Assignment 04

Norah Jones

November 21, 2024

# 1. Data Preparartion

## 1.1 Load data and review

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("data/lightcast\_job\_postings.csv")  
  
# Show Schema and Sample Data  
#print("---This is Diagnostic check, No need to print it in the final doc---")  
  
#df.printSchema() # comment this line when rendering the submission  
#df.show(5)

WARNING: Using incubator modules: jdk.incubator.vector  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/17 14:43:41 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
25/10/17 14:43:42 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.  
[Stage 1:> (0 + 1) / 1]

### 1.1.1 Pick salary as target variable, state name, NAICS2\_NAME name, remote type name, employment type name, city name, education levels name, min years experience , duration will be indepent variables for the analysis.

### 1.1.2 For min years experience and duration fill na with 0.

df = df.select("SALARY","STATE\_NAME","NAICS2\_NAME", "EDUCATION\_LEVELS\_NAME", "MIN\_YEARS\_EXPERIENCE", "DURATION")  
#df.show()  
df = df.na.fill({"MIN\_YEARS\_EXPERIENCE": 0, "DURATION":0})  
#df.show()

### 1.1.3 Visualize the nas to understand the magnitude. Over 50% of salary is na, remove those values.

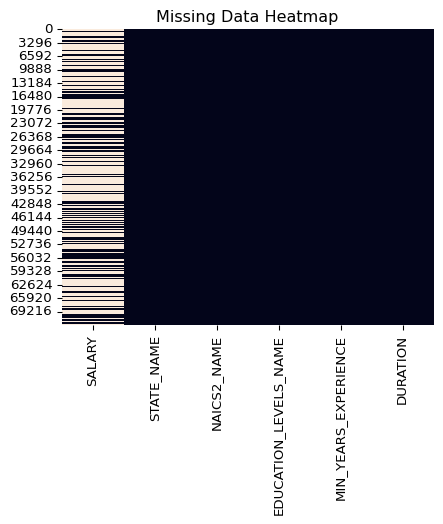
df\_pd = df.toPandas()  
#df\_pd.head(5)  
(df\_pd.isna().sum() / len(df\_pd)) \* 100

[Stage 2:> (0 + 1) / 1]

SALARY 57.505035  
STATE\_NAME 0.060691  
NAICS2\_NAME 0.060691  
EDUCATION\_LEVELS\_NAME 0.060691  
MIN\_YEARS\_EXPERIENCE 0.000000  
DURATION 0.000000  
dtype: float64

### 1.1.4 Using seaborn review a heat map of NA values. The independant variables are whole but salary is missing data.

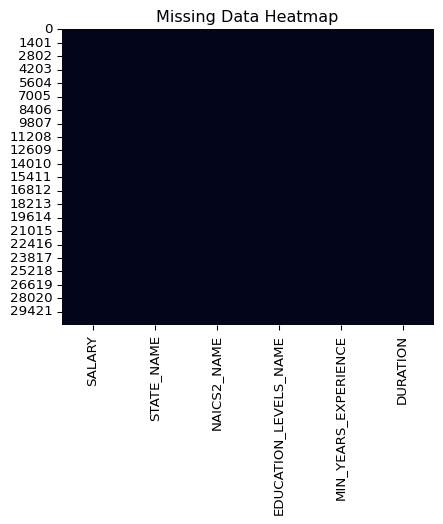
sns.heatmap(df\_pd.isna(), cbar=False)  
plt.title("Missing Data Heatmap")  
plt.show()



### 1.1.5 Drop all records where salary is NA.

df = df.na.drop(subset=["SALARY"])  
df\_pd = df.toPandas()  
  
sns.heatmap(df\_pd.isna(), cbar=False)  
plt.title("Missing Data Heatmap")  
plt.show()

[Stage 3:> (0 + 1) / 1]



### 1.1.6 The data is cleaned and is ready for modeling.

(df\_pd.isna().sum() / len(df\_pd)) \* 100

SALARY 0.0  
STATE\_NAME 0.0  
NAICS2\_NAME 0.0  
EDUCATION\_LEVELS\_NAME 0.0  
MIN\_YEARS\_EXPERIENCE 0.0  
DURATION 0.0  
dtype: float64

# 2. Feature Engineering

## 2.1 First take the input variables and split into numeric and non numeric goups. State name, education levels, NAICS\_NAME are all categoric variables. Min years experience and duration are numeric.

