Portfolio Part 4

Data set from: https://opendata.transport.nsw.gov.au/dataset/opal-trips-bus/resource/9862b4f1-37d9-495a-97b6-0c867fa91d83#%7B%7D

Description: Monthly Opal bus trips by contract area, month and card type, July 2016 to October 2021.

Questions / Challenges:

1. Are there seasonal changes between trends
2. View which Contract_Region / Trips that has the highest movement
3. What are the distribution rate of card types in NSW
4. Try different algoritm(s) to generate a model that is appropriate prediction of the chosen label

0. Import the Libraries

In [ ]:
```
#import Libraries First
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
color_pal = sns.color_palette()
%matplotlib inline
from sklearn.preprocessing import OrdinalEncoder
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import Normalizer
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import confusion_matrix, accuracy_score, r2_score
from sklearn.metrics import roc_curve, auc
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')
```

Exploratory Data Analysis

1.1 Load the dataset, check for null values, check the dimension of the dataframe.

Read the CSV and outout the intial 5 rows. As observed in our dataframe, the features are encoded in month-year format and the values in them are the number of trips.

In [ ]:

```
df = pd.read_csv('data/Bus_Card_Type.csv')
df.head(5)
```

Out[ ]:

	Contract_Type	Tap_Class	Jul-16	Aug-16	Sep-16	Oct-16	Nov-16	Dec-16	Jan-17	Feb-17	...	Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug
0	Sydney Metro Bus Contract 1	Adult	390,433	386,386	370,068	381,402	405,245	353,196	341,332	386,683	...	230,528	265,311	294,367	271,849	289,229	259,924	127,354	102,
1	Sydney Metro Bus Contract 1	Adult Single Bus Trip 1	NaN	25,753	23,416	23,138	23,352	22,065	21,306	21,277	...	3	3	10	4	8	8	7	
2	Sydney Metro Bus Contract 1	Adult Single Bus Trip 2	NaN	17,882	14,984	14,465	13,914	13,285	12,021	11,824	...	NaN	NaN	1	7	2	4	1	
3	Sydney Metro Bus Contract 1	Adult Single Bus Trip 3	NaN	1,880	1,459	1,355	1,198	1,102	1,098	1,009	...	NaN	3	NaN	NaN	NaN	4	NaN	
4	Sydney Metro Bus Contract 1	Child/Youth	62,800	67,010	70,022	78,951	76,328	77,290	80,779	80,920	...	46,574	45,469	42,069	52,658	42,457	34,715	11,205	6,

5 rows × 66 columns

Using tail, we noticed that there is a grand total row for each month-year. We can remove that to further clean our dataframe.

In [ ]:

```
df.tail(3)
```

Out[ ]:

	Contract_Type	Tap_Class	Jul-16	Aug-16	Sep-16	Oct-16	Nov-16	Dec-16	Jan-17	Feb-17	...	Jan-21	Feb-21	Mar-21	Apr-
610	Outer Sydney Metro Bus Contract 5	School Student	45,224	110,239	75,262	71,065	92,417	33,800	5,512	91,284	...	NaN	NaN	NaN	N
611	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	124,570	139,526	137,657	134,642	143,341	135,737	127,356	131,825	...	NaN	NaN	NaN	N
612	Grand Total	Total	21,198,509	25,210,772	23,011,540	23,095,302	24,422,383	20,880,865	19,550,895	22,534,683	...	11,567,325	14,964,876	17,032,074	15,471,

3 rows × 66 columns

Using the drop method, we can remove the last row in our dataframe.

In [ ]:

```
df = df.drop(index=df.index[-1], axis=0)
df.tail(3)
```

Out[ ]:

	Contract_Type	Tap_Class	Jul-16	Aug-16	Sep-16	Oct-16	Nov-16	Dec-16	Jan-17	Feb-17	...	Jan-21	Feb-21	Mar-21	Apr-21	May-21	Jun-21	Jul-21	Aug-21	Sep-21	Oct-21
609	Outer Sydney Metro Bus Contract 5	Free Travel	957	1,162	1,061	1,063	1,215	1,388	1,309	1,127	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
610	Outer Sydney Metro Bus Contract 5	School Student	45,224	110,239	75,262	71,065	92,417	33,800	5,512	91,284	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
611	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	124,570	139,526	137,657	134,642	143,341	135,737	127,356	131,825	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3 rows × 66 columns

The cell below will check to see the null values in our dataframe.

In [ ]:

```
busCardDF = df.copy()
busCardDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 612 entries, 0 to 611
Data columns (total 66 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Contract_Type  612 non-null    object
 1   Tap_Class      612 non-null    object
 2   Jul-16         227 non-null    object
 3   Aug-16         456 non-null    object
 4   Sep-16         451 non-null    object
 5   Oct-16         454 non-null    object
 6   Nov-16         448 non-null    object
 7   Dec-16         454 non-null    object
 8   Jan-17         450 non-null    object
 9   Feb-17         454 non-null    object
 10  Mar-17         443 non-null    object
 11  Apr-17         446 non-null    object
 12  May-17         444 non-null    object
 13  Jun-17         442 non-null    object
 14  Jul-17         453 non-null    object
 15  Aug-17         447 non-null    object
 16  Sep-17         462 non-null    object
 17  Oct-17         467 non-null    object
 18  Nov-17         440 non-null    object
 19  Dec-17         440 non-null    object
 20  Jan-18         450 non-null    object
 21  Feb-18         442 non-null    object
 22  Mar-18         342 non-null    object
 23  Apr-18         349 non-null    object
 24  May-18         353 non-null    object
 25  Jun-18         352 non-null    object
 26  Jul-18         369 non-null    object
 27  Aug-18         373 non-null    object
 28  Sep-18         369 non-null    object
 29  Oct-18         374 non-null    object
 30  Nov-18         378 non-null    object
 31  Dec-18         375 non-null    object
 32  Jan-19         379 non-null    object
 33  Feb-19         379 non-null    object
 34  Mar-19         381 non-null    object
 35  Apr-19         382 non-null    object
 36  May-19         370 non-null    object
 37  Jun-19         370 non-null    object
```

```
 38  Jul-19         370 non-null    object
 39  Aug-19         380 non-null    object
 40  Sep-19         392 non-null    object
 41  Oct-19         393 non-null    object
 42  Nov-19         378 non-null    object
 43  Dec-19         392 non-null    object
 44  Jan-20         386 non-null    object
 45  Feb-20         386 non-null    object
 46  Mar-20         385 non-null    object
 47  Apr-20         299 non-null    object
 48  May-20         296 non-null    object
 49  Jun-20         302 non-null    object
 50  Jul-20         295 non-null    object
 51  Aug-20         290 non-null    object
 52  Sep-20         281 non-null    object
 53  Oct-20         291 non-null    object
 54  Nov-20         286 non-null    object
 55  Dec-20         277 non-null    object
 56  Jan-21         274 non-null    object
 57  Feb-21         275 non-null    object
 58  Mar-21         286 non-null    object
 59  Apr-21         291 non-null    object
 60  May-21         278 non-null    object
 61  Jun-21         274 non-null    object
 62  Jul-21         274 non-null    object
 63  Aug-21         271 non-null    object
 64  Sep-21         259 non-null    object
 65  Oct-21         292 non-null    object
dtypes: object(66)
memory usage: 315.7+ KB
```

Check the nulls in our dataframe.

In [ ]:
```
busCardDF.isnull().sum()
```

Out[ ]:
```
Contract_Type      0
Tap_Class          0
Jul-16           385
Aug-16           156
Sep-16           161
                ...
Jun-21           338
Jul-21           338
Aug-21           341
Sep-21           353
Oct-21           320
Length: 66, dtype: int64
```

Check the total records that has nulls value

In [ ]:
```
busCardDF.isnull().sum().sum()
```

Out[ ]:
```
15650
```

There are 15,650 null values in our dataframe. If we remove them per row, some of the values that has initial filled in records in some features will be completely gone. So here, we could probably modify our dataframe and instead of making features with the month-year, we'll make it as a value instead. So the dataframe would be like these:

a. Contract_Type

b. Tap_Class

c. Date *(In year month format)*

d. Taps_Count

In [ ]:
```
tempBusCardDF = busCardDF.copy()
```

We would have the data wrangling method here where we will transform our dataframe into more readable and easier to understand to our algorithm, and for us.

In [ ]:
```
import datetime

wrangDF = pd.DataFrame(columns=['Contract_Type', 'Tap_Class', 'Year', 'Month', 'Day', 'Taps_Count'])
for index, row in tempBusCardDF.iterrows():
    colCount = 0
    columnNames = []
    for column in tempBusCardDF.loc[[index]]:
        # print(tempBusCardDF.loc[[index]][column].values)
        colCount+=1
```

```
        columnNames.append(tempBusCardDF.loc[[index]][column].values)
        if colCount > 2:
            _date = column.split("-")
            _month = _date[0]

            ##Convert the month to number
            month_name = _month
            datetime_object = datetime.datetime.strptime(month_name, "%b")
            _month = datetime_object.month

            _year = int(_date[1]) + 2000

            taps = tempBusCardDF.loc[[index]][column].values[0]
            if  isinstance(taps, float):
                taps = 0
                pass
            else:
                taps = taps.replace(',','')
                taps = int(taps)
                newRow = {'Contract_Type' : columnNames[0][0], 'Tap_Class':columnNames[1][0], 'Year': _year, 'Month':  _month, 'Day': 1, 'Taps_Cou
                wrangDF = wrangDF.append(newRow, ignore_index=True)
```

Below is our new dataframe, we'll use this dataframe until the end of this pipeline.

In [ ]:
```
wrangDF.tail(5)
```

Out[ ]:

	Contract_Type	Tap_Class	Year	Month	Day	Taps_Count
23513	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	2017	3	1	149897
23514	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	2017	4	1	125352
23515	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	2017	5	1	144478
23516	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	2017	6	1	130625
23517	Outer Sydney Metro Bus Contract 5	Senior/Pensioner	2017	7	1	11

So far the converted data looks good. Let's try to export the csv for future use.

In [ ]:
```
wrangDF.to_csv("data/Bus_Card_Type_Converted.csv")
```

As we created a new dataframe, it is important to check the data-type of our columns.

In [ ]:
```
wrangDF.dtypes
```

Out[ ]:
```
Contract_Type    object
Tap_Class        object
Year             object
Month            object
Day              object
Taps_Count       object
dtype: object
```

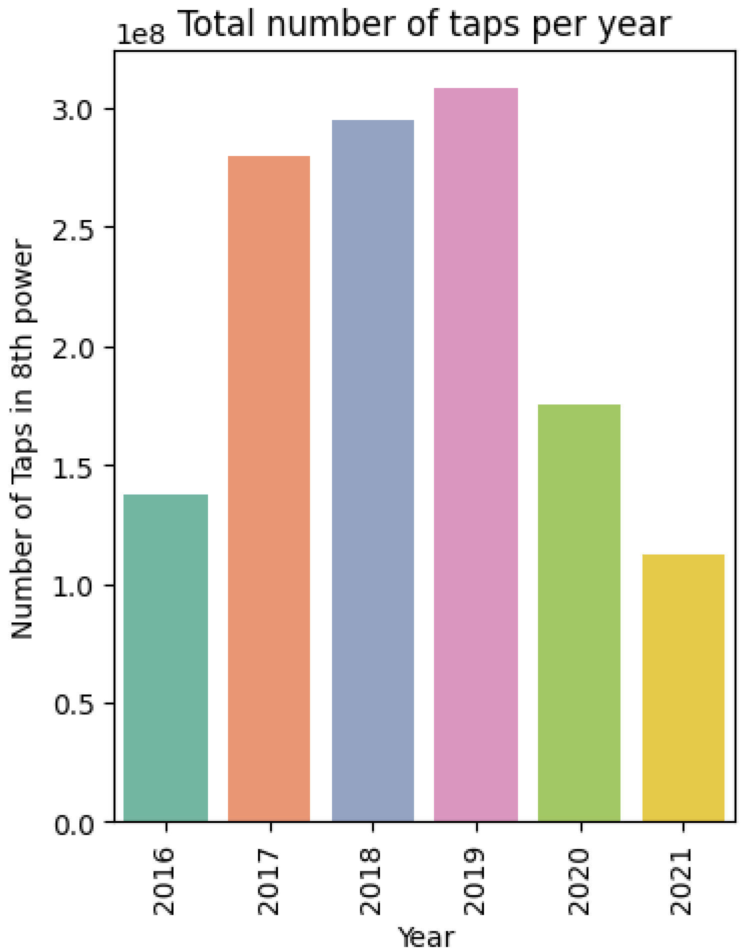
1.2 Generate graphs to quickly compare against the taps per year, months, class and contracts.

In [ ]:
```
#Converte the types of each column
wrangDF.Taps_Count = wrangDF.Taps_Count.astype('int64')
wrangDF.Year = wrangDF.Year.astype('int64')
wrangDF.Month = wrangDF.Month.astype('int64')
wrangDF.Day = wrangDF.Day.astype('int64')
```

As seen below, there is a significant growth going from 2016 - 2019 but it drops when 2020 came. The reason for such drop is because of the pandemic where NSW officials imposed a lockdown. Also to note, we only have data from July 2016, and that is the reason that we only have atleast 3/5 of data against 2017 when comparing it to 2016.

In [ ]:
```
yearDistribution = wrangDF.groupby(['Year'], as_index=False).sum()

sns.set_palette("Set2")
plt.figure(figsize=(4,5))
plt.xticks(rotation=90)
sns.barplot(data=yearDistribution, x="Year", y="Taps_Count")
plt.title('Total number of taps per year')
plt.xlabel('Year')
plt.ylabel('Number of Taps in 8th power')
```

Out[ ]:
```
Text(0, 0.5, 'Number of Taps in 8th power')
```

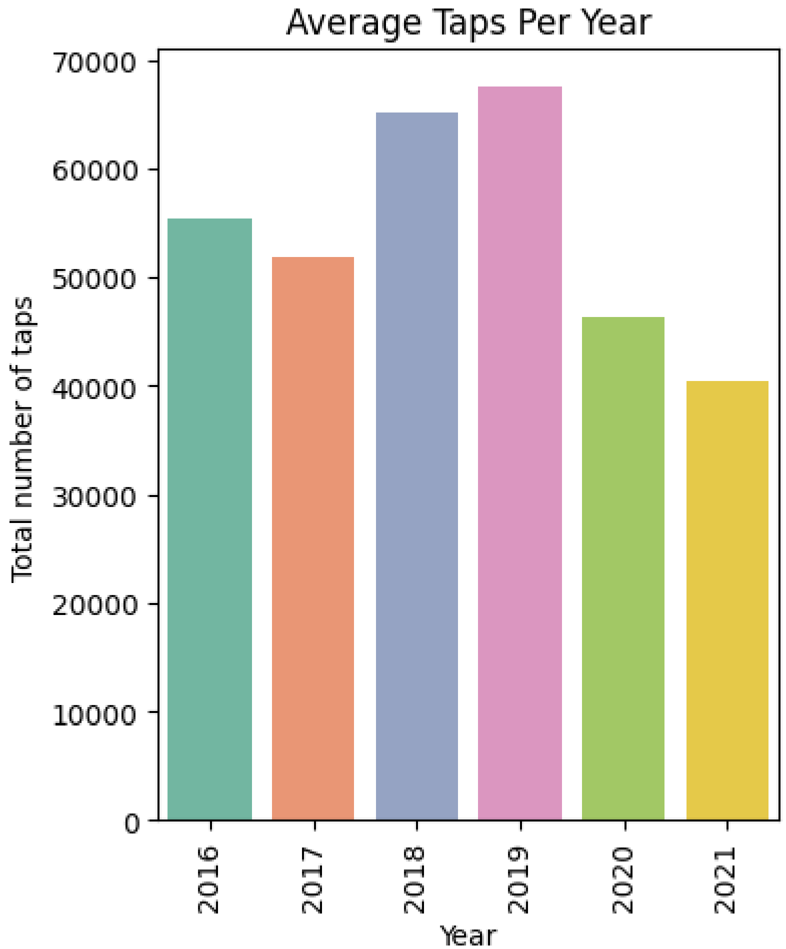


Check the average taps below, there is a dip of movement in 2017. The reason for this is some of the trips got lesser movement or taps in the same month on 2016, and some of the lines stopped their operation.

