Virtual competition

December 2, 2024

1 Virtual Competition

[420]:
```
import pandas as pd
import numpy as np
```

Consider the following Python dictionary `data` and Python list `labels`:

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

1. Create a DataFrame `df` from this dictionary `data` which has the index `labels`.

[421]:
```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat',␣
↪'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no',␣
↪'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(data,labels)
df
```

[421]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

2. Display a summary of the basic information about this DataFrame and its data (*hint: there is a single method that can be called on the DataFrame*).

[422]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   animal    10 non-null     object
 1   age       8 non-null      float64
 2   visits    10 non-null     int64
 3   priority  10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

3. Return the first 3 rows of the DataFrame `df`.

[423]:
```
df.head(3)
```

[423]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no

4. Display the 'animal' and 'age' columns from the DataFrame `df`

[424]:
```
df[["animal", "age"]]
```

[424]:

	animal	age
a	cat	2.5
b	cat	3.0
c	snake	0.5
d	dog	NaN
e	dog	5.0
f	cat	2.0
g	snake	4.5
h	cat	NaN
i	dog	7.0
j	dog	3.0

5. Display the data in rows `[3, 4, 8]` *and* in columns '['animal', 'age']'

[425]:
```
df[["animal","age"]].iloc[[2,3,7]]
```

[425]:

	animal	age
c	snake	0.5
d	dog	NaN
h	cat	NaN

6. Select only the rows where the number of visits is greater than 3.

[426]:
```
df[df["visits"]>3]
```

[426]:
```
Empty DataFrame
Columns: [animal, age, visits, priority]
Index: []
```

7. Select the rows where the age is missing, i.e. it is `NaN`.

[427]:
```
df[df["age"].isna()]
```

[427]:
```
  animal  age  visits priority
d    dog  NaN       3      yes
h    cat  NaN       1      yes
```

8. Select the rows where the animal is a cat *and* the age is less than 3.

[428]:
```
df[(df["animal"]=="cat") & (df["age"]<3)]
```

[428]:
```
  animal  age  visits priority
a    cat  2.5       1      yes
f    cat  2.0       3       no
```

9. Select the rows where the age is between 2 and 4 (inclusive)

[429]:
```
df[(df["age"]>=2) & (df["age"]<=4)]
```

[429]:
```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
f    cat  2.0       3       no
j    dog  3.0       1       no
```

10. Change the age in row 'f' to 1.5.

[430]:
```
print(df.loc['f'])
df["age"].loc['f'] = 1.5
print("\n", df.loc['f'])
```

```
animal      cat
age         2.0
visits        3
priority     no
Name: f, dtype: object

 animal      cat
age         1.5
visits        3
priority     no
Name: f, dtype: object
```

```
/var/folders/jw/jny11qx97778yjjc7534g4bm0000gn/T/ipykernel_3107/1448079181.py:2:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df["age"].loc['f'] = 1.5
/var/folders/jw/jny11qx97778yjjc7534g4bm0000gn/T/ipykernel_3107/1448079181.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["age"].loc['f'] = 1.5
```

11. Calculate the sum of all visits in `df` (i.e. the total number of visits).

[431]:
```
df["visits"].sum()
```

[431]:
```
19
```

12. Calculate the mean age for each different animal in `df`.

[432]:
```
df.groupby("animal")["age"].mean()
```

[432]:
```
animal
cat      2.333333
dog      5.000000
snake    2.500000
Name: age, dtype: float64
```

13. Append a new row 'k' to `df` with your choice of values for each column. Then delete that row to return the original DataFrame.

[433]:
```
new_values = ['lion', 5, 4,'yes']
df.loc['k'] = new_values
df
```

[433]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	1.5	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no
k	lion	5.0	4	yes

14. Count the number of each type of animal in `df`.

[434]:
```
df.groupby("animal")["animal"].count()
```

[434]:
```
animal
cat      4
dog      4
lion     1
snake    2
Name: animal, dtype: int64
```

15. Sort `df` first by the values in the 'age' in *decending* order, then by the value in the 'visits' column in *ascending* order (so row `i` should be first, and row `d` should be last).

[435]:
```
df.sort_values("age", ascending=False)
```

[435]:

	animal	age	visits	priority
i	dog	7.0	2	no
e	dog	5.0	2	no
k	lion	5.0	4	yes
g	snake	4.5	1	no
b	cat	3.0	3	yes
j	dog	3.0	1	no
a	cat	2.5	1	yes
f	cat	1.5	3	no
c	snake	0.5	2	no
d	dog	NaN	3	yes
h	cat	NaN	1	yes

16. The 'priority' column contains the values 'yes' and 'no'. Replace this column with a column of boolean values: 'yes' should be `True` and 'no' should be `False`.

