```python
from matplotlib.pyplot import subplots
import numpy as np
import pandas as pd
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF
from pandas.api.types import is_numeric_dtype
from scipy import stats
from statsmodels.stats.anova import anova_lm
import statsmodels.formula.api as smf
from statsmodels.graphics.regressionplots import influence_plot
import plotly.express as px
```

```python
# Display residuals plot function
def display_residuals_plot(results):
    """Display residuals plot
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                    [[https://www.statsmodels.org/stable/generated/statsmodels.regression.lin
    :return None
    """
    _, ax = subplots(figsize=(8, 8))
    ax.scatter(results.fittedvalues, results.resid)
    ax.set_xlabel("Fitted values for " + results.model.endog_names)
    ax.set_ylabel("Residuals")
    ax.axhline(0, c="k", ls="--")
```

```python
def display_studentized_residuals(results):
    """Display studentized residuals
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                       [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return None
    """
    _, ax = subplots(figsize=(8, 8))
    ax.scatter(results.fittedvalues, results.resid_pearson)
    ax.set_xlabel("Fitted values for " + results.model.endog_names)
    ax.set_ylabel("Standardized residuals")
    ax.axhline(0, c="k", ls="--")
    outliers_indexes = np.where(
        (results.resid_pearson > 3.0) | (results.resid_pearson < -3.0)
    )[0]
    for idx in range(len(outliers_indexes)):
        ax.plot(
            results.fittedvalues.iloc[outliers_indexes[idx]],
```

```
            results.resid_pearson[outliers_indexes[idx]],
            "ro",
        )
        print("Outlier rows: ")
        print(Auto.iloc[outliers_indexes])
```

```
def display_hat_leverage_plot(results):
    """Display hat leverage plot.
    The size of the bubble or point is an indicator of the influence the point has on the reg
    It is simply a multiplication of the leverage value and the absolute value of the student
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                       [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return None
    """
    student_residuals = results.resid_pearson
    infl = results.get_influence()
    hat_matrix_diag = infl.hat_matrix_diag
    data_point_indexes = np.arange(0, len(student_residuals))
    df = pd.DataFrame(
        {
            "Student Residuals": student_residuals,
            "Leverage": hat_matrix_diag,
            "Data Point": data_point_indexes,
            "Influence": np.abs(student_residuals) * hat_matrix_diag,
        }
    )
    fig = px.scatter(
        df,
        x="Leverage",
        y="Student Residuals",
        size="Influence",
        title="Influence Plot",
        hover_name="Data Point",
    )
    fig.show()
```

```
def get_influence_points(results):
    """Get high influential data points from a combination of hat_diagonal_matrix, DFBetas, 
    [[https://www.theopeneducator.com/doe/Regression/outlier-leverage-influential-points]]
    [[https://library.virginia.edu/data/articles/detecting-influential-points-in-regression-w
    [[https://online.stat.psu.edu/stat501/lesson/11/11.5]]
    We use the following cutoffs:
```

```
    Hat Leverage Cutoff: 2 * Average Hat Leverage
    DFBetas Cutoff: 3 / √n
    DFFITs Cutoff: 2 * √(p/n)
    Cooks Distance Threshold: 1.0
    Cooks p-value Cutoff: 0.05
    Studentized Residuals Cutoff: 3.0
    Studentized Residuals p-value Cutoff: 0.01
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                       [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return dataframe object that contains the high influential points as identified by the a
    """
    data_dictionary = {}
    infl = results.get_influence()
    summary_frame = infl.summary_frame()
    no_of_obs = results.nobs
    data_dictionary["n"] = no_of_obs
    no_of_parameters = len(results.params)
    data_dictionary["p"] = no_of_parameters
    print(f"n = {no_of_obs}, p = {no_of_parameters}")
    hat_matrix_diag = summary_frame["hat_diag"]
    average_hat_leverage = np.mean(hat_matrix_diag)
    data_dictionary["average_hat"] = average_hat_leverage
    print(f"Average Hat Leverage: {average_hat_leverage}")
    hat_leverage_cutoff = 2 * average_hat_leverage
    data_dictionary["hat_leverage_cutoff"] = hat_leverage_cutoff
    print(f"Hat Leverage  Cutoff = 2 * Average Hat Leverage = {hat_leverage_cutoff}")
    beta_cutoff = 3 / np.sqrt(no_of_obs)
    data_dictionary["dfbetas_cutoff"] = beta_cutoff
    dffits_cutoff = 2 * np.sqrt(no_of_parameters / no_of_obs)
    data_dictionary["dffits_cutoff"] = dffits_cutoff
    cooks_d_cutoff = 1.0
