

Project Ventoux, part 2: multi feature polynomial regression

A technical overview of part 1 [can be found here](#) and a blog post about it [can be found here](#).

Part 2 focuses on taking multiple feature inputs and putting them through a polynomial regression to predict one output. This is heavy on the "finding the right polynomial degree" and less on the "optimizing the parameters," which is something I covered a lot in part 1. Different parts, different skillsets being demonstrated. There is a quick and dirty run of some poly regression based on the output of the MSEs for different polynomial degrees.

The same data table is used in part 1 and part 2. I'm going to carry forward with the same underlying data up through part 4, at which point I'll decide on a new path forward.

Importing data

libraries and methods

```
In [12]: #I know best practice is to use tian for pandas but I chose the name of a fa
import pandas as tian #named after a DC zoo panda
tian.set_option('display.max_columns', None) #invoking pandas setting I always want to see all columns

#I also diverged from numpy as lumpnump because this makes me laugh and still
import numpy as lumpnump
import matplotlib.pyplot as plt
import os
from sklearn import metrics, linear_model, utils, preprocessing, model_selection
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from scipy.stats import norm, pearsonr
import math
import copy
```

File import

```
In [13]: top_tracks_stem = os.getenv('TOP_TRACKS_STEM')

toptracks_bpmeta_matrix_csv = f'{top_tracks_stem}/toptracks_bpmeta_matrix'
toptracks_bpmeta_matrix_df_csv = f'{top_tracks_stem}/toptracks_bpmeta_matrix.csv'

toptracks_bpmeta_matrix_df = tian.read_csv(toptracks_bpmeta_matrix_df_csv)
toptracks_bpmeta_matrix = lumpnump.genfromtxt(toptracks_bpmeta_matrix_csv, c
```

```
In [14]: toptracks_bpmeta_matrix_df.head()
```

	isrc_numeric	release_id	release_date	release_year	release_month	label_id
0	2613012100395	3650447	20211001	2021	10	100342
1	471072337324	4221505	20230818	2023	8	106937
2	72126162358275	4181814	20230707	2023	7	90543
3	72126162358279	4181814	20230707	2023	7	90543
4	72126162358286	4181814	20230707	2023	7	90543

Normalization functions

I'm going to use two different normalization techniques from the Scikit Learn machine learning library.

StandardScaler: one of the most common but also allegedly not great with outliers

PowerTransformer: less common but allegedly better equipped for outliers

The goal is to just see what everything looks like under two different types of normalization

StandardScaler function

Uses StandardScaler() from Scikit Learn to center data around 0, with negative and positive values allowed. StandardScaler is considered sensitive to large outliers.

This function takes train, test, and validate data as inputs and outputs scaled data.

```
In [15]: def standard_scaling(x_train, x_validate, x_test, y_train, y_validate, y_test):
    x_features_scaler = StandardScaler()
    y_output_scaler = StandardScaler()

    # Run on training data
    x_train_scaled = x_features_scaler.fit_transform(x_train)
    y_train_scaled = y_output_scaler.fit_transform(y_train)

    # Run on validation data
    x_validate_scaled = x_features_scaler.transform(x_validate)
    y_validate_scaled = y_output_scaler.transform(y_validate)
```

```
# Run on test data
x_test_scaled = x_features_scaler.transform(x_test)
y_test_scaled = y_output_scaler.transform(y_test)

return x_train_scaled, y_train_scaled, x_validate_scaled, y_validate_sca
```

PowerTransformer function

Applies a power transformation to each feature. This is supposed to make the data a bit more Gaussian-esque to stabilize variance and minimize skewness.

"...applies a power transformation to each feature to make the data more Gaussian-like in order to stabilize variance and minimize skewness."--from Scikit Learn documentation

```
In [16]: def power_scaling(x_train, x_validate, x_test, y_train, y_validate, y_test):
    x_features_scaler = PowerTransformer()
    y_output_scaler = PowerTransformer()

    # Run on training data
    x_train_scaled = x_features_scaler.fit_transform(x_train)
    y_train_scaled = y_output_scaler.fit_transform(y_train.reshape(-1, 1))

    # Run on validation data
    x_validate_scaled = x_features_scaler.transform(x_validate)
    y_validate_scaled = y_output_scaler.transform(y_validate.reshape(-1, 1))

    # Run on test data
    x_test_scaled = x_features_scaler.transform(x_test)
    y_test_scaled = y_output_scaler.transform(y_test.reshape(-1, 1))

    return x_train_scaled, y_train_scaled, x_validate_scaled, y_validate_sca
```

Preparing data for regression

Creating dictionary and mapping references

Just working with the genres in this public part because it's easier to create a flow to show. I'll work on ways to display some of the more insane data dives I do in private.

```
In [17]: toptracks_bpmeta_matrix_df_columns = toptracks_bpmeta_matrix_df.columns

toptracks_bpmeta_matrix_df_dict = {name: idx for idx, name in enumerate(topt

subset_genres = [6, 90, 93]
subset_genres_dict = {
    6: "Techno (Peak Time/Driving)",
    90: "Melodic House & Techno",
    93: "Organic House / Downtempo"
```

```

}

# grabs the index values for the metrics, y output, and values for IDs that
core_x_features = [16, 17, 18, 19, 20, 21, 22, 23]
core_y_target = 28
core_id_values = [0, 3, 5, 7]

# these are ratios needed for different parts of the train/test/split flow
train_ratio = 0.6
validate_ratio = 0.5
test_validate_ratio = 0.4

```

Cell In[17], line 3

```

    toptracks_bpmeta_matrix_df_dict = {name: idx for idx, name in enumerate
(toptracks_bpmeta_matrix_df_columns)
^
SyntaxError: '{' was never closed

```

Splitting and normalizing

Going to very clearly breakout the data for each normalization type so I can cleanly trace errors. When you're an absolute export, a lot of this can be consolidated but I'm finding it easier to trace and fix errors when they're isolated in their own copies of datasets.

Standard Scaling

In []:

```

standard_scaler_matrix = toptracks_bpmeta_matrix
standard_scaler_ids = standard_scaler_matrix[:, core_id_values]
standard_scaler_x_features = standard_scaler_matrix[:, core_x_features]
standard_scaler_y_output = standard_scaler_matrix[:, core_y_target].reshape(-1, 1)

# training and test/validate split
x_train, x_temp, y_train, y_temp, ids_train, ids_temp = train_test_split(standard_scaler_y_output, standard_scaler_ids, test_size=0.2)

# test/validate split
x_validate, x_test, y_validate, y_test, ids_validate, ids_test = train_test_split(x_temp, y_temp, ids_temp, test_size=0.5)

x_train_standard_scaled, y_train_standard_scaled, x_validate_standard_scaled, y_validate_standard_scaled, x_test_standard_scaled, y_test_standard_scaled = train_test_split(x_train, y_train, x_validate, y_validate, x_test, y_test, test_size=0.5)

```

In []:

```

# somewhere along the way my y values became tuples and I was over it so I'm
y_train_standard_scaled = lumpnump.array(y_train_standard_scaled).reshape(-1, 1)
y_validate_standard_scaled = lumpnump.array(y_validate_standard_scaled).reshape(-1, 1)
y_test_standard_scaled = lumpnump.array(y_test_standard_scaled).reshape(-1, 1)

# taking all the scaled and not scaled data and putting it back in one matrix
standard_scaling_train = lumpnump.hstack((ids_train, x_train_standard_scaled))
standard_scaling_validate = lumpnump.hstack((ids_validate, x_validate_standard_scaled))

```

```
standard_scaling_test = lumpnump.hstack((ids_test, x_test_standard_scaled, y

# Splits size check
print("train:", standard_scaling_train.shape)
print("validate:", standard_scaling_validate.shape)
print("test:", standard_scaling_test.shape)
```

Power Scaling

```
In [ ]: power_scaler_matrix = toptracks_bpmeta_matrix
power_scaler_ids = power_scaler_matrix[:, core_id_values]
power_scaler_x_features = power_scaler_matrix[:, core_x_features]
power_scaler_y_output = power_scaler_matrix[:, core_y_target].reshape(-1, 1)

# training and test/validate split
x_train, x_temp, y_train, y_temp, ids_train, ids_temp = train_test_split(power_scaler_y_output, power_scaler_ids, test_size=test_size)

#test/validate split
x_validate, x_test, y_validate, y_test, ids_validate, ids_test = train_test_split(x_temp, y_temp, ids_temp, test_size=validate_size)

x_train_power_scaled, y_train_power_scaled, x_validate_power_scaled, y_validate_power_scaled, x_test_power_scaled, y_test_power_scaled = power_scaler_matrix[:, core_x_features], power_scaler_matrix[:, core_y_target].reshape(-1, 1), power_scaler_matrix[:, core_x_features], power_scaler_matrix[:, core_y_target].reshape(-1, 1), power_scaler_matrix[:, core_x_features], power_scaler_matrix[:, core_y_target].reshape(-1, 1)

#somewhere along the way my y values became tuples and I was over it so I'm
y_train_power_scaled = lumpnump.array(y_train_power_scaled).reshape(-1, 1)
y_validate_power_scaled = lumpnump.array(y_validate_power_scaled).reshape(-1, 1)
y_test_power_scaled = lumpnump.array(y_test_power_scaled).reshape(-1, 1)

#taking all the scaled and not scaled data and putting it back in one matrix
power_scaling_train = lumpnump.hstack((ids_train, x_train_power_scaled, y_train_power_scaled))
power_scaling_validate = lumpnump.hstack((ids_validate, x_validate_power_scaled, y_validate_power_scaled))
power_scaling_test = lumpnump.hstack((ids_test, x_test_power_scaled, y_test_power_scaled))

# Splits size check
print("train:", power_scaling_train.shape)
print("validate:", power_scaling_validate.shape)
print("test:", power_scaling_test.shape)
```

defining polynomial regression and plot functions

This takes train and validate data as inputs and an array of degrees and outputs the mean squared errors and then I smash them together to create some graphs that were inspired by code from deeplearning.ai

```
In [ ]: def poly_degree_regress(x_train, y_train, x_validate, y_validate, degrees):
    train_mean_squared_errors = []
    validate_mean_squared_errors = []

    for degree in degrees:
        polynomial = PolynomialFeatures(degree, include_bias=False)
```

```

# Train polynomial scaling and regression
poly_xtrain_mapped = polynomial.fit_transform(x_train)
scaler = StandardScaler().fit(poly_xtrain_mapped)
poly_xtrain_mapped_scaled = scaler.transform(poly_xtrain_mapped)

model = linear_model.LinearRegression()
model.fit(poly_xtrain_mapped_scaled, y_train)

# Train MSE
train_y_hat = model.predict(poly_xtrain_mapped_scaled)
train_mean_squared_error = mean_squared_error(y_train, train_y_hat)
train_mean_squared_errors.append(train_mean_squared_error)

# Validate polynomial scaling and regression
poly_xvalidate_mapped = polynomial.transform(x_validate)
poly_xvalidate_mapped_scaled = scaler.transform(poly_xvalidate_mapped)

# Validate MSE
validate_y_hat = model.predict(poly_xvalidate_mapped_scaled)
validate_mean_squared_error = mean_squared_error(y_validate, validate_y_hat)
validate_mean_squared_errors.append(validate_mean_squared_error)

return train_mean_squared_errors, validate_mean_squared_errors

```

In []: #I was trying to sort through weird plotting due to MSE values that were all
#and then side by side. I did this over time and should have found a way to

```

def plot_train_cv_mses(degrees, mses, title):
    plt.plot(degrees, mses, marker="s", c='cornflowerblue', label='train MSE')
    plt.title(title)
    plt.xlabel("Poly degrees")
    plt.ylabel("MSEs")
    plt.legend()
    plt.show()

def train_validate_msse_2plots(degrees, train_msse, validate_msse, title):
    plt.plot(degrees, train_msse, marker="s", c='cornflowerblue', label='train MSE')
    plt.plot(degrees, validate_msse, marker="P", c='fuchsia', label='validate MSE')
    plt.title(title)
    plt.xlabel("Poly degrees")
    plt.ylabel("MSEs")
    plt.legend()
    plt.show()

```

Running and graphing different normalization methods

Standard Scaling

I wrote techno and melodic as very separate things, with the intent to combine into one function to make this cleaner. I kept it this way in the end because it was easier for me to

review and I didn't want to spend a lot more time on something that might have minimal value

Techno

```
In [19]: techno_degrees = [2, 3, 4, 5, 6]
techno_genre_mask = (standard_scaling_train[:, 3] == 6.)
techno_genre_validate_mask = (standard_scaling_validate[:, 3] == 6.)
techno_genre_test_mask = (standard_scaling_test[:, 3] == 6.)

techno_xtrain = standard_scaling_train[techno_genre_mask, 4:12]
techno_ytrain = standard_scaling_train[techno_genre_mask, 12]

techno_xvalidate = standard_scaling_validate[techno_genre_validate_mask, 4:12]
techno_yvalidate = standard_scaling_validate[techno_genre_validate_mask, 12]

techno_xtest = standard_scaling_test[techno_genre_test_mask, 4:12]
techno_ytest = standard_scaling_test[techno_genre_test_mask, 12]

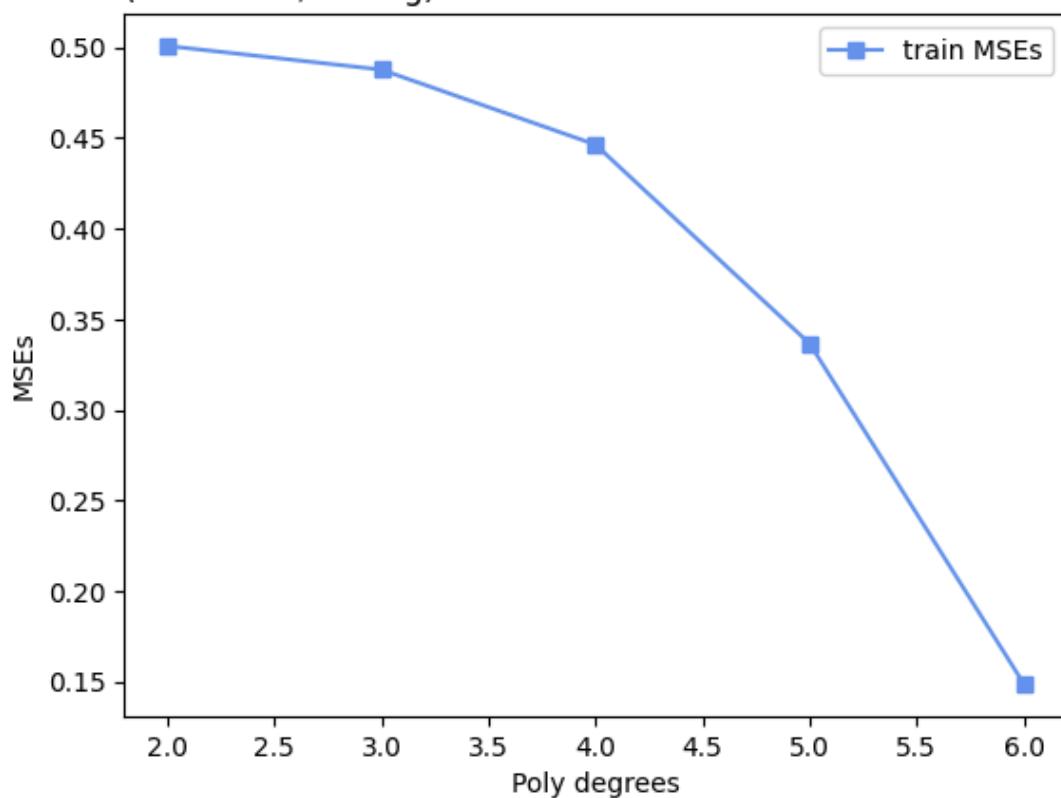
techno_train_mses, techno_validate_mses = poly_degree_regress(techno_xtrain,
# Print results
print("Techno training MSEs:", techno_train_mses)
print("Techno validation MSEs:", techno_validate_mses)

Techno training MSEs: [0.5009038903017673, 0.4878086227651276, 0.44640179981
73258, 0.33647495565648394, 0.14860510542836886]
Techno validation MSEs: [0.4971352920978839, 0.5376727052760927, 4.755932691
402215, 23880.81837281477, 2739904030.175498]
```

