Complete Project PDF

Sales Standards of Excellence

Table of Contents:

Diana Bowden

Eastern University: DTSC691

Fall Term 2023

Experiment A Source Code:

```
# Databricks notebook source
#  All of the data sources to pick from (Salesforce, LMS, Funnel, etc.)
root_directory = "/mnt/datalake/silver-adhoc/"
dbutils.fs.ls(root_directory)


# COMMAND ----------

#Helper Functions
def load_raw_data(root_dir):
  table_names = []
  for fileinfo in dbutils.fs.ls(root_dir):
    filename = fileinfo[1][:-1]
    sdf_tablename = spark.read.format("delta").option("header","true").load(root_dir+filename) # Load
each table from this DIR
    sdf_tablename.createOrReplaceTempView(filename) # Create spark view for each table
    table_names.append(filename)
  return table_names

# COMMAND ----------

# load salesforce tables
root_directory = "/mnt/datalake/silver-adhoc/salesforce/"
load_raw_data(root_directory)

# COMMAND ----------

# MAGIC %md
# MAGIC Load all dfs from prevous notebook before joining

# COMMAND ----------

oppdf = spark.sql("""
SELECT  DISTINCT OwnerId, Owner_Full_Name__c, sum(Amount) AS Total_Revenue, Avg(Amount) AS
Avg_Opp_Amount, Avg(Age__c)  AS Avg_Sell_Cycle, Avg(Number_of_Opportunity_Products__c) AS
Avg_Num_Products_Per_Opp
FROM opportunity
WHERE Order_Type__c != 'Renewal'
GROUP BY OwnerId, Owner_Full_Name__c
""")
display(oppdf)

# COMMAND ----------

#First, explore IsWon column values
```

```
closeddf = spark.sql("""
SELECT OwnerId, Owner_Full_Name__c, IsWon
FROM Opportunity

""")
display(closeddf)

# COMMAND ----------

#Aggregate IsWon values into 2 new columns (True vs. False)
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, when, col

# Creating new columns for counts of 'True' and 'False'
splitdf = closeddf.groupBy('OwnerId').agg(
    sum(when(col('IsWon') == True, 1).otherwise(0)).alias('Total_Opps_Won'),
    sum(when(col('IsWon') == False, 1).otherwise(0)).alias('Total_Opps_Lost')
)

display(splitdf)

# COMMAND ----------

#Add calculated fields of total opps count by rep and closure rate

from pyspark.sql import SparkSession
from pyspark.sql.functions import col

avg_close_ratedf = splitdf.withColumn(
    'Avg_Close_Rate',
    (col('Total_Opps_Won') / (col('Total_Opps_Won') + col('Total_Opps_Lost')))
)
display(avg_close_ratedf)

# COMMAND ----------

contractdf = spark.sql ("""
SELECT DISTINCT o.OwnerId, avg(Term) AS Avg_Term_Length
FROM opportunity o
JOIN servicecontract s
ON o.AccountId = s.AccountId
GROUP BY o.OwnerId
""")
display(contractdf)

# COMMAND ----------

quotadf = spark.sql("""
```

```
SELECT o.OwnerId, sum(f.QuotaAmount) AS Sum_Quota, sum(o.Amount) AS Sum_Amount
FROM opportunity o
LEFT JOIN forecastingquota f
ON o.OwnerId = f.QuotaOwnerId
GROUP BY o.OwnerId, o.Owner_Full_Name__c
ORDER BY o.Owner_Full_Name__c
""")
display(quotadf)

# COMMAND ----------

#add column with % to quota
percent_to_quotadf = quotadf.withColumn(
    'Percent_To_Quota',
    (col('Sum_Amount') / col('Sum_Quota' )))
display (percent_to_quotadf)

# COMMAND ----------

avg_opp_discdf = spark.sql("""
SELECT o.OwnerId, avg(q.SBQQ__AverageCustomerDiscount__c) AS Avg_Cust_Disc
FROM opportunity o
JOIN sbqq__quote__c q
ON o.OwnerId = q.OwnerId
GROUP BY o.OwnerId, o.Owner_Full_Name__c
ORDER BY o.Owner_Full_Name__c
""")
display(avg_opp_discdf)

# COMMAND ----------

datedf = spark.sql("""
SELECT OwnerId, ActivityDate
FROM task
""")
display(datedf)

# COMMAND ----------

tasksdf = spark.sql("""
SELECT DISTINCT OwnerId, count(Type), min(ActivityDate) AS Rep_Start_Date, max(ActivityDate) AS
Rep_Term_Date
FROM task t
WHERE Status = 'Completed'
GROUP BY OwnerId
""")
display(tasksdf)
```

```
# COMMAND ----------

from pyspark.sql import SparkSession
from pyspark.sql import functions as F

# Convert timestamp to date
datedf = tasksdf.withColumn("Rep_Start_Date_Date", F.to_date("Rep_Start_Date"))
datedf = datedf.withColumn("Rep_Term_Date_Date", F.to_date("Rep_Term_Date"))
# Add a calculated field ("Days_Worked")
daysdf = datedf.withColumn("Days_Worked", F.datediff("Rep_Term_Date_Date",
"Rep_Start_Date_Date"))
daysdf = daysdf.withColumn("Weeks_Worked", F.col("Days_Worked") / 7)

display(daysdf)

# COMMAND ----------

#drop unnecessary columns
daysdf = daysdf.drop("count(Type)", "Rep_Start_Date", "Rep_Term_Date", "Rep_Start_Date_Date",
"Rep_Term_Date_Date")
display(daysdf)

# COMMAND ----------

#consolidate all task types
ttypedf = spark.sql("""
SELECT OwnerId, Type
FROM task
GROUP BY OwnerId, Type
""")
display(ttypedf)

# COMMAND ----------

#Total all Type categories into new columns
split_taskdf = spark.sql("""
SELECT
   DISTINCT OwnerId,
   COUNT(Type) as Total_Activities,
   SUM(CASE WHEN Type= 'Email' THEN 1 ELSE 0 END) as Total_Emails,
   SUM(CASE WHEN Type= 'Customer Call' THEN 1 ELSE 0 END) as Total_Customer_Calls,
   SUM(CASE WHEN Type= 'Prospect Call' THEN 1 ELSE 0 END) as Total_Prospect_Calls,
   SUM(CASE WHEN Type= 'Conversation' THEN 1 ELSE 0 END) as Total_Conversations,
   SUM(CASE WHEN Type= 'Demo Request' THEN 1 ELSE 0 END) as Total_Demo_Requests,
   SUM(CASE WHEN Type= 'Set Demo' THEN 1 ELSE 0 END) as Total_Set_Demos,
   SUM(CASE WHEN Type= 'Performed Demo' THEN 1 ELSE 0 END) as Total_Performed_Demos,
   SUM(CASE WHEN Type= 'Online Meeting' THEN 1 ELSE 0 END) as Total_Online_Meetings
FROM
```

```
    task
GROUP BY
    OwnerId
""").na.fill(0)

display(split_taskdf)


# COMMAND ----------

from pyspark.sql import functions as F

# Join DataFrames
final_typedf = daysdf.join(split_taskdf, on="OwnerId")

# List of columns to calculate
columns_to_calculate = [
    "Total_Activities",
    "Total_Emails",
    "Total_Customer_Calls",
    "Total_Prospect_Calls",
    "Total_Conversations",
    "Total_Set_Demos",
    "Total_Performed_Demos",
    "Total_Demo_Requests",
    "Total_Online_Meetings"
]

# Iterate over the columns and calculate the corresponding new columns
for col_name in columns_to_calculate:
    new_col_name = f"{col_name}_Per_Week"
    final_typedf = final_typedf.withColumn(new_col_name, F.col(col_name) / F.col("Weeks_Worked"))

display(final_typedf)

# COMMAND ----------

#drop colunns not being used in final analysis
final_type_dropdf = final_typedf.drop(
    "Days_Worked",
    "Total_Activities",
    "Total_Emails",
    "Total_Customer_Calls",
    "Total_Prospect_Calls",
    "Total_Conversations",
    "Total_Demo_Requests",
    "Total_Set_Demos",
    "Total_Performed_Demos",
```

```
    "Total_Online_Meetings")

display(final_type_dropdf)

# COMMAND ----------

# MAGIC %md
# MAGIC JOIN following tables: oppdf, avg_close_ratedf, contractdf, percent_to_quotadf,
avg_opp_discdf, final_type_dropdf ON OwnerId

# COMMAND ----------

joindf = oppdf.join(avg_close_ratedf, on="OwnerId")
joindf = joindf.join(contractdf, on="OwnerId")
joindf = joindf.join(percent_to_quotadf, on="OwnerId")
joindf = joindf.join(avg_opp_discdf, on="OwnerId")
joindf = joindf.join(final_type_dropdf, on="OwnerId")

display(joindf)

# COMMAND ----------

#filter out reps / sales ops people that may not have quote using Percent_To_Quota field, the dependent
variable.
filtered_df = joindf.filter(joindf["Percent_To_Quota"].isNotNull())
display(filtered_df)

# COMMAND ----------

descriptive_stats = filtered_df.summary()
display(descriptive_stats)

# COMMAND ----------

#use this cell's visualizations to check outliers before and after removal
# filter out negative values in avg_sell_cycle and avg_cust_disc in df
remove_neg_out_sellcycle = filtered_df.where((col('Avg_Sell_Cycle') >= 0) & (col("Weeks_Worked")>0))
remove_neg_opps = remove_neg_out_sellcycle.where(col("Avg_Opp_Amount") > 0)
remove_opps = remove_neg_opps.where(col('Total_Opps_Won')> 0)
remove_opps_disc = remove_opps.where((col("Avg_Cust_Disc") > 0 ) & (col("Avg_Cust_Disc") <99))
remove_max_opp = remove_opps_disc.where(col("Avg_Opp_Amount") < 160000)

display(remove_max_opp)


# COMMAND ----------

#display descriptive stats again for comparison
```

```
descriptive_stats2 = remove_max_opp.summary()
display(descriptive_stats2)

# COMMAND ----------

descriptive_stats = filtered_df.summary()
display(descriptive_stats)

# COMMAND ----------



# COMMAND ----------

#After separating data, run correlations
remove_max_opp.stat.corr("Percent_To_Quota", "Avg_Opp_Amount")


# COMMAND ----------

#After separating data, run correlations
remove_max_opp.stat.corr("Percent_To_Quota", "Avg_Cust_Disc")

# COMMAND ----------

final_df = remove_max_opp
display(final_df)

# COMMAND ----------

from pyspark.sql.functions import col, expr
from pyspark.sql.window import Window

column_name = 'Percent_To_Quota'

# Calculate the 80th percentile value
percentile_value = final_df.approxQuantile(column_name, [0.8], 0.001)[0]

# Create a new column with binary values
binary_df = final_df.withColumn('binary_column', expr(f'CASE WHEN {column_name} >=
{percentile_value} THEN 1 ELSE 0 END'))

# Display the resulting DataFrame
display(binary_df)

# COMMAND ----------

#Run Logistic Regression with 80/20 split of training set.
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.sql.functions import col

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=['Avg_Opp_Amount'], outputCol='features')

# Create a Logistic Regression model
lr = LogisticRegression(featuresCol='features', labelCol='binary_column', maxIter=10, regParam=0.01)

# Create a pipeline with the VectorAssembler and Logistic Regression stages
pipeline = Pipeline(stages=[assembler, lr])

# Split the data into training and testing sets
(trainingData, testData) = binary_df.randomSplit([0.8, 0.2], seed=42)

# Fit the model on the training data
model = pipeline.fit(trainingData)

# Make predictions on the test data
predictions = model.transform(testData)

# Display the predictions
predictions.select('binary_column', 'prediction', 'probability').show()

# COMMAND ----------

#run ROC validation to determine the logisic regression model's performance based on 80/20 training
set split.
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.classification import LogisticRegression

# Create a BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol='binary_column', rawPredictionCol='prediction',
metricName='areaUnderROC')

# Calculate the AUC-ROC
auc_roc = evaluator.evaluate(predictions)
print(f"AUC-ROC: {auc_roc}")

# COMMAND ----------

#Try K-Fold with logistic regression model to see if logistic regression model performance improves

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
```