In [ ]:

```
yearDistribution = wrangDF.groupby(['Year'], as_index=False).mean()

sns.set_palette("Set2")
plt.figure(figsize=(4,5))
plt.xticks(rotation=90)
sns.barplot(data=yearDistribution, x="Year", y="Taps_Count")
plt.title('Average Taps Per Year')
plt.xlabel('Year')
plt.ylabel('Total number of taps')
```

```
Out[ ]: Text(0, 0.5, 'Total number of taps')
```



Exluding 2020 and 2021 we wish to see the pre-pandemic movement of the city.

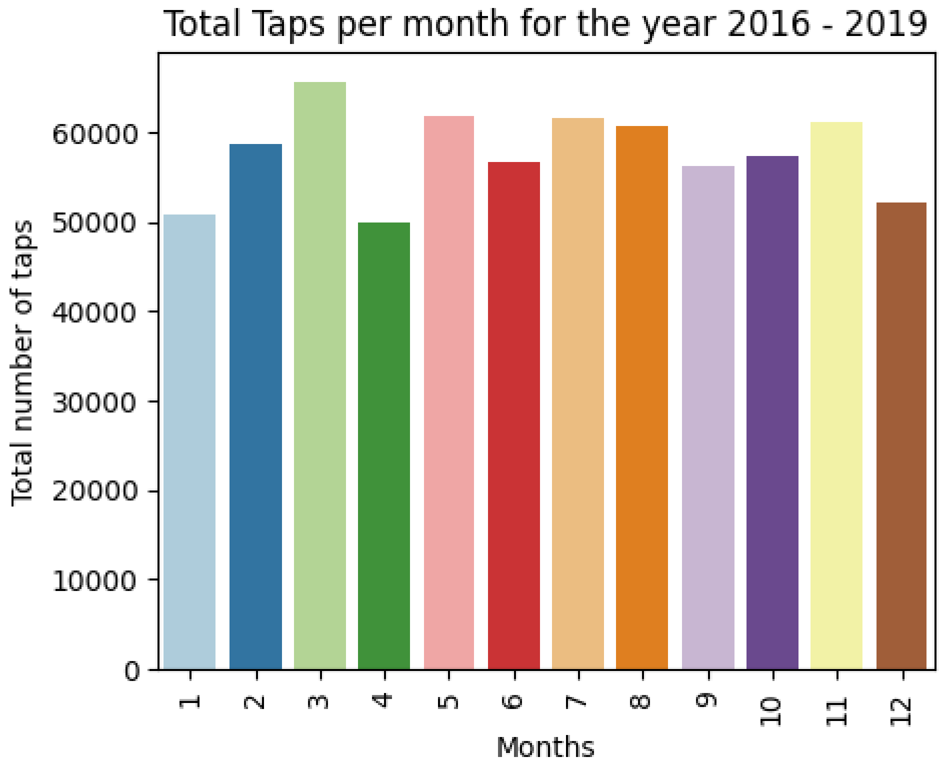
As seen, the month of March got the highest movement. This can be attributed to the several festivals happening in the city.

The low activity in January, April and December can be attributed to the session break of the universities, the holidays in public schools.

```
In [ ]: monthDistribution = wrangDF[wrangDF["Year"] < 2021].groupby('Month', as_index=False).mean()

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
```

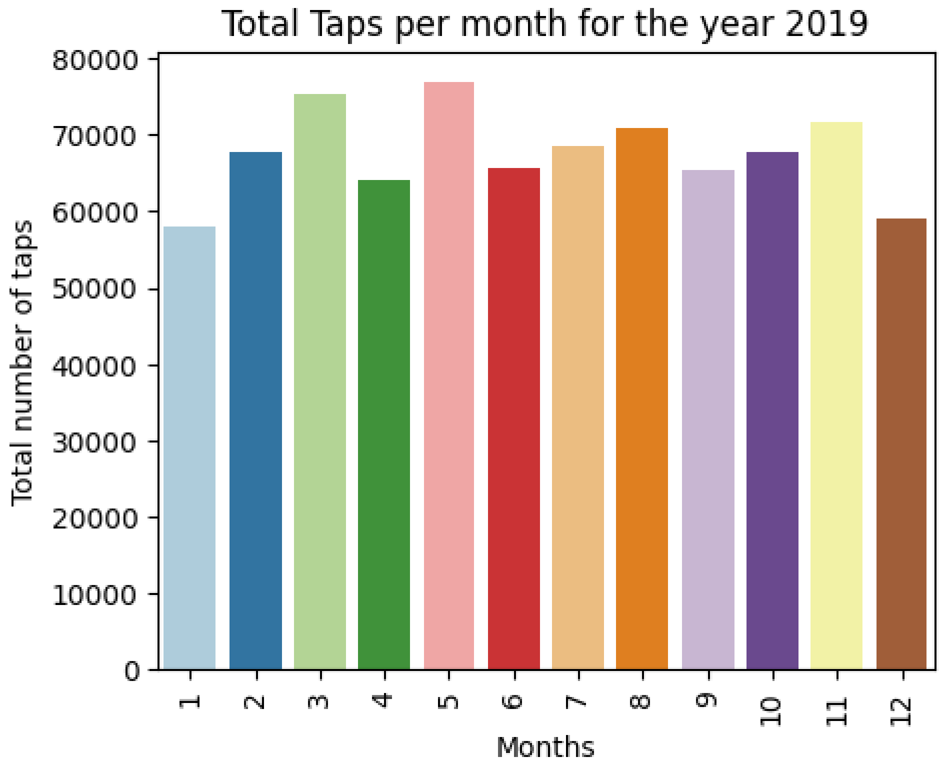
```python
sns.barplot(data=monthDistribution, x="Month", y="Taps_Count")
plt.title('Total Taps per month for the year 2016 - 2019')
plt.xlabel('Months')
plt.ylabel('Total number of taps')
```

Out[ ]: Text(0, 0.5, 'Total number of taps')



In [ ]:
```python
monthDistribution2019 = wrangDF[wrangDF["Year"] == 2019].groupby('Month', as_index=False).mean()

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=monthDistribution2019, x="Month", y="Taps_Count")
plt.title('Total Taps per month for the year 2019')
plt.xlabel('Months')
plt.ylabel('Total number of taps')
```

Out[ ]: Text(0, 0.5, 'Total number of taps')

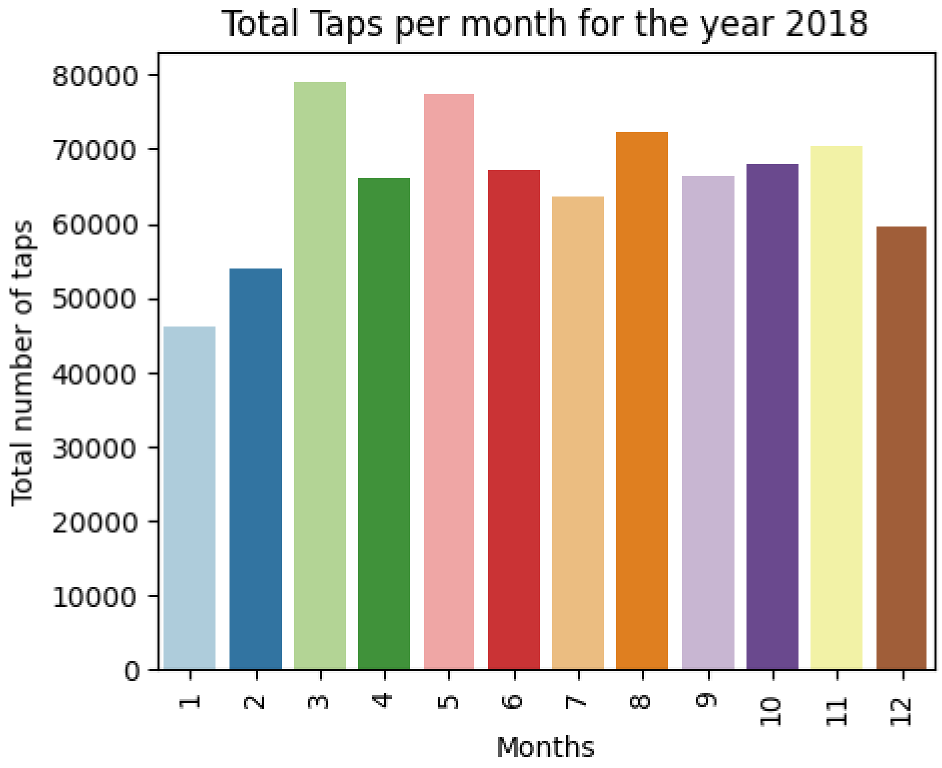


In [ ]:

```
monthDistribution2018 = wrangDF[wrangDF["Year"] == 2018].groupby('Month', as_index=False).mean()

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=monthDistribution2018, x="Month", y="Taps_Count")
plt.title('Total Taps per month for the year 2018')
plt.xlabel('Months')
plt.ylabel('Total number of taps')
```

Out[ ]:

```
Text(0, 0.5, 'Total number of taps')
```



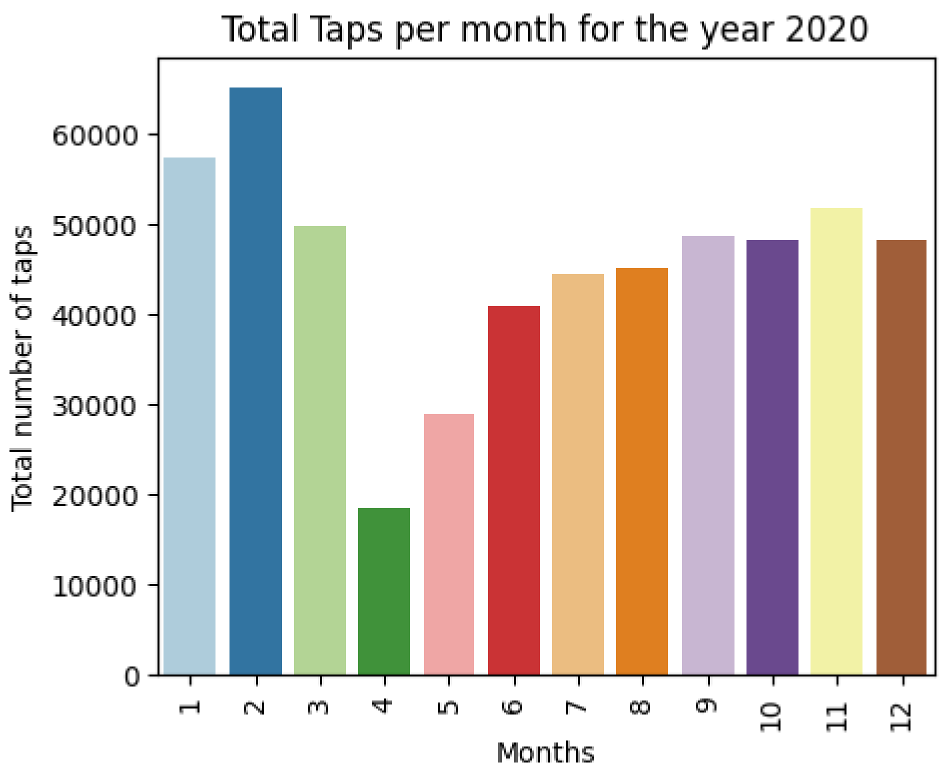
Noticed that in the graph below, there was a big hit in number of transactions / tap in April. The reason could be of the lock down imposed by the NSW government.

In [ ]:

```
monthDistribution2020 = wrangDF[wrangDF["Year"] == 2020].groupby('Month', as_index=False).mean()

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=monthDistribution2020, x="Month", y="Taps_Count")
plt.title('Total Taps per month for the year 2020')
plt.xlabel('Months')
plt.ylabel('Total number of taps')
```

Out[ ]:

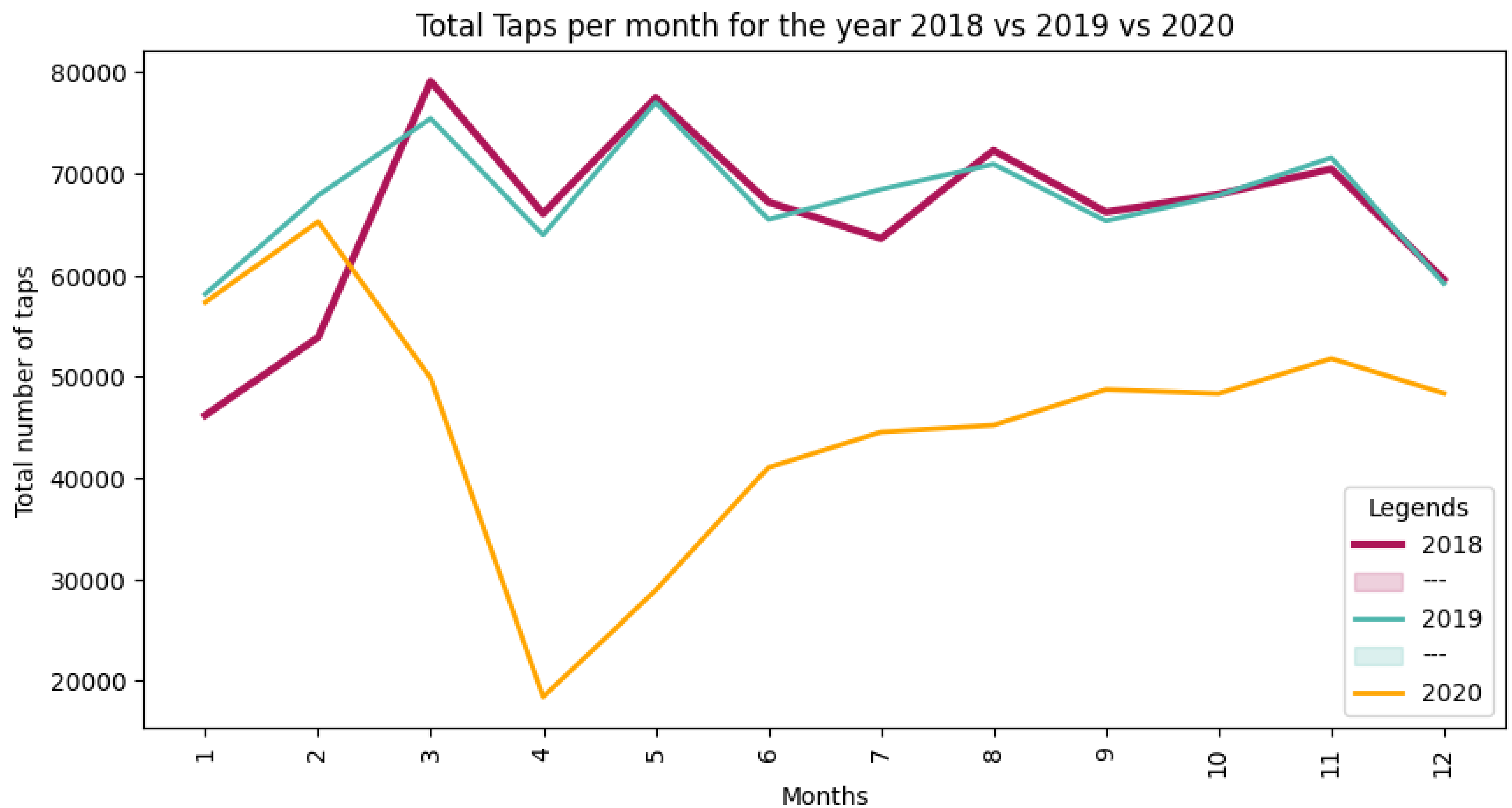
```
Text(0, 0.5, 'Total number of taps')
```



To see the trendline, we'll use the lineplot from sns. See the figure below for comparison. 2018 and 2019 almost has the same trend line. 2020 got the big hit.

