Machine Learning Methods

Regression, Classification, Topic Insights

Dakota Alder

Julio Garcia

# Data Loading and Cleaning

from pyspark.sql import SparkSession  
  
# Start a Spark session  
spark = SparkSession.builder.appName("JobPostingsAnalysis").getOrCreate()  
  
# Load the CSV file into a Spark DataFrame  
df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("../data/lightcast\_job\_postings.csv")

WARNING: Using incubator modules: jdk.incubator.vector  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/16 04:16:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
[Stage 1:> (0 + 1) / 1]

import pandas as pd  
from pyspark.sql.functions import when, col  
  
#Clean Data to convert to Pandas  
columns\_to\_drop = ["ID", "BODY", "URL", "ACTIVE\_URLS", "DUPLICATES", "LAST\_UPDATED\_TIMESTAMP",  
 "NAICS2", "NAICS3", "NAICS4", "NAICS5", "NAICS6",  
 "SOC\_2", "SOC\_3", "SOC\_4", "SOC\_5", "LAST\_UPDATED\_DATE", "LAST\_UPDATED\_TIMESTAMP", "EXPIRED", "SOURCE\_TYPES", "SOURCES", "ACTIVE\_SOURCES\_INFO", "MODELED\_EXPIRED", "MODELED\_DURATION", "NAICS2\_NAME", "NAICS3\_NAME", "NAICS4\_NAME", "NAICS5\_NAME", "NAICS6\_NAME",  
 "SOC\_2\_NAME", "SOC\_3\_NAME", "SOC\_4\_NAME", "SOC\_5\_NAME", "EDUCATION\_LEVELS", "MIN\_EDULEVELS"  
   
]  
cleaned\_data = df.drop(\*columns\_to\_drop)  
  
cleaned\_data = cleaned\_data.withColumn(  
 "REMOTE\_TYPE\_NAME",  
 when(col("REMOTE\_TYPE\_NAME") == "Remote", "Remote")  
 .when(col("REMOTE\_TYPE\_NAME") == "Hybrid Remote", "Hybrid")  
 .when(col("REMOTE\_TYPE\_NAME") == "[None]", "On-site")  
 .when(col("REMOTE\_TYPE\_NAME").isNull(), "On-site")  
 .when(col("REMOTE\_TYPE\_NAME") == "Not Remote", "On-site")  
 .otherwise(col("REMOTE\_TYPE\_NAME"))  
)  
  
cleaned\_data = cleaned\_data.withColumn(  
 "EMPLOYMENT\_TYPE\_NAME",  
 when(col("EMPLOYMENT\_TYPE\_NAME") == "Part-time / full-time", "Flexible")  
 .when(col("EMPLOYMENT\_TYPE\_NAME").isNull(), "Full-Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME") == "Part-time (â‰¤ 32 hours)", "Part-Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME") == "Full-time (> 32 hours)", "Full-Time")  
 .otherwise(col("EMPLOYMENT\_TYPE\_NAME"))   
)  
  
cleaned\_data = cleaned\_data.filter(col("NAICS\_2022\_2\_NAME") != "Unclassified Industry")  
  
median\_salary = cleaned\_data.approxQuantile("SALARY", [0.5], 0.01)[0]  
cleaned\_data = cleaned\_data.withColumn(  
 "SALARY",  
 when(col("SALARY").isNull(), median\_salary).otherwise(col("SALARY"))  
)  
  
#Convert to Pandas  
clean\_pdf = cleaned\_data.toPandas()

[Stage 2:> (0 + 1) / 1] 25/10/16 04:16:46 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.  
[Stage 3:> (0 + 1) / 1]

# Cleaning empty rows and dropping columns that are mostly empty  
  
  
fill\_cols = ["CITY\_NAME", "CITY", "LOCATION", "STATE", "STATE\_NAME", "COMPANY", "COMPANY\_NAME"]  
clean\_pdf[fill\_cols] = clean\_pdf[fill\_cols].fillna("Unknown")  
  
clean\_pdf = clean\_pdf.drop\_duplicates(subset=["TITLE", "COMPANY", "LOCATION", "POSTED"], keep="first")  
  
clean\_pdf.dropna(thresh=len(clean\_pdf)\*0.5, axis=1, inplace=True)  
  