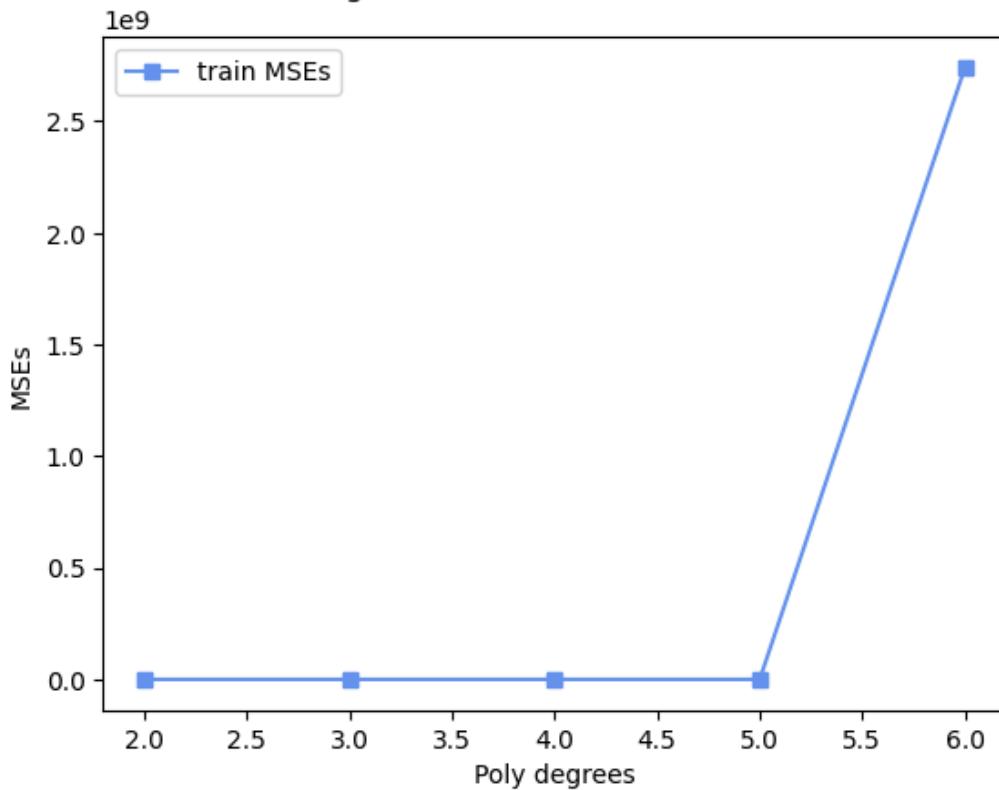
```
In [20]: techno = subset_genres_dict[6]
techno_title = f"{techno} train MSEs for standard scaler normalized data"
validate_title = f"{techno} validate MSEs for standard scaler normalized data"
techno_overall_title = f"{techno} train and validate mean squared errors for"
plot_train_cv_mses(techno_degrees, techno_train_mses, techno_title)
plot_train_cv_mses(techno_degrees, techno_validate_mses, validate_title)

train_validate_mses_2plots(techno_degrees, techno_train_mses, techno_validate_mses)
```

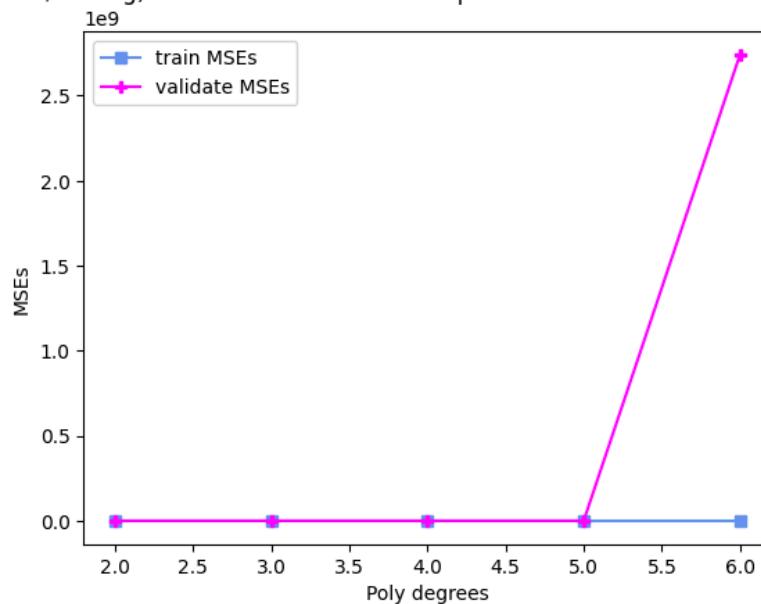
Techno (Peak Time/Driving) train MSEs for standard scaler normalized data



Techno (Peak Time/Driving) validate MSEs for standard scaler normalized data



Techno (Peak Time/Driving) train and validate mean squared errors for standard scaler normalized data



```
In [21]: techno_poly4 = PolynomialFeatures(degree=4)
techno_x_poly4_train = techno_poly4.fit_transform(techno_xtrain)
techno_x_poly4_validate = techno_poly4.fit_transform(techno_xvalidate)
techno_x_poly4_test = techno_poly4.fit_transform(techno_xtest)

techno_model4 = linear_model.LinearRegression()
techno_model4.fit(techno_x_poly4_train, techno_ytrain)
y_poly_pred4_train = techno_model4.predict(techno_x_poly4_train)

techno_model4.fit(techno_x_poly4_validate, techno_yvalidate)
y_poly_pred4_validate = techno_model4.predict(techno_x_poly4_validate)

techno_model4.fit(techno_x_poly4_test, techno_ytest)
y_poly_pred4_test = techno_model4.predict(techno_x_poly4_test)

techno_sorted_indices_train = lumpyump.argsort(techno_xtrain[:, 3])
techno_sorted_indices_validate = lumpyump.argsort(techno_xvalidate[:, 3])
techno_sorted_indices_test = lumpyump.argsort(techno_xtest[:, 3])

techno_sorted_xtrain = techno_xtrain[techno_sorted_indices_train, 3]
techno_sorted_xtrain_validate = techno_xvalidate[techno_sorted_indices_validate, 3]
techno_sorted_xtest = techno_xtest[techno_sorted_indices_test, 3]

techno_sorted_ytrain = techno_ytrain[techno_sorted_indices_train]
techno_sorted_ytrain_validate = techno_yvalidate[techno_sorted_indices_validate]
techno_sorted_ytest = techno_ytest[techno_sorted_indices_test]

techno_sorted_y_poly_pred4_train = y_poly_pred4_train[techno_sorted_indices_train]
techno_sorted_y_poly_pred4_validate = y_poly_pred4_validate[techno_sorted_indices_validate]
techno_sorted_y_poly_pred4_test = y_poly_pred4_test[techno_sorted_indices_test]

# Polynomial regression for degree 5
techno_poly5 = PolynomialFeatures(degree=5)
techno_x_poly5_train = techno_poly5.fit_transform(techno_xtrain)
techno_x_poly5_validate = techno_poly5.fit_transform(techno_xvalidate)
techno_x_poly5_test = techno_poly5.fit_transform(techno_xtest)
```

```
# Fitting the polynomial regression model for degree 5
techno_model5 = linear_model.LinearRegression()
techno_model5.fit(techno_x_poly5_train, techno_ytrain)
y_poly_pred5_train = techno_model5.predict(techno_x_poly5_train)

techno_model5.fit(techno_x_poly5_validate, techno_yvalidate)
y_poly_pred5_validate = techno_model5.predict(techno_x_poly5_validate)

techno_model5.fit(techno_x_poly5_test, techno_ytest)
y_poly_pred5_test = techno_model5.predict(techno_x_poly5_test)

techno_sorted_y_poly_pred5_train = y_poly_pred5_train[techno_sorted_indices]
techno_sorted_y_poly_pred5_validate = y_poly_pred5_validate[techno_sorted_indices]
techno_sorted_y_poly_pred5_test = y_poly_pred5_test[techno_sorted_indices]

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(techno_sorted_xtrain, techno_sorted_ytrain, color='blue', label='points')
plt.plot(techno_sorted_xtrain, techno_sorted_y_poly_pred4_train, color='red', label='fit')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_techno_train = r2_score(techno_ytrain, y_poly_pred4_train)
mse_4_techno_train = mean_squared_error(techno_ytrain, y_poly_pred4_train)
pearson_corr_4_techno_train, _ = pearsonr(techno_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f"R^2: {r2_4_techno_train:.4f}\nMSE: {mse_4_techno_train:.4f}\nPearson Corr: {pearson_corr_4_techno_train:.4f}")

plt.subplot(3, 2, 3)
plt.scatter(techno_sorted_xtrain_validate, techno_sorted_ytrain_validate, color='blue', label='points')
plt.plot(techno_sorted_xtrain_validate, techno_sorted_y_poly_pred4_validate, color='red', label='fit')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_techno_validate = r2_score(techno_yvalidate, y_poly_pred4_validate)
mse_4_techno_validate = mean_squared_error(techno_yvalidate, y_poly_pred4_validate)
pearson_corr_4_techno_validate, _ = pearsonr(techno_yvalidate, y_poly_pred4_validate)
plt.figtext(0.1, 0.62, f"R^2: {r2_4_techno_validate:.4f}\nMSE: {mse_4_techno_validate:.4f}\nPearson Corr: {pearson_corr_4_techno_validate:.4f}")

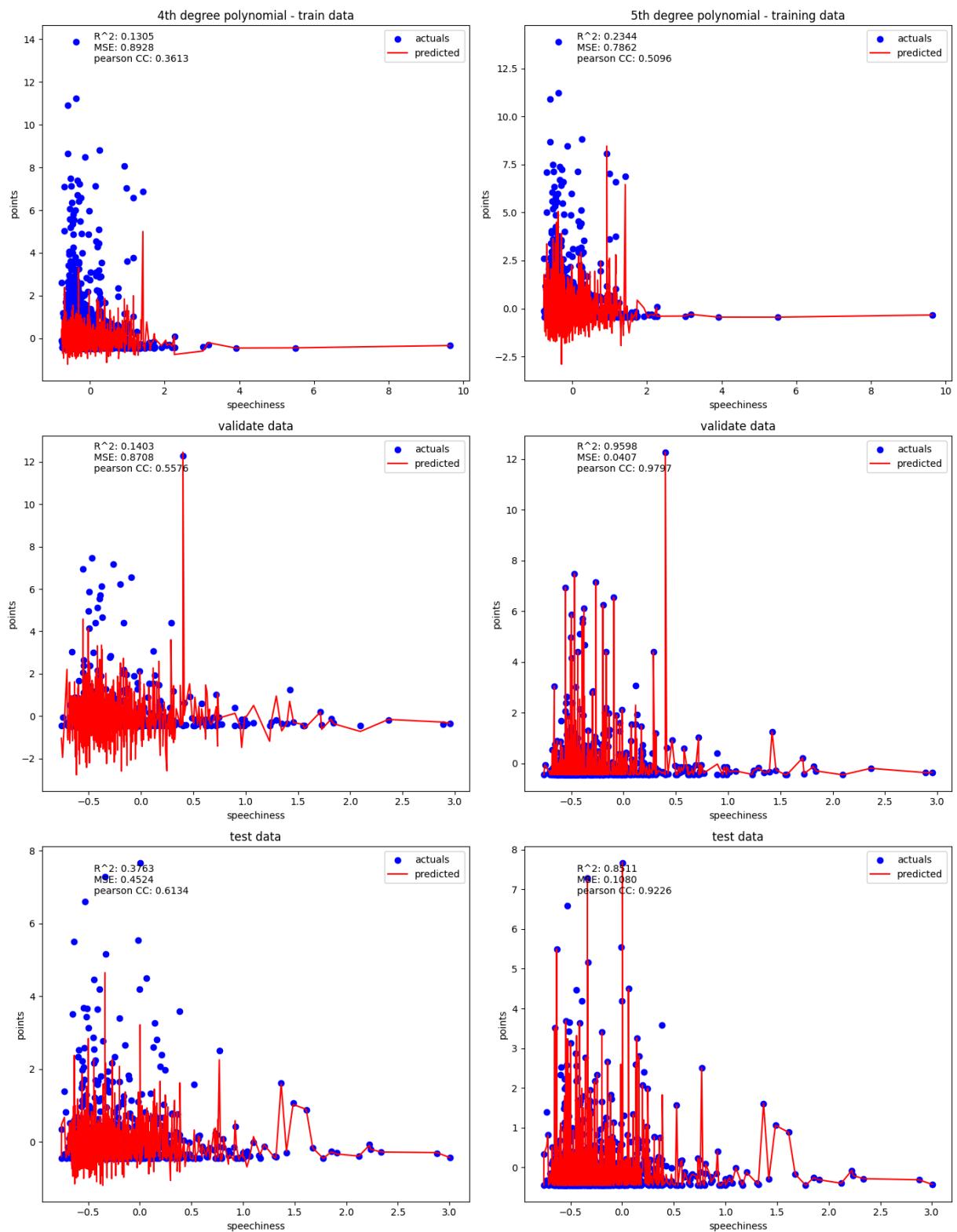
plt.subplot(3, 2, 5)
plt.scatter(techno_sorted_xtest, techno_sorted_ytest, color='blue', label='points')
plt.plot(techno_sorted_xtest, techno_sorted_y_poly_pred4_test, color='red', label='fit')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_techno_test = r2_score(techno_ytest, y_poly_pred4_test)
mse_4_techno_test = mean_squared_error(techno_ytest, y_poly_pred4_test)
pearson_corr_4_techno_test, _ = pearsonr(techno_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_techno_test:.4f}\nMSE: {mse_4_techno_test:.4f}\nPearson Corr: {pearson_corr_4_techno_test:.4f}")
```

```
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(techno_sorted_xtrain, techno_sorted_ytrain, color='blue', label='points')
plt.plot(techno_sorted_xtrain, techno_sorted_y_poly_pred5_train, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_techno_train = r2_score(techno_ytrain, y_poly_pred5_train)
mse_5_techno_train = mean_squared_error(techno_ytrain, y_poly_pred5_train)
pearson_corr_5_techno_train, _ = pearsonr(techno_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_techno_train:.4f}\nMSE: {mse_5_techno_train:.4f}")

plt.subplot(3, 2, 4)
plt.scatter(techno_sorted_xtrain_validate, techno_sorted_ytrain_validate, color='blue', label='points')
plt.plot(techno_sorted_xtrain_validate, techno_sorted_y_poly_pred5_validate, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_techno_validate = r2_score(techno_yvalidate, y_poly_pred5_validate)
mse_5_techno_validate = mean_squared_error(techno_yvalidate, y_poly_pred5_validate)
pearson_corr_5_techno_validate, _ = pearsonr(techno_yvalidate, y_poly_pred5_validate)
plt.figtext(0.6, 0.62, f"R^2: {r2_5_techno_validate:.4f}\nMSE: {mse_5_techno_validate:.4f}")

plt.subplot(3, 2, 6)
plt.scatter(techno_sorted_xtest, techno_sorted_ytest, color='blue', label='points')
plt.plot(techno_sorted_xtest, techno_sorted_y_poly_pred5_test, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_techno_test = r2_score(techno_ytest, y_poly_pred5_test)
mse_5_techno_test = mean_squared_error(techno_ytest, y_poly_pred5_test)
pearson_corr_5_techno_test, _ = pearsonr(techno_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_techno_test:.4f}\nMSE: {mse_5_techno_test:.4f}")

plt.tight_layout()
plt.show()
```



Melodic house and techno

```
In [22]: melodic_degrees = [2, 3, 4, 5, 6]
melodic_genre_mask = (standard_scaling_train[:, 3] == 90.)
melodic_genre_validate_mask = (standard_scaling_validate[:, 3] == 90.)
melodic_genre_test_mask = (standard_scaling_test[:, 3] == 90.)

melodic_xtrain = standard_scaling_train[melodic_genre_mask, 4:12]
```

```

melodic_ytrain = standard_scaling_train[melodic_genre_mask, 12]
melodic_xvalidate = standard_scaling_validate[melodic_genre_validate_mask, 4:12]
melodic_yvalidate = standard_scaling_validate[melodic_genre_validate_mask, 1]

melodic_xtest = standard_scaling_test[melodic_genre_test_mask, 4:12]
melodic_ytest = standard_scaling_test[melodic_genre_test_mask, 12]

melodic_train_mses, melodic_validate_mses = poly_degree_regress(melodic_xtrain,
    melodic_ytrain, melodic_xvalidate, melodic_yvalidate, 1, 6)

print("melodic training MSEs:", melodic_train_mses)
print("melodic validation MSEs:", melodic_validate_mses)

