Assignment 04 - Machine Learning on Scale

Cindy Guzman

October 3, 2025

## 1 Abstract

Assignment 04 focuses on building and evaluating salary prediction models using Lightcast job-posting data. After enforcing filters (positive salaries; non-negative experience), categorical fields were encoded and assembled for modeling. Four regressors were trained—Generalized Linear Regression (GLR), Linear Regression, Polynomial Regression (quadratic in MIN\_YEARS\_EXPERIENCE), and Random Forest—and report test RMSE/MAE/R², in addition to coefficient/t-value summaries for interpretability. Data is saved under \_output/.

# --- Setup ---  
from pyspark.sql import SparkSession  
from pyspark.sql import functions as F  
from pyspark.ml.feature import (  
 StringIndexer, OneHotEncoder, OneHotEncoderModel,  
 SQLTransformer, VectorAssembler  
)  
from pyspark.ml import Pipeline  
from pyspark.ml.regression import GeneralizedLinearRegression, LinearRegression, RandomForestRegressor  
  
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns  
import plotly.express as px, plotly.io as pio  
from math import erf, sqrt  
from pathlib import Path  
import os, sys, subprocess  
import warnings  
  
# Hide noisy warnings and printouts globally  
warnings.filterwarnings("ignore", message=".\*set\_ticklabels.\*")  
warnings.filterwarnings("ignore", category=UserWarning)  
warnings.filterwarnings("ignore", category=FutureWarning)  
  
# Required packages for image export  
subprocess.run([sys.executable, "-m", "pip", "install", "kaleido"], check=False)  
  
pio.renderers.default = "plotly\_mimetype"  
sns.set\_theme(style="whitegrid")  
  
# Start Spark  
try:  
 spark  
except NameError:  
 spark = SparkSession.builder.appName("LightcastData").getOrCreate()  
spark.sparkContext.setLogLevel("WARN")  
  
np.random.seed(42)  
OUTPUT\_DIR = Path("\_output")  
OUTPUT\_DIR.mkdir(parents=True, exist\_ok=True)  
  
# print("[OK] Environment initialized")

Requirement already satisfied: kaleido in ./.venv/lib/python3.12/site-packages (1.1.0)  
Requirement already satisfied: choreographer>=1.0.10 in ./.venv/lib/python3.12/site-packages (from kaleido) (1.1.1)  
Requirement already satisfied: logistro>=1.0.8 in ./.venv/lib/python3.12/site-packages (from kaleido) (1.1.0)  
Requirement already satisfied: orjson>=3.10.15 in ./.venv/lib/python3.12/site-packages (from kaleido) (3.11.3)  
Requirement already satisfied: packaging in ./.venv/lib/python3.12/site-packages (from kaleido) (25.0)  
Requirement already satisfied: pytest-timeout>=2.4.0 in ./.venv/lib/python3.12/site-packages (from kaleido) (2.4.0)  
Requirement already satisfied: simplejson>=3.19.3 in ./.venv/lib/python3.12/site-packages (from choreographer>=1.0.10->kaleido) (3.20.2)  
Requirement already satisfied: pytest>=7.0.0 in ./.venv/lib/python3.12/site-packages (from pytest-timeout>=2.4.0->kaleido) (8.4.2)  
Requirement already satisfied: iniconfig>=1 in ./.venv/lib/python3.12/site-packages (from pytest>=7.0.0->pytest-timeout>=2.4.0->kaleido) (2.1.0)  
Requirement already satisfied: pluggy<2,>=1.5 in ./.venv/lib/python3.12/site-packages (from pytest>=7.0.0->pytest-timeout>=2.4.0->kaleido) (1.6.0)  
Requirement already satisfied: pygments>=2.7.2 in ./.venv/lib/python3.12/site-packages (from pytest>=7.0.0->pytest-timeout>=2.4.0->kaleido) (2.19.2)

# Load CSV  
df = (  
 spark.read  
 .option("header", "true")  
 .option("inferSchema", "true")  
 .option("multiLine", "true")  
 .option("escape", "\"")  
 .csv("./data/lightcast\_job\_postings.csv")  
)  
  
# Target and key columns  
candidate\_y = ["Average\_Salary", "SALARY", "AVERAGE\_SALARY"]  
y\_col = next((c for c in candidate\_y if c in df.columns), None)  
if y\_col is None:  
 raise ValueError(f"None of {candidate\_y} found. Available: {df.columns}")  
  
# Use SALARY if present  
if "SALARY" in df.columns:  
 y\_col = "SALARY"  
  
min\_years = "MIN\_YEARS\_EXPERIENCE" if "MIN\_YEARS\_EXPERIENCE" in df.columns else None  
max\_years = "MAX\_YEARS\_EXPERIENCE" if "MAX\_YEARS\_EXPERIENCE" in df.columns else None  
if min\_years is None:  
 raise ValueError("Expected MIN\_YEARS\_EXPERIENCE in dataset for polynomial term.")  
  