In [ ]:
```
sns.set_palette("Paired")
plt.figure(figsize=(10,5))
plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12], rotation=90)
sns.lineplot(data=monthDistribution2018, x="Month", y="Taps_Count", linewidth=3, color="#AD1457")
sns.lineplot(data=monthDistribution2019, x="Month", y="Taps_Count", linewidth=2, color="#4DB6AC")
sns.lineplot(data=monthDistribution2020, x="Month", y="Taps_Count", linewidth=2, color="orange")
plt.title('Total Taps per month for the year 2018 vs 2019 vs 2020')
plt.legend(title='Legends', loc='lower right', labels=['2018','---', '2019', '---', '2020'])
plt.xlabel('Months')
plt.ylabel('Total number of taps')
```

Out[ ]:
```
Text(0, 0.5, 'Total number of taps')
```

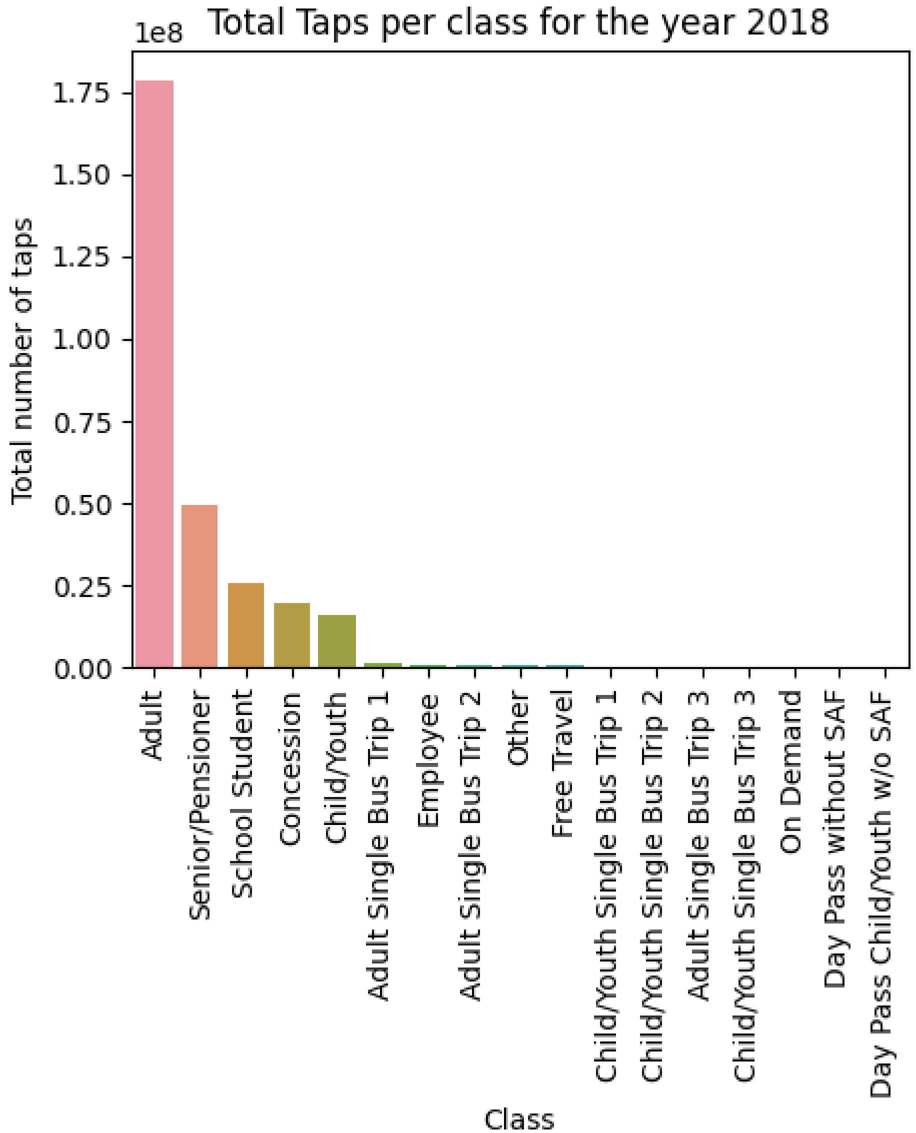


Based on the statistics, https://www.abs.gov.au/articles/snapshot-nsw-2021 , there are 20.4 million living in NSW right now. 9.4M of which are below 18. Suffice to say, the graph below looks relatively right.

In [ ]:

```
tapClass2018 = wrangDF[wrangDF['Year'] == 2018].groupby(['Tap_Class'], as_index=False).sum()
tapClass2018 = tapClass2018.sort_values('Taps_Count', ascending=False)

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=tapClass2018, x="Tap_Class", y="Taps_Count")
plt.title('Total Taps per class for the year 2018')
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

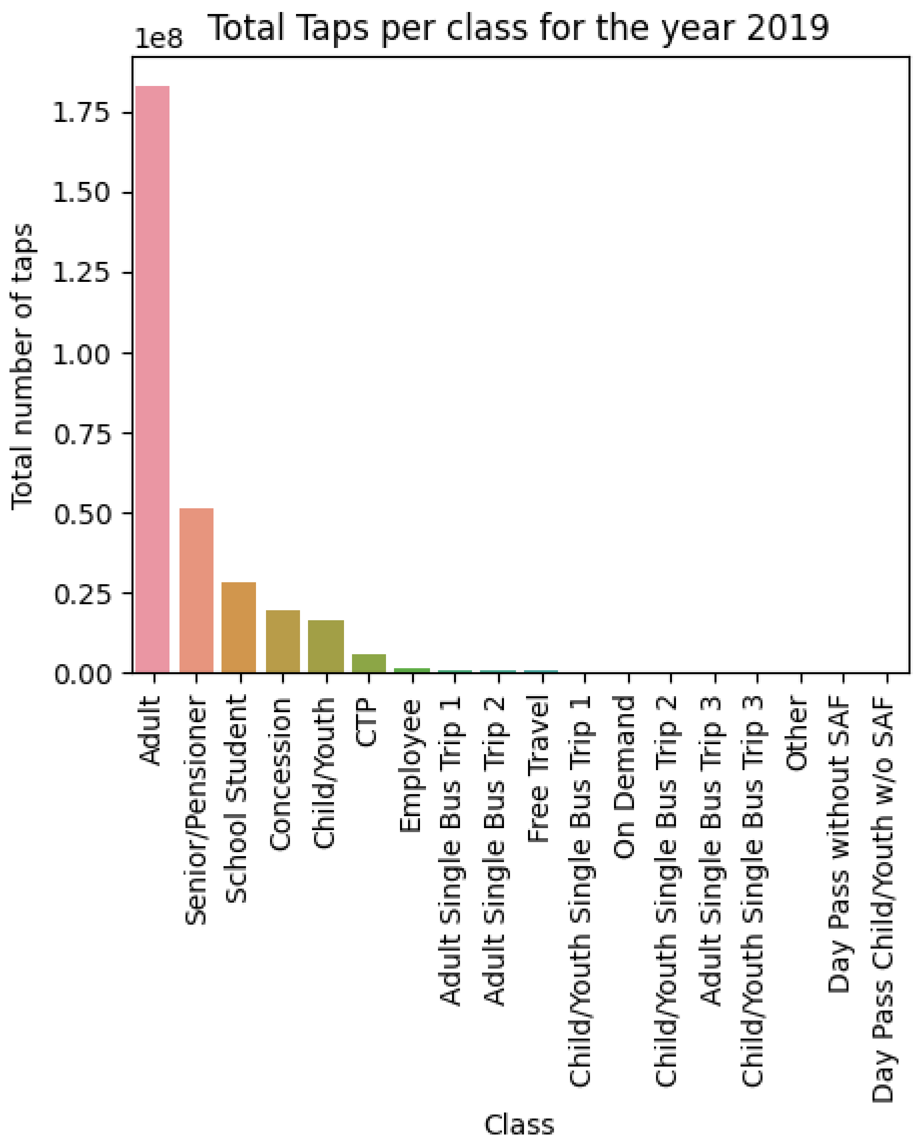
Out[ ]: Text(0, 0.5, 'Total number of taps')



In [ ]:
```
tapClass2019 = wrangDF[wrangDF['Year'] == 2019].groupby(['Tap_Class'], as_index=False).sum().sort_values('Taps_Count', ascending=False)

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=tapClass2019, x="Tap_Class", y="Taps_Count")
```

```
plt.title('Total Taps per class for the year 2019')
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

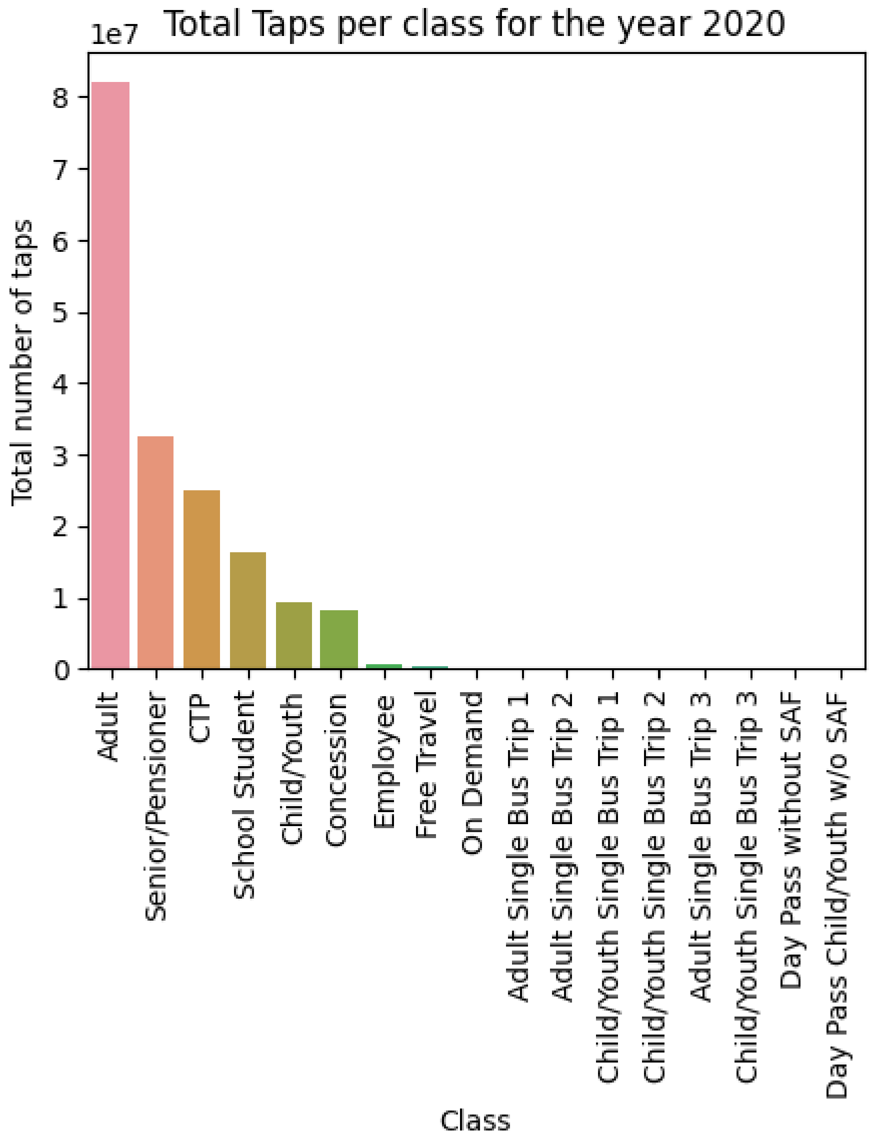
Out[ ]: Text(0, 0.5, 'Total number of taps')



There is an increase of movement for the CTP in the year 2020

In [ ]:
```
tapClass2020 = wrangDF[wrangDF['Year'] == 2020].groupby(['Tap_Class'], as_index=False).sum().sort_values('Taps_Count', ascending=False)

sns.set_palette("Paired")
plt.figure(figsize=(5,4))
plt.xticks(rotation=90)
sns.barplot(data=tapClass2020, x="Tap_Class", y="Taps_Count")
plt.title('Total Taps per class for the year 2020')
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

Out[ ]:
```
Text(0, 0.5, 'Total number of taps')
```



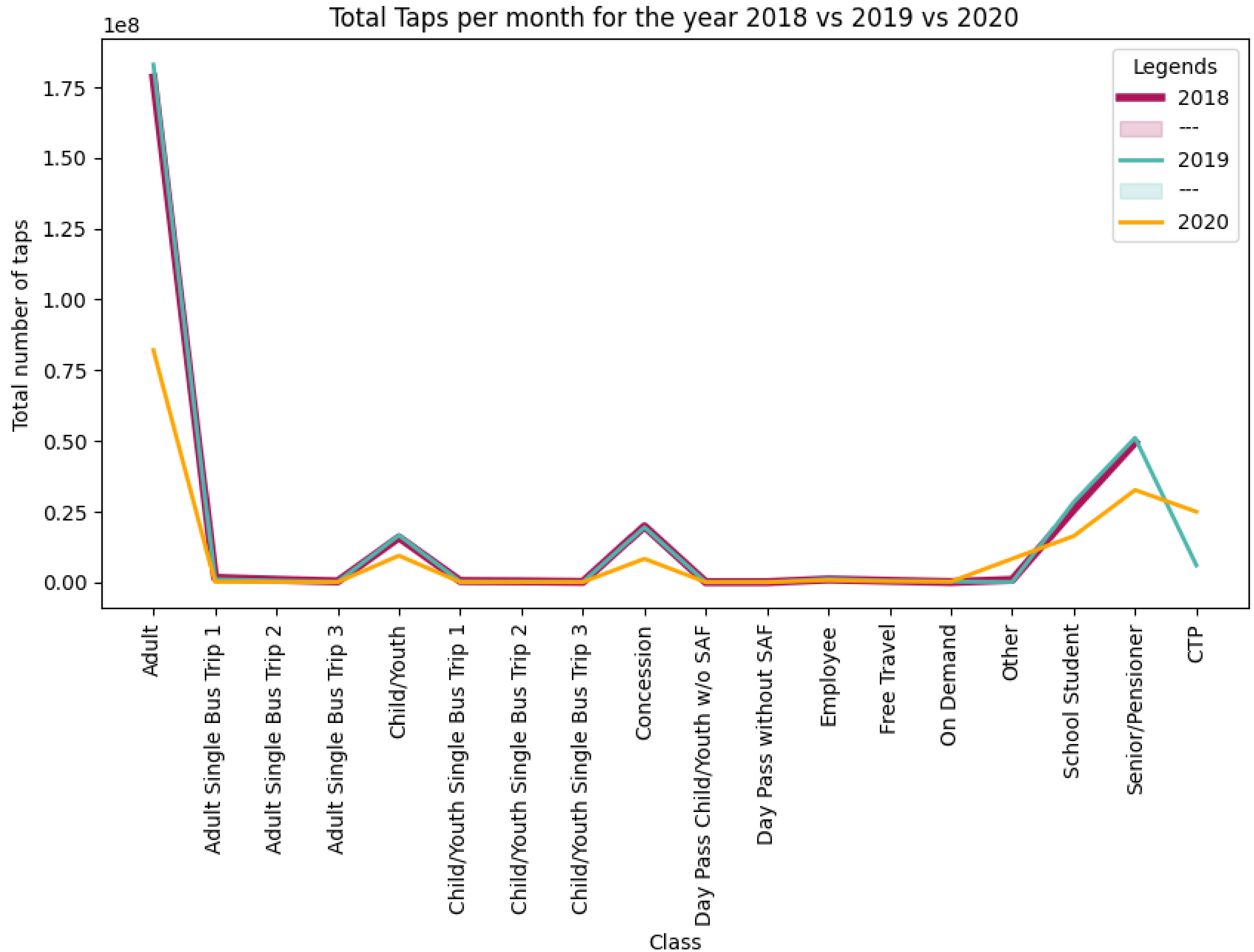
Looking in our graph below, we can say the same that there was a mejor decline in transaction during the 2020.

In [ ]:

```
tapClass2018 = tapClass2018.sort_values('Tap_Class')
tapClass2019 = tapClass2019.sort_values('Tap_Class')
tapClass2020 = tapClass2020.sort_values('Tap_Class')

sns.set_palette("Paired")
```

```python
plt.figure(figsize=(10,5))
plt.xticks(rotation=90)
sns.lineplot(data=tapClass2018, x="Tap_Class", y="Taps_Count", linewidth=4, color="#AD1457")
sns.lineplot(data=tapClass2019, x="Tap_Class", y="Taps_Count", linewidth=2, color="#4DB6AC")
sns.lineplot(data=tapClass2020, x="Tap_Class", y="Taps_Count", linewidth=2, color="orange")
plt.title('Total Taps per month for the year 2018 vs 2019 vs 2020')
plt.legend(title='Legends', loc='upper right', labels=['2018','---', '2019', '---', '2020'])
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

