

PUBLIC TRANSPORTATION EFFICIENCY ANALYSIS

Phase4 Submission:

INTRODUCTION :

Real-time data processing

1. Collect data from public transportation vehicles and infrastructure. This data can include vehicle location, passenger load, and traffic conditions.
2. Use a serverless event stream processing platform, such as AWS Kinesis Firehose or Google Cloud Dataflow, to process the data in real time. These platforms can scale automatically based on demand, so you don't have to worry about provisioning or managing servers.
3. Write code to process the data in real time. The code can perform various tasks, such as filtering the data, transforming the data, and aggregating the data.
4. Deploy your code to the serverless event stream processing platform.

Automation

5. Use a serverless workflow management platform, such as AWS Step Functions or Google Cloud Workflow Orchestration, to automate the public transportation efficiency analysis workflow. These platforms allow you to create visual workflows that can be triggered by events, such as new data arriving from public transportation vehicles and infrastructure.
6. Use serverless functions to automate specific tasks in the public transportation efficiency analysis workflow. For example, you can use a serverless function to calculate the on-time performance of public transportation vehicles.

Storage

7. Use a serverless database, such as Amazon DynamoDB or Google Cloud Firestore, to store the public transportation efficiency analysis

results. These databases are scalable and easy to use, and they can handle both structured and unstructured data.

8. Configure your serverless event stream processing platform to store the processed data in the serverless database.
9. Use serverless functions to access and analyze the data in the serverless database.

Example

Here is an example of how to implement real-time data processing, automation, and storage in public transportation efficiency analysis using AWS:

10. Collect data from public transportation vehicles and infrastructure using AWS IoT Core. AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices.
11. Use AWS Kinesis Firehose to process the data in real time. AWS Kinesis Firehose is a fully managed service for delivering real-time streaming data to data lakes, data stores, and analytics services.
12. Use AWS Lambda to automate the public transportation efficiency analysis workflow. AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers.
13. Use Amazon DynamoDB to store the public transportation efficiency analysis results. Amazon DynamoDB is a fully managed, multi-region, multi-master, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications.

Benefits of using a serverless architecture for public transportation efficiency analysis

There are several benefits to using a serverless architecture for public transportation efficiency analysis:

- **Scalability:** Serverless architectures can scale automatically based on demand, so you don't have to worry about provisioning or managing servers.
- **Cost-effectiveness:** Serverless architectures are pay-as-you-go, so you only pay for the resources that you use.
- **Ease of use:** Serverless architectures are easy to use and manage. You don't have to worry about provisioning or managing servers, and you can deploy your code with just a few clicks.

```
python
import pandas as pd

# Load data from a CSV file (replace 'your_data.csv' with the actual file path)
data = pd.read_csv('your_data.csv')

# Basic data exploration
print("Data Overview:")
print(data.head()) # Display the first few rows of data

# Calculate key metrics
# Example 1: Average passenger load per route
average_load_per_route = data.groupby('Route')['PassengerLoad'].mean()

# Example 2: On-time performance
data['OnTime'] = data['ActualArrivalTime'] <= data['ScheduledArrivalTime']
on_time_performance = data['OnTime'].sum() / len(data) * 100 # Percentage of on-time arrivals

print("\nAverage Load per Route:")
print(average_load_per_route)

print("\nOn-Time Performance:")
print(f"On-Time Percentage: {on_time_performance:.2f}%")

# More analysis can be performed, such as identifying routes with consistently low on-time
# performance, optimizing schedules, and more.
```

In this example, we assume you have a CSV file containing data on routes, passenger loads, scheduled arrival times, and actual arrival times. You can adapt this code to your specific dataset and analysis needs. Additional analysis can include identifying bottlenecks, optimizing routes, or suggesting schedule improvements.

For more complex analysis, I can certainly help you with the next phase of your project. To create visualizations and integrate code for data analysis, follow these steps:

1. *Data Preparation*:

Ensure that your data is cleaned and properly structured for analysis. This may include merging datasets, handling missing values, and preparing data for IBM Cognos and Python.

2. *IBM Cognos Visualizations*:

- a. Open IBM Cognos and create a new report or dashboard.
- b. Choose the relevant data sources and connect them to your project.
- c. Design visualizations to represent on-time performance, passenger feedback, and service efficiency metrics. Use charts, graphs, and tables to display this data effectively.