#New Column to Classify AI Jobs and Add Month of Posting Date  
  
ai\_keywords = [  
 "AI", "Machine Learning", "Data Scientist", "Data Analyst", "ML",   
 "Artificial Intelligence", "Deep Learning", "NLP", "Predictive Analytics"  
]  
  
#Function to classify AI vs Non-AI Jobs  
def classify\_ai(title):  
 title\_lower = str(title).lower()  
 for keyword in ai\_keywords:  
 if keyword.lower() in title\_lower:  
 return "AI"  
 return "Non-AI"  
  
clean\_pdf["AI\_JOB"] = clean\_pdf["TITLE\_RAW"].apply(classify\_ai)  
  
clean\_pdf["POSTED"] = pd.to\_datetime(clean\_pdf["POSTED"], errors="coerce")  
clean\_pdf["POSTED\_MONTH"] = clean\_pdf["POSTED"].dt.month  
  
#Add column for URBAN vs RURAL  
  
clean\_pdf["URBAN\_RURAL"] = clean\_pdf["MSA\_NAME"].apply(lambda x: "Urban" if pd.notnull(x) else "Rural")

# Model 1: KMeans Clustering

We applied KMeans clustering to identify patterns in the data. We wanted to each cluster to represent a group of jobs based on characteristics of Salary, AI jobs, remote work, and Urban or Rural locations. We used an elbow method to find the ideal number of clusters, which was 4, though the model didn’t reflect as concise groupings as wanted.

**Features:**

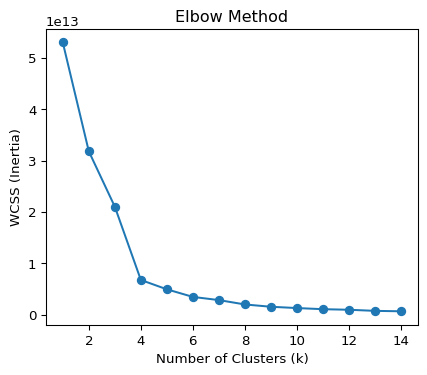
SALARY: Continuous numerical feature to reflect compensation.  
AI\_JOB: Binary feature (AI vs Non-AI), one-hot encoded.  
REMOTE\_TYPE\_NAME: One-hot encoded categories (On-site, Remote, Hybrid).  
URBAN\_RURAL: One-hot encoded (Urban or Rural).

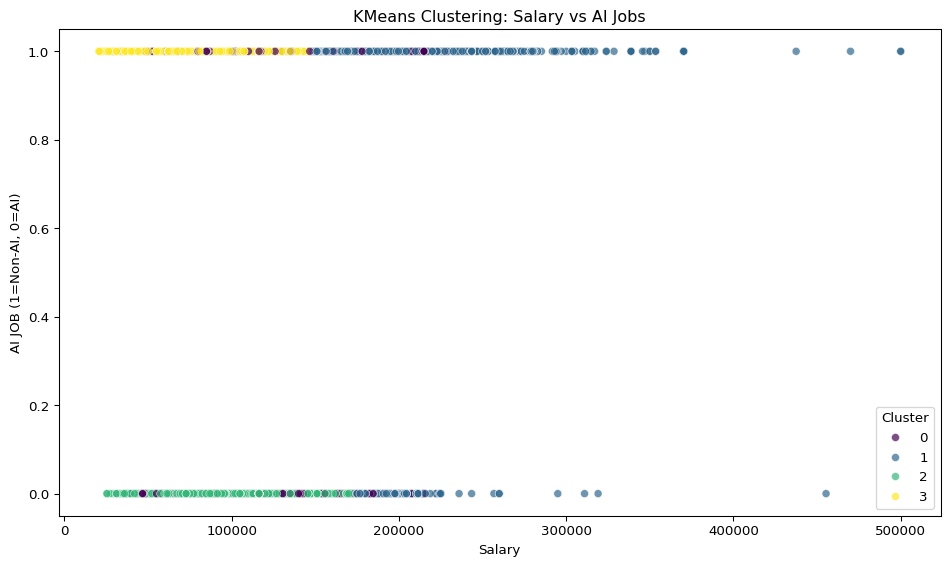
**Implications for Job Seekers**

A job seeker can use this model to see what type of characteristics of a job might be tied to others. For example, AI jobs might yield high salaries, and if we used a reference of industry might be able to find an industry of interest that falls in a cluster that is being regarded in the job hunt. This can also be tied into the Skills Gap Analysis to see what a job seeker should work on in order to be considered for a certain cluster.

