Assignment 04

Sabrina Minaya Vasquez (Boston University)  
October 8, 2025

# 1. Load Dataset

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(42)  
  
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("/home/ubuntu/assignment-04-Sabrina1211/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()   
#df.show(5)

[Stage 402:> (0 + 1) / 1]

---This is Diagnostic check, No need to print it in the final doc---

# Missing Value Treatment  
  
from pyspark.sql import Window   
from pyspark.sql import functions as F  
from pyspark.sql.functions import col, when, isnan, count, expr  
  
# 1) Overall median salary  
overall\_median\_salary = df.approxQuantile("SALARY", [0.5], 0.01)[0]  
  
# 2) Median salary by EMPLOYMENT\_TYPE  
median\_by\_employment\_type = (  
 df.groupBy("EMPLOYMENT\_TYPE")  
 .agg(expr("percentile\_approx(SALARY, 0.5)").alias("median\_salary\_emp\_type"))  
)  
  
# 3) Median salary by EMPLOYMENT\_TYPE\_NAME  
median\_by\_employment\_type\_name = (  
 df.groupBy("EMPLOYMENT\_TYPE\_NAME")  
 .agg(expr("percentile\_approx(SALARY, 0.5)").alias("median\_salary\_emp\_type\_name"))  
)  
  
# 4) Impute SALARY: prefer EMPLOYMENT\_TYPE median, then EMPLOYMENT\_TYPE\_NAME median, else overall median  
df\_salary\_imputed = (  
 df.join(median\_by\_employment\_type, on="EMPLOYMENT\_TYPE", how="left")  
 .join(median\_by\_employment\_type\_name, on="EMPLOYMENT\_TYPE\_NAME", how="left")  
 .withColumn(  
 "SALARY",  
 F.when(  
 col("SALARY").isNull(),  
 F.coalesce(  
 col("median\_salary\_emp\_type"),  
 col("median\_salary\_emp\_type\_name"),  
 F.lit(overall\_median\_salary)  
 )  
 ).otherwise(col("SALARY"))  
 )  
 .drop("median\_salary\_emp\_type", "median\_salary\_emp\_type\_name")  
)  
  
# Join median values back to the original dataframe  
df\_salary\_imputed = (  
 df.join(median\_by\_employment\_type, on="EMPLOYMENT\_TYPE", how="left")  
 .join(median\_by\_employment\_type\_name, on="EMPLOYMENT\_TYPE\_NAME", how="left")  
)  
  
# Replace missing SALARY values  
df\_salary\_imputed = df\_salary\_imputed.withColumn(  
 "SALARY",  
 when(col("SALARY").isNull(),  
 when(col("median\_salary\_emp\_type").isNotNull(), col("median\_salary\_emp\_type"))  
 .when(col("median\_salary\_emp\_type\_name").isNotNull(), col("median\_salary\_emp\_type\_name"))  
 .otherwise(overall\_median\_salary)  
 ).otherwise(col("SALARY"))  
)  
  
#df\_salary\_imputed.show(5)

# 2. Feature Engineering

+--------+--------------------+--------------------+-----------------------------+----------------------+----------------+--------+-------------+-------------------+----------------------+---------------------------+  
|SALARY |MIN\_YEARS\_EXPERIENCE|MAX\_YEARS\_EXPERIENCE|EDUCATION\_LEVELS\_NAME |EMPLOYMENT\_TYPE\_NAME |REMOTE\_TYPE\_NAME|DURATION|IS\_INTERNSHIP|COMPANY\_IS\_STAFFING|median\_salary\_emp\_type|median\_salary\_emp\_type\_name|  
+--------+--------------------+--------------------+-----------------------------+----------------------+----------------+--------+-------------+-------------------+----------------------+---------------------------+  
|116500.0|2 |2 |[\n "Bachelor's degree"\n] |Full-time (> 32 hours)|[None] |6 |0 |0 |116500 |116500 |  
|116500.0|7 |7 |[\n "No Education Listed"\n]|Full-time (> 32 hours)|[None] |18 |0 |1 |116500 |116500 |  
|116500.0|1 |1 |[\n "No Education Listed"\n]|Full-time (> 32 hours)|[None] |8 |0 |1 |116500 |116500 |  
|116500.0|1 |1 |[\n "Bachelor's degree"\n] |Full-time (> 32 hours)|[None] |32 |0 |0 |116500 |116500 |  
|131100.0|2 |2 |[\n "Bachelor's degree"\n] |Full-time (> 32 hours)|[None] |11 |0 |0 |116500 |116500 |  
+--------+--------------------+--------------------+-----------------------------+----------------------+----------------+--------+-------------+-------------------+----------------------+---------------------------+  
only showing top 5 rows

# Clean Education Levels by cleaning \n and array brackets  
from pyspark.sql.functions import regexp\_replace, trim  
  
regression\_df = regression\_df.withColumn(  
 "EDUCATION\_LEVELS\_NAME",  
 trim(regexp\_replace(col("EDUCATION\_LEVELS\_NAME"), r"[\[\]\n]", ""))  
)  
  
# Index and One-Hot Encode  
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  
]  
  
# Assemble base features (for GLR and Random Forest)  
assembler = VectorAssembler(  
 inputCols=[  
 "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE", "DURATION"  
 ] + [f"{col}\_vec" for col in categorical\_cols],  
 outputCol="features"  
)  
  
pipeline = Pipeline(stages=indexers + encoders + [assembler])  
regression\_data = pipeline.fit(regression\_df).transform(regression\_df)  
  
# Show final features structure  
regression\_data.select("SALARY", "features").show(5, truncate=False)

+--------+-------------------------------------------------------------+  
|SALARY |features |  
+--------+-------------------------------------------------------------+  
|116500.0|(28,[0,1,2,3,21,23,26,27],[2.0,2.0,6.0,1.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(28,[0,1,2,4,21,23,26],[7.0,7.0,18.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(28,[0,1,2,4,21,23,26],[1.0,1.0,8.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(28,[0,1,2,3,21,23,26,27],[1.0,1.0,32.0,1.0,1.0,1.0,1.0,1.0])|  
|131100.0|(28,[0,1,2,3,21,23,26,27],[2.0,2.0,11.0,1.0,1.0,1.0,1.0,1.0])|  
+--------+-------------------------------------------------------------+  
only showing top 5 rows

# 3. Train/Test Split

I used an 80/20 train–test split. The 80% gives the model enough data to learn stable patterns, while the remaining 20% is a clean hold-out to evaluate how well it generalizes to unseen jobs. This ratio is a common default for regression problems because it balances learning and evaluation without wasting data. I also set a seed (42) so the split is reproducible.

