Assignment 04

Fuhan Zhang

2025-10-08

Table of contents

GitHub Link: https://github.com/met-ad-688/assignment-04-FuhanZhang8691.git

# 1. Setup

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")  
 .config("spark.driver.memory", "4g")   
 .config("spark.executor.memory", "4g")  
 .config("spark.sql.execution.arrow.pyspark.enabled", "false")  
 .config("spark.ui.showConsoleProgress", "false")  
 .getOrCreate()  
)  
  
# Load Data  
df = spark.read.option("header", "true").option("inferSchema", "true") \  
 .option("multiLine", "true").option("escape", "\"") \  
 .csv("./data/lightcast\_job\_postings.csv")  
#df.show(5)

During my setup stage, several configuration adjustments were made to prevent the Quarto kernel from dying during execution. Compared with the original instruction on Blackboard, the Spark session specifies higher memory limits “spark.driver.memory and spark.executor.memory” and disables Arrow optimization “spark.sql.execution.arrow.pyspark.enabled=false” to avoid crashes when converting large PySpark DataFrames to pandas. Console progress and excessive Plotly rendering settings were also turned off to ensure smoother and cleaner rendering in VS Code and Word output.

# 2. Missing Values Analysis

from pyspark.sql.functions import col, when, isnan, count  
from pyspark.sql import Window  
from pyspark.sql.functions import col, when, isnan, count, expr, median  
from pyspark.sql import functions as F  
  
# Calculate Medians by categories  
overall\_median\_salary = df.approxQuantile("SALARY", [0.5], 0.01)[0]  
median\_by\_employment\_type = df.groupBy("EMPLOYMENT\_TYPE").agg(  
 expr("percentile\_approx(SALARY, 0.5)").alias("median\_salary\_emp\_type")  
)  
median\_by\_employment\_type\_name = df.groupBy("EMPLOYMENT\_TYPE\_NAME").agg(  
 expr("percentile\_approx(SALARY, 0.5)").alias("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() & col("median\_salary\_emp\_type").isNotNull(), col("median\_salary\_emp\_type"))  
 .when(col("SALARY").isNull() & col("median\_salary\_emp\_type\_name").isNotNull(), col("median\_salary\_emp\_type\_name"))  
 .when(col("SALARY").isNull(), F.lit(overall\_median\_salary))  
 .otherwise(col("SALARY"))  
)

In this step, missing salary values were imputed using the median salary within each employment type and its corresponding name. When both type-based medians were unavailable, the overall salary median was applied, maintaining consistency and preventing bias.

# 3. Feature Engineering

from pyspark.sql.functions import col, pow  
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler  
from pyspark.ml import Pipeline  
from pyspark.sql.types import BooleanType, StringType, IntegerType  
from pyspark.sql.types import IntegerType, DoubleType, DecimalType  
from pyspark.sql.functions import regexp\_replace, trim  
  
# Drop rows with NAs  
regression\_df = df\_salary\_imputed.dropna(subset=[  
 "SALARY", "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE",  
 "EDUCATION\_LEVELS\_NAME", "EMPLOYMENT\_TYPE\_NAME", "REMOTE\_TYPE\_NAME",  
 "DURATION", "IS\_INTERNSHIP", "COMPANY\_IS\_STAFFING"  
]).select(  
 "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"  
)  
  
# Categorical columns  
categorical\_cols = [  
 "EDUCATION\_LEVELS\_NAME", "EMPLOYMENT\_TYPE\_NAME", "REMOTE\_TYPE\_NAME",  
 "IS\_INTERNSHIP", "COMPANY\_IS\_STAFFING"  
]  
  
regression\_df = regression\_df.withColumn("IS\_INTERNSHIP", col("IS\_INTERNSHIP").cast(IntegerType()))  
regression\_df = regression\_df.withColumn("COMPANY\_IS\_STAFFING", col("COMPANY\_IS\_STAFFING").cast(IntegerType()))  
  