3. *Python Code for Data Analysis*:

a. Utilize Python for advanced data analysis. You can use libraries like Pandas, NumPy, and Matplotlib for this purpose.

b. Calculate service punctuality rates: Write code to compute on-time performance metrics, such as on-time arrival percentages, and display them visually.

c. Sentiment analysis on passenger feedback: Use Natural Language Processing (NLP) libraries like NLTK or spaCy for sentiment analysis. This can help you gauge passenger satisfaction based on their feedback.

4. *Integration*:

Integrate the visualizations and reports created in IBM Cognos with your Python analysis. You can export visualizations from Cognos and include them in your Python reports or dashboards.

5. *Testing*:

Thoroughly test your visualizations and Python code to ensure accuracy and functionality

```
%matplotlib inline
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import datetime
import os
from math import sqrt
import warnings

## For Multiple Output in single cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
warnings.filterwarnings('ignore')

data = pd.read_csv('../input/unisys/ptsboardingsummary/20140711.CSV')
data.shape
data.head(10)
```

TripID	RouteID	StopID	StopName	WeekBeginnin g	NumberOfBoa rdings	
0	23631	100	14156	181 Cross Rd	2013-06-30 00:00:00	1
1	23631	100	14144	177 Cross Rd	2013-06-30 00:00:00	1
2	23632	100	14132	175 Cross Rd	2013-06-30 00:00:00	1

3	23633	100	12266	Zone A Arndale Interchange	2013-06-30 00:00:00	2
4	23633	100	14147	178 Cross Rd	2013-06-30 00:00:00	1
5	23634	100	13907	9A Marion Rd	2013-06-30 00:00:00	1
6	23634	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
7	23634	100	13335	9A Holbrooks Rd	2013-06-30 00:00:00	1
8	23634	100	13875	9 Marion Rd	2013-06-30 00:00:00	1
9	23634	100	13045	206 Holbrooks Rd	2013-06-30 00:00:00	1

```

out_geo = pd.read_csv('../input/outgeo/output_geo.csv')
out_geo.shape
out_geo.head()

```

```
(4165, 10)
```

Out[3]:

	accuracy	formatted_address	google_place_id	input_street	latitude	longitude	number_of_results	postcode	status	type
0	ROOFTOP	181 Cross Rd, Westbourne Park SA 5041, Australia	ChIJKT7I9rbPsGoRVHMhkly-Oyk	181 Cross Rd	-34.966656	138.592148	1	5041	OK	street_address
1	ROOFTOP	177 Cross Rd, Westbourne Park SA 5041, Australia	ChIJ-VFZ87bPsGoRyfVgC5qbPpE	177 Cross Rd	-34.966607	138.592301	1	5041	OK	street_address

2	ROOFTOP	175 Cross Rd, Westbourne Park SA 5041, Australia	ChIJltlrbPsGoR38KRk76kPFI	175 Cross Rd	-34.966758	138.592715	1	5041	OK	street_address
3	GEOMETRIC_CENTER	Zone A Arndale Interchange - South side, Kilke...	ChIJn0C1hCPGsGoRIWvCdHf1RIg	Zone A Arndale Interchange	-34.875160	138.551628	1	5009	OK	bus_station,establishment,point_of_interest,transit
4	ROOFTOP	178 Cross Rd, Malvern SA 5061, Australia	ChIJycNiyvOsGoRdhfq9GKnpqQ	178 Cross Rd	-34.964960	138.611477	1	5061	OK	street_address

External Features

In [4]:

```

#DistanceFromCentre: Distance measure from the city centre
#For Calculating Distance between centre with other bus stops by using
Longitude and Latitude
#we have used the Haversine formula

from math import sin, cos, sqrt, atan2, radians
def calc_dist(lat1,lon1):
    ## approximate radius of earth in km
    R = 6373.0
    dlon = radians(138.604801) - radians(lon1)
    dlat = radians(-34.921247) - radians(lat1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(-34.921247)) *
sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    return R * c

```

```
out_geo['dist_from_centre'] =
out_geo[['latitude', 'longitude']].apply(lambda x: calc_dist(*x), axis=1)
```

In [6]:

```
out_geo.head()
```

accuracy	formatted_address	google_place_id	input_string	latitude	longitude	number_of_results	postcode	status	type	dist_from_centre	
0	ROOFTOP	181 Cross Rd, Westbourne Park SA 5041, Australia	ChIJKT7I9rbPsGoRVHMHkly-Oyk	181 Cross Rd	- 34.966 656	138.59 2148	1	5041	OK	street_addresses	5.1809 61
1	ROOFTOP	177 Cross Rd, Westbourne Park SA 5041, Australia	ChIJ-VFZ87bPsGoRyfVgC5qbPpE	177 Cross Rd	- 34.966 607	138.59 2301	1	5041	OK	street_addresses	5.1725 25
2	ROOFTOP	175 Cross Rd, Westbourne Park SA 5041, Australia	ChIJztIirbPsGoR38KRk76kPFI	175 Cross Rd	- 34.966 758	138.59 2715	1	5041	OK	street_addresses	5.1807 09