#KMeans clustering using NAICS as a reference but not a target  
  
from sklearn.preprocessing import StandardScaler  
  
# Select features  
features = clean\_pdf[["SALARY", "AI\_JOB", "REMOTE\_TYPE\_NAME", "URBAN\_RURAL"]]  
  
# One-hot encode categorical columns  
features\_encoded = pd.get\_dummies(features, columns=["AI\_JOB", "REMOTE\_TYPE\_NAME", "URBAN\_RURAL"], drop\_first=True)  
  
# Standardize numerical features (important for KMeans)  
scaler = StandardScaler()  
features\_scaled = scaler.fit\_transform(features\_encoded)  
  
from sklearn.cluster import KMeans  
  
k = 4  
kmeans = KMeans(n\_clusters=k, random\_state=42)  
clean\_pdf["CLUSTER"] = kmeans.fit\_predict(features\_scaled)  
  
#Use Industry Name (NAICS2022) as a reference label  
  
cluster\_summary = (  
 clean\_pdf.groupby(["CLUSTER", "NAICS\_2022\_2\_NAME"])  
 .size()  
 .reset\_index(name="count")  
 .sort\_values(["CLUSTER", "count"], ascending=[True, False])  
)  
  
print(cluster\_summary)  
  
#Used an Elbow Method to choose the correct number of clusters  
  
from sklearn.cluster import KMeans  
import matplotlib.pyplot as plt  
  
# Example features  
X = features\_encoded.values # your numerical features  
  
wcss = []  
for k in range(1, 15):  
 km = KMeans(n\_clusters=k, random\_state=42)  
 km.fit(X)  
 wcss.append(km.inertia\_)  
  
plt.plot(range(1, 15), wcss, marker='o')  
plt.xlabel('Number of Clusters (k)')  
plt.ylabel('WCSS (Inertia)')  
plt.title('Elbow Method')  
plt.show()  
  
cluster\_summary.head(20) # Show top 20 to see patterns  
  
one\_hot\_cols = ['AI\_JOB\_Non-AI', 'REMOTE\_TYPE\_NAME\_On-site', 'REMOTE\_TYPE\_NAME\_Remote', 'URBAN\_RURAL\_Urban']  
  
clean\_pdf = pd.concat([clean\_pdf, features\_encoded[one\_hot\_cols]], axis=1)  
  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10,6))  
sns.scatterplot(  
 data=clean\_pdf,  
 x='SALARY',  
 y='AI\_JOB\_Non-AI',   
 hue='CLUSTER',  
 palette='viridis',  
 alpha=0.7  
)  
plt.title("KMeans Clustering: Salary vs AI Jobs")  
plt.xlabel("Salary")  
plt.ylabel("AI JOB (1=Non-AI, 0=AI)")  
plt.legend(title="Cluster")  
plt.tight\_layout()  
plt.show()

CLUSTER NAICS\_2022\_2\_NAME count  
13 0 Professional, Scientific, and Technical Services 2968  
6 0 Finance and Insurance 1720  
1 0 Administrative and Support and Waste Managemen... 1598  
10 0 Manufacturing 734  
8 0 Information 696  
.. ... ... ...  
60 3 Accommodation and Food Services 353  
69 3 Management of Companies and Enterprises 146  
71 3 Mining, Quarrying, and Oil and Gas Extraction 95  
63 3 Arts, Entertainment, and Recreation 71  
62 3 Agriculture, Forestry, Fishing and Hunting 45  
  
[80 rows x 3 columns]

