```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
! mkdir ~/.kaggle
```

```
cp  /content/drive/MyDrive/Kaggle_Api/kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets download arshkon/linkedin-job-postings
```

```
Dataset URL: https://www.kaggle.com/datasets/arshkon/linkedin-job-postings
License(s): CC-BY-SA-4.0
```

```
! unzip linkedin-job-postings.zip
```

```
Archive:  linkedin-job-postings.zip
  inflating: companies/companies.csv
  inflating: companies/company_industries.csv
  inflating: companies/company_specialities.csv
  inflating: companies/employee_counts.csv
  inflating: jobs/benefits.csv
  inflating: jobs/job_industries.csv
  inflating: jobs/job_skills.csv
  inflating: jobs/salaries.csv
  inflating: mappings/industries.csv
  inflating: mappings/skills.csv
  inflating: postings.csv
```

Step1: Load and Explore Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy import stats
```

```
# Load the dataset
df = pd.read_csv('postings.csv')

# Get basic information about the dataset
print(f"Dataset shape: {df.shape}")
print("\nData types:")
print(df.dtypes)
print("\nBasic statistics:")
print(df.describe().T)
print("\nMissing values:")
print(df.isnull().sum())
```

```
closed_time            1073.0  1.712928e+12  3.622893e+08  1.712346e+12
listed_time          123849.0  1.713204e+12  3.989122e+08  1.711317e+12
sponsored            123849.0  0.000000e+00  0.000000e+00  0.000000e+00
normalized_salary     36073.0  2.053270e+05  5.097627e+06  0.000000e+00
zip_code             102977.0  5.040049e+04  3.025223e+04  1.001000e+03
fips                  96434.0  2.871388e+04  1.601593e+04  1.003000e+03

                              25%           50%           75%           max
job_id               3.894587e+09  3.901998e+09  3.904707e+09  3.906267e+09
max_salary           4.828000e+01  8.000000e+04  1.400000e+05  1.200000e+08
company_id           1.435200e+04  2.269650e+05  8.047188e+06  1.034730e+08
views                3.000000e+00  4.000000e+00  8.000000e+00  9.975000e+03
med_salary           1.894000e+01  2.550000e+01  2.510500e+03  7.500000e+05
min_salary           3.700000e+01  6.000000e+04  1.000000e+05  8.500000e+07
applies              1.000000e+00  3.000000e+00  8.000000e+00  9.670000e+02
original_listed_time 1.712863e+12  1.713395e+12  1.713478e+12  1.713573e+12
remote_allowed       1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
expiry               1.715481e+12  1.716042e+12  1.716088e+12  1.729125e+12
closed_time          1.712670e+12  1.712670e+12  1.713283e+12  1.713562e+12
listed_time          1.712886e+12  1.713408e+12  1.713484e+12  1.713573e+12
sponsored            0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
normalized_salary    5.200000e+04  8.150000e+04  1.250000e+05  5.356000e+08
zip_code             2.411200e+04  4.805900e+04  7.820100e+04  9.990100e+04
fips                 1.312100e+04  2.918300e+04  4.207700e+04  5.604500e+04

Missing values:
job_id                            0
company_name                   1719
title                             0
description                       7
max_salary                    94056
pay_period                    87776
location                          0
company_id                     1717
views                          1689
med_salary                   117569
min_salary                    94056
formatted_work_type               0
applies                      100529
original_listed_time              0
remote_allowed               108603
job_posting_url                   0
application_url               36665
application_type                  0
expiry                            0
closed_time                  122776
formatted_experience_level    29409
skills_desc                  121410
listed_time                       0
posting_domain                39968
sponsored                         0
work_type                         0
currency                      87776
compensation_type             87776
normalized_salary             87776
zip_code                      20872
fips                          27415
dtype: int64
```

Step 2: Data Cleaning - Handle Missing Values and Normalize Salary Data

```python
df_clean = df.copy()