Out[ ]:
```
Text(0, 0.5, 'Total number of taps')
```

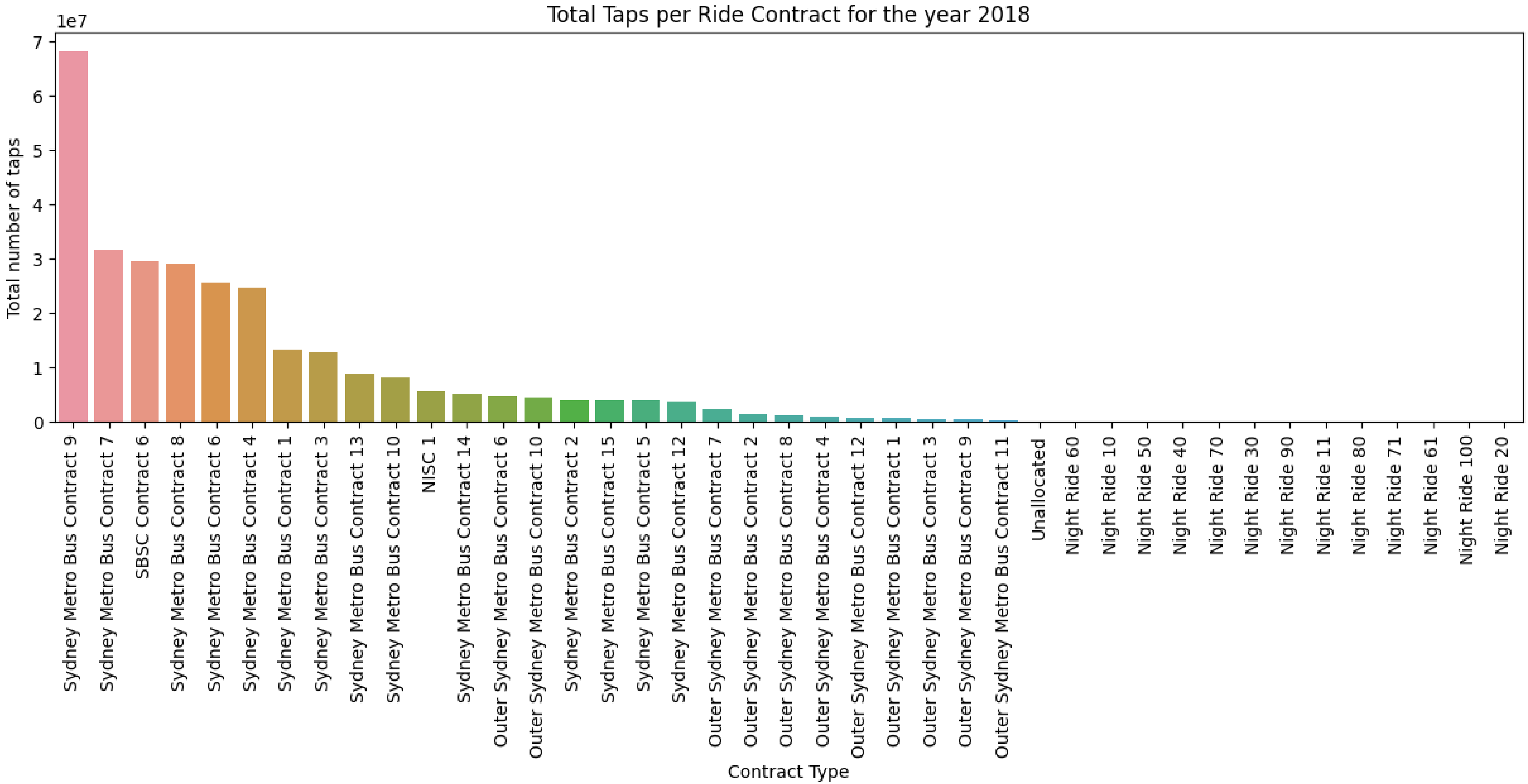


Sydney Metro Bus was the popular choice for mode of transportation. It just make sense as it connects to almost major sub urb in NSW.

In [ ]:
```
contract2018 = wrangDF[wrangDF["Year"] == 2018].groupby(['Contract_Type'], as_index=False).sum().sort_values('Taps_Count', ascending=False)
```

```
sns.set_palette("Paired")
plt.figure(figsize=(15,4))
plt.xticks(rotation=90)
sns.barplot(data=contract2018, x="Contract_Type", y="Taps_Count")
plt.title('Total Taps per Ride Contract for the year 2018')
plt.xlabel('Contract Type')
plt.ylabel('Total number of taps')
```

Out[ ]: Text(0, 0.5, 'Total number of taps')

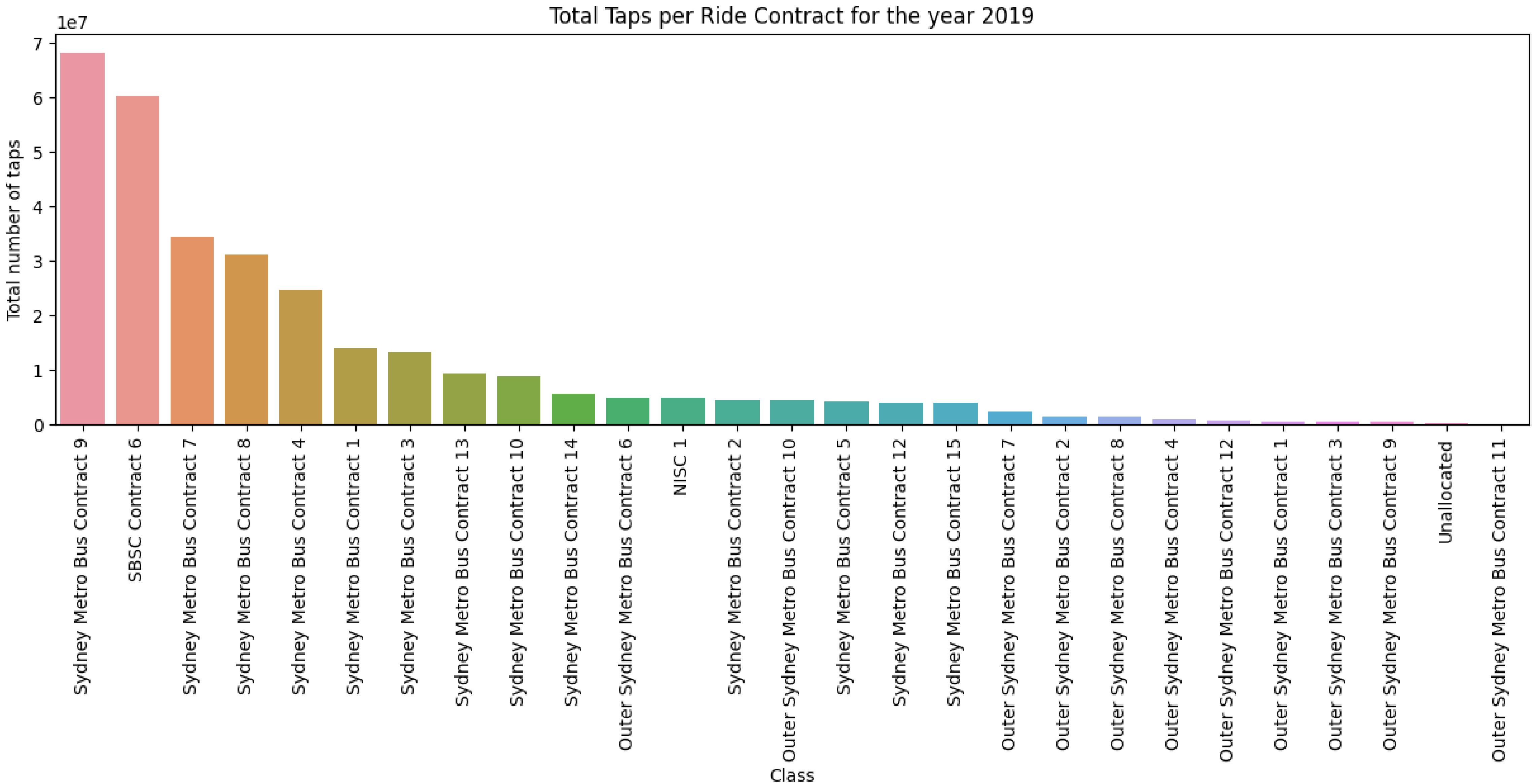


In [ ]:

```
contract2019 = wrangDF[wrangDF["Year"] == 2019].groupby(['Contract_Type'], as_index=False).sum().sort_values('Taps_Count', ascending=False)

sns.set_palette("Paired")
plt.figure(figsize=(15,4))
plt.xticks(rotation=90)
sns.barplot(data=contract2019, x="Contract_Type", y="Taps_Count")
plt.title('Total Taps per Ride Contract for the year 2019')
```

```python
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

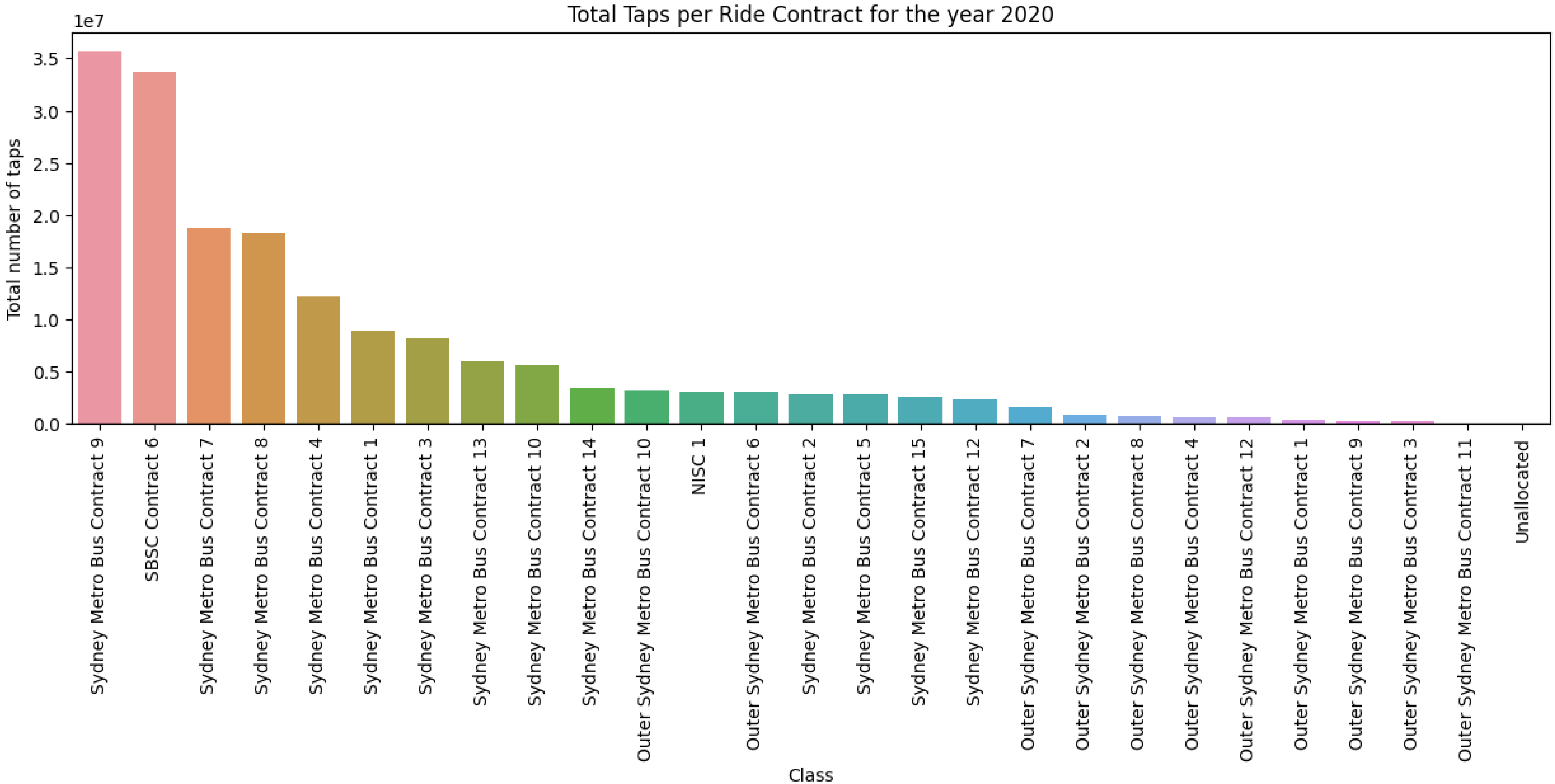
Out[ ]: Text(0, 0.5, 'Total number of taps')



In [ ]:
```python
contract2020 = wrangDF[wrangDF["Year"] == 2020].groupby(['Contract_Type'], as_index=False).sum().sort_values('Taps_Count', ascending=False)
```

```python
sns.set_palette("Paired")
plt.figure(figsize=(15,4))
plt.xticks(rotation=90)
sns.barplot(data=contract2020, x="Contract_Type", y="Taps_Count")
plt.title('Total Taps per Ride Contract for the year 2020')
plt.xlabel('Class')
plt.ylabel('Total number of taps')
```

Out[ ]: Text(0, 0.5, 'Total number of taps')

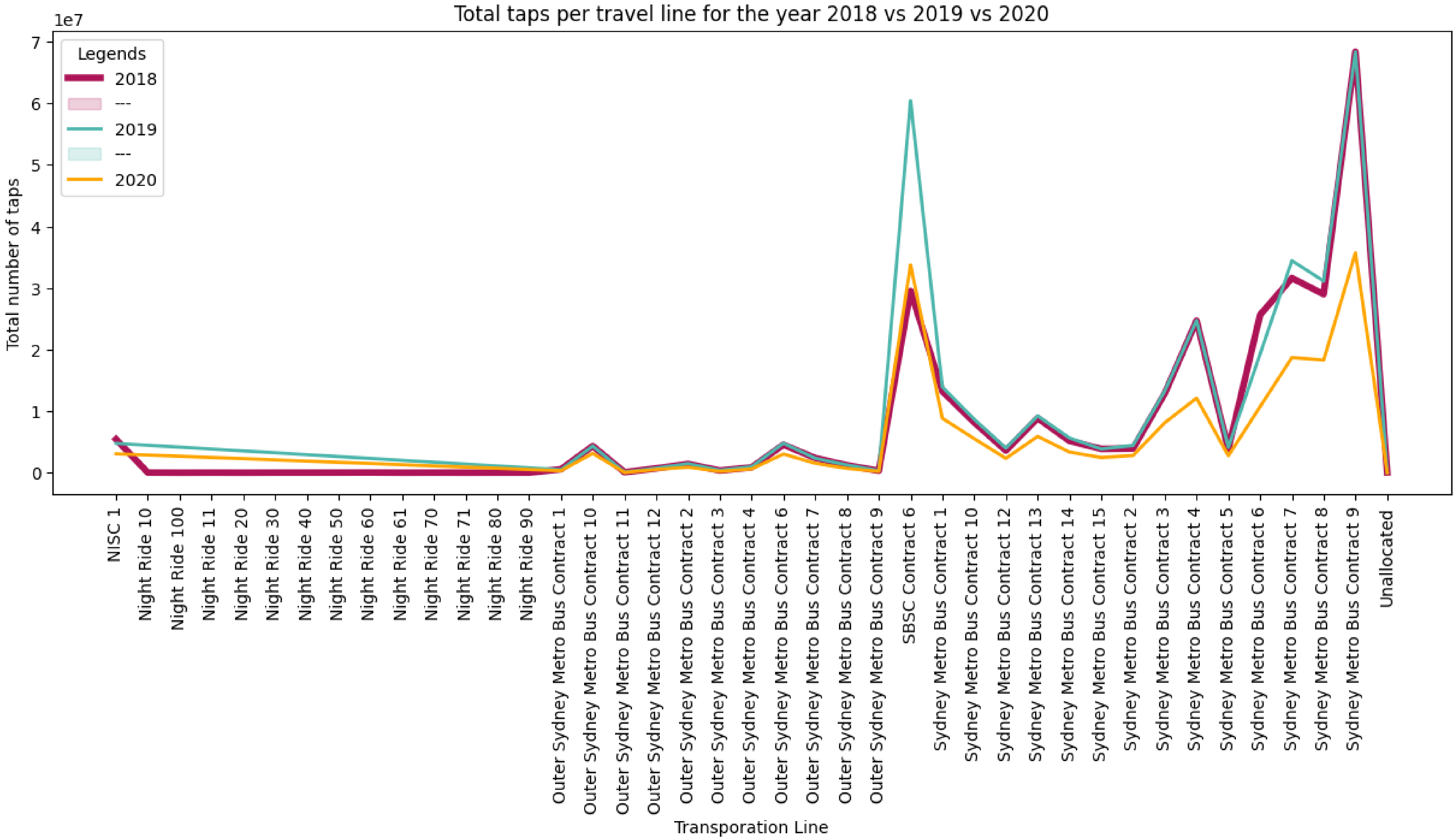


With the trend line, we can see that there is major hit during the 2020.

In [ ]:
```
contract2018 = contract2018.sort_values('Contract_Type')
contract2019 = contract2019.sort_values('Contract_Type')
contract2020 = contract2020.sort_values('Contract_Type')