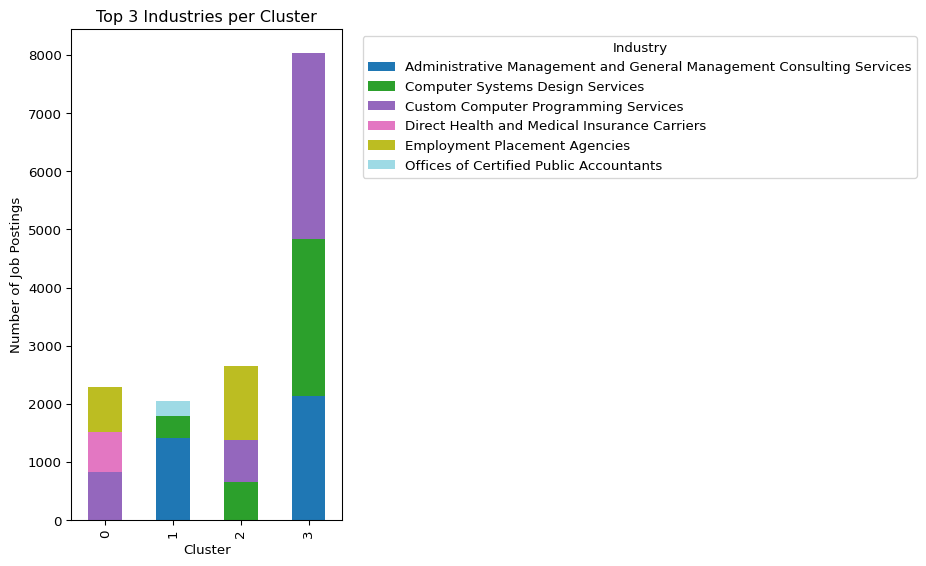




#Visualizing the NAICS Industries in reference to each cluster  
  
  
  
import matplotlib.pyplot as plt  
  
  
top\_industries = (  
 clean\_pdf.groupby(["CLUSTER", "NAICS\_2022\_6\_NAME"])  
 .size()  
 .reset\_index(name="count")  
)  
  
top3 = top\_industries.groupby("CLUSTER").apply(lambda x: x.nlargest(3, "count")).reset\_index(drop=True)  
  
  
pivot\_df = top3.pivot(index="CLUSTER", columns="NAICS\_2022\_6\_NAME", values="count").fillna(0)  
  
print(pivot\_df)  
  
pivot\_df.plot(kind='bar', stacked=True, figsize=(10,6), colormap='tab20')   
  
plt.title("Top 3 Industries per Cluster")  
plt.xlabel("Cluster")  
plt.ylabel("Number of Job Postings")  
plt.legend(title="Industry", bbox\_to\_anchor=(1.05, 1), loc='upper left')  
plt.tight\_layout()  
plt.show()

/tmp/ipykernel\_5387/24634743.py:14: FutureWarning:  
  
DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

NAICS\_2022\_6\_NAME Administrative Management and General Management Consulting Services \  
CLUSTER   
0 0.0   
1 1414.0   
2 0.0   
3 2139.0   
  
NAICS\_2022\_6\_NAME Computer Systems Design Services \  
CLUSTER   
0 0.0   
1 376.0   
2 663.0   
3 2694.0   
  
NAICS\_2022\_6\_NAME Custom Computer Programming Services \  
CLUSTER   
0 836.0   
1 0.0   
2 720.0   
3 3209.0   
  
NAICS\_2022\_6\_NAME Direct Health and Medical Insurance Carriers \  
CLUSTER   
0 685.0   
1 0.0   
2 0.0   
3 0.0   
  
NAICS\_2022\_6\_NAME Employment Placement Agencies \  
CLUSTER   
0 774.0   
1 0.0   
2 1268.0   
3 0.0   
  
NAICS\_2022\_6\_NAME Offices of Certified Public Accountants   
CLUSTER   
0 0.0   
1 257.0   
2 0.0   
3 0.0



## Key Insights

Out of the 4 clusters, it seems like there are 3 fairly clear clusters and 1 that is more ambiguous.

