Assignment 04

Caroline O’Sullivan (Boston University)  
September 27, 2025

from pyspark.sql import SparkSession  
import pandas as pd  
import plotly.express as px  
import plotly.io as pio  
import numpy as np  
  
np.random.seed(45)  
  
pio.renderers.default = "notebook+notebook\_connected+vscode"  
  
# Initialize Spark Session  
spark = SparkSession.builder.appName("LightcastData").getOrCreate()  
  
# Load Data  
df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/lightcast\_job\_postings.csv")  
  
# Show Schema and Sample Data  
#print("---This is Diagnostic check, No need to print it in the final doc---")  
  
# df.printSchema() # comment this line when rendering the submission  
#df.show(5)

Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/09 02:15:42 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

#Missing Value Treatment  
#1. Replace missing values in Salary by Median of Salary based on the Employment Type, if missing then replace with the overall median of Salary  
  
from pyspark.sql import Window  
from pyspark.sql.functions import col, when, isnan, count, lit, expr, avg, median, pow  
from pyspark.ml import Pipeline  
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler  
  
#select subset of columns  
eda\_cols = ["SALARY","MIN\_YEARS\_EXPERIENCE","EMPLOYMENT\_TYPE\_NAME","REMOTE\_TYPE\_NAME","STATE\_NAME","MIN\_EDULEVELS\_NAME","COMPANY\_IS\_STAFFING","IS\_INTERNSHIP"]  
df\_subset = df.select(eda\_cols)  
  
#Fix Remote Type Name incorrect labels to Remote, Hybrid, Onsite. None and NULL are Onsite.  
df\_subset = df\_subset.withColumn(  
 "REMOTE\_TYPE\_NAME",  
 when(col("REMOTE\_TYPE\_NAME") == "Remote", "Remote")  
 .when(col("REMOTE\_TYPE\_NAME") == "Not Remote", "On Site")  
 .when(col("REMOTE\_TYPE\_NAME") == "Hybrid Remote", "Hybrid")  
 .when((col("REMOTE\_TYPE\_NAME").isNull()) | (col("REMOTE\_TYPE\_NAME") == "[None]"), "On Site")  
 .otherwise("On Site")  
)  
  
#Clean Employment Type  
df\_subset = df\_subset.withColumn(  
 "EMPLOYMENT\_TYPE\_NAME",  
 when(col("EMPLOYMENT\_TYPE\_NAME").like("Part-time (â‰¤ 32%"), "Part Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME").like("Part-time / full-%"), "Flexible")  
 .when(col("EMPLOYMENT\_TYPE\_NAME").like("Full-time (> 32%"), "Full Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME").isNull(), "Full Time")  
 .otherwise("Full Time") # fallback for anything else  
)  
#Calculate the median of MIN\_YEARS\_EXPERIENCE column based on Employment Type  
median\_by\_emp\_type\_min\_exp = df.groupBy("EMPLOYMENT\_TYPE").agg(expr('percentile\_approx(MIN\_YEARS\_EXPERIENCE, 0.5)').alias('median\_min\_years\_experience\_by\_emp\_type'))  
  
#Median Salary  
median\_salary = df\_subset.approxQuantile("SALARY", [0.5], 0.25)[0]  
median\_by\_emp\_type = (  
 df\_subset.groupBy("EMPLOYMENT\_TYPE\_NAME")  
 .agg(expr("percentile\_approx(SALARY, 0.5)").alias("median\_salary\_by\_emp\_type"))  
)  
  
df\_subset = df\_subset.join(median\_by\_emp\_type, on="EMPLOYMENT\_TYPE\_NAME", how="left")  
df\_subset = df\_subset.withColumn(  
 "SALARY",  
 when(  
 col("SALARY").isNull(),  
 when(col("median\_salary\_by\_emp\_type").isNotNull(), col("median\_salary\_by\_emp\_type"))  
 .otherwise(lit(median\_salary))  
 ).otherwise(col("SALARY"))  
).drop("median\_salary\_by\_emp\_type")  
  
#Median MIN\_YEARS\_EXPERIENCE  
median\_by\_emp\_type\_min\_exp = df.groupBy("EMPLOYMENT\_TYPE\_NAME").agg(  
 expr('percentile\_approx(MIN\_YEARS\_EXPERIENCE, 0.5)').alias('median\_min\_years\_experience\_by\_emp\_type')  
)  
df\_subset = df\_subset.join(median\_by\_emp\_type\_min\_exp, on="EMPLOYMENT\_TYPE\_NAME", how="left")  
df\_subset = df\_subset.withColumn(  
 "MIN\_YEARS\_EXPERIENCE",  
 when(  
 col("MIN\_YEARS\_EXPERIENCE").isNull(),  
 when(col("median\_min\_years\_experience\_by\_emp\_type").isNotNull(), col("median\_min\_years\_experience\_by\_emp\_type"))  
 .otherwise(lit(0)) # fallback if needed  
 ).otherwise(col("MIN\_YEARS\_EXPERIENCE"))  
).drop("median\_min\_years\_experience\_by\_emp\_type")  
  