[436]:
```
for i in range(len(df)):
    if df["priority"].iloc[i] == 'yes':
        df["priority"].iloc[i] = True
    else:
```

```
        df["priority"].iloc[i] = False
df
```

```
/var/folders/jw/jny11qx97778yjjc7534g4bm0000gn/T/ipykernel_3107/4105441080.py:3:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df["priority"].iloc[i] = True
/var/folders/jw/jny11qx97778yjjc7534g4bm0000gn/T/ipykernel_3107/4105441080.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["priority"].iloc[i] = True
/var/folders/jw/jny11qx97778yjjc7534g4bm0000gn/T/ipykernel_3107/4105441080.py:3:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df["priority"].iloc[i] = True
```

```
    df["priority"].iloc[i] = True
```

[436]:
```
  animal  age  visits priority
a    cat  2.5       1     True
b    cat  3.0       3     True
c  snake  0.5       2    False
d    dog  NaN       3     True
e    dog  5.0       2    False
f    cat  1.5       3    False
g  snake  4.5       1    False
h    cat  NaN       1     True
i    dog  7.0       2    False
j    dog  3.0       1    False
k   lion  5.0       4     True
```

17. In the 'animal' column, change the 'snake' entries to 'python'.

[437]:
```
print(df)
for i in range(len(df)):
    if (df["animal"].iloc[i] == 'snake'):
        df["animal"].iloc[i] = 'python'
print("\n", df)
```

```
  animal  age  visits priority
a    cat  2.5       1     True
b    cat  3.0       3     True
c  snake  0.5       2    False
d    dog  NaN       3     True
e    dog  5.0       2    False
f    cat  1.5       3    False
g  snake  4.5       1    False
h    cat  NaN       1     True
i    dog  7.0       2    False
j    dog  3.0       1    False
k   lion  5.0       4     True

    animal  age  visits priority
a     cat  2.5       1     True
b     cat  3.0       3     True
c  python  0.5       2    False
d     dog  NaN       3     True
e     dog  5.0       2    False
f     cat  1.5       3    False
g  python  4.5       1    False
h     cat  NaN       1     True
i     dog  7.0       2    False
j     dog  3.0       1    False
k    lion  5.0       4     True
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["animal"].iloc[i] = 'python'
```

18. Load the ny-flights dataset to Python

[438]:
```
df_flights = pd.read_csv("ny-flights.csv")
df_flights
```

[438]:

	fl_date	unique_carrier	airline_id	tail_num	fl_num	origin
0	2014-01-01 00:00:00	AA	19805	N338AA	1	JFK
1	2014-01-01 00:00:00	AA	19805	N335AA	3	JFK
2	2014-01-01 00:00:00	AA	19805	N327AA	21	JFK
3	2014-01-01 00:00:00	AA	19805	N3EHAA	29	LGA
4	2014-01-01 00:00:00	AA	19805	N319AA	117	JFK
…	…	…	…	…	…	…
20812	2014-01-31 00:00:00	UA	19977	N54711	1253	ROC
20813	2014-01-31 00:00:00	UA	19977	N77525	1429	LGA
20814	2014-01-31 00:00:00	UA	19977	N37293	1456	LGA
20815	2014-01-31 00:00:00	UA	19977	N24729	1457	LGA
20816	2014-01-31 00:00:00	MQ	20398	N609MQ	3699	BUF

	dest	dep_time	dep_delay	arr_time	arr_delay	cancelled
0	LAX	914.0	14.0	1238.0	13.0	0.0
1	LAX	1157.0	-3.0	1523.0	13.0	0.0
2	LAX	1902.0	2.0	2224.0	9.0	0.0
3	PBI	722.0	-8.0	1014.0	-26.0	0.0
4	LAX	1347.0	2.0	1706.0	1.0	0.0
…	…	…	…	…	…	…
20812	ORD	801.0	-4.0	908.0	4.0	0.0
20813	CLE	1522.0	-10.0	1649.0	-31.0	0.0
20814	IAH	719.0	-6.0	1006.0	-20.0	0.0
20815	IAH	852.0	7.0	1156.0	-6.0	0.0
20816	ORD	1208.0	-12.0	1251.0	-19.0	0.0

	arr	dep
0	2014-01-01 12:38:00	2014-01-01 09:14:00
1	2014-01-01 15:23:00	2014-01-01 11:57:00
2	2014-01-01 22:24:00	2014-01-01 19:02:00
3	2014-01-01 10:14:00	2014-01-01 07:22:00
4	2014-01-01 17:06:00	2014-01-01 13:47:00
…	…	…
20812	2014-01-31 09:08:00	2014-01-31 08:01:00
20813	2014-01-31 16:49:00	2014-01-31 15:22:00
20814	2014-01-31 10:06:00	2014-01-31 07:19:00
20815	2014-01-31 11:56:00	2014-01-31 08:52:00
20816	2014-01-31 12:51:00	2014-01-31 12:08:00

```
[20817 rows x 14 columns]
```

