# New Scan 9:48 PM

Scanned on: 21:48 November 10, 2022 UTC

| | |
|---|---|
| 8.6% | Overall Similarity Score |
| 9 | Results Found |
| 1103 | Total Words in Text |

| | |
|---|---|
| Identical Words | 90 |
| Words with Minor Changes | 0 |
| Paraphrased Words | 5 |
| Omitted Words | 0 |

New Scan 9:48 PM

Scanned on: 21:48 November 10, 2022 UTC

# Results

Sources that matched your submitted document.

| | |
|---|---|
| **Copyleaks Internal Database** | 3% |
| **Copyleaks Internal Database** | 2% |
| **Copyleaks Internal Database** | 1% |
| **Copyleaks Internal Database** | 1% |
| **Copyleaks Internal Database** | 1% |
| **Copyleaks Internal Database** | 1% |
| **Copyleaks Internal Database** | 1% |
| **python - getting better results with sklearn's Kmeans - Stac...**<br>*https://stackoverflow.com/questions/69939793/getting-better-results-with-...* | 1% |

IDENTICAL

Identical matches are one to one exact wording in the text.

MINOR CHANGES

Nearly identical with different form, ie "slow" becomes "slowly".

PARAPHRASED

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

New Scan 9:48 PM

Scanned on: 21:48 November 10, 2022 UTC

| | |
|---|---|
| **python - How to cluster using just one feature - Stack Overfl...**<br>*https://stackoverflow.com/questions/71540843/how-to-cluster-using-just-...* | 1% |

# Scanned Text

Your text is highlighted according to the matched content in the results above.

● IDENTICAL ● MINOR CHANGES ● PARAPHRASED

**EXPLORATORY DATA ANALYSIS ON AIRPLANE ROUTE DATASETS**

```
%%capture
!pip install networkx==2.3
```

```
import pandas as pd
import networkx as nx
#%matplotlib notebook
import matplotlib.pyplot as plt
import os
import operator
import warnings
warnings.filterwarnings('ignore')
```

```
G_df = pd.read_csv('/content/routes1 (2).csv')
cols_list=["Airport ID","City","Country","IATA"]
airport_df = pd.read_csv('/content/airports.csv',usecols=cols_list)
```

```
G_df.head(2)
```

```
airport_df.head(2)
```

```
G_draw = nx.from_pandas_edgelist(G_df.head(1000), 'Source airport ', 'Destination airport
',create_using=nx.DiGraph())
```

```
plt.figure(figsize=(12,8))
nx.draw(G_draw,pos=nx.spring_layout(G_draw),with_labels=False)
```

```
G = nx.from_pandas_edgelist(G_df, 'Source airport ', 'Destination airport ',create_using=nx.DiGraph())
```

```
print(nx.info(G))
```

```
#does a route exist between every two airport? #is every airport reachable from every other airport?
nx.is_strongly_connected(G), nx.is_connected(G.to_undirected())
```

**WEAKLY AND STRONGLY CONNECTED COMPONENTS**

```
#How many nodes are in the largest (in terms of nodes) weakly connected component?
wccs = nx.weakly_connected_components(G)
x=len(max(wccs, key=len))
print(x)
```

```
#How many nodes are in the largest (in terms of nodes) strongly connected component?
sccs = nx.strongly_connected_components(G)
x=len(max(sccs, key=len))
print(x)
```

**Average path length**

```
scc_subs = (G.subgraph(c) for c in nx.strongly_connected_components(G))
G_sc = max(scc_subs, key=len) #the largest strongly connected subgraph
shortest_sc=nx.average_shortest_path_length(G_sc)
shortest_sc
```

```
wcc_subs = (G.subgraph(c) for c in nx.weakly_connected_components(G))
G_wc = max(wcc_subs, key=len) #the largest weakly connected subgraph
shortest_wc=nx.average_shortest_path_length(G_wc)
shortest_wc
```

**Diameter and radius**

1. The diameter represents the greatest possible no of airports between any two airports .
2. The radius represents an airport from which every other airport is at a minimum no of airports apart

```
diameter=nx.diameter(G_sc)
diameter
```

```
radius=nx.radius(G_sc)
radius
```

**Periphery and Center**

```
per=nx.periphery(G_sc)
per
```

```
airport_df.loc[airport_df['IATA'].isin(per)]
```

```
cen=nx.center(G_sc)
cen
```

```
airport_df.loc[airport_df['IATA'].isin(cen)]
```

**Which node in G_sc is connected to the most other nodes by a shortest path of length equal to the radius of G_sc**

```
d = radius
max_count = -1
result_node = None
for node in cen:
count = 0
sp = nx.shortest_path_length(G_sc, node)
for key, value in sp.items():
if value == radius:
count += 1
if count > max_count:
result_node = node
max_count = count
```

```
result_node, max_count
```

```
airport_df.loc[airport_df['IATA'] == result_node]
```

**Indegree and Outdegree**

**What are the top and bottom 5 airports with most incoming flights?**

```
in_deg=nx.in_degree_centrality(G_sc)
```

```
top5=sorted(in_deg.items(), key=operator.itemgetter(1),reverse=True)[:5]
```

```
l=[]
for i,j in top5:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

```
bot5=sorted(in_deg.items(), key=operator.itemgetter(1))[:5]
l=[]
for i,j in bot5:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**What are the top and bottom 5 airports with most outgoing flights?**

```
out_deg=nx.out_degree_centrality(G_sc)
```

```
top5=sorted(out_deg.items(), key=operator.itemgetter(1),reverse=True)[:5]
top5
```

```
l=[]
for i,j in top5:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

```
bot5=sorted(out_deg.items(), key=operator.itemgetter(1))[:5]
bot5
```

```
l=[]
for i,j in bot5:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**Closeness Centrality**