#False for IS\_INTERNSHIP and COMPANY\_IS\_STAFFING if NULL  
df\_subset = df\_subset.withColumn(  
 "IS\_INTERNSHIP",  
 when(col("IS\_INTERNSHIP").isNull(), lit(False)).otherwise(col("IS\_INTERNSHIP"))  
).withColumn(  
 "COMPANY\_IS\_STAFFING",  
 when(col("COMPANY\_IS\_STAFFING").isNull(), lit(False)).otherwise(col("COMPANY\_IS\_STAFFING"))  
)  
  
df\_subset = df\_subset.withColumn("IS\_INTERNSHIP\_num", col("IS\_INTERNSHIP").cast("int"))  
df\_subset = df\_subset.withColumn("COMPANY\_IS\_STAFFING\_num", col("COMPANY\_IS\_STAFFING").cast("int"))  
  
df\_subset = df\_subset.dropna()

Linear Regression Model

#Feature Engineering  
#String Indexing and One Hot Encoding for Categorical Variables  
categorical\_cols = ["EMPLOYMENT\_TYPE\_NAME","REMOTE\_TYPE\_NAME","MIN\_EDULEVELS\_NAME","STATE\_NAME"]  
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}\_idx", handleInvalid='skip') for col in categorical\_cols]  
encoders = [OneHotEncoder(inputCol=f"{col}\_idx", outputCol=f"{col}\_vec") for col in categorical\_cols]  
  
assembler = VectorAssembler(  
 inputCols=[  
 "MIN\_YEARS\_EXPERIENCE",  
 "COMPANY\_IS\_STAFFING",  
 "IS\_INTERNSHIP"  
 ] + [f"{col}\_vec" for col in categorical\_cols],  
 outputCol="features"  
)  
  
pipeline = Pipeline(stages=indexers + encoders + [assembler])  
data = pipeline.fit(df\_subset).transform(df\_subset)

#Test Train Split  
train\_data, test\_data = data.randomSplit([0.8, 0.2], seed=45)

from pyspark.ml.regression import GeneralizedLinearRegression  
from pyspark.sql import Row  
  
feature\_names = assembler.getInputCols()  
  
glr = GeneralizedLinearRegression(  
 featuresCol="features",  
 labelCol="SALARY",  
 family="gaussian",  
 link="identity",  
 maxIter=10,  
 regParam=0.3,  
)  
  
glr\_model = glr.fit(train\_data)  
glr\_summary = glr\_model.summary

#Model Summary  
  
print("Intercept: {:.4f}".format(glr\_model.intercept))  
print("Coefficients:")  
for i, coef in enumerate(glr\_model.coefficients):  
 print(f" Feature {i + 1}: {coef:.4f}")

Intercept: 102278.9407  
Coefficients:  
 Feature 1: 2378.7010  
 Feature 2: -2835.2513  
 Feature 3: -3435.8114  
 Feature 4: 12006.3027  
 Feature 5: -8290.8537  
 Feature 6: 1538.9051  
 Feature 7: 1088.4958  
 Feature 8: -5950.0787  
 Feature 9: -3615.2379  
 Feature 10: -22564.1717  
 Feature 11: -11979.7180  
 Feature 12: 6353.9330  
 Feature 13: -1553.1378  
 Feature 14: 2688.0705  
 Feature 15: -2274.1549  
 Feature 16: 490.1882  
 Feature 17: 576.8265  
 Feature 18: -164.8164  
 Feature 19: -1131.9933  
 Feature 20: -1786.4970  
 Feature 21: -2290.7826  
 Feature 22: 1624.7187  
 Feature 23: -2157.5368  
 Feature 24: 2.7998  
 Feature 25: -687.9366  
 Feature 26: -2652.3943  
 Feature 27: 3176.2818  
 Feature 28: -1391.5348  
 Feature 29: -3838.0329  
 Feature 30: -2622.0994  
 Feature 31: -1952.8745  
 Feature 32: -1542.7176  
 Feature 33: -641.1122  
 Feature 34: -3499.4323  
 Feature 35: -930.2304  
 Feature 36: -996.0715  
 Feature 37: 2871.9190  
 Feature 38: -2375.4212  
 Feature 39: -2173.8021  
 Feature 40: -1138.4543  
 Feature 41: -4351.5362  
 Feature 42: -1842.2851  
 Feature 43: -1461.6578  
 Feature 44: -1012.8358  
 Feature 45: -6805.4274  
 Feature 46: 2939.3664  
 Feature 47: -1612.1046  
 Feature 48: -2197.0598  
 Feature 49: -3211.0793  
 Feature 50: -760.8950  
 Feature 51: -226.1467  
 Feature 52: -78.5807  
 Feature 53: -3717.8571  
 Feature 54: -4033.0228  
 Feature 55: -7850.0135  
 Feature 56: -4341.2085  
 Feature 57: -9986.9455  
 Feature 58: -7546.0152  
 Feature 59: 3243.1689  
 Feature 60: 811.4712  
 Feature 61: -7436.7514  
 Feature 62: -10989.4340

#Summary Statistics  
print("Coefficient Standard Errors:", [f"{val:.4f}" for val in glr\_summary.coefficientStandardErrors])  
print("T Values:", [f"{val:.4f}" for val in glr\_summary.tValues])  
print("P Values:", [f"{val:.4f}" for val in glr\_summary.pValues])

[Stage 63:> (0 + 1) / 1]

