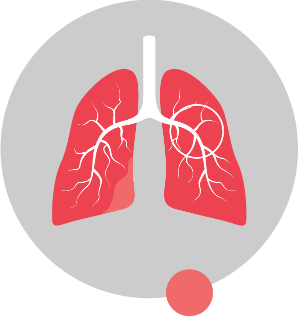
Tracking Tuberculosis Trends: A Historical Analysis in the U.S.



Coding Crew:
Daniel Kenet, Jennifer Kim, Kat McEldowney, Michelle Minkowitz

Index

About TB

U.S. Department of
Health and Human Services
Centers for Disease
Control and Prevention

Steps to clean the data

Merging dataframes

1

Removing

First, we removed any unnecessary columns

2

Separating

Separating the data sets into raw cases, and “rate per 100k”

3

Indexing

Turning unique values of each table into a column through a function

4

Merging

Merging all the tables based on year value in a state

5

Exporting

Turning the new data-frames into CSVs for our machine learning process

```
def clean_dataframes(dataframes):
    for df_name, df in dataframes.copy().items():
        if "M-F_cases" in df_name:
            dataframes[df_name] = process_cases(df, ['Year', 'Geography', 'Sex'])

        elif "M-F_rate" in df_name or "US vs Non_rate" in df_name:
            dataframes[df_name] = process_rate(df, ['Year', 'Geography', 'Sex'] if "M-F_rate" in df_name else ['Year', 'Geography', 'Country of birt

        elif "US vs Non_cases" in df_name:
            dataframes[df_name] = process_cases(df, ['Year', 'Geography', 'Country of birth'])

        elif "Age Groups_cases" in df_name:
            dataframes[df_name] = process_cases(df, ['Year', 'Geography', 'Age Group'])

        elif "Age Groups_rate" in df_name:
            dataframes[df_name] = process_rate(df, ['Year', 'Geography', 'Age Group'])

        elif "Race-Ethnicity_cases" in df_name:
            dataframes[df_name] = process_cases(df, ['Year', 'Geography', 'Race/Ethnicity'])

        elif "Race-Ethnicity_rate" in df_name:
            dataframes[df_name] = process_rate(df, ['Year', 'Geography', 'Race/Ethnicity'])

def process_cases(df, groupby_cols):
    df_cases_split = df.groupby(groupby_cols)['Cases'].sum().unstack() #allows the unique values to be indexed as columns
    df_cases_split.reset_index(inplace=True)
    return df_cases_split

def process_rate(df, groupby_cols):
    df_rate_split = df.groupby(groupby_cols)['Rate per 100000'].sum().unstack()
    df_rate_split.reset_index(inplace=True)
    return df_rate_split

# Call the function to clean the dataframes
clean_dataframes(dataframes)
```

Mapping Data

1

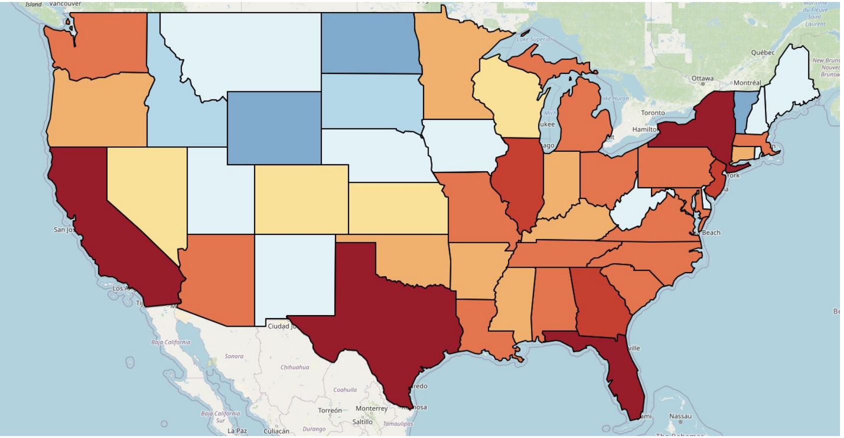
Select the Year

State outcomes are grouped by year

2

Hover over state

View data points via the callout boxes for each state



3

Case vs Rate

Total case count is the number of confirmed cases reported to the state/jurisdiction participating in the national TB surveillance system

Incidence rate is the number of cases per 100,000 persons

Linear Regression

Overview

Model Selection

Lowest Mean Square Error

Training Process

Data from 2000-2018

Time Series

Tried with and without lags and compared the models

Prediction

Predicted future values for 2019-2022

Experimented with testing training splits

and the other without a split in order to compare the accuracy of the two.