sns.set_palette("Paired")
```

```python
plt.figure(figsize=(15,5))
plt.xticks(rotation=90)
sns.lineplot(data=contract2018, x="Contract_Type", y="Taps_Count", linewidth=4, color="#AD1457")
sns.lineplot(data=contract2019, x="Contract_Type", y="Taps_Count", linewidth=2, color="#4DB6AC")
sns.lineplot(data=contract2020, x="Contract_Type", y="Taps_Count", linewidth=2, color="orange")
plt.title('Total taps per travel line for the year 2018 vs 2019 vs 2020')
plt.legend(title='Legends', loc='upper left', labels=['2018','---', '2019', '---', '2020'])
plt.xlabel('Transporation Line')
plt.ylabel('Total number of taps')
```

Out[ ]:
```
Text(0, 0.5, 'Total number of taps')
```

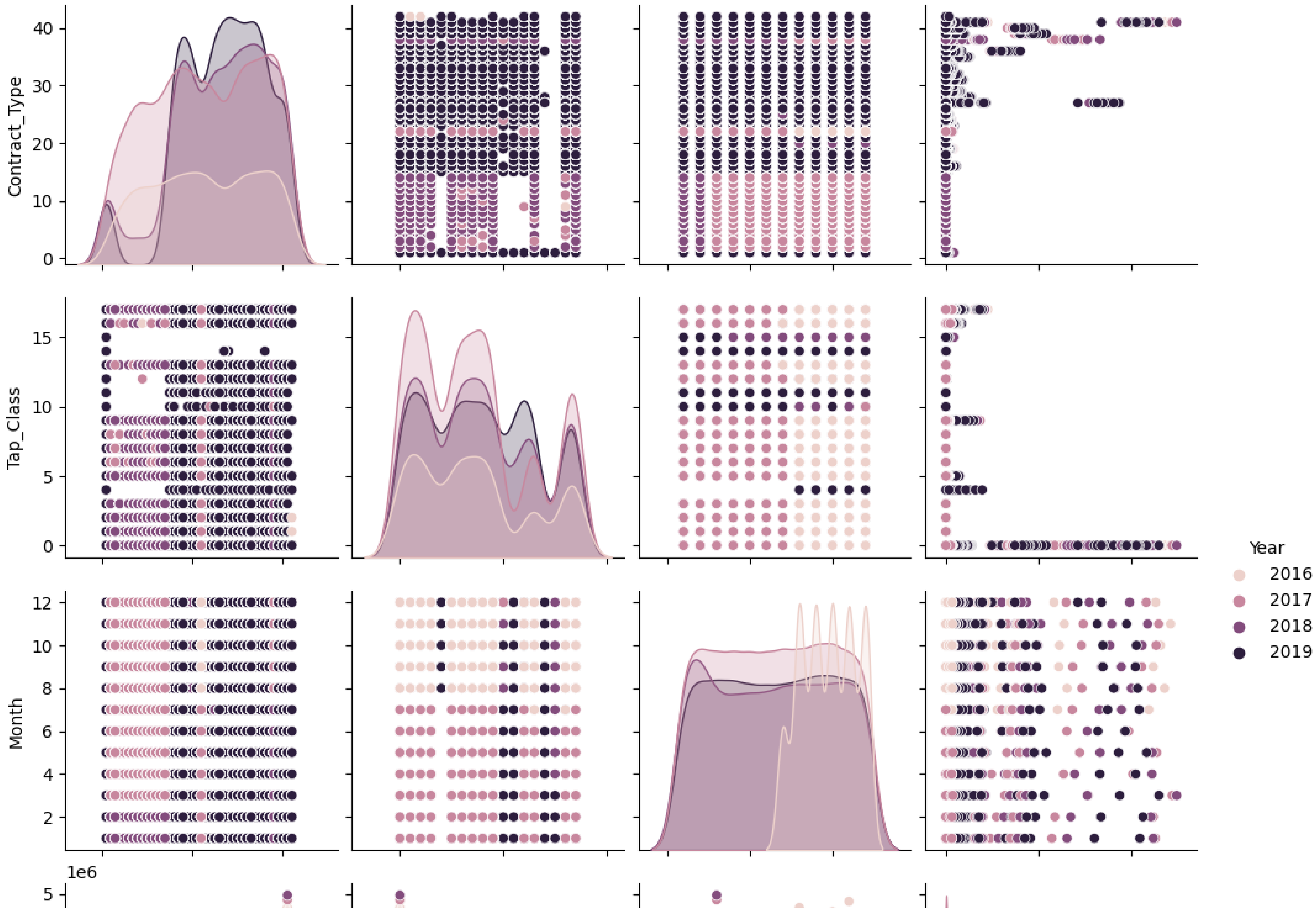


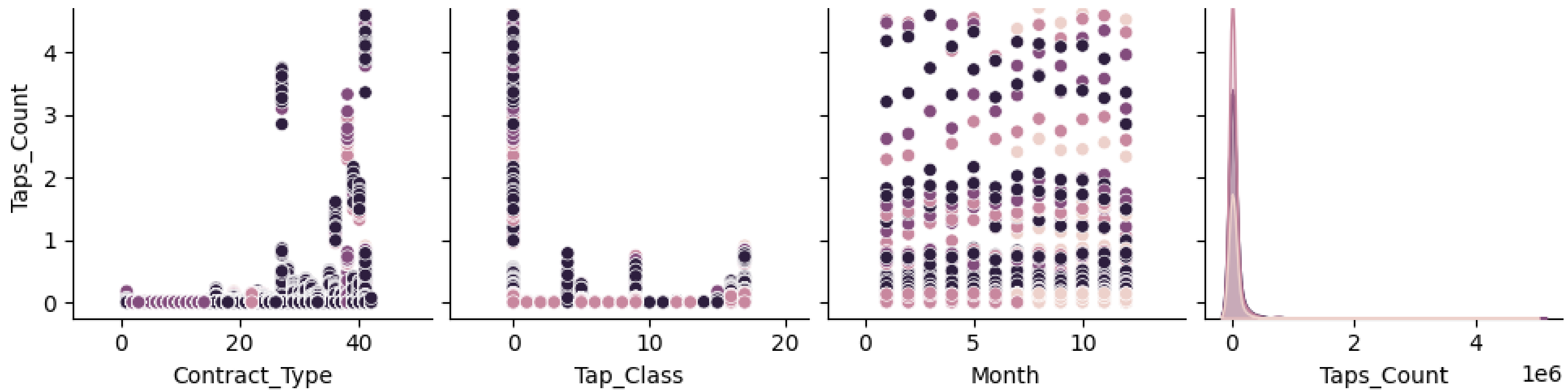
1.3 Transform the values of each row into an encoded order, check correlation and distribution.

In [ ]:
```
enc = OrdinalEncoder()
encodedWrangDF = wrangDF.copy()
encodedWrangDF[["Contract_Type","Tap_Class"]] = enc.fit_transform(encodedWrangDF[["Contract_Type","Tap_Class"]])
```

In [ ]:
```
#Exluding 2020 and 2021 as these are pandemic season.
sns.pairplot(data=encodedWrangDF[encodedWrangDF["Year"]<2020].drop(columns=['Day']), hue="Year")
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7fbc1d761a00>
```


Contract_Type
Tap_Class
Month
1e6
Year
2016
2017
2018
2019



In [ ]:
```
plt.figure(figsize=(8,5))
sns.heatmap(encodedWrangDF.drop(columns=['Day']).corr(),annot=True)
```

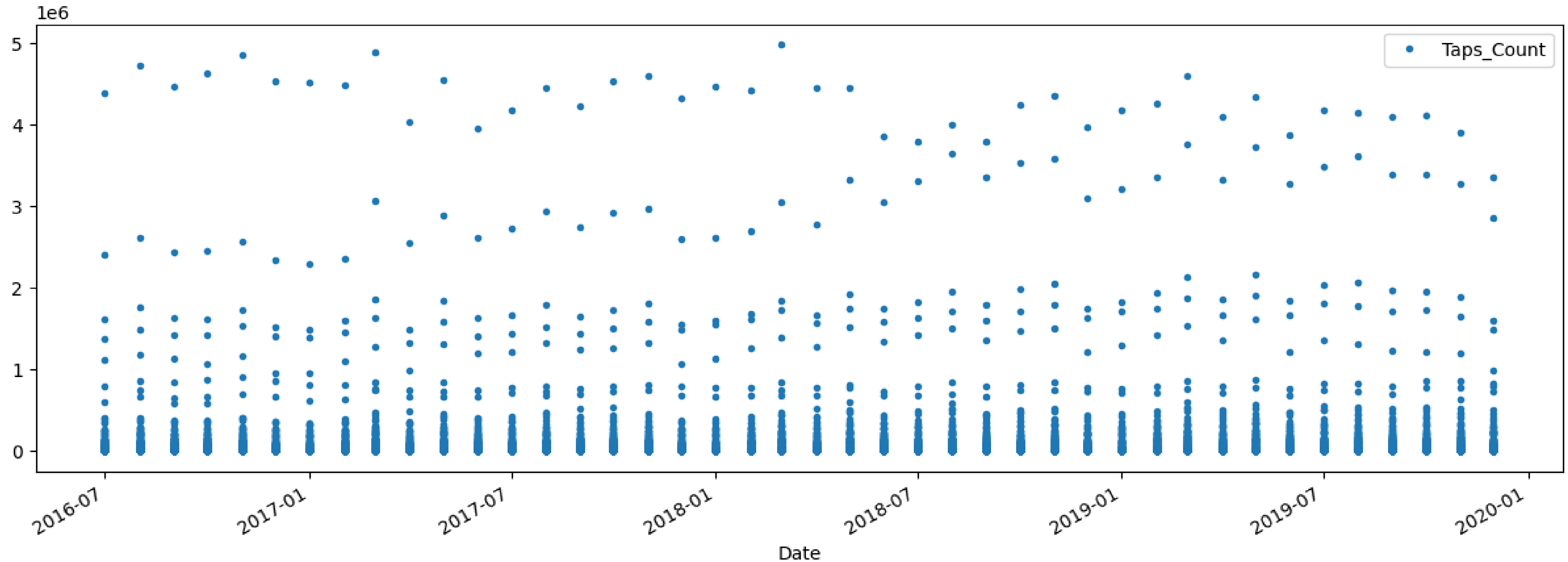
Out[ ]: <AxesSubplot: >

1.4 plot the dataframe using datetime column

```
In [ ]: tempWrangDF = encodedWrangDF[encodedWrangDF['Year'] < 2020].copy()
        tempWrangDF['Date'] = pd.to_datetime(tempWrangDF[['Year', 'Month', 'Day']])
        tempWrangDF = tempWrangDF.sort_values('Date', ascending=True)
```

```
In [ ]: tempWrangDF.plot(style='.',
            x="Date",
            y="Taps_Count",
            color=color_pal[0],
            figsize=(15, 5))
```

```
Out[ ]: <AxesSubplot: xlabel='Date'>
```



1.5 Normalize the dataframe

In [ ]:
```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler(feature_range=(1, 100))

taps_count = tempWrangDF['Taps_Count'].values
taps_count = taps_count.reshape(-1, 1)
x_scaled = scaler.fit_transform(taps_count)
tempWrangDF.Taps_Count = x_scaled
```

In [ ]:
```
tempWrangDF.head()
```

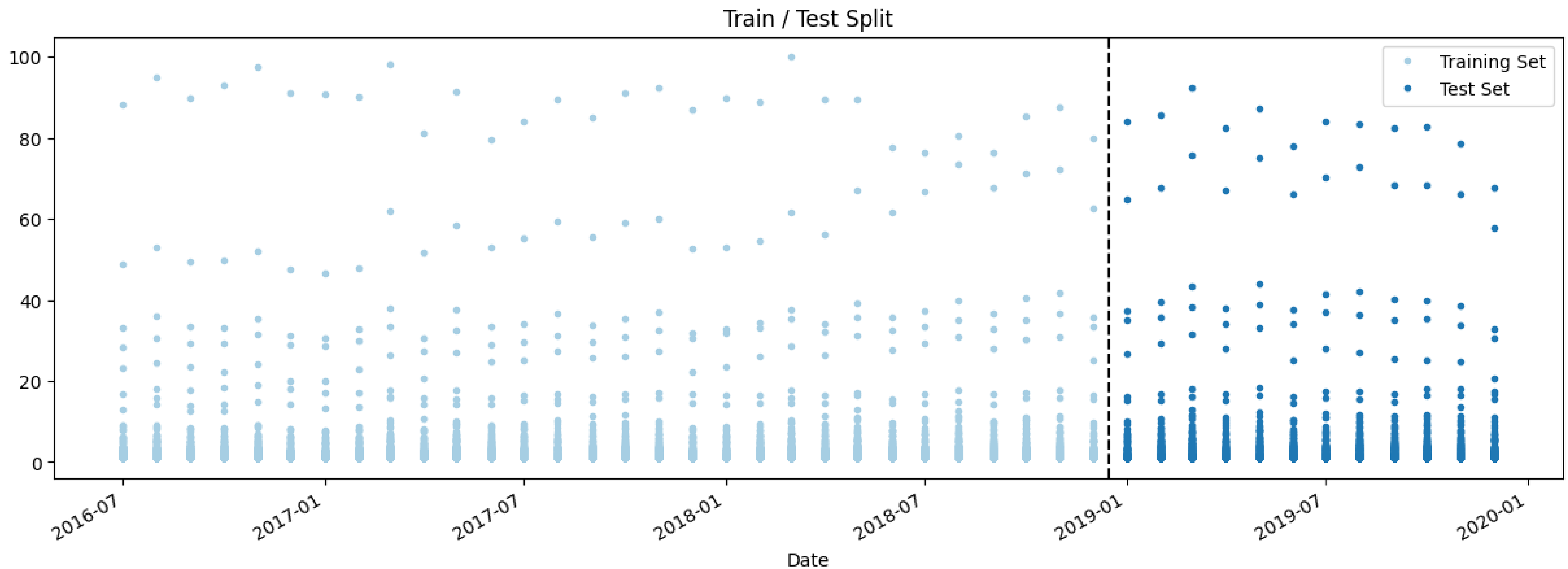
Out[ ]:

	Contract_Type	Tap_Class	Year	Month	Day	Taps_Count	Date
0	28.0	0.0	2016	7	1	8.769178	2016-07-01
16876	25.0	0.0	2016	7	1	1.622158	2016-07-01
4977	39.0	0.0	2016	7	1	33.149223	2016-07-01
16739	24.0	17.0	2016	7	1	2.057466	2016-07-01
16675	24.0	16.0	2016	7	1	1.945893	2016-07-01

2 Train / Test Split

In [ ]:

```
train = tempWrangDF.loc[tempWrangDF.Date < '2019-01-01']
test = tempWrangDF.loc[tempWrangDF.Date >= '2019-01-01']

fig, ax = plt.subplots(figsize=(15, 5))
train.plot(ax=ax, x="Date", y="Taps_Count", label="Training Set", style='.', title="Train / Test Split")
test.plot(ax=ax, x="Date", y="Taps_Count", label="Test Set", style='.')
ax.axvline('2018-12-15', color="black", ls="--")
plt.show()
```



3.1 Create the model - XGBOOST

In [ ]:
```
features = ['Contract_Type', 'Tap_Class', 'Year', 'Month']
target = 'Taps_Count'
```

In [ ]:
```
X_train = train[features]
y_train = train[target]

X_test = test[features]
y_test = test[target]
```

In [ ]:
```
reg = xgb.XGBRegressor(n_estimators = 10000, early_stopping_rounds=50, learning_rate = 0.001)
reg.fit(X_train,
    y_train,
```

