Introduction

The Office for National Statistics (ONS) provides a range of economic indicators that can be used to gain insights into the UK's economy. In this project, we will explore the predictive power of these ONS indicators to forecast economic trends by focusing on the Business Insights and Workforce indicator category which includes data about small businesses, job adverts, and VAT, just to mention a few. We will use machine learning techniques to analyze the data and identify the most important indicators in predicting economic performance.

We will start by importing the necessary libraries and datasets to accomplish this. Next, we will create a class to help us clean the data. We will then perform dataset cleaning, deal with missing values, and merge the datasets. Finally, we will use ensemble feature selection to identify the most important indicators in predicting economic performance.

SECTION 1 : Data Pre-Processing

To begin, the necessary libraries were imported, including NumPy, Pandas, Matplotlib, and Scikit-learn. We also imported the ONS datasets containing the economic indicators to be analysed.

In total, there were 5 datasets stored in Excel formats in the Business insights and workforce indicator categories. The majority of the data were daily data so resampling was also performed to transform this daily data to monthly data. The following table describes each dataset in detail along with their originating source.

Indicator	Source	Description	Back series
Business insights and workforce			
Online job adverts	Adzuna	Weekly experimental online job advert indices covering the UK job market, using data from job advert aggregating website Adzuna	07/02/2018
Redundancies	Insolvency services	Weekly advanced notification of potential redundancies from HR1 forms submitted by employers to the Insolvency Service's Redundancy Payments Service	07/04/2019
Company incorporations, voluntary dissolutions, and compulsory dissolutions	Companies House	Weekly data for company incorporations, voluntary dissolutions, and compulsory dissolution first gazettes in the UK	04/01/2019
Monthly data on sales and jobs in small businesses	Xero	Monthly data on sales and jobs in small businesses, taken from Xero, a global cloud-based accounting software platform with 785,000 small business subscribers in the UK	Jan 2019
VAT new businesses and business turnover	HMRC	Monthly Value Added Tax (VAT) diffusion indexes and new VAT reporters	Feb 2012

Figure 1: Business Insights and Workforce Indicator Details

Dataset Cleaning:
To make the data-cleaning process more efficient, we created a class to help us clean the data. This class contains functions to handle various data-cleaning tasks and parameters to handle peculiarities encountered within the datasets. They are described as follows.

```
class clean:

    def __init__ (self,df,query_dataset=False,setheader= False,header_int=0,droprows=False,rows_to_drop=[],
                  drop_cols=False,columns_to_drop=[],to_transpose=False,to_datetime=False,dayfirst=False,nancolumn=False,columns_to_keep=[],
                  converttopct=False,converttonan=False,rename_cols=False,to_append='',replace_nan=False):
```

Figure 2: Initialisation of the 'clean' class

query_dataset: Set to True if we want to filter the dataset based on an expression, for example, one of the datasets had data from over 30 countries but only observations from the United Kingdom were required. This was useful in that case.
setheader: Set to True to specify the header of the dataset as most of them had extra rows with sentences before the start of the actual data. This was from the source.
header_int: Specify the integer row of the header
droprows: Set to True if there are any unwanted rows that we'd like to drop
rows_to_drop: a list of row numbers
drop_cols: Set to True if there are any unwanted columns that we'd like to drop
columns_to_drop: a list of column numbers
to_transpose: Set to True if the dataset needs to be transposed
to_datetime: Set to True if the index of the dataset needs to be converted to a DateTime format
dayfirst: Set to True if we want the original dates to be parsed as with the 'day' as the first element. Works with to_datetime
nancolumn: Set to True if we want to exclude columns that have all missing values from the dataset
columns_to_keep: a list of the preferred columns to keep, works with nancolumn
converttopct: Set to True if we want the values of specified columns to be converted to percentages.
converttonan: Set to True if we want the values of specified columns to be converted to 'np.nan'. This was necessary as some of the missing data was originally represented as strings.
rename_cols: Set to True if we want to rename the columns
to_append: a string to append to the end of each column in the dataset. This is to allow for easy identification of the exact dataset to which the column belongs
replace_nan: Set to True if we want to replace missing values with another value

