**Name:** Rohan Deshmukh
**Position Applied For:** Data Scientist
**Submission Date:** 01/31/2022

**Query for table creation:**

```
CREATE TABLE `takehome`.rentals (
        rental_id BIGINT NULL,
        user_id BIGINT NULL,
        order_placed_ts TIMESTAMP NULL,
        product_category varchar(30) NULL,
        product varchar(3) NULL,
        order_canceled_ts TIMESTAMP NULL)
```

## Question 1 Response:

```
WITH cte AS
(SELECT DISTINCT product_category, product,
ROW_NUMBER() OVER (PARTITION BY product_category ORDER BY count(product) DESC)rn
FROM rentals
WHERE order_canceled_ts IS NULL
GROUP BY product_category,product)

SELECT product_category, product
FROM cte
WHERE rn<=5
```

**Query results attached in the appendix section**

## Question 2 Response:

```
WITH cte AS
(SELECT *, ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY order_canceled_ts
DESC) rn
FROM rentals
WHERE order_canceled_ts IS NOT NULL)

SELECT t1.user_id
FROM cte t1 JOIN cte t2
ON t1.user_id=t2.user_id
AND t1.rn=t2.rn+1
WHERE t1.user_id in (SELECT user_id FROM cte WHERE rn>=2)
AND abs(DATEDIFF(t1.order_canceled_ts,t2.order_canceled_ts))<30
```

| **user_id** |
|---|
| **11345263704829** |

## Question 3 Response:

```
SELECT rental_id
FROM rentals
WHERE SUBSTRING(product, 1, 1) != SUBSTRING(product_category, 1, 1)
```

**| rental_id       |**
**|-----------------|**
**| 247332742048526 |**

## Question 4 Response:

**Assumption**: Every unique user add at least 1 of the three options to the cart.

**Given values**

Overall conversion rate:  5.6%

# of sessions: 308000

# of total conversions: 5.6% of 308000 = 17248

**Monthly subscription**

Monthly Cart Conversion rate: 0.9%

Monthly Add to cart rate: 6.5% = (# of carts with monthly / # of sessions) * 100

# of carts with monthly: 20020

# of conversions for monthly: 0.9% * # of carts with monthly = 180

**Short term**

Short term Add to cart rate: 7.1% = (# of carts with short term / # of sessions) * 100

# of carts with short term: 21868

# Monthly Abandoned cart rate: 74% = ( 1 - (# of conversions short term / # of carts with short term))*100

# of conversions short term = 5685

Short term Car Conversion rate: 26%

**Purchase**

# of conversions purchase = # of total conversion - # of conversions monthly - # of conversions short term =  17248 -180 - 5685 = 11383

# of carts with purchase = # of sessions - # of carts with monthly - # of carts with short term = 308000 - 20020 - 21868 = 266132

Purchase conversion rate: # of conversions short term / # of carts with purchase * 100 = 11383 / 266132 *100 = 4.27%

**Conversion rate for clothing purchases: 4.27%**

## Question 5 Response:

**Null hypothesis: Proportions from the two populations are the same.**
**Alternative hypothesis: Proportions from the two populations are not the same.**

```
import numpy as np
from statsmodels.stats.proportion import proportions_ztest

# significance as it is two tailed test
significance = 0.025

# declaring the samples
sample_success_a, sample_size_a = (1748, 22039)
sample_success_b, sample_size_b = (1801, 21561)

# declaring the count and nobs parameter values
successes = np.array([sample_success_a, sample_success_b])
samples = np.array([sample_size_a, sample_size_b])

# two sided since we are just checking if the proportions are different rather than if one is greater than the other
stat, p_value = proportions_ztest(count=successes, nobs=samples,  alternative='two-sided')

# two proportion z-test
print('z_stat: %0.3f, p_value: %0.3f' % (stat, p_value))
if p_value > significance:
  print ("Fail to reject the null hypothesis.")
else:
  print ("Reject the null hypothesis - suggest the alternative hypothesis is true")
```

**z_stat: -1.610, p_value: 0.107**
**Fail to reject the null hypothesis.**

**Thus, the difference between two sample means is not statistically significant at a 95% confidence level.**

## Question 6 Response:

Upon analyzing the confusion matrix, we can see that True positive is 9% and False Positive is 8%. So, if you consider the positive (1) prediction, 53% of the prediction is correct. This 53% is calculated by measuring how much is 9 of 17 (9%+8%).