# Handle missing company names
df_clean['company_name'].fillna('Unknown Company', inplace=True)

# Normalize salary data - convert hourly wages to yearly
def normalize_salary(row):
    if pd.notna(row['max_salary']):
        if row['pay_period'] == 'HOURLY':
            # Assuming 40 hours per week, 52 weeks per year
            return row['max_salary'] * 40 * 52
        else:
            return row['max_salary']
    return row['normalized_salary']

# If normalized_salary is available, use it; otherwise calculate from max_salary
df_clean['annual_salary'] = df_clean.apply(normalize_salary, axis=1)

print("Normalized salary data:")
print(df_clean[['title', 'max_salary', 'pay_period', 'normalized_salary', 'annual_salary']].head())
```

```
Step 2: Data Cleaning
<ipython-input-12-69d59120ef21>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  df_clean['company_name'].fillna('Unknown Company', inplace=True)
Normalized salary data:
                                              title  max_salary pay_period  \
0                             Marketing Coordinator        20.0     HOURLY
1                 Mental Health Therapist/Counselor        50.0     HOURLY
2                       Assitant Restaurant Manager     65000.0     YEARLY
3  Senior Elder Law / Trusts and Estates Associat...   175000.0     YEARLY
4                                Service Technician     80000.0     YEARLY

   normalized_salary  annual_salary
0            38480.0        41600.0
1            83200.0       104000.0
2            55000.0        65000.0
3           157500.0       175000.0
4            70000.0        80000.0
```

Step 3: Feature Engineering

```
# Extract state from location
df_clean['state'] = df_clean['location'].str.extract(r',\s*(\w{2})$')

# Convert listed_time from epoch to datetime
df_clean['listed_date'] = pd.to_datetime(df_clean['listed_time'], unit='ms')

# Create a binary flag for remote jobs
df_clean['is_remote'] = df_clean['remote_allowed'].apply(lambda x: 1 if x == 1.0 else 0)

# Calculate post age in days (from listing to current date)
current_date = pd.Timestamp('2025-04-10')  # Using the current date from the prompt
df_clean['post_age_days'] = (current_date - df_clean['listed_date']).dt.days

print("Feature engineering results:")
print(df_clean[['location', 'state', 'listed_time', 'listed_date', 'remote_allowed', 'is_remote', 'post_age_days']].head())
```

```
Step 3: Feature Engineering
Feature engineering results:
            location state   listed_time         listed_date  remote_allowed  \
0      Princeton, NJ    NJ  1.713398e+12 2024-04-17 23:45:08             NaN
1   Fort Collins, CO    CO  1.712858e+12 2024-04-11 17:51:27             NaN
2     Cincinnati, OH    OH  1.713278e+12 2024-04-16 14:26:54             NaN
3  New Hyde Park, NY    NY  1.712896e+12 2024-04-12 04:23:32             NaN
4     Burlington, IA    IA  1.713452e+12 2024-04-18 14:52:23             NaN

   is_remote  post_age_days
0          0            357
1          0            363
2          0            358
3          0            362
4          0            356
```

Step 4: Exploratory Data Analysis - Salary Distribution

```
# Basic statistics for annual salary
print("Annual salary statistics:")
print(df_clean['annual_salary'].describe())

# Create a histogram of annual salaries
plt.figure(figsize=(10, 6))
sns.histplot(df_clean['annual_salary'].dropna(), kde=True)
plt.title('Distribution of Annual Salaries')
plt.xlabel('Annual Salary (USD)')
plt.ylabel('Frequency')
plt.savefig('salary_distribution.png')
plt.close()  # Close the figure to avoid displaying in notebook

print("Salary distribution analysis complete. Histogram would show the distribution pattern.")
```

```
Step 4: Exploratory Data Analysis - Salary Distribution
Annual salary statistics:
count    3.607300e+04
mean     2.275688e+05
std      5.540858e+06
min      0.000000e+00
25%      5.453760e+04
50%      9.000000e+04
75%      1.400000e+05
max      5.720000e+08
Name: annual_salary, dtype: float64
Salary distribution analysis complete. Histogram would show the distribution pattern.
```

Step 5: Exploratory Data Analysis - Job Engagement Metrics