In [37]:

```
#create function to run linear regression analysis on the data with training (2000-2017) and testing (2018, 2019)
def lr_model(df, model_info):
    #split data into testing and training
    train_data = df[df['Year'] < 2018]
    test_data = df[df['Year'] >= 2018]
    #split into features and targets
    X_train = train_data.drop(columns=["Sum"])
    X_test = test_data.drop(columns=["Sum"])
    y_train = train_data['Sum']
    y_test = test_data['Sum']
    #create and fit the model
    model = LinearRegression()
    model.fit(X_train, y_train)
    # Make predictions on testing data
    y_pred = model.predict(X_test)
    # Compute metrics
    score = model.score(X_test, y_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    std = np.std(y_test)
    # create dict to store results
    results_dict = {
        'Model Detail': model_info,
        'Score': score,
        'R2': r2,
        'Mean Squared Error': mse,
        'Root Mean Squared Error': rmse,
        'Standard Deviation': std
    }
    return results_dict
```

Saved multiple variations of the data

```
yearly_cases = tb_filtered.groupby("Year", as_index=False).sum()
yearly_cases.drop(columns = states, inplace=True)

#add lags to original df
tb_df_with_lags = tb_filtered.copy()
for lag in range(1, 3):
    tb_df_with_lags[f'lag_{lag}'] = tb_df_with_lags['Sum'].shift(lag)
tb_df_with_lags.dropna(inplace=True)

#group df with lags by year
yearly_cases_with_lags = tb_df_with_lags.groupby("Year", as_index=False).sum()
yearly_cases_with_lags.drop(columns = states, inplace=True)

#drop age bins
no_ages = tb_filtered.copy()
no_ages.drop(columns = ['0-4', '14-May', '15-24', '25-34', '35-44','45-54', '55-64', '65+'], inplace=True)

#group df with droped age bins by year
no_state_no_age = no_ages.copy()
no_state_no_age = no_state_no_age.groupby('Year', as_index=False).sum()
no_state_no_age.drop(columns = states, inplace=True)

#drop age bins from df with lags
no_ages_yes_lags = tb_df_with_lags.copy()
no_ages_yes_lags.drop(columns = ['0-4', '14-May', '15-24', '25-34', '35-44','45-54', '55-64', '65+'], inplace=True)

#group df with dropped age bins and added lag by year
no_state_no_ages_yes_lags = no_ages_yes_lags.groupby('Year', as_index=False).sum()
no_state_no_ages_yes_lags.drop(columns = states, inplace=True)

dfs_and_info = {
    "By States, with age bins, no lag": tb_filtered, "By year, with age bins, no lag": yearly_cases,
    "By state, with age bins, and lags": tb_df_with_lags,"By year, with age bins, and lags": yearly_cases_with_lags,
    "By state, no age bins, no lags": no_ages,"By year, no age bins, no lags": no_state_no_age,
    "By state, no age bins, and lags": no_ages_yes_lags,"By year, no age bins, and lags": no_state_no_ages_yes_lags
```

Created a DataFrame of the models metrics

```python
#create df and columns to hold the results from running 1st LR model with testing/training split
new_results_df_split = pd.DataFrame(columns=['Model Detail', 'Score', 'R2', 'Mean Squared Error',
       'Root Mean Squared Error', 'Standard Deviation'])

#loop through dict of dfs and info and add metrics to results df
for model_info, df in dfs_and_info.items():
    results_dict = lr_model(df, model_info)
    #turn dict that is output from function into df
    new_row = pd.DataFrame([results_dict])
    # concat to result df
    new_results_df_split = pd.concat([new_results_df_split, new_row], ignore_index=True)
new_results_df_split
```

6]:

	Model Detail	Score	R2	Mean Squared Error	Root Mean Squared Error	Standard Deviation
0	By States, with age bins, no lag	1.000000	1.000000	3.559302e-02	0.188661	336.179458
1	By year, with age bins, no lag	0.999229	0.999229	1.888920e+00	1.374380	49.500000
2	By state, with age bins, and lags	1.000000	1.000000	3.533480e-02	0.187976	336.179458
3	By year, with age bins, and lags	0.999075	0.999075	2.267242e+00	1.505736	49.500000
4	By state, no age bins, no lags	0.905029	0.905029	1.073331e+04	103.601678	336.179458
5	By year, no age bins, no lags	-799.897457	-799.897457	1.962399e+06	1400.856522	49.500000
6	By state, no age bins, and lags	0.904440	0.904440	1.079990e+04	103.922560	336.179458
7	By year, no age bins, and lags	0.996415	0.996415	8.785369e+00	2.964012	49.500000

Comparison of both models metrics

In [49]:
```
#sort not split by MSE
both_options2.sort_values('Mean Squared Error2')
```

Out[49]:

	Model Detail	Score	R2	Mean Squared Error	Root Mean Squared Error	Standard Deviation	Score2	R22	Mean Squared Error2	Root Mean Squared Error2	Standard Deviation2
0	By States, with age bins, no lag	1.000000	1.000000	3.559302e-02	0.188661	336.179458	0.999999	0.999999	0.101050	0.317884	430.440785
2	By state, with age bins, and lags	1.000000	1.000000	3.533480e-02	0.187976	336.179458	0.999999	0.999999	0.101130	0.318009	430.838780
3	By year, with age bins, and lags	0.999075	0.999075	2.267242e+00	1.505736	49.500000	1.000000	1.000000	1.222185	1.105525	2505.026349
1	By year, with age bins, no lag	0.999229	0.999229	1.888920e+00	1.374380	49.500000	1.000000	1.000000	1.542953	1.242157	2539.841401
7	By year, no age bins, and lags	0.996415	0.996415	8.785369e+00	2.964012	49.500000	0.999994	0.999994	37.247184	6.103047	2505.026349
6	By state, no age bins, and lags	0.904440	0.904440	1.079990e+04	103.922560	336.179458	0.962689	0.962689	6925.768466	83.221202	430.838780
4	By state, no age bins, no lags	0.905029	0.905029	1.073331e+04	103.601678	336.179458	0.962345	0.962345	6976.728711	83.526814	430.440785
5	By year, no age bins, no lags	-799.897457	-799.897457	1.962399e+06	1400.856522	49.500000	0.954083	0.954083	296202.533609	544.244921	2539.841401

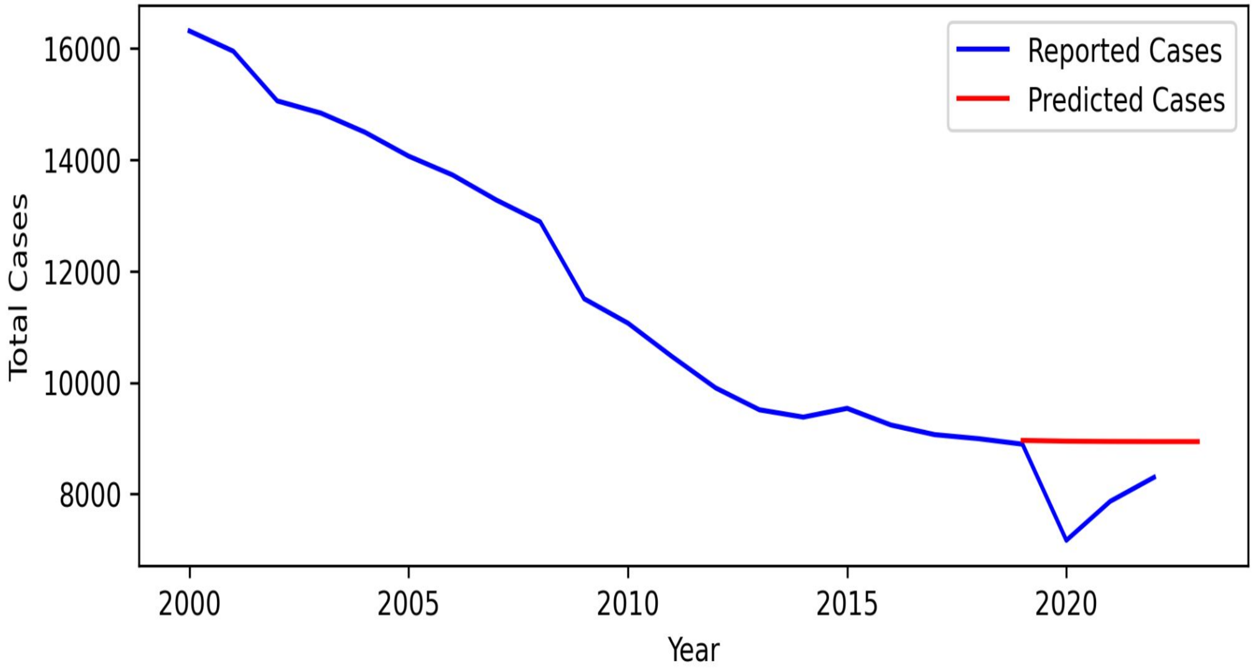
Selected Model

Model Detail	Score	R2	Mean Squared Error	Root Mean Squared Error	Standard Deviation
By year, no age bins, and lags	0.996415	0.996415	8.785369	2.964012	49.5

Year	Sum	LR_cases_predicted	Difference
2019	8895	8962.651209	67.651209
2020	7170	8949.332451	1779.332451
2021	7870	8943.980071	1073.980071
2022	8301	8942.116989	641.116989

Our Prediction

Reported TB cases and predicted TB cases without COVID



Time Series Forecasting

TB cases over the years with and without COVID to see the pandemic's impacts on TB trends

Let the model predict future values based on the data from a certain time period

Lags

Limitations and Implications

Study Limitations:

- Univariate analysis: using time as the single predictor variable
- Limited scope for understanding factors influencing TB
- Challenges in measuring the true relationship between COVID-19 and TB
- Future studies should utilize more granular data on individual patients

Public Health Implications:

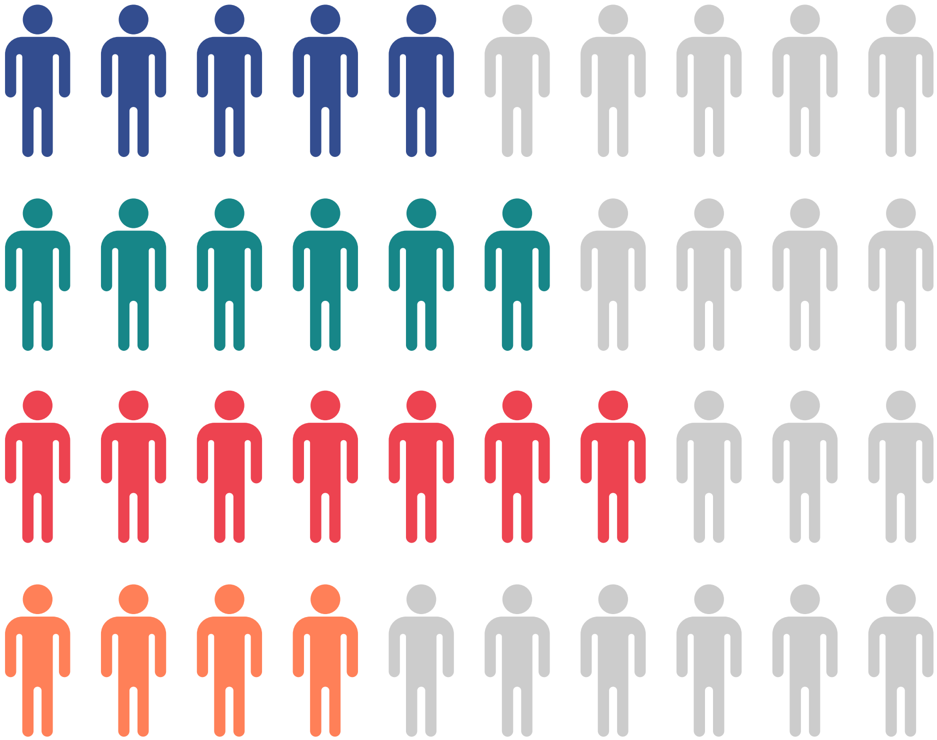
- Under-reporting: disruptions in disease surveillance activities
- Impaired access to healthcare facilities
- Delayed diagnosis and treatment may further complicate disease and health
- Increased risk of TB transmission within communities if untreated cases persist

Future Directions

Recommendations:

Advocate for stronger public health emergency preparedness.

Ensure essential diseases like TB receive adequate attention during future pandemics.



Thank you !

Feel free to ask any and all questions

Source: Slidesgo