```

melodic training MSEs: [0.35230948718228916, 0.33881540804014026, 0.303247947701971, 0.20500974623967386, 0.12096257941989522]
melodic validation MSEs: [0.34030771126277876, 0.3654972922996947, 55.72826580107606, 6584225.443401804, 61299082199.576385]

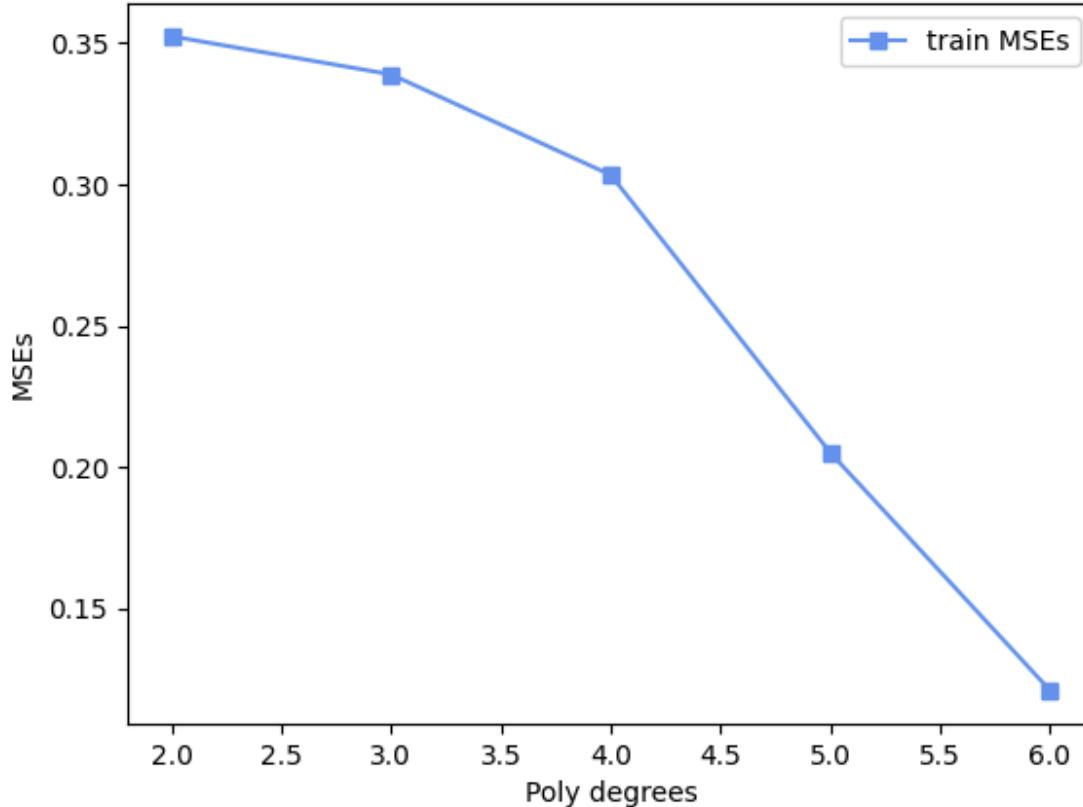
In [23]:

```

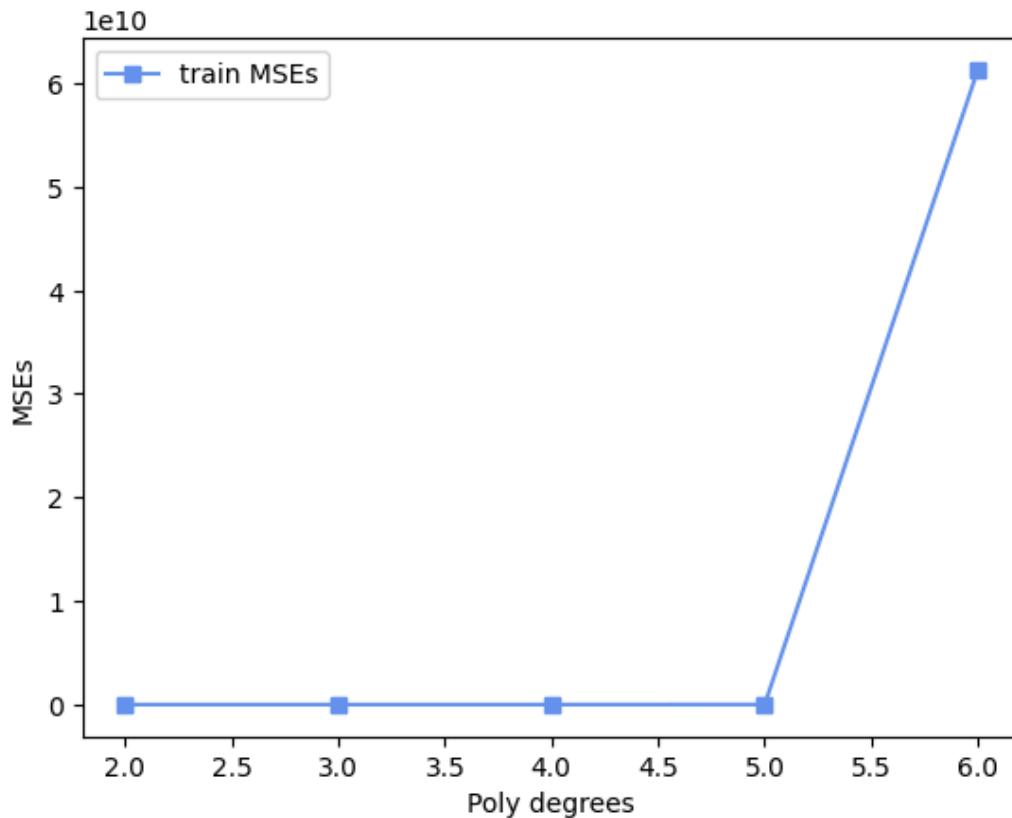
melodic = subset_genres_dict[90]
melodic_title = f"{melodic} train MSEs for standard scaler normalized data"
melodic_validate_title = f"{melodic} validate MSEs for standard scaler normalized data"
melodic_overall_title = f" {melodic} train and validate MSEs for standard scaler normalized data"
plot_train_cv_mses(melodic_degrees, melodic_train_mses, melodic_title)
plot_train_cv_mses(melodic_degrees, melodic_validate_mses, melodic_validate_title)
train_validate_mses_2plots(melodic_degrees, melodic_train_mses, melodic_validate_mses)

```

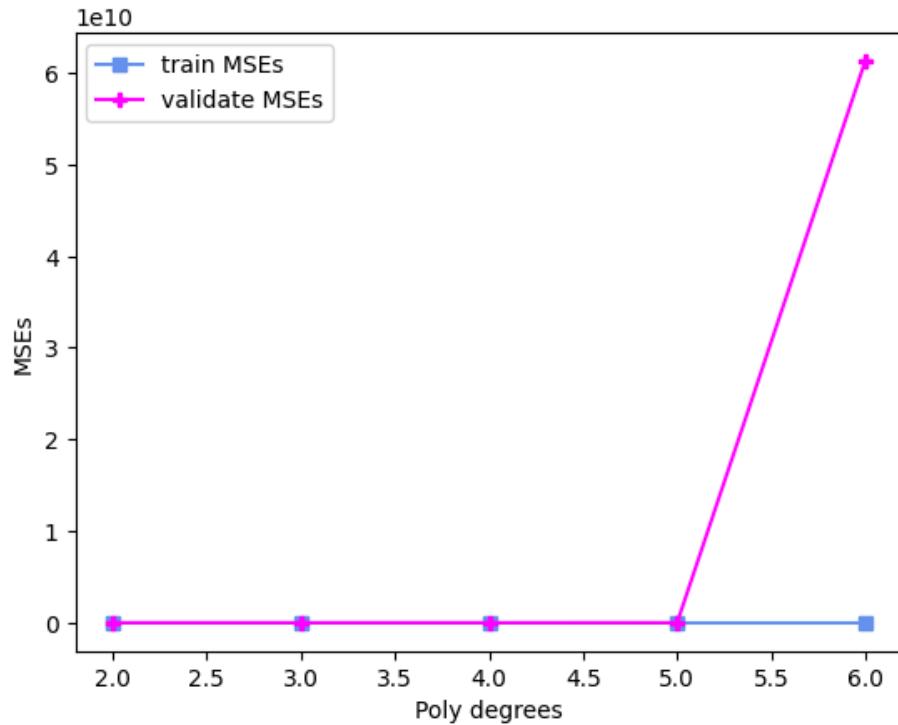
Melodic House & Techno train MSEs for standard scaler normalized data



Melodic House & Techno validate MSEs for standard scaler normalized data



Melodic House & Techno train and validate MSEs for standard scaler normalized data



```
In [24]: melodic_poly4 = PolynomialFeatures(degree=4)
melodic_x_poly4_train = melodic_poly4.fit_transform(melodic_xtrain)
melodic_x_poly4_validate = melodic_poly4.fit_transform(melodic_xvalidate)
melodic_x_poly4_test = melodic_poly4.fit_transform(melodic_xtest)
```

```

melodic_model4 = linear_model.LinearRegression()
melodic_model4.fit(melodic_x_poly4_train, melodic_ytrain)
y_poly_pred4_train = melodic_model4.predict(melodic_x_poly4_train)

melodic_model4.fit(melodic_x_poly4_validate, melodic_yvalidate)
y_poly_pred4_validate = melodic_model4.predict(melodic_x_poly4_validate)

melodic_model4.fit(melodic_x_poly4_test, melodic_ytest)
y_poly_pred4_test = melodic_model4.predict(melodic_x_poly4_test)

melodic_sorted_indices_train = lumpnump.argsort(melodic_xtrain[:, 3])
melodic_sorted_indices_validate = lumpnump.argsort(melodic_xvalidate[:, 3])
melodic_sorted_indices_test = lumpnump.argsort(melodic_xtest[:, 3])

melodic_sorted_xtrain = melodic_xtrain[melodic_sorted_indices_train, 3]
melodic_sorted_xtrain_validate = melodic_xvalidate[melodic_sorted_indices_va
melodic_sorted_xtest = melodic_xtest[melodic_sorted_indices_test, 3]

melodic_sorted_ytrain = melodic_ytrain[melodic_sorted_indices_train]
melodic_sorted_ytrain_validate = melodic_yvalidate[melodic_sorted_indices_va
melodic_sorted_ytest = melodic_ytest[melodic_sorted_indices_test]

melodic_sorted_y_poly_pred4_train = y_poly_pred4_train[melodic_sorted_indices_
melodic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[melodic_sorted_
melodic_sorted_y_poly_pred4_test = y_poly_pred4_test[melodic_sorted_indices_]

# Polynomial regression for degree 5
melodic_poly5 = PolynomialFeatures(degree=5)
melodic_x_poly5_train = melodic_poly5.fit_transform(melodic_xtrain)
melodic_x_poly5_validate = melodic_poly5.fit_transform(melodic_xvalidate)
melodic_x_poly5_test = melodic_poly5.fit_transform(melodic_xtest)

# Fitting the polynomial regression model for degree 5
melodic_model5 = linear_model.LinearRegression()
melodic_model5.fit(melodic_x_poly5_train, melodic_ytrain)
y_poly_pred5_train = melodic_model5.predict(melodic_x_poly5_train)

melodic_model5.fit(melodic_x_poly5_validate, melodic_yvalidate)
y_poly_pred5_validate = melodic_model5.predict(melodic_x_poly5_validate)

melodic_model5.fit(melodic_x_poly5_test, melodic_ytest)
y_poly_pred5_test = melodic_model5.predict(melodic_x_poly5_test)

melodic_sorted_y_poly_pred5_train = y_poly_pred5_train[melodic_sorted_indices_
melodic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[melodic_sorted_
melodic_sorted_y_poly_pred5_test = y_poly_pred5_test[melodic_sorted_indices_]

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(melodic_sorted_xtrain, melodic_sorted_ytrain, color='blue', labe
plt.plot(melodic_sorted_xtrain, melodic_sorted_y_poly_pred4_train, color='red'
plt.xlabel('speechiness')
plt.ylabel('points')

```

```

plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_melodic_train = r2_score(melodic_ytrain, y_poly_pred4_train)
mse_4_melodic_train = mean_squared_error(melodic_ytrain, y_poly_pred4_train)
pearson_corr_4_melodic_train, _ = pearsonr(melodic_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f"R^2: {r2_4_melodic_train:.4f}\nMSE: {mse_4_melodic_}

plt.subplot(3, 2, 3)
plt.scatter(melodic_sorted_xtrain_validate, melodic_sorted_ytrain_validate,
plt.plot(melodic_sorted_xtrain_validate, melodic_sorted_y_poly_pred4_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_melodic_validate = r2_score(melodic_yvalidate, y_poly_pred4_validate)
mse_4_melodic_validate = mean_squared_error(melodic_yvalidate, y_poly_pred4_validate)
pearson_corr_4_melodic_validate, _ = pearsonr(melodic_yvalidate, y_poly_pred4_validate)
plt.figtext(0.1, 0.62, f"R^2: {r2_4_melodic_validate:.4f}\nMSE: {mse_4_melodic_}

plt.subplot(3, 2, 5)
plt.scatter(melodic_sorted_xtest, melodic_sorted_ytest, color='blue', label='points')
plt.plot(melodic_sorted_xtest, melodic_sorted_y_poly_pred4_test, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_melodic_test = r2_score(melodic_ytest, y_poly_pred4_test)
mse_4_melodic_test = mean_squared_error(melodic_ytest, y_poly_pred4_test)
pearson_corr_4_melodic_test, _ = pearsonr(melodic_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_melodic_test:.4f}\nMSE: {mse_4_melodic_}

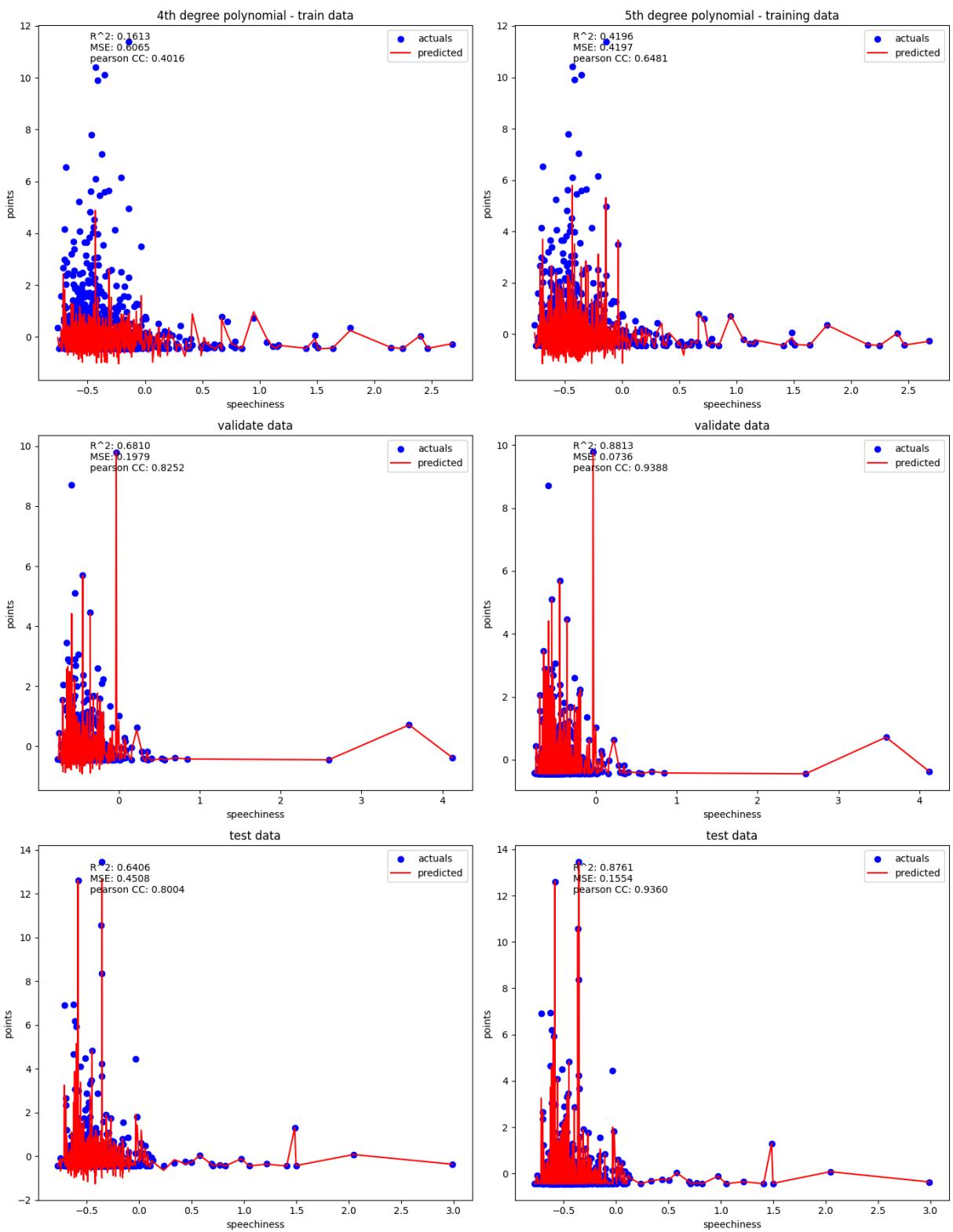
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(melodic_sorted_xtrain, melodic_sorted_ytrain, color='blue', label='points')
plt.plot(melodic_sorted_xtrain, melodic_sorted_y_poly_pred5_train, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_melodic_train = r2_score(melodic_ytrain, y_poly_pred5_train)
mse_5_melodic_train = mean_squared_error(melodic_ytrain, y_poly_pred5_train)
pearson_corr_5_melodic_train, _ = pearsonr(melodic_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_melodic_train:.4f}\nMSE: {mse_5_melodic_}

plt.subplot(3, 2, 4)
plt.scatter(melodic_sorted_xtrain_validate, melodic_sorted_ytrain_validate,
plt.plot(melodic_sorted_xtrain_validate, melodic_sorted_y_poly_pred5_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_melodic_validate = r2_score(melodic_yvalidate, y_poly_pred5_validate)
mse_5_melodic_validate = mean_squared_error(melodic_yvalidate, y_poly_pred5_validate)
pearson_corr_5_melodic_validate, _ = pearsonr(melodic_yvalidate, y_poly_pred5_validate)
plt.figtext(0.6, 0.62, f"R^2: {r2_5_melodic_validate:.4f}\nMSE: {mse_5_melodic_}