```
# Calculate view-to-application ratio (where data is available)
df_clean['view_apply_ratio'] = df_clean['applies'] / df_clean['views']

# Calculate the correlation between salary and views
salary_views_corr = df_clean['annual_salary'].corr(df_clean['views'])
print(f"Correlation between annual salary and views: {salary_views_corr:.2f}")

# Scatter plot of salary vs. views
plt.figure(figsize=(10, 6))
sns.scatterplot(x='annual_salary', y='views', data=df_clean)
plt.title('Job Views vs. Annual Salary')
plt.xlabel('Annual Salary (USD)')
plt.ylabel('Number of Views')
plt.savefig('salary_vs_views.png')
plt.close()

print("Job engagement analysis complete. Scatter plot would show relationship between salary and views.")
```

```
Step 5: Exploratory Data Analysis - Job Engagement Metrics
Correlation between annual salary and views: -0.00
Job engagement analysis complete. Scatter plot would show relationship between salary and views.
```

Step 6: Geographic Analysis

```
# Count jobs by state
state_counts = df_clean['state'].value_counts()
print("Job count by state:")
print(state_counts)

# Calculate average salary by state
avg_salary_by_state = df_clean.groupby('state')['annual_salary'].mean().sort_values(ascending=False)
print("\nAverage annual salary by state:")
print(avg_salary_by_state)

# Create a bar chart of average salaries by state
plt.figure(figsize=(12, 6))
avg_salary_by_state.plot(kind='bar')
plt.title('Average Annual Salary by State')
plt.xlabel('State')
plt.ylabel('Average Annual Salary (USD)')
plt.xticks(rotation=45)
plt.savefig('salary_by_state.png')
plt.close()

print("Geographic analysis complete. Bar chart would show salary differences by state.")
```

```
MA      111632.367371
```

```
MA    111622.267271
VA    110967.283502
AK    109444.260000
NJ    108935.327756
IL    104995.290957
CT    104889.521200
GA    104116.580026
PA    103560.264456
MD    102487.342844
DE     98904.891724
OR     97767.734355
CO     95088.937083
AZ     90629.793079
RI     89990.593846
NV     89340.292085
WV     86914.492500
NM     85258.026087
AL     84202.775355
UT     83970.373197
TN     83624.017346
WI     82592.077742
MO     81805.884222
WY     81232.014815
LA     79035.357404
NH     78809.804380
MI     78713.954177
IN     77680.016238
NE     77472.437500
IA     77001.970101
AR     76263.453731
ID     75231.750667
ME     74346.384124
MT     73362.462687
ND     72879.945902
KS     71872.760505
HI     71094.840419
VT     71039.405349
MS     68116.427692
SD     68042.502041
ON              NaN
QC              NaN
Name: annual_salary, dtype: float64
Geographic analysis complete. Bar chart would show salary differences by state.
```

Step 7: Salary Prediction Model (Basic)

```
# For this simple example, we'll create a basic linear regression model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Select features for the model
features = ['views', 'post_age_days', 'is_remote']
available_features = [f for f in features if f in df_clean.columns]

# Prepare the data (drop rows with missing values)
model_data = df_clean.dropna(subset=['annual_salary'] + available_features)

if len(model_data) >= 5:  # Need at least 5 rows for meaningful split
    # Split the data into training and testing sets
    X = model_data[available_features]
    y = model_data['annual_salary']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train the model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Model trained with features: {available_features}")
    print(f"Mean Squared Error: {mse:.2f}")
    print(f"R² Score: {r2:.2f}")

    # Feature importance
    feature_importance = pd.DataFrame({
        'Feature': available_features,
        'Importance': model.coef_
    })
```

```
    print("\nFeature importance:")
    print(feature_importance.sort_values('Importance', ascending=False))
else:
    print("Not enough complete data rows to build a prediction model.")
    print("In a real analysis, you would need more data points.")
```