# numeric & categorical candidates (kept small on purpose)  
cont\_cols = []  
for c in [max\_years, "DURATION", "SALARY\_FROM"]:  
 if c and c in df.columns:  
 cont\_cols.append(c)  
cat\_cols = [c for c in ["COMPANY\_NAME", "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"] if c in df.columns]  
  
required\_cols = [y\_col, min\_years] + cont\_cols + cat\_cols  
missing = [c for c in required\_cols if c not in df.columns]  
if missing:  
 raise ValueError(f"Missing expected columns: {missing}")  
  
# Cast numerics and filter sanity ranges  
df\_clean = df.dropna(subset=required\_cols)  
for c in [y\_col, min\_years] + [c for c in cont\_cols if c]:  
 df\_clean = df\_clean.withColumn(c, F.col(c).cast("double"))  
  
df\_clean = (  
 df\_clean  
 .filter(F.col(y\_col) > 0) # positive salary  
 .filter(F.col(min\_years) >= 0) # non-negative experience  
)  
  
# print("[OK] Cleaned rows:", df\_clean.count())  
# print("[vars] y:", y\_col)  
# print("[vars] numeric:", cont\_cols + [min\_years])  
# print("[vars] categorical:", cat\_cols)

# Indexing + OneHot Encoder  
indexers = [  
 StringIndexer(inputCol=c, outputCol=f"{c}\_idx",  
 handleInvalid="keep", stringOrderType="frequencyDesc")  
 for c in cat\_cols  
]  
encoder = OneHotEncoder(  
 inputCols=[f"{c}\_idx" for c in cat\_cols],  
 outputCols=[f"{c}\_ohe" for c in cat\_cols],  
 handleInvalid="keep"  
)  
  
# Quadratic term for MIN\_YEARS\_EXPERIENCE  
add\_square = SQLTransformer(  
 statement=f"SELECT \*, POW({min\_years}, 2.0) AS {min\_years}\_SQ FROM \_\_THIS\_\_"  
)  
  
# Numeric label  
add\_label = SQLTransformer(  
 statement=f"SELECT \*, CAST({y\_col} AS DOUBLE) AS label FROM \_\_THIS\_\_"  
)  
  
# Feature Selection and Vectorization of Features  
feature\_cols = cont\_cols + [min\_years] + [f"{c}\_ohe" for c in cat\_cols]  
assembler = VectorAssembler(inputCols=feature\_cols, outputCol="features")  
  
poly\_cols = [min\_years, f"{min\_years}\_SQ"]  
poly\_assembler = VectorAssembler(inputCols=poly\_cols, outputCol="features\_poly")  
  
pipeline = Pipeline(stages=indexers + [encoder, add\_square, add\_label, assembler, poly\_assembler])  
  
model = pipeline.fit(df\_clean)  
full\_df = model.transform(df\_clean)  
  
print("[OK] Pipeline fit complete")  
full\_df.select("label", min\_years, f"{min\_years}\_SQ", "features", "features\_poly").show(5, truncate=False)

[OK] Pipeline fit complete  
+--------+--------------------+-----------------------+------------------------------------------------------+-------------+  
|label |MIN\_YEARS\_EXPERIENCE|MIN\_YEARS\_EXPERIENCE\_SQ|features |features\_poly|  
+--------+--------------------+-----------------------+------------------------------------------------------+-------------+  
|131100.0|2.0 |4.0 |(848,[0,1,2,3,37,839],[2.0,11.0,113400.0,2.0,1.0,1.0])|[2.0,4.0] |  
|136950.0|3.0 |9.0 |(848,[0,1,2,3,7,839],[3.0,28.0,115300.0,3.0,1.0,1.0]) |[3.0,9.0] |  
|136950.0|3.0 |9.0 |(848,[0,1,2,3,7,839],[3.0,28.0,115300.0,3.0,1.0,1.0]) |[3.0,9.0] |  
|104000.0|3.0 |9.0 |(848,[0,1,2,3,107,837],[3.0,8.0,104000.0,3.0,1.0,1.0])|[3.0,9.0] |  
|80000.0 |3.0 |9.0 |(848,[0,1,2,3,21,840],[3.0,37.0,60000.0,3.0,1.0,1.0]) |[3.0,9.0] |  
+--------+--------------------+-----------------------+------------------------------------------------------+-------------+  
only showing top 5 rows

# Prune created dataframe  
from pyspark.sql.functions import monotonically\_increasing\_id  
  
# Ensure a stable key  
if "row\_id" not in full\_df.columns:  
 full\_df = full\_df.withColumn("row\_id", monotonically\_increasing\_id())  
  
# Minimal working set  
minimal\_cols = ["row\_id", "label", "features", "features\_poly",  
 "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE", "DURATION", "SALARY\_FROM"]  
final\_df = full\_df.select([c for c in minimal\_cols if c in full\_df.columns])  
  
# print("[OK] Pruned final\_df columns:", final\_df.columns)

# Train and Split  
train\_df, test\_df = final\_df.randomSplit([0.8, 0.2], seed=42)  
print("[OK] Split sizes:", train\_df.count(), test\_df.count())

[OK] Split sizes: 1848 395

The standard ratio of 80/20 for tabular regression was used. This will balance bias and variances by keeping enough training data while still holding back a meaningful test sample for performance validation.

# GLR (Gaussian / Identity)  
glr = GeneralizedLinearRegression(featuresCol="features", labelCol="label", family="gaussian", link="identity", maxIter=50)  
glr\_model = glr.fit(train\_df)  
glr\_summary = glr\_model.summary  
glr\_test = glr\_model.transform(test\_df)  
  
# Linear Regression (features)  
lr = LinearRegression(featuresCol="features", labelCol="label")  
lr\_model = lr.fit(train\_df)  
lr\_test = lr\_model.transform(test\_df)  
  
# Polynomial Linear Regression (features\_poly)  
poly\_lr = LinearRegression(featuresCol="features\_poly", labelCol="label")  
poly\_model = poly\_lr.fit(train\_df)  
poly\_test = poly\_model.transform(test\_df)  
  
