['fecha'] = pd.to_datetime(df['fecha'])
```

Ejemplo del Examen 2022:

```python
# Convertir texto con '$' a número
df['Total_Gross'] = df['Total_Gross'].str.replace('$', '').astype(float)
```

3.5 Combinación de DataFrames

```python
# Merge (similar a JOIN de SQL)
df_merged = pd.merge(df1, df2, on='id', how='inner')
# how puede ser: 'inner', 'outer', 'left', 'right'

# Concatenar DataFrames
df_concat = pd.concat([df1, df2], axis=0)  # Verticalmente
df_concat = pd.concat([df1, df2], axis=1)  # Horizontalmente
```

3.6 Valores Únicos y Conteos

```
# Valores únicos
unicos = df['columna'].unique()
num_unicos = df['columna'].nunique()

# Conteo de valores (MUY USADO EN EXÁMENES)
conteos = df['columna'].value_counts()
conteos_porcentaje = df['columna'].value_counts(normalize=True)
```

4. Limpieza y Preparación de Datos

4.1 Manejo de Valores Nulos

Identificación:

```
# Contar valores nulos
df.isnull().sum()
df.isna().sum()  # Equivalente

# Porcentaje de valores nulos
percent_missing = df.isnull().mean() * 100
print(percent_missing)
```

Tratamiento:

```
# Eliminar filas con valores nulos
df_clean = df.dropna()                    # Cualquier nulo
df_clean = df.dropna(subset=['col1'])     # Nulos en columna específica

# Eliminar columnas con valores nulos
df_clean = df.dropna(axis=1)

# Rellenar valores nulos
df_filled = df.fillna(0)                  # Con cero
df_filled = df.fillna(df.mean())          # Con la media
df_filled = df.fillna(method='ffill')     # Forward fill
df_filled = df.fillna(method='bfill')     # Backward fill
```

Patrón del Examen 2024:

```
# Eliminar columnas con más del 25% de valores nulos
percent_missing = df.isnull().mean() * 100
selected_columns = percent_missing < 25
df_clean = df.loc[:, selected_columns]
df_clean = df_clean.dropna()
```

Análisis: Esta es una estrategia común: eliminar columnas con muchos nulos, luego eliminar filas con pocos nulos restantes.

4.2 Detección y Manejo de Outliers

```
# Método IQR (Interquartile Range)
Q1 = df['columna'].quantile(0.25)
Q3 = df['columna'].quantile(0.75)
IQR = Q3 - Q1

# Definir límites
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

# Filtrar outliers
df_sin_outliers = df[(df['columna'] >= limite_inferior) &
                     (df['columna'] <= limite_superior)]
```

4.3 Codificación de Variables Categóricas

```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# One-Hot Encoding (crear columnas dummy)
df_encoded = pd.get_dummies(df, columns=['categoria'], drop_first=True)

# Label Encoding (convertir a números)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['categoria_encoded'] = le.fit_transform(df['categoria'])
```

Por qué es importante: Los modelos de machine learning requieren datos numéricos. One-hot encoding es preferible para variables nominales, label encoding para ordinales.

4.4 Normalización y Estandarización

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Estandarización (media=0, std=1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalización (rango 0-1)
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
```

Análisis: La estandarización es esencial para algoritmos sensibles a escalas (SVM, KNN, regresión regularizada). Usa `fit_transform` en train, `transform` en test.

5. Visualización de Datos

5.1 Matplotlib - Gráficos Básicos

Configuración inicial:

```
import matplotlib.pyplot as plt

# Configurar tamaño de figura
plt.figure(figsize=(10, 6))

# Gráfico de línea
plt.plot(x, y, label='Datos', color='blue', linewidth=2)
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Título del Gráfico')
plt.legend()
plt.grid(True)
plt.show()
```

Gráfico de barras:

```
# Vertical
plt.bar(categorias, valores)

# Horizontal
plt.barh(categorias, valores)

# Personalización
plt.bar(categorias, valores, color='skyblue', edgecolor='black', alpha=0.7)
```

Scatter plot:

```
plt.scatter(x, y, c=colores, s=tamaños, alpha=0.5)
plt.colorbar()  # Si usas colores basados en valores
```