Coefficient Standard Errors: ['32.1098', '345.5202', '758.5322', '998.1229', '1182.6848', '667.5953', '711.0274', '2935.8109', '2938.5952', '2974.3190', '2986.2789', '3017.8056', '3103.7287', '3105.9301', '3125.4587', '3126.0381', '3126.9004', '3129.9410', '3139.8876', '3141.4249', '3143.4861', '3143.4611', '3153.0125', '3157.2838', '3167.5916', '3177.9271', '3177.1376', '3188.2625', '3189.0496', '3192.7196', '3202.2389', '3205.5528', '3208.6622', '3227.4155', '3228.2393', '3244.4764', '3257.6197', '3281.2210', '3307.3549', '3317.3929', '3315.0877', '3323.9190', '3323.6527', '3334.4591', '3339.7964', '3331.9678', '3364.9221', '3402.6602', '3400.3004', '3395.5481', '3402.1817', '3423.3624', '3502.2973', '3560.7282', '3596.8296', '3635.7330', '3648.5880', '3711.9215', '3718.6007', '3826.0438', '3879.8121', '4034.6588', '4397.4497']  
T Values: ['74.0801', '-8.2057', '-4.5296', '12.0289', '-7.0102', '2.3051', '1.5309', '-2.0267', '-1.2303', '-7.5863', '-4.0116', '2.1055', '-0.5004', '0.8655', '-0.7276', '0.1568', '0.1845', '-0.0527', '-0.3605', '-0.5687', '-0.7287', '0.5169', '-0.6843', '0.0009', '-0.2172', '-0.8346', '0.9997', '-0.4365', '-1.2035', '-0.8213', '-0.6098', '-0.4813', '-0.1998', '-1.0843', '-0.2882', '-0.3070', '0.8816', '-0.7239', '-0.6573', '-0.3432', '-1.3126', '-0.5543', '-0.4398', '-0.3037', '-2.0377', '0.8822', '-0.4791', '-0.6457', '-0.9444', '-0.2241', '-0.0665', '-0.0230', '-1.0615', '-1.1326', '-2.1825', '-1.1940', '-2.7372', '-2.0329', '0.8721', '0.2121', '-1.9168', '-2.7238', '23.2587']  
P Values: ['0.0000', '0.0000', '0.0000', '0.0000', '0.0000', '0.0212', '0.1258', '0.0427', '0.2186', '0.0000', '0.0001', '0.0353', '0.6168', '0.3868', '0.4668', '0.8754', '0.8536', '0.9580', '0.7185', '0.5696', '0.4662', '0.6053', '0.4938', '0.9993', '0.8281', '0.4039', '0.3174', '0.6625', '0.2288', '0.4115', '0.5420', '0.6303', '0.8416', '0.2782', '0.7732', '0.7588', '0.3780', '0.4691', '0.5110', '0.7315', '0.1893', '0.5794', '0.6601', '0.7613', '0.0416', '0.3777', '0.6319', '0.5185', '0.3450', '0.8227', '0.9470', '0.9817', '0.2884', '0.2574', '0.0291', '0.2325', '0.0062', '0.0421', '0.3831', '0.8320', '0.0553', '0.0065', '0.0000']

#Dispersion  
print(f"Null Deviance: {glr\_summary.nullDeviance:.4f}")  
print(f"Residual DF Null: {glr\_summary.residualDegreeOfFreedomNull}")  
print(f"Residual DF: {glr\_summary.residualDegreeOfFreedom}")  
print(f"AIC: {glr\_summary.aic:.4f}")  
print(f"Deviance: {glr\_summary.deviance:.4f}")

Null Deviance: 51607738345410.0703  
Residual DF Null: 57956  
Residual DF: 57894

[Stage 80:> (0 + 1) / 1]

AIC: 1350095.6573  
Deviance: 44306505655818.4141

feature\_names = glr\_summary.\_call\_java("featureNames")  
features = ["Intercept"] + feature\_names  
coefs = [glr\_model.intercept] + list(glr\_model.coefficients)  
se = list(glr\_summary.coefficientStandardErrors)  
t\_values = list(glr\_summary.tValues)  
p\_values = list(glr\_summary.pValues)  
  
print("Length of features:", len(features))  
print("Length of coefficients:", len(coefs))  
print("Length of standard errors:", len(se))  
print("Length of t-values:", len(t\_values))  
print("Length of p-values:", len(p\_values))

Length of features: 63  
Length of coefficients: 63  
Length of standard errors: 63  
Length of t-values: 63  
Length of p-values: 63

import pandas as pd  
from tabulate import tabulate  
from IPython.display import HTML  
  
summary\_df = pd.DataFrame({  
 "Feature": features,  
 "Estimate": [f"{v:.4f}" if v is not None else None for v in coefs],  
 "Std. Error": [f"{v:.4f}" if v is not None else None for v in se],  
 "t-value": [f"{v:.4f}" if v is not None else None for v in t\_values],  
 "p-value": [f"{v:.4f}" if v is not None else None for v in p\_values]  
})  
summary\_df.to\_csv("output/glr\_model\_summary.csv", index=False)  
summary\_df

|  | Feature | Estimate | Std. Error | t-value | p-value |
| --- | --- | --- | --- | --- | --- |
| 0 | Intercept | 102278.9407 | 32.1098 | 74.0801 | 0.0000 |
| 1 | MIN\_YEARS\_EXPERIENCE | 2378.7010 | 345.5202 | -8.2057 | 0.0000 |
| 2 | COMPANY\_IS\_STAFFING | -2835.2513 | 758.5322 | -4.5296 | 0.0000 |
| 3 | IS\_INTERNSHIP | -3435.8114 | 998.1229 | 12.0289 | 0.0000 |
| 4 | EMPLOYMENT\_TYPE\_NAME\_vec\_Full Time | 12006.3027 | 1182.6848 | -7.0102 | 0.0000 |
| ... | ... | ... | ... | ... | ... |
| 58 | STATE\_NAME\_vec\_Alaska | -7546.0152 | 3718.6007 | 0.8721 | 0.3831 |
| 59 | STATE\_NAME\_vec\_Vermont | 3243.1689 | 3826.0438 | 0.2121 | 0.8320 |
| 60 | STATE\_NAME\_vec\_Montana | 811.4712 | 3879.8121 | -1.9168 | 0.0553 |
| 61 | STATE\_NAME\_vec\_West Virginia | -7436.7514 | 4034.6588 | -2.7238 | 0.0065 |
| 62 | STATE\_NAME\_vec\_North Dakota | -10989.4340 | 4397.4497 | 23.2587 | 0.0000 |