    cooks_d_pvalue_cutoff = 0.05
    studentized_residuals_cutoff = 3.0
    studentized_residuals_pvalue_cutoff = 0.01
    data_dictionary["studentized_residuals_cutoff"] = studentized_residuals_cutoff
    data_dictionary["studentized_residuals_pvalue_cutoff"] = (
        studentized_residuals_pvalue_cutoff
    )
    data_dictionary["cooks_d_cutoff"] = cooks_d_cutoff
    data_dictionary["cooks_d_pvalue_cutoff"] = cooks_d_pvalue_cutoff

    print(f"DFBetas Cutoff = 3 / sqrt(n) = {beta_cutoff}")
```

```python
print(f"DFFITS Cutoff = 2 * sqrt(p/n) = {dffits_cutoff}")
print(f"Cooks Distance Cutoff = {cooks_d_cutoff}")
print(f"Cooks Distance p-value Cutoff = {cooks_d_pvalue_cutoff}")
print(f"Studentized Residuals Cutoff = {studentized_residuals_cutoff}")
print(
    f"Studentized Residuals p-value Cutoff = {studentized_residuals_pvalue_cutoff}"
)
summary_frame["student_resid_pvalue"] = stats.t.sf(
    summary_frame["student_resid"], df=no_of_obs - no_of_parameters - 1
)
summary_frame["hat_influence"] = (
    np.abs(summary_frame["student_resid"]) * summary_frame["hat_diag"]
)
summary_frame["cooks_d_pvalue"] = infl.cooks_distance[1]

# Create query string for DFBetas Columns
dfb_cols = [col for col in summary_frame if col.startswith("dfb_")]
query_dfb = ""
for col in dfb_cols:
    query_dfb += " abs(`" + col + "`) > " + str(beta_cutoff) + " or "

# Construct query
# # Choose studentized residuals p-values that are less than p-value  cutoff
query = (
    "(student_resid_pvalue < " + str(studentized_residuals_pvalue_cutoff) + " or "
)
# Choose studentized residuals that are more than 3 SD away from mean of 0
query += "abs(student_resid) > " + str(studentized_residuals_cutoff) + ") and ("
# add DFBetas criteria
query += query_dfb
# add hat leverage criterion
query += "hat_diag > " + str(hat_leverage_cutoff) + " or "
# add DFFITS criterion
query += "abs(dffits) > " + str(dffits_cutoff) + " or "
# add Cooks distance criterion
query += " cooks_d > " + str(cooks_d_cutoff) + " or "
# add Cooks distance p-value criterion
query += "cooks_d_pvalue < " + str(cooks_d_pvalue_cutoff)
# close and
query += ")"

# Fire query for high influential points
```

```
    summary_frame = summary_frame.query(query)

    # Drop standardized residuals and DFFITS Internals from columns since we
    # choose to utilize studentized residuals and DFFITs externalized instead
    summary_frame = summary_frame.drop(columns=["standard_resid", "dffits_internal"])
    return summary_frame, data_dictionary
```

```
def display_hat_leverage_cutoffs(results):
    """Display hat leverage plot
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                       [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return None
    """
    # https://online.stat.psu.edu/stat501/lesson/11/11.2
    infl = results.get_influence()
    average_leverage_value = np.mean(infl.hat_matrix_diag)
    high_leverage_cutoff = 2 * average_leverage_value
    high_influence_cutoff = 3 * average_leverage_value
    no_of_obs = results.nobs
    _, ax = subplots(figsize=(8, 8))
    ax.scatter(np.arange(no_of_obs), infl.hat_matrix_diag)
    ax.set_xlabel("Index")
    ax.set_ylabel("Leverage")
    high_leverage_indices = np.argwhere(
        (infl.hat_matrix_diag > high_leverage_cutoff)
        & (infl.hat_matrix_diag < high_influence_cutoff)
    )
    high_leverage_values = infl.hat_matrix_diag[
        np.where(
            (infl.hat_matrix_diag > high_leverage_cutoff)
            & (infl.hat_matrix_diag < high_influence_cutoff)
        )
    ]
    ax.plot(high_leverage_indices, high_leverage_values, "yo")
    high_influence_indices = np.argwhere(infl.hat_matrix_diag > high_influence_cutoff)
    high_influence_values = infl.hat_matrix_diag[
        np.where(infl.hat_matrix_diag > high_influence_cutoff)
    ]
    ax.plot(high_influence_indices, high_influence_values, "ro")
    ax.axhline(high_leverage_cutoff, c="y", ls="--")
    ax.axhline(high_influence_cutoff, c="r", ls="-")
```

```python
def display_cooks_distance_plot(results):
    """Display cook's distance leverage plot
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                    [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return matplotlib.figure.Figurehttps://matplotlib.org/stable/api/_as_gen/matplotlib.figu
    [[https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figu
    """
    fig = influence_plot(results);
    return fig
```

```python
def display_DFFITS_plot(results):
    """Display DFFITS leverage plot
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                    [[https://www.statsmodels.org/stable/generated/statsmodels.regression
    :return matplotlib.figure.Figurehttps://matplotlib.org/stable/api/_as_gen/matplotlib.figu