3	GEOMETRIC_CENTER	Zone A Arndale Interchange - South side, Kilke...	ChIJn0C1hCPGsGoRIWvCdhF1RIg	Zone A Arndale Interchange	- 34.875160	138.551628	1	5009	OK	bus_station, establishment, point_of_interest, tr..	7.057549
4	ROOFTOP	178 Cross Rd, Malvern SA 5061, Australia	ChIJycNiyIvOsGoRdhfq9GKnpqQ	178 Cross Rd	- 34.964960	138.611477	1	5061	OK	street_addresses	4.900099

```
#exp_data = out_geo.head(10)
##Fill the missing values with mode
out_geo['type'].fillna('street_address', inplace=True)
out_geo['type'] = out_geo['type'].apply(lambda x: str(x).split(',')[0])
```

```
out_geo['type'].unique()
```

Out[8]:

```
array(['street_address', 'transit_station', 'premise', 'political',
      'school', 'route', 'intersection', 'point_of_interest',
      'subpremise', 'real_estate_agency', 'university', 'travel_agency',
      'restaurant', 'supermarket', 'store', 'post_office'], dtype=object)
```

In [9]:

```
data['WeekBeginning'] = pd.to_datetime(data['WeekBeginning']).dt.date
data['WeekBeginning'][1]
```

Out[9]:

```
datetime.date(2013, 6, 30)
```

Data Aggregation

In [10]:


```

#Combine the Geolocation and main input file to get final Output File.
data= pd.merge(data,out_geo,how='left',left_on = 'StopName',right_on =
'input_string')
data.head(5)
data.shape

```

Tri plD	Ro ut el D	Stop ID	Sto pN ame	We ekB egi nni ng	Nu mb erO fBo ard ing s	acc ura cy	for ma tte d_a ddr ess	goo gle _pl ace _id	inp ut_ stri ng	lati tud e	lon git ude	nu mb er_ of_ res ults	pos tco de	stat us	typ e	dist _fr om_ ce ntr e	
0	23 63 1	100	141 56	181 Cro ss Rd	201 3- 06- 30	1	RO OF TO P	181 Cro ss Rd, We stb our ne Par k SA 504 1, Aus tral ia	Chl JKT 719 rbP sGo RV Hkl y- Oy k	181 Cro ss Rd	- 34. 966 656	138 .59 214 8	1	504 1	OK	str eet _ad dres s	5.1 809 61

1	23 63 1	100	141 44	177 Cro ss Rd	201 3- 06- 30	1	RO OF TO P	177 Cro ss Rd, We stb our ne Par k SA 504 1, Aus tral ia	Chl J- VFZ 87b PsG oRy fVg C5 qb PpE	177 Cro ss Rd	- 34. 966 607	138 .59 230 1	1	504 1	OK	str et _ad dres s	5.1 725 25
2	23 63 2	100	141 32	175 Cro ss Rd	201 3- 06- 30	1	RO OF TO P	175 Cro ss Rd, We stb our ne Par k SA 504 1, Aus tral ia	Chl Jlztl irb PsG oR 38K Rk7 6kP FI	175 Cro ss Rd	- 34. 966 758	138 .59 271 5	1	504 1	OK	str et _ad dres s	5.1 807 09
3	23 63 3	100	122 66	Zon e A Arn dal e Int erc han ge	201 3- 06- 30	2	GE OM ETR IC_ CE NT ER	Zon e A Arn dal e Int erc han ge - Sou th sid e, Kilk e...	Chl Jn0 C1 hC PGs Go RI Wv Cd hF1 Rlg	Zon e A Arn dal e Int erc han ge	- 34. 875 160	138 .55 162 8	1	500 9	OK	tra nsit _st ati on	7.0 575 49
4	23 63 3	100	141 47	178 Cro ss Rd	201 3- 06- 30	1	RO OF TO P	178 Cro ss Rd, Ma lver n SA 506	Chl Jyc Niy lvO sGo Rd hfq 9G Kn	178 Cro ss Rd	- 34. 964 960	138 .61 147 7	1	506 1	OK	str et _ad dres s	4.9 000 99