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import col

# binary_df is the dataframe to run
independent_variable = 'Avg_Opp_Amount'
label_column = 'binary_column'

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=[independent_variable], outputCol='features')

# Create a Logistic Regression model
lr = LogisticRegression(featuresCol='features', labelCol=label_column)

# Create a pipeline with the VectorAssembler and Logistic Regression stages
pipeline = Pipeline(stages=[assembler, lr])

# Define the parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
   .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
   .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
   .build()

# Create a k-fold cross-validator; ;set number of folds to 5
crossval = CrossValidator(estimator = pipeline,
estimatorParamMaps = paramGrid,
evaluator = BinaryClassificationEvaluator(labelCol=label_column), numFolds =5 )

# Fit the model using cross-validation
cvModel = crossval.fit(binary_df)  # Use the entire dataset

# The model has been trained and evaluated on each fold
# Access the best model and view its performance
bestModel = cvModel.bestModel
predictions = bestModel.transform(binary_df)
areaUnderROC = evaluator.evaluate(predictions)

# Display the area under the ROC curve
print(f"Area Under ROC: {areaUnderROC}")

# COMMAND ----------

#run multiple regression model using K-fold validation to determine if this is a better model.

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col

# final_df is the DataFrame
target_column = 'Percent_To_Quota'
feature_columns = ['Avg_Opp_Amount', 'Avg_Sell_Cycle', 'Avg_Cust_Disc']

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=feature_columns, outputCol='features')

# Create a Linear Regression model
lr = LinearRegression(featuresCol='features', labelCol=target_column)

# Create a pipeline with the VectorAssembler and Linear Regression stages
pipeline = Pipeline(stages=[assembler, lr])

# Define the parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

# Create a k-fold cross-validator; set number of folds to 5
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(labelCol=target_column),  # Instantiate with labelCol
                          numFolds=5)

# Split the data into training and testing sets
(trainingData, testData) = final_df.randomSplit([0.8, 0.2], seed=42)

# Fit the model using cross-validation
cvModel = crossval.fit(trainingData)

# Make predictions on the test data
predictions = cvModel.transform(testData)

# Evaluate the model on the test data
evaluator = RegressionEvaluator(labelCol=target_column)  # Instantiate with labelCol
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# Display evaluation metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")


# COMMAND ----------
```

```
# Display evaluation metrics for the MLR  model
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# COMMAND ----------

#Explore other model options - try Random Forest

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator

target_column = 'Percent_To_Quota'
feature_columns = ['Avg_Opp_Amount', 'Avg_Sell_Cycle', 'Avg_Cust_Disc']

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=feature_columns, outputCol='features')

# Create a Random Forest Regressor model
rf = RandomForestRegressor(featuresCol='features', labelCol=target_column)

# Create a pipeline with the VectorAssembler and Random Forest Regressor stages
pipeline = Pipeline(stages=[assembler, rf])

# Split the data into training and testing sets
# Adjust the split ratio as needed
(trainingData, testData) = final_df.randomSplit([0.8, 0.2], seed=42)

# Fit the model on the training data
model = pipeline.fit(trainingData)

# Make predictions on the test data
predictions = model.transform(testData)

# Evaluate the model using a regression evaluator
evaluator = RegressionEvaluator(labelCol=target_column, predictionCol='prediction',
metricName='rmse')
rmse = evaluator.evaluate(predictions)

# Display the Root Mean Squared Error (RMSE)
print(f"Root Mean Squared Error (RMSE): {rmse}")

# View the feature importances
feature_importances = model.stages[-1].featureImportances
print("Feature Importances:")
for i, feature in enumerate(feature_columns):
```

```
print(f"{feature}: {feature_importances[i]}")
```

Experiment B Source Code:

```
# Databricks notebook source
# MAGIC %md
# MAGIC Load all Salesforce tables, helper functions

# COMMAND ----------

#  All of the data sources to pick from (Salesforce, LMS, Funnel, etc.)
root_directory = "/mnt/datalake/silver-adhoc/"
dbutils.fs.ls(root_directory)

#Helper Functions
def load_raw_data(root_dir):
  table_names = []
  for fileinfo in dbutils.fs.ls(root_dir):
    filename = fileinfo[1][:-1]
    sdf_tablename = spark.read.format("delta").option("header","true").load(root_dir+filename) # Load
each table from this DIR
    sdf_tablename.createOrReplaceTempView(filename) # Create spark view for each table
    table_names.append(filename)
  return table_names

# load salesforce tables
root_directory = "/mnt/datalake/silver-adhoc/salesforce/"
load_raw_data(root_directory)


# COMMAND ----------

# MAGIC %md
# MAGIC Examine tables, aggregate, filter and join

# COMMAND ----------

# MAGIC %sql
# MAGIC
# MAGIC SELECT *
# MAGIC FROM opportunity
# MAGIC LiMIT 100;

# COMMAND ----------

#Obtain total number of accounts by OwnerId
accountdf = spark.sql("""
SELECT DISTINCT OwnerId, count(Name) AS Total_Num_Accounts
```

```
FROM Account
GROUP BY OwnerId
""")
display(accountdf)


# COMMAND ----------

#Obtain calculated fields needed from opportunity table
oppdf = spark.sql("""
SELECT
   DISTINCT OwnerId, Owner_Full_Name__c,
   Sales_Division__c,
   SUM(Amount) AS Total_Revenue,
   AVG(Amount) AS Avg_Opp_Amount,
   AVG(Age__c) AS Avg_Sell_Cycle,
   AVG(Number_of_Opportunity_Products__c) AS Avg_Num_Products_Per_Opp
FROM
   opportunity
WHERE
   Sales_Division__c != "Digital Recruiting"
GROUP BY
   OwnerId,
   Owner_Full_Name__c,
   Sales_Division__c
""")

display(oppdf)

# COMMAND ----------

#join account table with opportunity table
acct_opp_df = accountdf.join(oppdf, on="OwnerId", how="left")
display(acct_opp_df)

# COMMAND ----------

#First, explore IsWon column values on IsWon
closeddf = spark.sql("""
SELECT OwnerId, Owner_Full_Name__c, IsWon
FROM Opportunity

""")
display(closeddf)

# COMMAND ----------

#Aggregate IsWon values into 2 new columns (True vs. False)
```

```
from pyspark.sql.functions import sum, when, col

# Creating new columns for counts of 'True' and 'False'
splitdf = closeddf.groupBy('OwnerId').agg(
    sum(when(col('IsWon') == True, 1).otherwise(0)).alias('Total_Opps_Won'),
    sum(when(col('IsWon') == False, 1).otherwise(0)).alias('Total_Opps_Lost')
)

#Add calculated fields of total opps count by rep and closure rate

avg_close_ratedf = splitdf.withColumn(
    'Avg_Close_Rate',
    (col('Total_Opps_Won') / (col('Total_Opps_Won') + col('Total_Opps_Lost')))
)
display(avg_close_ratedf)


# COMMAND ----------

#Obtain average contract term length
contractdf = spark.sql ("""
SELECT DISTINCT o.OwnerId, avg(Term) AS Avg_Term_Length
FROM opportunity o
JOIN servicecontract s
ON o.AccountId = s.AccountId
GROUP BY o.OwnerId
""")
display(contractdf)


# COMMAND ----------

#Calculate sum of quota and sum of all opportunities, by OwnerId
quotadf = spark.sql("""
SELECT o.OwnerId, o.Owner_Full_Name__c, sum(f.QuotaAmount) AS Sum_Quota, sum(o.Amount) AS
Sum_Amount
FROM opportunity o
LEFT JOIN forecastingquota f
ON o.OwnerId = f.QuotaOwnerId
GROUP BY o.OwnerId, o.Owner_Full_Name__c
ORDER BY o.Owner_Full_Name__c
""")

#add column with % to quota
percent_to_quotadf = quotadf.withColumn(
    'Percent_To_Quota',
    (col('Sum_Amount') / col('Sum_Quota' )))
display (percent_to_quotadf)
```

COMMAND ----------

COMMAND ----------

```
#Calculate average discount by OwnerId
avg_opp_discdf = spark.sql("""
SELECT o.OwnerId, avg(q.SBQQ__AverageCustomerDiscount__c) AS Avg_Cust_Disc
FROM opportunity o
JOIN sbqq__quote__c q
ON o.OwnerId = q.OwnerId
GROUP BY o.OwnerId, o.Owner_Full_Name__c
ORDER BY o.Owner_Full_Name__c
""")
display(avg_opp_discdf)
```

COMMAND ----------

```
#Obtain "weeks worked" for each rep by first taking min/max of when activity dates started and ended.