#Create a numeric summary DataFrame for further analysis  
summary\_numeric = pd.DataFrame({  
 "Feature": features,  
 "Estimate": coefs,  
 "Std\_Error": se,  
 "t\_value": t\_values,  
 "p\_value": p\_values  
})  
  
# Add significance flag  
summary\_numeric["Significant"] = summary\_numeric["p\_value"] < 0.05  
  
# Sort by absolute t-value  
summary\_numeric = summary\_numeric.reindex(summary\_numeric["t\_value"].abs().sort\_values(ascending=False).index)  
summary\_numeric.head(10)

|  | Feature | Estimate | Std\_Error | t\_value | p\_value | Significant |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | Intercept | 102278.940654 | 32.109834 | 74.080141 | 0.000000e+00 | True |
| 62 | STATE\_NAME\_vec\_North Dakota | -10989.434042 | 4397.449726 | 23.258695 | 0.000000e+00 | True |
| 3 | IS\_INTERNSHIP | -3435.811431 | 998.122865 | 12.028883 | 0.000000e+00 | True |
| 1 | MIN\_YEARS\_EXPERIENCE | 2378.701045 | 345.520191 | -8.205747 | 2.220446e-16 | True |
| 9 | MIN\_EDULEVELS\_NAME\_vec\_No Education Listed | -3615.237936 | 2974.318976 | -7.586332 | 3.330669e-14 | True |
| 4 | EMPLOYMENT\_TYPE\_NAME\_vec\_Full Time | 12006.302678 | 1182.684819 | -7.010197 | 2.405853e-12 | True |
| 2 | COMPANY\_IS\_STAFFING | -2835.251337 | 758.532194 | -4.529553 | 5.922656e-06 | True |
| 10 | MIN\_EDULEVELS\_NAME\_vec\_High school or GED | -22564.171663 | 2986.278876 | -4.011587 | 6.038759e-05 | True |
| 56 | STATE\_NAME\_vec\_Hawaii | -4341.208540 | 3648.588031 | -2.737208 | 6.198195e-03 | True |
| 61 | STATE\_NAME\_vec\_West Virginia | -7436.751355 | 4034.658810 | -2.723758 | 6.456314e-03 | True |

Analysis: I created a duplicate table and added a significance flag to better understand the model. I then limited it by the top 10 Features by T value. In the table above, I can see that the baseline salary in my dataset is $102,278. The model predicts that being based in North Dakota means that you will earn $10,989 LESS in salary. Additionally, an internship will earn you $3,435 less in salary. When it comes to your minimum years of experience, your salary will increase by $2378 for each additional year of experience you have. These observations are in line with what I would expect but the baseline salary is rather high in this dataset compared to national averages.

# Polynomial Regression

poly\_data = data.withColumn("MIN\_YEARS\_EXPERIENCE\_sq", pow(col("MIN\_YEARS\_EXPERIENCE"), 2))  
  
assembler\_poly = VectorAssembler(  
 inputCols=[  
 "MIN\_YEARS\_EXPERIENCE",  
 "MIN\_YEARS\_EXPERIENCE\_sq",  
 "COMPANY\_IS\_STAFFING",  
 "IS\_INTERNSHIP"  
 ] + [f"{col}\_vec" for col in categorical\_cols],  
 outputCol="features\_poly"  
)  
  
poly\_data = assembler\_poly.transform(poly\_data)  
poly\_data.select("SALARY","features\_poly").show(5,truncate=False)

[Stage 83:> (0 + 1) / 1]

+--------+---------------------------------------------+  
|SALARY |features\_poly |  
+--------+---------------------------------------------+  
|103573.0|(63,[0,1,5,6,8,24],[3.0,9.0,1.0,1.0,1.0,1.0])|  
|86390.0 |(63,[5,6,9,17],[1.0,1.0,1.0,1.0]) |  
|86390.0 |(63,[0,1,5,6,8,23],[3.0,9.0,1.0,1.0,1.0,1.0])|  
|52987.0 |(63,[5,6,8,17],[1.0,1.0,1.0,1.0]) |  
|86390.0 |(63,[5,7,9,21],[1.0,1.0,1.0,1.0]) |  
+--------+---------------------------------------------+  
only showing top 5 rows

#Test Train Split  
train\_poly\_data, test\_poly\_data = poly\_data.randomSplit([0.8, 0.2], seed=45)

poly\_feature\_names = assembler\_poly.getInputCols()  
  
poly\_reg\_min\_yrs = GeneralizedLinearRegression(  
 featuresCol="features\_poly",  
 labelCol="SALARY",  
 family="gaussian",  
 link="identity",  
 maxIter=10,  
 regParam=0.3,  
)  
  
poly\_yrs\_model = poly\_reg\_min\_yrs.fit(train\_poly\_data)  
poly\_yrs\_summary = poly\_yrs\_model.summary