Cluster 0: The ambiguous one, doesn’t seem to have any groupings along AI, Non-AI, and Salary.  
Cluster 1: This is a group of higher salary paid positions, but are grouped regardless of AI vs Non-AI.  
Cluster 2: These seem to be low paying, AI jobs.  
Cluster 3: These seem to be low paying Non-AI jobs.

**Industry Relationship:**

It seems that the industries don’t have as much of a bearing on each cluster. For example, we thought that the high paying clusters would be more tech industry focused, but the low paying cluster also has a large amount of job openings in the same industries. This may mean that salaries are most likely influenced less by Industry and AI impact and influenced more by Skills, seniority, and education.

# Model 2: Linear Regression for Salary Prediction

We ran a linear regression model to predict job salaries based on location, remote status, and urban/rural classification. It estimates/predicts how location and job type impact salary. It’s useful for identifying which factors contribute to higher salaries and preferred work type.

**Features used:**

STATE\_NAME: One-hot encoded categorical variable representing each state.  
REMOTE\_TYPE\_NAME: One-hot encoded (Remote, Hybrid, On-site).  
URBAN\_RURAL: One-hot encoded (Urban vs. Rural).  
Target variable: SALARY (numerical).

**Implications for job seekers:**

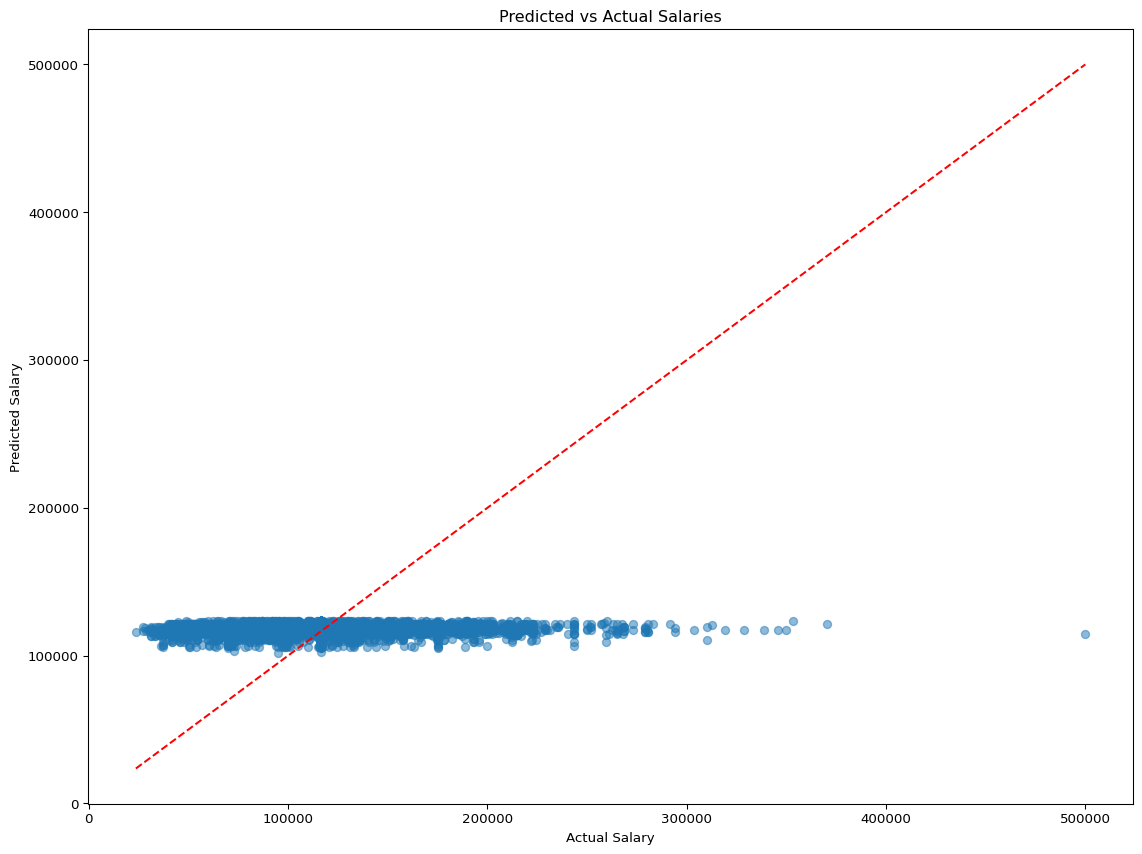
The model reveals which locations and job types may pay more, which is probably the most important consideration for job seekers.