datedf = spark.sql("""
SELECT OwnerId, ActivityDate
FROM task
""")

tasksdf = spark.sql("""
SELECT DISTINCT OwnerId, count(Type), min(ActivityDate) AS Rep_Start_Date, max(ActivityDate) AS
Rep_Term_Date
FROM task t
WHERE Status = 'Completed'
GROUP BY OwnerId
""")

#Next, convert timestamp to date so that a calculated field can be created to subtract min from max
activity date.
from pyspark.sql import functions as F

#Divid days_worked by 7 to get weeks worked, so that all rep activities are normalized by week,
regardless of tenure.
datedf = tasksdf.withColumn("Rep_Start_Date_Date", F.to_date("Rep_Start_Date"))
datedf = datedf.withColumn("Rep_Term_Date_Date", F.to_date("Rep_Term_Date"))
# Add a calculated field ("Days_Worked")
```

```
daysdf = datedf.withColumn("Days_Worked", F.datediff("Rep_Term_Date_Date",
"Rep_Start_Date_Date"))
daysdf = daysdf.withColumn("Weeks_Worked", F.col("Days_Worked") / 7)

display(daysdf)



# COMMAND ----------

#drop unnecessary columns
daysdf = daysdf.drop("count(Type)", "Rep_Start_Date", "Rep_Term_Date", "Rep_Start_Date_Date",
"Rep_Term_Date_Date")


#consolidate all task types
ttypedf = spark.sql("""
SELECT OwnerId, Type
FROM task
GROUP BY OwnerId, Type
""")


#Total all Type categories into new columns
split_taskdf = spark.sql("""
SELECT
   DISTINCT OwnerId,
   COUNT(Type) as Total_Activities,
   SUM(CASE WHEN Type= 'Email' THEN 1 ELSE 0 END) as Total_Emails,
   SUM(CASE WHEN Type= 'Customer Call' THEN 1 ELSE 0 END) as Total_Customer_Calls,
   SUM(CASE WHEN Type= 'Prospect Call' THEN 1 ELSE 0 END) as Total_Prospect_Calls,
   SUM(CASE WHEN Type= 'Conversation' THEN 1 ELSE 0 END) as Total_Conversations,
   SUM(CASE WHEN Type= 'Demo Request' THEN 1 ELSE 0 END) as Total_Demo_Requests,
   SUM(CASE WHEN Type= 'Set Demo' THEN 1 ELSE 0 END) as Total_Set_Demos,
   SUM(CASE WHEN Type= 'Performed Demo' THEN 1 ELSE 0 END) as Total_Performed_Demos,
   SUM(CASE WHEN Type= 'Online Meeting' THEN 1 ELSE 0 END) as Total_Online_Meetings
FROM
   task
GROUP BY
   OwnerId
""").na.fill(0)

display(split_taskdf)



# COMMAND ----------
```

```
#join tasks with days worked
from pyspark.sql import functions as F

# Join DataFrames
final_typedf = daysdf.join(split_taskdf, on="OwnerId")

# List of columns to calculate
columns_to_calculate = [
    "Total_Activities",
    "Total_Emails",
    "Total_Customer_Calls",
    "Total_Prospect_Calls",
    "Total_Conversations",
    "Total_Set_Demos",
    "Total_Performed_Demos",
    "Total_Demo_Requests",
    "Total_Online_Meetings"
]

# Iterate over the columns and calculate the corresponding new columns
for col_name in columns_to_calculate:
    new_col_name = f"{col_name}_Per_Week"
    final_typedf = final_typedf.withColumn(new_col_name, F.col(col_name) / F.col("Weeks_Worked"))

#drop colunns not being used in final analysis
final_type_dropdf = final_typedf.drop(
    "Days_Worked",
    "Total_Activities",
    "Total_Emails",
    "Total_Customer_Calls",
    "Total_Prospect_Calls",
    "Total_Conversations",
    "Total_Demo_Requests",
    "Total_Set_Demos",
    "Total_Performed_Demos",
    "Total_Online_Meetings")

#JOIN following tables: oppdf, avg_close_ratedf, contractdf, percent_to_quotadf, avg_opp_discdf,
final_type_dropdf ON OwnerId
joindf = acct_opp_df.join(avg_close_ratedf, on="OwnerId")
joindf = joindf.join(contractdf, on="OwnerId")
joindf = joindf.join(percent_to_quotadf, on="OwnerId")
joindf = joindf.join(avg_opp_discdf, on="OwnerId")
joindf = joindf.join(final_type_dropdf, on="OwnerId")

display(joindf)
```

```
# COMMAND ----------

#filter out reps / sales ops people that may not have quote using Percent_To_Quota field, the dependent variable.
filtered_df = joindf.filter(joindf["Percent_To_Quota"].isNotNull())

#check stats:   descriptive_stats = filtered_df.summary()
#display(descriptive_stats)

#use this cell's visualizations to check outliers before and after removal
# filter out negative values in avg_sell_cycle and avg_cust_disc in df
remove_neg_out_sellcycle = filtered_df.where((col('Avg_Sell_Cycle') >= 0) & (col("Weeks_Worked")>0))
remove_neg_opps = remove_neg_out_sellcycle.where(col("Avg_Opp_Amount") > 0)
remove_opps = remove_neg_opps.where(col('Total_Opps_Won')> 0)
outliers_removed_df = remove_opps.where((col("Avg_Cust_Disc") > 0 ) & (col("Avg_Cust_Disc") <99))
outliers_removed_df= outliers_removed_df.where((col("Sales_Division__c") != "Field") &
(col("Sales_Division__c") != "Field Sales") & (col("Sales_Division__C") != "Migrations"))
outliers_removed_df = outliers_removed_df.where((col("Sales_Division__c") != "MM"))

display(outliers_removed_df)



# COMMAND ----------

#Run Visualizations
display(outliers_removed_df)

# COMMAND ----------

# MAGIC %md
# MAGIC SMB Data

# COMMAND ----------

#Run visualizations on Avg_Opp_Amount by Sales_Division__c against target: Percent_To_Quota: SMB

smb_df = outliers_removed_df.filter(col("Sales_Division__c") == "SMB")
display(smb_df)


# COMMAND ----------

#Review descriptive statistics for outliers and distribution of data
smb_stats = smb_df.summary()
display(smb_stats)

# COMMAND ----------
```

```
# Run correlations from visualizations that had the best potential: SMB
from pyspark.sql.functions import corr
smb_opp_corr = smb_df.select(corr("Percent_To_Quota", "Avg_Opp_Amount")).collect()[0][0]
smb_disc_corr = smb_df.select(corr("Percent_To_Quota", "Avg_Cust_Disc")).collect()[0][0]
smb_activity_corr = smb_df.select(corr("Percent_To_Quota",
"Total_Activities_Per_Week")).collect()[0][0]

# Display the correlation
print(f"Correlation between Percent To Quota and Average Opp Amount: {smb_opp_corr}")
print(f"Correlation between Percent To quota and Average Cust Discount: {smb_disc_corr}")
print(f"Correlation between Percent To quota and Total Avg Weekly Activities: {smb_activity_corr}")

# COMMAND ----------

# MAGIC %md
# MAGIC SMB Linear Regression

# COMMAND ----------

#Run Linear Regression Model on SMB with Avg_Cust_Discount on Percent To Quota

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator

# Assemble features, create a pipeline, and set up cross-validation
target_column = 'Percent_To_Quota'
feature_column_smb = ['Avg_Cust_Disc']
assembler = VectorAssembler(inputCols=feature_column_smb, outputCol='smb_feature')

lr = LinearRegression(featuresCol='smb_feature', labelCol=target_column)
pipeline = Pipeline(stages=[assembler, lr])

paramGrid = ParamGridBuilder() \
   .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
   .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
   .build()

crossval = CrossValidator(estimator=pipeline,
                  estimatorParamMaps=paramGrid,
                  evaluator=RegressionEvaluator(labelCol=target_column),
                  numFolds=5)

(trainingData, testData) = smb_df.randomSplit([0.8, 0.2], seed=42)
```

```
# Fit the model using cross-validation
cvModel = crossval.fit(trainingData)

# Access coefficient estimates
coefficients = cvModel.bestModel.stages[-1].coefficients

# Access the intercept
intercept = cvModel.bestModel.stages[-1].intercept

# Print coefficients and intercept
print(f"Intercept: {intercept}")
for i, coef in enumerate(coefficients):
    print(f"Coefficient {i}: {coef}")

# Make predictions on the test data
predictions = cvModel.transform(testData)

# Evaluate the model on the test data
evaluator = RegressionEvaluator(labelCol=target_column)
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# Display evaluation metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Access the t-values and p-values from the summary
t_values = cvModel.bestModel.stages[-1].summary.tValues
p_values = cvModel.bestModel.stages[-1].summary.pValues

# Print t-values and p-values
for i in range(len(t_values)):
    print(f"Coefficient {i}: T-statistic = {t_values[i]}, P-value = {p_values[i]}")


# COMMAND ----------

#Create user interface that forecasts percent to quota
coefficient = 0.0032698978873837653
intercept = 0.2121573067286658

def predict_y(x):
    # Predict Y using the linear regression equation
    y = intercept + coefficient * x
    return y

# Example usage
feature_value = 12000
```

```
predicted_y = predict_y(feature_value)

print(f"For feature value {feature_value}, predicted Y is: {predicted_y}")

# COMMAND ----------

# MAGIC %md
# MAGIC SMB Multiple Regression

# COMMAND ----------

#Run MLR for SMB with avg cust disc and avg opp amount on percent to quota to see if this is better
model

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
import scipy.stats as stats

# smb_df is DataFrame

target_column = 'Percent_To_Quota'
features_columns_smb = ['Avg_Opp_Amount', 'Avg_Cust_Disc']

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=features_columns_smb, outputCol='smb_features')

# Create a Linear Regression model
lr = LinearRegression(featuresCol='smb_features', labelCol=target_column)

# Create a pipeline with the VectorAssembler and Linear Regression stages
pipeline = Pipeline(stages=[assembler, lr])

# Define the parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
   .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
   .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
   .build()

# Create a k-fold cross-validator; set number of folds to 5
crossval = CrossValidator(estimator=pipeline,
                    estimatorParamMaps=paramGrid,
                    evaluator=RegressionEvaluator(labelCol=target_column),  # Instantiate with labelCol
                    numFolds=5)