19. Which airline ID is present maximum times in the dataset

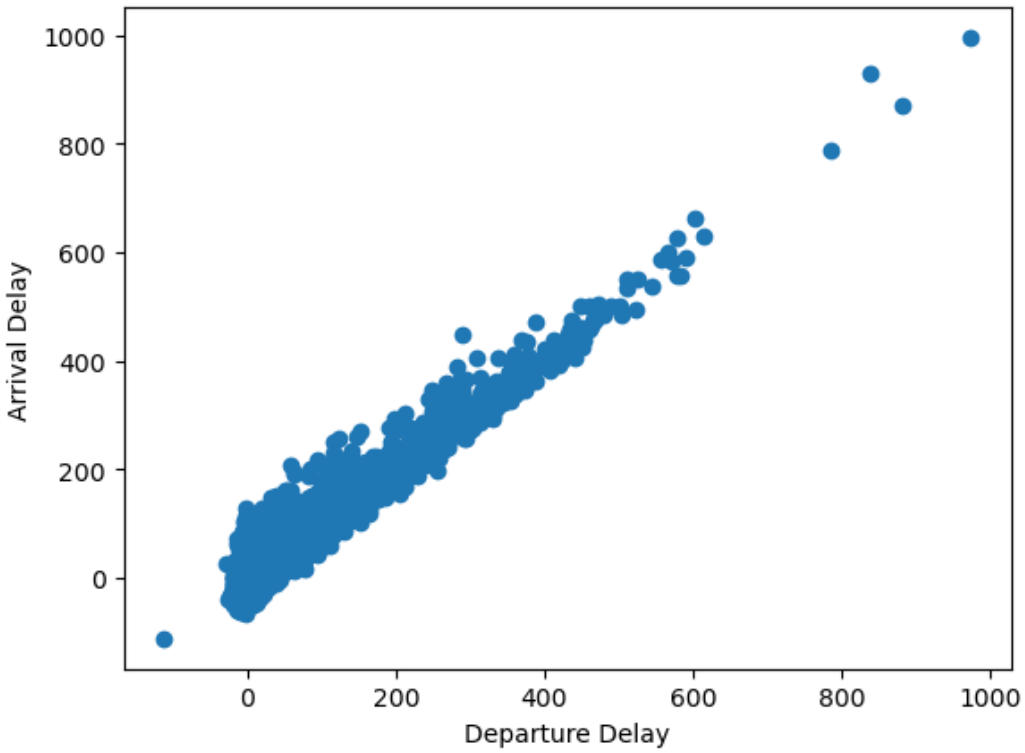
[439]:
```
airline = df_flights.groupby("airline_id")["airline_id"].count().
  ↪sort_values(ascending=False).head(1)
print("Airline ID: ", airline.index[0])
print("Count: ", airline.iloc[0])
```

```
Airline ID:  20409
Count:  4902
```

20. Draw a plot between dep_delay and arr_delay

[440]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[441]:
```
x_points = df_flights["dep_delay"]
y_points = df_flights["arr_delay"]
plt.scatter(x_points, y_points)
plt.xlabel("Departure Delay")
plt.ylabel("Arrival Delay")
plt.show()
```



2 ny_flights Dataset Tasks:

1. What is the shape and size of the dataset?

[442]:
```
print("Number of rows: ", df_flights.shape[0])
print("\nNumber of columns: ", df_flights.shape[1])
```

```
Number of rows:  20817

Number of columns:  14
```

[443]:
```
df_flights.size
```

[443]:
```
291438
```

2. what is the column-wise information of the dataset?

[444]:
```
df_flights.describe()
```

[444]:

	airline_id	fl_num	dep_time	dep_delay	arr_time	\
count	20817.000000	20817.000000	18462.000000	18462.000000	18412.000000	
mean	20109.614882	1826.098813	1319.991713	22.772127	1493.698566	
std	370.715209	1548.188325	479.999940	59.766152	518.851657	
min	19393.000000	1.000000	1.000000	-112.000000	1.000000	
25%	19790.000000	472.000000	858.000000	-4.000000	1110.750000	
50%	20355.000000	1457.000000	1336.000000	0.000000	1519.000000	
75%	20409.000000	2701.000000	1720.000000	22.000000	1923.000000	
max	21171.000000	6258.000000	2400.000000	973.000000	2400.000000	

	arr_delay	cancelled
count	18383.000000	20817.00000
mean	21.380732	0.11505
std	64.605591	0.31909
min	-112.000000	0.00000
25%	-12.000000	0.00000
50%	3.000000	0.00000
75%	28.000000	0.00000
max	996.000000	1.00000

3. Display the data types of each column

[445]:
```
df_flights.dtypes
```

[445]:
```
fl_date            object
unique_carrier     object
airline_id          int64
```

```
tail_num            object
fl_num               int64
origin              object
dest                object
dep_time           float64
dep_delay          float64
arr_time           float64
arr_delay          float64
cancelled          float64
arr                 object
dep                 object
dtype: object
```

4. Show all the flights from 'JFK' to 'LAX' which departed on time.

[446]:
```
df_flights[(df_flights["origin"]=='JFK') & (df_flights["dest"]=='LAX') &␣
↪(df_flights["dep_delay"]==0)]
```