```
Step 7: Salary Prediction Model (Basic)
Model trained with features: ['views', 'post_age_days', 'is_remote']
Mean Squared Error: 7656380123681.46
R² Score: -0.00

Feature importance:
         Feature    Importance
2      is_remote  74075.208572
0          views   -110.530881
1  post_age_days  -1192.381103
```

Step 8: Text Analysis of Job Descriptions

```
# Basic text analysis
def count_words(text):
    if pd.isna(text):
        return 0
    return len(str(text).split())

df_clean['description_word_count'] = df_clean['description'].apply(count_words)
df_clean['skills_word_count'] = df_clean['skills_desc'].apply(count_words)

print("Job description statistics:")
print(df_clean[['description_word_count', 'skills_word_count']].describe())

# Example of text processing for keyword extraction
from collections import Counter
import re

def extract_keywords(text_series):
    # Combine all text
    all_text = ' '.join(text_series.dropna().astype(str))

    # Clean text and tokenize
    words = re.findall(r'\b[a-zA-Z]{3,}\b', all_text.lower())

    # Remove common stopwords (simplified list)
    stopwords = {'and', 'the', 'for', 'with', 'are', 'this', 'that', 'you', 'our', 'has', 'have'}
    words = [word for word in words if word not in stopwords]

    # Count word frequencies
    word_counts = Counter(words)
    return word_counts.most_common(10)

top_description_keywords = extract_keywords(df_clean['description'])
print("\nTop keywords in job descriptions:")
print(top_description_keywords)

top_skills_keywords = extract_keywords(df_clean['skills_desc'])
print("\nTop keywords in skills requirements:")
print(top_skills_keywords)
```

```
Step 8: Text Analysis of Job Descriptions
Job description statistics:
       description_word_count  skills_word_count
count           123849.000000      123849.000000
mean               523.027929           0.500166
std                301.940688          11.201057
min                  0.000000           0.000000
25%                298.000000           0.000000
50%                477.000000           0.000000
75%                696.000000           0.000000
max               3400.000000         529.000000

Top keywords in job descriptions:
[('experience', 374182), ('work', 349453), ('will', 278659), ('all', 274650), ('team', 250455), ('your', 204374), ('other', 197064),

Top keywords in skills requirements:
[('skills', 879), ('experience', 629), ('position', 527), ('following', 478), ('requires', 473), ('ability', 436), ('work', 380), (
```

Step 9: Job Market Segmentation

```
# Prepare data for clustering
cluster_features = ['annual_salary', 'views', 'remote_allowed']
cluster_data = df_clean[cluster_features].dropna()

if len(cluster_data) >= 5:  # Need at least 5 rows for meaningful clustering
    # Scale the data
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(cluster_data)

    # Determine optimal number of clusters (using elbow method, simplified for this example)
    max_clusters = min(4, len(cluster_data) - 1)  # Simplified for small dataset
    wcss = []
    for i in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
        kmeans.fit(scaled_data)
        wcss.append(kmeans.inertia_)

    # Apply K-Means clustering
    k = 2  # Simplified choice for this example
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    cluster_data['cluster'] = kmeans.fit_predict(scaled_data)

    # Analyze clusters
    cluster_analysis = cluster_data.groupby('cluster').mean()
    print("Job market segments by salary and engagement:")
    print(cluster_analysis)

    # Visualize clusters (2D projection)
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x='annual_salary', y='views', hue='cluster', data=cluster_data, palette='viridis')
    plt.title('Job Market Segments')
    plt.xlabel('Annual Salary (USD)')
    plt.ylabel('Number of Views')
    plt.savefig('job_clusters.png')
    plt.close()

    print("Segmentation analysis complete. Scatter plot would show different job market segments.")
else:
    print("Not enough complete data rows for clustering analysis.")
    print("In a real analysis, you would need more data points.")
```