# Suppose you have these columns  
categorical\_cols = ["STATE\_NAME", "EDUCATION\_LEVELS\_NAME","NAICS2\_NAME"]  
numeric\_cols = ["MIN\_YEARS\_EXPERIENCE", "DURATION"]

### 2.1.1 For the categorical columns, assign index values to each column and then one hot encode the columns as a vector.

indexers = [StringIndexer(inputCol=col, outputCol=f"{col}\_indexed") for col in categorical\_cols]  
encoders = [OneHotEncoder(inputCols=[f"{col}\_indexed"], outputCols=[f"{col}\_encoded"]) for col in categorical\_cols]

### 2.1.2 Next compile the one hot encoded columns with the numeric columns in a vector to be used in feature modeling.

assembler\_inputs = [f"{col}\_encoded" for col in categorical\_cols] + numeric\_cols  
assembler = VectorAssembler(inputCols=assembler\_inputs, outputCol="features")

### 2.1.3 Store these data preparation steps as a pipeline for further use

pipeline = Pipeline(stages=indexers + encoders + [assembler])

### 2.1.4 For polynomial square min years experience

from pyspark.sql.functions import col, pow  
  
df\_poly = df.withColumn("MIN\_YEARS\_EXPERIENCE\_SQ", pow(col("MIN\_YEARS\_EXPERIENCE"), 2))

### 2.1.5 Assemble vector using min years and min years experience for polynomial features.

from pyspark.ml.feature import VectorAssembler  
  
assembler\_poly = VectorAssembler(  
 inputCols=["MIN\_YEARS\_EXPERIENCE", "MIN\_YEARS\_EXPERIENCE\_SQ"],  
 outputCol="features\_poly"  
)  
  
df\_poly = assembler\_poly.transform(df\_poly)

df\_poly.printSchema()

root  
 |-- SALARY: integer (nullable = true)  
 |-- STATE\_NAME: string (nullable = true)  
 |-- NAICS2\_NAME: string (nullable = true)  
 |-- EDUCATION\_LEVELS\_NAME: string (nullable = true)  
 |-- MIN\_YEARS\_EXPERIENCE: integer (nullable = false)  
 |-- DURATION: integer (nullable = false)  
 |-- MIN\_YEARS\_EXPERIENCE\_SQ: double (nullable = false)  
 |-- features\_poly: vector (nullable = true)

#df\_poly.show()

### 2.1.6 Now split the data for training and testing in a 70/30% split.

train\_df, test\_df = df.randomSplit([0.7, 0.3], seed=42)

### 2.1.7 Use the previusly created pipeline the prepare the training and test data for use.

pipeline\_model = pipeline.fit(train\_df)  
train\_ready = pipeline\_model.transform(train\_df)  
test\_ready = pipeline\_model.transform(test\_df)

[Stage 4:> (0 + 1) / 1] [Stage 10:> (0 + 1) / 1] [Stage 16:> (0 + 1) / 1]

### 2.1.8 Parse target variable and features vector for modeling.

train\_ready = train\_ready["SALARY","features"]  
#train\_ready.show()  
  
test\_ready = test\_ready["SALARY","features"]

### 2.1.9 Confirm schema for linear regression model.

train\_ready.printSchema()

root  
 |-- SALARY: integer (nullable = true)  
 |-- features: vector (nullable = true)

# 3. Modeling

## 3.1 Build and fit model

### 3.1.1 Penalalize large coefficients and keep all features. Allow for intercept and standardize the data accross variables.

lrm = LinearRegression(  
 featuresCol="features",  
 labelCol="SALARY",  
 predictionCol="prediction",  
 maxIter=100,  
 regParam=0.1,  
 elasticNetParam=0.0,  
 fitIntercept=True,  
 standardization=True,  
)  
  
modelLR = lrm.fit(train\_ready)

[Stage 22:> (0 + 1) / 1] [Stage 23:> (0 + 1) / 1]

## 3.2 Linear Regression Model Summary on Train and test data

summary = modelLR.summary

### 3.2.1 Issue existed with spillover so control with se, tvals, pvals. For features, loop through for only the number of coefficients so to avoid spillover.

