IFQ509 - Assignment 1

This report concerns a subset of the Nexoid COVID-19 Survival Calculator data, which offers insights into demographic, health, and lifestyle variables influencing COVID-19 infection and mortality risk. We will focus on the essential steps for transforming raw data into a suitable format for analysis. Emphasis is placed on understanding the data, identifying and mitigating any data quality issues, and preparing the data for data mining. This process will enhance the dataset's utility for modelling and risk factor analysis in COVID-19 survival outcomes.

Task 1: Examine and correct data types

Screenshot of original column information

```
#show all column information
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5789 entries, 0 to 5788
Data columns (total 36 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   survey_date             5789 non-null   object
 1   region                  1562 non-null   object
 2   country                 5786 non-null   object
 3   ip_latitude             5789 non-null   float64
 4   ip_longitude            5789 non-null   float64
 5   gender                  5789 non-null   object
 6   age                     5789 non-null   object
 7   height                  5789 non-null   int64
 8   weight                  5789 non-null   int64
 9   bmi                     5789 non-null   float64
 10  blood_type              5689 non-null   object
 11  insurance               4497 non-null   object
 12  income                  4636 non-null   object
 13  smoking                 5753 non-null   object
 14  alcohol                 5751 non-null   float64
 15  cocaine                 1430 non-null   float64
 16  contacts_count          5747 non-null   float64
 17  public_transport_count  4572 non-null   float64
 18  working                 5747 non-null   object
 19  worried                 4518 non-null   float64
 20  covid19_positive        5789 non-null   int64
 21  covid19_symptoms        5789 non-null   int64
 22  covid19_contact         5789 non-null   int64
 23  asthma                  5789 non-null   int64
 24  kidney_disease          5789 non-null   int64
 25  liver_disease           5789 non-null   int64
 26  compromised_immune      5789 non-null   int64
 27  heart_disease           5789 non-null   int64
 28  lung_disease            5789 non-null   int64
 29  diabetes                5789 non-null   int64
 30  hiv_positive            5789 non-null   int64
 31  other_chronic           5789 non-null   int64
 32  nursing_home            5789 non-null   int64
 33  health_worker           5789 non-null   int64
 34  risk_infection          5789 non-null   float64
 35  risk_mortality          5789 non-null   float64
dtypes: float64(10), int64(16), object(10)
memory usage: 1.6+ MB
None
```

Table 1: Changes made to originally-assigned data types as needed

Variable	Data type	Data type subsequently assigned in Python	Has the data type changed?
survey_date	quantitative - continuous	datetime64	yes
region	qualitative - nominal	string (category)	yes
country	qualitative - nominal	string (category)	yes

Variable	Data type	Data type subsequently assigned in Python	Has the data type changed?
ip_latitude	quantitative - continuous	float	no
ip_longitude	quantitative - continuous	float	no
gender	qualitative - nominal	string (category)	yes
age	qualitative - ordinal	string (category)	yes
height	quantitative - discrete	int64 - integer	no
weight	quantitative - discrete	int64 - integer	no
bmi	quantitative - continuous	float	no
blood_type	qualitative - nominal	string (category)	yes
insurance	qualitative - nominal	string (category)	yes
income	qualitative - ordinal	string (category)	yes
smoking	qualitative - ordinal	string (category)	yes
alcohol	quantitative - discrete	int64 - integer	yes
cocaine	quantitative - discrete	int64 - integer	yes
contacts_count	quantitative - discrete	int64 - integer	yes
public_transport_count	quantitative - discrete	int64 - integer	yes
working	qualitative - nominal	string (category)	yes
worried	qualitative - ordinal	string (category)	yes
covid19_positive	qualitative - nominal	int64 - integer	no
covid19_symptoms	qualitative - nominal	int64 - integer	no
covid19_contact	qualitative - nominal	int64 - integer	no
asthma	qualitative - nominal	int64 - integer	no
kidney_disease	qualitative - nominal	int64 - integer	no
liver_disease	qualitative - nominal	int64 - integer	no
compromised_immune	qualitative - nominal	int64 - integer	no
heart_disease	qualitative - nominal	int64 - integer	no

Variable	Data type	Data type subsequently assigned in Python	Has the data type changed?
lung_disease	qualitative - nominal	int64 - integer	no
diabetes	qualitative - nominal	int64 - integer	no
hiv_positive	qualitative - nominal	int64 - integer	no
other_chronic	qualitative - nominal	int64 - integer	no
nursing_home	qualitative - nominal	int64 - integer	no
health_worker	qualitative - nominal	int64 - integer	no
risk_infection	quantitative - discrete	int64 - integer	yes
risk_mortality	quantitative - continuous	float64	no

Discussion of assigned data types:

a. Configuring import of datasets into Jupyter notebooks

It's important to ensure the dataset is being read into the notebook correctly. We encountered an issue where the region was incorrectly being set as NaN by pandas' read_csv function when it was actually 'NA' (for North America). The read_csv function by default recognises the string NA as being NaN (Pandas Development Team. (n.d.)). We handled for this by setting keep_default_na to False and specifying only "" values (that is, empty fields in the dataset) as NaN.

b. string vs. category assignment

The pandas library supports a number of data types, including string and category. Although string is the base type, category dtype has been developed to optimise for performance and subsequent queries (Pandas development team. (n.d.)). Under the hood, it allows for each category to be expressed as a number when the dtype is assigned to the field, with mappings made to the original category fields. We have chosen to use pandas category dtype for our string-based fields.

c. Fields containing blank vs. fields containing no data in the data set

Fields containing the word blank represent instances where the respondent elected not to fill in that particular field. This in itself may indicate some related factor, for example, people with high incomes being more reluctant to disclose this. This is again different from an empty field in the dataset, read into the Jupyter notebook as a NaN value. This could be caused by data loss or some other external factor. Simply assigning all NaN fields as blank confuses these two meanings.

d. survey_date as datetime64[ns], and other categorical fields

Although datetime64[ns] currently suits the data in the survey_date column well, we will have to modify this field as part of the cleaning and analysis process, as well as other categorical fields. In order to use Python modules like sklearn to process the dataset, non-numerical fields will need to be converted into numerical data. We will do this down the track in a few ways:

- we will express the survey_date field as an integer, indicating the days since the minimum day on which a survey was submitted
- we will use one-hot encoding to turn categorical (qualitative) fields in the dataset into a number of numerical fields each denoting a part of the information in the original categorical field. These can then be merged back into the single categorical column for readability once analysis is complete.

e. Int64 vs. int64

A standard Numpy integer type (int64) used for storing integers , does not support nullable integers, meaning that a column contains NaN values (missing values) will result in a error. In that case, a padas-specific nullable integer type (Int64) is introduced to handle that special value for missing data in pandas (pd.NA)

Screenshot of adjusted data types

```
dtypes: Int64(4), category(10), datetime64[ns](1), float64(4), int64(17)
memory usage: 1.2 MB
```

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5789 entries, 0 to 5788
Data columns (total 36 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   survey_date             5789 non-null   datetime64[ns]
 1   region                  5787 non-null   category
 2   country                 5787 non-null   category
 3   ip_latitude             5789 non-null   float64
 4   ip_longitude            5789 non-null   float64
 5   gender                  5789 non-null   category
 6   age                     5789 non-null   category
 7   height                  5789 non-null   int64
 8   weight                  5789 non-null   int64
 9   bmi                     5789 non-null   float64
 10  blood_type              5689 non-null   category
 11  insurance               4497 non-null   category
 12  income                  4636 non-null   category
 13  smoking                 5753 non-null   category
 14  alcohol                 5751 non-null   Int64
 15  cocaine                 1430 non-null   Int64
 16  contacts_count          5747 non-null   Int64
 17  public_transport_count  4572 non-null   Int64
 18  working                 5747 non-null   category
 19  worried                 4518 non-null   category
 20  covid19_positive        5789 non-null   int64
 21  covid19_symptoms        5789 non-null   int64
 22  covid19_contact         5789 non-null   int64
 23  asthma                  5789 non-null   int64
 24  kidney_disease          5789 non-null   int64
 25  liver_disease           5789 non-null   int64
 26  compromised_immune      5789 non-null   int64
 27  heart_disease           5789 non-null   int64
 28  lung_disease            5789 non-null   int64
 29  diabetes                5789 non-null   int64
 30  hiv_positive            5789 non-null   int64
 31  other_chronic           5789 non-null   int64
 32  nursing_home            5789 non-null   int64
 33  health_worker           5789 non-null   int64
 34  risk_infection          5789 non-null   int64
 35  risk_mortality          5789 non-null   float64
```

Task 2: Prepare data

2.1 Determining data skew

Skewness is typically applied to numerical (quantitative) or categorical (qualitative) data.

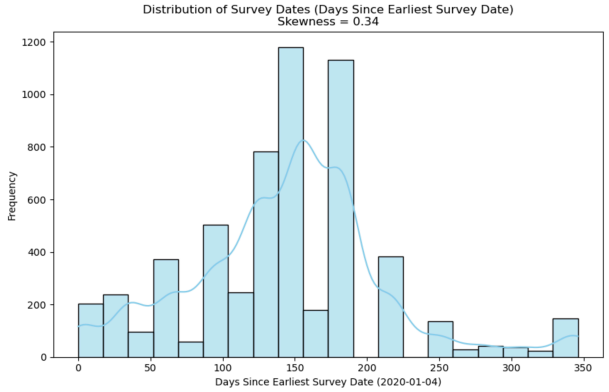
- numerical values: ip_longtitude & ip_longtitude, height, weight, bmi, alcohol, cocaine, contacts_count, public_transport_count, covid19_positive, covid19_symptoms, covid19_contact, asthma, kidney_disease, liver_disease, compromised_immune, heart_disease, lung_disease, diabetes, hiv_positive, other_chronic, nursing_home, health_worker, risk_infection and risk_mortality.

- categorical values: region, country, sex, age, blood_type, insurance, income, smoking, and worried.

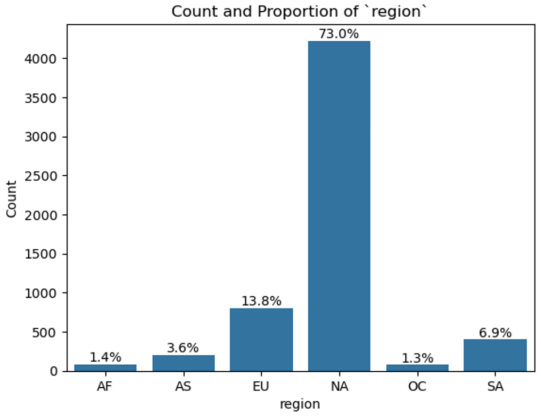
Screenshot of skewness type found from the dataset

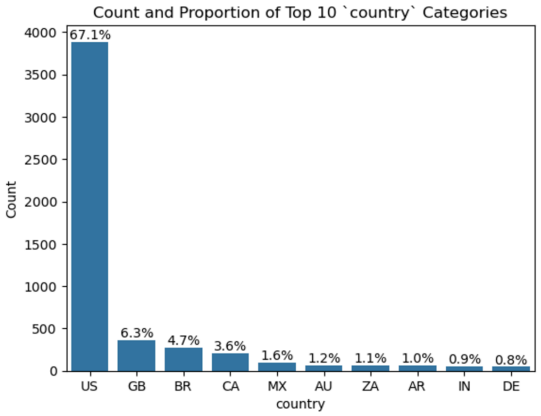
```
Skewness Direction Based on Mean and Median:
 ip_latitude               Negative Skew
ip_longitude              Positive Skew
height                      Symmetrical
weight                    Positive Skew
bmi                       Positive Skew
alcohol                   Positive Skew
cocaine                   Positive Skew
contacts_count            Positive Skew
public_transport_count    Positive Skew
covid19_positive          Positive Skew
covid19_symptoms          Positive Skew
covid19_contact           Positive Skew
asthma                    Positive Skew
kidney_disease              Symmetrical
liver_disease               Symmetrical
compromised_immune        Positive Skew
heart_disease               Symmetrical
lung_disease                Symmetrical
diabetes                  Positive Skew
hiv_positive                Symmetrical
other_chronic             Positive Skew
nursing_home                Symmetrical
health_worker             Positive Skew
risk_infection            Positive Skew
risk_mortality            Positive Skew
survey_date_as_int        Negative Skew
dtype: object
```

a. survey_date

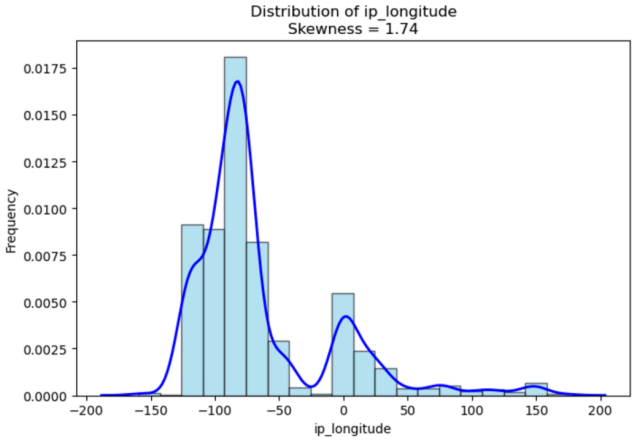


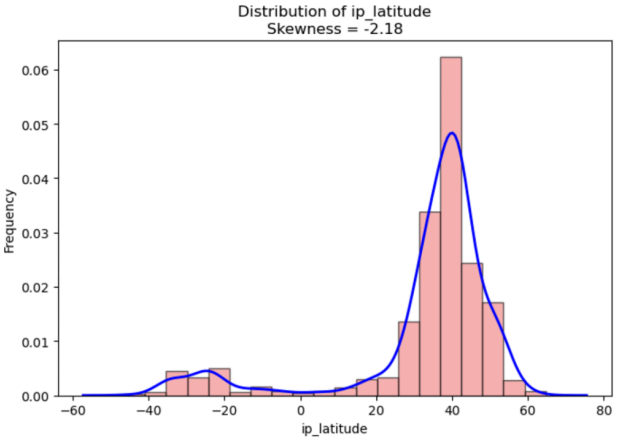
b. region & country



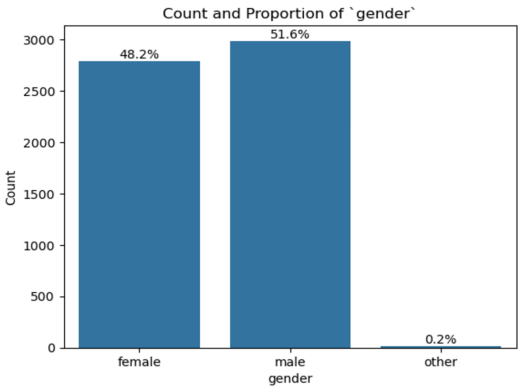


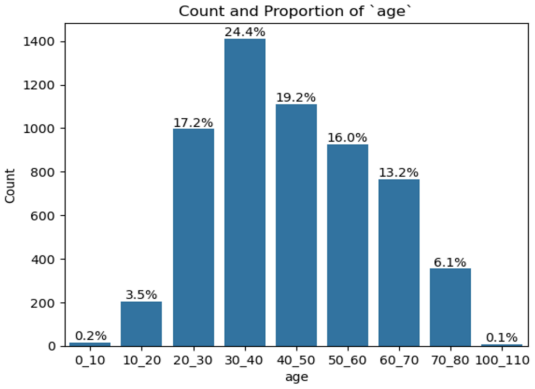
c. ip_longitude & ip_latitude



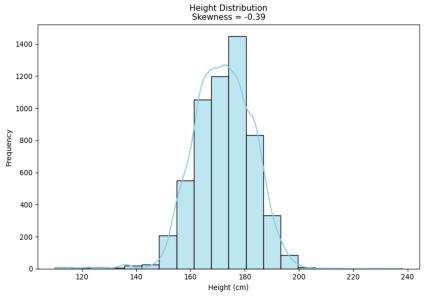


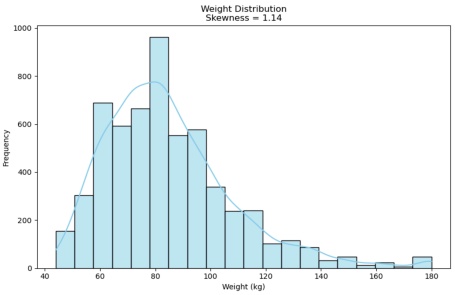
d. gender & age

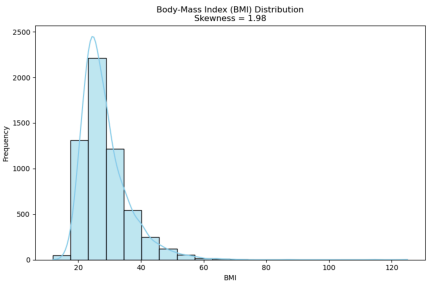




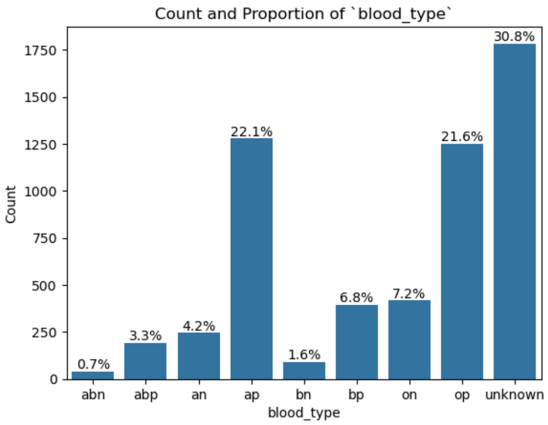
e. height, weight & bmi

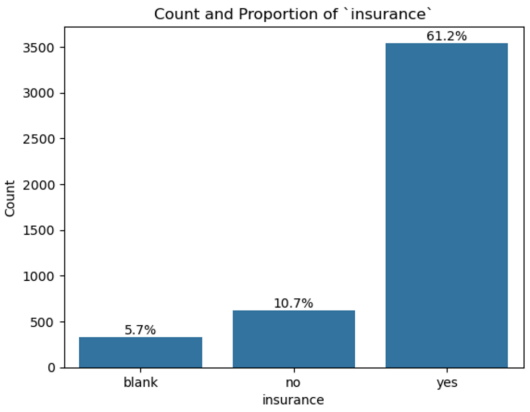




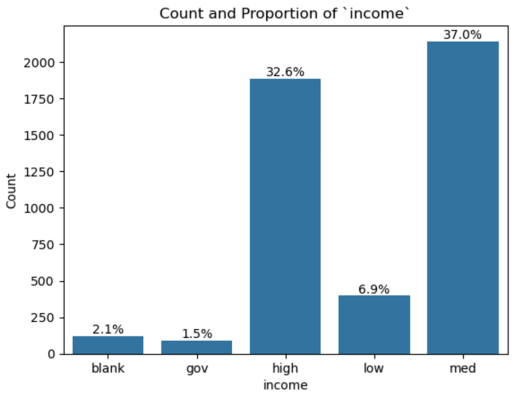


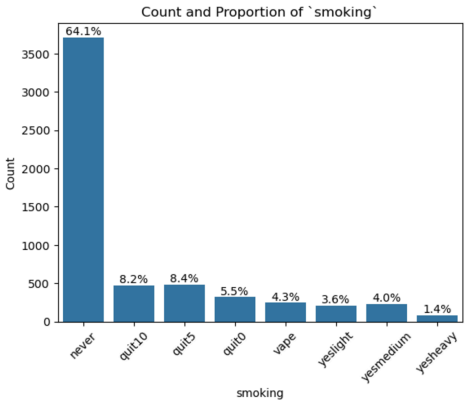
f. blood_type & insurance



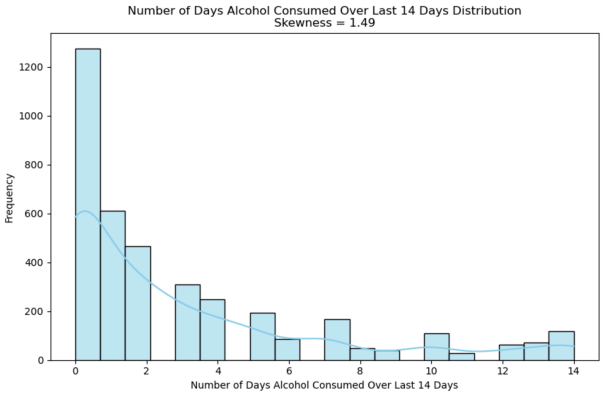


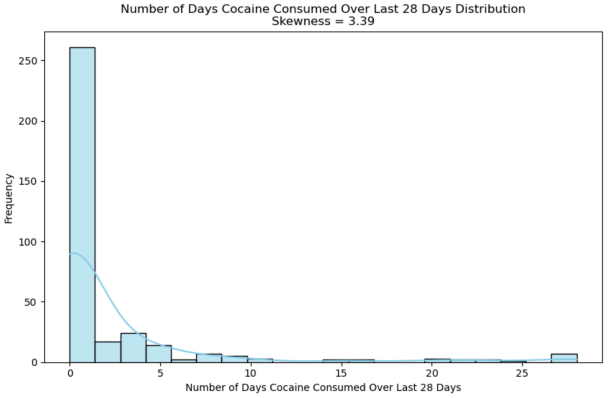
g. income & smoking



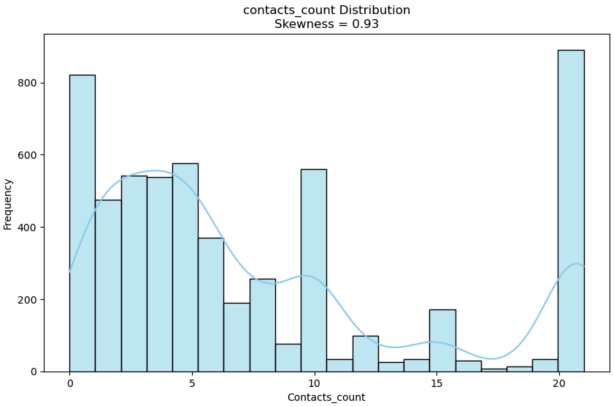


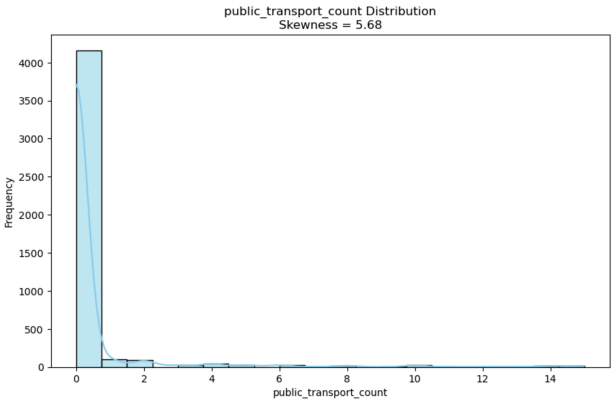
h. alcohol & cocaine



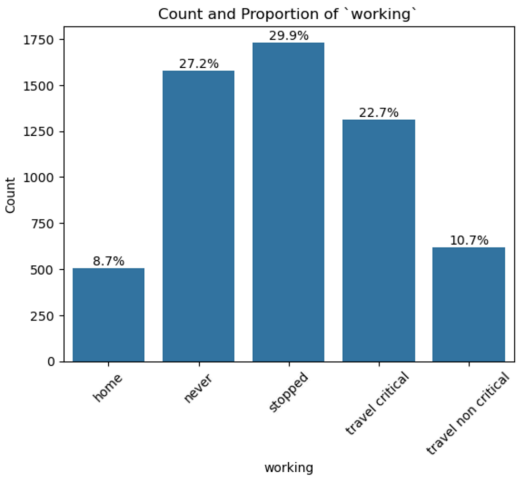


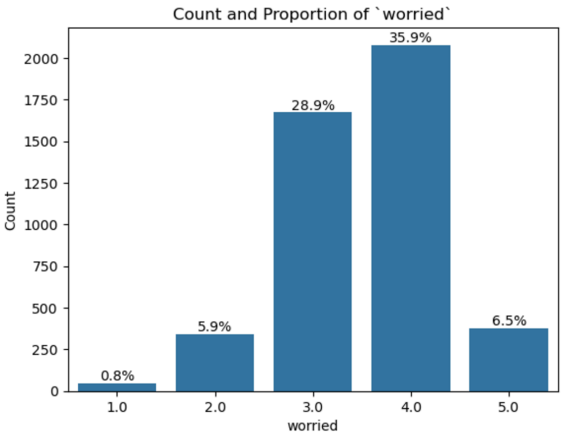
i. contacts_count & public_transport_count



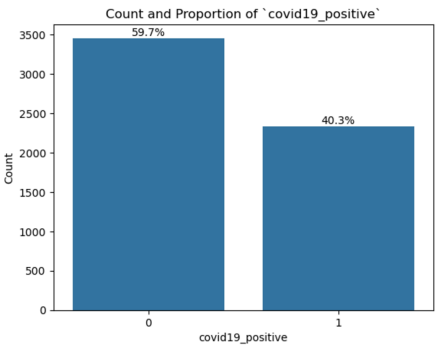


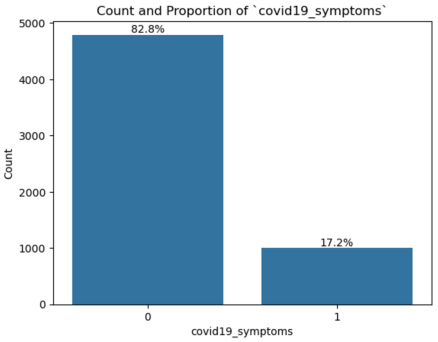
j. working & worried

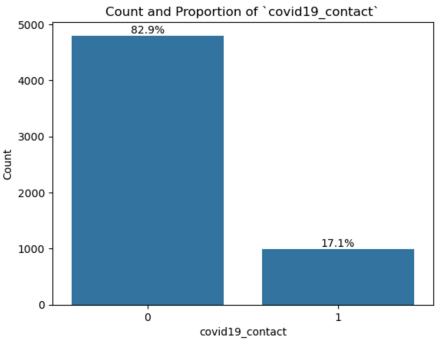




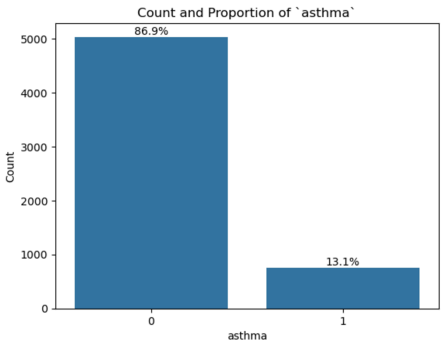
k. covid19_positive, covid19_symtoms & covid19_contact

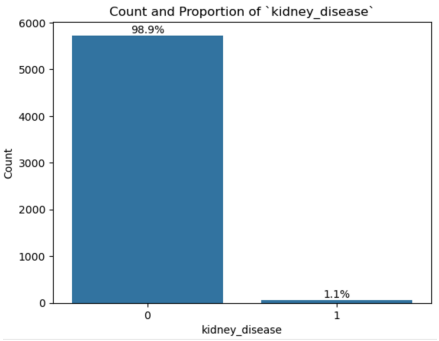


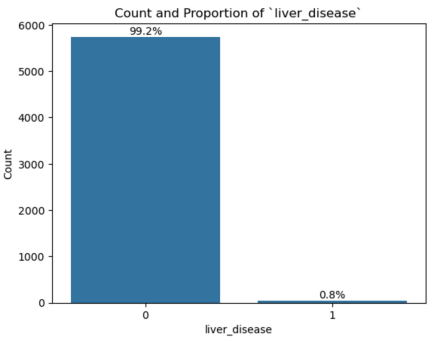




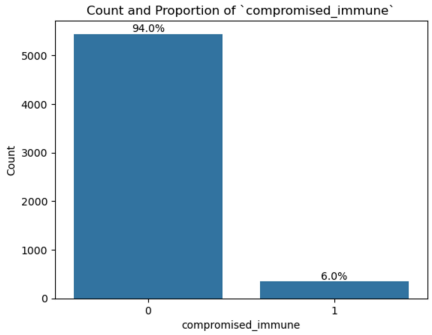
l. asthma, kidney_disease & liver_disease

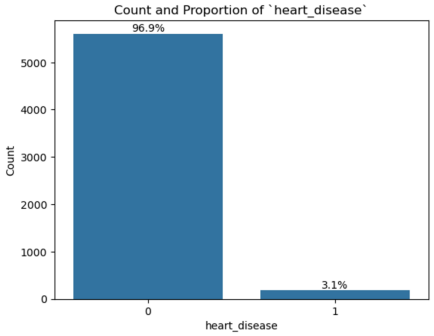


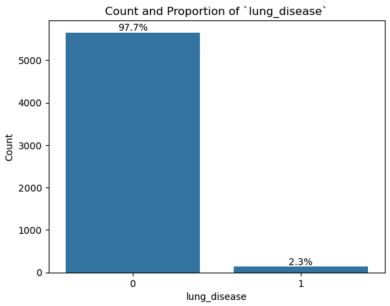




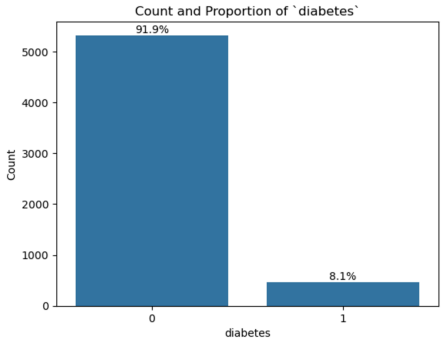
m. compromised_immune, heart_disease & lung_disease

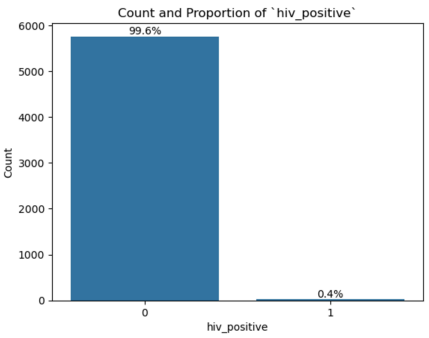


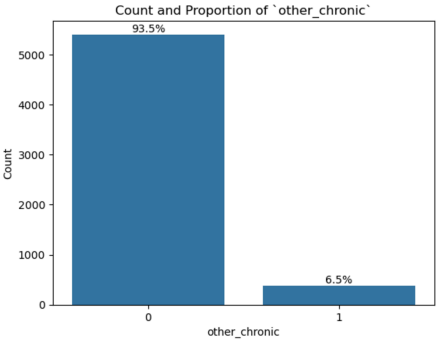




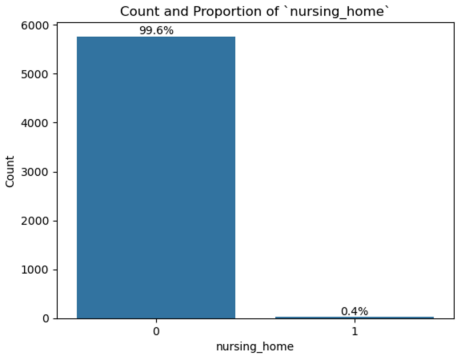
diabetes, hiv_positive & other_chronic

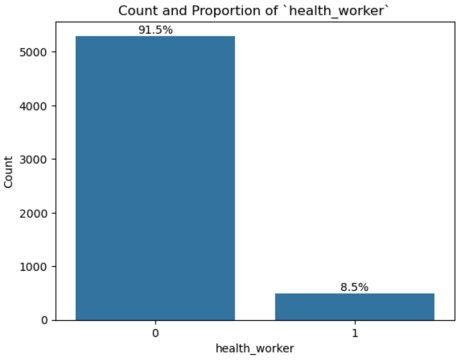




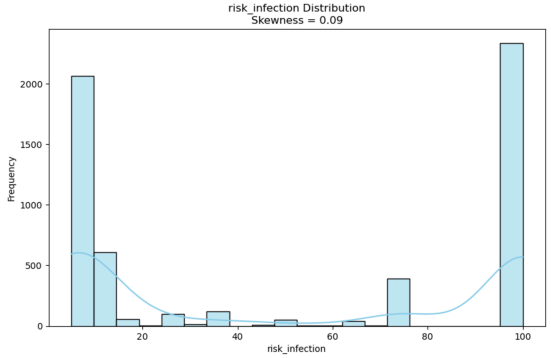


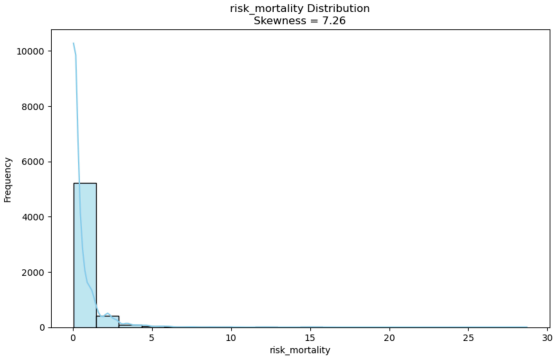
n. nursing_home & health_worker





o. risk_infection & risk_mortality





2.2 Data imputation

a. region & country

There were two rows in the dataset provided that had missing country and region information.

As the fields region, country, and ip_latitude & ip_longitude contain overlapping data about the respondent's location, the missing values for region and country were found by using ip_latitude & ip_longitude to approximate the respondent's location.

Screenshot of missing values before and after adjustment

	region	country	ip_latitude	ip_longitude
3543	NaN	NaN	35.6896	51.4332
5788	NaN	NaN	51.5875	-0.1807

	region	country	ip_latitude	ip_longitude
3543	AS	IR	35.6896	51.4332
5788	EU	GB	51.5875	-0.1807

In addition, it might not be necessary to maintain all the given location-related fields as part of the data mining work later on.

b. blood_type

There are 100 NaN values in the blood_type field. Dropping these rows would constitute 1.72% of the rows in the provided data set. This is quite low and we could consider dropping these. We could impute by mode, by finding the most frequent blood_type per country as blood_type is highly heritable. The ideal value to group by for blood_type heritability would be race, which exists in the original dataset as information on ethnicity.

c. insurance

There doesn't seem to be any relationship between null insurance values and other variables. insurance has taken the following populated values in the data provided:

- yes - 3544 rows
- no - 622 rows
- blank - 331 rows

```
data['insurance'].value_counts()
```

```
insurance
yes      3544
no        622
blank     331
Name: count, dtype: int64
```

There are 1292 rows with null insurance data - this is just over 22% of the whole dataset. Dropping all of these rows to handle null insurance values would result in significant data loss over the other fields.

We could simply assign blank to all the null insurance values as we don't know what those values are. However, we don't know if the respondent originally entered data in the survey or not, or if the data was lost at some other point. We could assign the general mode of populated insurance values to rows with null insurance values.

d. income

There are 1153 NaN values in the income field.

This makes up 19.9% of rows in the provided data set. Dropping such a large quantity of rows would result in significant data loss over the other fields. Imputation would be ideal.

As there is no numerical value associated with the categories and it is simply a personal rating of one's income (outside of blank, where the person explicitly did not answer), it would be difficult to relate with other fields in the dataset.

e. smoking

There are 36 NaN values in the smoking field. This is only 0.62% of rows in the total dataset; this would not result in a lot of data loss if the NaN fields were dropped. Again, mode imputation is possible.

f. alcohol

There are 38 NaN values in the alcohol field. This is only 0.66% of rows in the total dataset, this would not result in a lot of data loss if the NaN fields were dropped. Again, mode imputation is possible.

g． cocaine

There are 4359 NaN values in the cocaine field. This is about 75.3% of the total dataset. Dropping rows is not going to be an appropriate resolution. It might be better to drop the whole column when it comes to data mining.

h． contacts_count

There are 42 NaN values in the contacts_count field. This is about 0.73% of the rows in the dataset. Dropping these rows would be a reasonable solution.

i. public_transport_count

There are 1217 NaN values in the public_transport_count field. This is about 21% of the dataset. Imputation via mode or other means is preferable.

j. working

There are 42 NaN values in the working field. This is about 0.73% of the rows in the dataset. Dropping these rows would not result in significant data loss.

k. worried

There are 271 NaN values in the worried field. This is about 22% of the entire data set: we should look into imputation for this field.

Final decision re: data cleaning

We can impute all remaining 10 fields with data via IterativeImputer, which imputes data based on existing relationships between other populated variables in the dataset (Scikit-learn. (n.d.)). However, it's important to note that IterativeImputer is an experimental feature in the sklearn module, and that predictions may change without sklearn module update.

We will use the default BayesianRidge estimator for IterativeImputer. Other estimators we could use with IterativeImputer include RandomForestRegressor and KNeighborsRegressor (Scikit-learn developers. (n.d.)).

Process of data cleaning

- Began with one-hot encoding via pandas' get_dummies function.
    - Noticed categorical information being dropped when running get_dummies function across whole dataset

- Performed one-hot encoding across each categorical column individually before merging all dummied categorical columns together
- Ensured that NaN values were represented in the dummied categorical columns

```
# ... and set dummies of each category to `NaN` when the corresponding `..._nan` field is 1, so that the `IterativeImputer` knows what to replace

data_categorical_columns.loc[data_categorical_columns['insurance_nan'] == 1, ['insurance_yes', 'insurance_no', 'insurance_blank']] = np.nan
data_categorical_columns.loc[data_categorical_columns['blood_type_nan'] == 1, ['blood_type_abp', 'blood_type_an', 'blood_type_ap', 'blood_type_bn', 'blood_type_bp', 'blood_type_on', 'blood_type_op', 'blood_type_unknown']] = np.nan
data_categorical_columns.loc[data_categorical_columns['income_nan'] == 1, ['income_high', 'income_med', 'income_low', 'income_gov', 'income_blank']] = np.nan
data_categorical_columns.loc[data_categorical_columns['age_nan'] == 1, ['age_0_10', 'age_10_20', 'age_20_30', 'age_30_40', 'age_40_50', 'age_50_60', 'age_60_70', 'age_70_80', 'age_100_110']] = np.nan
data_categorical_columns.loc[data_categorical_columns['region_nan'] == 1, ['region_AF', 'region_AS', 'region_EU', 'region_NA', 'region_OC', 'region_SA']] = np.nan
data_categorical_columns.loc[data_categorical_columns['country_nan'] == 1, ['country_AO', 'country_AE', 'country_AM', 'country_AR', 'country_AT',
                                                                             'country_AU', 'country_AW', 'country_AZ', 'country_BA', 'country_BD',
                                                                             'country_BE', 'country_BG', 'country_BH', 'country_BM', 'country_BR',
                                                                             'country_BS', 'country_CA', 'country_CH', 'country_CL', 'country_CO',
                                                                             'country_CR', 'country_CZ', 'country_DE', 'country_DK', 'country_DO',
                                                                             'country_DZ', 'country_EC', 'country_EG', 'country_ES', 'country_ET',
                                                                             'country_FI', 'country_FR', 'country_GB', 'country_GE', 'country_GR',
                                                                             'country_GT', 'country_HK', 'country_HN', 'country_HR', 'country_HU',
                                                                             'country_ID', 'country_IE', 'country_IL', 'country_IN', 'country_IQ',
                                                                             'country_IR', 'country_IT', 'country_JE', 'country_JP', 'country_KE',
                                                                             'country_KR', 'country_KY', 'country_KZ', 'country_LT', 'country_LU',
                                                                             'country_LV', 'country_MA', 'country_MD', 'country_MK', 'country_MM',
                                                                             'country_MO', 'country_MT', 'country_MU', 'country_MX', 'country_MY',
                                                                             'country_NZ', 'country_NA', 'country_NL', 'country_NO', 'country_NP',
                                                                             'country_NZ', 'country_PA', 'country_PE', 'country_PH', 'country_PK',
                                                                             'country_PL', 'country_PR', 'country_PS', 'country_PT', 'country_QA',
                                                                             'country_RO', 'country_RS', 'country_RU', 'country_SA', 'country_SE',
                                                                             'country_SG', 'country_SI', 'country_SK', 'country_SV', 'country_TH',
                                                                             'country_TR', 'country_UA', 'country_US', 'country_VI', 'country_ZA']] = np.nan
data_categorical_columns.loc[data_categorical_columns['gender_nan'] == 1, ['gender_male', 'gender_female', 'gender_other']] = np.nan
data_categorical_columns.loc[data_categorical_columns['smoking_nan'] == 1, ['smoking_never', 'smoking_quit0', 'smoking_quit5', 'smoking_quit10',
                                                                             'smoking_vape','smoking_yeslight', 'smoking_yesmedium', 'smoking_yesheavy']] = np.nan
data_categorical_columns.loc[data_categorical_columns['worried_nan'] == 1, ['worried_1.0', 'worried_2.0', 'worried_3.0', 'worried_4.0','worried_5.0']] = np.nan
data_categorical_columns.loc[data_categorical_columns['working_nan'] == 1, ['working_home', 'working_never', 'working_stopped',
                                                                             'working_travel critical', 'working_travel non critical']] = np.nan
```

- Converted survey_date from datetime64[ns]to int64-type column, as the number of days since the minimum date in the dataset.
- Merged categorical and numerical columns together into one dataset ready for imputation

- Used sklearn's IterativeImputer with BayesianRidge estimator to impute data

```
[423]: # using IterativeImputer from sklearn module to impute missing values using the default Bayesian ridge regression setting

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imputer = IterativeImputer(random_state=0)

data_imputed = imputer.fit_transform(numerical_and_categorical_columns)

## converting to dataframe

data_imputed = pd.DataFrame(data_imputed, columns=numerical_and_categorical_columns.columns)
```

- Rounded imputed values to whole numbers, retaining the imputed value closest to 1 and setting all other imputed values to 0

```
[426]: # we have estimates for the NaN values! let's take the largest value for each row for each group of dummies with NaN values, set that to 1 and set the others as 0.

def set_largest_to_one(row, columns):
    max_column_value = row[columns].idxmax()
    row[columns] = 0
    row[max_column_value] = 1
    return row

# set up lists for each group of dummies with NaN values that have been replaced

insurance_column_list = ['insurance_yes', 'insurance_no', 'insurance_blank']
blood_type_column_list = ['blood_type_abn', 'blood_type_abp', 'blood_type_an', 'blood_type_ap', 'blood_type_bn', 'blood_type_bp', 'blood_type_on', 'blood_type_op', 'blood_type_unknown']
income_column_list = ['income_gov', 'income_blank', 'income_high', 'income_low', 'income_med']
age_column_list = ['age_0_10', 'age_100_110', 'age_10_20', 'age_20_30', 'age_30_40', 'age_40_50', 'age_50_60', 'age_60_70', 'age_70_80']
region_column_list = ['region_AF', 'region_AS', 'region_EU', 'region_NA', 'region_OC', 'region_SA']
country_column_list = ['country_AO', 'country_AE', 'country_AM', 'country_AR', 'country_AT',
                       'country_AU', 'country_AW', 'country_AZ', 'country_BA', 'country_BD',
                       'country_BE', 'country_BG', 'country_BH', 'country_BM', 'country_BR',
                       'country_BS', 'country_CA', 'country_CH', 'country_CL', 'country_CO',
                       'country_CR', 'country_CZ', 'country_DE', 'country_DK', 'country_DO',
                       'country_DZ', 'country_EC', 'country_EG', 'country_ES', 'country_ET',
                       'country_FI', 'country_FR', 'country_GB', 'country_GE', 'country_GR',
                       'country_GT', 'country_HK', 'country_HN', 'country_HR', 'country_HU',
                       'country_ID', 'country_IE', 'country_IL', 'country_IN', 'country_IQ',
                       'country_IR', 'country_IT', 'country_JE', 'country_JP', 'country_KE',
                       'country_KR', 'country_KY', 'country_KZ', 'country_LT', 'country_LU',
                       'country_LV', 'country_MA', 'country_MD', 'country_MK', 'country_MM',
                       'country_MO', 'country_MT', 'country_MU', 'country_MX', 'country_MY',
                       'country_MZ', 'country_NA', 'country_NL', 'country_NO', 'country_NP',
                       'country_NZ', 'country_PA', 'country_PE', 'country_PH', 'country_PK',
                       'country_PL', 'country_PR', 'country_PS', 'country_PT', 'country_QA',
                       'country_RO', 'country_RS', 'country_RU', 'country_SA', 'country_SE',
                       'country_SG', 'country_SI', 'country_SK', 'country_SV', 'country_TH',
                       'country_TR', 'country_UA', 'country_US', 'country_VI', 'country_ZA']
gender_column_list = ['gender_male', 'gender_female', 'gender_other']
smoking_column_list = ['smoking_never', 'smoking_quit0', 'smoking_quit10', 'smoking_quit5',
                       'smoking_vape', 'smoking_yesheavy', 'smoking_yeslight','smoking_yesmedium']
worried_column_list = ['worried_1.0', 'worried_2.0', 'worried_3.0', 'worried_4.0','worried_5.0']
working_column_list = ['working_home', 'working_never', 'working_stopped',
                       'working_travel critical', 'working_travel non critical']

list_of_lists = [insurance_column_list, blood_type_column_list, income_column_list, age_column_list, region_column_list, country_column_list, gender_column_list, smoking_column_list, worried_column_list, working_column_list]

# Apply the function to each row in the DataFrame for each group of dummy fields
for list in list_of_lists:
    data_imputed = data_imputed.apply(lambda row: set_largest_to_one(row, list), axis=1)
```

- Ensure imputed results make sense with respect to column definitions

```
[429]: # Round and adjust the imputed numerical columns so that they fit into their original definitions

data_imputed_rounded = data_imputed.copy()

data_imputed_rounded[['alcohol', 'cocaine', 'contacts_count', 'public_transport_count']] = data_imputed_rounded[['alcohol', 'cocaine', 'contacts_count', 'public_transport_count']].round(decimals=0)

# rows with values less than -1 should be set to -1 for `alcohol` field (to represent non-drinkers)

data_imputed_rounded.loc[data_imputed_rounded['alcohol'] < -1, ['alcohol']] = -1

# rows with values less than -1 should be set to -1 for `cocaine` field (to represent those who have never used cocaine)

data_imputed_rounded.loc[data_imputed_rounded['cocaine'] < -1, ['cocaine']] = -1

# for `contacts_count` field, 21 is reserved for >20 contacts

data_imputed_rounded.loc[data_imputed_rounded['contacts_count'] > 20, ['contacts_count']] = 21

# rows with values less than -1 should be set to 0 for `public_transport_count` field (to represent those who did not use public transport)

data_imputed_rounded.loc[data_imputed_rounded['public_transport_count'] < -1, ['public_transport_count']] = 0

# looking at the larger Nexoid dataset, there are no rows containing a value for greater than `15` for `public_transport_count` across more than 900,000 records.
# it's likely that `public_transport_count` works the same way as the `alcohol` field - if you took public transport more than 14 times then you are assigned `15` as the ceiling, but we don't know this for sure.
```

- Merge dummied columns back together

No NaN fields remain in the processed dataset:

```
[690]: # check

var = cleaned_dataset.isnull().sum()
print(var.to_string())
```

```
ip_latitude                0
ip_longitude               0
height                     0
weight                     0
bmi                        0
alcohol                    0
cocaine                    0
contacts_count             0
public_transport_count     0
covid19_positive           0
covid19_symptoms           0
covid19_contact            0
asthma                     0
kidney_disease             0
liver_disease              0
compromised_immune         0
heart_disease              0
lung_disease               0
diabetes                   0
hiv_positive               0
other_chronic              0
nursing_home               0
health_worker              0
risk_infection             0
risk_mortality             0
income_nan                 0
insurance_nan              0
blood_type_nan             0
age_nan                    0
region_nan                 0
country_nan                0
gender_nan                 0
smoking_nan                0
working_nan                0
worried_nan                0
survey_date                0
insurance                  0
blood_type                 0
income                     0
age                        0
region                     0
country                    0
gender                     0
smoking                    0
worried                    0
working                    0
```

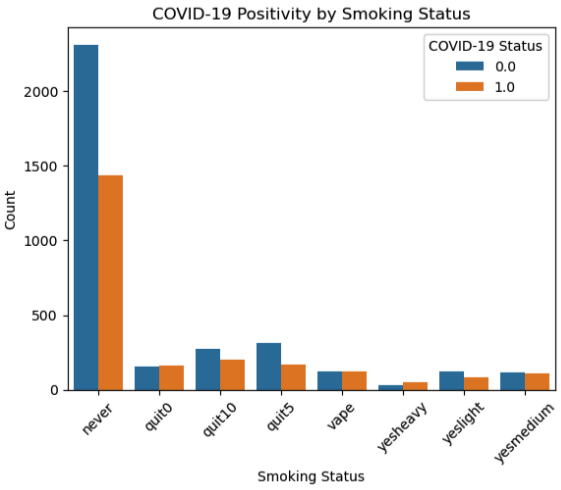
Note that with the selection and rounding of the largest imputed value for categorical fields, the `IterativeImputer` has effectively used mode replacement for a number of the categorical fields missing data. Further actions could include normalisation of numerical fields and using `KNeighborsRegressor` estimator with the `IterativeImputer` instead.

Task 3: Select data mining task and feature selection
Relationship between `covid19_positive` and `smoking`

We began by visualising `covid19_positive` against `smoking` data, using a stacked bar graph:

```
[714]: # Create the bar plot
sns.barplot(data=smoking_covid_19_positive_status_with_count, x='smoking', y='count', hue='covid19_positive')

# Customize the plot
plt.title('COVID-19 Positivity by Smoking Status')
plt.ylabel('Count')
plt.xlabel('Smoking Status')
plt.legend(title='COVID-19 Status')
plt.xticks(rotation=45)
plt.show()
```



We also performed some basic statistical analysis via `scipy.stats` module, using chi-square test, resultant p-value and Cramer's V.

[451]:
```
# Chi-square test via stats
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

# Cramer's V via numpy
n = contingency_table.to_numpy().sum()
cramers_v = np.sqrt(chi2 / (n * (min(contingency_table.shape) - 1)))

print(f"Chi-square: {chi2}, p-value: {p}, Cramer's V: {cramers_v}")
```

```
Chi-square: 54.596748253386885, p-value: 1.791932522763504e-09, Cramer's V: 0.09711394885699592
```

All of this shows a strong relationship between `covid19_positive` and `smoking`.

It is important to remember that there may be a number of confounding factors and that correlation does not equal causation. For example, another variable or a combination of variables may be related to smoking and covid-19 positivity and it is via this undetermined secondary relationship that smoking and covid-19 positivity are connected.

Regardless, this strong relationship between `smoking` status and `covid19_positive` status may cause inaccuracies and redundancy in modelling down the track - it may be useful to drop `smoking`.

We should also identify if any other variables are strongly correlated and need to be dropped from the dataset for analysis. We can do this by building a correlation matrix.

Discovery of other correlated variables

[449]:
```
import scipy.stats as stats

smoking_covid_19_positive_status = cleaned_dataset[['smoking','covid19_positive']]

smoking_covid_19_positive_status_with_count = smoking_covid_19_positive_status.groupby(['smoking', 'covid19_positive']).size().reset_index(name='count')

contingency_table = pd.pivot_table(smoking_covid_19_positive_status_with_count, values='count', index='covid19_positive', columns='smoking', aggfunc='sum', fill_value=0)

print(contingency_table)
```

```
smoking           never  quit0  quit10  quit5  vape  yesheavy  yeslight  \
covid19_positive
0.0                2313    158     273    315   122        33       121
1.0                1436    161     200    170   125        47        85

smoking           yesmedium
covid19_positive
0.0                     119
1.0                     111
```

[455]:
```
## going back to a version of the dataset with dummied, numerical data

# build the correlation matrix
corr_matrix = data_imputed_rounded.corr()
```

[727]:
```
# setting a threshold - if higher than 0.8 then these pairs point towards multicollinearity
threshold = 0.8

# finding pairs
highly_correlated = []

for column in corr_matrix.columns:
    for row in corr_matrix.index:
        if abs(corr_matrix.loc[row, column]) > threshold and row != column:
            highly_correlated.append((row, column, corr_matrix.loc[row, column]))

# display
for pair in highly_correlated:
    print(f"Variables: {pair[0]} and {pair[1]} have correlation {pair[2]:.2f}")
```

```
Variables: region_NA and ip_longitude have correlation -0.81
Variables: bmi and weight have correlation 0.86
Variables: weight and bmi have correlation 0.86
Variables: risk_infection and covid19_positive have correlation 0.92
Variables: covid19_positive and risk_infection have correlation 0.92
Variables: insurance_nan and income_nan have correlation 0.93
Variables: worried_nan and income_nan have correlation 0.92
Variables: income_nan and insurance_nan have correlation 0.93
Variables: worried_nan and insurance_nan have correlation 0.85
Variables: country_ZA and region_AF have correlation 0.91
Variables: ip_longitude and region_NA have correlation -0.81
Variables: country_US and region_NA have correlation 0.87
Variables: country_AU and region_OC have correlation 0.94
Variables: country_BR and region_SA have correlation 0.81
Variables: country_nan and region_nan have correlation 1.00
Variables: region_OC and country_AU have correlation 0.94
Variables: region_SA and country_BR have correlation 0.81
Variables: region_NA and country_US have correlation 0.87
Variables: region_AF and country_ZA have correlation 0.91
Variables: region_nan and country_nan have correlation 1.00
Variables: gender_male and gender_female have correlation -1.00
Variables: gender_female and gender_male have correlation -1.00
Variables: working_nan and smoking_nan have correlation 0.93
Variables: smoking_nan and working_nan have correlation 0.93
Variables: income_nan and worried_nan have correlation 0.92
Variables: insurance_nan and worried_nan have correlation 0.85
```

As per the above, it makes sense to retain only one of:

- `height`,`weight` and `bmi`
- `country`, `region` and `ip_latitude`/`ip_longitude`

The NaN columns are shown to be correlated, meaning there might be a minimum subset of rows we can drop with maximum missing data to minimise the amount of other data being lost. Note at this point we will not include the `-_nan` suffixed columns in the final dataset.

Retaining `bmi` and `country` makes the most sense - they result in the most readable information being preserved in the dataset without overlap.

`risk_infection` and `covid19_positive` are also correlated. This makes sense - the Nexoid analysts would have used the survey data to build models around risk of infection and death based on the respondents' data. If we were to build our own models re: risk of covid19 positivity, using `risk_infection`, which is itself partially derived from `covid19_positive`, would not work.

Appropriate data mining tasks

We can certainly use this data to build a predictive model of COVID19 mortality and infection risk (e.g. classification). Association mining is more suited to transactional data, however, we could also build up a profile of those likely to be COVID19 positive via clustering. This overlaps with classification, however - it makes sense to build the models around mortality and infection risk and use the information gathered as part of this in the mentioned profile determination.

Variables to be included in final dataset

- ~~`ip_latitude`~~ redundant
- ~~`ip_longitude`~~ redundant
- ~~`height`~~ redundant
- ~~`weight`~~ redundant
- `bmi`
- `alcohol`
- ~~`cocaine`~~ too much missing data
- `contacts_count`
- `public_transport_count`
- `covid19_positive`
- `covid19_symptoms`
- `covid19_contact`
- `asthma`
- `kidney_disease`
- `liver_disease`
- `compromised_immune`
- `heart_disease`
- `lung_disease`
- `diabetes`
- `hiv_positive`
- `other_chronic`
- `nursing_home`
- `health_worker`
- ~~`risk_infection`~~ derived value based on other values
- ~~`risk_mortality`~~ derived value based on other values
- ~~`survey_date`~~ irrelevant
- `insurance`
- `blood_type`
- `income`
- `age`
- ~~`region`~~ redundant
- `country`
- `gender`
- `smoking`
- `worried`
- `working`

References and reading

eNidaan. (n.d.). *COVID-19 analysis*. GitHub. Retrieved November 4, 2024, from https://github.com/eNidaan/covid19_analysis?tab=readme-ov-file

COVID-19 survival calculator. (n.d.). Retrieved November 4, 2024, from https://www.covid19survivalcalculator.com/en/home

Ogedegbe, G., Ravenell, J., Adhikari, S., Butler, M., Cook, T., Francois, F., Iturrate, E., Jean-Louis, G., Jones, S. A., Onakomaiya, D., Shah, N., & Troxel, A. B. (2021). Assessment of racial/ethnic disparities in hospitalization and mortality in patients with COVID-19 in New York City. *medRxiv*. https://doi.org/10.1101/2021.03.02.21252269

Nustat. (n.d.). *Data cleaning and preparation*. Retrieved November 6, 2024, from https://nustat.github.io/DataScience_Intro_python/Data%20cleaning%20and%20preparation.html

Pandas development team. (n.d.). *Categorical data*. Pandas documentation. Retrieved November 17, 2024, from https://pandas.pydata.org/docs/user_guide/categorical.html

Pandas Development Team. (n.d.). *pandas.read_csv* — pandas 2.2.3 documentation. Pandas Documentation. Retrieved November 17, 2024, from https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Scikit-learn. (n.d.). *sklearn.impute.IterativeImputer*. Retrieved November 10, 2024, from https://scikit-learn.org/1.5/modules/generated/sklearn.impute.IterativeImputer.html

Scikit-learn developers. (n.d.). *Iterative imputer variants comparison*. Scikit-learn documentation. Retrieved November 10, 2024, from https://scikit-learn.org/stable/auto_examples/impute/plot_iterative_imputer_variants_comparison.html