```
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=500)
```

```
[0]	validation_0-rmse:6.10242	validation_1-rmse:6.61483
[500]	validation_0-rmse:3.85963	validation_1-rmse:4.21052
[1000]	validation_0-rmse:2.50591	validation_1-rmse:2.74982
[1500]	validation_0-rmse:1.70445	validation_1-rmse:1.86742
[2000]	validation_0-rmse:1.25059	validation_1-rmse:1.36417
[2500]	validation_0-rmse:1.00050	validation_1-rmse:1.09377
[3000]	validation_0-rmse:0.86179	validation_1-rmse:0.94920
[3500]	validation_0-rmse:0.77690	validation_1-rmse:0.88027
[4000]	validation_0-rmse:0.72704	validation_1-rmse:0.84135
[4500]	validation_0-rmse:0.68635	validation_1-rmse:0.81858
[5000]	validation_0-rmse:0.62631	validation_1-rmse:0.80220
[5500]	validation_0-rmse:0.59032	validation_1-rmse:0.79326
[6000]	validation_0-rmse:0.57191	validation_1-rmse:0.78861
[6500]	validation_0-rmse:0.55909	validation_1-rmse:0.78322
[7000]	validation_0-rmse:0.54826	validation_1-rmse:0.77771
[7500]	validation_0-rmse:0.53839	validation_1-rmse:0.77265
[8000]	validation_0-rmse:0.52690	validation_1-rmse:0.76402
[8500]	validation_0-rmse:0.51342	validation_1-rmse:0.74964
[9000]	validation_0-rmse:0.50135	validation_1-rmse:0.74374
[9500]	validation_0-rmse:0.48802	validation_1-rmse:0.73280
[9999]	validation_0-rmse:0.47869	validation_1-rmse:0.72224
```

Out[ ]:

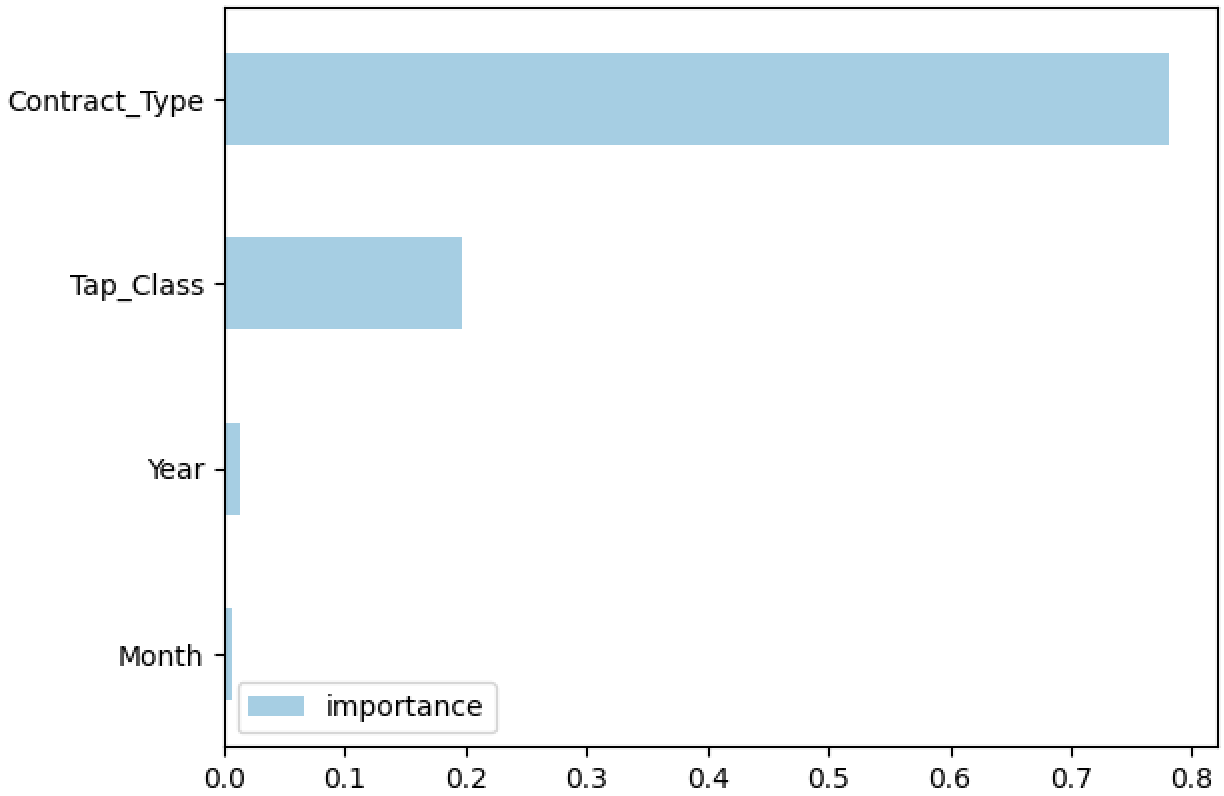
```
XGBRegressor

XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=50, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.001, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=10000,
             n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
```

3.1.1 Feature Importance

In [ ]:

```
fi = pd.DataFrame(data=reg.feature_importances_, index=reg.feature_names_in_, columns=['importance'])
```

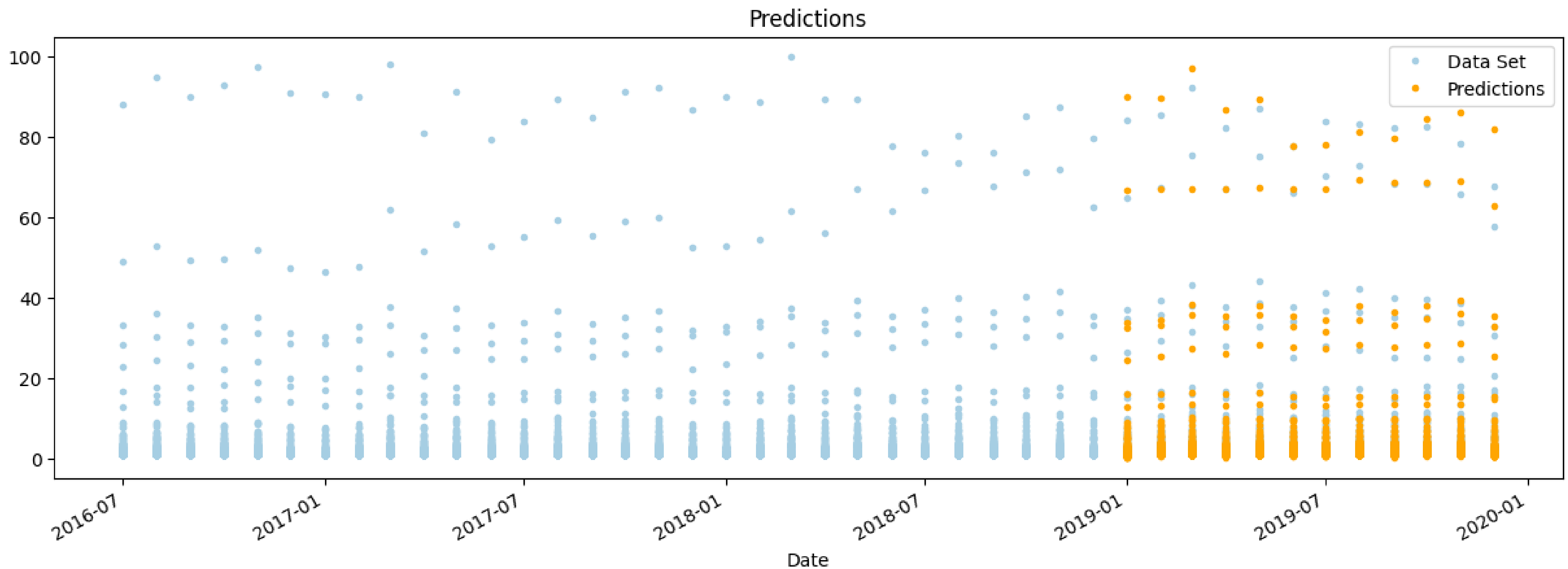
```
fi.sort_values('importance').plot(kind="barh")
plt.show()
```



3.1.2 Forecast on Test

In [ ]:
```
test['prediction'] = reg.predict(X_test)
tempWrangDF = tempWrangDF.merge(test[['prediction']], how='left', left_index=True, right_index=True)
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 5))
tempWrangDF.plot(ax=ax, x="Date", y="Taps_Count", label="Data Set", style='.', title="Predictions")
tempWrangDF.plot(ax=ax, x="Date", y="prediction", label="Predictions", style='.', color="orange")
plt.show()
```




```
In [ ]: score = np.sqrt(mean_squared_error(test['Taps_Count'], test['prediction']))
        score
```

```
Out[ ]: 0.7222387237925278
```

```
In [ ]: # Evaluation
        maeLog = mean_absolute_error(y_test, test['prediction'])
        mseLog = mean_squared_error(y_test, test['prediction'])
        rmseLog = np.sqrt(mean_squared_error(y_test, test['prediction']))
        r2Log = r2_score(y_test, test['prediction'])

        #Metric Logs
        print('Metric Logs')
        print(f'Mean absolute error: {maeLog:.3f}')
        print(f'Mean squared error: {mseLog:.3f}')
        print(f'Root mean squared error: {rmseLog:.2f}')
        print(f'Adjusted R-Squared: {r2Log:.3f}')
```

```
Metric Logs
Mean absolute error: 0.245
Mean squared error: 0.522
Root mean squared error: 0.72
Adjusted R-Squared: 0.987
```

3.1.3 Create a dictionary to compare the Root Mean Square Error and some metrics

In [ ]:
```
metrics_data = dict({'ModelName': ['XGBRegressor'], 'MAE': [maeLog], 'MSE': [mseLog], 'RMSE': [rmseLog], 'AdjustedRSquared':[r2Log]})
test_results = {}
test_results['XGBRegressor'] = np.sqrt(mean_squared_error(y_test, test['prediction']))
```

In [ ]:
```
test['error'] = np.abs(test[target] - test['prediction'])
test.sort_values('error', ascending=True).head(10)
```

Out[ ]:

	Contract_Type	Tap_Class	Year	Month	Day	Taps_Count	Date	prediction	error
13839	21.0	1.0	2019	5	1	1.030883	2019-05-01	1.030903	0.000020
12367	19.0	3.0	2019	10	1	1.013193	2019-10-01	1.013246	0.000053
12532	19.0	7.0	2019	9	1	1.005930	2019-09-01	1.005857	0.000073
12423	19.0	5.0	2019	12	1	1.147968	2019-12-01	1.148043	0.000075
14253	21.0	12.0	2019	2	1	1.002069	2019-02-01	1.001985	0.000085
12859	19.0	17.0	2019	9	1	1.554303	2019-09-01	1.554434	0.000131
15798	23.0	12.0	2019	1	1	1.023321	2019-01-01	1.023488	0.000166
9793	31.0	11.0	2019	2	1	1.000298	2019-02-01	1.000468	0.000169
2438	35.0	11.0	2019	12	1	1.001075	2019-12-01	1.000905	0.000170
14173	21.0	8.0	2019	9	1	1.002388	2019-09-01	1.002185	0.000203

3.2 Create the model - Linear Regression

In [ ]:
```
reg = LinearRegression()
regModel = reg.fit(X_train, y_train)
logModelPrediction = regModel.predict(X_test)
```

In [ ]:
```
# check actual and predicted scores
dfPreds = pd.DataFrame({'Actual': y_test.squeeze(), 'Predicted': logModelPrediction.squeeze()})
dfPreds.head()
```

Out[ ]:

	Actual	Predicted
2898	3.148183	3.137136
3741	1.248875	3.241419
549	1.040594	1.918643
2014	1.070800	3.224964
989	1.045210	3.600962

In [ ]:
```
# Evaluation
maeLog = mean_absolute_error(y_test, logModelPrediction)
mseLog = mean_squared_error(y_test, logModelPrediction)
rmseLog = np.sqrt(mean_squared_error(y_test, logModelPrediction))
r2Log = r2_score(y_test, logModelPrediction)

#Metric Logs
print('Metric Logs')
print(f'Mean absolute error: {maeLog:.3f}')
print(f'Mean squared error: {mseLog:.3f}')
print(f'Root mean squared error: {rmseLog:.2f}')
print(f'Adjusted R-Squared: {r2Log:.3f}')

test_results['LinearRegression'] = np.sqrt(mean_squared_error(y_test, logModelPrediction))

#Metrics Data

metrics_data['ModelName'].append('LinearRegression')
metrics_data['MAE'].append(maeLog)
metrics_data['MSE'].append(mseLog)
metrics_data['RMSE'].append(rmseLog)
metrics_data['AdjustedRSquared'].append(r2Log)
```

```
Metric Logs
Mean absolute error: 2.118
Mean squared error: 39.024
Root mean squared error: 6.25
Adjusted R-Squared: 0.035
```

3.3 Create the model - Neural Network Regression and Deep Neural Network Regression

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

```
2022-10-12 19:49:39.051057: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Netwo
rk Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
No supported GPU was found.
2.10.0
```

3.3.1 Preparing the dataset

In [ ]:
```
train_dataset = encodedWrangDF.loc[encodedWrangDF.Year < 2019]
test_dataset = encodedWrangDF.loc[encodedWrangDF.Year == 2019]
```

In [ ]:
```
transformer = Normalizer()
```

In [ ]:
```
train_features = train_dataset.drop(columns=['Taps_Count']).copy()
test_features = test_dataset.drop(columns=['Taps_Count']).copy()

train_labels = train_dataset.drop(columns=['Contract_Type', 'Tap_Class', 'Year', 'Day','Month'])
test_labels = test_dataset.drop(columns=['Contract_Type', 'Tap_Class', 'Year', 'Day','Month'])

train_labels.Taps_Count = transformer.transform(train_labels[['Taps_Count']])
test_labels.Taps_Count = transformer.transform(test_labels[['Taps_Count']])
```

In [ ]:
```
normalizer = tf.keras.layers.Normalization(axis=-1)
```

```
normalizer.adapt(np.array(train_features))
```

```
2022-10-12 19:49:44.162504: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Netwo
rk Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [ ]:

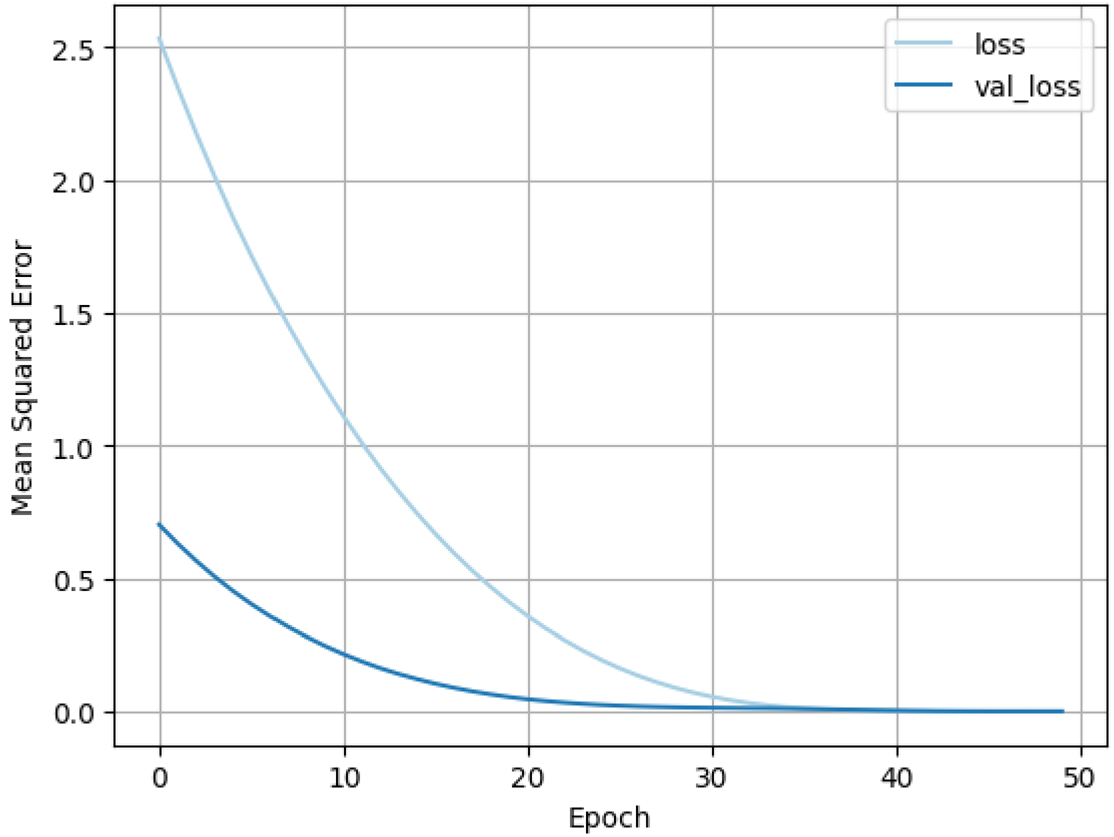
```
def plot_loss(history, _label):
  label = _label
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.xlabel('Epoch')
  plt.ylabel(label)
  plt.legend()
  plt.grid(True)
```