### 3.2.2 Linear regression model performed well. All features had statistically significant p values, moderately large t values, reasonably equivalent coefficient magnitudes with respect to salary, which is reflected in the stndard error being significantly smaller than the respective coefficient.

se = summary.coefficientStandardErrors[1:]  
tvals = summary.tValues[1:]  
pvals = summary.pValues[1:]  
  
coef\_df = pd.DataFrame({  
 "Feature": [f"feature\_{i+1}" for i in range(len(modelLR.coefficients))],  
 "Coefficient": modelLR.coefficients.toArray(),  
 "StdError": se,  
 "tValue": tvals,  
 "pValue": pvals  
})  
  
coef\_df.head(100)

|  | Feature | Coefficient | StdError | tValue | pValue |
| --- | --- | --- | --- | --- | --- |
| 0 | feature\_1 | 27160.771987 | 5586.524673 | 3.197485 | 1.388322e-03 |
| 1 | feature\_2 | 17862.826140 | 5604.708248 | 3.970201 | 7.204534e-05 |
| 2 | feature\_3 | 22251.815915 | 5628.321573 | 2.900919 | 3.724464e-03 |
| 3 | feature\_4 | 16327.306545 | 5639.799141 | 3.565083 | 3.645294e-04 |
| 4 | feature\_5 | 20106.349447 | 5650.125035 | 3.825928 | 1.306483e-04 |
| ... | ... | ... | ... | ... | ... |
| 93 | feature\_94 | -13950.827368 | 9974.182772 | -0.376460 | 7.065783e-01 |
| 94 | feature\_95 | -3754.885150 | 10060.013381 | 0.278668 | 7.805023e-01 |
| 95 | feature\_96 | 2803.403880 | 73.807420 | 74.588198 | 0.000000e+00 |
| 96 | feature\_97 | 5505.162463 | 15.692949 | -6.188578 | 6.180532e-10 |
| 97 | feature\_98 | -97.117036 | 36792.595281 | 0.164635 | 8.692328e-01 |

## 3.3 Now validate and run the model using the test data

testmodelLR = lrm.fit(test\_ready)

[Stage 24:> (0 + 1) / 1] [Stage 25:> (0 + 1) / 1]

testsummary = testmodelLR.summary  
  
t\_values = summary.tValues  
p\_values = summary.pValues  
  
stats\_df = pd.DataFrame({  
 "t\_value": t\_values,  
 "p\_value": p\_values  
})  
  
desc\_stats = stats\_df.describe()  
print(desc\_stats)

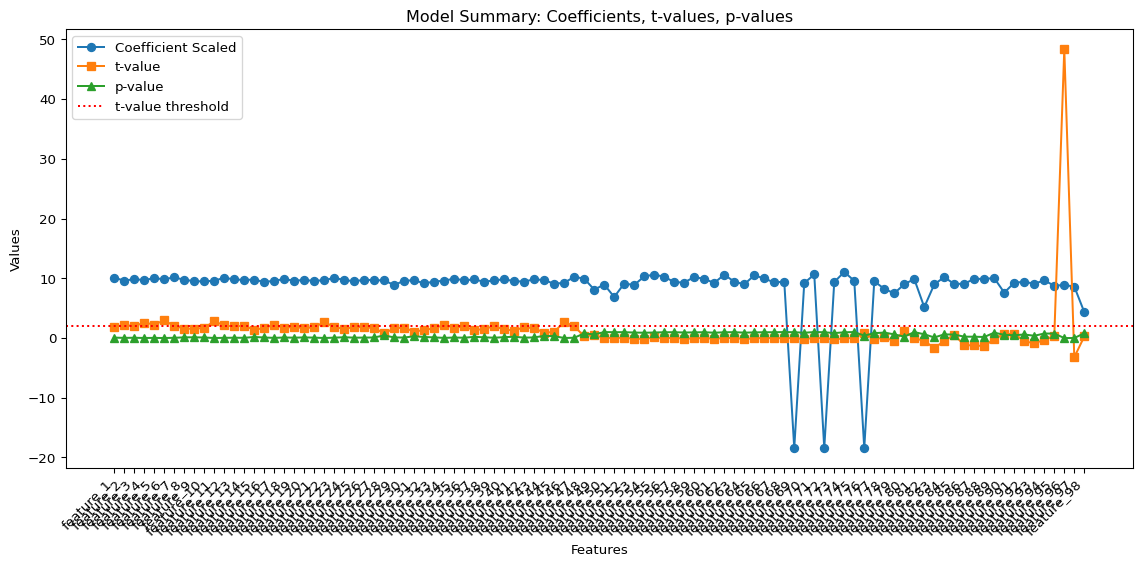
t\_value p\_value  
count 99.000000 99.000000  
mean 2.496936 0.176394  
std 7.503150 0.265126  
min -6.188578 0.000000  
25% 0.898590 0.003787  
50% 2.039629 0.036848  
75% 2.887619 0.279912  
max 74.588198 0.981592