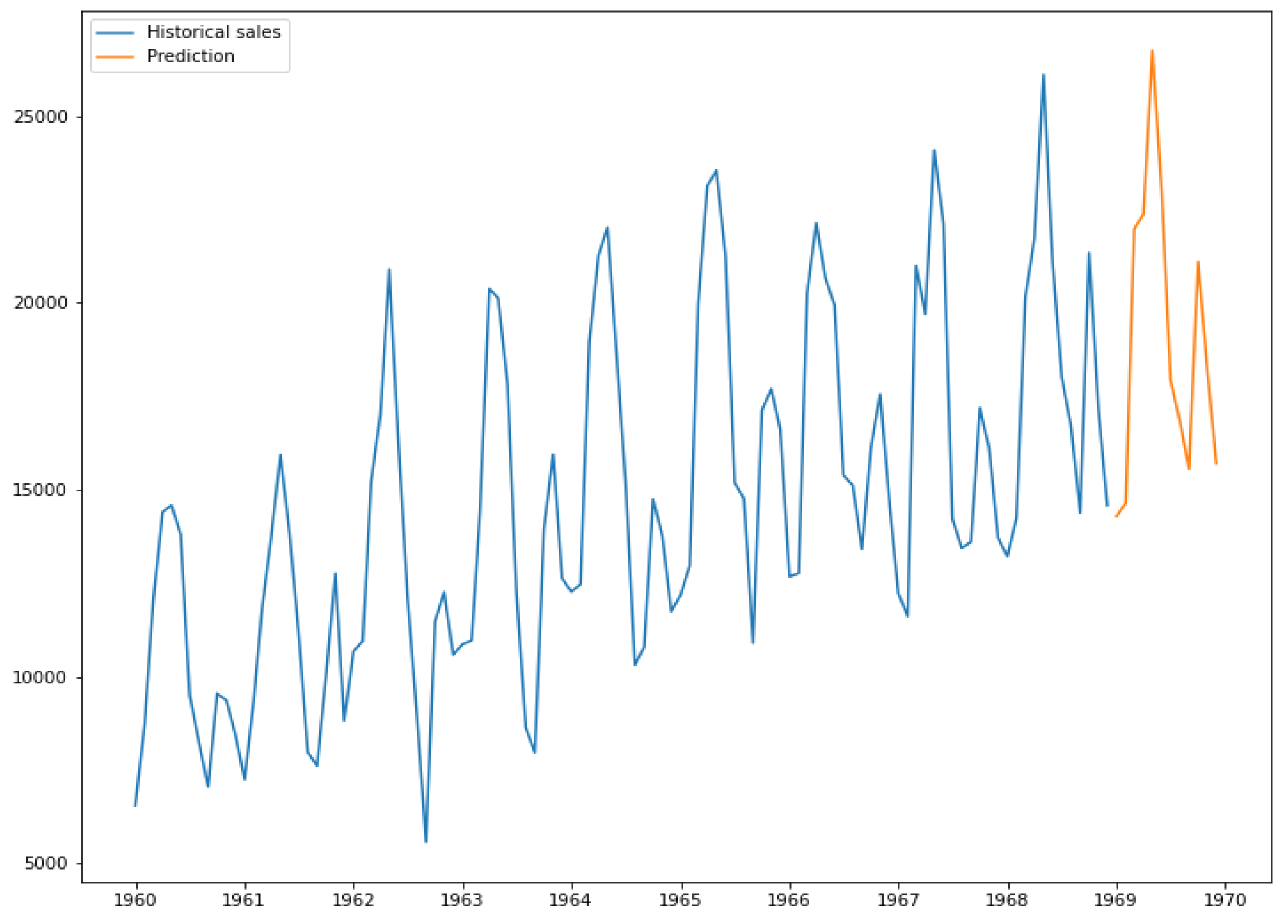
For instance, if our model predicts that 100 customers churn next month (1-positive), the model correctly identifies 53 customers that are going to churn. If we denote X as promotional amount per customer, we have an equation where money spent is equal to money earned for breaking even:

**Money spent on promotional campaign = Money earned on retained subscriber**

*100X = 200 * 53*
*X = (200 * 53)/100*
*X = 106*
**Thus, spending $106 per customer for promotion, the campaign can break even.**