**Which airports will allow you to reach all other airports with the lowest average number of airports in between?**

```
closeness = nx.closeness_centrality(G_sc, wf_improved=True)
```

```
close=sorted(closeness.items(), key=operator.itemgetter(1),reverse=True)[:5]
l=[]
for i,j in close:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**Which airports will make you reach all other airports with the highest average number of airports in between?**

```
close=sorted(closeness.items(), key=operator.itemgetter(1))[:18]
l=[]
for i,j in close:
l.append(i)
```

```
airport_df.loc[airport_df['IATA'].isin(l)]
```

**Betweenness Centrality**

**Which airports often act as bridges between other pairs of airports?**

```
betweeness = nx.betweenness_centrality(G_sc, normalized=True)
close=sorted(betweeness.items(), key=operator.itemgetter(1),reverse=True)[:5]
l=[]
for i,j in close:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**PageRank**

**5 airports with highest and lowest pagerank?**

Important airports based on highest number of connections within shortest path

```
pr = nx.pagerank(G_sc, alpha=0.85)
pager=sorted(pr.items(), key=operator.itemgetter(1),reverse=True)[:5]
l=[]
for i,j in pager:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

```
pager=sorted(pr.items(), key=operator.itemgetter(1))[:5]
l=[]
for i,j in pager:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**Identifying hubs and authorities**

```
hits = nx.hits(G_sc)
```

```
hubs=sorted(hits[0].items(), key=operator.itemgetter(1))[:5]
l=[]
for i,j in hubs:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

```
auth=sorted(hits[1].items(), key=operator.itemgetter(1))[:5]
l=[]
for i,j in auth:
l.append(i)
airport_df.loc[airport_df['IATA'].isin(l)]
```

**FLIGHT FARE PREDICTION**

```
import numpy as np
import pandas as pd
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from prettytable import PrettyTable

t_df = pd.read_csv("/content/routes1 (2).csv")

t_df.columns

t_df.info()

t_df.describe()

t_df.isnull().head()

t_df.isnull().sum()

t_df.dropna(inplace = True)

t_df[t_df.duplicated()].head()

t_df.drop_duplicates(keep='first',inplace=True)
t_df.head()

t_df.shape

t_df["Airline"].unique()

sns.catplot(y = "Price", x = "Airline", data = t_df.sort_values("Price", ascending = False), kind="boxen", height
= 8, aspect = 3)
plt.show()

sns.catplot(y = "Price", x = "Source", data = t_df.sort_values("Price", ascending = False), kind="violin", height
= 4, aspect = 3)
plt.show()
```

```
sns.catplot(y = "Price", x = "Destination", data = t_df.sort_values("Price", ascending = False), kind="box",
height = 4, aspect = 3)
plt.show()
```

```
t_df.head()
```

```
t_df["Journey_day"] = t_df['Date_of_Journey'].str.split('/').str[0].astype(int)
t_df["Journey_month"] = t_df['Date_of_Journey'].str.split('/').str[1].astype(int)
t_df.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

Plotting Bar chart for Months (Duration) vs Number of Flights

```
plt.figure(figsize = (10, 5))
plt.title('Count of flights month wise')
ax=sns.countplot(x = 'Journey_month', data = t_df)
plt.xlabel('Month')
plt.ylabel('Count of flights')
for p in ax.patches:
ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1), va='bottom', color= 'black')
```

Plotting Bar chart for Types of Airline vs Number of Flights

```
plt.figure(figsize = (20,5))
plt.title('Count of flights with different Airlines')
ax=sns.countplot(x = 'Airline', data =t_df)
plt.xlabel('Airline')
plt.ylabel('Count of flights')
plt.xticks(rotation = 45)
for p in ax.patches:
ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1), va='bottom', color= 'black')
```

```
plt.figure(figsize = (15,4))
plt.title('Price VS Airlines')
plt.scatter(t_df['Airline'], t_df['Price'])
plt.xticks
plt.xlabel('Airline')
plt.ylabel('Price of ticket')
plt.xticks(rotation = 90)
```

Plotting Ticket Prices VS Airlines

**CLUSTERING BASED ON AIR TRAFFIC ON AIRLINES**

```
df = pd.read_csv("/content/Air_Traffic_Passenger_Statistics.csv")
df.head()
```

```
plt.figure(figsize = (15,15))
sns.countplot(x = "Operating Airline", palette = "Set3",data = df)
plt.xticks(rotation = 90)
plt.ylabel("Number of fights held")
plt.show()
plt.figure(figsize = (15,15))
sns.countplot(x = "GEO Region", palette = "Set3",data = df)
plt.xticks(rotation = 90)
plt.ylabel("Number of fights held")
plt.show()


flight_count = df["Operating Airline"].value_counts()
flight_count.sort_index(inplace=True)
traveler_count = df.groupby("Operating Airline").sum()["Passenger Count"]
traveler_count.sort_index(inplace=True)
from sklearn.preprocessing import scale
x = flight_count.values
y = traveler_count.values
plt.figure(figsize = (10,10))
plt.scatter(x, y)
plt.xlabel("No of Flights held")
plt.ylabel("No of Passengers")
for i, txt in enumerate(flight_count.index.values):
plt.annotate(txt, (x[i], y[i]))
plt.show()


z = np.array(list(zip(x,y)))
km = KMeans(n_clusters=6)
km.fit(z)
y_km = km.predict(z)
plt.figure(figsize = (14,14))
plt.xlabel("No of Flights held")
plt.ylabel("No of Passengers")
plt.scatter(z[:, 0], z[:, 1], c=y_km, s=300, cmap='Set1')
for i, txt in enumerate(flight_count.index.values):
plt.annotate(txt, (z[i,0], z[i,1]), size = 7)
plt.show()
```