### 3.3.1 Issue existed with spillover so control with se, tvals, pvals. For features, loop through for only the number of coefficients so to avoid spillover.

### 3.3.2 The linear regression model performed okay on the test data. Accross the 99 features, the average p value was not significant, however the median value was significant at appx. .037 - the 75th percentile is not significant but the 25th is. THere were a few features that produced extreme results in all of the t value (~-6/74), p value (0/.98), and coefficient (a few spikes). R squared states ~35% of variation in salary can be explained by the features in the data. On average predictions are off by ~$35,834.

se = testsummary.coefficientStandardErrors[1:]  
tvals = testsummary.tValues[1:]  
pvals = testsummary.pValues[1:]  
  
testcoef\_df = pd.DataFrame({  
 "Feature": [f"feature\_{i+1}" for i in range(len(testmodelLR.coefficients))],  
 "Coefficient": testmodelLR.coefficients.toArray(),  
 "StdError": se,  
 "tValue": tvals,  
 "pValue": pvals  
})  
  
print("R2:", testsummary.r2)  
print("RMSE:", testsummary.rootMeanSquaredError)  
  
#coef\_df.head(100)  
# Scale coefficients to t-values range  
testcoef\_df["Coefficient\_log"] = np.log(np.abs(testcoef\_df["Coefficient"]) + 1e-8)   
  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(12,6))  
  
# x-axis = features  
x = testcoef\_df["Feature"]  
  
# plot each statistic as a line  
plt.plot(x, testcoef\_df["Coefficient\_log"], marker='o', label="Coefficient Scaled")  
plt.plot(x, testcoef\_df["tValue"], marker='s', label="t-value")  
plt.plot(x, testcoef\_df["pValue"], marker='^', label="p-value")  
  
plt.axhline(y=2, color='red', linestyle=':', label='t-value threshold')  
  
plt.xticks(rotation=45, ha='right') # rotate feature names for readability  
plt.xlabel("Features")  
plt.ylabel("Values")  
plt.title("Model Summary: Coefficients, t-values, p-values")  
plt.legend()  
plt.tight\_layout()  
plt.show()

R2: 0.35636023407578243  
RMSE: 35834.08039301742



# 4. Polynomial Regression

from pyspark.ml.feature import PolynomialExpansion, VectorAssembler

## 4.1 Combine numeric and non numeric columns.

assembler\_poly = VectorAssembler(  
 inputCols=["MIN\_YEARS\_EXPERIENCE", "DURATION"],  
 outputCol="numeric\_features"  
)  
  
#expand to polynomial  
poly\_expansion = PolynomialExpansion(  
 degree=2,   
 inputCol="numeric\_features",   
 outputCol="poly\_features"  
)

### 4.1.1 Use the pipeline created earlier to prepare the polynomial features.

assembler\_final = VectorAssembler(  
 inputCols=["poly\_features"] + [f"{col}\_encoded" for col in categorical\_cols],  
 outputCol="features"  
)  
  
poly\_pipeline = Pipeline(stages=indexers + encoders + [assembler\_poly, poly\_expansion, assembler\_final])

### 4.1.2 Fit model and apply same parameters as the linear regression. ~36.8% of variation in salary can be explained by the feature variables. And on average the prediction was off by $36,025.