# Random Forest  
rf = RandomForestRegressor(featuresCol="features", labelCol="label", numTrees=200, maxDepth=6, seed=42)  
rf\_model = rf.fit(train\_df)  
rf\_test = rf\_model.transform(test\_df)  
  
print("[OK] All models trained")

[OK] All models trained

# Reconstruct expanded feature names (continuous + OHE columns)  
enc\_model = next(s for s in model.stages if isinstance(s, OneHotEncoderModel))  
drop\_last = enc\_model.getDropLast() if hasattr(enc\_model, "getDropLast") else True  
  
expanded\_features = cont\_cols + [min\_years]  
for input\_col, size\_in in zip(enc\_model.getInputCols(), enc\_model.categorySizes):  
 base = input\_col.replace("\_idx", "")  
 size\_out = int(size\_in) - (1 if drop\_last else 0)  
 expanded\_features += [f"{base}\_ohe\_{i}" for i in range(size\_out)]  
  
print(f"[OK] Expanded feature count = {len(expanded\_features)}")

[OK] Expanded feature count = 846

# Try native SE/t/p; if not present, bootstrap-estimate SE/t/p (SE = standard errors, t = t-statistics, p = p-values)  
coefs = np.array(glr\_model.coefficients.toArray(), dtype=float)  
  
def norm\_cdf(z):   
 return 0.5 \* (1.0 + erf(z / sqrt(2.0)))  
  
use\_native = True  
try:  
 se = np.array(glr\_summary.coefficientStandardErrors, dtype=float)  
 tval= np.array(glr\_summary.tValues, dtype=float)  
 pval= np.array(glr\_summary.pValues, dtype=float)  
except Exception:  
 use\_native = False  
  
if not use\_native:  
 print("[warn] Spark did not provide SE/t/p; estimating via bootstrap...")  
 B = 40  
 boot = []  
 for b in range(B):  
 samp = train\_df.sample(withReplacement=True, fraction=1.0, seed=1234+b)  
 m = GeneralizedLinearRegression(featuresCol="features", labelCol="label",  
 family="gaussian", link="identity", maxIter=50).fit(samp)  
 c = np.array(m.coefficients.toArray(), dtype=float)  
 if len(c) == len(coefs): boot.append(c)  
 boot = np.array(boot)  
 if boot.shape[0] >= 3:  
 se = boot.std(axis=0, ddof=1)  
 with np.errstate(divide='ignore', invalid='ignore'):  
 tval = coefs / se  
 pval = 2.0 \* (1.0 - np.vectorize(norm\_cdf)(np.abs(tval)))  
 else:  
 se = np.full\_like(coefs, np.nan); tval = np.full\_like(coefs, np.nan); pval = np.full\_like(coefs, np.nan)  
  
# Align to names to ensure df can be built   
L = min(len(coefs), len(expanded\_features), len(se), len(tval), len(pval))  
glr\_df = pd.DataFrame({  
 "feature": expanded\_features[:L],  
 "coefficient": coefs[:L],  
 "std\_error": se[:L],  
 "t\_value": tval[:L],  
 "p\_value": pval[:L]  
})  
  
# Intercept row (if native arrays include intercept, they're length L+1; if not, put NaNs)  
int\_se = float(se[-1]) if len(se) >= len(coefs)+1 else np.nan  
int\_tv = float(tval[-1])if len(tval)>= len(coefs)+1 else np.nan  
int\_pv = float(pval[-1])if len(pval)>= len(coefs)+1 else np.nan  
  
glr\_df = pd.concat([glr\_df, pd.DataFrame([{  
 "feature":"\_intercept\_",  
 "coefficient": float(glr\_model.intercept),  
 "std\_error": int\_se,  
 "t\_value": int\_tv,  
 "p\_value": int\_pv  
}])], ignore\_index=True)  
  
# 95% Confidence Interval  
glr\_df["CI\_lower"] = glr\_df["coefficient"] - 1.96\*glr\_df["std\_error"]  
glr\_df["CI\_upper"] = glr\_df["coefficient"] + 1.96\*glr\_df["std\_error"]  
  
glr\_df.to\_csv(OUTPUT\_DIR / "glr\_summary.csv", index=False)  
print("Saved:", OUTPUT\_DIR / "glr\_summary.csv")

[warn] Spark did not provide SE/t/p; estimating via bootstrap...

Saved: \_output/glr\_summary.csv

# Polynomial Linear Regression Summary   
poly\_names = [min\_years, f"{min\_years}\_SQ"]  
  
lr\_summ = poly\_model.summary  
p\_coefs = np.array(poly\_model.coefficients.toArray(), dtype=float)  
p\_se = np.array(lr\_summ.coefficientStandardErrors, dtype=float) if lr\_summ.coefficientStandardErrors else np.full\_like(p\_coefs, np.nan)  
p\_tv = np.array(lr\_summ.tValues, dtype=float) if lr\_summ.tValues else np.full\_like(p\_coefs, np.nan)  
p\_pv = np.array(lr\_summ.pValues, dtype=float) if lr\_summ.pValues else np.full\_like(p\_coefs, np.nan)  
  
poly\_df = pd.DataFrame({  
 "feature": poly\_names[:len(p\_coefs)],  
 "coefficient": p\_coefs[:len(poly\_names)],  
 "std\_error": p\_se[:len(poly\_names)],  
 "t\_value": p\_tv[:len(poly\_names)],  
 "p\_value": p\_pv[:len(poly\_names)]  
})  
  