3.3.2 Linear Regression Neural Network with multiple inputs

In [ ]:

```
linear_model = tf.keras.Sequential([
    normalizer,
    layers.Dense(units=1)
])

linear_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='mean_squared_error')

lin_model = linear_model.fit(
    train_features,
    train_labels['Taps_Count'],
    epochs=50,
    # Suppress logging.
    verbose=0,
    # Calculate validation results on 20% of the training data.
    validation_split = 0.2)

test_results['linear_neural_network'] = np.sqrt(linear_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0))
test_results

metrics_data['ModelName'].append('Linear_Neural_Networ')
metrics_data['MSE'].append(linear_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0))
metrics_data['RMSE'].append(np.sqrt(linear_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0)))
```

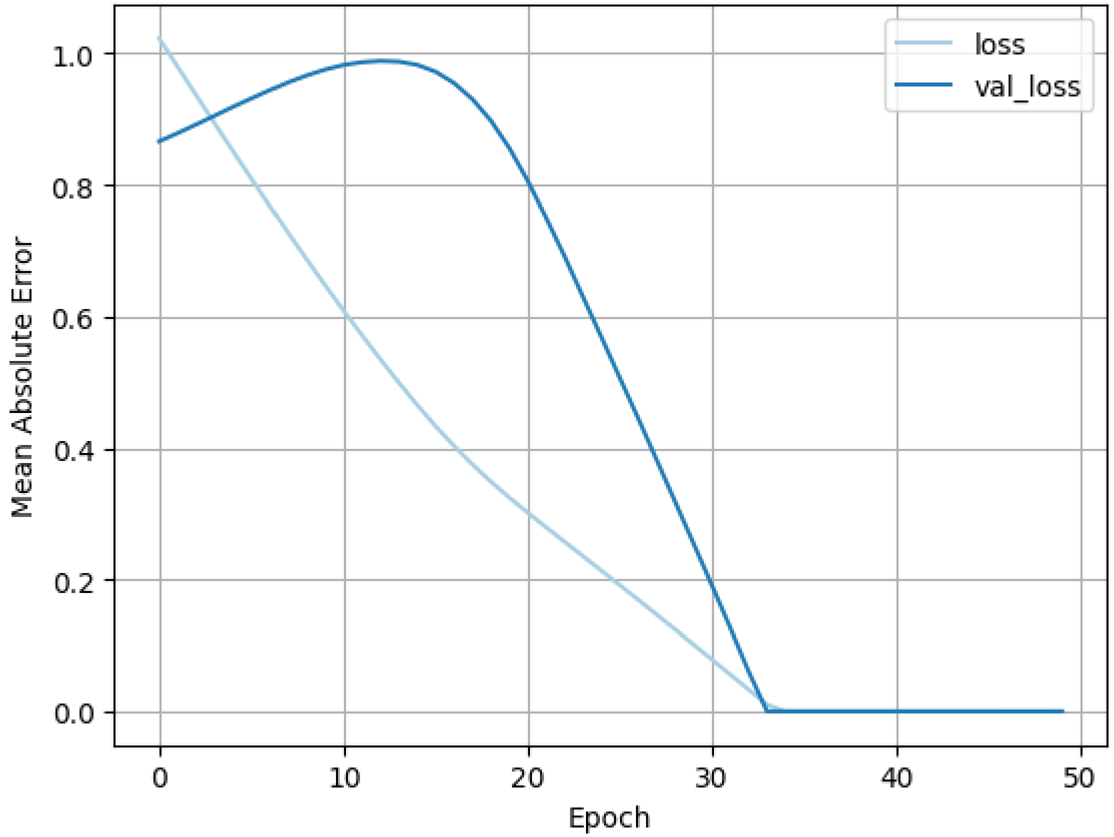
```
plot_loss(lin_model, 'Mean Squared Error')
```



In [ ]:

```
##Mean Absolute Error
linear_model = tf.keras.Sequential([
    normalizer,
    layers.Dense(units=1)
])

linear_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='mean_absolute_error')

lin_model = linear_model.fit(
    train_features,
    train_labels['Taps_Count'],
    epochs=50,
```

```
    # Suppress logging.
    verbose=0,
    # Calculate validation results on 20% of the training data.
    validation_split = 0.2)

metrics_data['MAE'].append(linear_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0))
metrics_data['AdjustedRSquared'].append(0)

plot_loss(lin_model, 'Mean Absolute Error')
```

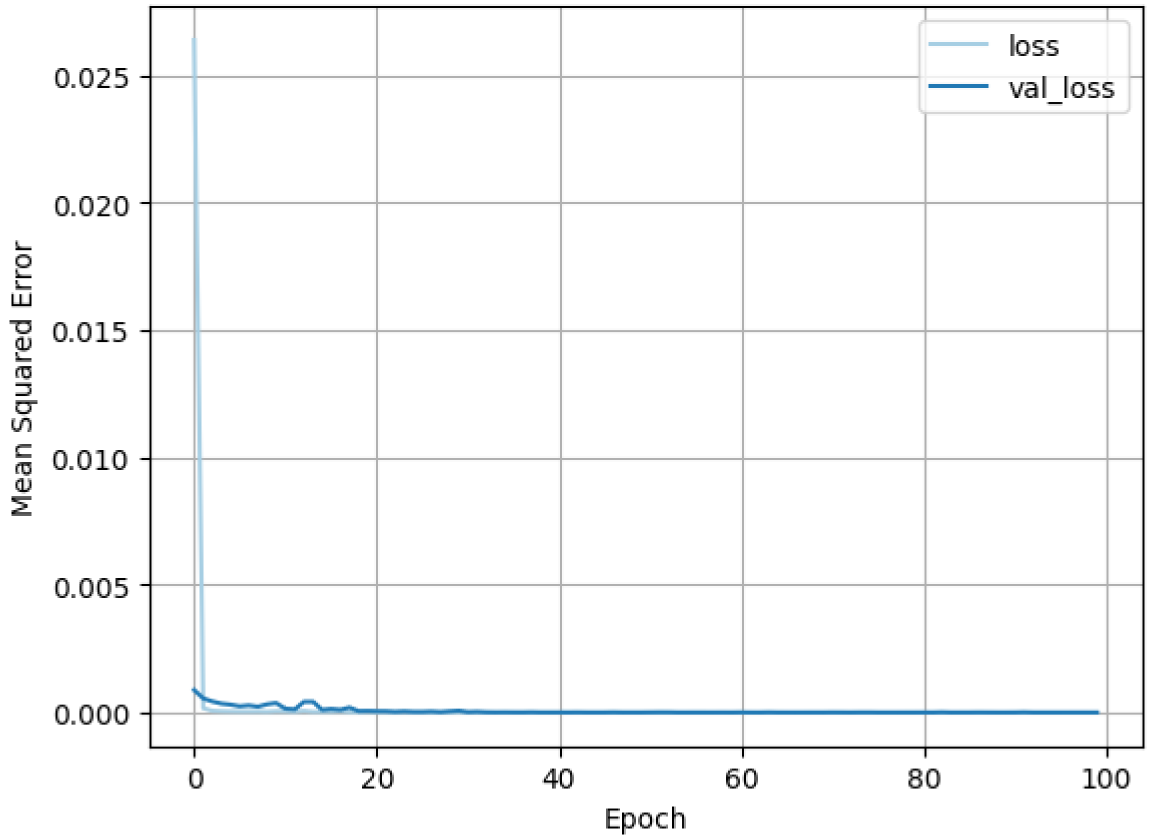


3.3.3 Regression using a DNN and multiple inputs

In [ ]:
```
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(32, activation='relu'),
```

```
      layers.Dense(64, activation='relu'),
      layers.Dense(128, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(32, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_squared_error',
    optimizer=tf.keras.optimizers.Adam(0.001))

  return model

dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()

dnn_model_ = dnn_model.fit(
    train_features,
    train_labels,
    validation_split=0.2,
    verbose=0,
    epochs=100)

test_results['Linear_DNN'] = np.sqrt(dnn_model.evaluate(test_features, test_labels, verbose=0))

metrics_data['ModelName'].append('Linear_DNN')
metrics_data['MSE'].append(dnn_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0))
metrics_data['RMSE'].append(np.sqrt(dnn_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0)))

plot_loss(dnn_model_, 'Mean Squared Error')
```

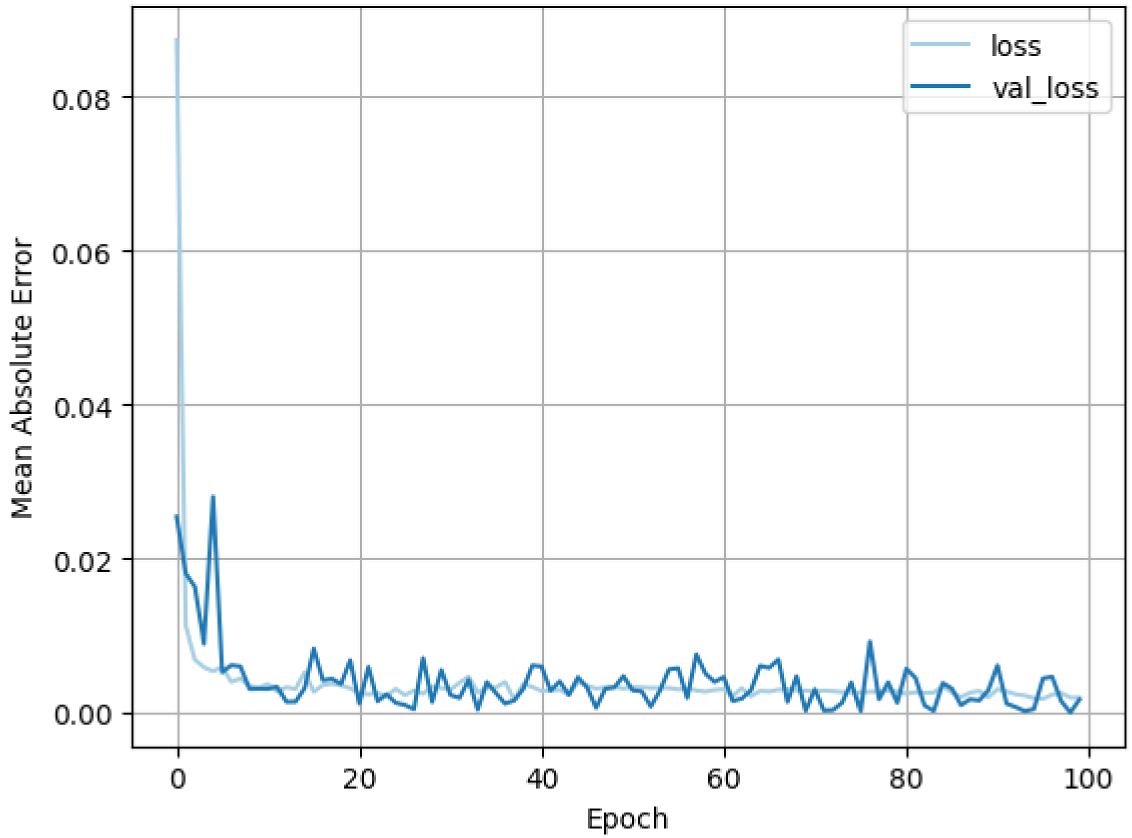
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 5)                11
 n)

 dense_2 (Dense)             (None, 32)                192

 dense_3 (Dense)             (None, 64)                2112

 dense_4 (Dense)             (None, 128)               8320

 dense_5 (Dense)             (None, 64)                8256

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 21,004
Trainable params: 20,993
Non-trainable params: 11
_________________________________________________________________
```



In [ ]:
```
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(32, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(128, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(32, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_absolute_error',
    optimizer=tf.keras.optimizers.Adam(0.001))

  return model

dnn_model = build_and_compile_model(normalizer)
```

```python
dnn_model.summary()

dnn_model_ = dnn_model.fit(
    train_features,
    train_labels,
    validation_split=0.2,
    verbose=0,
    epochs=100)

metrics_data['MAE'].append(dnn_model.evaluate(test_features, test_labels['Taps_Count'], verbose=0))
metrics_data['AdjustedRSquared'].append(0)

plot_loss(dnn_model_, 'Mean Absolute Error')
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 normalization (Normalizatio  (None, 5)                11        
 n)                                                              
                                                                 
 dense_8 (Dense)             (None, 32)                192       
                                                                 
 dense_9 (Dense)             (None, 64)                2112      
                                                                 
 dense_10 (Dense)            (None, 128)               8320      
                                                                 
 dense_11 (Dense)            (None, 64)                8256      
                                                                 
 dense_12 (Dense)            (None, 32)                2080      
                                                                 
 dense_13 (Dense)            (None, 1)                 33        
                                                                 
=================================================================
Total params: 21,004
Trainable params: 20,993
Non-trainable params: 11
_________________________________________________________________
```

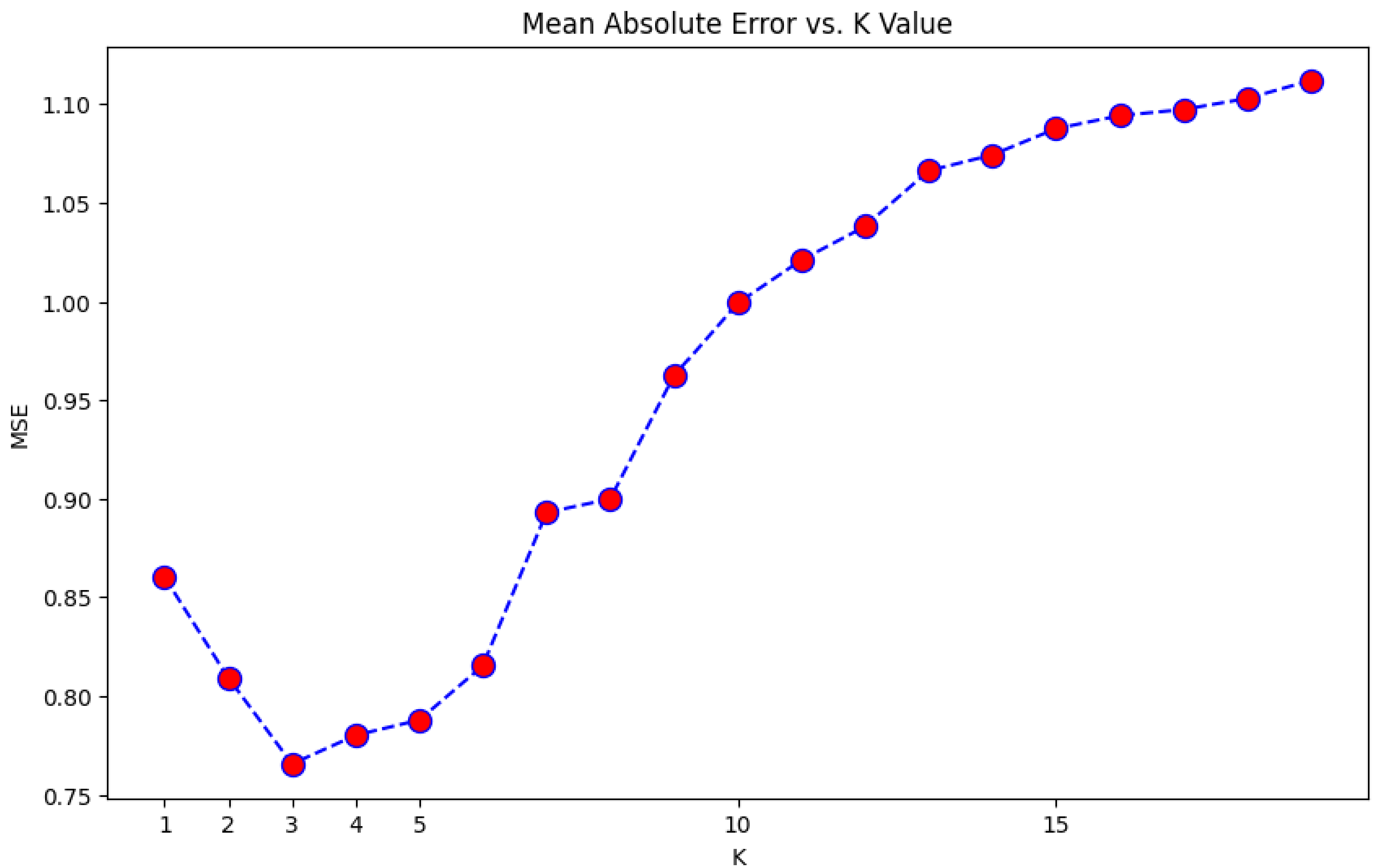


3.4 Create the model - KNearest Regression

In [ ]:
```
train_dataset = tempWrangDF.copy()