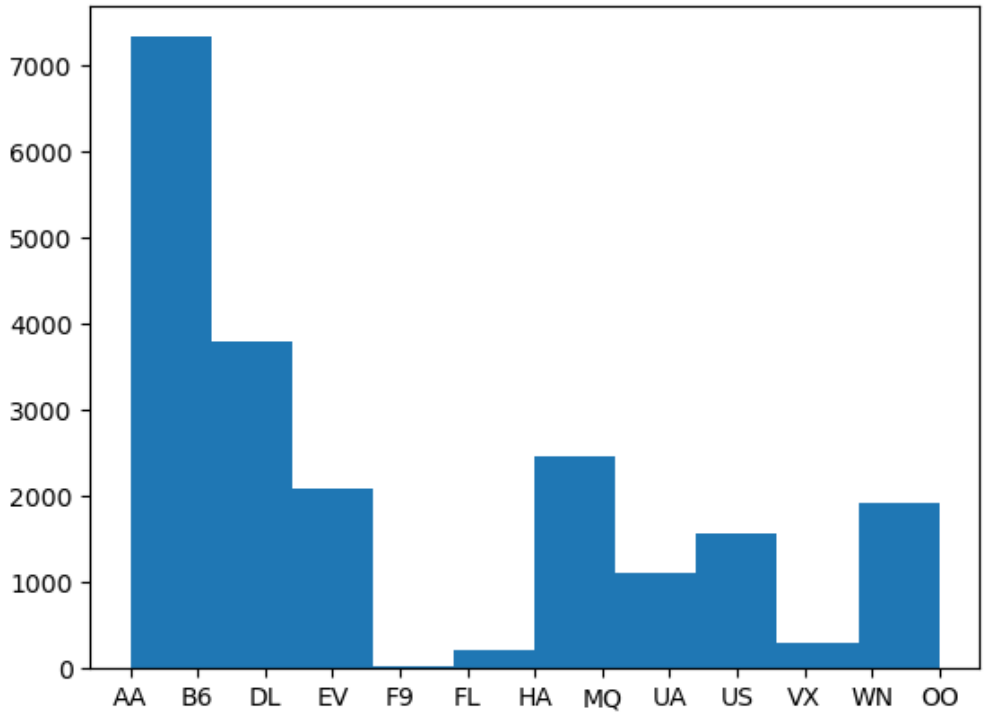
[446]:
```
                   fl_date unique_carrier  airline_id tail_num  fl_num origin  \
537    2014-01-01 00:00:00             UA       19977   N502UA     841    JFK
3421   2014-01-06 00:00:00             VX       21171   N640VA     411    JFK
3799   2014-01-06 00:00:00             AA       19805   N332AA     185    JFK
4055   2014-01-06 00:00:00             DL       19790   N722TW     423    JFK
4943   2014-01-08 00:00:00             UA       19977   N568UA     274    JFK
…                      …              …           …        …      …    …
18325  2014-01-28 00:00:00             DL       19790   N188DN     427    JFK
18340  2014-01-28 00:00:00             DL       19790   N713TW     477    JFK
18653  2014-01-28 00:00:00             B6       20409   N524JB     423    JFK
19575  2014-01-30 00:00:00             UA       19977   N502UA     274    JFK
20780  2014-01-31 00:00:00             UA       19977   N597UA     314    JFK

      dest  dep_time  dep_delay  arr_time  arr_delay  cancelled  \
537    LAX    1445.0        0.0    1816.0        3.0        0.0
3421   LAX    1300.0        0.0    1607.0      -13.0        0.0
3799   LAX    2135.0        0.0      54.0       -1.0        0.0
4055   LAX    1200.0        0.0    1456.0      -25.0        0.0
4943   LAX    1129.0        0.0    1428.0      -11.0        0.0
…    …        …          …        …          …          …
18325  LAX    2015.0        0.0    2327.0      -13.0        0.0
18340  LAX    1645.0        0.0    1955.0      -24.0        0.0
18653  LAX    1635.0        0.0    1937.0      -26.0        0.0
19575  LAX    1129.0        0.0    1455.0       16.0        0.0
20780  LAX    2025.0        0.0    2351.0        9.0        0.0

                       arr                  dep
537    2014-01-01 18:16:00  2014-01-01 14:45:00
3421   2014-01-06 16:07:00  2014-01-06 13:00:00
3799   2014-01-06 00:54:00  2014-01-06 21:35:00
```

```
4055   2014-01-06 14:56:00  2014-01-06 12:00:00
4943   2014-01-08 14:28:00  2014-01-08 11:29:00
…                      …                    …
18325  2014-01-28 23:27:00  2014-01-28 20:15:00
18340  2014-01-28 19:55:00  2014-01-28 16:45:00
18653  2014-01-28 19:37:00  2014-01-28 16:35:00
19575  2014-01-30 14:55:00  2014-01-30 11:29:00
20780  2014-01-31 23:51:00  2014-01-31 20:25:00

[61 rows x 14 columns]
```