# Intercept (if arrays include it as an extra element, use it; else NaN)  
p\_int\_se = float(p\_se[-1]) if len(p\_se) >= len(p\_coefs)+1 else np.nan  
p\_int\_tv = float(p\_tv[-1]) if len(p\_tv) >= len(p\_coefs)+1 else np.nan  
p\_int\_pv = float(p\_pv[-1]) if len(p\_pv) >= len(p\_coefs)+1 else np.nan  
  
poly\_df = pd.concat([poly\_df, pd.DataFrame([{  
 "feature": "\_intercept\_",  
 "coefficient": float(poly\_model.intercept),  
 "std\_error": p\_int\_se,  
 "t\_value": p\_int\_tv,  
 "p\_value": p\_int\_pv  
}])], ignore\_index=True)  
  
poly\_df["CI\_lower"] = poly\_df["coefficient"] - 1.96\*poly\_df["std\_error"]  
poly\_df["CI\_upper"] = poly\_df["coefficient"] + 1.96\*poly\_df["std\_error"]  
  
poly\_df.to\_csv(OUTPUT\_DIR / "polylr\_summary.csv", index=False)  
print("Saved:", OUTPUT\_DIR / "polylr\_summary.csv")

Saved: \_output/polylr\_summary.csv

Interpretation of Polynomial Linear Regression. Adding a quadratic term in Min Yrs Exp does not improve generalization for this dataset. The linear term carries most of the signal, the smaller or non-significant t-value for the squared term is suggesting added variance with a limited predictive value.

# T-values significance and top drivers  
ALPHA\_T = 1.96 # quick rule-of-thumb for statistical significance at ~5% (two-sided) under a normal/large-sample approximation  
  
def export\_sig\_tables(df, prefix):  
 df = df.copy()  
 if "t\_value" not in df.columns:  
 print(f"[warn] No t\_value in {prefix}, skipping.")  
 return  
 df["abs\_t"] = df["t\_value"].abs()  
 sig = df[df["abs\_t"] >= ALPHA\_T].sort\_values("abs\_t", ascending=False)  
 top\_pos = df.sort\_values("t\_value", ascending=False).head(15)  
 top\_neg = df.sort\_values("t\_value", ascending=True).head(15)  
 sig.to\_csv(OUTPUT\_DIR / f"{prefix}\_significant.csv", index=False)  
 top\_pos.to\_csv(OUTPUT\_DIR / f"{prefix}\_top\_positive\_t.csv", index=False)  
 top\_neg.to\_csv(OUTPUT\_DIR / f"{prefix}\_top\_negative\_t.csv", index=False)  
 print(f"Saved: {prefix}\_significant/top\_positive\_t/top\_negative\_t")  
  
export\_sig\_tables(glr\_df, "glr")  
export\_sig\_tables(poly\_df, "poly")

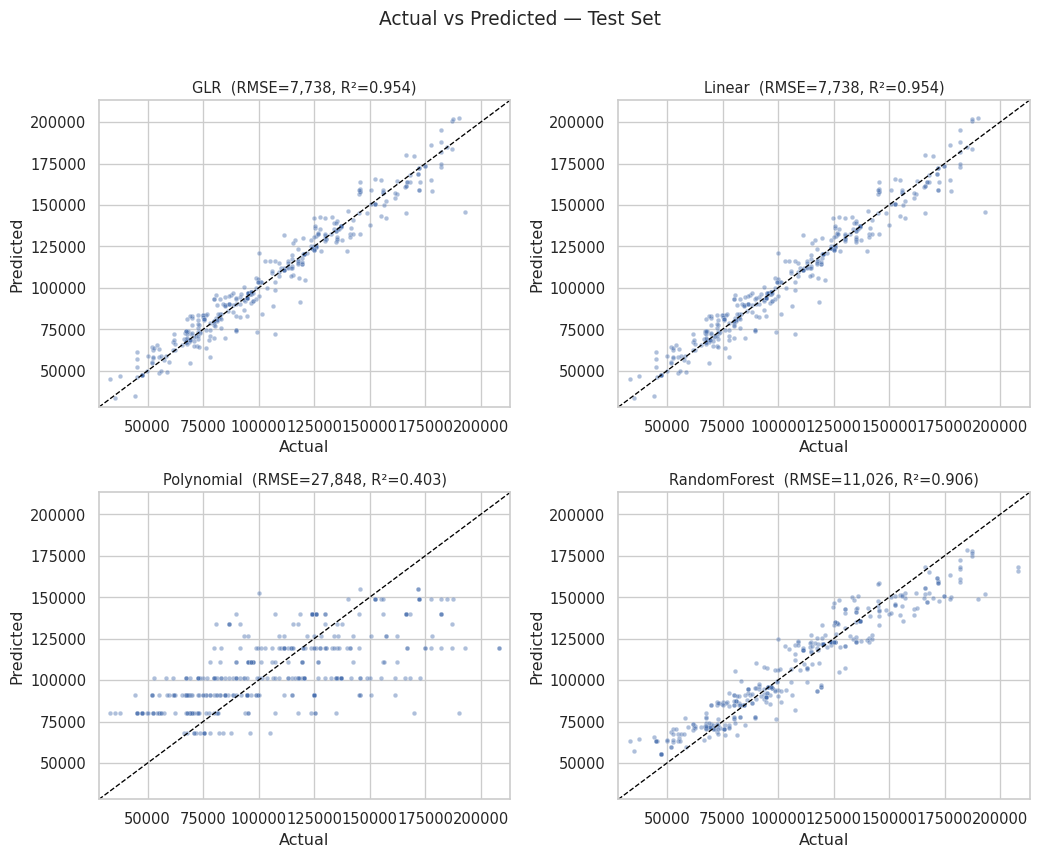
Saved: glr\_significant/top\_positive\_t/top\_negative\_t  
Saved: poly\_significant/top\_positive\_t/top\_negative\_t