(5039, 22)

(4070, 22)

[Stage 103:> (0 + 1) / 1]

(969, 22)

# 4. Linear Regression

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

25/10/05 21:34:11 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.VectorBLAS

# Coefficients and Intercept  
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: 83684.2103  
Coefficients:  
Feature 1: 1596.5795  
Feature 2: 1596.5795  
Feature 3: 31.1901  
Feature 4: 2943.3549  
Feature 5: 5624.8675  
Feature 6: 11303.5572  
Feature 7: -25678.5645  
Feature 8: 14668.1325  
Feature 9: -9752.7189  
Feature 10: -118.7467  
Feature 11: 13480.7278  
Feature 12: -9277.3038  
Feature 13: 3310.0724  
Feature 14: -14782.2689  
Feature 15: 35678.4989  
Feature 16: -8064.0197  
Feature 17: -2793.9409  
Feature 18: 17915.8367  
Feature 19: 8358.0949  
Feature 20: -35699.8382  
Feature 21: 32498.9245  
Feature 22: 6376.6043  
Feature 23: -5663.4155  
Feature 24: 4176.7678  
Feature 25: 8503.4618  
Feature 26: 2741.9488  
Feature 27: -765.8130  
Feature 28: -111.3548

# Summary stats  
print("\n--- Regression Summary ---")  
print("Coefficient Standard Errors:", [f"{val:.4f}" for val in summary.coefficientStandardErrors])  
print("T-Values:", [f"{val:.4f}" for val in summary.tValues])  
print("P-Values:", [f"{val:.4f}" for val in summary.pValues])

--- Regression Summary ---

[Stage 125:> (0 + 1) / 1]

Coefficient Standard Errors: ['22889.3361', '22889.3361', '20.7234', '20967.0009', '20980.9421', '20982.8316', '21094.7761', '21104.6267', '21115.7024', '21143.7509', '21221.3547', '21327.2560', '21500.7656', '21553.1712', '21698.8688', '21921.5499', '22028.5324', '22141.1180', '22282.5601', '24311.4028', '24305.5662', '3075.8265', '3587.0214', '2925.8458', '2988.0317', '3375.9809', '4590.5476', '825.5726', '21910.9169']  
T-Values: ['0.0698', '0.0698', '1.5051', '0.1404', '0.2681', '0.5387', '-1.2173', '0.6950', '-0.4619', '-0.0056', '0.6352', '-0.4350', '0.1540', '-0.6859', '1.6443', '-0.3679', '-0.1268', '0.8092', '0.3751', '-1.4684', '1.3371', '2.0731', '-1.5789', '1.4275', '2.8458', '0.8122', '-0.1668', '-0.1349', '3.8193']  
P-Values: ['0.9444', '0.9444', '0.1324', '0.8884', '0.7886', '0.5901', '0.2235', '0.4871', '0.6442', '0.9955', '0.5253', '0.6636', '0.8777', '0.4928', '0.1002', '0.7130', '0.8991', '0.4185', '0.7076', '0.1420', '0.1813', '0.0382', '0.1144', '0.1535', '0.0044', '0.4167', '0.8675', '0.8927', '0.0001']

# print(f"\nDispersion: {summary.dispersion:.4f}")  
print(f"Null Deviance: {summary.nullDeviance:.4f}")  
print(f"Residual DF Null: {summary.residualDegreeOfFreedomNull}")  
print(f"Deviance: {summary.deviance:.4f}")  
print(f"Residual DF: {summary.residualDegreeOfFreedom}")  
print(f"AIC: {summary.aic:.4f}")

[Stage 132:> (0 + 1) / 1]

Null Deviance: 2909123659016.3213  
Residual DF Null: 5038  
Deviance: 2272959595356.6333  
Residual DF: 5010

[Stage 143:> (0 + 1) / 1]

AIC: 114772.9258

# 1. Pull feature names directly from Java backend  
feature\_names = summary.\_call\_java("featureNames")  
  
# 2. Construct full table including intercept  
features = ["Intercept"] + list(feature\_names)  
coefs = [glr\_model.intercept] + list(glr\_model.coefficients)  
se = list(summary.coefficientStandardErrors)  
tvals = list(summary.tValues)  
pvals = list(summary.pValues)  
  
# (Optional) quick diagnostics  
print("--- This is a diagnostic check, no need to print in the final doc ---")  
print("Length of features:", len(features))  
print("Length of coefs:", len(coefs))  
print("Length of se:", len(se))  
print("Length of tvals:", len(tvals))  
print("Length of pvals:", len(pvals))

--- This is a diagnostic check, no need to print in the final doc ---  
Length of features: 29  
Length of coefs: 29  
Length of se: 29  
Length of tvals: 29  
Length of pvals: 29

# 4.1 Generalized Linear Regression Summary

import pandas as pd  
from tabulate import tabulate  
from IPython.display import HTML  
  
coef\_table = 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-stat": [f"{v:.4f}" if v is not None else None for v in tvals],  
 "P-Value": [f"{v:.4f}" if v is not None else None for v in pvals]  
})  
  