poly\_model = poly\_pipeline.fit(train\_df)  
train\_poly = poly\_model.transform(train\_df)  
test\_poly = poly\_model.transform(test\_df)  
  
lrm\_poly = LinearRegression(  
 featuresCol="features",  
 labelCol="SALARY",  
 predictionCol="prediction",  
 maxIter=100,  
 regParam=0.1,  
 elasticNetParam=0.0  
)  
  
model\_poly = lrm\_poly.fit(train\_poly)  
summary\_poly = model\_poly.summary  
  
print("R2:", summary\_poly.r2)  
print("RMSE:", summary\_poly.rootMeanSquaredError)

[Stage 26:> (0 + 1) / 1] [Stage 32:> (0 + 1) / 1] [Stage 38:> (0 + 1) / 1] [Stage 44:> (0 + 1) / 1] [Stage 45:> (0 + 1) / 1] 25/10/17 14:45:23 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.  
[Stage 46:> (0 + 1) / 1] [Stage 47:> (0 + 1) / 1]

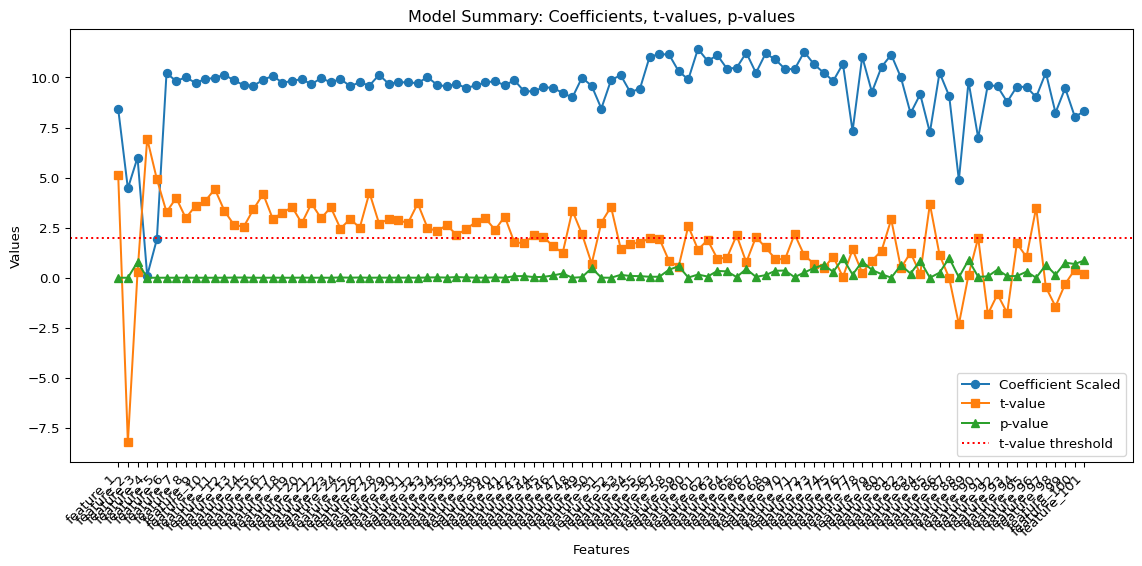
R2: 0.36831696100365496  
RMSE: 36025.51327867095

se = summary\_poly.coefficientStandardErrors[1:]  
tvals = summary\_poly.tValues[1:]  
pvals = summary\_poly.pValues[1:]  
  
coef\_df = pd.DataFrame({  
 "Feature": [f"feature\_{i+1}" for i in range(len(model\_poly.coefficients))],  
 "Coefficient": model\_poly.coefficients.toArray(),  
 "StdError": se,  
 "tValue": tvals,  
 "pValue": pvals  
})  
  
coef\_df.head()

|  | Feature | Coefficient | StdError | tValue | pValue |
| --- | --- | --- | --- | --- | --- |
| 0 | feature\_1 | 4534.036373 | 16.938884 | 5.113944 | 3.182049e-07 |
| 1 | feature\_2 | 86.624507 | 49.042361 | -8.224292 | 2.220446e-16 |
| 2 | feature\_3 | -403.338710 | 4.098090 | 0.271263 | 7.861914e-01 |
| 3 | feature\_4 | 1.111660 | 0.982271 | 6.941764 | 3.982370e-12 |
| 4 | feature\_5 | 6.818695 | 5553.886969 | 4.923227 | 8.575835e-07 |

### 4.1.3 Overall the polynomial model performed worse than the linear regression model. All of the p value, t value, and coefficients experience high volatilitity and varying results.