```
Step 9: Job Market Segmentation
Job market segments by salary and engagement:
         annual_salary     views  remote_allowed
cluster
0         1.400893e+05  58.06402             1.0
1         3.120000e+08  22.00000             1.0
Segmentation analysis complete. Scatter plot would show different job market segments.
```

Step 10: Insights and Recommendations

```
# Calculate job market attractiveness score
# Higher score = more competitive compensation and more engagement
df_clean['market_score'] = (
    df_clean['annual_salary'].fillna(df_clean['annual_salary'].median()) / df_clean['annual_salary'].median() * 0.7 +
    df_clean['views'].fillna(df_clean['views'].median()) / df_clean['views'].median() * 0.3
)

# Sort by market score
top_jobs = df_clean.sort_values('market_score', ascending=False).head(3)
print("Top jobs by market attractiveness:")
print(top_jobs[['title', 'company_name', 'annual_salary', 'views', 'market_score']])

# Identify potential undervalued jobs
# Jobs with high view counts but lower salaries
df_clean['salary_percentile'] = df_clean['annual_salary'].rank(pct=True)
df_clean['views_percentile'] = df_clean['views'].rank(pct=True)
df_clean['value_gap'] = df_clean['views_percentile'] - df_clean['salary_percentile']

undervalued_jobs = df_clean.sort_values('value_gap', ascending=False).head(3)
print("\nPotentially undervalued jobs (high interest, lower salary):")
print(undervalued_jobs[['title', 'company_name', 'annual_salary', 'views', 'value_gap']])

# Summary insights
print("\nSummary Insights:")
```

```
print("\nSummary Insights:")
print(f"1. Average annual salary across all jobs: ${df_clean['annual_salary'].mean():.2f}")
print(f"2. Most jobs are {df_clean['formatted_work_type'].mode()[0]}")
print(f"3. Remote-friendly jobs: {df_clean['is_remote'].sum()} out of {len(df_clean)}")
print(f"4. Average job view count: {df_clean['views'].mean():.1f}")
print(f"5. Most job postings are from {df_clean['state'].mode()[0] if not df_clean['state'].mode().empty else 'various states'}")

print("\nRecommendations:")
print("1. Focus job search on high market score positions for best compensation")
print("2. Consider undervalued jobs for positions with high interest but potentially less competition")
print("3. Research companies in states with higher average salaries")
print("4. For employers: Include comprehensive skills descriptions to attract more qualified candidates")
print("5. For job seekers: Target positions with detailed job descriptions as they tend to offer better compensation")
```

```
Step 10: Insights and Recommendations
Top jobs by market attractiveness:
                                         title  \
9237   Intellectual Property Associate (246215)
98888        Case Manager RN, Pedi Rheumatology
89082                    Cloud Domain Architect

                       company_name  annual_salary  views  market_score
9237   Eastridge Workforce Solutions    572000000.0    4.0   4449.188889
98888              Kaiser Permanente    408865600.0    4.0   3180.365778
89082                     Applicantz    312000000.0   42.0   2429.816667

Potentially undervalued jobs (high interest, lower salary):
                                title    company_name  annual_salary  \
46750                   UX/UI Designer       MIDIScale            0.0
29777         Human Resources Generalist       Lionsgate           32.0
40894  Remote Software Engineer (Python)  Insight Global           63.0

       views  value_gap
46750  260.0   0.994201
29777  476.0   0.993421
40894  306.0   0.988282

Summary Insights:
1. Average annual salary across all jobs: $227568.81
2. Most jobs are Full-time
3. Remote-friendly jobs: 15246 out of 123849
4. Average job view count: 14.6
5. Most job postings are from CA

Recommendations:
1. Focus job search on high market score positions for best compensation
2. Consider undervalued jobs for positions with high interest but potentially less competition
3. Research companies in states with higher average salaries
4. For employers: Include comprehensive skills descriptions to attract more qualified candidates
5. For job seekers: Target positions with detailed job descriptions as they tend to offer better compensation
```