#Model Summary  
  
print("Intercept: {:.4f}".format(poly\_yrs\_model.intercept))  
print("Coefficient Standard Errors:", [f"{val:.4f}" for val in poly\_yrs\_summary.coefficientStandardErrors])  
print("T Values:", [f"{val:.4f}" for val in poly\_yrs\_summary.tValues])  
print("P Values:", [f"{val:.4f}" for val in poly\_yrs\_summary.pValues])  
print(f"Null Deviance: {poly\_yrs\_summary.nullDeviance:.4f}")  
print(f"Residual DF Null: {poly\_yrs\_summary.residualDegreeOfFreedomNull}")  
print(f"Residual DF: {poly\_yrs\_summary.residualDegreeOfFreedom}")  
print(f"AIC: {poly\_yrs\_summary.aic:.4f}")  
print(f"Deviance: {poly\_yrs\_summary.deviance:.4f}")

Intercept: 102770.4046

Coefficient Standard Errors: ['89.6343', '7.7088', '345.2028', '761.6878', '997.1704', '1181.6420', '666.9488', '710.4609', '2934.0575', '2935.7966', '2972.0945', '2983.4991', '3015.2172', '3100.7369', '3102.9532', '3122.4368', '3123.0461', '3123.8888', '3126.9238', '3136.8676', '3138.3902', '3140.4516', '3140.4866', '3149.9636', '3154.2721', '3164.5396', '3174.8576', '3174.0643', '3185.1746', '3185.9610', '3189.6462', '3199.1404', '3202.4498', '3205.5803', '3224.3000', '3225.1381', '3241.3518', '3254.4646', '3278.0500', '3304.2059', '3314.1859', '3311.9843', '3320.7134', '3320.4628', '3331.2482', '3336.5663', '3328.7495', '3361.6903', '3399.3659', '3397.0262', '3392.2679', '3398.8985', '3420.0516', '3498.9091', '3557.2962', '3593.3572', '3632.2547', '3645.0567', '3708.3337', '3715.0051', '3822.3433', '3876.0671', '4030.7522', '4393.4332']  
T Values: ['16.5970', '10.6460', '-8.1069', '-5.5854', '12.0973', '-7.1567', '2.3084', '1.7295', '-1.7378', '-1.1709', '-7.3683', '-4.1078', '2.2659', '-0.5332', '0.8185', '-0.7476', '0.1061', '0.1496', '-0.0848', '-0.3999', '-0.5932', '-0.7565', '0.4462', '-0.7038', '-0.0568', '-0.2512', '-0.8600', '0.9841', '-0.4351', '-1.2077', '-0.8587', '-0.6250', '-0.4927', '-0.2427', '-1.0583', '-0.3307', '-0.3425', '0.8803', '-0.7466', '-0.7189', '-0.3232', '-1.3996', '-0.5853', '-0.4848', '-0.3396', '-2.0221', '0.9076', '-0.5224', '-0.6373', '-0.9810', '-0.2004', '-0.0946', '-0.0051', '-1.0468', '-1.1664', '-2.2112', '-1.2470', '-2.7276', '-2.0138', '0.8537', '0.2298', '-1.9459', '-2.7340', '23.3918']  
P Values: ['0.0000', '0.0000', '0.0000', '0.0000', '0.0000', '0.0000', '0.0210', '0.0837', '0.0823', '0.2417', '0.0000', '0.0000', '0.0235', '0.5939', '0.4131', '0.4547', '0.9155', '0.8810', '0.9324', '0.6892', '0.5531', '0.4494', '0.6554', '0.4815', '0.9547', '0.8017', '0.3898', '0.3251', '0.6635', '0.2272', '0.3905', '0.5320', '0.6222', '0.8082', '0.2899', '0.7409', '0.7320', '0.3787', '0.4553', '0.4722', '0.7465', '0.1616', '0.5583', '0.6278', '0.7341', '0.0432', '0.3641', '0.6014', '0.5239', '0.3266', '0.8412', '0.9246', '0.9959', '0.2952', '0.2435', '0.0270', '0.2124', '0.0064', '0.0440', '0.3932', '0.8183', '0.0517', '0.0063', '0.0000']

[Stage 116:> (0 + 1) / 1]

Null Deviance: 51607738345410.0703  
Residual DF Null: 57956  
Residual DF: 57893

[Stage 123:> (0 + 1) / 1]

AIC: 1349984.3354  
Deviance: 44219958881865.4297

poly\_feature\_names = poly\_yrs\_summary.\_call\_java("featureNames")  
poly\_features = ["Intercept"] + poly\_feature\_names  
poly\_coefs = [poly\_yrs\_model.intercept] + list(poly\_yrs\_model.coefficients)  
poly\_se = list(poly\_yrs\_summary.coefficientStandardErrors)  
poly\_t\_values = list(poly\_yrs\_summary.tValues)  
poly\_p\_values = list(poly\_yrs\_summary.pValues)  
  
poly\_summary\_df = pd.DataFrame({  
 "Feature": poly\_features,  
 "Estimate": [f"{v:.4f}" if v is not None else None for v in poly\_coefs],  
 "Std. Error": [f"{v:.4f}" if v is not None else None for v in poly\_se],  
 "t-value": [f"{v:.4f}" if v is not None else None for v in poly\_t\_values],  
 "p-value": [f"{v:.4f}" if v is not None else None for v in poly\_p\_values]  
})  
poly\_summary\_df.to\_csv("output/poly\_model\_summary.csv", index=False)  
poly\_summary\_df