5. what is the distribution of the carriers

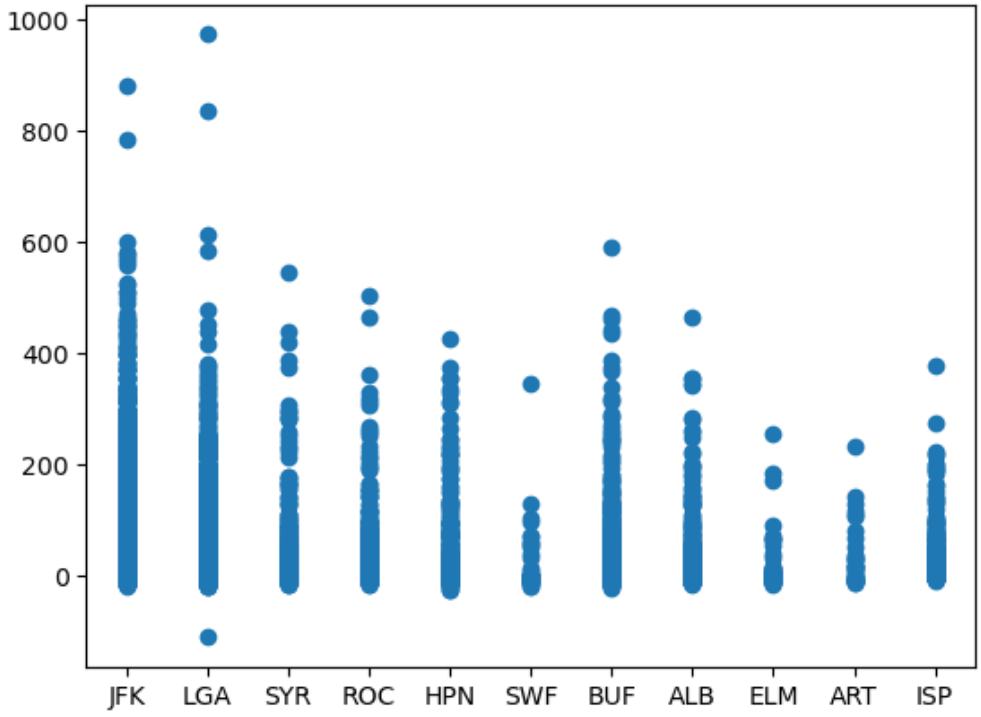
[460]:
```
# Univariate Analysis
plt.hist(df_flights["unique_carrier"])
```

[460]:
```
(array([7329., 3803., 2080.,   33.,  206., 2463., 1106., 1568.,  301.,
        1928.]),
 array([ 0. ,  1.2,  2.4,  3.6,  4.8,  6. ,  7.2,  8.4,  9.6, 10.8, 12. ]),
 <BarContainer object of 10 artists>)
```



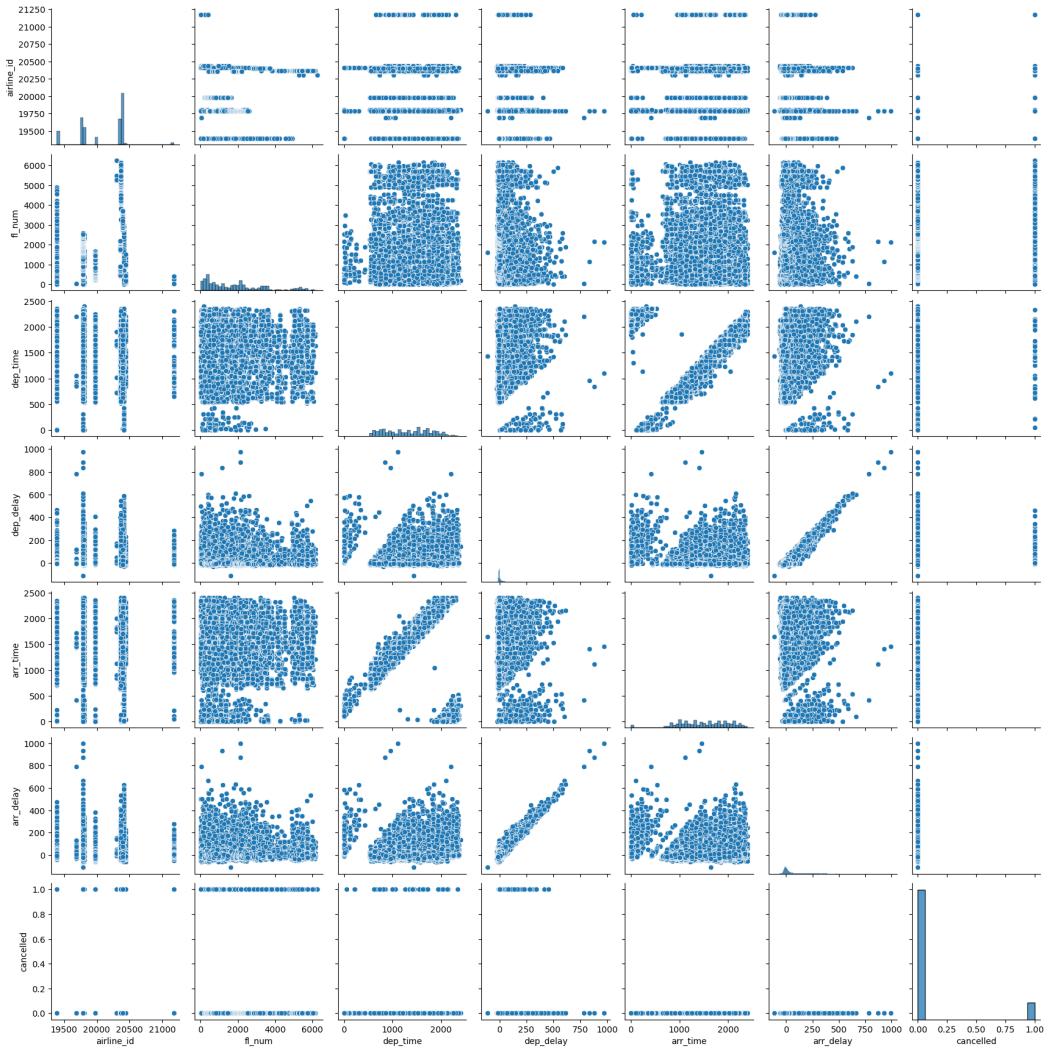
[458]:
```
# Bivariate Analysis
plt.scatter(df_flights["origin"],df_flights["dep_delay"])
```