# Split data into training and testing sets
```

```
(trainingData, testData) = smb_df.randomSplit([0.8, 0.2], seed=42)

# Fit model using cross-validation
cvModel = crossval.fit(trainingData)

# Access coefficient estimates and standard errors
coefficients = cvModel.bestModel.stages[-1].coefficients
t_values = cvModel.bestModel.stages[-1].summary.tValues

# Print coefficients
print("Intercept:", cvModel.bestModel.stages[-1].intercept)
print("Coefficients:", cvModel.bestModel.stages[-1].coefficients)

# Calculate p-values
degrees_of_freedom = cvModel.bestModel.stages[-1].summary.degreesOfFreedom
p_values = [2 * (1 - stats.t.cdf(abs(t), df=degrees_of_freedom)) for t in t_values]

# Print coefficients, t-values, and p-values
for i, (coef, t_val, p_val) in enumerate(zip(coefficients, t_values, p_values)):
    print(f"Coefficient {i}: Estimate = {coef}, T-value = {t_val}, P-value = {p_val}")

# Make predictions on test data
predictions = cvModel.transform(testData)

# Evaluate model on test data
evaluator = RegressionEvaluator(labelCol=target_column)  # Instantiate with labelCol
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# Display evaluation metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")



# COMMAND ----------

import matplotlib.pyplot as plt
import numpy as np
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline

# smb_df is DataFrame

# Extract coefficients and intercept
intercept = 0.16387927163782137
coefficients = [4.152605107643032e-06, 0.0032705427993116173]
```

```
# Assemble features
label_column = ['Percent_To_Quota']
feature_columns = ['Avg_Opp_Amount', 'Avg_Cust_Disc']
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
smb2_df = assembler.transform(smb_df)

# Create a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="Percent_To_Quota", predictionCol="prediction")
model = lr.fit(smb2_df)

# Generate predictions
predictions = model.transform(smb2_df)

# Extract feature values and predictions
feature_values = np.array([row.features.toArray() for row in predictions.select("features").collect()])
predictions_values = np.array(predictions.select("prediction").rdd.map(lambda row: row[0]).collect())

# Create a scatter plot
plt.figure(figsize=(10, 6))

# Scatter plot for feature 1
plt.subplot(1, 2, 1)
plt.scatter(feature_values[:, 0], smb2_df.select("Percent_To_Quota").rdd.map(lambda row:
row[0]).collect(), label="Actual")
plt.scatter(feature_values[:, 0], predictions_values, label="Predicted")
plt.xlabel("Avg_Opp_Amount")
plt.ylabel("Percent_To_Quota")
plt.title("Scatter Plot for Avg_Opp_Amount")

# Add best fit line for feature 1
x_range_1 = np.linspace(np.min(feature_values[:, 0]), np.max(feature_values[:, 0]), 100)
y_range_1 = coefficients[0] * x_range_1 + intercept
plt.plot(x_range_1, y_range_1, color='red', label='Best Fit Line')
plt.legend()

# Scatter plot for feature 2
plt.subplot(1, 2, 2)
plt.scatter(feature_values[:, 1], smb2_df.select("Percent_To_Quota").rdd.map(lambda row:
row[0]).collect(), label="Actual")
plt.scatter(feature_values[:, 1], predictions_values, label="Predicted")
plt.xlabel("Avg_Cust_Disc")
plt.ylabel("Percent_To_Quota")
plt.title("Scatter Plot for Avg_Cust_Disc")

# Add best fit line for feature 2
x_range_2 = np.linspace(np.min(feature_values[:, 1]), np.max(feature_values[:, 1]), 100)
y_range_2 = coefficients[1] * x_range_2 + intercept
```

```
plt.plot(x_range_2, y_range_2, color='red', label='Best Fit Line')
plt.legend()

plt.tight_layout()
plt.show()

# COMMAND ----------

# MAGIC %md
# MAGIC Mid-Market Data

# COMMAND ----------

#Run visualizations on Avg_Opp_Amount by Sales_Division__c against target: Percent_To_Quota: Midmarket
midmarket_df = outliers_removed_df.filter(col("Sales_Division__c") == "Mid-Market")

#filter outlier
midmarket_df = midmarket_df.where((col("OWner_Full_Name__c") != "LJ Yarborough"))

display(midmarket_df)

# COMMAND ----------

#Explore descriptive statistics for outliers / data variability, naans, anomalies
mm_stats = midmarket_df.summary()
display(mm_stats)

# COMMAND ----------

# Run correlations from visualizations that had the best potential: Mid-Market
from pyspark.sql.functions import corr
mm_opp_corr = midmarket_df.select(corr("Percent_To_Quota", "Avg_Opp_Amount")).collect()[0][0]
mm_disc_corr = midmarket_df.select(corr("Percent_To_Quota", "Avg_Cust_Disc")).collect()[0][0]
mm_convo_corr = midmarket_df.select(corr("Percent_To_Quota",
"Total_Conversations_Per_Week")).collect()[0][0]
mm_accounts_corr = midmarket_df.select(corr("Percent_To_Quota",
"Total_Num_Accounts")).collect()[0][0]
mm_online_corr = midmarket_df.select(corr("Percent_To_Quota",
"Total_Online_Meetings_Per_Week")).collect()[0][0]

# Display the correlation
print(f"Correlation between Percent To Quota and Average Opp Amount: {mm_opp_corr}")
print(f"Correlation between Percent To quota and Average Cust Discount: {mm_disc_corr}")
print(f"Correlation between Percent To quota and Total Avg Weekly Conversations: {mm_convo_corr}")
print(f"Correlation between Percent To quota and Total Number of Accounts: {mm_accounts_corr}")
print(f"Correlation between Percent To quota and Total Avg Online Meetings Per Week:
{mm_online_corr}")
```

```
# COMMAND ----------

# MAGIC %md
# MAGIC Mid-Market Linear Regression

# COMMAND ----------

#Run Linear Regression Model on MM data using feature Avg Cust Disc on Percent to Quota

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator

# Assemble features, create a pipeline, and set up cross-validation
target_column = 'Percent_To_Quota'
feature_column_mm = ['Avg_Cust_Disc']
assembler = VectorAssembler(inputCols=feature_column_mm, outputCol='mm_feature')

lr = LinearRegression(featuresCol='mm_feature', labelCol=target_column)
pipeline = Pipeline(stages=[assembler, lr])

paramGrid = ParamGridBuilder() \
   .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
   .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
   .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(labelCol=target_column),
                          numFolds=5)

(trainingData, testData) = midmarket_df.randomSplit([0.8, 0.2], seed=42)

# Fit the model using cross-validation
cvModel = crossval.fit(trainingData)

# Access coefficient estimates
coefficients = cvModel.bestModel.stages[-1].coefficients

# Access the intercept
intercept = cvModel.bestModel.stages[-1].intercept

# Print coefficients and intercept
print(f"Intercept: {intercept}")
for i, coef in enumerate(coefficients):
```

```
    print(f"Coefficient {i}: {coef}")

# Make predictions on the test data
predictions = cvModel.transform(testData)

# Evaluate the model on the test data
evaluator = RegressionEvaluator(labelCol=target_column)
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# Display evaluation metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Access the t-values and p-values from the summary
t_values = cvModel.bestModel.stages[-1].summary.tValues
p_values = cvModel.bestModel.stages[-1].summary.pValues

# Print t-values and p-values
for i in range(len(t_values)):
    print(f"Coefficient {i}: T-statistic = {t_values[i]}, P-value = {p_values[i]}")




# COMMAND ----------

# MAGIC %md
# MAGIC Mid-Market Multiple Regression

# COMMAND ----------

midmarket_df = midmarket_df.filter("Total_Online_Meetings_Per_Week IS NOT NULL AND
Avg_Sell_Cycle IS NOT NULL").cache()

# COMMAND ----------


#Run MLR for MM with avg cust disc and total online meetings per week on percent to quota to see if
this is better model

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
import scipy.stats as stats
import numpy as np
```

```
from pyspark.sql.functions import col

# midmarket_df is dataframe

target_column = 'Percent_To_Quota'
features_columns_mm = ['Avg_Cust_Disc', 'Total_Online_Meetings_Per_Week']

# Create a VectorAssembler to assemble the feature vector
assembler = VectorAssembler(inputCols=features_columns_mm, outputCol='mm_features')

# Create a Linear Regression model
lr = LinearRegression(featuresCol='mm_features', labelCol=target_column)

# Create a pipeline with the VectorAssembler and Linear Regression stages
pipeline = Pipeline(stages=[assembler, lr])

# Define the parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
   .addGrid(lr.regParam, [0.1, 0.01, 0.001]) \
   .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
   .build()

# Create a k-fold cross-validator; set number of folds to 5
crossval = CrossValidator(estimator=pipeline,
                    estimatorParamMaps=paramGrid,
                    evaluator=RegressionEvaluator(labelCol=target_column),
                    numFolds=5)

# Split data into training and testing sets
(trainingData, testData) = midmarket_df.randomSplit([0.8, 0.2], seed=42)

# Fit model using cross-validation
cvModel = crossval.fit(trainingData)

# Access coefficient estimates
coefficients = cvModel.bestModel.stages[-1].coefficients

# Extract feature values from the test data
feature_values = np.array([testData.select(col(col_name)).collect() for col_name in
features_columns_mm]).squeeze().T

# Calculate the t-values for each coefficient, including intercept
t_values = [coef / (float(std_dev_resid) / np.sqrt(np.sum([float(f) ** 2 for f in feature_values[:, i]]))) for i,
coef in enumerate(coefficients)]

# Calculate p-values from t-values
p_values = [2 * (1 - stats.t.cdf(abs(t), df=n - len(features_columns_mm))) for t in t_values]
```

```
# Access the intercept
intercept = cvModel.bestModel.stages[-1].intercept

# Print intercept
print(f"Intercept: {intercept}")

# Print coefficients, t-values, p-values, and standard errors
for i, (coef, t_val, p_val, se) in enumerate(zip([intercept] + list(coefficients), t_values, p_values,
standard_errors)):
    print(f"Coefficient {i}: Estimate = {coef}, T-value = {t_val}, P-value = {p_val}, Standard Error = {se}")

# Make predictions on test data
predictions = cvModel.transform(testData)

# Evaluate model on test data
evaluator = RegressionEvaluator(labelCol=target_column)  # Instantiate with labelCol
rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})