								1, Aus tral ia	pq 0								
--	--	--	--	--	--	--	--	-------------------------	---------	--	--	--	--	--	--	--	--

Out[10]:

(10857234, 17)

In [11]:

```
#Columns to keep for further analysis
col = ['TripID', 'RouteID', 'StopID', 'StopName',
'WeekBeginning', 'NumberOfBoardings',
'latitude', 'longitude', 'postcode', 'type', 'dist_from_centre']
data = data[col]
```

In [12]:

```
##saving the final dataset
#data.to_csv('Weekly_Boarding.csv',index=False)
```

Aggregate the Data According to Weeks and Stop names

- **NumberOfBoardings_sum** Number of Boardings within particular week for each Bus stop
- **NumberOfBoardings_count** Number of times data is recorded within week
- **NumberOfBoardings_max** Maximum number of boarding done at single time within week

In [13]:

```
grouped = data.groupby(['StopName', 'WeekBeginning', 'type'])
#grouped.head()
```

In [14]:

```
# st_week_grp1 =
pd.DataFrame(data.groupby(['StopName', 'WeekBeginning', 'type']).agg({'NumberOfBoardings': ['sum', 'count']})).reset_index()
grouped =
data.groupby(['StopName', 'WeekBeginning', 'type']).agg({'NumberOfBoardings': ['sum', 'count', 'max']})
grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]
```

In [15]:

```
grouped.head(10)
grouped.columns
```

Out[15]:

			NumberOfBoardings_sum	NumberOfBoardings_count	NumberOfBoardings_max
StopName	WeekBeginning	type			
1 Anzac Hwy	2013-06-30	street_address	1003	378	51
	2013-07-07	street_address	783	360	28
	2013-07-14	street_address	843	343	45
	2013-07-21	street_address	710	356	28
	2013-07-28	street_address	898	379	41
	2013-08-04	street_address	799	378	40
	2013-08-11	street_address	1012	358	71
	2013-08-18	street_address	793	333	41
	2013-08-25	street_address	897	354	45
	2013-09-01	street_address	1368	431	59

Out[15]:

```
Index(['NumberOfBoardings_sum', 'NumberOfBoardings_count',
      'NumberOfBoardings_max'],
      dtype='object')
```

In [16]:

```
st_week_grp = pd.DataFrame(grouped).reset_index()
st_week_grp.shape
st_week_grp.head()
```

Out[16]:

(207864, 6)

Out[16]:

	StopName	WeekBeginning	type	NumberOfBoardings_sum	NumberOfBoardings_count	NumberOfBoardings_max
0	1 Anzac Hwy	2013-06-30	street_address	1003	378	51
1	1 Anzac Hwy	2013-07-07	street_address	783	360	28
2	1 Anzac Hwy	2013-07-14	street_address	843	343	45
3	1 Anzac Hwy	2013-07-21	street_address	710	356	28
4	1 Anzac Hwy	2013-07-28	street_address	898	379	41

In [17]:

```

st_week_grp1 =
pd.DataFrame(st_week_grp.groupby('StopName')['WeekBeginning'].count()).reset_index()
st_week_grp1.head()

```

Out[17]:

	StopName	WeekBeginning
0	1 Anzac Hwy	54
1	1 Bartels Rd	54
2	1 Botanic Rd	54
3	1 Frome Rd	54
4	1 Fullarton Rd	54

In [18]:

```

#Gathering only the Stop Name which having all 54 weeks of Dat
aa = list(st_week_grp1[st_week_grp1['WeekBeginning'] == 54]['StopName'])
aa[1:10]

```

Out[18]:

```

['1 Bartels Rd',
 '1 Botanic Rd',
 '1 Frome Rd',
 '1 Fullarton Rd',
 '1 George St',
 '1 Glen Osmond Rd',
 '1 Goodwood Rd',
 '1 Henley Beach Rd',
 '1 Kensington Rd']

```

In [19]:

```

bb = st_week_grp[st_week_grp['StopName'].isin(aa)]
bb.head()
bb.shape

type(bb)

```

Out[19]:

	StopName	WeekBeginnin g	type	NumberOfBoa rdings_sum	NumberOfBoa rdings_count	NumberOfBoa rdings_max
0	1 Anzac Hwy	2013-06-30	street_address	1003	378	51