X = train_dataset[['Contract_Type', 'Tap_Class', 'Year', 'Month']]
y = train_dataset[['Taps_Count']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

mseList = []
# Will take some time
for i in range(1,20):
    neigh = KNeighborsRegressor(n_neighbors = i).fit(X_train, y_train)
    yhat = neigh.predict(X_test)
    mseList.append(mean_absolute_error(y_test, yhat))
```

```python
plt.figure(figsize=(10,6))
plt.xticks([1,2,3,4,5,10,15,20])
plt.plot(range(1,20),mseList,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Mean Absolute Error vs. K Value')
plt.xlabel('K')
plt.ylabel('MSE')
print("Maximum accuracy:-",max(mseList),"at K =",mseList.index(max(mseList)))
```

```
Maximum accuracy:- 1.1120953509194582 at K = 18
```



In [ ]:

```python
#fit the model
reg = KNeighborsRegressor(n_neighbors = 3)
knnModel = reg.fit(X_train, y_train)
knnModelPrediction = knnModel.predict(X_test)

# Evaluation
maeLog = mean_absolute_error(y_test, knnModelPrediction)
mseLog = mean_squared_error(y_test, knnModelPrediction)
rmseLog = np.sqrt(mean_squared_error(y_test, knnModelPrediction))
r2Log = r2_score(y_test, knnModelPrediction)

#Metric Logs
print('Metric Logs')
print(f'Mean absolute error: {maeLog:.3f}')
print(f'Mean squared error: {mseLog:.3f}')
print(f'Root mean squared error: {rmseLog:.2f}')
print(f'Adjusted R-Squared: {r2Log:.3f}')

test_results['KNearestRegressor = 3'] = np.sqrt(mean_squared_error(y_test, knnModelPrediction))

#Metrics Data

metrics_data['ModelName'].append('KNearestRegressor = 3')
metrics_data['MAE'].append(maeLog)
metrics_data['MSE'].append(mseLog)
metrics_data['RMSE'].append(rmseLog)
metrics_data['AdjustedRSquared'].append(r2Log)
```

```
Metric Logs
Mean absolute error: 0.766
Mean squared error: 14.451
Root mean squared error: 3.80
Adjusted R-Squared: 0.715
```

4. Visualize, Compare and Analyze the Results

In [ ]:

```python
metricsDF = pd.DataFrame(metrics_data)
metricsDF
```

Out[ ]:

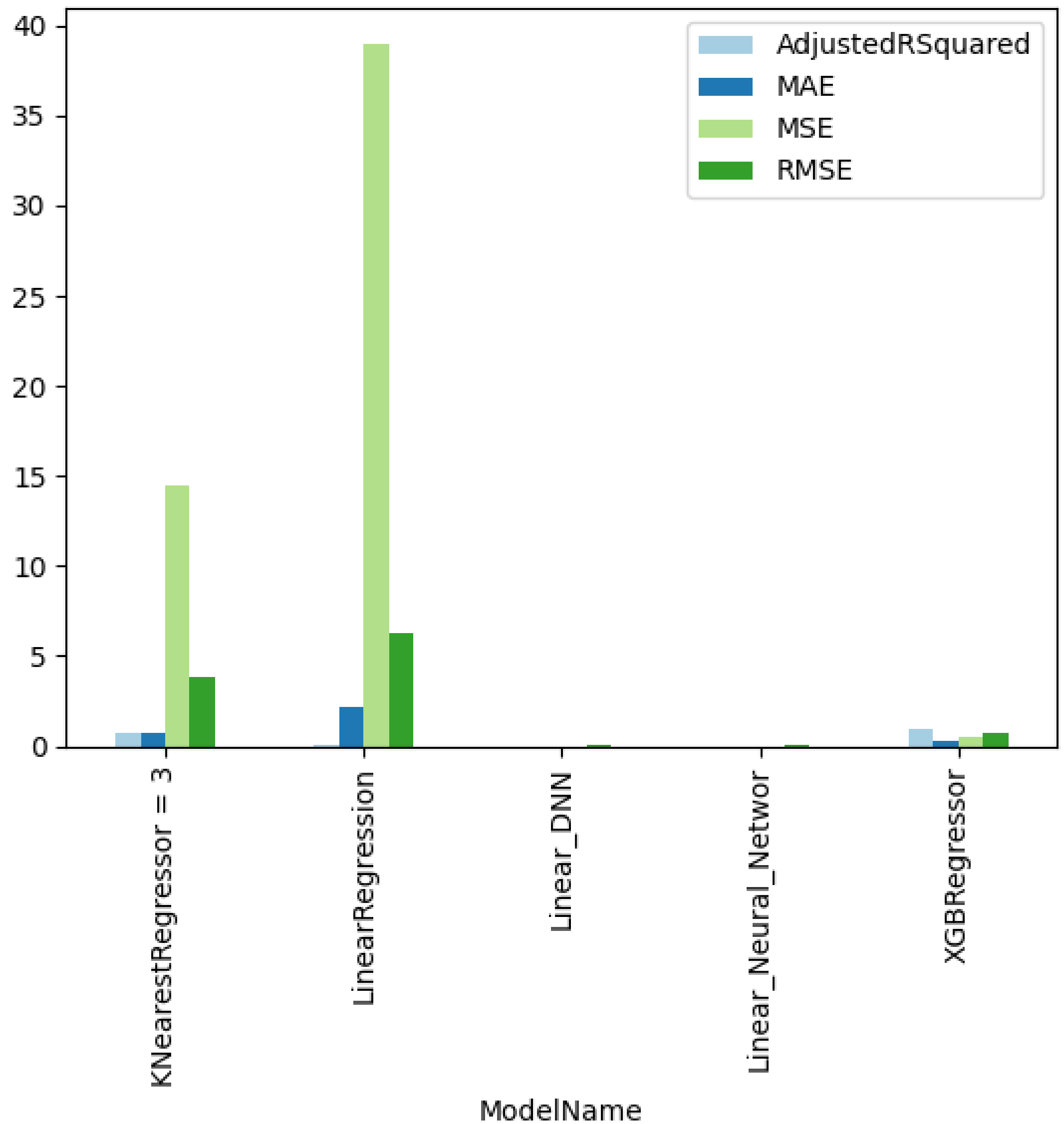
	ModelName	MAE	MSE	RMSE	AdjustedRSquared
0	XGBRegressor	0.245038	0.521629	0.722239	0.987099
1	LinearRegression	2.118230	39.023988	6.246918	0.034884
2	Linear_Neural_Networ	0.000525	0.000437	0.020912	0.000000
3	Linear_DNN	0.002231	0.000438	0.020927	0.000000
4	KNearestRegressor = 3	0.765523	14.450976	3.801444	0.714789

In [ ]:

```
metricsDF.pivot_table(index="ModelName").plot(kind='bar')
```

Out[ ]:

```
<AxesSubplot: xlabel='ModelName'>
```

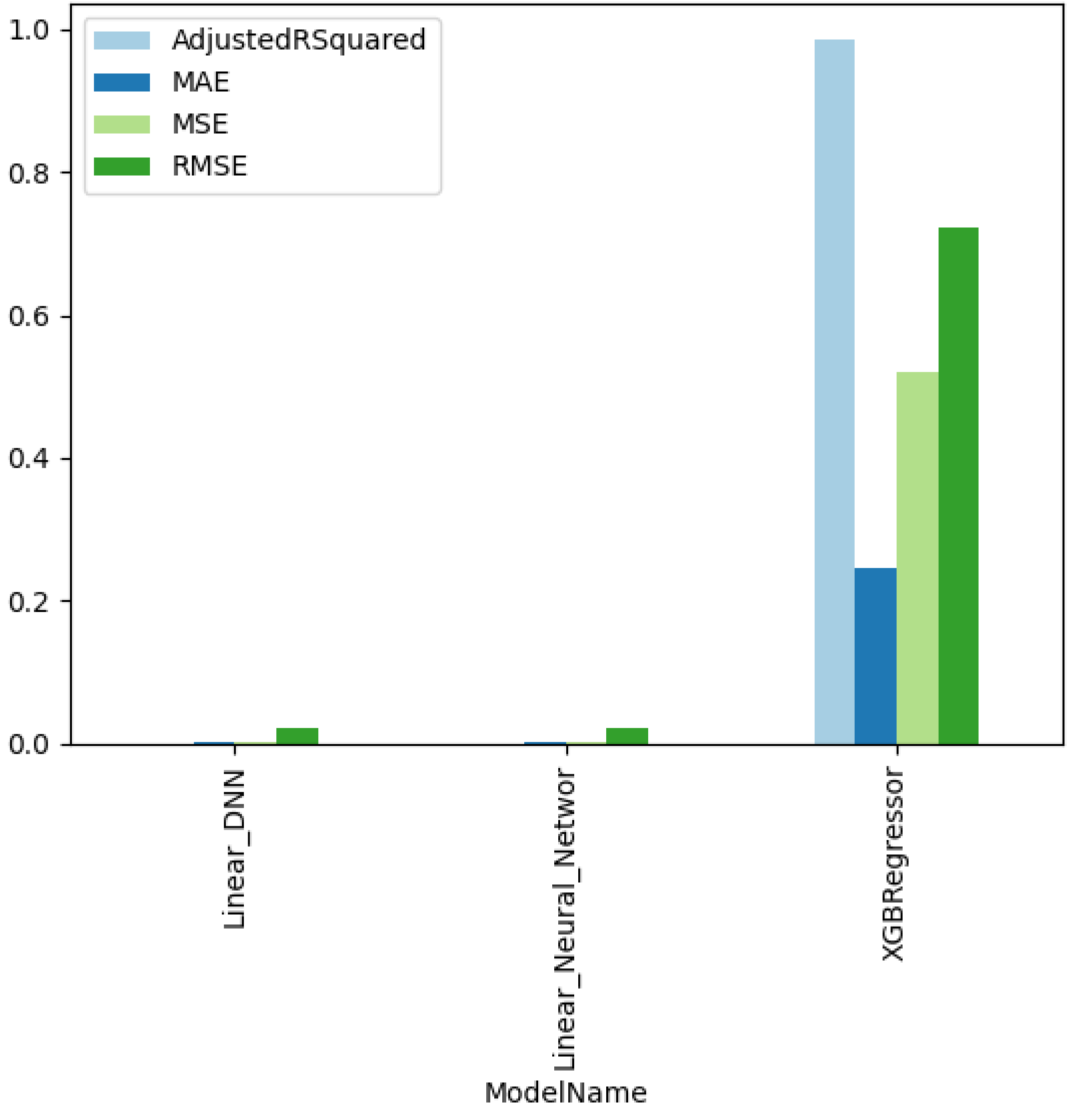


In [ ]:

```
_metricsDF = metricsDF.drop(index=metricsDF.index[1], axis=0)
_metricsDF = _metricsDF.drop(index=_metricsDF.index[3], axis=0)
_metricsDF.pivot_table(index="ModelName").plot(kind='bar')
```

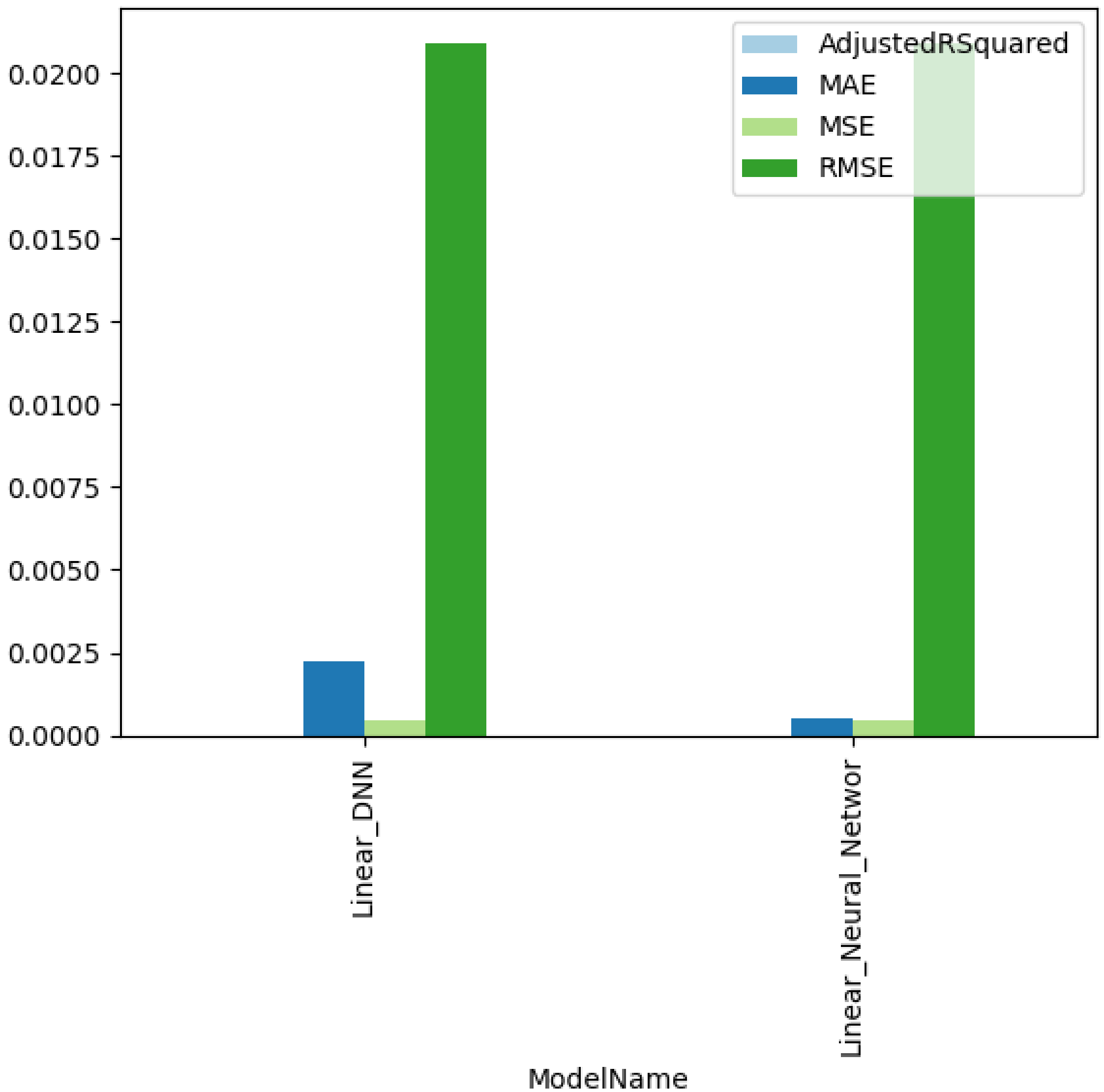
Out[ ]:

```
<AxesSubplot: xlabel='ModelName'>
```



In [ ]:
```
_metricsDF.drop(index=metricsDF.index[0], axis=0).pivot_table(index="ModelName").plot(kind='bar')
```

Out[ ]: <AxesSubplot: xlabel='ModelName'>



Conclusion

Model Conclusion: RMSE or Root Mean Square Error is one of the popular measure for evaludating the quality of the predictions. Thus if we checked the metrics of our generated model, we can deduce that Linear Regression with Deep Neural Network and Linear Neural Network (non-deep model) are currently the best models. During re-runs, sometimes Deep Learning has the best metric sometimes, the Neural Network has the best metric. So far, there is no way to reproduce the generated model performance into another machine but generally, they will be still performant. The amount of time to get the best parameter in neural network sometimes does not weigh the benefist. In some cases, altho neural network is a very powerful algorithm to generate a model, using an ensemble model can give us a run for our time too.

Data Conclusion: On the other hand, the dataset that was pulled from open data of NSW transporation is a relatively clean and a good dataset.