```

```
plt.subplot(3, 2, 6)
plt.scatter(melodic_sorted_xtest, melodic_sorted_ytest, color='blue', label=
plt.plot(melodic_sorted_xtest, melodic_sorted_y_poly_pred5_test, color='red'
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_melodic_test = r2_score(melodic_ytest, y_poly_pred5_test)
mse_5_melodic_test = mean_squared_error(melodic_ytest, y_poly_pred5_test)
pearson_corr_5_melodic_test, _ = pearsonr(melodic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f'R^2: {r2_5_melodic_test:.4f}\nMSE: {mse_5_melodic_t

plt.tight_layout()
plt.show()
```



Organic house/down tempo

```
In [25]: organic_degrees = [2, 3, 4, 5, 6]
organic_genre_mask = (standard_scaling_train[:, 3] == 93.)
organic_genre_validate_mask = (standard_scaling_validate[:, 3] == 93.)

organic_xtrain = standard_scaling_train[organic_genre_mask, 4:12]
organic_ytrain = standard_scaling_train[organic_genre_mask, 12]
```

```

organic_xvalidate = standard_scaling_validate[organic_genre_validate_mask, 4:12]
organic_yvalidate = standard_scaling_validate[organic_genre_validate_mask, 12]

organic_xtest = standard_scaling_test[organic_genre_validate_mask, 4:12]
organic_ytest = standard_scaling_test[organic_genre_validate_mask, 12]

organic_train_mses, organic_validate_mses = poly_degree_regress(organic_xtrain,
    organic_ytrain, 2, 6)

# Print results
print("organic training MSEs:", organic_train_mses)
print("organic validation MSEs:", organic_validate_mses)

organic training MSEs: [0.3539071110949449, 0.3456354596496505, 0.3158095241
3206176, 0.20404768046254645, 0.10603118671301336]
organic validation MSEs: [0.21188266813074344, 0.2644059867134607, 5.4818102
037996015, 286224.22296324244, 208151071.39585665]

```

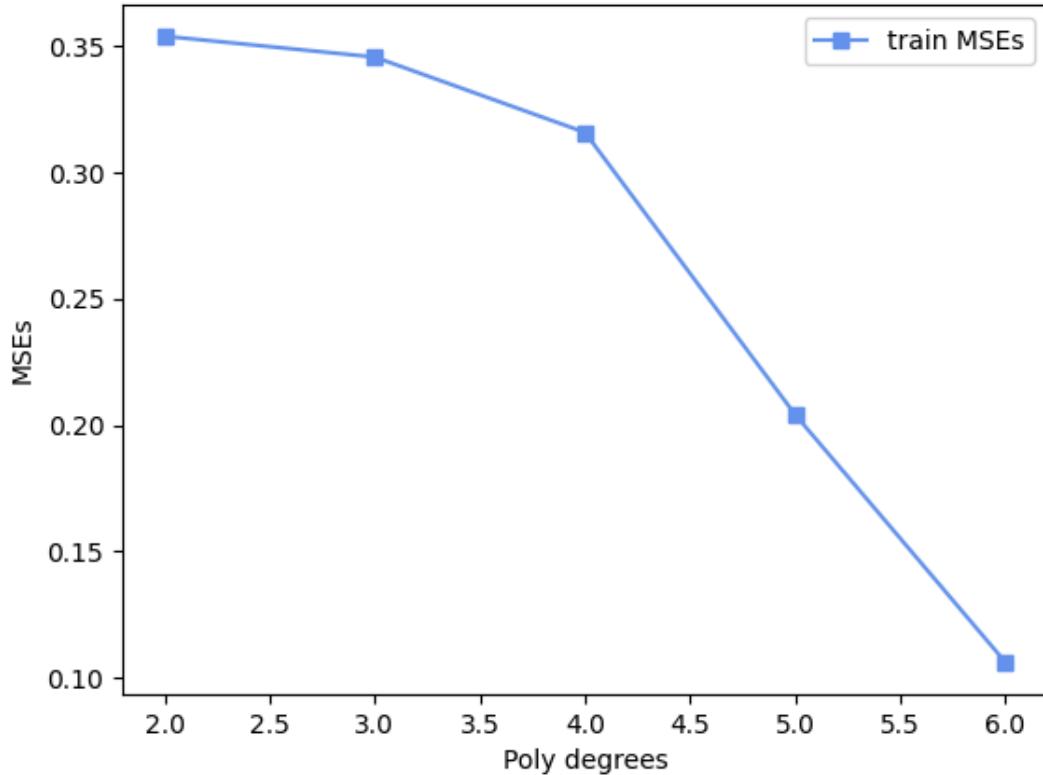
In [26]:

```

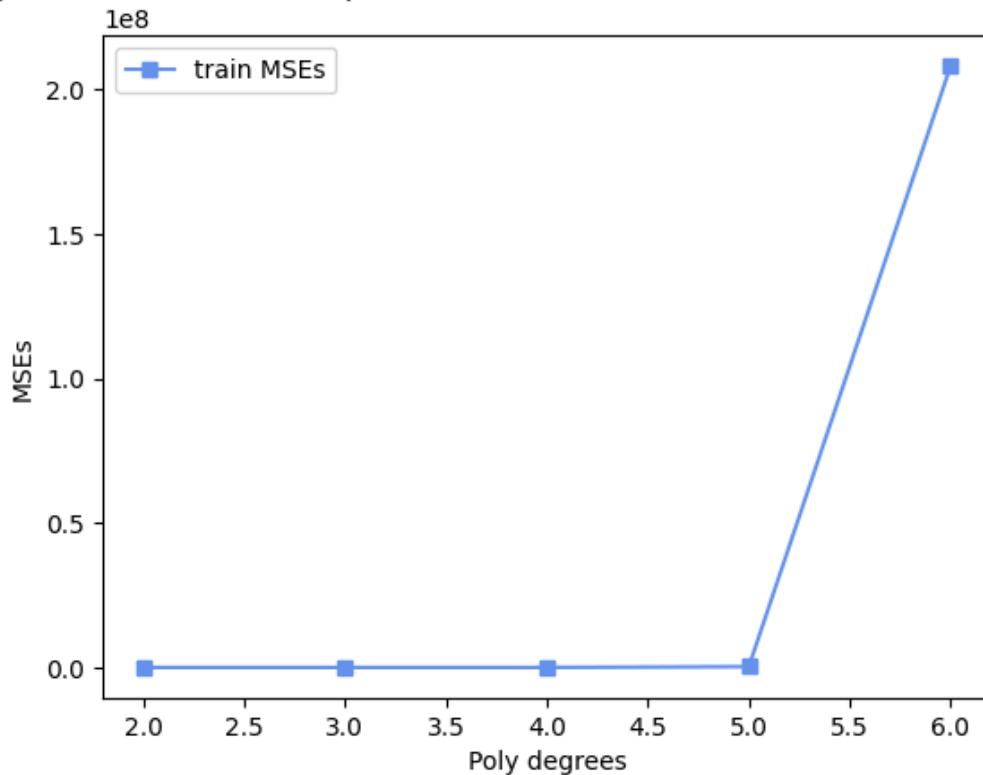
organic = subset_genres_dict[93]
organic_title = f"{{organic}} train MSEs for standard scaler normalized data"
organic_validate_title = f"{{organic}} validate MSEs for standard scaler normalized data"
organic_overall_title = f" {{organic}} train and validate MSEs for standard scaler normalized data"
plot_train_cv_mses(organic_degrees, organic_train_mses, organic_title)
plot_train_cv_mses(organic_degrees, organic_validate_mses, organic_validate_title)
train_validate_mses_2plots(organic_degrees, organic_train_mses, organic_validate_mses)

```

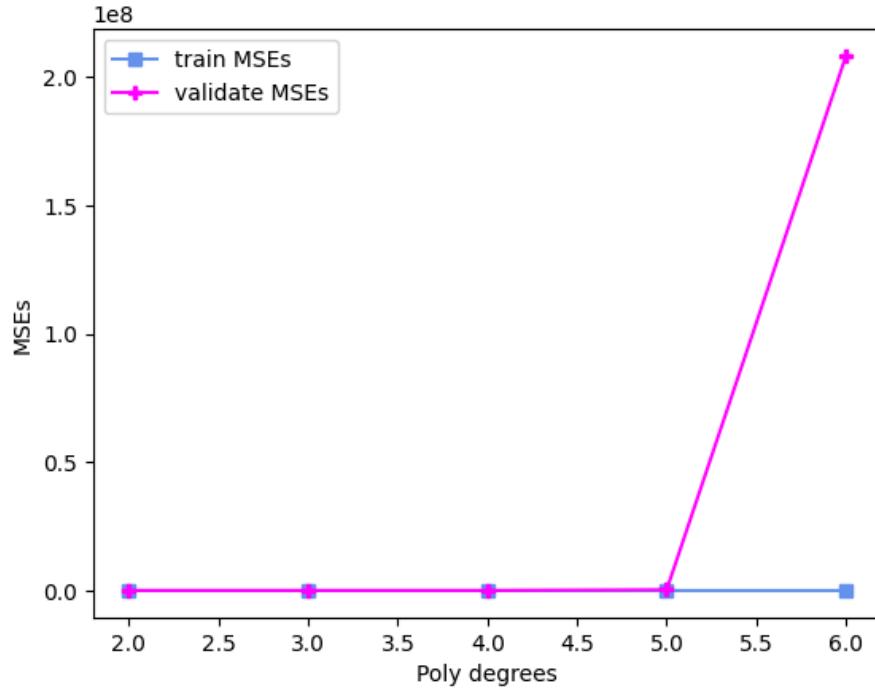
Organic House / Downtempo train MSEs for standard scaler normalized data



