```
import pandas as pd

# Load dataset (Replace 'sampled_job_descriptions.csv' with actual file path)
df = pd.read_csv("sampled_job_descriptions.csv")

# Step 1: Display column names
print(" Column Names in Dataset:")
print(df.columns.tolist())

# Step 2: Check for missing values across all columns
missing_values = df.isnull().sum().reset_index()
missing_values.columns = ["Column Name", "Missing Values"]

# Display missing values
print("\n Missing Values in Each Column:")
print(missing_values)

#Step 2: Primary Key & Referential Integrity Checks
null_job_ids = df[df["Job Id"].isnull()]
print("\n Null Job Ids Count:", len(null_job_ids))

# Step 4: Check for Duplicate Job Ids
duplicate_job_ids = df[df["Job Id"].duplicated()]
print(" Duplicate Job Ids Count:", len(duplicate_job_ids))


#Field-Level Integrity Checks
import re

# Step 5: Validate Latitude & Longitude Ranges
invalid_latitude = df[(df["latitude"].notna()) & ((df["latitude"] < -90) | (df["latitude"] > 90))]
invalid_longitude = df[(df["longitude"].notna()) & ((df["longitude"] < -180) | (df["longitude"] > 180))]

print("\n Invalid Latitude Count:", len(invalid_latitude))
print(" Invalid Longitude Count:", len(invalid_longitude))


# Step 6: Validate Job Posting Date format
df["Job Posting Date"] = pd.to_datetime(df["Job Posting Date"], errors='coerce')  # Convert invalid dates to NaT
invalid_dates = df[df["Job Posting Date"].isna()]

print("\n Invalid Job Posting Date Count:", len(invalid_dates))

    # Step 7: Validate Contact Numbers (Minimum 10 Digits)
invalid_contacts = df[~df["Contact"].astype(str).str.match(r'^\d{10,}$', na=False)]

print("\n Invalid Contact Count:", len(invalid_contacts))

# Display first 10 invalid contacts
print("\n Invalid Contacts (First 10):")
print(invalid_contacts["Contact"].head(10).tolist())

# Save invalid records

# Step 8: Validate Work Type Categories
unique_work_types = df["Work Type"].unique()

# Display unique work type values
print("\n Unique Work Types:", unique_work_types)

# Step 9: Validate Location Format (Letters, Spaces, Commas, Hyphens, Periods)
location_pattern = r'^[A-Za-z\s,.-]+$'
invalid_location = df[~df["location"].astype(str).str.match(location_pattern, na=False)]

print("\n Invalid Location Count:", len(invalid_location))
print("\n Invalid Locations (First 10):")
print(invalid_location["location"].head(10).tolist())

salary_pattern = r'^\$\d{2,3}K-\$\d{2,3}K$'
invalid_salary = df[~df["Salary Range"].astype(str).str.match(salary_pattern, na=False)]

# Step 11: Validate Experience Format (X to Y Years)
experience_pattern = r'^\d+ to \d+ Years$'
invalid_experience = df[~df["Experience"].astype(str).str.match(experience_pattern, na=False)]

# Ensure Experience range is valid (X ≤ Y)
def check_experience_range(exp):
    match = re.match(r'(\d+) to (\d+) Years', str(exp))
    if match:
        x, y = int(match.group(1)), int(match.group(2))
        return x <= y  # True if X ≤ Y, False otherwise
    return False

invalid_experience_range = df[~df["Experience"].apply(check_experience_range)]


print("\n Invalid Salary Range Count:", len(invalid_salary))
print(" Invalid Salary Values (First 10):", invalid_salary["Salary Range"].unique())

print("\n Invalid Experience Format Count:", len(invalid_experience))
print(" Invalid Experience Values (First 10):", invalid_experience["Experience"].unique())

print("\n Invalid Experience Range Count (X > Y):", len(invalid_experience_range))
print(" Invalid Experience Range Values (First 10):", invalid_experience_range["Experience"].unique())
```