[1]:
```
# ===== Güçlü (Robust) Tek CSV Veri Yükleyici ve Yardımcı Fonksiyonlar =====
import os, re
import pandas as pd
import numpy as np

# Şu anda data/ klasöründe bulunan tek CSV dosyasının adı:
CSV_FILE = "all_upwork_jobs_2024-02-07-2024-03-24.csv"  # Eğer dosya adını değiştirirsen bu satırı da güncelle.
```

[2]:
```
def find_data_dir(start_dir="."):
    """Verilen klasörden yukarıya doğru çıkarak 'data' klasörünü bulur."""
    start = os.path.abspath(start_dir)
    cur = start
    for _ in range(5):
        cand = os.path.join(cur, "data")
        if os.path.isdir(cand):
            return cand
        cur = os.path.dirname(cur)
    # Ek tahminler: bazen çalışma dizini karışabiliyor
    guesses = [os.path.join(start, "data"), os.path.join(os.getcwd(), "data")]
    for g in guesses:
        if os.path.isdir(g):
            return g
    # Hiçbir şekilde bulunamadıysa hata fırlat
    raise FileNotFoundError("‘data’ klasörü bulunamadı. Lütfen bir ‘data’ klasörü oluşturup içine CSV dosyasını koy.")
```

[3]:
```
# data klasörünün yolunu bul
DATA_DIR = find_data_dir(".")
csv_path = os.path.join(DATA_DIR, CSV_FILE)
print("DATA_DIR =", DATA_DIR)
print("CSV_PATH =", csv_path)
```

```
DATA_DIR = C:\Users\karm1\OneDrive\Desktop\Upwork_Project\data
CSV_PATH = C:\Users\karm1\OneDrive\Desktop\Upwork_Project\data\all_upwork_jobs_2024-02-07-2024-03-24.csv
```

[4]:
```python
def safe_read_csv(path):
    """CSV dosyasını güvenli şekilde okur; karakter kodlaması (encoding) veya bozuk satır hatalarına karşı dayanıklıdır."""
    try:
        return pd.read_csv(path, low_memory=False)
    except UnicodeDecodeError:
        try:
            return pd.read_csv(path, encoding="latin1", low_memory=False)
        except Exception:
            return pd.read_csv(path, encoding_errors="ignore", engine="python", on_bad_lines="skip", low_memory=False)
    except Exception:
        return pd.read_csv(path, engine="python", on_bad_lines="skip", low_memory=False)
```

[5]:
```python
# Dosya mevcut mu kontrol et
if not os.path.isfile(csv_path):
    raise FileNotFoundError(f"Dosya bulunamadı: {csv_path}")

# ℹ️ Sadece tek dosya okunur (birleştirme yok)
df = safe_read_csv(csv_path)
df["__source_file"] = os.path.basename(csv_path)
print("Loaded one file shape:", df.shape)
```

```
Loaded one file shape: (244828, 9)
```

[6]:
```
def normcols(d):
    d2 = d.copy()

    # "budget" (bütçe) yerine kullanılabilecek alternatif sütun isimleri
    for c in list(d2.columns):
        cl = c.lower()
        if cl in ["fixed_price", "price", "amount"] and "budget" not in d2.columns:
            d2 = d2.rename(columns={c: "budget"})

    # Saatlik ücret için alternatif isim kontrolü
    cols_lower = {c.lower(): c for c in d2.columns}
    if "hourly_rate" not in d2.columns:
        if "hourly_low" in cols_lower and "hourly_high" in cols_lower:
            low_col  = cols_lower["hourly_low"]
            high_col = cols_lower["hourly_high"]
            hr = (pd.to_numeric(d2[low_col], errors="coerce") + pd.to_numeric(d2[high_col], errors="coerce"))/2
            d2["hourly_rate"] = hr
        else:
            hmins = [c for c in d2.columns if "hourly" in c.lower() and "min" in c.lower()]
            hmaxs = [c for c in d2.columns if "hourly" in c.lower() and "max" in c.lower()]
            if hmins and hmaxs:
                hr = (pd.to_numeric(d2[hmins[0]], errors="coerce") + pd.to_numeric(d2[hmaxs[0]], errors="coerce"))/2
                d2["hourly_rate"] = hr

    # "payment_verified" sütunu
    cand_pay = [c for c in d2.columns if "payment" in c.lower() and "verif" in c.lower()]
    if cand_pay and "payment_verified" not in d2.columns:
        d2 = d2.rename(columns={cand_pay[0]: "payment_verified"})

    # Açıklama ve beceri sütunları
    for c in list(d2.columns):
        cl = c.lower()
        if cl == "job_description" and "description" not in d2.columns:
            d2 = d2.rename(columns={c: "description"})
        if cl in ["skills_str", "job_skills"] and "skills" not in d2.columns:
            d2 = d2.rename(columns={c: "skills"})

    return d2
```

[7]:
```
#Normalize edilmiş DataFrame
df = normcols(df)
```

[8]:
```
#Güvenli strip fonksiyonu
def safe_strip(x):
    """Metinlerden baştaki ve sondaki boşlukları güvenli şekilde temizler."""
    try:
        return str(x).strip()
    except:
        return x
```

[9]:
```
#Metin sütunlarını temizle
def safe_strip(x):
    """Metinlerden baştaki ve sondaki boşlukları güvenli şekilde temizler."""
    try:
        return str(x).strip()
    except:
        return x
```

[10]:
```
#Tarih formatı tespiti
for c in df.select_dtypes(include=["object"]).columns:
    df[c] = df[c].apply(safe_strip)
```

[15]:
```
#Tarih formatı tespitiS
for c in df.columns:
    if any(k in c.lower() for k in ["time","date","posted","created","updated","published"]):
        try:
            df[c] = pd.to_datetime(df[c], errors="coerce")
        except:
            pass
```

[16]:
```python
#Sayısal uç değer temizliği
for c in df.columns:
    cl = c.lower()
    if cl in ["budget","hourly_rate","hourly_low","hourly_high"]:
        df[c] = pd.to_numeric(df[c], errors="coerce")
        if df[c].notna().any():
            q99 = df[c].quantile(0.99)
            df.loc[df[c] < 0, c] = np.nan
            df.loc[df[c] > q99*5, c] = np.nan
```

[22]:
```python
#Sonuç bilgileri
print("Columns available:", list(df.columns))
print("Final shape after light cleaning:", df.shape)
```

```
Columns available: ['title', 'link', 'published_date', 'is_hourly', 'hourly_low', 'hourly_high', 'budget', 'country', '__source_file', 'hourly_rate']
Final shape after light cleaning: (244828, 10)
```