For example, remote AI jobs in urban hubs may offer higher salaries than non-AI roles in rural areas.

Limitations: This model focuses only on geographic and job-type features, so salary effects of skills, experience, or certifications are not captured. As we show, geographical data is not a greate predictor or estimator in Salary. We assume that education levels, experience, and skills may be a better predictor.

#Predicting Salaries based on Location Data through Linear Regression  
#\*Decided to run it in Pandas with Scikit as it's already been converted and cleaned  
  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import numpy as np  
import matplotlib.pyplot as plt  
  
reg\_data = clean\_pdf[["SALARY", "STATE\_NAME", "REMOTE\_TYPE\_NAME", "URBAN\_RURAL"]]  
  
X = pd.get\_dummies(reg\_data[["STATE\_NAME", "REMOTE\_TYPE\_NAME", "URBAN\_RURAL"]], drop\_first=True)  
y = reg\_data["SALARY"]  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=42)  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
y\_pred = model.predict(X\_test)  
rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Evaulation Metrics:")  
print(f"RMSE: {rmse:,.2f}")  
print(f"R2: {r2:.3f}")  
  
df\_coef = pd.DataFrame({  
 "Feature": X.columns,  
 "Coefficient": model.coef\_  
}).sort\_values(by="Coefficient", ascending=False)  
  
print("\nTop PositiveInfluences on Salary:")  
print(df\_coef.head(10))  
  
print("\nTop Negative Influences on Salary:")  
print(df\_coef.tail(10))  
  
  
plt.figure(figsize=(12,9))  
plt.scatter(y\_test, y\_pred, alpha=0.5)  
plt.xlabel("Actual Salary")  
plt.ylabel("Predicted Salary")  
plt.title("Predicted vs Actual Salaries")  
plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--')  
plt.tight\_layout()  
plt.show()

Evaulation Metrics:  
RMSE: 29,689.15  
R2: 0.006  
  
Top PositiveInfluences on Salary:  
 Feature Coefficient  
45 STATE\_NAME\_Washington 6361.179175  
5 STATE\_NAME\_Connecticut 5837.080401  
43 STATE\_NAME\_Vermont 5302.264696  
3 STATE\_NAME\_California 4726.943831  
51 REMOTE\_TYPE\_NAME\_Remote 4274.612550  
50 REMOTE\_TYPE\_NAME\_On-site 4176.813834  
2 STATE\_NAME\_Arkansas 4109.440464  
46 STATE\_NAME\_Washington, D.C. (District of Colum... 2818.294899  
28 STATE\_NAME\_New Jersey 2786.906817  
11 STATE\_NAME\_Illinois 2575.153394  
  
Top Negative Influences on Salary:  
 Feature Coefficient  
49 STATE\_NAME\_Wyoming -3610.797563  
47 STATE\_NAME\_West Virginia -3985.698913  
22 STATE\_NAME\_Mississippi -4039.741194  
15 STATE\_NAME\_Kentucky -5710.215799  
0 STATE\_NAME\_Alaska -5941.167716  
42 STATE\_NAME\_Utah -6390.023354  
26 STATE\_NAME\_Nevada -7679.185737  
39 STATE\_NAME\_South Dakota -9360.455547  
32 STATE\_NAME\_North Dakota -10354.011369  
29 STATE\_NAME\_New Mexico -10744.064282



## Key Insights

**Evaulation Metrics:**  
RMSE: 29,689.15  
R2: 0.006

These metrics are saying that our features are not very influential on the salary, meaning that there are most likely better predictors of Salary than geographical predictors. Like previously stated, things like seniority, education, and skills may be better predictors to higher or lower salaries.

Another insight that might be able to be used by a job seeker might be the highest and negative influencers on salary. Even though our model isn’t great, it seems that we can deduce that seeking a job in Washington might yield a higher salary vs. the baseline salary, but seeking a job in New Mexico might yield a lower salary than the baseline.