## Question 7 Response:

Prediction for 1969:




*#!/usr/bin/env python*
*# coding: utf-8*

*# # Car Sales Forecasting*

*# #### Problem statement: Given historical monthly car sales data, predict car sales for 12 months in future*

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, acf, pacf,arma_order_select_ic
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
from pmdarima.arima import auto_arima

import warnings
warnings.filterwarnings("ignore")

pl.rcParams['figure.figsize'] = (12, 10)

# ### Loading data & basic preprocessing

# load data
ts = pd.read_csv('monthly-car-sales.csv')
ts.tail()

# **As per the given dataset, 'Sales' column has monthly sales for corresponding month in the
'Month' column**

ts['Month'] = pd.to_datetime(ts['Month'])
ts

# **It is imperative to check for the datatype of the columns before we proceed with our EDA and
modeling steps.
# It seems the 'Month' column is loaded as an object datatype. We will have to format the column
correctly to date datatype.**
# **Also, no missing values can be seen in the dataset**

ts['Month'] = pd.to_datetime(ts['Month'])
ts.set_index('Month', inplace=True)
ts.info()

# ### Exploratory Data Analysis

# Plotting car sales through the years
plt.figure(figsize=(14, 10))
plt.title('Total Car Sales in Quebec province between 1960-1968')
plt.xlabel('Year')
plt.ylabel('Unit of cars sold')
```

```
plt.plot(ts);

# **Some of the observations from the above plot.**
# **1. There is an obvious "seasonality" (for eg: peak car sales around a particular time of year**
# **2. There is an increasing "Trend" (sales have increased with each passing year).**
# **3. The seasonal signal appears to be growing over time, suggesting a multiplicative relationship
(increasing change)**
# **4. There appears to be 1 obvious outlier somewhere towards the end of 1962.**
# **5. The seasonality suggests that the series is almost certainly non-stationary.**

# Time series decomposition using multiplicative model
res = sm.tsa.seasonal_decompose(ts, period=12, model="multiplicative")
res.plot()
plt.show()

# **Stationarity refers to time-invariance of a series. Two points in a time series are related to each
other by only how far apart they are, and not by the direction(forward/backward). When a time series
is stationary, it can be easier to model. Statistical modeling methods assume or require the time series
to be stationary.**
#
# **There are multiple tests that can be used to check stationarity. One such test is ADF(Augmented
Dicky Fuller Test)**

# Checking for stationarity of the time series
def test_stationarity(ts):
   df_adf = adfuller(ts, autolag='AIC')
   dfoutput = pd.Series(df_adf[0:4], index=['Test Statistic','p-value','Lags Used','Number of
Observations Used'])
   for key,value in df_adf[4].items():
       dfoutput['Critical Value (%s)'%key] = value
   print (dfoutput)

test_stationarity(ts)

# **The ADF test is a type of statistical test called a unit root test.
# The null hypothesis of the test is that the time series can be represented by a unit root, that it is not
stationary. The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.
# From the above test statistics, we can see the p-value > 0.05. Thus, we fail to reject the null
hypothesis and this indicates that the time series has a unit root and it is non-stationary.**

# **Observing from the decomposed plot of the time series, we can see sales has an increasing trend.
One of the preprocessing steps we can perform is detrending (removing trend of from the time series)
the timeseries. This helps to more easily observe subtrends in the data that are seasonal.**

# Detrending the time series
def difference(data, interval=1):
```

```
    diff = list()
    for i in range(interval, len(data)):
        value = data[i] - data[i - interval]
        diff.append(value)
    return pd.Series(diff)

# Plotting OG, detrended and differenced ts
plt.figure(figsize=(16,16))

plt.subplot(311)
plt.title('Original')
plt.xlabel('Time')
plt.ylabel('Sales')
plt.plot(ts)

plt.subplot(312)
plt.title('After De-trend')
plt.xlabel('Time')
plt.ylabel('Sales')
new_ts=difference(ts.values)
plt.plot(new_ts)
plt.plot()

plt.subplot(313)
plt.title('After Deseasonalization')
plt.xlabel('Time')
plt.ylabel('Sales')
new_ts=difference(ts.values, 12)       # assuming the seasonality is 12 months long
plt.plot(new_ts)
plt.plot()

# Using ADF to check if the detrending and deseasonalization made the time series stationary
test_stationarity(new_ts)

# **From the above test statistics, p-value for the ADF test is less than 0.05. Hence we can reject null hypothesis and assume stationarity of the series**
# **The original series can be derived back using the inverse transform function.**

# ## Modeling
# **Now that we have made the time series stationary, we can start with the modeling process. The baseline model that we can experiment with is ARIMA as it is one of the basic yet effective statistical model for forecasting**
# **For ARIMA, we have the following parameters:**
# p: The number of lag observations included in the model, also called the lag order.
# d: The number of times that the raw observations are different, also called the degree of differencing.
```

```
# q: The size of the moving average window, also called the order of moving average.
# **In our case, we have already observed that the d = 1 as after differencing the time series, it
became stationary.**

# prepare training dataset
X = ts
split = int(len(X) * 0.75)
train, test = X[0:split], X[split:]

# plotting train and test dataset
plt.plot(train)
plt.plot(test)

# ### Approach I: ARIMA model and manual optimization of p, d, q parameters

# Modeling approach I: Evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):

    # prepare training dataset
    split = int(len(X) * 0.66)
    train, test = X[0:split], X[split:]
    history = [x for x in train]

    # make predictions
    predictions = list()
    for t in range(len(test)):
        try:
            mdl = ARIMA(history, order=arima_order)
            fitted_mdl = mdl.fit()
            yhat = fitted_mdl.forecast()[0]
            predictions.append(yhat)
            history.append(test[t])
        except:
            continue
    # calculate out of sample error
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse

# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(data, p_values, d_values, q_values):
    data = data.astype('float32')
    best_score, best_pdq = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
```

```
        try:
            rmse = evaluate_arima_model(data, order)
            if rmse < best_score:
                best_score, best_pdq = rmse, order
            print('ARIMA%s RMSE=%.3f' % (order,rmse))
        except:
            continue
    print('Best ARIMA%s RMSE=%.3f' % (best_pdq, best_score))

X = ts.values
p_values = [0, 1, 2, 4, 5, 6, 7, 8]
d_values = range(1, 4)
q_values = range(1, 4)
evaluate_models(X, p_values, d_values, q_values)

mdl1 = ARIMA(train, order=(8,1,2))
fitted_mdl1 = mdl1.fit()

# Test set predictions using model I
prediction1 = fitted_mdl1.forecast(steps=27)

plt.figure(figsize=(12,10))
plt.plot(train,label="Training")
plt.plot(test,label="Test")
plt.plot(prediction1,label="Predicted")
plt.legend()
plt.show()

# Model evaluation of model I and prediction for out of sample data
print(f'R2_score: {(r2_score(test.Sales, prediction1))}')
print(f'RMSE: {(np.sqrt(mean_squared_error(test.Sales, prediction1)))}')

# ### Approach I: ARIMA model and optimization of p, d, q parameters using auto_arima() function
from pmdarima library


# Modeling approach II: Optimizing parameters using Auto Arima lib
fitted_mdl2 =  auto_arima(train,start_p=0, d=1, start_q=0,
                          max_p=8, max_d=5, max_q=5, start_P=0,
                           D=1, start_Q=0, max_P=8, max_D=5,
                           max_Q=5, m=12, seasonal=True,
                           error_action='warn',
                           supress_warnings=True,stepwise = True,
                           random_state=20,n_fits = 50 )

# Test set predictions using model II
```

```
prediction1 = fitted_mdl2.predict(n_periods=27)
plt.clf()
plt.figure(figsize=(12,10))
plt.plot(train,label="Training")
plt.plot(test,label="Test")
plt.plot(prediction2,label="Predicted")
plt.legend()
plt.show()

# Model evaluation of model I and prediction for out of sample data
print(f'R2_score: {(r2_score(test.Sales, prediction2))}')
print(f'RMSE: {(np.sqrt(mean_squared_error(test.Sales, prediction2)))}')

# **We can see that the model II is a better fit both in terms of r2_score and rmse. So, lets train the
model II on the entire dataset and predict for out of sample (1969)**