Organic House / Downtempo validate MSEs for standard scaler normalized data



Organic House / Downtempo train and validate MSEs for standard scaler normalized data



```
In [27]: organic_poly4 = PolynomialFeatures(degree=4)
organic_x_poly4_train = organic_poly4.fit_transform(organic_xtrain)
organic_x_poly4_validate = organic_poly4.fit_transform(organic_xvalidate)
organic_x_poly4_test = organic_poly4.fit_transform(organic_xtest)

organic_model4 = linear_model.LinearRegression()
organic_model4.fit(organic_x_poly4_train, organic_ytrain)
y_poly_pred4_train = organic_model4.predict(organic_x_poly4_train)
```

```
organic_model4.fit(organic_x_poly4_validate, organic_yvalidate)
y_poly_pred4_validate = organic_model4.predict(organic_x_poly4_validate)

organic_model4.fit(organic_x_poly4_test, organic_ytest)
y_poly_pred4_test = organic_model4.predict(organic_x_poly4_test)

organic_sorted_indices_train = lumpnump.argsort(organic_xtrain[:, 3])
organic_sorted_indices_validate = lumpnump.argsort(organic_xvalidate[:, 3])
organic_sorted_indices_test = lumpnump.argsort(organic_xtest[:, 3])

organic_sorted_xtrain = organic_xtrain[organic_sorted_indices_train, 3]
organic_sorted_xtrain_validate = organic_xvalidate[organic_sorted_indices_validate, 3]
organic_sorted_xtest = organic_xtest[organic_sorted_indices_test, 3]

organic_sorted_ytrain = organic_ytrain[organic_sorted_indices_train]
organic_sorted_ytrain_validate = organic_yvalidate[organic_sorted_indices_validate]
organic_sorted_ytest = organic_ytest[organic_sorted_indices_test]

organic_sorted_y_poly_pred4_train = y_poly_pred4_train[organic_sorted_indices_train, 3]
organic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[organic_sorted_indices_validate, 3]
organic_sorted_y_poly_pred4_test = y_poly_pred4_test[organic_sorted_indices_test, 3]

# Polynomial regression for degree 5
organic_poly5 = PolynomialFeatures(degree=5)
organic_x_poly5_train = organic_poly5.fit_transform(organic_xtrain)
organic_x_poly5_validate = organic_poly5.fit_transform(organic_xvalidate)
organic_x_poly5_test = organic_poly5.fit_transform(organic_xtest)

# Fitting the polynomial regression model for degree 5
organic_model5 = linear_model.LinearRegression()
organic_model5.fit(organic_x_poly5_train, organic_ytrain)
y_poly_pred5_train = organic_model5.predict(organic_x_poly5_train)

organic_model5.fit(organic_x_poly5_validate, organic_yvalidate)
y_poly_pred5_validate = organic_model5.predict(organic_x_poly5_validate)

organic_model5.fit(organic_x_poly5_test, organic_ytest)
y_poly_pred5_test = organic_model5.predict(organic_x_poly5_test)

organic_sorted_y_poly_pred5_train = y_poly_pred5_train[organic_sorted_indices_train, 3]
organic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[organic_sorted_indices_validate, 3]
organic_sorted_y_poly_pred5_test = y_poly_pred5_test[organic_sorted_indices_test, 3]

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(organic_sorted_xtrain, organic_sorted_ytrain, color='blue', label='points')
plt.plot(organic_sorted_xtrain, organic_sorted_y_poly_pred4_train, color='red', label='4th degree polynomial')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_organic_train = r2_score(organic_ytrain, y_poly_pred4_train)
```

```

mse_4_organic_train = mean_squared_error(organic_ytrain, y_poly_pred4_train)
pearson_corr_4_organic_train, _ = pearsonr(organic_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f"R^2: {r2_4_organic_train:.4f}\nMSE: {mse_4_organic_}

plt.subplot(3, 2, 3)
plt.scatter(organic_sorted_xtrain_validate, organic_sorted_ytrain_validate,
plt.plot(organic_sorted_xtrain_validate, organic_sorted_y_poly_pred4_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_organic_validate = r2_score(organic_yvalidate, y_poly_pred4_validate)
mse_4_organic_validate = mean_squared_error(organic_yvalidate, y_poly_pred4_
pearson_corr_4_organic_validate, _ = pearsonr(organic_yvalidate, y_poly_pred4_
plt.figtext(0.1, 0.62, f"R^2: {r2_4_organic_validate:.4f}\nMSE: {mse_4_organic_

plt.subplot(3, 2, 5)
plt.scatter(organic_sorted_xtest, organic_sorted_ytest, color='blue', label=
plt.plot(organic_sorted_xtest, organic_sorted_y_poly_pred4_test, color='red'
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_organic_test = r2_score(organic_ytest, y_poly_pred4_test)
mse_4_organic_test = mean_squared_error(organic_ytest, y_poly_pred4_test)
pearson_corr_4_organic_test, _ = pearsonr(organic_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_organic_test:.4f}\nMSE: {mse_4_organic_t

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(organic_sorted_xtrain, organic_sorted_ytrain, color='blue', label=
plt.plot(organic_sorted_xtrain, organic_sorted_y_poly_pred5_train, color='red'
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_organic_train = r2_score(organic_ytrain, y_poly_pred5_train)
mse_5_organic_train = mean_squared_error(organic_ytrain, y_poly_pred5_train)
pearson_corr_5_organic_train, _ = pearsonr(organic_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_organic_train:.4f}\nMSE: {mse_5_organic_

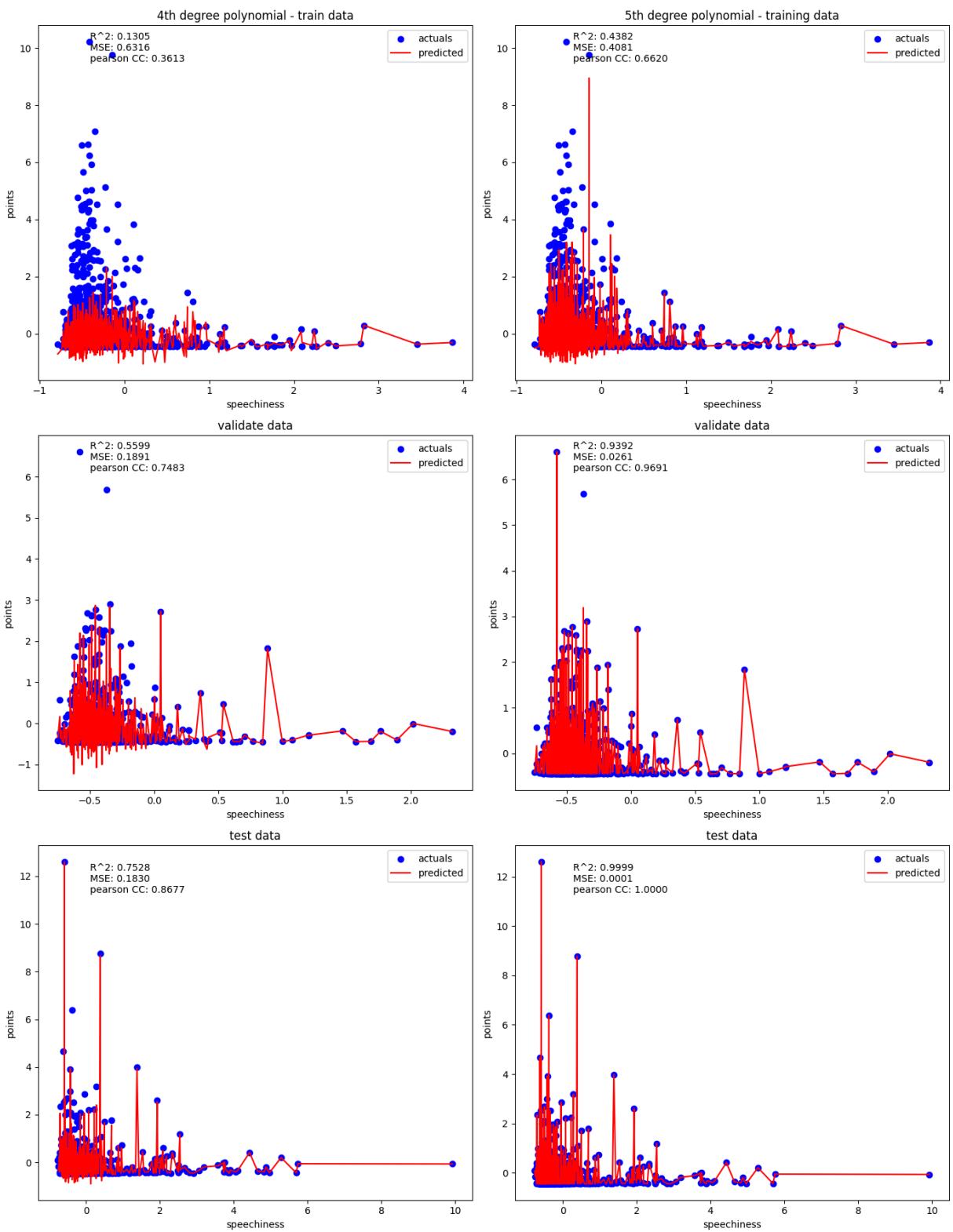
plt.subplot(3, 2, 4)
plt.scatter(organic_sorted_xtrain_validate, organic_sorted_ytrain_validate,
plt.plot(organic_sorted_xtrain_validate, organic_sorted_y_poly_pred5_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_organic_validate = r2_score(organic_yvalidate, y_poly_pred5_validate)
mse_5_organic_validate = mean_squared_error(organic_yvalidate, y_poly_pred5_
pearson_corr_5_organic_validate, _ = pearsonr(organic_yvalidate, y_poly_pred5_
plt.figtext(0.6, 0.62, f"R^2: {r2_5_organic_validate:.4f}\nMSE: {mse_5_organic_

plt.subplot(3, 2, 6)
plt.scatter(organic_sorted_xtest, organic_sorted_ytest, color='blue', label=
plt.plot(organic_sorted_xtest, organic_sorted_y_poly_pred5_test, color='red'

```

```
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_organic_test = r2_score(organic_ytest, y_poly_pred5_test)
mse_5_organic_test = mean_squared_error(organic_ytest, y_poly_pred5_test)
pearson_corr_5_organic_test, _ = pearsonr(organic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_organic_test:.4f}\nMSE: {mse_5_organic_t

plt.tight_layout()
plt.show()
```



Power Scaling

Techno

```
In [28]: power_techno_degrees = [2, 3, 4, 5, 6]
power_techno_genre_mask = (power_scaling_train[:, 3] == 6.)
power_techno_genre_validate_mask = (power_scaling_validate[:, 3] == 6.)
```

```

power_techno_genre_test_mask = (power_scaling_test[:, 3] == 6.)

power_techno_xtrain = power_scaling_train[power_techno_genre_mask, 4:12]
power_techno_ytrain = power_scaling_train[power_techno_genre_mask, 12]

power_techno_xvalidate = power_scaling_validate[power_techno_genre_validate_
power_techno_yvalidate = power_scaling_validate[power_techno_genre_validate_

power_techno_xtest = power_scaling_test[power_techno_genre_test_mask, 4:12]
power_techno_ytest = power_scaling_test[power_techno_genre_test_mask, 12]

power_techno_train_mses, power_techno_validate_mses = poly_degree_regress(po

# Print results
print("Techno training MSEs:", power_techno_train_mses)
print("Techno validation MSEs:", power_techno_validate_mses)

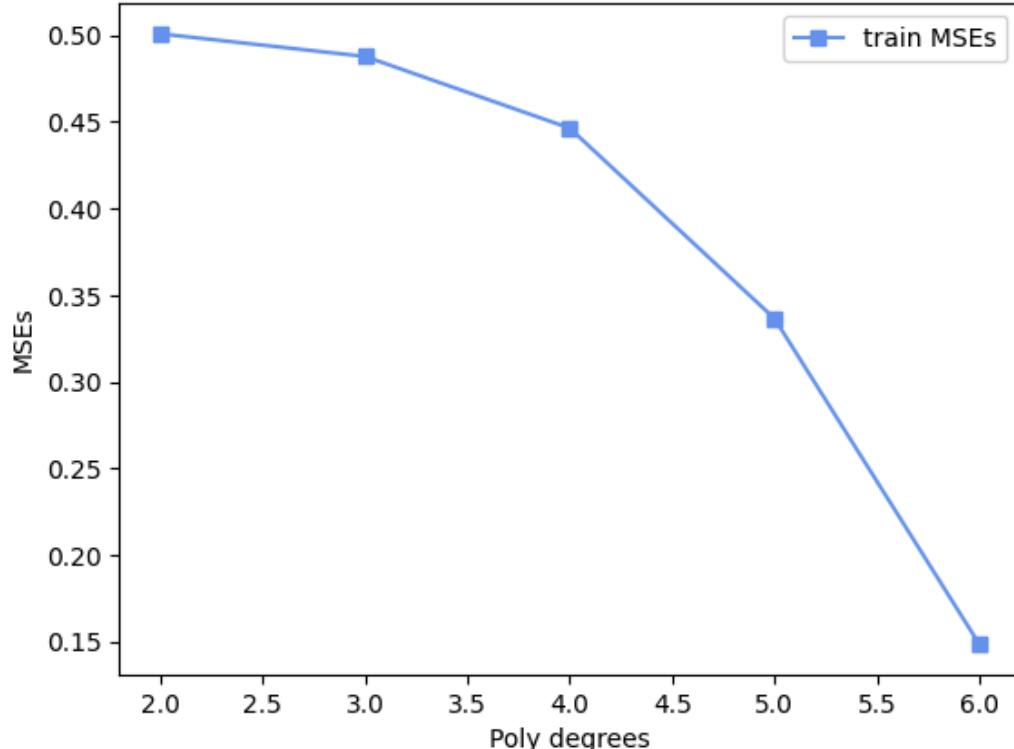
techno = subset_genres_dict[6]
techno_title = f"{techno} train MSEs for power transformed normalized data"
validate_title = f"{techno} validate MSEs for power transformed normalized
techno_overall_title = f"{techno} train and validate mean squared errors for
plot_train_cv_mses(techno_degrees, techno_train_mses, techno_title)
plot_train_cv_mses(techno_degrees, techno_validate_mses, validate_title)

train_validate_mses_2plots(techno_degrees, techno_train_mses, techno_validate_

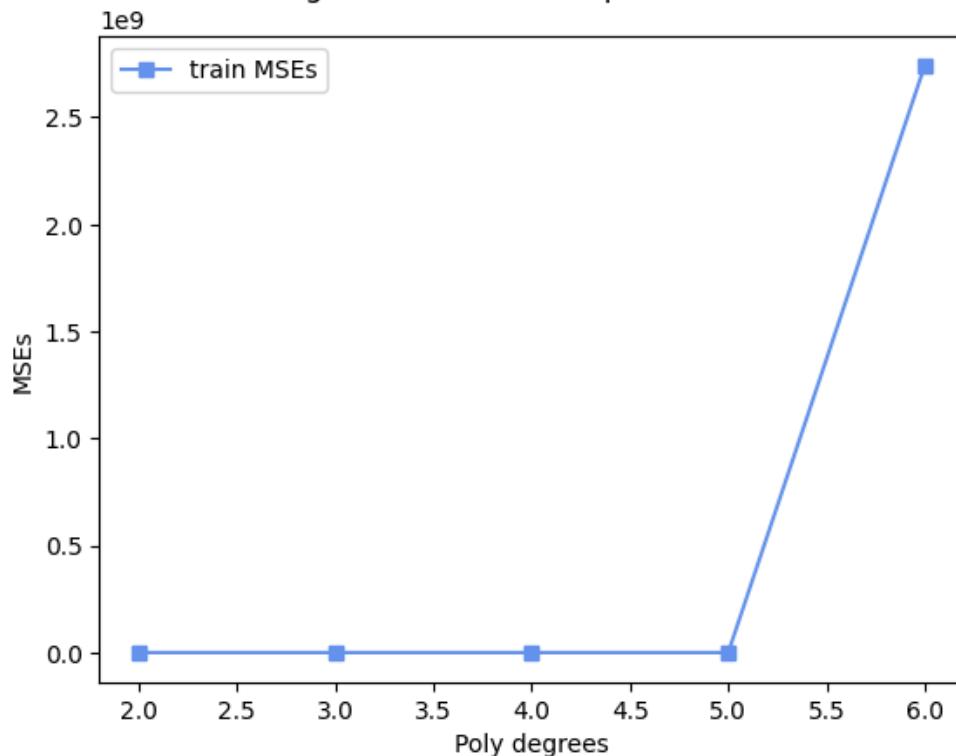
```

Techno training MSEs: [0.49103616096361785, 0.47043376169919926, 0.421427191
1001973, 0.28684657005670333, 0.07362797367523534]
Techno validation MSEs: [0.5042744545195588, 0.5211371599036397, 0.683231348
4270566, 13.809691321430849, 38683.93853297744]

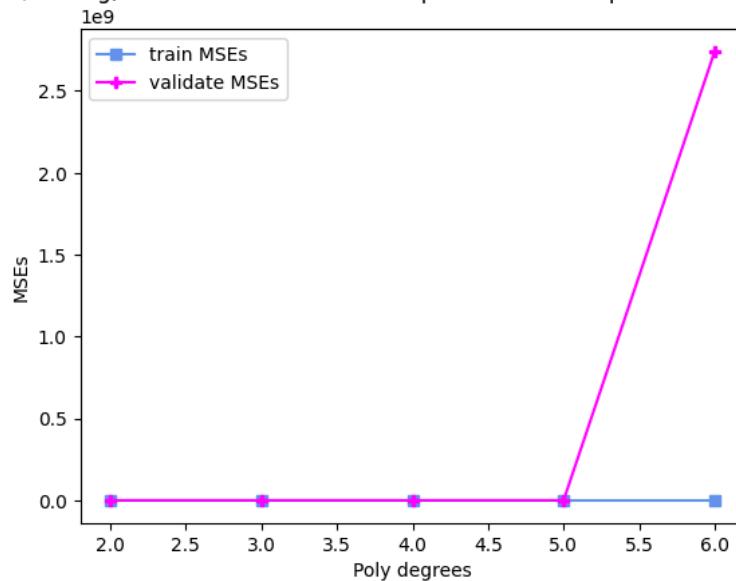
Techno (Peak Time/Driving) train MSEs for power transformed normalized data



Techno (Peak Time/Driving) validate MSEs for power transformed normalized data



Techno (Peak Time/Driving) train and validate mean squared errors for power transformed normalized data



```
In [29]: power_techno_poly4 = PolynomialFeatures(degree=4)
power_techno_x_poly4_train = power_techno_poly4.fit_transform(power_techno_x)
power_techno_x_poly4_validate = power_techno_poly4.fit_transform(power_techno_x)
power_techno_x_poly4_test = power_techno_poly4.fit_transform(power_techno_x)

power_techno_model4 = linear_model.LinearRegression()
power_techno_model4.fit(power_techno_x_poly4_train, power_techno_ytrain)
y_poly_pred4_train = power_techno_model4.predict(power_techno_x_poly4_train)

power_techno_model4.fit(power_techno_x_poly4_validate, power_techno_yvalidate)
y_poly_pred4_validate = power_techno_model4.predict(power_techno_x_poly4_val)
```

```

power_techno_model4.fit(power_techno_x_poly4_test, power_techno_ytest)
y_poly_pred4_test = power_techno_model4.predict(power_techno_x_poly4_test)

power_techno_sorted_indices_train = lumpnump.argsort(power_techno_xtrain[:, 1])
power_techno_sorted_indices_validate = lumpnump.argsort(power_techno_xvalidate[:, 1])
power_techno_sorted_indices_test = lumpnump.argsort(power_techno_xtest[:, 1])

power_techno_sorted_xtrain = power_techno_xtrain[power_techno_sorted_indices_train]
power_techno_sorted_xvalidate = power_techno_xvalidate[power_techno_sorted_indices_validate]
power_techno_sorted_xtest = power_techno_xtest[power_techno_sorted_indices_test]

power_techno_sorted_ytrain = power_techno_ytrain[power_techno_sorted_indices_train]
power_techno_sorted_yvalidate = power_techno_yvalidate[power_techno_sorted_indices_validate]
power_techno_sorted_ytest = power_techno_ytest[power_techno_sorted_indices_test]

power_techno_sorted_y_poly_pred4_train = y_poly_pred4_train[power_techno_sorted_indices_train]
power_techno_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_techno_sorted_indices_validate]
power_techno_sorted_y_poly_pred4_test = y_poly_pred4_test[power_techno_sorted_indices_test]

# Polynomial regression for degree 5
power_techno_poly5 = PolynomialFeatures(degree=5)
power_techno_x_poly5_train = power_techno_poly5.fit_transform(power_techno_xtrain)
power_techno_x_poly5_validate = power_techno_poly5.fit_transform(power_techno_xvalidate)
power_techno_x_poly5_test = power_techno_poly5.fit_transform(power_techno_xtest)

# Fitting the polynomial regression model for degree 5
power_techno_model5 = linear_model.LinearRegression()
power_techno_model5.fit(power_techno_x_poly5_train, power_techno_ytrain)
y_poly_pred5_train = power_techno_model5.predict(power_techno_x_poly5_train)

power_techno_model5.fit(power_techno_x_poly5_validate, power_techno_yvalidate)
y_poly_pred5_validate = power_techno_model5.predict(power_techno_x_poly5_validate)

power_techno_model5.fit(power_techno_x_poly5_test, power_techno_ytest)
y_poly_pred5_test = power_techno_model5.predict(power_techno_x_poly5_test)

power_techno_sorted_y_poly_pred5_train = y_poly_pred5_train[power_techno_sorted_indices_train]
power_techno_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_techno_sorted_indices_validate]
power_techno_sorted_y_poly_pred5_test = y_poly_pred5_test[power_techno_sorted_indices_test]

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_techno_sorted_xtrain, power_techno_sorted_ytrain, color='blue')
plt.plot(power_techno_sorted_xtrain, power_techno_sorted_y_poly_pred4_train, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_techno_train = r2_score(power_techno_ytrain, y_poly_pred4_train)
mse_4_techno_train = mean_squared_error(power_techno_ytrain, y_poly_pred4_train)
pearson_corr_4_techno_train, _ = pearsonr(power_techno_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f'R^2: {r2_4_techno_train:.4f}\nMSE: {mse_4_techno_train:.4f}')

```

```

plt.subplot(3, 2, 3)
plt.scatter(power_techno_sorted_xvalidate, power_techno_sorted_yvalidate, color='blue')
plt.plot(power_techno_sorted_xvalidate, power_techno_sorted_y_poly_pred4_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_techno_validate = r2_score(power_techno_yvalidate, y_poly_pred4_validate)
mse_4_techno_validate = mean_squared_error(power_techno_yvalidate, y_poly_pred4_validate)
pearson_corr_4_techno_validate, _ = pearsonr(power_techno_yvalidate, y_poly_pred4_validate)
plt.figtext(0.1, 0.62, f"R^2: {r2_4_techno_validate:.4f}\nMSE: {mse_4_techno_validate:.4f}")

plt.subplot(3, 2, 5)
plt.scatter(power_techno_sorted_xtest, power_techno_sorted_ytest, color='blue')
plt.plot(power_techno_sorted_xtest, power_techno_sorted_y_poly_pred4_test)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_techno_test = r2_score(power_techno_ytest, y_poly_pred4_test)
mse_4_techno_test = mean_squared_error(power_techno_ytest, y_poly_pred4_test)
pearson_corr_4_techno_test, _ = pearsonr(power_techno_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_techno_test:.4f}\nMSE: {mse_4_techno_test:.4f}")

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_techno_sorted_xtrain, power_techno_sorted_ytrain, color='blue')
plt.plot(power_techno_sorted_xtrain, power_techno_sorted_y_poly_pred5_train)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_techno_train = r2_score(power_techno_ytrain, y_poly_pred5_train)
mse_5_techno_train = mean_squared_error(power_techno_ytrain, y_poly_pred5_train)
pearson_corr_5_techno_train, _ = pearsonr(power_techno_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_techno_train:.4f}\nMSE: {mse_5_techno_train:.4f}")

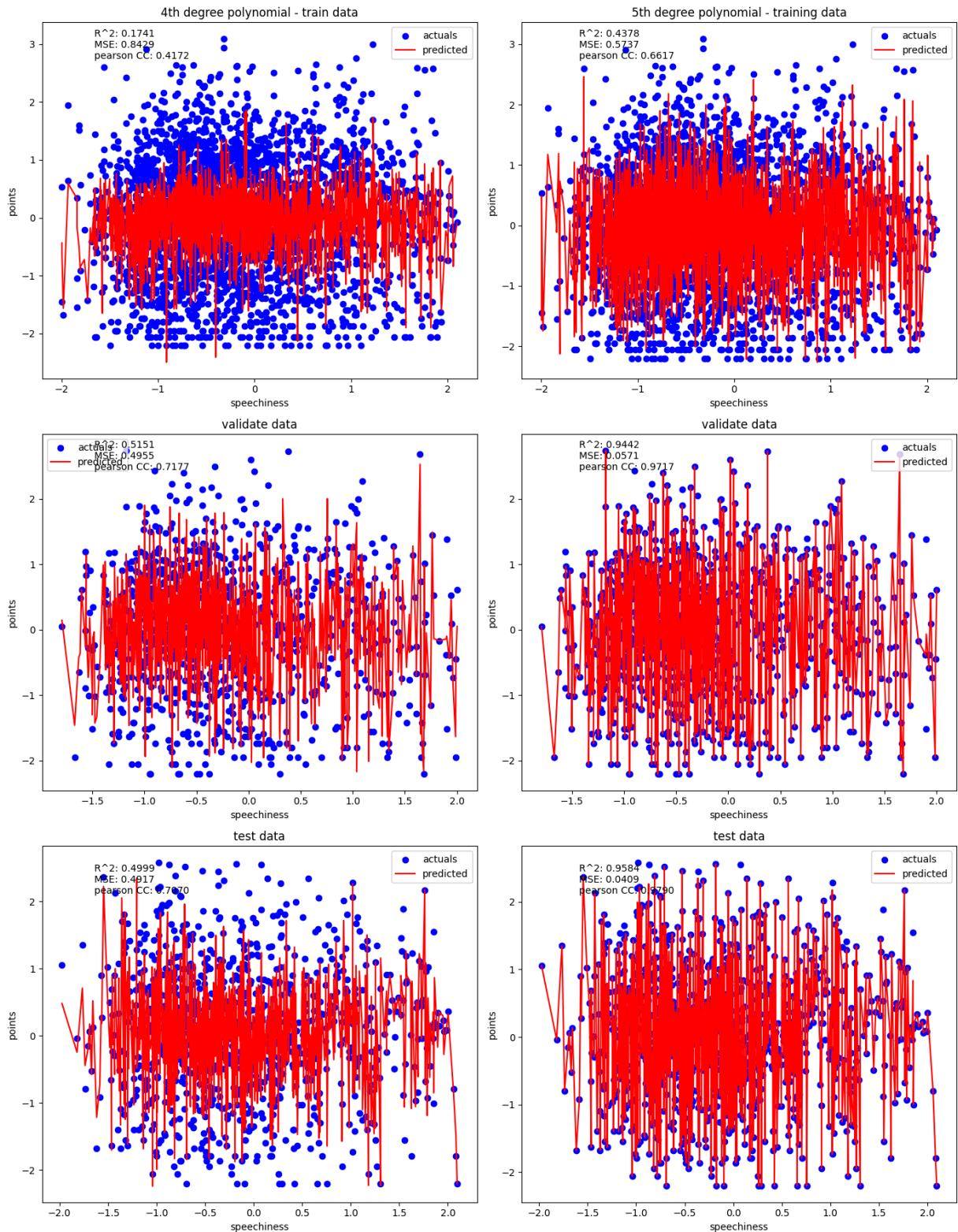
plt.subplot(3, 2, 4)
plt.scatter(power_techno_sorted_xvalidate, power_techno_sorted_yvalidate, color='blue')
plt.plot(power_techno_sorted_xvalidate, power_techno_sorted_y_poly_pred5_validate)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_techno_validate = r2_score(power_techno_yvalidate, y_poly_pred5_validate)
mse_5_techno_validate = mean_squared_error(power_techno_yvalidate, y_poly_pred5_validate)
pearson_corr_5_techno_validate, _ = pearsonr(power_techno_yvalidate, y_poly_pred5_validate)
plt.figtext(0.6, 0.62, f"R^2: {r2_5_techno_validate:.4f}\nMSE: {mse_5_techno_validate:.4f}")

plt.subplot(3, 2, 6)
plt.scatter(power_techno_sorted_xtest, power_techno_sorted_ytest, color='blue')
plt.plot(power_techno_sorted_xtest, power_techno_sorted_y_poly_pred5_test)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')

```

```
r2_5_techno_test = r2_score(power_techno_ytest, y_poly_pred5_test)
mse_5_techno_test = mean_squared_error(power_techno_ytest, y_poly_pred5_test)
pearson_corr_5_techno_test, _ = pearsonr(power_techno_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f'R^2: {r2_5_techno_test:.4f}\nMSE: {mse_5_techno_test:.4f}\npearson CC: {pearson_corr_5_techno_test:.4f}')

plt.tight_layout()
plt.show()
```



Melodic House and Techno

```
In [30]: power_melodic_degrees = [2, 3, 4, 5, 6]
power_melodic_genre_mask = (power_scaling_train[:, 3] == 90.)
power_melodic_genre_validate_mask = (power_scaling_validate[:, 3] == 90.)
power_melodic_genre_test_mask = (power_scaling_test[:, 3] == 90.)

power_melodic_xtrain = power_scaling_train[power_melodic_genre_mask, 4:12]
power_melodic_ytrain = power_scaling_train[power_melodic_genre_mask, 12]

power_melodic_xvalidate = power_scaling_validate[power_melodic_genre_validate_
power_melodic_yvalidate = power_scaling_validate[power_melodic_genre_validate_

power_melodic_xtest = power_scaling_test[power_melodic_genre_test_mask, 4:12]
power_melodic_ytest = power_scaling_test[power_melodic_genre_test_mask, 12]

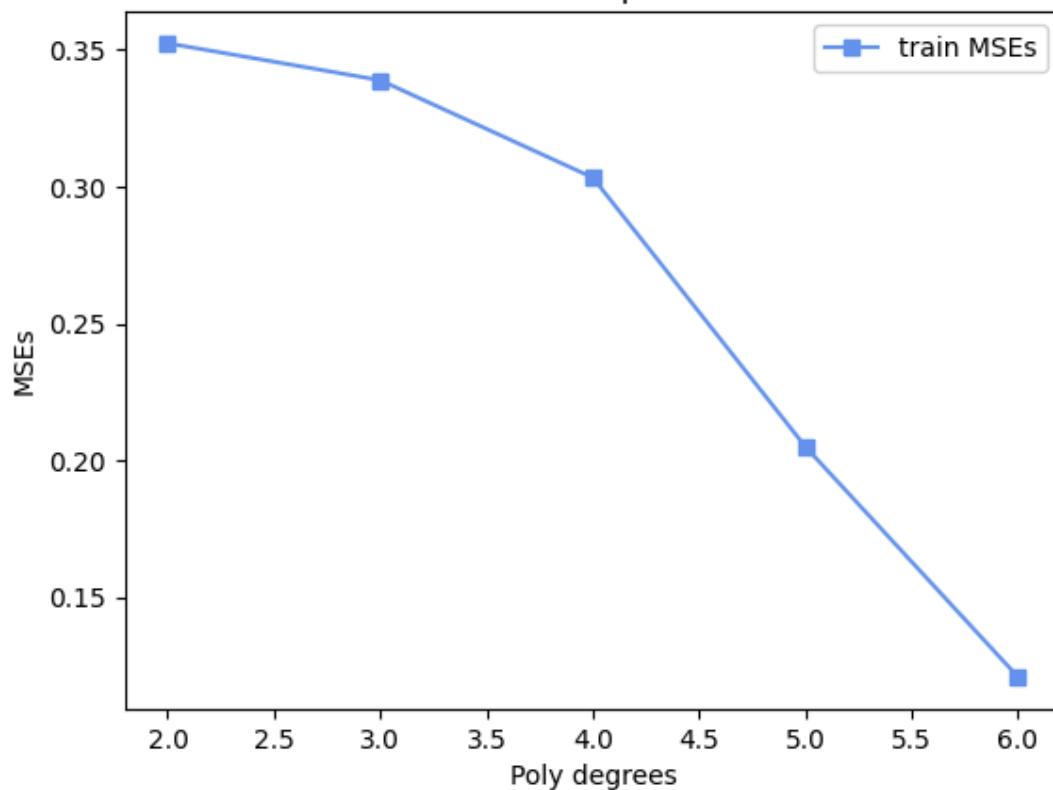
power_melodic_train_mses, power_melodic_validate_mses = poly_degree_regress(
    power_melodic_degrees)

# Print results
print("melodic training MSEs:", power_melodic_train_mses)
print("melodic validation MSEs:", power_melodic_validate_mses)

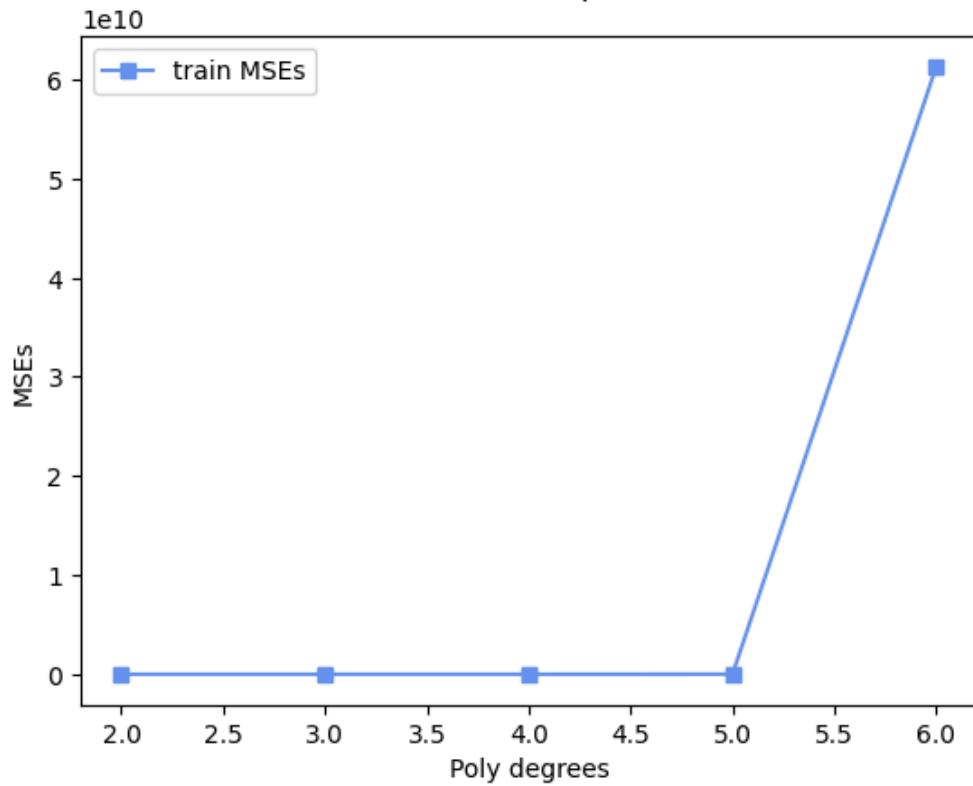
melodic = subset_genres_dict[90]
melodic_title = f"{melodic} train MSEs for power transformed normalized data"
validate_title = f"{melodic} validate MSEs for power transformed normalized data"
melodic_overall_title = f"{melodic} train and validate mean squared errors for all genres"
plot_train_cv_mses(melodic_degrees, melodic_train_mses, melodic_title)
plot_train_cv_mses(melodic_degrees, melodic_validate_mses, validate_title)

train_validate_mses_2plots(melodic_degrees, melodic_train_mses, melodic_vali
melodic training MSEs: [0.4768052306472042, 0.4505406617871817, 0.3891511467
4022633, 0.23767197060791803, 0.07171417256272077]
melodic validation MSEs: [0.4733992991744044, 0.5451002986634893, 0.88053613
68195503, 159.49396327652036, 5.8732486058404125e+20]
```

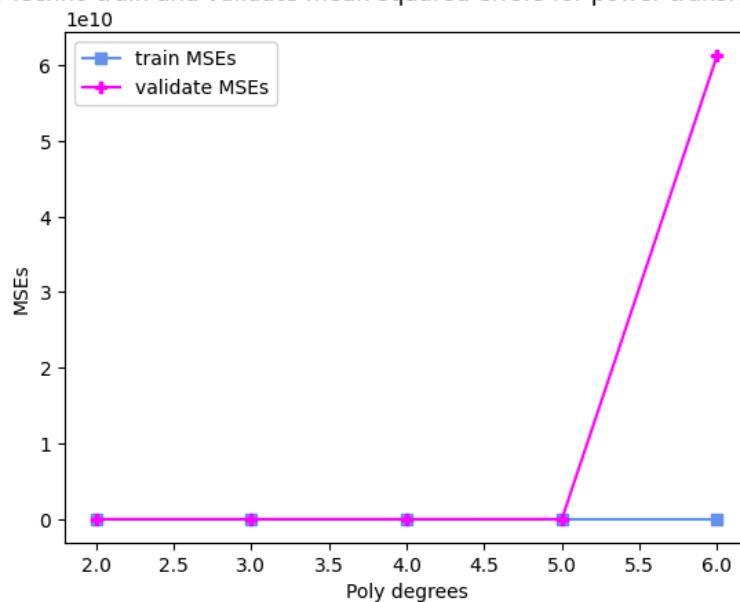
Melodic House & Techno train MSEs for power transformed normalized data



Melodic House & Techno validate MSEs for power transformed normalized data



Melodic House & Techno train and validate mean squared errors for power transformed normalized data



```
In [31]: power_melodic_poly4 = PolynomialFeatures(degree=4)
power_melodic_x_poly4_train = power_melodic_poly4.fit_transform(power_melodic_xtrain)
power_melodic_x_poly4_validate = power_melodic_poly4.fit_transform(power_melodic_xvalidate)
power_melodic_x_poly4_test = power_melodic_poly4.fit_transform(power_melodic_xtest)

power_melodic_model4 = linear_model.LinearRegression()
power_melodic_model4.fit(power_melodic_x_poly4_train, power_melodic_ytrain)
y_poly_pred4_train = power_melodic_model4.predict(power_melodic_x_poly4_train)

power_melodic_model4.fit(power_melodic_x_poly4_validate, power_melodic_yvalidate)
y_poly_pred4_validate = power_melodic_model4.predict(power_melodic_x_poly4_validate)

power_melodic_model4.fit(power_melodic_x_poly4_test, power_melodic_ytest)
y_poly_pred4_test = power_melodic_model4.predict(power_melodic_x_poly4_test)

power_melodic_sorted_indices_train = lumpyump.argsort(power_melodic_xtrain[:, 0])
power_melodic_sorted_indices_validate = lumpyump.argsort(power_melodic_xvalidate[:, 0])
power_melodic_sorted_indices_test = lumpyump.argsort(power_melodic_xtest[:, 0])

power_melodic_sorted_xtrain = power_melodic_xtrain[power_melodic_sorted_indices_train]
power_melodic_sorted_xvalidate = power_melodic_xvalidate[power_melodic_sorted_indices_validate]
power_melodic_sorted_xtest = power_melodic_xtest[power_melodic_sorted_indices_test]

power_melodic_sorted_ytrain = power_melodic_ytrain[power_melodic_sorted_indices_train]
power_melodic_sorted_yvalidate = power_melodic_yvalidate[power_melodic_sorted_indices_validate]
power_melodic_sorted_ytest = power_melodic_ytest[power_melodic_sorted_indices_test]

power_melodic_sorted_y_poly_pred4_train = y_poly_pred4_train[power_melodic_sorted_indices_train]
power_melodic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_melodic_sorted_indices_validate]
power_melodic_sorted_y_poly_pred4_test = y_poly_pred4_test[power_melodic_sorted_indices_test]

# Polynomial regression for degree 5
power_melodic_poly5 = PolynomialFeatures(degree=5)
power_melodic_x_poly5_train = power_melodic_poly5.fit_transform(power_melodic_xtrain)
power_melodic_x_poly5_validate = power_melodic_poly5.fit_transform(power_melodic_xvalidate)
power_melodic_x_poly5_test = power_melodic_poly5.fit_transform(power_melodic_xtest)
```

```

# Fitting the polynomial regression model for degree 5
power_melodic_model5 = linear_model.LinearRegression()
power_melodic_model5.fit(power_melodic_x_poly5_train, power_melodic_ytrain)
y_poly_pred5_train = power_melodic_model5.predict(power_melodic_x_poly5_train)

power_melodic_model5.fit(power_melodic_x_poly5_validate, power_melodic_yvalidate)
y_poly_pred5_validate = power_melodic_model5.predict(power_melodic_x_poly5_validate)

power_melodic_model5.fit(power_melodic_x_poly5_test, power_melodic_ytest)
y_poly_pred5_test = power_melodic_model5.predict(power_melodic_x_poly5_test)

power_melodic_sorted_y_poly_pred5_train = y_poly_pred5_train[power_melodic_sorted_index]
power_melodic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_melodic_sorted_index]
power_melodic_sorted_y_poly_pred5_test = y_poly_pred5_test[power_melodic_sorted_index]

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_melodic_sorted_xtrain, power_melodic_sorted_ytrain, color='blue')
plt.plot(power_melodic_sorted_xtrain, power_melodic_sorted_y_poly_pred4_train, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_melodic_train = r2_score(power_melodic_ytrain, y_poly_pred4_train)
mse_4_melodic_train = mean_squared_error(power_melodic_ytrain, y_poly_pred4_train)
pearson_corr_4_melodic_train, _ = pearsonr(power_melodic_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f"R^2: {r2_4_melodic_train:.4f}\nMSE: {mse_4_melodic_train:.4f}\nPearson Corr: {pearson_corr_4_melodic_train:.4f}")

plt.subplot(3, 2, 3)
plt.scatter(power_melodic_sorted_xvalidate, power_melodic_sorted_yvalidate, color='blue')
plt.plot(power_melodic_sorted_xvalidate, power_melodic_sorted_y_poly_pred4_validate, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_melodic_validate = r2_score(power_melodic_yvalidate, y_poly_pred4_validate)
mse_4_melodic_validate = mean_squared_error(power_melodic_yvalidate, y_poly_pred4_validate)
pearson_corr_4_melodic_validate, _ = pearsonr(power_melodic_yvalidate, y_poly_pred4_validate)
plt.figtext(0.1, 0.62, f"R^2: {r2_4_melodic_validate:.4f}\nMSE: {mse_4_melodic_validate:.4f}\nPearson Corr: {pearson_corr_4_melodic_validate:.4f}")

plt.subplot(3, 2, 5)
plt.scatter(power_melodic_sorted_xtest, power_melodic_sorted_ytest, color='blue')
plt.plot(power_melodic_sorted_xtest, power_melodic_sorted_y_poly_pred4_test, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_melodic_test = r2_score(power_melodic_ytest, y_poly_pred4_test)
mse_4_melodic_test = mean_squared_error(power_melodic_ytest, y_poly_pred4_test)
pearson_corr_4_melodic_test, _ = pearsonr(power_melodic_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_melodic_test:.4f}\nMSE: {mse_4_melodic_test:.4f}\nPearson Corr: {pearson_corr_4_melodic_test:.4f}")

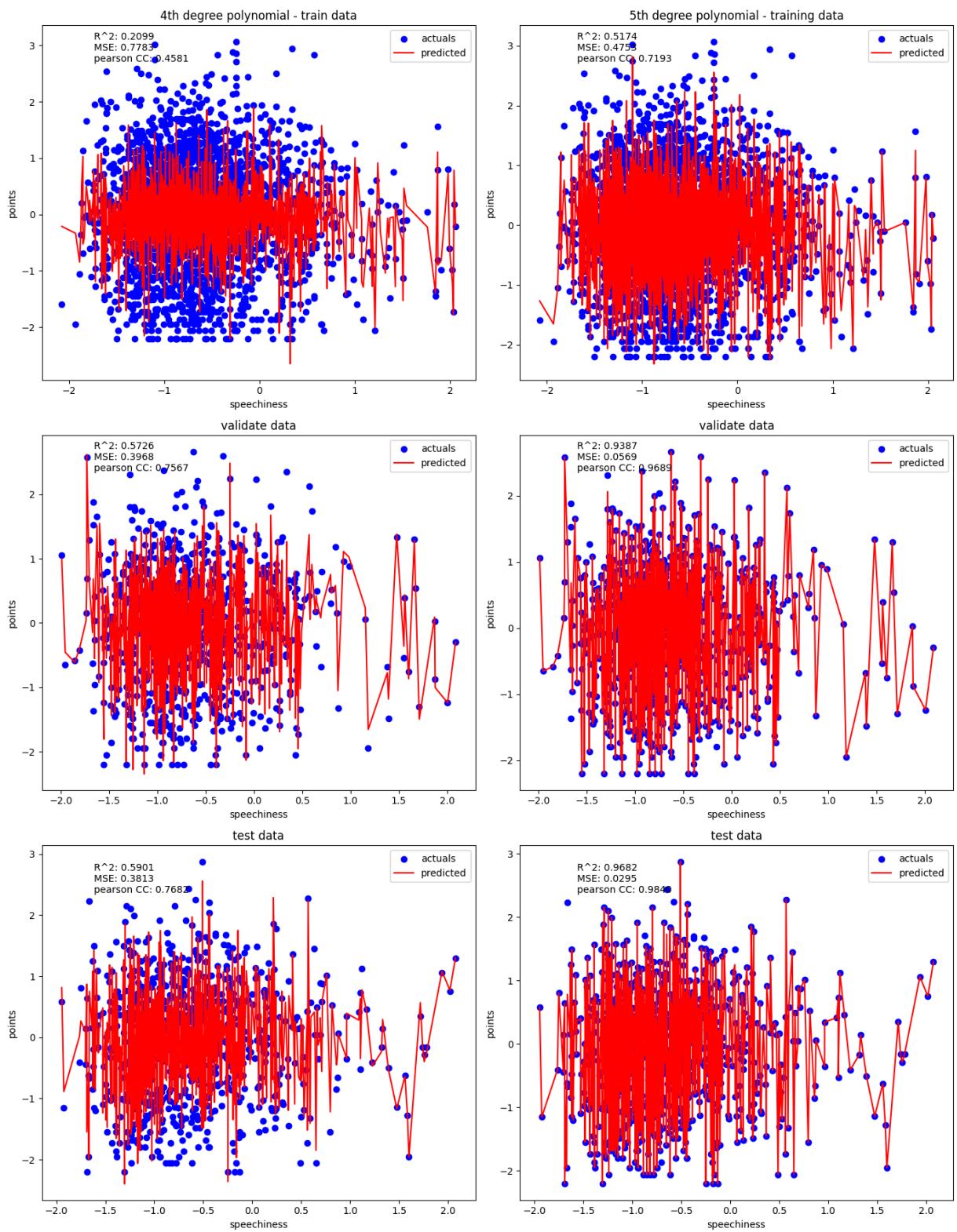
```

```
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_melodic_sorted_xtrain, power_melodic_sorted_ytrain, color='blue')
plt.plot(power_melodic_sorted_xtrain, power_melodic_sorted_y_poly_pred5_train, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_melodic_train = r2_score(power_melodic_ytrain, y_poly_pred5_train)
mse_5_melodic_train = mean_squared_error(power_melodic_ytrain, y_poly_pred5_train)
pearson_corr_5_melodic_train, _ = pearsonr(power_melodic_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_melodic_train:.4f}\nMSE: {mse_5_melodic_train:.4f}")

plt.subplot(3, 2, 4)
plt.scatter(power_melodic_sorted_xvalidate, power_melodic_sorted_yvalidate, color='blue')
plt.plot(power_melodic_sorted_xvalidate, power_melodic_sorted_y_poly_pred5_validate, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_melodic_validate = r2_score(power_melodic_yvalidate, y_poly_pred5_validate)
mse_5_melodic_validate = mean_squared_error(power_melodic_yvalidate, y_poly_pred5_validate)
pearson_corr_5_melodic_validate, _ = pearsonr(power_melodic_yvalidate, y_poly_pred5_validate)
plt.figtext(0.6, 0.62, f"R^2: {r2_5_melodic_validate:.4f}\nMSE: {mse_5_melodic_validate:.4f}")

plt.subplot(3, 2, 6)
plt.scatter(power_melodic_sorted_xtest, power_melodic_sorted_ytest, color='blue')
plt.plot(power_melodic_sorted_xtest, power_melodic_sorted_y_poly_pred5_test, color='red')
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_melodic_test = r2_score(power_melodic_ytest, y_poly_pred5_test)
mse_5_melodic_test = mean_squared_error(power_melodic_ytest, y_poly_pred5_test)
pearson_corr_5_melodic_test, _ = pearsonr(power_melodic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_melodic_test:.4f}\nMSE: {mse_5_melodic_test:.4f}")

plt.tight_layout()
plt.show()
```



Organic house/down tempo

```
In [32]: power_organic_degrees = [2, 3, 4, 5, 6]
power_organic_genre_mask = (power_scaling_train[:, 3] == 93.)
power_organic_genre_validate_mask = (power_scaling_validate[:, 3] == 93.)
power_organic_genre_test_mask = (power_scaling_test[:, 3] == 93.)
```

```

power_organic_xtrain = power_scaling_train[power_organic_genre_mask, 4:12]
power_organic_ytrain = power_scaling_train[power_organic_genre_mask, 12]

power_organic_xvalidate = power_scaling_validate[power_organic_genre_validate]
power_organic_yvalidate = power_scaling_validate[power_organic_genre_validate]

power_organic_xtest = power_scaling_test[power_organic_genre_test_mask, 4:12]
power_organic_ytest = power_scaling_test[power_organic_genre_test_mask, 12]

power_organic_train_mses, power_organic_validate_mses = poly_degree_regress()

# Print results
print("organic training MSEs:", power_organic_train_mses)
print("organic validation MSEs:", power_organic_validate_mses)

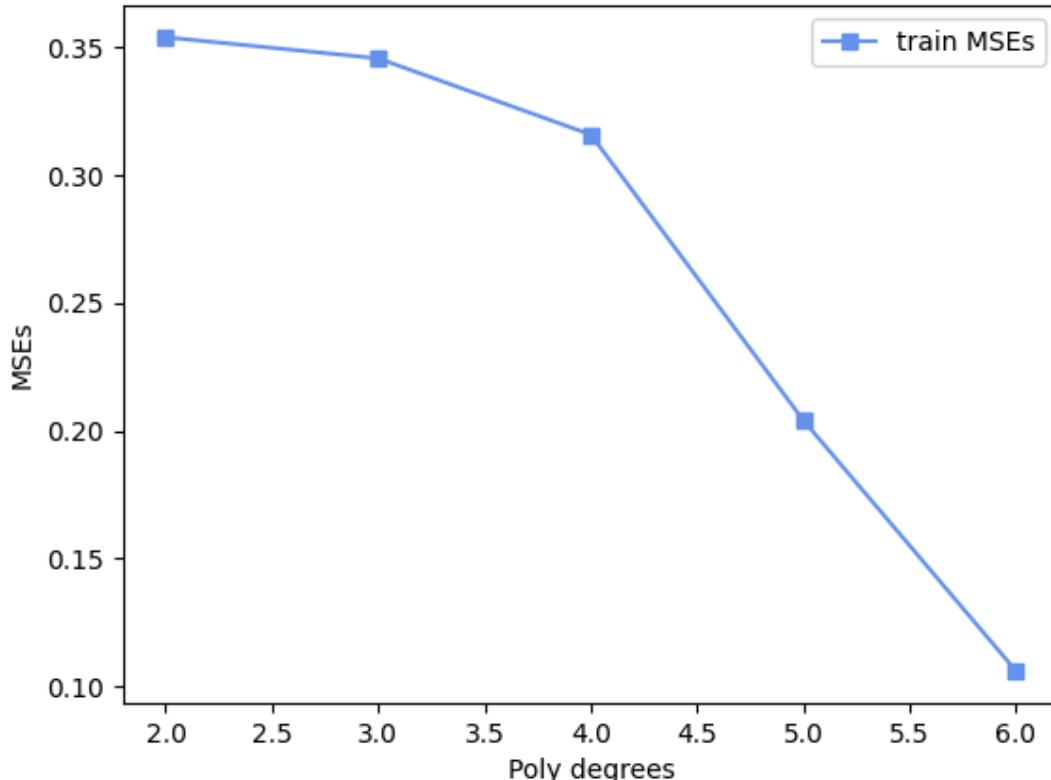
organic = subset_genres_dict[90]
organic_title = f"{organic} train MSEs for power transformed normalized data"
validate_title = f"{organic} validate MSEs for power transformed normalized data"
organic_overall_title = f"{organic} train and validate mean squared errors for power transformed normalized data"
plot_train_cv_mses(organic_degrees, organic_train_mses, organic_title)
plot_train_cv_mses(organic_degrees, organic_validate_mses, validate_title)

train_validate_mses_2plots(organic_degrees, organic_train_mses, organic_validate_mses)

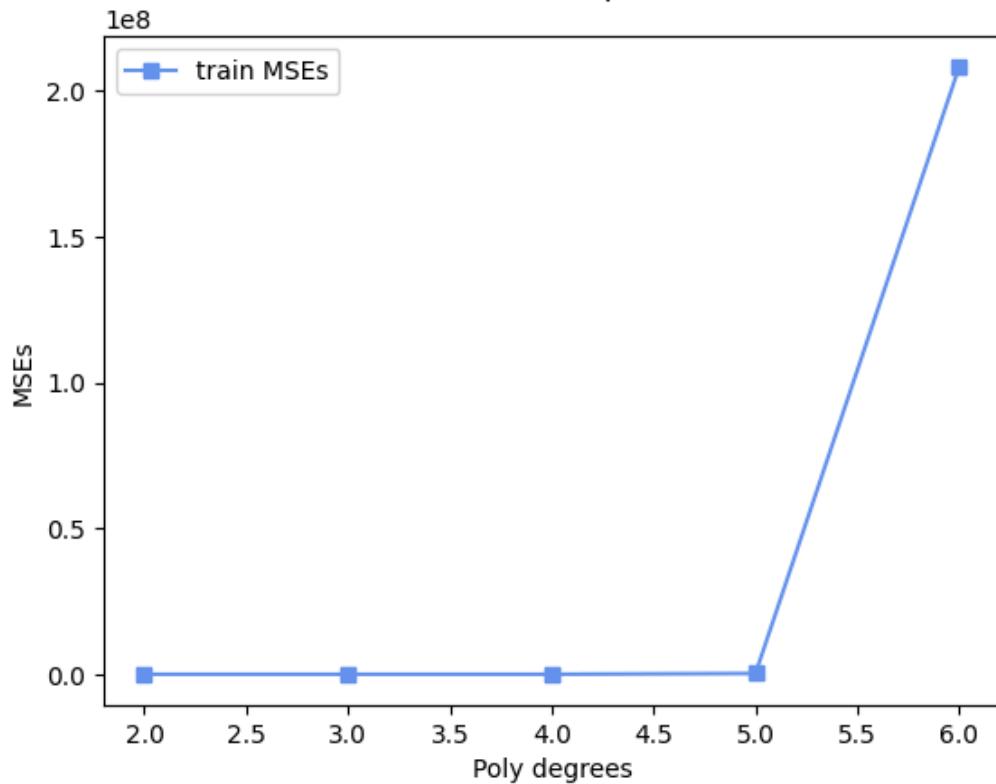
```

organic training MSEs: [0.4650892795882737, 0.43810757859862703, 0.37981071864055105, 0.21383612568540866, 1.0571771216814398]
 organic validation MSEs: [0.4908675822819797, 0.537852638696692, 1.299767994052033, 1015.6403481821587, 1.420483053473572e+24]

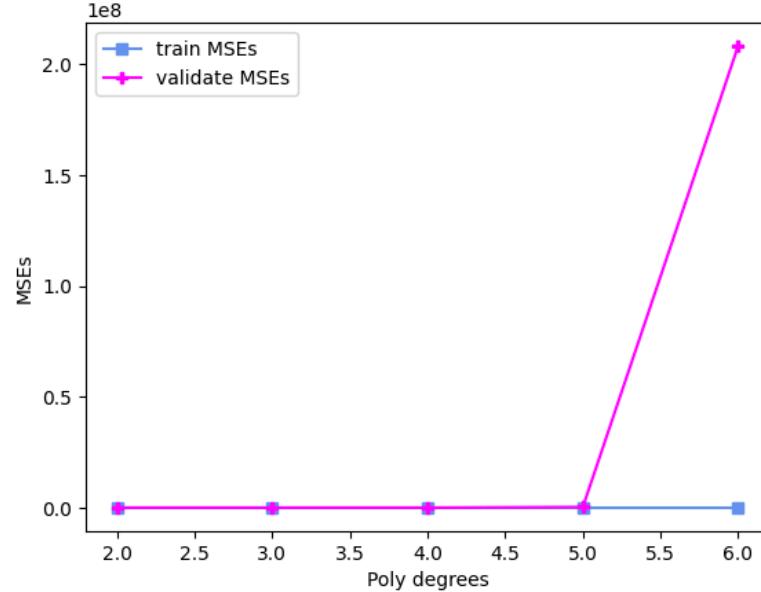
Melodic House & Techno train MSEs for power transformed normalized data



Melodic House & Techno validate MSEs for power transformed normalized data



Melodic House & Techno train and validate mean squared errors for power transformed normalized data



```
In [33]: power_organic_poly4 = PolynomialFeatures(degree=4)
power_organic_x_poly4_train = power_organic_poly4.fit_transform(power_organic_x)
power_organic_x_poly4_validate = power_organic_poly4.fit_transform(power_organic_x)
power_organic_x_poly4_test = power_organic_poly4.fit_transform(power_organic_x)

power_organic_model4 = linear_model.LinearRegression()
power_organic_model4.fit(power_organic_x_poly4_train, power_organic_ytrain)
y_poly_pred4_train = power_organic_model4.predict(power_organic_x_poly4_train)

power_organic_model4.fit(power_organic_x_poly4_validate, power_organic_yvalidate)
y_poly_pred4_validate = power_organic_model4.predict(power_organic_x_poly4_validate)
```

```

power_organic_model4.fit(power_organic_x_poly4_test, power_organic_ytest)
y_poly_pred4_test = power_organic_model4.predict(power_organic_x_poly4_test)

power_organic_sorted_indices_train = lumpnump.argsort(power_organic_xtrain[:,])
power_organic_sorted_indices_validate = lumpnump.argsort(power_organic_xvali
power_organic_sorted_indices_test = lumpnump.argsort(power_organic_xtest[:,],

power_organic_sorted_xtrain = power_organic_xtrain[power_organic_sorted_index
power_organic_sorted_xvalidate = power_organic_xvalidate[power_organic_sorted_
power_organic_sorted_xtest = power_organic_xtest[power_organic_sorted_index

power_organic_sorted_ytrain = power_organic_ytrain[power_organic_sorted_index
power_organic_sorted_yvalidate = power_organic_yvalidate[power_organic_sorted_
power_organic_sorted_ytest = power_organic_ytest[power_organic_sorted_index

power_organic_sorted_y_poly_pred4_train = y_poly_pred4_train[power_organic_sor
power_organic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_organ
power_organic_sorted_y_poly_pred4_test = y_poly_pred4_test[power_organic_sor

# Polynomial regression for degree 5
power_organic_poly5 = PolynomialFeatures(degree=5)
power_organic_x_poly5_train = power_organic_poly5.fit_transform(power_organic_x
power_organic_x_poly5_validate = power_organic_poly5.fit_transform(power_organ
power_organic_x_poly5_test = power_organic_poly5.fit_transform(power_organic_x

# Fitting the polynomial regression model for degree 5
power_organic_model5 = linear_model.LinearRegression()
power_organic_model5.fit(power_organic_x_poly5_train, power_organic_ytrain)
y_poly_pred5_train = power_organic_model5.predict(power_organic_x_poly5_train)

power_organic_model5.fit(power_organic_x_poly5_validate, power_organic_yvali
y_poly_pred5_validate = power_organic_model5.predict(power_organic_x_poly5_v

power_organic_model5.fit(power_organic_x_poly5_test, power_organic_ytest)
y_poly_pred5_test = power_organic_model5.predict(power_organic_x_poly5_test)

power_organic_sorted_y_poly_pred5_train = y_poly_pred5_train[power_organic_sor
power_organic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_organ
power_organic_sorted_y_poly_pred5_test = y_poly_pred5_test[power_organic_sor

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_organic_sorted_xtrain, power_organic_sorted_ytrain, color=
plt.plot(power_organic_sorted_xtrain, power_organic_sorted_y_poly_pred4_train)
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_organic_train = r2_score(power_organic_ytrain, y_poly_pred4_train)
mse_4_organic_train = mean_squared_error(power_organic_ytrain, y_poly_pred4_
pearson_corr_4_organic_train, _ = pearsonr(power_organic_ytrain, y_poly_pred4_
plt.figtext(0.1, 0.95, f'R^2: {r2_4_organic_train:.4f}\nMSE: {mse_4_organic_

```

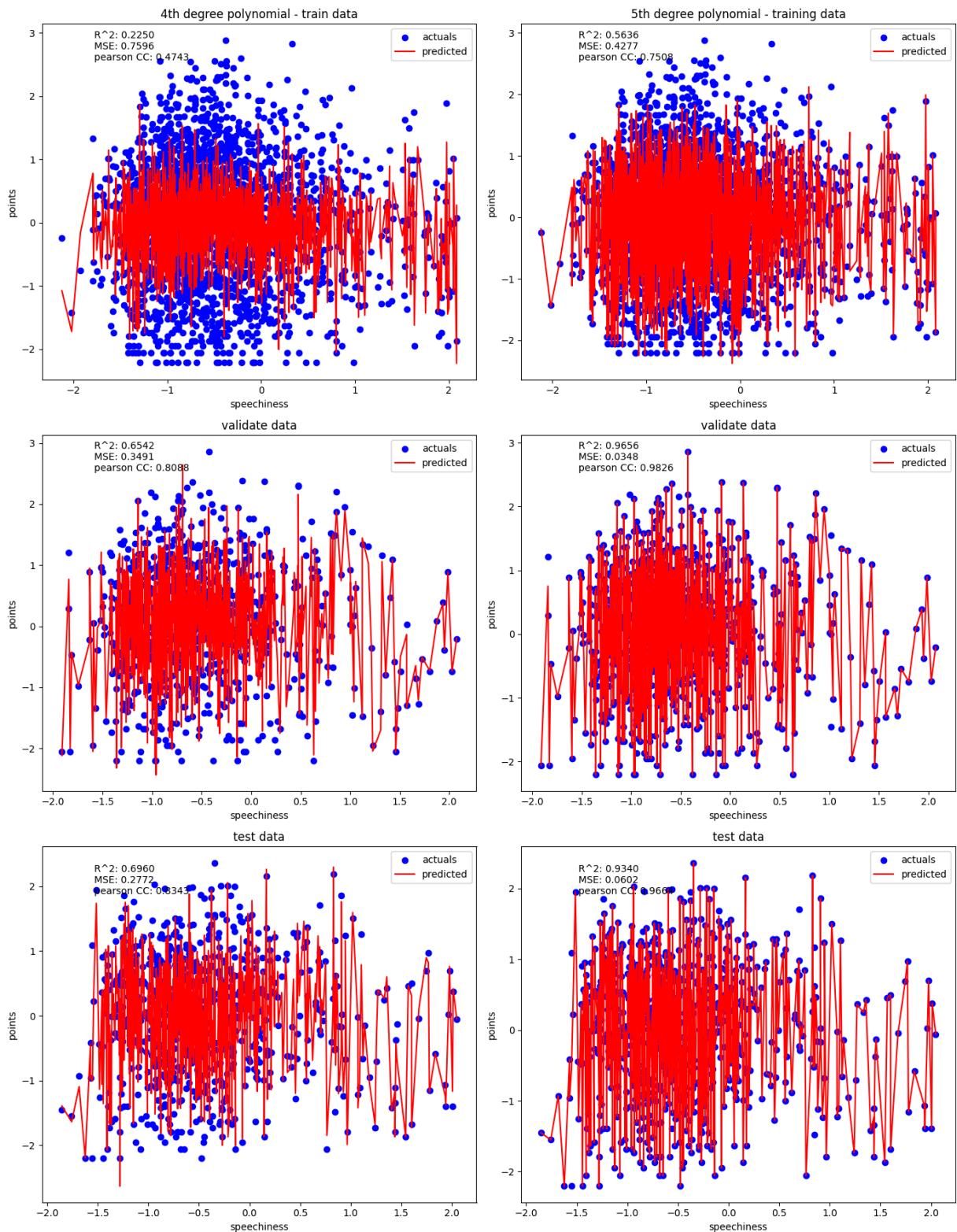
```
plt.subplot(3, 2, 3)
plt.scatter(power_organic_sorted_xvalidate, power_organic_sorted_yvalidate,
plt.plot(power_organic_sorted_xvalidate, power_organic_sorted_y_poly_pred4_
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_organic_validate = r2_score(power_organic_yvalidate, y_poly_pred4_vali
mse_4_organic_validate = mean_squared_error(power_organic_yvalidate, y_poly_
pearson_corr_4_organic_validate, _ = pearsonr(power_organic_yvalidate, y_poly_
plt.figtext(0.1, 0.62, f"R^2: {r2_4_organic_validate:.4f}\nMSE: {mse_4_organic_
plt.subplot(3, 2, 5)
plt.scatter(power_organic_sorted_xtest, power_organic_sorted_ytest, color='b'
plt.plot(power_organic_sorted_xtest, power_organic_sorted_y_poly_pred4_test,
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_organic_test = r2_score(power_organic_ytest, y_poly_pred4_test)
mse_4_organic_test = mean_squared_error(power_organic_ytest, y_poly_pred4_te
pearson_corr_4_organic_test, _ = pearsonr(power_organic_ytest, y_poly_pred4_
plt.figtext(0.1, 0.28, f"R^2: {r2_4_organic_test:.4f}\nMSE: {mse_4_organic_t
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_organic_sorted_xtrain, power_organic_sorted_ytrain, color='r'
plt.plot(power_organic_sorted_xtrain, power_organic_sorted_y_poly_pred5_train,
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_organic_train = r2_score(power_organic_ytrain, y_poly_pred5_train)
mse_5_organic_train = mean_squared_error(power_organic_ytrain, y_poly_pred5_tr
pearson_corr_5_organic_train, _ = pearsonr(power_organic_ytrain, y_poly_pred5_
plt.figtext(0.6, 0.95, f"R^2: {r2_5_organic_train:.4f}\nMSE: {mse_5_organic_
plt.subplot(3, 2, 4)
plt.scatter(power_organic_sorted_xvalidate, power_organic_sorted_yvalidate,
plt.plot(power_organic_sorted_xvalidate, power_organic_sorted_y_poly_pred5_vali
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_organic_validate = r2_score(power_organic_yvalidate, y_poly_pred5_vali
mse_5_organic_validate = mean_squared_error(power_organic_yvalidate, y_poly_
pearson_corr_5_organic_validate, _ = pearsonr(power_organic_yvalidate, y_poly_
plt.figtext(0.6, 0.62, f"R^2: {r2_5_organic_validate:.4f}\nMSE: {mse_5_organic_
plt.subplot(3, 2, 6)
plt.scatter(power_organic_sorted_xtest, power_organic_sorted_ytest, color='b'
plt.plot(power_organic_sorted_xtest, power_organic_sorted_y_poly_pred5_test,
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
```

```

plt.title('test data')
r2_5_organic_test = r2_score(power_organic_ytest, y_poly_pred5_test)
mse_5_organic_test = mean_squared_error(power_organic_ytest, y_poly_pred5_test)
pearson_corr_5_organic_test, _ = pearsonr(power_organic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f'R^2: {r2_5_organic_test:.4f}\nMSE: {mse_5_organic_test:.4f}\npearson CC: {pearson_corr_5_organic_test:.4f}')

plt.tight_layout()
plt.show()

```



In []: