[1]:
```
# ============================================================
# 4) Hiperparametre Ayarlama (Hyperparameter Tuning)
# Upwork Jobs (Tek CSV, Hafif Versiyon)
# Çalışan sütunlar: ['title','link','published_date','is_hourly','hourly_low','hourly_high','budget','country']
# ============================================================

import os, numpy as np, pandas as pd, tempfile
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import (
    accuracy_score, f1_score, classification_report,
    r2_score, mean_absolute_error, root_mean_squared_error
)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
```

[2]:
```
# --------------------------
# Tek CSV dosyasını yükleme (birleştirme yok)
# --------------------------
DATA_DIR = r"C:\Users\user\OneDrive\Desktop\Upwork_Project\data"
CSV_FILE = "all_upwork_jobs_2024-02-07-2024-03-24.csv"   # Farklı dosya kullanıyorsan burayı güncelle
csv_path = os.path.join(DATA_DIR, CSV_FILE)
if not os.path.isfile(csv_path):
    raise FileNotFoundError(f"CSV not found at: {csv_path}")

# CSV dosyasını oku
df = pd.read_csv(csv_path, low_memory=False)
print("Loaded:", df.shape, list(df.columns))
```

```
Loaded: (244828, 8) ['title', 'link', 'published_date', 'is_hourly', 'hourly_low', 'hourly_high', 'budget', 'country']
```

[3]:
```python
# --------------------------
# 1) Özellik Mühendisliği (Feature Engineering)
# --------------------------

# hourly_low + hourly_high ortalamasından hourly_rate sütunu oluştur
if {"hourly_low","hourly_high"}.issubset(df.columns):
    df["hourly_rate"] = (pd.to_numeric(df["hourly_low"], errors="coerce")
                         + pd.to_numeric(df["hourly_high"], errors="coerce"))/2

# is_hourly sütununu 0 / 1 formatına dönüştür (sabit mi, saatlik mi)
def map_hourly(v):
    s = str(v).strip().lower()
    if s in ["1","true","yes","y","hourly","hourly_job","t"]: return 1
    if s in ["0","false","no","n","fixed","fixed_price","f"]: return 0
    try: return int(v)
    except: return np.nan
if "is_hourly" in df.columns:
    df["is_hourly"] = df["is_hourly"].apply(map_hourly)

# Sayısal sütunları dönüştür
for c in ["budget","hourly_rate"]:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")
```

[4]:
```python
# Eksik değerleri doldur, metin sütunlarını hazırla
df["title"] = df.get("title","").astype(str).fillna("")
df["country"] = df.get("country","unknown").astype(str).fillna("unknown")
```

[5]:
```python
# --------------------------
# 2) Örnekleme (Sample for speed)
# --------------------------
# Büyük veriyle GridSearch çok yavaş olabileceği için örnekleme yapılıyor
SAMPLE_MAX = 40000   # Hızlı test için 40K satır, gerekirse artırılabilir
df_s = df.sample(SAMPLE_MAX, random_state=42) if len(df) > SAMPLE_MAX else df.copy()
print("Working sample:", df_s.shape)
```

```
Working sample: (40000, 9)
```

[6]:
```
# ---------------------------
# 3) Ön İşleme (Preprocessing)
# ---------------------------
# TF-IDF (title) + OneHot (country)
pre_sparse = ColumnTransformer(
    transformers=[
        ("txt", TfidfVectorizer(max_features=4000, ngram_range=(1,2)), "title"),  # Metin özellikleri
        ("cty", OneHotEncoder(handle_unknown="ignore"), ["country"]),             # Ülke özellikleri
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

# Pipeline cache (geçici dizin) — tekrar eden modelleri hızlandırır
CACHE_DIR = tempfile.mkdtemp()
print("Pipeline cache:", CACHE_DIR)
```

```
Pipeline cache: C:\Users\user\AppData\Local\Temp\tmpy1akzx54
```

[7]:
```
# --------------------------------
# A) Sınıflandırma (Classification): is_hourly tahmini
# --------------------------------
if "is_hourly" in df_s.columns and df_s["is_hourly"].dropna().nunique()==2:
    dcls = df_s.dropna(subset=["is_hourly"]).copy()
    Xc = dcls[["title","country"]]   # Girdi özellikleri
    yc = dcls["is_hourly"].astype(int)  # Hedef: 0 veya 1

    # Veriyi eğitim (%80) ve test (%20) olarak ayır
    Xc_train, Xc_test, yc_train, yc_test = train_test_split(
        Xc, yc, test_size=0.2, random_state=42, stratify=yc
    )
    print("\n[A] CLS Train:", Xc_train.shape, "Test:", Xc_test.shape)

    # --- Logistic Regression modeli için hiperparametre ayarı ---
    pipe_log = Pipeline(steps=[
        ("pre", pre_sparse),
        ("clf", LogisticRegression(max_iter=800, solver="liblinear"))
    ], memory=CACHE_DIR)

    # C (regularization strength) için grid araması
    grid_log = GridSearchCV(
        pipe_log,
        param_grid={"clf__C":[0.5, 1.0, 2.0]},   # Küçük grid (hızlı)
        scoring="f1",
        cv=3, n_jobs=-1, verbose=1
    )
    grid_log.fit(Xc_train, yc_train)
    print("Best LOG params:", grid_log.best_params_, " best F1:", grid_log.best_score_)
```

```
[A] CLS Train: (32000, 2) Test: (8000, 2)
```

```
[A] CLS Train: (32000, 2) Test: (8000, 2)
Fitting 3 folds for each of 3 candidates, totalling 9 fits
Best LOG params: {'clf__C': 0.5}  best F1: 0.7227389165193155
```

[8]:
```python
# --- Random Forest modeli için hiperparametre ayarı ---
# RandomForest, sparse veriyle çalışamadığı için TruncatedSVD ile boyut indirgeme yapılır
pipe_rf = Pipeline(steps=[
    ("pre", pre_sparse),
    ("svd", TruncatedSVD(n_components=150, random_state=42)),   # Boyut indirgeme (daha hafif)
    ("clf", RandomForestClassifier(random_state=42, n_jobs=-1))
], memory=CACHE_DIR)

param_rf = {
    "clf__n_estimators":[200, 300],  # Ağaç sayısı
    "clf__max_depth":[None, 25]      # Maksimum derinlik
}
grid_rf = GridSearchCV(
    pipe_rf, param_grid=param_rf, scoring="f1", cv=3, n_jobs=-1, verbose=1
)
grid_rf.fit(Xc_train, yc_train)
print("Best RF params:", grid_rf.best_params_, " best F1:", grid_rf.best_score_)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best RF params: {'clf__max_depth': None, 'clf__n_estimators': 300}  best F1: 0.7140229564901711
```

[9]:
```python
if "is_hourly" in df_s.columns and df_s["is_hourly"].dropna().nunique()==2:

    # --- En iyi modelleri test kümesinde değerlendir ---
    for name, model in [("LogReg", grid_log.best_estimator_), ("RF", grid_rf.best_estimator_)]:
        pred = model.predict(Xc_test)
        acc = accuracy_score(yc_test, pred)
        f1  = f1_score(yc_test, pred)
        print(f"\n[name={name}] ACC={acc:.4f} F1={f1:.4f}")
        print(classification_report(yc_test, pred, digits=3))

else:
    print("\n[A] Classification skipped: 'is_hourly' not binary/available.")
```

```
[name=LogReg] ACC=0.6498 F1=0.7226
              precision    recall  f1-score   support

           0      0.616     0.458     0.525      3384
           1      0.665     0.791     0.723      4616

    accuracy                         0.650      8000
   macro avg      0.641     0.624     0.624      8000
weighted avg      0.644     0.650     0.639      8000


[name=RF] ACC=0.6440 F1=0.7201
              precision    recall  f1-score   support
```

```
[name=LogReg] ACC=0.6498 F1=0.7226
              precision    recall  f1-score   support

           0      0.616     0.458     0.525      3384
           1      0.665     0.791     0.723      4616

    accuracy                         0.650      8000
   macro avg      0.641     0.624     0.624      8000
weighted avg      0.644     0.650     0.639      8000


[name=RF] ACC=0.6440 F1=0.7201
              precision    recall  f1-score   support

           0      0.610     0.440     0.511      3384
           1      0.659     0.794     0.720      4616

    accuracy                         0.644      8000
   macro avg      0.634     0.617     0.616      8000
weighted avg      0.638     0.644     0.632      8000
```