# Display evaluation metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# COMMAND ----------

# MAGIC %md
# MAGIC Additional Information on Account Management and Enterprise

# COMMAND ----------

#THIS IS NOT PART OF THE PROJECT BUT IS INTENDED TO PROVIDE MORE INFORMATION TO SALES MGT
#Run visualizations on Avg_Opp_Amount by Sales_Division__c against target: Percent_To_Quota:
Account Management
am_df = outliers_removed_df.filter(col("Sales_Division__c") == "Account Management")
display(am_df)


# COMMAND ----------

#NOT PART OF PROJECT - Run visualizations on Avg_Opp_Amount by Sales_Division__c against target:
Percent_To_Quota: Enterprise

enterpriseall_df = outliers_removed_df.filter(
    (col("Sales_Division__c") == "Enterprise HHS") |
    (col("Sales_Division__c") == "Enterprise Acute") |
    (col("Sales_Division__c") == "Enterprise Post-Acute")
)
display(enterpriseall_df)
```

COMMAND ----------

from pyspark.sql.functions import col, avg

Calculate the average of Avg_Opp_Amount for each Sales_Division__c
avg_opp_by_division =
outliers_removed_df.groupBy("Sales_Division__c").agg(avg("Avg_Opp_Amount").alias("Avg_Avg_Opp_Amount"))
sorted = avg_opp_by_division.orderBy(col("Avg_Avg_Opp_Amount").desc())
display(sorted)

COMMAND ----------

#display descriptive stats again for comparison
descriptive_stats2 = remove_max_opp.summary()
display(descriptive_stats2)
descriptive_stats = filtered_df.summary()
#display(descriptive_stats)

#After separating data, run correlations
remove_max_opp.stat.corr("Percent_To_Quota", "Avg_Opp_Amount")

#After separating data, run correlations
remove_max_opp.stat.corr("Percent_To_Quota", "Avg_Cust_Disc")
final_df = remove_max_opp
display(final_df)

COMMAND ----------

#Final deliverable is to identify benchmarks for sales management which includes averages and 75th percentiles for highes performers. Use descriptive statistics on dataframe to capture averages, min and max as thresholds for each feautre, by sales division.

smb_stats = smb_df.summary()
display(smb_stats)

mm_stats = midmarket_df.summary()
display(mm_stats)

DTSC691: APPLIED DATA SCIENCE

CUSTOM PROJECT RESULTS

PROJECT: SALES STANDARDS OF EXCELLENCE

Diana Bowden, Fall Term 2023

AGENDA

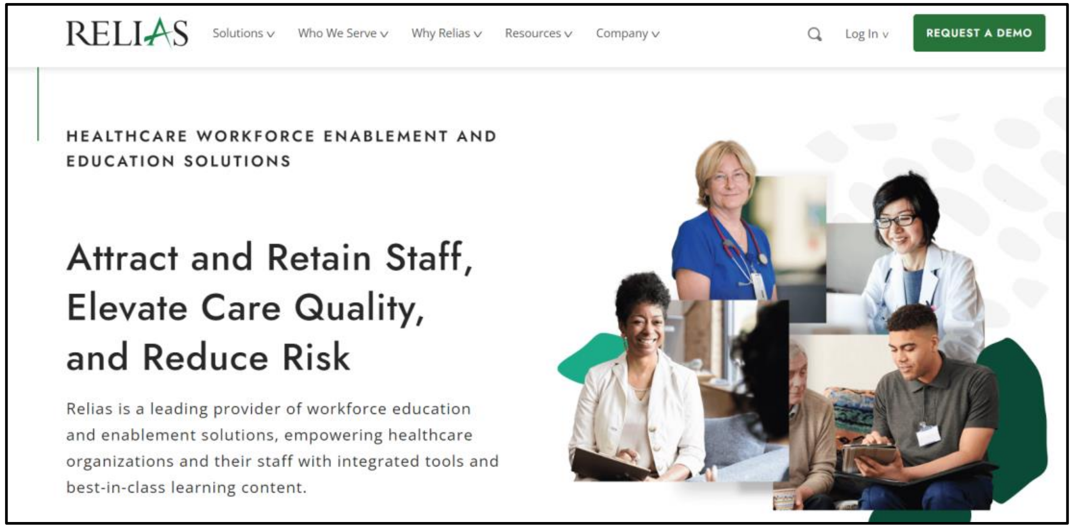
BACKGROUND INFORMATION:
- COMPANY PROBLEM
- PROJECT GOAL

SOLUTION:
- SOFTWARE
- DATA EXPLORATION & CLEANING
- EXPERIMENTS
- RESULTS

PROJECT CODE:
- WALKTHROUGH ON DATABRICKS

CHALLENGES:
- INTEGRATION STRATEGY
- COMPLEXITY
- WHAT I HAVE LEARNED
- FUTURE RECOMMENDATIONS

BACKGROUND INFORMATION



Project Goal:

Model the most successful sales representatives for use as best-selling practices to increase annual revenue results.

What are the top predictors for success for sales representatives at Relias?

SOLUTION

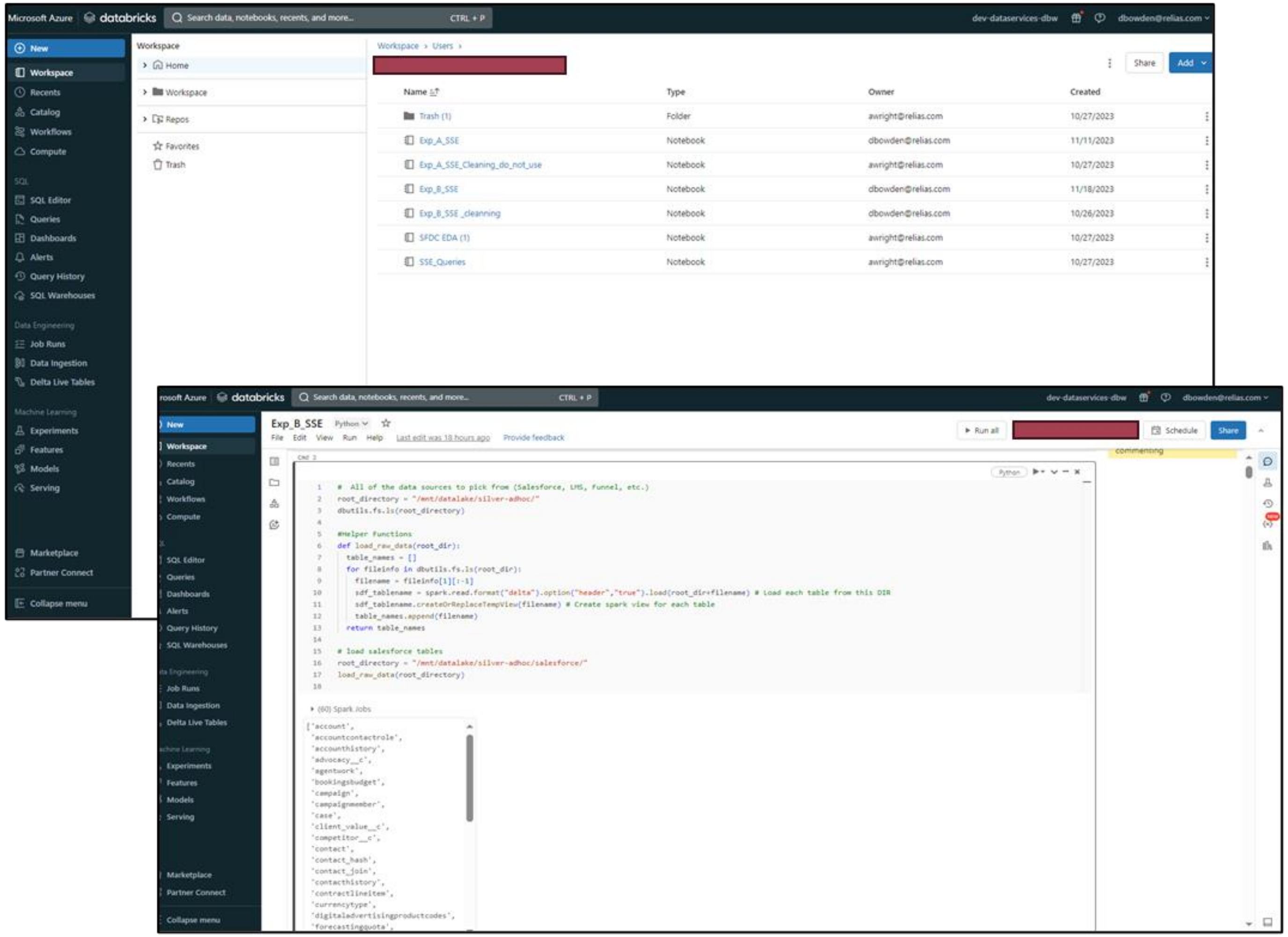
Leverage the sales tracking data from CRM (Salesforce)

Explore the attributes exhibited by the top sales representatives

Model the data using regression analysis to predict quota attainment from these attributes

Provide additional activity metrics as benchmarks for sales success

SOLUTION



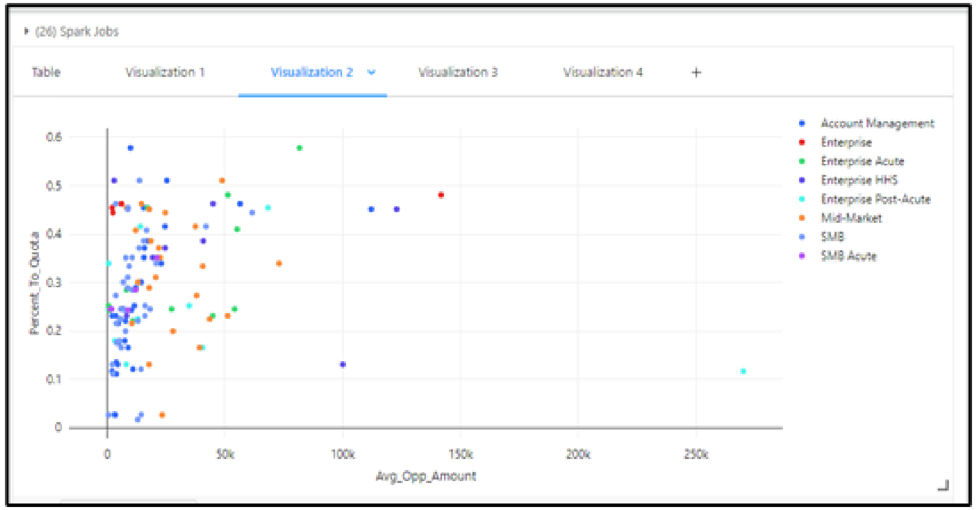
Platform & Software

- Apache Spark - core of Azure Databricks platform
- Spark libraries/packages
- Python and SQL compatible programs run on Spark
- Matplotlib
- Databricks integrated visualization tool
- Excel/Word/PowerPoint
- WebEx (recording)

SOLUTION

```
Edit  View  Run  Help  Last edit was 24 days ago  Provide feedback

final_query = spark.sql("""
SELECT
  a.OwnerId, a.Name AS Account_Name, a.Type AS Account_Type, a.BillingState AS Billing_State, a.NumberOfEmployees AS Employee_Count,
  o.Name, o.AccountId, o.Amount, o.TotalOpportunityQuantity, o.CloseDate, o.Type, o.IsWon,
  t.Subject, t.ActivityDate, t.Type, t.WhatId, t.Status,
  s.SBQQ__AverageCustomerDiscount__c,
  f.PeriodId, f.StartDate, f.QuotaAmount,
  oli.Billing_Frequency__c, oli.Vertical__c,
  sc.Quote_Price_Cap__c, sc.Term,
  r.Account_Name_Client_Size, r.Account_Owner, r.Account_Name_Vertical
FROM account a
LEFT JOIN task t
ON a.OwnerId = t.OwnerId
AND a.Id = t.AccountId
LEFT JOIN opportunity o
ON o.AccountId = a.Id
LEFT JOIN forecastingquota f
ON  t.OwnerId = f.QuotaOwnerId
LEFT JOIN sbqq__quote__c s
ON t.OwnerId = s.OwnerId
LEFT JOIN opportunitylineitem oli
ON oli.OpportunityId = o.Id
LEFT JOIN servicecontract sc
ON sc.AccountId = o.AccountId
LEFT JOIN rubybenchmarking r
ON r.Account_Name_18_digit_ID = o.AccountId
LEFT JOIN opportunityhistory oh
ON oh.OpportunityId = o.Id
WHERE IsWon = TRUE AND t.Status = 'Completed' AND o.Type = 'New Customer'
""")
display(final_query)
```



Data Exploration & Cleaning

- 43 Salesforce (SFDC) tables
- SQL queries on predictor candidates
- Predictor, Dependent variables identified
- 9 Tables in first join
- 6 Tables in second join
- Descriptive statistics analyzed
- Data visualizations
- Outliers, null values removed
- Data formats (ie, Timestamp to Date)
- Categories and counts created from Tasks
- 30 + calculated fields

SOLUTION: EXPERIMENTS

- **Experiment A** - sales divisions aggregated:
  - Logistic Regression (80/20 train and test split)
  - Logistic Regression (K-Fold validation)
  - Random Forest (K-Fold validation)

- **Experiment B** - feature engineering modifications, SMB/MM sales divisions:
  - Linear Regression for SMB (K-fold validation, t and p testing)
  - Linear Regression for Mid-Market (K-fold validation t and p testing)
  - Multiple Linear Regression models for SMB (K-fold validation, t and p testing)
  - Multiple Linear Regression models for MM (K-fold validation, t and p testing)

Multiple types of Regression and Validation Techniques used to Identify Best Model

SOLUTION: PEARSON CORRELATIONS

Experiment A: All Sales Divisions

Feature	Dependent Variable: Percent To Quota (0 to 1)
Avg_Opp_Amount	0.8070
Avg_Sell_Cycle	0.2990
Total_Demo_Request_Per_Week	-0.0300
Avg_Close_Rate	-0.1890
Avg_Num_Products_Per_Opp	-0.0200
Avg_Term_Length	-0.1190
Total_Emails_Per_Week	-0.0923
Total_Prospect_Calls_Per_Week	No Correlation run because visuals did not support
Total_Customer_Calls_Per_Week	No Correlation run because visuals did not support
Total_Conversations_Per_Week	-0.0312
Total_Set_Demos_Per_Week	-01830
Total_Performed_Demos_Week	-0.1400
Total_Demo_Requests_Per_Week	-0.0300
Total_Online_Meetings_Per_Week	-0.2310

Experiment B: SMB vs. MM Sales Divisions

Experiment B. SMB Pearson's Correlation Results:

- Correlation between Percent To Quota and Average Opp Amount: 0.37218218071266135
- Correlation between Percent To quota and Average Cust Discount: 0.423386881349258
- Correlation between Percent To quota and Total Avg Weekly Activities: -0.20538575028381872

Experiment B. MM Pearson's Correlation Results:

- Correlation between Percent To Quota and Average Opp Amount: -0.12022508786589424
- Correlation between Percent To quota and Average Cust Discount: 0.31834755561553063
- Correlation between Percent To quota and Total Avg Weekly Conversations: 0.010416490833213958
- Correlation between Percent To quota and Total Number of Accounts: -0.27548335889903575
- Correlation between Percent To quota and Total Online Meetings Per Week: 0.2249661298099106

SOLUTION: RESULTS & MODEL VALIDATION

Experiment	Division	Regression	Validation	Features	Results	T-Test and P-Test Results
Experiment A1	All	Logistic	80/20	Avg_Opp_Amount	ROC = 0.5378	N/A = Did not perform because ROC was too low and comparable to random chance
Experiment A2	All	Logistic	K-Fold	Avg_Opp_Amount	ROC = 0.6127	N/A – Did not perform because although improved, ROC value was still too low and comparable to random chance
Experiment A3	All	Random Forest	K-Fold	Avg_Opp_Amount Avg_Sell_Cycle Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.12396054358188907 Feature Importances: Avg_Opp_Amount: .5582428319309904 Avg_Sell_Cycle: 0.23379201565840949 Avg_Cust_Disc: 0.2079651524106001	N/A – Did not perform because RMSE was too high given the Percent To Quota scale of 0-1
Experiment B1	SMB	Linear	K-Fold	Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.11367827283577361 R-squared (R2): -0.939068077221632	Intercept: 0.21334034795651072 Coefficient 0: 0.00328007258819299T- Coefficient 0 : T-statistic = 2.5941296695425082, P-value = 0.014919053689924855 Coefficient 1: T-statistic = 6.560752017245371, P-value = 4.103771282792934e-07
Experiment B2	SMB	Multiple	K-Fold	Avg_Opp_Amount Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.08129527906544977 R-squared (R2): 0.5695292561191588	Intercept: 0.16387927163782137 Coefficient 0: Estimate = 4.152605107643032e-06, T-value = 2.451509636119704, P-value = 0.020981244195025672 Coefficient 1: Estimate = 0.0032705427993116173, T-value = 2.810376139203201, P-value = 0.009096325199308364
Experiment B3	MM	Linear	K-Fold	Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.13518621444979825 R-squared (R2): -0.07320713457562689	Intercept: 0.3226327201988182 Coefficient 0: T-statistic = 1.0496002531095305, P-value = 0.3105194110454699 Coefficient 1: T-statistic = 9.730423949339313, P-value = 7.150270620037702e-08
Experiment B4	MM	Multiple	K-Fold	Avg_Cust_Disc Total_Online_Meetings_Per_Week	Root Mean Squared Error (RMSE): 0.1272495380644176 R-squared (R2): 0.04910805985992739	Intercept: 0.3123193943052424 Coefficient 0: Estimate = 0.3123193943052424, T-value = 0.22607205349133713, P-value = 0.8356722409559736 Coefficient 1: Estimate = 0.0006449522076622271, T-value = 0.20832566399276212, P-value = 0.8483167022891438

SOLUTION: MODEL SELECTION

- P-values significant (<.05) for both features
  - P-value for Avg_Opp_Amount = 0.020981244195025672
  - P-value for Avg_Cust_Disc = 0.009096325199308364

- Indicates both variables are predictors
- RMSE, R2 (goodness of fit) values acceptable
  - RMSE = 0.08129527906544977 lowest error of all models
  - R2 = 0.5695292561191588 ; highest of all models

- Indicates 8% error and nearly 60% variance in dependent variable that can be explained by independent variable

SMB Multiple Linear Regression Model Selected

SOLUTION: MODEL SELECTION

Multiple Linear Regression for SMB - best fitting model.

Multiple Regression Equation:

$Y = \beta 0 + \beta 1 X1 + \beta 2 X2$,

where β0 is the intercept, β1 is X1 coefficient, β2 is the X2 coefficient

Final Model Equation:

"Percent_To_Quota" = 0.16387927163782137 +
4.152605107643032e-06 x (Feature_1) +
0.003270542799311617 3 x (Feature_2),

where Feature 1 = Avg_Opp_Amount and Feature 2 = Avg_Cust_Disc.

SOLUTION: SMB

2 Predictors and Relevant Benchmarks can be used as performance indicators

SMB Benchmarks: Standards of Performance

summary	Total_Num_Accounts	Avg_Opp_Amount	Avg_Sell_Cycle	Avg_Close_Rate	Avg_Term_Length	Avg_Cust_Disc	Total_Activities_Per_Week	Total_Emails_Per_Week	Total_Set_Demos_Per_Week	Total_Performed_Demos_Per_Week
count	36	36	36	36	36	36	36	36	36	36
mean	1405.75	11854.56	587.7357	0.429846	33.7318	18.37899	109.4613	61.52405	0.940713	1.03234
stddev	1400.527	11266.95	483.1692	0.115733	7.346555	16.50292	44.46695	34.16769	0.929752	0.817602
min	1	562.95	68.5	0.129808	17.98619	0.391009	5.2883	3.758743	0	0.030852
25%	403	5584.667	152.874	0.368952	28.54474	8.126098	78.04559	33.24151	0.060049	0.365462
50%	1040	8751.83	336	0.415	33.89443	14.16049	99.2165	57.11348	0.659341	0.8262
75%	1718	14325.87	1029.333	0.490518	38.0725	22.47764	138.6264	78.27534	1.359223	1.64572
max	5432	61531.43	1433.889	0.707521	47.41805	95.72791	208.335	143.2419	3.533981	3.479612

Reject null hypotheses; conclude Avg_Opp_Amount and Avg_Cust_Disc are positively correlated with quota attainment for SMB

SOLUTION: MID-MARKET

- Relevant Benchmarks -Descriptive Statistics still very helpful in use as performance indicators

MM Benchmarks: Standards of Performance

summary	Total_Num_Accounts	Avg_Opp_Amount	Avg_Sell_Cycle	Avg_Close_Rate	Avg_Term_Length	Avg_Cust_Disc	Total_Activities_Per_Week	Total_Emails_Per_Week	Total_Set_Demos_Per_Week	Total_Performed_Demos_Per_Week
count	22	22	22	22	22	22	22	22	22	22
mean	1084.455	28156.34	944.896	0.440551	36.65357	21.35331	112.1941	66.73564	0.785048	0.863651
stddev	904.3352	16331.04	504.2868	0.101498	5.414118	18.21201	46.81782	33.16598	0.842456	0.8077
min	19	8149.73	118.7039	0.181595	28.45822	4.898037	5.2883	3.758743	0	0.030852
25%	471	17760.42	533.1429	0.370569	33.87805	11.68766	94.00776	47.30941	0.060049	0.365462
50%	894	21933.75	1191.25	0.421137	35.83924	20.0265	99.2165	61.48953	0.448109	0.552941
75%	1398	39193.74	1365	0.491098	40.54902	23.19539	142.4739	78.31481	1.269641	1.31165
max	4168	73008.17	1499.5	0.677712	47.41805	95.72791	208.335	143.2419	3.214563	3.479612

Accept null hypotheses; conclude there are no known predictors correlated with quota attainment for MM

CODE WALKTHROUGH: DATABRICKS

CHALLENGES

CHALLENGES: INTEGRATION STRATEGY

- Results incorporated into PowerPoint for Sales Leadership
- Explore adoption of new weekly activity metrics
- Re-aligning sales territories not possible
- Integration of predictors into reports not recommended - discounting controversial
- Integrate benchmarks as threshold metrics required for success.

CHALLENGES: CAPSTONE COMPLEXITY

- 43 SFDC object tables analyzed
- Unfamiliar Cloud Computing and Software
- 9 tables joined initially; 6 tables joined in final project
- Strategy change on joins - 1 week
- Unexpected null values for key data
- Extensive data wrangling and cleaning
- Summary statistics performed to understand distribution and anomalies
- 7 different regression models, multiple validation techniques
- Data Complexity- millions of rows of activity data
- Data results illogical at times
- Results interpretation more complex
- Unanticipated small n size after data cleaning

CHALLENGES: WHAT HAVE I LEARNED?

- There is no shame in asking for help
- You know far more than you think you know
- Coding error messages are your friends
- Regardless of software differences, coding concepts still apply
- Adaptability and ongoing learning are key to success as a Data Scientist
- Projects are never really “done” – there are always areas for improvement
- Dig deeper on illogical data and results
- Be agile on changing direction but document

WHAT'S NEXT

RECOMMENDATIONS

FUTURE RECOMMENDATIONS

MORE DATA COLLECTION

Continue to collect more data to improve correlations

FEATURE ENGINEERING

Change dependent variable, predictors so that more data can be used

EVOLUTION OF BENCHMARK DATA

Analyze the positive and negative trends in benchmark data

SALES OPERATIONS / DATA CONSISTENCY

Demo request vetting / completion, Territory Assignments

REFERENCES

Mckinney, Wes (2022). Python for Data Analysis, Third Edition. Published by O'Reilly Media,

Geron, Aurelien (2023). Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 3rd Edition. Published by O'Reilly Media Inc.

Lubanovic, Bill (2020). Introducing Python, Second Edition. Published by O'Reilly Media, Inc.

VanderPlas, Jake. (December 2016). Python Data Science Handbook, First Edition. Published by O'Reilly Media, Inc.

Matthes, Eric (2019). Python Crash Course, Second Edition. Published by No Starch Press.
.
Etaati, Leila (June 26, 2018). Azure Databricks Part 2. Published by Radacad.com.

Sruthi E R (Updated October 26, 2023). Understand Random Forest Algorithms With Examples. Analytics Vidhya (https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/#:~:text=A.%20Random%20Forest%20is%20a,and%20outliers%20in%20the%20data.)

Almaliki, Zaid Alisse (March 19, 2019). Do you know how to choose the right learning algorithm array among 7 different types? Towards Data Science (www.towardsdatascience.com)

VanDerwerf, Paul (August 26, 2020). Train-Test Split for Evaluating Machine Learning Algorithms. Machine Learning Mastery (www.machinelearningmastery.com)

Sap.com (2023). SAP Predictive Analytics https://help.sap.com/docs/SAP_PREDICTIVE_ANALYTICS/41d1a6d4e7574e32b815f1cc87c00f42/5e5198fd4afe4ae5b48fefe0d3161810.html

Ray, Sunil (Updated September 25, 2023). 8 Ways to Improve Accuracy of Machine Learning Models. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/#:~:text=There%20are%20several%20ways%20to,bagging%2C%20boosting%2C%20and%20stacking.

Stackexchange (2015). Linear regression: Intercept isn't significant. https://stats.stackexchange.com/questions/160628/linear-regression-intercept-isnt-significant

Wikipedia (2023). Coefficient of Determination.https://en.wikipedia.org/wiki/Coefficient_of_determination#:~:text=There%20are%20cases%20where%20R,fitting%20procedure%20using%20those%20data.

Databricks.com

Spark.apache.org

Coding Assistance and Troubleshooting:
Databricks Assistant (integrated AI assistant in Databricks to troubleshoot coding errors and issues)
Google.com
Stackoverflow.com

Sales Standards of Excellence

Summary of Findings

PRESENTER

Diana Bowden

Account Executive

Agenda

Project Goal

Process

Results

Benchmarks

Integration Strategy

Project Goal

Model the most successful sales representatives to replicate best selling practices.

What are the top predictors for success for sales representatives at Relias?

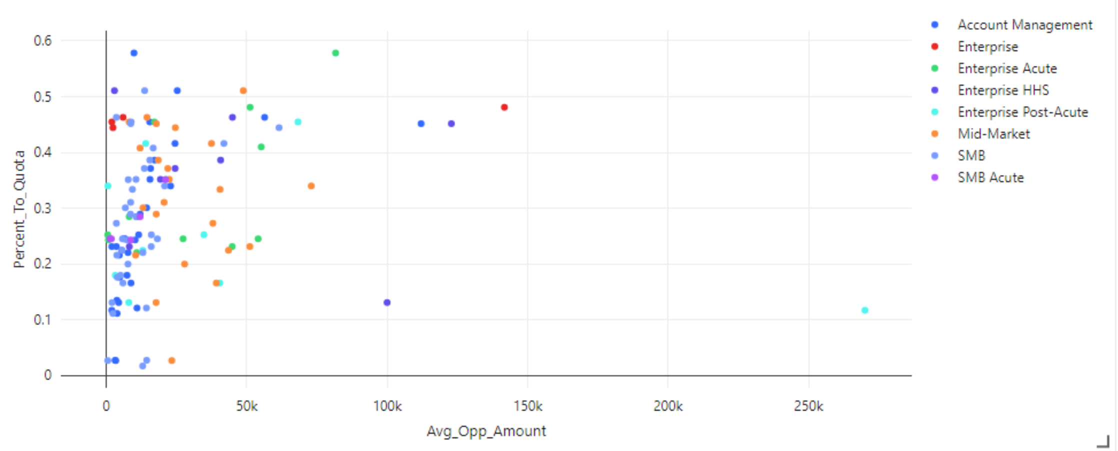
Process

Edit View Run Help Last edit was 24 days ago Provide feedback

```
final_query = spark.sql("""
SELECT
  a.OwnerId, a.Name AS Account_Name, a.Type AS Account_Type, a.BillingState AS Billing_State, a.NumberOfEmployees AS Employee_Count,
  o.Name, o.AccountId, o.Amount, o.TotalOpportunityQuantity, o.CloseDate, o.Type, o.IsWon,
  t.Subject, t.ActivityDate, t.Type, t.WhatId, t.Status,
  s.SBQQ__AverageCustomerDiscount__c,
  f.PeriodId, f.StartDate, f.QuotaAmount,
  oli.Billing_Frequency__c, oli.Vertical__c,
  sc.Quote_Price_Cap__c, sc.Term,
  r.Account_Name_Client_Size, r.Account_Owner, r.Account_Name_Vertical
FROM account a
LEFT JOIN task t
ON a.OwnerId = t.OwnerId
AND a.Id = t.AccountId
LEFT JOIN opportunity o
ON o.AccountId = a.Id
LEFT JOIN forecastingquota f
ON  t.OwnerId = f.QuotaOwnerId
LEFT JOIN sbqq__quote__c s
ON t.OwnerId = s.Ownerid
LEFT JOIN opportunitylineitem oli
ON oli.OpportunityId = o.Id
LEfT JOIN servicecontract sc
ON sc.AccountId = o.AccountId
LEFT JOIN rubybenchmarking r
ON r.Account_Name_18_digit_ID = o.AccountId
LEFT JOIN opportunityhistory oh
ON oh.OpportunityId = o.Id
WHERE IsWon = TRUE AND t.Status = 'Completed' AND o.Type = 'New Customer'
""")
display(final_query)
```

(26) Spark Jobs

Table Visualization 1 Visualization 2 Visualization 3 Visualization 4 +



- 43 Salesforce (SFDC) tables
- SQL queries on predictor candidates
- Predictor, Dependent variables identified
- Joined tables
- Analyzed descriptive statistics
- Data visualizations
- Outliers, null values removed
- Data formats changes
- Consolidated Tasks categories
- 30 + calculated fields
- Regression used to find predictors

Extensive data exploration, cleaning and manipulation performed

Experiments

- **Experiment A** - sales divisions aggregated:
  - Logistic Regression (80/20 train and test split)
  - Logistic Regression (K-Fold validation)
  - Random Forest (K-Fold validation)

- **Experiment B** - feature engineering modifications, SMB/MM sales divisions:
  - Linear Regression for SMB (K-fold validation, t and p testing)
  - Linear Regression for Mid-Market (K-fold validation t and p testing)
  - Multiple Linear Regression models for SMB (K-fold validation, t and p testing)
  - Multiple Linear Regression models for MM (K-fold validation, t and p testing)

Multiple types of Regression and Validation Techniques used to Identify Best Model

 RELIAS

Results : Pearson Correlations

Experiment A: Aggregated sales divisions

Feature	Dependent Variable: Percent To Quota (0 to 1)
Avg_Opp_Amount	0.8070
Avg_Sell_Cycle	0.2990
Total_Demo_Request_Per_Week	-0.0300
Avg_Close_Rate	-0.1890
Avg_Num_Products_Per_Opp	-0.0200
Avg_Term_Length	-0.1190
Total_Emails_Per_Week	-0.0923
Total_Prospect_Calls_Per_Week	No Correlation run because visuals did not support
Total_Customer_Calls_Per_Week	No Correlation run because visuals did not support
Total_Conversations_Per_Week	-0.0312
Total_Set_Demos_Per_Week	-01830
Total_Performed_Demos_Week	-0.1400
Total_Demo_Requests_Per_Week	-0.0300
Total_Online_Meetings_Per_Week	-0.2310

Experiment B: SMB vs. MM sales divisions

Experiment B. SMB Pearson's Correlation Results:

- Correlation between Percent To Quota and Average Opp Amount: 0.37218218071266135
- Correlation between Percent To quota and Average Cust Discount: 0.4233868881349258
- Correlation between Percent To quota and Total Avg Weekly Activities: -0.20538575028381872

Experiment B. MM Pearson's Correlation Results:

- Correlation between Percent To Quota and Average Opp Amount: -0.12022508786589424
- Correlation between Percent To quota and Average Cust Discount: 0.31834755561553063
- Correlation between Percent To quota and Total Avg Weekly Conversations: 0.010416490833213958
- Correlation between Percent To quota and Total Number of Accounts: -0.27548335889903575
- Correlation between Percent To quota and Total Online Meetings Per Week: 0.2249661298099106

Results: Model Validation

Experiment	Division	Regression	Validation	Features	Results	T-Test and P-Test Results
Experiment A1	All	Logistic	80/20	Avg_Opp_Amount	ROC = 0.5378	N/A = Did not perform because ROC was too low and comparable to random chance
Experiment A2	All	Logistic	K-Fold	Avg_Opp_Amount	ROC = 0.6127	N/A – Did not perform because although improved, ROC value was still too low and comparable to random chance
Experiment A3	All	Random Forest	K-Fold	Avg_Opp_Amount Avg_Sell_Cycle Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.12396054358188907 Feature Importances: Avg_Opp_Amount: .5582428319309904 Avg_Sell_Cycle: 0.23379201565840949 Avg_Cust_Disc: 0.2079651524106001	N/A – Did not perform because RMSE was too high given the Percent To Quota scale of 0-1
Experiment B1	SMB	Linear	K-Fold	Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.11367827283577361 R-squared (R2): -0.939068077221632	Intercept: 0.21334034795651072 Coefficient 0: 0.00328007258819299T- Coefficient 0 : T-statistic = 2.5941296695425082, P-value = 0.014919053689924855 Coefficient 1: T-statistic = 6.560752017245371, P-value = 4.103771282792934e-07
Experiment B2	SMB	Multiple	K-Fold	Avg_Opp_Amount Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.08129527906544977 R-squared (R2): 0.5695292561191588	Intercept: 0.16387927163782137 Coefficient 0: Estimate = 4.152605107643032e-06, T-value = 2.451509636119704, P-value = 0.020981244195025672 Coefficient 1: Estimate = 0.0032705427993116173, T-value = 2.810376139203201, P-value = 0.009096325199308364
Experiment B3	MM	Linear	K-Fold	Avg_Cust_Disc	Root Mean Squared Error (RMSE): 0.13518621444979825 R-squared (R2): -0.07320713457562689	Intercept: 0.3226327201988182 Coefficient 0: T-statistic = 1.0496002531095305, P-value = 0.3105194110454699 Coefficient 1: T-statistic = 9.730423949339313, P-value = 7.150270620037702e-08
Experiment B4	MM	Multiple	K-Fold	Avg_Cust_Disc Total_Online_Meetings_Per_Week	Root Mean Squared Error (RMSE): 0.1272495380644176 R-squared (R2): 0.04910805985992739	Intercept: 0.3123193943052424 Coefficient 0: Estimate = 0.3123193943052424, T-value = 0.22607205349133713, P-value = 0.8356722409559736 Coefficient 1: Estimate = 0.0006449522076622271, T-value = 0.20832566399276212, P-value = 0.8483167022891438

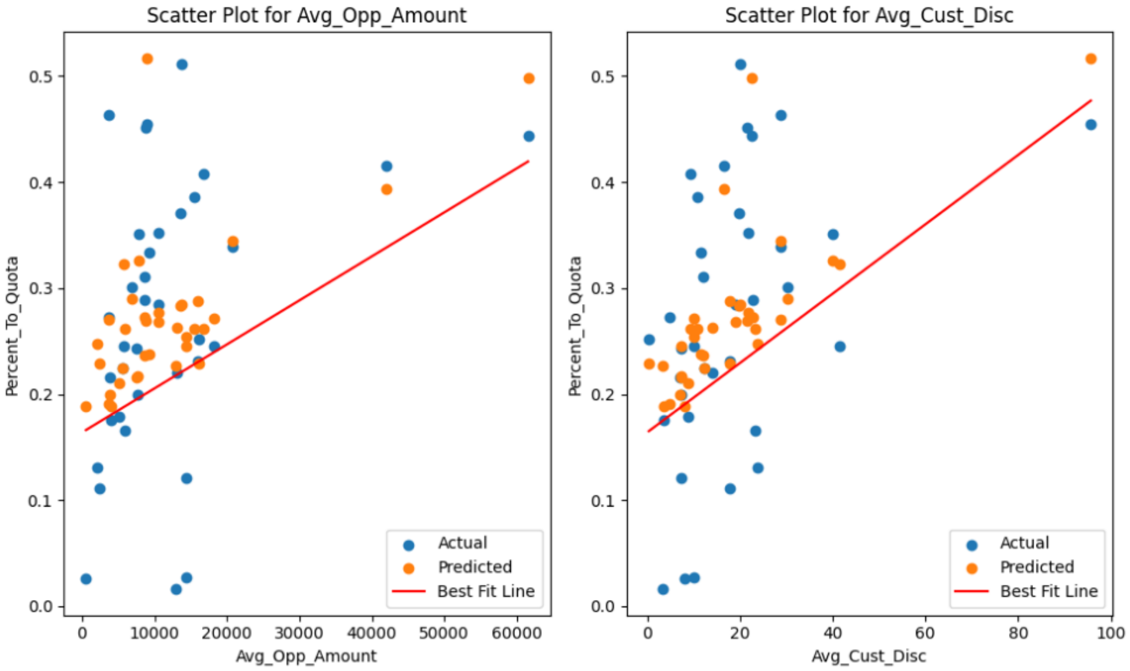
Results

Multiple Linear Regression for SMB - best fitting model.

Multiple Regression Equation:

$Y = \beta 0 + \beta 1 X1 + \beta 2 X2$,

where ß0 is the intercept, ß1 is X1 coefficient, ß2 is the X2 coefficient



Final Model Equation:

“Percent_To_Quota” = 0.16387927163782137 + 4.152605107643032e-06 x (Feature_0) + 0.0032705427993116173 x (Feature_1),

where Feature 0 = Avg_Opp_Amount and Feature 1 = Avg_Cust_Disc.

Conclusion: SMB

- Relevant Benchmarks
  - Highly correlated predictors: Avg Opp Amount, Avg Cust Disc
  - Activity metrics

SMB Benchmarks: Standards of Performance

summary	Total_Num_Accounts	Avg_Opp_Amount	Avg_Sell_Cycle	Avg_Close_Rate	Avg_Term_Length	Avg_Cust_Disc	Total_Activities_Per_Week	Total_Emails_Per_Week	Total_Set_Demos_Per_Week	Total_Performed_Demos_Per_Week
count	36	36	36	36	36	36	36	36	36	36
mean	1405.75	11854.56	587.7357	0.429846	33.7318	18.37899	109.4613	61.52405	0.940713	1.03234
stddev	1400.527	11266.95	483.1692	0.115733	7.346555	16.50292	44.46695	34.16769	0.929752	0.817602
min	1	562.95	68.5	0.129808	17.98619	0.391009	5.2883	3.758743	0	0.030852
25%	403	5584.667	152.874	0.368952	28.54474	8.126098	78.04559	33.24151	0.060049	0.365462
50%	1040	8751.83	336	0.415	33.89443	14.16049	99.2165	57.11348	0.659341	0.8262
75%	1718	14325.87	1029.333	0.490518	38.0725	22.47764	138.6264	78.27534	1.359223	1.64572
max	5432	61531.43	1433.889	0.707521	47.41805	95.72791	208.335	143.2419	3.533981	3.479612

Reject null hypotheses; conclude Avg Opp Amount and Avg Cust Discount positiviely correlated with quota attainment for SMB

Conclusion: MM

- Relevant Benchmarks -Descriptive Statistics still very helpful in use as performance indicators

MM Benchmarks: Standards of Performance

summary	Total_Num_Accounts	Avg_Opp_Amount	Avg_Sell_Cycle	Avg_Close_Rate	Avg_Term_Length	Avg_Cust_Disc	Total_Activities_Per_Week	Total_Emails_Per_Week	Total_Set_Demos_Per_Week	Total_Performed_Demos_Per_Week
count	22	22	22	22	22	22	22	22	22	22
mean	1084.455	28156.34	944.896	0.440551	36.65357	21.35331	112.1941	66.73564	0.785048	0.863651
stddev	904.3352	16331.04	504.2868	0.101498	5.414118	18.21201	46.81782	33.16598	0.842456	0.8077
min	19	8149.73	118.7039	0.181595	28.45822	4.898037	5.2883	3.758743	0	0.030852
25%	471	17760.42	533.1429	0.370569	33.87805	11.68766	94.00776	47.30941	0.060049	0.365462
50%	894	21933.75	1191.25	0.421137	35.83924	20.0265	99.2165	61.48953	0.448109	0.552941
75%	1398	39193.74	1365	0.491098	40.54902	23.19539	142.4739	78.31481	1.269641	1.31165
max	4168	73008.17	1499.5	0.677712	47.41805	95.72791	208.335	143.2419	3.214563	3.479612

Accept null hypotheses; conclude there are no predictors correlated with quota attainment for MM

Integration Strategy

Recommendations

- Gather more data to increase n and improve results
- Explore adoption of new weekly activity metrics
- Re-aligning sales territories not possible at this time
- Re-examine discounting practices and policies
- Integrate benchmarks as threshold metrics for success.
- Explore changes in key performance metrics over time

THANK YOU