|  | Feature | Estimate | Std. Error | t-value | p-value |
| --- | --- | --- | --- | --- | --- |
| 0 | Intercept | 102770.4046 | 89.6343 | 16.5970 | 0.0000 |
| 1 | MIN\_YEARS\_EXPERIENCE | 1487.6582 | 7.7088 | 10.6460 | 0.0000 |
| 2 | MIN\_YEARS\_EXPERIENCE\_sq | 82.0679 | 345.2028 | -8.1069 | 0.0000 |
| 3 | COMPANY\_IS\_STAFFING | -2798.5163 | 761.6878 | -5.5854 | 0.0000 |
| 4 | IS\_INTERNSHIP | -4254.3237 | 997.1704 | 12.0973 | 0.0000 |
| ... | ... | ... | ... | ... | ... |
| 59 | STATE\_NAME\_vec\_Alaska | -7467.6895 | 3715.0051 | 0.8537 | 0.3932 |
| 60 | STATE\_NAME\_vec\_Vermont | 3171.6806 | 3822.3433 | 0.2298 | 0.8183 |
| 61 | STATE\_NAME\_vec\_Montana | 878.2150 | 3876.0671 | -1.9459 | 0.0517 |
| 62 | STATE\_NAME\_vec\_West Virginia | -7542.6035 | 4030.7522 | -2.7340 | 0.0063 |
| 63 | STATE\_NAME\_vec\_North Dakota | -11020.2597 | 4393.4332 | 23.3918 | 0.0000 |

#Create a numeric summary DataFrame for further analysis  
poly\_summary\_numeric = pd.DataFrame({  
 "Feature": poly\_features,  
 "Estimate": poly\_coefs,  
 "Std\_Error": poly\_se,  
 "t\_value": poly\_t\_values,  
 "p\_value": poly\_p\_values  
})  
  
# Add significance flag  
poly\_summary\_numeric["Significant"] = poly\_summary\_numeric["p\_value"] < 0.05  
  
# Sort by absolute t-value  
poly\_summary\_numeric = poly\_summary\_numeric.reindex(poly\_summary\_numeric["t\_value"].abs().sort\_values(ascending=False).index)  
poly\_summary\_numeric.head(10)

|  | Feature | Estimate | Std\_Error | t\_value | p\_value | Significant |
| --- | --- | --- | --- | --- | --- | --- |
| 63 | STATE\_NAME\_vec\_North Dakota | -11020.259746 | 4393.433199 | 23.391821 | 0.000000e+00 | True |
| 0 | Intercept | 102770.404563 | 89.634274 | 16.596980 | 0.000000e+00 | True |
| 4 | IS\_INTERNSHIP | -4254.323692 | 997.170407 | 12.097298 | 0.000000e+00 | True |
| 1 | MIN\_YEARS\_EXPERIENCE | 1487.658219 | 7.708798 | 10.646001 | 0.000000e+00 | True |
| 2 | MIN\_YEARS\_EXPERIENCE\_sq | 82.067877 | 345.202790 | -8.106876 | 4.440892e-16 | True |
| 10 | MIN\_EDULEVELS\_NAME\_vec\_No Education Listed | -3437.400514 | 2972.094524 | -7.368299 | 1.751932e-13 | True |
| 5 | EMPLOYMENT\_TYPE\_NAME\_vec\_Full Time | 12063.067849 | 1181.641990 | -7.156699 | 8.362200e-13 | True |
| 3 | COMPANY\_IS\_STAFFING | -2798.516307 | 761.687811 | -5.585390 | 2.342199e-08 | True |
| 11 | MIN\_EDULEVELS\_NAME\_vec\_High school or GED | -21899.280992 | 2983.499125 | -4.107798 | 3.999975e-05 | True |
| 62 | STATE\_NAME\_vec\_West Virginia | -7542.603506 | 4030.752155 | -2.734045 | 6.258041e-03 | True |

Analysis: The polynominal regression has slightly different values. Only min years experience really changed which is the variable I squared in this model.