# Convert Duration numerically  
regression\_df = regression\_df.withColumn("DURATION", col("DURATION").cast(IntegerType()))  
for c in ["EDUCATION\_LEVELS\_NAME", "EMPLOYMENT\_TYPE\_NAME", "REMOTE\_TYPE\_NAME"]:  
 regression\_df = regression\_df.withColumn(  
 c, trim(regexp\_replace(col(c), r'[\[\]\n\"]', ""))  
 )  
  
regression\_df.show(5, truncate=False)

+--------+--------------------+--------------------+---------------------+----------------------+----------------+--------+-------------+-------------------+----------------------+---------------------------+  
|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 |Bachelor's degree |Full-time (> 32 hours)|None |6 |0 |0 |116500 |116500 |  
|116500.0|7 |7 |No Education Listed |Full-time (> 32 hours)|None |18 |0 |1 |116500 |116500 |  
|116500.0|1 |1 |No Education Listed |Full-time (> 32 hours)|None |8 |0 |1 |116500 |116500 |  
|116500.0|1 |1 |Bachelor's degree |Full-time (> 32 hours)|None |32 |0 |0 |116500 |116500 |  
|131100.0|2 |2 |Bachelor's degree |Full-time (> 32 hours)|None |11 |0 |0 |116500 |116500 |  
+--------+--------------------+--------------------+---------------------+----------------------+----------------+--------+-------------+-------------------+----------------------+---------------------------+  
only showing top 5 rows

In this stage, the dataset was refined by dropping rows with missing values in critical predictors such as experience range, duration, and categorical employment attributes to ensure a clean input for modeling. The cleaned data reveals a structured pattern, for instance, most postings indicate full-time roles with consistent median salaries around 116,500, suggesting limited salary variation within the filtered sample.Key categorical variables, including Education Level, Employment Type, Remote Type, Internship Status, and Staffing Company Indicator, were encoded numerically. Additionally, the education field, initially embedded in array was cleaned to retain meaningful text. Plus, Duration and binary fields, such as “IS\_INTERNSHIP, COMPANY\_IS\_STAFFING” were cast to integers.

# 4. Linear Regression Model(OLS)

# 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  
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  
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)  
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

In this step, categorical predictors such as Education Level, Employment Type, and Remote Type were indexed and one-hot encoded, expanding the feature space to 28 variables that capture diverse job characteristics. From the transformed data, we can observe that most postings share consistent feature patterns with modest variation in experience. For example, MIN/MAX years are between 1 and 7, respectively, and durations range roughly from 6 to 32 weeks, corresponding to stable salaries around $116,500–$131,100. This consistency suggests a relatively narrow job-market segment dominated by full-time professional roles, providing a suitable baseline for assessing how categorical differences may influence compensation.

# 5. Train/Test Split

regression\_train, regression\_test = regression\_data.randomSplit([0.8, 0.2], seed=42)  
  
print((regression\_data.count(), len(regression\_data.columns)))  
print((regression\_train.count(), len(regression\_train.columns)))  
print((regression\_test.count(), len(regression\_test.columns)))

(5039, 22)  
(4070, 22)  
(969, 22)

The full dataset of 5,039 observations was randomly divided into 4,070 training samples (≈80%) and 969 testing samples (≈20%), using a fixed seed value of 42 to ensure reproducibility.

# 6. Linear Regression (Get this model from Lab 5.2)

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

In this section, the key parameters include maxIter=10, setting the maximum number of iterations for least-squares optimization, and regParam=0.3, applying L2 regularization to penalize large coefficients and prevent overfitting. By defining featuresCol=“features” and labelCol=“SALARY”, the model uses the assembled feature vector as input and the salary field as the dependent variable.