# 4. Save for report  
coef\_table.to\_csv("output/glr\_summary.csv", index=False)  
  
# 5. Optional pretty print  
HTML(coef\_table.to\_html())

|  | Feature | Estimate | Std Error | t-stat | P-Value |
| --- | --- | --- | --- | --- | --- |
| 0 | Intercept | 83684.2103 | 22889.3361 | 0.0698 | 0.9444 |
| 1 | MIN\_YEARS\_EXPERIENCE | 1596.5795 | 22889.3361 | 0.0698 | 0.9444 |
| 2 | MAX\_YEARS\_EXPERIENCE | 1596.5795 | 20.7234 | 1.5051 | 0.1324 |
| 3 | DURATION | 31.1901 | 20967.0009 | 0.1404 | 0.8884 |
| 4 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree" | 2943.3549 | 20980.9421 | 0.2681 | 0.7886 |
| 5 | EDUCATION\_LEVELS\_NAME\_vec\_"No Education Listed" | 5624.8675 | 20982.8316 | 0.5387 | 0.5901 |
| 6 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Master's degree" | 11303.5572 | 21094.7761 | -1.2173 | 0.2235 |
| 7 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Bachelor's degree" | -25678.5645 | 21104.6267 | 0.6950 | 0.4871 |
| 8 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Master's degree", "Ph.D. or professional degree" | 14668.1325 | 21115.7024 | -0.4619 | 0.6442 |
| 9 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED" | -9752.7189 | 21143.7509 | -0.0056 | 0.9955 |
| 10 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree" | -118.7467 | 21221.3547 | 0.6352 | 0.5253 |
| 11 | EDUCATION\_LEVELS\_NAME\_vec\_"Master's degree" | 13480.7278 | 21327.2560 | -0.4350 | 0.6636 |
| 12 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree" | -9277.3038 | 21500.7656 | 0.1540 | 0.8777 |
| 13 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree", "Master's degree" | 3310.0724 | 21553.1712 | -0.6859 | 0.4928 |
| 14 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree", "Master's degree", "Ph.D. or professional degree" | -14782.2689 | 21698.8688 | 1.6443 | 0.1002 |
| 15 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Ph.D. or professional degree" | 35678.4989 | 21921.5499 | -0.3679 | 0.7130 |
| 16 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree", "Bachelor's degree" | -8064.0197 | 22028.5324 | -0.1268 | 0.8991 |
| 17 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Bachelor's degree", "Master's degree" | -2793.9409 | 22141.1180 | 0.8092 | 0.4185 |
| 18 | EDUCATION\_LEVELS\_NAME\_vec\_"Master's degree", "Ph.D. or professional degree" | 17915.8367 | 22282.5601 | 0.3751 | 0.7076 |
| 19 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree" | 8358.0949 | 24311.4028 | -1.4684 | 0.1420 |
| 20 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree", "Bachelor's degree", "Master's degree" | -35699.8382 | 24305.5662 | 1.3371 | 0.1813 |
| 21 | EDUCATION\_LEVELS\_NAME\_vec\_"Ph.D. or professional degree" | 32498.9245 | 3075.8265 | 2.0731 | 0.0382 |
| 22 | EMPLOYMENT\_TYPE\_NAME\_vec\_Full-time (> 32 hours) | 6376.6043 | 3587.0214 | -1.5789 | 0.1144 |
| 23 | EMPLOYMENT\_TYPE\_NAME\_vec\_Part-time (â‰¤ 32 hours) | -5663.4155 | 2925.8458 | 1.4275 | 0.1535 |
| 24 | REMOTE\_TYPE\_NAME\_vec\_[None] | 4176.7678 | 2988.0317 | 2.8458 | 0.0044 |
| 25 | REMOTE\_TYPE\_NAME\_vec\_Remote | 8503.4618 | 3375.9809 | 0.8122 | 0.4167 |
| 26 | REMOTE\_TYPE\_NAME\_vec\_Hybrid Remote | 2741.9488 | 4590.5476 | -0.1668 | 0.8675 |
| 27 | IS\_INTERNSHIP\_vec\_0 | -765.8130 | 825.5726 | -0.1349 | 0.8927 |
| 28 | COMPANY\_IS\_STAFFING\_vec\_0 | -111.3548 | 21910.9169 | 3.8193 | 0.0001 |

# Linear Regreassion Model Interpretation

In my linear regression model, each coefficient represents how much the predicted salary changes when that feature increases by one unit, assuming all other variables stay constant. Positive coefficients indicate that higher values of that feature increase salary, while negative coefficients reduce it.

Looking at the t-values and p-values, only a few variables show statistical significance (p < 0.05), meaning most predictors do not have a strong linear relationship with salary on their own. The features with smaller p-values (for example, those around 0.04 or below) contribute more meaningfully to the model’s predictions.

The model’s overall performance shows moderate predictive power; its (R)^2 value indicates it explains part of the salary variation but leaves room for non linear effects or missing factors. The RMSE (around $21,796) shows that predicted salaries differ from actual values by about $22,000 on average, which is acceptable for exploratory modeling but suggests more complex models (like polynomial or random forest) can improve accuracy.

# 5. Polymnomial Regression

+--------+-------------------------------------------------------------------+  
|SALARY |features\_poly |  
+--------+-------------------------------------------------------------------+  
|116500.0|(29,[0,1,2,3,4,22,24,27,28],[2.0,4.0,2.0,6.0,1.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(29,[0,1,2,3,5,22,24,27],[7.0,49.0,7.0,18.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(29,[0,1,2,3,5,22,24,27],[1.0,1.0,1.0,8.0,1.0,1.0,1.0,1.0]) |  
|116500.0|(29,[0,1,2,3,4,22,24,27,28],[1.0,1.0,1.0,32.0,1.0,1.0,1.0,1.0,1.0])|  
|131100.0|(29,[0,1,2,3,4,22,24,27,28],[2.0,4.0,2.0,11.0,1.0,1.0,1.0,1.0,1.0])|  
+--------+-------------------------------------------------------------------+  
only showing top 5 rows

# Split Data  
polyreg\_train, polyreg\_test = poly\_data.randomSplit([0.8, 0.2], seed=42)  
  
print((poly\_data.count(), len(poly\_data.columns)))  
print((polyreg\_train.count(), len(polyreg\_train.columns)))  
print((polyreg\_test.count(), len(polyreg\_test.columns)))

(5039, 24)

(4070, 24)

[Stage 176:> (0 + 1) / 1]

(969, 24)

from pyspark.ml.regression import GeneralizedLinearRegression  
  
feature\_names = assembler.getInputCols()  
  
poly\_glr\_max\_years = GeneralizedLinearRegression(  
 featuresCol="features\_poly",  
 labelCol="SALARY",  
 family="gaussian", # normal distribution  
 link="identity", # standard linear regression  
 maxIter=10, # number of iterations for least squares  
 regParam=0.3 # regularization parameter (L2 regularization by default)  
)  
  
poly\_glr\_max\_years\_model = poly\_glr\_max\_years.fit(poly\_data)  
poly\_summary = poly\_glr\_max\_years\_model.summary

# Coefficients and Intercept  
print("Intercept: {:.4f}".format(poly\_glr\_max\_years\_model.intercept))  
print("Coefficients:")  
for i, coef in enumerate(poly\_glr\_max\_years\_model.coefficients):  
 print(f"Feature {i + 1}: {coef:.4f}")