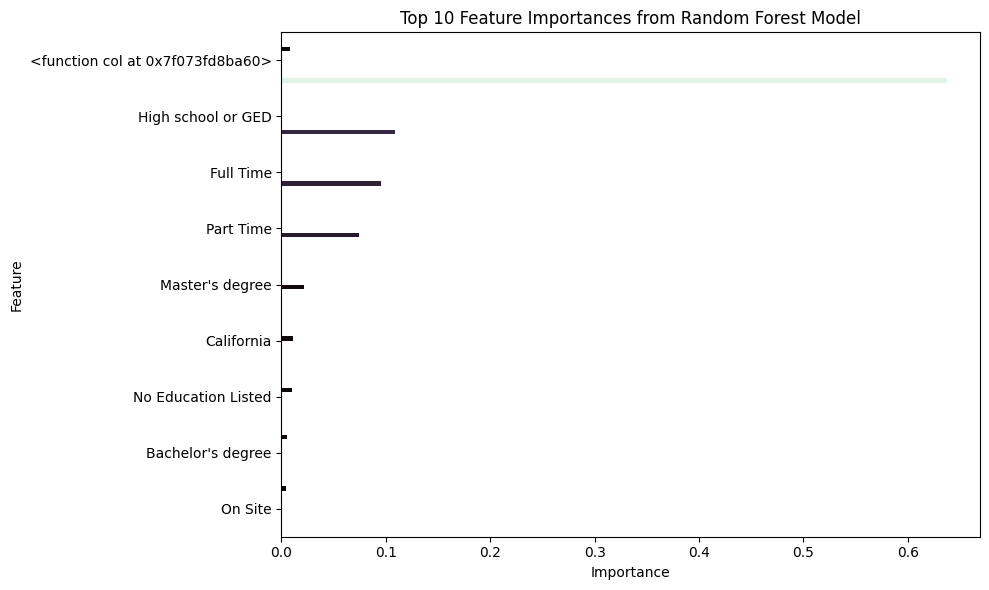
# Random Forest

from pyspark.ml.regression import RandomForestRegressor  
  
rf = RandomForestRegressor(featuresCol="features", labelCol="SALARY", numTrees=200, maxDepth=4, seed=45)  
rf\_model = rf.fit(train\_data.select("features","SALARY"))  
rf\_preds = rf\_model.transform(test\_data.select("features","SALARY"))

WARNING: An illegal reflective access operation has occurred   
WARNING: Illegal reflective access by org.apache.spark.util.SizeEstimator$ (file:/opt/spark-3.5.6-bin-hadoop3/jars/spark-core\_2.12-3.5.6.jar) to field java.nio.charset.Charset.name  
WARNING: Please consider reporting this to the maintainers of org.apache.spark.util.SizeEstimator$  
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations  
WARNING: All illegal access operations will be denied in a future release

import matplotlib.pyplot as plt  
import seaborn as sns  
  
#Feature Importance  
def get\_actual\_feature\_names(data, assembler, encoded\_cols):  
 full\_feature\_names = []  
 for col\_name in assembler.getInputCols():  
 if col\_name in encoded\_cols:  
 try:  
 attr\_meta = data.schema[col\_name].metadata["ml\_attr"]["attrs"]  
 for attr\_group in attr\_meta.values():  
 for attr in attr\_group:  
 full\_feature\_names.append(attr["name"])  
 except:  
 full\_feature\_names.append(col\_name)  
 else:  
 full\_feature\_names.append(col)  
 return full\_feature\_names  
  
encoded\_cols = [f"{col}\_vec" for col in categorical\_cols]  
feature\_names = get\_actual\_feature\_names(data, assembler, encoded\_cols)  
importances = rf\_model.featureImportances.toArray()

def clean\_feature\_names(feature\_list):  
 clean\_names = []  
 for name in feature\_list:  
 if isinstance(name, list):  
 clean\_names.append("\_".join(str(n) for n in name))  
 elif isinstance(name, str) and name.startswith("["):  
 clean\_names.append(name.replace("[","").replace("]","").replace("'","").replace('"','').strip())  
 else:  
 clean\_names.append(str(name))  
 return clean\_names  
  
importance\_df = pd.DataFrame({  
 "Feature": feature\_names,  
 "Importance": importances  
}).sort\_values(by="Importance", ascending=False)  
  
importance\_df["Feature"] = clean\_feature\_names(importance\_df["Feature"])  
  
plt.figure(figsize=(10,6))  
sns.barplot(data=importance\_df.head(10), x="Importance", y="Feature", hue="Importance", palette="mako", legend=False)  
plt.title("Top 10 Feature Importances from Random Forest Model")  
plt.xlabel("Importance")  
plt.ylabel("Feature")  
plt.tight\_layout()  
plt.savefig("output/rf\_feature\_importance.png")  
plt.show()



# Compare Models

from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.sql.functions import col, pow, sqrt, avg  
import numpy as np  
  
#Generate predications for poly and linear  
poly\_preds = poly\_yrs\_model.transform(test\_poly\_data)  
glr\_preds = glr\_model.transform(test\_data)  
rf\_preds = rf\_model.transform(test\_data)  
  
evaluatorR2 = RegressionEvaluator(labelCol="SALARY", predictionCol="prediction", metricName="r2")  
residuals\_df = glr\_summary.predictions.withColumn("squared\_errors", pow(col("SALARY") - col("prediction"), 2))  
  
glr\_df = glr\_summary.predictions.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "GLR\_Prediction"})  
glr\_r2 = evaluatorR2.evaluate(glr\_summary.predictions)  
glr\_rmse = np.sqrt(residuals\_df.select(avg("squared\_errors")).first()[0])  
glr\_aic = glr\_summary.aic  
glr\_bic = len(glr\_summary.coefficients) \* np.log(glr\_summary.numInstances) + glr\_summary.numInstances \* np.log(glr\_summary.deviance / glr\_summary.numInstances)  
  
poly\_df = poly\_preds.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "Poly\_Prediction"})  
poly\_r2 = evaluatorR2.evaluate(poly\_preds)  
poly\_rmse = np.sqrt(((poly\_data["SALARY"] - poly\_data["Poly\_Prediction"]) \*\* 2).mean())  
poly\_aic = None  
poly\_bic = None  
  
rf\_df = rf\_preds.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "RF\_Prediction"})  
rf\_r2 = evaluatorR2.evaluate(rf\_preds)  
rf\_rmse = np.sqrt(((rf\_df["SALARY"] - rf\_df["RF\_Prediction"]) \*\* 2).mean())  
rf\_aic = None  
rf\_bic = None