[458]: <matplotlib.collections.PathCollection at 0x327f2aea0>



[449]:
```
# Multivariate Analysis
sns.pairplot(df_flights)
```

[449]: <seaborn.axisgrid.PairGrid at 0x31dd5f950>



3 Diabetes Dataset EDA:

[450]:
```
df_diabetes = pd.read_csv("diabetes_model.csv")
df_diabetes
```

[450]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	5	77	82	41	42	35.8	
1	9	122	56	0	0	33.3	
2	0	113	76	0	0	33.3	
3	1	139	62	41	480	40.7	
4	10	161	68	23	132	25.5	
..	…	…	…	…	…	…	

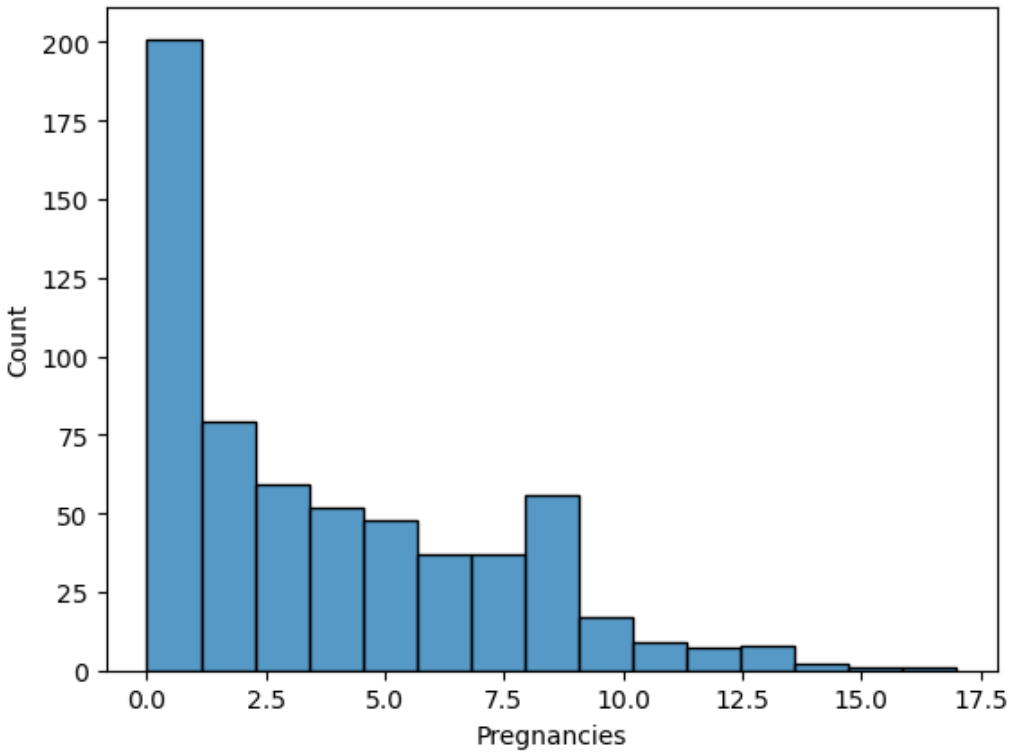
```
609            4      114             64              0        0  28.9
610            2      175             88              0        0  22.9
611            3      121             52              0        0  36.0
612            7      136             74             26      135  26.0
613            4      156             75              0        0  48.3

     DiabetesPedigreeFunction  Age  Outcome
0                       0.156   35        0
1                       1.114   33        1
2                       0.278   23        1
3                       0.536   21        0
4                       0.326   47        1
..                        …  …       …
609                     0.126   24        0
610                     0.326   22        0
611                     0.127   25        1
612                     0.647   51        0
613                     0.238   32        1

[614 rows x 9 columns]
```

3.1 Univariate Analysis:

- Analyze individual variables (e.g., histograms, box plots).