# Scale coefficients with log  
coef\_df["Coefficient\_log"] = np.log(np.abs(coef\_df["Coefficient"]) + 1e-8)   
  
plt.figure(figsize=(12,6))  
  
# x-axis = features  
x = coef\_df["Feature"]  
  
# plot each statistic as a line  
plt.plot(x, coef\_df["Coefficient\_log"], marker='o', label="Coefficient Scaled")  
plt.plot(x, coef\_df["tValue"], marker='s', label="t-value")  
plt.plot(x, coef\_df["pValue"], marker='^', label="p-value")  
  
plt.axhline(y=2, color='red', linestyle=':', label='t-value threshold')  
  
plt.xticks(rotation=45, ha='right') # rotate feature names for readability  
plt.xlabel("Features")  
plt.ylabel("Values")  
plt.title("Model Summary: Coefficients, t-values, p-values")  
plt.legend()  
plt.tight\_layout()  
plt.show()



# 5. Random Forest

## 5.1 Setup model

### 5.1.1 100 trees, 10 levels of depth to each branch, max of 18 bins for feature categorization.

from pyspark.ml.regression import RandomForestRegressor  
  
rf = RandomForestRegressor(  
 labelCol="SALARY",  
 featuresCol="features",  
 numTrees=100,   
 maxDepth=10,   
 maxBins=18,   
 seed=42  
)

### 5.1.2 Like the other models, assemble the data and prepare it using the pipeline setup earlier in the project.

rf\_pipeline = Pipeline(stages=indexers + encoders + [assembler, rf])

### 5.1.3 Train and then make predictions on test data.

rf\_model = rf\_pipeline.fit(train\_df)  
predictions = rf\_model.transform(test\_df)

[Stage 48:> (0 + 1) / 1] [Stage 54:> (0 + 1) / 1] [Stage 60:> (0 + 1) / 1] [Stage 66:> (0 + 1) / 1] [Stage 67:> (0 + 1) / 1] [Stage 68:> (0 + 1) / 1] [Stage 70:> (0 + 1) / 1] [Stage 72:> (0 + 1) / 1] [Stage 74:> (0 + 1) / 1] [Stage 76:> (0 + 1) / 1] [Stage 78:> (0 + 1) / 1] 25/10/17 14:46:20 WARN DAGScheduler: Broadcasting large task binary with size 1319.0 KiB  
[Stage 80:> (0 + 1) / 1] 25/10/17 14:46:23 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB  
[Stage 82:> (0 + 1) / 1] 25/10/17 14:46:27 WARN DAGScheduler: Broadcasting large task binary with size 3.4 MiB  
[Stage 84:> (0 + 1) / 1][Stage 85:> (0 + 1) / 1] 25/10/17 14:46:30 WARN DAGScheduler: Broadcasting large task binary with size 5.1 MiB  
[Stage 86:> (0 + 1) / 1][Stage 87:> (0 + 1) / 1] 25/10/17 14:46:35 WARN DAGScheduler: Broadcasting large task binary with size 7.4 MiB  
[Stage 88:> (0 + 1) / 1][Stage 89:> (0 + 1) / 1]

### 5.1.4 Initial results suggest the random forest performed reasonably well. The r squared suggests ~43% of variation in salary can be explained by the features in the forest. Error is comparable to the other models as on average the prediction of salary was $25,175 off.

evaluator = RegressionEvaluator(  
 labelCol="SALARY",  
 predictionCol="prediction",  
 metricName="r2"  
)  
  
r2 = evaluator.evaluate(predictions)  
rmse = RegressionEvaluator(  
 labelCol="SALARY", predictionCol="prediction", metricName="rmse"  
).evaluate(predictions)  
  
mae = RegressionEvaluator(  
 labelCol="SALARY", predictionCol="prediction", metricName="mae"  
).evaluate(predictions)  
  
print(f"R²: {r2:.3f}")  
print(f"RMSE: {rmse:.3f}")  
print(f"MAE: {mae:.3f}")

[Stage 90:> (0 + 1) / 1] [Stage 91:> (0 + 1) / 1] [Stage 92:> (0 + 1) / 1]