Histograma:

```
plt.hist(datos, bins=20, edgecolor='black', alpha=0.7)
plt.xlabel('Valor')
plt.ylabel('Frecuencia')
```

5.2 Seaborn - Visualizaciones Estadísticas

Por qué Seaborn: Construido sobre Matplotlib, ofrece visualizaciones estadísticas más sofisticadas con menos código.

```
import seaborn as sns

# Configurar estilo
sns.set_style('whitegrid')
sns.set_palette('husl')
```

Barplot con estadísticas:

```
# Automáticamente muestra media con intervalo de confianza
sns.barplot(x='categoria', y='valor', data=df)

# Personalizar
sns.barplot(x='categoria', y='valor', data=df,
            ci=95,           # Intervalo de confianza
            estimator=np.median)  # Usar mediana en lugar de media
```

Patrón del Examen 2023:

```
# Visualizar quejas por compañía y tipo de envío
plt.figure(figsize=(12, 6))
sns.barplot(x='company', y='count', hue='submitted_via', data=df_plot)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Boxplot:

```
sns.boxplot(x='categoria', y='valor', data=df)
# Útil para detectar outliers y comparar distribuciones
```

Heatmap - Matriz de correlación:

```
# Calcular correlación
correlation_matrix = df.corr()

# Visualizar
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Matriz de Correlación')
plt.show()
```

Pairplot - Visualización multivariada:

```
sns.pairplot(df, hue='categoria')
# Muestra relaciones entre todas las variables numéricas
```

Violinplot:

```
sns.violinplot(x='categoria', y='valor', data=df)
# Combina boxplot con estimación de densidad
```

5.3 Mejores Prácticas de Visualización

```
# 1. Siempre etiquetar ejes y añadir título
plt.xlabel('Descripción del eje X', fontsize=12)
plt.ylabel('Descripción del eje Y', fontsize=12)
plt.title('Título Descriptivo', fontsize=14, fontweight='bold')

# 2. Ajustar tamaño de figura según contenido
plt.figure(figsize=(12, 6))  # Para gráficos anchos

# 3. Rotar etiquetas si son largas
plt.xticks(rotation=45, ha='right')

# 4. Usar tight_layout para evitar solapamientos
plt.tight_layout()

# 5. Guardar figuras con alta resolución
plt.savefig('grafico.png', dpi=300, bbox_inches='tight')
```

6. Estadística

6.1 Estadística Descriptiva

```
import numpy as np
import scipy.stats as stats

# Medidas de tendencia central
media = np.mean(datos)
mediana = np.median(datos)
moda = stats.mode(datos)

# Medidas de dispersión
std = np.std(datos)           # Desviación estándar
varianza = np.var(datos)      # Varianza
rango = np.max(datos) - np.min(datos)

# Cuartiles
Q1 = np.percentile(datos, 25)
Q2 = np.percentile(datos, 50)  # Mediana
Q3 = np.percentile(datos, 75)
```

6.2 Correlación

```
from scipy.stats import pearsonr, spearmanr

# Correlación de Pearson (lineal)
corr, p_value = pearsonr(x, y)
print(f"Correlación: {corr:.3f}, p-value: {p_value:.4f}")

# Correlación de Spearman (monotónica, no necesariamente lineal)
corr, p_value = spearmanr(x, y)

# Matriz de correlación con pandas
correlation_matrix = df.corr()
```

Interpretación:

- Correlación cercana a 1: relación positiva fuerte
- Correlación cercana a -1: relación negativa fuerte

- Correlación cercana a 0: no hay relación lineal
- p-value < 0.05: correlación estadísticamente significativa

6.3 Tests Estadísticos

Patrón del Examen 2022 - Test de Wilcoxon:

```
from scipy.stats import ranksums

# Comparar dos grupos
stat, p_value = ranksums(grupo1, grupo2)

if p_value < 0.05:
    print("Hay diferencias significativas entre los grupos")
else:
    print("No hay diferencias significativas")
```

Test de normalidad:

```
from scipy.stats import shapiro

stat, p_value = shapiro(datos)
if p_value > 0.05:
    print("Los datos siguen una distribución normal")
```

7. Machine Learning con Scikit-learn

7.1 Workflow General de Machine Learning

```
# 1. Importar bibliotecas
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# 2. Preparar datos (X: features, y: target)
X = df[['feature1', 'feature2', 'feature3']]
y = df['target']

# 3. Dividir en train y test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 4. Normalizar/Estandarizar (si es necesario)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Solo transform!

# 5. Entrenar modelo
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# 6. Predecir
y_pred = model.predict(X_test_scaled)

# 7. Evaluar
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse:.2f}, R²: {r2:.3f}")
```

Análisis crítico: Nota que usamos `fit_transform` solo en train y `transform` en test. Esto evita data leakage.

7.2 Regresión Lineal

Simple:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)

# Coeficientes
print(f"Intercepto: {model.intercept_}")
print(f"Coeficientes: {model.coef_}")

# Predicción
y_pred = model.predict(X_test)
```

Con validación cruzada:

```
from sklearn.model_selection import cross_val_score

# Evaluar modelo con 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='r2')
print(f"R² medio: {scores.mean():.3f} (+/- {scores.std():.3f})")
```

7.3 Regresión con Regularización

Ridge (L2):

```
from sklearn.linear_model import Ridge

model = Ridge(alpha=1.0)  # alpha controla la regularización
model.fit(X_train, y_train)
```

Elastic Net (L1 + L2):

```python
from sklearn.linear_model import ElasticNetCV

# CV selecciona automáticamente el mejor alpha
model = ElasticNetCV(cv=5, random_state=42)
model.fit(X_train, y_train)
print(f"Mejor alpha: {model.alpha_}")
```

Patrón del Examen 2024:

```python
from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import explained_variance_score

# Entrenar con validación cruzada
model = ElasticNetCV(cv=5, random_state=0)
model.fit(X_train, y_train)

# Predecir y evaluar
y_pred = model.predict(X_test)
score = explained_variance_score(y_test, y_pred)
print(f"Explained Variance: {score:.3f}")
```

Por qué regularización: Previene overfitting penalizando coeficientes grandes. Ridge mantiene todas las features, Lasso puede eliminar algunas (selección de features).

7.4 Regresión Logística (Clasificación)

```python
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.metrics import accuracy_score, classification_report

# Con validación cruzada para encontrar mejor C
model = LogisticRegressionCV(cv=5, random_state=42)
model.fit(X_train, y_train)

# Predecir
y_pred = model.predict(X_test)

# Evaluar
accuracy = accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

Probabilidades de predicción:

```
# Obtener probabilidades en lugar de clases
probs = model.predict_proba(X_test)
```

Patrón del Examen 2023:

```
from sklearn.metrics import precision_recall_curve, auc

# Calcular curva precision-recall
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)
auc_score = auc(recall, precision)
print(f"AUC: {auc_score:.3f}")
```

7.5 K-Means Clustering

Uso básico:

```
from sklearn.cluster import KMeans

# Entrenar modelo
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# Añadir clusters al DataFrame
df['cluster'] = clusters

# Centros de clusters
centers = kmeans.cluster_centers_
```

Encontrar número óptimo de clusters - Método del codo:

```python
inertias = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Visualizar
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Número de Clusters')
plt.ylabel('Inercia')
plt.title('Método del Codo')
plt.show()
```

Silhouette score:

```python
from sklearn.metrics import silhouette_score

score = silhouette_score(X, clusters)
print(f"Silhouette Score: {score:.3f}")
# Valores cercanos a 1 indican clusters bien definidos
```

Patrón del Examen 2024:

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# 1. Aplicar K-means
kmeans = KMeans(n_clusters=3, random_state=0)
clusters = kmeans.fit_predict(X_scaled)

# 2. Reducir dimensionalidad para visualizar
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 3. Visualizar
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=200, label='Centros')
plt.legend()
plt.show()
```

7.6 Clustering Jerárquico

```
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Crear dendrograma
Z = linkage(X, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title('Dendrograma')
plt.show()

# Clustering jerárquico
hierarchical = AgglomerativeClustering(n_clusters=3)
clusters = hierarchical.fit_predict(X)
```

7.7 PCA (Análisis de Componentes Principales)

```
from sklearn.decomposition import PCA

# Reducir a 2 componentes
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Varianza explicada
print(f"Varianza explicada: {pca.explained_variance_ratio_}")
print(f"Varianza total: {pca.explained_variance_ratio_.sum():.3f}")

# Visualizar
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.2%})')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.2%})')
plt.show()
```

Por qué PCA: Reduce dimensionalidad preservando máxima varianza. Útil para visualización y cuando hay muchas features correlacionadas.

7.8 Métricas de Evaluación

Regresión:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from statsmodels.tools.eval_measures import rmse

# Errores
mse = mean_squared_error(y_test, y_pred)
rmse_score = rmse(y_test, y_pred)  # o np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)

# R² score (coeficiente de determinación)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse_score:.2f}")
print(f"MAE: {mae:.2f}")
print(f"R²: {r2:.3f}")
```

Interpretación R^2:

- $R^2 = 1$: predicción perfecta
- $R^2 = 0$: modelo no mejor que predecir la media
- $R^2 < 0$: modelo peor que predecir la media

Clasificación:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report

# Métricas básicas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(cm)

# Reporte completo
print(classification_report(y_test, y_pred))
```

8. Patrones Comunes en Exámenes

8.1 Estructura Típica de un Examen

Patrón observado en los 3 exámenes:

1. **Carga y exploración de datos** (1 punto)
   - Leer CSV
   - Explorar estructura (info, describe, shape)
   - Identificar tipos de datos
2. **Limpieza y preparación** (1-2 puntos)
   - Manejar valores nulos
   - Convertir tipos de datos

- Filtrar datos relevantes
- Seleccionar columnas de interés

3. **Análisis exploratorio y visualización** (2-3 puntos)
    - Agrupar datos (groupby)
    - Crear visualizaciones (barplot, scatter, etc.)
    - Identificar patrones
4. **Análisis estadístico o ML** (3-4 puntos)
    - Construir modelo predictivo
    - Aplicar clustering
    - Realizar test estadístico
    - Evaluar resultados

8.2 Ejercicio Tipo: Análisis de Datos

Plantilla general:

```python
# 1. CONFIGURACIÓN INICIAL
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Configuración de visualización
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

# 2. CARGA DE DATOS
df = pd.read_csv('datos.csv')

# 3. EXPLORACIÓN INICIAL
print("Shape:", df.shape)
print("\nInfo:")
print(df.info())
print("\nPrimeras filas:")
print(df.head())
print("\nEstadísticas:")
print(df.describe())
print("\nValores nulos:")
print(df.isnull().sum())

# 4. LIMPIEZA
# Seleccionar columnas relevantes
columnas_analisis = ['col1', 'col2', 'col3', 'target']
df_clean = df[columnas_analisis].copy()

# Manejar nulos
df_clean = df_clean.dropna()

# Convertir tipos si es necesario
df_clean['columna'] = df_clean['columna'].astype(float)

# 5. ANÁLISIS EXPLORATORIO
# Agrupar y agregar
resultado = df_clean.groupby('categoria')['valor'].agg(['mean', 'count', 'std'])
print(resultado)

# Visualizar
```

```
plt.figure(figsize=(12, 6))
sns.barplot(x='categoria', y='valor', data=df_clean)
plt.xticks(rotation=45)
plt.title('Análisis por Categoría')
plt.tight_layout()
plt.show()

# 6. PREPARACIÓN PARA ML
X = df_clean[['feature1', 'feature2', 'feature3']]
y = df_clean['target']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Escalar
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 7. MODELO
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# 8. EVALUACIÓN
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.3f}")

# Visualizar predicciones
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Predicciones vs Valores Reales')
plt.show()
```

8.3 Ejercicio Tipo: Clustering

Plantilla basada en Examen 2024:

```python
# 1. Preparar datos (solo features, no target)
X = df[['feature1', 'feature2', 'feature3']].copy()

# 2. Estandarizar
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Determinar número óptimo de clusters
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertias = []
silhouette_scores = []
K = range(2, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(X_scaled)
    inertias.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, clusters))

# Visualizar método del codo
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(K, inertias, 'bo-')
ax1.set_xlabel('Número de Clusters')
ax1.set_ylabel('Inercia')
ax1.set_title('Método del Codo')

ax2.plot(K, silhouette_scores, 'ro-')
ax2.set_xlabel('Número de Clusters')
ax2.set_ylabel('Silhouette Score')
ax2.set_title('Silhouette Score por K')

plt.tight_layout()
plt.show()

# 4. Aplicar K-means con k óptimo
k_optimal = 3
kmeans = KMeans(n_clusters=k_optimal, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)

# 5. Analizar clusters
```

```
print("Distribución de clusters:")
print(df['cluster'].value_counts().sort_index())

print("\nCaracterísticas por cluster:")
print(df.groupby('cluster')[['feature1', 'feature2', 'feature3']].mean())

# 6. Visualizar con PCA
from sklearn.decomposition import PCA

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

plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
                      c=df['cluster'], cmap='viridis', alpha=0.6)
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%})')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%})')
plt.title('Clusters en Espacio PCA')
plt.colorbar(scatter, label='Cluster')
plt.show()
```

8.4 Errores Comunes y Cómo Evitarlos

1. Data Leakage:

```
# ❌ INCORRECTO - fit_transform en todo el dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test = train_test_split(X_scaled, ...)

# ✅ CORRECTO - fit solo en train
X_train, X_test = train_test_split(X, ...)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2. No manejar valores nulos antes de ML:

```
# ✅ Siempre verificar y manejar nulos
print(df.isnull().sum())
df = df.dropna()  # o fillna()
```

3. Olvidar estandarizar para algoritmos sensibles:

```
# Algoritmos que REQUIEREN estandarización:
# - Regresión con regularización (Ridge, Lasso, ElasticNet)
# - SVM
# - KNN
# - K-means
# - PCA
# - Regresión logística (recomendado)

# No la necesitan:
# - Árboles de decisión
# - Random Forest
# - Gradient Boosting
```

4. Usar variables categóricas sin codificar:

```
# ✅ Codificar antes de usar en modelo
df_encoded = pd.get_dummies(df, columns=['categoria'], drop_first=True)
```

5. No hacer split aleatorio reproducible:

```
# ✅ Usar random_state para reproducibilidad
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

9. Resumen General

9.1 Workflow Completo de Data Science

```
1. ENTENDER EL PROBLEMA
   ↓
2. CARGAR Y EXPLORAR DATOS
   - read_csv()
   - info(), describe(), head()
   ↓
3. LIMPIAR DATOS
   - Valores nulos (dropna/fillna)
   - Tipos de datos (astype)
   - Outliers
   ↓
4. ANÁLISIS EXPLORATORIO
   - groupby, value_counts
   - Visualizaciones (matplotlib/seaborn)
   - Correlaciones
   ↓
5. PREPARAR PARA MODELADO
   - Seleccionar features
   - Train/test split
   - Codificar categóricas
   - Estandarizar
   ↓
6. ENTRENAR MODELO
   - Elegir algoritmo apropiado
   - fit() con datos de entrenamiento
   ↓
7. EVALUAR
   - Métricas apropiadas
   - Visualizar resultados
   - Validación cruzada
   ↓
8. INTERPRETAR Y COMUNICAR
   - Conclusiones
   - Visualizaciones finales
   - Recomendaciones
```

9.2 Principales Técnicas por Tipo de Problema

Regresión (predecir valor continuo):

- LinearRegression
- Ridge/Lasso/ElasticNet (con regularización)
- Métricas: MSE, RMSE, MAE, R^2

Clasificación (predecir categoría):

- LogisticRegression
- Métricas: Accuracy, Precision, Recall, F1, AUC

Clustering (agrupar sin etiquetas):

- K-Means
- Hierarchical Clustering
- Métricas: Silhouette Score, Inercia

Reducción de Dimensionalidad:

- PCA
- Para visualización o pre-procesamiento

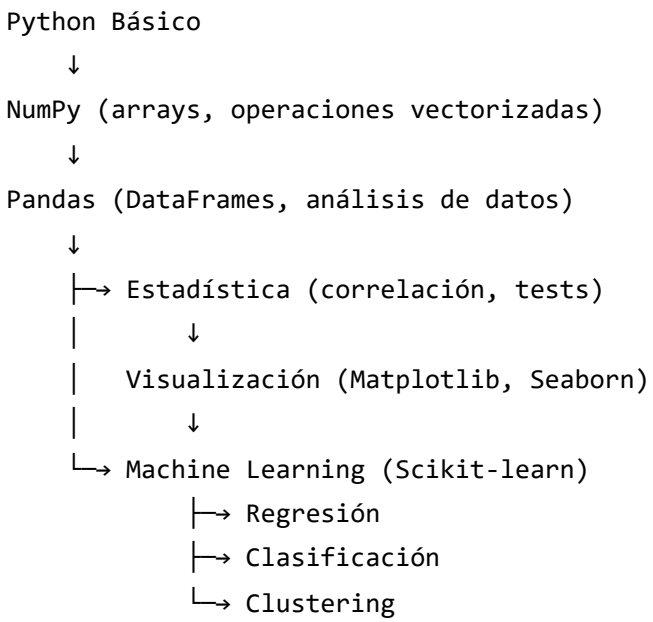
9.3 Librerías Esenciales

```
# Manipulación de datos
import pandas as pd
import numpy as np

# Visualización
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression, Ridge, LogisticRegression
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

# Estadística
from scipy import stats
```

9.4 Consejos Finales para Exámenes

1. **Lee cuidadosamente** qué pide cada ejercicio
2. **Explora los datos primero** - usa info(), describe(), head()
3. **Verifica valores nulos** antes de continuar
4. **Divide train/test ANTES** de cualquier preprocessing
5. **Estandariza** cuando uses regularización o clustering
6. **Visualiza** para validar tus resultados
7. **Comenta tu código** brevemente para explicar tu razonamiento
8. **Verifica que tus gráficos** tengan etiquetas y títulos
9. **Usa random_state** para reproducibilidad
10. **Gestiona el tiempo** - no te quedes atascado en un ejercicio

9.5 Relación entre Temas



Todos los temas se conectan: NumPy proporciona arrays eficientes, Pandas los usa para DataFrames, la visualización y ML trabajan sobre estos datos estructurados.

📝 Notas Aclaratorias

Diferencias entre Ejercicios y Exámenes

Ejercicios de clase:

- Más guiados, con explicaciones paso a paso
- Enfoque didáctico en entender conceptos
- Código más extenso y comentado
- Ejemplos más simples

Exámenes:

- Requieren aplicar múltiples conceptos
- Menos guía, más autonomía
- Problemas realistas con datasets reales
- Tiempo limitado - necesitas ser eficiente

Soluciones Alternativas

En varios ejercicios se observan diferentes enfoques:

Para agrupar y contar:

```
# Opción 1: groupby + size
df.groupby('columna').size()

# Opción 2: value_counts (más directo)
df['columna'].value_counts()

# Opción 3: groupby + count (cuenta por cada columna)
df.groupby('columna').count()
```

Para filtrar top N:

```
# Opción 1: sort + head
df.sort_values('valor', ascending=False).head(5)

# Opción 2: nlargest
df.nlargest(5, 'valor')
```

Ambos enfoques son válidos. Elige el que te resulte más natural y legible.

Recursos Adicionales

Los archivos de chuletas existentes en la carpeta `chuleta/` son complementarios:

- `py_chu_numpy.ipynb` - Referencia rápida de NumPy
- `py_chu_pandas.ipynb` - Referencia rápida de Pandas
- `py_chu_matplotlib.ipynb` - Referencia rápida de Matplotlib
- `py_chu_seaborn.ipynb` - Referencia rápida de Seaborn
- `py_chu_stats.ipynb` - Referencia rápida de estadística
- `py_chu_skl.ipynb` - Referencia rápida de Scikit-learn

Última actualización: Este documento se basa en el análisis de todos los ejercicios de `clase_jf/`, `clase_io/` y los exámenes de 2022, 2023 y 2024 del repositorio Python_examen_Tecnun.

Autor: Generado automáticamente mediante análisis completo del repositorio.

Uso: Este documento es una guía de estudio. Practica con los notebooks originales para consolidar los conceptos.