AttributeError: 'GeneralizedLinearRegressionTrainingSummary' object has no attribute 'coefficients'  
[31m---------------------------------------------------------------------------[39m  
[31mAttributeError[39m Traceback (most recent call last)  
[36mCell[39m[36m [39m[32mIn[23][39m[32m, line 17[39m  
[32m 15[39m glr\_rmse = np.sqrt(residuals\_df.select(avg([33m"[39m[33msquared\_errors[39m[33m"[39m)).first()[[32m0[39m])  
[32m 16[39m glr\_aic = glr\_summary.aic  
[32m---> [39m[32m17[39m glr\_bic = [38;5;28mlen[39m([43mglr\_summary[49m[43m.[49m[43mcoefficients[49m) \* np.log(glr\_summary.numInstances) + glr\_summary.numInstances \* np.log(glr\_summary.deviance / glr\_summary.numInstances)  
[32m 19[39m poly\_df = poly\_preds.select([33m"[39m[33mSALARY[39m[33m"[39m, [33m"[39m[33mprediction[39m[33m"[39m).toPandas().rename(columns={[33m"[39m[33mprediction[39m[33m"[39m: [33m"[39m[33mPoly\_Prediction[39m[33m"[39m})  
[32m 20[39m poly\_r2 = evaluatorR2.evaluate(poly\_preds)  
  
[31mAttributeError[39m: 'GeneralizedLinearRegressionTrainingSummary' object has no attribute 'coefficients'

from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.sql.functions import col, pow, sqrt, avg  
import numpy as np  
  
# Generate predictions  
poly\_preds = poly\_yrs\_model.transform(test\_poly\_data)  
glr\_preds = glr\_model.transform(test\_data)  
rf\_preds = rf\_model.transform(test\_data)  
  
# Metrics for GLR  
evaluatorR2 = RegressionEvaluator(labelCol="SALARY", predictionCol="prediction", metricName="r2")  
residuals\_df = glr\_summary.predictions.withColumn("squared\_errors", pow(col("SALARY") - col("prediction"), 2))  
  
glr\_df = glr\_summary.predictions.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "GLR\_Prediction"})  
glr\_r2 = evaluatorR2.evaluate(glr\_summary.predictions)  
glr\_rmse = np.sqrt(residuals\_df.select(avg("squared\_errors")).first()[0])  
glr\_aic = glr\_summary.aic  
glr\_bic = len(glr\_model.coefficients) \* np.log(glr\_summary.numInstances) + \  
 glr\_summary.numInstances \* np.log(glr\_summary.deviance / glr\_summary.numInstances)  
  
# Metrics for Polynomial Model  
poly\_df = poly\_preds.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "Poly\_Prediction"})  
poly\_r2 = evaluatorR2.evaluate(poly\_preds)  
poly\_rmse = np.sqrt(((poly\_df["SALARY"] - poly\_df["Poly\_Prediction"]) \*\* 2).mean())  
poly\_aic = None  
poly\_bic = None  
  
# Metrics for Random Forest  
rf\_df = rf\_preds.select("SALARY", "prediction").toPandas().rename(columns={"prediction": "RF\_Prediction"})  
rf\_r2 = evaluatorR2.evaluate(rf\_preds)  
rf\_rmse = np.sqrt(((rf\_df["SALARY"] - rf\_df["RF\_Prediction"]) \*\* 2).mean())  
rf\_aic = None  
rf\_bic = None

#Plot  
merged = glr\_df.copy()  
merged["Polynomial"] = poly\_df["Poly\_Prediction"]  
merged["Random\_Forest"] = rf\_df["RF\_Prediction"]  
  
#Melt  
plot\_df = merged.melt(id\_vars="SALARY", var\_name="Model", value\_name="Predicted")  
plot\_df.head()

|  | SALARY | Model | Predicted |
| --- | --- | --- | --- |
| 0 | 20800.0 | GLR\_Prediction | 79700.536267 |
| 1 | 24960.0 | GLR\_Prediction | 97928.452843 |
| 2 | 24960.0 | GLR\_Prediction | 98416.110750 |
| 3 | 24960.0 | GLR\_Prediction | 103947.174075 |
| 4 | 25480.0 | GLR\_Prediction | 99272.377344 |

#Plot  
plt.figure(figsize=(7,16.5))  
sns.set\_style("whitegrid")  
  
models = {  
 "GLR": (glr\_rmse,glr\_r2,glr\_aic,glr\_bic),  
 "Polynomial": (poly\_rmse,poly\_r2,"NA","NA"),  
 "RandomForest": (rf\_rmse,rf\_r2,"NA","NA")  
}  
  
for idx, model in enumerate(models.keys(), 1):  
 plt.subplot(3, 1, idx)  
 temp\_df = plot\_df[plot\_df["Model"] == model]  
 sns.scatterplot(data=temp\_df, x="SALARY", y="Predicted", alpha=0.5, label=model)  
 sns.lineplot(x=temp\_df["SALARY"], y=temp\_df["SALARY"], color='red', label='Ideal Fit')  
   
 rmse, r2, aic, bic = models[model]  
 plt.title(f"{model} Model: RMSE={models[model][0]:.2f}, R²={models[model][1]:.4f}, AIC={models[model][2]}, BIC={models[model][3]}")  
 plt.xlabel("Actual Salary")  
 plt.ylabel("Predicted Salary")  
 plt.grid(True, linestyle='--', alpha=0.5, linewidth=0.75,color='#676767')  
  
plt.tight\_layout()  
plt.savefig("output/model\_comparison.png")  
plt.show()