1	1 Anzac Hwy	2013-07-07	street_address	783	360	28
2	1 Anzac Hwy	2013-07-14	street_address	843	343	45
3	1 Anzac Hwy	2013-07-21	street_address	710	356	28
4	1 Anzac Hwy	2013-07-28	street_address	898	379	41

Out[19]:

(175446, 6)

Out[19]:

pandas.core.frame.DataFrame

In [20]:

```
#removing the stoppage which are not having the data of whole 54 weeks
new_data = data[data['StopName'].isin(aa)]
new_data.shape
print("data without stopage removing: ", data.shape)
print("data, after removing stoppage not having the data of whole 54 weeks: ", new_data.shape)
```

Out[20]:

(10567931, 11)

```
data without stopage removing: (10857234, 11)
data, after removing stoppage not having the data of whole 54 weeks:
(10567931, 11)
```

In [21]:

```
new_data.head(2)
filtered_data = new_data[new_data['dist_from_centre'] <= 100]
filtered_data.shape
```

Out[21]:

	TripID	RouteID	StopID	StopName	WeekBeginning	NumberOfBoardings	latitude	longitude	postcode	type	dist_from_centre
0	23631	100	14156	181 Cross Rd	2013-06-30	1	-34.966656	138.592148	5041	street_addresses	5.180961
1	23631	100	14144	177 Cross Rd	2013-06-30	1	-34.966607	138.592301	5041	street_addresses	5.172525

Out[21]:

(10341468, 11)

In [22]:

```
data = filtered_data.copy()
data.shape
```

Out[22]:

```
(10341468, 11)
```

In [23]:

```
#No of boarding for each stopage in all weeks
#bb["StopName"].groupby(NumberOfBoardings_sum)
stopageName_with_boarding =
bb.groupby(['StopName']).agg({'NumberOfBoardings_sum': ['sum']})

#stopageName_with_boarding.columns = ["_".join(x) for x in
stopageName_with_boarding.columns.ravel()]
#stopageName_with_boarding.head()
stopageName_with_boarding =
pd.DataFrame(stopageName_with_boarding.reset_index())
```

In [24]:

```
#type(stopageName_with_boarding)
stopageName_with_boarding.columns = ["StopName",
"Total_boarding_on_the_stopage"]
#stopageName_with_boarding.shape
stopageName_with_boarding.head()
```

Out[24]:

	StopName	Total_boarding_on_the_stopage
0	1 Anzac Hwy	39429
1	1 Bartels Rd	8412
2	1 Botanic Rd	14868
3	1 Frome Rd	67458
4	1 Fullarton Rd	585

In [25]:

```
## save the aggregate data
#bb.to_csv('st_week_grp.csv', index=False)
```

Data Exploration

In [26]:

```
data.nunique()
#data.isnull().sum()
#data['WeekBeginning'].unique()
```

Out[26]:

```
TripID          39211
RouteID         616
StopID          5838
StopName        3127
WeekBeginning    54
NumberOfBoardings 359
latitude        2393
longitude       2379
postcode        138
type            8
dist_from_centre 2397
dtype: int64
```

Data Visualization

In [27]:

```
##can assign the each chart to one axes at a time
fig,axrr=plt.subplots(2,2,figsize=(15,15))

ax=axrr[0][0]
ax.set_title("No of Boardings")
data['NumberOfBoardings'].value_counts().sort_index().head(20).plot.bar(ax=axrr[0][0])

ax=axrr[0][1]
ax.set_title("WeekBeginning")
data['WeekBeginning'].value_counts().plot.area(ax=axrr[0][1])

ax=axrr[1][0]
ax.set_title("most Busiest Route")
data['RouteID'].value_counts().head(10).plot.bar(ax=axrr[1][0])

ax=axrr[1][1]
ax.set_title("least Busiest Route")
data['RouteID'].value_counts().tail(10).plot.bar(ax=axrr[1][1])
```

Out[27]:

```
Text(0.5,1,'No of Boardings')
```


Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff880af0940>
```

Out[27]:

```
Text(0.5,1,'WeekBeginning')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff709a6bb38>
```

Out[27]:

```
Text(0.5,1,'most Busiest Route')
```

Out[27]:

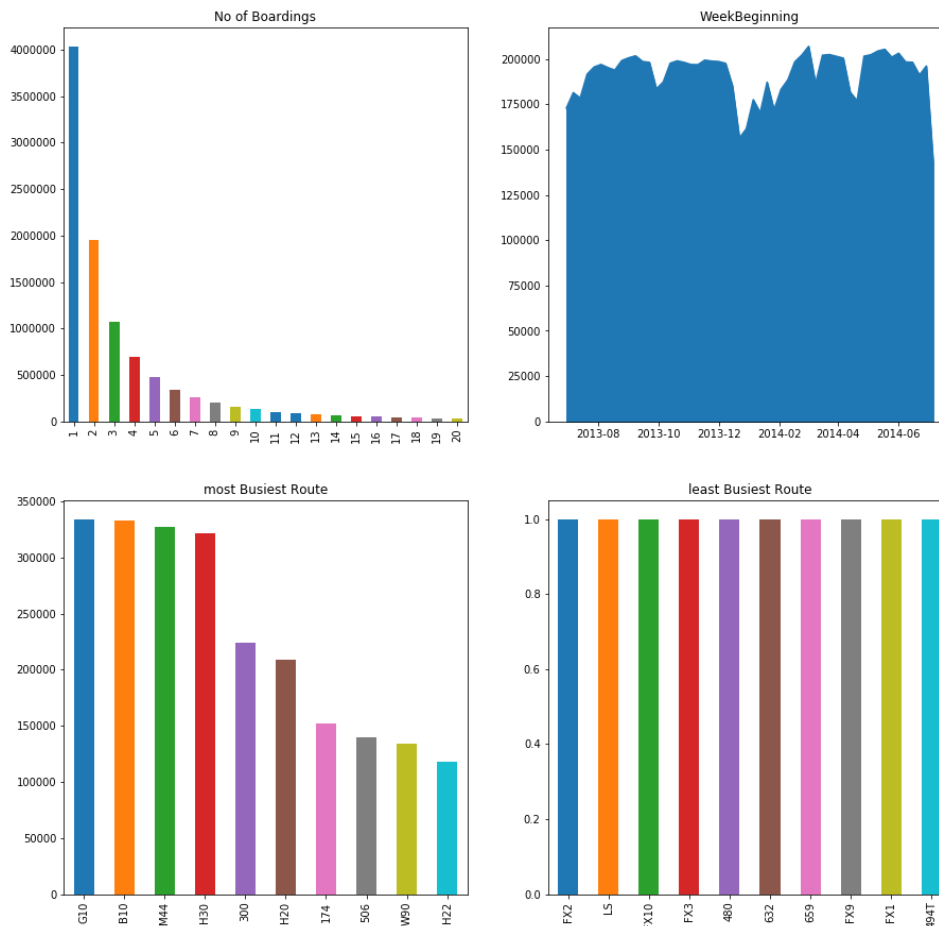
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff709a48e10>
```

Out[27]:

```
Text(0.5,1,'least Busiest Route')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff736bbafd0>
```



In [28]:

```
stopageName_with_boarding =  
stopageName_with_boarding.sort_values('Total_boarding_on_the_stopage',  
ascending = False)  
#stopage with most no of boarding  
stopageName_with_boarding.head(10)
```

Out[28]:

	StopName	Total_boarding_on_the_stopage
3054	I2 North Tce	628859
3125	X1 King William St	622099
3032	F2 Grenfell St	604149
3130	X2 King William St	583227
3021	E1 Currie St	550396
3207	Zone C Paradise Interchange	547709
3015	D1 King William St	541046
3211	Zone C Tea Tree Plaza Intercha	451960
3025	E3 Currie St	399351
3039	G3 Grenfell St	356518

In [29]:

```
#stopage with least no of boarding  
stopageName_with_boarding.tail(10)
```

Out[29]:

	StopName	Total_boarding_on_the_stopage
1845	45 McIntyre Rd	292
2318	57 Philip Hwy	281
2732	75B Frick St	275
58	109 Regency Rd	274
1633	39D Glenloth Dr	266
170	127 Lyndoch Rd	264
3086	Strathalbyn South Tce	227
1231	31 Glenroy St	221