# Prediction for next 12 months using model II
#Training on the entire dataset and predicting for 1969
final_mdl =  auto_arima(X, start_p=0, d=1, start_q=0,
                        max_p=8, max_d=5, max_q=5, start_P=0,
                        D=1, start_Q=0, max_P=8, max_D=5,
                        max_Q=5, m=12, seasonal=True,
                        error_action='warn',
                        supress_warnings=True,stepwise = True,
                        random_state=20,n_fits=50)

# Model prediction using model II AND prediction for out of sample data
prediction = final_mdl.predict(n_periods = 12)
idx = pd.date_range(start='1/1/1969', end='12/1/1969', freq='MS')
sales_1969 = pd.DataFrame({'Sales':prediction}, index=idx)
sales_1969.index.name = 'Month'

# Plotting historical sales till 1968 and predicted sales for 1969
plt.figure(figsize=(12,10))
plt.plot(X, label='Historical sales')
plt.plot(sales_1969, label='Prediction')
plt.legend()
plt.show()

print(Predicted sales for the year of 1969: \n\n {sales_1969}')
```

***Predicted sales for the year of 1969:***

```
            Sales
Month
1969-01-01  14284.0
```

***1969-02-01  14639.0***
***1969-03-01  21970.0***
***1969-04-01  22366.0***
***1969-05-01  26746.0***
***1969-06-01  22991.0***
***1969-07-01  17933.0***
***1969-08-01  16844.0***
***1969-09-01  15541.0***
***1969-10-01  21105.0***
***1969-11-01  18222.0***
***1969-12-01  15701.0***

## Appendix

**Question 1 query results**

| product_category | product |
| --- | --- |
| Active | A7 |
| Active | A4 |
| Active | A22 |
| Active | A25 |
| Active | A18 |
| Bottom | B23 |
| Bottom | B8 |
| Bottom | B10 |
| Bottom | B12 |
| Bottom | B21 |
| Dress | D9 |
| Dress | D16 |
| Dress | D25 |
| Dress | D7 |
| Dress | D18 |
| Full Skirt | F16 |
| Full Skirt | F2 |
| Full Skirt | F6 |
| Full Skirt | F11 |
| Full Skirt | F12 |
| Gown | G14 |
| Gown | G15 |
| Gown | G13 |
| Gown | G4 |
| Gown | G23 |
| Hourglass | H25 |
| Hourglass | H15 |

| product_category | product |
| --- | --- |
| Active | A7 |
| Active | A4 |
| Active | A22 |
| Active | A25 |
| Active | A18 |
| Bottom | B23 |
| Bottom | B8 |
| Bottom | B10 |
| Bottom | B12 |
| Hourglass | H23 |
| Hourglass | H18 |
| Hourglass | H17 |
| Jumpsuit | J25 |
| Jumpsuit | J13 |
| Jumpsuit | J1 |
| Jumpsuit | J20 |
| Jumpsuit | J14 |
| Kids Apparel | K5 |
| Kids Apparel | K19 |
| Kids Apparel | K6 |
| Kids Apparel | K14 |
| Kids Apparel | K25 |
| Mini skirt | M14 |
| Mini skirt | M17 |
| Mini skirt | M25 |
| Mini skirt | M1 |
| Mini skirt | M21 |

| product_category | product |
|---|---|
| Active | A7 |
| Active | A4 |
| Active | A22 |
| Active | A25 |
| Active | A18 |
| Bottom | B23 |
| Bottom | B8 |
| Bottom | B10 |
| Bottom | B12 |
| Outerwear/Jacket | O5 |
| Outerwear/Jacket | O18 |
| Outerwear/Jacket | O12 |
| Outerwear/Jacket | O15 |
| Outerwear/Jacket | O21 |
| Sweater/Knit | S20 |
| Sweater/Knit | S16 |
| Sweater/Knit | S19 |
| Sweater/Knit | S3 |
| Sweater/Knit | S8 |
| Top | T12 |
| Top | T18 |
| Top | T6 |
| Top | T24 |
| Top | T1 |
| Vest | V20 |
| Vest | V24 |
| Vest | V10 |

| product_category | product |
|---|---|
| Active | A7 |
| Active | A4 |
| Active | A22 |
| Active | A25 |
| Active | A18 |
| Bottom | B23 |
| Bottom | B8 |
| Bottom | B10 |
| Bottom | B12 |
| Vest | V23 |
| Vest | V17 |