Also, within the 'clean' class, a function 'final_clean' was created to sequentially apply the various cleaning functions based on the specified parameters as seen in the figure below.

```
    def final_clean(self):
        if self.query_dataset == True:
            self.df = self.df.query("Name == 'United Kingdom'")
        if self.setheader == True:
            header_array = self.df.iloc[self.header_int].ravel()
            self.df.set_axis(header_array,axis=1,inplace=True)
        if self.droprows == True:
            self.df.drop(self.rows_to_drop,inplace=True)
        self.df.set_index(self.df.columns[0] ,inplace=True)
        self.df.index.names = ['Date']
        if self.drop_cols==True:
             self.df.drop(columns=self.columns_to_drop,inplace=True)
        if self.to_transpose == True:
            self.df = self.df.T
        if self.to_datetime == True and self.dayfirst==True:
            self.df.index = pd.to_datetime(self.df.index,dayfirst=True)
        else:
            self.df.index = pd.to_datetime(self.df.index)
        if self.replace_nan == True:
            self.df.iloc[:,0] = self.df.iloc[:,0].replace(np.nan, '0%')
        if self.nancolumn==True:
            self.df = self.df.loc[:,self.columns_to_keep]
        if self.converttopct==True:
            self.convert_to_pct()
        if self.converttonan==True:
            self.convert_to_nan()
        self.df= self.resampleandchangeformat()
        a = dt.strptime(self.df.index[0], '%b %Y')
        b = dt.strptime('Jan 2020', '%b %Y')
        if a < b:
            self.df = self.df['Jan 2020':'Dec 2022']
        else:
            self.df = self.df[self.df.index[0]:]
        if self.renamecols == True:
            column_list = self.getnewheaders()
            for label in column_list:
                to_append = '_' + self.toappend
                new_column_name = str(label)+to_append
                self.df.rename(columns={label:new_column_name},inplace=True)
        return self.df
```

Figure 3: ‘ Final_clean’ function

Next, for each of the datasets, we set the class instance, specified the required parameters and then called the final_clean function.

[85]:
```
#Before cleaning
company_incorporations_vol.head()
```

[85]:

	Number of weekly company voluntary dissolution applications recorded by Companies House [note 1][note 2]	Unnamed: 1	Unnamed: 2
0	This worksheet contains one table. Some cells ...	NaN	NaN
1	[x] = Data are unavailable	NaN	NaN
2	Companies House	NaN	NaN
3	Week ending to	Weekly	4 week average
4	2019-01-04 00:00:00	3095	[x]

Figure 4: Company Involuntary dissolutions dataset before cleaning

[86]:
```
#Setting required variables
civ_header=3
civ_droprows = [0,1,2,3]
civ_dropcols = ['4 week average']

#class instance
dataset_cleaning = clean(company_incorporations_vol,setheader= True,header_int=civ_header,droprows=True,rows_to_drop=civ_droprows,
                  drop_cols=True,columns_to_drop=cii_dropcols,rename_cols=True,to_append='Voluntary_Dissolutions')
company_incorporations_vol = dataset_cleaning.final_clean()

#After cleaning
company_incorporations_vol.head()
```

[86]:

Date	Weekly_Voluntary_Dissolutions
Jan 2020	5901.40
Feb 2020	6094.75
Mar 2020	5683.50
Apr 2020	4183.50
May 2020	4476.00

Figure 5: Company Involuntary dissolutions dataset after cleaning

Merging Datasets:
After cleaning the datasets, they were merged into a single dataset to analyze the data more effectively and identify patterns and trends. The final single dataset consisted of 216 columns.

```
[111]: business_insights.info(verbose=True, show_counts=True)

<class 'pandas.core.frame.DataFrame'>
Index: 36 entries, Jan 2020 to Dec 2022
Data columns (total 216 columns):
```

Figure 6: Business Insights and Workforce Merged Datasets

Missing Values:
The data the ONS uses originates from various sources and new economic indicators are created regularly, implying that the data will have varying start dates and end dates. This led to missing values in the data which were handled using the KNN imputer. Some of the columns in the data had all null values, therefore a function was created to handle this issue.

```
[118…
#Create function for imputation of missing values usin KNN
def missingvalues(X):
    # check for columns that have all null values and convert to a list
    empty_train_columns = X.columns[X.isnull().all()].tolist()
    #drop those columns from the dataset
    X = X.drop(empty_train_columns, axis=1)
    #Impute missing values
    KNN_imputer = KNNImputer(n_neighbors=3)
    #Create new dataset
    X_new = pd.DataFrame(data=KNN_imputer.fit_transform(X), columns=X.columns)
    return X_new

X_new = missingvalues(X)
```

Figure 7: Missing values through KNN imputation

After imputation, 57 columns which consisted of only null values were dropped from the dataset.

SECTION 2: Modelling

Ensemble Feature Selection:
After all the necessary data pre-processing steps were taken, the dataset was split into the target variable, ‘y’ which was ‘Monthly GDP’ and the predictor variables, ‘X’ from the business insights merged dataset. It was then divided into training and testing data with a ratio of 70/30.
Before applying the feature selection methods, the features of the data were standardised to ensure that each feature contributes equally to the analysis and to prevent features with large values from dominating the model.

```
[119]: #Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.3, random_state=19042351)

#Scale data
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

Figure 8: Split data into training and Testing

An ensemble method was utilised for feature selection, which combined the results of different feature selection algorithms, in this case, three models. A voting procedure was then used to determine the final feature selection.
The first model was the Lasso, which operates by reducing the coefficients of minor features to zero. The LassoCV function was used to discover the best regularisation parameter before training the Lasso model. This function evaluates the Lasso regression model at various regularisation parameter values using a k-fold cross-validation approach. It fits the model to the data on k-1 folds and then assesses it on the remaining fold, repeating this process k times with different folds each time. After training the Lasso model, the feature coefficients that were not reduced to zero were saved in a variable for later use
In the following two models in the ensemble feature selection process, the Recursive Feature Elimination with Cross-Validation (RFECV) strategy was utilised, which uses a recursive approach to eliminate features that are deemed less important.
For the second model, the RFE used the gradient-boosting regressor as its estimator for choosing the required features. The gradient-boosting regressor was included in the ensemble because it is less prone to overfitting than other ensemble methods, as it also uses a regularization parameter to control the complexity of the model. After training the Gradient boosting regressor, wrapped in the RFE model, the selected features were also saved to a variable for later use
The Random Forest Regressor was utilised as the estimator for the RFE variable selection in the third model. This was also a preferred model due to its robustness to noise and overfitting because the predictions are based on an average of numerous decision trees rather than a single decision tree.
After fitting the ensemble model, the votes of the selected features were summed up and the criteria for final feature selection was the features that received a minimum of two out of three votes from the models in the ensemble.

The results are as follows:

```
Number of columns before feature selection:159
Number of columns after feature selection:73
Average accuracy score : 85.8%
 Lasso model : 81.4%
 Gradient Boosting model : 91.0%
 Random Forest model : 84.9%
['Potential redundancies' 'Employers proposing redundancies'
 'Admin / clerical / secretarial_category'
 'Catering and hospitality_category' 'Charity / voluntary_category'
 'Construction / trades_category'
 'Creative / design / arts & media_category'
 'Customer service / support_category' 'Domestic help_category'
```

Figure 9: Selected features after training ensemble model

In the end, a total of 73 features were selected to predict Monthly GDP with an average accuracy score of roughly 86%. The code contains the complete list of selected features.

Conclusion

Through this analysis, we found that some ONS indicators are more important in predicting economic performance than others. Policymakers and businesses may make informed judgements about the economy and plan for probable future trends by identifying these indicators.

In addition, our project demonstrated the value of data cleaning and preprocessing in machine learning. We were able to increase the quality of our models and derive more useful insights from the data by cleaning the data and dealing with missing values.

Overall, our effort sheds light on the predictive capacity of ONS indicators and proves the utility of machine learning in anticipating economic changes.