# Join predictions on row\_id, maintain a compact table  
p\_glr = glr\_test.select("row\_id", F.col("label").alias("label"), F.col("prediction").alias("pred\_glr"))  
p\_lr = lr\_test.select("row\_id", F.col("prediction").alias("pred\_lr"))  
p\_pr = poly\_test.select("row\_id", F.col("prediction").alias("pred\_poly"))  
p\_rf = rf\_test.select("row\_id", F.col("prediction").alias("pred\_rf"))  
  
joined = p\_glr.join(p\_lr, "row\_id", "inner").join(p\_pr, "row\_id", "inner").join(p\_rf, "row\_id", "inner")  
pdf = joined.select("label","pred\_glr","pred\_lr","pred\_poly","pred\_rf").toPandas().dropna()  
  
def rmse(y, yhat):   
 return float(np.sqrt(np.mean((y - yhat)\*\*2)))  
  
y = pdf["label"].values  
preds = {  
 "GLR": pdf["pred\_glr"].values,  
 "Linear": pdf["pred\_lr"].values,  
 "Polynomial": pdf["pred\_poly"].values,  
 "RandomForest": pdf["pred\_rf"].values  
}  
  
# Parameter counts (linear = k = (# coefficients) + 1) and a proxy for RF (k\_rf = len(expanded\_features) + 1)  
k\_glr = len(glr\_model.coefficients) + 1  
k\_lr = len(lr\_model.coefficients) + 1  
k\_poly = len(poly\_model.coefficients) + 1  
k\_rf = len(expanded\_features) + 1 # heuristic for AIC/BIC proxy  
  
def aic\_bic(y, yhat, k):  
 n = len(y)  
 sse = float(np.sum((y - yhat)\*\*2))  
 if n <= 0 or sse <= 0: return np.nan, np.nan  
 sigma2 = sse / n  
 logL = -0.5 \* n \* (np.log(2\*np.pi\*sigma2) + 1.0)  
 AIC = 2\*k - 2\*logL  
 BIC = k\*np.log(n) - 2\*logL  
 return float(AIC), float(BIC)  
  
rows = []  
for name, yhat in preds.items():  
 k = {"GLR": k\_glr, "Linear": k\_lr, "Polynomial": k\_poly, "RandomForest": k\_rf}[name]  
 R2 = 1 - np.sum((y - yhat)\*\*2) / np.sum((y - np.mean(y))\*\*2) if np.sum((y - np.mean(y))\*\*2) > 0 else np.nan  
 AIC, BIC = aic\_bic(y, yhat, k)  
 rows.append({"Model": name, "RMSE": rmse(y, yhat), "MAE": float(np.mean(np.abs(y - yhat))), "R2": R2, "AIC": AIC, "BIC": BIC})  
  
metrics\_df = pd.DataFrame(rows)  
  
# === GLR exact log-likelihood & BIC ===  
n\_glr = int(glr\_summary.numInstances)  
disp = float(glr\_summary.dispersion)  
dev = float(glr\_summary.deviance)  
logL\_glr = -0.5 \* ( n\_glr \* np.log(2\*np.pi) + n\_glr \* np.log(disp) + dev / disp )  
bic\_glr\_exact = k\_glr \* np.log(n\_glr) - 2.0 \* logL\_glr  
  
# Override GLR BIC with exact formula; add logL column  
metrics\_df.loc[metrics\_df["Model"] == "GLR", "BIC"] = bic\_glr\_exact  
if "logL" not in metrics\_df.columns:  
 metrics\_df["logL"] = np.nan  
metrics\_df.loc[metrics\_df["Model"] == "GLR", "logL"] = logL\_glr  
  
metrics\_df = metrics\_df.sort\_values("RMSE")  
metrics\_df.to\_csv(OUTPUT\_DIR / "metrics\_table.csv", index=False)  
pdf.to\_csv(OUTPUT\_DIR / "predictions\_clean.csv", index=False)  
  
print("Saved:", OUTPUT\_DIR / "metrics\_table.csv")  
print("Saved:", OUTPUT\_DIR / "predictions\_clean.csv")  
metrics\_df

Saved: \_output/metrics\_table.csv  
Saved: \_output/predictions\_clean.csv

|  | Model | RMSE | MAE | R2 | AIC | BIC | logL |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | GLR | 7737.758101 | 5163.626094 | 0.953926 | 9892.516587 | 42929.874096 | -18271.907796 |
| 1 | Linear | 7737.758101 | 5163.626094 | 0.953926 | 9892.516587 | 13270.590602 | NaN |
| 3 | RandomForest | 11026.262059 | 8246.128017 | 0.906442 | 10168.309223 | 13538.425465 | NaN |
| 2 | Polynomial | 27847.577892 | 22108.809590 | 0.403238 | 9212.217446 | 9224.154103 | NaN |