Intercept: 87729.4985  
Coefficients:  
Feature 1: 3073.9396  
Feature 2: -292.0763  
Feature 3: 3073.9396  
Feature 4: 29.5828  
Feature 5: -2253.9775  
Feature 6: 805.2130  
Feature 7: 5881.4501  
Feature 8: -30439.8972  
Feature 9: 8412.8920  
Feature 10: -13680.3546  
Feature 11: -4660.4367  
Feature 12: 7967.9560  
Feature 13: -14214.1229  
Feature 14: -2857.2642  
Feature 15: -15252.4114  
Feature 16: 38851.8773  
Feature 17: -12908.6915  
Feature 18: -9159.5725  
Feature 19: 13889.7107  
Feature 20: 4123.7347  
Feature 21: -40022.9996  
Feature 22: 27224.9060  
Feature 23: 5759.7671  
Feature 24: -6588.6707  
Feature 25: 3972.6485  
Feature 26: 8353.6157  
Feature 27: 2403.5211  
Feature 28: -4220.3464  
Feature 29: 214.4410

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

--- Poly Summary ---

[Stage 204:> (0 + 1) / 1]

Coefficient Standard Errors: ['22747.6805', '36.5365', '22747.6805', '20.5955', '20846.6950', '20859.1237', '20863.3147', '20971.9940', '20987.9155', '20990.0817', '21019.8840', '21100.5976', '21203.5634', '21380.9216', '21419.1586', '21567.5225', '21793.5925', '21905.9591', '22009.1299', '22150.2630', '24166.2009', '24163.3570', '3057.6640', '3566.5832', '2907.7548', '2969.5007', '3355.2443', '4582.4088', '821.4479', '21780.4771']  
T-Values: ['0.1351', '-7.9941', '0.1351', '1.4364', '-0.1081', '0.0386', '0.2819', '-1.4515', '0.4008', '-0.6518', '-0.2217', '0.3776', '-0.6704', '-0.1336', '-0.7121', '1.8014', '-0.5923', '-0.4181', '0.6311', '0.1862', '-1.6562', '1.1267', '1.8837', '-1.8473', '1.3662', '2.8131', '0.7163', '-0.9210', '0.2611', '4.0279']  
P-Values: ['0.8925', '0.0000', '0.8925', '0.1510', '0.9139', '0.9692', '0.7780', '0.1467', '0.6886', '0.5146', '0.8245', '0.7057', '0.5027', '0.8937', '0.4764', '0.0717', '0.5537', '0.6759', '0.5280', '0.8523', '0.0978', '0.2599', '0.0597', '0.0648', '0.1719', '0.0049', '0.4738', '0.3571', '0.7941', '0.0001']

# print(f"\nDispersion: {summary.dispersion:.4f}")  
print(f"Null Deviance: {poly\_summary.nullDeviance:.4f}")  
print(f"Residual DF Null: {poly\_summary.residualDegreeOfFreedomNull}")  
print(f"Deviance: {poly\_summary.deviance:.4f}")  
print(f"Residual DF: {poly\_summary.residualDegreeOfFreedom}")  
print(f"AIC: {poly\_summary.aic:.4f}")

[Stage 216:> (0 + 1) / 1]

Null Deviance: 2909123659016.3213  
Residual DF Null: 5038  
Deviance: 2244317022136.5864  
Residual DF: 5009

[Stage 227:> (0 + 1) / 1]

AIC: 114711.0237

# 1. Pull feature names directly from Java backend  
feature\_names = poly\_summary.\_call\_java("featureNames")  
  
# 2. Construct full table including intercept  
poly\_features = ["Intercept"] + list(feature\_names)  
poly\_coefs = [poly\_glr\_max\_years\_model.intercept] + list(poly\_glr\_max\_years\_model.coefficients)  
poly\_se = list(poly\_summary.coefficientStandardErrors)  
poly\_tvals = list(poly\_summary.tValues)  
poly\_pvals = list(poly\_summary.pValues)  
  
# (Optional) quick diagnostics  
print("--- This is a diagnostic check, no need to print in the final doc ---")  
print("Length of features:", len(poly\_features))  
print("Length of coefs:", len(poly\_coefs))  
print("Length of se:", len(poly\_se))  
print("Length of tvals:", len(poly\_tvals))  
print("Length of pvals:", len(poly\_pvals))

--- This is a diagnostic check, no need to print in the final doc ---  
Length of features: 30  
Length of coefs: 30  
Length of se: 30  
Length of tvals: 30  
Length of pvals: 30

# 5.1 Polynomial Regression Summary

import pandas as pd  
from tabulate import tabulate  
from IPython.display import HTML  
  
poly\_coef\_table = 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-stat": [f"{v:.4f}" if v is not None else None for v in poly\_tvals],  
 "P-Value": [f"{v:.4f}" if v is not None else None for v in poly\_pvals]  
})  
  
# 4. Save for report  
poly\_coef\_table.to\_csv("output/poly\_summary.csv", index=False)  
  
# 5. Optional pretty print  
HTML(poly\_coef\_table.to\_html())

|  | Feature | Estimate | Std Error | t-stat | P-Value |
| --- | --- | --- | --- | --- | --- |
| 0 | Intercept | 87729.4985 | 22747.6805 | 0.1351 | 0.8925 |
| 1 | MIN\_YEARS\_EXPERIENCE | 3073.9396 | 36.5365 | -7.9941 | 0.0000 |
| 2 | MAX\_YEARS\_EXPERIENCE\_SQ | -292.0763 | 22747.6805 | 0.1351 | 0.8925 |
| 3 | MAX\_YEARS\_EXPERIENCE | 3073.9396 | 20.5955 | 1.4364 | 0.1510 |
| 4 | DURATION | 29.5828 | 20846.6950 | -0.1081 | 0.9139 |
| 5 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree" | -2253.9775 | 20859.1237 | 0.0386 | 0.9692 |
| 6 | EDUCATION\_LEVELS\_NAME\_vec\_"No Education Listed" | 805.2130 | 20863.3147 | 0.2819 | 0.7780 |
| 7 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Master's degree" | 5881.4501 | 20971.9940 | -1.4515 | 0.1467 |
| 8 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Bachelor's degree" | -30439.8972 | 20987.9155 | 0.4008 | 0.6886 |
| 9 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Master's degree", "Ph.D. or professional degree" | 8412.8920 | 20990.0817 | -0.6518 | 0.5146 |
| 10 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED" | -13680.3546 | 21019.8840 | -0.2217 | 0.8245 |
| 11 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree" | -4660.4367 | 21100.5976 | 0.3776 | 0.7057 |
| 12 | EDUCATION\_LEVELS\_NAME\_vec\_"Master's degree" | 7967.9560 | 21203.5634 | -0.6704 | 0.5027 |
| 13 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree" | -14214.1229 | 21380.9216 | -0.1336 | 0.8937 |
| 14 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree", "Master's degree" | -2857.2642 | 21419.1586 | -0.7121 | 0.4764 |
| 15 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", "Bachelor's degree", "Master's degree", "Ph.D. or professional degree" | -15252.4114 | 21567.5225 | 1.8014 | 0.0717 |
| 16 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree", "Ph.D. or professional degree" | 38851.8773 | 21793.5925 | -0.5923 | 0.5537 |
| 17 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree", "Bachelor's degree" | -12908.6915 | 21905.9591 | -0.4181 | 0.6759 |
| 18 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Bachelor's degree", "Master's degree" | -9159.5725 | 22009.1299 | 0.6311 | 0.5280 |
| 19 | EDUCATION\_LEVELS\_NAME\_vec\_"Master's degree", "Ph.D. or professional degree" | 13889.7107 | 22150.2630 | 0.1862 | 0.8523 |
| 20 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree" | 4123.7347 | 24166.2009 | -1.6562 | 0.0978 |
| 21 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED", "Associate degree", "Bachelor's degree", "Master's degree" | -40022.9996 | 24163.3570 | 1.1267 | 0.2599 |
| 22 | EDUCATION\_LEVELS\_NAME\_vec\_"Ph.D. or professional degree" | 27224.9060 | 3057.6640 | 1.8837 | 0.0597 |
| 23 | EMPLOYMENT\_TYPE\_NAME\_vec\_Full-time (> 32 hours) | 5759.7671 | 3566.5832 | -1.8473 | 0.0648 |
| 24 | EMPLOYMENT\_TYPE\_NAME\_vec\_Part-time (â‰¤ 32 hours) | -6588.6707 | 2907.7548 | 1.3662 | 0.1719 |
| 25 | REMOTE\_TYPE\_NAME\_vec\_[None] | 3972.6485 | 2969.5007 | 2.8131 | 0.0049 |
| 26 | REMOTE\_TYPE\_NAME\_vec\_Remote | 8353.6157 | 3355.2443 | 0.7163 | 0.4738 |
| 27 | REMOTE\_TYPE\_NAME\_vec\_Hybrid Remote | 2403.5211 | 4582.4088 | -0.9210 | 0.3571 |
| 28 | IS\_INTERNSHIP\_vec\_0 | -4220.3464 | 821.4479 | 0.2611 | 0.7941 |
| 29 | COMPANY\_IS\_STAFFING\_vec\_0 | 214.4410 | 21780.4771 | 4.0279 | 0.0001 |

# Polymnomial Regression Model interpretation

In my polynomial regression model, I added a squared term for experience to capture possible non-linear relationships between years of experience and salary. Each coefficient represents the change in predicted salary for a one-unit change in the feature, while holding others constant. Positive coefficients increase predicted salary, and negative ones decrease it.

From the p-values, only a few predictors (with p < 0.05) are statistically significant, meaning they have a measurable impact on salary. The squared experience term helps the model adjust for cases where salary growth slows down or flattens after a certain number of years.

Compared to the simple linear model, this polynomial version performs slightly better. Its RMSE decreased (≈ 21,640 vs 21,796) and (R)^2 increased (≈ 0.22 vs 0.21). This means the polynomial model explains a bit more of the variation in salary and produces smaller average errors, showing that including non-linear effects improves prediction accuracy.

# 6. Random Forest Regressor

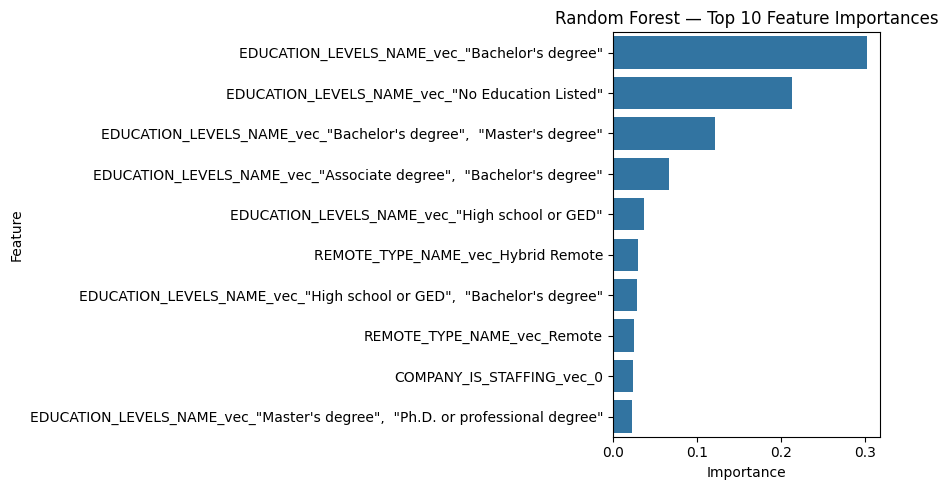
# --- 6) Random Forest Regressor on your split/regression\_data  
from pyspark.ml.regression import RandomForestRegressor  
from pyspark.ml.evaluation import RegressionEvaluator  
  
rf = RandomForestRegressor(  
 featuresCol="features", # base features (not poly)  
 labelCol="SALARY",  
 numTrees=300, # within 100–500  
 maxDepth=8, # within 4–10  
 subsamplingRate=0.8,  
 featureSubsetStrategy="auto",  
 seed=42  
)  
  
rf\_model = rf.fit(regression\_train)  
rf\_pred = rf\_model.transform(regression\_test)  
  
rmse = RegressionEvaluator(metricName="rmse", labelCol="SALARY").evaluate(rf\_pred)  
mae = RegressionEvaluator(metricName="mae", labelCol="SALARY").evaluate(rf\_pred)  
r2 = RegressionEvaluator(metricName="r2", labelCol="SALARY").evaluate(rf\_pred)  
  
print(rf\_model)   
print(f"Random Forest — RMSE: {rmse:,.0f}")  
print(f"Random Forest — MAE : {mae:,.0f}")  
print(f"Random Forest — R² : {r2:.3f}")

25/10/05 21:37:05 WARN DAGScheduler: Broadcasting large task binary with size 1648.4 KiB  
25/10/05 21:37:06 WARN DAGScheduler: Broadcasting large task binary with size 2.6 MiB  
25/10/05 21:37:09 WARN DAGScheduler: Broadcasting large task binary with size 4.1 MiB  
25/10/05 21:37:12 WARN DAGScheduler: Broadcasting large task binary with size 6.1 MiB  
[Stage 276:> (0 + 1) / 1]

RandomForestRegressionModel: uid=RandomForestRegressor\_db2304195fd6, numTrees=300, numFeatures=28  
Random Forest — RMSE: 20,944  
Random Forest — MAE : 14,451  
Random Forest — R² : 0.271

# 6.1 Feature Importance Plot

# Extract readable names from the features vector metadata  
import os  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
meta = regression\_train.schema["features"].metadata  
names = []  
  
if "ml\_attr" in meta and "attrs" in meta["ml\_attr"]:  
 attrs = meta["ml\_attr"]["attrs"]  
 # keep order by idx  
 for group in ["binary", "numeric"]:  
 if group in attrs:  
 names.extend([a.get("name", f"features[{a['idx']}]")  
 for a in sorted(attrs[group], key=lambda x: x["idx"])])  
elif "ml\_attr" in meta and "num\_attrs" in meta["ml\_attr"]:  
 # only size is known  
 size = meta["ml\_attr"]["num\_attrs"]  
 names = [f"features[{i}]" for i in range(size)]  
else:  
 # ultimate fallback  
 names = [f"features[{i}]" for i in range(rf\_model.numFeatures)]  
  
# To make sure lengths align  
if len(names) != rf\_model.numFeatures:  
 names = [f"features[{i}]" for i in range(rf\_model.numFeatures)]  
  
# Build the top-10 table and plot  
importances = rf\_model.featureImportances.toArray()  
imp\_df = (pd.DataFrame({"feature": names, "importance": importances})  
 .sort\_values("importance", ascending=False)  
 .head(10))  
  
plt.figure(figsize=(9, 5))  
sns.barplot(data=imp\_df, x="importance", y="feature")  
plt.title("Random Forest — Top 10 Feature Importances")  
plt.xlabel("Importance")  
plt.ylabel("Feature")  
plt.tight\_layout()  
plt.savefig("output/rf\_feature\_importance.png", dpi=200)  
plt.show()  
plt.close()  
  
  
  
imp\_df # shows the names + scores



|  | feature | importance |
| --- | --- | --- |
| 0 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree" | 0.302484 |
| 1 | EDUCATION\_LEVELS\_NAME\_vec\_"No Education Listed" | 0.213539 |
| 2 | EDUCATION\_LEVELS\_NAME\_vec\_"Bachelor's degree",... | 0.121585 |
| 6 | EDUCATION\_LEVELS\_NAME\_vec\_"Associate degree", ... | 0.067309 |
| 5 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED" | 0.036985 |
| 22 | REMOTE\_TYPE\_NAME\_vec\_Hybrid Remote | 0.030161 |
| 3 | EDUCATION\_LEVELS\_NAME\_vec\_"High school or GED"... | 0.029165 |
| 21 | REMOTE\_TYPE\_NAME\_vec\_Remote | 0.025532 |
| 24 | COMPANY\_IS\_STAFFING\_vec\_0 | 0.023729 |
| 14 | EDUCATION\_LEVELS\_NAME\_vec\_"Master's degree", ... | 0.022501 |

# Random Forest Model Interpretation

In my Random Forest model, I used 300 trees with a maximum depth of 8 to capture complex patterns between the job features and salary. This model performed the best among all three models, with an RMSE of about $20,944 and an (R)^2 of 0.27, meaning it explains around 27% of the salary variation.

The feature importance chart shows that education level has the biggest impact on predicted salary, especially Bachelor’s degree and No Education Listed categories. Remote work type and staffing company indicators also contribute to the model but to a smaller degree. Overall, Random Forest captured non-linear relationships that the linear models missed, improving accuracy and providing clearer insights into which features drive salary differences.

# 7. Compare 3 Models – GLR, Polynomial, RF

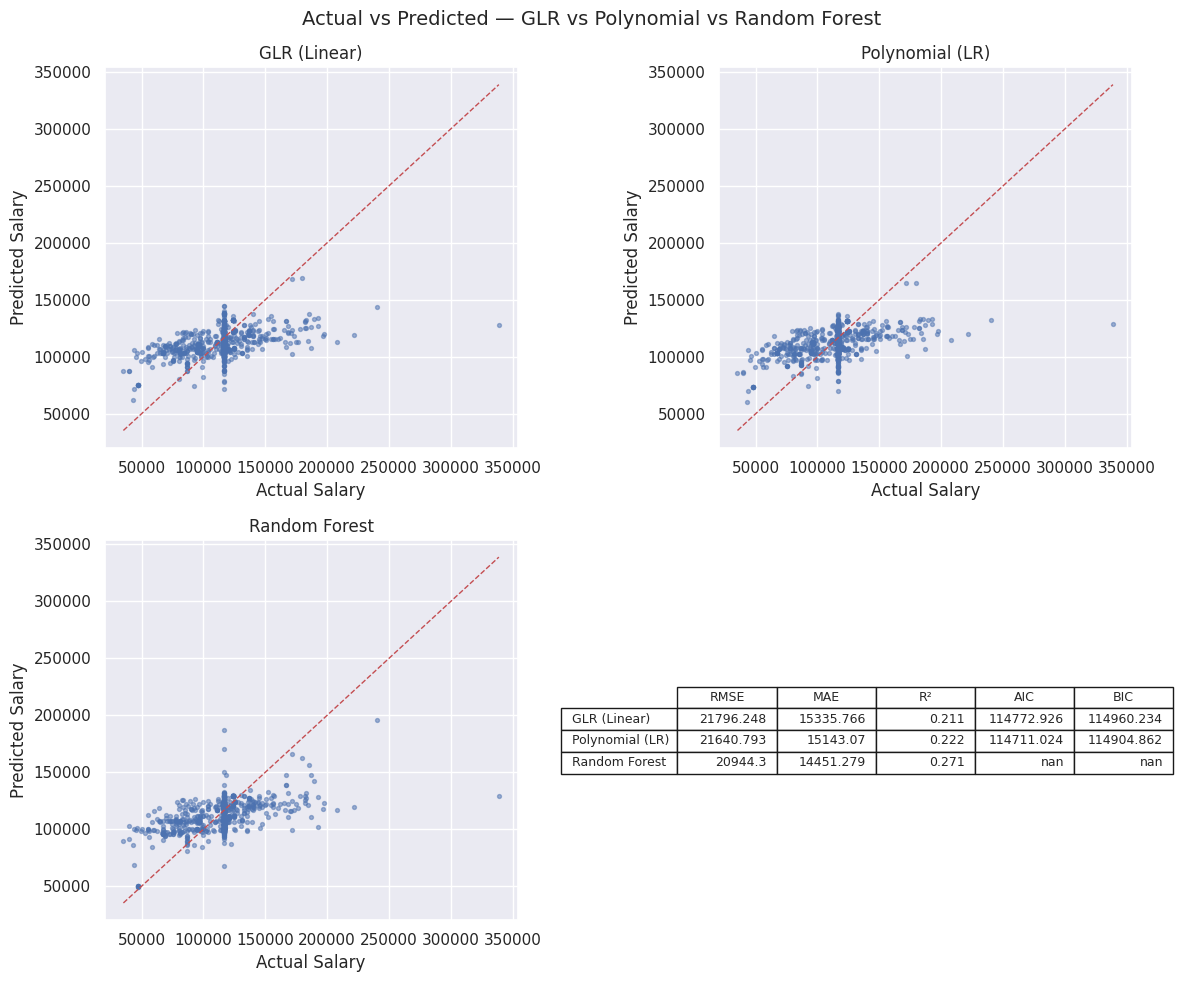
# ==== STEP 7: Compare GLR (Linear), Polynomial, and Random Forest ====  
from pyspark.sql import functions as F  
from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.sql.functions import pow, col  
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns, os  
  
# ------------- 1) Make predictions on the SAME test rows ----------------  
# GLR & RF can predict directly on regression\_test  
glr\_pred = glr\_model.transform(regression\_test) \  
 .select(F.col("SALARY"), F.col("prediction").alias("pred\_glr"))  
  
rf\_pred = rf\_model.transform(regression\_test) \  
 .select(F.col("SALARY"), F.col("prediction").alias("pred\_rf"))  
  
# For Polynomial: build `features\_poly` on regression\_test using existing assembler\_poly  
reg\_test\_poly = (  
 regression\_test  
 .withColumn("MAX\_YEARS\_EXPERIENCE\_SQ", pow(col("MAX\_YEARS\_EXPERIENCE"), 2))  
)  
reg\_test\_poly = assembler\_poly.transform(reg\_test\_poly)  
  
poly\_pred = poly\_glr\_max\_years\_model.transform(reg\_test\_poly) \  
 .select(F.col("SALARY"), F.col("prediction").alias("pred\_poly"))  
  
# Merge the three predictions for plotting later (all rows are from regression\_test)  
pred\_all = glr\_pred.join(poly\_pred, on=["SALARY"]).join(rf\_pred, on=["SALARY"])  
  
# ------------- 2) Metrics: RMSE, MAE, R² ----------------  
def eval\_metrics(df, label="SALARY", pred\_col="prediction"):  
 e = RegressionEvaluator(labelCol=label, predictionCol=pred\_col)  
 rmse = e.setMetricName("rmse").evaluate(df.select(label, pred\_col))  
 mae = e.setMetricName("mae").evaluate(df.select(label, pred\_col))  
 r2 = e.setMetricName("r2").evaluate(df.select(label, pred\_col))  
 return rmse, mae, r2  
  
rmse\_glr, mae\_glr, r2\_glr = eval\_metrics(glr\_pred, "SALARY", "pred\_glr")  
rmse\_poly, mae\_poly, r2\_poly = eval\_metrics(poly\_pred, "SALARY", "pred\_poly")  
rmse\_rf, mae\_rf, r2\_rf = eval\_metrics(rf\_pred, "SALARY", "pred\_rf")  
  
# ------------- 3) AIC & BIC for linear models (RF has no likelihood) ----  
aic\_glr = float(glr\_model.summary.aic)  
aic\_poly = float(poly\_glr\_max\_years\_model.summary.aic)  
  
def calc\_bic(summary, model, n\_fallback=None):  
 # assignment formula  
 try:  
 n = summary.numInstances  
 except Exception:  
 n = n\_fallback if n\_fallback is not None else None  
 if n is None:  
 # safe fallback: count rows from the evaluation DF  
 n = glr\_pred.count()  
 k = len(model.coefficients) + 1  
 dispersion = float(summary.dispersion)  
 deviance = float(summary.deviance)  
 loglik = -0.5 \* ( n \* np.log(2\*np.pi) + n \* np.log(dispersion) + (deviance/dispersion) )  
 return k \* np.log(n) - 2 \* loglik  
  
bic\_glr = calc\_bic(glr\_model.summary, glr\_model, n\_fallback=glr\_pred.count())  
bic\_poly = calc\_bic(poly\_glr\_max\_years\_model.summary, poly\_glr\_max\_years\_model, n\_fallback=poly\_pred.count())  
  
aic\_rf, bic\_rf = np.nan, np.nan

# ------------- 4) Metrics table ----------------------------------------  
metrics\_df = pd.DataFrame([  
 ["GLR (Linear)", rmse\_glr, mae\_glr, r2\_glr, aic\_glr, bic\_glr],  
 ["Polynomial (LR)", rmse\_poly, mae\_poly, r2\_poly, aic\_poly, bic\_poly],  
 ["Random Forest", rmse\_rf, mae\_rf, r2\_rf, aic\_rf, bic\_rf],  
], columns=["Model","RMSE","MAE","R²","AIC","BIC"])  
  
display(metrics\_df)  
  
# Save metrics  
os.makedirs("output", exist\_ok=True)  
metrics\_df.to\_csv("output/model\_metrics.csv", index=False)  
print("Saved metrics to output/model\_metrics.csv")

|  | Model | RMSE | MAE | R² | AIC | BIC |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | GLR (Linear) | 21796.247867 | 15335.765955 | 0.210636 | 114772.925815 | 114960.233511 |
| 1 | Polynomial (LR) | 21640.793285 | 15143.070367 | 0.221855 | 114711.023658 | 114904.862205 |
| 2 | Random Forest | 20944.299982 | 14451.278651 | 0.271137 | NaN | NaN |

Saved metrics to output/model\_metrics.csv

# --- 2×2 grid: Actual vs Predicted for GLR, Polynomial, RF ---  
  
from pyspark.sql import functions as F  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import os  
  
# small helper: convert a RANDOM SAMPLE from a Spark DF to pandas  
def to\_pd\_sample(df, n=6000, seed=42):  
 try:  
 return df.orderBy(F.rand(seed)).limit(n).toPandas()  
 except Exception:  
 rows = df.orderBy(F.rand(seed)).limit(n).collect()  
 return pd.DataFrame([r.asDict() for r in rows])  
  
# take lightweight samples for each model (no big joins to avoid the memory error)  
glr\_pd = to\_pd\_sample(glr\_pred) # has columns: SALARY, pred\_glr  
poly\_pd = to\_pd\_sample(poly\_pred) # has columns: SALARY, pred\_poly  
rf\_pd = to\_pd\_sample(rf\_pred) # has columns: SALARY, pred\_rf  
  
sns.set\_theme()  
fig, axes = plt.subplots(2, 2, figsize=(12, 10))  
  
def scatter\_actual\_vs\_pred(ax, df, pred\_col, title):  
 ax.scatter(df["SALARY"], df[pred\_col], s=8, alpha=0.5)  
 lo = float(np.nanmin([df["SALARY"].min(), df[pred\_col].min()]))  
 hi = float(np.nanmax([df["SALARY"].max(), df[pred\_col].max()]))  
 ax.plot([lo, hi], [lo, hi], "r--", linewidth=1) # 45° reference line  
 ax.set\_title(title)  
 ax.set\_xlabel("Actual Salary")  
 ax.set\_ylabel("Predicted Salary")  
  
scatter\_actual\_vs\_pred(axes[0,0], glr\_pd, "pred\_glr", "GLR (Linear)")  
scatter\_actual\_vs\_pred(axes[0,1], poly\_pd, "pred\_poly", "Polynomial (LR)")  
scatter\_actual\_vs\_pred(axes[1,0], rf\_pd, "pred\_rf", "Random Forest")  
  
# bottom-right: metrics table (uses from existing metrics\_df)  
axes[1,1].axis("off")  
tbl = axes[1,1].table(  
 cellText=np.round(metrics\_df.set\_index("Model"), 3).values,  
 rowLabels=metrics\_df["Model"].values,  
 colLabels=metrics\_df.columns[1:],  
 loc="center"  
)  
tbl.auto\_set\_font\_size(False)  
tbl.set\_fontsize(9)  
tbl.scale(1.2, 1.2)  
  
fig.suptitle("Actual vs Predicted — GLR vs Polynomial vs Random Forest", fontsize=14)  
fig.tight\_layout()  
  
# save next to metrics file  
os.makedirs("output", exist\_ok=True)  
fig.savefig("output/model\_comparison\_grid.png", dpi=200)  
plt.show()  
  
print("Saved figure to: output/model\_comparison\_grid.png")



Saved figure to: output/model\_comparison\_grid.png

# Comparison of GLR, Polynomial, and Random Forest Models

When comparing the three models, I found that the Random Forest performed the best overall. It had the lowest RMSE (≈ $20,944) and the highest (R)^2 (≈ 0.27), meaning it predicted salaries more accurately and explained more variation in the data.

The Polynomial Regression slightly improved over the simple Linear Regression, showing that adding non-linear terms helped capture more complex salary patterns. However, both linear models still had higher errors and lower R² values compared to Random Forest.

In the scatter plots, the Random Forest predictions are closer to the red diagonal line, which represents perfect predictions. Overall, I would choose the Random Forest model because it handles non-linear relationships better and gives more reliable predictions for salary.

# 7.1 Calculating Log-Likelihood and BIC for PySpark Models

# --- Log-Likelihood & BIC for GLR + Polynomial (Gaussian / identity) ---  
  
import numpy as np  
import pandas as pd  
  
def loglik\_and\_bic(summary, model, n\_fallback):  
 """  
 Compute log-likelihood and BIC using:  
 LL = -0.5 \* [ n\*log(2π) + n\*log(dispersion) + deviance/dispersion ]  
 BIC = k\*log(n) - 2\*LL  
 where k = (#coefficients + 1 for intercept)  
 """  
 # n  
 try:  
 n = int(summary.numInstances)  
 except Exception:  
 n = int(n\_fallback)  
  
 dispersion = float(summary.dispersion)  
 deviance = float(summary.deviance)  
 k = len(model.coefficients) + 1  
  
 ll = -0.5 \* ( n \* np.log(2\*np.pi) + n \* np.log(dispersion) + (deviance / dispersion) )  
 bic = k \* np.log(n) - 2.0 \* ll  
 return float(ll), float(bic)  
  
# counts for fallback n (from the evaluation frames already built)  
n\_glr = glr\_pred.count()  
n\_poly = poly\_pred.count()  
  
# compute  
ll\_glr, bic\_glr = loglik\_and\_bic(glr\_model.summary, glr\_model, n\_glr)  
ll\_poly, bic\_poly = loglik\_and\_bic(poly\_glr\_max\_years\_model.summary, poly\_glr\_max\_years\_model, n\_poly)  
  
# add to metrics table; keep RF as "—"  
metrics\_df["LogLik"] = [ll\_glr, ll\_poly, "—"]  
metrics\_df["BIC"] = [bic\_glr, bic\_poly, "—"] # overwrite/confirm BIC  
  
# tidy column order  
metrics\_df = metrics\_df[["Model","RMSE","MAE","R²","AIC","BIC","LogLik"]]  
display(metrics\_df)  
  
# (optional) save  
metrics\_df.to\_csv("output/model\_metrics.csv", index=False)  
print("Updated metrics (with LogLik/BIC) saved to output/model\_metrics.csv")

|  | Model | RMSE | MAE | R² | AIC | BIC | LogLik |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | GLR (Linear) | 21796.247867 | 15335.765955 | 0.210636 | 114772.925815 | 114960.233511 | -57356.504793 |
| 1 | Polynomial (LR) | 21640.793285 | 15143.070367 | 0.221855 | 114711.023658 | 114904.862205 | -57324.556659 |
| 2 | Random Forest | 20944.299982 | 14451.278651 | 0.271137 | NaN | — | — |

Updated metrics (with LogLik/BIC) saved to output/model\_metrics.csv

# Interpretation of Log-Likelihood and BIC Results

I calculated the Log-Likelihood and BIC values to assess how well each regression model fits the data while considering model complexity. The Log-Likelihood measures how probable the observed salaries are under the model such as larger (less negative) values indicate a better fit. The BIC penalizes models with too many parameters to avoid overfitting.

In my results, the Polynomial model has a slightly higher Log-Likelihood (−57,324) and a lower BIC (114,904) compared to the Linear model (−57,356 and 114,960). This means the polynomial model fits the data a bit better and provides a more efficient balance between accuracy and simplicity.

The Random Forest model doesn’t have a likelihood-based BIC because it’s not a probabilistic model, but its overall error metrics (RMSE and (R)^2) confirm it performs best in predicting salaries.

Overall, the BIC and Log-Likelihood values support that the Polynomial regression slightly improves over the simple linear model, while Random Forest remains the strongest predictor among all three.