[10]:
```python
# ------------------------------------------------
# B) Regresyon (Fixed Jobs): 'budget' tahmini
# ------------------------------------------------
reg_fixed_done = False
if {"is_hourly","budget"}.issubset(df_s.columns):
    dregF = df_s[(df_s["is_hourly"]==0) & (df_s["budget"].notna())].copy()
    if len(dregF) >= 200:
        XrF = dregF[["title","country"]]
        yrF = dregF["budget"].astype(float)

        XrF_train, XrF_test, yrF_train, yrF_test = train_test_split(
            XrF, yrF, test_size=0.2, random_state=42
        )
        print(f"\n[B] REG Fixed Rows={len(dregF)} Train:{XrF_train.shape} Test:{XrF_test.shape}")

        # Pipeline: TF-IDF + OneHot + SVD + RandomForestRegressor
        pipe_rfr = Pipeline(steps=[
            ("pre", pre_sparse),
            ("svd", TruncatedSVD(n_components=150, random_state=42)),
            ("reg", RandomForestRegressor(random_state=42, n_jobs=-1))
        ], memory=CACHE_DIR)

        # Hiperparametre aralığı (n_estimators ve max_depth)
        grid_rfr = GridSearchCV(
            pipe_rfr,
            param_grid={"reg__n_estimators":[200, 300], "reg__max_depth":[None, 25]},
            scoring="neg_root_mean_squared_error",  # Hedef: RMSE'yi minimize etmek
            cv=3, n_jobs=-1, verbose=1
        )
        grid_rfr.fit(XrF_train, yrF_train)

        # En iyi modelin performansını test setinde ölç
        best_regF = grid_rfr.best_estimator_
        predF = best_regF.predict(XrF_test)
        r2F = r2_score(yrF_test, predF)
        maeF = mean_absolute_error(yrF_test, predF)
        rmseF = root_mean_squared_error(yrF_test, predF)
        print("Best RFReg (Fixed) params:", grid_rfr.best_params_,
              " | Test -> R2={:.4f} MAE={:.2f} RMSE={:.2f}".format(r2F, maeF, rmseF))
        reg_fixed_done = True

if not reg_fixed_done:
    print("\n[B] Regression (Fixed) skipped: need many rows with is_hourly==0 and budget numeric.")
```

```
[B] REG Fixed Rows=16920 Train:(13536, 2) Test:(3384, 2)
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best RFReg (Fixed) params: {'reg__max_depth': None, 'reg__n_estimators': 300}  | Test -> R2=-1.1926 MAE=1597.68 RMSE=13704.81
```

[11]:
```
# ----------------------------------------------------
# C) Regresyon (Hourly Jobs): 'hourly_rate' tahmini
# ----------------------------------------------------
reg_hourly_done = False
if {"is_hourly","hourly_rate"}.issubset(df_s.columns):
    dregH = df_s[(df_s["is_hourly"]==1) & (df_s["hourly_rate"].notna())].copy()
    if len(dregH) >= 200:
        XrH = dregH[["title","country"]]
        yrH = dregH["hourly_rate"].astype(float)

        XrH_train, XrH_test, yrH_train, yrH_test = train_test_split(
            XrH, yrH, test_size=0.2, random_state=42
        )
        print(f"\n[C] REG Hourly Rows={len(dregH)} Train:{XrH_train.shape} Test:{XrH_test.shape}")

        pipe_rfrH = Pipeline(steps=[
            ("pre", pre_sparse),
            ("svd", TruncatedSVD(n_components=150, random_state=42)),
            ("reg", RandomForestRegressor(random_state=42, n_jobs=-1))
        ], memory=CACHE_DIR)

        grid_rfrH = GridSearchCV(
            pipe_rfrH,
            param_grid={"reg__n_estimators":[200, 300], "reg__max_depth":[None, 25]},
            scoring="neg_root_mean_squared_error",
            cv=3, n_jobs=-1, verbose=1
        )
        grid_rfrH.fit(XrH_train, yrH_train)

        best_regH = grid_rfrH.best_estimator_
        predH = best_regH.predict(XrH_test)
        r2H = r2_score(yrH_test, predH)
        maeH = mean_absolute_error(yrH_test, predH)
        rmseH = root_mean_squared_error(yrH_test, predH)
        print("Best RFReg (Hourly) params:", grid_rfrH.best_params_,
              " | Test -> R2={:.4f} MAE={:.2f} RMSE={:.2f}".format(r2H, maeH, rmseH))
        reg_hourly_done = True

if not reg_hourly_done:
    print("\n[C] Regression (Hourly) skipped: need rows with is_hourly==1 and hourly_rate numeric.")
```

```
[C] REG Hourly Rows=16159 Train:(12927, 2) Test:(3232, 2)
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best RFReg (Hourly) params: {'reg__max_depth': 25, 'reg__n_estimators': 300}  | Test -> R2=-0.0278 MAE=16.42 RMSE=30.08
```

[12]:
```
print("\n Hiperparametre ayarlama başarıyla tamamlandı.")
```

```
 Hiperparametre ayarlama başarıyla tamamlandı.
```