    [[https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figu
    """
    fig = influence_plot(results, criterion="DFFITS")
    return fig
```

```python
# Identify least statistically significant variable or column
def identify_least_significant_feature(results, alpha=0.05):
    """Identify least significant feature
    :param results - the statsmodels.regression.linear_model.RegressionResults object
                  [[https://www.statsmodels.org/stable/generated/statsmodels.regression.li
    :param alpha - the level of significance chosen
    :return None
    """
    index = np.argmax(results.pvalues)
    highest_pvalue = results.pvalues.iloc[index]
    if highest_pvalue > alpha:
        variable = results.pvalues.index[index]
        coeff = results.params.iloc[index]
        print(
            "We find the least significant variable in this model is "
            + variable
            + " with a p-value of "
            + str(highest_pvalue)
            + " and a coefficient of "
            + str(coeff)
        )
        print(
```

```python
            "Using the backward methodology, we suggest dropping "
            + variable
            + " from the new model"
        )
    else:
        print("No variables are statistically insignificant.")
        print("The model " + results.model.formula + " cannot be pruned further.")
```

```python
# Calculate [Variance Inflation Factors(VIFs) for features
# in a model](https://www.statology.org/how-to-calculate-vif-in-python/)
def calculate_VIFs(formula, df):
    """Calculate VIFs
    :param formula - the regression formula
    :param df - the pandas dataframe
    :return the pandas datafame containing VIF information for each feature.
    """
    _, X = dmatrices(formula, data=df, return_type="dataframe")
    # calculate VIF for each explanatory variable
    vif = pd.DataFrame()
    vif["VIF"] = [VIF(X.values, i) for i in range(1, X.shape[1])]
    vif["Feature"] = X.columns[1:]
    vif = vif.set_index(["Feature"])
    return vif
```

```python
# Identify feature with highest VIF
def identify_highest_VIF_feature(vifdf, threshold=10):
    """Identify highest VIF feature
    :param vifdf - the pandas dataframe containing vif information
    :param threshold - the threshold specified to identify multicollinearity in features usi
    :return tuple with variable name and its VIF when threshold is breached
            else None
    """
    highest_vif = vifdf["VIF"].iloc[np.argmax(vifdf)]
    if highest_vif > threshold:
        variable = vifdf.index[np.argmax(vifdf["VIF"])]
        print(
            "We find the highest VIF in this model is "
            + variable
            + " with a VIF of "
            + str(highest_vif)
        )
        print("Hence, we drop " + variable + " from the model to be fitted.")
```

```
        return variable, highest_vif
    else:
        print("No variables are significantly collinear.")
```

```
# Function to standardize numeric columns
def standardize(series):
    """Standardize
    :param series - series to be standardized
    :return the standardized series if the series is a numeric datatype
            else the original series
    """
    if is_numeric_dtype(series):
        return stats.zscore(series)
    return series


# Function to produce linear regression analysis
def perform_analysis(response, formula, dataframe):
    """Perform analysis
    :param response - the name of the response feature
    :param formula - the regression formula after the ~ sign
    :param dataframe - the pandas dataframe object
    :return the statsmodels.regression.linear_model.RegressionResults object
      [[https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.Reg
    """
    model = smf.ols(f"{response} ~ {formula}", data=dataframe)
    results = model.fit()
    print(results.summary())
    print(anova_lm(results))
    return results
```

```
# Function to get results of regression in a data frame compactly
def get_results_df(results):
    result_df = pd.DataFrame(
        {
            "coefficient": results.params,
            "se": results.bse,
            "tstatistic": results.tvalues,
            "p-value": results.pvalues,
            "r-squared": results.rsquared,
            "pearson_coefficient": np.sqrt(results.rsquared),
            "rss": results.ssr,
```

```
            "sd_residuals": np.sqrt(results.mse_resid),
        }
    )
    return result_df
```

```
def is_pos_def(x):
      return np.all(np.linalg.eigvals(x) > 0)

def check_symmetric(a, rtol=1e-05, atol=1e-08):
    return np.allclose(a, a.T, rtol=rtol, atol=atol)

def is_symmetric_pos_def(x):
  return (is_pos_def(x) & check_symmetric(x))
```