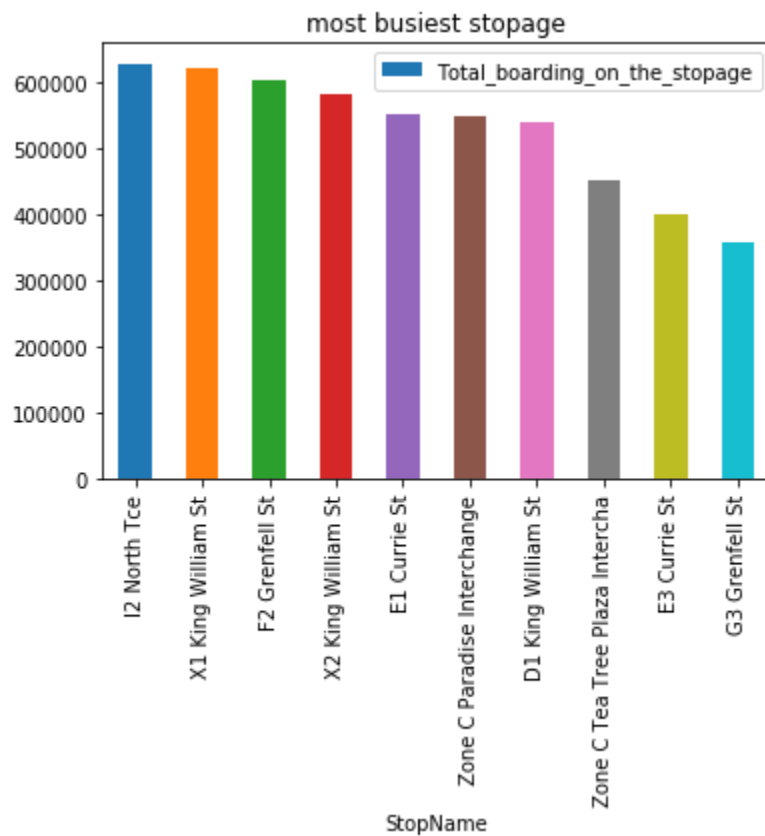
558	19 Gilles Rd	215
294	145 The Esplanade	175

In [30]:

```
ax = stopageName_with_boarding.head(10).plot.bar(x='StopName',
y='Total_boarding_on_the_stopage', rot=90)
ax.set_title("most busiest stopage")
```

Out[30]:

Text(0.5,1,'most busiest stopage')

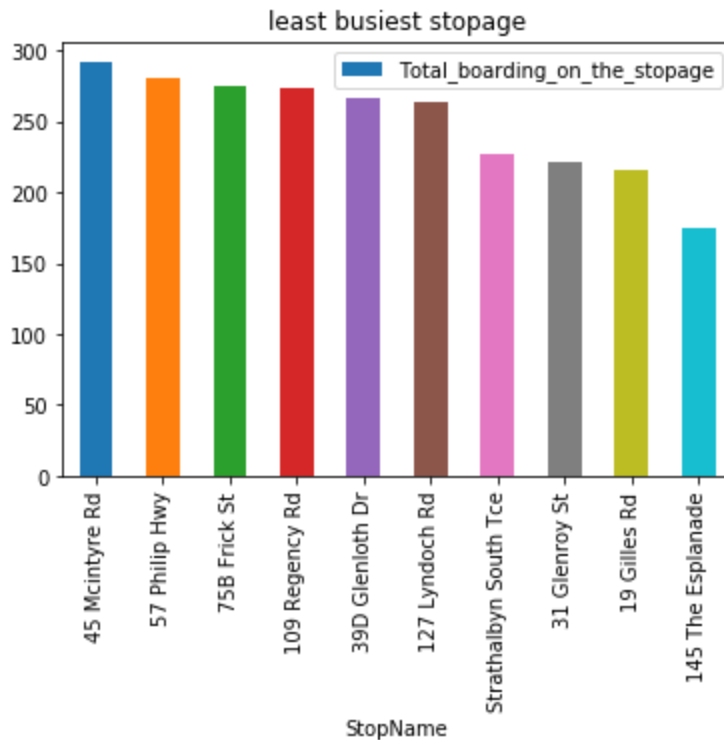


In [31]:

```
ax = stopageName_with_boarding.tail(10).plot.bar(x='StopName',
y='Total_boarding_on_the_stopage', rot=90)
ax.set_title("least busiest stopage")
```

Out[31]:

Text(0.5,1,'least busiest stopage')



In [32]:

```
data['WeekBeginning'].value_counts().mean()
```

Out[32]:

```
191508.66666666666
```

In [33]:

```
# data['dist_from_centre'].nunique()
bb_grp = data.groupby(['dist_from_centre']).agg({'NumberOfBoardings':
['sum']}).reset_index()
bb_grp.columns = bb_grp.columns.get_level_values(0)
bb_grp.head()
bb_grp.columns
bb_grp.tail()
```

Out[33]:

	dist_from_centre	NumberOfBoardings
0	0.000018	1892435
1	0.131368	167535
2	0.309089	356518
3	0.314937	1484824