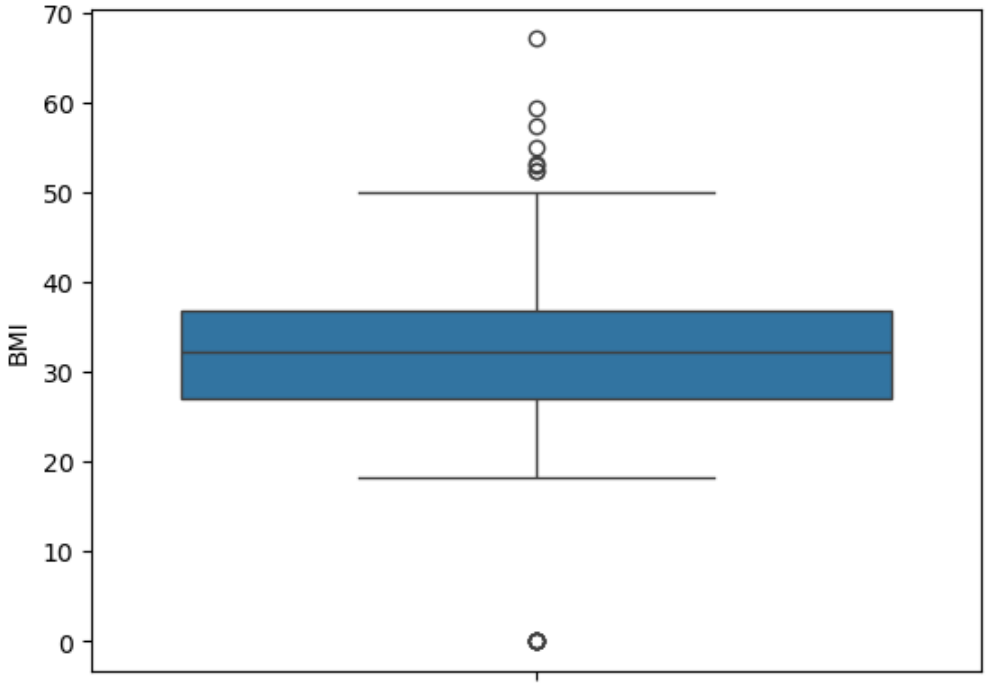
[451]:
```
# histogram of count of pregnancies
sns.histplot(df_diabetes["Pregnancies"])
```

[451]:
```
<Axes: xlabel='Pregnancies', ylabel='Count'>
```



[452]:
```
# Box plot of all the BMIs
sns.boxplot(df_diabetes["BMI"])
```

[452]: <Axes: ylabel='BMI'>

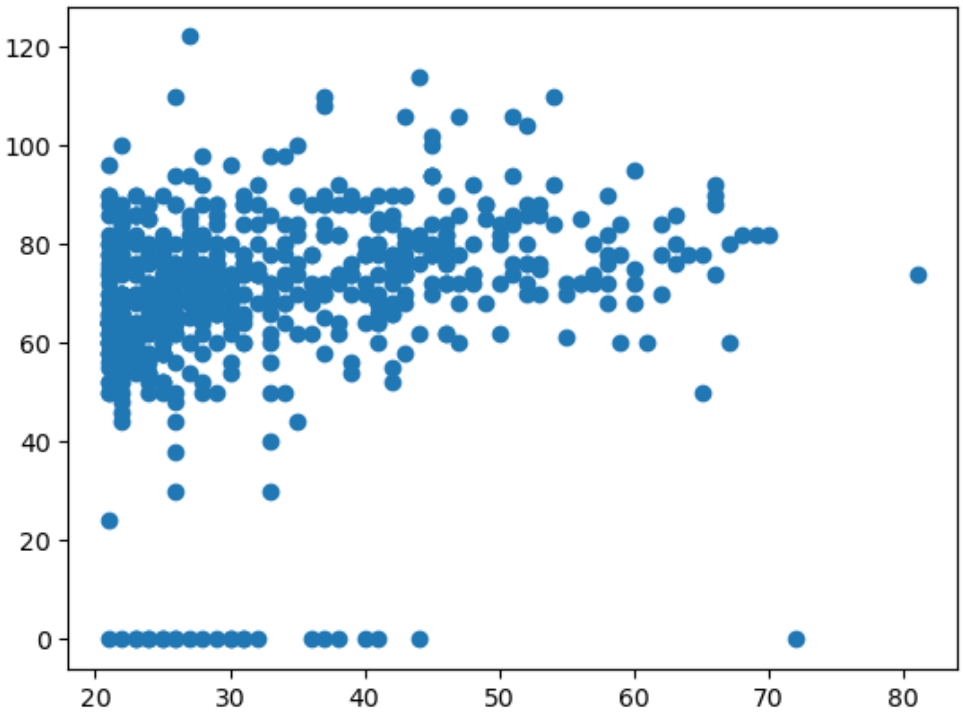


3.2 Bivariate Analysis:

- Explore relationships between pairs of variables (e.g., scatter plots, correlation matrices).