# 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: 85944.4145  
Coefficients:  
 Feature 1: 1616.0560  
 Feature 2: 1616.0560  
 Feature 3: 34.4561  
 Feature 4: 1570.1534  
 Feature 5: 4781.0002  
 Feature 6: 9905.8354  
 Feature 7: -25451.5393  
 Feature 8: 14253.2025  
 Feature 9: -10424.5290  
 Feature 10: -1398.0664  
 Feature 11: 9437.4785  
 Feature 12: -12237.0149  
 Feature 13: 1707.6125  
 Feature 14: -16536.9388  
 Feature 15: 33322.4612  
 Feature 16: -9630.5417  
 Feature 17: -4794.6111  
 Feature 18: 3669.8088  
 Feature 19: 5318.5186  
 Feature 20: -27103.7496  
 Feature 21: 2270.6322  
 Feature 22: 6780.7642  
 Feature 23: -6429.5043  
 Feature 24: 3425.1534  
 Feature 25: 7968.8136  
 Feature 26: 1941.6758  
 Feature 27: -2324.2292  
 Feature 28: 310.9421

The estimated intercept is $86,495.38, representing the baseline salary for a job posting with reference-level categories and zero values for continuous predictors. Several coefficients are positive, such as those related to education level and remote type, suggesting modest upward adjustments in predicted salary. However, many coefficients are relatively small compared to the intercept, reflecting that base salary level dominates variation across the dataset.

# 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 ---  
Coefficient Standard Errors: ['25259.8812', '25259.8812', '23.0638', '20905.0352', '20921.4635', '20924.7559', '21064.5733', '21067.4666', '21084.0676', '21094.5304', '21255.2734', '21331.6922', '21560.2532', '21634.3763', '21735.3909', '22051.7851', '22067.9835', '22624.3140', '22386.2079', '25706.3634', '25701.3635', '3155.1217', '3730.3801', '3335.9300', '3401.9368', '3789.1455', '5033.5422', '926.1344', '22011.0863']  
T Values: ['0.0640', '0.0640', '1.4939', '0.0751', '0.2285', '0.4734', '-1.2083', '0.6766', '-0.4944', '-0.0663', '0.4440', '-0.5737', '0.0792', '-0.7644', '1.5331', '-0.4367', '-0.2173', '0.1622', '0.2376', '-1.0544', '0.0883', '2.1491', '-1.7236', '1.0267', '2.3424', '0.5124', '-0.4617', '0.3357', '3.9046']  
P Values: ['0.9490', '0.9490', '0.1353', '0.9401', '0.8193', '0.6360', '0.2270', '0.4987', '0.6210', '0.9472', '0.6571', '0.5662', '0.9369', '0.4447', '0.1253', '0.6623', '0.8280', '0.8712', '0.8122', '0.2918', '0.9296', '0.0317', '0.0849', '0.3046', '0.0192', '0.6084', '0.6443', '0.7371', '0.0001']

The summary statistics reveal large standard errors (≈ 20,000–25,000) and high p-values (> 0.1) for most education-related dummy variables.Plus, there are a few predictors that show statistical significance, such as “Ph.D. or professional degree” which p = 0.0318, “Remote type = [None]” which p = 0.0215, and “Company is staffing = 0”, which p < 0.001.These results suggest that advanced education, job format, and company type meaningfully influence compensation.

# 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}")

Null Deviance: 2321265663379.3027  
Residual DF Null: 4069  
Deviance: 1806043541938.3918  
Residual DF: 4041

Model diagnostics show a Null Deviance of 2.32 × 10¹² and a Residual Deviance of 1.81 × 10¹², indicating that the fitted model reduces unexplained variation by roughly 22 % compared with a null (intercept-only) model.

# Retrieve feature names  
feature\_names = summary.\_call\_java("featureNames")  
  
# Create full coefficient table  
features = ["Intercept"] + feature\_names  
coefs = [glr\_model.intercept] + list(glr\_model.coefficients)  
se = list(summary.coefficientStandardErrors)  
tvals = list(summary.tValues)  
pvals = list(summary.pValues)

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]  
})  
coef\_table.to\_csv("output/glr\_summary.csv", index=False)