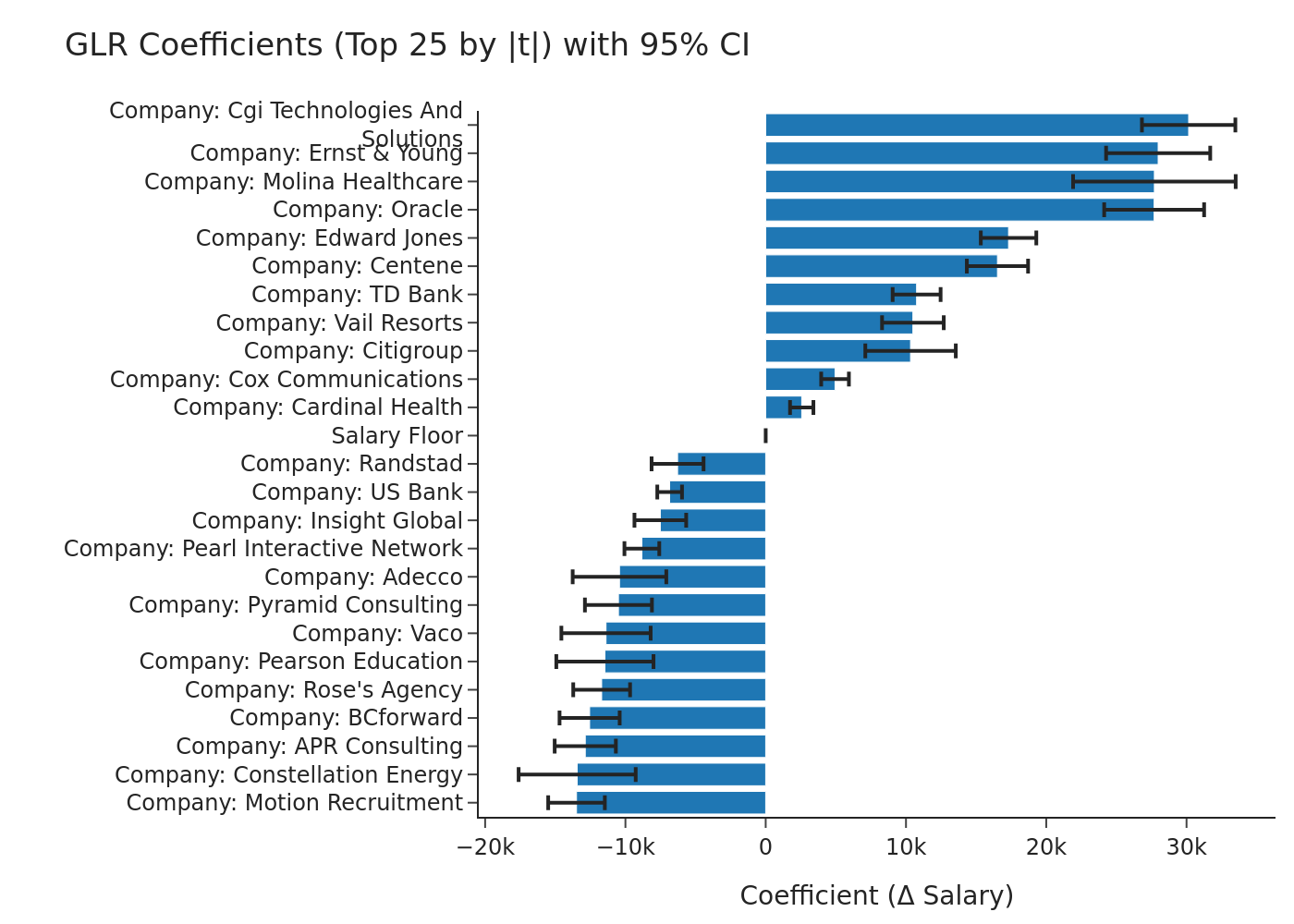
# Actual vs Predicted  
import seaborn as sns  
sns.set\_theme(style="whitegrid")  
  
models\_order = ["GLR", "Linear", "Polynomial", "RandomForest"]  
ymin, ymax = float(np.min(y)), float(np.max(y))  
pad = 0.03 \* (ymax - ymin)  
lo, hi = ymin - pad, ymax + pad  
  
fig, axes = plt.subplots(2, 2, figsize=(11, 9))  
axes = axes.ravel()  
  
for ax, m in zip(axes, models\_order):  
 yhat = preds[m]  
 n = len(y)  
 idx = np.arange(n if n <= 4000 else 4000)  
 if n > 4000:  
 idx = np.random.choice(n, 4000, replace=False)  
 sns.scatterplot(x=y[idx], y=yhat[idx], s=12, alpha=0.45, ax=ax, legend=False)  
 ax.plot([lo, hi], [lo, hi], linestyle="--", linewidth=1, color="black")  
 ax.set\_xlim(lo, hi); ax.set\_ylim(lo, hi)  
 rmse\_val = float(np.sqrt(np.mean((y - yhat)\*\*2)))  
 r2\_val = 1 - np.sum((y - yhat)\*\*2) / np.sum((y - np.mean(y))\*\*2)  
 ax.set\_title(f"{m} (RMSE={rmse\_val:,.0f}, R²={r2\_val:.3f})", pad=6, fontsize=11)  
 ax.set\_xlabel("Actual"); ax.set\_ylabel("Predicted")  
  
plt.suptitle("Actual vs Predicted — Test Set", fontsize=14, y=0.99)  
plt.tight\_layout(rect=[0,0,1,0.97])  
plt.savefig(OUTPUT\_DIR / "actual\_vs\_pred\_2x2.png", dpi=300, bbox\_inches="tight")  
plt.show()  
print("Saved:", OUTPUT\_DIR / "actual\_vs\_pred\_2x2.png")



Saved: \_output/actual\_vs\_pred\_2x2.png

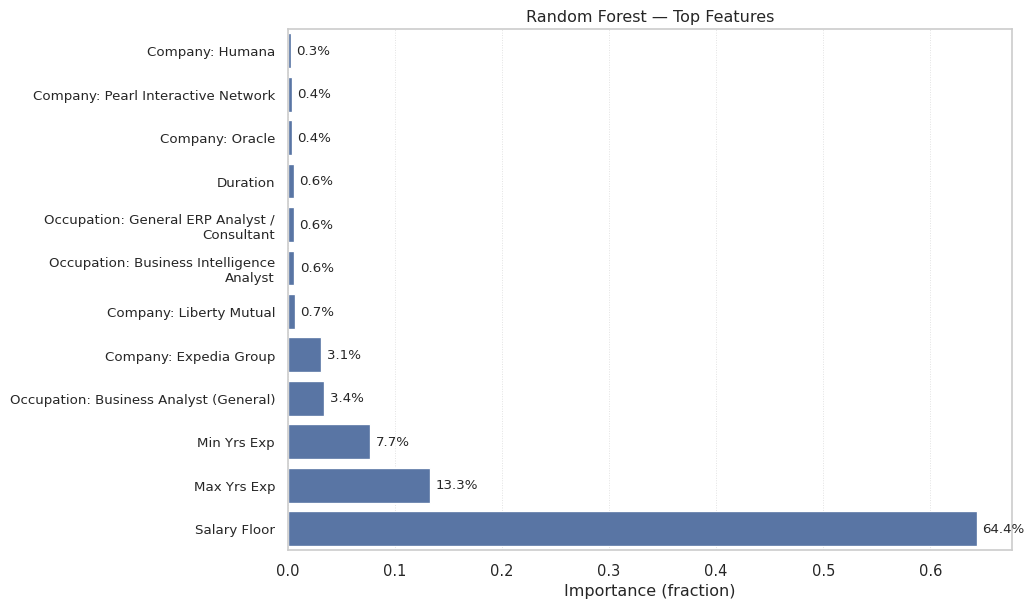
Interpretation of Model Comparison. All four models achieve similar predictive and performance accuracy. GLR and Random Forest show the lowest RMSE and comparable R². Essentially what this tells us is that the salary grows at a steady pace with experience and company type, without any drastic curves or complex variable interactions.

# GLR coefficients with CI   
import os, textwrap  
import numpy as np, pandas as pd  
from IPython.display import Image, display  
from pathlib import Path  
from pyspark.ml.feature import StringIndexerModel  
import plotly.express as px  
import plotly.io as pio  
  
TOP\_N = 25 # show top 25 effects by |t|  
WRAP\_CHARS = 36 # wrap long labels for readability  
  
# map original categorical col -> StringIndexerModel  
indexer\_models = {}  
for s in model.stages:  
 if isinstance(s, StringIndexerModel):  
 indexer\_models[s.getInputCol()] = s  
  
def humanize\_base(col):  
 col = col.replace("\_idx","").replace("\_ohe","")  
 col = col.replace("LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME", "Occupation")  
 col = col.replace("COMPANY\_NAME", "Company")  
 col = col.replace("MIN\_YEARS\_EXPERIENCE", "Min Yrs Exp")  
 col = col.replace("MAX\_YEARS\_EXPERIENCE", "Max Yrs Exp")  
 col = col.replace("DURATION", "Duration")  
 col = col.replace("SALARY\_FROM", "Salary Floor")  
 return col  
  
# Turn OHE feature names like "Company\_ohe\_12" into "Company: <label>"  
pretty\_names = []  
for f in glr\_df["feature"]:  
 if f == "\_intercept\_":  
 pretty\_names.append("Intercept")  
 continue  
 if "\_ohe\_" in f:  
 base, idx = f.split("\_ohe\_")  
 base\_h = humanize\_base(base)  
 idx\_model = indexer\_models.get(base)  
 label = None  
 try:  
 ii = int(idx)  
 if idx\_model is not None and ii < len(idx\_model.labels):  
 label = idx\_model.labels[ii]  
 except Exception:  
 pass  
 label = label if label else f"level\_{idx}"  
 pretty\_names.append(f"{base\_h}: {label}")  
 else:  
 pretty\_names.append(humanize\_base(f))  
  
plot\_df = glr\_df.copy()  
plot\_df["pretty"] = pretty\_names  
  
# choose top by |t| when available; otherwise by |coef|  
if "t\_value" in plot\_df.columns and plot\_df["t\_value"].notna().any():  
 plot\_df = plot\_df.assign(abs\_t=plot\_df["t\_value"].abs()).sort\_values("abs\_t", ascending=False)  
else:  
 plot\_df = plot\_df.assign(abs\_c=plot\_df["coefficient"].abs()).sort\_values("abs\_c", ascending=False)  
  
plot\_df = plot\_df.head(TOP\_N).sort\_values("coefficient") # plot from negative to positive  
plot\_df["pretty\_wrapped"] = plot\_df["pretty"].apply(lambda s: "<br>".join(textwrap.wrap(s, WRAP\_CHARS))) # wrap labels  
  
fig\_coef = px.bar(  
 plot\_df,  
 x="coefficient", y="pretty\_wrapped", orientation="h",  
 title=f"GLR Coefficients (Top {TOP\_N} by |t|) with 95% CI",  
 error\_x=plot\_df["CI\_upper"] - plot\_df["coefficient"],  
 error\_x\_minus=plot\_df["coefficient"] - plot\_df["CI\_lower"],  
 template="simple\_white",  
)  
fig\_coef.update\_layout(  
 margin=dict(l=10, r=10, t=60, b=10),  
 xaxis\_title="Coefficient (Δ Salary)",  
 yaxis\_title="",  
)  
fig\_coef.update\_traces(hovertemplate="<b>%{y}</b><br>Coef=%{x:.2f}<extra></extra>")  
  
OUTPUT\_DIR = Path("\_output")  
OUTPUT\_DIR.mkdir(parents=True, exist\_ok=True)  
  
# Save static + html  
png\_path = OUTPUT\_DIR / "glr\_coefficients\_ci.png"  
html\_path = OUTPUT\_DIR / "glr\_coefficients\_ci.html"  
  
pio.write\_html(fig\_coef, str(html\_path), include\_plotlyjs="cdn")  
  
pio.write\_image(fig\_coef, str(png\_path), scale=2)  
  
assert png\_path.exists() and png\_path.stat().st\_size > 0, (  
 "Failed to create \_output/glr\_coefficients\_ci.png. "  
 "Make sure kaleido is installed in the Quarto Python environment."  
)  
  
display(Image(filename=str(png\_path)))



Interpretation of GLR. The model shows that experience and the salary floor variable have the strongest positive correlation with salary, while several company/occupation levels contribute smaller adjustments around that baseline. Narrow confidence intervals and large absolute t-values show stable effects for the main drivers. While, wide intervals flag sparse categories where there is less certainity in these estimates.

# Random Forest   
import numpy as np, pandas as pd, matplotlib.pyplot as plt, textwrap  
from pyspark.ml.feature import OneHotEncoderModel, StringIndexerModel  
import seaborn as sns  
  
TOP\_K = 12  
WRAP\_CHARS = 38  
  
# 1) pull encoder + indexers  
enc\_model = next(s for s in model.stages if isinstance(s, OneHotEncoderModel))  
drop\_last = enc\_model.getDropLast() if hasattr(enc\_model, "getDropLast") else True  
  
indexer\_models = {}  
for s in model.stages:  
 if isinstance(s, StringIndexerModel):  
 indexer\_models[s.getInputCol()] = s  
  
# 2) humanizing helpers  
def humanize\_base(col):  
 col = col.replace("\_idx","").replace("\_ohe","")  
 col = col.replace("LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME", "Occupation")  
 col = col.replace("COMPANY\_NAME", "Company")  
 col = col.replace("MIN\_YEARS\_EXPERIENCE", "Min Yrs Exp")  
 col = col.replace("MAX\_YEARS\_EXPERIENCE", "Max Yrs Exp")  
 col = col.replace("DURATION", "Duration")  
 col = col.replace("SALARY\_FROM", "Salary Floor")  
 return col  
  
# assembler order  
assembler\_inputs = cont\_cols + [min\_years] + [f"{c}\_ohe" for c in cat\_cols]  
  
# emitted width for each OHE output after dropLast  
ohe\_width = {}  
for input\_c, output\_c, sz in zip(enc\_model.getInputCols(), enc\_model.getOutputCols(), enc\_model.categorySizes):  
 ohe\_width[output\_c] = int(sz) - (1 if drop\_last else 0)  
  
# Build pretty names to match the vector order used by RF  
pretty\_feature\_names = []  
for c in assembler\_inputs:  
 if c.endswith("\_ohe"):  
 base\_col = c.replace("\_ohe","")  
 idx\_model = indexer\_models.get(base\_col)  
 labels = idx\_model.labels if idx\_model is not None else []  
 width = ohe\_width.get(c, 0)  
 for i in range(width):  
 label\_i = labels[i] if i < len(labels) else f"level\_{i}"  
 pretty\_feature\_names.append(f"{humanize\_base(base\_col)}: {label\_i}")  
 else:  
 pretty\_feature\_names.append(humanize\_base(c))  
  
# 3) align & pick top-K  
imps = np.array(rf\_model.featureImportances.toArray(), dtype=float)  
L = min(len(pretty\_feature\_names), len(imps))  
names = pretty\_feature\_names[:L]  
imps = imps[:L]  
  
fi = (  
 pd.DataFrame({"feature": names, "importance": imps})  
 .sort\_values("importance", ascending=False)  
 .head(TOP\_K)  
 .sort\_values("importance", ascending=True) # nicer bars bottom-up  
)  
  
# 4) plot   
plt.figure(figsize=(11, 6.5))  
ax = sns.barplot(data=fi, y="feature", x="importance", orient="h")  
ax.grid(axis="x", linestyle=":", linewidth=0.7, alpha=0.6)  
ax.set\_xlabel("Importance (fraction)")  
ax.set\_ylabel("")  
ax.set\_title("Random Forest — Top Features")  
  
# wrap labels  
wrapped = ["\n".join(textwrap.wrap(lbl, WRAP\_CHARS)) for lbl in fi["feature"]]  
ax.set\_yticklabels(wrapped, fontsize=10)  
  
# annotate percents  
for p, val in zip(ax.patches, fi["importance"].to\_numpy()):  
 width = p.get\_width()  
 ax.text(width + 0.005, p.get\_y() + p.get\_height()/2,  
 f"{100.0\*val:,.1f}%", va="center", fontsize=10)  
  
plt.subplots\_adjust(left=0.36)  
plt.tight\_layout()  
plt.savefig(OUTPUT\_DIR / "rf\_feature\_importance.png", dpi=300, bbox\_inches="tight")  
plt.show()  
print("Saved:", OUTPUT\_DIR / "rf\_feature\_importance.png")



Saved: \_output/rf\_feature\_importance.png

Interpretation of Random Forest Importances. Random Forest confirms what the linear models above showed, salary floor and experience are the dominant predictors, with company and occupation contributing small tweaks. This suggests the main structure is additive, with limited non-linear gains from tree splits.