[453]:
```
# Relationship between blood pressure and age
plt.scatter(df_diabetes["Age"],df_diabetes["BloodPressure"])
```

[453]:
```
<matplotlib.collections.PathCollection at 0x322d338f0>
```



[454]:
```
print("Pearson Correlation Coefficient = \n", np.corrcoef(df_diabetes["Age"],␣
↪df_diabetes["BloodPressure"]))
```

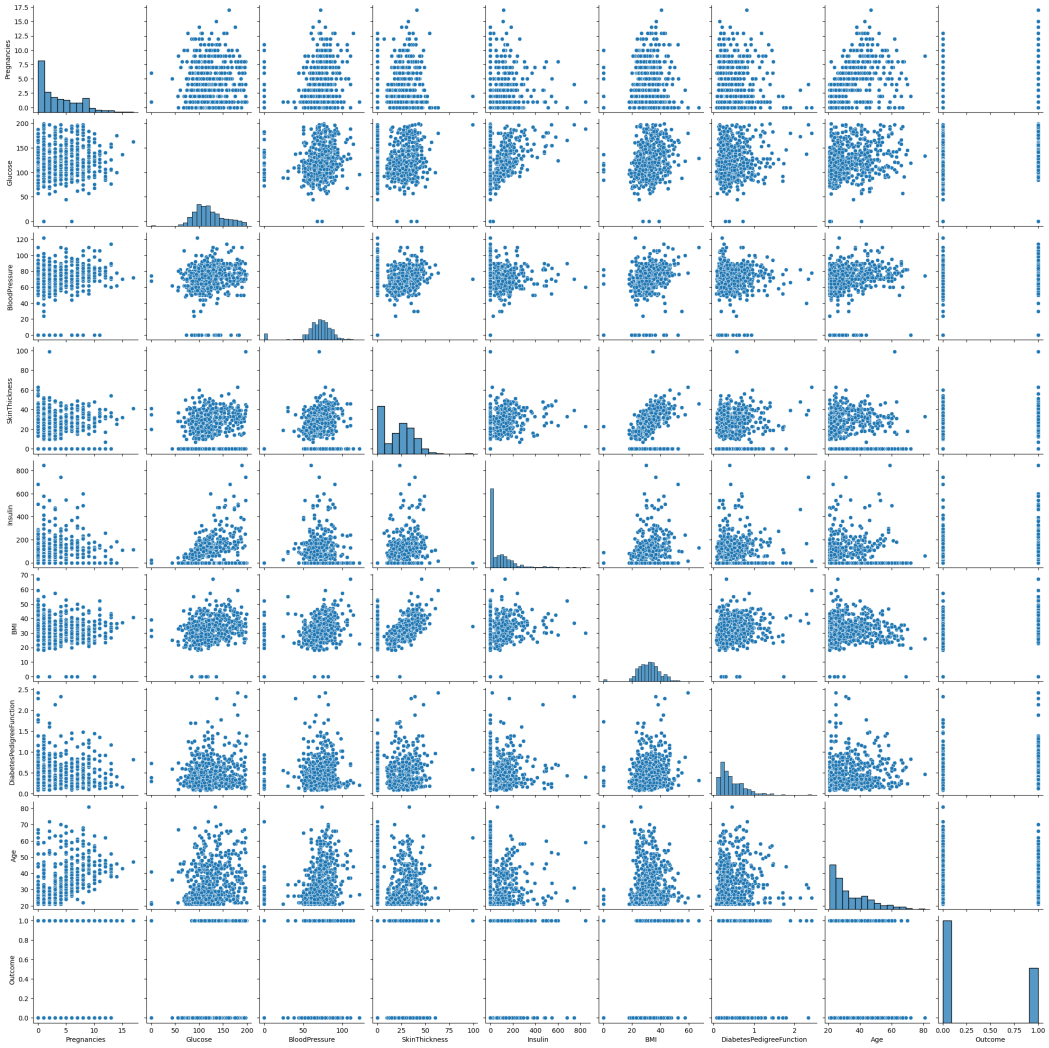
```
Pearson Correlation Coefficient =
 [[1.         0.22766343]
 [0.22766343 1.        ]]
```

3.3 Multivariate Analysis:

- Investigate interactions among multiple variables (e.g., pair plots, 3D plots).

[455]:
```
sns.pairplot(df_diabetes)
```

[455]: `<seaborn.axisgrid.PairGrid at 0x323a6f020>`


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome