Machine Learning project

Oil price prediction

Bogdana Kolic, Clemence Mottez, Matheo Le Masson

Motivations

- Crucial for industries and decision-makers
  - Impact on the global economy, investment and trading strategies, supply and demand dynamics, risk management and market forecasting
- Usual data: year and price
  - More comprehensive approach with 23 features
- Feature selection
  - determine what contribute to oil prices
  - deepen our understanding of the complex dynamics driving oil prices
  - provides us with enhanced predictive capabilities

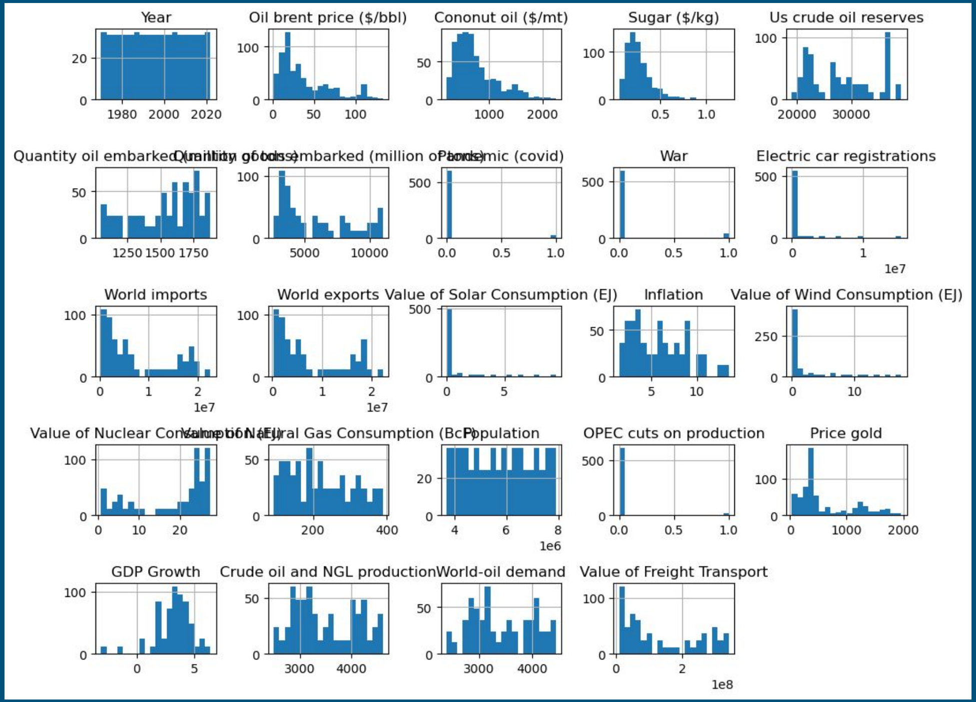
Data

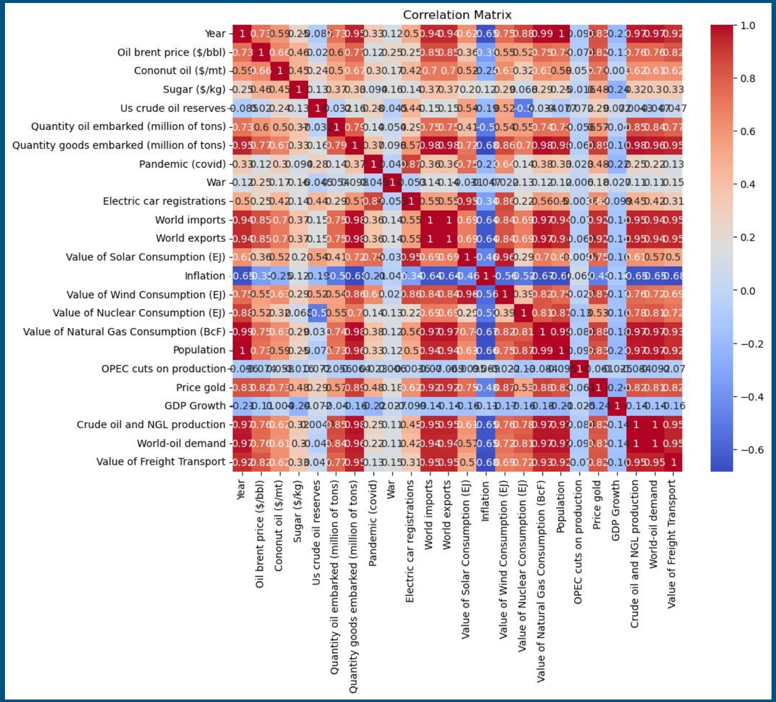
- How did we create our data set
  - Decided by ourselves what we thought could influence the price, asked domain experts
  - Sources: macrotrends.net, unctadstat.unctad.org, datasource.kapsarc.org, data.worldbank.org, tradingeconomics.com
  - Handle missing values
  - Normalization

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

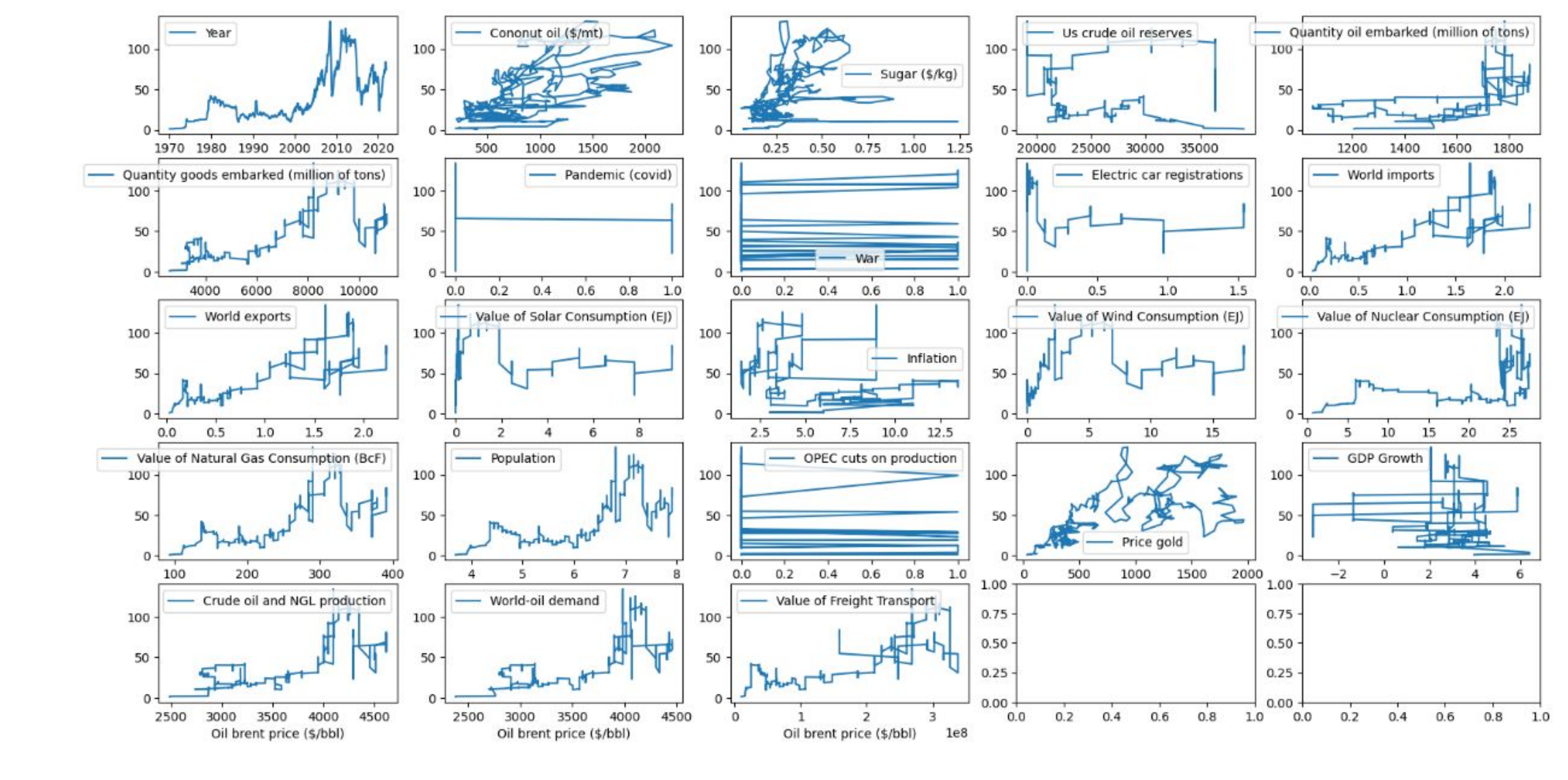
- Our dataset
  - 23 parameters, total of 24 columns
  - Monthly data from 1970 to 2022, total of 624 rows

Data visualization





Data visualization



Data preprocessing

- Removing instances with missing values
- Allow for the option of averaging the data over several months
    - 24th feature - average price of oil in the past months?
- Manual feature selection
- Normalization
- Splitting the data into training and test sets

Manual feature selection and averaging

In [6]:
```python
# discovered function rolling() at:
# https://stackoverflow.com/questions/60274621/calculate-average-of-specified-range-of-values-in-pandas-column-and-store-as-ano
# and function shift() at:
# https://stackoverflow.com/questions/10982089/how-to-shift-a-column-in-pandas-dataframe

# features that shouldn't ever be averaged
no_avg_features = ['Year', 'War', 'OPEC cuts on production', 'Pandemic (covid)']
# features we choose to use
features = ['Year', 'Cononut oil ($/mt)', 'Sugar ($/kg)', 'Us crude oil reserves',
            'Quantity oil embarked (million of tons)', 'Quantity goods embarked (million of tons)', 'Pandemic (covid)', 'War',
            'Electric car registrations', 'World imports', 'World exports', 'Value of Solar Consumption (EJ)', 'Inflation',
            'Value of Wind Consumption (EJ)', 'Value of Nuclear Consumption (EJ)', 'Value of Natural Gas Consumption (BcF)',
            'Population', 'OPEC cuts on production', 'Price gold', 'GDP Growth', 'Crude oil and NGL production',
            'World-oil demand', 'Value of Freight Transport']

df = data.loc[:, features]

num_months_avg = 1 # over how many months to take average

take_average = 0 # 1 or 0, indicates whether we do the averaging

for feature in features:
    if feature not in no_avg_features:
        df[feature] = df[feature].rolling(num_months_avg).mean() # average
    elif num_months_avg > 1:
        df[feature] = df[feature].shift(-1) # if taking average, adjust the other features for forecasting

df.dropna(inplace=True)
df.describe()
```

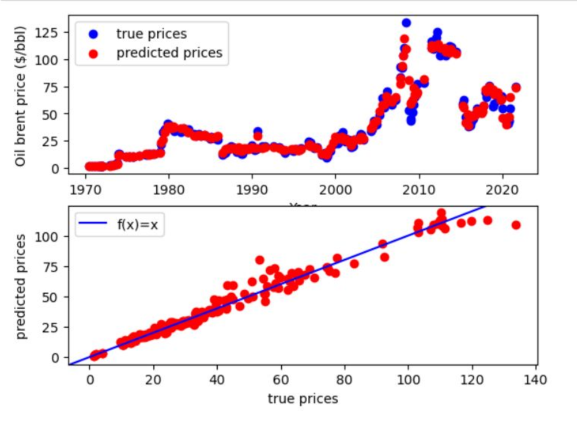
Out[6]:

Models

- Which models did we study?
  - Linear and Polynomial regression, decision trees, Random forest
- Hyperparameters
  - Ridge and Lasso penalty for Linear and Polynomial regression
  - Number of estimators, maximal depth, … for decision trees
- Which one did we choose for feature selection
  - xgbRegressor + random forest
  - The decision trees seem to work the best, different scores based on different metrics and features

Training

- Scoring and error metrics
  - Minimizing the *Mean squared error*
  - Maximizing *R2 score*
  - Minimizing the *Cross-validation with mean squared error*
    - Over the whole dataset
- Visualizing the results
  - two plots
  - ex: bagging regressor performance, all features



Hyperparameter tuning

- Initialize several options and test all the possibilities
- Evaluation done by cross-validation on the training set
- example on AdaBoost model:

AdaBoost

In [27]:
```
# tuning the hyper-parameters:
depths = [1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
estimators = [1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
min_mse = math.inf
max_depth = 0
num_estimators = 0

for d in depths:
     for e in estimators:
            for sf in samples_features:
                adaboost_model = AdaBoostRegressor(DecisionTreeRegressor(max_depth=d),n_estimators=e)
                adaboost_model.fit(x_train, y_train)
                prediction = adaboost_model.predict(x_test)
                mse = mean_squared_error(prediction, y_test)
                if mse<min_mse:
                    min_mse = mse
                    max_depth = d
                    num_estimators = e
                    max_samples_features = sf
adaboost_model = AdaBoostRegressor(DecisionTreeRegressor(max_depth=max_depth),n_estimators=num_estimators)
adaboost_model.fit(x_train, y_train)
y_predict_adaboost = adaboost_model.predict(x_test)
adaboost_r2_score = r2_score(y_test, y_predict_adaboost)
adaboost_mse = mean_squared_error(y_test, y_predict_adaboost)
adaboost_cv = cross_validate(adaboost_model, df, oil_prices, 4, mean_squared_error, np.mean, False)
print("max_depth: ", max_depth)
print("num_estimators: ", num_estimators)
print("mse: ", adaboost_mse)
print("r2 score: ", adaboost_r2_score)
print("cv: ", adaboost_cv)
model_results["adaboost"] = (adaboost_r2_score, adaboost_mse, adaboost_cv)
```

Interpretation of the results

- By sorting the final scores/ errors
  - example on the left: raw features
  - example on the right: with additional average oil price feature and averaging

Summing up the results

```
In [35]: comparison_r2 = sorted(model_results.items(), key = lambda x:x[1], reverse = True)
         print(comparison_r2)

[('adaboost', (0.9829371845001577, 15.492112234042551, 713.3105756410255)), ('random_forest', (0.9775000018764277, 20.428779541
06389, 698.034808000225)), ('polynomial_ridge', (0.9762947996355764, 21.52303789368814, 6762.762089856493)), ('bagging', (0.975
035427040476, 22.666479993730352, 640.6171660665955)), ('gradient_boosting', (0.9715275884435935, 25.851407430958524, 620.13537
79207891)), ('xgb_regressor', (0.9698048834023651, 27.415530294802856, 621.124659375191)), ('polynomial_lasso', (0.967417781600
923, 29.582889428608727, 2343.74914969669)), ('linear_regression', (0.9361874946807662, 57.938298304294435, 1308536400.38078
8)), ('linear_ridge', (0.9359970842441161, 58.11118066682978, 67234304.81083053)), ('linear_lasso', (0.9192127144482721, 73.350
47928418226, 2546211.101777478)), ('polynomial', (0.894844984516612, 95.47505813774859, 4.681250740188116e+23))]

In [36]: comparison_mse = sorted(model_results.items(), key = lambda x:(x[1][1], x[1][0], x[1][2]))
         print(comparison_mse)

[('adaboost', (0.9829371845001577, 15.492112234042551, 713.3105756410255)), ('random_forest', (0.9775000018764277, 20.428779541
06389, 698.034808000225)), ('polynomial_ridge', (0.9762947996355764, 21.52303789368814, 6762.762089856493)), ('bagging', (0.975
035427040476, 22.666479993730352, 640.6171660665955)), ('gradient_boosting', (0.9715275884435935, 25.851407430958524, 620.13537
79207891)), ('xgb_regressor', (0.9698048834023651, 27.415530294802856, 621.124659375191)), ('polynomial_lasso', (0.967417781600
923, 29.582889428608727, 2343.74914969669)), ('linear_regression', (0.9361874946807662, 57.938298304294435, 1308536400.38078
8)), ('linear_ridge', (0.9359970842441161, 58.11118066682978, 67234304.81083053)), ('linear_lasso', (0.9192127144482721, 73.350
47928418226, 2546211.101777478)), ('polynomial', (0.894844984516612, 95.47505813774859, 4.681250740188116e+23))]

In [37]: comparison_cv = sorted(model_results.items(), key = lambda x:(x[1][2], x[1][0], x[1][1]))
         print(comparison_cv)

[('gradient_boosting', (0.9715275884435935, 25.851407430958524, 620.1353779207891)), ('xgb_regressor', (0.9698048834023651, 27.
415530294802856, 621.124659375191)), ('bagging', (0.975035427040476, 22.666479993730352, 640.6171660665955)), ('random_forest',
(0.9775000018764277, 20.42877954106389, 698.034808000225)), ('adaboost', (0.9829371845001577, 15.492112234042551, 713.310575641
0255)), ('polynomial_lasso', (0.967417781600923, 29.582889428608727, 2343.74914969669)), ('polynomial_ridge', (0.97629479963557
64, 21.52303789368814, 6762.762089856493)), ('linear_lasso', (0.9192127144482721, 73.35047928418226, 2546211.101777478)), ('lin
ear_ridge', (0.9359970842441161, 58.11118066682978, 67234304.81083053)), ('linear_regression', (0.9361874946807662, 57.93829830
4294435, 1308536400.380788)), ('polynomial', (0.894844984516612, 95.47505813774859, 4.681250740188116e+23))]
```

Summing up the results

```
In [35]: comparison_r2 = sorted(model_results.items(), key = lambda x:x[1], reverse = True)
         print(comparison_r2)

[('polynomial_ridge', (0.9843756617933525, 14.826663207770741, 18610.141423557034)), ('adaboost', (0.9684282869268801, 29.95987
0967741924, 279.1755549790532)), ('xgb_regressor', (0.9669511247550565, 31.36161904405848, 401.37278861869487)), ('gradient_boo
sting', (0.961981446048883, 36.077578942821695, 304.7947511354112)), ('bagging', (0.9565630346940646, 41.219362179750945, 595.2
714603866647)), ('polynomial_lasso', (0.9547047134762644, 42.98280985118291, 2548.975667086903)), ('random_forest', (0.95060119
43072742, 46.87683056940849, 251.59578204175645)), ('linear_regression', (0.9416399040316498, 55.380616846079405, 294858597.990
079)), ('linear_ridge', (0.9415360493915167, 55.47916935764513, 266136460.37427446)), ('linear_lasso', (0.9333589963836735, 63.
23875631588165, 644679336.2288257)), ('polynomial', (-601.8732784942586, 572094.5705972947, 1.2229615249613017e+22))]

In [36]: comparison_mse = sorted(model_results.items(), key = lambda x:(x[1][1], x[1][0], x[1][2]))
         print(comparison_mse)

[('polynomial_ridge', (0.9843756617933525, 14.826663207770741, 18610.141423557034)), ('adaboost', (0.9684282869268801, 29.95987
0967741924, 279.1755549790532)), ('xgb_regressor', (0.9669511247550565, 31.36161904405848, 401.37278861869487)), ('gradient_boo
sting', (0.961981446048883, 36.077578942821695, 304.7947511354112)), ('bagging', (0.9565630346940646, 41.219362179750945, 595.2
714603866647)), ('polynomial_lasso', (0.9547047134762644, 42.98280985118291, 2548.975667086903)), ('random_forest', (0.95060119
43072742, 46.87683056940849, 251.59578204175645)), ('linear_regression', (0.9416399040316498, 55.380616846079405, 294858597.990
079)), ('linear_ridge', (0.9415360493915167, 55.47916935764513, 266136460.37427446)), ('linear_lasso', (0.9333589963836735, 63.
23875631588165, 644679336.2288257)), ('polynomial', (-601.8732784942586, 572094.5705972947, 1.2229615249613017e+22))]

In [37]: comparison_cv = sorted(model_results.items(), key = lambda x:(x[1][2], x[1][0], x[1][1]))
         print(comparison_cv)

[('random_forest', (0.9506011943072742, 46.87683056940849, 251.59578204175645)), ('adaboost', (0.9684282869268801, 29.959870967
741924, 279.1755549790532)), ('gradient_boosting', (0.961981446048883, 36.077578942821695, 304.7947511354112)), ('xgb_regresso
r', (0.9669511247550565, 31.36161904405848, 401.37278861869487)), ('bagging', (0.9565630346940646, 41.219362179750945, 595.2714
603866647)), ('polynomial_lasso', (0.9547047134762644, 42.98280985118291, 2548.975667086903)), ('polynomial_ridge', (0.98437566
17933525, 14.826663207770741, 18610.141423557034)), ('linear_ridge', (0.9415360493915167, 55.47916935764513, 266136460.3742744
6)), ('linear_regression', (0.9416399040316498, 55.380616846079405, 294858597.990079)), ('linear_lasso', (0.9333589963836735, 6
3.23875631588165, 644679336.2288257)), ('polynomial', (-601.8732784942586, 572094.5705972947, 1.2229615249613017e+22))]
```

- Conclusion:
  - similar results that change with each run
  - decision trees perform well
  - our choice is the XGBRegressor

Feature selection

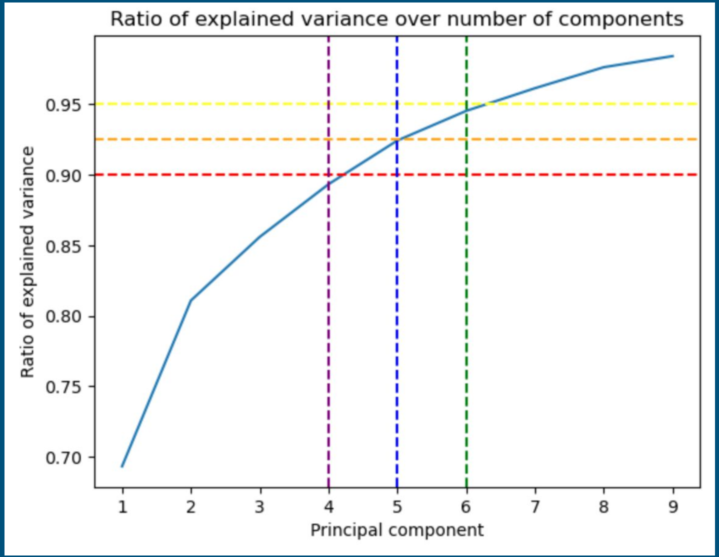
Why?

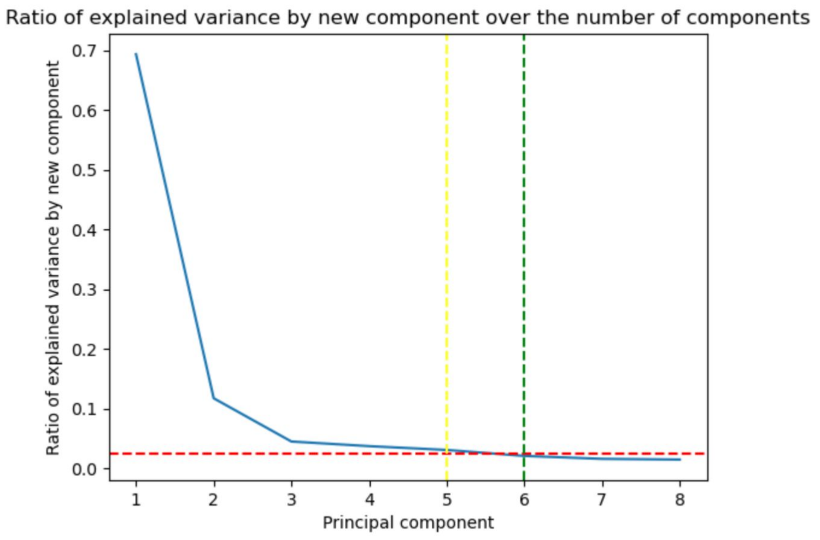
- Data 624x24!!!
    - Reduce computational cost

- Noise, redundant, irrelevant information have negative impact
    - Improve the performance of the model

- Capture essential dynamics of the oil market
    - Facilitate interpretation
    - What really predicts the price of oil?

PCA as an answer to: How many features should we keep?



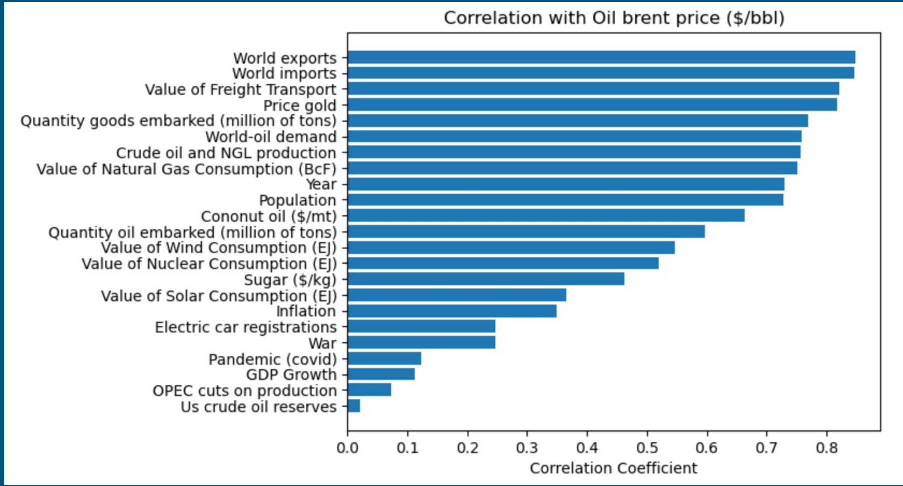


5 components explain 90% of variance

Components after the 5th explain less than 2.5% of variation

Filter models

- Computationally efficient
- Involve statistical measures such as correlation
    - Measures the linear relationship between each feature and the target variable



Embedded models

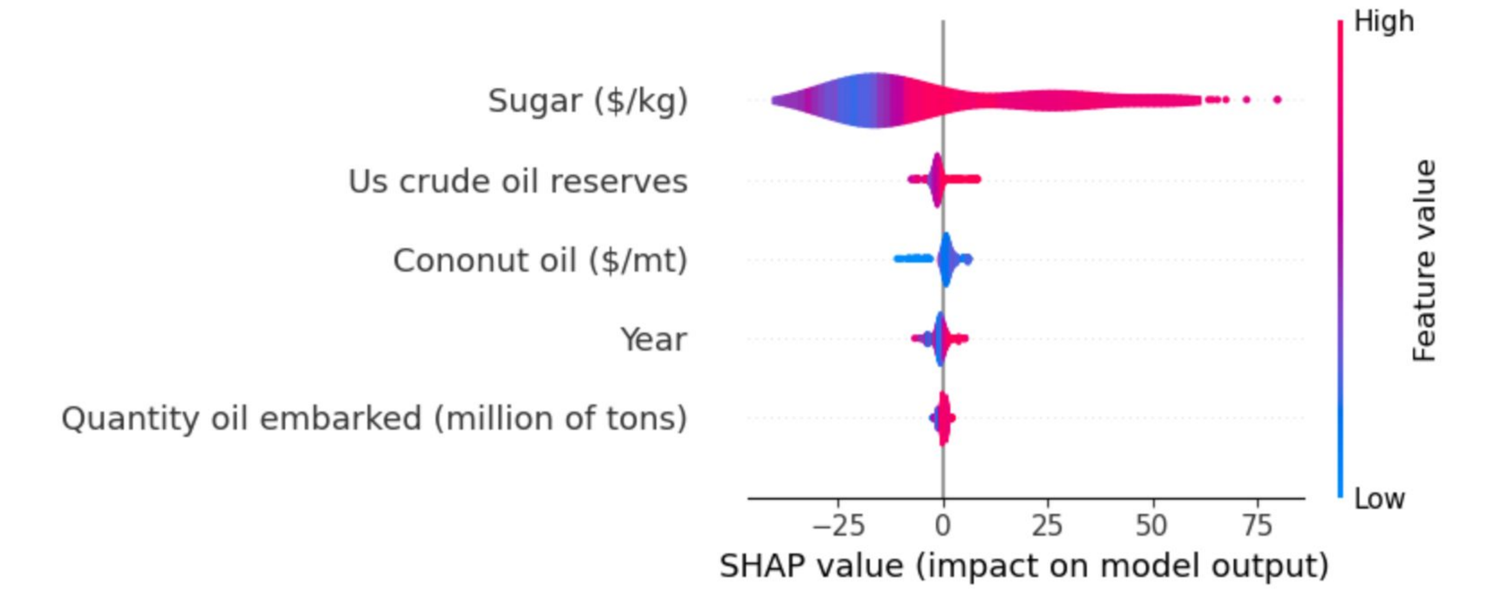
- Incorporates feature selection within the model training process
  - Optimizes both the model's performance and feature selection simultaneously
- Select relevant features based on their contribution to the model's accuracy

SHAP

- Interpretability tool
- Gain insights into feature importance and guide the feature selection process
- Show global contribution
    - Computed for each feature and used to rank the importance of features
- Show local feature contribution
    - for each instance

Embedded models

- XGboost + SHAP



Wrapper models

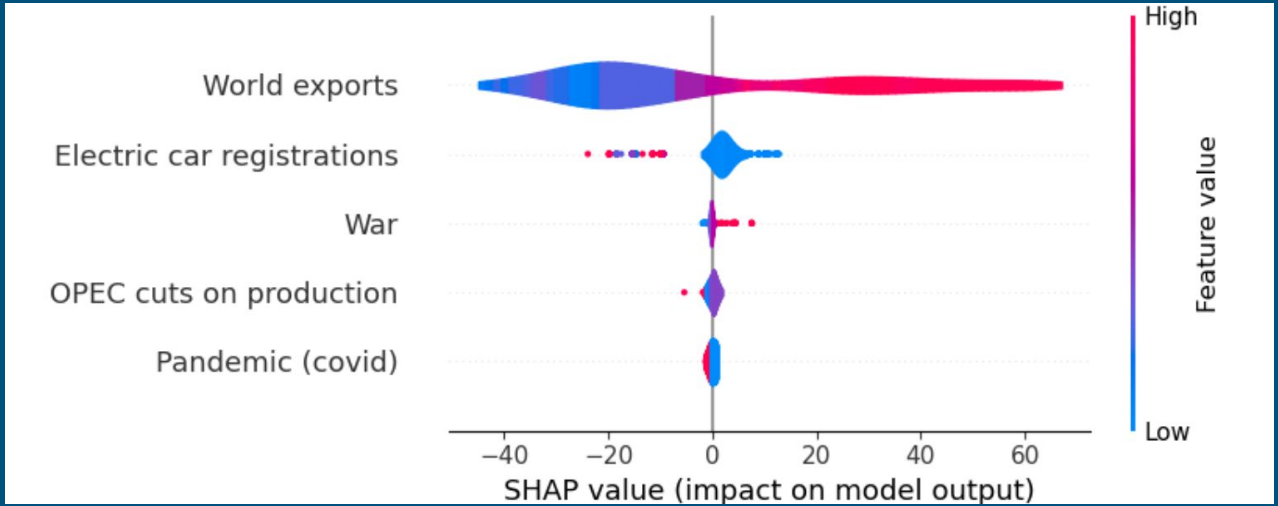
- Iteratively select and evaluate different subsets of features to find the subset that yields the best performance.
- Computationally expensive but accurate results

- Used Random Forest and XGB models

Wrapper models

Sequential Forward Selection

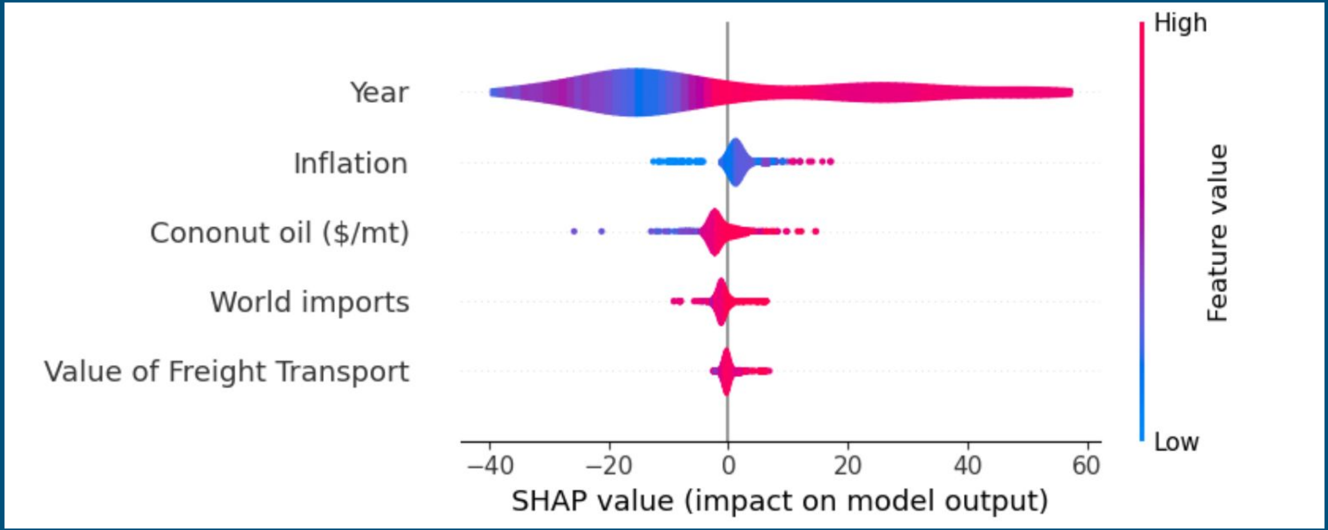
- Starts with an empty set of features
- Iteratively adds features based on the best-performing subset



Wrapper models

Recursive Feature Elimination (RFE)

- Starts with all features
- iteratively eliminates the least important feature



Wrapper models

Boruta

- Compared importance of features with random shadow features
  - A feature is important if it can do better than the best randomized feature

- Used eBoruta (extension of Boruta that already uses the SHAP importance and that is mode agnostic)

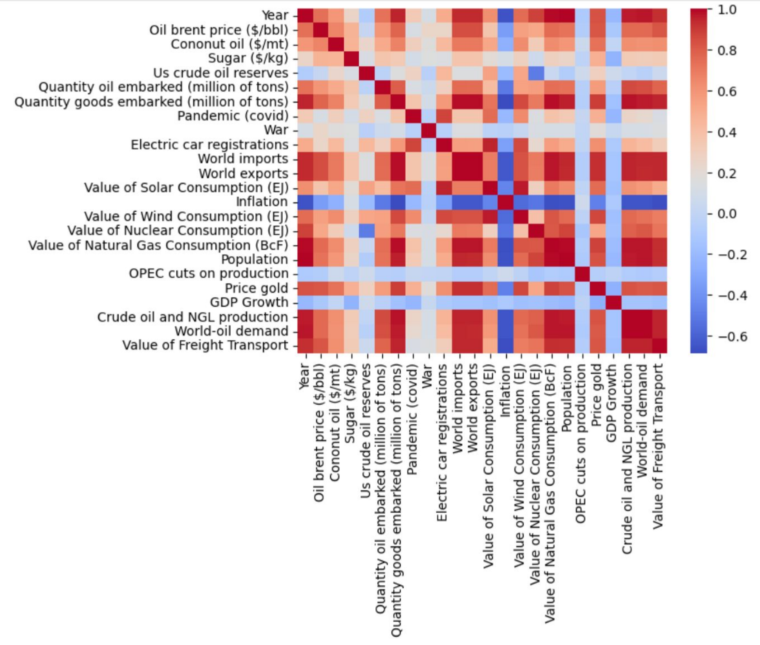
Feature	Importance
Year	20.353813
Inflation	2.827759
Cononut oil ($/mt)	2.487869
World imports	1.869873
Sugar ($/kg)	1.062382

Selection

- Year
- World imports
- World exports
- Inflation
- Price of Gold
- War
- OPEC cuts on production

Correlation

- Small selection so we don’t want too correlated variables
- Threshold at 0.95

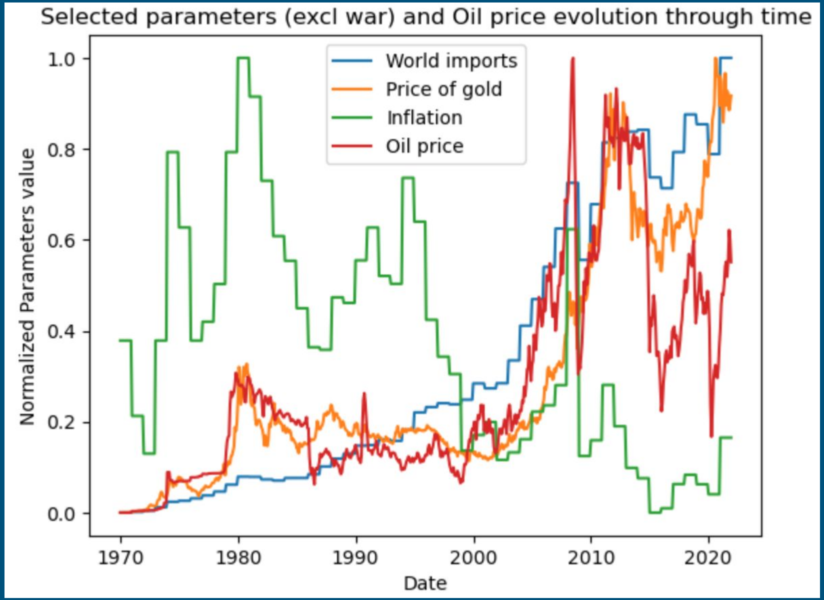


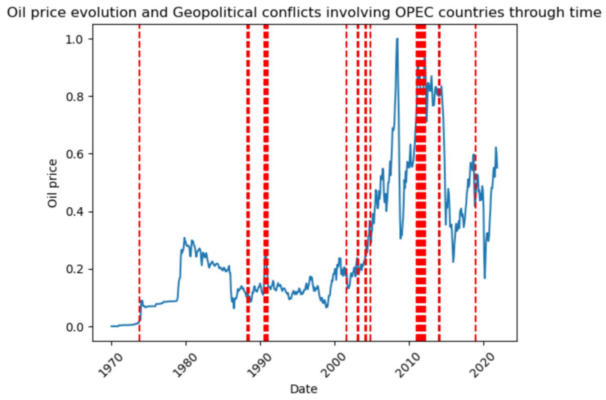
Result:

- World export and world imports

Final selection

- Year
- World imports
- ~~World exports~~ <- too correlated
- Inflation
- Price of Gold
- War
- ~~OPEC cuts on production~~ <- only 5 features selected





MSE

Data / Model	XGB Regressor	Decision Tree	Random Forest
Raw Data	**621**	643	680
Data with PCA	**253**	390	318
Selected Data	**372**	474	555

Limits - What could do after?

- Limits:
  - don’t know if causation or correlation
    for example price of gold certainly a correlation
- After:
  - Network model
  - Future analysis and predictions
    make predictions for future oil prices
    Incorporate new data as it becomes available
  - How to optimally compare models? With data modifications
  - Optimize algorithm we used (for example SHAP is computationally very expensive)
  - New model: RNNs with LSTM to avoid Vanishing Gradient problem

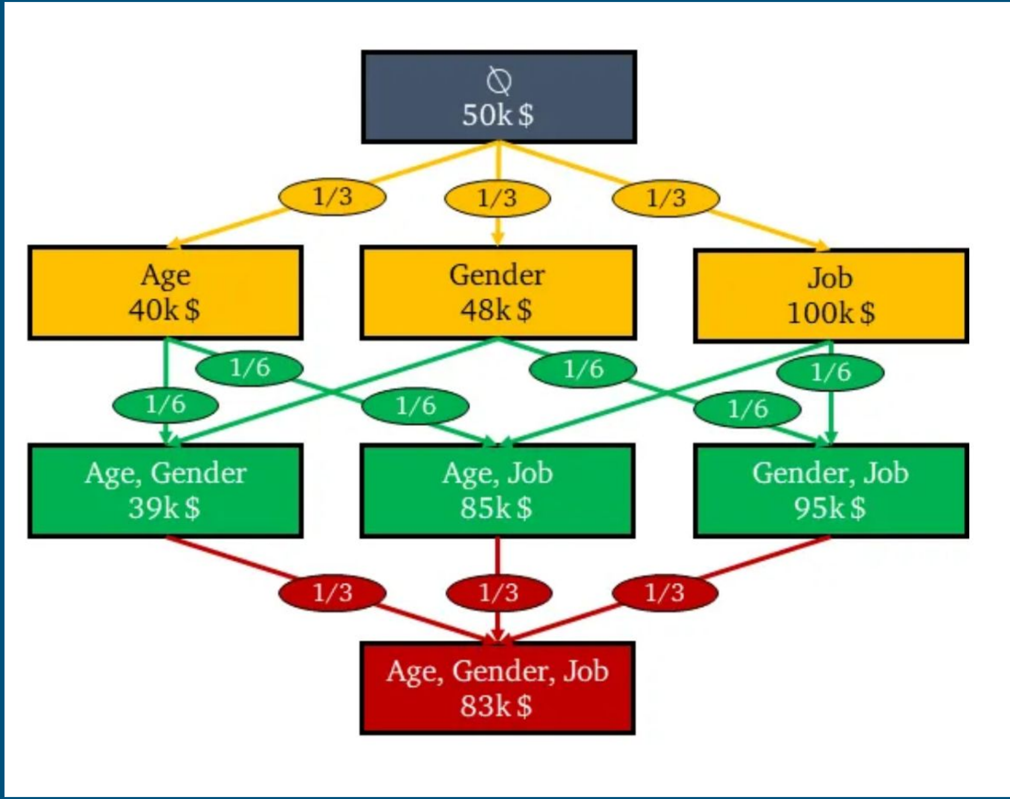
Network model

- Much better accuracy
    - MSE from cross-validation around 220 (compared to XGB around 400)
- Relatively simple:
    - 3 Hidden Dense layers with Relu activation
    - Dropout regularization to avoid overfitting

Other info for questions

- slides to explain the different models we used
- slides with term explanations, …

SHAP values



∅
50k $
1/3
1/3
1/3
Age
40k $
Gender
48k $
Job
100k $
1/6
1/6
1/6
1/6
1/6
1/6
Age, Gender
39k $
Age, Job
85k $
Gender, Job
95k $
1/3
1/3
1/3
Age, Gender, Job
83k $