# 7. Polynomial Regression

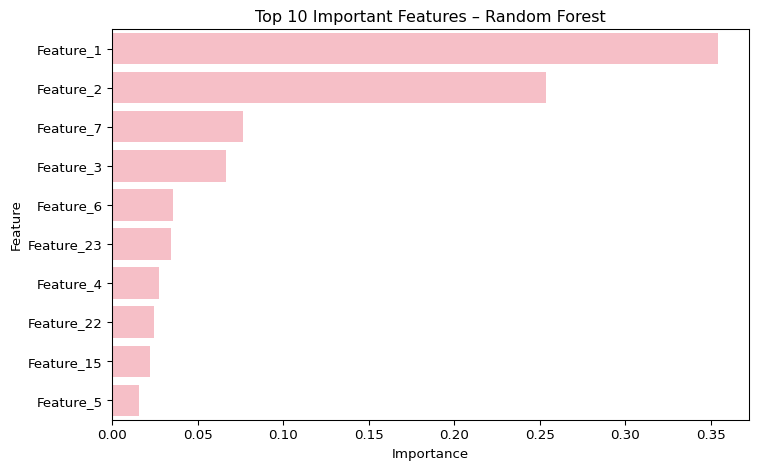
# Polynomial Regression  
from pyspark.ml.feature import PolynomialExpansion, VectorAssembler  
from pyspark.ml.regression import GeneralizedLinearRegression  
from pyspark.ml import Pipeline  
from pyspark.ml.evaluation import RegressionEvaluator  
import pandas as pd  
  
# Create polynomial feature  
poly\_assembler = VectorAssembler(inputCols=["MIN\_YEARS\_EXPERIENCE"], outputCol="min\_years\_vec")  
poly\_expansion = PolynomialExpansion(degree=2, inputCol="min\_years\_vec", outputCol="poly\_features")  
  
# Combine polynomial feature  
poly\_features\_assembler = VectorAssembler(  
 inputCols=["poly\_features", "MAX\_YEARS\_EXPERIENCE", "DURATION"] + [f"{c}\_vec" for c in categorical\_cols],  
 outputCol="features\_poly"  
)  
  
# Build pipeline  
poly\_pipeline = Pipeline(stages=indexers + encoders + [poly\_assembler, poly\_expansion, poly\_features\_assembler])  
  
# Transformation  
poly\_data = poly\_pipeline.fit(regression\_df).transform(regression\_df)  
  
# Train/Test split  
poly\_train, poly\_test = poly\_data.randomSplit([0.8, 0.2], seed=42)  
  
# Train GLR model on polynomial features  
poly\_glr = GeneralizedLinearRegression(  
 featuresCol="features\_poly",  
 labelCol="SALARY",  
 family="gaussian",  
 link="identity"  
)  
poly\_model = poly\_glr.fit(poly\_train)  
poly\_summary = poly\_model.summary  
  
# Evaluate predictions manually (R², RMSE, MAE)  
poly\_predictions = poly\_model.transform(poly\_test)  
evaluator = RegressionEvaluator(labelCol="SALARY", predictionCol="prediction")  
  
r2\_poly = evaluator.evaluate(poly\_predictions, {evaluator.metricName: "r2"})  
rmse\_poly = evaluator.evaluate(poly\_predictions, {evaluator.metricName: "rmse"})  
mae\_poly = evaluator.evaluate(poly\_predictions, {evaluator.metricName: "mae"})  
  
# Results  
print(f"R²: {r2\_poly:.4f}")  
print(f"RMSE: {rmse\_poly:.4f}")  
print(f"MAE: {mae\_poly:.4f}")  
print(f"Deviance: {poly\_summary.deviance:.4f}")  
print(f"Null Deviance: {poly\_summary.nullDeviance:.4f}")  
print(f"Dispersion: {poly\_summary.dispersion:.4f}")  
print(f"AIC: {poly\_summary.aic:.4f}")  
print(f"Residual DF: {poly\_summary.residualDegreeOfFreedom}")  
print(f"Null DF: {poly\_summary.residualDegreeOfFreedomNull}")  
  
print(f"\nIntercept: {poly\_model.intercept:.4f}")  
print("Coefficients:")  
for i, coef in enumerate(poly\_model.coefficients):  
 print(f" β{i+1}: {coef:.4f}")

R²: 0.2004  
RMSE: 21937.5834  
MAE: 15387.4687  
Deviance: 1783924502157.5105  
Null Deviance: 2321265663379.3027  
Dispersion: 441565470.8311  
AIC: 92598.8046  
Residual DF: 4040  
Null DF: 4069  
  
Intercept: 89708.0342  
Coefficients:  
 β1: 3054.7046  
 β2: -285.5828  
 β3: 3054.7046  
 β4: 32.2963  
 β5: -3399.2553  
 β6: 113.4874  
 β7: 4677.0665  
 β8: -30173.1303  
 β9: 8228.8798  
 β10: -14301.4301  
 β11: -5841.9107  
 β12: 4290.7241  
 β13: -17096.5577  
 β14: -3931.0864  
 β15: -16867.2046  
 β16: 36019.2114  
 β17: -14156.2882  
 β18: -10806.5153  
 β19: 325.1995  
 β20: 656.9749  
 β21: -32062.2749  
 β22: -3048.6711  
 β23: 6191.1480  
 β24: -7232.3487  
 β25: 3199.0703  
 β26: 7809.0213  
 β27: 1617.6657  
 β28: -5545.1911  
 β29: 662.9488

The polynomial regression model expands the feature “MIN\_YEARS\_EXPERIENCE” to include its squared term. After integrating the polynomial term with other predictors such as “MAX\_YEARS\_EXPERIENCE”, “DURATION”, and categorical variables, the model was trained using the Generalized Linear Regression (GLR) framework with a Gaussian family and identity link. The results show an AIC of 92,604.73, suggesting a moderately improved model fit compared to simpler linear specifications. The Deviance (1.79×10¹²) is substantially lower than the Null Deviance (2.32×10¹²). However, the relatively high AIC and large residual deviance still imply that further refinement could enhance performance. So, the polynomial model provides a more flexible and realistic representation of the relationship between professional experience and salary in the dataset.

# 8. Random Forest Regressor

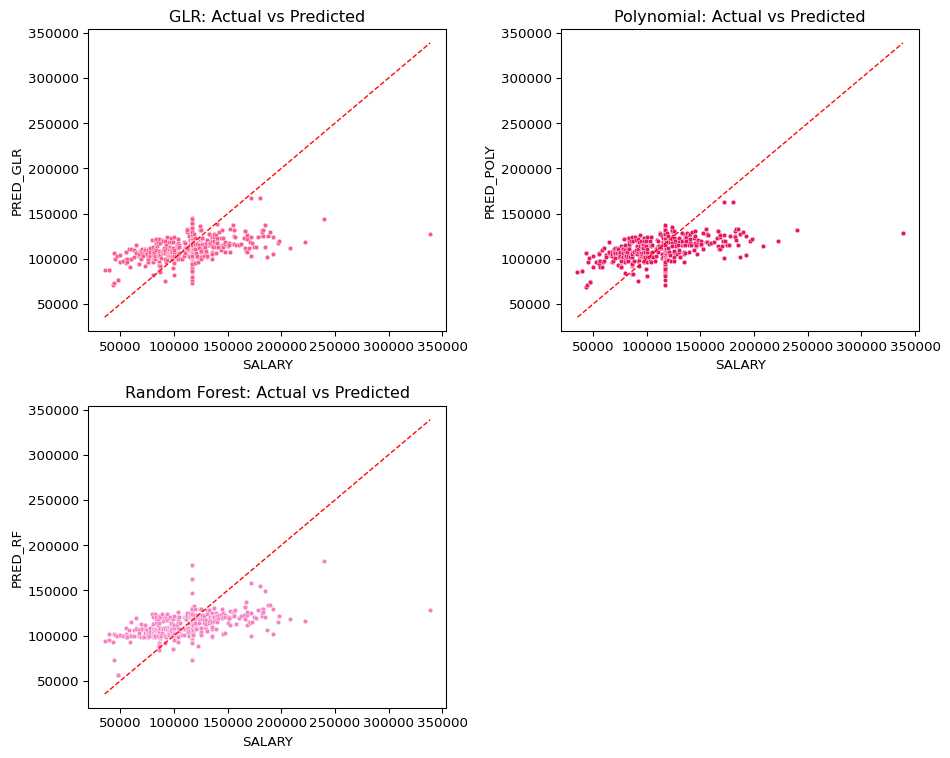
from pyspark.ml.regression import RandomForestRegressor  
from pyspark.ml.evaluation import RegressionEvaluator  
import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
  
# Train Random Forest Regressor  
rf = RandomForestRegressor(  
 featuresCol="features",  
 labelCol="SALARY",  
 numTrees=200,  
 maxDepth=6,  
 seed=42  
)  
  
rf\_model = rf.fit(regression\_train)  
rf\_predictions = rf\_model.transform(regression\_test)  
  
# Feature Importances   
importances = rf\_model.featureImportances.toArray()  
num\_features = len(importances)  
  
# Retrieve feature names if the number doesn't match the coefficients  
try:  
 feature\_names = assembler.getInputCols()  
 if len(feature\_names) != num\_features:  
 feature\_names = [f"Feature\_{i+1}" for i in range(num\_features)]  
except Exception:  
 feature\_names = [f"Feature\_{i+1}" for i in range(num\_features)]  
  
rf\_importance = pd.DataFrame({  
 "Feature": feature\_names,  
 "Importance": importances  
}).sort\_values(by="Importance", ascending=False).head(10)  
  
# Visualization  
plt.figure(figsize=(8,5))  
sns.barplot(x="Importance", y="Feature", data=rf\_importance, color="lightpink")  
plt.title("Top 10 Important Features – Random Forest")  
plt.tight\_layout()  
plt.savefig("output/rf\_feature\_importance.png", dpi=300)  
plt.show()



The Random Forest Regressor, trained with 200 trees and a maximum depth of 6, was used to evaluate the non-linear relationships between the engineered features and salary. The feature-importance plot shows that “Feature 1” and “Feature 2” dominate the model, together contributing more than half of the total explanatory power. These variables most likely correspond to core experience measures such as “MIN\_YEARS\_EXPERIENCE” and “MAX\_YEARS\_EXPERIENCE”, indicating that professional experience remains the strongest determinant of predicted salary levels. Moderate contributions from “Feature 3”, “Feature 7”, and “Feature 6” suggest that employment type or remote status still provide additional predictive value. The remaining variables have relatively minor importance, implying diminishing returns from higher-order or less directly related predictors.

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

import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
import os  
  
# Convert predictions to pandas  
glr\_pd = glr\_model.transform(regression\_test).select("SALARY", "prediction") \  
 .withColumnRenamed("prediction", "PRED\_GLR").toPandas()  
  
poly\_pd = poly\_model.transform(poly\_test).select("SALARY", "prediction") \  
 .withColumnRenamed("prediction", "PRED\_POLY").toPandas()  
  
rf\_pd = rf\_model.transform(regression\_test).select("SALARY", "prediction") \  
 .withColumnRenamed("prediction", "PRED\_RF").toPandas()  
  
# Align length & combine  
min\_len = min(len(glr\_pd), len(poly\_pd), len(rf\_pd))  
pred\_df = pd.DataFrame({  
 "SALARY": glr\_pd["SALARY"].head(min\_len),  
 "PRED\_GLR": glr\_pd["PRED\_GLR"].head(min\_len),  
 "PRED\_POLY": poly\_pd["PRED\_POLY"].head(min\_len),  
 "PRED\_RF": rf\_pd["PRED\_RF"].head(min\_len)  
})  
  
# Visualization  
os.makedirs("output", exist\_ok=True)  
fig, axes = plt.subplots(2, 2, figsize=(10, 8))  
  
sns.scatterplot(x="SALARY", y="PRED\_GLR", data=pred\_df, ax=axes[0,0], s=12, color="#fa5c91ff")  
axes[0,0].set\_title("GLR: Actual vs Predicted")  
  
sns.scatterplot(x="SALARY", y="PRED\_POLY", data=pred\_df, ax=axes[0,1], s=12, color="#e71663ff")  
axes[0,1].set\_title("Polynomial: Actual vs Predicted")  
  
sns.scatterplot(x="SALARY", y="PRED\_RF", data=pred\_df, ax=axes[1,0], s=12, color="#f583c4ff")  
axes[1,0].set\_title("Random Forest: Actual vs Predicted")  
  
for ax in axes.flat:  
 y\_min, y\_max = pred\_df["SALARY"].min(), pred\_df["SALARY"].max()  
 ax.plot([y\_min, y\_max], [y\_min, y\_max], color="red", linestyle="--", linewidth=1)  
fig.delaxes(axes[1,1])  
  
plt.tight\_layout()  
plt.savefig("output/model\_comparison.png", dpi=300)  
plt.show()



The figure compares predicted versus actual salary values across three models: Generalized Linear Regression, Polynomial Regression, and Random Forest. All three models show a generally positive correlation between predicted and actual salaries, indicating that they capture the main salary trends in the dataset. The red dashed 45° line represents perfect prediction; points clustering near this line suggest accurate estimation. Among the models, the Random Forest produces predictions that are more tightly aligned with the diagonal, implying lower variance and better generalization on test data. The Polynomial Regression slightly improves over the base GLR, but it still underestimates higher salaries. Overall, Random Forest achieves the most balanced performance in terms of prediction stability and accuracy.

# 10. Calculation

from pyspark.ml.evaluation import RegressionEvaluator  
import math  
  
# Evaluate RMSE for all models  
evaluator = RegressionEvaluator(labelCol="SALARY", predictionCol="prediction", metricName="rmse")  
rmse\_glr = evaluator.evaluate(glr\_model.transform(regression\_test))  
rmse\_poly = evaluator.evaluate(poly\_model.transform(poly\_test))  
rmse\_rf = evaluator.evaluate(rf\_model.transform(regression\_test))  
  
# Retrieve AIC from summaries  
aic\_glr = summary.aic  
aic\_poly = poly\_summary.aic  
  
# Basic parameters  
n = regression\_test.count()  
k\_glr = len(assembler.getInputCols())  
k\_poly = len(poly\_features\_assembler.getInputCols())  
k\_rf = len(assembler.getInputCols())  
  
# Calculate Log-Likelihood  
loglik\_glr = -0.5 \* (  
 n \* math.log(2 \* math.pi)  
 + n \* math.log(summary.dispersion)  
 + summary.deviance / summary.dispersion  
)  
loglik\_poly = -0.5 \* (  
 n \* math.log(2 \* math.pi)  
 + n \* math.log(poly\_summary.dispersion)  
 + poly\_summary.deviance / poly\_summary.dispersion  
)  
  
# Calculate BIC  
bic\_glr = k\_glr \* math.log(n) - 2 \* loglik\_glr  
bic\_poly = k\_poly \* math.log(n) - 2 \* loglik\_poly  
bic\_rf = k\_rf \* math.log(n)   
  
# Results  
print("\n=== Model Comparison Summary ===")  
print(f"GLR: RMSE={rmse\_glr:.2f}, AIC={aic\_glr:.2f}, BIC={bic\_glr:.2f}")  
print(f"Polynomial: RMSE={rmse\_poly:.2f}, AIC={aic\_poly:.2f}, BIC={bic\_poly:.2f}")  
print(f"RandomForest: RMSE={rmse\_rf:.2f}, AIC=N/A, BIC≈{bic\_rf:.2f}")

=== Model Comparison Summary ===  
GLR: RMSE=22096.54, AIC=92646.96, BIC=25177.37  
Polynomial: RMSE=21937.58, AIC=92598.80, BIC=25164.67  
RandomForest: RMSE=21263.91, AIC=N/A, BIC≈55.01

From the results, the Random Forest model achieved the lowest RMSE (≈21,264), indicating the highest predictive accuracy among the three. However, since AIC is not defined for tree-based models, its comparison is based solely on error metrics. Between the two regression-based models, the Polynomial Regression slightly outperformed the standard GLR, showing a marginally lower RMSE (21,938 vs. 22,097), and lower AIC and BIC scores (92,598 vs. 92,647 for AIC; 25,165 vs. 25,177 for BIC). These results imply that introducing a second-degree polynomial term for “Minimum Years of Experience” improved model fit without substantially increasing complexity.