4	0.326005	120061
---	----------	--------

Out[33]:

```
Index(['dist_from_centre', 'NumberOfBoardings'], dtype='object')
```

Out[33]:

	dist_from_centre	NumberOfBoardings
2392	86.471064	18905
2393	94.826409	321
2394	99.625655	1101
2395	99.665190	4373
2396	99.748995	21216

In [34]:

```
import plotly.graph_objs as go
from plotly.offline import iplot

trace0 = go.Scatter(
    x = bb_grp['dist_from_centre'],
    y = bb_grp['NumberOfBoardings'], mode = 'lines+markers', name = 'X2 King
William St')

data1 = [trace0]
layout = dict(title = 'Distance Vs Number of boarding',
              xaxis = dict(title = 'Distance from centre'),
              yaxis = dict(title = 'Number of Boardings'))
fig = dict(data=data1, layout=layout)
iplot(fig)
```

In [35]:

```
#clustering Technique// based on the distance from city centre
```

```
x = data["dist_from_centre"]
distance_10 = []
distance_10_50 = []
distance_50_100 = []
#distance_100_ = []
distance_100_more = []
total = 0
outlier = []
outlier_ = 0
for i in x:
```

```

if(i<=10):
    distance_10.append(i)
    total += 1
elif(i<=50):
    distance_10_50.append(i)
    total += 1
elif(i<=100):
    distance_50_100.append(i)
    total += 1
#elif(i>100 and i< 2000):
#distance_100_more.append(i)
#total += 1
#elif(i>2000):
#outlier.append(i)
#outlier_ += 1

```

In [36]:

```
print(outlier_)
```

0

In [37]:

```

y = len(distance_10)+len(distance_10_50)+len(distance_50_100)
#+len(distance_100_more)
#print(y)
#print(total)

```

In [38]:

```

print(total)
print("passangers, boarding the buses in the radius of 10Km from the city
center = ", (len(distance_10)/total)*100)
print("passanger, boarding the buses from the distance of 10Km to 50Km
from the city center = ", (len(distance_10_50)/total)*100)
print("passanger, boarding the buses from the distance of 50Km to 100 from
the city center = ", (len(distance_50_100)/total)*100)
#print("passanger, boarding the buses from the distance of 100Km and more
from the city center = ", (len(distance_100_more)/total)*100)

```

10341468

passangers, boarding the buses in the radius of 10Km from the city center
= 64.31275521038212

passanger, boarding the buses from the distance of 10Km to 50Km from the

```
city center = 33.16731241638035
passanger, boarding the buses from the distance of 50Km to 100 from the
city center = 2.5199323732375327
```

In [39]:

```
#busiest route on weekly basis
#data.head(10)
# st_week_grp1 =
pd.DataFrame(data.groupby(['StopName', 'WeekBeginning', 'type']).agg({'Number
OfBoardings': ['sum', 'count']})).reset_index()
grouped_route = data.groupby(['RouteID']).agg({'NumberOfBoardings':
['sum', 'max']})
grouped_route.columns = ["_".join(x) for x in
grouped_route.columns.ravel()]
```

In [40]:

```
"""grouped_route = grouped_route.head().reset_index()
type(grouped_route)
grouped_route = grouped_route.sort_values("NumberOfBoardings_sum",
ascending = True)
#stopageName_with_boarding =
stopageName_with_boarding.sort_values('Total_boarding_on_the_stopage',
ascending = False)
#stopage with most no of boarding
#stopageName_with_boarding.head(10)
#grouped_route["NumberOfBoardings_sum"] =
grouped_route["NumberOfBoardings_sum"] / 365
grouped_route.head(10)
grouped_route.shape"""
```

Out[40]:

```
'grouped_route =
grouped_route.head().reset_index()\n
type(grouped_route)\n
grouped_route =
grouped_route.sort_values("NumberOfBoardings_sum", ascending =
True)\n
#stopageName_with_boarding =
stopageName_with_boarding.sort_values(\'Total_boarding_on_the_stopage\',
ascending = False)\n
#stopage with most no of
boarding\n
#stopageName_with_boarding.head(10)\n
#grouped_route["NumberOfBoa
rdings_sum"] = grouped_route["NumberOfBoardings_sum"] /
365\n
grouped_route.head(10)\n
grouped_route.shape'
```

....

In [41]:

```
"""route_data = grouped_route[grouped_route['RouteID'] == "G10"]  
route_data.head()"""
```

Out[41]:

```
'route_data = grouped_route[grouped_route['RouteID'] ==  
"G10"]\nroute_data.head()'
```