R²: 0.429  
RMSE: 33744.372  
MAE: 25175.654

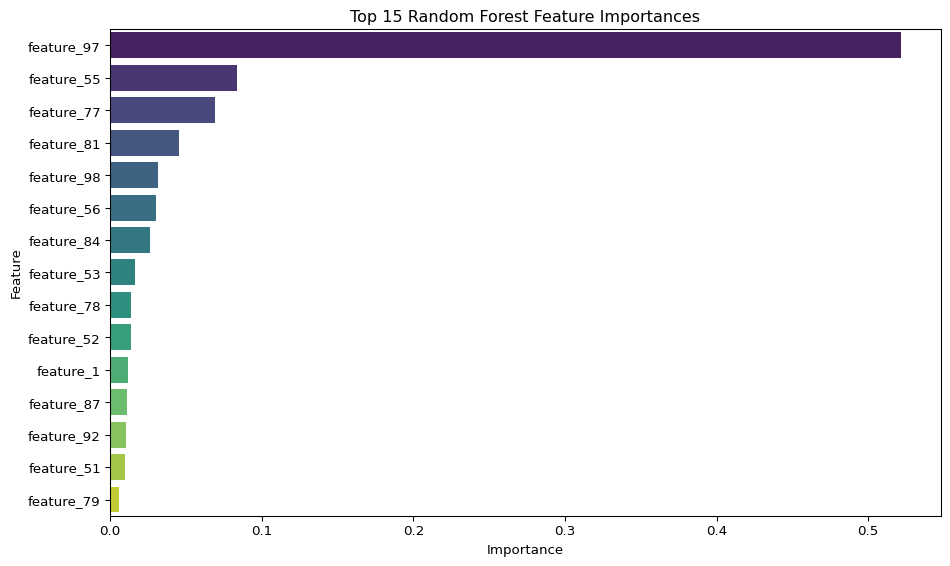
### 5.1.5 Feature 97 was by far the most important feature.

rf\_stage = rf\_model.stages[-1]   
  
importances = rf\_stage.featureImportances.toArray()  
  
feat\_imp = pd.DataFrame({  
 "Feature": [f"feature\_{i+1}" for i in range(len(importances))],  
 "Importance": importances  
}).sort\_values(by="Importance", ascending=False)  
  
feat\_imp.head(10)

|  | Feature | Importance |
| --- | --- | --- |
| 96 | feature\_97 | 0.521858 |
| 54 | feature\_55 | 0.083882 |
| 76 | feature\_77 | 0.069038 |
| 80 | feature\_81 | 0.045592 |
| 97 | feature\_98 | 0.031267 |
| 55 | feature\_56 | 0.029943 |
| 83 | feature\_84 | 0.025972 |
| 52 | feature\_53 | 0.016110 |
| 77 | feature\_78 | 0.013780 |
| 51 | feature\_52 | 0.013470 |

import matplotlib.pyplot as plt  
import seaborn as sns  
  
plt.figure(figsize=(10,6))  
sns.barplot(data=feat\_imp.head(15), y="Feature", x="Importance", palette="viridis")  
plt.title("Top 15 Random Forest Feature Importances")  
plt.tight\_layout()  
plt.show()

/tmp/ipykernel\_5663/3261347940.py:5: FutureWarning:  
  
  
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.



# 6. Model Comparisons

## 6.1 Summary

### 6.1.1 Overall the linear regression model was the model that produced the most consistent results on the test data. Generally speaking across all features the T-stat was consistently high (coeff/error), the p[ value was consistently at a level of significance (<.05), and this was representative of all variables, meaning all of the variables seemed to contribute reasonably to the outcome in terms of the coefficient magnitude, the t-stat and the p-value. The random foorest performed better in r squared (percent variation in salary explained by the features) but the rest of the supporting data was inconsinstent, especially the expnential impact of feature 97 on the random forest. The polynomial model also had inconsistencies, which make it less consistenly effective than the linear regression.

### 6.1.2 In total the linear regression accross the 99 features, the average p value was not significant, however the median value was significant at appx. .037 - the 75th percentile is not significant but the 25th is. There were a few features that produced extreme results in all of the t value (~-6/74), p value (0/.98), and coefficient (a few spikes). R squared states ~35% of variation in salary can be explained by the features in the data. On average predictions are off by ~$35,834.